Temporal Analysis Flow Based on an Enabling Rate Characterization for Multi-Rate Applications Executed on MPSoCs with Non-Starvation-Free Schedulers

Joost P.H.M. Hausmans §  Stefan J. Geuns §  Maarten H. Wiggers Marco J.G. Bekooij ¶
joost.hausmans@utwente.nl

§ University of Twente, Enschede, The Netherlands
¶ NXP Semiconductors, Eindhoven, The Netherlands

ABSTRACT
Real-time stream processing applications often contain multi-rate behavior. This multi-rate behavior can be accurately modeled using Synchronous Dataflow (SDF) graphs. However, no temporal analysis technique exists which is applicable for arbitrary cyclic SDF graphs and can handle cyclic resource dependencies.

This paper presents a temporal analysis flow for SDF graphs which is applicable for systems with non-starvation-free schedulers such as static priority preemptive schedulers. The analysis flow uses an enabling rate characterization to calculate response times. This enabling rate characterization is determined using multi-dimensional periodic schedules and allows a more accurate modeling of enabling patterns than is possible with a characterization that is based on periods and enabling jitters.

The presented approach is applicable for arbitrary (cyclic) graph topologies and can take buffer capacity constraints into account during analysis. Also cyclic resource dependencies can be analyzed. The presented analysis flow is the first approach that considers arbitrary SDF graph topologies in combination with cyclic resource dependencies that are caused by non-starvation-free schedulers.

The proposed analysis flow is evaluated using a radio processing application. The analysis results are obtained using a tool in which the analysis flow is implemented. This case-study illustrates that the used enabling characterization achieves up to 87% better response times than with an enabling jitter based characterization.

Categories and Subject Descriptors
D.4.8 [Operating Systems]: Performance—Modeling and prediction

General Terms
Performance, Theory, Verification

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© ACM 2014 978-1-4503-2941-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2609248.2609262

1. INTRODUCTION
Real-time stream processing applications are often executed on Multiprocessor Systems-on-Chip (MPSoCs) for power/performance reasons. The tasks of these applications are usually executed data-driven. Such stream processing applications often have a strictly periodic source or sink that imposes a throughput constraint, though latency constraints are also common. The modern MPSoCs on which these applications execute typically have an individual scheduler per processor.

The temporal behavior of such systems can be efficiently analyzed using timed dataflow models. Streaming applications can be modeled accurately [15, 21] and dataflow models are available with different expressivity, from static multi-rate models (e.g. SDF [14]) to dynamic models (e.g. SADF [23], VPDF [31]). However, in case of for example SDF models the current state-of-the art demands that all schedulers are run-time budget schedulers [28, 30]. Such budget schedulers (e.g. Time-Division Multiplexing (TDM)) guarantee a minimum time budget in a maximum time interval. Therefore, by construction a bound is imposed on the maximum interference that tasks cause on other tasks sharing a resource.

This paper shows that also multiprocessor systems with schedulers different from budget schedulers can be analyzed with timed SDF models. This is achieved by computing upper and lower bounds on the interference experienced by each task. In a properly managed multiprocessor system these upper and lower bounds will not indefinitely diverge because of the strictly periodic source or sink of the real-time stream processing application. The considered class of schedulers is called non-starvation-free schedulers and is broader than budget schedulers. For such non-starvation-free schedulers (e.g. static priority pre-emptive), the interference from other tasks can only be bounded with knowledge about how often tasks are started [30].

Recent work introduced dataflow analysis techniques based on the same insight [7]. However, their scope is limited to Homogeneous Synchronous Dataflow (HSDF) models which can only describe applications in which all tasks have the same firing rate. This restriction allows for a sufficiently accurate description of the upper and lower bounds on in-
terference in terms of two parameters: period and jitter.

Our contribution is a temporal analysis flow for cyclic SDF models which is based on enabling characterizations. An enabling characterization is a function that describes for each time interval the minimum and maximum number of possible task enablings. The enabling characterization which is used in this paper allows to model bursts accurately. These bursts are caused by multi-rate behavior in an application and are modeled using SDF graphs. In order to temporally analyze the multi-rate behavior, a two-actor dataflow component is required instead of the one-actor component of [7]. The parameters of this two-actor component are computed differently than the two-actor component of [28]. The class of non-starvation-free schedulers requires the use of fixed-point computation to determine these characteristics.

The outline of this paper is as follows. We first discuss related work in Section 2. The proposed temporal analysis flow is presented in Section 3 and the steps of this flow are detailed in Sections 4, 5 and 6. Section 7 contains a case-study performed using the presented analysis flow and we conclude this paper in Section 8.

2. RELATED WORK

The focus of this work is on real-time stream processing applications of which the tasks are executed data-driven on MPSoCs. Data-driven execution of tasks allows to cope with varying execution times, i.e., shorter executions can compensate for long executions of a task. Such varying execution times can be captured in workload functions which can be included in temporal analysis flows [16, 18, 8]. Non data-driven approaches, such as methods that are based on the periodic task model [3, 4, 1], use strictly periodic schedules. Such strictly periodic schedules require the use of the Worst-Case Execution Times (WCETs) of tasks and can therefore not use this additional knowledge about varying execution times. Next to that, these strictly periodic schedules can not handle response times of tasks that are larger than the periods of the tasks.

The multi-rate behavior of stream processing applications can be accurately modeled using SDF models [14]. Dataflow models have been extended with time in [21] and in [28] it has been shown how the effects of run-time schedulers that belong to the class of starvation-free schedulers can be included in SDF analysis models. The effects of these starvation-free schedulers are modeled using latency-rate actor components. Using the SDF analysis model, the minimum throughput can be computed with a polynomial algorithm given only upper bounds on execution times of tasks. The analysis models can also be used to compute the minimum buffer capacities given a throughput constraint.

In [7] an analysis method is introduced which can also be used to analyze the effects of the broader class of non-starvation-free schedulers in combination with HSDF models. With such HSDF models, only single-rate behavior can be modeled. This analysis method uses the period and enabling jitter of tasks to bound the maximum number of enablings of tasks. The enabling characterization which we use in this paper is more general than such a period and enabling jitter based characterization and using this more general enabling characterization leads to more accurate analysis results for SDF graphs. Next to that, the method in this paper requires the use of a two-actor (latency-rate) component in contrast to the one-actor component of [7]. This two-actor component is required to model the combination of the response times of tasks and the multi-rate behavior correctly.

The approach from [6] combines systems with First-Come First-Served (FCFS) schedulers with HSDF models. This approach considers a subclass of the systems we consider. FCFS schedulers belong to the class of non-starvation-free which is supported by our method. Next to that, only cyclic graphs are considered while we also support cyclic graphs and, furthermore, pipelined schedules are not considered in [6] which means that the source can only be triggered when the previous result is produced.

Compared to the analysis of starvation-free run-time schedulers with SDF analysis models, the generalization to non-starvation-free schedulers comes at the cost of an exponential computational complexity of the analysis algorithms. The accuracy of the analysis results can be improved by making use of knowledge about best-case execution times, which is not considered for the analysis of starvation-free run-time schedulers. Due to the required fixed-point computation in our analysis flow, buffer-sizing can only be performed after the analysis flow finishes in contrast to dataflow analysis methods for starvation-free schedulers, which can minimize buffer sizes by modifying the schedules [17, 27].

An iterative procedure of traffic characterization and response time calculation is used by the SymTA/S approach [9]. However, this approach uses propagation of traffic instead of the computation of worst-case and best-case schedules which our method uses. Using traffic propagation can result in a low accuracy because the applied traffic characterization does not capture the correlation between different streams accurately [12]. The dataflow analysis techniques presented in this paper, do not suffer from this disadvantage because the schedules used to compute the enabling characterization do capture the correlation of events. The use of best- and worst-case schedules in our solution is possible thanks to monotonicity of dataflow graphs [30]. The enabling characterizations which we use in this paper are highly similar to the standard event models of the SymTA/S approach [9].

In [20] the SymTA/S approach is extended by calculating more accurate temporal end-to-end properties. However, it relies on similar traffic models as the original approach and these traffic models cannot be determined for arbitrary cyclic graphs [10].

The original MPA [26] approach, based on Real-Time Calculus [5], also makes use of traffic propagation. However, MPA for cyclic HSDF graphs [24] solves the problems of traffic propagation by making use of a characterization of traffic in which the correlation between streams is captured by a characterization in the time-domain instead of the time-interval domain. In the same paper a schedulability check given a throughput constraint is presented. However, it only discusses (potentially cyclic) data dependencies and does not include cyclic resource dependencies. Cyclic resource dependencies are considered in [11] but the combination with cyclic data dependencies is not considered. This combination is difficult because it requires an accurate translation between a traffic characterization in the time-domain and in the time-interval domain.

The result from [24] can be generalized to SDF graphs by converting the SDF graph into an equivalent HSDF graph [14, 21]. However, we see several issues with this approach. When an SDF actor is modeled using multiple
of an actor component, as is presented in Section 6 (step 2). The SDF model is then translated into an equivalent HSDF model which is used to determine upper and lower bounds on the schedules of the SDF model. The calculated upper and lower bounds are used to derive an enabling characterization for each task which is also discussed in Section 6 (step 3). Initially, an optimistic enabling characterization is used which underestimates the amount of enablings.

Steps 1, 2 and 3 are repeated until the enabling characterizations no longer change and thus also the response times have converged. The analysis flow also stops when a violation of the temporal constraints is detected. Both the response time analysis step (step 1) and the dataflow analysis step (step 3) can return infeasible which means that the analysis flow concludes that the application under the given mapping and given constraints is not schedulable.

Finally, if final response times are found, sufficient buffer sizes can be determined (step 5) such that the temporal results, obtained in the previous steps, are ensured. Buffer sizes are chosen such that the schedules from step 4 remain feasible. The method from [29] can be used to find such buffer sizes.

The complete analysis flow has a monotonic behavior. This monotonic behavior of the flow is guaranteed by the fact that an iteration of the analysis flow can never lead to an enabling characterization with less enablings. This is because the response time computation is monotonic in the enabling characterization and because of monotonicity of dataflow graphs which is explained in Section 6. As a result of this monotonicity, a conservative temporal analysis result is found after convergence of the response times.

4. APPLICATION CHARACTERISTICS
The characteristics of an application consists of the application described by one or more task graphs, and the mapping of the tasks of the application to the MPSoC (task to processor specification). Each task is mapped to one processor of the MPSoC. The tasks executed data-driven and are scheduled by the run-time scheduler of the processor to which the task is mapped.

A task graph is a connected directed graph of tasks in which the directed edges represent first-in-first-out (FIFO) buffers. A task graph contains one source, $\tau_s$, which executes strictly periodic with a period $P_s$ and starts at time $s(\tau_s)$. This strictly periodic execution of the source imposes a throughput constraint on the other tasks in the task graph.

We use $P_i$ for the average period of a task $\tau_i$.

All tasks in a task graph are (eventually) activated by the source of that task graph and we require that no (non-source) task can execute before the source of the task graph starts releasing data for the first time. This corresponds with the situation where the application has finished the initial behavior.

Tasks communicate using FIFO buffers which are directed from one task to another. We use $c_{ij}$ for a FIFO buffer from task $\tau_i$ to $\tau_j$. FIFO buffers consists of an optional predefined (maximum) capacity of $\bar{\phi}_{ij}$ containers which contain the data. The capacity of the FIFO buffers for which only a maximum capacity is specified, can be determined using buffer sizing techniques. The containers of a FIFO buffer can either be filled with data or be empty. We use $\bar{\phi}_{ij}$ to denote containers in FIFO $c_{ij}$ which are initially full and $\bar{\phi}_{ij}$ to denote the initially empty containers. Note that $\bar{\phi}_{ij}$

Figure 1: Overview of the analysis flow.

HSDF actors, also multiple MPA components correspond to one task. Each of these components transfers remaining service to the next component but these service curves are in the time interval domain. As a result, the moment at which a resource is used by a component is lost. This leads to a severe accuracy reduction in the case of SDF graphs because the correlation between the usage of a resource by consecutive executions of a task, is lost. Next to that, modeling a task with multiple MPA components leads to a cyclic resource dependency (the last MPA component transfers remaining service back to the first component) and it is unclear how this can be analyzed in the MPA approach.

Our method also uses the conversion to an equivalent HSDF graph. However, converting the SDF graph to an HSDF graph and then applying the analysis method of [7] is not straightforward. The one-actor model of [7] does not suffice for multi-rate applications as we will show in Section 6.3. Next to that, we show in Section 7 that the period and enabling jitter based enabling characterization from [7] leads to pessimistic analysis results in the case of multi-rate behavior. We use the schedules of the HSDF graph to determine the enabling characterizations of the tasks. The schedules of the HSDF actors are combined in such a way that the correlation between the resource usage of consecutive executions of a task is preserved.

3. ANALYSIS FLOW

Figure 1 shows the analysis flow used in this paper. This section gives a brief overview of the different steps in the flow and indicates the section in which they are discussed.

The characteristics of an application are discussed in Section 4. The temporal constraints of such an application are included in this characterization by means of periodic sources. Given these characteristics, initial upper and lower bounds on the response times of tasks can be found using iterative fixed-point computation as is discussed in Section 5 (step 1).

The flow maintains a one-to-one relation between the given task graph and an SDF model. The one-to-one relation between the task graph and the SDF model ensures that the obtained analysis results also hold for the application. The one-to-one relation relies on the properties of the response times and on the derivation of the two parameters $\bar{\phi}_{ij}$ and $\bar{\phi}_{ij}$ which are initially full and $\bar{\phi}_{ij}$ to denote the initially empty containers. Note that $\bar{\phi}_{ij}$
is equal to $\tilde{r}_{ij} + \tilde{r}_{ij'}$.

At the beginning of every execution, task $\tau_i$ acquires $\chi(c_{ki})$ full containers from each FIFO $c_{ki}$ directed towards $\tau_i$ and acquires $\psi(c_{ij})$ empty containers from each output FIFO buffer. These empty containers are filled with data produced by task $\tau_i$ and then released as $\psi(c_{ij})$ full containers on the same FIFO from which they were acquired at the end of the execution of $\tau_i$. The acquired full containers are released as $\chi(c_{ki})$ empty containers on FIFO $c_{ki}$ at the end of the execution of $\tau_i$. The number of acquired/released containers is constant for every execution of a task.

An execution of task $\tau_i$ can only start after $\tau_i$ is enabled. Task $\tau_i$ is enabled if the previous execution of task $\tau_i$ is finished and if it can acquire the required number of containers from the FIFO buffers.

The minimum amount of processor time a task $\tau_i$ requires to complete an execution is specified by its Best-Case Execution Time (BCET), $B_i$, and the maximum required processor time is specified by its WCET, $C_i$. The time between an enabling and a corresponding finish of a task is defined by the response time of that execution of the task. Bounds on the response times of tasks are presented in the next section.

5. RESPONSE TIMES

The response time of an execution of a task is the time between the enabling of the execution and the corresponding finish of the execution. In this section we provide upper and lower bounds on the response times of tasks. The bounds on response times will be used in the next sections to give guarantees on the temporal behavior of the application. Similar to [9], we define the bounds on the response time of a task $\tau_i$ using its average period $P_i$ and the enabling characterizations of the other tasks.

Similar to [19, 20, 24], only schedulers are supported for which the response times are monotonic in the enabling characterization. If the response times are not monotonic in the enabling characterizations of the tasks, then the complete analysis flow is not monotonic and convergence of the analysis cannot be guaranteed.

To find response time bounds, we use local scheduling analysis techniques [19, 25]. The basic idea is that the maximum/minimum interference of other tasks during an execution is added to the time that the task takes to execute.

Using local scheduling analysis techniques enables to analyze the response times of the tasks running on one processor in isolation. Each processor of the MPSoC can thus be analyzed separately. The schedules that are obtained by dataflow analysis (see Section 6.4) ensure that all the precedence constraints are satisfied. For ease of understanding we use the most simple minimum response time definition, where the interference of other tasks is assumed to be 0.

Therefore, the minimum response time $\hat{R}_i$ of a task $\tau_i$ is defined as $\hat{R}_i = B_i$. More accurate definitions of the minimum response time of a task are discussed in [19].

As discussed in [9], the maximum number of enablings of a task in a time interval $\Delta t$ can be captured in an $\hat{\eta}($interval of length $\Delta t$).

**Definition 2: (Minimum Enabling Distance Function)**

The minimum enabling distance function $\hat{\eta}(n)$ specifies the minimum distance between $n \geq 2$ consecutive enablings of task $\tau_i$.

The $\hat{\eta}(n)$ function can be converted into $\hat{\eta}(\Delta t)$ without loss of accuracy using:

$$\hat{\eta}(\Delta t) = \max_{n \in \mathbb{N}, n \geq 2} \{\{n \mid \hat{\eta}(n) < \Delta t\} \cup \{1\}\}$$  (1)

This maximum number of enablings can be used to provide upper bounds on the response time of a task. We show this for a static priority pre-emptive scheduler and a TDM scheduler.

**Static Priority Pre-emptive Scheduling.**

Each task mapped to a processor is assigned a distinct priority level when a static priority pre-emptive scheduler is used on that processor. The scheduler serves at each point in time the highest priority task that is enabled at that point in time.

The maximum response time $\hat{R}_i$ of a task $\tau_i$ scheduled using a static priority pre-emptive scheduler, can be determined using (3) [23].

$$w_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}_j(w_i(q)) \cdot C_j$$  (2)

$$\hat{R}_i = \max_{1 \leq q} \left(w_i(q) - (q - 1) \cdot P_i\right)$$  (3)

Equation (2) calculates $w_i(q)$ which is the maximum amount of time it takes to finish $q$ executions of a task $\tau_i$ after it is enabled. We use $hp(i)$ as the set of tasks running on the same processor as $\tau_i$ and which have a higher priority than $\tau_i$. As is shown in [25], iterative fixed-point computation can be used to compute $w_i(q)$. Only values of $q$ for which $w_i(q) \geq q \cdot P_i$ holds need to be considered [25]. As long as this holds, no lower priority task is executed. We define this as task $\tau_i$ being in a consecutive execution.

Note that for the focus of this paper we choose to use the WCET for each task. Improvements are available which can exploit knowledge about varying execution times [16, 18, 8]. Such methods use workload functions which capture the worst-case cumulative execution time of a task instead of its WCET. Workload functions can be easily incorporated in the computation of response times by using these workload functions instead of $C_i$, see [18].

**TDM Scheduling.**

A TDM scheduler belongs to the class of budget schedulers and guarantees every task a minimum time budget in a maximum replenishment interval. Despite the fact that TDM schedulers belong to the class of starvation-free schedulers the temporal analysis results might benefit from the analysis methods of this paper.

We consider an TDM scheduler as described in [22] where the tasks use a cooperative yield mechanism. Such a yield mechanism implies that when a task does not have sufficient data or space available, it stops its execution and allows other tasks to execute. Because of this mechanism, better temporal analysis results than for example [28] can be ob-
tained when it is known when and how often tasks execute.

Identical to the method of [28] we use the fact that a task has a guaranteed time budget. The maximum time that a task \( \tau_i \) has to wait for \( \Delta t \) processing time is equal to 
\[
\left\lfloor \frac{\Delta t}{R_i} \right\rfloor \cdot (Q - R_i)
\]
where \( R_i \) is the time budget of task \( \tau_i \), and \( Q \) is the replenishment interval.

This upper bound on the waiting time can be combined with information on the interference from other tasks to obtain a tighter response time. We use \( T(\cdot) \) for the set of tasks running on the same processor as task \( \tau_i \) without task \( \tau_i \) itself. Again iterative fixed-point computation is used to compute \( w_i(q) \) and similar to the method for static priority preemptive scheduling, only values of \( q \) for which \( w_i(q) \geq q \cdot P_i \) holds need to be considered.

\[
w_i(q) = q \cdot C_i + \min \left( \sum_{j \in T(i)} \frac{\sum_{j \in T(i)} \hat{\eta}_j(w_i(q)) \cdot C_j}{\left\lfloor \frac{\sum_{j \in T(i)} \hat{\eta}_j(w_i(q)) \cdot C_j}{R_i} \right\rfloor} \cdot (Q - R_i) \right)
\]

\[
\hat{R}_i = \max \left( w_i(q) - (q - 1) \cdot P_i \right)
\]

### 6. TEMPORAL ANALYSIS

This section provides a method to determine enabling characterizations of tasks based on dataflow analysis techniques. We use an SDF model for which we show that it is a temporally conservative abstraction of the task graphs of an application. Such SDF models can be used to analyze applications with multi-rate behavior. The analysis method uses a conversion from the SDF model to an equivalent HSDF model [14, 21]. We use the structure of this conversion to derive the enabling characterization for each task.

#### 6.1 Analysis Model

An SDF graph is a directed graph \( G = (V, E, \delta, \rho, \pi, \gamma) \), with a set of actors \( V \) and a set of directed edges \( E \) between those actors. Actors communicate by producing and consuming tokens over the edges, which represent unbounded queues. An edge \( e_{ij} \) initially has \( \delta(e_{ij}) \) tokens. An actor is enabled when the number of tokens that will be consumed is available on each of its input edges. The number of tokens that will be consumed is specified by \( \gamma : E \rightarrow \mathbb{N} \) and is equal to the number of acquired containers in the corresponding FIFO buffer. The \( \gamma(e_{ij}) \) tokens are consumed in an atomic action from all input edges of an actor at the start of a firing of that actor. A firing of actor \( v_i \) finishes \( \rho_i \) time units after its start. At the finish of a firing, actor \( v_i \) atomically produces \( \pi(e_{ij}) \) tokens on each output queue \( e_{ij} \) which is equal to the number of released containers in the corresponding FIFO buffer.

Functionally deterministic dataflow graphs, such as SDF graphs, have a monotonic temporal behavior [30]. This has a number of important implications. Increasing the firing duration of one actor in the graph can never lead to an earlier enabling of any of the actors in the graph and also, decreasing the firing durations of one of the actors can never lead to any later enabling. Similarly, increasing the amount of initial tokens in the graph can never lead to a later enabling of any of the actors in the graph and the opposite, decreasing the amount of initial tokens can never lead to any earlier enabling. See [30] for more details on monotonicity of SDF graphs.

### Figure 2: One-to-one relation between SDF model and task graph.

Consistency of an SDF model is an important property. Algorithms exist to verify consistency for a connected SDF model [13]. For an inconsistent SDF model, either no schedule is deadlock-free or there is an unbounded accumulation of tokens.

For consistent SDF models a repetition vector \( q \) can be determined. This repetition vector contains the relative firing frequencies between the actors. In the remainder of this paper, \( q_i \in q \) is used for the repetition factor of actor \( v_i \). If each actor \( v_i \) in the SDF model fires exactly \( q_i \) times then it holds for each edge in the SDF model that the number of tokens produced on an edge is equal to the number of consumed tokens from that edge.

The average period at which a task executes is different than the period of the source in the task graph, \( P^* \). The period of a task can be computed by using the relative firing frequencies. The average rate at which tasks consume data should also be equal to the rate at which the source produces data. The average period of a task is equal to the period of its corresponding SDF actor and can be calculated with (6).

\[
v_{\forall v_i \in V} : P_i = \frac{q_i}{q_i} \cdot P_s
\]
Figure 3: Actor component for modeling the worst-case behavior.

6.3 Temporally conservative modeling

A dataflow model is temporally conservative to a task graph if the worst-case temporal behavior of the task graph is captured conservatively by the dataflow model. This can be shown by proving that all of the container arrival times are bounded from above by the arrival times of the corresponding tokens [30]. Thus for each pair of a task and corresponding actor component we have to prove:

$$\forall k: e(k) \leq \hat{e}(k) \implies \forall k: f(k) \leq \hat{f}(k)$$  \hspace{1cm} (7)

Where $e(k)$ is the enabling time of the execution $k$ of a task and $\hat{e}(k)$, the enabling of the corresponding actor component firing. The finish time of execution $k$ of the task is denoted with $f(k)$ and $\hat{f}(k)$ represents the finish time of the firing of the corresponding actor component. In [30] it is shown that if (7) holds for every task and corresponding actor component pair, the dataflow model is temporally conservative to the task graph.

If we want to bound the best-case behavior of the task graph we have to show a similar property:

$$\forall k: e(k) \geq \hat{e}(k) \implies \forall k: f(k) \geq \hat{f}(k)$$  \hspace{1cm} (8)

The worst-case behavior of task $\tau_i$ is modeled with the actor component of Figure 3. We now prove that this actor component is conservative given the definition of response times of Section 5.

Without loss of generality we can state that execution $k$ of task $\tau_i$ belongs to a consecutive execution that starts at execution $n \leq k$. Furthermore, the function $w_i(k)$ is defined as the maximum amount of time it takes to finish $q$ executions of task $\tau_i$. The finish time of an execution $k$ of task $\tau_i$ is thus bounded by:

$$f_i(k) \leq \max_{n \leq j \leq k} (e_i(j) + w_i(k - j + 1))$$

Furthermore, given the definition of response times of (3) we know that:

$$\forall q: w_i(q) \leq \hat{R}_i + (q - 1) \cdot P_i$$

When the begin of the consecutive execution $(n)$ is known and by using the maximum response time of $\tau_i$, we can define an upperbound, $f_i^n(k)$, on the finish time of execution $k$ of task $\tau_i$ with Equation (9).

$$f_i(k) \leq f_i^n(k) = \max_{n \leq j \leq k} (e_i(j) + \hat{R}_i + (k - j) \cdot P_i)$$  \hspace{1cm} (9)

However, during analysis it is unknown at which execution a consecutive execution starts. Therefore, we derive a second upperbound, $f_i^\infty(k)$, which is independent of this start of the consecutive execution. After that it is shown with this upperbound that the actor component of Figure 3 is conservative to the corresponding task.

**Theorem 1:**

An upper bound on the finish time of execution $k$ of task $\tau_i$ is given by:

$$f_i(k) \leq f_i^\infty(k) = \max(e_i(k) + \max(0, \hat{R}_i - P_i), f_i(k - 1)) + \min(\hat{R}_i, P_i)$$  \hspace{1cm} (10)

**Proof.** We can distinguish two cases: either $\hat{R}_i \leq P_i$ or $\hat{R}_i > P_i$ holds.

If $\hat{R}_i \leq P_i$, then $f_i^\infty(k) = \max(e_i(k) + f_i^\infty(k - 1)) + \hat{R}_i$ and using the definition of enabling and response times of tasks it holds that: $f_i(k) \leq \max(e_i(k), f_i(k - 1)) + \hat{R}_i$. Thus $f_i(k) \leq f_i^\infty(k)$ holds.

Otherwise we have $\hat{R}_i > P_i$. To prove this case, the upperbound function $f_i^\infty(k)$ from (9) is used. If $\hat{R}_i > P_i$, then $f_i^\infty(k) = \max(e_i(k) + \hat{R}_i - P_i, f_i^\infty(k - 1)) + P_i$. If for $f_i^\infty(k)$, $j = k$ gives the maximum value then $f_i^\infty(k) = e_i(k) + \hat{R}_i$ and thus $f_i^\infty(k) \leq f_i^\infty(k)$.

Otherwise, $j = k$ does not give the maximum value and $f_i^\infty(k) \leq f_i^\infty(k)$. Because $f_i(k) \leq f_i^\infty(k)$ holds, (9) holds, also $f_i(k) \leq f_i^\infty(k)$ holds.

Using max-plus algebra on the actor component of Figure 3 it can be shown that:

$$\hat{f}_i(k) = \max(e_i(k) + \max(0, \hat{R}_i - P_i), \hat{f}_i(k - 1)) + \min(\hat{R}_i, P_i)$$  \hspace{1cm} (11)

When $e_i(k) \leq \hat{e}_i(k)$, then according to Theorem 1 and (11), $f_i(k) \leq f_i^\infty(k) \leq f_i(k)$ holds and then also (7) holds. The dataflow component of Figure 3 thus models the worst-case temporal behavior of the corresponding task conservatively.

The best-case behavior of task $\tau_i$ is modeled by choosing the firing duration of actor $v_{i,0}$ equal to 0 and $v_{i,1}$ equal to $\hat{R}_i$ which basically gives us a one-actor component consisting of $v_{i,1}$. With this model, (8) can be proven similarly.

Note that, in contrast to the method of [7], it is not possible to model the worst-case temporal behavior of a task with a one-actor component. It is shown in the next section that the schedule of the two-actor SDF component is multi-dimensional periodic. The average period of this SDF component is equal to the average period of the corresponding task but the time between two consecutive firings of the SDF component can be different. The response time definition of task $\tau_i$ that is used requires a minimal distance of $P_i$ which is, as we proved, ensured by the two-actor SDF component of Figure 3.

6.4 Determine Enabling Characterizations

To determine the enabling characterization of tasks, we first transform the SDF models for the best- and worst-case behavior into equivalent HSDF models using the method of [14, 21]. This conversion transforms each SDF actor $v_a$ of the SDF model into $q_a$ actors in the HSDF model. Each of these $q_a$ actors model a certain firing of the SDF actor $v_a$ and each of these firings can cause different enablings of other actors in the SDF model. After $q_a$ firings of actor $v_a$, all the different enablings have occurred. We call this the repetition period of an actor. We number the actors in
the HSDF as $v_0^0 \ldots v_k^{q-1}$ where $v_0^0$ corresponds to the first firing in the repetition period of actor $v_0$ and $v_k^{q-1}$ to the last firing in this repetition period. Furthermore, we use $V_H$ as the set of HSDF actors and $E_H$ as the set of edges in the HSDF graph. Each SDF actor $v_0$ has an average period $P_0$ as defined in Equation (6). Because the SDF actor is split in $q_0$ actors in the HSDF graph, on average, each of these actors have to fire once per $q_0 \cdot P_0$, which is equal to $q_0 \cdot P_s$ (see Equation (6)).

Also the SDF actor $v_a$, corresponding to the the source task $\tau_a$, is modeled using $q_a$ actors. Each of these actors should fire periodically with a period equal to $q_a \cdot P_s$. Actor $v_0^0$ corresponds to the first firing of actor $v_a$ and therefore starts at time $s(\tau_a)$. Each next firing corresponds to a new execution of the source and thus is defined to start exactly $P_s$ time later: $s(v_k^j) = s(v_k^{j-1}) + P_s$.

With the HSDF models for the best- and the worst-case behavior and the derived constraints we can use the algorithms from [7] to determine the periodic upper and lower bound schedules, $\hat{s}$ and $\tilde{s}$ respectively.

Algorithm 1 determines the earliest possible enabling time of the first firing of the HSDF actors of the best-case HSDF model. Edges that initially contain tokens are omitted because they will not impose a precedence constraint on the first firing and thus do not influence the enabling time of the first firing. Because all the actors are enabled by a source, the earliest possible schedule after the first firing of HSDF actor $v_a$ is a periodic schedule with period $q_a \cdot P_a$. We use $\tilde{s}_a$ for the earliest enabling time of the first firing of HSDF actor $v_a$.

Algorithm 1

Minimize

$$\sum_{v_i \in V_H} \hat{s}_i$$

Subject to

$$0 \leq k < q_a : \hat{s}_{a,k} = s(\tau_a) + k \cdot P_s$$

$$\forall e_{ij} \in E_H : \tilde{s}_j - \hat{s}_i \geq \hat{\rho}_i$$

$$\text{with } E_H' = \{e_{ij} \mid e_{ij} \in E_H \land \delta(e_{ij}) = 0\}$$

The worst-case enabling times of the HSDF actors in the worst-case HSDF model are determined with Algorithm 2. Enabling times are calculated such that all the precedence constraints are satisfied and such that the number of initial tokens allows a periodic schedule that starts at the determined enabling time. We use $\tilde{s}_a$ for the latest enabling time of the first firing of HSDF actor $v_a$.

Algorithm 2

Minimize

$$\sum_{v_i \in V_H} \tilde{s}_i$$

Subject to

$$0 \leq k < q_a : \tilde{s}_{a,k} = s(\tau_a) + k \cdot P_s$$

$$\forall e_{ij} \in E_H : \tilde{s}_j - \tilde{s}_i \geq \hat{\rho}_i - \delta(e_{ij}) \cdot P_i$$

These determined schedules give for each of the HSDF actors strictly periodic upper and lower bounds on the enabling times of the actors. Although the schedule for every HSDF actor $v_a$ is periodic (with a period $q_a \cdot P_a$), they are not necessarily strictly periodic for the schedule of the corresponding SDF actor. The time between two consecutive enablings of the SDF actor $v_a$ can be different than $P_a$. The only information that is known is that the schedules are periodic over $q_a$ firings. The SDF schedules are thus multi-dimensional periodic.

Using the best-case schedules for the HSDF actor components, we define a lower bound, $\tilde{s}_{a,0}$, on the enabling times of an SDF actor $v_{a,1}$ in Equation (16). Note that $\tilde{s}_{a,0}(i)$ is defined recursively and makes use of the periodicity (period $q_a \cdot P_a$) of the SDF actors.

$$\tilde{s}_{a,0}(i) = \begin{cases} \tilde{s}(v_{a,1}) & \text{if } 0 \leq i < q_a \\ \tilde{s}_{a,0}(i - q_a) + q_a \cdot P_a & \text{otherwise} \end{cases}$$

An SDF actor component in the best-case model consists of only actor $v_{a,1}$ and it thus holds that this lower bound is also a lower bound on the enabling times of task $\tau_a$. We use $\hat{\varepsilon}_a$ for the lower bound on the enabling times of task $\tau_a$:

$$\hat{\varepsilon}_a(i) = \tilde{s}_{a,0}(i)$$

We define the upper bound on the enabling time of task $\tau_a$ by using the maximum finish time, $\hat{f}_{a,1}$, of the SDF actor component and the worst-case enabling time of the SDF actor component. The maximum finish time is defined in Equation (18) using the finish times of the HSDF actor components which are equal to the worst-case enabling of actors $v_{a,1}$ plus their firing durations. We define the finish time of non existing firings to be equal to $-\infty$ to ease the definition of the enabling times of the tasks.

$$\hat{f}_{a,1}(i) = \begin{cases} -\infty & \text{if } i < 0 \\ \hat{s}(v_{a,1}) + \rho(v_{a,1}) & \text{if } 0 \leq i < q_a \\ \hat{f}_{a,1}(i - q_a) + q_a \cdot P_a & \text{otherwise} \end{cases}$$

Task $\tau_a$ always finishes its execution $i$ earlier than actor $v_{a,1}$ finishes its firing $i$ (see Section 6.3). Furthermore, the enabling time of SDF actor $v_{a,0}$ forms an upper bound on the time that task $\tau_a$ can acquire the required number of containers (see Section 6.3). This enabling time, $\hat{\varepsilon}_{a,0}$, is recursively defined using the HSDF schedule as follows:

$$\hat{s}_{a,0}(i) = \begin{cases} \hat{s}(v_{a,0}) & \text{if } 0 \leq i < q_a \\ \hat{s}_{a,0}(i - q_a) + q_a \cdot P_a & \text{otherwise} \end{cases}$$

As defined in Section 4, an execution of a task is enabled when the previous execution is finished and when it can acquire the required number of containers from the buffers. Therefore, task $\tau_a$ is never later enabled than:

$$\hat{\varepsilon}_a(i) = \max(\hat{f}_{a,1}(i - 1), \hat{s}_{a,0}(i))$$

With these earliest and latest enabling times, the distance between $n \geq 2$ consecutive enablings, $\theta_a(n)$, can be determined. The smallest distance between $n$ enablings can be computed as follows:

$$\hat{\varepsilon}_a(n \geq 2) = \min_{0 \leq k < q_a} (\hat{\varepsilon}_a(k + n - 1) - \hat{\varepsilon}_a(k))$$

$$\hat{\theta}_a(n \geq 2) = \max(0, \hat{\varepsilon}_a(n))$$

Note that the bounded domain of the min term in (21) is a direct result of the periodic nature of (16), (18) and (19). Furthermore, it can be shown that $\hat{\varepsilon}_a$ is periodic:
Lemma 1. $\hat{\varepsilon}_a(n + q_a) = \hat{\varepsilon}_a(n) + q_a \cdot P_a$

Proof. For $i \geq q_a$ we have $\hat{\varepsilon}_a(i) = \hat{\varepsilon}_a(i - q_a) + q_a \cdot P_a$ and thus:

$$\hat{\varepsilon}_a(n + q_a) = \min_{0 \leq k \leq q_a} (\hat{\varepsilon}_a(k + n + q_a - 1) - \hat{\varepsilon}_a(k))$$

$$= \min_{0 \leq k \leq q_a} (\hat{\varepsilon}_a(k + n - 1) - \hat{\varepsilon}_a(k)) + q_a \cdot P_a$$

Because $\hat{\varepsilon}_a$ is periodic, $\hat{\theta}_a$ will eventually also be periodic and $\hat{\theta}_a$ thus has a finite length.

The function $\hat{\theta}_a(n)$ can be transformed into $\hat{\theta}_a(\Delta t)$ by using Equation (1) to find the new enabling characterizations for each task.

6.5 Enabling Jitter

Also the enabling jitter based characterization of a task can be derived. On the one hand, such an enabling jitter characterization can be useful because it belongs to the class of standard event models [9]. On the other hand, we use this enabling jitter characterization to compare its accuracy with the accuracy of the more general enabling characterization which is derived in the previous section. This comparison is performed in Section 7.

The minimal distances between enablings of a task $\tau_a$ is conservatively expressed by its enabling jitter $J_a$ and its period $P_a$ with Equation (23) [9].

$$\hat{\theta}_a(n) = \max (0, (n - 1) \cdot P_a - J_a) \quad \text{with } n \geq 2 \quad (23)$$

With this expression we can derive an enabling jitter such that the minimal distances are conservative (smaller) to the minimal distances derived in the previous section.

Lemma 2. The enabling jitter of task $\tau_a$ is conservative when we choose:

$$J_a = \max_{2 \leq n < (\eta_a + 2)} ((n - 1) \cdot P_a - \hat{\varepsilon}_a(n))$$

Proof. From Lemma 1 we know that $\hat{\varepsilon}_a$ has a periodic behavior. Therefore, when we calculate jitter as above we have:

$$\forall n \geq 2 : (n - 1) \cdot P_a - \hat{\varepsilon}_a(n) \leq J_a$$

This can be rewritten to show that the minimal distances are smaller, and thus conservative, with the enabling jitter based characterization:

$$\forall n \geq 2 : (n - 1) \cdot P_a - J_a \leq \hat{\varepsilon}_a(n)$$

$$\forall n \geq 2 : \max (0, (n - 1) \cdot P_a - J_a) \leq \max (0, \hat{\varepsilon}_a(n))$$

$$\forall n \geq 2 : \hat{\theta}_a(n) \leq \hat{\theta}_a(n)$$

7. CASE-STUDY

Radio processing applications often have feedback loops such as time and frequency synchronization loops, automatic gain control loops and loops for re-encoding and adaptive channel equalization. These feedback loops add cyclic constraints which complicates analysis for such applications.

Figure 4 contains the task graph of an example of a radio processing application with a fine frequency tracking loop. The feedback loop is formed by the demodulation task, which determines the amount of frequency offset, and the mixer, which compensates for this offset. The tokens in the feedback loop behave as delays which means that more tokens in this loop result in a reduced adaptation speed of the tracking loop which is undesirable. To illustrate the effect of multi-rate behavior we choose a configurable block size of $k$ items that are processed at once by the mixer task. The ADC source task also produces the same amount of items at once. The period of the ADC source also depends on the block size and is equal to $k \cdot 16$ time units.

The radio processing application is executed on a MPSoC. The processors of this MPSoC use a static priority preemptive scheduler to schedule the tasks. The ADC source does have dedicated hardware support in the MPSoC. The red tasks (mixer and demodulation) are mapped to one processor of the MPSoC and the blue tasks (deinterleaving and frame-formatting) are mapped to another processor. The channel decoding task is mapped to a dedicated hardware accelerator. The demodulation, deinterleaving and frame-formating all process at a granularity of one item. The channel decoding task uses redundancy in the bit stream to correct errors caused by a noisy channel. For example the Viterbi algorithm can be used to decode the input stream. For this case-study the redundancy in the bit stream has a ratio of two to one which means that the channel decoding task produces one corrected item per two consumed items.

Figure 5: Simplified SDF model of a radio processing application with a fine frequency compensation. Analysis results shown for $v_b$ for $k = 3$
In this case-study we focus on the most interesting part of this application, namely, the fine frequency tracking loop. The mixer task and the demodulation task share one processor and the demodulation task is configured with the highest priority. The mixer task has the lowest priority. Due to the priorities of the tasks, this part of the radio processing application has a cyclic resource dependency, i.e., the mixer task delays the demodulation task which in turn pre-empts the mixer task. Because of the feedback loop, this part of the application thus has both a cyclic resource as well as a cyclic data dependency.

Figure 5a shows the SDF model of the part of the application on which we focus. The ADC source is modeled with actor \( v_3 \) and the mixer and demodulation tasks are modeled with actor components \((v_{a,0}, v_{a,1})\) and \((v_{b,0}, v_{b,1})\) respectively. The BCET and the WCET of the demodulation task are equal to \( k \cdot 2 \) and \( k \cdot 4 \) time units respectively and the BCET and WCET of the mixer task are equal to 4 and 6 time units respectively. The source is assumed to produce instantaneously. Figure 5a shows the firing durations belonging to the worst-case temporal model of the application.

The equivalent HSDF model of the SDF model is shown in Figure 5b for which the block size, \( k \), is chosen to be equal to 3. The repetition factors of the SDF actor component \( v_a \) and \( v_b \) are then equal to 1, 1 and 3 respectively. Therefore, \( v_3 \) is modeled with 3 HSDF actor components and the other two SDF actor components with one HSDF actor component. The period of the source actor, \( v_3 \), is equal to \( 16 \cdot 3 = 48 \) time units. The periods of \( v_a \) and \( v_b \) are thus equal to 48 and 16 time units respectively.

After three iterations of the analysis flow, the enabling characterizations have converged and the analysis flow ends. The conclusion is that the number of tokens in the feedback loop, 6 when \( k \) equals 3, are sufficient to meet the throughput, i.e., the ADC source can execute once per 48 time units. If less tokens are chosen in the feedback loop, the accuracy of the frequency compensation is better. However, from the analysis flow we conclude that with less than 6 tokens, the throughput cannot be met. The response time of the mixer task is after the first iteration of the flow equal to 30 time units, after the second iteration 42 time units and it converges in the third iteration to 48 time units.

Each iteration of the analysis flow determines the schedules of the HSDF model. Figure 5d shows these schedules of actor \( v_{b,3} \) in the last iteration of the analysis flow when \( k \) is equal to 3. With the schedules, the enabling characterization is determined using (22) and (1). This enabling characterization, \( \eta_2 \), is shown in Figure 5d for task \( v_3 \) (the demodulation task) with the red solid line. Note that this enabling characterization is multi-dimensional periodic which cannot be expressed with a period and jitter characterization accurately. The blue dashed line in Figure 5d illustrates the jitter based enabling characterization from Section 6.5, denoted by \( \eta'_j \). As is shown in Figure 5d, the interference is higher according to this jitter based enabling characterization which results in larger response times.

Table 1 illustrates this in more detail. It contains the different worst-case response times of the mixer task given different block sizes \( k \). The second column contains the response times obtained by using the proposed enabling characterization. The third column shows the converged response times when an enabling characterization based on enabling jitters is used and the fourth shows the difference between the two. When the block size is equal to 1, the results obtained with the enabling jitter characterization are equal to the results obtained with the proposed enabling jitter. However, for larger block sizes the difference rapidly increases. For a block size of 32 items, the worst-case response time obtained with the enabling jitter characteristics is 86.7% larger than the response time that is calculated by using the proposed enabling characterization.

Over-approximating the worst-case response times by using an enabling jitter based characterization is even less desirable because it can also have an effect on the temporal results of the other tasks in the application. The tasks later in the pipeline than the mixer task, for example the deinterleaving task, will observe a larger difference between the best-case and the worst-case schedule when the worst-case response time of the mixer task is larger. With a larger difference between the best-case and the worst-case schedule, the interference bounds will also increase. This can again lead to larger response times of the tasks that share a processor with such a task.

Table 1: Comparison between the worst-case response times of the mixer task obtained by the proposed enabling characterization and the jitter based characterization

<table>
<thead>
<tr>
<th>( k )</th>
<th>( R_{mixer} )</th>
<th>( R'_{mixer} )</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>16</td>
<td>+ 0.0%</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>44</td>
<td>+ 37.5%</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>78</td>
<td>+ 62.5%</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>106</td>
<td>+ 65.6%</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>230</td>
<td>+ 79.7%</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>472</td>
<td>+ 84.4%</td>
</tr>
<tr>
<td>32</td>
<td>512</td>
<td>956</td>
<td>+ 86.7%</td>
</tr>
</tbody>
</table>

8. CONCLUSION

The temporal analysis method presented in this paper is applicable to analyze systems with non-starvation-free schedulers using dataflow analysis techniques for SDF graphs. It is shown that the analysis techniques do not make assumptions on the graph topology and do therefore allow cyclic graph topologies. Also the combination between cyclic...
resource dependencies and cyclic data dependencies is supported. Furthermore, it is shown that buffer capacity constraints can be taken into account during the analysis.

Upper and lower bound schedules are used to determine enabling characterizations. It is shown in the case-study that this enabling characterization allows more modeling freedom than characterizations based on periods and enabling jitters. This is illustrated in the case-study. The case-study also showed that a radio processing application with both cyclic resource and cyclic data dependencies can be analyzed. Next to that it is illustrated in the case-study that the used enabling characterization is more accurate than an enabling characterization based on periods and enabling jitters. The response times in the case-study improve up to 87% compared to the enabling jitter based characterization.

We expect that the presented throughput analysis approach based on enabling characterizations can be generalized, enabling the accurate analysis of systems with non-starvation-free schedulers in combination with more expressive dataflow models, such as Cyclo-Static Dataflow (CSDF) [2] and Variable-Rate Dataflow (VRDF) [29].

References


