Buffer Sizing to Reduce Interference and Increase Throughput of Real-Time Stream Processing Applications

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

***Abstract*—Existing temporal analysis and buffer sizing techniques for real-time stream processing applications ignore that FIFO buffers bound interference between tasks on the same processor. By considering this effect it can be shown that a reduction of buffer capacities can result in a higher throughput. However, the relation between buffer capacities and throughput is non-monotone in general, which makes an exploitation of the effect challenging.**

In this paper a buffer sizing approach is presented which exploits that FIFO buffers bound interference between tasks on shared processors. The approach combines temporal analysis using a cyclic dataflow model with computation of buffer capacities in an iterative manner and thereby enables higher throughput guarantees at smaller buffer capacities. It is shown that convergence of the proposed analysis flow is guaranteed.

The benefits of the presented approach are demonstrated using a WLAN 802.11p transceiver application executed on a multiprocessor system with shared processors. If buffers without blocking writes are used an up to 25% higher guaranteeable throughput and up to 23% smaller buffer capacities can be determined compared to existing approaches. For systems using buffers with blocking writes the guaranteeable throughput is even up to 43% higher and buffer capacities up to 11% smaller.

I. Introduction

Executing real-time stream processing applications such as Software Defined Radios (SDRs) on embedded multiprocessor systems usually imposes the necessity of giving throughput guarantees at design time, to ensure that temporal constraints can always be satisfied. It has been shown that dataflow analysis techniques are suitable for the verification of such temporal constraints, as they support the combination of cyclic stream processing applications, processor sharing and run-time schedulers [10], [11], [18].

Real-time stream processing applications regulary consist of multiple tasks that communicate via First-In-First-Out (FIFO) buffers. Such an application is exemplified in Figure 1. Synchronization between tasks using FIFO buffers ensures that a reading task is delayed until a writing task writes data to the buffer. Regarding write operations on FIFO buffers, it must be differed between buffers for which writing tasks are blocked or are not blocked when buffers are full.

If buffers with blocking writes are used then a writing task can be delayed in its execution when one of its output buffers is full, such that it has to wait until a reading task finishes its execution and thereby frees locations in the buffer. Due to this additional synchronization, buffers with blocking writes can compensate for variations in the enabling times of tasks, whereas buffers with non-blocking writes have to account for such differences with larger capacities. Buffers with blocking writes therefore require less memory than buffers with non-blocking writes to guarantee the same minimum throughput, provided that the additional synchronization overhead due to blocking on writes is neglectable.

Existing buffer sizing techniques in the scope of dataflow analysis assume that the relation between buffer capacities and minimum throughput is monotone [18]. This is illustrated by the dotted curve in Figure 1. However, it has been recently shown in [20] that interference between tasks, which occurs when a high priority (HP) task preempts and delays a low priority (LP) task, can be reduced by decreasing the capacities of FIFO buffers with blocking writes connecting these tasks. Due to this effect, the relation between buffer capacities and minimum throughput can become non-monotone. It can occur that smaller buffer capacities result in higher throughput guarantees, as illustrated by the solid curve in Figure 1.

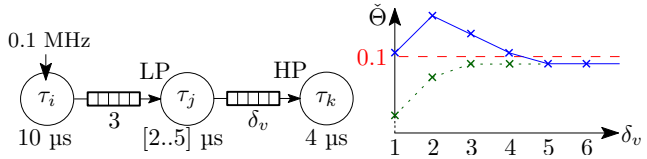

Fig. 1. Left: Didactic example of a task graph containing buffers with blocking writes and tasks scheduled by a static priority preemptive scheduler. Right: Relation between buffer capacity δ_v and minimum throughput $\check{\Theta}$.

However, buffer sizing is regularly performed after temporal analysis [4], [16], [20], which prevents an exploitation of the effect that smaller buffer capacities can lead to higher throughput guarantees. This issue can be addressed by an integrated temporal analysis and buffer sizing approach.

In this paper a buffer sizing approach is presented which combines temporal analysis and computation of buffer capacities in an iterative manner and thereby exploits that FIFO buffers bound interference between tasks. The approach is suitable for the analysis of cyclic stream processing applications executed on multiprocessor systems with processor sharing and static priority preemptive schedulers. It is shown that calculating buffer capacities iteratively instead of once after temporal analysis can result in reduced buffer capacities and higher throughput guarantees.

On the one hand, this is due to the fact that the capacities of both buffers with blocking and non-blocking writes represent bounds on interference between tasks, which can be smaller than bounds obtained solely based on jitter of tasks. Considering estimates on buffer capacities in the interference calculation can therefore lead to an accuracy improvement. And on the other hand, buffers with blocking writes can be sized in such way that early enabled tasks are delayed until another task finishes its execution, effectively reducing jitter and thereby also interference between tasks. In that case, not only analysis accuracy is improved, but the temporal behavior of the analyzed application is changed, such that interference between tasks is reduced. It is shown that convergence of the presented iterative analysis flow is guaranteed.

The remainder of this paper is structured as follows. In Section II related work is presented. Section III sketches the basic idea of bounding interference with FIFO buffers. Section IV presents the iterative analysis flow and the analysis model, derives the required equations for buffer sizing and interference and gives the proof of convergence. In Section V the benefits of the presented approach are evaluated in a case study and Section VI presents the conclusions.

 DOI 10.1109/ISORC.2015.14.

II. Related Work

Our approach is based on the temporal analysis framework introduced in [3]. Apart from that framework there exist two other major frameworks suitable for the analysis of data-driven real-time applications on multiprocessor systems: The SymTA/S approach [4] and Real-Time Calculus (RTC) [13].

The SymTA/S approach combines standard event models in the time-interval domain with response time analysis. It makes use of an iterative procedure of traffic characterization and response time calculation, which enables the analysis of applications with cyclic resource dependencies. However, it lacks support for (arbitrary) cyclic data dependencies, which is required to accurately capture FIFO buffers with limited capacities in the analysis model. It is therefore restricted to a post-analysis computation of sufficient buffer capacities. In [5] a backlog-based, post-analysis buffer sizing for buffers with non-blocking writes is presented.

The Modular Performance Analysis (MPA) approach [16] is based on RTC, allows for arbitrary event models and derives its traffic characterization in the time-interval domain. Cyclic data dependencies cannot be captured in the analysis model, which restricts the approach to buffer sizing techniques after temporal analysis. In [14] a generalization of the MPA framework is presented which supports modeling of cyclic data dependencies by deriving a traffic characterization in the time domain. Consequently, buffer capacity constraints can be modeled in this framework. However, both buffer sizing and a combination of cyclic data dependencies with cyclic resource dependencies are not discussed. Another approach based on RTC is presented in [6] and allows for a consideration of buffers with blocking writes in the analysis model. The approach lacks support for cyclic dependencies other than those imposed by finite buffers. Cyclic resource dependencies are not considered as well.

As neither the SymTA/S approach nor RTC consider combinations of cyclic data and resource dependencies, the effect that reducing buffer capacities can lead to higher throughput guarantees also cannot be exploited in these frameworks.

Buffer sizing of buffers with blocking writes has been studied extensively in the context of dataflow analysis, e.g. in [17], [12], [7]. In [9] it is shown how dataflow models can be used to compute buffer capacities of buffers with non-blocking writes as well. However, it is assumed that best-case and worst-case response times are given, which prevents an exploitation of the relation between buffer capacities, interference and response times. [19] presents a dataflow analysis approach for systems with starvation-free schedulers like round-robin, for which the minimum service of a task can be determined independently of the enabling rates of other tasks. This approach shows that dataflow analysis techniques can combine cyclic data and resource dependencies to derive the minimum throughput of real-time applications executed on multiprocessor systems.

In [3] a dataflow analysis framework is introduced that combines dataflow modeling with response time analysis. It derives an enabling rate characterization of tasks and thereby extends the scope of dataflow analysis techniques to systems with non-starvation-free schedulers like static priority preemptive. Moreover, the usage of an enabling rate characterization allows for an accuracy improvement for starvation-free schedulers as well. An extension of this approach is presented in [20]. In this work response time equations are introduced which take the effect that cyclic data dependencies bound interference into account. However, buffer sizing is performed after temporal analysis, which prevents an exploitation of the effect that FIFO buffers bound interference.

In contrast to all aforementioned works, our approach exploits that FIFO buffers bound interference. It makes use of the response time equations from [20] and modifies the analysis flow from [3] such that estimates on buffer capacities are computed iteratively during temporal analysis instead of once afterwards. The problematic, in general non-monotone relation between buffer capacities and interference is thereby taken into account. To the best of our knowledge, our temporal analysis and buffer sizing approach is consequently the only approach which exploits the effect that reducing buffer capacities can lead to higher throughput guarantees.

III. Basic Idea

This section illustrates the non-monotone relation between buffer sizing, interference and minimum throughput with an example. Using this example, it is shown that performing buffer sizing iteratively during temporal analysis instead of once afterwards can result in both smaller buffer capacities and higher throughput guarantees.

Consider the task graph depicted on the left side of Figure 1. The tasks τ_j and τ_k are executed on a shared processor using a static priority preemptive scheduler, with task τ_k having a higher priority than task τ_j. Executions of task τ_k can preempt and thereby delay executions of task τ_j due to the given priority assignment, which can make the maximum response time $\hat{R}_j$ of task τ_j larger than its Worst-Case Execution Time (WCET) of 5 µs. Task τ_i is enabled by a periodic source with a frequency of 0.1 MHz and the tasks communicate via FIFO buffers with blocking writes of the capacities 3 and δ_v, respectively.

In a dataflow model of the task graph the FIFO buffers with blocking writes correspond to cyclic data dependencies. According to the Maximum Cycle Mean (MCM) equation [10], both cyclic data dependencies limit the minimum throughput $\check{\Theta}$ as follows:

$$\check{\Theta} = \min\left(\frac{3}{10\ \mu\text{s} + \hat{R}_j}, \frac{\delta_v}{\hat{R}_j + 4\ \mu\text{s}}\right)$$

Classical response time analysis [15], [3] considers maximum response times to be independent of buffer capacities. Under this assumption, the minimum throughput $\check{\Theta}$ is monotonically increasing in the buffer capacity δ_v, until it is not limited by the rightmost buffer, but by the leftmost buffer with the fixed capacity of 3. This behavior can be explained by the fact that larger buffer capacities generally allow for more pipeline parallelism. For the given example the minimum throughput allowed by the cyclic data dependencies, which is indicated by the dotted curve in Figure 1, is not large enough for any δ_v to meet the throughput constraint imposed by the source, indicated by the horizontal, dashed line.

However, if the response time equations from [20] are applied, then the interference of task τ_k on task τ_j is bounded by the cyclic data dependency between the tasks, leading to smaller maximum response times $\hat{R}_j$ for smaller δ_v. This is due to the fact that the buffer between the tasks τ_j and τ_k blocks on both reads and writes. The buffer thus effectively bounds jitter between the tasks, and thereby also interference of task τ_k on task τ_j.

For instance, if the capacity of the buffer between the tasks τ_j and τ_k would be equal to $\delta_v = 1$, then both tasks would have to wait for the other to finish its execution before they can start theirs. Consequently, the executions of the tasks τ_j and τ_k would be mutually exclusive and the interference of task τ_k on task τ_j zero.

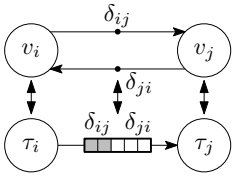

Fig. 2. One-to-one relation between dataflow model and task graph.

If the positive effect of smaller buffer capacities on response times exceeds the negative effect of smaller buffer capacities on pipeline parallelism, then an increased minimum throughput $\check{\Theta}$ can be observed for these buffer capacities. The resulting, in general non-monotone relation between buffer capacities and minimum throughput is illustrated by the solid curve in Figure 1. As it can be concluded from the graph, the required throughput guarantee indicated by the dashed line can be met for buffer capacities δ_v between 1 and 4.

Note that existing temporal analysis and buffer sizing approaches do not come to this conclusion. This is due to the fact that unknown buffer capacities are not considered during temporal analysis. Instead, buffer sizing is performed only once after temporal analysis finishes. Not considering buffer capacities during temporal analysis is equivalent to assuming unknown buffer capacities as unbounded, i.e. $\delta_v = \infty$. As it can be seen from the graph in Figure 1, the minimum throughput $\check{\Theta}$ is constant for any $\delta_v \geq 5$, a buffer capacity of $\delta_v = \infty$ would therefore result in a throughput guarantee below the required throughput. Existing temporal analysis approaches would hence conclude that the required throughput is infeasible for the given example. This shortcoming motivates the introduction of an iterative, integrated temporal analysis and buffer sizing approach, as it is presented in the remainder of this paper.

IV. Temporal Analysis and Buffer Sizing

Our temporal analysis and buffer sizing approach is presented in this section. Section IV-A describes the analysis model and Section IV-B the iterative analysis flow used for buffer sizing. Section IV-C presents equations for the calculation of upper bounds on the response times of tasks, which consider the effect that cyclic data dependencies bound interference. The employed relation between cyclic data dependencies and interference is established in Section IV-D. Section IV-E presents algorithms that are used to derive upper and lower bounds on the enabling times of tasks, as well as upper bounds on enabling jitters. Section IV-F describes our technique to determine suitable buffer capacities for a satisfaction of throughput constraints. Finally, Section IV-G derives a criterion for which our iterative analysis and buffer sizing flow converges.

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 enabling times maximum (minimum) enabling times and upper bounds on enabling jitters maximum enabling jitters.

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 enabling jitters of tasks 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. An actor $v_i \in V$ communicates with other actors by producing tokens on and consuming tokens from 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. An actor consumes one token from all its incoming edges at the start of a firing and produces one token on each of its outgoing edges when it finishes.

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 $\tau_i \in \mathcal{T}$ representing tasks and its directed edges representing FIFO buffers. Each task graph is single-rate and has a single strictly periodic source τ_s activating all other tasks of the task graph. Without loss of generality we require that no task is enabled before the first execution of the source τ_s and assume that all times of a task graph are relative to the first execution of the source. 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 executions. This behavior can be guaranteed by a scheduler that performs the required acquire operations when the execution of a task is started and the corresponding release operations when it is finished.

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. 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. These start times are used to compute bounds on the enabling times of the corresponding tasks. The minimum start times form a lower bound on the enabling times, whereas the maximum start times determine an upper bound on the enabling times of tasks. Note that the minimum and maximum start times thus form bounds on the best-case and worst-case schedules of tasks. A schedule is called admissible if no actor must fire before it is enabled according to the schedule. This is equivalent to a schedule that does not violate any constraints imposed on the schedule.

B. Iterative Analysis Flow for Buffer Sizing

Figure 3 depicts the flow of our temporal analysis and buffer sizing approach. We use this analysis flow for the verification of throughput constraints and for the computation of buffer capacities that allow for a satisfaction of these constraints. The main difference compared to the analysis flow

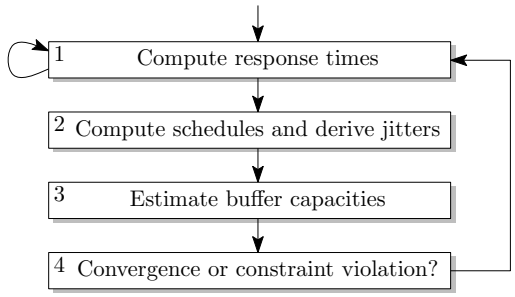

Fig. 3. Overview of the analysis flow.

from [3], on which this analysis flow is based on, as well as to other temporal analysis and buffer sizing techniques, is that estimated buffer capacities are computed iteratively during temporal analysis instead of once afterwards.

Input to our analysis flow are 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. Moreover, upper bounds on unknown buffer capacities can be specified, which is required to guarantee convergence of the flow. Based on these inputs, minimum and maximum response times of tasks are derived in step 1. This step uses the maximum response time equations from [20], which take into account that cyclic data dependencies bound interference.

In step 2, the transition from task graph to dataflow model is performed. Using the correspondence depicted in Figure 2 and the response times computed in step 1, two HSDF graphs are derived, the one a best-case and the other a worst-case model of the task graph. Given these HSDF graphs, two periodic schedules are computed, the first forming a lower bound and the second forming an upper bound on the enabling times of tasks. Based on these schedules, the maximum enabling jitters of tasks are derived.

Estimates on buffer capacities are computed in step 3 of the flow. The capacities are computed based on the schedules derived in step 3 and used in the response time calculations of subsequent iterations. In step 4, the two schedules are checked against temporal constraints, estimates on buffer capacities are compared with their bounds and it is verified whether all maximum enabling jitters and buffer capacities 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 maximum enabling jitters and buffer capacities have converged, the algorithm either finishes, or repeats the steps 1 to 4 until either convergence is achieved or constraints are violated.

C. Maximum Response Times of Tasks

In this section it is shown that the effect of cyclic data dependencies bounding interference between tasks can be included into equations for the calculation of maximum response times of tasks. The maximum response time $\hat{R}_i$ of a task τ_i denotes an upper bound on the time between an external enabling and finish of a task execution. External enablings of a task τ_i thereby describe enablings due to the arrival of containers produced by other tasks than task τ_i itself.

If a task τ_i is executed on a shared processor then it is not sufficient to consider the WCET of the task to calculate $\hat{R}_i$, but also executions of other tasks must be taken into account. For non-starvation-free schedulers like static priority preemptive, it has been shown that the response time of a task τ_i can be only bounded from above if the enabling rate characterization of other tasks on the same processor is taken into account [19].

Let P_j be the period of a task τ_j and J_j its maximum enabling jitter. The enabling rate characterization $\eta_j(\Delta t)$ of a task τ_j, which is an upper bound on the maximum number of enablings a task τ_j can have during a time interval Δt, can then be determined as follows [8]:

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

Using this enabling rate characterization, the busy period $w_i(q)$ of a task τ_i, which is an upper bound on the maximum amount of time between the first enabling and last finish of q consecutive executions of a task τ_i, can be determined by the following equation [15]:

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

C_i denotes the WCET of one execution of a task τ_i and the set $hp(i)$ contains all tasks τ_j with a higher priority than task τ_i. Provided that external enablings of a task τ_i are periodic with a period P_i, its maximum response time $\hat{R}_i$ can be derived as follows:

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

According to [15] only values of q for which $w_i(q) \geq q \cdot P_i$ holds need to be considered. Note that we will compute upper bounds on the external enabling times of tasks in Section IV-E. These bounds are periodic and therefore can be used in conjunction with maximum response times calculated with Equation 3.

In order to include the effect that cyclic data dependencies bound interference in the maximum response time equations, the enabling rate characterization $\eta_j(\Delta t)$ of a task τ_j can be replaced by a more accurate characterization $\eta'_{j \to i}(\Delta t, q)$. This characterization denotes the maximum number of enablings a task τ_j can have during a time interval Δt and q consecutive executions of a task τ_i [20]:

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

The first term of the minimum function denotes the interference of a task τ_j on a task τ_i, given that their enablings are independent of each other. It 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 presented in the subsequent section, describes the maximum number of enablings of a task τ_j during q consecutive iterations of a task τ_i due to cyclic data dependencies between them. Using this new upper bound on the maximum number of enablings of a task τ_j to reduce the busy period calculated with Equation 2 results in the following maximum response time equations:

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

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

In [20] it is proven that the maximum response times calculated with Equation 6 are temporally conservative and more accurate than the maximum response times calculated with Equation 3.

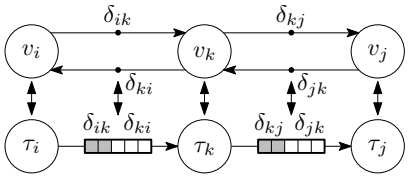

Fig. 4. Example of three tasks connected via FIFO buffers.

D. Limiting Interference with Cyclic Data Dependencies

In this section an intuitive derivation of the function $\gamma_{j\to i}(q)$ is presented, which describes interference of a task τ_j on q consecutive iterations of a task τ_i due to cyclic data dependencies. For a formal derivation of the function $\gamma_{j\to i}(q)$ please refer to [20].

We focus on the most simple case first, which is depicted in Figure 2. This task graph contains a producing task τ_i that is connected to a consuming task τ_j via a FIFO buffer of a capacity $\delta_{ij} + \delta_{ji}$. We further assume that both tasks execute on the same processor and that task τ_j has a higher priority than task τ_i. Therefore we derive a bound on interference of task τ_j on executions of task τ_i. We do this derivation using the corresponding HSDF graph.

According to the semantics of HSDF graphs, actor v_i is allowed to fire once if there is at least one token on its incoming edge, corresponding to one free location in the buffer. As one token is consumed by actor v_i at the beginning of a firing and not produced until the firing finishes, at most $\delta_{ij} + \delta_{ji} - 1$ tokens can be on the incoming edge of actor v_j during a firing of actor v_i. These tokens in turn allow for maximally $\delta_{ij} + \delta_{ji} - 1$ enablings of actor v_j during one firing of actor v_i.

Figure 4 depicts a slightly more complex example. As in the first example, we assume that the tasks τ_i and τ_j are executed on the same processor, with task τ_j having a higher priority than task τ_i, and in addition that task τ_k is executed on a different processor than the tasks τ_i and τ_j.

Actor v_j can be enabled maximally $\delta_{kj} + \delta_{jk}$ times without any firing of actor v_k, corresponding to the buffer between the tasks τ_k and τ_j being full. In addition, each firing of actor v_k can lead to an additional enabling of actor v_j. As actor v_k can fire maximally $\delta_{ik} + \delta_{ki} - 1$ times during one firing of actor v_i we derive that actor v_j can be enabled an additional $\delta_{ik} + \delta_{ki} - 1$ times as well. This results in maximally $\delta_{ik} + \delta_{ki} + \delta_{kj} + \delta_{jk} - 1$ enablings of actor v_j during one firing of actor v_i.

The observations obtained from the two examples can be generalized as follows. We define $\mathcal{P}_{ij}$ as the set of all directed paths from an actor v_i to actor v_j and $\delta(\mathcal{P}_{ij})$ as the minimum number of tokens on any path in $\mathcal{P}_{ij}$, with $\delta(\mathcal{P}_{ij}) = \infty$ if $\mathcal{P}_{ij} = \emptyset$. According to [20], the function $\gamma_{j\to i}(q)$ then equals to the following:

$$\gamma_{j\to i}(q) = \delta(\mathcal{P}_{ij}) + \delta(\mathcal{P}_{ji}) + q - 2 = \delta^{\circ}_{ij} + q - 2 \quad (7)$$

Note that the sum $\delta^{\circ}_{ij} = \delta(\mathcal{P}_{ij}) + \delta(\mathcal{P}_{ji})$ is just the minimum number of tokens on any directed cycle between two actors v_i and v_j. Such a cycle does not necessarily have to be a simple cycle, i.e. a cycle on which each actor is only traversed once, but can be any cyclic data dependency between the two actors. Moreover, it can be shown that adding an edge with an infinite number of initial tokens to an HSDF graph does not affect the start times of any actors. This relation is used in the definition of $\delta(\mathcal{P}_{ij})$ for $\mathcal{P}_{ij} = \emptyset$ to allow for a temporally conservative consideration of actors without cyclic data dependencies between them as well.

E. Enabling Times and Maximum Enabling Jitters

A method to derive periodic bounds on the minimum and maximum enabling times of tasks is presented in this section. Given these bounds, we can compute maximum enabling jitters of tasks, as well as buffer capacities for buffers with blocking and with non-blocking writes that allow for these enabling times. To derive these bounds we use the two different dataflow models which reflect the best-case and worst-case behavior of an application. From these dataflow models we then derive two periodic schedules consisting of the start times of actors. Afterwards, we show that these start times can be used as bounds on the enabling times of the corresponding tasks.

According to Figure 2, we model each task as a dataflow actor. We set the firing durations of actors in the best-case dataflow model to the minimum response times of the corresponding tasks and the firing durations of actors in the worst-case model to the maximum response times. The minimum response times of tasks are equal to their Best-Case Execution Times (BCETs), whereas the maximum response times are determined as presented in Section IV-C.

In the best-case model we have to consider that cyclic data dependencies with limited numbers of tokens can delay periodic start times of actors, while it can occur that the corresponding tasks do not experience the same delays. Consequently, we assume both fixed and unknown buffer capacities to be infinite in the best-case model, which is equivalent to removing all edges containing tokens.

In [3] it has been shown that the Linear Program (LP) in Algorithm 1 can be used to derive a periodic schedule consisting of the start times of actors in the best-case model. By setting the start time of the source actor v_s to $\check{s}_s = 0$, all start times are computed relative to the start time of the source. As the enabling times of tasks are also relative to the first execution of the strictly periodic source τ_s, we can use the minimum start time $\check{s}_i$ of an actor v_i to derive a lower bound $\check{\varepsilon}_i(k)$ on the enabling time $\varepsilon_i(k)$ of the corresponding task τ_i in iteration k as follows:

$$\forall_{k\geq 0}\colon\ \check{\varepsilon}_i(k) = \check{s}_i + k \cdot P_i \leq \varepsilon_i(k)$$

In the worst-case model, we consider all buffer capacities that are fixed before analysis as cyclic data dependencies, as shown in Figure 2. Buffer capacities that are determined with our analysis flow are set to the upper bounds specified as input to the flow. We do this assignment as increasing buffer capacities can lead to both larger maximum response times and earlier start times. A consideration of estimates on buffer capacities in the worst-case model would consequently break convergence of the analysis flow. Moreover, it is allowed that maximum response times of tasks can be larger than the source period, as such tasks can still execute at the rate of the source. That follows from Equation 6 with $w'_i(q)$ the maximum time required for q consecutive executions of a task τ_i:

$$\forall_{q\geq 1}\colon\ w'_i(q) \leq \hat{R}'_i + (q-1) \cdot P_i$$

Usually, self-edges with one token are used in dataflow models to capture that a task cannot be enabled before its previous execution is finished. However, setting firing durations of actors to maximum response times larger than periods would violate the temporal constraints imposed by such self-edges. Therefore, we have to omit self-edges in the worst-case model. Due to this omission it holds that the start times of actors derived in the worst-case model are not bounds on the enabling times of tasks, but only on the external enabling times, i.e. the times at which a task is enabled due to the arrival of containers coming from other tasks than itself.

Algorithm 1

$$\begin{aligned}\text{Minimize } & \sum_{v_i \in V} \check{s}_i \\ \text{Subject to: } & \check{s}_s = 0 \\ & \forall_{e_{ij} \in E'} \colon \ \check{s}_j - \check{s}_i \geq \check{\rho}_i \\ & \text{with } E' = \{e \,|\, e \in E \ \wedge \ \delta(e) = 0\}\end{aligned}$$

Algorithm 2

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

A periodic schedule consisting of the start times of actors in the worst-case model can be computed by solving the LP presented in Algorithm 2, with the start time of the source actor v_s set to $\hat{s}_s = 0$. We use the maximum start time $\hat{s}_i$ of an actor v_i to determine an upper bound $\hat{\varepsilon}_i^{ext}(k)$ on the external enabling time $\varepsilon_i^{ext}(k)$ of a task τ_i in iteration k:

$$\forall_{k \geq 0} \colon \ \varepsilon_i^{ext}(k) \leq \hat{\varepsilon}_i^{ext}(k) = \hat{s}_i + k \cdot P_i$$

Remember that the maximum response time of a task τ_i is defined as an upper bound on the time between the external enabling and the finish of a task execution, provided that the external enablings are periodic with a period P_i. As both minimum and maximum external enabling times of a task τ_i are periodic with P_i, we can bound the finish times $f_i(k)$ of a task τ_i with $\check{f}_i(k)$ and $\hat{f}_i(k)$, respectively, as follows:

$$\forall_{k \geq 0} \colon \ \check{f}_i(k) = \check{s}_i + k \cdot P_i + \check{R}_i \leq f_i(k) \leq \hat{f}_i(k) = \hat{s}_i + k \cdot P_i + \hat{R}_i$$

In [3], [20] it is assumed that a task τ_i is always enabled externally. However, as tasks can have maximum response times that are larger than their periods, this assumption does not hold in general and can result in an underapproximation of enabling jitters. A correction for the calculation of bounds on enabling times is presented in [2], where it is made explicit that the enabling time $\varepsilon_i(k)$ of a task τ_i in iteration k depends on both the external enabling time of the task in iteration k and on the time at which the previous execution of the task in iteration $k-1$ is finished. We define $f_i(-1) = -\infty$, thus it also holds that $f_i(-1) \leq \hat{f}_i(-1)$. An upper bound $\hat{\varepsilon}_i(k)$ on the enabling time $\varepsilon_i(k)$ of a task τ_i can then be determined as follows:

$$\begin{aligned}\forall_{k \geq 0} \colon \ \varepsilon_i(k) &= \max(\varepsilon_i^{ext}(k), f_i(k-1)) \\ &\leq \max(\hat{\varepsilon}_i^{ext}(k), \hat{f}_i(k-1)) \\ &= \max(\hat{s}_i + k \cdot P_i, \hat{s}_i + (k-1) \cdot P_i + \hat{R}_i) \\ &= \hat{s}_i + k \cdot P_i + \max(0, \hat{R}_i - P_i) = \hat{\varepsilon}_i(k)\end{aligned}$$

Given that both best-case and worst-case schedules are admissible, the maximum enabling jitter J_i of a task τ_i can be derived from the difference between minimum and maximum enabling times:

$$J_i = \max_{k \geq 0}(\hat{\varepsilon}_i(k) - \check{\varepsilon}_i(k)) = \hat{s}_i + \max(0, \hat{R}_i - P_i) - \check{s}_i \quad (8)$$

F. Buffer Sizing

In this section equations for our iterative buffer sizing flow are derived that can compute buffer capacities for both buffers with blocking and non-blocking writes.

Consider the relation between a FIFO buffer and the corresponding cycle in the dataflow graph depicted in Figure 2. The capacity of the modeled buffer is $\delta_{ij} + \delta_{ji}$. We assume that the number of initially full containers δ_{ij} is given, as this number is usually determined by the functional behavior of the analyzed application. Using the best-case and worst-case schedules determined with Algorithms 1 and 2, respectively, we can determine sufficiently large numbers of empty containers δ_{ji} for buffers with blocking and non-blocking writes.

1) Buffers with non-blocking writes:

To prevent an overflow of a buffer with non-blocking writes, that can lead to a functionally incorrect behavior of the application, it must be ensured that there is at least one empty container in the buffer whenever task τ_i is enabled. If the buffer initially holds δ_{ji} empty containers, then task τ_i can execute δ_{ji} times before another empty container must be freed by a finished execution of task τ_j. To prevent a buffer overflow it therefore must hold that an iteration k of task τ_j must be finished before iteration $k + \delta_{ji}$ of task τ_i is enabled, i.e.:

$$\forall_{k \geq 0} \colon \ f_j(k) \leq \varepsilon_i(k + \delta_{ji})$$

Using the periodic bounds on enabling and finish times presented in Section IV-E, it can be seen that:

$$\forall_{k \geq 0} \colon \ \hat{f}_j(k) \leq \check{\varepsilon}_i(k + \delta_{ji}) \Rightarrow f_j(k) \leq \varepsilon_i(k + \delta_{ji})$$

With substitution of the periodic bounds and $P_i = P_j$, which always holds for tasks of the same task graph, it follows:

$$\begin{aligned}& \forall_{k \geq 0} \colon \ \hat{s}_j + k \cdot P_j + \hat{R}_j \leq \check{s}_i + (k + \delta_{ji}) \cdot P_j \qquad (9) \\ \Leftrightarrow \quad & \delta_{ji} \geq \tfrac{\hat{s}_j + \hat{R}_j - \check{s}_i}{P_j}\end{aligned}$$

Hence it can be concluded that a buffer overflow cannot occur if the number of initially free containers δ_{ji} is large enough to satisfy the constraint in Equation 9.

2) Buffers with blocking writes:

In contrast to buffers with non-blocking writes it holds for buffers with blocking writes that a buffer overflow can never occur. Due to the blocking on writes, an early enabling of task τ_i is delayed until task τ_j finishes its execution and thereby frees a location in the buffer. Given best-case and worst-case schedules that represent bounds on the enabling times of tasks, the buffer capacities therefore do not have to be large enough to account for any difference between best-case and worst-case. Instead, it only has to be guaranteed that both bounds remain valid, to ensure that the calculated maximum enabling jitters remain temporally conservative.

In the computation of the best-case schedule with Algorithm 1, unknown buffer capacities are assumed to be infinite. From the monotonicity of dataflow graphs [18] it follows that reducing a number of tokens cannot lead to earlier enablings of actors. Therefore any finite number of initially empty containers δ_{ji} cannot lead to earlier start times in the best-case model, the best-case schedule remains a valid lower bound on the enabling times for any δ_{ji}.

With respect to the bound on the worst-case schedule, it has to be ensured that the number of initially empty containers δ_{ji} is large enough such that the start times computed with Algorithm 2 cannot become larger. Considering the number

of empty containers δ_{ji} in Algorithm 2 would result in an additional constraint on the maximum start time of actor v_i:

$$\hat{s}_i - \hat{s}_j \geq \hat{\rho}_j - \delta_{ji} \cdot P_j$$

It can be seen that if this constraint is not violated for the start times computed by Algorithm 2, then the periodic worst-case schedule remains admissible and the upper bounds on the enabling times of tasks remain valid. Substituting $\hat{\rho}_j$ with $\hat{R}_j$ and resolving the constraint to δ_{ji} results in the following constraint on the number of sufficient empty containers:

$$\delta_{ji} \geq \frac{\hat{s}_j + \hat{R}_j - \hat{s}_i}{P_j} \tag{10}$$

From this follows that the bounds on enabling times remain valid if the number of initially free containers δ_{ji} is large enough to satisfy the constraint in Equation 10.

3) Buffer sizing in the iterative analysis flow:

We denote a number of initially empty containers estimated in iteration n of the analysis flow as δ_{ji}^n. In our analysis flow, numbers of empty containers are initialized as follows:

$$\delta_{ji}^0 = \begin{cases} 1 & \delta_{ij} = 0 \\ 0 & \text{otherwise} \end{cases}$$

The reason for this initial assignment is that any smaller initial values would create cycles with zero tokens in the corresponding dataflow model, which would cause deadlock. Using Equations 9 and 10, the numbers of initially empty containers are estimated in step 3 of the analysis flow as follows:

$$\delta_{ji}^n = \begin{cases} \max(\left\lceil \frac{\hat{s}_j + \hat{R}_j - \check{s}_i}{P_j} \right\rceil, 0) & \text{buffer with non-blocking writes} \\ \max(\left\lceil \frac{\hat{s}_j + \hat{R}_j - \hat{s}_i}{P_j} \right\rceil, \delta_{ji}^{n-1}) & \text{buffer with blocking writes} \end{cases} \tag{11}$$

The numbers of initially empty containers are thus set to the smallest non-negative integers that satisfy the constraints in Equations 9 and 10. For buffers with blocking writes, a clamping of initially empty containers by the initially empty containers of the previous iteration of the analysis flow is required. Otherwise, convergence of the flow would not be guaranteed, as will be shown in Section IV-G.

4) Effects of FIFO buffers on interference between tasks:

This section compares the capacities of buffers with blocking and non-blocking writes obtained with Equation 11 and discusses the resulting effects on interference between tasks.

Due to $\forall_{v_i \in V} : \check{s}_i \leq \hat{s}_i$ it holds that capacities of buffers with blocking writes are always smaller or equal to capacities of buffers with non-blocking writes for the same bounds on schedules. The reason for this is that buffers with non-blocking writes cannot compensate for differences between best-case and worst-case schedules, while buffers with blocking writes can delay tasks such that the differences between enablings become smaller. Buffers with blocking writes thus effectively reduce the maximum enabling jitter between tasks. Consequently, using buffers with blocking writes instead of buffers with non-blocking writes does not only result in smaller buffer capacities, but also less interference and thereby a higher minimum throughput, provided that the additional synchronization overhead is neglectable.

Buffers with non-blocking writes do not change the temporal behavior of tasks, the jitter reduction observed for buffers with blocking writes consequently cannot occur as well. However, the iteratively calculated estimates on buffer capacities still represent bounds on the maximum enabling jitters between tasks, which can be more accurate than the absolute enabling jitters obtained with Equation 8. A consideration of these estimates in the maximum response time calculations can therefore result in tighter bounds on interference, which can lead to both smaller buffer capacities and a higher minimum throughput.

G. Convergence of the Analysis Flow

As all parts of the analysis flow are temporally conservative, it is ensured that the obtained results are also temporally conservative on convergence of the flow. However, it still remains to be shown that the analysis flow converges. To prove convergence, we first show that buffer capacities and maximum enabling jitters do not decrease in successive iterations of our analysis flow. Then we prove that the flow converges if all tasks of an analyzed task graph communicate exclusively via FIFO buffers and if upper bounds on the capacities are specified. These propositions are captured by the following two lemmas:

Lemma 1: Let J_i^n be the maximum enabling jitter of a task τ_i that is calculated in iteration n of the analysis flow. Analogously, we define δ_{ij}^n as the number of initial tokens on an edge e_{ij} calculated in iteration n. The initial maximum enabling jitters are denoted as J_i^0, the initial numbers of initial tokens as δ_{ij}^0. Then it holds that:

$$\forall_{\tau_i \in \mathcal{T}, e_{ij} \in E, n \geq 0} \colon \; J_i^n \leq J_i^{n+1} \wedge \delta_{ij}^n \leq \delta_{ij}^{n+1}$$

Proof: For space reasons we only present the sketch of a proof, which is based on mathematical induction.

In the induction base we verify that maximum enabling jitters and numbers of initial tokens cannot become smaller than their initial values. Maximum enabling jitters are initialized to zero. It can be seen that maximum enabling times can never become smaller than minimum enabling times, therefore our analysis flow also cannot compute enabling jitters smaller than zero. Numbers of initial tokens are initially set to the minimum values for which no deadlock can occur. The numbers of initial tokens computed with Equation 11 are also at least large enough to ensure that deadlock cannot occur. This is due to the fact that both blocking and non-blocking buffers are sized in such way that the best-case and worst-case schedules remain admissible, which would not be the case if a buffer would cause deadlock. Hence it can be concluded that:

$$\forall_{\tau_i \in \mathcal{T}, e_{ij} \in E} \colon \; J_i^0 \leq J_i^1 \;\wedge\; \delta_{ij}^0 \leq \delta_{ij}^1$$

For the induction step we assume as induction hypothesis that both maximum jitters and numbers of initial tokens computed in iteration n of the analysis flow are larger or equal than in the previous iteration, i.e.:

$$\forall_{\tau_i \in \mathcal{T}, e_{ij} \in E} \colon \; J_i^{n-1} \leq J_i^n \;\wedge\; \delta_{ij}^{n-1} \leq \delta_{ij}^n$$

The maximum response times computed in iteration n of the analysis flow are parameterized in the maximum jitters and numbers of initial tokens computed in iteration $n-1$. Equation 6, which is used for the maximum response time calculation, is monotonically increasing in both maximum jitters and numbers of initial tokens. Hence it follows:

$$\begin{aligned} &\forall_{\tau_i \in \mathcal{T}, e_{ij} \in E} \colon && J_i^{n-1} \leq J_i^n \;\wedge\; \delta_{ij}^{n-1} \leq \delta_{ij}^n \\ \Rightarrow\; &\forall_{\tau_i \in \mathcal{T}} \colon && \hat{R}_i'^n \leq \hat{R}_i'^{n+1} \end{aligned}$$

The minimum start times of actors calculated with Algorithm 1 in step 2 of the flow are constant throughout different iterations

of the flow. In contrast, in the computation of maximum start times with Algorithm 2 the firing durations of actors are set to the maximum response times calculated in step 1. Moreover, unknown numbers of initial tokens are assumed to be infinite in the algorithm, making its results independent of these numbers of initial tokens. From the monotonicity of dataflow graphs [19] it follows that increasing firing durations cannot lead to decreasing start times. Hence it holds that minimum start times are constant throughout different iterations of the flow, whereas maximum start times are monotonically increasing in increasing maximum response times:

$$\forall_{\tau_i \in \mathcal{T}}\colon\ \hat{R}_i'^{n} \leq \hat{R}_i'^{n+1} \Rightarrow \forall_{v_i \in V}\colon\ \hat{s}_i^n \leq \hat{s}_i^{n+1}$$

The maximum enabling jitters computed with Equation 8 are monotonically increasing in maximum start times and maximum response times. As we have shown that both maximum start times and maximum response times are larger or equal to the start times and response times of the previous iteration, we can conclude that also maximum enabling jitters are larger or equal than in the previous iteration:

$$\begin{aligned} &\forall_{\tau_i \in \mathcal{T}, v_j \in V}\colon && \hat{R}_i'^{n} \leq \hat{R}_i'^{n+1} \ \wedge\ \hat{s}_j^n \leq \hat{s}_j^{n+1} \\ &\Rightarrow \forall_{\tau_i \in \mathcal{T}}\colon && J_i^n \leq J_i^{n+1} \end{aligned} \tag{12}$$

For buffers with non-blocking writes it holds that the numbers of initial tokens computed with Equation 11 are monotonically increasing in increasing maximum response times and maximum start times. Due to the fact that minimum start times are constant throughout different iterations of the flow, we can conclude that the numbers of initial tokens for buffers with non-blocking writes are larger or equal than in the previous iteration. However, Equation 10, that is used to compute numbers of initial tokens for buffers with blocking writes, contains a maximum start time $\hat{s}_j$ with a negative sign, which can lead to decreasing buffer capacities in increasing maximum start times. Therefore it has to be enforced that the estimated numbers of initial tokens are always larger or equal than the numbers from the previous iteration, which is ensured by the clamping as shown in Equation 11. From this follows:

$$\begin{aligned} &\forall_{\tau_i \in \mathcal{T}, v_j \in V}\colon && \hat{R}_i'^{n} \leq \hat{R}_i'^{n+1} \ \wedge\ \hat{s}_j^n \leq \hat{s}_j^{n+1} \\ &\Rightarrow \forall_{e_{ij} \in E}\colon && \delta_{ij}^n \leq \delta_{ij}^{n+1} \end{aligned} \tag{13}$$

Equation 12 and Equation 13 conclude the induction step, thus it is proven that Lemma 1 holds. ■

Lemma 2: If the presented analysis flow is applied on a task graphs whose tasks all communicate via FIFO buffers and if for each of these FIFO buffers an upper bound is specified, then the analysis flow converges.

Proof: Let $\hat{\delta}_{ij}$ be an upper bound on the number of initial tokens $\delta(e_{ij})$ of an edge e_{ij}. From the correspondence in Figure 2 it follows that the requirement of bounded FIFO buffers between tasks is fulfilled if it holds that:

$$\forall_{e_{ij} \in E}\colon\ \exists_{e_{ji} \in E} \text{ with } \hat{\delta}_{ij} + \hat{\delta}_{ji} < \infty$$

From Lemma 1 it follows that both maximum enabling jitters and numbers of initial tokens are monotonically increasing. Moreover, if no constraints are violated, then the analysis flow does not converge as long as at least one maximum enabling jitter or number of initial tokens increases throughout two iterations of the flow. This in turn can only occur if at least one maximum response time increases throughout two iterations.

The results of $\eta'_{j \to i}(q, \Delta t)$ from Equation 4 are integer. Therefore it holds that maximum response times increase in minimum steps defined by the WCETs of higher priority tasks. Furthermore, it holds according to Equation 11 that if all communication between tasks of an analyzed task graph is realized with FIFO buffers, then a maximum response time cannot increase indefinitely without eventually increasing one of the numbers of initial tokens as well.

Finally, as in each iteration at least one maximum response time increases with a minimum step, eventually also a number of initial tokens increases. And as each number of initial tokens is bounded from above with $\delta(e_{ij}) \leq \hat{\delta}_{ij}$, it can be concluded that the analysis flow converges. ■

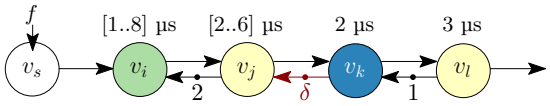

Fig. 5. HSDF graph of a synthetic application. Priorities of the corresponding tasks: Case 1: Task τ_j LP, task τ_l HP, Case 2: Task τ_j HP, task τ_l LP.

V. Case Study

This section demonstrates the benefits of our approach using two examples. The first example is synthetic, although it could well be part of a realistic application, and the second example is the simplified task graph of the packet decoding mode of a WLAN 802.11p transceiver application. We show that for both buffers with blocking and non-blocking writes improved results can be obtained compared to a post-analysis buffer sizing [3], [20] if our iterative buffer sizing approach is applied. Note that neither RTC [13] nor the SymTA/S approach [4] can analyze the presented applications, as both applications combine cyclic data with cyclic resource dependencies.

A. Synthetic Application

Figure 5 depicts the HSDF model of the task graph of a synthetic application. We use this example for an illustration of the different effects that occur on application of our iterative buffer sizing approach.

The tasks of the corresponding task graph are enabled by a periodic source τ_s with the frequency f. The source frequency f can be either $^1/_{12}$ MHz, $^1/_{11}$ MHz or $^1/_{10}$ MHz. Tasks are executed on three different processors, which is indicated by the different colors of vertices, and the BCETs and WCETs of tasks are denoted next to the vertices. The tasks τ_j and τ_l are executed on a shared processor with a static priority preemptive scheduler. Moreover, the FIFO buffers between tasks τ_i and τ_j and between tasks τ_k and τ_l are buffers with blocking writes and fixed capacities of two and one, respectively. The capacity of the FIFO buffer between τ_j and τ_k remains to be determined and is denoted with δ.

For the different frequencies we apply our integrated temporal analysis and buffer sizing approach, calculate the smallest sufficient buffer capacity δ and determine whether any constraints are violated. Thereby we assume that the buffer between τ_j and τ_k either blocks or does not block on writes and compare our results to the results obtained with the analysis flow from [20], in which unknown buffer capacities are assumed to be infinite during temporal analysis.

For the first case, in which task τ_l has a higher priority than task τ_j, the results are presented in Table I. It can be seen that if our iterative buffer sizing is applied, both the resulting buffer capacity δ and the maximum response time of task τ_j are smaller or equal compared to the results of a post-analysis buffer sizing for all source frequencies. This is due to the fact that a consideration of the iteratively computed estimates on buffer capacities in the interference calculation leads to an accuracy improvement of temporal analysis.

Buffer Sizing	Post-Analysis		Iterative	
Writes	Non-blocking	Blocking	Non-blocking	Blocking
$^1/_{12}$ MHz	$\delta = 2$ $\hat{R}'_j = 15$	$\delta = 2$ $\hat{R}'_j = 15$	$\delta = 2$ $\hat{R}'_j = 12$	$\delta = 1$ $\hat{R}'_j = 9$
$^1/_{11}$ MHz	constraint violation	constraint violation	$\delta = 2$ $\hat{R}'_j = 12$	$\delta = 1$ $\hat{R}'_j = 9$
$^1/_{10}$ MHz	constraint violation	constraint violation	constraint violation	$\delta = 2$ $\hat{R}'_j = 12$

TABLE I. BUFFER SIZING RESULTS FOR FIGURE 5 ($\hat{R}'_j$ IN µs).

For instance, the interference calculated for the frequency $f = {}^1/_{12}$ MHz and buffers with non-blocking writes is smaller if estimates on the buffer capacity δ are considered during temporal analysis. This can be seen by comparing the parameters of the minimum function in $\eta'_{l\to j}(\Delta t, q)$ from Equation 4. On convergence of the analysis flow, the function $\gamma_{l\to j}(q)$ from Equation 7, which bounds interference based on cyclic data dependencies, resolves to the following:

$$\gamma_{l\to j}(1) = \delta + 1 - 1 = \left\lceil \frac{\hat{s}_k + \hat{R}'_k - \check{s}_j}{P_k} \right\rceil = \left\lceil \frac{21}{12} \right\rceil = 2$$

This is smaller than the jitter-based bound on interference $\eta_l(\Delta t)$ from Equation 1:

$$\eta_l(\hat{R}'_j) = \left\lceil \frac{J_l + \hat{R}'_j}{P_l} \right\rceil = \left\lceil \frac{17 + 12}{12} \right\rceil = 3$$

In case of a post-analysis buffer sizing, the buffer capacity δ is assumed to be infinite during temporal analysis. The interference is therefore only bounded by $\eta_l(\hat{R}'_j)$, resulting in a decreased accuracy compared to an iterative buffer sizing.

Moreover, according to Equation 11, the capacity δ computed for a buffer with blocking writes is always smaller or equal to the capacity of a buffer with non-blocking writes, which also results in smaller or equal interference. For instance, for the frequency of $f = {}^1/_{11}$ MHz it holds on convergence if a buffer with blocking writes is used:

$$\delta = \left\lceil \frac{\hat{s}_k + \hat{R}'_k - \hat{s}_k}{P_k} \right\rceil = \left\lceil \frac{17 + 2 - 8}{11} \right\rceil = 1$$

This is obviously smaller than the computed convergent capacity for the case of a buffer with non-blocking writes:

$$\delta = \left\lceil \frac{\hat{s}_k + \hat{R}'_k - \check{s}_k}{P_k} \right\rceil = \left\lceil \frac{20 + 2 - 1}{11} \right\rceil = 2$$

Remember that the interference limitation observed when buffers with blocking writes are used is not only an accuracy improvement in the analysis model, but a limitation of interference of the actual tasks. Whenever a buffer with blocking writes has a smaller capacity than a sufficiently large buffer with non-blocking writes, the interference between the tasks connected by the buffer is effectively reduced. For instance, the buffer with blocking writes of the capacity $\delta = 1$ makes the executions of tasks τ_j and τ_k mutually exclusive, from which follows that task τ_l can preempt task τ_j only once per execution of task τ_i.

Due to these effects it can be concluded that all buffer sizing approaches converge without a violation of constraints for the source frequency $f = {}^1/_{12}$ MHz, with the buffer capacity δ being the smallest for an iterative buffer sizing of a buffer with blocking writes. For $f = {}^1/_{11}$ MHz, the methods with a post-analysis buffer sizing report a violation of constraints. For $f = {}^1/_{10}$ MHz only the combination of iterative buffer sizing and a buffer with blocking writes results in convergence without violation of constraints.

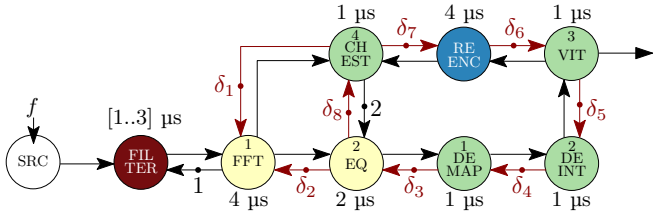

Fig. 6. HSDF graph of the packet decoder of a WLAN 802.11p transceiver.

In the second case the priorities of the tasks τ_j and τ_l are reversed, such that task τ_j has a higher priority than task τ_l. For the source frequency of $f = {}^1/_{10}$ MHz, a violation of temporal constraints can be observed for all four buffer sizing techniques. To be more precise, the maximum response time of the task τ_l is computed as $\hat{R}'_l = 21$ µs for the post-analysis buffer sizing techniques, $\hat{R}'_l = 15$ µs for the iterative buffer sizing of a buffer with non-blocking writes and $\hat{R}'_l = 9$ µs for the iterative buffer sizing of a buffer with blocking writes. Although all these maximum response times are too large for the temporal constraint imposed by the rightmost cycle, the results still indicate that the iterative buffer sizing for both buffers with blocking and non-blocking writes can lead better analysis results, no matter how the priorities of tasks are distributed. Finally, for the frequencies $f = {}^1/_{11}$ MHz and $f = {}^1/_{12}$ MHz only the iterative buffer sizing of a buffer with blocking writes leads to convergence without a violation of constraints, with $\delta = 1$ and $\hat{R}'_l = 9$ µs.

B. WLAN 802.11p Transceiver

In this section it is illustrated that applying our iterative buffer sizing flow can be beneficial for more realistic applications as well. We analyze the task graph of a WLAN 802.11p transceiver [1]. This application has several modes and is executed on a multiprocessor system for performance reasons. We only consider the part of the task graph that is active during packet decoding mode.

An HSDF model corresponding to the task graph of the packet decoding mode is shown in Figure 6. A periodic source with the frequency f models the input of this dataflow graph. For illustration purposes, the source frequency f can be either $^1/_{10}$ MHz, $^1/_8$ MHz or $^1/_7$ MHz. All received symbols are first processed by a filter with a variable WCET and then processed by an FFT. Filter and FFT communicate via a FIFO buffer with blocking writes of the capacity one, which is represented by the leftmost cyclic data dependency. For the other buffers we will use our analysis flow to determine sufficiently large buffer capacities δ_1 to δ_8.

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).

All tasks are mapped to four 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. We apply our integrated temporal analysis and buffer sizing approach for the different source frequencies and,

Buffer Sizing	Post-Analysis		Iterative	
Writes	Non-blocking	Blocking	Non-blocking	Blocking
$^1/_{10}$ MHz	$\sum \delta = 13$ $\sum \hat{R}' = 25$	$\sum \delta = 9$ $\sum \hat{R}' = 25$	$\sum \delta = 10$ $\sum \hat{R}' = 21$	$\sum \delta = 8$ $\sum \hat{R}' = 21$
$^1/_8$ MHz	constraint violation	constraint violation	$\sum \delta = 11$ $\sum \hat{R}' = 21$	$\sum \delta = 9$ $\sum \hat{R}' = 21$
$^1/_7$ MHz	constraint violation	constraint violation	constraint violation	$\sum \delta = 9$ $\sum \hat{R}' = 21$

TABLE II. BUFFER SIZING RESULTS FOR FIGURE 6 ($\sum \hat{R}'$ IN µs).

in case no constraint is violated, compute the smallest sufficient buffer capacities δ_1 to δ_8 for both buffers with blocking and non-blocking writes.

The results of our iterative buffer sizing compared to a post-analysis buffer sizing are presented in Table II. It can be seen that also for this more realistic example a higher guaranteed throughput can be obtained, i.e. convergence without violation of constraints can be achieved for higher source frequencies, if an iterative buffer sizing is used instead of a post-analysis buffer sizing. Moreover, buffer capacities and interference are smaller when buffers with blocking writes are used instead of buffers with non-blocking writes. For buffers with non-blocking writes it follows that with an iterative buffer sizing the obtainable guaranteed throughput is 25% higher than with a post-analysis buffer sizing, while 23% smaller buffer capacities can be achieved for the same guaranteed throughput. For buffers with blocking writes the obtainable guaranteed throughput is even 43% higher and buffer capacities are 11% smaller for the same guaranteed throughput.

VI. CONCLUSION

In this paper we presented an iterative buffer sizing approach for real-time stream processing applications. The novelty of the presented approach is that it takes into account that FIFO buffers bound interference between tasks, which can result in a higher guaranteed throughput.

For the case that buffers with blocking writes are used, it was shown that the relation between buffer capacities and minimum throughput is non-monotone in general, a reduction of buffer capacities can lead to higher throughput guarantees. This is due to the fact that buffers with blocking writes effectively reduce jitter between tasks, allowing for less interference and smaller response times.

Such a jitter reduction cannot occur if buffers with non-blocking writes are used. However, it was demonstrated that bounding interference in the analysis flow by iteratively calculated estimates on buffer capacities can nevertheless result in tighter bounds on both buffer capacities and minimum throughput, compared to existing approaches.

We showed that maximum enabling jitters and buffer capacities cannot decrease throughout successive iterations of our analysis flow. Moreover, we derived that our analysis flow converges if communication between tasks is limited to FIFO buffers and if an upper bound for all buffer capacities is specified before analysis.

The benefits of our approach were illustrated in a case study using a WLAN 802.11p transceiver application. It was shown that if buffers with non-blocking writes were used, an up to 25% higher guaranteeable throughput and up to 23% smaller buffer capacities could be determined, compared to a backlog-based calculation of buffer capacities after analysis. For systems using buffers with blocking writes the guaranteeable throughput was even up to 43% higher and buffer capacities up to 11% smaller.

While for other applications the extent of improvement can vary, it is guaranteed that tighter bounds are computed with the presented buffer sizing approach, as it exploits an effect that other approaches do not take into account.

REFERENCES

[1] P. Alexander, D. Haley, and A. Grant. Outdoor mobile broadband access with 802.11. *IEEE Communications Magazine*, 45(11):108–114, 2007.

[2] J. Hausmans, S. Geuns, M. Wiggers, and M. Bekooij. Temporal analysis flow based on an enabling rate characterization for multi-rate applications executed on MPSoCs with non-starvation-free schedulers. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 108–117, 2014.

[3] 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)*, pages 13–22, 2013.

[4] R. Henia et al. System level performance analysis – the SymTA/S approach. *IEE Proc. of Computers and Digital Techniques*, 152(2):148–166, 2005.

[5] M. Jersak. *Compositional performance analysis for complex embedded applications.* PhD thesis, Braunschweig University of Technology, 2005.

[6] C.-W. Lin et al. Timing analysis of process graphs with finite communication buffers. In *Real-Time and Embedded Technology and Applications Symp. (RTAS)*, pages 227–236, 2013.

[7] O. Moreira, T. Basten, M. Geilen, and S. Stuijk. Buffer sizing for rate-optimal single-rate data-flow scheduling revisited. *IEEE Trans. on Computers*, 59(2):188–201, 2010.

[8] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *IEEE Real-Time Systems Symp. (RTSS)*, pages 236–245, 2003.

[9] H. Salunkhe, O. Moreira, and K. van Berkel. Buffer allocation for real-time streaming on a multi-processor without back-pressure. In *IEEE Symp. on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, 2014.

[10] S. Sriram and S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. CRC Press, 2009.

[11] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Design Automation Conf. (DAC)*, pages 899–904, 2006.

[12] S. Stuijk, M. Geilen, and T. Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. on Computers*, 57(10):1331–1345, 2008.

[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)*, volume 4, pages 101–104, 2000.

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

[15] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[16] 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*, 8(6):649–667, 2006.

[17] M. Wiggers, M. Bekooij, and G. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Design Automation Conf. (DAC)*, pages 658–663, 2007.

[18] M. Wiggers, M. Bekooij, and G. Smit. Modelling run-time arbitration by latency-rate servers in dataflow graphs. In *Int'l Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pages 11–22, 2007.

[19] M. Wiggers, M. Bekooij, and G. Smit. Monotonicity and run-time scheduling. In *ACM Int'l Conf. on Embedded Software (EMSOFT)*, pages 177–186, 2009.

[20] P. Wilmanns, J. Hausmans, S. Geuns, and M. Bekooij. Accuracy improvement of dataflow analysis for cyclic stream processing applications scheduled by static priority preemptive schedulers. In *Euromicro Conf. on Digital System Design Architectures, Methods and Tools (DSD)*, pages 9–18, 2014.