Incremental Synthesis and Support for Manual Binding in the MIMOLA Hardware Design System

O. Broß, P. Marwedel, W. Schenk
University Kiel
Institute f. Informatik u. Prakt. Mathematik
Olshausenstr. 40-60
D-2300 Kiel, W. Germany

1 Introduction

In the following, we study the case of high-level synthesis algorithms generating structures at the RT-level. The constituents of these structures are RT-components (ALUs, registers, RAMs, etc.) and their interconnections.

A number of problems exists with current high-level synthesis algorithms, but two related important problems are generally present: the phase-coupling problem and the complexity problem.

1.1 Phase Coupling

Most synthesis systems partition the synthesis algorithm into a number of phases solving subproblems. The subproblems include e.g. the binding of variables to storage elements, decomposition of complex expressions, scheduling, module selection, binding of operations to modules, the generation of control, module generation, floor planning, routing, and layout generation. There have been attempts to combine several of these subproblems and to solve them in a single step. However, no one has succeeded in and no one probably will ever succeed in solving all subproblems in a single step.

Solving subproblem after subproblem does not guarantee an optimum or even an acceptable solution for the complete problem. Current synthesis systems spend much time in predicting the effects of a decision on subsequent steps. Nevertheless early design decisions reduce the design space without trading area against time consumption.

1.2 Complexity

With a design space as big as the design space for high-level synthesis and with the problems of predicting the precise effects of early design decisions on the final design, it is extremely important to reduce the design space as much as possible. Unreasonable designs should be excluded by the reduction of the design space. Only then will we be able to generate several reasonable designs and to fully explore the remaining design space.

2 Concepts for a new Synthesis Algorithm

2.1 Area-oriented Designs

Synthesis decisions should trade the area occupied by the structure versus the time needed to execute the program. Almost every design decision has a potential to result in an unacceptable layout. The execution time can easily be calculated from the number of control steps and the mimimal cycle time, whereas the area can not be simply estimated from the size of the incorporated modules [McF87].

2.1.1 Floor-Planning

All relevant design decisions should be based on a preliminary floor-plan. Peng [Pen87] was the first to propose such an approach. His algorithm, however, is not general enough to handle real design problems. For example, backtracking is restricted in order to avoid infinite loops.

The floor-planner required for synthesis has to be tightly coupled to synthesis. It should be an incremental floor-planner, which is able to predict the effect of a small design change rapidly.

2.1.2 Busses

Many synthesis systems use multiplexers for selection of the appropriate data in a certain control step. It is well known that this is not area-efficient. There have been attempts to use busses [deM86]. However, current systems just minimize the number of busses. This number serves only as a very rough estimate of the required area. Floor-planning based design decisions are required for implementing area-efficient data selection.

2.2 Bindings and Partial Structures

Two means for the reduction of the design space can be identified: partial structures and binding information.

Below we discuss the use of partial structures and binding information during architectural synthesis. We will explain, how the bindings can be conveniently expressed in a design language. Our own language MIMOLA will be used as an example. The binding information cannot change semantics and hence it is compatible with the "correctness by construction" principle. The additional information has a potential to speed up synthesis algorithms.

2.2.1 Binding Information

High level synthesis deals with different kinds of bindings. We consider the binding of variables to storage, the operation to control step binding (i.e. scheduling), and the binding of operations to hardware components. The more the user specifies the bindings, the simpler is the task of the synthesis system and the resulting structure will be closer to the user's intention. This is only possible, if the input language allows the description of such bindings.

Prebound Variables:
Procedural descriptions of the required behaviour contain a set of abstract variables. It is not obvious, how these should be bound to hardware registers and RAMs.

In order to simplify the storage allocation problem, MIMOLA allows the user to define variable to location bindings. We distinguish between user and temporary variables. User variables can be bound in the variable declaration.

Example

```
VAR
  counter : integer AT CounterReg;
```

Here, CounterReg is the name of a visible hardware component.

Variables, which have not been bound by the user, will be bound to one of the locations set aside for user variables. In MIMOLA (version 4.0), these locations are defined by a construct similar to VHDL configurations.

Example:

```
RESERVED locations IS
  FOR Variables USE Main[0..4095];
END;
```

Fixed Control Sequence:
The scheduling phase of high-level synthesis can be omitted, if the mapping to control steps is already done by the user. Some mechanism is required to express this mapping. The mechanism has to represent a program as as set of control steps, each of which potentially contains several parallel assignments. An elegant solution is a special block structure of the form

```
SEQBEGIN
  PARBEGIN
    set of statements
  PAREND;
  PARBEGIN
    set of statements
  PAREND;
SEQEND;
```

Each parallel block represents a control step. SEQBEGIN ... SEQEND means that the sequence cannot be changed by any tool.

Preselection of Key Components:
Frequently, the user has a pretty good knowledge about the type and number of required key hardware components. Key components can be defined to be those modules which perform some operation that is explicitly listed in the required system behaviour. Hence, key components include RAMs, registers, ALUs etc.

The following example presents a partial description of a key component using MIMOLA:

Example:

```
PARTS                -- components
  alu : MODULE add_sub(IN a, b: word;
        IN c:twobits) RETURN f : word
        BEHAVIOUR AtRtLevel IS
          BEGIN
            f <- CASE c OF
              0 : a + b;
              1 : a - b;
              2 : a OR b;
              3 : a AND b
            END;
          END;
```

It is relatively easy to use this information during scheduling, because now all the essential resources are known. Scheduling now reduces to the type of scheduling implemented in microcode generators.

Prebound Operations:
As a result of previous design iterations, users sometimes realize, that better designs would be generated, if the operation to hardware component binding would be changed.

Example:

a := a +'alu b

All versions of MIMOLA provided constructs for this binding. For the sake of standardization, we are currently moving towards a straight-forward extension of attributes in VHDL.

2.2.2 Partial Structures

Partial structures are normally available for most design problems. Users of our own MIMOLA hardware design system frequently have a pretty good knowledge of some parts of the hardware structure. They want to express this knowledge in a partial description of the hardware structure and to use the synthesis system to design some additional circuitry like decoders, clocks and control. The existence of a partial designs should speed up the design process and should not slow it down.

Partial Description of the Interconnection Structure:
Until now we have discussed information concerning components and the binding of operations to components. The next step is to include at least a partial description of the interconnection structure in the synthesis input. This information shall be used to evaluate the effects of design decisions.

A simple way of using this information is easy to implement: Many of the synthesis tools are not globally optimizing. They consider hardware requirements control step by control step. For each of the control steps the hardware is augmented, taking into consideration the hardware requirements by previous control steps. With this approach, it is relatively easy to take advantage of information about the interconnection structure. This information is presented to the synthesis tool as a partial structure generated by a virtual 0th control step. This is the concept employed in a new synthesis algorithm for the MIMOLA design system [Bal89].

Full Structure Completely Fixed:
As an extreme, the specification may contain a single block with a fixed structure. This is a limiting case of incremental synthesis: only the control code can be changed during synthesis. This special case occurs due to the following reasons:

The results of automatic synthesis systems normally are analysed by human designers. Frequently they discover possible optimizations, changes required to improve testability and changes required to meet some company standards. After such changes, guaranteed correctness normally is lost. Tools, checking whether or not a manually modified hardware still meets the requirements, are needed in order to avoid this situation. In the case of a procedural specification this means: it has to be checked, that the specification can be compiled onto the changed structure. Our retargetable code generator MSSC handles this special case of incremental synthesis [Now89]. A typical design process including synthesis and generation of control code for a fixed structure is shown in fig. 1.

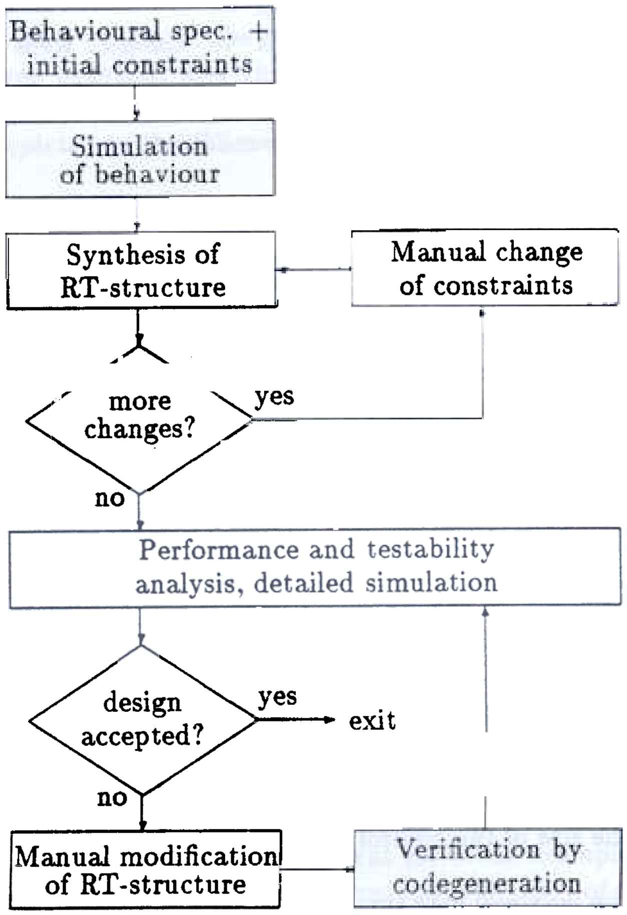
Behavioural spec. + initial constraints
Simulation of behaviour
Synthesis of RT-structure
Manual change of constraints
more changes?
yes
no
Performance and testability analysis, detailed simulation
design accepted?
yes
exit
no
Manual modification of RT-structure
Verification by codegeneration

Figure 1 Control flow during a design project with MSS

3 Implementation of the Concept

We have implemented a new synthesis system [Bro89] that incorporates the features described above. At each step the system deals with a complete structure, which is subject to improving transformations (c.f. table 1). Therefore the system is called SuccAss (successive approximation synthesis system).

3.1 Integration of Subproblems into a single Synthesis Algorithm

In this approach to synthesis we employ the MIMOLA frontend, the RT-mapping of the program via transformation rules, mapping of variables to memory locations and the parallelization of the RT-program. The following phases are integrated into the SuccAss transformations:

The introduction of temporary variables as required by decomposition of complex expressions,

- The allocation of hardware modules,
- The binding of operations to control steps (scheduling),
- The binding of operations to modules,
- The generation of control.

These transformations change the current structure. In order to preserve the required behaviour, the bindings of operations to control steps and the bindings of operations to modules are updated accordingly. The gain in chip area is estimated via a simple floorplanner, which is also integrated into the system. The design constrains, the system deals with, are as follows:

the library of available components,

the area of components,

the delay of operations,

the maximal area for the design,

the size of the control storage,

the maximal cyle time,

- the maximal number of temporary variables,

all kinds of bindings mentioned in section 2.2.1.

3.1.1 Transformation on the Structure

The SuccAss algorithm generates an initial structure that is large enough to perform each of the parallel blocks of the program in a single instruction. The algorithm applies transformations reducing chip area. The iteration consecutively identifies partial structures that may be replaced by alternatives. A transformation is called folding, if a single part is removed from the current structure. Another transformation substitutes e.g. a bus by a few multiplexers. The algorithm estimates the changes in terms of area and time. The estimates of the required area are calculated by a floor-planner. A transformation is applied, if it enhances the structure; i.e. the area shrinks and no additional control step has to be introduced, or the area is enlarged within a certain limit and the time is reduced.

```
Generate Initial Structure;
WHILE structure does not fit on the chip DO
BEGIN
  apply transformations
  not generating new control steps
  in order to reduce area;
  IF more transformations applicable
    THEN
      apply transformation
      with least increase in control steps
      and largest decrease in chip area;
  IF size of control storage exhausted THEN
    EXIT design constraints could not be met;
END;
Generate hardwired constants;
Generate Busses;
PRINT structure;
```

Table 1: The SuccAss algorithm

3.1.2 Changes to the Program

There are two basic changes on the program. The first breaks up expressions into smaller subexpressions by introducing a temporary variable. It is applied if the constraints on the available area are violated and cannot be met by any transformation of the structure. The second changes the statement to control step binding. It binds the statements to a feasible control step; i.e. a time, where idle hardware resources are available to execute the statement.

use the designers knowledge about efficient designs requires mechanism allowing the user to express such bindings.

We propose that such mechanisms should be present in design languages supporting synthesis. Synthesis systems using such an enhanced language should have an improved performance in the sense of reduced computation time and better results.

The new synthesis system SuccAss implements the proposed properties. The results indicate that the step to integrated synthesis systems is feasible and attractive as we generate structures closer to the designers intent.

References

[Bal89] M. Balakrishnan, P. Marwedel: Integrated Scheduling and Binding: A Synthesis Aproach for Design Space Exploration, 26th Design Automation Conference, 1989

[Bro89] O. Broß: Synthese von RT-Strukturen durch schrittweise Annäherung, master's thesis, Institut für Informatik, University of Kiel, (in german)

[deM86] H. De Man, J. Rabaey, P. Six : CATHEDRAL II: A Synthesis and Module Generation System for Multiprocessor Systems on a Chip, in: G. De Micheli, A. Sangiovanni-Vincentelli, P. Antognetti (ed.): Design Systems for VLSI Circuits, Logic Synthesis and Silicon Compilation, Martinus Nijhoff Publishers, 1987

[McF87] M.C. McFarland: Reevaluating the Design Space for Register-Transfer Synthesis, Proc. ICCAD, 1987

[Now89] L. Nowak, P. Marwedel: Verification of Hardware Descriptions by Retargetable Code Generation, 26th Design Automation Conference, 1989

[Pen87] Z. Peng: A Formal Methodology for Automated Synthesis of VLSI Systems, Ph.D. thesis, Department of Computer and Information Science, Linköping University, 1987