Abstract
In this paper we present an integrated design concept that focuses on the correctness of concurrent controllers. Our approach is based on the specification of concurrent tasks with aid of structured flow charts. We introduce a new formal proceeding for the verification of the correct behavior. Our algorithms reduce the verification process to a polynomial amount of computational effort. Thus, our method can be applied even for the design of large systems.

1. Introduction
A convenient approach for the design of digital hardware systems is to divide an entire system into two individual parts: One part consists of processing units which are exclusively tailored for data processing. In order to supervise the data processing activities in the processing units we need controlling units. In the design process the use of standard cells and previously defined processing units with dedicated functions has become 'state of the art' in the last recent years. Thus, the designers activities more and more are moving towards the organization of functional blocks and in respect of that, there is an increasing need to improve the communication management by suited controllers.

As the system complexity increases controlling becomes more complicated, especially, if the control task has to be distributed among several controlling units. For the designer two problems are gaining ever greater significance:

- First, he precisely has to specify how the system control should behave. To reduce modeling complexity the specification should be done at an abstract and realization independent level.
- Secondly, the correctness of the specification has to be verified before implementation in order to prevent runtime conflicts such as livelock and deadlock situations. The costs of verification increases exponentially with the system complexity.

The latter aspect is an indispensable issue because conflicts detected in an implementation will lead to expensive and time consuming redesign cycles.

Regarding the design support for distributed controllers there is a lack of suited specification and analysis tools: The technology provided for software development (OCCAM, Concurrent PASCAL etc.) generally does not meet the requirements of hardware design because there is neither a reference to hardware cell libraries nor to clock based timing models. Conventional methods for system specification, such as Dijkstra’s P/V-systems [1] or the concepts of message passing also are not suitable for our applications: P/V-systems are restricted in modeling power [2]. Message passing systems [3] require a realization dependent block architecture and they are difficult to analyze. Since petri nets have been proposed by C.A. Petri [4] they are widely used as a model base for concurrent systems. A lot of analysis tools have been developed to check the behavioral characteristics of the specified systems. However, it is well known that computational complexity limits the applicability to very large systems [5,6]. To cope with modeling complexity several high order petri nets have been introduced [7] but there is still a lack of support for the investigation of their characteristic properties. Interaction based abstract programming languages such as Milner’s process algebra CCS [8] are also promising candidates for concurrent process specification, but due to the inherent state explosion problem complexity is a limiting factor even here.

Faced with this unsatisfactory situation we felt that there was a need for an integrated specification and analysis concept well tailored for the design of concurrent controllers. In our work we de-
developed a new method for this purpose. By use of a clear design paradigm and structuring aids we provide a proper formalism for the specification and the early verification of correctness as well. In this paper we briefly introduce the main ideas of our design concept and then we focus on our efficient approach for the correctness analysis of concurrent controller specifications.

2. Design Concept

In our research group we concentrate on the development of formal concepts for the specification and analysis of concurrent control tasks and we provide the means for a systematic design and implementation of these tasks in form of distributed control units. Ordinarily, the design of small modules for the solution of sequential partial processes can be accomplished by use of 'state of the art' techniques. For this objective mature concepts and reliable tools are available. A complex system however, simply synthesized by composing several distinct modules, each of them with a perfect functionality, is not guaranteed to work as a conflict free unit. The main difficulties raise owing to the interaction of well organized subsystems. The management of this crisis basically requires a global view on the entire system behavior. To handle the issues of specification, correctness analysis, simulation, and implementation in a profitable manner, it is advantageous to embed these tasks in an integrated environment.

Facing this we initiated our work with the development of an open design concept. For a detailed explanation refer to [9, 17]. Within our concept the entire design process - from the system description at a behavioral level to the generation of communicating finite state machines (FSM) - is partitioned into a series of partial transformation steps. The backbone of this concept is a new graph theoretical calculus, especially tailored for the specification and analysis of concurrent processes.

For the description of the control task we start at behavioral level with our novel concurrent controller specification language COCOS. COCOS is syntactically based on VHDL and it excellently meets controller specific requirements. Therefore, our open concept at this stage offers an interface for the integration of VHDL models of the processing units that have to be controlled. The processing units themselves are not matter of our research.

In our first transformation we algorithmically generate an internal graph representation of the COCOS control flow. This calculus originally has been proposed by Thurn [10]. Because structuring is an essential feature within this calculus we refer to this representation form by 'concurrently structured flow charts (CSF)'. We give a detailed description of this calculus in the following chapters.

A CSF serves as a platform for our efficient correctness verification algorithms and for subsequent design steps as well. After verification the set of concurrent operations in an CSF is partitioned. The resulting sequential subsets are associated with individual controlling units. To organize the interaction of the controlling units coordination mechanisms are derived from the relational directives in the CSF. The coordination is algorithmically added to the individual basic subcontroller tasks. As the result we obtain functional models in standard VHDL language for all controller units. At this stage we are able to use any conventional VHDL simulator for a common simulation of controlling units together with processing units.

Based on the functional VHDL modules we move on to optimize the control flow and thereupon we provide a link to conventional logic synthesis tools taking charge of lower level design.

3. Demands for correctness

Dealing with concurrency the designer is confronted with a lot of problems that will not arise in sequential systems. The system reliability depends on the 'correctness' of the designers specification. Runtime conflicts, such as deadlocks, lead to unpredictable and undesirable behavior. Fairness conditions have to be guaranteed to prevent concurrent processes from unnecessary mutual blocking. Furthermore, the designer must ensure that the system specification can be mapped onto a physical implementation. To prevent from such failures it is mainly important to concentrate on the inherent behavior of the control flow.

In this chapter we briefly introduce inherent properties for concurrent systems and we give a definition for 'correctness' as it is used in this paper. Our definition is based on the research that has been made for the characteristics of petri net specifications. Because there exists an equivalent petri net representation for each CSF we can easily apply this definition to our calculus. We assume that the reader is familiar with the petri net theory thus, we need not to explain it in detail. For more information see [7, 12].

One basic issue for a system specification is the question whether a petri net model is 'bounded'
('safe') or not: A petri net is 'bounded' ('safe') if there is no place $p_i$ with more than $k$ (1) token $t$. 'Boundedness' is strongly related with the occurrence of backward conflicts in a petri net. Only 'bounded' net representations can be realized as physical implementations, i.e. with finite state space. Moreover it is interesting whether a petri net is FSM decomposable. For that reason it is important to know whether the 'conservation' criterion is satisfied: A petri net is 'conservative' if the weighted sum $\sum (t_i \times w_i)$ for all places $p_i$ has a constant value at any time. For $w_i > 0$ the petri net is 'bounded'. Regarding our design concept FSM decomposability comes out to be a really important requirement.

Due to the fact that a concurrent system should behave in reproducible manner the modeling petri net must be 'deterministic': A petri net is 'deterministic' if firing of transitions with common input places is unambiguously arranged. The most important characteristic in concurrent processing is 'liveness'. In terms of the petri net net theory we first define 'liveness' as follows: A petri net is 'live' if none of its transitions is ever potentially unfireable. However, a control flow may contain initial start sequences or it can stop in exception sequences. Thus, it will not be modeled by a strongly connected petri net and some transitions will fire only once. We find three subcriteria, sufficient enough to ensure 'liveness' in practice: In concurrent systems it is essential to exclude 'deadlocks' and 'traps' [7, 13]: A 'deadlock' is a set $P_d$ of places $p_i$ in a petri net where the number of tokens never can be increased. A 'trap' is a set $P_t$ of places $p_i$ where the number of tokens never can be decreased. A 'deadlock' that loses all its tokens causes the transitions in $P_d$ to be unfireable. A 'trap' collects tokens and may be a reason for unfireable transitions outside $P_t$. In addition to 'deadlocks' and 'traps' as defined above there exists a third reason for non firing transitions [14] that we call 'blocking': Let a petri net $P_N$ be free of 'deadlocks' and 'traps'. A 'blocking' occurs if at least one token in an input place $p_i$ of a transition $t$ is missing and due to the actual distribution of tokens in $P_N$ no firing sequence can move a token to $p_i$. The reason for blockings are bad decisions in the token flow.

With reference to the above properties we now give our definition of 'correctness':

**Def. 3.1:**
A concurrent system will behave 'correct' if its equivalent petri net model is bounded, conservative, deterministic, and free of deadlocks, traps, and blockings.

It is well known that within the conventional petri net calculus the proof of 'correctness' requires exponentially increasing cost in time and space. In this paper we show how polynomial algorithms based on the CSF calculus can efficiently reduce this computational amount of effort.

**4. Relational Specification**

As mentioned above the central point of our concept is the CSF calculus. The basic idea lies in the relational description of indivisible control sequences that we call 'operations'. An operation

![Figure 1: Operations and Structural Relations](image)

(Figure 1a) contains sequential instructions and decisions to perform local control over associated processing units. We demand that the internal control flow of an operation is neither directly nor indirectly influenced by other operations. Operations are concatenated by the 'direct predecessor relation $<dp>$', graphically represented by a directed edge. The transitive relationship is defined by the 'indirect predecessor relation $<ip>$'.

The causal dependence of operations can be characterized by the 'selection relation $<se>$' (Figure 1b). In the control flow a 'selection begin (SB)' symbol marks a place where a selection is made from different successors of an operation. A 'selection end (SE)' symbol labels a join point for all operations that previously have been separated. Selections are made in dependence of the value of event variables (EV) that encode the feedback information from the processing units. In contrast to selections that cause operations to run alternatively the 'concurrence relation $<co>$' (Figure 1c) specifies concurrent activation of operations. Concurrent operations are causally independent. 'Concurrence begin (CB)' and 'concurrence end (CE)' symbols are used to enclose concurrent sections in the control flow.
Due to the strong relationship to the structure of a control flow chart the \(<dp>, <se>, \) and \(<co>\) relations are summarized by the term 'structural relations'. In order to influence concurrent operations with regard to the time behavior a second group of relations which we call 'temporal relations' (Figure 2) is put at disposal. Temporal relations are powerful instructions that allow the specification of behavioral rules at a realization independent level. The temporal relations include:

Simultaneity relation \(<si>\); (Figure 2a). This relation between two concurrent operations forces the synchronous start of execution.

Before relation \(<bf>\); This relation means that \(O_i\) has a preparatory effect on \(O_j\) and must therefore take place before \(O_j\) (Figure 2b).

Non-simultaneity relation; operations with this relation are hindered to run simultaneously. We distinguish between relations for which no priority \(<ns>\) is required (Figure 2c).

Figure 2: Temporal Relations

5. The concept of consistent relations

In this chapter we will explain how the correctness of a relational CSF-specification depends on the consistency of structural and temporal relations. It is turned out that the consistence of the temporal relations mainly can be proven by examination of cycles.

5.1. Consistent structural relations

The correctness of relational specifications is related to the consistency of relations. Thurn [10] has proven that a relational flow chart corresponds to an equivalent correct Petri net if all groups of relations are consistent, i.e. there are no contradictions with respect to relational properties. This view of correctness is restricted of the properties described in Chapter 3. Basically, there are two different ways to prove the consistency and respectively the aspects of correctness as defined in Chapter 3 is guaranteed.

5.2 Consistent temporal relations

In this chapter we consider the analysis of CSF specification completed by use of the temporal relations: \(<bf>, <si>, <ns>\) and \(<sns>\) (see Chapter 4). For this completion of the relational specification we present our approach to ensure the correctness, as defined in Chapter 3. Again proving consistency is the fundamental idea to satisfy the demands of Definition 3.1. In an equivalent Petri net the amount of analysis would significantly increase because the temporal demands must be modeled by complex specific Petri net extensions [15]. Modeling with consistent structural and temporal relations will be performed by use of a constructive approach. Again, the starting point is a set of rules:

Def. 5.2:
A CSF specification extended by the temporal relations: \'<bf>, <si>, <ns>\) and \'<sns>' is called consistent, if

1. the temporal relations are exclusively applied to concurrent operations.
temporal relations between two operations, are not used as follows:

a) \(<bf>\) together with \(<si>\),
b) \(<ns>\) together with \(<si>\),
c) \(<sns>\) together with \(<si>\) and

d) all combinations of \(<bf>\) and \(<sns>\)

relations in an opposite direction.

(3) temporal relations are not used in loops or between selective operations.

(4) no inconsistent cycles of temporal and structural relations exist.

It should be mentioned that we developed refinements for rule (3) in order to allow the use of temporal demands in loops and between selective operations in a consistent way. For this paper however, these specific rules are not important, because our intention here is to emphasize just the principal ideas of consistent relations. If the temporal relations are used as defined above, the correctness of the entire CSF-specification will be ensured. For explanation of Definition 5.2 we give some examples:

In Figure 6a violations of the rules (3) of Definition 3.2 are illustrated. Operation i (loop SB1-i-SE1-SB1) is specified to execute before operation j. But if the loop is repeated more than once the temporal demand by the relation \(<bf>\) causes a blocking for operation i. This denotes that a violation of the rule (3) also implicates a conflict in the structural and temporal demands of CSF-specification.

a) Blockings  b) \(<bf>\) cycle  c) \(<sns>\) cycle

Figure 6: Examples of Temporal Conflicts

The example of Figure 6b illustrates rule (4). In this example a conflict between temporal and structural relations arises. Operation i intentionally executes before operation j. In the same way k is specified to run before i is started. The specification j \(<dp>\) k demands, that j has to work preceding to k. Thus we get a conflict, because it is implied that i already has to run before its first occurrence. An other conflict is described in Figure 6c with respect to the rule (4). Because it is possible that the operations i, j and k are simultaneously ready for start, the relations \(<sns>\) yield a conflict due to the following fact: operation j takes priority over operation i, k over j and i over k. This is like a typical right of way problem, no operation will be enabled to start.

In case of the rules (1), (2) and (3) an algorithmic proof consists of search and comparison of temporal and structural relational properties. The amount of computation is polynomial. However, the proof for rule (4) comes out to be much more complicated. It requires the search and assessment of relational cycles.

Our algorithms base on the following ideas:

The search is done in a simplified version of the CSF specification, an unweighted directed graph. The operations O and the SB-, SE-, CB- and CE-elements of the CSF-specification are transformed into the vertices of the directed graph, with respect to the relational properties the directed relations \(<dp>\), \(<bf>\) and \(<sns>\) are described by directed edges and the symmetrical relations \(<si>\) and \(<ns>\) are represented by antiparallel edges. For the search algorithms there is no difference made between structural and temporal relations. But all critical relational cycles are cycles in this graph, too. Not every cycle in this graph denotes an inconsistent cycle in the CSF-specification. Therefore, an accurate analysis of the cycles will be necessary. It seems to be an easy way to find inconsistent cycles of the CSF-specification by a preceding determination of all possible cycles in the unweighted, directed graph. After that an analysis of the relational composition of the cycles in the CSF-specification examines the consistence. The problem is that such a general search of all cycles is an exponential problem, as well. Owing to this, the aim of our research has been significant reduction of costs.

Fortunately, we can take advantage of some favorable facts that allows us to reduce the effort. At first, we identify three types of cycles which do not result in conflicts:

a) Loops intended by the programmer (Figure 7a)
b) Cycles which only include relation \(<si>\) or only \(<ns>\). These specifications satisfy the rules of Definition 5.2 (Figure 7b).
c) Cycles with temporal relations containing concurrent processes in parts. These are no conflict because the temporal relations occur only within concurrent meshes (Figure 7c).
Secondly, it is sufficient to find one inconsistent cycle violating the rules of the definition given above and the amount of effort to find a single cycle is polynomial.

In order to avoid unnecessary search for cycles in the three exceptional cases we have developed an adaptive version of the algorithm of Neumann [16]. In our improved cycle search exclusively real conflicts are detected. The essential idea of our search algorithm is to start cycle construction with critical temporal relations. Thus we generally avoid to search for cycles as in Figure 7a. Furthermore previous sorting of vertices and well suited selection of edges prevent from searching cycles in Figures 7b and 7c. The costs of the adaptive search algorithm to find one single cyclic conflict is polynomial.

As a result, the generally exponential cycle search problem is reduced to some essential cases, that purposefully can be found with polynomial amount of computational effort.

6. Conclusion

In complex system design one of the core problems lies in the verification of the correct behavior. Checking for liveness and safeness characteristics is an indispensable task especially for such systems that are distributed among several concurrent processes. To prevent deadlock and livelock conflicts at runtime the proof of correctness has to be offered before implementation. Since the costs of verification usually increases exponentially with the system complexity the checking performance is an essential issue. Correctness is mapped onto inherent characteristics of a high level formal specification. In our work we show that suited structuring rules within our calculus result in a significant cost reduction for algorithmic verification of correctness.

Concurrent control comes out to be an important task for a wide field of future applications. Within our integrated design we provide the means to facilitate specification, analysis, and implementation as well. Therefore, our contribution could be very promising to fill a gap in the conventional design procedure.

7. References