

# Modeling, Analysis, and Hard Real-time Scheduling of Adaptive Streaming Applications

Jiali Teddy Zhai, Sobhan Niknam, and Todor Stefanov, *Member, IEEE*

**Abstract**—In real-time systems, the application’s behavior has to be predictable at compile-time to guarantee timing constraints. However, modern streaming applications which exhibit adaptive behavior due to mode switching at run-time, may degrade system predictability due to unknown behavior of the application during mode transitions. Therefore, proper temporal analysis during mode transitions is imperative to preserve system predictability. To this end, in this paper, we initially introduce Mode Aware Data Flow (MADF) which is our new predictable Model of Computation (MoC) to efficiently capture the behavior of adaptive streaming applications. Then, as an important part of the operational semantics of MADF, we propose the Maximum-Overlap Offset (MOO) which is our novel protocol for mode transitions. The main advantage of this transition protocol is that, in contrast to self-timed transition protocols, it avoids timing interference between modes upon mode transitions. As a result, any mode transition can be analyzed independently from the mode transitions that occurred in the past. Based on this transition protocol, we propose a hard real-time analysis as well to guarantee timing constraints by avoiding processor overloading during mode transitions. Therefore, using this protocol, we can derive a lower bound and an upper bound on the earliest starting time of the tasks in the new mode during mode transitions in such a way that hard real-time constraints are respected.

## I. INTRODUCTION

**T**O handle the ever-increasing computational demands and meet hard real-time constraints in streaming applications, where the huge amount of streaming data should be processed in a short time interval, embedded systems have relied on Multi-Processor System-on-Chip (MPSoC) platforms to benefit from parallel processing. To efficiently exploit the computational capacity of MPSoCs, however, streaming applications must be expressed primarily in a parallel fashion. The common practice for expressing the parallelism in an application is to use parallel Models of Computation (MoCs) [1]. Within a parallel MoC, a streaming application is modeled as a directed graph, where graph nodes represent actors (i.e., tasks) and graph edges represent data dependencies. Actors are executed concurrently and communicate data explicitly via FIFOs. For example, Synchronous Data Flow (SDF) [2] and Cyclo-Static Data Flow (CSDF) [3] are two popular parallel MoCs because of their compile-time analyzability. Due to the static nature of SDF and CSDF MoCs, the actors are restricted to produce and consume data with fixed rates per firing or, in case of CSDF, with fixed periodic patterns.

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEK-TCAD special issue.

The authors contributed to the paper equally. The authors are with the Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands, Email: teddyzhai@gmail.com, {s.niknam,t.p.stefanov}@liacs.leidenuniv.nl. This research is supported by the Dutch Technology Foundation STW under the Robust Cyber Physical Systems program (Project 12695).

Nowadays, many modern streaming applications, in the domain of multimedia, image, and signal processing, increasingly show adaptive behavior at run-time. For example, a computer vision system processes different parts of an image continuously to obtain information from several regions of interest depending on the actions taken by the external environment. This adaptive behavior, however, cannot be effectively expressed with an SDF or CSDF model due to their limited expressiveness. As a result, more expressive models, e.g., Scenario-Aware Data Flow (SADF) [4], Finite State Machine (FSM)-based Scenario-Aware Data Flow (FSM-SADF) [5], Variable-rate Phased Data Flow (VPDF) [6], and Mode-controlled Data Flow (MCDF) [7], have been proposed and deployed as extensions of the (C)SDF model. These MoCs are able to capture the behavior of an adaptive streaming application as a collection of different static behaviors, called scenarios or modes, which are individually predictable in performance and resource usage at compile-time.

Moreover, to guarantee tight timing constraints in modern streaming applications with adaptive behavior nature, proper temporal analysis for application execution during mode transitions, when the application’s behavior is switching from one mode to another mode, is imperative at compile-time. However, such analysis can be difficult due to the fact that different actors in different modes are concurrently executing during mode transitions. This difficulty comes directly from the protocol adopted for the mode transitions. In the existing adaptive MoCs, like MCDF [7] and FSM-SADF [5], a protocol, referred as self-timed transition protocol, has been adopted which specifies that actors are scheduled as soon as possible not only in each mode individually, but also during mode transitions. This protocol, however, introduces interference of one mode execution with another one, as explained in Section IV-C1. As a consequence, the temporal analysis of a mode transition is tightly dependent on the mode transitions that occurred in the past. Another consequence of the incurred interference between modes is the high time complexity of analyzing mode transitions, as the mode transitions cannot be analyzed independently, e.g., see the state-space exploration approach proposed in [5].

Therefore, to overcome the aforementioned interference issue and consequent problems caused by the self-timed transition protocol, in this paper, we propose a new MoC called Mode Aware Data Flow (MADF) to model adaptive streaming applications, that is armed by a novel transition protocol called Maximum-Overlap Offset. This transition protocol enables an independent analysis for mode transitions. The specific novel contributions of this paper are the following:

- We propose a new MoC, Mode-Aware Data Flow (MADF), that has the advantages of SADF [4] and VPDF [6]. Inspired

by SADF, we characterize the behavior of adaptive streaming applications with individual modes and transitions between them. Similar to VPDF, the length of production/consumption sequences for an actor varies from one mode to another. The length is only fixed when the mode is known. Then, based on the clear distinction between modes and transitions, we define analyzable operational semantics for MADF;

- As an important part of the operational semantics of MADF, we propose the Maximum-Overlap Offset (MOO) which is our novel protocol for mode transitions. The main advantage of this transition protocol is that, in contrast to the self-timed transition protocol, adopted in [5], [7], it avoids timing interference between modes upon mode transitions. As a result, this transition protocol enables an independent analysis for mode transitions. This means, the analysis of any mode transition is independent from the mode transitions that occurred in the past. This independent analysis significantly reduces the complexity of the analysis as the complexity merely depends on the number of allowed transitions. This is crucial for applications with a large number of modes and possible transitions;
- Based on the novel MOO transition protocol, we propose a hard real-time analysis approach to guarantee the timing constraints by avoiding processor overloading, i.e., avoiding that the total utilization of allocated tasks on a processor exceeds its capacity, during mode transitions. Our analysis is much simpler and faster than the computationally intensive state-of-the-art timing analysis approaches such as [5].

The remainder of this paper is organized as follows: Section II gives an overview of the related work. Section III introduces the background needed for understanding the contributions of this paper. Our novel adaptive MoC and transition protocol are then introduced in Section IV. Based on the novel transition protocol, in Section V, we present our hard real-time analysis approach to guarantee the timing constraints during mode transitions. In Section VI, two case studies are presented to illustrate the practical applicability of our proposed MADF mode, transition protocol, and real-time analysis. Finally, Section VII ends the paper with conclusions.

## II. RELATED WORK

To model the adaptive behavior of modern streaming applications while having certain degree of compile-time analyzability, different MoCs such as Scenario-Aware Data Flow (SADF) [4], Finite State Machine (FSM)-based Scenario-Aware Data Flow (FSM-SADF) [5], Variable-rate Phased Data Flow (VPDF) [6], Mode-controlled Data Flow (MCDF) [7], and Parameterized SDF (PSDF) [8] have been already proposed in the literature.

In SADF [4] and FSM-SADF [5], *detector* actors are introduced to parameterize the SDF model. All valid scenarios and their possible order of occurrence, which is shown either by using a Markov chain [4] or finite state machine [5], must be predefined at compile-time. Each scenario consists of a set of valid parameter combination that determines a scenario of SADF. This guarantees the consistency of SADF in individual scenarios, therefore, no run-time consistency check is required. In a scenario, the SADF model behaves the same way as the

SDF model. Therefore, an SADF graph can be seen as a set of SDF graphs. In the initial FSM-SADF definition, all the production and consumption rates of the data-flow edges are constant within a graph iteration of a scenario.

For the FSM-SADF MoC [5], the authors proposed an approach to compute worst-case performance among all mode transitions, assuming the self-timed transition protocol. Although it is an exact analysis, the approach has inherently exponential time complexity. Moreover, this approach leads to timing interference between modes upon mode transitions. In contrast, our approach does not introduce interference between modes due to the novel MOO transition protocol proposed in Section IV-C2. The timing behavior of individual modes and during mode transitions can be analyzed independently. In addition, our approach considers allocation of actors on processors, which by itself is a harder problem than the one addressed in [5].

In [9], the author proposes to use a linear model to capture worst-case transition delay and period during scenario transitions of FSM-SADF. Our transition protocol is conceptually similar to the linear model. However, we obtain the linear model in a different way, specifically simplified for the adopted hard real-time scheduling framework. For instance, finding a reference schedule is not necessary in our case, but being crucial in the tightness of the analysis proposed in [9]. Moreover, our approach solves the problem of changing the application graph structure during mode transitions, which was not studied in [9].

For VPDF [6], the analysis has been limited to computing buffer sizes under throughput constraints so far. The execution of a VPDF graph on MPSoC platforms under hard real-time constraints has not been studied. In particular, the allocation of actors and how to switch from one mode to another one are not discussed. Moreover, delay due to mode transitions has not been investigated. Our approach, on the other hand, takes these important factors into account. Therefore, our analysis results are directly reflected in a real implementation.

MCDF [7] is another adaptive MoC which properties can be partly analyzed at compile-time. The MCDF MoC primarily focuses on Software-Defined Radio applications, where different sub-graphs need to be active in different modes. This is achieved by using *switch* and *select* actors. The author implicitly assumes self-timed scheduling during mode transitions. Based on this assumption, a worst-case timing analysis is developed. Similar to the case of SADF, the use of the self-timed scheduling introduces timing interference between modes. As a consequence, the analysis must take into account the sequence of mode transitions of interest. Although the author provides an upper bound of timing behavior for a parameterized sequence of mode transitions, the accuracy is still unknown. In contrast, our approach results in a timing analysis of mode transitions that is independent from already occurred transitions. Moreover, the analysis results are directly reflected in the final implementation. In this sense, our analysis is exact in the timing behavior of mode transitions.

In [8], a meta-modeling technique is proposed to augment the expressive power of wide range of existing data-flow models which have the graph iteration concept. In [8], the proposed

technique is especially applied to the SDF model which is called Parameterized SDF (PSDF). In PSDF, separate *init* and *sub-init* graphs are proposed to reconfigure the body graph in a hierarchical manner. In this model, functional properties can only be partially decided at compile-time, and thus run-time verification is needed. To this end, for all configurations, computing a schedule and verifying consistency for both graphs and specifications need to be fulfilled at run-time which is pretty complex procedure. In addition, temporal analysis to find the worst-case system reconfiguration delay to preserve model predictability is not proposed. In contrast, our MADF model does not require run-time consistency check as every mode in our model is predefined at compile-time and represented as a CSDF graph. In addition, our MADF provides the temporal analysis of the mode transitions at compile-time using the MOO transition protocol.

In [10], [11], an analysis is proposed to reason about worst-case response time of a task graph in case of a mode change. However, the task graph has very limited expressiveness and is not able to model the behavior of adaptive streaming applications. Instead, in our paper, we define a more expressive MoC that is amenable to adaptive application behavior and real-time analysis.

In [12], [13], the authors focus on timing analysis for mode changes of real-time tasks. The starting times of new mode tasks need to be delayed to avoid overloading of processors during mode changes. In [12], [13], however, it is assumed that tasks are independent. The proposed algorithms are thus not applicable to adaptive MoCs, since the starting times of tasks in adaptive MoCs depend on each other due to data dependencies. Moreover, the algorithms in [12], [13] involve high computational complexity because fixed-point equations must be solved at every step in the algorithms. In contrast, in our paper, we propose an adaptive MoC and analysis for applications with data-dependent tasks, which is more realistic and applicable to wider range of real-life streaming applications. Moreover, our analysis is simpler with low computational and time complexity.

### III. BACKGROUND

In this section, we provide a brief overview of our system model, the CSDF MoC, and the scheduling framework presented in [14]. This background is needed to understand the novel contributions of our work.

#### A. System Model

The considered MPSoC platforms in this work are homogeneous, i.e., they may contain multiple, but the same type of programmable Processing Elements (PEs) with distributed memories. Moreover, the platform must be predictable, which means timing guarantees are provided on the response time of hardware components and OS schedulers. The precision-timed (PRET) [15] platform is such an example. On the software side, we assume partitioned scheduling algorithms, i.e., no migration of tasks between PEs is allowed. The considered scheduling algorithms on each PE include Fixed-Priority Preemptive Scheduling (FPPS) algorithms, such as RM [16], or dynamic scheduling algorithms, such as EDF [16].

#### B. Cyclo-Static Data Flow (CSDF)

An application modeled as a CSDF [3] is defined as a directed graph  $G = (\mathcal{A}, \mathcal{E})$  that consists of a set of actors  $\mathcal{A}$  which communicate with each other through a set of edges  $\mathcal{E}$ . Actors represent computation while edges represent data dependency due to communication and synchronization. In CSDF, every actor  $A_i \in \mathcal{A}$  has an execution sequence  $C_i = [c_1, c_2, \dots, c_{\phi_i}]$  of length  $\phi_i$ . This means, the  $x$ -th time that actor  $A_i$  is fired, it performs the computation  $C_i((x-1) \bmod \phi_i + 1)$ . Similarly, production and consumption of data tokens are also sequences of length  $\phi_i$  in CSDF. The token production of actor  $A_i$  to edge  $E_j$  is represented as a sequence of constant integers  $PRD_j = [prd_1, prd_2, \dots, prd_{\phi_i}]$ , called *production sequence*. Analogously, token consumption from every input edge  $E_k$  of actor  $A_i$  is a predefined sequence  $CNS_k = [cns_1, cns_2, \dots, cns_{\phi_i}]$ , called *consumption sequence*. The  $x$ -th time that actor  $A_i$  is fired, it produces  $PRD_j((x-1) \bmod \phi_i + 1)$  tokens to channel  $E_j$  and consumes  $CNS_k((x-1) \bmod \phi_i + 1)$  tokens from channel  $E_k$ .

An important property of the CSDF model that is proven in [3], is that a valid static schedule of a CSDF graph can be generated at design-time if the graph is consistent and live. A CSDF graph  $G$  is said to be consistent if a non-trivial solution exists for the repetition vector  $\vec{q} = [q_1, q_2, \dots, q_n]^T \in \mathbb{N}^n$ . An entry  $q_i \in \vec{q}$  denotes how many times an actor  $A_i \in \mathcal{A}$  has to be executed in every graph iteration of  $G$ . If a deadlock-free schedule can be found,  $G$  is then said to be live. For more details, we refer the reader to [3].

#### C. Strictly Periodic Scheduling of CSDF

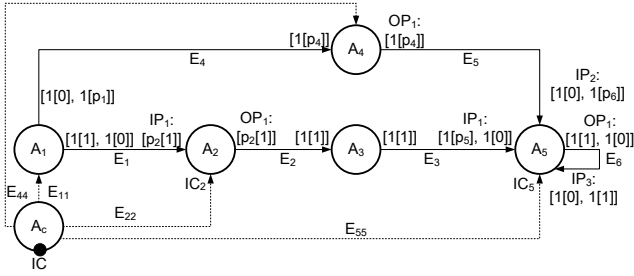
In [14], a real-time strictly periodic scheduling (SPS) framework for CSDF graphs is proposed. In this framework, the actors in a CSDF graph are converted to a set of real-time implicit-deadline periodic tasks. Therefore, such a real-time task corresponding to a CSDF actor is associated with two parameters, namely period  $T$  and earliest starting time  $S$ , where the deadline of the task is equal to its period (i.e., implicit deadline). The minimum period  $T_i$  [14] of any actor  $A_i \in \mathcal{A}$  under SPS can be computed as:

$$T_i = \frac{lcm(\vec{q})}{q_i} \left\lceil \frac{\max_{A_i \in \mathcal{A}} \{\mu_i q_i\}}{lcm(\vec{q})} \right\rceil, \quad (1)$$

where  $q_i$  is the number of repetitions of actor  $A_i$  per graph iteration, and  $\mu_i$  is the worst-case execution time (WCET) of actor  $A_i$ . In general, the derived period vector  $\vec{T}$  must satisfy the condition  $q_1 T_1 = q_2 T_2 = \dots = q_n T_n = H$ , where  $H$  is the iteration period, also called hyper period, that represents the duration needed by the graph to complete one iteration. The minimum period of the sink actor for a CSDF graph determines the maximum throughput that this graph can achieve. In addition, the utilization of any actor  $A_i \in \mathcal{A}$ , denoted by  $u_i$ , can be computed as  $u_i = \mu_i / T_i$ , where  $u_i \in (0, 1]$ .

To sustain a strictly periodic execution with the period derived by Equation (1), the earliest starting time  $S_i$  [14] of any actor  $A_i \in \mathcal{A}$  can be obtained as:

$$S_i = \begin{cases} 0 & \text{if } \text{prec}(A_i) = 0 \\ \max_{A_j \in \text{prec}(A_i)} (S_{j \rightarrow i}) & \text{otherwise,} \end{cases} \quad (2)$$


 Fig. 1. An example of MADF graph ( $G_1$ ).

where  $\text{prec}(A_i)$  represents the set of predecessor actors of  $A_i$  and  $S_{j \rightarrow i}$  is given by:

$$S_{j \rightarrow i} = \min_{t \in [0, S_j + H]} \left\{ t : \begin{array}{l} \text{Prd}_{[t_s, t_e]}(A_j, E_u) \\ \geq \text{Cns}_{[t_s, t_e]}(A_i, E_u), \forall k \in [0, H], k \in \mathbb{N} \end{array} \right\} \quad (3)$$

where  $\text{Prd}_{[t_s, t_e]}(A_j, E_u)$  is the total number of tokens produced by  $A_j$  to edge  $E_u$  during the time interval  $[t_s, t_e]$ ,  $\text{Cns}_{[t_s, t_e]}(A_i, E_u)$  is the total number of tokens consumed by  $A_i$  from edge  $E_u$  during the time interval  $[t_s, t_e]$ , and  $S_j$  is the earliest start time of a predecessor task  $A_j$ .

#### IV. MODE-AWARE DATA FLOW (MADF)

In this section, we introduce our new MoC called Mode-Aware Data Flow (MADF). MADF can capture multiple modes associated with an adaptive streaming application, where each individual mode is a CSDF graph [3]. Details and formal definitions of the MADF model and its operational semantics are given later in this section. Here, we explain the MADF intuitively by an example. Throughout this paper, we use graph  $G_1$  shown in Fig. 1 as the running example to illustrate the definition of MADF and the hard real-time scheduling analysis related to MADF. This graph consists of 5 computation actors  $A_1$  to  $A_5$  that communicate data over edges  $E_1$  to  $E_5$ . Also, there is an extra actor  $A_c$  which controls the switching between modes through control edges  $E_{11}$ ,  $E_{22}$ ,  $E_{44}$ , and  $E_{55}$  at run-time. Each edge contains a production and a consumption pattern, and some of these production and consumption patterns are parameterized. Having different values of parameters and worst-case execution times (WCET) of the actors determine different modes. For example, to specify the consumption pattern with variable length on edge  $E_1$  in graph  $G_1$ , the parameterized notation  $[p_2[1]]$  is used on edge  $E_1$  that is interpreted as a sequence of  $p_2$  elements with integer value 1, e.g.,  $[2[1]] = [1, 1]$ . Similarly, the notation  $[1[p_4]]$  on edge  $E_4$  is interpreted as a sequence of 1 element with integer value  $p_4$ , e.g.,  $[1[2]] = [2]$ . Assume in this particular example that parameter vector  $(p_1, p_2, p_4, p_5, p_6)$  can take only two values  $(0, 2, 0, 2, 0)$  and  $(1, 1, 1, 1, 1)$ . Then,  $A_c$  can switch the application between two corresponding modes  $SI^1$  and  $SI^2$  by setting the parameter vector to value  $(0, 2, 0, 2, 0)$  and  $(1, 1, 1, 1, 1)$ , respectively, at run-time. Fig. 2(a) and (b) show the corresponding CSDF graphs of mode  $SI^1$  and  $SI^2$ .

##### A. Formal Definition of MADF

In this section, we provide a brief formal definition of MADF. The detailed formal definition of MADF can be found in [17].

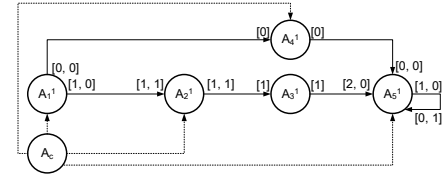
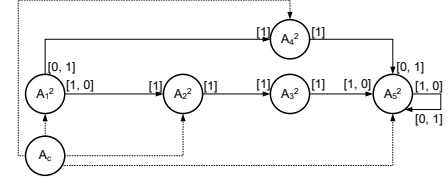

 (a) CSDF graph  $G_1^1$  of mode  $SI^1$ .

 (b) CSDF graph  $G_1^2$  of mode  $SI^2$ .

Fig. 2. Two modes of the MADF graph in Fig. 1.

#### Definition 1 (Mode-Aware Data Flow (MADF)).

A Mode-Aware Data Flow (MADF) is a multi-graph defined by a tuple  $(\mathcal{A}, A_c, \mathcal{E}, \Pi)$ , where

- $\mathcal{A} = \{A_1, \dots, A_{|\mathcal{A}|}\}$  is a set of dataflow actors;
- $A_c$  is the control actor to determine modes and their transitions;
- $\mathcal{E}$  is the set of edges for data/parameter transfer;
- $\Pi = \{\vec{p}_1, \dots, \vec{p}_{|\mathcal{A}|}\}$  is the set of parameter vectors, where each  $\vec{p}_i \in \Pi$  is associated with a dataflow actor  $A_i$ .

**Definition 2 (Dataflow Actor).** A dataflow actor  $A_i$  is described by a tuple  $(\mathcal{I}_i, IC_i, O_i, C_i, M_i)$ , where

- $\mathcal{I}_i = \{IP_1, \dots, IP_{|\mathcal{I}_i|}\}$  is the set of data input ports of actor  $A_i$ ;
- $IC_i$  is the control input port that reads parameter vector  $\vec{p}_i$  for actor  $A_i$ ;
- $O_i = \{OP_1, \dots, OP_{|O_i|}\}$  is the set of data output ports of actor  $A_i$ ;
- $C_i = \{c_1, \dots, c_{|C_i|}\}$  is the set of computations. When actor  $A_i$  fires, it performs a computation  $c_k \in C_i$ ;
- $M_i : \vec{p}_i \rightarrow \{\phi, \bar{C}_i\}$  is a mapping relation, where  $\vec{p}_i \in \Pi$ ,  $\phi \in \mathbb{N}^+$ , and  $\bar{C}_i \subseteq C_i$  is a sequence of computations  $[\bar{C}_i(1), \dots, \bar{C}_i(k), \dots, \bar{C}_i(\phi)]$  with  $\bar{C}_i(k) \in C_i$ ,  $1 \leq k \leq \phi$ .

**Definition 3 (Control Actor).** The control actor  $A_c$  is described by a tuple  $(IC, O_c, \mathcal{S}, \mathcal{M}_c)$ , where

- $\mathcal{S} = \{SI^1, \dots, SI^{|\mathcal{S}|}\}$  is a set of mode identifiers, each of which specifies a unique mode;
- $IC$  is the control input port which is connected to the external environment. Mode identifiers are read through the control input port from the environment;
- $O_c = \{OC_1, \dots, OC_{|\mathcal{A}|}\}$  is a set of control output ports. Parameter vector  $\vec{p}_i$  is sent through  $OC_i \in O_c$  to actor  $A_i$ ;
- $\mathcal{M}_c = \{MC_1, \dots, MC_{|\mathcal{A}|}\}$  is a set of functions defined for each actor  $A_i \in \mathcal{A}$ . For each  $MC_i \in \mathcal{M}_c$ ,  $MC_i : \mathcal{S} \rightarrow \mathbb{N}^{|\vec{p}_i|}$  is a function that takes a mode identifier and outputs a vector of non-negative integer values.

**Definition 4 (Edge).** An edge  $E \in \mathcal{E}$  is defined by a tuple  $((A_i, OP), (A_j, IP))$ , where

- actor  $A_i$  produces a parameterized number of tokens to edge  $E$  through output port  $OP$ ;

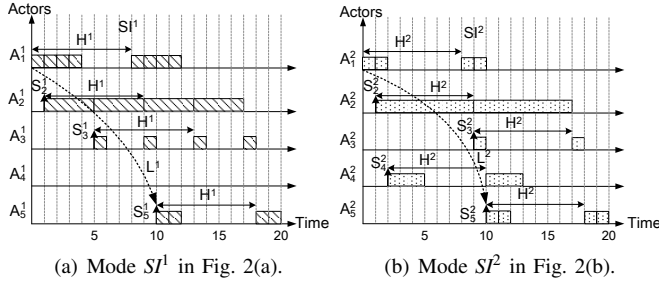


Fig. 3. Execution of two iterations of both modes  $SI^1$  and  $SI^2$  under self-timed scheduling.

- actor  $A_j$  consumes a parameterized number of tokens from  $E$  through input port  $IP$ .

**Definition 5** (Mode of MADF). A mode  $SI^l$  of MADF is a consistent and live CSDF graph, denoted as  $G^l$ , obtained by setting values of  $\Pi$  in Definition 1 as follows:

$$\forall \vec{p}_k \in \Pi : \vec{p}_k = MC_k(SI^l), \quad (4)$$

where function  $MC_k$  is given in Definition 3.

**Definition 6** (Mode of MADF Actor). An actor  $A_k$  in mode  $SI^l$ , denoted by  $A_k^l$ , is a CSDF actor obtained from  $A_k$  as follows:

$$\vec{p}_k = MC_k(SI^l). \quad (5)$$

## B. Operational Semantics

During execution of a MADF graph, it can be either in a steady-state or mode transition.

**Definition 7** (Steady-state). A MADF graph is in a steady-state of a mode  $SI^l$ , if it satisfies Equation (4) with the same  $SI^l$  for all its actors.

**Definition 8** (Mode Transition). A MADF graph is in a mode transition from mode  $SI^o$  to  $SI^l$ , where  $o \neq l$ , if some actors have  $SI^o$  for Equation (5) and the remaining active actors (Definition 9 in [17]) have  $SI^l$  for Equation (5).

In the steady-state of a MADF graph, all active actors execute in the same mode. As defined previously in Definition 5 and shown in Fig. 2(a) and Fig. 2(b), the steady-state of the MADF graph has the same operational semantics as a CSDF graph. We use  $\langle A_i^k, x \rangle$  to denote the  $x$ -th firing of actor  $A_i$  in mode  $SI^k$ . At  $\langle A_i^k, x \rangle$ , it executes computation  $\bar{C}_i(((x-1) \bmod \phi) + 1)$ , where  $\bar{C}_i$  is given in Definition 2. The number of tokens consumed and produced are specified according to Definitions 4 and 5 in [17], respectively. For instance, the  $x$ -th firing of  $A_i^k$  produces  $PRD(((x-1) \bmod \phi) + 1)$  tokens through an output port  $OP$ . In each mode  $SI^k$ , the MADF graph is a consistent and live CSDF graph and thus has the notion of graph iterations with a non-trivial repetition vector  $\vec{q}^k \in \mathbb{N}^{|\mathcal{A}|}$ . Next, we further define mode iterations.

**Definition 9** (Mode Iteration). One iteration  $It^k$  of a MADF graph in mode  $SI^k$  consists of one firing of control actor  $A_c$  and  $q_i^k \in \vec{q}^k$  firings of each MADF actor  $A_i^k$ .

Consider the two modes shown in Fig. 2(a) and Fig. 2(b). Repetition vectors  $\vec{q}^1$  and  $\vec{q}^2$  are:

$$\vec{q}^1 = [4, 2, 2, 0, 2], \quad \vec{q}^2 = [2, 1, 1, 1, 2]. \quad (6)$$

TABLE I  
ACTOR PARAMETER FOR  $G_1$  IN FIG. 1.

Mode	$SI^1$					$SI^2$				
	$A_1^1$	$A_2^1$	$A_3^1$	$A_4^1$	$A_5^1$	$A_1^2$	$A_2^2$	$A_3^2$	$A_4^2$	$A_5^2$
WCET ( $\mu_i$ )	1	4	1	1	1	8	1	3	1	
period ( $T_i$ )	2	4	4	4	4	8	8	8	8	4
starting time ( $S_i$ )	0	2	6	14	0	4	12	8	8	20
utilization ( $u_i$ )	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	1	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	

For any mode of a MADF graph, i.e., a live CSDF graph, under *any* valid schedule, it has (eventually) periodic execution in time. This holds for CSDF graphs under self-timed schedule [18], K-periodic schedule [19], and SPS [14]. The length of the periodic execution, called *iteration period*, determines the minimum time interval to complete one graph iteration (cf. Definition 9). The iteration period, denoted by  $H^k$ , is equal for any actor in the same mode  $SI^k$ . During a periodic execution, the starting time of each actor  $A_i^k$ , denoted by  $S_i^k$ , indicates the time distance between the start of source actor  $A_{src}^k$  and the start of actor  $A_i^k$  in the same iteration period. Based on the notion of starting times, we define *iteration latency*  $L^k$  of a MADF graph in mode  $SI^k$  as follows:

$$L^k = S_{snk}^k - S_{src}^k, \quad (7)$$

where  $S_{snk}^k$  and  $S_{src}^k$  are the earliest starting times of the sink and source actors, respectively. Fig. 3 illustrates the execution of both modes  $SI^1$  and  $SI^2$  given in Fig. 2 under the self-timed schedule. A rectangle denotes the WCET of an actor firing. The WCETs of all actors in both modes are given in the third row of Table I. Now, it can be seen in Fig. 3 that iteration period  $H^1 = H^2 = 8$ . Based on the starting time of each actor, we obtain iteration latencies  $L^1 = S_5^1 - S_1^1 = 10 - 0 = 10$  and  $L^2 = S_5^2 - S_1^2 = 10 - 0 = 10$  as shown in Fig. 3.

## C. Mode Transition

While the operational semantics of a MADF graph in steady-state are the same as that of a CSDF graph, the transition of MADF graph from one mode to another is the crucial part that makes it fundamentally different from CSDF. The protocol for mode transitions has strong impact on the compile-time analyzability and implementation efficiency. In this section, we propose a novel and efficient protocol of mode transitions for MADF graphs.

During execution of a MADF graph, mode transitions may be triggered at run-time by receiving a Mode Change Request (MCR) from the external environment. We first assume that a MCR can be only accepted in the steady-state of a MADF graph, not in an ongoing mode transition. This means that any MCR occurred during an ongoing mode transition will be ignored. Consider a mode transition from  $SI^o$  to  $SI^l$ . The transition is accomplished by the control actor reading mode identifier  $SI^l$  from its control input port (see the black dot in Fig. 1) and writing parameter values of  $\vec{p}_i$  to the control output port connected to each dataflow actor  $A_i^l$  according to function  $MC_i$  given in Definition 3. Then,  $A_i^l$  reads new parameter values  $\vec{p}_i$  from its control input port and sets the sequence of computations according to mapping relation  $M_i$  in

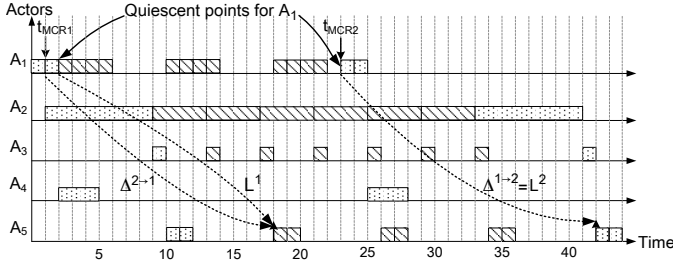


Fig. 4. An execution of  $G_1$  in Fig. 1 with two mode transitions under the ST transition protocol.  $MCR1$  at time  $t_{MCR1}$  denotes a transition request from mode  $S^2$  to  $S^1$ , and  $MCR2$  at time  $t_{MCR2}$  denotes a transition request from mode  $S^1$  to  $S^2$ .

**Definition 2.** The production and consumption sequences are specified according to Definitions 4 and 5 in [17], respectively. We further define/require that mode transitions are only allowed at quiescent points [20].

**Definition 10** (Quiescent Point of MADF). *For mode  $S^l$ , a quiescent point of MADF actor  $A_i$  is firing  $\langle A_i^l, x \rangle$  in mode iteration  $I^l$  that satisfies*

$$\neg \exists \langle A_i^l, y \rangle \in I^l : y < x. \quad (8)$$

Definition 10 simply refers to the first firing of actor  $A_i$  in each iteration  $I^l$  of mode  $S^l$ . Recall that each iteration of mode  $S^l$  consists of  $q_i^l$  firings of actor  $A_i$ . Therefore, our requirement that a mode transition is only allowed at a quiescent point implies that a transition from mode  $S^l$  to  $S^o$  of actor  $A_i$  happens when all firings of actor  $A_i$  are completed in the iteration of  $S^l$  when MCR occurs. Fig. 4 shows an execution of  $G_1$  in Fig. 1 with two mode transitions. For instance, the MCR at time  $t_{MCR1} = 1$  denotes a transition request from mode  $S^2$  to  $S^1$ . The mode transition of actor  $A_1$  happens when all firings of actor  $A_1$  are completed, that is at time 2 in Fig. 4 in this particular example.

Definition 10 defines mode transitions of MADF graphs as partially ordered actor firings. However, it does not specify at which time instance a mode transition actually starts. Therefore, below, we focus on the transition protocol that defines the points in time for occurrences of mode transitions. To quantify the transition protocol, we introduce a metric, called *transition delay*, to measure the responsiveness of a protocol to a MCR.

**Definition 11** (Transition Delay). *For a MCR at time  $t_{MCR}$  calling for a mode transition from mode  $S^o$  to  $S^l$ , the transition delay  $\Delta^{o \rightarrow l}$  of a MADF graph is defined as*

$$\Delta^{o \rightarrow l} = \sigma_{snk}^{o \rightarrow l} - t_{MCR}, \quad (9)$$

where  $\sigma_{snk}^{o \rightarrow l}$  is the earliest starting time of the sink actor in the new mode  $S^l$ .

In Fig. 4, we can compute the transition delay for  $MCR1$  occurred at time  $t_{MCR1} = 1$  as  $\Delta^{2 \rightarrow 1} = 18 - 1 = 17$ .

1) *Self-timed Transition Protocol:* In the existing adaptive MoCs like FSM-SADF [5], a protocol, referred here as *Self-Timed* (ST) transition protocol, is adopted. The ST protocol specifies that actors are scheduled in the self-timed manner not only in the steady-state, but also during a mode transition. For FSM-SADF upon a MCR, a firing of a FSM-SADF actor in

the new mode can start immediately after the firing of the actor completes the old mode iteration. The only possible delay is introduced due to availability of input data. One reason behind the ST protocol is that the ST schedule for a (C)SDF graph (steady-state of FSM-SADF<sup>1</sup>) leads to its highest achievable throughput. However, the ST protocol generally introduces interference of one mode execution with another one. The time needed to complete mode transitions also fluctuates as the transition delay of an ongoing transition depends on the transitions that occurred in the past. We consider this as an undesired effect because mode transitions using the ST protocol become potentially slow and unpredictable. Another consequence of the incurred interference between modes using the ST transition protocol is the high time complexity of analyzing transition delays, because transition delays cannot be analyzed independently for each mode transition. The analysis proposed in [5] uses an approach based on state-space exploration, which has the exponential time complexity.

Consider  $G_1$  in Fig. 1 and an execution of  $G_1$  with the two mode transitions illustrated in Fig. 4. The execution is assumed under the ST schedule for both steady-state and mode transitions of  $G_1$ . After  $MCR1$  at time  $t_{MCR1}$ , the transition from mode  $S^2$  to  $S^1$  introduces interference to execution of the new mode  $S^1$  from execution of the old mode  $S^2$ . The interference increases the iteration latency of the new mode  $S^1$  to  $L^1 = S_5^1 - S_1^1 = 18 - 2 = 16$  from initially 10 as shown in Fig. 3(a) when  $G_1$  is only executed in the steady-state of mode  $S^1$ . Even worse, the interference is further propagated to the second mode transition after  $MCR2$  at time  $t_{MCR2}$ . In this case, the iteration latency  $L^2 = S_5^2 - S_1^2 = 42 - 23 = 19$  is increased from initially 10 as shown in Fig. 3(b) when  $G_1$  is only executed in the steady-state of mode  $S^2$ . This example thus clearly shows the problem of the ST protocol. That is, it introduces interference between the old and new modes due to mode transitions, thereby increasing the iteration latency of the new mode in the steady-state after the transition. Furthermore, the increase of iteration latency also potentially increases transition delays as it will be shown in the next section.

2) *Maximum-Overlap Offset Transition Protocol:* To address the problem of the ST transition protocol explained above, we propose a new transition protocol, called *Maximum-Overlap Offset* (MOO).

**Definition 12** (Maximum-Overlap Offset (MOO)). *For a MADF graph and a transition from mode  $S^o$  to  $S^l$ , Maximum-Overlap Offset (MOO), denoted by  $x$ , is defined as*

$$x = \begin{cases} \max_{A_i \in \mathcal{A}^o \cap \mathcal{A}^l} (S_i^o - S_i^l) & \text{if } \max_{A_i \in \mathcal{A}^o \cap \mathcal{A}^l} (S_i^o - S_i^l) > 0 \\ 0 & \text{otherwise,} \end{cases} \quad (10)$$

where  $\mathcal{A}^o \cap \mathcal{A}^l$  is set of actors active in both modes  $S^o$  and  $S^l$ .

Basically, we first assume that the new mode  $S^l$  starts immediately after the source actor  $A_{src}^o$  of the old mode  $S^o$  completes its last iteration  $I^o$ . All actors  $A_i^l$  of the new mode

<sup>1</sup>The steady-state of SADF is defined similarly to that of MADF. The only difference is that a scenario of FSM-SADF is a SDF graph, whereas a mode of MADF is a CSDF graph.

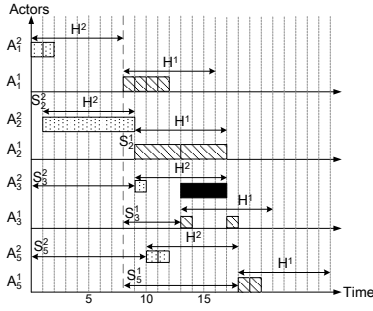


Fig. 5. An illustration of the Maximum-Overlap Offset (MOO) calculation.

execute according to the earliest starting times  $S_i^l$  and iteration period  $H^l$  in the steady-state. Under this assumption, if the execution of the new mode overlaps with the execution of the old mode in terms of iteration periods  $H^o$  and  $H^l$ , we then need to offset the starting time of the new mode by the maximum overlap among all actors. In this way, the execution of the new mode will have the same iteration latency as that of the new mode in the steady-state, i.e., no interference between the execution of both old and new modes.

Consider  $MCR1$  at time  $t_{MCR1}$  shown in Fig. 4. Obtaining MOO  $x$  is illustrated in Fig. 5. We first assume that the new mode  $SI^1$  starts at the time when the source actor  $A_1^2$  completes the last iteration at time 8 (see bold, dashed line in Fig. 5). Actors  $A_i^1$  in the new mode start as if they executed in the steady-state of mode  $SI^1$ . Then, we can see that, for actor  $A_3$ , the execution of  $A_3^1$  in the new mode  $SI^1$  according to  $S_3^1$  in Fig. 3(a) overlaps 4 time units (solid bar in Fig. 5) with the execution of  $A_3^2$  in the old mode  $SI^2$  in terms of iteration periods  $H^2$  and  $H^1$ . This is also the maximum overlap between the execution of actors in modes  $SI^2$  and  $SI^1$ . According to Definition 12,  $x$  can be obtained through the following equations:  $S_1^2 - S_1^1 = 0 - 0 = 0$ ,  $S_2^2 - S_2^1 = 1 - 1 = 0$ ,  $S_3^2 - S_3^1 = 9 - 5 = 4$ ,  $S_5^2 - S_5^1 = 10 - 10 = 0$ . Therefore, it results in an offset  $x = \max(0, 0, 4, 0) = 4$  to the start of mode  $SI^1$  and is shown in Fig. 6. The starting time of the new mode  $SI^1$ , namely the source actor  $A_1^1$ , must be first delayed to the time when  $A_2^1$  completes the iteration period  $H^2$  in the last iteration, namely time 8 shown as the first bold dashed line in Fig. 6. In addition, the MOO  $x = 4$  must be further added to the starting time of  $A_1^1$  (the second bold dashed line in Fig. 6). Fig. 6 also shows another transition from mode  $SI^1$  to  $SI^2$  with a MCR occurred at time  $t_{MCR2} = 23$ . The starting time of the source actor  $A_2^1$  in the new mode  $SI^2$  must be first delayed to the time 28 (the third bold dashed line in Fig. 6), namely the time when  $A_1^1$  completes the last iteration in the old mode  $SI^1$ . To calculate the MOO  $x$  for this transition, the following equations hold:  $S_1^1 - S_1^2 = 0 - 0 = 0$ ,  $S_2^1 - S_2^2 = 1 - 1 = 0$ ,  $S_3^1 - S_3^2 = 5 - 9 = -4$ ,  $S_5^1 - S_5^2 = 10 - 10 = 0$ . Thus, the equations above result in  $x = \max(0, 0, -4, 0) = 0$ . For this transition, the new mode  $SI^2$  starts at time 28 as shown in Fig. 6.

The MOO protocol offers several advantages over the ST protocol. Essentially, the MOO protocol retains the iteration latency of the MADF graph in the new mode the same as the initial value, thereby avoiding the interference between the old and new modes. For instance, after  $MCR1$  and  $MCR2$  in

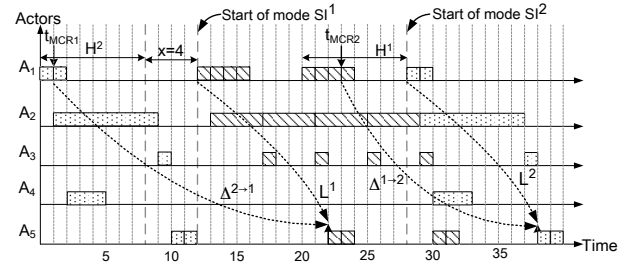


Fig. 6. The execution of  $G_1$  with two mode transitions under Maximum-Overlap Offset (MOO) protocol.

Fig. 6, mode  $SI^1$  and  $SI^2$  still have the initial iteration latency  $L^1 = 10$  and  $L^2 = 10$  as shown in Fig. 3. Therefore, efficiently computing the starting time of MADF actors in the new mode becomes feasible and it plays an important role in deriving a hard-real time schedule for the MADF actors. As a result, analysis of the worst-case transition delay is much simpler (see Theorem 1) than that of the ST protocol, because the transition delay does not depend on the order of the transitions that occurred previously.

Concerning the transition delay, it may be the case that the MOO protocol results in initially longer transition delay than the ST protocol does due to the offset given in Definition 12. For  $MCR1$  occurred at time  $t_{MCR1}$ , the transition delay of the MOO protocol is  $\Delta^{2 \rightarrow 1} = 22 - 1 = 21$  as shown in Fig. 6, whereas the transition delay of the ST protocol is equal to  $\Delta^{2 \rightarrow 1} = 18 - 1 = 17$  as shown in Fig. 4. On the other hand, let us consider the same transition request  $MCR2$  occurred at time  $t_{MCR2} = 23$  shown in Fig. 4 and Fig. 6. For  $MCR2$ , the ST protocol results in transition delay  $\Delta^{1 \rightarrow 2} = 42 - 23 = 19$  as shown in Fig. 4. In contrast, the transition delay for the MOO protocol is  $\Delta^{1 \rightarrow 2} = 38 - 23 = 16$  as shown in Fig. 6. The MOO protocol could provide shorter transition delay than the ST protocol, thereby faster responsiveness to a mode transition.

## V. HARD REAL-TIME ANALYSIS AND SCHEDULING OF MADF

Based on the proposed MOO protocol for mode transitions, in this section, we propose a hard real-time analysis and scheduling framework for MADF. More specifically, we propose an analysis technique for mode transitions in MADF to reason about transition delays, such that timing constraints can be guaranteed. The hard real-time scheduling framework for MADF graphs is an extension of the SPS [14] framework initially developed for CSDF graphs.

As explained in Section III-C, the key concept of the SPS framework is to derive a periodic taskset representation for a CSDF graph. Since the steady-state of a mode can be considered as a CSDF graph according to Definitions 5 and 7, it is thus straightforward to represent the steady-state of a MADF graph as a periodic taskset and schedule the resulting taskset using any well-known hard real-time scheduling algorithm. Using the SPS framework, we can derive the two main parameters for each MADF actor in mode  $SI^k$ , namely the period ( $T_i^k$  in Equation (1)) and the earliest starting time ( $S_i^k$  in Equation (2)). Under SPS, the iteration period in mode  $SI^k$  is obtained as  $H^k = q_i^k T_i^k$ ,  $\exists A_i^k \in \mathcal{A}$ . Below, we focus on determining the earliest starting time of each actor in the new mode upon

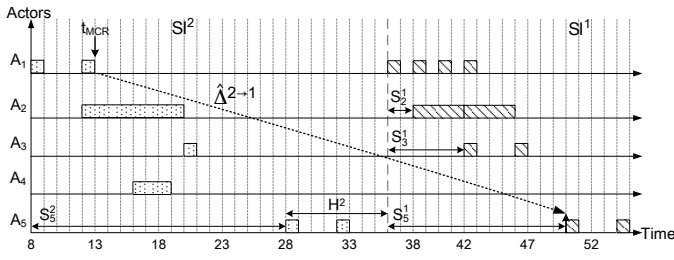


Fig. 7. Upper bounds of earliest starting times for transition from mode  $SI^2$  to  $SI^1$ .

a transition. From the earliest starting time, we can reason about the transition delay to quantify the responsiveness of a transition.

Upon a MCR, a MADF graph can safely switch to the new mode if all of its actors have completed their last iteration in the old mode upon synchronous protocol. In this case, the firings of MADF actors in the new mode do not overlap with the firings of actors in the old mode. This is called synchronous protocol [12] in real-time systems with mode change. One of its advantages is the simplicity, i.e., the synchronous protocol does not require any schedulability test at both compile-time and run-time. However, other protocols lead to earlier starting times than the synchronous protocol. Therefore, the synchronous protocol sets an upper bound on the earliest starting time for each MADF actor in the new mode.

**Lemma 1.** For a MADF graph  $G$  under SPS and a MCR from mode  $SI^o$  to  $SI^l$  at time  $t_{MCR}$ , the earliest starting time of actor  $A_i^l$ ,  $\hat{\sigma}_i^{o \rightarrow l}$ , is upper bounded by

$$\hat{\sigma}_i^{o \rightarrow l} = F_{src}^o + S_{snk}^o + S_i^l, \quad (11)$$

where  $F_{src}^o$  indicates the time when the source actor  $A_{src}^o$  completes its last iteration  $I^o$  of the old mode  $SI^o$  and is given by

$$F_{src}^o = t_S^o + \left\lceil \frac{t_{MCR} - t_S^o}{H^o} \right\rceil H^o. \quad (12)$$

$t_S^o$  is the starting time of mode  $SI^o$  and  $H^o$  is the iteration period of mode  $SI^o$ .

*Proof.* The proof can be found in [17], [21].  $\square$

Let us consider the actor parameters given in Table I for  $G_1$  in Fig. 1. The third row shows the WCET for each actor in modes  $SI^l$  and  $SI^2$ . Based on WCETs, the period (fourth row in Table I) and the earliest starting time (fifth row in Table I) for each actor in the steady-state of both modes are obtained according to Equation (1) and Equation (2), respectively. Given  $\bar{q}^2$  in Equation (6), we can also compute iteration period  $H^2 = q_1^2 T_1^2 = 2 \times 4 = 8$ . Now consider the mode transition from mode  $SI^2$  to  $SI^1$  shown in Fig. 7. Assume that the MCR occurs at time  $t_{MCR} = 13$  and mode  $SI^2$  starts at time  $t_S^2 = 8$ . The completion time of the last iteration  $I^2$  is equal to the completion time of the sink actor  $A_5^2$  computed as  $F_{snk}^2 = t_S^2 + \left\lceil \frac{t_{MCR} - t_S^2}{H^2} \right\rceil H^2 + S_5^2 = 8 + \left\lceil \frac{13-8}{8} \right\rceil 8 + 20 = 36$ . In Fig. 7,  $F_{snk}^2$  corresponds to the earliest starting time of the source actor  $A_1^1$  (bold dashed line). Finally, we can compute the earliest starting time for each actor in the new mode  $SI^1$  by adding  $S_i^1$ . Considering for

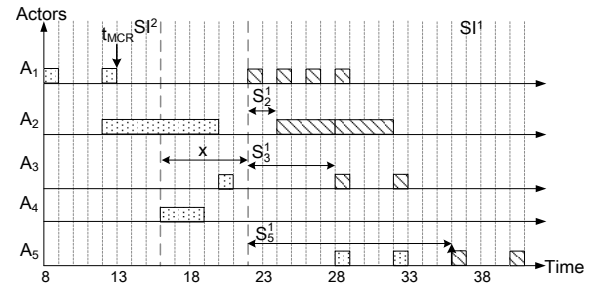


Fig. 8. Earliest starting times for transition from mode  $SI^2$  to  $SI^1$  with the MOO protocol.

instance the sink actor  $A_5^1$  in the new mode with  $S_5^1 = 14$ , the upper bound of its earliest starting time can be obtained as  $\hat{\sigma}_5^{2 \rightarrow 1} = F_{src}^2 + S_5^2 + S_5^1 = F_{snk}^2 + S_5^1 = 36 + 14 = 50$ . We can thus compute the transition delay (cf. Definition 11) as  $\hat{\Delta}^{2 \rightarrow 1} = \hat{\sigma}_5^{2 \rightarrow 1} - t_{MCR} = 50 - 13 = 37$ .

Although the upper bound of the earliest starting times is easy to obtain for MADF actors in the new mode, it does not provide a responsive mode transition. Therefore, here we aim at deriving a lower bound of the earliest starting times with the proposed MOO protocol.

**Lemma 2.** For a MADF graph under SPS and a MCR from mode  $SI^o$  to  $SI^l$  at time  $t_{MCR}$ , the earliest starting time of actor  $A_i^l$  using the MOO protocol is lower bounded by  $\check{\sigma}_i^{o \rightarrow l}$  given as

$$\check{\sigma}_i^{o \rightarrow l} = F_{src}^o + x + S_i^l, \quad (13)$$

where  $F_{src}^o$  is given in Equation (12) and  $x$  is given in Equation (10).

*Proof.* The proof can be found in [17], [21].  $\square$

Let us consider again the transition from mode  $SI^2$  to  $SI^1$ . With the MOO protocol, the mode transition is illustrated in Fig. 8. Upon the MCR at time  $t_{MCR} = 13$  and  $t_S^2 = 8$ , source actor  $A_1^2$  completes its last iteration  $I^2$  in the old mode  $SI^2$  at the time (cf. Equation (12)) given as  $F_{src}^2 = F_1^2 = t_S^2 + \left\lceil \frac{t_{MCR} - t_S^2}{H^2} \right\rceil H^2 = 8 + \left\lceil \frac{13-8}{8} \right\rceil 8 = 16$ . This is the earliest possible time at which mode transition is allowed. For MOO,  $x$  can be computed according to Equation (10). Therefore, the following equations hold:  $S_1^2 - S_1^1 = 0 - 0 = 0$ ,  $S_2^2 - S_2^1 = 4 - 2 = 2$ ,  $S_3^2 - S_3^1 = 12 - 6 = 6$ ,  $S_5^2 - S_5^1 = 20 - 14 = 6$ . It thus yields  $x = \max(0, 2, 6, 6) = 6$ , i.e., an offset  $x = 6$  is added to  $F_{src}^2$ . It can be seen in Fig. 8 that the source actor  $A_1^1$  starts at time  $F_{src}^2 + x = 16 + 6 = 22$ . Finally, the earliest starting times of actors in mode  $SI^1$  can be determined by adding  $S_i^1$ . Considering for instance  $A_5^1$  in the new mode, the lower bound of its earliest starting time can be obtained as  $\check{\sigma}_5^{2 \rightarrow 1} = F_{src}^2 + x + S_5^1 = 16 + 6 + 14 = 36$ . Now, the transition delay (cf. Definition 11) can be obtained as  $\hat{\Delta}^{2 \rightarrow 1} = \check{\sigma}_5^{2 \rightarrow 1} - t_{MCR} = 36 - 13 = 23$ .

#### A. Scheduling Analysis under a Fixed Allocation of Actors

During a mode transition of a MADF graph according to the MOO protocol, actors execute simultaneously in the old and new modes. The derived starting time in Lemma 2 for each actor is only the lower bound because the allocation of actors on PEs



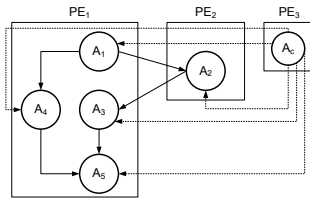


Fig. 9. Allocation of all MADF actors in Fig. 1 to 3 PEs.

is not taken into account yet. That means, the derived starting times according to Lemma 2 can be only achieved during mode transitions when each actor is allocated to a separate PE. In a practical system where multiple actors are allocated to the same PE, the PE may be potentially overloaded during mode transitions. To avoid overloading of PEs, the earliest starting times of actors may be further delayed.

**Lemma 3.** For a MADF graph under SPS, a MCR from mode  $SI^o$  to  $SI^l$ , and a  $m$ -partition of all actors  $\Psi = \{\Psi_1, \dots, \Psi_m\}$ , where  $m$  is the number of PEs, the earliest starting time of an actor  $A_i^l$  without overloading the underlying PE is given by

$$\sigma_i^{o \rightarrow l} = F_{src}^o + \delta^{o \rightarrow l} + S_i^l, \quad (14)$$

where  $F_{src}^o$  is computed by Equation (12) and  $\delta^{o \rightarrow l}$  is obtained as

$$\delta^{o \rightarrow l} = \min_{t \in [t, S_{snk}^o]} \{t : U_j(k) \leq UB, \forall k \in [t, S_{snk}^o] \wedge \forall \Psi_j \in \Psi\}. \quad (15)$$

$UB$  denotes the utilization bound of the scheduling algorithm used to schedule actors on each PE.  $\Psi_j$  contains the set of actors allocated to  $PE_j$ .  $U_j(k)$  is the total utilization of  $PE_j$  at time  $k$  demanded by both mode  $SI^o$  and  $SI^l$  actors, and is given by

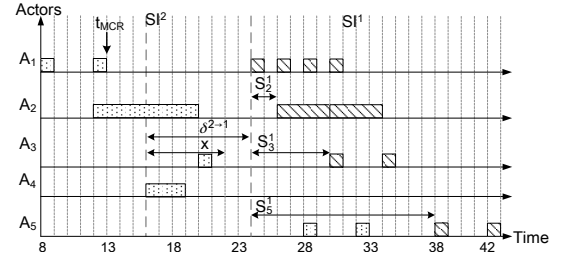
$$U_j(k) = \underbrace{\sum_{A_d^o \in \Psi_j} (u_d^o - h(k - S_d^o) \cdot u_d^o)}_{U_j^o(k)} + \underbrace{\sum_{A_d^l \in \Psi_j} (h(k - S_d^l - t) \cdot u_d^l)}_{U_j^l(k)}, \quad (16)$$

$A_d^o \in \Psi_j$  is an actor active in the old mode  $SI^o$  and allocated to  $PE_j$ .  $A_d^l \in \Psi_j$  is an actor active in the new mode  $SI^l$  and allocated to  $PE_j$ .  $h(t)$  is the Heaviside step function.

*Proof.* The proof can be found in [17], [21].  $\square$

Fig. 9 shows all actors of  $G_1$  in Fig. 1 allocated to 3 PEs and let us assume that the actors allocated to each PE are scheduled using the EDF scheduling algorithm [16]. The utilization bound of EDF is given in [16] as  $UB = 1$ . Given this allocation and the transition from mode  $SI^2$  to  $SI^1$  shown in Fig. 8, the lower bound of the earliest starting time  $\check{\sigma}_1^{2 \rightarrow 1} = 22$  for actor  $A_1^1$  cannot be achieved. At time 22, only actor  $A_1^1$  has completed the last iteration  $I^2$  on  $PE_1$ . Starting the new mode  $SI^1$  at time 22 corresponds to  $\delta^{2 \rightarrow 1} = x = 6$ . The total utilization of  $PE_1$  demanded by the actors in the old mode  $SI^2$  at time 22, i.e.,  $U_1^2(6)$ , can be computed as follows:

$$\begin{aligned} U_1^2(6) &= \sum_{A_d^2 \in \Psi_1} u_d^2 - h(6 - S_d^2) \cdot u_d^2, \quad d \in \{1, 3, 4, 5\} \\ &= u_1^2 - h(6) \cdot u_1^2 + u_3^2 - h(-6) \cdot u_3^2 + u_4^2 - h(-2) \cdot u_4^2 + u_5^2 - h(-14) \cdot u_5^2 \\ &= 0 + u_3^2 + u_4^2 + u_5^2 = \frac{1}{8} + \frac{3}{8} + \frac{1}{4} = \frac{3}{4}. \end{aligned}$$


 Fig. 10. Earliest starting times for transition  $SI^2$  to  $SI^1$  on 3 PEs shown in Fig. 9.

Enabling  $A_1^1$  in the new mode  $SI^1$  at time 22 would yield  $U_1(6) = U_1^2(6) + u_1^1 = \frac{3}{4} + \frac{1}{2} > UB = 1$ , thereby leading to being unschedulable on  $PE_1$ . In this case, the earliest starting times of all actors in mode  $SI^1$  must be delayed by  $\delta^{2 \rightarrow 1} = 8$  to time 24 as shown in Fig. 10. At time 24, the total utilization demanded by mode  $SI^2$  actors is

$$\begin{aligned} U_1^2(8) &= \sum_{A_d^2 \in \Psi_1} u_d^2 - h(8 - S_d^2) \cdot u_d^2, \quad d \in \{1, 3, 4, 5\} \\ &= u_1^2 - h(8) \cdot u_1^2 + u_3^2 - h(-4) \cdot u_3^2 + u_4^2 - h(0) \cdot u_4^2 + u_5^2 - h(-12) \cdot u_5^2 \\ &= 0 + u_3^2 + 0 + u_5^2 = \frac{1}{8} + \frac{1}{4} = \frac{3}{8}. \end{aligned}$$

Now, enabling  $A_1^1$  in the new mode at time 24 results in the total utilization of  $PE_1$  as  $U_1(8) = U_1^2(8) + u_1^1 = \frac{3}{8} + \frac{1}{2} < 1$ . Next, assuming that the new mode  $SI^1$  starts at time 24, we need to check that the remaining actors in the new mode  $SI^1$ , namely  $A_3^1$  and  $A_5^1$ , can start with  $S_3^1$  and  $S_5^1$  respectively without overloading  $PE_1$ . For instance, enabling  $A_3^1$  at time 24 results in starting time  $\sigma_3^{2 \rightarrow 1} = 24 + S_3^1 = 24 + 6 = 30$ . At time 30, the total utilization of  $PE_1$  can be obtained according to Equation (16) as follows:

$$\begin{aligned} U_1^2(8+6) &= \sum_{A_d^2 \in \Psi_1} u_d^2 - h(14 - S_d^2) \cdot u_d^2, \quad d \in \{1, 3, 4, 5\} \\ &= u_1^2 - h(14) \cdot u_1^2 + u_3^2 - h(2) \cdot u_3^2 + u_4^2 - h(6) \cdot u_4^2 + u_5^2 - h(-6) \cdot u_5^2 \\ &= 0 + 0 + 0 + u_5^2 = \frac{1}{4}, \\ U_1^1(8+6) &= \sum_{A_d^1 \in \Psi_1} (h(14 - S_d^1 - 8) \cdot u_d^1), \quad d \in \{1, 3, 5\} \\ &= h(6)u_1^1 + h(0)u_3^1 + h(-8)u_5^1 = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}, \\ U_1(8+6) &= U_1^2(8+6) + U_1^1(8+6) = 1 = UB. \end{aligned}$$

Hence, actors  $A_5^2$ ,  $A_1^1$ , and  $A_3^1$  are schedulable on  $PE_1$  using EDF. Similarly, starting  $A_5^1$  at time  $\sigma_5^{2 \rightarrow 1} = 24 + S_5^1 = 38$  still keeps the resulting set of actors schedulable on  $PE_1$ .

Using Lemma 3, we can quantify the maximum and minimum transition delays for any transition from mode  $SI^o$  to  $SI^l$ .

**Theorem 1.** For a MADF graph under SPS, a fixed allocation of all MADF actors  $\Psi = \{\Psi_1, \dots, \Psi_m\}$  to  $m$  PEs, and a MCR from mode  $SI^o$  to  $SI^l$ , the minimum transition delay is given by

$$\Delta_{min}^{o \rightarrow l} = \delta^{o \rightarrow l} + S_{snk}^l \quad (17)$$

and the maximum transition delay is given by

$$\Delta_{max}^{o \rightarrow l} = \delta^{o \rightarrow l} + S_{snk}^l + H^o, \quad (18)$$

where  $\delta^{o \rightarrow l}$  is computed by Lemma 3,  $S_{snk}^l$  is the starting time of the sink actor in the new mode  $SI^l$ , and  $H^o$  is the iteration period of the old mode  $SI^o$ .

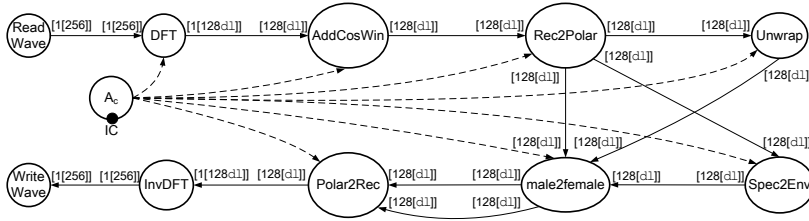


Fig. 11. MADF graph of Vocoder.

TABLE II  
WCETs OF ALL ACTORS IN VOCODER (IN CLK. CYCLES).

Mode	ReadWave	DFT	AddCosWin	Rec2Polar	Unwrap	Spec2Env	male2female	Polar2Rec	InvDFT	WriteWave
$SI^8$	3704	16775	16	90	359	7168	1093	3	236	3660
$SI^{16}$	3704	35121	35	183	691	1163	138	260	644	3660
$SI^{32}$	3704	71337	75	366	1393	1392	210	507	988	3660
$SI^{64}$	3704	144531	150	1156	2346	1696	426	1056	3630	3660

*Proof.* The proof can be found in [17], [21].  $\square$

It can be seen from Theorem 1 that the maximum and minimum transition delays solely depend on the allocation of MADF actors and the old and new modes in question, irrespective of the previously occurred transitions. The old and new modes determine  $H^o$  and  $S^l_{snk}$ , respectively, while the allocation of MADF actors determines the value of  $\delta^{o \rightarrow l}$ . Here, the offset  $x$  due to our MOO protocol is captured in  $\delta^{o \rightarrow l}$  and can be considered as performance overhead if  $x \neq 0$ . The other parts, namely  $H^o$  and  $S^l_{snk}$ , in the maximum and minimum transition delays cannot be avoided as they will be present in any transition protocol.

## VI. CASE STUDIES

To evaluate our proposed MADF MoC and MOO protocol, in this section, we present two case studies. In the first case study, we model a real-life adaptive streaming application, called Vocoder, with our MADF MoC proposed in Section IV and apply the hard real-time analysis proposed in Section V. With this case study, we show that the MADF MoC is capable of capturing different application modes and the transitions between them. Then, in the second case study, we model another real-life adaptive streaming application, called MP3decoder, with MADF and we focus on analyzing the transition delays and demonstrating the effectiveness of our MADF model armed with the proposed MOO transition protocol compared to the well-known FSM-SADF model [5] which also can capture modes/scenarios. In this case study, we adopt self-timed scheduling for both our MADF and FSM-SADF models in the steady-state. The major difference between these models in this case study is their transition protocol which is the MOO protocol in our MADF model and the self-timed protocol in FSM-SADF. Another example of the application of our MOO protocol can be found in [22].

### A. Case Study 1

In this section, we consider a real-life adaptive application from the StreamIT benchmark suit [23], called Vocoder,

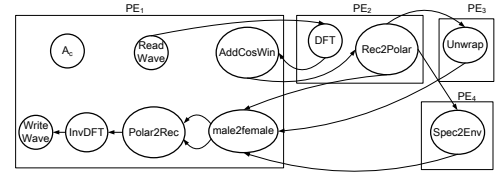


Fig. 12. Allocation of dataflow actors of Vocoder to 4 PEs. The control edges are omitted to avoid cluttering.

which implements a phase voice encoder and performs pitch transposition of recorded sounds from male to female. We modeled Vocoder using a MADF graph with 4 modes, which capture different workloads. The MADF graph of Vocoder is shown in Fig. 11. Depending on the desired quality of audio encoding and various performance requirements, the resource manager as a middle-ware or OS-like component for the MPSoC may switch between four different modes of Vocoder at run-time. The four modes  $S = \{SI^8, SI^{16}, SI^{32}, SI^{64}\}$  specify different lengths of the Discrete Fourier Transform (DFT), denoted by  $dl \in \{8, 16, 32, 64\}$ . Mode  $SI^8$  ( $dl = 8$ ) requires the least amount of computation at the cost of the worst voice encoding quality among all DFT lengths. Mode  $SI^{64}$  ( $dl = 64$ ) produces the best quality of voice encoding among all modes, but is computationally intensive. The other two modes  $SI^{16}$  and  $SI^{32}$  explore the trade-off between the quality of the encoding and computational workload. The resource manager, therefore, can take advantage of this trade-off and adjust the quality of the encoding according to the available resources, such as energy budget and number of PEs, at run-time. A transition from one mode to any other one is possible, thereby resulting in totally 12 possible transitions. At run-time, reconfiguration of the parameter  $dl$  is triggered by the environment, e.g., the resource manager in this case. Subsequently, control actor  $A_c$  propagates  $dl$  to the data-flow actors shown in Fig. 11 through the dashed-lined edges.

We measured the WCETs of all dataflow actors in Fig. 11 in the four modes on an ARM Cortex-A9 [24] processor. All dataflow actors were compiled using the compiler `arm-xilinx-eabi-gcc 4.7.2` with the vectorization option. The WCETs of all actors in all four modes are given in Table II. It is worth to note that in mode  $SI^8$ , actors `Spec2Env` and `male2female` exhibit exceptionally high WCETs. It is because parameter  $dl$  represents the size of the inner-most loop in the computation of actors `Spec2Env` and `male2female`. Small  $dl$  (in this case  $dl = 8$ ) leads to the fact that the inner-most loop cannot be vectorized by the compiler. In the other modes from  $SI^{16}$  to  $SI^{64}$ , larger sizes of the inner-most loop ( $dl$  equal to 16, 32, and 64, respectively) lead to full vectorization of the computation of actors `Spec2Env` and `male2female`. Therefore, in these three modes, the WCETs of actors `Spec2Env` and `male2female` are even smaller than the ones in mode  $SI^8$ . The dataflow actors of Vocoder are allocated to 4 PEs as shown in Fig. 12. This allocation guarantees that the shortest periods (maximum throughput) in the steady-states of all modes can be achieved.

Table III shows the performance results for the four modes in their steady-state under SPS. For instance, the second column

TABLE III

PERFORMANCE RESULTS OF FOUR MODES OF VOCODER IN THE STEADY-STATE.

Mode	Period ( $T$ in clk.)	Total utilization ( $U$ )	Iteration latency ( $L$ )
$SI^8$	917504	1.24	7339608
$SI^{16}$	148864	2.36	1191436
$SI^{32}$	178176	3.19	1425448
$SI^{64}$	300288	3.4	2402550

TABLE IV

PERFORMANCE RESULTS FOR ALL MODE TRANSITIONS OF VOCODER.

Transition ( $SI^i$ to $SI^j$ )	$\Delta_{\min}^{o \rightarrow l}$ (in clk.)	$\Delta_{\max}^{o \rightarrow l}$ (in clk.)	$x$ (in clk.)	$\delta^{o \rightarrow l}$ (in clk.)
$SI^8 \rightarrow SI^{64}$	3636815	4554266	1234264	1234264
$SI^8 \rightarrow SI^{32}$	2903988	3821439	1478540	1478540
$SI^8 \rightarrow SI^{16}$	2728479	3645930	<b>1537043</b>	1537043
$SI^{16} \rightarrow SI^{64}$	2402550	2551480	0	0
$SI^{16} \rightarrow SI^{32}$	1425448	1574378	0	0
$SI^{16} \rightarrow SI^8$	7339608	7488538	0	0
$SI^{32} \rightarrow SI^{64}$	2402550	2580731	0	0
$SI^{32} \rightarrow SI^{16}$	1425448	1603629	234012	234012
$SI^{32} \rightarrow SI^8$	7339608	7517789	0	0
$SI^{64} \rightarrow SI^{32}$	2402550	2702869	977102	977102
$SI^{64} \rightarrow SI^{16}$	2402550	2702869	1211114	1211114
$SI^{64} \rightarrow SI^8$	7339608	7639927	0	0

at the first row in Table III indicates that it is guaranteed for sink actor WriteWave to produce 256 samples per 917451 clock cycles in mode  $SI^8$ . This is the “worst-case” performance among all four modes because the Spec2Env actor exhibits exceptionally high workload (*cf.* WCETs in Table II) in mode  $SI^8$ . Consequently, actor Spec2Env becomes the “bottleneck” actor, so that mode  $SI^8$  cannot be scheduled with higher throughput (shorter period). Nevertheless, all mode  $SI^8$  actors as a whole require a total processor utilization ( $U$ ) of only 1.24 (see the third column in Table III) which is the least among all modes. From Table III, we can see that MADF together with the SPS framework brings another advantage of efficiently utilizing PE resources. For example, in case that Vocoder is switched to a mode with lower processor utilization, idle capacity of PEs can be efficiently utilized by admitting other applications at run-time without introducing interference to the currently running Vocoder.

Now, we focus on the performance results of the MOO protocol, namely transition delays, for all possible transitions between the four modes of Vocoder. Table IV shows both the minimum and maximum transition delays in accordance with Theorem 1 for all transitions. We can see in the second column of Table IV that, in the best case, the transition delays for 6 out of 12 transitions remain the same as the iteration latencies of the new modes. This can be seen as  $x = 0$  shown in the fourth column. In these 6 transitions, the proposed MOO protocol does not introduce any extra delay. In the 6 remaining transitions, as expected, the MOO protocol introduces offset  $x > 0$  to the transitions from an old mode with a longer iteration latency to a new mode with a shorter iteration latency. For instance, the largest  $x$  (in bold shown in Table IV) happens in case of a transition from mode  $SI^8$  with the longest iteration latency (see the fourth column in Table III) to mode  $SI^{16}$  with the shortest iteration latency. To quantify  $x$ , we compute the percentage of  $x$  compared to both minimum and maximum transition delays as  $\Omega_{\min} = \frac{x}{\Delta_{\min}^{o \rightarrow l}} \times 100\%$ ,  $\Omega_{\max} = \frac{x}{\Delta_{\max}^{o \rightarrow l}} \times 100\%$ .  $\Omega_{\min}$  varies from the worst-case 56% to the best case 16% with an average of

TABLE V

THE PERIOD AND ITERATION LATENCY OF MODES IN MP3 DECODER IN CLK. CYCLES.

Mode	s-s	s-l	l-s	l-l	m
Period ( $T$ )	5830000	5785970	5830000	4640000	5760000
Iteration latency ( $L$ )	9434720	9234570	9278600	7466400	9089900

41%, whereas  $\Omega_{\max}$  varies from the worst-case 44% to the best case 14% with an average of 33%. Therefore, the increase of the transition delays due to the MOO protocol is reasonable for this real-life application.

Next, we consider the effect of the actor allocation shown in Fig. 12 on the earliest starting times of actors in the new mode upon a transition (*cf.* Lemma 3). In this particular example, we find out that no extra delay is incurred to any actor in all transitions due to the fixed actor allocation. This can be seen from the fourth and fifth columns in Table IV, where  $\delta^{o \rightarrow l} = x$ .

### B. Case Study 2

To further evaluate the MOO protocol, presented in Section IV-C2, in this section, we performed an experiment with the MP3 decoder application, which is a real-life adaptive streaming application, taken from [5]. This MP3 decoder is a frame-based algorithm that retrieves audio frames from the incoming compressed bitstream. In the MP3 decoder, each audio frame can be decoded using a different method. In total, MP3 decoder has five individual decoding methods for audio frames that are denoted as {s-s, l-l, l-s, s-l, m}.

Each of these methods can be represented accurately by an SDF graph. Therefore, the application behavior can be accurately captured using FSM-SADF [5] rather than conservatively capture these methods in a static dataflow model. Consequently, a much tighter performance can be guaranteed by FSM-SADF graph than SDF. Note that since each mode in our MADF model is represented as a CSDF graph, our MADF is more expressive than FSM-SADF and therefore, the MP3 decoder can be also properly modeled with MADF. The period and iteration latency of each mode are given in Table V.

Let us now compare the throughput of MP3 decoder modeled as MADF and FSM-SADF graphs. To compute the throughput of MP3 decoder modeled by the FSM-SADF, we use the publicly available SDF<sup>3</sup> tool set [25]. Since the type of frames may change non-deterministically in arbitrary orders, SDF<sup>3</sup> detects the worst-case mode transition using the state-space exploration approach developed in [5] for FSM-SADF to lower bound the throughput. To compute the worst-case throughput of the application, we use the `sdf3analysis-fsmsadf` tool from SDF<sup>3</sup>. Similarly, we use the same approach to compute the throughput of our MADF model that uses the MOO protocol. For both models, the same throughput of  $1.75 \cdot 10^{(-7)}$  frame per clock cycle is achieved. Therefore, both models perform equally well in terms of the worst-case throughput they can guarantee and the delay introduced by our MOO protocol during mode transitions has no impact on the worst-case throughput.

Now, we focus on the performance results of our MADF and FSM-SADF models in terms of the iteration latency of the modes and the transition delay. The results of this comparison for four different mode transition sequences is given in Table VI. In this table, for each mode transition sequence,

TABLE VI  
PERFORMANCE RESULTS OF MP3 DECODER FOR FOUR DIFFERENT MODE TRANSITION SEQUENCES USING MADF AND FSM-SADF MODELS.

Mode Sequence	FSM-SADF [5]								MADF						
	Iteration latency				Transition delay				Iteration latency			Transition delay			
	$L^{s-s}$	$L^{s-l}$	$L^m$	$L^{l-l}$	$\Delta^{s-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	$\Delta^{m \rightarrow l-l}$		$L^{s-s}$	$L^{s-l}$	$L^m$	$L^{l-l}$	$\Delta^{s-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	$\Delta^{m \rightarrow l-l}$
s-s→s-l→m→l-l	9434720	9434670	9310400	<b>9310400</b>	9434670	<b>9310400</b>	9310400	9434720	9234570	9089900	<b>7466400</b>	10032600	9261700	9089900	
s-s→l-l→s-l→m	$L^{s-s}$	$L^{l-l}$	$L^{s-l}$	$L^m$	$\Delta^{s-s \rightarrow l-l}$	$\Delta^{l-l \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	$L^{s-s}$	$L^{l-l}$	$L^{s-l}$	$L^m$	$\Delta^{s-s \rightarrow l-l}$	$\Delta^{l-l \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	
	9434720	<b>9434700</b>	9434670	9217800	9434700	9434670	<b>9217800</b>	9434720	<b>7466400</b>	9234570	9089900	9434700	9234500	<b>9261700</b>	
l-s→s-l→m→l-l	$L^{l-s}$	$L^{s-l}$	$L^m$	$L^{l-l}$	$\Delta^{l-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	$\Delta^{m \rightarrow l-l}$	$L^{l-s}$	$L^{s-l}$	$L^m$	$L^{l-l}$	$\Delta^{l-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow m}$	$\Delta^{m \rightarrow l-l}$	
	9278600	9278570	9197200	<b>9197200</b>	9278570	<b>9197200</b>	9197200	9278600	9234570	9089900	<b>7466400</b>	9876500	<b>9261700</b>	9089900	
s-s→l-s→s-l→l-l	$L^{s-s}$	$L^{l-s}$	$L^{s-l}$	$L^{l-l}$	$\Delta^{s-s \rightarrow l-s}$	$\Delta^{l-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow l-l}$	$L^{s-s}$	$L^{l-s}$	$L^{s-l}$	$L^{l-l}$	$\Delta^{s-s \rightarrow l-s}$	$\Delta^{l-s \rightarrow s-l}$	$\Delta^{s-l \rightarrow l-l}$	
	9434720	9434700	9434670	<b>8661500</b>	9434700	9434670	8661500	9434720	9278600	9234570	<b>7466400</b>	10032600	9876500	9234600	

the iteration latency of each mode and the transition delay of each mode transition are given for our MADF model that uses the MOO protocol and the FSM-SADF model that uses the self-timed protocol. From this table, we can clearly see that our MADF retains the iteration latency of each mode irrespective of the mode transition sequences. Using the FSM-SADF model, however, the iteration latency of modes in the steady-state is accordingly changed with respect to the order of mode transitions. For instance, mode l-l has different iteration latency,  $L^{l-l}$ , of 9310400, 9434700, 9197200, and 8661500 for the different mode transition sequences, when using FSM-SADF. In contrast, the same mode l-l has a constant iteration latency of 7466400 under our MADF model (bolded in Table VI). Therefore, the iteration latency of modes in the steady-state can not be guaranteed under the FSM-SADF model as it is highly dependent on the order of mode transitions which is not known beforehand at design-time.

From Table VI, we can also see that by changing the iteration latency of the modes, the transition delays are also changed. Although the transition delays are sometimes shorter in the FSM-SADF model, the FSM-SADF model is potentially unpredictable. Our MADF model, however, is completely predictable because the (minimum) transition delays for all mode transitions can be computed beforehand at design-time according to Theorem 1. For instance, the transition from mode s-l to mode m has different transition delay,  $\Delta^{s-l \rightarrow m}$ , of 9310400, 9217800, and 9197200 for different mode transition sequences under the FSM-SADF model whereas this mode transition has a constant transition delay of 9261700 under our MADF model (bolded in Table VI).

## VII. CONCLUSION

In this paper, we have proposed the novel Mode-Aware Data Flow (MADF) model which can capture effectively the adaptive nature of modern streaming applications. Moreover, as an important part of the operational semantics of MADF, we have proposed a novel protocol for mode transitions. The main advantage of this transition protocol is that, in contrast to the self-timed transition protocol, it avoids timing interference between modes upon mode transitions. As a result, any mode transition can be analyzed independently from others that occurred in the past. Furthermore, based on the transition protocol, we have proposed a hard real-time analysis and scheduling framework to reason and guarantee timing constraints by avoiding processor overloading during mode transitions. Finally, we evaluate the effectiveness of our MADF model compared with the well-know FSM-SADF model

by conducting two case studies using two real-life adaptive streaming applications.

## REFERENCES

- [1] A. Gerstlauer et al. Electronic system-level synthesis methodologies. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2009.
- [2] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on computers*, 1987.
- [3] G. Bilsen et al. Cycle-static dataflow. *IEEE Trans. Signal Process.*, 1996.
- [4] B. D. Theelen et al. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *MEMOCODE*, 2006.
- [5] M. Geilen and S. Stuijk. Worst-case performance analysis of synchronous dataflow scenarios. In *CODES+ISSS*, 2010.
- [6] M. H. Wiggers et al. Buffer capacity computation for throughput-constrained modal task graphs. *ACM Trans. Embed. Comput. Syst.*, 2010.
- [7] O. Moreira. Temporal analysis and scheduling of hard real-time radios running on a multi-processor. *ser. PHD Thesis, Technische Universiteit Eindhoven*, 2012.
- [8] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for dsp systems. *IEEE Trans. Signal Process.*, 2001.
- [9] M. Geilen. Synchronous dataflow scenarios. *ACM Trans. Embed. Comput. Syst.*, 2010.
- [10] R. Henia and R. Ernst. Scenario aware analysis for complex event models and distributed systems. In *RTSS*, 2007.
- [11] M. Negrean et al. Bounding mode change transition latencies for multi-mode real-time distributed applications. In *ETFA*, 2011.
- [12] J. Real and A.s Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-time Systems*, 2004.
- [13] N. Stoimenov et al. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *DATE*, 2009.
- [14] M. Bamakhrama and T. Stefanov. On the hard-real-time scheduling of embedded streaming applications. *DAES*, 2013.
- [15] B. Lickly et al. Predictable programming on a precision timed architecture. In *CASES*, 2008.
- [16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 1973.
- [17] J. T. Zhai, S. Niknam, and T. Stefanov. Modeling, analysis, and hard real-time scheduling of adaptive streaming applications. *arXiv:1807.04835*, 2018. <https://arxiv.org/abs/1807.04835>.
- [18] S. Stuijk et al. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. on Computers*, 2008.
- [19] B. Bodin et al. Periodic schedules for cyclo-static dataflow. In *ESTIMedia*, 2013.
- [20] S. Neuendorffer and E. Lee. Hierarchical reconfiguration of dataflow models. In *MEMOCODE*, 2004.
- [21] J. T. Zhai. *Adaptive streaming applications: analysis and implementation models*. PhD thesis, PhD dissertation, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, 2015. <https://openaccess.leidenuniv.nl/handle/1887/32963>.
- [22] S. Niknam and T. Stefanov. Energy-efficient scheduling of throughput-constrained streaming applications by periodic mode switching. In *SAMOS*, 2017.
- [23] M. I. Gordon, W. Thies, and S. Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. *ACM SIGOPS Operating Systems Review*, 2006.
- [24] ARM Cortex. A9 processor. <http://www.arm.com/products/processors/cortex-a/cortex-a9.php>, 2013.
- [25] S. Stuijk, M. Geilen, and T. Basten. Sdf\* 3: Sdf for free. In *ACSD*, 2006.