ABSTRACT

In this paper, a novel approach for integrating time slice based resource access in global performance analysis of distributed real-time critical embedded systems is presented. The performance analysis approach itself is based on bottom-up analysis of communicating processes under consideration of synchronization by inter-process communication and complex internal control flows of the processes. This general analysis methodology is extended concerning concurrent occupation of shared resources using time slice based access methods. The defined extensions are parameterizable for describing arbitrary communication media access schedules and software schedules on shared computation resources, although the explicit focus in this paper is on software scheduling. The applicability of the analysis extensions is presented by a case study of a multimedia subsystem implemented in SystemC.

Categories and Subject Descriptors
B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids

General Terms
Performance, Design, Verification

Keywords
Performance Analysis, Communicating Processes, Scheduling

1. INTRODUCTION

The advent of embedded systems in areas of life and engineering, joint with increasing quantitative and qualitative requirements in the areas of functionality, performance and comfort, pose novel challenges to system design. New methods need to be evolved for handling increasing design complexity and accompanied computational demands of distributed embedded system applications. As the ability to create high performance computational components and subsystems is raised, the demanded number of utility functions to be realized on each of these platform parts is steadily increasing. Further, a specialization of manufacturers and suppliers to some few parts of system design and development takes place. Due to that, less complete systems can be fully realized by only one responsible entity. Moreover, the integration of single software and hardware components into the overall system is getting a more and more lavish and time consuming, but essential and crucial step in the development process. Often, general purpose platforms are used for reducing the costs of hardware design and for decoupling the dependencies on multiple suppliers with the issues of oversized resources and unit costs as well as reduced customizability.

These platforms provide optimization degrees and customization potential to a slight extent. Architectural properties like clock frequencies and caches as well as runtime parameters like communication priorities and software scheduling policies can be modified. But even this slight number of parameters in combination with growing amounts of functional components spans rapidly growing design spaces. Even under these conditions, the integrability of software components provided by suppliers must be enabled.

Further, these systems are often real-time critical. This means, that hard performance requirements and constraints have to be guaranteed. Methods are needed to analyze timing properties like guaranteed data throughput or worst case end-to-end latencies. Currently, analysis of such systems is usually performed by simulation with the issues of incomplete coverage or infinite simulation times. This deduces the necessity of formal methods based on abstraction of system implementations for identifying corner cases in finite time. This allows coping even with future system complexities and an efficient evaluation of design decisions at system level of abstraction. Further challenges for system architects consist of integrating and scheduling software components that are potentially not yet implemented, available or that will be included later for providing additional features. Amongst real time requirements, robustness aspects concerning a potential failure propagation of these unknown components towards the total system have to be considered. Especially in the domain of distributed embedded automotive systems, wherein a majority of components is provided by suppliers, time slice based access methods have given an excellent account of itself due to robustness concerning a failure of single components and a predictability concerning timing properties. As result, potential failures are isolated and safety critical components, that are fully formal verified and correct except for external effects, are not affected. A common proceed for establishing a preemptive, predictable, robust scheduling in systems is the static allocation of exclusive resource access times for each competitor (communication instance or software process). This way,
the order of resource accesses can be fixed and isolated from each other. Additionally, a potentially unbounded resource occupation by competitors with long occupation times is prevented by making resource occupation preemptible. If the full time slice length is guaranteed to each software process, whether or not it requires the resource, this scheduling policy is similar to TDMA (Time Division Multiple Access) media access control. Hence, TDMA is considered as one general class for media access and software scheduling.

This paper describes new methods for formal analysis of safety critical embedded applications with hard timing constraints executed on multiple processors under time slice based resource access scheduling. The inclusion in a general framework for bottom-up worst case performance analysis as depicted in Figure 1 is described in Section 3. Unlike other approaches in this area that operate on explicitly modeled abstract task graphs, the presented analysis approach can be directly applied on a given C/C++ or SystemC model as functional system representation (bottom-up approach).

The influence of blocking communication instances on the global system timing is considered by identifying synchronizing communication and by determining the time a process is stalled at such a communication. The related analysis methodology called communication analysis is briefly described in Section 3.1. Section 4 analyzes the impact of TDMA scheduling on the global system timing and presents approaches for reducing analysis pessimism by enhanced cycle analysis under consideration of aspects like for example context switching costs. Although the focus in this paper is on software scheduling aspects, the defined analysis methods could be directly applied on temporal communication aspects. Due to that requires the description of formal analysis concerning temporal communication aspects (like e.g. communication latency and transmission duration), this is out of scope of this paper. The following sections will review existing approaches from literature. Finally, the applicability of the evolved methods is presented on the SystemC model of a multimedia system in Section 5.

2. RELATED WORK

Many approaches in pertinent and recent literature tackle the issue of global performance analysis under shared resource utilization. These can be distinguished between simulation based approaches and analytic approaches. Simulation based methodologies lack a sufficient corner case coverage in a tolerable amount of time. Although methodologies as TLM [4] allow the consideration of different parts of the system at different levels of abstraction for speeding up real-time analysis, TLM based approaches (like e.g. [14, 15]) can not break these limitations.

Analytic approaches have to be distinguished whether the underlying model is derived from functional system implementations [16, 2, 5] (white box, bottom-up approach) or whether the analysis model is explicitly modeled [19, 12, 13] or derived from system specifications like UML models [6] (black box approaches). Black box approaches abstract from the internal behavior of components and consider task graphs as representation of the internal communication and inter-task communication. Based on this model, efficient analysis towards performance and schedulability can be performed (e.g. TDMA scheduling and optimization [12, 19]). But the missing link from a direct extraction of the analysis model from functional implementations and lack of representing aspects like synchronization by communication and complex interaction schemes as well as task-internal control flow descriptions like loops and branches limit the practical applicability.

Other approaches like timed automata [7] or Petri nets [20] require an explicit description of the system using the analysis model and are very fine grained. The problem sizes to be analyzed are not suitable for complex systems. The application of these methods is more likely in the area of single component verification at late stages of the design flow.

White box approaches derive the analysis model directly from a functional implementation of a system. The approach presented in [2] extracts an abstract analysis component model from functional implementations. Issues are the incorporation of complex component interaction schemes as well as the consideration of different synchronization primitives by communication. In [5] MP-SoC application implementations are extracted and optimized with regard to performance properties and constraints. Differences consist of a lower level of abstraction, and the missing consideration of high-level communication protocols and software scheduling policies. In [1], parameters for optimized communication architectures are derived from a SystemC TLM model. The approach mainly concerns the interconnection and communication of the system, whereas we consider aspects of computation in this paper.

In [16] an approach for extracting a model called communication dependency graph from a functional implementation and information on component platform mapping is presented. The model can handle complex interaction schemes as well as synchronization by communication. The analysis methodology is restricted to include concurrent communication resource utilization by guaranteeing conflict free resource access using a method for conflict anal-
ysis. Further, only cooperative [17] and no preemptive software scheduling is considered during analysis. The approach presented in this paper is based on that approach and extends it towards time slice based resource access scheduling.

2.1 Communication dependency graph

For representing a system of communicating processes, a formal model called communication dependency graph (CDG) is used. A CDG is a directed, cyclic graph that can be constructed based on the control-flow graph (CFG) of each process and information on inter-process communication.

The edges $E_{com}$ are given by inter-process communication in the system. Edges $E_{cdg}$ represent the control-flow between two communication endpoints in the CFG. An edge $e_{cdg} \in E_{cdg}$ between two nodes in the CDG exists if a path in the CFG between the corresponding communication basic blocks exists. The minimal/maximal computation latencies $(c_{s_{min}}, c_{s_{max}})$ are attributed to each edge $e_{cdg}$. They represent the execution time of the longest and shortest path between the corresponding nodes in the CFG. Figure 2 shows the CDG of a Viterbi decoder which consists of nine communicating processes. In this figure, double enlined circles represent the blocking property of a communication endpoint whereas dashed lines between nodes represent communication instances.

**Definition 1 (CDG).** A communication dependency graph (CDG) is denoted by:

$$CDG := \langle V_{CDG}, E_{CDG}, E_{COM}, \tau_{CDG}, l_{CDG} \rangle$$

- $V_{CDG}$ is a set of nodes representing communication endpoints.
- $E_{CDG} \subseteq V_{CDG} \times V_{CDG}$ is a set of directed edges describing the precedence dependencies between nodes.
- The function $\tau_{CDG}(v)$: $V_{CDG} \rightarrow \{ \text{init}, \text{send}_{asycnc}, \text{receive}_{asycnc}, \text{receive}_{async} \}$ denotes the type of each node.
- $E_{COM} \subseteq V_{send} \times V_{rec}$, with $V_{send} = v \in V_{CDG}: \tau_{CDG}(v) = \text{send}_{async}$ and $V_{rec} = v \in V_{CDG}: \tau_{CDG}(v) \in \{ \text{receive}_{asycnc}, \text{receive}_{async} \}$ is a set of directed edges describing the communication.
- The edge weights are represented by the function $l_{CDG}: E_{CDG} \rightarrow N \times N$, with minimum and maximum execution time $l_{CDG}(v_1, v_2) = (c_{s_{min}}, c_{s_{max}})$ between the nodes $v_1, v_2 \in V_{CDG}$.

2.2 TDMA based scheduling

Classic time slice based approaches assign the same amount of time to each task in a fixed order. As result, a fixed task schedule as depicted in Figure 3 is created. Each task $\{v_1, ..., v_n\}$ is assigned to a time slice $\{\theta_1, ..., \theta_n\}$. The time slice cycle consisting of all time slices $\{\theta_1, ..., \theta_n\}$ is characterized by the quantum of time $Q$. If $S(\nu)$ denotes the length of time slice $\nu$, $Q$ can be calculated by

$$Q = \sum_{i=1}^{n} S(\nu_i)$$

In each cycle, each task can only be active in its assigned time slice. As result, the overall execution time $d_i$ is increased by the time in which the task can not be active due to time slice allocation of the same resource to other tasks. The interference with other tasks in each cycle is calculated by $Q - S(\nu_i)$. The number $N$ of necessary (full) cycles is denoted by the number of necessary time slices.

$$N = \left\lfloor \frac{d_i}{S(\nu_i)} \right\rfloor$$

For WCRT calculation using Equation (2), an equal time slice length is not a precondition. The interference $I_Q$ with other tasks in $Q$ is determined by [11]:

$$I_Q(\nu_i) = Q - S(\nu_i)$$

Therein the length of the time slice $\nu_i$ is subtracted from cycle length $Q$. As result, the interference in a cycle can be determined independently from other time slice lengths. On account of the elimination of this restriction, a priorization of processes can be applied. Important tasks with high resource demand can be assigned to longer time slices. In practical applications, the length of a time slice is often determined by the multiplication of an integer $n$ with a basic time slice length $b$. As result, a task is assigned to multiple time slices of length $b$. This model does neither consider the allocation of multiple time slices (with different lengths) at different positions of a cycle nor overhead due to context switching cost impact. Extensions towards these limitations are presented in Section 4.

3. DESIGN AND ANALYSIS FLOW

Due to the focus on TDMA software scheduling integration in this paper, the general design and analysis flow depicted in Fig-

![Diagram of Viterbi decoder CDG](image-url)
ure I is explained briefly with emphasis on software aspects of embedded system designs. It is based on the design flow presented in [16]. The starting point is the so-called system design which consists of communicating processes (CP) in SystemC/C++, an abstract platform description, a description of the temporal system environment, information on functional mapping and parameterization as well as requirements on the envisaged system. The first step in the design flow consists of an abstraction of the system design, mainly the functionality of the communicating processes. For this issue, the overall abstraction process is decomposed to three domains:

I. Extraction of abstract control-flow and process interaction

II. Timing analysis of communication properties

III. Timing analysis of computation properties

For determining an abstract untimed control-flow per process and the inter-process communication dependencies (I), the functional SystemC model is parsed to an intermediate abstract syntax tree (AST) representation [9]. Based on the AST, communication endpoints are identified and the internal syntax tree is collapsed to a CFG with special annotations for communication endpoints. Based on the CFG, a reachability analysis is performed for extracting precedence information on communication endpoints in each process. Further, a static inspection of loop boundaries delivers repetition numbers of CDG edges. The association of communication endpoints and communication channels is preserved during the transformation process for associating the single CDG processes to a global untimed CDG.

The determination of temporal communication properties (II) is based on the determination of communication data sizes out of the AST. Subsequently, a model-based integration of high-level communication protocols is performed for incorporating the impact of communication protocols on data sizes (e.g. overhead caused by check sums), on synchronization properties (e.g. 2-way-handshake), and on the number of resulting communication instances (e.g. protocol startup). Based on the derived data sizes temporal communication properties are determined under consideration of the interconnect properties. These temporal communication properties represent the access timing based on guaranteed exclusive access or dedicated communication resources. For integrating the impact of concurrent communication media access on the global system timing, communication scheduling analysis which is comparable to the software scheduling analysis presented in this paper will be applied.

The proceed for timing analysis of software computation properties (III) is at first to cross compile each single C++ process using the parameters of the target platform description. If communicating processes are given in SystemC, the tool PORTOS [10] is applied for deriving a single C++ file for each process. The result of compilation is one assembler file for each communicating process. Subsequently, these files are processed by a tool called splitter. This tool determines communication calls in each assembler file and decomposes the basic blocks between all directly connected communication endpoints. As result, each assembler file is split to multiple assembler files. Further, the global communication structure as representation of untimed CDG processes is extracted for integrating the execution times into the CDG determined in (I). The split assembler files are processed with the tool composer for determining the timing properties between communication endpoints of hardware components. Subsequently, obtained execution time intervals CET and the untimed CDG processes are composed using the tool composer to the timed CDG representation of the communicating processes.

Up from this point, the global timing analysis (i.e. communication analysis) for incorporating the impact of blocking communication primitives on the global timing of the system can be applied if no TDMA scheduling will be integrated in the system. After communication analysis, performance analysis based on the determined global system timing and requirement evaluation can take place. The calculated properties can then be used for an exploration of the system design. The new aspects presented in this paper are necessary to be integrated in the design flow if TDMA scheduling is utilized in the system design. The required parameters (e.g. time slice length, context switching costs, time slice assignment of processes) are extracted from the mapping/deploymet specification. Together with the statically determined CDG, all information necessary for the worst case incorporation of time slice based scheduling are available and the CDG timing adaption concerning the methodology presented in Section 4 can take place.

3.1 Communication analysis

The objective of communication analysis is the determination of the global system behavior based on the global communication and interconnection structure and local best case and worst case execution times (CET). Communication analysis is based on the synchronization properties of communication endpoints (i.e. CDG nodes \(V_{CDG}\)). A system of \(\{\text{max}, +\}\) equations [3] is formulated that describes the synchronization and timing properties of the entire system which is represented by the CDG. Due to the focus on software scheduling aspects (temporal aspects of communication considered to be 0 in this paper), communication analysis is considered simplified. Nevertheless, this does not limit the applicability of the described methods due to that they can be identically applied with consideration of temporal communication aspects. Given are two asynchronous communication instances \(C_{pre} = (s_{pre}, r_{pre})\) and \(C_{cur} = (s_{cur}, r_{cur})\) \(\in \epsilon_{COM}\) with \(\tau(r_{pre}) = \tau(r_{cur}) = r_{exec}\). Let \(P_1, P_2\) be the shortest paths between \(C_{pre}\) and \(C_{cur}\) with

\[
P_1 = \text{path}_{C}(C_{pre} \leadsto s_{cur})
\]

\[
= \begin{cases} 
  p_s : p_s = \text{path}(s_{pre} \leadsto s_{cur}) \text{ if } r_{pre} \notin p_s \\
  p_r : p_r = \text{path}(r_{pre} \leadsto s_{cur}) \text{ else }
\end{cases}
\]

\(5\)

and

\[
P_2 = \text{path}_{C}(C_{pre} \leadsto r_{cur})
\]

\[
= \begin{cases} 
  p_s : p_s = \text{path}(s_{pre} \leadsto r_{cur}) \text{ if } r_{pre} \notin p_s \\
  p_r : p_r = \text{path}(r_{pre} \leadsto r_{cur}) \text{ else }
\end{cases}
\]

\(6\)

Although temporal communication properties are not considered in this paper, the paths between \(C_{pre}\) and \(C_{cur}\) are distinguished for generality reasons towards the integration of temporal communication properties. The \((\text{minimum}/\text{maximum})\) length of a path \(p\) is denoted \(l(p)\) and is represented as tuple \((p, l)\). The control flow leaves a blocking communication endpoint after both, the control flow and the communication instance, have arrived. The \((\text{minimum}/\text{maximum})\) departure time \(t_P\) of the control-flow in \(r_{cur}\) after \(s_{pre}\) is defined as:

\[
t_P = \max(\max(P_1, P_2), \max(F, T))
\]

\(7\)
In these equations, \( \text{max} \) refers to the maximum of two integer values whereas \( \text{max}_2 \) refers to the pairwise maximum of two tuples. The minimum and maximum waiting time \( x(r_{\text{wait}}) \) in \( r_{\text{wait}} \) can be calculated by

\[
x(r_{\text{wait}}) = \max_i((t_r \ominus i)(P_2), (0, 0))
\]

\[
x(r_{\text{wait}}) = \max_i((\max(P_1, P_2), \max(T_1, T_2))
\]

\[
\oplus(P_2, T_2), (0, 0)) = \max_i((\max(P_1, P_2) - P_2, \max(T_1, T_2) - T_2), (0, 0))
\]

\[
\max_i((\max(P_1, P_2) - P_2, \max(T_1, T_2) - T_2), (0, 0)) = \max_i((\max(P_1, P_2) - P_2, \max(T_1, T_2) - T_2), (0, 0))
\]

\[
x(r_{\text{wait}}) = \max_i((l(P_1) \ominus l(P_2), (0, 0))) \quad (8)
\]

In this formulation, \( \ominus \) represents the pairwise difference of two tuples.

**Communication analysis** iteratively processes the system until an already computed system state (set of waiting times at all blocking communication endpoints) is determined. At this point, a fixpoint where the temporal system behavior recurs is detected and the analysis terminates.

### 4. INCORPORATION OF TDMA RESOURCE SCHEDULING

The analytic inclusion of time slice scheduling is based on the equations introduced in Section 2.2. These equations are valid for independent tasks which represent enclosed blocks of computation without any internal communication endpoints. A CDG represents processes, which communicate during execution using synchronous and asynchronous communication primitives. The temporal and synchronization interdependencies of interprocess communication have a major impact on WCRT of these processes. Obviously due to these restrictions, an entire CDG process can not be directly analyzed under consideration of scheduling parameters. The atomicity of CDG edges, which are encapsulated blocks of computation (without communication endpoints between the two bounding communication endpoints of the edge) provides the same prerequisites for scheduling analysis as the task model. Due to that only one edge \( e \in E_{\text{CDG}} \) per process can be executed simultaneously, scheduling transformations are applied at this level of granularity. The objective within the presented approach is the calculation of response times of CDG edges \( \text{CDG} \) of a process \( \nu \). The derived temporal edge parameters by scheduling analysis are represented the same way as edge weights using CET intervals. One difference is, that no exact bounds are derived by scheduling analysis. The derived numbers are pessimistic worst case/ best case estimations. As result, additional pessimism is inherited during later analysis steps. Nevertheless it will be shown by a case study that the pessimism is acceptable and the analysis methodology is powerful and efficient.

#### 4.1 Prerequisites

The CDG model did not yet explicitly define processes. For resolving this issue, the model is extended by processes and a mapping function that describes the association of CDG nodes to processes.

**Definition 2 (CDG processes).** The CDG definition is extended by:

- The set \( P_{\text{CDG}} \) of communicating processes

#### 4.2 Application of TDMA analysis

Context switching effects pose execution time impacts with regard to worst case assumptions in performance analysis. These penalties reduce the effective computation power in comparison to an exclusive resource usage by communicating processes. The deactivation of the previously running process and the activation overhead of the activated process including penalties (e.g. caches effects, TLB flushes) have to be considered. Regarding these issues, the terms \( o_a(v) \) and \( o_d(v) \) represent the activation and deactivation overhead of process \( v \). Under application of the previously introduced function \( \zeta \), the following Equation (9) for incorporating the context switching costs during a time slice \( \theta \) can be given. The overall overhead \( O_{\text{slot}} \) for a time slice \( \theta \) can be expressed using:

\[
O_{\text{slot}}(\theta) = o_a(\zeta(\theta)) + o_d(\zeta(\theta)) \quad (9)
\]

with \( 0 \leq i < m \) and \( j = \begin{cases} (i - 1), & \text{if } i \neq 0 \\ (m - 1), & \text{else} \end{cases} \)

This is illustrated in Figure 4.

**Figure 4: TDMA cycle analysis considering context switching penalties**

At this point, the model assumes an assignment of each process \( \nu \) to only one time slice \( \theta \) with and \( m = n \). For calculating the available computation time \( C_{\text{slot}} : P_{\text{CDG}} \rightarrow N_0 \times N_0 \) of a process per cycle, the overhead \( O_{\text{slot}} \) has to be subtracted from the available time slice length \( S_{\text{slot}} \). Due to the overhead characterization as timing interval, the resulting computation time \( C_{\text{slot}} \) is characterized as interval as well:

\[
C_{\text{slot}}(\theta) = S_{\text{slot}}(\zeta(\theta)) - O_{\text{slot}}(\theta) \quad (10)
\]

Further, the calculation of the interference with other processes in a cycle is determined as described in Equation (4) under additional consideration of context switching overhead.

\[
I_{\text{cycle}}(\zeta(\theta)) = Q - C_{\text{slot}}(\zeta(\theta)) \quad (11)
\]

The upper and lower bounds of necessary cycles (of length \( Q \)) for finishing demanded computation time \( I_{\text{CDG}}(e) \) of an edge \( e \in E_{\text{CDG}} \) in slot \( \theta \) can be calculated under application of Equation (3) and under consideration of overhead costs.

- The mapping function \( \nu_{\text{CDG}} : V_{\text{CDG}} \rightarrow P_{\text{CDG}} \) describes the affiliation of a node \( v \in V_{\text{CDG}} \) to a process \( p \in P_{\text{CDG}} \).
\[ N(\theta, e) = \left[ \frac{i_{\text{edge}}(e)}{C_{\text{slot}}(\zeta(\theta))} \right] \]  

(12)

### 4.3 Response time estimation

Up from this point, supremum \( \sup \) and infimum \( \inf \) denote the upper/lower bounds of one or more variables.

**Worst case.**

Due to the direct proportionality of the execution time of edge \( e \) to the resulting execution time and due to the varying inversely slot computation time \( C_{\text{slot}} \), to the resulting execution time, the maximum execution time \( \sup(\{\text{edge}(e)\}) \) and the maximum overhead \( \sup(\{O_{\text{slot}}\}) \) result in the following equation:

\[
t_{\text{wc}}(e) = \sup\{N(\theta, e) \cdot Q\} = \sup\{N(\theta, e) \cdot (C_{\text{slot}}(\zeta(\theta)) + I_{\text{cycle}}(\zeta(\theta)))\} + N(\theta, e) \cdot I_{\text{cycle}}(\zeta(\theta))
\]

(13)

**Best case.**

Due to guaranteed time slices, the interference in a TDMA cycle is fixed. The least critical moment is at the point of time, when best case. The least critical moment is at the point of time, when the priorization capabilities of processes. The worst case (Equation (14)) and best case (Equation (15)) calculations allow the upper/lower bounds of one or more variables.

\[
t_{\text{bc}}(e) = \inf\{i_{\text{edge}}(e) + N(\theta, e) \cdot I_{\text{cycle}}(\nu_i)\}
\]

(14)

The calculation of worst case and best case intervals \( t_{\text{wc}} \) and \( t_{\text{bc}} \) using the given equations, do not require the conditions that all time slices have the same length. The calculation of interference \( I_{\text{cycle}} \) and cycle length \( Q \) already consider different time slice lengths.

### 4.4 Extension of TDMA parameters

The assignment of each process to only one time slice limits the priorization capabilities of processes. The worst case (Equation (13)) and best case (Equation (14)) calculations allow the analysis of different time slice lengths and the analysis of process assignment to multiple time slices if they are next to each other. In this case, the assumption is that they were considered as long time slice (without consideration of multiple context switching overheads). But if processes with very short execution times but frequent activations exist, good resource utilization would benefit from multiple time slices at different positions in the cycle. Figure 5 depicts a cycle with allocation of multiple slots to single processes at different positions. The cycle of length \( Q \) is divided to 7 time slices which are assigned to the process set \( P_{\text{CDDG}}^{\nu} \) with the elements \( \{\nu_0, \nu_1, \nu_2, \nu_3\} \). As example, process \( \nu_i \) is assigned to 3 time slices \( \{\theta_0, \theta_1, \theta_2\} \). For the following analysis, an additional utility function \( C_{\text{red}} \) is defined: \( C_{\text{red}} : \Theta \times P_{\text{CDDG}} \rightarrow \mathbb{N}_0 \)

\[
C_{\text{red}}(\nu, \theta) = \begin{cases} 
C_{\text{slot}}(\zeta(\theta)), & \text{if } \zeta(\theta) = \nu \\
0, & \text{else}
\end{cases}
\]

(15)

This utility function returns the usable length \( C_{\text{slot}}(\zeta(\theta)) \) (under consideration of overhead) of a slot \( \theta \) if a process \( \nu \) is assigned to it and else 0. It is applied for determining the sum of all time slices \( C_{\text{cycle}}(\nu) \) of a process \( \nu \) in cycle \( Q \) under consideration of overhead costs:

\[
C_{\text{cycle}}(\nu) = \sum_{j=0}^{m} C_{\text{red}}(\nu, \theta_j)
\]

(16)

The calculation of interference (based on Equation (11)) is formulated as follows:

\[
I_{\text{cycle}}(\nu) = Q - C_{\text{cycle}}(\nu)
\]

(17)

For determining the bounds of the maximum cycle number, the available computation time has to be considered.

\[
N(\nu, e) = \left[ \frac{i_{\text{edge}}(e)}{C_{\text{cycle}}(\nu)} \right]
\]

(18)

Under application of this formulation of \( N(\nu, e) \) instead of \( N(\theta, e) \), Equations (13) and (14) can be utilized for calculating best case and worst case timing properties under consideration of process assignment to multiple time slices.

### 4.5 Reduction of pessimism

The previously made assumptions are quite conservative according the consideration of the last cycle. For reducing the resulting intervals, the time slice parameters and process assignment have to be examined more closely. For determining more tight (best/ worst case) intervals, the consideration of the starting point and the associated computation time during integer executed cycles is determined. This difference \( i_{\text{cycle}}(\nu) \) can be computed using Equation (19).

\[
i_{\text{cycle}}(\nu) = \{l_{\text{edge}}(e) - C_{\text{cycle}}(\nu) \cdot (N(\nu, e) - 1)\}
\]

(19)

The determination of the remaining execution time bounds of edge \( e \) in the last cycle enables an exact analysis of the resulting response times under consideration of scheduling. The application of static analysis does not determine the point in time, when the initial computation of an edge starts. As result, any starting point in a cycle is possible. The worst case assumption, that a process is activated directly after the end of its time slice, is retained under consideration of all time slices a process is assigned to.
The worst case starting point at the end of a time slice in a cycle is the one for which the largest overall remaining computation time of a process \( \nu \) results. The objective for calculating the worst case is now the determination of the time slice, after which the worst case starting point is located. For deriving the best case, the starting point of the time slice with the least resulting interference is determined, so that the least response time results. Both cases are based on Equation (19) for calculating the already spent execution time of edge \( e \) in the previous \((N - 1)\) cycles. The response time of an edge \( e \) depends on the length of the time slices as well as on the interference between them. A utility function is defined for denoting the overhead of a process \( \nu \) in dependency on a time slice \( \theta \). If a process \( \nu \) is assigned to \( \theta \), \( O_{\text{slot}} \) from Equation (10) is returned. Otherwise, 0 is returned.

\[
O(\nu, \theta) = \begin{cases} O_{\text{slot}}(\theta), & \text{if } \zeta(\theta) \neq \nu \\ 0, & \text{else} \end{cases} \quad (20)
\]

Equation (21) redefines the function for calculating the interference \( I_r : P_{\text{CDG}} \times \Theta \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \) for a process. This function determines the minimum and maximum required time from a starting time slice \( \theta_j \) to finish computation of edge \( e \) under consideration of overhead costs. For determining interference over the bounds of a time slice, the modulo variable \( K \) is used for indexing.

\[
I_r(\nu, \theta_j) = \sum_{l = j}^{R} S_{\text{rel}}(\nu, \theta_K), \quad \text{if } \zeta(\theta_K) \neq \nu
\]

\[
I_r(\nu, \theta_j) = O(\nu, \theta_K), \quad \text{else}
\]

with \( K = l \mod m \) \quad (21)

As illustration, Figure 4 depicts a cycle of length \( Q \) including time slices of a processes \( \nu \), and interfering cycle parts with other processes. \( S_1 \) and \( S_2 \) are possible worst case starting points. For incorporating the starting point \( S_2 \), summing up interferences over the bounds of the cycle is necessary. The termination condition \( R : \Theta \times \mathbb{N}_0 \rightarrow \mathbb{N}_0 \) calculates the index of the time slice that is necessary for providing enough computation time for process \( \nu \) to terminate calculation of \( l_e \) by summing up the difference between time slice length and overhead.

The worst case does not incorporate the starting time slice \( \theta_j \) due to the assumption that the process is activated directly after this time slice. Additionally, the maximum remaining computation time in the last cycle is considered.

\[
R_{\text{max}}^{\text{rel}} = \inf_{j < k \leq j + 2} \left\{ \sup \left\{ \sum_{l = j + 1}^{k+1} C_{\text{rel}}(\nu_l, \theta_K) \geq l_e(e) \right\} \right\}
\]

with \( K = l \mod m \) \quad (22)

The best case incorporates time slice length \( \theta_j \) and the minimum execution time of process \( \nu_l \) in the last cycle.

\[
R_{\text{min}}^{\text{rel}} = \inf_{j < k \leq j + 2} \left\{ \inf \left\{ \sum_{l = j}^{k} C_{\text{rel}}(\nu_l, \theta_K) \geq l_r(e) \right\} \right\}
\]

with \( K = l \mod m \) \quad (23)

The formulation of Equation (21) allows the determination of interference times until finishing execution of \( l_e \) starting at all possible time slices. The function \( L : P_{\text{CDG}} \rightarrow \mathbb{N}_0 \times \mathbb{N}_0 \) determines the minimum and maximum interference time for the determination of \( l_e \) in the last cycle. For solving this issue, the interference of a process \( \nu \) for all relevant starting points in a cycle is determined using Equation (22) and (23).

\[
L(\nu) = I_r(\nu, \theta_j), \forall j : \zeta(\theta_j) = \nu \quad (24)
\]

By adding up this interference time with the execution time of edge \( e \) and the interference time of the completely run through \((N - 1)\) cycles, the response time \( l_e(e) \) of edge \( e \) can be calculated for best and worst case.

\[
t(e) = l_{\text{cdg}}(e) + (N(\nu, e) - 1) \cdot I_{\text{cycle}}(\nu) + L(\nu)
\]

with \( e = (v_1, v_2) \in E_{\text{CDG}} \land g(v_1) = g(v_2) = \nu \) \quad (25)

The determined worst case response time of edge \( e \) equals the worst case of a round robin strategy with the same cycle configuration \((\Theta, S_{\text{rel}}(\nu_1, \theta_j))\) if no additional knowledge on activation and deactivation of the single processes is given [11].

4.6 Dynamic time slices

The integration of dynamic system aspects can be represented by additional dynamically assigned time slices like \( \theta_c \) in Figure 6. Dynamic time slices are not exclusively assigned to a process. The assignment is realized with an on-line fixed priority scheduling mechanism. When a dynamic time slice is active, the runnable process with the highest priority can use this time slice for computation. A function \( \omega(\psi, \nu) \) is introduced which returns the priority \( \rho \) of a process \( \nu \) on a resource \( \psi \) (or \( -1 \) if the process is not mapped to the \( \psi \)).

Further, a utility function \( \gamma(\nu) = \psi \) describes the mapping of a process \( \nu \) to a resource \( \psi \).

\[
\omega(\psi, \nu) = \begin{cases} \rho, & \text{if } \gamma(\nu) = \psi \\ 1, & \text{else} \end{cases} \quad (26)
\]

For each valid mapping of a process \( \nu \) on a resource \( \psi \),

\[
\omega(\psi, \nu) \geq 0 \quad (27)
\]

If dynamic time slice scheduling is non-preemptive, each active process can use the resource until it terminates or the time slice runs out. If the process is in a blocked state or finished with its computation, the runnable process with the highest priority is chosen by the scheduler. If the scheduling strategy is preemptive, the process could be additionally preempted by higher priority processes that become runnable.

Under consideration of fixed priority scheduling, each process could use the dynamic time slice for its computation in the best case consideration of preemptive and non-preemptive behavior. Therefore it is assumed that a dynamic time slice is assigned to each examined process.

When analyzing the worst case behavior of fixed priority pre-emptive on a resource in a static way, only the process with the
highest priority can use the dynamic time slice in every cycle. The process \( \nu_i \) assigned with the highest priority on a resource \( \phi \) is specified by

\[
\inf \{ \omega(\phi, \nu_i) \} \geq 0
\]  

(28)

During worst case analysis, a dynamic time slice is assigned to an ascertained process \( \nu \) with the utility function \( \zeta(\theta) \). In the example given in Figure 6, \( \zeta(\theta) = \nu_i \) if \( i \) is returned by Equation 28.

Each other process depends on the state of the highest priority process. Due to the static analysis proceed, the time slice is not assigned to them in the worst case calculation. If the scheduling is not preemptive it can not be guaranteed that the dynamic time slice is used by one particular process. As result, the calculation of the worst case response time does not consider the assignment of the time slice to any process.

Further reduction of pessimism of dynamic TDMA aspects (e.g. towards Round-Robin analysis) is limited by the static proceed proposed in this paper and is motivation for current work which integrates interference analysis during communication analysis (CA).

Based on the global timing behavior determined by CA, major reductions of pessimism can be achieved and very tight bounds of static and dynamic TDMA scheduling timing properties can be calculated. Although the interference analysis is based on the formulations presented in this paper, the analysis enhancements require major extensions of the formulation of communication analysis, which is out of scope of this paper. Nevertheless, the proposed analysis extensions in this paper are very applicable which is shown in a promising case study presented in the following section.

5. CASE STUDY

As an example, the described methods were applied to a SystemC description of a JPEG decoder and a Viterbi decoder provided by Infineon Technologies. Both decoders consist of several parallel processes, communicating via signals with non-blocking send and blocking receive synchronization properties. Both applications were mapped on a general purpose platform depicted in Figure 7. The memory processes \( \text{vitmem}_w (P_5), \text{bytemem}_w \) and \( \text{bytemem}_r (P_5) \) are mapped to dual port SRAM memories with latencies \((d_0, d_1, d_2) \) of 60 ns. The processes with the main functionalities \((P_1, P_2, P_3, P_5, P_9, P_{10}, P_{11}) \) were mapped to microprocessor cores as depicted in Figure 7. CPU1 is a PPC 750 with 233 MHz. CPU2 is a PPC 750 with 100 MHz. Due to simplicity and comparability, the basic mappings of processes to processors stay fixed in the analyzed configurations. Of course, the mappings, the number, parameters and heterogeneity of the microprocessors can be exploited during exploration [8]. According to these basic mappings, the CDGs of both decoders were obtained using the bottom-up analysis framework [16] and are depicted in Figure 8 and Figure 2. The time unit of annotated execution times is ns. These CDGs provide the basis for the formal scheduling and performance analysis implemented in the SysXplorer tool.

![Figure 7: Platform model and process mapping](image)

![Figure 8: JPEG decoder CDG](image)

The assumed activation and deactivation overhead \( a_0(\nu) \) and \( a_d(\nu) \) (in ns) of each process \( \nu \in P_{CDG} \) on the associated computation resource are depicted in the table in Figure 9.

<table>
<thead>
<tr>
<th>( \nu )</th>
<th>vit_bwd</th>
<th>vit_fwd</th>
<th>idct</th>
<th>ileh</th>
<th>squash</th>
<th>izigzag</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_0(\nu) ) [ns]</td>
<td>20.37</td>
<td>14.21</td>
<td>19.28</td>
<td>4.5</td>
<td>31.50</td>
<td>29.55</td>
</tr>
<tr>
<td>( a_d(\nu) ) [ns]</td>
<td>5.12</td>
<td>3.7</td>
<td>4.62</td>
<td>9.17</td>
<td>6.14</td>
<td>6.14</td>
</tr>
</tbody>
</table>

![Figure 9: Process activation and deactivation overheads](image)

The parameters depicted in Figure 10 were used for analyzing six different scenario configurations. The entries in the table characterize [CPU #/time slice #/time slice length (ns)].

![Figure 10: TDMA scheduling parameterization](image)

Configuration 1 uses one time slice per process and equal time slices of length 1000 ns. Configuration 2 uses one time slice per process and different time slice sizes. Configuration 3 uses potentially multiple time slices per process with equal length. The Configuration 4, 5, and 6 respectively use potentially multiple time slices per process with different sizes. Further, Configurations 5 and 6 include a dynamic time slice per resource of length 250 ns. The prioritization of the processes on the resources is shown in Figure 11. In Configuration 5, a preemptive policy for the dynamic time slice assignment is utilized, whereas Configuration 6 utilizes a non-preemptive policy.
For evaluating the impact of scheduling parameters, the maximum guaranteed input data rate \( l(input \rightarrow output) \) for computing 16 packets (Viterbi) respectively 64 packets (JPEG) with the maximum guaranteed input rate were determined for both applications and all configurations using the formal analysis approach (FA). For validating the latency analysis results, a framework for generating abstract SystemC simulation models from CDG models was used [18]. A library was implemented for allowing suspension and re-activation of SystemC processes by inserting a transparent layer between the global SystemC clock and \texttt{request2run(sc\_time)} calls which replace \texttt{wait(sc\_time)} calls.

The analysis results (FA) and a comparison with timing properties derived during simulation (SIM) is shown in the table in Figure 12. The depicted simulation-based results represent the maximal occurred timing values after 10,000 simulation runs of both decoders. A further result for all analyzed configurations is that they were not continuously desynchronizing due to too high input rates of single processes. This means that finite buffers can be utilized for a guaranteed prevention of data loss.

The maximum guaranteed input data rate of the Viterbi example due to the additional dynamic time slice. As result, the resulting input rate numbers are equal for both configurations. The predicted end-to-end latency has its least value for Configuration 5 due to that the second Viterbi stage (\( P_2 \)) has the highest priority on \( CPU_2 \). Due to the preemptive policy, \( P_2 \) can always use the dynamic time slice which means twice the time slice length in a cycle with an increase of 2250 ns. In contrast, the end-to-end latency increases between Configurations 4 and 6 due to that the static analysis cannot guarantee that \( P_2 \) is always able to use the time slice in the non-preemptive configuration. So the increased cycle length \( Q \) leads to an increase of the end-to-end latency. But simulation results show, that this estimation is rather pessimistic. The monitored worst case results for the end-to-end latency of Configuration 6 are around 9 % below the monitoring results of Configuration 4.

The JPEG decoder input data rate of Configuration 1 is limited by the inverse run-length encoding stage which is implemented in \( P_9 \). This is mainly caused by the high computational effort represented by the latency of edge (\( R_5, S_5 \)). The guaranteed JPEG decoder input data rate has the smallest value in Configuration 2 due to the short time slice for process \( P_{10} \) and the long one for process \( P_4 \) on \( CPU_1 \), as well as high context switching costs. Especially the worst case execution time inf\((l(R_4, R_5)) = 561\) ns in a cycle of length \( Q = 3500\) ns (WCRT of 13981 ns) effects the input data rate of the inverse quantization stage (in comparison to e.g. \( P_9 \)). As result, process \( P_{10} \) causes the low input rate of the entire JPEG decoder. Further, the end-to-end latency for the 64 packets is the highest because of the low input datarate and the comparably slow pipeline stages \( P_6 \) and \( P_{10} \).

The time slice parameters of Configuration 3 with a reduced cycle length \( Q \) for \( CPU_2 \) leads to an increased input datarate in comparison to Configuration 1. The slight increase of the end-to-end latency is caused by higher response times of the processes \( P_3 \) and \( P_{10} \), which suffer from shorter assigned time slice length with regard to the overall cycle length. The better overall performance of Configuration 3 in comparison to Configuration 4 is due to a better time slice balancing. The limiting factor of the JPEG decoder input data rate in Configuration 4 is the inverse quantization (\( P_{10} \)) which suffers from the longer overall cycle length \( Q \) in comparison to Configuration 3.

The JPEG decoder performance benefits from the preemptive dynamic time slice added in Configurations 5. Due to that process \( P_{10} \) has the highest priority on \( CPU_1 \), the maximal guaranteed data rate increases. Nevertheless, the end-to-end latency increases due to increased overall cycle lengths \( Q \). The input data rate drops about 23 % in Configuration 6 because of the static analysis properties of non-preemptive scheduling of \( P_{10} \) which were identified in Section 4.6.

All in all, the comparison between formal analysis results and simulation-based results validates the proposed analysis approach. Whereas abstract simulation of one system configuration took about 13 minutes, the overall time of the formal analysis (communication analysis + performance analysis) was around 6.2 s on an AMD Athlon 3400+. Further, the presented numbers show an acceptable degree of potential overestimation with regard to the simulation results which do not guarantee to cover corner cases.

### Figure 12: Performance analysis results

<table>
<thead>
<tr>
<th>Conf.</th>
<th>Viterbi max((d_1^{-1}))</th>
<th>JPEG max((d_1^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FA [KB/s]</td>
<td>FA [ms]</td>
</tr>
<tr>
<td>1</td>
<td>169.22</td>
<td>3.29 : 2.98</td>
</tr>
<tr>
<td>2</td>
<td>176.36</td>
<td>4.55 : 4.20</td>
</tr>
<tr>
<td>3</td>
<td>267.32</td>
<td>2.50 : 2.21</td>
</tr>
<tr>
<td>4</td>
<td>417.19</td>
<td>2.19 : 2.01</td>
</tr>
<tr>
<td>5</td>
<td>389.31</td>
<td>1.86 : 1.58</td>
</tr>
<tr>
<td>6</td>
<td>389.31</td>
<td>2.23 : 1.83</td>
</tr>
</tbody>
</table>

### 6. CONCLUSION

In this paper, an approach for bottom-up analysis of concurrent communicating processes under consideration of time slice based software scheduling has been presented. Unlike other approaches in this area, the internal control-flow of the processes and complex interaction patterns and protocols are considered during analysis. The general analysis approach is based on a method for determin-
ing communication instances that cause an implicit synchronization of processes by timing properties and for incorporating this synchronization behavior in global system timing analysis. Based on this automated bottom-up methodology for analyzing processes that are exclusively mapped on computation resources, a methodology for incorporating time slice based software scheduling has been proposed. This analysis is performed on abstract model level for being able to efficiently analyze and optimize a systems configuration. In addition to the basic incorporation of time slice based scheduling, several analysis extensions were proposed for being able to incorporate multiple assignments, dynamic time slices, and platform specific parameters like for example context switching overhead penalties and the allocation of multiple time slices with different lengths to one software process. Additionally, the described parameter analysis can be directly applied for deriving temporal communication properties, opening a broad application area.

The described methods have been applied to an example, the SystemC description of a multimedia subsystem consisting of 11 communicating processes. In the example, the existence of synchronization between processes, the maximum guaranteed input data rate, and the worst case end-to-end latency under consideration of time slice based software scheduling were determined.

The comparison of the analysis results with an abstract simulation of the formal system model shows that the pessimism inherited by the interference estimation is acceptable low. Furthermore, the formal analysis has a speed-up of more than 100 in comparison to the abstract simulation of the system which can not even guarantee corner case coverage. A concrete simulation of such a system implementation, even as TLM model, would take orders of magnitude longer.

The evolved methods provide a powerful basis for formal timing analysis of embedded systems and enable a very fast evaluation of many different system configurations towards an exploration of the design space of the system configuration.

Extensions addressed in Future will be towards an analysis of dynamic scheduling policies like round robin or FPP scheduling as well as a reduction of pessimism by reducing the time slice candidates for interval analysis. These extensions will be based on an integration of interference analysis during communication analysis.

7. REFERENCES


