End-to-end latency analysis of dataflow scenarios mapped onto shared heterogeneous resources

End-to-end Latency Analysis of Dataflow Scenarios Mapped onto Shared Heterogeneous Resources

Firew Siyoum, *Student Member, IEEE,* Marc Geilen, *Member, IEEE,* and Henk Corporaal, *Member, IEEE*

***Abstract*—The design of embedded wireless and multimedia applications requires temporal analysis to verify if real-time constraints such as throughput and latency are met. This paper presents a design-time analytical approach to derive a conservative upper-bound to the maximum end-to-end latency of a streaming application. Existing analytical approaches often assume static application models, which cannot cope with the data-dependent execution of dynamic streaming applications. Consequently, they give overly pessimistic upper-bounds. In this work, we use an expressively richer dataflow model of computation as an application model. The model supports adaptive applications that change their graph structure, execution times and data rates, depending on their mode of operation, or *scenario*. We first formalize the latency analysis problem in the presence of dynamically switching scenarios. We characterize each scenario with a compact matrix in $(max, +)$ algebra using a symbolic execution of one graph iteration. The resulting matrices are then composed to derive a bound to the end-to-end latency under a periodic source. Aperiodic sources such as sporadic streams can be analyzed by reduction to a periodic reference. We demonstrate the applicability of the technique with dataflow models from the wireless application domain. Moreover, the method is illustrated with a trade-off analysis in resource reservation under a throughput constraint. The evaluation shows that the approach has a low run-time, which enables it to be effectively integrated in multiprocessor design-flows of streaming applications.**

***Index Terms*—end-to-end latency, synchronous dataflow, temporal analysis, max-plus algebra, real-time, dynamism, embedded streaming applications**

I. Introduction

Contemporary embedded wireless and multimedia applications are typically implemented on a Multiprocessor Systems-on-Chip (MPSoC) for power and performance reasons. The MPSoC commonly comprises heterogeneous resources that are shared between multiple applications under different scheduling policies. These applications have strict real-time constraints such as worst-case throughput and maximum end-to-end latency. It is crucial to guarantee that such constraints are satisfied under all operating conditions. Simulation and measurement based analysis techniques cannot guarantee temporal bounds, since it is impractical, if not impossible, to cover all possible system behaviors. Thus, analytical techniques are often used to compute conservative temporal bounds. Dataflow models of computation (MoCs), in particular, have been widely used to model streaming applications and analyze their worst-case temporal properties at design-time. In this section, we skim through such dataflow analysis models and highlight the analysis challenges. The section presents the problem addressed in this work and outlines our approach.

Firew Siyoum, Marc Geilen and Henk Corporaal are with the Department of Electrical Engineering, Eindhoven University of Technology, Den Dolech 2 5612 AZ Eindhoven, The Netherlands.
E-mail: {f.m.siyoum, m.c.w.geilen, h.corporaal}@tue.nl

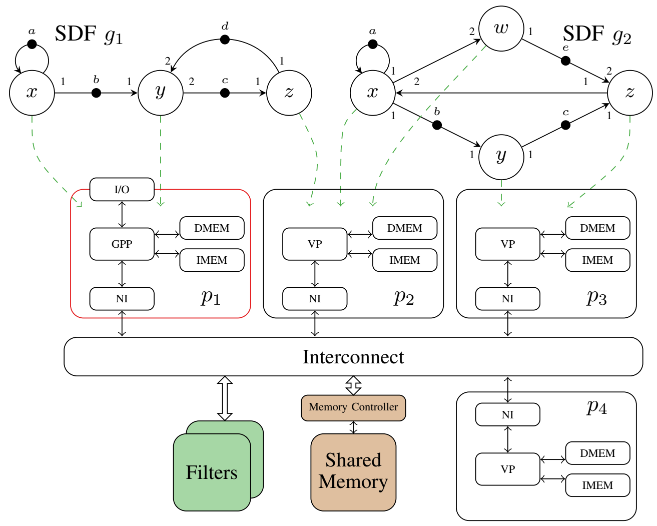


Fig. 1: Dataflow graphs mapped on a heterogeneous MPSoC

A. Dataflow Analysis Models

A dataflow model is a directed graph, where the nodes, called *actors*, denote computation entities and the edges denote FIFO buffer *channels*. Actors communicate by sending data *tokens* through their *ports*. There are different variants of dataflow models, which vary with their levels of expressiveness and analysability (cf. Section II). A prominent dataflow model is Synchronous Dataflow (SDF) [1]. Figure 1 shows two SDF graphs that are mapped onto a MPSoC. The black-dots in the figure are initial tokens of channels. The numbers on the edges indicate token production and consumption rates of ports. The numbers inside the actors denote Worst-Case Execution Times (WCETs). A *firing* of a SDF actor consumes from each input port as many tokens as the input port rate. After a timing delay, given by its WCET, it then produces at each output port as many tokens as the output port rate.

B. Challenges to Dataflow-based Temporal Analysis

A challenge to dataflow-based analysis of present-day streaming applications is their dynamic behavior. These applications change their graph structure, data rates and computation loads, depending on their mode of operation. Take for instance the 802.11a WLAN application. The WLAN baseband packet decoding consists of 4 modes [2]. These are Synchronization, Header decoding, Payload decoding and Cyclic redundancy checking (CRC). Once a packet is detected, mode 1 executes repeatedly until synchronization succeeds. Then, the application switches to mode 2 to decode the packet header. Mode 2 determines the size of the payload, which may vary from 1 to 256 OFDM symbols, each with a length of $4\mu s$. After header decoding, mode 3 is executed as many times as the number of symbols. The final mode performs CRC and sends an acknowledgment if the packet is received in a good order. Each mode may activate a different set of tasks and may have a different computational load than others.

A conservative static dataflow model, such as Synchronous Dataflow (SDF) [1], abstracts from such varying operating modes for the sake of analysability. It models an application with a static graph that captures the worst-case behavior across all modes. However, the abstraction leads to overly pessimistic temporal bounds. This further leads to unnecessarily large budget reservation of resources such as processors and communication interconnects to guarantee real-time latency and throughput requirements. Thus, a refined temporal analysis that considers the different operating scenarios is crucial to compute tight real-time temporal bounds and, consequently, avoid unnecessary over-allocation of scarce MPSoC resources.

C. Modeling Dynamic Applications using Scenarios

In this work, we use an expressively richer dataflow model, called *Scenario-aware Dataflow (SADF)*, as an application model. SADF [3] has been proposed to model adaptive applications, which are characterized by dynamic mode switchings. It splits the dynamic behavior of an application into a group of static operating modes, called *scenarios*. Each scenario is explicitly modeled by a SDF graph, which can have a different graph structure as well as graph parameters than the other scenarios. In a finite-state machine (FSM) based SADF, the possible orders of execution of scenarios are specified by the language of a FSM [3].

The intent of SADF is to improve the expressiveness of SDF, while maintaining design-time analyzability. The worst-case throughput analysis of SADF has been studied in [3]–[5]. The techniques follow a compositional approach, where each scenario is first analyzed separately. Then the results are composed, making use of the possible scenario sequences given by the FSM. Due to the refined analysis, SADF enables a more accurate temporal analysis than approaches that do not consider scenario sequences [6]. Nevertheless, the maximum end-to-end latency, which is a key real-time requirement, has never been studied in the presence of dynamically switching scenarios. In WLAN, for instance, an acknowledgment packet must be sent within $16\mu s$ of the reception of the last symbol if the cyclic redundancy check is successful. This time guard of $16\mu s$, known as the Short Intra-Frame Spacing, is a latency constraint that must be satisfied.

D. This work

In this paper, we present a design-time analytical approach to compute the end-to-end latency of an adaptive streaming application, which has a set of dynamically switching scenarios modeled with a FSM-based SADF. We define latency in terms of two causally related actor firings between a source and a sink actor, in a source and a sink scenario, respectively. This implies that the latency analysis of SADF requires computing the temporal distance that separates two causally related events across (possibly multiple) scenarios. This is unlike the latency analysis of static SDF, which is only concerned with source and sink actor firings within the same graph iteration [7], [8].

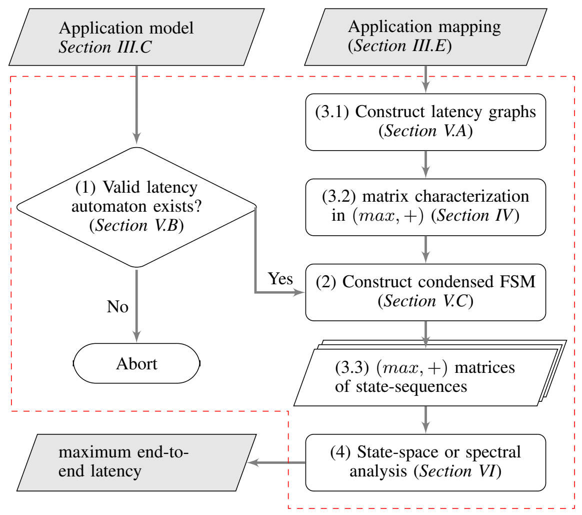

Application model
Section III.C
Application mapping
(Section III.E)
(3.1) Construct latency graphs
(Section V.A)
(1) Valid latency automaton exists?
(Section V.B)
(3.2) matrix characterization in $(max, +)$ (Section IV)
Yes
(2) Construct condensed FSM
(Section V.C)
No
Abort
(3.3) $(max, +)$ matrices of state-sequences
maximum end-to-end latency
(4) State-space or spectral analysis (Section VI)

Fig. 2: Latency analysis framework for dynamic applications

Figure 2 outlines the steps in our latency analysis approach. (1) It begins with asserting whether the FSM qualifies for the latency analysis (cf. Section V-B *latency automaton*). This is required because a bound to the maximum latency may not exist for an arbitrary FSM, which has an unbounded length scenario sequence, due to cyclic transitions, between the two causally-related firings.

(2) If the FSM qualifies, all possible scenario sequences between the source and sink scenarios are extracted from the FSM. This results in a *condensed FSM* (cf. Section V-C), each of whose states is labelled with a scenario sequence.

(3) Each scenario sequence is then characterized by a matrix in $(max, +)$ algebra [9] that captures its end-to-end timing behavior. The input to the matrix characterization is a given *application mapping* (cf. Section III-E), which specifies the resource allocations of each scenario on a shared heterogeneous MPSoC platform. (3.1) First, the causality between source and sink actors is enforced by constructing the *latency graphs* (cf. Section V-A) of source and sink scenarios. (3.2) Then, a $(max, +)$ characterization matrix is constructed for each scenario mapping through a symbolic execution of the scenario for one complete iteration (cf. Section IV-B). (3.3) The matrix of each scenario sequence is then constructed from the product of the matrices of the constituent scenario mappings.

(4) The latency is finally derived in either of two different ways: 1) from a state-space, constructed in a breadth-first-search manner, by considering the possible transitions between scenario sequences, and 2) from a spectral analysis of the single-matrix characterization of all scenario sequences.

E. Contributions

This is the first work on the latency analysis of FSM-based SADF application models. Previous works [3]–[5], [10] on such applications are primarily concerned with the worst-case throughput, which is in essence a Maximum Cycle Mean (MCM) analysis on a certain throughput or state-space graph. Unlike the MCM analysis, the knowledge of the possible execution cycles of the state-space is not relevant for the latency analysis. The latency computation rather requires trailing all possible finite length execution paths (scenario sequences) in the state-space between two events, which correspond to the firings of a source actor and a sink actor.

The state-space has potentially many occurrences of such events, following the repeated executions of source and sink scenarios. This requires appropriate problem formulation to track all pairs of causally-related source and sink actor firings. In this work, we achieve this by systematically containing all such causal relations within scenario-sequences, which end up to be nodes/states in the state-space graph. As a result, the latency analysis reduces to finding the state, which has the event pair with the longest temporal distance.

As a relatively scalable alternative, we also present the *spectral analysis*. It involves a recurrence execution ($\gamma_{k+1} = M \cdot \gamma_k$) until the state-vector γ converges. Such recurrent equations are used in [3], [9] for eigenvalue computation (and hence for throughput). Because of our problem formulation using condensed FSMs, we extended the algorithm in such a way that each visited state-vector encodes a conservative bound to a source-to-sink delay of a scenario sequence. As such, the set of visited state-vectors form a conservative state-space for the latency analysis, as opposed to the exact state-space of the first approach.

We demonstrate the analysis technique with dataflow models from the wireless application domain. Moreover, we present a trade-off analysis in resource reservation under a throughput constraint, which is set by the source's period. Our analysis detects if the throughput constraint is not met, which implies that the maximum latency cannot be bounded. In this case, the analysis gives the minimum period the application supports. The evaluation shows that the approach has a low run-time that enables it to be effectively integrated in multiprocessor design flows of streaming applications.

II. Related Work

Exploiting classical uni-processor scheduling to multi-core systems has been widely done to compute end-to-end latency in distributed real-time systems. These works can be categorized as modular or holistic. SymTA/S [11] is a modular approach where each system component is analyzed locally with classical algorithms. Then, the local results are propagated to other connected components through appropriate event model interfaces. The analysis is completed successfully if all event streams converge towards a fixed-point through an iterative process. An end-to-end latency analysis based on the SymTA/S approach has been studied in [12]. A related approach that integrates dataflow MoCs in the SymTA/S flow has also been presented in [13]. It allows a SymTA/S component to be modeled as a single-rate SDF graph, also called Homogeneous SDF (HSDF), which restricts all port-rates to 1. Holistic approaches [14]–[16] extend classical algorithms for specific combinations of task model, resource sharing and communication policy. As such, they integrate task and communication scheduling into a single analysis framework. Thus, they may give tight performance bounds by taking system-level correlations into account. However, they are not modular and may require a completely new analysis for a new combination of schedulers and task model [17].

Another modular latency analysis that supports arbitrary event models, so-called *arrival curves*, has been presented in [18]. The approach is based on real-time calculus [19], which adopts concepts from network calculus to distributed real-time applications. Real-time calculus employs *service curves* to model the availability of resources. The approach propagates the output service curves of local resource analysis to connected components for a compositional performance analysis. By using service curves, the approach supports a wide range of scheduling policies.

Dataflow MoCs have also been widely used to analyze streaming applications. In [8], maximum latency analysis for HSDF graphs has been studied. The technique supports periodic and sporadic sources, arbitrary cycles and delay tokens. However, analysis of SDF requires conversion to an HSDF equivalent. In [7], a technique has been presented to compute the minimum achievable latency between a designated pair of SDF actors. The approach does not require conversion to HSDF and uses a state-space analysis, similar to [20], to find an execution scheme for the minimum latency.

The aforementioned and similar methods assume application models that are less suitable to model modal applications. Even within a static mode, streaming applications require a task graph that supports cyclic-dependencies, multi-input tasks, multi-rate tasks and a natural way of handling back-pressured buffer communication. The above approaches [8], [12], [13], [16], [18] support only single-rate tasks. Conversion from multi-rate to single-rate (HSDF) graphs is possible [21]. However, the conversion leads to an exponential growth in the graph size, which hinders scalability [20].

The compositional analysis framework of [22] and multi-mode real-time calculus [23] have shown considerable gains by extending uni-modal real-time systems analysis to multi-mode. [22] supports a compositional analysis of multiple multi-modal components through an automaton-based resource interface, whose states map to valid combinations of component modes. This improves the service guarantee required to ensure schedulability compared to a uni-modal analysis, which assumes pessimistic workload scenarios, such as worst-case task combinations. Nevertheless, the analysis framework is limited to a single resource shared by multiple components through a hierarchical scheduling policy. [23], on the other hand, supports multiple resources, despite its limitation to simpler multi-stage/sequential structures. It extends the arrival/service curves of real-time calculus to arrival/service automata, where each state is augmented with the arrival/service curve of a certain mode. Each automaton captures only local modal variations of a particular resource. As a result, unlike our global finite-state machine, system-level correlations between mode changes, across different resources, are not taken into account. This possibly leads to pessimistic workload assumptions and overallocation of resources.

Different dataflow models are also proposed that enhance the expressiveness of SDF [4], [24]–[27]. The majority of these dataflow models, however, are either not sufficiently analysable or do not have known analysis techniques. In expressively Turing-complete dataflow models, such as DDF [24], HDF [28], KPN [29], BDF [25] and CFDF [26], basic properties such as deadlock-freedom are undecidable. On the other hand, FSM-SADF [4], which is the application model used in this paper, improves the expressiveness of SDF while still allowing for design-time analysability. Throughput analysis techniques for FSM-SADF are presented in [3]–[5]. In this paper, we analyse the maximum latency of FSM-SADF models. Unlike existing dataflow approaches [2], [30], [31], we avoid constructing resource-aware dataflow models to analyze resource sharing. Instead, we use worst-case resource availability curves to model resource sharing. As a result, we keep the graph size intact and improve scalability.

III. Preliminaries

This section recaps basic dataflow modeling concepts and gives their formal definitions. Section III-A introduces notation and Section III-B briefly introduce the $(max, +)$ algebra, which we use to characterize the execution of dataflow graphs as linear systems. Sections III-C, III-D and III-E formally define the application model, the MPSoC platform resource model and the application-to-platform mapping, respectively.

A. Notation

We use upper-case letters (A, Θ) to denote sets and sequences, except for letters M and N that denote matrices. We use lower-case Latin letters (a) for individual elements, lower-case Greek letters $(\alpha : A \to B)$ for functions, $\mathcal{P}(A)$ for the power set of A and bar accents $(\bar{\gamma})$ for vectors. We use $|\cdot|$ to denote the cardinality of a set, the length of a sequence or size of a vector. In addition, we use $\mathbb{N}, \mathbb{N}^0$ and $\mathbb{R}$ for natural numbers starting from 1, natural numbers starting from 0 and real numbers, respectively. We denote the set of real numbers extended with $-\infty$ as $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$. The set of real numbers extended with $+\infty$ and $-\infty$ is denoted as $\overline{\mathbb{R}}_{\max} = \mathbb{R} \cup \{+\infty, -\infty\}$.

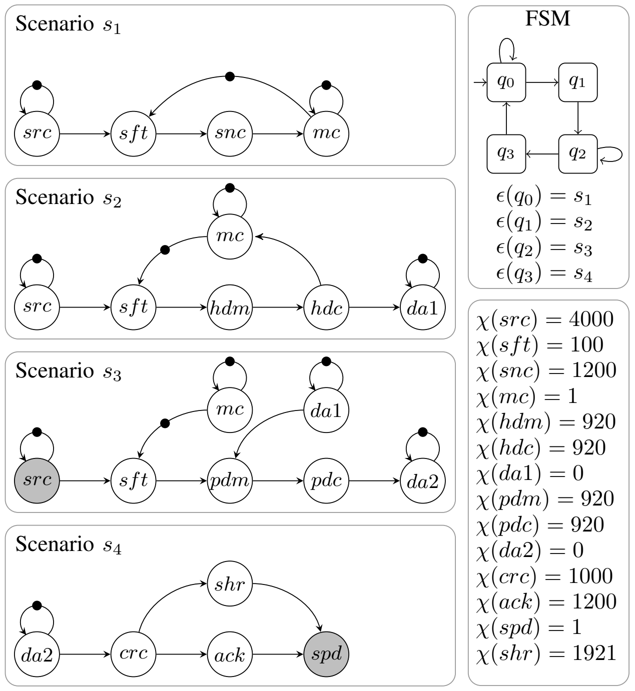
B. $(max, +)$ Algebra Notation

For elements $a, b \in \mathbb{R}_{\max}$, $(max, +)$ algebra defines $a \oplus b \overset{\text{def}}{=} \max(a, b)$ and $a \otimes b \overset{\text{def}}{=} a + b$. We use the standard max and addition notation in this paper for better readability. For $a \in \mathbb{R}_{\max}$, by definition, $\max(-\infty, a) = \max(a, -\infty) = a$ and $a + -\infty = -\infty + a = -\infty$. The algebra is extended to vectors and matrices as follows.

For $n \in \mathbb{N}$, define $\underline{n} \overset{\text{def}}{=} \{1, 2, \cdots, n\}$. An n dimensional column vector is an element of the set $\mathbb{R}^n_{\max}$. For vector $\bar{\gamma} \in \mathbb{R}^n_{\max}$, the entry at row $i \in \underline{n}$ is denoted as $[\bar{\gamma}]_i$. For $c \in \mathbb{R}_{\max}$, scalar to vector addition and multiplication are given as $[c + \gamma]_i = c + [\gamma]_i$ and $[c\gamma]_i = c[\gamma]_i$, respectively. Vector addition, subtraction and max operation are element-wise operations, i.e. for $\bar{\theta} \in \mathbb{R}^n_{\max}$, $[\bar{\gamma} \pm \bar{\theta}]_i = [\bar{\gamma}]_i \pm [\bar{\theta}]_i$ and likewise $[\max(\bar{\gamma}, \bar{\theta})]_i = \max([\bar{\gamma}]_i, [\bar{\theta}]_i)$. The *norm* of vector $\bar{\gamma}$ is the maximum entry of the vector, denoted as $\|\bar{\gamma}\| = \max_i [\bar{\gamma}]_i$. For vector $\bar{\gamma}$ with $\|\bar{\gamma}\| > -\infty$, the normalized vector is denoted as $\bar{\bar{\gamma}}$, where $[\bar{\bar{\gamma}}]_i = [\bar{\gamma}]_i - \|\bar{\gamma}\|$. We write $\bar{\gamma} \preceq \bar{\theta}$ if $\forall i \in \underline{n}, [\bar{\gamma}]_i \leq [\bar{\theta}]_i$. $\bar{\gamma} \succeq \bar{\theta}$ if $\bar{\theta} \preceq \bar{\gamma}$. Vector dot-product is max of sums, which is analogous to sum of products of standard algebra: I.e. $\bar{\gamma} \cdot \bar{\theta} = \max_i([\bar{\gamma}]_i + [\bar{\theta}]_i)$.

The set of $m \times n$ matrices is denoted as $\mathbb{R}^{m \times n}_{\max}$. Row $i \in \underline{m}$ is denoted as $[M]_{i:}$ and column $j \in \underline{n}$ as $[M]_{:j}$. An entry at row $i \in \underline{m}$ and column $j \in \underline{n}$ is denoted as $[M]_{ij}$. For $M, N \in \mathbb{R}^{n \times n}_{\max}$, we write $M \preceq N$ if $\forall i, j \in \underline{n}, [M]_{ij} \leq [N]_{ij}$. Similarly, $M \succeq N$ if $N \preceq M$. Given matrix $M \in \mathbb{R}^{m \times n}_{\max}$ and matrix $N \in \mathbb{R}^{n \times o}_{\max}$, matrix multiplication is defined using vector dot-products as $[MN]_{ij} = [M]_{i:} \cdot [N]_{:j}$. Matrix-vector product is given as $[M\bar{\gamma}]_i = [M]_{i:} \cdot \bar{\gamma}$.

Two interesting properties of matrix multiplication are *linearity* and *monotonicity*. Monotonicity says if $\bar{\gamma} \preceq \bar{\theta}$, then $M\bar{\gamma} \preceq M\bar{\theta}$. From the linearity property, we have the following two relations: $M(c + \bar{\gamma}) = c + M\bar{\gamma}$ and $M(\max(\bar{\gamma}, \bar{\theta})) = \max(M\bar{\gamma}, M\bar{\theta})$.



$\chi(src) = 4000$
$\chi(sft) = 100$
$\chi(snc) = 1200$
$\chi(mc) = 1$
$\chi(hdm) = 920$
$\chi(hdc) = 920$
$\chi(da1) = 0$
$\chi(pdm) = 920$
$\chi(pdc) = 920$
$\chi(da2) = 0$
$\chi(crc) = 1000$
$\chi(ack) = 1200$
$\chi(spd) = 1$
$\chi(shr) = 1921$

Fig. 3: FSM-SADF model of WLAN baseband processing

C. Application Model

We use the FSM-based Scenario-aware Dataflow (SADF) [4] to model applications. SADF can model adaptive streaming applications, which change their graph structure, data rates and execution times depending on the input workload. For design-time analysability, SADF splits an adaptive application into a set of static modes of operation, known as *scenarios*. Each scenario is modeled by a SDF graph (SDFG). A SDFG $g = (A, C, \iota, \chi, \rho)$ is a tuple, comprising a set A of actors, a set $C \subseteq A \times A$ of channels, number of initial tokens of channels $\iota : C \to \mathbb{N}^0$, the WCETs of actors in some time-unit $\chi : A \to \mathbb{N}^0$ and the source and destination port rates of channels $\rho : C \to \mathbb{N} \times \mathbb{N}$.

The execution of a SDFG is a timed simulation of the executions of its actors. The set of actor firings that returns the graph back to its initial tokens distribution is referred to as *iteration*. The number of firings of each actor within an iteration is expressed by the *repetition vector* of the graph. E.g. the repetition vector of graph g_1 of Figure 1 is $[1, 1, 2]$ for actors x, y and z, respectively.

An execution of an application is a sequence of executions of the different scenarios, where each scenario is executed for one iteration. The possible orders of scenario executions are captured by the language of infinite words of a finite-state machine (FSM). Given a set S of scenarios, a FSM f on S is a tuple $f = (Q, q_0, T, \epsilon)$. Q is a set of states, $q_0 \in Q$ is an initial state, $T \subseteq Q \times Q$ is a transition relation and $\epsilon : Q \to S$ is a scenario labeling. Hence, an adaptive streaming application is given by the tuple (S, f).

Figure 3 shows the SADF model of WLAN 802.11a baseband processing. This model is an adaptation to SADF from the model in [2]. WLAN packets arrive sporadically and the decoding consists of 4 scenarios, namely Synchronization (s_1), Header decoding (s_2), Payload decoding (s_3) and cyclic-

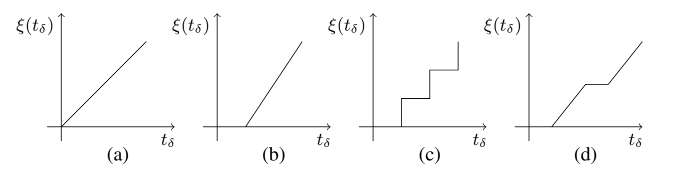


Fig. 4: Example WCRCs: Appropriate service curves have been provided from real-time calculus for different schedulers. (a) a fully available resource, (b) a bounded delay resource that guarantees a certain service rate after a bounded delay, (c) a periodic resource that periodically processes a request, (d) a TDM resource that replenishes the allocated slice every frame.

redundancy checking (s_4). Once a new packet is detected, scenario s_1 executes repeatedly until synchronization succeeds. Then, scenario s_2 decodes the packet header to determine the size of the payload that may vary from 1 to 256 OFDM symbols, each with a length of $4\mu s$. After header decoding, scenario s_3 is executed as many times as the number of OFDM symbols. The FSM approximates this conservatively by allowing an arbitrary number of payload symbols. Finally, scenario s_4 performs cyclic redundancy check (CRC). If CRC is successful, an acknowledgment packet must be sent within $16\mu s$ of the reception of the last OFDM symbol. This time guard of $16\mu s$, known as the Short Intra-Frame Spacing (SIFS), specifies a latency requirement that must be satisfied.

D. Resource Model

The MPSoC platform comprises a set Π of heterogeneous processor tiles. The interconnect is abstracted with a set Θ of connections. The platform (Π, Θ) can be shared between multiple applications under different scheduling policies. Mapping an application (S, f) to platform (Π, Θ) allocates resources such as processor budgets. E.g. on a processor under TDM scheduling, an application is allocated a slice (say 25%) of the TDM frame. At run-time, the application is preempted by other applications when it runs out of its allocated slice and has to wait for the next TDM frame.

Following the same approach as our previous work [10], we characterize the minimum share an application (S, f) has on a platform (Π, Θ) with *worst-case resource curves (WCRCs)*. WCRCs are similar to lower-bound service curves of RTC [19]. A WCRC $\xi(t_\delta)$ (Definition 1) specifies the minimum amount of resource in *service units* that an application is guaranteed to get in a given *time interval* $t_\delta \in \mathbb{N}$. Figure 4 shows some examples of WCRCs. Service units can be, for example, processor cycles or connection bandwidth in bytes.

Definition 1 (Worst-case Resource Curve - WCRC)**.** *A WCRC ξ conservatively models a resource r if for any time interval $[u, v) : u, v \in \mathbb{N}, u \leq v$, if $r[u, v)$ denotes the amount of service units that an application is guaranteed to get from the resource over interval $[u, v)$, then $r[u, v) \geq \xi(v - u)$.*

E. Application Mapping

The application-to-platform mapping decides actor-to-processor and channel-to-connection/tile bindings. It allocates resources such as processor budgets and buffer-sizes. It also constructs a static order (SO) schedule between actors that are mapped on the same processor. The mapping may vary from one scenario to another. Definitions 2 defines a scenario mapping, where Ξ denotes the set of WCRC functions.

Definition 2 (Scenario mapping)**.** *Given a platform (Π, Θ), a scenario mapping $(g, \tau, \kappa, \pi, \theta, \beta, \sigma)$ is a 7-tuple, where g is a scenario graph, $\tau : A \rightarrow \Pi$ is actor-to-processor binding and $\kappa : C \rightarrow \Theta \cup \Pi$ is channel binding. $\pi : \Pi \rightarrow \Xi$ and $\theta : \Theta \rightarrow \Xi$ are the worst-case resource curves of processors and connections, respectively. $\beta : C \rightarrow \mathbb{N}$ is the allocated buffer-sizes of channels. σ is a function that returns the SO schedule $\sigma(p) = \langle a_1, a_2, \cdots, a_n \rangle$ of actors $a_i \in A$ on processor tile $p \in \Pi$.*

An application mapping (S, f, μ) is a 3-tuple. S is a set of scenarios, f is a FSM on S and μ is a function that gives the scenario mapping $\mu(s)$ of $s \in S$.

TABLE I: A brief reference to frequently used symbols

Symbol	Description	Symbol	Description
$\bar{\gamma}$	time-stamp vector	$\phi(x, k)$	finish time of k^{th} firing of x
$[\bar{\gamma}]_i$	i^{th} entry of vector $\bar{\gamma}$	$\psi(x, k)$	buffer availability time of x
M	$(max, +)$ Matrix	$\bar{t}_i$	symbolic time-stamp of a token
$\chi(x)$	WCET of actor x	ξ	worst-case resource curve
$\omega(x)$	WCRT of actor x	c_{xy}	channel from actor x to y
s_i	example scenario	$\bar{\phi}_i$	the input stream, i^{th} element
q_i	example FSM state	$\bar{\psi}_i$	the output stream, i^{th} element

IV. $(\max, +)$ Matrix Characterization of an Application Mapping

In this section, we present a $(max, +)$ matrix characterization of an application mapping. The characterization constructs a matrix in $(max, +)$ algebra for each scenario mapping. The resulting set of matrices are the basis for the latency analysis presented later in Section VI. The matrix of a scenario mapping is constructed through a symbolic execution of the scenario for one complete iteration, as discussed next.

A. Execution of a Scenario Mapping

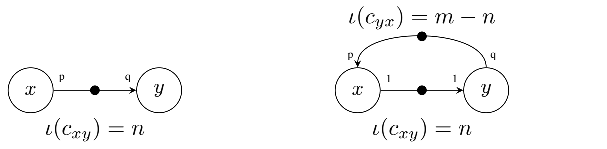
Given a scenario mapping, an actor is enabled for firing if 1) the actor is next in the static-order schedule, 2) all input tokens have arrived and 3) there is sufficient output buffer space. Thus, the enabling time of an actor is determined by the last satisfied condition, i.e. the *maximum* of the enabling times of the above three conditions. An upper-bound to the completion time of an actor's firing can be obtained by *adding* its worst-case response time (WCRT) to its start time, as shown in Equation 1. $\phi(x, k)$ denotes the upper-bound for the k^{th} firing of actor x, where t_i, t_o, t_p denote input token, output buffer and processor availability times, respectively.

$$\phi(x, k) = \max(t_i, t_o, t_p) + \omega(x) \tag{1}$$

$\omega(x)$ denotes the WCRT of actor x. It depends on the scheduling policy used to share the processor tile with other applications. It is computed from the WCRC ξ of the tile, as shown in Equation (2). Recall that $\chi(x)$ is the WCET of the actor (cf. Section III-C).

$$\omega(x) = \inf\{t_\delta \in \mathbb{N} \mid \xi(t_\delta) \geq \chi(x)\}. \tag{2}$$

Equation (1) illustrates how the entire timing behavior of a scenario mapping can be analyzed using $(max, +)$ expressions. The input data availability time t_i is determined by the last arriving token, i.e. the maximum of the production times of all input tokens. The processor availability time t_p



(a) c_{xy} has n initial tokens (b) c_{xy} has a buffer-size of $\beta(c_{yx}) = m$

Fig. 5: Modeling the allocated buffer-size of a channel in RAD

can be derived from the completion time of the last actor that is executed on the processor. Output buffer availability time t_o can be computed from the completion times of previous actor firings. This is because we employ an *acquire-release* FIFO management, where buffer spaces of input tokens are conservatively released only after the actor firing is completed. E.g. Figure 5a shows channel c_{xy} from actor x to y with $n \in \mathbb{N}^0$ initial tokens. The channel is allocated a buffer-size of $m \in \mathbb{N}$ where $m \geq n$. The availability times of free buffer spaces on this channel can be derived by considering how buffer spaces are modeled in RAD. Buffer spaces are modeled in RAD by adding another channel in the reverse direction with as many initial tokens as the number of free buffer spaces [32], as shown in Figure 5b. The tokens on the reverse channel c_{yx} represent the free buffer spaces. The output buffer availability time on channel c_{xy} for the k^{th} firing of x is given by Equation (3). The availability time is obtained from the completion time of a previous firing of actor y.

$$\psi(x, k, c_{xy}) = \phi\left(y, \left\lceil \frac{p \times k - m + n}{q} \right\rceil\right) \tag{3}$$

The output buffer availability time $t_o = \psi(x, k)$ of x for the k^{th} firing is then the latest buffer space availability time among all outgoing channels of x, i.e. $C_{xy} = \{c_{xy} \in C \mid y \in A, c_{xy} = (x, y)\}$, as shown in Equation 4.

$$\psi(x, k) = \max_{c_{xy} \in C_{xy}} \psi(x, k, c_{xy}) \tag{4}$$

Example: We show the execution of a scenario mapping of graph g_1 of Figure 1. Assume actor x is mapped on processor tile p_1 and actors y and z are mapped on p_2. The allocated buffer sizes of channels are $\beta(c_{xy}) = 1$, $\beta(c_{yz}) = 3$ and $\beta(c_{zy}) = 2$. The buffers allow the SO schedules $\sigma(p_1) = \langle x \rangle$ and $\sigma(p_2) = \langle z, y, z \rangle$. We assume TDM scheduling on both tiles with a TDM frame size of 4 and allocated TDM slice of 2 time-units for both p_1 and p_2, respectively. To simplify the example, we assume delay-less interconnect. Accounting for interconnect delays in the analysis is presented in [10].

An execution of one iteration comprises four actor firings as given by the repetition vector $[1, 1, 2]$. Initially, the only possible firing is actor z. Actor x cannot fire since it does not have enough output buffer space and actor y is not next in the SO schedule. When actor z fires, it consumes token c and produces one token on channel c_{zy}. The firing completes after its WCRT $\omega(z) = 3$. It then produces one token on channel c_{zy}, time-stamped with its production time 3. Then, actor y fires and completes its firing at $\phi(y, 1) = \max(t_i, \psi(y, 1), t_p) + \omega(y) = \max(3, 0, 3) + 7 = 10$. Note that $\psi(y, 1) = 0$ since channel c_{yz} had already 2 free buffer spaces at the beginning of the iteration. This can also be derived from Equation (4), which gives $\psi(y, 1) = \phi(z, 0) = 0$. Similarly, the remaining two firings complete at $\phi(x, 1) = 10 + 4 = 14$ and $\phi(z, 2) = 10 + 3 = 13$. The collection of the time-stamps of the final tokens marks a bound to the end of the iteration. This enables temporal analysis of a scenario mapping such as throughput, which is given by iterations per time-unit [4].

B. $(max, +)$ *matrix of a scenario mapping*

Next, we execute the above example using *symbolic time-stamps* of tokens. By symbolic execution, as opposed to concrete execution like the above example, we compute a bound for any iteration, given the collection of time-stamps that mark the start of the iteration [4]. This is given by a recurrent relation $\bar{\gamma}_{k+1} = M \cdot \bar{\gamma}_k$, where $\bar{\gamma}_k \in \mathbb{R}^n_{\max}$ is a vector of time-stamps that mark a bound to iteration k. $M \in \mathbb{R}^{n \times n}_{\max}$ is a $(max, +)$ characterization matrix of the scenario mapping.

Following [4], the production time of a token can be expressed symbolically as $t = \max_i(t_i + g_i)$, where t_i are time-stamps of initial tokens and resource availability, and g_i are suitable constants. This can be written in a $(max, +)$ vector dot-product $\bar{t}.\bar{g}$, where $\bar{t} = [t_a, t_b, t_c, t_d, t_{p1}, t_{p2}]^T$ and $\bar{g} \in \mathbb{R}^6_{\max}$. The time-stamp vector has 6 entries: 4 for the initial tokens (a, b, c and d) and 2 for processors p_1 and p_2.

Let the time-stamps $\bar{t}_a, \bar{t}_b, \bar{t}_c$ and $\bar{t}_d$ correspond to the time-stamp vectors $[0; -\infty; -\infty; -\infty; -\infty; -\infty]^T$, $[-\infty; 0; -\infty; -\infty; -\infty; -\infty]^T$, $[-\infty; -\infty; 0; -\infty; -\infty; -\infty]^T$ and $[-\infty; -\infty; -\infty; 0; -\infty; -\infty]^T$, respectively. Similarly, the time-stamps $\bar{t}_{p1}$ and $\bar{t}_{p2}$ encode processor availability times. They correspond to $[-\infty; -\infty; -\infty; -\infty; 0; -\infty]^T$ and $[-\infty; -\infty; -\infty; -\infty; -\infty; 0]^T$. Using symbolic versions of Equation (1) and (4), where all time-stamps are vectors, the completion time of the first firing of actor z, $\bar{\phi}(z, 1)$, is

$$\begin{aligned} \bar{\phi}(z, 1) &= \max(\bar{t}_c, \bar{\psi}(z, 1), \bar{t}_{p2}) + \omega(z) \\ &= \max(\bar{t}_c, \bar{t}_{p2}) + \omega(z) \\ &= [-\infty; -\infty; 0; -\infty; -\infty; 0] + 3 \\ &= [-\infty; -\infty; 3; -\infty; -\infty; 3]. \end{aligned} \tag{5}$$

Note that $\bar{t}_c$, $\bar{\psi}(z, 1)$ and $\bar{t}_{p2}$ are the input token, output buffer and processor availability times, respectively. According to Equation (4), $\bar{\psi}(z, 1)$ evaluates to $\bar{\phi}(y, 0)$, which is the completion time of actor y in the previous iteration on tile p_2. Hence, $\bar{\psi}(z, 1)$ occurs before the current availability time of p_2: i.e. $\bar{\psi}(z, 1) \preceq \bar{t}_{p2}$. This simplifies the evaluation as shown in Equation (5). Equation (5) also updates $\bar{t}_{p2}$ to $\bar{\phi}(z, 1)$. Similarly, the completion times of the other firings are $\bar{\phi}(y, 1) = [-\infty; 7; 10; 7; -\infty; 10]$, $\bar{\phi}(z, 2) = [-\infty; 10; 13; 10; -\infty; 13]$ and $\bar{\phi}(x, 1) = [4; 11; 14; 11; 4; 14]$. At the end of the iteration, the new four tokens have the time-stamps $\bar{t'}_a = \bar{\phi}(x, 1)$, $\bar{t'}_b = \bar{\phi}(x, 1)$, $\bar{t'}_c = \bar{\phi}(y, 1)$ and $\bar{t'}_d = \bar{\phi}(z, 2)$. The time-stamps of the two processors become $\bar{t'}_{p1} = \bar{\phi}(x, 1)$ and $\bar{t'}_{p2} = \bar{\phi}(z, 2)$. Collecting the new symbolic time-stamps as $[\bar{t'}_a, \bar{t'}_b, \bar{t'}_c, \bar{t'}_d, \bar{t'}_{p1}, \bar{t'}_{p2}]$ gives Equation (6), which captures the worst-case timing behavior of an iteration. The relation enables to compute a bound for an iteration, given the start of the iteration. I.e. $\bar{\gamma}_{k+1} = M \cdot \bar{\gamma}_k$.

$$\begin{bmatrix} t'_a \\ t'_b \\ t'_c \\ t'_d \\ t'_{p1} \\ t'_{p2} \end{bmatrix} = \begin{bmatrix} 4 & 11 & 14 & 11 & 4 & 14 \\ 4 & 11 & 14 & 11 & 4 & 14 \\ -\infty & 7 & 10 & 7 & -\infty & 10 \\ -\infty & 10 & 13 & 10 & -\infty & 13 \\ 4 & 11 & 14 & 11 & 4 & 14 \\ -\infty & 10 & 13 & 10 & -\infty & 13 \end{bmatrix} \cdot \begin{bmatrix} t_a \\ t_b \\ t_c \\ t_d \\ t_{p1} \\ t_{p2} \end{bmatrix} \tag{6}$$

An algorithm to construct the $(max, +)$ matrix of a given scenario mapping is presented in [10]. Once a matrix is constructed for each scenario mapping, the resulting set of matrices are used to derive a bound to the maximum end-to-end latency, which is formulated next in Section V.

V. Latency Analysis Problem Formulation

A streaming application processes a sequence of input data objects (packets, frames, etc). It then produces the corresponding sequence of output data objects. We define a *stream* as a sequence of arrival/production time-stamps of these data objects, as defined in Definition 3.

Definition 3 (Stream)**.** *A stream $\bar{\phi}$ is an element of the set $\mathbb{R}_{\max}^{\mathbb{N}}$ such that for $\bar{\phi} = \langle \bar{\phi}_1, \bar{\phi}_2, \cdots, \bar{\phi}_n, \cdots \rangle$ and $n \in \mathbb{N}$, $\bar{\phi}_n \in \mathbb{R}_{\max}$ is the time-stamp of the n^{th} element of the stream.*

The time-stamp of the n^{th} element of a stream denotes the arrival time of the n^{th} input or the production time of the n^{th} output. As further discussed in Section V-A, we construct streams in such a way that there exists a one-to-one causal relation between the elements of an input stream and its corresponding output stream. Given an input stream and its corresponding output stream, the maximum end-to-end latency is determined by the longest timing separation between their elements, as defined in Definition 4.

Definition 4 (Maximum latency)**.** *Given an input stream $\bar{\phi}$, its corresponding output stream $\bar{\psi}$ and a dependency distance $d \in \mathbb{N}$ between corresponding input-output elements, the maximum latency $l \in \overline{\mathbb{R}}_{\max}$ is defined as*

$$l = \max_{i \in \mathbb{N}} \left(\bar{\psi}_{i+d} - \bar{\phi}_i \right) \tag{7}$$

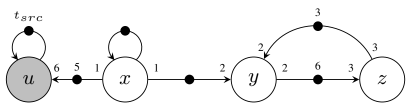
where $\bar{\psi}_{i+d}$ is the corresponding output element of input $\bar{\phi}_i$.

In Definition 4, the dependency distance $d \in \mathbb{N}$ denotes a delay where d output elements are produced even before the first input arrives. This is due to the initial state of the application. This section discusses the construction of input and output streams from scenario graphs and the identification of the causal relations between elements of the streams.

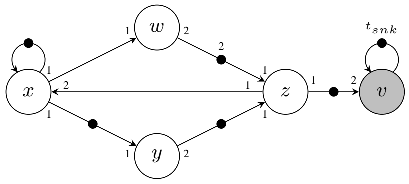
A. Causality in Input-Output Pairs and Dependency Distance

Given an application mapping (S, f, μ), the input stream $\bar{\phi}$ is constructed from the firings of a single *source actor* a_{src} and the output stream $\bar{\psi}$ is constructed from the firings of a single *sink actor* a_{snk}. We refer to a scenario that a_{src} belongs to as a *source scenario* $s_{src} \in S$. Likewise, we refer to a scenario that a_{snk} belongs to as a *sink scenario* $s_{snk} \in S$. There can be multiple source and sink scenarios, since source and sink actors may be active in multiple scenarios. A scenario can also be at the same time a source and a sink scenario.

We require the source and sink actors to fire exactly once per iteration to enable identification of causally-related source and sink actor firings. Hence, the repetition factors of the source and sink actors should be 1 in the repetition vectors of their corresponding scenario graphs. Otherwise, new actors, which serve as source and sink actors, can be introduced with appropriate port-rates. The new actors ensure a repetition factor of 1, as shown in Figure 6. In the figure, actors u and v are introduced to record the firing times of actors x and z, respectively. The original source and sink actors (i.e. x and z) fire 6 and 2 times per iteration, respectively. The number of



(a) Source scenario. Actor u is the source actor.



(b) Sink scenario. Actor v is the sink actor.

Fig. 6: Example source and sink scenarios. Actors u and v are added to construct input and output streams, through the time-stamps of tokens t_{src} and t_{snk}.

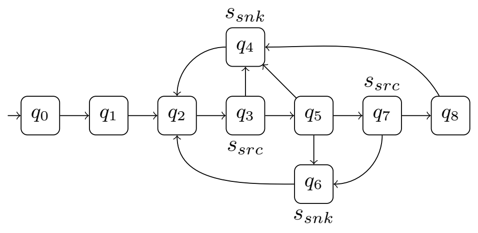
initial tokens on the channel c_{xu} determines which one of the 6 firings of actor x is recorded by the production time of token t_{src}. In the example, this would be the first firing. A similar approach also works for channel c_{zv}. The newly added actors u and v become the source and sink actors, by replacing the original source and sink actors x and z. The time-stamps of token t_{src} constitute the input stream $\bar{\phi}$. Similarly, the time-stamps of token t_{snk} constitute the output stream $\bar{\psi}$.

Dependency Distance: Consider the execution of a sequence of scenarios $\langle s_{src}, s_1, s_2, \cdots, s_{snk} \rangle$ that begins with a source scenario and ends with a sink scenario and no other source or sink scenario in between. A firing of the source actor eventually influences some firing of the sink actor. However, the influence does not necessarily occur in a single execution of the sequence. This is due to initial tokens that may exist in the graph, which may cause the sink actor to fire, even before the source actor fires. We refer to the number of iterations of the sink scenario before a source firing influences the sink actor as its *dependency distance*. E.g. assume Figure 6a is both a source and a sink scenario, where x and z are source and sink actors, respectively. z can fire twice and complete its firings of an iteration before x fires. Token t_{snk}, which can have only one time-stamp per iteration, records one of these two firings of z. This results in a dependency distance of 1, where the first firing of x influences only the second time-stamp of t_{snk}. If there are multiple scenarios between a source and a sink, the computation of the dependency distance becomes more involved (cf. Section VII-C for further discussion).

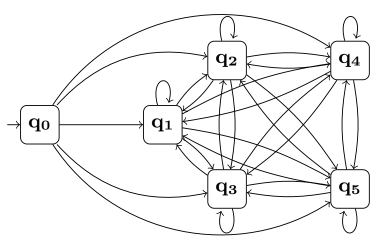
B. Latency Automaton, State-sequence and Latency-sequence

We perform latency analysis by considering scenario sequences allowed by the FSM. However, the FSM can potentially specify infinitely many and infinitely long scenario sequences, due to cycles in the FSM. Scenario sequences that indefinitely stay within cycles without reaching a sink scenario may have no practical relevance. The latency of such sequences is also unbounded. This section discusses a *proper latency automaton* that suffices to guarantee a bounded latency.

A common property of FSMs of real-life streaming applications is a *recurrent state* $q_r \in Q$. Streams are processed in fragments of data such as packets and frames. Applications often start processing a data fragment at a particular defined state. Then, they go through a sequence of states to process the data fragment and, eventually, return back to the recurrent state



(a) A proper latency automaton



(b) A condensed FSM of Figure 7a

Fig. 7: A proper latency automaton and its condensed FSM

to process the next data fragment. However, the sequence of states an application goes through may vary with the content, size or type of the data fragment to be processed.

We define a *state-sequence* as a finite-length sequence $\langle q_r, q_1, q_2, \cdots, q_n\rangle$ of states that starts with the recurrent state q_r. Section VI later presents the latency analysis of FSM-SADF whose FSM specifies a finite number of state-sequences with a common recurrent state. This requirement is formally given in Definition 5. It defines a proper FSM for latency analysis, which has a recurrent state q_r such that for every state $q \in Q$, any path starting from q must lead to q_r (i.e. has no infinitely long path, without revisiting q_r).

Definition 5 (Latency Automaton)**.** *Given an application mapping (S, f, μ), a latency automaton of FSM $f = (Q,\ q_0,T,\epsilon)$ is a tuple $(f, Q_\downarrow)$ such that $Q_\downarrow \subseteq Q$ is a set of final states. We say $(f, Q_\downarrow)$ is proper for latency analysis if it recognizes the language $\mathcal{L} = \{q_0T(q_rP)^* \in Q^* \mid T, P \in Q^* : |T|, |P| \leq m\}$, where $q_r \in Q$ is a recurrent state, Q^* is the set of strings over Q and $m \in \mathbb{N}^0$.*

According to Definition 5, a proper latency automaton specifies transient sequences of states (denoted by q_0T), followed by repetitive concatenation of state-sequences (denoted by q_rP). In the sequel, we use the term state-sequence to refer to also transient sequences. Figure 7a shows an example of a proper latency automaton. It has six state-sequences, where q_2 is the recurrent state: $\langle q_0, q_1\rangle$, $\langle q_2, q_3, q_4\rangle$, $\langle q_2, q_3, q_5, q_4\rangle$, $\langle q_2, q_3, q_5, q_6\rangle$, $\langle q_2, q_3, q_5, q_7, q_6\rangle$ and $\langle q_2, q_3, q_5, q_7, q_8, q_4\rangle$. Among the six, q_0q_1 is the only transient sequence.

A state-sequence may contain source and sink scenarios in different possible orders. We define a *latency-sequence* recursively as a sub-sequence of a state-sequence such that the first state is labeled with a source scenario, the last state with a sink scenario and there is no other latency-sequence in between. The first and the last states of a latency-sequence can be conveniently referred to as source and sink states, respectively. Some examples of latency-sequences of Figure 7 are q_3q_4, $q_3q_5q_4$, $q_3q_5q_6$ and q_7q_6.

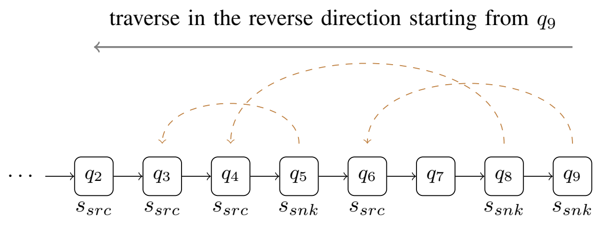


Fig. 8: Identifying latency-sequences in a state-sequence

A state-sequence may have multiple latency-sequences. These latency-sequences can be identified by associating or pairing each sink state with a source state. To do the pairing, we start from the last sink state and go in the reverse direction, as shown by the example of Figure 8. The last sink state is paired with the first source state that is encountered while going in the reverse direction. In Figure 8, this gives the pair (q_9, q_6), which defines the latency-sequence $q_6q_7q_8q_9$. By moving further in the reverse direction, the next sink state is paired with the first source state, which is not yet paired with another sink state. This gives the pair (q_8, q_4), which defines the latency-sequence $q_4q_5q_6q_7q_8$. If the number of source and sink states are not properly matched, some source or sink states will be left unpaired (in front of the other source-sink pairs). The implication of such unpaired states in the latency analysis is discussed in Section VII-D.

C. Condensed FSM

A proper latency automaton has a *condensed FSM* representation, which is constructed by replacing each state-sequence by a new FSM state. Figure 7b shows the condensed FSM of the latency automaton of Figure 7a, which has 6 state-sequences. Each of these state-sequences corresponds to a state in the condensed FSM of Figure 7b, where $\mathbf{q_0} = \langle q_0, q_1\rangle$, $\mathbf{q_1} = \langle q_2, q_3, q_4\rangle$, $\mathbf{q_2} = \langle q_2, q_3, q_5, q_4\rangle$, $\mathbf{q_3} = \langle q_2, q_3, q_5, q_6\rangle$, $\mathbf{q_4} = \langle q_2, q_3, q_5, q_7, q_6\rangle$ and $\mathbf{q_5} = \langle q_2, q_3, q_5, q_7, q_8, q_4\rangle$. $\mathbf{q_0}$ is a transient state. The other states are fully connected, which implies that they may occur in any arbitrary order.

Algorithms 1 and 2 outline the construction of the states of a condensed FSM. The algorithms do not consider transient states for readability reasons. All transient states can be simply found from paths that start with the initial state of the FSM and end at the recurrent state q_r. Algorithm 1 starts from the recurrent state q_r and searches for state-sequences in every outgoing transition t (line 3). Each outgoing transition potentially leads to a new state-sequence. Hence, a new state-sequence is started for each outgoing transition (line 4) that begins with q_r (line 5). Then, the search for valid state-sequences continues from the destination state of transition t (line 6) using the recursive procedure of Algorithm 2.

Algorithm 2 recursively searches for state-sequences starting from a given transition t of destination state q_s (line 2). Every outgoing transition of state q_s (line 12) potentially starts a new state-sequence (line 13). The new state-sequence basically contains all sequences that led upto, and including, state q_s (line 14) and (line 15). Then, the search continues from each of the immediately reachable states of q_s (line 16). A state-sequence is found if a path reaches a recurrent state q_r (line 3-6). A path may also return to an already traversed state before reaching the recurrent state (line 8). This indicates

Algorithm 1 Construct the states of a condensed FSM

```
1: CondensedFSMStates ($f = (Q, q_0, T, \epsilon), q_r$)
2: $\mathbf{Q} \leftarrow$ the set of state-sequences
3: for every outgoing transition $t = (q_r, q_s)$ of $q_r$ do
4:     $\mathbf{q_r}$ := new state-sequence
5:     push back $q_r$ to $\mathbf{q_r}$
6:     findStateSequences($\mathbf{Q}, \mathbf{q_r}, q_r, t$)
7: end for
8: return Q
```

Algorithm 2 Find state-sequences from a given transition t

```
1: findStateSequences ($\mathbf{Q}, \mathbf{q}, q_r, t$)
2: $q_s$ := destination state of tranition $t$ of FSM $f$
3: if $q_s$ equals $q_r$ then
4:     $\mathbf{Q} := \mathbf{Q} \cup \{\mathbf{q}\}$
5:     return terminate path          //state-sequence found
6: end if
7: for every $q'$ in $\mathbf{q}$ do
8:     if $q'$ equals $q_s$ then
9:         return search fail         //unbounded loop detected
10:    end if
11: end for
12: for every outgoing transition $t = (q_s, q_u)$ of $q_s$ do
13:    $\mathbf{q_s}$ := new state-sequence
14:    copy $\mathbf{q}$ into $\mathbf{q_s}$
15:    push back $q_s$ at the end of $\mathbf{q_s}$
16:    findStateSequences($\mathbf{Q}, \mathbf{q_s}, q_r, t$)
17: end for
```

a cycle that creates an infinitely long (or unbounded) state-sequence. In this case, the algorithm terminates unsuccessfully (line 9), as the input FSM is not a proper automaton.

If the algorithm succeeds, it returns a set of state-sequences. Each state-sequence forms a state of the condensed FSM. Each state of the condensed FSM is annotated with a matrix, which is the matrix product of the state-sequence matrices. A state-sequence $\langle q_r, q_1, q_2, \cdots, q_n\rangle$ has a corresponding scenario sequence $\langle s_r, s_1, s_2, \cdots, s_n\rangle$, where $s_x = \epsilon(q_x)$ is the scenario associated with state q_x. The execution of this scenario sequence from a time-stamp vector $\bar{\gamma}_{k-1}$ yields a new vector $\bar{\gamma}_k = M_n \cdot M_{n-1} \cdots M_1 \cdot M_r \cdot \bar{\gamma}_{k-1}$, where M_x is the matrix of the scenario mapping of s_x (cf. Section IV). Due to associativity of $(max, +)$ matrix multiplication, $\bar{\gamma}_k = M \cdot \bar{\gamma}_{k-1}$, where M is given as $M = M_n \cdot M_{n-1} \cdots M_1 \cdot M_r$. This way, each state of the condensed FSM gets a matrix that relates the end-to-end timing behavior of its scenario sequence.

Once a condensed FSM is constructed and its states are annotated with matrices, a bound to the maximum end-to-end latency is derived with respect to a given source actor and a sink actor. Any actor, except the source actor, can be a sink actor. However, the source actor is treated differently. A source is an external entity from an application's perspective. The arrival of input data objects from a given source, such as the RF frontend of a baseband modem, can be periodic or aperiodic. Section VI next derives a bound to the maximum latency under a periodic source. The analysis of aperiodic sources is handled by reduction to a periodic reference source using the approach of [8], as discussed in Section VII-A.

VI. Analysing Latency under a Periodic Source

This section analyses latency under a periodic source for the *basic case*. The basic case assumes a dependency distance of 0 (cf. Section V-A) and a maximum of one latency-sequence per state-sequence (cf. Section V-B). Section VII discusses later the general case, where these assumptions do not hold.

A periodic source is characterized by a period $p \in \mathbb{N}$ between consecutive firings. In dataflow graphs, a periodic source can be modeled by a SDF actor, with a self-edge, as shown by actor x of Figure 6. The period of the source is the WCET of the actor; i.e. $p = \chi(x)$. Next, we present two approaches to derive a bound to the maximum latency under a periodic source: *state-space analysis* and *spectral analysis*.

A. State-Space Analysis

An execution of FSM-SADF is an execution of a sequence of states allowed by the condensed FSM. The execution of a state is given by the relation $\bar{\gamma}_k = M \cdot \bar{\gamma}_{k-1}$, where M is the $(max, +)$ matrix of the state (cf. Section V-C). In the basic case, there is at most one latency-sequence per state. If a latency-sequence exists in a state, the source and sink actors fire exactly once during the execution of the scenario sequence of the state. The firing times of these two actors are then recorded by the time-stamp tokens t_{src} and t_{snk}, respectively. We define the latency of a scenario sequence in isolation as the relative timing distance between the time-stamps of the source and the sink time-stamp tokens, i.e. t_{src} and t_{snk}. These two time-stamps form the i^{th} input-output element pair, $\bar{\phi}_i$ and $\bar{\psi}_i$ (cf. Definition 4), since dependency distance $d = 0$ in the basic case. The latency of the scenario sequence is given by Equation (8). Note that $[\bar{\gamma}]_t$ denotes the entry of token t in vector $\bar{\gamma}$, and $\bar{\bar{\gamma}}$ is a normalized vector.

$$\begin{aligned} l_i &= \bar{\psi}_i - \bar{\phi}_i \\ &= [\bar{\gamma}_k]_{t_{snk}} - [\bar{\gamma}_k]_{t_{src}} \\ &= [\bar{\bar{\gamma}}_k + \|\bar{\gamma}_k\|]_{t_{snk}} - [\bar{\bar{\gamma}}_k + \|\bar{\gamma}_k\|]_{t_{src}} \\ &= [\bar{\bar{\gamma}}_k]_{t_{snk}} - [\bar{\bar{\gamma}}_k]_{t_{src}} \end{aligned} \quad (8)$$

Equation (8) needs to be computed for all reachable time-stamp vectors to find the maximum latency. The set of all reachable time-stamp vectors can be found using *state-space exploration*. The state-space is constructed in a breadth-first-search manner from the condensed FSM, following the approach of [4]. A state of the state-space consists of a state-vector pair $(\mathbf{q}, \bar{\gamma}_{k-1})$, where $\mathbf{q}$ is a state of the condensed FSM, executed from a time-stamp vector $\bar{\gamma}_{k-1}$. Executing $\mathbf{q}$ from vector $\bar{\gamma}_{k-1}$ yields a new vector $\bar{\gamma}_k$. Since the FSM is non-deterministic, there may be multiple outgoing transitions from state $\mathbf{q}$. This results in multiple state-vector pairs $(\mathbf{q}', \bar{\gamma}_k)$, where $\mathbf{q}'$ is a directly reachable state from $\mathbf{q}$. Continuing the execution from each of these new pairs yields either new or already-visited pairs. If there are no more new pairs, the exploration is terminated and the set of reachable time-stamp vectors are finite. The maximum latency is then derived from the set S_γ of all reachable vectors $\bar{\gamma}_k$, as given by Equation (9).

$$l = \max_{\bar{\gamma}_k \in S_\gamma} ([\bar{\bar{\gamma}}_k]_{t_{snk}} - [\bar{\bar{\gamma}}_k]_{t_{src}}) \quad (9)$$

Figure 9 shows the state-space of WLAN where the payload is limited to 2 symbols. This gives three state-sequences: $\langle q_0, q_1, q_2, q_2, q_3\rangle$, $\langle q_0, q_1, q_2, q_3\rangle$ and $\langle q_0\rangle$. The initial state $\mathbf{I}$ enables to execute all the three state-sequences starting from

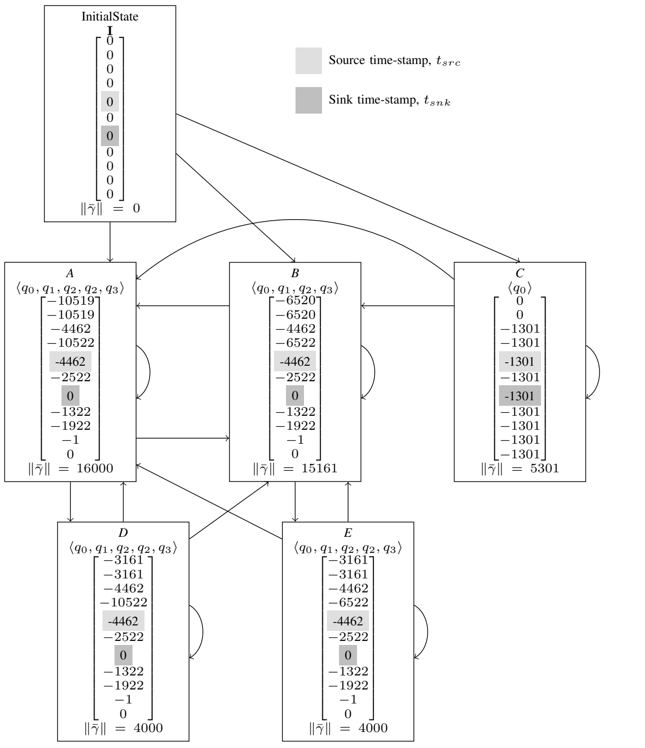


Fig. 9: State-space of WLAN with a max payload of 2 symbols

a zero vector. The execution results in three new vectors, A, B and C. Further executions give two more new vectors D and E. Additional executions do not lead to new vectors and, as a result, the state-space terminates. For this state-space, Equation (9) gives a latency bound of $4462\mu s$.

B. Spectral Analysis

The state-space approach gives the exact worst-case latency. However, it may suffer from state-space explosion [33], as the number of states increases. As a relatively more scalable alternative, we present the spectral analysis approach. The basis of the analysis is the recurrent execution $\bar{\gamma}_k = M \cdot \bar{\gamma}_{k-1}$, where $\bar{\gamma}_0$ is the initial time-stamp vector. The execution of the recurrence relation is terminated once the time-stamp vector returns to a recurrent vector i.e. $\bar{\bar{\gamma}}_k = \bar{\bar{\gamma}}_{k-n}$ for some $n \in \mathbb{N}$. This is because the execution enters a periodic phase and the vectors will repeat themselves afterwards [9]. Such recurrence equations are used in the literature [9] to compute the eigenvalue of the matrix. In [4], the technique is applied to analyse the worst-case throughput of scenarios under a fully-connected FSM, since the eigenvalue gives the maximum cycle mean, which determines the worst-case throughput. This is achieved by constructing a single-matrix M representation, applying the max operator on the matrices of the individual scenarios. In this paper, we adopt the technique for latency analysis. First, the single-matrix representation is constructed, taking the maximum of the matrices of all scenario sequences (or states of the condensed FSM). I.e. $M = \max_{\mathbf{q} \in \mathbf{Q}} M_{\mathbf{q}}$ where $M_{\mathbf{q}}$ is the matrix of state $\mathbf{q}$ and $\mathbf{Q}$ is the set of states of the condensed FSM. Once such a matrix is constructed, the latency analysis is carried out through the recurrence equation. However, unlike throughput, the analysis is not an eigenvalue computation. Instead, the maximum latency is computed according to Equation (9), where S_γ is the set of all encountered vectors until the periodic phase is reached.

VII. Extensions

In Section VI, the analysis is presented for a periodic source, where the basic case is assumed (dependency distance of 0 and at most one latency-sequence per state-sequence). This section discusses the extension of the analysis for a more general case.

A. Aperiodic Sources

It is shown in [8] that the latency of aperiodic sources, such as sporadic and bursty sources, can be analysed by reduction to a periodic reference. The same approach can also be applied to FSM-SADF models. In this section, we show the analysis for the case of a sporadic source. A sporadic source produces events at arbitrary moments, but with some minimum inter-arrival time of $d \in \mathbb{N}$, as shown in Figure 10. The corresponding events of a reference periodic source are also shown in dashed lines, whose period equals the minimum inter-arrival time d.

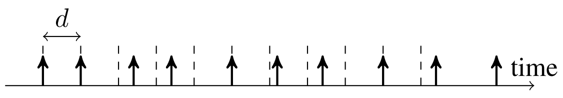


Fig. 10: Sporadic events with minimum inter-arrival time of d

The *linear timing* property of FSM-SADF enables to analyse latency under a sporadic source using a reference periodic source, by applying the same technique as [8]. Linear timing states that the production times of tokens cannot be delayed by more than the delays in the availability of input tokens [2], [31]. The linearity property of FSM-SADF can be seen from the linearity of $(max, +)$ matrices, as shown in Equation 10, where $\bar{\delta} \in \mathbb{R}^n_{\max}$ is a delay vector, whose entries are all non-negative; i.e. $\bar{\delta} \succeq \mathbf{u}[0]$.

$$M \cdot (\bar{\gamma} + \bar{\delta}) \preceq M \cdot \bar{\gamma} + \|\bar{\delta}\| \qquad (10)$$

Equation 10 shows that FSM-SADF has linear timing property, since M can also denote the matrix of a sequence of scenarios. The maximum latency under a sporadic source is then at most equal to the maximum latency computed for the reference periodic source. If an input from a sporadic source arrives later than its periodic reference by $\|\bar{\delta}\| \geq 0$ time units, the firing of the sink will be delayed in the worst-case by $\|\bar{\delta}\|$, due to linear timing. Thus, the relative maximum timing distance between the sink and the source time-stamp tokens does not increase; i.e. $(t_{snk} + \|\bar{\delta}\|) - (t_{src} + \|\bar{\delta}\|) \leq t_{snk} - t_{src}$.

B. Multiple latency-sequences per state-sequence

In this case, the computed latency from Equation 8 would be based on only the last latency-sequence of the state-sequence. The other latency-sequences would not be part of the analysis, since the source and sink time-stamp tokens t_{src} and t_{snk} keep the record of the last firings of the source and sink actors by overwriting previous firings. This may lead to an underestimation of the latency. One solution to handle such cases is to carry out the analysis in multiple steps, by considering only one latency-sequence at a time. For instance, Figure 8 shows an example of a state-sequence that has three latency-sequences. In this case, the analysis is carried out three times. First, only (q_3 and q_5) are labeled as source and sink scenarios. In the second and the third analyses, only (q_4 and q_8) and (q_6 and q_9) are labeled as source and sink scenarios, respectively. The maximum latency is then obtained from the maximum among these three possibilities.

C. Non-zero dependency distance

If the dependency distance $d > 0$, the $(i+d)^{th}$ output element $\bar{\psi}_{i+d}$ is causally related to the i^{th} input element $\bar{\phi}_i$. The first d sink firings do not have corresponding source firings and, hence, latency cannot be defined for them. As a result, the analysis can be reduced to the case where the dependency distance is zero. This can be achieved by starting the analysis from the state of the application where the first d sink firings are completed. To realize this, each state-sequence needs to be executed while blocking (not firing) the source actor until the sequence deadlocks. The resulting scenario graphs get new distributions of initial tokens, where the first sink firing has a causal dependency with the first source firing.

D. Unpaired source and sink scenarios in a state-sequence

It is possible that some source (or sink) scenarios can be left without being paired with sink (or source) scenarios. Such unpaired scenarios are always in front of the other source-sink pairs of a state-sequence. The analysis of Section VI permits unpaired source scenarios. (A good example is the case of WLAN, which is discussed in Section VIII-A.) This is because all unpaired source scenarios are in front of the latency-sequences of a state-sequence. As a result, the source time-stamp token t_{src} gets overwritten as these latency-sequences are executed later in the sequence. Hence, the timing distance between t_{src} and t_{snk} eventually depends only on the latency-sequences (i.e. the paired source-sink scenarios). However, it is not possible for some sink scenarios to be left unpaired, since this would imply a non-zero dependency distance, where a sink actor can fire before the source fires.

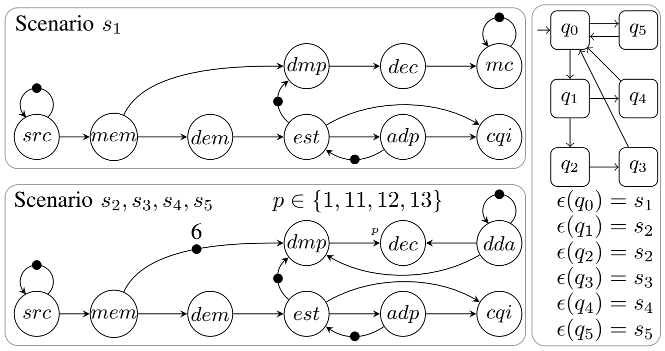
VIII. Evaluation

This section evaluates the presented analysis techniques. Section VIII-A and VIII-B analyze the latencies of two wireless applications: WLAN 802.11a and 3GPP's LTE. Section VIII-C binds these applications onto a multi-core platform model and discusses trade-offs between resource allocation and achievable temporal bounds. Section VIII-D evaluates run-time and scalability. The techniques are implemented in a publicly available dataflow analysis tool-set, SDF3 [30]. The experiments are carried out in a 2.4GHz dual-core Linux PC.

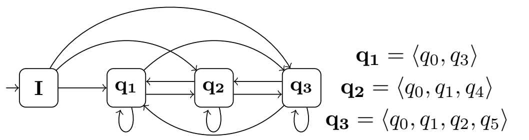
A. WLAN 802.11a baseband processing

The payload size of a WLAN packet may vary from 1 to 256 OFDM symbols, each of which is $4\mu s$ long (cf. Figure 3). Since packets arrive sporadically, the maximum latency is equivalent to the case of a periodic source of period $4\mu s$ (cf. Section VII-A). After these symbols are demodulated and decoded, CRC is performed. If CRC is successful, an acknowledgment must be sent within $16\mu s$ of the end of reception. This defines a latency requirement between the reception of the last symbol and the sending of the acknowledgement. Hence, the source and sink actors are actors src and spd, where the source scenario is s_3 i.e. $\epsilon(q_2)$ and the sink scenario is s_4 i.e. $\epsilon(q_3)$. Scenario s_3 is executed as many times as the number of OFDM symbols. The FSM of Figure 3 conservatively approximates this behavior by allowing an arbitrary number of payload symbols, as indicated by the self-transition in state q_2. As a result of this self-transition, Algorithm 1 fails to construct a valid condensed FSM. To address this issue, the analysis is carried out on the original exact FSM, which has 256 FSM states labeled with scenario s_3. The 256 states have the form q_2^i, where i runs from 0 upto 255. This results in a condensed FSM that has 257 state-sequences. One state-sequence is $\langle q_0 \rangle$, which captures the repeated execution of scenario s_1 until synchronization succeeds. The remaining 256 sequences have the form $\langle q_0, q_1, q_2^0, q_2^1, \cdots, q_2^i, q_3 \rangle$, which specifies the scenario sequence needed to decode a packet of length $i+1$ symbols. In these scenario sequences, only the last source scenario at q_2^i is paired with a sink scenario, while the other source scenarios are unpaired. This is allowed by the analysis as discussed in Section VII-D.

The latency of the model has been analysed with both the state-space (SS) and the spectral analysis (SA) techniques. The constructed state-space has 514 states (state-vector pairs). The obtained maximum latency bound equals $4.46\mu s$ and the analysis took $1200ms$. In fact, all visited states, which have a latency-sequence, have given the same latency bound as the maximum latency. The SA also gives the same latency bound in this case and its analysis run-time is only $84ms$.



(a) A dynamically changing graph structure of LTE


I
$\mathbf{q_1}$
$\mathbf{q_2}$
$\mathbf{q_3}$
$\mathbf{q_1} = \langle q_0, q_3 \rangle$
$\mathbf{q_2} = \langle q_0, q_1, q_4 \rangle$
$\mathbf{q_3} = \langle q_0, q_1, q_2, q_5 \rangle$

(b) condensed FSM

Fig. 11: FSM and condensed FSM of LTE

B. Long Term Evolution (LTE) baseband processing

LTE is a recent (pre-4G) standard in cellular wireless communication. LTE's downlink frame consists of 10 sub-frames. Each sub-frame is $1000\mu s$ long and has 14 OFDM symbols. Hence, the source fires periodically with a period of $1000\mu s/14 = 71.4\mu s$. The 14 symbols are allocated to different data and control channels. The three possible channel allocations are 1, 2 or 3 control channels, followed by 13, 12, or 11 data channels respectively. The FSM-based SADF model of LTE comprises five scenarios $s_1 - s_5$ [33]. A compact version of this model is shown in Figure 11a.

Scenario s_1 decodes a control format indicator channel that is allocated to symbol 1. Scenario s_2 decodes a control channel that is allocated to symbols 2 and 3. Scenarios s_3, s_4 and s_5 decode a data channel that is allocated from symbol 2 to 14, 3 to 14, and 4 to 14, respectively. Sub-frame processing always starts at scenario s_1 (i.e. source scenario) and terminates at s_3, s_4 or s_5 (i.e. sink scenarios). This results in three possible state-sequences (one for each of the three possible sub-frame types), as shown by the condensed FSM of Figure 11b.

All the three state-sequences of LTE are *unbounded* for the given source period. This means that the state-sequences are slower than the rate of the source. As a result, the temporal distance between the source and the sink diverges indefinitely and the latency becomes unbounded. This requires speeding-up critical actors to get the state-sequences bounded. A state-sequence is bounded if and only if its matrix has a cycle-time vector [5] whose entries are all the same. This indicates that all actors have the same execution rate. For instance, if actor dec is accelerated by a factor of 4, all state-sequences become bounded for the source period of $71.4\mu s$ and the application achieves the throughput constraint set by the source. In this case, the derived latency bound is $1568\mu s$, according to both the SA and SS techniques. The state-space has 13 states and the maximum latency is observed for the sequence $\langle q_0, q_5\rangle$. The run-time is under $12ms$ for both techniques.

If critical actors, such as actor dec, are not accelerated, the minimum period the graph can support is only $127\mu s$. Any period lower than this becomes too fast to be supported by the application. When the application is run under this minimum period, the latency bound is $3752\mu s$, according to both SS and SA techniques. The state-space has 157 states. The maximum latency is observed for state-sequence $\langle q_0, q_5\rangle$ and the other sequences have a lower latency. The analysis run-time is under $20ms$ for both techniques.

Conservativeness of the spectral analysis: The computed latencies in the above cases are the same for both SS and SA techniques. This holds in all cases where the condensed FSM is fully connected and has no transient sequences. Otherwise, the SS gives the exact worst-case latency, while the SA gives only a conservative bound. Table II shows one such case. The evaluation is based on the LTE model of Figure 11, where the FSM is modified to make sequence $\langle q_0, q_5\rangle$ transient and the other two sequences periodic. Moreover, actor dec is accelerated to keep the latency bounded. The acceleration factor is $\frac{1}{4}\times\chi(dec)$ in all scenarios except in scenario s_5, for which it is varied as shown in the first column.

TABLE II: Conservativeness of the spectral analysis

Acceleration factor in Scenario s_5	$\chi(dec)$	Latency SS (μs)	Latency SA (μs)	Difference (%)
$\frac{1}{1}\times\chi(dec)$	970	-	-	-
$\frac{5}{6}\times\chi(dec)$	808	2700	3212	18.9
$\frac{4}{5}\times\chi(dec)$	776	2636	3084	16.9
$\frac{3}{4}\times\chi(dec)$	727	2538	2888	13.7
$\frac{2}{3}\times\chi(dec)$	646	2376	2564	7.9
$\frac{1}{2}\times\chi(dec)$	485	2054	2054	0
$\frac{1}{4}\times\chi(dec)$	242	1568	1568	0

C. Resource Reservation vs. Temporal Bound Trade-offs

This section demonstrates how the presented latency analysis techniques can be used for a trade-off analysis between resource reservation and achievable temporal bounds for a software-defined radio (SDR) design using the same WLAN and LTE FSM-SADF models. A SDR comprises a heterogeneous MPSoC platform that is shared between multiple wireless radio applications. For instance, the baseband processor of high-feature phones is a MPSoC [34] that supports multiple applications, such as WLAN for wireless connectivity and WiMax and LTE for 3G/4G cellular connectivity.

TABLE III: Constructed actor-to-processor bindings

Appl.	GPP1	VP1	VP2	VP3	WPA1	WPA2	GPP2
			Binding-1				
WLAN	*mc, dda1, dda2, crc*	*shift, sync, hdem, pdem*	-	-	*hdec, pdec*	-	*ack, spld, shdr*
LTE	*mc,dda, mem, dem*	*est, adp, cqi, dmp*	-	-	-	*dec*	-
			Binding-2				
WLAN	*mc, dda1, dda2, crc*	*shift, sync*	*hdem*	*pdem*	*hdec, pdec*	-	*ack, spld, shdr*
LTE	*mc,dda, mem, dem*	*est*	*adp, cqi*	*dmp*	-	*dec*	-

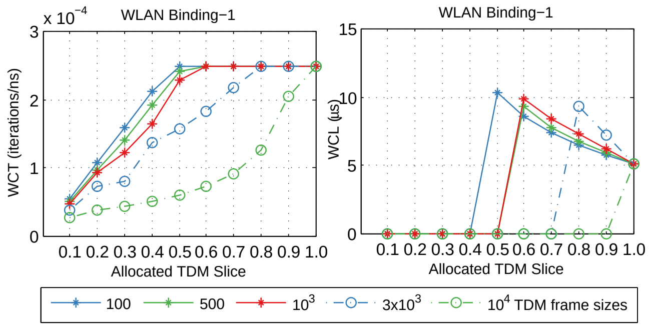


Fig. 12: Timing Analysis of WLAN Mapping

1) Platform: A typical SDR MPSoC (cf. Figure 1) comprises a general-purpose processor (GPP) (e.g. ARM) for control and generic tasks, a set of vector processors (VPs) (e.g. EVP [35]) for tasks such as synchronization and demodulation, and a set of weakly-programmable hardware accelerators (WPAs) for tasks such as Turbo decoding. Each processor tile has a local data and instruction memory, but has no cache. It is assumed these processors are connected through a predictable interconnect [36] that uses a contention-free routing, based on TDM switching. Hence, each connection has a guaranteed bandwidth and maximum latency. For all connections, we assume the same bounded delay WCRC (cf. Figure 4b).

2) Application mapping: We constructed two sets of actor-to-processor bindings, where control-oriented tasks are mapped on the GPP and signal processing tasks are mapped on the VPs, as shown in Table III. A static-order schedule is constructed per tile between actors of the same application. The buffer-size of each channel is set to a fixed value, which equals twice the total number of tokens produced on the channel in one iteration. We set the buffer-sizes to fixed values in order to control the effects of variable buffer-sizes on the temporal analysis. Both applications are intended to run together on the same platform, where TDM scheduling is used to arbitrate the processor tiles between them.

3) Temporal Analysis: Figure 12 shows the worst-case throughput (WCT) and the worst-case latency (WCL) of WLAN Binding-1 for different TDM frame sizes and slice allocations. The results for Binding-2 are similar to Binding-1, as the serial dependencies from actor sft to $hdem$ and $pdem$ limit potential gains from parallelization. The required throughput is 2.5×10^{-4} per ns. The figure on the left shows that this throughput is not met until the slice allocation is sufficiently large. A bound to the maximum latency does not also exist if the throughput requirement is not met. E.g. for a frame size of 100, TDM slices less than 0.5 make the application unbounded. Once the throughput requirement is met, increasing the slice allocation does not lead to a further increase in throughput but helps only to reduce the latency.

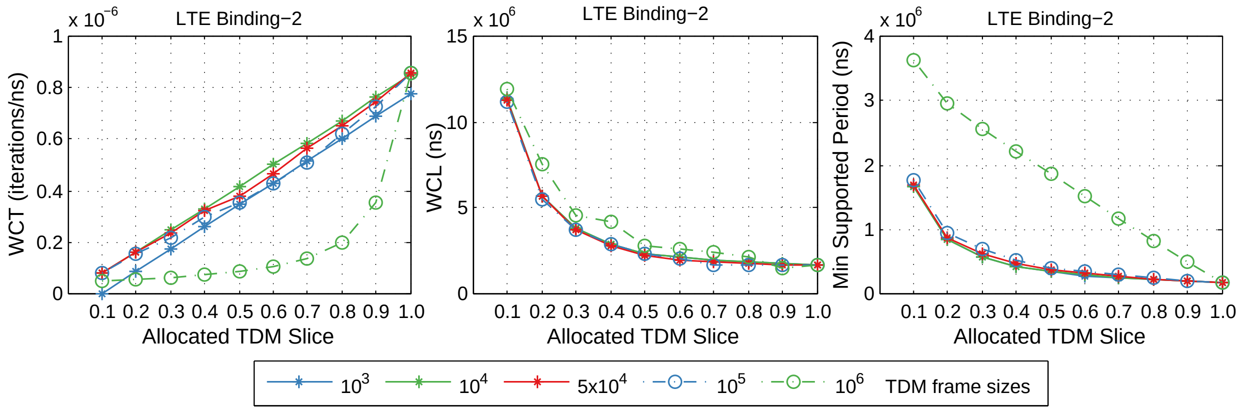


Fig. 13: Latency analysis of LTE mapping

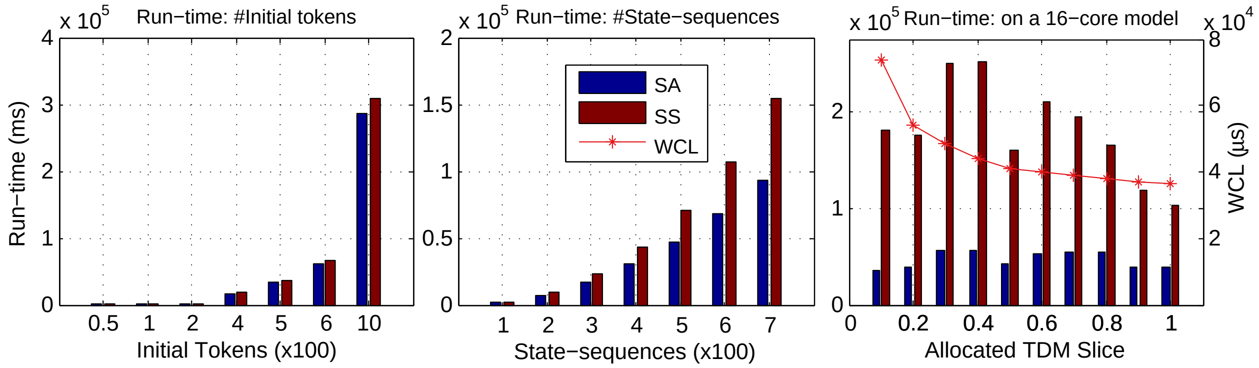


Fig. 14: Analysis run-time for large problem sizes

Figure 13 shows the timing analysis of the LTE processing in Binding-2. This binding gives almost twice the performance of Binding-1 (not shown) for a similar resource reservation. The left-most figure shows that a TDM frame size of $10^4 ns$ gives the best performance. Shorter frame sizes have lower performance due to extra overhead from frequent context switching. Longer frame sizes (e.g. $10^6 ns$), on the other hand, have lower performance due to longer response times. However, this frame size of $10^4 ns$ gives a degraded performance for WLAN if both applications are running together. As a compromise, one may select a frame size of $10^3 ns$.

None of the resource reservations of the two LTE bindings satisfies the throughput requirement, which is dictated by the period of the source, $71.4\mu s$. Thus, a bound to the maximum latency also does not exist for this source period. The middle and the right-most figures of Figure 13 show the maximum latency and the minimum supported period, respectively, for different frame sizes and TDM slice allocations. For instance, for a TDM frame size of $10^3 ns$ and allocated slice of 0.7, the minimum supported period equals $238\mu s$ and a bound to the maximum end-to-end latency equals $1982\mu s$.

D. Scalability

Two key parameters that influence the run-time of the analysis techniques are the number of state-sequences and the size of the $(max, +)$ matrices. The number of state-sequences is the same as the number of states of the condensed FSM. The size of the matrices (i.e. $n \times n$) is determined by the total number n of initial tokens. The two left-most plots of Figure 14 show how the run-time of the analysis scales with increasing number of initial tokens and state-sequences. The experiments are conducted by randomly adding synthetic initial tokens and FSM transitions in the LTE model of Figure 11. The right-most plot of Figure 14 shows the run-times of the SS and SA techniques for a synthetic binding, which is constructed by duplicating LTE Binding-2 of Table III over a 16-core MPSoC platform model, where the two instances of LTE are sequentially connected, the source period is set to $2500\mu s$ and the TDM frame size is $100\mu s$.

The matrix of each scenario is constructed through a symbolic execution of one graph iteration, as presented in Section IV. Given these matrices, the analysis with the SA technique involves three separate steps. The first step is constructing the matrices of the state-sequences, which has a complexity of $\mathcal{O}(l \cdot m \cdot n^3)$, where l is number of state-sequences, m is length of state-sequences and $\mathcal{O}(n^3)$ is the complexity of matrix-to-matrix multiplication. The second step is a max operation on these matrices, which is linear with the number of state-sequences and the size of the matrices. The third step is a spectral (eigenvalue) analysis of the final matrix using Algorithm 3 of [3]. The algorithm is based on the recurrent execution $\gamma_{k+1} = M \cdot \gamma_k$, until a periodic phase is reached. Thus, it has a complexity of $\mathcal{O}(t \cdot n^2)$, where t is the length of the transient phase and $\mathcal{O}(n^2)$ is complexity of matrix-to-vector multiplication.

The analysis with the state-space technique also first requires constructing the matrices of the state-sequences. It further requires a matrix-to-vector computation of complexity $\mathcal{O}(s \cdot n^2)$, where s is the size of the state-space, which is constructed through a breadth-first-search algorithm. As a result, it has time complexity, which is polynomial with the number of initial tokens, similar to the SA technique. However, the size of the state-space may increase exponentially with the number of state-sequences. This is because the execution of a FSM of l states gives at most l and l^l new states, respectively, after the first and second rounds of execution. Consequently, the performance of the state-space technique may degrade with increasing number of states. E.g. in the right-most plot of Figure 14, the state-space has about 10^4 states). However, the results show comparable run-time between the two techniques for increasing number of initial tokens, as shown by the left-most plot. This is justifiable since the number of state-sequences in this experiment is as low as three state-sequences.

IX. Conclusion

We have presented a systematic analytical approach to compute a conservative bound to the maximum end-to-end latency of dataflow applications that are mapped onto a shared heterogeneous platform. The approach targets dynamic application graphs that may change their graph structure as they switch between different modes. A latency bound is derived with respect to a given periodic source, which also sets a throughput constraint. The technique has time complexity, which is polynomial with the number of initial tokens. Thus, reducing the number of tokens becomes crucial for a scalable analysis. To that end, the approach constructs compact $(max, +)$ matrices, which characterize the end-to-end temporal behavior of scenario mappings over a graph iteration. An upper-bound to the maximum latency is then derived through a state-space exploration or spectral analysis over the matrices of the possible scenario sequences. The technique can be combined with existing throughput and boundedness analysis methods for resource reservation under real-time constraints. The evaluation results show that the techniques can be effectively integrated in multiprocessor design flows of streaming applications.

References

[1] E. Lee and D. Messerschmitt, "Synchronous dataflow," *Proceedings of IEEE*, 1987.

[2] O. Moreira, "Temporal Analysis and Scheduling of Hard-Real Time Radios on a Multi-processor," Ph.D. dissertation, Eindhoven University of Technology, 2011.

[3] M. Geilen, "Synchronous dataflow scenarios," *ACM Transactions on Embedded Computing Systems*, 2011.

[4] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES/ISSS*, 2010.

[5] F. Siyoum *et. al*, "Worst-case throughput analysis of real-time dynamic streaming applications," in *CODES+ISSS*, 2012.

[6] S. Stuijk *et al.*, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *DSD*, 2010.

[7] A. Ghamarian *et. al*, "Latency minimization for synchronous data flow graphs," in *DSD*, 2007.

[8] O. Moreira *et al.*, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proc. EMSOFT*, 2007.

[9] B. Heidergott *et. al*, *Max Plus at Work: Modeling and Analysis of Synchronized Systems*. Princeton University Press, 2006.

[10] F. Siyoum, M. Geilen, and H. Corporaal, "Symbolic analysis of dataflow applications mapped onto shared heterogeneous resources," in *DAC*, 2014.

[11] R. Henia *et. al*, "System level performance analysis-the SymTA/S approach," *Computers and Digital Techniques, IEEE Proc.*, 2005.

[12] S. Schliecker and R. Ernst, "A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems," in *CODES+ISSS*, 2009.

[13] S. Schliecker, S. Stein, and R. Ernst, "Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis," in *DATE*, 2007.

[14] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, 1994.

[15] P. Pop, P. Eles, and Z. Peng, "Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems," in *DATE*, 2003.

[16] J. Kim *et. al*, "A novel analytical method for worst case response time estimation of distributed embedded systems," in *DAC*, 2013.

[17] S. Perathoner *et. al*, "Influence of different system abstractions on the performance analysis of distributed real-time systems," in *EMSOFT*, 2007.

[18] L. Thiele and N. Stoimenov, "Modular performance analysis of cyclic dataflow graphs," ser. EMSOFT, 2009.

[19] S. Chakraborty *et. al*, "A general framework for analysing system properties in embedded system designs," in *DATE*, 2003.

[20] A. Ghamarian *et. al*, "Throughput analysis of synchronous data flow graphs," in *ACSD*, 2006.

[21] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synch.*, 2009.

[22] Phan, L.T.X. and Insup Lee and Sokolsky, O., "Compositional Analysis of Multi-mode Systems," in *ECRTS*, 2010.

[23] Phan, L.T.X. and Chakraborty, S. and Thiagarajan, P.S., "A Multi-mode Real-Time Calculus," in *Real-Time Systems Symposium*, 2008.

[24] J. Buck, "A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications," in *CODES*, 1994.

[25] E. Lee, "Consistency in dataflow graphs," *IEEE Trans. on Parallel and Distributed Systems*, 1991.

[26] W. Plishker *et. al*, "Functional DIF for rapid prototyping," in *International Symposium on Rapid System Prototyping*, 2008.

[27] P. Wauters *et. al*, "Cyclo-dynamic dataflow," *IEEE Transactions on Signal Processing*, 1996.

[28] A. Girault *et. al*, "Hierarchical finite state machines with multiple concurrency models," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 1999.

[29] G. Khan, "The semantics of a simple language for parallel programming," in *IFIP*, 1974.

[30] S. Stuijk *et. al*, "SDF3: SDF For Free," in *ACSD*, 2006.

[31] M. Wiggers, "Aperiodic multiprocessor scheduling for real-time stream processing applications," Ph.D. dissertation, 2009.

[32] P. Poplavko, "An accurate analysis for guaranteed performance of multiprocessor streaming applications," Ph.D. dissertation, Eindhoven University of Technology, 2008.

[33] F. Siyoum *et. al*, "Analyzing Synchronous Dataflow Scenarios for Dynamic Software-defined Radio Applications," in *SOC*, 2011.

[34] F. Clermidy *et. al*, "A 477mW NoC-based digital baseband for MIMO 4G SDR," in *ISSCC*, 2010.

[35] K. van Berkel *et. al*, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Appl. Signal Process.*, 2005.

[36] K. Goossens *et al.*, "The Æthereal network on chip: Concepts, architectures, and implementations," vol. 22, no. 5, 2005.

Firew Siyoum Firew Siyoum holds a Ph.D., in the areas of model-based design of streaming applications, from Eindhoven University of Technology, The Netherlands. He received his B.Sc. in Electrical Engineering from Mekelle University, Ethiopia, and his M.Sc. in Embedded Systems from Eindhoven University of Technology. His research interests include model-based design, analysis and synthesis of embedded systems and multiprocessor system-on-chips, with a special focus on multimedia and wireless application domains.

Marc Geilen Marc Geilen is an assistant professor in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an M.Sc. in Information Technology and a Ph.D., in the areas of formal verification, from Eindhoven University of Technology. His research interests include modeling, simulation and programming of multimedia systems, multiprocessor systems-on-chip, networked embedded systems and cyber-physical systems, and multi-objective optimization and trade-off analysis.

Henk Corporaal Henk Corporaal is a professor in Embedded System Architectures in the Department of Electrical Engineering at Eindhoven University of Technology. He holds an M.Sc. in Theoretical Physics from University of Groningen, The Netherlands, and a Ph.D. in Electrical Engineering, in the area of Computer Architecture, from Delft University of Technology, The Netherlands. His current research interests are low-power single and multi-processor architectures, their programmability, and the predictable design of real-time systems.