2014 17th Euromicro Conference on Digital System Design

Accuracy Improvement of Dataflow Analysis for Cyclic Stream Processing Applications Scheduled by Static Priority Preemptive Schedulers

Philip S. Wilmanns* philip.wilmanns@utwente.nl
Joost P.H.M. Hausmans* joost.hausmans@utwente.nl
Stefan J. Geuns* stefan.geuns@utwente.nl
Marco J.G. Bekooij*‡ marco.bekooij@nxp.com
*University of Twente, Enschede, The Netherlands ‡NXP Semiconductors, Eindhoven, The Netherlands

***Abstract*—Stream processing applications executed on embedded multiprocessor systems regularly contain cyclic data dependencies due to the presence of feedback loops and bounded FIFO buffers. Dataflow modeling is suitable for the temporal analysis of such applications. However, the accuracy can be unsatisfactory as existing temporal analysis techniques ignore that cyclic data dependencies limit interference between tasks executed on shared processors.**

This paper presents a dataflow analysis approach that increases the analysis accuracy by taking into account that cyclic data dependencies limit interference between tasks. It is shown that the approach is applicable for single-rate stream processing applications that are executed on multiprocessor systems using static priority preemptive schedulers.

The improvement of accuracy is demonstrated in a case study employing a WLAN 802.11p transceiver application that is executed on a multiprocessor system with shared processors.

I. Introduction

Real-time stream processing applications such as Software Defined Radios (SDRs) are usually executed on embedded multiprocessor systems in a data-driven fashion. A number of dataflow analysis techniques exist which can be used to verify whether throughput and latency constraints can be satisfied [1], [2]. These analysis techniques are also used for the computation of required buffer capacities [3], scheduler settings [4] and a suitable task-to-processor assignment [5]. Furthermore, they also form the basis for synchronization overhead minimization techniques such as task clustering [6] and resynchronization [7].

In particular, dataflow analysis is suitable for the analysis of stream processing applications with cyclic data dependencies as well as modal behavior [8], [9]. Cyclic data dependencies are regularly found in models of SDRs due to the presence of feedback loops and bounded First-In-First-Out (FIFO) buffers used for inter-task communication. Examples of modes in receiver applications are acquisition, synchronization and decoding, with each mode activating different parts of the application.

Dataflow analysis techniques in the context of data-driven systems were until recently only applicable for systems with starvation-free schedulers such as Time Division Multiplex (TDM) [2]. In [10] it has been shown that dataflow analysis techniques can also be used for the broader class of non-starvation-free schedulers such as static priority preemptive.

Figure 1 depicts two tasks with a cyclic data dependency that are scheduled by such a static priority preemptive scheduler. As task τ_j has a higher priority than task τ_i, its executions

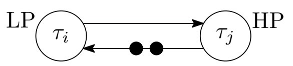

Fig. 1. Two tasks with a cyclic data dependency that are scheduled by a static priority preemptive scheduler.

can preempt and delay executions of task τ_i. This interference has to be incorporated into the response time of task τ_i in order to obtain temporally conservative analysis results.

By definition, the interference of tasks scheduled with non-starvation-free schedulers can be only limited in the analysis model if it can be derived how often these tasks are maximally enabled per time interval. In [10] it is shown that such an enabling characterization can be calculated using the periods and jitters of interfering tasks.

However, existing analysis techniques do not capture that cyclic data dependencies limit interference as well, which can lead to an unsatisfactory accuracy of analysis results. This relation can be illustrated with the cyclic data dependency in the given example. The execution of a task begins with the consumption of one token from all its incoming edges and finishes with the production of one token on all its outgoing edges. This implies that a task cannot be executed if there are no tokens on one of its incoming edges. The presented cyclic data dependency contains two tokens, for instance corresponding to a FIFO buffer with two containers. One of the two tokens is always held by task τ_i during each of its executions, as an execution cannot begin without a consumption of one token and a production cannot happen before the end of an execution. Therefore at most one token can be available on the incoming edge of task τ_j during each execution of task τ_i. This in turn allows for only one execution of task τ_j per execution of task τ_i, which effectively limits interference between the tasks.

In this paper we present a dataflow analysis approach for single-rate real-time stream processing applications that realizes an accuracy improvement compared to existing temporal analysis techniques by taking into account that cyclic data dependencies limit interference. This is incorporated into the temporal analysis by the derivation of maximum response time equations that are not only parameterized in periods and jitters, but also in the number of tokens on cyclic data dependencies between interfering tasks. We show that the presented analysis approach, which has an exponential time-complexity, is applicable for systems with static priority preemptive schedulers and can be used for the verification of temporal constraints as well as a calculation of sufficient buffer capacities.

The remainder of this paper is structured as follows. Section II presents related work. In Section III we present our analysis flow and derive the maximum response time equations which capture that cyclic data dependencies limit interference. Section IV shows the improvement of analysis accuracy in a case study and Section V states the conclusions.

II. Related Work

In [2] it has been shown that dataflow analysis can be used to derive the minimum throughput of applications that are executed on multiprocessor systems in a data-driven fashion. This approach is restricted to systems that employ starvation-free schedulers, for which the minimum service of a task can be determined independently of the enabling characterization of other tasks. Recently, a dataflow analysis approach has been introduced in [10], which takes the enabling characterization of tasks into account. This approach extends the scope of dataflow analysis techniques by allowing an analysis for systems with non-starvation-free schedulers as well, at the cost of an exponential time-complexity. Moreover, the usage of an enabling characterization of tasks enables an accuracy improvement for starvation-free schedulers. However, it is not considered that cyclic data dependencies limit interference, which causes a lower accuracy of analysis results compared to our approach.

The SymTA/S approach [11] uses an iterative procedure of traffic characterization and response time calculation. However, the employed traffic characterization is derived in the time-interval domain. Therefore the correlation between streams cannot be captured accurately, which can lead to an unsatisfactory accuracy of analysis results. Moreover, it is not taken into account that cyclic data dependencies limit interference.

Modular Performance Analysis (MPA) [12] is based on Real-Time Calculus (RTC) [13] and, as SymTA/S, derives its traffic characterization in the time-interval domain. In [14] it has been shown how (potentially cyclic) data dependencies can be handled in a modified MPA framework. The main difference between the modified MPA framework and the one presented in [12] is that the traffic characterization is not derived in the time-interval, but in the time domain, which allows for an accurate capturing of correlated streams and hence for more accurate analysis results. However, the effect that cyclic data dependencies limit interference between tasks of the same application is not discussed, the combination of cyclic data and resource dependencies is not considered at all. Cyclic resource dependencies in the original MPA framework from [12] are discussed in [15], but not in combination with cyclic data dependencies. This combination is difficult due to the requirement of an accurate translation between a traffic characterization in the time domain and the resulting resource usage characterization in the time-interval domain.

Time offsets on the executions of tasks make use of data dependencies to limit interference between tasks as well. Employing time offsets to limit interference was firstly proposed in [16]. This approach makes use of static time offsets, which only allow for the correct characterization of systems with strictly periodic schedules. Therefore, multiple generalizations of this approach were introduced, e.g. [17], [18], [19], extending the applicability of time offsets to systems with data-driven schedulers. However, none of these methods is applicable for arbitrary (cyclic) task graphs. In contrast, our approach does not only allow for cyclic data dependencies, but exploits the presence of cyclic data dependencies for an accuracy improvement of analysis results.

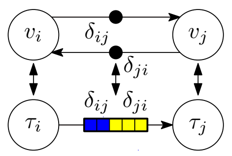

Fig. 2. One-to-one relation between HSDF model and task graph.

III. Temporal Analysis

In this section we explain our temporal analysis approach. Section III-A describes the analysis model and Section III-B the analysis flow. In Section III-C we present equations for the calculation of an upper bound on the response time of tasks that consider the effect of cyclic data dependencies limiting interference. The employed interference limitation due to cyclic data dependencies is detailed in Section III-D. Section III-E presents Linear Program (LP) algorithms that are used to derive upper and lower bounds on the start times of tasks, as well as upper bounds on their jitters, and Section III-F describes a technique for determining sufficient buffer capacities.

In the remainder of this paper we will refer to the upper (lower) bounds on the response times of tasks as maximum (minimum) response times. Analogously, we will call the upper (lower) bounds on start times maximum (minimum) start times and upper bounds on jitters maximum jitters, respectively.

A. Analysis Model

We make use of Homogeneous Synchronous Dataflow (HSDF) graphs to calculate lower bounds on the best-case and upper bounds on the worst-case schedule of an analyzed application. These schedules are used for the verification of temporal constraints, the derivation of maximum jitters and a calculation of sufficient buffer capacities.

An HSDF graph is a directed graph $G = (V, E, \delta, \rho)$ that consists of a set of actors V and a set of directed edges E connecting these actors. Actors $v_i \in V$ communicate by producing tokens on and consuming tokens from the edges, which represent unbounded queues. An edge $e_{ij} = (v_i, v_j) \in E$ initially contains $\delta(e_{ij})$ tokens. An actor v_i is enabled to fire if a token is available on each of its incoming edges. Furthermore, the firing duration ρ_i specifies the difference between the start and finish time of a firing of an actor v_i. At the start of a firing an actor consumes one token from all its incoming edges and when it finishes it produces one token on each of its outgoing edges.

With our temporal analysis approach we analyze applications that can be described by one or more task graphs. We specify a task graph as a weakly connected directed graph, with its vertices representing tasks and its directed edges representing FIFO buffers. Each task graph is single-rate and has a single, strictly periodic source τ_s enabling all other tasks in the task graph. Write operations on FIFO buffers are characterized by an acquisition of space, followed by the actual writing of data and finalized by a release of data. Analogously, read operations are described by an acquisition of data, the reading of data and a release of space.

As depicted in Figure 2, we model each task of a task graph as a single HSDF actor. Such a one-to-one relation between tasks and actors can be maintained if it is ensured that all acquisition operations of a task happen at the beginning and all release operations at the end of its execution. This behavior can be guaranteed by a scheduler that performs the required acquire operations when a task is started and the corresponding release operations when a task finishes.

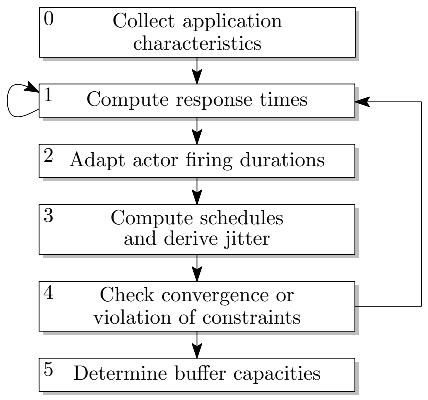
0 Collect application characteristics
1 Compute response times
2 Adapt actor firing durations
3 Compute schedules and derive jitter
4 Check convergence or violation of constraints
5 Determine buffer capacities

Fig. 3. Overview of the analysis flow.

Exchanging data between tasks over a FIFO buffer can then be modeled by a directed cycle in an HSDF graph as depicted in Figure 2, with the number of initial tokens δ_{ij} on the edge from actor v_i to actor v_j being equal to the number of initially full containers in the corresponding FIFO buffer and the number of initial tokens δ_{ji} on the edge from actor v_j to actor v_i being equal to the number of initially free containers. The consumption of a token by actor v_i then corresponds to an acquisition of space, whereas a token production by that actor corresponds to a release of data on the modeled FIFO buffer. Analogously, the consumption of a token by actor v_j corresponds to an acquisition of data and the production of a token to a release of space.

In the following, we will derive such HSDF graphs from task graphs to compute minimum and maximum start times of actors, which are bounds on the start times of the corresponding tasks. The minimum start times hence form a lower bound on the best-case schedule of a task graph, whereas the maximum start times determine an upper bound on the worst-case schedule. A schedule is called admissible if no task in the schedule is started before it is enabled. The start times of an admissible schedule therefore do not violate any temporal constraints.

B. Analysis Flow

Figure 3 depicts the flow of our temporal analysis approach. We use this analysis flow for the verification of throughput constraints and for the calculation of sufficient buffer capacities.

In step 0, application characteristics are collected, which form the input of our analysis. These characteristics include a task graph as specified in the previous section, a fixed task-to-processor mapping, a specification of scheduler settings and a set of temporal constraints, which are usually derived from the period of the source. Based on these characteristics, minimum and maximum response times of tasks are derived in step 1. This step makes use of the maximum response time equations which take into account that cyclic data dependencies limit interference.

Step 2 makes use of an HSDF graph corresponding to the analyzed task graph, with the minimum and maximum firing durations of the actors set to the minimum and maximum response times of the corresponding tasks. Given this HSDF graph, two periodic schedules are computed in step 3, the first a lower bound on the best-case schedule and the second an upper bound on the worst-case schedule. Using these schedules, the maximum jitters of tasks are derived. In step 4, the two schedules are checked against the temporal constraints and it is verified whether all maximum jitters have converged, i.e. have not changed since the previous iteration of the algorithm. If a constraint is violated then the algorithm stops. Otherwise, depending on whether the maximum jitters have converged, the algorithm either continues with step 5, or repeats the steps 1 to 4 until either maximum jitter convergence or constraint violation is observed.

If buffer capacities are given then they are considered in both maximum response times and maximum start time calculations, using the correspondence depicted in Figure 2. However, our approach can also be used to determine sufficient buffer capacities, which is done in step 5 of the analysis flow.

C. Maximum Response Times of Tasks

In this section we will include the effect that cyclic data dependencies limit interference into equations for the calculation of maximum response times of tasks.

Let P_j be the period of a task τ_j and J_j its maximum jitter. The maximum number of enablings a task τ_j can have during a time interval Δt can then be determined as follows [20]:

$$\hat{\eta}_j(\Delta t) = \left\lceil \frac{J_j + \Delta t}{P_j} \right\rceil \tag{1}$$

Using this enabling characterization, the response time of a task scheduled by a static priority preemptive scheduler can be bounded from above with the following maximum response time equations [21]:

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

$$\hat{R}_i = \max_{1 \le q}(w_i(q) - (q-1) \cdot P_i) \tag{3}$$

The busy period $w_i(q)$ is an upper bound on the maximum amount of time required to finish q consecutive executions of a task τ_i, C_i is the Worst-Case Execution Time (WCET) of one firing of task τ_i and the set $hp(i)$ contains all tasks τ_j with a higher priority than task τ_i. Besides $q = 1$, only values of $q > 1$ for which $w_i(q-1) > (q-1) \cdot P_i$ holds need to be considered [21].

In order to include the effect that cyclic data dependencies limit interference into the maximum response time equations, we introduce a new upper bound on the maximum number of enablings a task τ_j can have during a time interval Δt in which a task τ_i is executed q consecutive times:

$$\hat{\eta}'_{j \to i}(\Delta t, q) = \min(\hat{\eta}_j(\Delta t), \gamma_{j \to i}(q)) \tag{4}$$

The first term of the minimum function denotes the maximum interference of a task τ_j on a task τ_i, given that their enablings are independent of each other. This term ensures that the maximum response time of a task τ_i cannot become more pessimistic than by applying Equation 1. The function $\gamma_{j \to i}(q)$, which will be derived in the next section, represents an upper bound on the maximum number of enablings a task τ_j can have during q consecutive executions of a task τ_i due to cyclic data dependencies between the tasks. As both terms of the minimum function are upper bounds on the number of enablings of a task τ_j, $\hat{\eta}'_{j \to i}(\Delta t, q)$ can be used to reduce the busy period calculated with Equation 2, leading to the following maximum response time equations:

$$w'_i(q) = q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}'_{j \to i}(w_i(q), q) \cdot C_j \tag{5}$$

$$\hat{R}'_i = \max_{1 \le q}(w'_i(q) - (q-1) \cdot P_i) \tag{6}$$

We require the maximum response time calculated with Equation 6 to be conservative and to be more accurate than the maximum response time calculated with Equation 3. This is the case if the same holds for the reduced busy period $w'_i(q)$. Hence we have to prove the following lemma:

Lemma 1: Let $w^*_i(q)$ be the actual, and thus in general unknown, maximum amount of time required to finish q consecutive executions of a task τ_i scheduled by a static priority preemptive scheduler. Then it holds that the reduced busy period $w'_i(q)$ is an upper bound on $w^*_i(q)$ and that $w'_i(q)$ is a tighter upper bound than $w_i(q)$, i.e. $w^*_i(q) \le w'_i(q) \le w_i(q)$.

Proof: At first we prove that $w'_i(q) \le w_i(q)$. From the relation $\forall_{x,y}: \min(x,y) \le x$ it directly follows that $\forall_{\Delta t,q}: \hat{\eta}'_{j\to i}(\Delta t, q) \le \hat{\eta}_j(\Delta t)$. Therefore it also holds that:

$$\begin{aligned} w'_i(q) &= q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}'_{j\to i}(w_i(q), q) \cdot C_j \\ &\le q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}_j(w_i(q)) \cdot C_j = w_i(q) \end{aligned}$$

For the proof of $w^*_i(q) \le w'_i(q)$ we firstly define $\hat{\eta}^*_{j\to i}(\Delta t, q)$ as the actual maximum number of enablings a task τ_j can have during a time interval Δt and q consecutive executions of a task τ_i. As both terms in the minimum function in $\hat{\eta}'_{j\to i}(\Delta t, q)$ are upper bounds on the number of enablings of a task τ_j it holds that $\forall_{\Delta t,q}: \hat{\eta}^*_{j\to i}(\Delta t, q) \le \hat{\eta}'_{j\to i}(\Delta t, q)$. Furthermore, $\hat{\eta}'_{j\to i}(\Delta t, q)$ is monotonically increasing in Δt, due to the monotonicity of the ceiling and minimum functions in Equations 1 and 4. With $w_i(q) \ge w^*_i(q)$ it then follows:

$$\hat{\eta}'_{j\to i}(w_i(q), q) \ge \hat{\eta}'_{j\to i}(w^*_i(q), q) \ge \hat{\eta}^*_{j\to i}(w^*_i(q), q)$$

Using this relation it follows for the reduced busy period $w'_i(q)$:

$$\begin{aligned} w'_i(q) &= q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}'_{j\to i}(w_i(q), q) \cdot C_j \qquad (7) \\ &\ge q \cdot C_i + \sum_{j \in hp(i)} \hat{\eta}^*_{j\to i}(w^*_i(q), q) \cdot C_j \end{aligned}$$

For static priority preemptive schedulers, the last term is an overapproximation of $w^*_i(q)$ since no other components than the time required for q executions of task τ_i and the time required for $\hat{\eta}^*_{j\to i}(w^*_i(q), q)$ executions of all tasks $\tau_j \in hp(i)$ can contribute to $w^*_i(q)$. Thus it holds that $w^*_i(q) \le w'_i(q)$. ■

D. Limiting Interference with Cyclic Data Dependencies

In this section we will derive the function $\gamma_{j\to i}(q)$ that calculates the maximum number of enablings a task τ_j can have during q consecutive executions of a task τ_i, based on cyclic data dependencies between the tasks. We will make use of HSDF modeling in this derivation. Hence we will not refer to tasks in the remainder of this section, but to HSDF actors corresponding to tasks, according to Figure 2. We employ precedence constraints on firings of actors in the derivation of $\gamma_{j\to i}(q)$, which are defined as follows:

Definition 1: We define $v_j(n)$ as firing n of an actor v_j. If a firing $v_j(n)$ of an actor v_j cannot start before a firing $v_i(m)$ of an actor v_i has finished, then we say that $v_i(m)$ precedes $v_j(n)$ and denote this relation as $v_j(n) \succ v_i(m)$. The firing number m is defined in a sequential manner such that a firing $n > m$ of an actor v_j cannot start before its firing m has started. $v_j(0)$ denotes the first firing of an actor v_j.

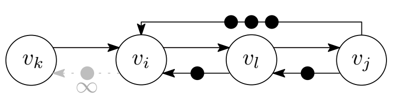

Fig. 4. Paths in a cyclic HSDF graph.

According to the definition of an HSDF graph an actor can only fire if there is at least one token on all its incoming edges. When an actor completes its firing it produces one token on each of its outgoing edges. If an edge e_{ij} contains $\delta(e_{ij})$ initial tokens then actor v_j can fire $\delta(e_{ij})$ times before it must get enabled by a completed firing of actor v_i for a subsequent firing. This implies that firing m of actor v_j cannot start before firing $m - \delta(e_{ij})$ of actor v_i has finished, which is captured by the following precedence constraint:

$$\forall_{m \ge \delta(e_{ij})}: \; v_j(m) \succ v_i(m - \delta(e_{ij})) \qquad (8)$$

Definition 2: A directed path on an HSDF graph G is defined as a sequence of edges:

$$p = \langle e_0, e_1, e_2, \ldots, e_{|p|-1} \rangle$$

with $e_k = (v_k, v_{k+1}) \in E$ and $|p|$ the number of edges on the path. We speak of a path from an actor v_i to an actor v_j if $e_0 = (v_i, v_1)$ and $e_{|p|-1} = (v_{|p|-1}, v_j)$. The set $\mathcal{P}_{ij}$ contains all paths from an actor v_i to an actor v_j.

Definition 3: The number of initial tokens on a path p is defined as the sum of initial tokens on its edges:

$$\delta(p) = \sum_{k=0}^{|p|-1} \delta(e_k)$$

Using this definition we define the minimum number of initial tokens on a path from an actor v_i to an actor v_j as:

$$\delta(\mathcal{P}_{ij}) = \begin{cases} \min_{p \in \mathcal{P}_{ij}} \delta(p) & \text{if } \mathcal{P}_{ij} \ne \emptyset \\ \infty & \text{otherwise} \end{cases} \qquad (9)$$

The relation between edges and paths is illustrated by the HSDF graph depicted in Figure 4, which contains one path from actor v_i to actor v_j and two paths from actor v_j to actor v_i. The minimum number of initial tokens on paths from actor v_i to actor v_j is $\delta(\mathcal{P}_{ij}) = 0$, due to $\delta(e_{il}) + \delta(e_{lj}) = 0$, whereas the minimum number of initial tokens on paths from actor v_j to actor v_i equals to $\delta(\mathcal{P}_{ji}) = \min(\delta(e_{jl}) + \delta(e_{li}), \delta(e_{ji})) = 2$.

Consider the dotted edge e_{ik} from actor v_i to actor v_k. As $\delta(e_{ik}) = \infty$, there are always sufficient tokens on this edge for actor v_k to fire, independent of firings of actor v_i. Due to this independence of firings, an edge with an infinite number of initial tokens can be considered equivalent to the absence of that edge. Consequently, a non-existent path between two actors can also be considered equivalent to a path containing a single edge with an infinite number of initial tokens. This equivalence is reflected by the definition of $\delta(\mathcal{P}_{ij}) = \infty$ for $\mathcal{P}_{ij} = \emptyset$.

The Floyd-Warshall Algorithm [22] can be used to compute minimum distances between all pairs of vertices of a weighted directed graph. We apply this algorithm on HSDF graphs, considering actors as vertices, using the same edges between vertices as between actors and setting the required edge weights to numbers of initial tokens. The resulting minimum distances between all pairs of vertices then just equal to $\delta(\mathcal{P}_{ij})$ for all pairs of actors in the HSDF graph. Note that in our case edge weights cannot be negative, as an HSDF graph cannot contain edges with negative numbers of initial tokens. Therefore there also cannot be negative cycles in the graph processed by the Floyd-Warshall Algorithm, guaranteeing that the correct $\delta(\mathcal{P}_{ij})$ can be always obtained.

Until now we have only defined precedence constraints for single edges. Proving the following lemma allows us to use precedence constraints for paths as well.

Lemma 2: For two actors $v_i, v_j \in V$ it holds that the following precedence constraint is the tightest precedence constraint imposed by a path from actor v_i to actor v_j:

$$\forall_{m \geq \delta(\mathcal{P}_{ij})}: \ v_j(m) \succ v_i(m - \delta(\mathcal{P}_{ij}))$$

Proof: All edges of a path $p \in \mathcal{P}_{ij}$ from an actor v_i to an actor v_j impose precedence constraints as defined in Equation 8. By recursively substituting the firing numbers of the actors on such a path we derive the following relation:

$$\begin{aligned} v_j(m) = v_{|p|}(m) &\succ v_{|p|-1}(m - \delta(e_{|p|-1})) \\ &\succ v_{|p|-2}(m - \delta(e_{|p|-1}) - \delta(e_{|p|-2})) \\ &\succ \dots \\ &\succ v_0(m - \sum_{k=0}^{|p|-1} \delta(e_k)) = v_i(m - \delta(p)) \end{aligned} \tag{10}$$

Due to the definition of $\delta(\mathcal{P}_{ij})$, there must be a path $p \in \mathcal{P}_{ij}$ with $\delta(p) = \delta(\mathcal{P}_{ij})$. As $\forall_{p \in \mathcal{P}_{ij}}: \ \delta(\mathcal{P}_{ij}) \leq \delta(p)$, it holds that no path from an actor v_i can impose a tighter precedence constraint on the firings of actor v_j than a path $p \in \mathcal{P}_{ij}$ with $\delta(p) = \delta(\mathcal{P}_{ij})$. By substituting $\delta(p)$ in Equation 10 with $\delta(\mathcal{P}_{ij})$ we obtain the precedence constraint from Lemma 2. ■

Based on Definition 3 and the definition of HSDF graphs we define sets of firings of actors that can overlap in time with a single firing of an actor v_i.

Definition 4: The outgoing interference set $\mathcal{I}^{\text{out}}_{v_j \to v_i(m)}$ contains all firings of an actor v_j that can overlap in time with firing m of an actor v_i, despite the precedence constraints imposed by the paths from actor v_i to actor v_j.

From Lemma 2 we derive:

$$\begin{aligned} &\forall_{m \geq \delta(\mathcal{P}_{ij})}: \quad v_j(m) \succ v_i(m - \delta(\mathcal{P}_{ij})) \\ \Leftrightarrow \ &\forall_{m \geq 0}: \quad v_j(m + \delta(\mathcal{P}_{ij})) \succ v_i(m) \end{aligned}$$

which implies together with the sequentiality of actor firings that only firings $v_j(n)$ with $n < m + \delta(\mathcal{P}_{ij})$ can occur before the end of firing $v_i(m)$. With this observation we can formalize Definition 4 as follows:

$$\mathcal{I}^{\text{out}}_{v_j \to v_i(m)} = \{v_j(n) \mid n < m + \delta(\mathcal{P}_{ij})\}$$

Analogous to the outgoing interference set we define an interference set for incoming paths of actor v_i:

Definition 5: The incoming interference set $\mathcal{I}^{\text{in}}_{v_j \to v_i(m)}$ contains all firings of an actor v_j that can take place during one firing m of an actor v_i, despite the precedence constraints imposed by the paths from actor v_j to actor v_i.

By using Lemma 2 we obtain:

$$\forall_{m \geq \delta(\mathcal{P}_{ji})}: \ v_i(m) \succ v_j(m - \delta(\mathcal{P}_{ji}))$$

From this it follows with the sequentiality of actor firings that only firings $v_j(n)$ with $n > m - \delta(\mathcal{P}_{ji})$ can occur after the start of firing $v_i(m)$. This leads to the formalization of $\mathcal{I}^{\text{in}}_{v_j \to v_i(m)}$:

$$\mathcal{I}^{\text{in}}_{v_j \to v_i(m)} = \{v_j(n) \mid n > m - \delta(\mathcal{P}_{ji})\}$$

Algorithm 1

```
Minimize $\sum_{v_i \in V} \check{s}_i$
Subject to: $\check{s}_s = 0$
    $\forall_{e_{ij} \in E'}: \ \check{s}_j - \check{s}_i \geq \check{\rho}_i$
    with $E' = \{e \mid e \in E \ \wedge \ \delta(e) = 0\}$
```

As we are not only interested in the firings of an actor v_j that can take place during a single firing of an actor v_i, but the firings that can take place during q consecutive firings of an actor v_i, we derive interference sets for q consecutive firings as the union of interference sets for single firings:

$$\begin{aligned} \mathcal{I}^{\text{out}}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}} &= \bigcup_{k=m}^{m+q-1} \mathcal{I}^{\text{out}}_{v_j \to v_i(k)} \\ &= \{v_j(n) \mid n < m + q - 1 + \delta(\mathcal{P}_{ij})\} \\ \mathcal{I}^{\text{in}}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}} &= \bigcup_{k=m}^{m+q-1} \mathcal{I}^{\text{in}}_{v_j \to v_i(k)} \\ &= \{v_j(n) \mid n > m - \delta(\mathcal{P}_{ji})\} \end{aligned}$$

To derive all firings of an actor v_j that can take place during q consecutive firings of an actor v_i, we draw the intersection between outgoing and incoming interference sets:

$$\begin{aligned} &\mathcal{I}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}} \\ &\quad = \mathcal{I}^{\text{out}}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}} \cap \mathcal{I}^{\text{in}}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}} \\ &\quad = \{v_j(n) \mid m - \delta(\mathcal{P}_{ji}) < n < m + q - 1 + \delta(\mathcal{P}_{ij})\} \end{aligned}$$

The number of firings of actor v_j that can interfere with q consecutive firings of actor v_i then equals to the number of elements in $\mathcal{I}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}}$ (with $\delta^{\circ}_{ij} = \delta(\mathcal{P}_{ij}) + \delta(\mathcal{P}_{ji})$ a shorthand notation):

$$\gamma_{j \to i}(q) = \left|\mathcal{I}_{v_j \to \{v_i(m),\dots,v_i(m+q-1)\}}\right| = \delta^{\circ}_{ij} + q - 2$$

Considering Equation 4 it follows that $\hat{\eta}'_{j \to i}(w_i(q), q)$ is equal to $\hat{\eta}_j(w_i(q))$ if there is no directed cycle containing both actors v_i and v_j, as then either $\delta(\mathcal{P}_{ij})$, $\delta(\mathcal{P}_{ji})$ or both are infinite. However, if the actors v_i and v_j are both part of a single directed cycle then $\gamma_{j \to i}(q)$ is finite, which can lead to a tighter upper bound on the response time of actor v_i. Note that a cycle does not necessarily have to be a simple cycle, as it is illustrated by the cycle in Figure 4 consisting of the paths $\{e_{il}, e_{lj}\}$ and $\{e_{jl}, e_{li}\}$.

E. Start Times and Maximum Jitters

In [10] it has been shown that a lower bound on the best-case schedule for a given task graph can be computed by solving the LP presented in Algorithm 1. In a similar fashion, an upper bound on the worst-case schedule can be computed by solving the LP presented in Algorithm 2. We set the minimum firing durations $\check{\rho}_i$ in Algorithm 1 to the Best-Case Execution Times (BCETs) of the corresponding tasks and the maximum firing durations $\hat{\rho}_i$ in Algorithm 2 to the maximum response times obtained by Equation 6. The minimum and maximum start times obtained by solving the LPs then just form the bounds on the best-case and worst-case schedules, respectively.

Given that both bounds on best-case and worst-case schedules are admissible, the maximum jitters of tasks can be conservatively bounded by $J_i = \hat{s}_i - \check{s}_i$.

Algorithm 2

$$\begin{aligned}
\text{Minimize } & \sum_{v_i \in V} \hat{s}_i \\
\text{Subject to: } & \hat{s}_s = 0 \\
& \forall_{e_{ij} \in E}\colon\ \hat{s}_j - \hat{s}_i \geq \hat{\rho}_i - \delta(e_{ij}) \cdot P_i
\end{aligned}$$

F. Sufficient Buffer Capacities

Our temporal analysis approach can handle FIFO buffers of given capacities, due to the correspondence between FIFO buffers and cyclic data dependencies presented in Section III-A. Moreover, the approach can be also used to determine sufficiently large buffer capacities.

As in [10], we assume unknown buffer capacities to be infinite in the steps 2 to 4 of our analysis flow, which corresponds to setting $\delta(e_{ji}) = \infty$ in Figure 2. Moreover, we also set unknown buffer capacities to infinite in the maximum response time calculations of step 1. If for this scenario all maximum jitters converge without a violation of temporal constraints, we obtain two admissible schedules, one a lower bound on the best-case and the other an upper bound on the worst-case schedule. Based on these results we can then determine sufficient $\delta(e_{ji})$ such that both schedules remain admissible.

The lower bound on the best-case schedule computed with Algorithm 1 is not dependent on buffer capacities, as it only considers the subset of edges that do not contain any initial tokens. However, the upper bound on the worst-case schedule computed with Algorithm 2 can become inadmissible if too small buffer capacities are chosen. Directly following from the constraints in Algorithm 2 we conclude that a buffer capacity is sufficiently large to keep the upper bound on the worst-case schedule admissible, if it is chosen larger or equal to the smallest integer that satisfies:

$$\delta(e_{ji}) \geq \frac{\hat{\rho}_j + \hat{s}_j - \hat{s}_i}{P_j} \tag{11}$$

Note that the relation between tokens and maximum response times does not have to be taken into account in the calculation of sufficient buffer capacities, which is due to the fact that lower buffer capacities cannot lead to larger maximum response times, and that smaller maximum response times cannot lead to larger maximum start times. However, we will demonstrate in Section IV that assuming buffer capacities to be smaller than infinite during analysis can be used to reduce the jitters of tasks, such that previously unsatisfiable temporal constraints become satisfiable.

IV. Case Study

In this section we illustrate with a practical example that a consideration of cyclic data dependencies in the maximum response time calculations leads to an improved accuracy of temporal analysis results. Furthermore, we will demonstrate the counter-intuitive effect that a reduction of buffer capacities to smaller than infinite can lead to previously unsatisfiable constraints becoming satisfiable.

We analyze the task graph of a WLAN 802.11p transceiver [23]. This application has several modes and is executed on a multiprocessor system for performance reasons. We will consider only the part of the task graph that is active during packet decoding mode.

An HSDF model corresponding to a realistic task graph of the packet decoding mode is shown in Figure 5. The

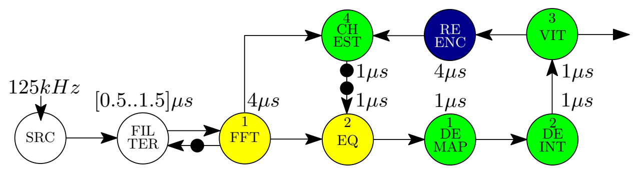

Fig. 5. HSDF graph of the packet decoder of a WLAN 802.11p transceiver.

unit of all times in the remainder of this section is $1\mu s$. A periodic source with a frequency of $125kHz$ models the input of this dataflow graph, which corresponds to a source period of $P_{\text{SRC}} = 8$. All received symbols are firstly processed by a hardware filter with a variable WCET and then processed by an FFT. The hardware filter and the FFT communicate via a FIFO buffer of the capacity one, which is represented by the leftmost cyclic data dependency. For the other tasks we will use our analysis flow to determine sufficient buffer capacities. The backward edges, whose numbers of initial tokens are set to infinity before buffer capacities are determined, are omitted in Figure 5. This is allowed due to the correspondence between edges with an infinite number of initial tokens and non-existent edges. In addition, the dataflow graph contains a feedback loop, as the settings of the channel equalizer (EQ) for the reception of symbol i are based on an estimate of the channel (CHEST) during the reception of symbol $i-2$. This estimate of the channel is based on the received symbol $i-2$ and the reencoded symbol $i-2$, which is obtained by reencoding the error corrected bits of symbol $i-2$ produced by the viterbi channel decoder (VIT).

We assume all software tasks (all tasks except source and hardware filter) being mapped to three different processors, which is indicated by the different colors of the actors in the dataflow graph. If multiple tasks are mapped to a shared processor, then they are scheduled by a static priority preemptive scheduler, with their priorities denoted in the upper parts of the corresponding actors. For instance, the tasks τ_{FFT} and τ_{EQ} share a processor, with task τ_{EQ} having a higher priority than task τ_{FFT}.

In the following, we will apply the original analysis flow that calculates maximum response times without consideration of cyclic data dependencies and then compare its results to the results obtained by an application of our accuracy-improved analysis flow. Before applying the analysis flows we resolve the inequalities from the LPs in Algorithms 1 and 2 such that we get compact formulations of maximum start times and maximum jitters.

At first, we iteratively substitute the inequalities from Algorithm 2 for each cycle in Figure 5 to derive the temporal constraints that must be met for an admissible upper bound on the worst-case schedule. The tasks τ_{FILTER}, τ_{EQ}, τ_{CHEST} and τ_{REENC} either have the highest priorities on their shared processors or are executed on separate processors. Therefore they do not experience any interference from other tasks and the maximum firing durations of the corresponding actors are equal to their WCETs. This lets us derive the following constraint from the cycle between the actors v_{FILTER} and v_{FFT}:

$$\hat{s}_{\text{FFT}} - \hat{s}_{\text{FILTER}} \geq \hat{\rho}_{\text{FILTER}} \ \wedge\ \hat{s}_{\text{FILTER}} - \hat{s}_{\text{FFT}} \geq \hat{\rho}_{\text{FFT}} - 1 \cdot P_{\text{SRC}}$$
$$\Leftrightarrow\ \hat{\rho}_{\text{FILTER}} + \hat{\rho}_{\text{FFT}} \leq 1 \cdot P_{\text{SRC}} \ \Leftrightarrow\ \ \hat{\rho}_{\text{FFT}} \leq 1 \cdot 8 - 1.5 = 6.5$$

Analogously, we derive the temporal constraint from the rightmost cycle:

$$\hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} + \hat{\rho}_{\text{VIT}} \leq 10 \tag{12}$$

By making use of the maximum jitter equation from Section III-E we derive dependencies between maximum jitters and maximum response times. For instance, the maximum jitter of task τ_{EQ} is defined as $J_{\text{EQ}} = \hat{s}_{\text{EQ}} - \check{s}_{\text{EQ}}$. With Algorithm 1 it follows for the minimum start time of actor v_{EQ}:

$$\check{s}_{\text{EQ}} \geq \check{s}_{\text{FFT}} + \check{\rho}_{\text{FFT}} \geq \check{\rho}_{\text{FILTER}} + \check{\rho}_{\text{FFT}} = 4.5 \qquad (13)$$

Note that the edge $e_{\text{CHEST,EQ}}$ is not considered in Algorithm 1 as it contains initial tokens. From Algorithm 2 we obtain the following constraints on the maximum start time of actor v_{EQ}:

$$\begin{aligned}
\hat{s}_{\text{EQ}} &\geq \hat{s}_{\text{FFT}} + \hat{\rho}_{\text{FFT}} \geq \hat{\rho}_{\text{FILTER}} + \hat{\rho}_{\text{FFT}} = \hat{\rho}_{\text{FFT}} + 1.5 && (14)\\
\hat{s}_{\text{EQ}} &\geq \hat{s}_{\text{CHEST}} + \hat{\rho}_{\text{CHEST}} - 2 \cdot P_{\text{SRC}} && (15)\\
&\geq \hat{s}_{\text{FFT}} + \hat{\rho}_{\text{FFT}} + \hat{\rho}_{\text{CHEST}} - 2 \cdot P_{\text{SRC}} &&\\
&\geq 1.5 + \hat{\rho}_{\text{FFT}} + 1 - 2 \cdot 8 = \hat{\rho}_{\text{FFT}} - 13.5 &&\\
\hat{s}_{\text{EQ}} &\geq \hat{s}_{\text{CHEST}} + \hat{\rho}_{\text{CHEST}} - 2 \cdot P_{\text{SRC}} \geq \dots && (16)\\
&\geq \hat{s}_{\text{EQ}} + \hat{\rho}_{\text{EQ}} + \hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} &&\\
&\quad + \hat{\rho}_{\text{VIT}} + \hat{\rho}_{\text{REENC}} + \hat{\rho}_{\text{CHEST}} - 2 \cdot P_{\text{SRC}} &&\\
&= \hat{s}_{\text{EQ}} + \hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} + \hat{\rho}_{\text{VIT}} - 10 &&
\end{aligned}$$

Equation 14 imposes a tighter constraint on $\hat{s}_{\text{EQ}}$ than Equation 15 and Equation 16 equals to the temporal constraint in Equation 12, which must be true for an admissible schedule. Therefore $\check{s}_{\text{EQ}}$ is only constrained by Equation 13 and $\hat{s}_{\text{EQ}}$ only by Equation 14. The LPs in Algorithm 1 and Algorithm 2 both minimize start times and as the Equations 13 and 14 are the only constraints that have to be considered, we can replace the inequalities in these constraints by equalities. This leads to the following maximum jitter of actor v_{EQ}:

$$J_{\text{EQ}} = \hat{s}_{\text{EQ}} - \check{s}_{\text{EQ}} = \hat{\rho}_{\text{FFT}} + 1.5 - 4.5 = \hat{\rho}_{\text{FFT}} - 3$$

Analogously, we derive for the other maximum jitters:

$$\begin{aligned}
&J_{\text{FILTER}} = 0,\; J_{\text{FFT}} = 1,\; J_{\text{EQ}} = J_{\text{DEMAP}} = \hat{\rho}_{\text{FFT}} - 3\\
&J_{\text{DEINT}} = \hat{\rho}_{\text{FFT}} + \hat{\rho}_{\text{DEMAP}} - 4,\; J_{\text{VIT}} = \hat{\rho}_{\text{FFT}} + \hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} - 5\\
&J_{\text{REENC}} = J_{\text{CHEST}} = \hat{\rho}_{\text{FFT}} + \hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} + \hat{\rho}_{\text{VIT}} - 6
\end{aligned}$$

Taking these temporal constraints and maximum jitters into account we now apply the original analysis flow which does not consider that cyclic data dependencies limit interference. The analysis flow is applied by iteratively calculating maximum response times and maximum jitters until either all maximum jitters converge or constraints are violated. For the first iteration of the analysis flow we initialize all maximum jitters to zero. For task τ_{DEINT} it follows with Equations 1 to 3:

$$\begin{aligned}
w_{\text{DEINT}}(1) &= 1 \cdot C_{\text{DEINT}} + \left\lceil \frac{J_{\text{VIT}} + w_{\text{DEINT}}(1)}{P_{\text{SRC}}} \right\rceil \cdot C_{\text{VIT}}\\
&\quad + \left\lceil \frac{J_{\text{CHEST}} + w_{\text{DEINT}}(1)}{P_{\text{SRC}}} \right\rceil \cdot C_{\text{CHEST}}\\
&= 1 \cdot 1 + \left\lceil \frac{0+1}{8} \right\rceil \cdot 1 + \left\lceil \frac{0+1}{8} \right\rceil \cdot 1 = 3\\
w_{\text{DEINT}}(1) &= 1 \cdot 1 + \left\lceil \frac{0+3}{8} \right\rceil \cdot 1 + \left\lceil \frac{0+3}{8} \right\rceil \cdot 1 = 3
\end{aligned}$$

Hence we can conclude that the fixed point of the busy period $w_{\text{DEINT}}(1)$ is three. As $w_{\text{DEINT}}(1) \leq 1 \cdot P_{\text{SRC}}$ we do not have to consider $q > 1$, resulting in $\hat{R}_{\text{DEINT}} = 3$.

Similarly we calculate the maximum response times for all other actors, which can be found in the first column of Table I. Based on these maximum response times we derive the maximum jitters in the second column. It can be verified that no maximum response time of the first iteration violates any of the temporal constraints. Therefore we calculate the maximum response times and maximum jitters in the second iteration, considering the maximum jitters calculated in the first. However, the maximum response times in the second iteration, which are presented in the third column of Table I, lead to a violation of the temporal constraint in Equation 12:

$$\hat{\rho}_{\text{DEMAP}} + \hat{\rho}_{\text{DEINT}} + \hat{\rho}_{\text{VIT}} = 7 + 5 + 3 = 15 > 10$$

	Original Analysis Flow				Accuracy-Improved Analysis Flow			
	1st iteration		2nd iteration		1st iteration		2nd iteration	
x	$\hat{R}_x$	J_x	$\hat{R}_x$	J_x	$\hat{R}'_x$	J_x	$\hat{R}'_x$	J_x
v_{FILTER}	1.5	0	1.5	0	1.5	0	1.5	0
v_{FFT}	5	1	5	1	5	1	5	1
v_{EQ}	1	2	1	2	1	2	1	2
v_{DEMAP}	4	2	7	2	4	2	4	2
v_{DEINT}	3	5	5	8	3	5	3	5
v_{VIT}	2	7	3	12	2	7	2	7
v_{REENC}	4	8	4	14	4	8	4	8
v_{CHEST}	1	8	1	14	1	8	1	8

TABLE I. TEMPORAL ANALYSIS RESULTS FOR FIGURE 5.

Now we apply our accuracy-improved analysis flow which takes into account that cyclic data dependencies limit interference. Using the maximum response time equations from Equation 4 to 6 results in the maximum response times and maximum jitters presented in the fifth and sixth column of Table I. These are just the same as in the first iteration of the original analysis flow. However, the results are different in the second iteration. For instance, the maximum response time of task τ_{VIT} in the second iteration is calculated as follows:

$$\begin{aligned}
w_{\text{VIT}}(1) &= 1 + \left\lceil \frac{J_{\text{CHEST}} + w_{\text{VIT}}(1)}{P_{\text{SRC}}} \right\rceil \cdot C_{\text{CHEST}} = \dots = 3\\
w'_{\text{VIT}}(1) &= 1 + \max(\left\lceil \frac{J_{\text{CHEST}} + w_{\text{VIT}}(1)}{P_{\text{SRC}}} \right\rceil, \delta^{\circ}_{\substack{\text{VIT,}\\\text{CHEST}}} - 1) \cdot C_{\text{CHEST}}\\
&= 1 + \max(\left\lceil \frac{8+3}{8} \right\rceil, 1) \cdot 1 = 2
\end{aligned}$$

As $w'_{\text{VIT}}(1) \leq 1 \cdot P_{\text{SRC}}$ it follows that $\hat{R}'_{\text{VIT}} = 2$. The number of tokens on the cycle between the actors v_{VIT} and v_{CHEST} effectively limits interference, which results in the same maximum response time of task τ_{VIT} as in the first iteration. The same happens to all other maximum response times in our example as well, resulting in the maximum response times and maximum jitters of the second iteration being equal to those of the first. Hence we conclude a convergence of maximum jitters without a violation of temporal constraints after the second iteration of our accuracy-improved analysis flow.

Based on these results we can now also determine sufficient FIFO buffer capacities. This is done by inserting edges in the reverse direction wherever two actors are connected by a single edge. For those edges we can then calculate sufficient numbers of initial tokens using Equation 11. For this calculation we require the maximum start times of all actors which define the upper bound on the worst-case schedule. The maximum start times can be calculated by adding up maximum response times on the path from actor v_{SRC} to actor v_{CHEST} over the rightmost cycle. For instance, the sufficient number of initial tokens on the inserted edge from actor v_{CHEST} to actor v_{FFT} can be calculated as the smallest integer that satisfies:

$$\delta(e_{\text{CHEST,FFT}}) \geq \frac{\rho_{\text{CHEST}} + \hat{s}_{\text{CHEST}} - \hat{s}_{\text{FFT}}}{P_{\text{SRC}}} = \frac{1 + 20.5 - 1.5}{8} = 2.5$$

Therefore it can be concluded that the sufficient buffer capacity for a FIFO buffer between the tasks τ_{FFT} and τ_{CHEST} is three. The sufficient numbers of initial tokens for the other additional

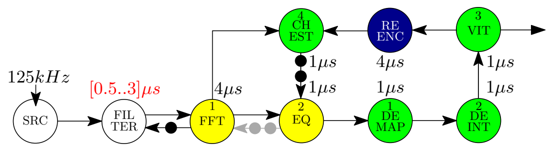

Fig. 6. Modification of the HSDF graph from Figure 5.

edges are all one, except for the edge $e_{\text{EQ,CHEST}}$, on which zero initial tokens suffice. Note that the FIFO buffer between the tasks τ_{CHEST} and τ_{EQ} must contain at least $0 + 2 = 2$ containers, as the number of initial tokens on the edge $e_{\text{CHEST,EQ}}$ is two, corresponding to two initially full containers.

Now consider that we replace the hardware filter by another implementation with a higher WCET. This modification is depicted in Figure 6. The larger maximum jitter coming from the filter does not increase the maximum response times of the actors on the rightmost cycle, as these are already limited by the number of tokens on that cycle. However, the maximum response time of task τ_{FFT} would increase to $\hat{R}_{\text{FFT}} = 6$. In addition, the temporal constraint imposed by the leftmost cycle would become $\hat{\rho}_{\text{FFT}} \leq 5$, due to the higher WCET of task τ_{FILTER}. Altogether, this would lead to a violation of the temporal constraint imposed by the cycle between the tasks τ_{FILTER} and τ_{FFT}.

However, setting the capacity of the FIFO buffer between the tasks τ_{FFT} and τ_{EQ} to two containers, as indicated by the additional edge $e_{\text{EQ,FFT}}$ with two initial tokens in Figure 6, leads to a different conclusion. This additional cyclic data dependency limits the interference of task τ_{EQ} on task τ_{FFT}, resulting in a maximum response time of $\hat{R}_{\text{FFT}} = 5$. This maximum response time would not violate the temporal constraint coming from the cycle between the tasks τ_{FILTER} and τ_{FFT}, which demonstrates that a reduction of buffer capacities to smaller than infinity can result in previously unsatisfiable temporal constraints becoming satisfiable.

V. Conclusion

In this paper a dataflow analysis approach is presented for single-rate real-time stream processing applications executed on multiprocessor systems with shared processors. The presented approach improves the accuracy compared to state-of-the-art by taking into account that cyclic data dependencies limit interference between tasks sharing a processor. The accuracy improvement results from the usage of enhanced maximum response time equations which take into account that the numbers of tokens on cyclic data dependencies limit interference. These maximum response time equations are embedded into an iterative analysis flow with an exponential time-complexity that is applicable for systems employing static priority preemptive schedulers and that can be used for the verification of temporal constraints, as well as a calculation of sufficient buffer capacities.

A WLAN 802.11p transceiver application containing cyclic data dependencies is used to illustrate that by applying our approach an accuracy improvement can be obtained for realistic applications, such that satisfaction of temporal constraints can be concluded where existing approaches would indicate a constraint violation. Furthermore, we demonstrate for the same application that a reduction of buffer capacities can lead to a reduction of jitters, making previously unsatisfiable temporal constraints satisfiable. We intend to develop a more systematic buffer sizing approach that exploits this counter-intuitive effect. Moreover, we plan to extend the applicability of our approach to a broader set of schedulers than static priority preemptive.

References

[1] O. Moreira and M. Bekooij, “Self-timed scheduling analysis for real-time applications,” *EURASIP Journal on Advances in Signal Processing*, no. 1, 2007.

[2] M. Wiggers, M. Bekooij, and G. Smit, “Monotonicity and run-time scheduling,” in *ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2009, pp. 177–186.

[3] ——, “Computation of buffer capacities for throughput constrained and data dependent inter-task communication,” in *Design, Automation and Test in Europe (DATE)*, 2008, pp. 640–645.

[4] M. Wiggers, M. Bekooij, M. Geilen, and T. Basten, “Simultaneous budget and buffer size computation for throughput-constrained task graphs,” in *Design, Automation and Test in Europe (DATE)*, 2010, pp. 1669–1672.

[5] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs,” in *Design Automation Conf. (DAC)*, 2007, pp. 777–782.

[6] J. Falk, J. Keinert, C. Haubelt, J. Teich, and S. Bhattacharyya, “A generalized static data flow clustering algorithm for MPSoC scheduling of multimedia applications,” in *ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2008, pp. 189–198.

[7] J. Hausmans, M. Bekooij, and H. Corporaal, “Resynchronization of cyclo-static dataflow graphs,” in *Design, Automation and Test in Europe (DATE)*, 2011, pp. 1–6.

[8] S. Geuns, J. Hausmans, and M. Bekooij, “Automatic dataflow model extraction from modal real-time stream processing applications,” in *Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 2013, pp. 143–152.

[9] B. Theelen *et al.*, “A scenario-aware data flow model for combined long-run average and worst-case performance analysis,” in *Int'l Conf. on Formal Methods and Models for Codesign (MEMOCODE)*, 2006, pp. 185–194.

[10] J. Hausmans, M. Wiggers, S. Geuns, and M. Bekooij, “Dataflow analysis for multiprocessor systems with non-starvation-free schedulers,” in *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2013, pp. 13–22.

[11] R. Henia *et al.*, “System level performance analysis – the SymTA/S approach,” *IEE Proc. of Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[12] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, “System architecture evaluation using modular performance analysis: A case study,” *Int'l Journal on Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 649–667, 2006.

[13] L. Thiele, S. Chakraborty, and M. Naedele, “Real-time calculus for scheduling hard real-time systems,” in *IEEE Int'l Symp. on Circuits and Systems (ISCAS)*, vol. 4, 2000, pp. 101–104.

[14] L. Thiele and N. Stoimenov, “Modular performance analysis of cyclic dataflow graphs,” in *ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2009, pp. 127–136.

[15] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi, “Cyclic dependencies in modular performance analysis,” in *ACM Int'l Conf. on Embedded Software (EMSOFT)*, 2008, pp. 179–188.

[16] K. Tindell, *Adding time-offsets to schedulability analysis*. University of York, Department of Computer Science, 1994.

[17] J. Palencia and M. Gonzalez Harbour, “Schedulability analysis for tasks with static and dynamic offsets,” in *IEEE Real-Time Systems Symp. (RTSS)*, 1998, pp. 26–37.

[18] T.-Y. Yen and W. Wolf, “Performance estimation for real-time distributed embedded systems,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1125–1136, 1998.

[19] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, “A novel analytical method for worst case response time estimation of distributed embedded systems,” in *Design Automation Conf. (DAC)*, 2013, pp. 129:1–129:10.

[20] K. Richter, R. Racu, and R. Ernst, “Scheduling analysis integration for heterogeneous multiprocessor SoC,” in *IEEE Real-Time Systems Symp. (RTSS)*, 2003, pp. 236–245.

[21] K. Tindell, A. Burns, and A. Wellings, “An extendible approach for analyzing fixed priority hard real-time tasks,” *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.

[22] R. Floyd, “Algorithm 97: Shortest path,” *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[23] P. Alexander, D. Haley, and A. Grant, “Outdoor mobile broadband access with 802.11,” *IEEE Communications Magazine*, vol. 45, no. 11, pp. 108–114, 2007.