Communication Analysis for Network-on-Chip Design

A. Siebenborn, O. Bringmann, W. Rosenstiel

FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
[siebenborn,bringmann]@fzi.de

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

Abstract

In this paper we present an approach for the analysis of systems of parallel communicating processes, with regard to Network-on-Chip applications. We present a method to detect communications that synchronize the program flow of two or more processes. These synchronization points set the processes into relation and allow the determination of the global timing behavior of such a system. Using the results of our method for communication analysis, we present a new method to detect communications that might produce conflicts on shared communication resources. This information can be used to determine static routing in a packet routing network.

1. Introduction

Network-on-Chip (NoC) is a new paradigm to enable an efficient design of future Systems-on-Chip (SoC), which exploit the multi-billion transistor capacity of single ASICs by composing hundreds of interconnected IP blocks, such as CPU and DSP cores, application specific hardware and large amounts of memory. The major challenge for SoC designers is to master the strongly increased system functionality being accompanied by increasing problems of deep submicron effects (e.g. cross-talk, interference) under timing, area, and low power constraints. Networks-on-Chip are a promising approach to address these problems by replacing global wires and on-chip busses with packet routing networks. The main advantages are reduction of electro-magnetic effects by introducing a well-structured interconnection layout instead of long-distance wiring as well as providing a good scalability and network utilization by using dynamic packet routing instead of on-chip busses and direct networks. The major drawback of on-chip packet-routing architectures is the strongly decreased timing predictability, due to dynamic packet routing, so that NoC architectures are not applicable to a wide bandwidth of embedded real-time application. Therefore, this approach aims to eliminate this problem by investigating new analysis and synthesis methods to determine temporal dependencies for communications, to calculate parameters for automatic network generation, and to refine analysis methods for NoC communication protocols. Parameter calculation for network generation include the component placement to minimize communication latencies and conflicts on shared communication resources and the determination of efficient routing at design time to guarantee a predictable timing satisfying given timing constraints.

This paper presents an approach to eliminate this problem by a combined approach for static communication analysis and network generation in order to determine the timing behavior of the entire systems and to detect bottlenecks in the communication structure. Based on the determined results, the communication network can be dimensioned, conflicts to shared communication channels can be avoided and routes for different communications between two or more modules can be calculated during design time for guaranteeing predictable timing satisfying all given constraints.

2. Related Work

The complexity of systems is steadily increasing and already today, there are designs integrating more than one processor cores with different instruction sets, various interfaces, memory and specialized hardware onto a single chip. Even today, the interconnection of the components is a key issue in the design of such systems. On-chip busses with standard interfaces have been developed to interconnect IP components the same way, as boards a few years ago. This imposes the need for new design and analysis tools, allowing to cope even with the existing complexity and move design decisions to higher abstraction levels. Most SoCs are used in embedded systems, that interact with their environment under more or less hard time constraints. The communication in such systems has a strong influence on the global timing behavior. Methods are needed to analyze the timing, as average throughput as well as worst case response time.

Relevant research areas for this work include the analysis of the communication behavior between parallel processes, timing analysis for hardware and software systems, and the rapidly emerging field of Network-on-Chip. Goal of communication
analysis is to determine the temporal relations between parallel, communicating processes. This knowledge is important for an efficient usage of shared communication resources. However, the accuracy depends on a precise execution time analysis of each communicating process.

Due to the influence of architectural features of modern processors the timing behavior of software processes is not easy to determine. Today it is not only sufficient simply to count instructions. There are several approaches to determine the worst-case-execution-time (WCET) of a program, executed on a dedicated processor. However existing approaches for WCET estimations of software do not consider communication between software tasks, that may suspend their execution.

3. Communication Analysis for SoC Design

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 analysis of concurrent, communicating systems are based on Petri nets, process algebras and communicating automata. The prerequisite for all presented approaches is the ability to incorporate timing information of the underlying communication network. Therefore, all un-timed approaches (e.g. un-timed Petri nets) are not discussed in the following. Petri nets provide a universal technique for modeling concurrent systems. While early approaches are mainly simulation-based, modern approaches are based on efficient analysis techniques able to cope with timed Petri nets [8, 16]. However, the applied communication types have to be modeled implicitly, so that the analysis is performed without knowledge on concrete communication types, processes, communication channels and the corresponding places and transitions.

A promising approach for analyzing worst-case timing behavior of concurrent systems is based on communicating automata [13], an extension to timed automata [1, 3]. However, these approaches rely also on totally synchronous communications. This quite restrictive model reduces the possible degree of concurrency and is not realistic for communication with buffers and latencies in the communication channel.

Beside all discussed approaches addressing distributed systems, several approaches exist tailored to the requirements of real-time systems. Here, we have to distinguish between hardware-oriented and software-oriented approaches. Typically, hardware-oriented approaches assume a parallel execution of the specified processes on dedicated resources, while software-oriented processes assume a sequential execution of the processes on a single processor by use of task scheduling. The combination of both aspects are tackled by software-oriented approaches addressing parallel and distributed real-time systems, where all specified processes have to be mapped onto several processing elements [17].

A lot of software-oriented approaches abstract from the internal process behavior and operate on a acyclic task graph. This model bases on the assumption, that each task has a statistically determined execution time, and each task starts execution once all input signals are available. The main drawback of this model is the missing support of conditional control structures. Therefore, newer approaches extend the basic task graph model by adding control dependencies [9, 15]. But due to the acyclic structure, different communication protocols and data-dependent loops can not be modeled.

One of the first global hardware-oriented approaches that analyze the timing behavior of a system of communicating processes is presented in [6]. The system is modeled according to its original specification. 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.

For a restricted class of applications, first analytical approaches exist for timing separation of discrete events [14, 7]. The main problem of these approaches are the exclusion of conditional behavior in the control flow and the restriction to synchronous communications.

Recent approaches address general hardware/software platforms, very often with respect to user-specified I/O event models [10, 5]. The main problems are caused due to the focus on the I/O stream behavior and the missing consideration of functional and communicational dependencies of the application.

In [11] an analysis technique has been presented that combines methods for WCET analysis with an approach for communication analysis for hardware synthesis [4]. This approach has been extended to consider latencies on communication channels and to determine possible conflicts on shared communication resources [12].

4. Network-on-Chip Design

The major drawback of packet-routing network architectures is the strongly decreased determinism and predictability of the on-chip timing behavior, due to the dynamically performed network routing and segment arbitration. Therefore, SoCs with packet-routing network architecture are not applicable to real-time systems with hard timing constraints, if the following issue is not mastered, as well:

*Preventing indeterminism and unpredictable timing behavior by applying statically routing techniques that allow the usage of different routes at different clock steps for two communicating modules.*

The determinism and timing predictability of such a network depend on the scheduling strategy of the routers. By using static routing schedules, worst-case guarantees may be given, but this is difficult in the case of dynamic scheduling. On the one hand, static scheduling needs a global control making systems more difficult to scale to larger designs. On the other hand dynamic routing is a complex task introducing a significant area and time overhead. One solution to combine the advantages of both approaches is to schedule statically the routes for each communication and perform the physical routing dynamically by sending the routing information for the entire route in the header of
each message. This is possible, due to the fixed network structure in contrast to general computer networks. Furthermore, the overhead in each message header can be kept low since only the router at the current scheduled route have to be switched accordingly. In order to find an optimized solution, static analysis methods are needed to determine the requirements and the schedule for the network. For that purpose the bottlenecks in the design have to be identified. Requirements for communication latencies and bandwidths have to be determined in order to fulfill the global timing constraints on the system. It should be noted, that different media access methods, routing, and quality of service mechanisms open a large design space. A method to analyze the temporal behavior of a system of concurrent processes, that communicate with each other, and where communication suspends the execution of the tasks will play a very important role in the design of reliable systems. Based on the given temporal constraints on the system, the temporal requirements for the communications, like bandwidth and latency can be determined. This information can be used to take decisions for the network protocol, like determine routing or assign virtual channels and priorities. At the same time, the need for analysis and design methods for such systems arises. A major goal concerning NoC, is the automated generation of a specialized network from a system description. The generation of a specialized network opens a huge design space. Even if a platform with integrated network will be used, an optimized component placement in the network has to be determined, considering possible conflicts at the communication medium. After a component placement, the location of the communication participants remains fixed, and possible routes can be determined at design time. Figure 1 shows the platform-based design for NoC. The functional description should be mapped on a platform with an integrated network. The network interconnects processing elements directly or via one or more routing elements. The processing elements may represent micro processor cores, DSP cores, application-specific hardware components and IP blocks. By applying communication analysis the bottlenecks and timing requirements for the network can be determined, and with this information a specialized network can be generated. Specialized estimation algorithms are performed for each processing element in order to determine the worst-case and the best-case execution time for functional blocks separated by successive communications. During iterative design steps, the system can be analyzed by communication analysis to check, if the system with the generated network fulfills the required timing behavior. For the communication analysis not only the inter connection structure between the modules is of interest. In this case, a precise temporal relationship between the modules can not be derived. Nevertheless, a lot of approaches try to schedule bus assignments and bus accesses by taking only the inter module communications into account. Therefore, we propose to determine the temporal order of the communications and the number of wait cycles in each blocking communication node by analyzing the inter module communications together with the internal control flow of each module (Figure 2). This allows a very precise analysis of the temporal behavior of a system of communicating processes.

5. Analysis of parallel, communicating processes

Our approach for communication analysis allows the validation of the real-time behavior of parallel software processes. Hereby two methods that work in two different problem domains are combined: (1) static timing analysis and (2) communication analysis. Static timing analysis handles code sequences with control structures, including loops with bounded iteration counts. Communication that blocks process execution and loops with unknown iteration counts can not be handled. On the other hand, for communication analysis only communication points and the timing behavior between those nodes are of interest. For that reason, the problem has to be decomposed for the two analysis domains.
For the communication analysis only information on the communications, and the temporal behavior between these communications is of interest. This information can compactly be represented as a communication dependency graph \( CDG \). The nodes in the \( CDG \) represent communication points. This graph contains two types of edges: Edges that represent the control flow between communication points, and edges that represent the communications.

A communication dependency graph \( CDG \) is a directed, cyclic graph, that can be constructed based on the \( CFG \) of each process. The edges \( e_{com} \) are given by the communications. 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 the edges \( e_{cdg} \), which are the execution times of the longest and shortest path between the corresponding nodes in the control flow graph. These latencies are determined by static timing analysis. For that purpose subgraphs of the control flow graph have to be constructed, that contain all possible paths from one communication node to another in the \( CFG \). Figure 3 shows an example for a \( CDG \) with three processes.

![Figure 3. Communication Schedule Graph](image)

In a real system the transmission of a message will need a certain amount of time, depending on the amount of data. Additionally there will be a certain delay between sending and receiving a message. For that reason, the parameters \( l_i \) for the message delay and the parameter \( d_i \) for the message duration are annotated to the communication edges \( e_{com} \) of the \( CDG \). For the determination of synchronization points, these values have to be taken into account. Communication partners leave synchronization points not at the same time, but time shifted by the delay \( l_i \). Also, the communication duration has to be considered for the determination of path latencies between communication points. Figure 4 depicts these parameters. The duration of the message \( d \) depends on the amount of data and the channel bandwidth. Process \( P_2 \) receives a message sent by \( P_1 \) delayed by the parameter \( l \).

![Figure 4. Communication Latency and Duration](image)

Based on the communication dependency graph, a condition can be formulated that has to be fulfilled for synchronization points. Each communication pair \((v_s, v_r)\) is a potential candidate for a synchronization point. An obvious condition is, that the blocking communication partner is reached before or together with the non-blocking partner.

The set of nodes on the shortest path from \( v_1 \) to \( v_2 \) is denoted by \( path_{min}(v_1 \rightarrow v_2) \). Analogically the set of nodes on the longest acyclic path is denoted by \( path_{max}(v_1 \rightarrow v_2) \). The function \( L(p) \) means the sum of all edge latencies \( l \) on a path \( p \). A communication \( C = (v_s, v_r) \) with \((v_s, v_r) \in E_{COM}\) is a synchronization point if

\[
L(path_{max}(v_{sync} \rightarrow v_r)) \leq L(path_{min}(v_{sync} \rightarrow v_s))
\]

Herein the set \( SP_{prec}((v_s, v_r)) \) refers to all previous synchronization points from which the communication nodes \( v_s \) or \( v_r \) can be reached directly, without passing an other synchronization point. This synchronization condition represents an actual criterion for the verification of existing synchronization points. By introducing a slack variable \( x \), for each communication \( C_i \) that represents the number of wait cycles at the communication, the condition can be formulated as an equation. 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 with an iterative algorithm, where communications with a result less than zero will not be considered for the next iteration. The algorithm stops, when all variables in the current system of equation become a positive value or zero.

The determined synchronization points and the slack variables can be used to determine the WCRT of a system. The influence of parallel processes to one process of the system is embodied in the slack variables. To determine the worst-case turn around time of one process, only the path latencies of this processes together with the slack variables have to be considered.

6. Access to Communication Channels

The information gained during communication analysis can be used to determine conflicts on shared communication resources, e.g. busses. The determined synchronization points set the parallel processes into relation and allow to put all communications into a temporal order.

The relative min. and max. latencies of the path between communications, together with the determined slack variables
allow to calculate absolute time intervals for each communication. Therefore SPs 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 5 and Figure 6. For example, the communications $C_2$ and $C_3$ in Figure 5 start execution during the time intervals $[L(path_{min}(I_2 \rightarrow S_2)), L(path_{max}(I_2 \rightarrow S_2))] = [7, 15]$ and $[L(path_{min}(I_2 \rightarrow S_3)), L(path_{max}(I_2 \rightarrow S_3))] = [12, 26]$. This time intervals are depicted in Figure 6. It has to be mentioned, that these intervals are related to the start times of communications, not to the duration. Even if these intervals overlap, with a look at the control flow of process $P_1$, it is obvious, that these communications will not be executed at the same time. That means, that it is necessary to consider also the order of the communications, that is determined by the control flow of the processes. Based on the synchronization points, that set the parallel processes of the system into relation, a global temporal order can be determined. This order can be represented by a graph: Parallel communications are grouped to a single node; edges denote the order of the communication and are given by the control flow of the program.

In the first step, only the order of communication points in the CDG is considered. During this step, path latencies are ignored. In principle, the algorithm is shown in Figure 7. It starts with the initial synchronization points as the first node in the communication schedule graph. The current communication points for each process, i.e. the system state, are stored in an array $A$. 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 $A$ is stored on a stack $V$, together with the current node $v$ of the $V$. The function $update()$ determines a set $A$ 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 $A$, 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 8 shows the CSG for the example in Figure 5. 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 communications, where both communication partners are contained in the array. Since the state vector $A_3$ contains the two communications $C_3$, $C_4$, these communications form one node in the CSG and are possibly executed at the same time.

![Figure 5. Possible conflicts on communication resources](image)

![Figure 6. Start time intervals of the communication in figure 5](image)

![Figure 7. Construction of Communication Schedule Graph](image)

![Figure 8. Communication Schedule Graph for the Example in Figure 5](image)
allel communications can be checked, if their time intervals are overlapping. Since $C_2$ in the example is a synchronization point, 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 $R_2 \rightsquigarrow S_3$ 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 be executed at the same time and accordingly the CSG can be reordered.

7. Resource Allocation and Binding

In the previous section, the detection of parallel communications has been explained. This information is now used to detect possible conflicts when transmitting communications via a shared media. In the following, a method for the allocation of a minimal number of communication resources e.g. a bus and the binding of communications to these resources will be shown. As already mentioned, allocation and binding is an important and well understood topic in the area of high-level synthesis and optimizing compiler. 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 onto 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 communications.

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 communications that are present in one node in the communication schedule graph.

8. Conclusion

Today, SoC consist of several components that communicate with each other via shared communication resources, like network-on-chip. For the design of such systems, formal methods are needed to guarantee the specified temporal behavior, particularly in safety critical domains. For the design of systems, with a predictable temporal behavior, the detection of possible resource conflicts, and a conflict free resource allocation and binding are crucial. Based on an approach for communication analysis, this paper presents a new method for the detection of parallel communications, that might produce a conflict on a shared resource e.g. a bus. Moreover, the allocation and binding of communications to a minimum number of communication resources has been shown. This information can be used to identify bottlenecks in the system, determine constraints for communication channels and assign priorities to single communications in case of QoS mechanism. For network generation, the gained information can be used for design space exploration with many options concerning component placement, routing, media access, flow control and priorities.

References