ABSTRACT

With current VLSI technology it has become possible to integrate complex systems in a single chip. Therefore much more efficient design and verification methods are needed especially at system level. We are using Real-Time Structured Analysis / Structured Design in logical behaviour specification and design of systems. We have developed automatic transformation from this graphical analysis and specification method to VHDL-Hardware Description Language. The resulting code can be simulated and so the behaviour of system can be verified at an early design phase. This paper presents the transformation principles and also describes the whole design process of ASICs.

1. INTRODUCTION

As VLSI devices become more complex, new methods are needed especially for specifying and verifying the design at high level. A single chip can contain a microprocessor with software as well as peripherals and user logic. The resulting system can be a subsystem of a larger system or a stand alone controller itself. Such an embedded ASIC must be modelled at behavioural level before detailed design of hardware and software can be started. Spending more time in specifying and verifying at high level reduces the time and cost of debugging and resimulating the design at lower design levels.

Our approach is to specify and simulate the whole behaviour of systems and ASICs at high level. We have selected graphics oriented Real-Time Structured Analysis / Structured Design (RT-SA/SD or SA/SD) method for specification and design of embedded ASICs. The high level specification is converted automatically to a behavioural hardware description language (HDL) code, which can be simulated. A rule based transformation has been developed from formalized SA/SD-diagrams to VHDL (VHSC Hardware Description Language). A similar transformation to DABL (Daisy Behavioral Language) has been realized earlier [1]. DABL is a behavioural HDL of Daisy Systems.

The verified high level specification is the starting point of further design phases. The reliability of the design process increases because flaws caused by inadequate specifications are detected before implementation design. In addition, it is possible to automate some parts of logic and layout design on the basis of the accurate specification.

2. DESIGN METHODOLOGY OF EMBEDDED ASICs

2.1 The SA/SD Method

The SA/SD method was originally developed for the specification of software systems. The method origins from the work of DeMarco [2], who presented the system as a hierarchy of data flows. The method was further developed by Yourdon [3] and the real time extensions needed for modelling computer systems were introduced by Ward and Mellor [4]. Recently the method has been adapted also to hardware design of embedded systems and ASICs [5].

A SA/SD description consists of hierarchical data flow diagrams, state transition diagrams and textual minispecifications. The data flow diagrams are constructed from data transformations, control transformations, stores and flows. The highest level data flow diagram is called the context diagram. It describes the system as a single data transformation with interfaces to its environment. At the bottom of hierarchy the behaviour of data transformations is described with textual minispecifications. The behaviour of control transformations is described with graphical state transition diagrams.

The SA/SD method must be formalized in order to make the automatic transformation to HDL possible. We have used Sokrates-SA [6], which is a refinement of SA/SD method. It is a formal specification method for real time systems. The minispecification language of Sokrates-SA is an ADA-like formal language. The semantics of Sokrates-SA also defines the exact behaviour of data and control transformations.

SA/SD offers a powerful tool for specifying and designing embedded systems. Its main advantages are the simple graphics interface, different levels of abstraction and good representation of concurrency. With minor extensions and modifications it can be used in all design phases. A combination of graphics and textual minispecifications makes it easy to specify system behaviour.
The SA/SD method is much more efficient for true behavioural modelling of systems than ordinary textual HDLs or block diagram based graphical interfaces.

2.2 The Design Process

In designing systems it is useful to distinguish the difference between the behavioural description of the system and the actual implementation of the system. Usually the designer starts with the implementation architecture of the system and describes this high level description misleadingly as a behavioural model.

Our design process (Fig.1) starts with technology independent logical behavioural modelling. The purpose is to define the logical behaviour of the system according to the original behavioural requirements. The resulting logical model handles only abstract data and it describes how the input data is manipulated to form the output and how the parallel or sequential data transformations operate in the system.

After the logical model has been designed the outside environmental details and restrictions are taken into account. The resulting functional model includes all the behavioural functions the system must perform. All the input and output data is typed according to the real-life environment. The functional model is implementation technology independent. At this stage the most efficient way to describe the system behaviour is to use detailed data flow diagrams and state transition diagrams, i.e. the SA/SD modelling.

The functional SA/SD model is then automatically transformed to a behavioural VHDL description by rule-based Sokrates-SA compiler. VHDL was selected as a target language because it has been standardized by IEEE and adopted by many CAE vendors.

With a VHDL logic simulator we can then verify the technology independent functional model of the system. At this stage we can get assured that we have specified the system behaviour correctly. The simulation files can be stored for later use and comparison with the results obtained from the actual implementation.

After the verification we proceed into the technology allocation of the system. The resulting allocated technology model contains behavioural models for both the software and hardware parts of the system. The allocated technology model contains also the description of the hardware/software interface.

The next step is the architecture design where the functions are allocated to different resources. At this stage we end up with technology dependent descriptions of the hardware and software, i.e. the implementation models. Even at this design phase we can use the graphical SA/SD notation.

The final step is the implementation design. The implementation design of hardware and software can proceed in parallel. As the design tools improve, it will become possible to automate more and more of this design stage.

3. TRANSFORMATION FROM SA/SD TO VHDL

3.1 VHDL As a Design Language

VHDL is a description language intended for use in all phases of the creation of electronic systems. It supports development, verification, synthesis, and testing of hardware designs. Also it helps maintaining and modifying designs. It is the first description language, which has been standardized [7].

The development of the VHDL was initiated by US Department of Defence. The IEEE standard 1076-1987 was ratified in December 1987. US Department of Defence requires ASIC design documentation using VHDL, which stresses the importance of VHDL as description and documentation language.
In the first place VHDL has been intended for representing the function and structure of hardware. In principle the system can be described in three different ways allowing designers to specify structural, data-flow, and behavioural models. All the three models represent the same design from different point of view.

All VHDL descriptions consist of an entity declaration and one or more architecture bodies. An entity declaration defines the connections between a description and its environment. Architecture bodies may be described by any of the three different description styles mentioned above. A single entity declaration may include several architecture bodies describing different views of design entity.

In structural description the architecture is expressed as a hierarchical arrangement of interconnected components. Each instance of a component represents a portion of the design that may be described by a lower level design entity. A component may be a single gate, a chip or an entire subsystem.

Data-flow descriptions (sometimes called register-transfer descriptions) consist of a set of concurrent register assignments. They may be register-transfer statements or timed, conditional, and guarded signal assignment statements. Data-flow descriptions resemble register transfer languages.

The third modelling style is behavioural description, which describes only the behaviour of entities by means of algorithmic definition. The process statement is the basic structure of behavioural description. There can be many concurrent process statements in one description. They model the parallelism inherent in hardware operations. A process statement contains sequentially executable statements. Behavioural descriptions resemble high level programming language.

3.2 Mapping of SA/SD to VHDL

VHDL 1076-1987 was selected as the target language because it is the only standardized HDL and because many CAE vendors are introducing tools with VHDL compatibility. Sokrates-SA was selected because of the refined semantics and because it earlier has been used for automatic software generation.

The first problem was to define correspondences between elements of SA/SD and VHDL. The transformation rules used by Sokrates-SA compiler were developed and coded on the basis of these correspondences.

SA/SD-diagrams form a hierarchical representation of system. The representation consists of graphical data flow diagrams, state transition diagrams and textual minispecifications. Corresponding structures of these elements have been specified in VHDL. Also the correspondencies between the Ada-like minispecification syntax and VHDL have been developed.

VHDL-description produced by the automatic transformation maintains the original hierarchy of SA/SD. The hierarchy is formed by structural description of VHDL. One advantage of this approach is that the VHDL description can be partitioned and then simulated in smaller parts.

The graphical data flow diagrams of SA/SD are mapped to structural descriptions of VHDL. The minispecifications and state transition diagrams, which are located at the lowest level of the hierarchy, are mapped to behavioural descriptions. The Ada-like structures used in minispecifications have nearly similar structures in VHDL.

Fig. 2 and Fig. 3 are examples of correspondences between SA/SD elements and VHDL code.

![Diagram](image-url)
### 3.3 The Transformation Method

A graphical CASE SA/SD tool Prosa [8] is used for entering SA/SD diagrams to PC/ATs or workstations. The Sokrates-SA compiler produces target code from Prosa-files. The compiler is a rule-based knowledge system developed in a Finnish national research project. It can generate different target codes from the same SA/SD diagram on the basis of language specific transformation rules.

Sokrates-SA compiler generates target code in two phases. First the compiler translates the Prosa-files into intermediate description (Fig.4). This is usually performed in PC/AT computer. This general component model is common to all target languages. Then it is transferred to VaxStation 2000 workstation, where the kernel of the compiler is situated.

In the workstation environment the translation is continued by typifying the general component model to typified component model. It is further compiled by means of language specific transformation rules into a syntax tree. The final target code is produced from the syntax tree by using printing rules.

### 4. RESULTS

The SA/SD method is widely used in electronics industry for software specification in Finland. Many companies are adopting the SA/SD method also for hardware specification and design.

Transformation rules for DABL have been developed earlier. Several examples have been translated to DABL and simulated. The simulation results were consistent with the semantics of Sokrates-SA diagrams.
Transformation rules for VHDL-language have been developed. The rules were coded in If-Prolog and the number of the rules is about 60.

The developed transformation rules have been tested by compiling several Sokrates-SA diagrams to VHDL. One of the designs was a fluid level controller example. It was composed of three data flow diagrams and one state transition diagram. The size of the resulting code was about 400 lines.

The transformation time for the fluid level controller from Sokrates-SA format to VHDL code was about fifteen minutes. A simple turning lights example, which consists of two state transition diagrams, takes about ten minutes. The size of the VHDL code was about 200 lines.

The syntax of resulting code has been checked by CAD Language Systems VHDL Analyzer. The syntax was found to be correct according to IEEE 1076-1987 standard. Also preliminary simulation results of the generated VHDL code have been achieved.

5. CONCLUSIONS

The main advantage of the proposed method is the ability to convert a formal high level specification into a behavioural HDL description, which can be simulated. The verified specification contains all the required functions of the system and further design phases are based on it. The use of system-level verification greatly reduces the time spent in lower design levels, because specification errors can be detected and corrected already at an early stage.

SA/SD method has strong descriptive power and it can be used in all design phases from logical behaviour modelling to implementation design. The presented automatic transformation helps to combine the advantages of a graphic specification method and system-level verification with VHDL. As a standardized language VHDL can permit the movement of design data between different design automation tools: simulators, synthesis tools, logic optimization tools and silicon compilers.

6. ACKNOWLEDGEMENTS

We are grateful to Prof. Jorma Lammasniemi and Prof. Seppo Säynäjäkangas for supporting the project team. We will also thank Mr Antti Auer and Mr Ari Okkonen for their helpful advice. This research work was sponsored by the Technology Development Centre of Finland and a number of industrial companies in the National Microelectronics Programme in Finland.

7. REFERENCES