Towards State-Based RT Analysis of FSM-SADFGs on MPSoCs with Shared Memory Communication

Conference Paper · January 2017

DOI: 10.1145/3023973.3023979

4 authors, including:

Maher Fakih
OFFIS e.V.
15 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Kim Grüttner
OFFIS e.V.
81 PUBLICATIONS **217** CITATIONS
SEE PROFILE

Towards State-Based RT Analysis of FSM-SADFGs on MPSoCs with Shared Memory Communication

Ralf Stemmer
University of Oldenburg
ralf.stemmer@uol.de

Maher Fakih
OFFIS–Institute for Information Technology
maher.fakih@offis.de

Kim Grüttner
OFFIS–Institute for Information Technology
kim.gruettner@offis.de

Wolfgang Nebel
University of Oldenburg
wolfgang.nebel@uol.de

ABSTRACT

Scenario-Aware Data-Flow Graphs (SADFGs) were introduced to capture the behavior of embedded applications achieving a good trade-off between expressiveness and analyzability. On the one side, they support the timing analysis of real-time applications, especially those running on MPSoCs, due to the clean separation of computation and communication phases in their executing nodes. On the other side, SADFGs allow the expression of a more dynamic behaviors than Synchronous dataflow graphs by allowing dynamic token-rates of single nodes depending on pre-defined typical scenarios. The fact which leads to more efficiency and better throughput.

In this paper, we describe the extension of a previous model-checking based real-time analysis approach to allow the analysis of timing bounds for FSM-SADFGs mapped on a shared memory multiprocessor architecture. We demonstrate our approach on an MPEG decoder application being viable to obtain the worst-case end-to-end latency of its implementation under different scenarios on a 2-tiles MPSoC.

CCS Concepts

•Computer systems organization → System on a chip; Embedded systems;

Keywords

Scenario-aware dataflow graphs; Real-time analysis; SDF; FSM-SADF; MPSoCs; Model-checking

1. INTRODUCTION

In the last decade, video processing applications are moving from merely being utilized in the infotainment domain to become a major topic in the embedded systems domain. For example video processing applications are nowadays used in safety-critical automotive applications to detect pedestrians crossing the street or to recognize traffic signs. To guarantee the safety of such systems, real-time analysis methods must be utilized to validate the fulfillment of their hard real-time requirements. The behavior of such data-driven applications like video processing can be captured in the form of Synchronous Dataflow Graphs (SDFGs) introduced by Lee [10]. The advantage of SDFGs lies in the simplicity of the model which makes them easy to analyze. Yet, due to their simplicity and static behavior, capturing applications with behaviors of high dynamism is not supported by SDFGs.

Scenario-Aware Data-Flow graphs (SADFGs), first introduced in [15], achieve a good trade-off between expressiveness (allowing the expression of more dynamic behaviors than SDFGs) and analyzability. They extend SDF by the possibility of a dynamic data rate.

Finite-State-Machine Scenario-Aware Data-Flow (FSM-SADF) [14] graphs are a simplification of the general SADF graphs. Both extend SDFGs with scenarios. In an FSM-SADF Model of Computation (MoC) a set of typical scenarios is predefined through a finite state machine for a specific SDF application. The SDF application reacts to every scenario in a different manner leading to more efficiency and better throughput.

Fig. 1 shows an example FSM-SADFG of an MPEG decoder as introduced by [15]. The vertices of the graph can be either kernels (`VLD`, `MC`, `IDCT` and `RC`) or detectors (`FD`). The solid edges are called *data* channels while the dashed edges are called *control* channels. Every input port of the kernels/detector in an FSM-SADF has a consumption rate and likewise each output port has a production rate. In each scenario, kernels can have different production and consumption rates. Those ports with different varying rates are denoted by variables while ports with a fix rate over all scenarios are denoted by the rate itself. While general SADFGs enable multiple scenario changes during one *iteration*[1] by allowing multiple scenario detectors, FSM-SADFGs support only one detector per graph. This simplifies the analysis of such models. Initial tokens are visualized by dots on the edges. For example the data channel from the kernel `RC` to the detector `FD` is initialized by three tokens, so that the detector can be executed three times before the kernel `RC`

RAPIDO '17, January 23 - 25, 2017, Stockholm, Sweden

ISBN 978-1-4503-4840-9/17/01…$15.00

DOI: http://dx.doi.org/10.1145/3023973.3023979

[1] An *iteration* is the minimum non-zero execution (i.e. at least one kernel has been executed) such that the initial state of the graph is obtained [5].

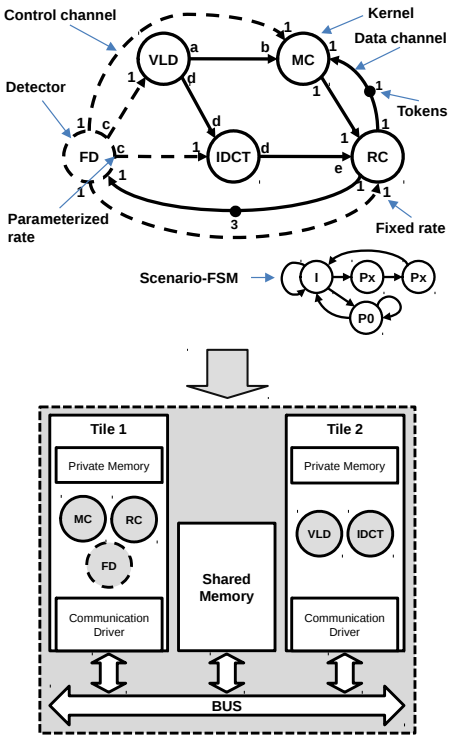

Figure 1: FSM-SADF graph of a MPEG-4 decoder inspired by [14] mapped on a MPSoC.

must be executed again to generate new tokens.

In this work, a mathematical (static) analysis is performed on a formal but abstract representation of both the software application and the hardware platform. This analysis takes into consideration all possible inputs and combinations of the running applications (i.e. abstracted as FSM-SADF actor activation, computation and communication) with all different hardware states (i.e. abstracted as resource access patterns) of the platform. We use timed automata (TA) and the UPPAAL model-checker [3] to capture and verify the behavior of the considered MPSoC system.

We claim the following contributions:

1. We enable a state-based real-time analysis (based on [5]) of multiple FSM-SADF applications mapped to an MPSoC platform with shared communication resources. State-based RT analysis methods, deliver very accurate results especially when handling heavy state-depended properties (e.g. First-Come-First-Serve (FCFS) arbitration protocol).

2. We will also present a technique which simplifies the finite state machine representing the scenarios by limiting the nondeterminism. This technique allows to reduce the state space of the resulting model under analysis.

2. RELATED WORK

Utilizing purely analytical approaches to obtain upper and lower timing bound of dataflow applications execution time is a wide spread research topic. Closest to our research are [12, 9, 11, 2]. While such approaches are fast and able to handle large systems, unfortunately they deliver pessimistic results especially when handling state-based bus arbitration protocols typically used in Multi-Processor Systems on a Chip (MPSoCs). In previous work [5] it was shown how far state-based RT analysis of SDFGs on MPSoCs can tighten the results compared to an analytical approach [12].

In the following, we give an excerpt of the main research related to our work mainly using state-based methods to analyze the performance of dataflow applications on MPSoCs.

Some previous work [6, 8] used model-checking to optimize buffer sizes in SDF applications. Yang et al. [16] introduce a state-space exploration approach to verify the hard real-time performance of applications modeled with SDFGs that are mapped to a platform with shared resources. Nevertheless, it does not consider a shared communication resource. A more recent work [1] followed the same path of our work, presenting a translation of single SDFGs to timed automata templates in order to analyze their behavior using model-checking. In contrast to our work, they focused on finding a maximal throughput on a given number of processors.

In [17, 18] the authors (similar to the work in [5]) transform a system model which includes an SDFG and a multiprocessor platform to a (priced) timed automata network and utilize an extended model-checker (UPPAAL CORA) to obtain optimal schedules combining optimization goals with optimal throughput and energy consumption.

In this work, we extend a previous model-checking based real-time analysis approach [5] for the analysis of timing bounds for FSM-SADFGs mapped on a shared memory multi-core architecture. In [5] only analysis of SDFGs was supported. We utilize timed automata (TA) as a common semantic model to represent worst-case execution times (WCET) of kernels, detectors and shared communication resource. The analysis model furthermore supports analysis of access protocols for buses, DMA, private (local) and shared memories of the MPSoC. For a given FSM-SADFGs and an MPSoC architecture different mappings and scheduling strategies can be examined by analyzing the resulting network of TA. Using the UPPAAL model-checker, safe timing bounds of the FSM-SADFG implemented on a MPSoC can be provided. The closest work to ours was published recently in [13] where a translation from FSM-SADF graphs to TA was presented. In difference to our approach, the authors concentrated in their translation and analysis only on the FSM-SADF MoC and did not consider MPSoC mapping and their resulting resource sharing aspects (i.e. contention on communication resources).

To the best of our knowledge, no other approach uses model-checking for the timing validation of hard real-time FSM-SADFGs on a MPSoC platform, considering the contention on shared on-chip components.

3. SYSTEM MODEL

Fig. 2 shows the models used in this paper in the context of the synthesis process (as defined by [7]) which will be described in the following sections.

3.1 Model of Computation (MoC)

The formal syntax of a FSM-SADF graph (such as in Fig. 1) are defined as follows (inspired by [14, 13]). Details of the semantic are described in [14].

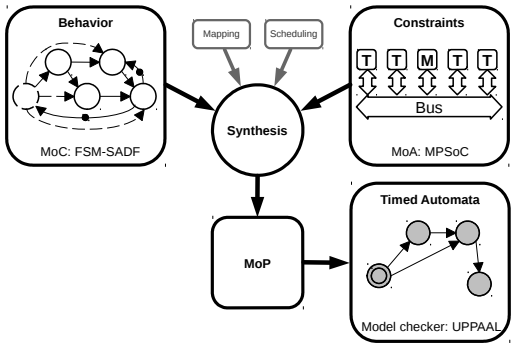

Figure 2: Synthesis process showing utilized models

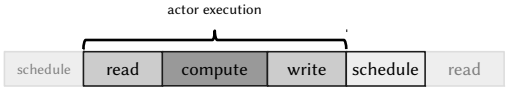

Figure 3: Execution phases of an FSM-SADF actor

Definition 1. *(FSM-SADF graph). An FSM-SADFG is a tuple* $G = (\mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{D}, \mathbb{S}, \mathbb{T}, \iota, \Phi)$*, where*

1. $\mathcal{S}$ *is the nonempty finite set of scenarios,*
2. *An actor is a tuple* $A = (\mathcal{P}, F)$ *consisting of a finite set* $\mathcal{P} \subseteq \mathbb{P}$*, and* F *a label, representing the functionality of the actor. If* $\mathcal{K}$ *is the nonempty finite set of kernels and* $d \notin \mathcal{K}$ *denotes the unique detector then* $\mathcal{A} = \mathcal{K} \cup \{d\}$ *is the total set of actors. The different execution phases of an SADF actor are shown in Fig. 3.*
3. $\mathbb{P}$ *is the set of ports of all actors. A Port* $P \in \mathbb{P}$ *is defined as a tuple* $P = (Dir, Rate)$ *where* $Dir \in \{I, O\}$ *defines whether* P *is an input or output port, and* $Rate = [r_1, r_2, \ldots r_n]$ *is an array of rates for each scenario* $s \in \mathcal{S}$*. The rate* $r_i \in \mathbb{N}_0$ *of a port specifies the number of tokens consumed (*$Rate_c$*) or produced (*$Rate_p$ *) by the corresponding port when the corresponding actor* $a \in \mathcal{A}$ *fires. If* $\mathcal{P}_c$ *is the set of all input ports (consumer) where* $\forall p \in \mathcal{P}_c : p.Dir = I$ *and* $\mathcal{P}_p$ *the set of all output ports (producer) where* $\forall p \in \mathcal{P}_p : p.Dir = O$*, then* $\mathbb{P} = \mathcal{P}_c \cup \mathcal{P}_p$*,*
4. $\mathcal{D} \subseteq \mathcal{K} \times \mathcal{A}$ *is the set of edges (called channels in context of FSM-SADF). If* $\mathcal{D}_{ctrl}$ *is the set of control channels in the FSM-SADF and* $\mathcal{D}_{data}$ *is the set of data channels then* $\mathcal{D}$ *is defined as* $\mathcal{D} = \mathcal{D}_{ctrl} \cup \mathcal{D}_{data}$*. A channel* $D \in \mathcal{D}$ *is a triple* $D = (P_p, P_c, i)$ *where* $P_p \in \mathcal{P}_p$ *is the port of the producer and* $P_c \in \mathcal{P}_c$ *is the port of a consumer and* $i \in \mathbb{N}_0$ *is the number of initial tokens on an channel. All ports of all actors are connected to exactly one channel, and all channels are connected to ports,*
5. $(\mathbb{S}, \mathbb{T}, \iota, \Phi)$ *is the FSM of the detector, where* $\mathbb{S}$ *is the nonempty set of states,* $\mathbb{T} : \mathbb{S} \to 2^{\mathbb{S}}$ *is the transition function,* $\iota \in \mathbb{S}$ *is the initial state, and* $\Phi : \mathbb{S} \to \mathcal{S}$ *associates each state with a scenario,*

3.2 Model of Architecture (MoA)

Fig. 1 (bottom) depicts an example of our proposed architecture. A tile is made up of a processing element (PE) which has a configurable bus connection. In addition, every PE has a private memory. A bus is used to connect the tiles to shared memory. This enables actors mapped to different tiles to communicate via buffers mapped to shared memory using non-preemptive arbitration protocols. Only explicit communication (message passing) between actors will be visible on the interconnect and the shared memory.

Definition 2. *(Tile) A tile is a tuple* $T = (PE, M_p)$ *with processing element* $PE = (PE_{type}, f)$ *where* PE_{type} *is the type of the processor and* f *is its clock frequency, and* M_p *is the size of memory.*

Definition 3. *(Execution Platform) An execution platform is defined as* $EP = (\mathcal{T}, \mathcal{I}, \mathcal{M}_\mathcal{S})$ *consisting of*

1. *a finite set* $\mathcal{T}$ *of tiles* T*,*
2. *a finite set* $\mathcal{I}$ *of shared interconnects with* $I \in \mathcal{I}$ *and* $I = (B_i, AP, CS)$ *with* B_i *being the bandwidth in bits/cycle,* AP *is the arbitration protocol (FCFS, Fixed-Priority, Round-Robin, TDMA) and* CS *is the communication style supported by the interconnect,*
3. *a finite set* $\mathcal{M}_\mathcal{S}$ *of shared storage resources (such as memories)* $M_S = (B_s, m_s)$*, each of them having specific size* m_s *in bits and a bandwidth* B_s *in bits/cycle.*

3.3 Synthesis

The system synthesis (see Fig. 2) includes the processes of binding and scheduling the behavioral model onto the defined architecture. Mapping the FSM-SADFG onto our MoA is defined as follows:

Definition 4. *(Mapping) If* $\mathcal{A}$ *is the set of actors of all FSM-SADFGs,* $\mathcal{D}$ *the set of all channels,* $\mathcal{T}$ *the set of all tiles of the platform configuration,* $\mathcal{I}$ *the set of all interconnects,* $\mathcal{M}_\mathcal{S}$ *the set of all shared storage resources,* $\mathcal{M}_\mathcal{P}$ *the set of all private memories, then a mapping can be defined as a tuple* $M = (\alpha, \beta, \delta, \zeta)$ *with*

1. *the function* $\alpha : \mathcal{A} \to \mathcal{T}$ *maps every actor to a unique tile (multiple actors can be assigned to one tile).*
2. *the function* $\beta : \mathcal{D} \to (\mathcal{M}_\mathcal{P} \cup (\mathcal{I} \times \mathcal{M}_\mathcal{S}))$*, where:* $\mathcal{M}_\mathcal{P}$*: mapping to private memory and* $(\mathcal{I} \times \mathcal{M}_\mathcal{S})$*: mapping to shared storage resource that can be accessed using an interconnect* $(i \in \mathcal{I})$
3. *the function* $\delta : \mathcal{D} \to \mathbb{N}_0$ *which assigns for every channel* $(d \in D)$ *the maximum number of tokens it can hold (buffer-size). In the case of FSM-SADF, the size of the buffer of a specific channel is equal to the highest number of tokens on this channel among all scenarios.*
4. *the function* $\zeta : \mathcal{D} \to \mathbb{N}_0$ *which assigns for every channel* $(d \in D)$ *the token size attribute* T_s *(in bits).*

The channel mapped to a private or to a shared storage resource represents a consumer-producer FIFO buffer in an actual implementation.

The following definitions allow us to express the scheduling behavior of each SADFG of a scenario mapped to tiles on the platform:

DEFINITION 5. *(Self-timed (static-order) schedule) For an FSM-SADFG with a repetition vector γ for each scenario, a static-order schedule SO is an ordered list of the actors (to be executed on some tile), where every actor a is included $\gamma(a)$ times.*

A repetition vector of an FSM-SADFG is defined as the vector that specifies the number of times every actor has to be executed in a specific scenario such that the initial state of the graph is obtained.

Because the number of executions of an actor depends on the scenario, each scenario may have a different scheduling. Self-timed means that FSM-SADFGs are executed in a static cyclic order as soon as the input data is available.

DEFINITION 6. *(Scheduling Assignment) Let $\mathcal{SO}$ be the set of all SO schedules for all FSM-SADFGs as result of a scenario. A scheduling assignment is a function $S : \mathcal{T} \rightarrow \mathbf{so}$, which assigns to every tile $t \in \mathcal{T}$ a subset $\mathbf{so} \subseteq \mathcal{SO}$.*

3.4 Model of Performance (MoP) Extraction

In order to verify that the performance of the FSM-SADFG stays within the required bounds, we must keep track of all possible timing delays in all scenarios of all mapped FSM-SADFGs to the MPSoC platform. To achieve this, a MoP is extracted from the synthesis process which includes all the SW/HW components with their properties influencing the timing in the considered system.

From the hardware abstraction point of view, we consider a *Time-accurate Bus-Functional-Model* (BFM) [4] abstraction. In this model, the application layer issues read/write transactions on the interconnect (see for example the FCFS-arbitrated bus in Fig. 1) and upper/lower latency bounds are calculated, abstracting details of the communication protocol. This model is appropriate in case no accurate (constant) timings can be obtained when transferring data of specific size through an interconnect with a specific communication protocol.

After synthesis, the following system components are annotated with execution times and delays: *communication drivers*, *schedulers*, *actors*, *interconnects*, *private* and *shared memory*.

DEFINITION 7. *(Delay annotations) If $\mathcal{FSM}-\mathcal{SADFG}$ is the set of FSM-SADFGs, $\mathcal{A}$ is the set of actors, $\mathcal{H}$ the set of schedulers, $\mathcal{D}$ the set of edges, $\mathcal{C}$ the set of communication drivers, I is the set of interconnects, $\mathcal{M}_\mathcal{S}$ the set of shared storage resources, and $\mathcal{M}_\mathcal{P}$ the set of private memories, the following delay functions are defined:*

- $\Delta_A : \mathcal{A} \times \mathcal{S} \times \mathcal{T} \rightarrow \mathbb{N}_{>0} \times \mathbb{N}_{>0}$ *which provides an execution time interval $[BCET, WCET]$ for each actor representing the cycles needed to execute the actor behavior (compute phase) for every scenario on the corresponding tile. We assume this delay can be obtained through static timing analysis or an appropriate measurement approach.*
- $\Delta_H : \mathcal{H} \times \mathcal{T} \rightarrow \mathbb{N}_{>0} \times \mathbb{N}_{>0}$, $\Delta_C : \mathcal{C} \times \mathcal{T} \rightarrow \mathbb{N}_{>0} \times \mathbb{N}_{>0}$ *assigns (in analogy to Δ_A) to every scheduler and communication driver a delay interval, which can be estimated in the same way as Δ_A.*
- $\Delta_D : \mathcal{D} \times (\mathcal{M}_\mathcal{P} \cup (\mathcal{I}, \mathcal{M}_\mathcal{S})) \rightarrow \mathbb{N}_{>0} \times \mathbb{N}_{>0}$ *assigns to each communicating edge $d \in D$ mapped to a communication primitive a delay Δ_D which depends on:*

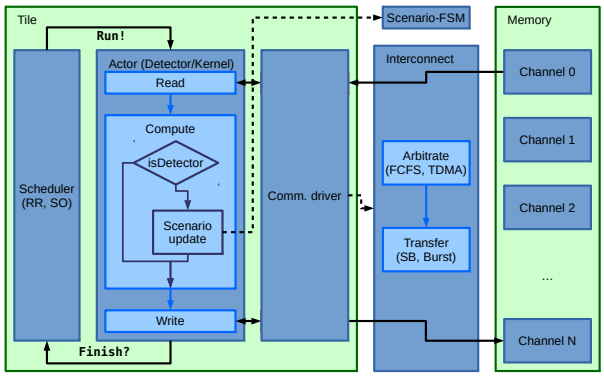

Figure 4: Abstract representation of the TA-templates (blue) capturing the temporal behavior of different components of the MoP.

1. *the number and size of the tokens being transported,*
2. *the type of transaction (read or write),*
3. *Δ_I: latency of the communication interconnect to transport current transaction and*
4. *Δ_M: latency of the target storage resource ($\Delta_{\mathcal{M}_\mathcal{S}}$ or $\Delta_{\mathcal{M}_\mathcal{P}}$)*

- $\Delta_P : \mathcal{A} \times \mathcal{T} \rightarrow \mathbb{N}_{>0} \times \mathbb{N}_{>0}$ *which provides a polling delay that should be waited by an actor when blocking on a shared storage resource.*

Regardless of whether the channel is a control- or data channel, the buffer size is fixed to the maximum size required among all scenarios by an actor within an iteration.

4. CAPTURING MOPS AS TA

In this chapter we present the steps that are necessary to get a representation of an FSM-SADF application mapped and scheduled on a MPSoC platform. Furthermore we present a technique to abstract the scenario state machine to get a deterministic scenario sequence to avoid state explosion when applying model checking techniques.

4.1 Actors and Channels on MPSoC

Figure 4 shows an abstract representation of the timed-automata templates used to capture the model of performance (MoP). It shows one actor of an FSM-SADFG running on one tile (compared to Fig. 1 where 5 actors are mapped on 2 tiles) The actor's channels are mapped to a shared memory. An actor can be either a kernel or a detector, as defined in section 3.1 and the shared memory is connected via an interconnect to the tile.

For every actor, an actor-TA-template is instantiated capturing its three phases of execution (see in Fig. 3). The first phase is the `Read` phase in which all tokens are read from all channels. Next the `Computation` phase in which the read data gets processed. At the end of this phase, if the current actor is of the type `Detector`, it triggers a scenario change (see Fig. 4). And finally the `Write` phase is executed in which new tokens are written to the output ports.

The scenario-state-machine TA is modeled as a part of the detector and therefore implicitly mapped to the tile where

the detector is running. Additionally to the FSM-SADFG actors, a scheduler and an interconnect-driver are modeled as TA for each tile. The scheduler signals an actor to fire. The interconnect-driver models the behavior of the communication to shared memory using arbitration protocols. In addition, an interconnect-TA-template models the interconnect temporal behavior, the number of transfered tokens, the interconnect latency and the arbitration strategy used. The memory latency is captured by the channels (FIFO buffers) timed automata, where for every channel a dedicated TA is instantiated. For more details about the implementation of the different TA templates please refer to [5].

A scenario update can take place at every detector firing. The detector signalizes the scenario update for dependent actors through the control channels. During the `Compute` phase (see Fig. 4) of the detector, the scenario-FSM can be triggered to change to a successor scenario. The scenario is valid for other actors after the `Write` phase of the detector is completed where the control tokens are written to the corresponding control channels. When a kernel actor fires it should read the control token in the control channel first in order to detect the scenario. This requires a prioritization of the channels leading to read the control channel before all other channels.

Since the change of scenarios takes place in the FSM which is activated in the `Compute` phase of the detector, the scenario change timing delay is considered to be a part of the $[BCET; WCET]$ interval of the `Compute` phase.

Kernels of the FSM-SADF that can work in different scenarios get notified of their scenario by control tokens produced by the detector.

Control tokens are propagated from the detector to other kernels using the same shared resources that data tokens use. Therefore control channels are modeled just like data channels.

4.2 Simplification of the Scenario-FSM

To handle non-determinism in the scenario-FSM of an FSM-SADF graph, multiple possible successor states of an initial state must be abstracted to a single state. The method to do this simplification is shown in Fig. 5.

First all scenarios need to be analysed separately to get their worst case and best case execution times. These times will be the WCET- and BCET-cost of the nodes of the graph that shall be simplified. To get a worst case scenario sequence, the path between two scenarios with the highest WCET-cost must be determined. The best case scenario sequence is the path with the lowest BCET-cost. The new state sequence is not necessarily the longest path of the original graph for the worst case scenario sequence, or the shortest path for the best case sequence since the WCET and BCET of each node gets considered.

The simplification for our MPEG-example shown in Fig. 5 was done for the path from node I to node I of the next iteration. With the highest cost for the path I, P_{80}, P_{99} and the lowest for I, P_0, P_{30}.

This abstraction is valid since we are interested in getting the worst-case value of some timing constraint (such as *end-to-end latency*). That is why a complex state machine can be simplified into a deterministic sequence of states. The same procedure can be done for the best-case scenario sequence. This leads to new deterministic FSMs which describe the token rates in a more coarse way.

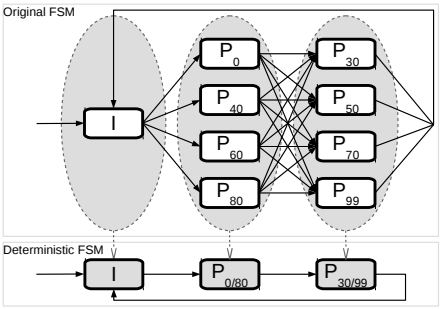

Figure 5: Top: original scenario-FSM [13], Bottom: Simplified FSM

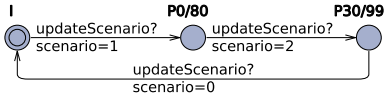

Figure 6: The deterministic scenario FSM as TA

5. EVALUATION

Table 1: The token rates in each scenario.

Rate	*Nondet. Scenarios*			*Det. Scenarios (BC/WC)*		
	I	P_0	P_x	I	$P_{0/80}$	$P_{30/99}$
a	0	0	1	0	0 / 1	1 / 1
b	0	0	x	0	0 / 80	30 / 99
c	99	1	x	99	1 / 80	30 / 99
d	1	0	1	1	0 / 1	1 / 1
e	99	0	x	99	0 / 80	30 / 99

For the evaluation, we analyze the worst-case end-to-end latency of an MPEG decoder from [14] running on a 2-tiles MPSoC with a shared memory accessible via a FCFS arbitrated bus (Fig. 1). The kernels `MC` and `RC` and the detector `FD` are mapped to *Tile 1*, the kernels `VLD` and `IDCT` to *Tile 2*. For the interprocess communication of actors running on the same tile, the private memory is used. Channels between actors on different tiles are mapped to shared memory.

The port rates of the worst-case-scenario and the best-case-scenario are shown in the right part of Tab. 1. For example the port rate b (input of kernel `MC`) can be either 0 or 80 depending if worst-case or best-case analysis gets applied. Before the abstraction it had multiple rates ranging between $0, 40, 60$ and 80 (see Fig. 5). All of them would have been considered for the analysis in a non-deterministic way. So a previous analysis of each of the possible scenarios (see Tab. 2) showed that the WCET and BCET for the rates of scenario P_{40} and P_{60} are between the BCET of scenario P_0 and the WCET of scenario P_{80}. For this reason in Fig. 5 the state $P_{0/80}$ is equivalent to the original state P_{80} in the worst case (see Tab. 1) since in this scenario, the kernels `RC` and `MC` (see Fig. 1) are executed most frequently and the highest number of data tokens needs to be transferred between kernels. For the best-case scenario, $P_{0/80}$ becomes equivalent to the original state P_0 (Tab. 1) where most data channels transfers holding zero rates and kernels are activated at most once per iteration.

The separate analysis of each of the possible scenarios (Tab. 2) confirmed this. This abstracted deterministic FSM constructed in Fig. 5 was implemented as a TA as shown in

Table 2: Results of the separate end-to-end latency analysis. To determine the latency of P_{99} the Convex Hull over-approximation optimisation was used.

Property	P_0	P_{30}	P_{40}	P_{50}	P_{60}	P_{70}	P_{80}	P_{99}
latency	1109	3002	3845	4680	5566	6443	7305	9618
RAM in MB	312	439	815	1149	1672	2397	3232	28
CPU time	1h	2h	5h	8h	15h	26h	43h	0.5h
state explored	10e6	15e6	29e6	41e6	62e6	89e6	122e6	98e6

Fig. 6. Each time the detector `FD` is executed, the FSM gets triggered by the `updateScenario` sync.-channel inside the TA model to change the scenario (see `scenario` in Fig. 6).

For the implementation of the TA model, we used the UPPAAL model checker [3]. In our experiments we used the UPPAAL 64 Bit version 4.1.19 for Linux. The hardware was a system with a 2.5 GHz AMD Opteron^{TM}Processor 6282 SE and 512 GB of RAM. All experiments took about two weeks, consumed less than 29 GB of RAM. In addition, the largest number of states explored was about 656 million states. The results of our state-based RT analysis are shown in Tab. 3. The latency of the worst-case scenario sequence using the worst-case rates for the scenarios $P_{0/80}$ and $P_{30/99}$ (see in Tab. 1) ranges between 8928 and 8969 cycles. For the best-case scenario sequence using the best-case rates (see rates in Tab. 1) the latency ranged between 2770 and 6758.

Table 3: Worst-case and best-case latency and resources needed for analysis

Scenario	*Latency*	*States*	*Mem.*	*Time*
WC scenario seq. WC lat.	8969	656e6	28GB	370h
WC scenario seq. BC lat.	8928	372e6	16GB	223h
BC scenario seq. WC lat.	6758	656e6	28GB	414h
BC scenario seq. BC lat.	2770	372e6	16GB	219h

6. CONCLUSION

In this work, we presented a state-based real-time analysis method which enables the real-time verification of FSM-SADF applications mapped to MPSoCs with shared communication resources. We showed that our method was able to analyze the worst-case latency and the best-case-latency of both the worst-case and the best-case scenario sequence of a MPEG4 decoder captured in the FSM-SADF MoC and run on an MPSoC of two tiles with shared memory. To avoid an unmanageable state space of the already complex model, we avoided nondeterminism by reducing the scenario state machine to a deterministic worst-case or best-case version of the original one.

To manage more complex applications executed on more complex MPSoCs, we will consider other methods, for instance the promising probabilistic RT approaches, in future work.

Acknowledgement

This work has been partially supported by the SAFEPOWER project with funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646531.

7. REFERENCES

[1] W. Ahmad, E. de Groote, P. K. Hölzenspies, M. I. A. Stoelinga, and J. C. van de Pol. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. In *Proceedings of 14th IEEE International Conference on Application of Concurrency to System Design (ACSD)*. IEEE, 2014.

[2] J. Bastos, S. Stuijk, J. Voeten, R. Schiffelers, J. Jacobs, and H. Corporaal. Modeling resource sharing using fsm-sadf. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*, pages 96–101, Sept 2015.

[3] J. Bengtsson and W. Yi. Timed Automata: Semantics, Algorithms and Tools. In *In Lecture Notes on Concurrency and Petri Nets. LNCS 3098*, pages 87–124. Springer-Verlag, 2004.

[4] L. Cai and D. Gajski. Transaction Level Modeling: an Overview. In *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2003*, pages 19–24, Oct. 2003.

[5] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg. State-based real-time analysis of SDF applications on mpsocs with shared communication resources. *Journal of Systems Architecture - Embedded Systems Design*, 61(9):486–509, 2015.

[6] M. Geilen, T. Basten, and S. Stuijk. Minimising buffer requirements of synchronous dataflow graphs with model checking. In *Proceedings of the 42nd annual Design Automation Conference*, pages 819–824, 2005.

[7] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich. Electronic System-Level Synthesis Methodologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1517 –1530, Oct. 2009.

[8] Z. Gu, M. Yuan, N. Guan, M. Lv, X. He, Q. Deng, and G. Yu. Static scheduling and software synthesis for dataflow graphs with symbolic model-checking. pages 353–364. IEEE, Dec. 2007.

[9] J. P. H. M. Hausmans, M. H. Wiggers, S. J. Geuns, and M. J. G. Bekooij. Dataflow analysis for multiprocessor systems with non-starvation-free schedulers. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems*, M-SCOPES '13, pages 13–22, New York, NY, USA, 2013. ACM.

[10] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[11] A. Lele, O. Moreira, J. Bastos, R. Almeida, P. Pedreiras, and K. van Berkel. Analyzing preemptive fixed priority scheduling of data flow graphs. In *Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on*, pages 50–59. IEEE, 2014.

[12] A. Shabbir, A. Kumar, S. Stuijk, B. Mesman, and H. Corporaal. CA-MPSoC: An Automated Design Flow for Predictable Multi-processor Architectures for Multiple Applications. *Journal of Systems Architecture*, 56(7):265–277, 2010.

[13] M. Skelin, E. R. Wognsen, M. C. Olesen, R. R. Hansen, and K. G. Larsen. Model checking of finite-state machine-based scenario-aware dataflow using timed automata. In *Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on*, pages 1–10. IEEE, 2015.

[14] S. Stuijk, A. H. Ghamarian, B. D. Theelen, M. C. W. Geilen, and T. Basten. FSM-based SADF. Technical report, Eindhoven University of Technology, Department of Electrical Engineering, 2008.

[15] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proceedings of the Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings.*, MEMOCODE '06, pages 185–194, Washington, DC, USA, 2006. IEEE Computer Society.

[16] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal. Automated bottleneck-driven design-space exploration of media processing systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 1041–1046, 3001 Leuven, Belgium, Belgium, 2010. European Design and Automation Association.

[17] X.-Y. Zhu, R. Yan, Y.-L. Gu, and G. Zhang. Static Optimal Scheduling and Mapping of Synchronous Dataflow Graphs on a Heterogeneous Multiprocessor Platform with Model Checking. 2014.

[18] X.-Y. Zhu, R. Yan, Y.-L. Gu, J. Zhang, W. Zhang, and G. Zhang. Static Optimal Scheduling for Synchronous Data Flow Graphs with Model Checking. In *FM 2015: Formal Methods*, pages 551–569. Springer, 2015.