Control-Flow Aware Communication and Conflict Analysis of Parallel Processes*

Axel Siebenborn[1], Alexander Viehl[1], Oliver Bringmann[1], Wolfgang Rosenstiel[1,2]

[1]FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
[siebenbo,viehl,bringman]@fzi.de

[2]Universität Tübingen
Sand 13
72076 Tübingen, Germany
rosenstiel@informatik.uni-tuebingen.de

Abstract

In this paper, we present an approach for control-flow aware communication and conflict analysis of systems of parallel communicating processes. This approach allows to determine the global timing behavior of such a system and to detect communication that might produce conflicts on shared communication resources. Furthermore, we show the incorporation of temporal environment models in order to analyze their influence on the system behavior. Based on the determined conflicts, an automated allocation and binding approach for shared resources to resolve potential access conflicts is proposed. All analysis steps can be performed starting with a TLM SystemC model of the entire system without any need for user interaction. Finally, a SystemC model of a Viterbi decoder is used as case study to demonstrate the capability of our approach.

I. Introduction

The complexity of systems is steadily increasing and already today, there are designs integrating more than one processor core with different instruction sets, various interfaces, memories and specialized hardware onto a single chip. Already today, the interconnection and communication of the components is a key issue in the design of such systems. Many embedded systems are composed of different processors and communicate via shared resources. Embedded systems interact with their environment under more or less hard timing constraints. The communication in such systems has a strong influence on the global timing behavior. Methods are needed to analyze timing properties, average throughput, as well as worst case response time (WCRT). Currently, analysis of such systems is usually performed by simulation with the issue of incomplete coverage. Formal methods are needed to identify the corner cases.

This paper describes new methods for formal analysis of safety-critical embedded applications with hard time constraints executed on multiple processors. These systems are usually statically scheduled and do not allow task preemption. However, a static schedule can be included as a process by our analysis methods. Unlike other approaches that operate on task graphs, our analysis approach can be applied directly on a given SystemC model as functional representation of the entire system without any user interaction.The influence of blocking communication on the timing of the system is considered by identifying synchronizing communication and by determining the time a process is stalled at such a communication. With this knowledge, the response time of the system can be determined. Whereas the environment of the system plays an important role the integration of temporal environment models into our analysis methods is another important issue. Conflicting access to shared communication resources imposes a major problem on the determination of the timing of a system. This paper presents a new approach that allows the determination of potential conflicts on shared ressources and the avoidance by a conflict free binding of communication to ressources. A major part of this paper will be the detailed demonstration of our methods on the model of a Viterbi decoder.

The following section will review existing approaches from literature. In Section III, we present our method for communication analysis. In Section IV, the modeling of the system environment is discussed including the well established event models. In Section V, we present a method to determine conflicting access to shared resources and the conflict-free binding of abstract communication to communication resources. Finally, we demonstrate the applicability of our methods on the SystemC model of a Viterbi decoder in Section VI.

II. Related Work

Existing approaches on communication analysis can be distinguished depending on the underlying model of computation and the addressed application domain. Many approaches introduced in the area of distributed software systems for modeling and analysing of concurrent, communicating systems are based on Petri nets, process algebras and communicating automata. Petri nets provide a universal technique for modeling concurrent systems. While early approaches are mainly simulation-based, modern approaches use efficient analysis techniques which are able to cope with timed Petri nets [8, 14]. However, the applied communication types have to be modeled implicitly. Otherwise, communication analysis is performed without knowledge on concrete communication types, the control-flow of the processes and communication channels using corresponding places and transitions. A promising method for analyzing worst case timing behavior of concurrent systems is based on communicating automata [12], an extension to timed automata [1, 2]. However, these methods rely on totally synchronous communication. This quite restrictive model reduces the possible degree of concurrency and is not realistic for communication with buffering and latencies of the communication channel.

Several approaches exist tailored to the requirements of real-time systems. Here, we have to distinguish between a parallel execution of the specified processes on dedicated resources and a sequential execution of the processes on a single processor us-

*This work was partially supported by the BMBF project VISION under grant 01M3078B and by the DFG project "Communication Analysis for Network-on-Chip" under grant BR 2321/1-1

1-4244-0630-7/07/$20.00 ©2007 IEEE.

ing task scheduling. The combination of both aspects are tackled by an approach [15] addressing parallel and distributed real-time systems, where all specified processes have to be mapped onto several processing elements. However, they abstract from the internal processes and operate on an acyclic task graph. This model is based on the assumption that each task has a statically determined execution time and each task starts its execution once all input signals are available. The main drawback of this model is the missing support of conditional control structures. Therefore, the basic task graph model is extended by adding control dependencies [9, 13]. Due to the acyclic structure, the modeling of different communication protocols and data-dependent loops is not allowed. Recent approaches address general hardware/software platforms, with respect to user-specified I/O event models [6, 4]. The main issues are caused due to the focus on the I/O stream behavior and the missing consideration of the control-flow of the processes.

One of the first approaches that analyze the timing behavior of a system of communicating processes considering the control-flow is presented in [5]. This approach is able to handle loops by a bottom-up evaluation of communicating loop bodies in the loop hierarchy. It is not applicable for systems with data dependent loops and branches. In addition, it relies on a correct specified system without data loss and dead locks. In [10] an analysis technique has been presented that combines methods for WCET analysis with an approach for communication analysis for hardware synthesis [3]. This approach has been extended to consider latencies on communication channels and to determine possible conflicts on shared communication resources [11]. In this paper we present our approach for including temporal environment models in system analysis. As a case study, we apply our methodology on a SystemC model of a Viterbi decoder. Furthermore, a novel approach for back-annotation of determined execution times on processor models in order to improve simulation accuracy is explained. We compare the results of our formal analysis approach with simulation based results.

III. Architectural Exploration

Our approach for communication analysis allows the validation of the real-time behavior of parallel processes. These processes can represent software running on processors or hardware processes. In this paper, we will focus on software processes. Hereby two methods that work in two different problem domains are combined: (1) static timing analysis and (2) communication analysis [10]. Static timing analysis handles code sequences with control structures, including loops with bounded iteration counts. On the other hand, for communication analysis only communication and the timing behavior between the communication nodes are of interest.

Timing and communication analysis is performed according to a given architecture. An example of an architectural specification is depicted in Figure 1. We use the architectural description for mapping of functionality and communication to resources and as constraint to identify potential conflicts on shared resources like e.g. busses and memories. A modification of the architecture, the mapping or the functionality allows architectural exploration based on our analysis methods.

Accordingly, the first step is to decompose the problem for the two analysis domains. For control-flow aware communication analysis, only information on the communication and the temporal and causal behavior between them is of interest. These information can be compactly represented as a communication dependency graph *CDG* [3]. The nodes in the *CDG* represent communication endpoints, i.e. sending and receiving events. The graph contains two types of edges: Edges that represent the flattened control-flow between communication endpoints of a process, and edges that represent the communication.

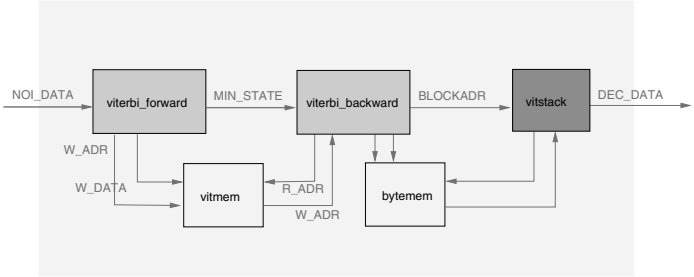

Fig. 1. Internal structure of Viterbi decoder

A communication dependency graph *CDG* is a directed, cyclic graph, that can be constructed based on the control-flow graph *CFG* of each process. The edges e_{com} are given by the communication. Edges e_{cdg} represent the control-flow between two communication points in the *CFG*. An edge e_{cdg} between two nodes in the *CDG* exists, if there exists a path in the *CFG* between the corresponding basic blocks. The latencies c_{min}, c_{max} are attributed to each edge e_{cdg}, which represent the execution time of the longest and shortest path between the corresponding nodes in the control-flow graph. These latencies are determined by static timing analysis [7].

Based on the communication dependency graph, a condition can be formulated that has to be fulfilled for a synchronization point, that represents a communication synchronizing the control-flow of the two communicating processes. Each communication pair (v_s, v_r) is a potential candidate for a synchronization point. Let us assume without loss of generality that the receiving node v_r blocks execution, whereas the sender v_s does not block. However, a blocking sender and a non-blocking receiver are handled in the same way. An obvious condition for synchronization is that the blocking communication partner is reached before or at the same time as the non-blocking partner.

The set of nodes on the shortest path from v_1 to v_2 is denoted by $path_{min}(v_1 \rightsquigarrow v_2)$. In the same way, the set of nodes on the longest acyclic path is denoted by $path_{max}(v_1 \rightsquigarrow v_2)$. The function $L(p)$ calculates the latency l of a path p.

A communication $C = (v_s, v_r)$ with $(v_s, v_r) \in E_{COM}$ is a synchronization point *SP* if

$$L(path_{max}(v_{sync} \rightsquigarrow v_r)) \leq L(path_{min}(v'_{sync} \rightsquigarrow v_s)) \quad \forall (v_{sync}, v'_{sync}) \in SP_{pre}((v_s, v_r))$$

Herein the set $SP_{pre}((v_s, v_r))$ refers to all previous *SP* from which the communication nodes v_s or v_r can be reached directly, without passing an other *SP*. The set of initialization nodes I of the processes is considered as initial *SP* for the recursive analysis of the system. This synchronization condition represents a criterion for verifying existing synchronization points. By introducing slack variables $\underline{x}_i$ and $\overline{x}_i$ for each communication C_i that represent the number of minimal and maximal wait cycles at the blocking communication endpoint, the condition can be formulated as two synchronization equations(*SE*).

$$\underline{SE}(C_{isync} \rightsquigarrow C_i) : L(path_{max}(v_{isync} \rightsquigarrow v_s)) = L(path_{min}(v'_{isync} \rightsquigarrow v_r)) + \overline{x}_i \quad (1)$$

$$\overline{SE}(C_{isync} \rightsquigarrow C_i) : L(path_{min}(v_{isync} \rightsquigarrow v_s)) = L(path_{max}(v'_{isync} \rightsquigarrow v_r)) + \underline{x}_i \quad (2)$$

The condition is fulfilled, if the slack variable has a value greater than or equal to zero. For the whole system, a system of equations can be set up. Synchronization points are determined using an iterative algorithm. Communication with a slack less than zero will not be considered for the next iteration. The algorithm stops when the values of all variables in the current system of equations recur. The determined synchronization points and the slack variables can be used to determine the WCRT of a system. The influence of parallel processes to a certain process of the system is embodied in the slack variables. To determine the worst-case turn-around time of a process, only the path latencies of this processes together with the slack variables have to be considered.

Back-Annotation of Execution Times The determined latencies that are annotated to the edges in the *CDG* can also be used for a timed simulation of the original untimed description. This procedure allows a time-enhanced simulation of the original functional model concerning the target platform. The execution times of the paths that were estimated with the analysis tool GROMIT[7] can be annotated in the SystemC description by `wait`-instructions. The mapping between *CDG* and source code is performed using debug information and the consideration of structural elements like branches and loops. This way, it is possible to include the temporal behavior of the system in simulation. Figure 2 presents an example for the annotation in SystemC.

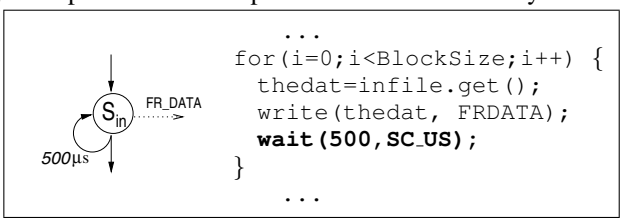

Fig. 2. Timing annotation for simulation in SystemC

IV. Environment Models

The environment of a system defines the occurrence of external events relating the system. The knowledge of the intended temporal environment of a system allows the verification of constraints and requirements given by the system specification. The environment can be modeled by additional processes in the *CDG*. This way it is possible to model common atomic event models. Moreover, additional processes provide a powerful way to describe complex sequences of events.

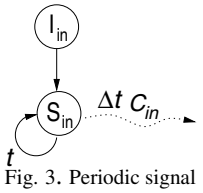

Fig. 3. Periodic signal

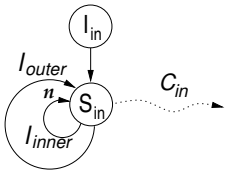

Fig. 4. Burst signal

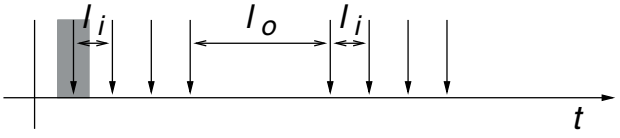

Fig. 5. Burst scenario latency of the inner loop l_i and the outer loop l_o

Figure 3 shows a process sending a periodic signal with period t and jitter Δt. Figure 4 depicts the modeling of signal bursts according to the burst scenario shown in Figure 5. The latency of the inner loop l_{inner} describes the interval between the signals in a burst, the loop counter n denotes the number of signals of a burst and the latency l_{outer} denotes the time between two bursts. These atomic event models can be combined to model a complex environment. Incorporating receive nodes, a real interaction between a system and its environment can be modeled. Figure 6 shows an example for modeling a simple reactive environment.

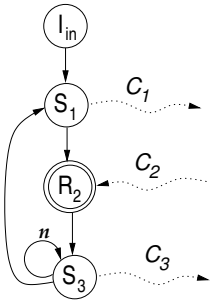

Fig. 6. Reactive environment model

V. Access to Shared Resources

The information gained during communication analysis can be used to determine conflicts on shared communication resources, like e.g. busses. The determined synchronization points relate the parallel processes and allow to put all communication into a temporal order. The relative minimum and maximum latencies of the

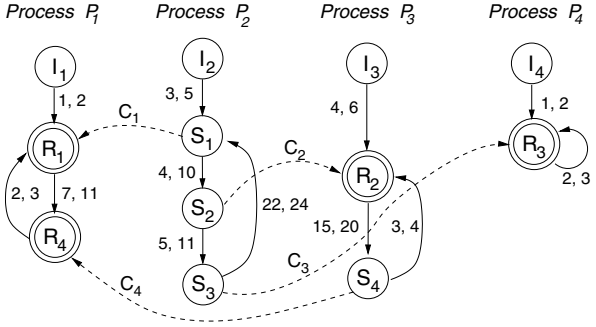

Fig. 7. CDG with possible conflicts on communication resources

path between communication nodes, together with the determined slack variables allow to calculate absolute time intervals for each communication. Therefore, synchronization points provide absolute timing information, but communication interdependencies in the entire process system have to be detected. Hence, considering only absolute time intervals to determine possible conflicts on communication resources is not sufficient. This problem is illustrated in Figure 7 and Figure 8. For example, the communication C_2 and C_3 in Figure 7 start their execution during the time intervals $[L(path_{min}(I_2 \rightsquigarrow S_2)), L(path_{max}(I_2 \rightsquigarrow S_2))] = [7, 15]$ and $[L(path_{min}(I_2 \rightsquigarrow S_3)), L(path_{max}(I_2 \rightsquigarrow S_3))] = [12, 26]$. These time intervals are depicted in Figure 8. It has to be mentioned that these intervals are related to the starting times of communication, not to the duration. Even if these intervals overlap, a look at the control-flow of process P_2 shows obviously that these communication will not take place at the same time. This means that it is indispensable to consider the order of the different communication. This order is determined using the control-flow of the processes. Based on the synchronization points, that relate the parallel processes, a global temporal order can be determined. This order can be represented by a so called communica-

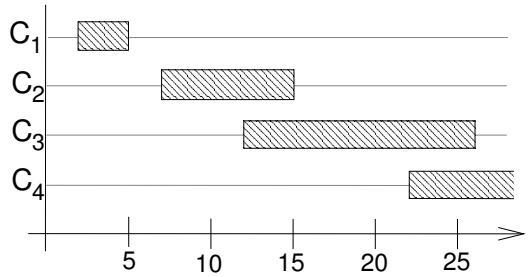

Fig. 8. Starting time intervals of the communication in Figure 7

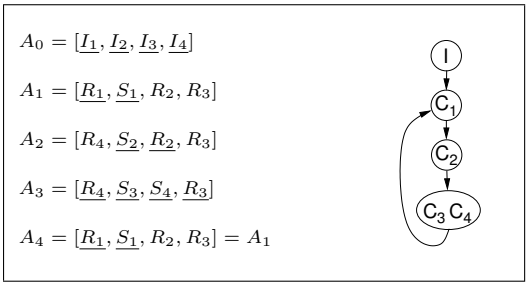

Fig. 10. Communication schedule graph for the example in Figure 7

tion scheduling graph *(CSG)*: Potentially parallel communication are grouped to a single node; edges denote the temporal order of the communication.

In the first step, only the order of communication endpoints in the *CDG* is considered. During this step, path latencies are ignored. In principle, the algorithm is shown in Figure 9. It starts with the initial synchronization points as the first node in the communication schedule graph. The current communication endpoints for each process, i.e. the system state, are stored in a vector A. The algorithm terminates, if the sequence of A's is recurring. To test this criterion, A has to be stored in a set S for each iteration of the algorithm. The current vector A_{cur} is stored on a stack $\mathbf{V}$, together with the current node v of the *CSG*. The function $update()$ determines a set $\mathbf{A}$ of system states A, based on a previous system state A_{prev}. Since the control-flow graph may include data dependent branches, $update()$ returns a set of state vectors $\mathbf{A}$, one for each branch in the control-flow graph. State vectors that are already contained in S will not be considered for further iteration steps. Figure 10 shows the *CSG* for the example

```
A_0 := [I_1, I_2, ..., I_n];
S := {A_0};
V.push(⟨v_0, A_0⟩);
while V ≠ ∅ begin
  ⟨v_prev, A_prev⟩ := V.pop();
  A := update(A_prev);
  for each A_cur ∈ A
    if A_cur ∉ S begin
      v_cur := ∅;
      for each c_send, c_rec ∈ A_cur with ⟨c_send, c_rec⟩ = e ∈ E_com then
        v_cur = v_cur ∪ {e};
      create edge e = ⟨v_prev, v_cur⟩;
      V.push(⟨v_cur, A_cur⟩);
      S := S ∪ {A_cur};
    end;
  end;
end;
```
Fig. 9. Construction of communication schedule graph

in Figure 7. Since there are no data dependent branches in this example, *update()* returns just one state vector for each iteration of the main loop. The underlined entries mark communication, where both communication partners are contained in the vector. Since the state vector A_3 contains the two communication C_3, C_4, these communication form one node in the *CSG* and are possibly executed at the same time. In the second step, with the knowledge of path latencies and based on the determined synchronization points potentially parallel communication can be checked, if their time intervals are overlapping. Since C_2 is a synchronization point in the example, the control-flow will leave S_2 and R_2 at the same time, so only the path latencies $S_2 \rightsquigarrow S_3$ and $R_2 \rightsquigarrow S_4$ have to be considered. Since the time intervals for these paths do not overlap ($[5, 11] \cap [15, 20] = \emptyset$), C_3 and C_4 are never executed at the same time and accordingly the *CSG* can be reordered.

Resource Allocation and Binding In the following, a method for allocating a minimal number of communication resources, e.g. a bus, and for binding communication to these resources will proposed. Allocation and binding is an important and well understood topic in the area of high-level synthesis and optimizing compilers. Approaches in this area are based on coloring of conflict graphs and clique partitioning of compatibility graphs. With these two types of graphs, the relation between elements to be mapped to shared resources can be expressed. Conflict graphs have edges between nodes that can not share a resource while compatibility graphs have edges between nodes which may share a resource. A proper node coloring of the conflict graph provides a solution to the resource sharing problem. Each color corresponds to a resource and the minimum number of colors corresponds to the minimum number of shared resources. In our case, the nodes in these graphs represent communication. In order to construct a method based on these approaches, the communication schedule graph has to be transformed accordingly into a conflict graph. For the construction of the conflict graph, the communication schedule graph is traversed and edges are drawn between all communication that are present in one node in the communication schedule graph.

VI. Case study: Viterbi decoder

As an example, the described methods will be applied to a SystemC description of a Viterbi decoder provided by Infineon. The decoder consists of several parallel processes, communicating via signals with a non-blocking send and a blocking receive behavior. The decoding is performed by the processes `viterbi_forward` and `viterbi_backward`. The system is implemented on a multiprocessor platform, whereas the two decoding processes are mapped to μC cores, `vitmem` and `bytemem` are mapped to memory modules, and `vitstack` is realized as hardware. To validate the system, the following information has to be determined: (1) the maximum data rate, the system is capable to process, (2) the maximum delay between input and output signal, and (3) the maximum output data rate.

Additionally, the internal communication have to be checked for deadlocks or data loss. For that purpose, the methods for communication analysis described in Section III will be applied. Figure 11 shows the analysis work flow. The source code is compiled as assembler code of the target machine. The program SPLITTER performs the decomposition of the problem into the communication structure of each process and the control-flow graph of the code between two communication. Using our tool for execution time analysis (GROMIT) the minimal and maximal execution time of the communication free parts are determined. The results of the execution time analysis are annotated on the edges e_{cdg} of the single processes. The global *CDG* is composed of the communication structure of all processes. The resulting *CDG* is shown in Figure 12. The annotated latencies in this figure rep-

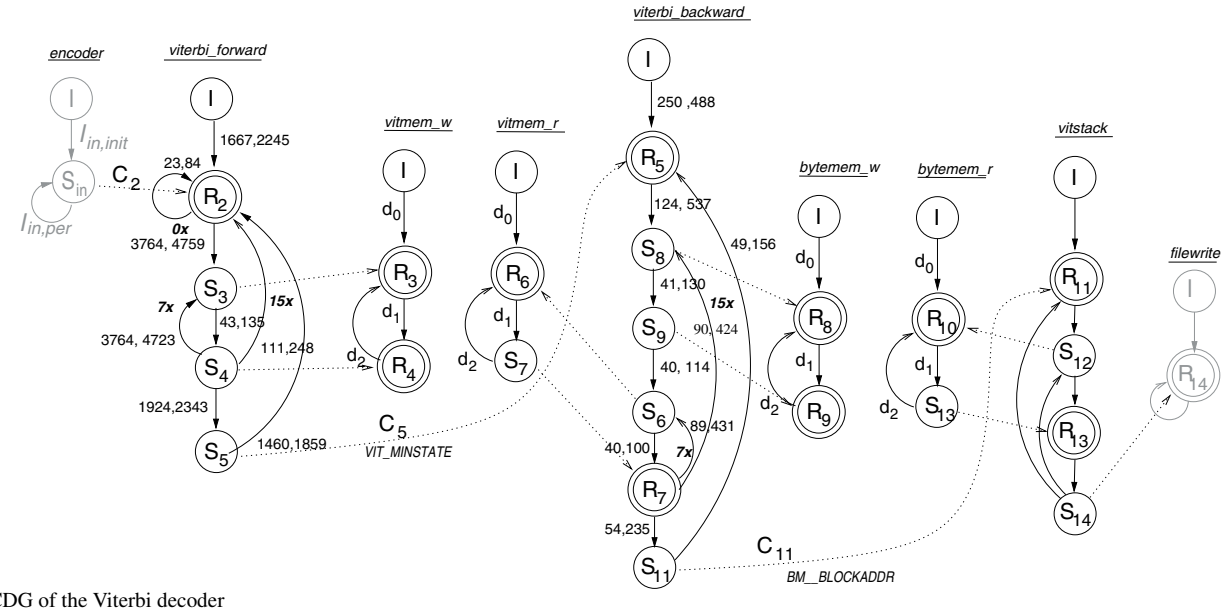

Fig. 12. CDG of the Viterbi decoder

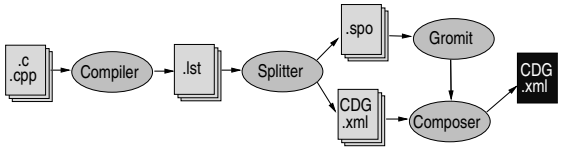

Fig. 11. Analysis work flow

resent the best-case and worst-case execution time of the code between the communication in clock cycles on a PowerPC750-processor. The processes `encoder` and `filewrite` represent the environment. To calculate the maximum input data rate, the smallest value for $l_{in,per}$ has to be determined that leads to a synchronization:

$$\begin{aligned} I \rightsquigarrow C_2: & \\ \min(l_{in,init}) &\leq 2254 \\ C_2 \rightsquigarrow C_2: & \\ \min(l_{in,per}) &\leq 4759 + 8 * 135 + 7 * 4723 + 2343 + 1859 \\ &\leq 43102 \end{aligned}$$

To prevent data loss, communication C_5 has to be a synchronization point. The synchronization equation (according Equation (2)) for that communication can be formulated as follows:

$$\begin{aligned} & C_5 \rightsquigarrow C_5: \\ & 1859 + \underline{x}_{2.2} + 15 * (4759 + 8 * 135 + 7 * 4723 + 248 + \underline{x}_{2.1}) \\ & +3764 + 8 * 43 + 7 * 3764 + 1924 \\ & = 537 + 130 + 114 + 100 + \underline{x}_{7.3} \\ & +15 * (424 + 130 + 114 + 100 + \underline{x}_{7.2} + \\ & 7 * (431 + 100 + \underline{x}_{7.1})) + 235 + 156 + \underline{x}_{5.1} \end{aligned}$$

Obviously, it is necessary to set up the system of equations for all involved communication to solve this equation. Considering the restricted space, in this paper we just present the result:

$$\Rightarrow \qquad \underline{x}_{5.1} = 557403$$

This means that the signal `VIT_MINSTATE` leads to a synchronization of the two processes.

WCRT Based on the determined slack times, the WCRT of the system can be calculated. The following equations determine the time between the first occurrence of the input signal and the emission of `BM_BLOCKADDR` (communication C_{11}) .

$$\begin{aligned} & L_{path_{max}}(C_2 \rightsquigarrow C_5) \\ & = 15 * (3764 + 8 * 43 + 7 * 3764 + 111 + \overline{x}_{2.1}) \\ & +4759 + 8 * 135 + 7 * 4723 + 2343 \\ & = 740243 \\ & L_{path_{max}}(C_5 \rightsquigarrow C_{11}) \\ & = 124 + 41 + 40 + 40 + \overline{x}_{7.3} + 7 * (89 + 40 + \overline{x}_{7.1}) \\ & +15 * (90 + 41 + 40 + 40 + \overline{x}_{7.2} + \\ & 7 * (89 + 40 + \overline{x}_{7.1})) + 235 \\ & = 122088 \\ & L_{path_{max}}(C_2 \rightsquigarrow C_{11}) \\ & = 740243 + 122088 = 862331 \end{aligned}$$

The calculated maximum slacks can be used as well for determining the maximum time between the regular occurrence of a signal. The calculation of the maximum period of communication C_5 returns the following result:

$$\begin{aligned} & L_{path_{max}}(C_5 \rightsquigarrow C_5) \\ & +4759 + 8 * 135 + 7 * 4723 + 2343 \\ & = 754463 \end{aligned}$$

These results have been validated by the annotation of execution times back to the SystemC description according to the example in Figure 2 and its simulation. However, using our method an error has been detected in the original description. Due to a non-blocking communication, there was no synchronization between the processes `vitmem_r` and `viterbi_backward` resulting in data loss. Our analysis approach allows further modification of the architecture towards design space exploration. We analyzed the Viterbi decoder using multiple configuration. Some selected results are depicted in Figure13. We calculated the global minimum slack for communication `VIT_MINSTATE` (C_5). A positive value guarantees the synchronization of the two processes that are mapped to processors. A positive value of $\underline{x}_2$ represents a guaranteed synchronization of communication C_2 with-

conf. #	`viterbi_forward`	`viterbi_backward`	$I_{in,per}$	$\underline{x}_5$	$\underline{x}_2$
1	233 MHz I-cache D-cache	233 MHz I-cache D-cache	200 μs	2906 μs	6 μs
2	233 MHz I-cache D-cache	100 MHz no cache no cache	200 μs	1495 μs	6 μs
3	100 MHz I-cache D-cache	100 MHz I-cache D-cache	500 μs	6294 μs	49 μs
4	233 MHz no cache D-cache	100 MHz I-cache D-cache	1500 μs	22283 μs	46 μs

Fig. 13. Analyzed system configuration

out any need to buffer incoming data from the temporal environment. A further decreased clock frequency of the processor running `viterbi_forward` down to 50 MHz in configuration 1-3 would cause data loss in communication C_2 and C_5.

The calculated results can be used in further computations for identifying possible conflicts while accessing shared resources. Based on synchronization points, the global sequential order of all communication can be determined. For that purpose, the method presented in Section V will be applied to the Viterbi example. The resulting *CSG* is shown in Figure 14. It is apparent that there are multiple potentially conflicting accesses on shared communication resources. Based on the *CSG*, the conflict graph shown in Figure 15 has been constructed. The nodes represent communication whereas edges between the nodes embody the competition for communication resources. The coloring of the conflict graph

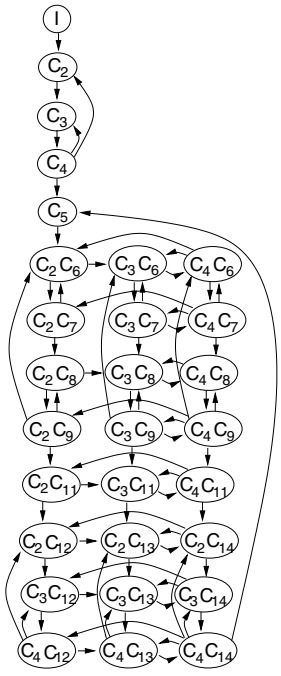

Fig. 14. CSG for Viterbi decoder

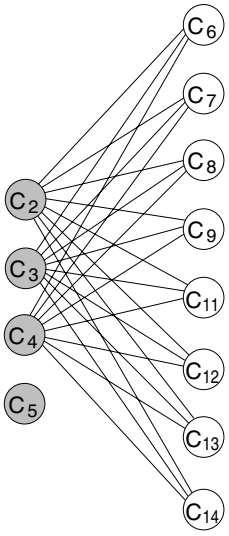

Fig. 15. Conflict graph

delivers the solution for the issue of static binding and allocation. The presented example requires two communication resources for guaranteed conflict free access.

VII. Conclusion

In this paper, an approach for analyzing parallel, communicating processes has been presented. Unlike other approaches in this area, the control-flow of the processes is considered during analysis. A method has been presented to determine communication that cause a synchronization of processes. These synchronization points temporally relate processes and global performance parameters like the response time of the whole system can be determined. The modeling of the temporal environment of the system by additional processes and the incorporation with our analysis approach have been introduced. Furthermore, we showed the ability of this approach to provide much more possibilities for accurate environment description than the common event stream models. Based on an approach for communication analysis, an algorithm for detecting parallel and potentially conflicting communication has been presented. The result of this method can be used to perform a conflict free binding of communication to communication resources. However, this approach is not restricted to communication resources. It can be applied to determine conflicting access to arbitrary shared resources, opening a broad application area. Furthermore, we briefly explained a method to back-annotate determined execution times to SystemC models for enabling a time-enhanced simulation. The described methods have been applied to an example, the SystemC description of a Viterbi decoder consisting of multiple communicating processes. In this example, the existence of synchronization between processes, the maximum input data rate, and the worst case response time have been determined. The evolved methods provide a powerful basis for formal timing analysis of embedded systems.

References

[1] R. Alur. Timed Automata. In *Proceedings of Computer-Aided Verification*, 1999.

[2] S. Bradley, W. Henderson, and D. Kendall. Using Timed Automata for Response Time Analysis of Distributed Real-Time Systems. In *Proceedings of Workshop on Real-Time Programming WRTP*, 1999.

[3] O. Bringmann. *Synchronisationsanalyse zur Multi-Prozess-Synthese*. Logos Verlag Berlin, 2003.

[4] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proceedings of DATE*, Munich, 2003.

[5] S. Dey and S. Bommu. Performance Analysis of a System of Communicating Processes. In *Proceedings of ICCAD*, 1997.

[6] R. Henia and R. Ernst. Context-Aware Scheduling Analysis of Distributed Systems with Tree-shaped Task-Dependencies. In *Proceedings of DATE*, 2005.

[7] A. Hergenhan and W. Rosenstiel. Static Timing Analysis of Embedded Software on Modern Processor Architectures. In *Proceedings of the DATE 2000 Conference, Paris, France*, 2000.

[8] M. A. Marsan, A. Bobbio, and S. Donantelli. Petri Nets in Performance Analysis: An Introduction . In *Lecture Notes in Computer Science*, volume 1491. Springer-Verlag, 1998.

[9] P. Pop, P. Eles, and Z. Peng. Performance Estimation for Embedded Systems with Data and Control Dependencies. In *CODES*, 2000.

[10] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Worst-case performance analysis of parallel, communicating software processes. In *Proceedings of the Tenth International Symposium on Hardware/Softwarw Codesign*, 2002.

[11] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Communication Analysis for System on Chip Design. In *Proceedings of the Design Automation and Test in Europe Conference (DATE)*, 2004.

[12] W. Stark and S. A. Smolka. Compositional Analysis of Expected Delays in Network of Probalistic I/O Automata. In *IEEE Symposium on Logic in Computer Science*, 1998.

[13] Y. Xie and W. Wolf. Allocation and Scheduling of Conditional Task Graph in Co-Synthesis. In *Proceedings of DATE*, Munich, 2001.

[14] A. Yakovlev, L. Gomes, and L. Lavagno. *Hardware Design and Petri Nets*. Kluwer, 2000.

[15] T.-Y. Yen and W. Wolf. Performance Estimation for Real-Time Distributed Embedded Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 9, November 1998.