Throughput Analysis of Run-Time Scheduled Multi-Rate Systems with Backpressure
Marco Bekooij\textsuperscript{1}, Maarten Wiggers\textsuperscript{2}, and Jef van Meerbergen\textsuperscript{3,4}
\textsuperscript{1}NXP semiconductors, Eindhoven, The Netherlands
\textsuperscript{2}University of Twente, Enschede, The Netherlands
\textsuperscript{3}Philips Research, Eindhoven, The Netherlands
\textsuperscript{4}Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract.
The throughput analysis technique described in this paper is intended for applications that are executed on heterogeneous embedded multiprocessor systems. A mix of run-time arbitration policies is often applied in these systems. Backpressure prevents buffer overflow because tasks do not start before sufficient buffer space is available.

We show that the effects of run-time arbitration can be included in the so-called worst-case response time (WCRT) of the actors of an MRDF graph. Given this model we compute conservative estimates of the arrival times of data in the system. Furthermore, we show that latency constraints can be included in a multi-rate dataflow model.

The analysis technique is demonstrated on a real-life car-radio application. In this application two independent streams are processed. One of these streams has a latency constraint.

1 Introduction
In many consumer systems, such as smart-phones, car-radios, and set-top boxes, a mix of firm real-time and soft real-time applications are executed simultaneously on a heterogeneous embedded multiprocessor system. For performance reasons, these applications are partitioned in tasks that communicate via FIFO buffers and are executed on several processors. A mix of arbitration policies, such as time division multiplex (TDM), round-robin and static priority preemptive, can be applied in these embedded multiprocessor systems.

In our system, tasks do not execute if there is insufficient space in the output buffers, which results in backpressure. Therefore, unlike the event-model approach [11][6], no other means are needed to prevent buffer overflow, such as traffic shapers or knowledge about the best-case execution time of tasks. Another advantage of backpressure is that the system can be made work-conserving, i.e. progress is made if sufficient data and space are available. The check, whether sufficient space is available before tasks start, can be implemented in software [13].

In this paper we describe an analysis technique to derive the minimum throughput of an application that processes a data stream. The tasks of the application are executed on a multiprocessor system in which a mix of run-time arbitration policies are applied. The tasks start if sufficient data and space is available and communicate containers filled with data via FIFO buffers with a predefined capacity. The application is modeled as a multi-rate dataflow (MRDF) graph [1][8][12]. End-to-end latency constraints are expressed in this MRDF graph such that they are verified.

We show that the effects of run-time arbitration can be included in the so-called worst-case response time (WCRT) of the actors of an MRDF graph. Given this MRDF graph we show that conservative estimates of the arrival time of containers in our system can be computed efficiently, which is the main contribution of this paper. We require that the WCRT of the actors can be computed given the arbiter settings and the worst-case execution time (WCET) of the tasks. Suitable arbitration policies include TDM and round-robin.

The key observation is that the arrival time of containers can be bounded from above because the corresponding dataflow model has a monotonic temporal behavior. Monotonicity implies that a decreasing response time does not result in a later arrival time of containers. We can therefore derive the minimum throughput with maximum cycle mean (MCM) analysis [2][4]. In case of an excessive run-time of the MCM algorithm, we use a conservative approximation algorithm with a linear computational complexity in the number of MRDF nodes and edges to check whether a specified minimum throughput constraint is satisfied [15]. The same algorithm is used to compute the FIFO buffer capacities.

In our system we apply run-time arbitration. Scheduling approaches that do not make use of run-time arbitration, like the time triggered [7] and the static-order [12] approaches, are difficult to apply for systems that process a mix of streams with firm or soft real-time constraints. This is because tasks of soft real-time applications often have execution times that are impractical to bound, and can have a data dependent number of task executions. Another complication is that starting or stopping a stream should not affect the temporal behavior of other streams. However, if static-order scheduling is applied and one task does not receive input data then this task prevents the start of another task that is executed on the same processor.

Scheduling approaches that make use of run-time arbitration, as for instance presented by Jersak [6], God-
standard [5] or Maxiaguine [9], do not allow cyclic dependencies that influence the temporal behavior of the system. Not only do these cyclic dependencies occur, because of functional data dependencies, this also means that backpressure through FIFO buffers with a predefined capacity cannot be applied.

The outline of this paper is as follows. In Section 2 we describe the MRDF model and present its well known properties. In Section 3 we describe an application as a task graph. The construction of a MRDF model with a one-to-one correspondence with the task graph is described in Section 4. In Section 5 we use the one-to-one correspondence between the MRDF model and the task graph together with the properties of the MRDF model, to arrive at the important observation that worst-case production times of containers can be derived with the MRDF model. One of the inputs of the analysis is the WCRT of each task. In Section 6 we show that if TDMA arbitration or round-robin arbitration is applied, the WCRT of the tasks can be computed from the WCET of the tasks and the scheduler settings. How latency constraints can be included in the MRDF model is described in Section 7. The case study that illustrates the use of the analysis technique, is described in Section 8. In the last section, we present the conclusions.

2 Dataflow model

In this section we describe the MRDF graph and its properties. An MRDF graph is a digraph \( G_M = (V, E, \delta, \gamma, \pi, \rho) \). The vertices \( V \) in this MRDF graph are called actors. The edges \( E = \{(v_i, v_j) | v_i, v_j \in V\} \) represent queues. The number of initial tokens on an edge \( e \) equals \( \delta(e) \), with \( \delta : E \rightarrow \mathbb{N} \). An actor is enabled when the number of tokens that will be consumed during its execution is available on all its input edges. The execution of an actor can start after it is enabled. An actor \( v_j \) consumes \( \gamma(e) \) tokens per execution from input edge \( e = (v_i, v_j) \), with \( \gamma : E \rightarrow \mathbb{N} \). The tokens are consumed at the moment the actor starts. An actor \( v_j \) produces \( \pi(e) \) tokens per execution on output edge \( e = (v_j, v_k) \), with \( \pi : E \rightarrow \mathbb{N} \). The finish time of actor \( v_i \) is its worst-case response time \( \rho(v_i) \) after its enabling, with \( \rho : V \rightarrow \mathbb{R} \). The tokens are produced at the moment the actor finishes its execution.

We can describe the topology of an MRDF graph with a topology matrix \( \Gamma \) [8]. The matrix \( \Gamma \) is a \( |E| \times |V| \) matrix, where:

\[
\Gamma_{ij} = \begin{cases} 
\pi(e_i) & \text{if } e_i = (v_j, v_k) \\
-\gamma(e_i) & \text{if } e_i = (v_k, v_j) \\
\pi(e_i) - \gamma(e_i) & \text{if } e_i = (v_j, v_k) \\
0 & \text{otherwise}
\end{cases}
\]

If the rank of \( \Gamma \) is \( |V| - 1 \), then a connected MRDF graph is said to be consistent. A consistent MRDF graph requires queues with a finite capacity, while an inconsistent graph requires queues with an infinite capacity. The vector \( \mathbf{x} \) of length \( |V| \), for which \( \Gamma \mathbf{x} = 0 \), is the repetition vector of the MRDF. The repetition vector contains the repetition factor of each actor. The repetition factors define the relative execution frequencies of the actors.

2.1 Self-timed execution

If an MRDF graph is executed in a self-timed manner, then actors start as soon as they are enabled. If each actor has either a constant response time or has a self-cycle with one initial token, then FIFO ordering is maintained by the actors because tokens do not overtake each other.

2.2 Maximum Cycle Mean analysis

The throughput during self-timed execution of a consistent MRDF graph for which FIFO ordering is maintained, can be derived analytically with maximum cycle mean (MCM) analysis [12]. Every consistent MRDF graph in which FIFO order is maintained can be transformed into a single rate dataflow (SRDF) graph [12], which we can use to derive properties of the MRDF graph.

An SRDF graph is a digraph \( G = (V, E, \delta, \gamma, \pi, \rho) \) for which \( \forall e \in E, \gamma(e) = 1 \) and \( \pi(e) = 1 \). MCM analysis is performed on SRDF graphs. The self-timed execution of an SRDF graph has the following properties that are relevant for this paper.

First of all, the self-timed execution of an SRDF graph is deadlock-free if there is at least one initial token on every cycle in the SRDF graph. Secondly, a deadlock free SRDF graph enters a periodic regime after a transition phase. More precisely, there exist a \( K \in \mathbb{N} \), an \( N \in \mathbb{N} \) and a \( \mu \in \mathbb{R} \), such that for an actor \( v \in V \), given \( k > K \) the start time \( s(v, k + N) \) in iteration \( k + N \) is described by:

\[
s(v, k + N) = s(v, k) + \mu \cdot N
\]

Equation 1 states that the execution of an actor \( v \) enters a periodic regime after \( K \) executions. The time one period spans is \( \mu \cdot N \). The number of executions of an actor \( v \) in one period is denoted by \( N \). Thus, \( \mu \) is equal to the inverse of the throughput measured over one period.

The Cycle Mean (CM) of a simple cycle \( c \) in the SRDF graph \( G \) is given by (2). In this equation, \( c \) denotes a simple cycle \( <(v_i, v_j), (v_j, v_k), ..., (v_l, v_i)> \). The Maximum Cycle Mean (MCM) [12] of an SRDF, which is equal to \( \mu \), is given by (3). In this equation, \( C_G \) is the set of simple cycles in the graph \( G \). The critical cycles are the cycles with a CM that is equal to \( \mu(G) \).

\[
\text{CM}(c) = \frac{\sum_{(v_i, v_j) \in c} \rho(v_i)}{\sum_{(v_i, v_j) \in c} \delta((v_i, v_j))}
\]

\[
\mu(G) = \max_{c \in C_G} \text{CM}(c)
\]
The \( \mu(G) \) of an SRDF graph \( G \) can be calculated with a polynomial algorithm \([2][3]\) or an algorithm which has a low run-time for many practical cases \([4]\). In case of an excessive run-time of the MCM algorithm we check with a conservative approximation algorithm \([15]\) whether a periodic schedule exists that satisfies the specified throughput constraint. This algorithm has a linear computational complexity in the number of MRDF nodes and edges, even for complex examples run-times below a second are reported.

### 2.3 Monotonic execution

The self-timed execution of a strongly connected MRDF graph that maintains FIFO ordering is monotonic in the response time. Monotonicity implies that decreasing actor response times cannot result in a later enabling of actors. The reason is that a decrease of a response time of an actor results in earlier production of tokens, and therefore in an earlier actor enabling.

Given that the self-timed execution of an MRDF is monotonic, the minimum throughput during self-timed execution of a consistent MRDF graph for which FIFO ordering is maintained can be determined with MCM analysis.

Given that the self-timed execution of an MRDF is monotonic, we can observe worst-case token arrival times with a dataflow simulator. In this dataflow simulator the actors start as soon as they are enabled and have a response time equal to their worst-case response time. Worst-case token arrival times are observed during the transition phase as well as during the periodic regime.

### 3 Task graph definition

An application is described by a task graph. A task graph is a weakly connected digraph \( G_T = (T, B, \zeta, \eta, \lambda, \xi, \sigma, \psi) \). A weakly connected directed graph is a graph in which for every pair of vertices \( a \) and \( b \) either a path exists from \( a \) to \( b \) or from \( b \) to \( a \). The vertices \( T \) in the task graph represent tasks. The directed edges \( B = \{(t_i,t_j)\mid t_i, t_j \in T\} \) represent FIFO buffers. The capacity of a FIFO buffer is the number of containers that can be stored in the buffer. The capacity equals \( \zeta(b) \), with \( \zeta : B \rightarrow \mathbb{N} \). The amount of data that can be stored in a container is defined per FIFO buffer. The number of containers stored in a buffer \( b \) at the start of the application is \( \eta(b) \), with \( \eta : B \rightarrow \mathbb{N} \).

A task is enabled when the number of containers that will be consumed during its execution is available in its input buffers and space for the number of containers that will be produced is available in its output buffers. A task checks if it is enabled before it starts. This check prevents that a task needs to wait for additional data or space before it can finish its execution. A task \( t_j \) consumes \( \lambda(b) \) containers per execution from buffer \( b = (t_k, t_j) \), with \( \lambda : B \rightarrow \mathbb{N} \). The containers are consumed before the task finishes its execution. A task \( t_j \) produces \( \xi(b) \) containers per execution in the buffer \( b = (t_j, t_k) \), with \( \xi : B \rightarrow \mathbb{N} \). The response time is the difference between enabling and finish of a task. The response time of task \( t_j \) is smaller than its worst-case response time \( \sigma(t_j) \), with \( \sigma : T \rightarrow \mathbb{R} \). The containers are produced before the task finishes its execution. After its start, a task finishes its execution within its WCET, if it is not preempted during its execution. The WCET of task \( t_i \) is \( \psi(t_i) \), with \( \psi : T \rightarrow \mathbb{R} \).

![Figure 1. A task graph example with \( b = (t1,t2), \xi(b) = 2, \lambda(b) = 3, \eta(b) = 1 \).](image1)

![Figure 2. An MRDF graph that has a one-to-one correspondence with the task graph in figure 1](image2)

The MRDF graph in Figure 2 corresponds with the task-graph in Figure 1 if the capacity of the FIFO buffer is 6 containers.

Since the task graph is weakly connected and each buffer in the task graph results in two edges in opposite direction in the MRDF graph, a task graph is modeled with a strongly connected MRDF graph. FIFO ordering is maintained because each actor has a self-edge with one initial token. In Section 2.3 we have shown that...
the self-timed execution of a strongly connected MRDF graph that maintains FIFO ordering is monotonic.

There is a one-to-one correspondence between the task graph and the MRDF model because each task has a corresponding actor with the same enabling condition. Each edge in the task graph corresponds with two edges in the MRDF model. The availability of data or space in a FIFO buffer corresponds with the presence of tokens on one of these edges.

Space that is made available in a FIFO buffer if a container is consumed by a task can be seen as the production of an empty container. The production of an empty container corresponds with the production of a token in the MRDF graph. If a task \( t_i \) and its corresponding actor \( v_i \) are enabled at the same point in time then an empty container is produced earlier than the corresponding token. The reason is that task \( t_i \) produces empty containers before it finishes, i.e. before \( \sigma(t_i) \) after its enabling, while an actor \( v_i \) produces tokens when it finishes, i.e. exactly \( \rho(v_i) = \sigma(t_i) \) after its enabling. The earlier production of empty containers results in an earlier enabling of tasks than actors.

**5 Worst-case production times**

In this section we show that containers are not produced later than the corresponding tokens are produced during self-timed execution. We arrive at this conclusion for the following reasons. First of all, there is a one-to-one correspondence between the task graph and the MRDF graph. The only relevant difference is that the response time of the tasks is shorter than or equal to the response time of the corresponding actors. Furthermore we know that the self-timed execution of the MRDF graph has a monotonic temporal behavior. Therefore, we arrive at the following conclusion: if during self-timed execution, the first execution of each task starts not later than the execution of the corresponding actor, then a shorter response time of a task cannot result in a later production of containers compared to the production of the corresponding tokens.

Sriram [12] bases his work on a similar observation about the arrival time of containers compared to tokens. An important difference is that [12] does not introduce the concept of a response time and therefore a task has to start immediately when it is enabled. Given the concept of response times, a task starts at a point in time after being enabled, and finishes within its WCRT after being enabled. This allows to model run-time arbiters that schedule enabled tasks later than the actual enabling. For a specific class of arbiters we can compute the WCRT of a task based on the WCET of the tasks and the arbiter settings, and therefore independent of the time interval between consecutive container arrivals. Two arbiters from this class are discussed in Section 6.

The arrival times of tokens which are a conservative upperbound for the arrival times of the containers in the system can be found with a dataflow simulator or by computing a schedule at design-time.

The arrival time of the tokens during self-timed execution of the MRDF graph can be observed with a dataflow simulator because actors start in the dataflow simulator as soon as they are enabled and produce tokens after their WCRT. Simulation can be stopped after the first period of the periodic regime. Unfortunately, there can be an extremely long preamble which results in a long simulation time.

Another option is to compute a strictly periodic [15] or a multidimensional periodic schedule at design-time [14] that satisfies the throughput constraint. Computation of a strictly periodic schedule has a linear computational complexity in the number of MRDF nodes and edges. In this schedule, the start time of the actors cannot be earlier than during self-timed execution. If actors start during self-timed execution earlier than in the at design-time computed schedule they will also produce their tokens earlier. An earlier production of tokens can only result in an earlier enabling of actors.

**6 Response time calculation**

One of the inputs of the analysis is the WCRT of each task. The presented analysis technique is therefore only applicable if the WCRT of a task can be derived from the arbiter settings and the WCET of the tasks that are executed on the same processor. This is the case for TDM arbitration, and round-robin arbitration. If, however, static priority preemptive arbitration is applied then the response time of a task also depends on the interval of time between the arrival of containers that are consumed by a task with a higher priority.

The WCRT of a task \( t_i \) can be calculated with (4) if TDM arbitration is applied. The length of the TDM period is \( p \) and the length of the time slice of task \( t_i \) is \( s_i \). The WCRT can be computed with this equation because task \( t_i \) is at most \( r = \lceil \frac{\psi(t_i)}{s_i} \rceil \) times preempted during its execution. The total time that the task cannot make progress because it is preempted, is thus at most \( (p - s_i)r \).

\[
\sigma(t_i) = \psi(t_i) + (p - s_i)\left\lfloor \frac{\psi(t_i)}{s_i} \right\rfloor
\]

A tight bound on the WCRT is computed with (4) if \( s_i \) is small compared to \( \psi(t_i) \). In this case, the effect of rounding to the next larger integer value is small.

The task \( t_i \) is busy waiting if it is not enabled. The time that task \( t_i \) is busy waiting is not included in the response time because the response time is defined as the interval of time between enabling and finish.

A too early arrival of a container allows for a longer stay in a FIFO buffer. Therefore, a potentially too early arrival of containers should not increase the WCRT estimate. According to (4) this is the case for dataflow models. However this is not the case for event-models [6]. As
a consequence, a more accurate estimate of the minimal throughput and maximum end-to-end latency can be obtained with dataflow models than with event-models.

If round-robin arbitration is applied, then the WCRT of a task $t_i$ can be calculated with Equation 5. In this equation $P$ is the set of tasks in the round-robin list and $c$ is the time it takes to check whether the task is enabled. The equation produces the WCRT because after the container arrives that enables $t_i$ it takes at most $\sum_{t_k \in P \setminus \{t_i\}} \psi(t_k) + c$ before it is checked whether $t_i$ is enabled. The time $c$ is added because the container can arrive just after it was checked whether the task is enabled. However, this check still takes some time to return before other tasks can start their execution. After it is detected that the actor is enabled it takes at most $\psi(t_i)$ before the task finishes its execution.

$$\sigma(t_i) = \sum_{t_k \in P} \psi(t_k) + c \quad (5)$$

### 7 Latency constraints

A maximum latency constraint between two actors with the same repetition factor of which one is executed strictly periodically can be modeled in an MRDF graph.

Equation 6 defines a maximum latency $l$ between the start-time of the $i$-th execution of actor $v1$ and the start-time of the $i$-th execution of actor $v2$. Under the assumption that actor $v2$ executes strictly periodically and has a constant response time $\rho(v2)$, this latency constraint can be included by introducing a cycle with an additional actor $v3$ and three edges, as is shown in Figure 3. If the constant response time of $v3$ is $\rho(v3)$, then the start time of actor $v1$ is defined by (7) and (8). Equation 8 can be rewritten in Equation 9 because actor $v2$ executes strictly periodically with a period $q$. After another rewriting step this results in Equation 10 which is equal to Equation 6 with $l = nq - \rho(v3) - \rho(v2)$.

$$s(v2, i) \leq s(v1, i) + l \quad (6)$$

$$s(v1, i) \geq s(v3, i - n) + \rho(v3) \quad (7)$$

$$s(v1, i) \geq s(v2, i - n) + \rho(v2) + \rho(v3) \quad (8)$$

$$s(v1, i) \geq s(v2, i) - nq + \rho(v2) + \rho(v3) \quad (9)$$

$$s(v2, i) \leq s(v1, i) + nq - \rho(v2) - \rho(v3) \quad (10)$$

A maximum latency constraint ($l \geq 0$) needs only to be verified at design time. The containers will arrive in time due to monotonicity.

### 8 Case-study

In this section we use the described analysis technique to check whether the throughput and latency constraints of a car-radio application are met. Given these constraints we will also derive the capacity of the FIFO buffers.

In this car-radio application, a phone call from a cell phone with bluetooth (BT) is handled while playing music from an MP3 encoded song. Audio echo cancellation is used to prevent the howling effect and to cancel the sound from the loudspeakers, such that a signal for the bluetooth device is obtained that only contains the speech of the user. The latency between the microphone and the BT devices should be shorter than 30 ms.

The block diagram of the application is shown in Figure 4. In this application, a block reader (BR) task accesses a compact disk and sends an encoded audio stream to an MP3 decoder task. The stream produced by the MP3 decoder task is converted by a sample rate converter (SRC) into a 44.1 kHz sample stream. The MP3 decoder task has a WCET of 4.7 ms. At the same time an 8 kHz stream from the bluetooth device is converted into a 44.1 kHz sample stream. After mixing and audio post processing (APP) the stream is sent to the loudspeakers. At the same time a stream from a microphone is converted into a 8 kHz sample stream. The speech signal of the user is obtained by removing the sound from the loudspeakers with an audio echo cancellation (AEC) task. The stream from the AEC task is sent to the output (OUT) task that produces a strictly periodically 8 kHz sample stream. The WCET of the AEC task is 5 ms. The algorithmic delay of the AEC task is 6 ms because it contains a 48 tap filter.

The stream from the MP3 decoder should be independent from the stream originating from the microphone because it must be possible to start or stop one
stream while continuing the other without interruption of the sound. For example, the speech signal should not be interrupted if the MP3 decoder task stops at the end of a song.

In our system [10], we execute the BR task on an ARM7 micro-processor. Preemptive task switching is applied because non real-time system configuration software is executed on the same processor. The computational load generated by the APP task plus the OUT and SRC tasks requires one DSP. The MP3 decoder and the AEC task execute on another DSP. Each sample rate converter is implemented with 2 tasks. Each SRC task runs in its own interrupt service routine that is activated with a fixed frequency. The MP3 decoder and the AEC task are round-robin scheduled because the applied DSPs do not support preemptive task switching. Round-robin scheduling is applied instead of static order scheduling because the MP3 decoder can stop at any moment and this should not prevent execution of the AEC task.

MP3 decoding and audio echo cancellation can be studied in isolation because round-robin arbitration is applied and the SRC tasks and the OUT task are executed strictly periodically. In the next section, we check with the presented analysis technique whether the throughput constraint for MP3 decoding is met. We also derive suitable FIFO buffer capacities. We repeat the same analysis for audio echo cancellation in Section 8.2.

8.1 MP3 decoding

The task graph with the MP3 decoding task is shown in Figure 5. The SRC task is executed every 1/48 ms. The WCRT of the BR task depends on the behavior of the compact disk player. We will assume a WCRT of 11 ms. The FIFO buffer \( b1 = (\text{MP3}, \text{SRC}) \) has a capacity of \( m1 \) containers and is initially empty. The FIFO buffer \( b2 = (\text{MP3}, \text{BR}) \) has a capacity of two containers and is initially full.

![Figure 5. Task graph with the MP3 task.](image)

The BR task stores in FIFO buffer \( b3 = (\text{BR}, \text{MP3}) \) encoded MP3 data. The MP3 decoder task consumes each execution a variable number of bytes from this buffer. This buffer is not shown in Figure 5 because we take care that it can be omitted during analysis such that the MP3 decoder can be represented in an MRDF graph. It can be omitted during analysis if sufficient space and data is always available in buffer \( b3 \). To achieve this, the MP3 task informs the BR how many bytes with encoded data it has consumed during its previous execution. This is achieved by sending a container to the BR task via buffer \( b2 \). The value stored in this container is equal to the number of bytes consumed during the previous execution of the MP3 task. The BR task will write this number of bytes in buffer \( b3 \). Initially two containers are stored in buffer \( b2 \). The initial value in each container is equal to the maximum number of bytes that the MP3 task can consume per execution.

The corresponding MRDF graph with an MP3 actor is shown in Figure 6. The response time of the SRC actor and BR actor is 1/48 ms and 11 ms, respectively. Because round-robin arbitration is applied, the response time of the MP3 actor is equal to the sum of the WCET of the AEC and MP3 task plus \( c = 1 \mu s \), i.e. \( 5 + 4.7 + 0.001 = 9.701 \) ms.

The repetition vector of the MRDF graph in Figure 6 is \([1 \ 1 \ 576]^T \). After expansion into an SRDF graph, the cycle with one token through all 576 SRC actors should be a critical cycle in the graph that determines the MCM such that the SRC task can execute strictly periodically during the periodic regime. The desired MCM of this graph is therefore \( 576 \cdot \frac{48}{43} = 12 \) ms. With our buffer calculation algorithm [15] we found in less than a second that 1042 tokens is a sufficient capacity for \( m1 \) to obtain an MCM of 12 ms. With a dataflow simulator we have verified that the SRC task executes indeed strictly periodically after the first tokens arrive.

8.2 Audio echo cancellation

The task graph with the AEC task is shown in Figure 7. The SRC, the ADC and the OUT task are executed every 1/8 ms. The FIFO buffers have a capacity of \( n1, n2 \) and \( n3 \) containers and are initially empty. The corresponding MRDF graph is shown in Figure 8. The response time of the AEC task is equal to the sum of the WCET of the AEC and MP3 task plus \( c = 1 \mu s \), i.e. \( 5 + 4.7 + 0.001 = 9.701 \) ms. The response time of the SRC, the ADC and the OUT actor is 1/8 ms.

![Figure 7. Task graph with the AEC task](image)
ADC actor is 30 − 6 = 24 ms because the algorithmic delay is not modeled in the MRDF graph. This latency constraint is modeled with the edges \( e_1 = (\text{OUT}, v_3) \), \( e_2 = (v_3, \text{ADC}) \), \( e_3 = (\text{ADC}, \text{OUT}) \) with \( \pi(e_1) = \pi(e_2) = \pi(e_3) = 1 \), \( \gamma(e_1) = \gamma(e_2) = \gamma(e_3) = 1 \) and \( \delta(e_1) = \delta(e_3) = 0 \). If \( \rho(v_3) = 0 \) and \( \delta(e_2) = 193 \) we model in the MRDF graph the maximum latency constraint \( l = \delta(e_2) + \rho(v_3) + \rho(\text{OUT}) \) which is equal to \( l = 193 \cdot \frac{1}{(8 \cdot 10^3)} + 0 - \frac{1}{(8 \cdot 10^3)} = 24 \) ms.

The repetition vector of the MRDF graph in Figure 6 is \([80 \ 80 \ 80 \ 1]^T\). After expansion into an SRDF graph, the cycle with one token through all 80 OUT actors should be a critical cycle in the graph. Therefore, the desired MCM of this graph is \( 80 \cdot \frac{1}{8} = 10 \) ms. This guarantees that the OUT, SRC and ADC actor execute strictly periodic during the periodic regime. With our buffer calculation algorithm [15] we found that if \( n_1, n_2 \) and \( n_3 \) are all larger than 158 then an MCM of 10 ms is obtained. With a dataflow simulator we have verified that the SRC, ADC and OUT actors indeed execute strictly periodically. These actors execute strictly periodically if the SRC and ADC actors start at the same point in time and the OUT actor starts 19.7 ms later

9 Conclusion

This paper describes a novel analysis technique to derive the minimum throughput of multiprocessor systems in which backpressure is applied. Backpressure prevents buffer overflow without traffic shapers or knowledge about the best-case execution time of tasks. In these systems a mix of run-time arbitration policies can be used. These systems are suitable for the simultaneous processing of a number of streams with different real-time requirements.

The minimum throughput is derived with a multi-rate dataflow model that has a one-to-one correspondence with the task graph that is executed on the multiprocessor system. Each actor in this model has a worst-case response time that includes the effects of time-division multiplex arbitration or round-robin arbitration. The minimum throughput of the system can be derived with maximum cycle mean analysis because the self-timed execution of a multi-rate dataflow graph is monotonic. A latency constraint between an actor that is executed strictly periodically and another actor can be included in the dataflow model.

The analysis technique is applied on a real-life car-radio application in which two streams are processed simultaneously. For this application we are able to check the feasibility of the throughput and latency constraints and derive suitable FIFO buffer capacities.

We are currently setting up a mapping flow, for network-on-chip based multiprocessor systems, that determines the task to processor assignment, scheduler settings and buffer capacities. Most steps in this flow make use of the presented dataflow analysis technique.

References


