Received June 12, 2020, accepted July 21, 2020, date of publication July 30, 2020, date of current version August 11, 2020.
Digital Object Identifier 10.1109/ACCESS.2020.3013013

Latency Evaluation of SDFGs on Heterogeneous Processors Using Timed Automata

SIVASHANKARI RAJADURAI[1], MAMOUN ALAZAB[2], (Senior Member, IEEE), NEERAJ KUMAR[3,4], (Senior Member, IEEE), AND THIPPA REDDY GADEKALLU[1]
[1]School of Information Technology and Engineering, Vellore Institute of Technology, Vellore 632014, India
[2]College of Engineering, IT and Environment, Charles Darwin University, Casuarina, NT 0909, Australia
[3]Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala 147001, India
[4]Department of Computer Science and Information Engineering, Asia University, Taichung City 41354, Taiwan
Corresponding author: Mamoun Alazab (mamoun.alazab@cdu.edu.au)

ABSTRACT Synchronous Data Flow (SDF) is a graphical computation model used for analyzing digital signal processing and real time multimedia applications. In general, these applications have two primary performance metrics – throughput and latency. Latency is important in multimedia processing applications such as video-conferencing, Internet telephony and games since latency surpassing a specific limit results in poor quality of service (QoS). Past work had focused on computing the latency of SDF graphs on homogeneous multiprocessor platforms. In this paper, we present an approach to compute the latency of a static schedule for a given unfolding factor with an optimal throughput for an SDF graph on a *heterogeneous multiprocessor platform* using timed automata. We use timed automata as a semantic model to represent the system model, which includes a synchronous data flow graph and an execution platform. We use the UPPAAL model-checker to specify the resulting network of timed automata and compute the latency.

INDEX TERMS Synchronous dataflow, throughput, latency, timed automata, UPPAAL.

I. INTRODUCTION

Real-time systems usually have throughput and latency constraints. System designers need to verify whether the system meets these constraints. SDF [11] is a graphical computation model used for analyzing digital signal processing and real time multimedia applications. These applications are often implemented on multiprocessor platforms, under real time and resource constraints [15], [16]. An important problem is to predict the timing behavior of such applications. In general, these applications have two primary performance metrics – throughput and latency. Throughput is the rate at which the system can process inputs. Latency is the duration between receiving a certain input sample and producing the processed sample.

The throughput analysis of Homogeneous SDF (HSDF) graph has been studied in [6], [10]. The throughput analysis of SDF graphs is carried out by transforming an SDF graph to an equivalent Homogeneous SDF (HSDF) graph [11], [13], which leads to a large increase in the size of the graph. In [8] the throughput of SDF graphs is measured by exploring the state-space. The maximal throughput is achieved by firing every actor as soon as it is enabled (known as *self-timed execution*). This approach works directly on an SDF graph, avoiding the conversion to an HSDF graph. For homogeneous multiprocessor platforms self-timed execution [8] gives maximal throughput but it is not true for heterogeneous multiprocessor platforms [17]. The paper [17] presents a method to find the maximal throughput on a heterogeneous multiprocessor platform for a given unfolding factor f.

Latency is important in multimedia processing applications such as video-conferencing, Internet telephony and games since latency surpassing a specific limit results in poor quality of service (QoS). The paper [9] proposed an algorithm that gives the minimum achievable latency between the executions of any two actors in the SDF graph on a homogeneous multiprocessor platform. It also proposed a heuristic that optimizes the latency under a given throughput. The algorithm presented in the paper [9] can not be used to compute the minimum latency for an SDF graph on heterogeneous multiprocessor platform as explained in section II-B.

Ahmad *et al.* [1], Zhu *et al.* [17] and Fakih *et al.* [7] have previously used timed automata (TA) [2] to model the timing behavior of SDF graphs. The paper [1] finds the maximal throughput for an SDF graph. The paper [17] presents a method for scheduling SDF graphs

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Collotta.

on heterogeneous multiprocessor platforms. The schedules are either throughput-optimal with best energy consumption or energy consumption-optimal with the best throughput for a given unfolding factor. The paper [7] finds the timing bounds of multiple (hard real-time) SDF-based applications mapped into a multicore platform. There is no existing work that presents a technique to compute the latency of an SDF graph using timed automata, especially for heterogeneous multiprocessor platforms.

In this paper, we propose an approach to compute the latency of a static schedule for a given unfolding factor with an optimal throughput for an SDF graph on a *heterogeneous multiprocessor platform* using timed automata. This approach can be used to explore the trade-off between throughput and latency by changing the unfolding factor. This approach can also be used to find the minimum achievable latency between the executions of any two actors in the SDF graph on a homogeneous multiprocessor platform.

II. SYNCHRONOUS DATA FLOW GRAPHS (SDFGs)

In DSP and multimedia applications, there are sets of tasks to be executed periodically in a certain order. These tasks consume and produce fixed amounts of data in each execution. An SDFG [11] is a natural model for describing this class of applications. An SDFG is a directed, connected graph consisting of nodes and edges. Nodes model tasks are called *actors* and edges model data flow among actors are called *channels*. The execution of an actor is defined in terms of firing. The data items communicated between actors are called *tokens*. An actor is enabled for firing when enough tokens are available on all of its inputs. When an actor fires it consumes some pre-specified number of tokens from every input channel and produces a pre-specified number of tokens on every output channel.

Let $\mathbb{N}_+$ be the set of positive natural numbers, $\mathbb{N}$ the natural numbers.

Definition 1 (SDFG): An SDFG is a tuple $G = (A, D, C_0)$ where:

- *A is finite set of actors,*
- *D is finite set of channels $D \subseteq A^2 \times {\mathbb{N}_+}^2$,*
- *C_0: $D \to \mathbb{N}$ is a function where $C_0(e)$ is the number of initial tokens on channel e in D.*

For an SDFG (A, D, C_0), a channel $d = (s, t, p, c)$ denotes the data dependency of actor t on actor s, where p and c are the production and consumption rates of tokens of s and t respectively. When actor s fires it produces p tokens on channel d and when actor t fires it consumes c tokens from channel d. For a channel $d = (s, t, p, c)$ in D, the consumption rate of d is defined by $CR(d) = c$ and the production rate of d is defined by $PR(d) = p$. An SDFG where all the production and consumption rates are one is called HSDF.

Definition 2 (Channel State): A channel state of an SDFG (A, D, C_0) is a mapping $C : D \mapsto \mathbb{N}$ that associates with each channel the number of tokens present in that channel at that state. C_0 represents the initial channel state of the SDFG.

For every actor $a \in A$ of an SDFG (A, D, C_0), we denote its set of input (output) channels by $InC(a)$ ($OutC(a)$) where:

$$InC(a) = \left\{(a', a, p, c) \in D \mid a' \in A \wedge p, c \in \mathbb{N}_+\right\}$$
$$OutC(a) = \left\{(a, a', p, c) \in D \mid a' \in A \wedge p, c \in \mathbb{N}_+\right\}$$

Definition 3 (Enabled Actor): An actor $a \in A$ of an SDFG (A, D, C_0) is called enabled in a channel state C_i if $C_i(d) \geq q_i$ for each channel $d = (a', a, p_i, q_i)$ in $Inc(a)$.

Definition 4 (Actor Firing): If an actor $a \in A$ of an SDFG (A, D, C_0) is enabled in channel state C_i and it fires, the resulting channel state C_{i+1} is defined by $C_{i+1}(e) = C(e) - c$ for each input channel $e = (s, a, p, c)$ in $InC(a)$, $C_{i+1}(e) = C(e) + p$ for each output channel $e = (a, t, p, c)$ in $OutC(a)$, and $C_{i+1}(e) = C(e)$ for all remaining channels. The channel state transition is denoted as $C_i \xrightarrow{a_i} C_{i+1}$.

Definition 5 (Execution): An execution σ of an SDFG (A, D, C_0) is an infinite sequence of channel states $C_0, C_1, \cdots$ starting from the initial state C_0, such that $\forall i \geq 0$, $C_i \xrightarrow{a_i} C_{i+1}$.

An SDFG (A, D, C_0) has a deadlock if and only if it has an execution that leads to a channel state C_i where $\forall a \in A$, $\exists d \in InC(a)$ such that $C_i(d) \ngeq CR(d)$.

Definition 6 (Repetition Vector): A repetition vector of an SDFG (A, D, C_0) is a function $\gamma : A \to \mathbb{N}_+$ such that for every channel $(s, t, p, c) \in D$, the following conditions holds

$$p \cdot \gamma(s) = c \cdot \gamma(t)$$

A repetition vector γ is called non-trivial if $\forall a \in A$, $\gamma(a) > 0$. An SDFG is *consistent* if it has a non-trivial repetition vector. A repetition vector indicates the number of times each of the actors of the SDFG must fire so that there is no change in the token distribution. An *iteration* is a set of firings such that for each $a_i \in A$, the set contains $\gamma(a_i)$ firing of a_i. In the rest of the paper, we assume that SDFGs are strongly connected, consistent and deadlock free.

Example 1: Fig. 1(a) shows an SDFG G_1, consisting of three actors, labeled by their names. The numbers at either end of the channel connecting u and v indicate that, when u fires it produces two tokens on this channel whereas when v fires it consumes one token from this channel. Initially channels may contain some tokens, denoted by black dots. The repetition vector of the SDFG G_1 is $\langle(u, 1), (v, 2), (w, 2)\rangle$.

For performance analysis an SDFG is extended with timing information. Let an execution platform P be a set of processors. In an execution platform P, a function $E : A \times P \to \mathbb{N}$ associates with each actor $a \in A$, the amount of time it needs to fire on processor $p \in P$. If processors are identical in P then the execution platform P is called a *homogeneous multiprocessor platform*, else it is called a *heterogeneous multiprocessor platform*.

Definition 7 (System Model): A system model includes an SDFG G and its execution platform P, denoted by $M = (G, P)$.

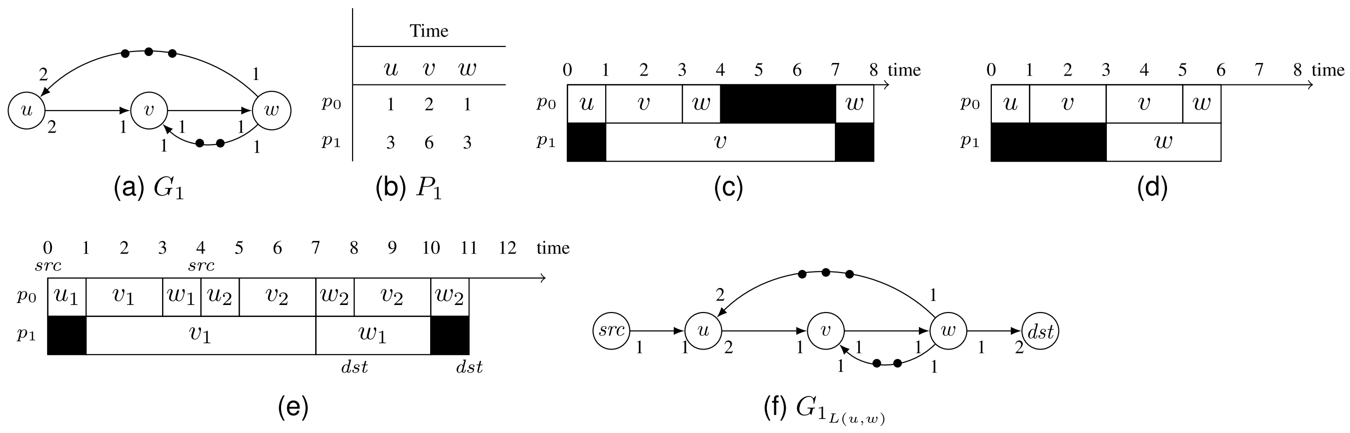


FIGURE 1. The system model M_1 and its schedule (a) The SDFG G_1; (b) the execution platform P_1 and execution time of actors in G_1 on different processor; (c) the Self-timed execution schedule; (d) a periodic schedule; (e) a periodic schedule with unfolding factor 2; (f) Latency Graph $G_{1_{L(u,w)}}$.

A. THROUGHPUT

Definition 8 (Throughput): The throughput of an SDFG G is the long run average of iterations per unit time, i.e.,

$$Throughput(G) = \lim_{t\to\infty} \frac{ite(t)}{t}$$

where $ite(t)$ is the number of iterations in time t.

Given a system model $M = (G, P)$, the *self-timed execution* provides the maximal throughput for an SDFG G on a homogeneous multiprocessor platform [13]. In a self-timed execution all actors fire as soon as they are enabled. For every consistent and strongly connected SDFG, the state-space of a self timed execution consists of a finite sequence of states, called the *transient phase* followed by a sequence of states which is repeated infinitely often, called the *periodic phase* [8]. On a heterogeneous multiprocessor platform the self-timed execution does not provide the maximal throughput [17]. Scheduling f iterations as one schedule cycle may increase the throughput. This schedule is called the *unfolding-schedule* [12] and f is called the *unfolding factor*.

Example 2 ([17]): Fig. 1 shows a system model $M_1 = (G_1, P_1)$, where G_1 is an SDFG shown in Fig. 1(a) and P_1, shown in Fig. 1(b), is a heterogeneous multiprocessor platform with two processors. Fig. 1(c) illustrates the self-timed execution of G_1 on the platform P_1. For this self-timed schedule the throughput of G_1 is $1/8$. Another periodic schedule of G_1 shown in Fig. 1(d). Here the throughput of G_1 is $1/6$, which is more than the throughput of the self-timed execution. Fig. 1(e) is a periodic schedule of G_1 with unfolding factor 2. This schedule provides the throughput $2/11$, which is more than that of the schedule shown in Fig. 1(d).

B. LATENCY

Latency is the time between initiating an action and receiving the response. In an SDFG the actions are firing of actors and the responses are consumption of produced tokens by some other actors. To define latency of an SDFG we need to define the notions of corresponding firing [9].

Definition 9 (Corresponding Firing [9]): For a path a_1, $a_2, \cdots, a_k$ from actor a_1 to a_k of an SDFG (A, D, C_0) the j_1-th firing of a_1 corresponds to the j_i-th firing of a_i for $1 < i \le k$ if for each $1 < l < i$, the j_l-th firing of a_l is its first firing that consumes at least one token produced by the j_{l-1}-th firing of a_{l-1}. The firing of a_k corresponding to the j_1-th firing of a_1 is denoted by $cf(a_1, j_1, a_k)$.

In an HSDF there is a one-to-one correspondence between the firing of some source actor and some destination since the firing rates are one. In an SDFG, the firing rates of actors are different therefore a one-to-one correspondence does not exist between actors. In order to define the latency for an SDFG, [9] introduces the latency graph of an SDFG. The latency graph is an SDFG with an explicit source actor added to the source of latency analysis and a destination actor added to the intended destination.

Definition 10 (Latency Graph [9]): The latency graph for actors a, b of an SDFG G is $G_{L(a,b)} = (A_L, D_L, C_{0L})$ where

- $A_L = A \cup \{src, dst\}$ *and* $\{src, dst\} \cap A = \emptyset$,
- $D_L = D \cup \{(src, a, \gamma(a), 1), (b, dst, 1, \gamma(b))\}$,
- $C_{0L} = C_0 \cup \{((src, a, \gamma(a), 1), 0), ((b, dst, 1, \gamma(b), 0)\}$,

The *src* and *dst* actors fire exactly once in every iteration of the graph and take zero time to execute, and hence the timing behavior of the graph does not change. Fig. 1(f) shows the latency graph for actors u and w of the SDFG shown in Fig. 1(a).

Let $F^{\sigma}_{a,k}$ denote the finishing time of the kth firing of actor $a \in A$ in execution σ. The set FE contains all feasible executions.

Definition 11 (Latency [9]): Let $G_{L(a,b)} = (A_L, D_L, C_{0L})$ be the latency graph. For an execution σ the time delay between the k-th firing of src and its corresponding firing of dst is called the k-th latency of a and b, denoted as $L^{\sigma}_k(a, b)$,

$$L^{\sigma}_k(a,b) = F^{\sigma}_{dst,cf(src,k,dst)} - F^{\sigma}_{src,k}$$

The latency of actors a and b in execution σ, $L^{\sigma}(a, b)$, is the maximum k-th latency of a and b for all firings of a:

$$L^{\sigma}(a, b) = \max_{k \in \mathbb{N}_+} L^{\sigma}_{k}(a, b)$$

The minimum latency of actors a and b, $L^{min}(a, b)$, is the minimum over all feasible executions in FE:

$$L^{min}(a, b) = \min_{\sigma \in FE} L^{\sigma}(a, b)$$

Proposition 1 ([9]): On a homogeneous multiprocessor platform for every latency graph $G_{L(a,b)}$, there is a $\delta \in \mathbb{N}$, such that $\forall k \in \mathbb{N}_+$, $cf(src, k, dst) = k + \delta$.

Proposition 1 states that there is a one-to-one correspondence between *src* and *dst* firings for a latency graph on a homogeneous multiprocessor platform. On a heterogeneous multiprocessor platform a one-to-one correspondence between *src* and *dst* firings for a latency graph may not exist. Consider the execution shown in Fig. 1(e) of latency graph $G1_{L(u,w)}$ shown in Fig. 1(f) on the platform P_1 shown in Fig. 1(b). For this execution the *dst* firing at time 8 corresponds to *src* firings at time 0 and time 4.

The paper [9] proposed an algorithm that gives the minimum achievable latency between the executions of any two actors in an SDFG on a homogeneous multiprocessor platform. The algorithm consists of 4 phases:

1) Except *src* execute actors firings until *src* is the only enabled actor.
2) Fire *src* once.
3) Execute without unnecessary delays all actors firings required for enabling the *dst* actor.
4) Fire *dst* once.

Fig. 2(d) shows a minimum latency execution of the latency graph shown in Fig. 2(a) on the platform P_2 shown in Fig. 2(b). The latency between actors *a* and *b* for this execution is 10 which is the total delay between actors *src* and *dst*. Fig. 2(c) shows another execution of the latency graph $G2_{L(a,b)}$ on the platform P_2. The latency between actors *a* and *b* for this execution is 8 which is less than the latency computed by the algorithm presented in paper [9]. Hence the algorithm presented in [9] cannot be used to compute the minimum achievable latency between two actors in an SDFG on a heterogeneous multiprocessor platform.

We can find the firing of *dst* corresponding to a firing of *src* for a latency graph using a scheme for numbering tokens as follows:

- Whenever *src* fires, assign a successive number to all the tokens produced by it starting from one. All initial tokens are numbered zero.
- An intermediate actor consumes the tokens in FIFO manner and produces tokens with the number which is the maximum of all the token numbers consumed by it from all its incoming edges.
- When *dst* consumes the token numbered *i* for the first time this indicates that this firing of *dst* corresponds to the *i*-th firing of *src*.
- In some cases, *dst* may receive token numbered *i* followed by token numbered $i + k$ for some $k > 0$. In such a case, we consider *dst* to have received all the intermediate tokens numbered $i+1, \cdots, i+k-1$ at the same time as the token numbered $i + k$.

In this paper, we are interested in finding the latency of a static schedule for a given unfolding factor with an optimal throughput for an SDFG on a heterogeneous multiprocessor platform. Since these schedules are periodic, we will find the firing of *dst* corresponding to all the firings of *src* in the first schedule cycle of f iterations.

III. TIMED AUTOMATA

Alur and Dill proposed *timed automata* as a model to represent the behavior of time-critical systems [2]. A timed automaton is a finite automaton extended with real-valued variables called *clocks*. All the clocks progress synchronously with time. Clocks may only be inspected, and reset to zero. After resetting the clocks, they start increasing their values implicitly as time progresses, i.e., the value of the clock denotes the time that has been elapsed since its last reset. Conditions on the clock values (*clock constraints*) are used as enabling conditions (*guards*) of transitions. Clock constraints are also used to limit the amount of time to be spent in a location.

Definition 12 (Clock Constraints): The clock constraints over the set Clk of clocks denoted CC(Clk), is defined by the grammar:

$$g ::= x \bowtie c \mid x - y \bowtie c \mid g \wedge g \mid true$$

where $x, y \in Clk$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$

Definition 13 (Timed Automata): A timed automaton is a tuple $TA = (L, Act, Clk, T, l^0, Inv)$ where:

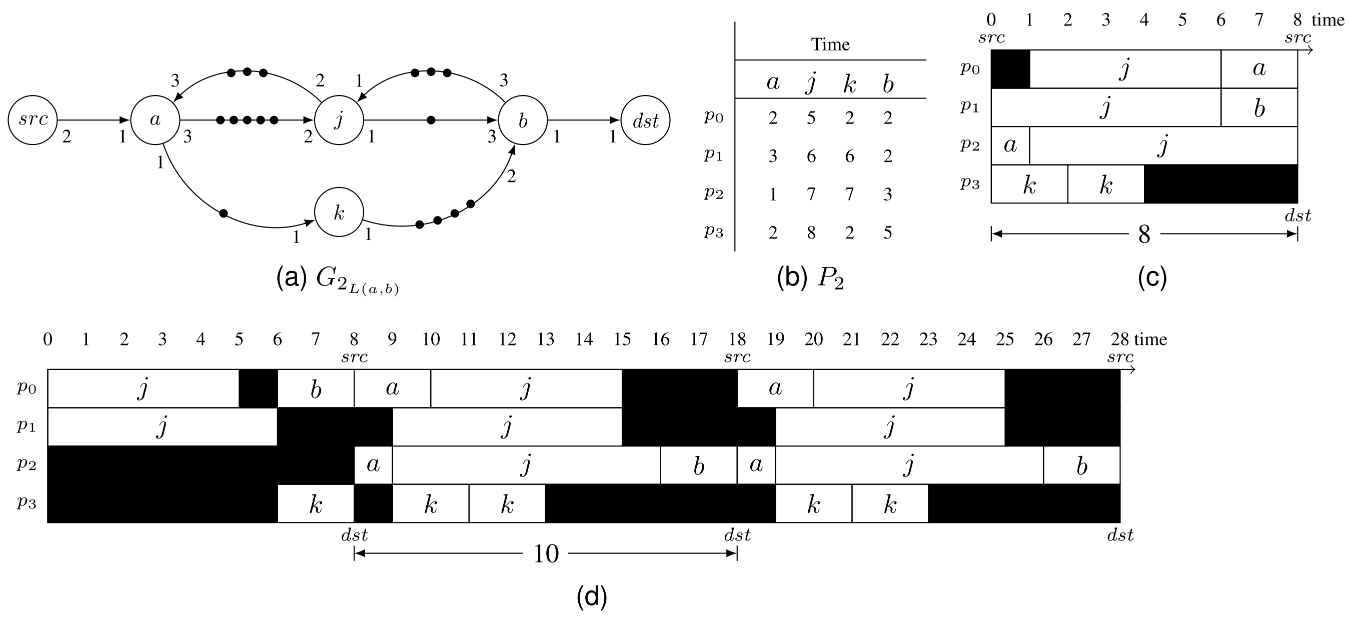
- *L is a finite set of locations;*
- *$l^0 \in L$ is the initial location;*
- *Act is finite set of actions;*
- *Clk is finite set of clocks.*
- *$T \subseteq L \times CC(Clk) \times 2^{Clk} \times L$ is a transition relation.*
- *$Inv : L \rightarrow CC(Clk)$ is an invariant-assignment function.*

The location invariant $Inv(l)$ specifies how long the timed automaton may stay in that location, i.e., location l should be left before the invariant $I(l)$ becomes invalid.

We represent the timed automaton by a graph where vertices represent the locations and edges are labeled with tuple (g, α, r) where g is a guard, α is an action, $r \subseteq Clk$ is set of clocks. The interpretation of $l \xrightarrow{g,\alpha,r} l'$ is that the timed automaton can move from location l to l' when guard g holds. While moving from location l to l', any clocks in r will be reset to zero and action α is performed.

Definition 14 (Clock Valuation): A clock valuation η for the set Clk is a function $\eta : Clk \rightarrow \mathbb{R}_+$, which assign a time value to each clock, i.e., for $x \in Clk$ its current value is $\eta(x)$. Let $\eta_0(x) = 0$ for all x in Clk. We write $\eta \in g$ to mean that clock values denoted by η satisfy the guard g.

The set of all clock valuation over *Clk* is denoted by $Eval(Clk)$. For positive real d, $\eta+d$ is defined by $(\eta+d)(x) =$



	a	j	k	b
p_0	2	5	2	2
p_1	3	6	6	2
p_2	1	7	7	3
p_3	2	8	2	5

FIGURE 2. **The system model M_2 and its schedule (a) the Latency Graph $G_{2_{L(a,b)}}$; (b) the execution platform P_2 and execution time of actors in $G_{2_{L(a,b)}}$ on different processor and the execution time of *src* and *dst* are 0; (c) a periodic schedule with unfolding factor 1; (d) A minimum latency execution of $G_{2_{L(a,b)}}$.**

$\eta(x) + d$. For $r \subseteq Clk$, let $[r \mapsto 0]\eta$ denote the clock assignment that maps all clocks in r to 0 and agrees with η for other clocks in $Clk \setminus r$.

There are two possible ways in which a timed automaton can proceed: by taking a transition in the timed automaton (*discrete transition*), or letting the time progress while staying in a location (*delay transition*). The state of a timed automaton is determined by its current location and the current values of all its clocks. Due to the continuous time domain timed automata have infinitely many states. Thus the timed automata can be considered as finite description of an infinite transition systems.

Definition 15 (Semantics of Timed Automata): Let $(L, Act, Clk, T, l^0, Inv)$ be a timed automaton. The semantic is defined as a transition system $(S, s_0, \rightarrow)$, where $S \subseteq L \times Eval(Clk)$ is the set of the states, $s_0 = \langle l^0, \eta_0\rangle$ is the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}_+ \cup Act) \times S$ is the transition relation defined by the rules:

- *delay transition, $\langle l, \eta\rangle \xrightarrow{d} \langle l, \eta + d\rangle$ if $\forall d' : 0 \leq d' \leq d \Rightarrow \eta + d' \in Inv(l)$ where $d \in \mathbb{R}_+$.*
- *discrete transition, $\langle l, \eta\rangle \xrightarrow{a} \langle l', \eta'\rangle$ if $l \xrightarrow{g,a,r} l', \eta \in g, \eta' = [r \mapsto 0]\eta$ and $\eta' \in Inv(l')$.*

A. NETWORK OF TIMED AUTOMATA

A network of timed automata [4] is the parallel composition $A_1 \parallel \cdots \parallel A_n$ of a set of timed automata $A_1, \cdots, A_n$ where $A_i = (L_i, Act, Clk, T_i, l_i^0, Inv_i)$, $1 \leq i \leq n$ over a common set of clocks and actions. A location vector is a vector $\bar{l} = (l_1, \ldots, l_n)$ and the invariant at $\bar{l}$ is defined by $Inv(\bar{l}) = \wedge_i Inv_i(l_i)$. Synchronous communication between timed automaton is carried out by handshaking synchronized input and output actions. The notation $\bar{l}[l_i'/l_i]$ represents the vector where ith element l_i of $\bar{l}$ is replaced by l_i'. For an action $a \in Act$, $a?$ denotes an input action and $a!$ denotes an output action and the internal action is represented by τ.

Definition 16 (Semantics of a Network of Timed Automata): Let $A_i = (L_i, Act, Clk, T_i, l_i^0, Inv_i)$ be a network of n timed automata. Let $\bar{l}_0 = (l_1^0, \ldots, l_n^0)$ be the initial location vector. The semantic is given as for the single timed automata in terms of transition system $(S, s_0, \rightarrow)$, where $S = (L_1 \times \cdots \times L_n) \times Eval(Clk)$ is the set of states, $s_0 = (\bar{l}_0, \eta_0)$ is the initial state, and $\rightarrow \subseteq S \times (\mathbb{R}_+ \cup Act) \times S$ is the transition relation such that:

- *$\langle \bar{l}, \eta\rangle \xrightarrow{d} \langle \bar{l}, \eta + d\rangle$ if $\forall d' : 0 \leq d' \leq d \Rightarrow \eta + d' \in Inv(\bar{l})$ where $d \in \mathbb{R}_+$.*
- *$\langle \bar{l}, \eta\rangle \xrightarrow{a} \langle \bar{l}[l_i'/l_i], \eta'\rangle$ if $l_i \xrightarrow{g,\tau,r} l_i', \eta \in g, \eta' = [r \mapsto 0]\eta$ and $\eta' \in Inv(\bar{l}[l_i'/l_i])$.*
- *$\langle \bar{l}, \eta\rangle \xrightarrow{a} \langle \bar{l}[l_i'/l_i, l_j'/l_j], \eta'\rangle$ if $l_i \xrightarrow{g_i,c?,r_i} l_i'$ and $l_j \xrightarrow{g_j,c!,r_j} l_j'$ $\eta \in g_i \wedge g_j, \eta' = [r_i \cup r_j \mapsto 0]\eta$ and $\eta' \in Inv(\bar{l}[l_i'/l_i, l_j'/l_j])$.*

IV. TIMED AUTOMATA FOR THE SYSTEM MODEL

In this section we will describe how the system model $M = (G_{L(a,b)}, P)$ can be formalized using timed automata, where $G_{L(a,b)} = (A_L, D_L, C_{0L})$ is the latency graph of an SDFG $G = (A, D, C_0)$ and P is an execution platform. The entire system model can be defined as `System`, which can be described as follows:

$$\texttt{System} = A_{G_{L(a,b)}} \parallel Processor_1 \parallel \cdots \parallel Processor_n$$

where $\parallel$ denotes parallel composition of timed automata in UPPAAL [3], shown in Fig. 3. The timed automaton $A_{G_{L(a,b)}}$

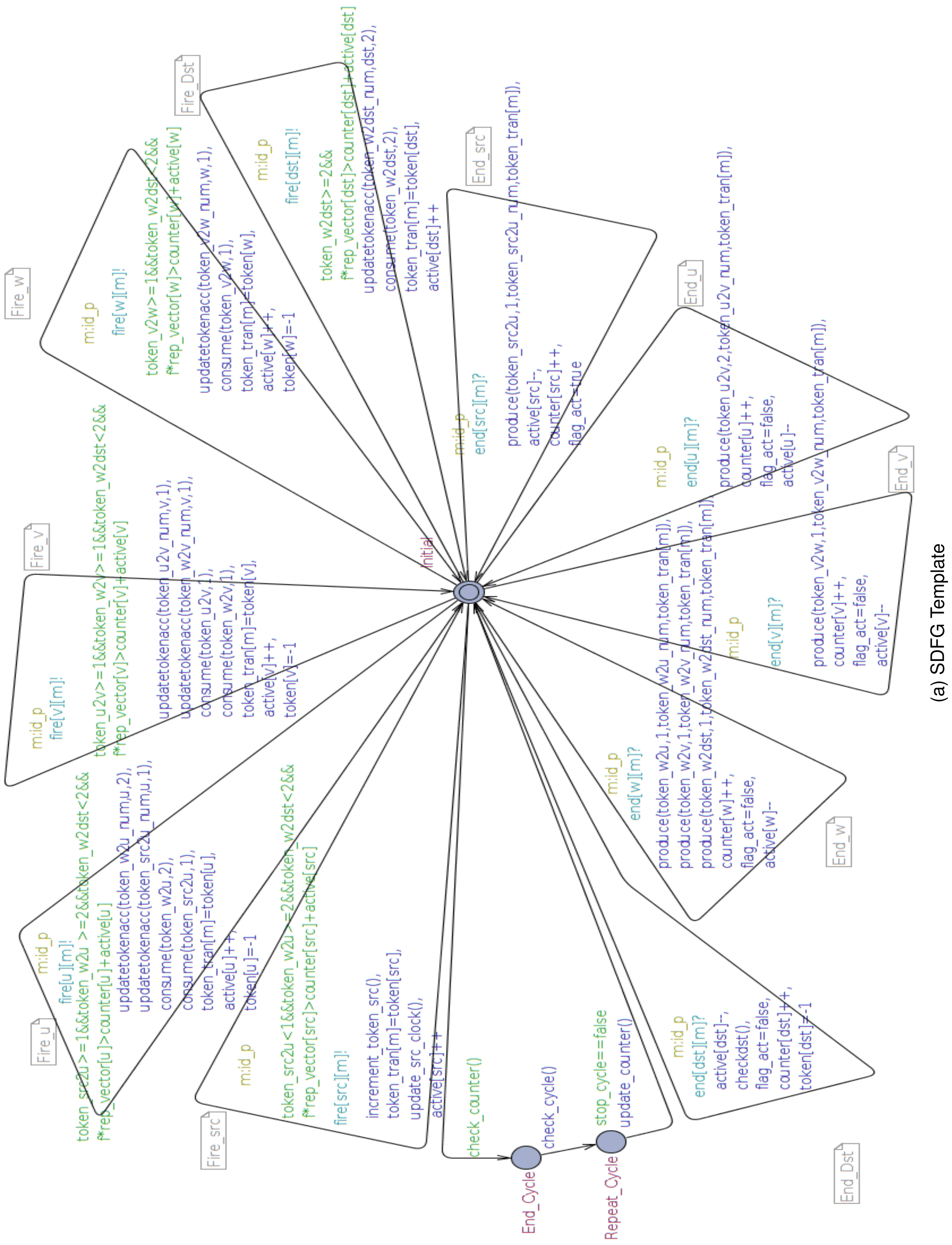
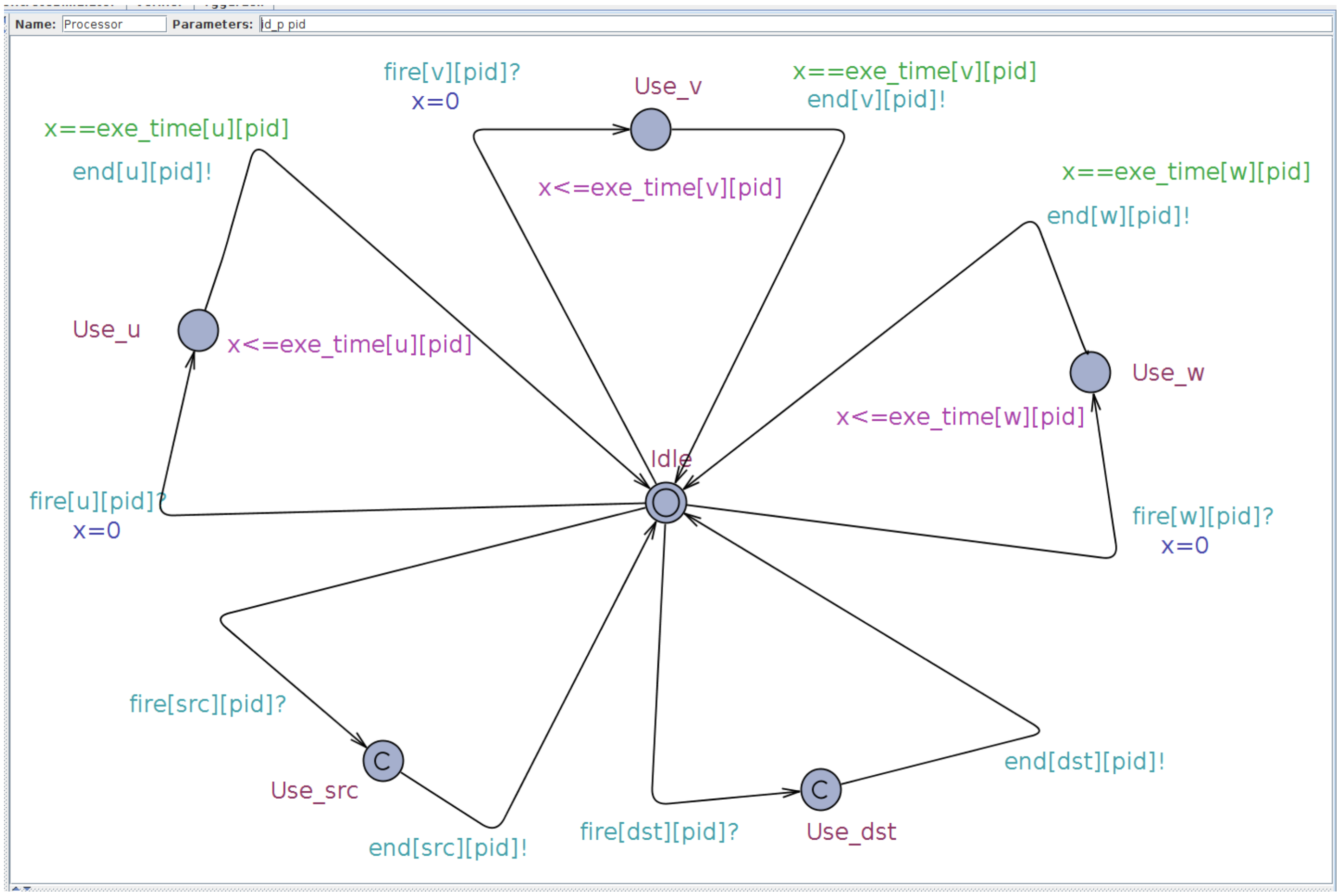


FIGURE 3. UPPAAL model for system model $M_1 = (G_{1_{L(u,w)}}, P_1)$.



(b) Processor Template

FIGURE 3. *(Continued.)* **UPPAAL model for system model $M_1 = (G_{1_{L(u,w)}}, P_1)$.**

models the latency graph $G_{L(a,b)}$ as shown in Fig. 3(a). The timed automata $Processor_0, \ldots, Processor_{n-1}$ model the processors $p_0, \ldots, p_{n-1}$ of P, as shown in Fig. 3(b). These timed automata are adapted from [1] with appropriate modification for computing latency as defined below. $A_{G_{L(a,b)}}$ is defined as,

$$A_{G_{L(a,b)}} = \{L, Act, Clk, T, l^0, Inv\}$$

where $A_{G_{L(a,b)}}$ has three locations: $L = \{$`Initial`, `RepeatCycle`, `EndCycle`$\}$, with initial location $l^0 =$ `Initial` (marked with double circle). When the timed automaton $A_{G_{L(a,b)}}$ is in the `EndCycle` location all actors a in A_L have completed firing $f \times \gamma(a)$ times, where f is the unfolding factor and $\gamma(a)$ is the number of times an actor a must fire in each iteration. At the location `EndCycle`, $A_{G_{L(a,b)}}$ decides whether it should repeat another schedule cycle of f iterations.

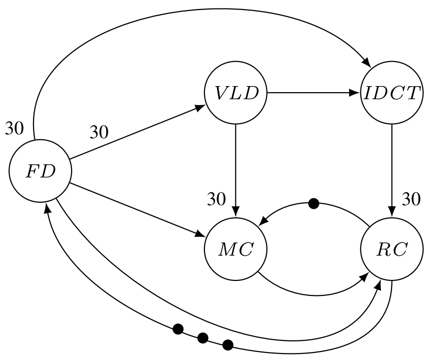
The action set Act contains two actions `fire`! and `end`? to synchronize with the timed automata $Processor_0, \ldots, Processor_{n-1}$. When an actor a is enabled it synchronizes with a timed automaton $Pocessor_i$ by `fire[a][i]`!. The $Processor_i$ has a corresponding `Fire[a][i]`?. For each $p_i \in P$ and $a \in A_L$, `Fire[a][i]`? represents the start of the firing of the actor a on the processor p_i and `end[a][i]`? represents its ending.

In $A_{G_{L(a,b)}}$ there are no location invariants and Clk is the set $\{src\_start_1, \ldots, src\_start_f, dst\_end_1, \ldots, dst\_end_f\}$ of $2f$ clocks. The src_start_i clock is used to find the time duration between the i-th firing of the src and the end of the schedule. The dst_end_i clock is used to find the time duration between the firing of dst corresponding to i-th firing of src and the end of the schedule.

For each $a \in A_L$ and all $d \in InC(a)$, T contains the following two edges:

- `Initial` $\xrightarrow{g,\texttt{fire[a][i]!},\emptyset}$ `Initial`, where $g = (C(d) \geq CR(d))$ && $(counter[a] < f \times \gamma(a))$
- `Initial` $\xrightarrow{\texttt{true},\texttt{end[a][i]?},\emptyset}$ `Initial`

The guard g signifies that an actor $a \in A_L$ must be enabled and the total number of firings for a must be less than $f \times \gamma(a)$ in order to execute the action `fire`!. An actor completes its firing by executing the action `end`?. There is no input edge for the actor $src \in A_L$, so it can fire at any time. We control its firing and it fires only when its output edge connecting to the actor a does not contain any token and the actor a is ready to

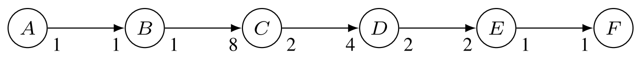


(a) G_3

	Time				
	FD	VLD	$IDCT$	MC	RC
p_0	0	1	1	9	15
p_1	0	3	2	18	25

(b) P_3

FIGURE 4. The system model M_3 of the MPEG-4 decoder (a) the SDFG G_3; (b) the execution platform P_3 and execution time of actors in G_3 on different processors.



(a) G_4

	Time					
	A	B	C	D	F	F
p_0	1	2	4	4	2	1
p_1	1	1	2	1	3	1

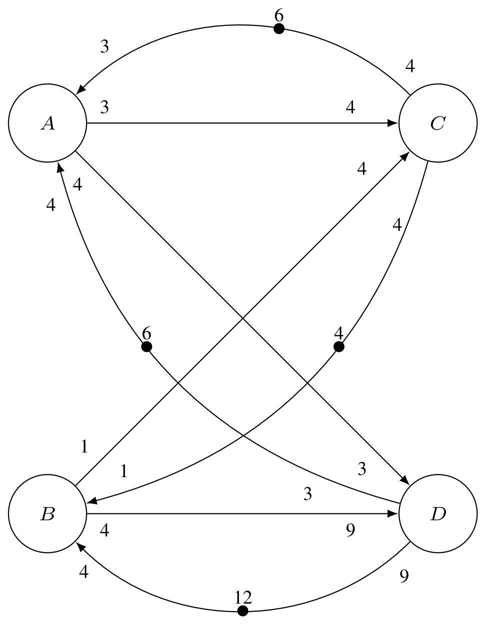
(b) P_4

FIGURE 5. The system model M_4 of the Modem (a) the SDFG G_4; (b) the execution platform P_4 and execution time of actors in G_4 on different processors.

fire. In the timed automaton $A_{G_{L(a,b)}}$, T contains the following additional edges:

- `Initial` $\xrightarrow{g,null,\emptyset}$ `EndCycle`, where $g = \forall a\ counter[a] == f \times \gamma(a)$.
- `EndCycle` $\xrightarrow{\mathtt{true},null,\emptyset}$ `RepeatCycle`
- `RepeatCycle` $\xrightarrow{Stop\_Cycle==0,null,\emptyset}$ `Initial`

The guard g in the first transition signifies that if all actors a in A_L have fired $f \times \gamma(a)$ times, then $A_{G_{L(a,b)}}$ must move to the location `EndCyle` from the location



(a) G_5

	Time			
	A	B	C	D
p_0	1	3	1	1
p_1	1	1	2	1

(b) P_5

FIGURE 6. The system model M_5 of the bipartite SDFG (a) the SDFG G_5; (b) the execution platform P_5 and execution time of actors in G_5 on different processors.

`Initial`. In the second transition above, the automaton $A_{G_{L(a,b)}}$ moves from the location `EndCycle` to the location `RepeatCycle`. In the same transition the automaton $A_{G_{L(a,b)}}$ sets the boolean variable *Stop_Cycle* to true if the actor *dst* completes all its firings corresponding to all the firings of *src* in the first schedule cycle of f iterations. In the third transition above, if the variable *Stop_Cycle* is `false` then the automaton $A_{G_{L(a,b)}}$ moves from the location `RepeatCycle` to the location `Initial` and it resets the *counter*$[a]$ to 0 for all actors $a \in A_L$. If the variable *Stop_Cycle* is `true` then the automaton $A_{G_{L(a,b)}}$ will halt in the location `RepeatCycle`, i.e., `System` will be in deadlock.

The timed automata $Processor_0, \ldots, Processor_{n-1}$ are defined as:

$$Processor_i = \{L_i, Act_i, Clk_i, T_i, l_i^0, Inv_i\};\ 0 \le i \le n-1$$

where l_i^0 = `idle` is the initial location. This location has no invariant. Clk_i contains only one clock x_i. For each actor $a \in A_L$ there is a location `Use_a` in the set L_i. The location `Use_a` indicates that processor $p_i \in P$ is currently used by

TABLE 1. Repetition vectors.

SDFG	Repetition Vector	Latency	
		Source	Destination
Modem in Fig. 5(a)	$\langle(A,16),(B,16),(C,2),(D,1),(E,1),(F,1)\rangle$	A	F
MPEG-4 Decoder in Fig. 4(a)	$\langle(FD,1),(VLD,30),(IDCT,30),(MC,1),(RC,1)\rangle$	FD	RC
Example SDFG in Fig. 1(a)	$\langle(u,1),(v,2),(w,2)\rangle$	u	w
Bipertite SDFG in Fig. 6(a)	$\langle(A,12),(B,36),(C,9),(D,16)\rangle$	A	D

TABLE 2. Experimental results.

System Model	Unfolding Factor f	Optimal Throughput	Latency	Memory(MB)	Time(s)
Modem	1	0.0434	23	8.236	0.12
	2	0.0465	40	48.780	2.45
	3	0.0476	63	782.408	47.745
	4	0.0481	82	9142.272	604
MPEG-4 Decoder	1	0.0161	62	12.628	0.3
	2	0.0172	116	194.520	12.62
	3	0.0176	170	2526.548	182.43
	4	0.0177	192	9588.632	659.02
	5	0.0179	203	15701.316	1106.79
Example SDFG	1	0.166	6	6.364	0.01
	2	0.181	8	6.576	0.02
	3	0.187	8	7.100	0.07
	4	0.181	8	9.180	0.13
	5	0.185	8	10.512	0.18
Bipertite SDFG	1	0.0175	57	7.052	0.04
	2	0.0175	114	10.116	0.04
	3	0.0175	114	13.492	0.36
	4	0.0175	114	19.776	0.49
	5	0.0175	114	25.328	0.18

actor $a \in A_L$. The location `Use_a` has an invariant $x_i <= exe\_time[a][p_i]$ and its outgoing transition has the constraint $x_i == exe\_time[a][p_i]$. This means that the timed automaton $Processor_i$ can stay in `Use_a` for exactly $exe\_time[a][p_i]$ time units. The action set contains two actions `fire`? and `end`! to synchronize with the timed automaton $A_{G_{L(a,b)}}$. For each actor $a \in A_L$, T_i contains two edges:

- `Idle` $\xrightarrow{\texttt{true,fire[a][i]?},x_i}$ `Use_a`.
- `Use_a` $\xrightarrow{x_i==exe\_time[a][p_i],\texttt{end[a][i]!},\emptyset}$ `Idle`.

In the location `Idle`, the timed automaton *Processor_i* waits for the firing of an actor $a \in A_L$ with the `fire[a][i]`? synchronization. When an actor a executes the action `fire[a][i]`! the timed automaton $Processor_i$ moves to the location `Use_a`. *Processor_i* stays in the location `Use_a` for exactly $exe\_time[a][p_i]$ time units and leaves the location `Use_a` by executing the `end[a][i]`! action.

V. MODELING SYSTEM MODELS IN UPPAAL

In this section we describe the specification of the timed automata used to capture the system model $M = (G, P)$ using UPPAAL [3]. Consider the system model $M_1 = (G1_{L(u,w)}, P_1)$ where $G1_{L(u,w)}$ is the latency graph shown in Fig. 1(f) and P_1 is an execution platform shown in Fig. 1(b). Fig. 3 shows the UPPAAL model of the system model M_1. In UPPAAL, we have separate templates for $G1_{L(u,w)}$ and execution platform P_1, namely `SDFG` and `Processor` respectively.

In the template `SDFG` shown in Fig. 3(a), the `select` statement $e : id\_p$ is used to select a processor with e ranging over the user defined type id_p. The model consists of two instances of `Processor` derived from the same template. The template has an argument $id\_p\ pid$ that defines its identifier.

A. LATENCY CALCULATION

To obtain the schedule of the system model with optimal throughput for a given unfolding factor, we ask UPPAAL to check `E <> deadlock` and to return a fastest trace. The latency for the system model is calculated as follows:

$$L_f^\sigma = \max_{1 \le i \le f} (src\_start_i - dst\_end_i)$$

VI. CASE STUDIES

We evaluate our methodology on a real DSP and a multimedia application modeled as SDFGs. From the DSP domain we used a modem [5] and from the multimedia domain, we used an MPEG-4 decoder [14] with scenario P30 for the evaluation of the trade-off between throughput and latency for different unfolding factors. The MPEG-4 decoder is modeled as a scenario-aware dataflow (SADF) model in [14]. Each scenario in an SADF model is an SDFG. We consider the scenario P30. We also analyzed the examples shown in Fig. 1 and the bipartite SDFG with buffer capacities from [5].

The system model of an MPEG-4 decoder, modem and bipartite SDFG are shown in Fig. 4,5 and 6, respectively. Table 2 shows the repetition vector and source and destination actors for latency evaluation of each SDFG. In the SDFG G_3, consumption and production rates are omitted when they are 1. We have computed the latency betweens the actors of a static schedule with optimal throughput by varying the unfolding factor from 1 to 5.

All experiments were performed on a laptop that runs on Intel core-i7 processor with 2 GHz and 16GB of RAM. The results of the experiments are shown in Table 2. The second column is the unfolding factor f. The third and fourth column show the optimal throughput and latency of a static schedule for the unfolding factor varying form 1 to 5. The fifth and sixth column show the execution time and memory consumption of our approach finding the optimal throughput and latency. For the MPEG-4 decoder an unfolding factor higher than 5 and for the modem an unfolding factor higher than 4 resulted in a running time of more than an hour.

VII. CONCLUSION

In this paper, we have presented an approach to compute the latency of a static schedule for a given unfolding factor with an optimal throughput for an SDFG on a heterogeneous multiprocessor platform using timed automata. The experimental evaluation shows that our technique can manage a model of moderate size within reasonable execution time, and can find how the unfolding factor affects the throughput and latency. Increasing the unfolding factor provides more scope for expanding the throughput at the cost of longer latency. Our approach allows the system designer to find the unfolding factor such that throughput and latency constraints of the system are met. We encountered the state-space explosion problem for bigger models while using UPPAAL. Future work includes computing the minimal achievable latency of a system model on a heterogeneous multiprocessor platform. We also plan to compute the latency considering other constraints like energy consumption and buffer size.

REFERENCES

[1] W. Ahmad, R. D. Groote, P. K. F. Hölzenspies, M. Stoelinga, and J. V. D. Pol, "Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata," in *Proc. 14th Int. Conf. Appl. Concurrency Syst. Design*, Tunis La Marsa, Tunisia, Jun. 2014, pp. 72–81.

[2] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[3] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems* (Lecture Notes in Computer Science), vol. 3185, M. Bernardo and F. Corradini, Eds. Berlin, Germany: Springer, Sep. 2004 pp. 200–236.

[4] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets: Advances in Petri Nets* (Lecture Notes in Computer Science), vol. 3098, J. Desel, W. Reisig, and G. Rozenberg, Eds. Berlin, Germany: Springer, 2004, pp. 87–124.

[5] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *J. VLSI Signal Process. Syst. Signal, Image Video Technol.*, vol. 21, no. 2, pp. 151–166, Jun. 1999.

[6] R. de Groote, J. Kuper, H. Broersma, and G. J. M. Smit, "Max-plus algebraic throughput analysis of synchronous dataflow graphs," in *Proc. 38th Euromicro Conf. Softw. Eng. Adv. Appl.*, Çeşme, Izmir, Sep. 2012, pp. 29–38.

[7] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "Towards performance analysis of SDFGs mapped to shared-bus architectures using model-checking," in *Proc. Conf. Design, Automat. Test Eur. (DATE)*. San Jose, CA, USA: EDA Consortium, 2013, pp. 1167–1172.

[8] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij, "Throughput analysis of synchronous data flow graphs," in *Proc. 6th Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, Turku, Finland, Jun. 2006, pp. 25–36.

[9] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen, "Latency minimization for synchronous data flow graphs," in *Proc. 10th Euromicro Conf. Digit. Syst. Design Architectures, Methods Tools (DSD)*, Lübeck, Germany, Aug. 2007, pp. 189–196.

[10] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Math.*, vol. 23, no. 3, pp. 309–311, 1978.

[11] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 24–35, Jan. 1987.

[12] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 178–195, Feb. 1991.

[13] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2009.

[14] B. Theelen, J.-P. Katoen, and H. Wu, "Model checking of scenario-aware dataflow with CADP," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*. San Jose, CA, USA: EDA Consortium, Mar. 2012, pp. 653–658.

[15] J. A. F. F. Dias, J. J. P. C. Rodrigues, N. Kumar, V. Korotaev, and G. Han, "REMA: A resource management tool to improve the performance of vehicular delay-tolerant networks," *Veh. Commun.*, vol. 9, pp. 135–143, Jul. 2017.

[16] A. Dua, N. Kumar, and S. Bawa, "Game theoretic approach for real-time data dissemination and offloading in vehicular ad hoc networks," *J. Real-Time Image Process.*, vol. 13, no. 3, pp. 627–644, Sep. 2017.

[17] X. Zhu, R. Yan, Y. Gu, J. Zhang, W. Zhang, and G. Zhang, "Static optimal scheduling for synchronous data flow graphs with model checking," in *FM 2015: Formal Methods* (Lecture Notes in Computer Science), vol. 9109, N. Bjørner and F. de Boer, Eds. Cham, Switzerland: Springer, 2015, pp. 551–569.