Published in IEEE International Symposium on Circuits and Systems (ISCAS), 387-390, 1994
which should be used for any reference to this work

High-Level Specification of Behavioural Hardware Models with MODES

Hans Peter Amann[1], Philippe Moeschler[1], Fausto Pellandini[1], Alain Vachoux[2], Charles Munk[2], Daniel Mlynek[2]

[1]Institute for Microtechnology
University of Neuchâtel
Rue de Tivoli 28
CH-2003 Neuchâtel, Switzerland

[2]Integrated Systems Center
Swiss Federal Institute for Technology
EPFL-Ecublens, DE-C3i
CH-1015 Lausanne, Switzerland

ABSTRACT

The ever increasing complexity of digital hardware, forces the hardware designer to switch from a low-level capture-and-simulation process to a high-level design-and-synthesis process. In order to assist the designer during the specification phase, we designed MODES, a new high-level specification environment. MODES integrates graphical specification capture in various behavioural representations, a facility to merge various specification formats, a high-level simulation tool and, finally, code generation for simulation and synthesis, e.g. in VHDL-1076, IEEE's hardware description language. Furthermore, some verification and consistency-check mechanisms have been integrated.

1. INTRODUCTION

In the last few years, the hardware design process began switching from a low-level capture-and-simulate process to a high-level design-and-synthesise process. This new paradigm shift allows a design to be described on higher abstraction levels with the final implementation achieved through automatic synthesis instead of manual refinement, thus relieving designers from the tedious tasks of logic mapping and optimisation [1].

While the specification formats at the logic level are straightforward, a design at higher levels of abstraction can comprise many different descriptions; the designer should be free to choose the most convenient one for each part of the design. Furthermore, to increase the acceptance of the high-level design approach, specific CAD-tools must be available, enabling for capture and merge of specifications, user assistance as well as automated code generation in a standard HDL language, e.g. VHDL.

We distinguish three sets of problems: i) the (graphical) capture tools for high-level design specifications and their respective formalisms with syntax and semantics, ii) the merge of data specified using various tools, the common internal format and the consistency checks and verifications which can be applied to it, and iii) the output code generation for synthesis and simulation, in the later case optionally with accompanying code for tests and simulation at the HDL level.

Several attempts have been made to rise the level of abstraction in hardware modelling. The StateChart approach [2] uses state diagrams for the description of the behaviour of blocks of hardware. It formed the base for StateMate [3] system and the SpecCharts language [4], both designed for the generation of VHDL model from state diagrams. In the view of a more general approach to system level design, DEBYS/CAS [5] - a knowledge-based system for computer-aided specification of mixed digital, analogue, optoelectronic, etc. units - as well as MCSE [6], a top-down specification methodology for digital hardware - and the respective tools - have to be mentioned.

In the present paper, we first present the global concept of MODES, defining a black-box behaviour (§2). In paragraph 3, we list the specification capture tools a user can dispose of for his design. The captured data, internally stored using a particular formalism for each specification tool, has then to be transformed to a common internal representation such that they can be merged (§4). Furthermore, this representation eases in an important way the application of consistency checks and verification algorithms whose results can be back-annotated in the original specifications (§5). This same representation can also be used as an input to the internal high-level simulator (§6) which allows i) the animation of the specifications at the level of the graphical specification tools and ii) the display in the form of waves as well as in log files. Paragraph 7 is devoted to the output code generation. Paragraph 8 presents briefly the framework on which MODES is built. Finally, the results and the open issues are discussed.

2. THE CONCEPT

MODES has been conceived in order to ease the user's specification job by increasing the level of abstraction from the HDL (Hardware Description Language) level to a more abstract and convenient one [7].

The user shall no more be confronted with the tedious details of HDLs, and, important for the continuously moving market of hardware synthesis tools, become independent of the language sub-sets supported. The MODES project aims to lower the level of expertise needed for the realisation of behavioural models, promoting hence a top-down specification approach for digital hardware devices.

Figure 1 shows the basic concept of MODES. The high-level specifications are introduced module by module using various behavioural editors. A block editor or project editor enables to express the connectivity between the modules. The formalised internal representations of the captured designs are then combined and can be translated to simulatable and/or synthesisable HDL code.

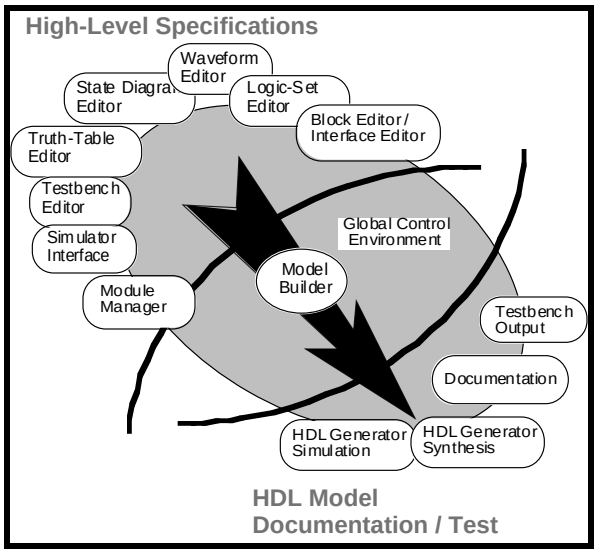

Figure 1. The concept of MODES

Since we do want to lower the expertise required for high-level specifications, we give the user also the possibility to observe the simulation at the same level of abstraction used for the specification. Therefore, MODES incorporates a high-level simulation unit, working at the level of the internal representation and enabling the user to visualise the results of a simulation either in the designs themselves or in the more classical form of signal wave forms.

3. SPECIFICATION CAPTURE

We distinguish two types of editors: i) behavioural editors for the description of the functionality. ii) structural editors for the specification of the connectivity, this being reduced to an interface specification for the case of a single module.

Block editor or project editor / interface specification editor
Each design can be specified as a single behavioural unit. Nevertheless, it may be interesting to split up a design, either for conceptual reasons or for convenience, enabling the user to specify each module with the best fitting behavioural editor. Therefore, and also in order to be able to integrate previously specified blocks (§4), a schematic or block editor with hierarchy has been integrated. Implicitly, it allows to specify the I/O pins of new modules to be created and to reuse the interface information of pre-existing modules.

BEMCharts: an editor for extended finite state diagrams
Classical finite state machines (FSM) reflect the control structure of a design through their states and transitions. Our extended FSM approach - BEMCharts[8] - is based on [2], [3]. The implementation, which also has been commercialised independently with many extensions [9], enables the user to specify concurrent and hierarchical annotated FSMs. The approach is particularly well adapted, but not limited, to the specification of controller circuits and communication protocols.
The *hierarchy* concept allows the refinement of the functionality of a state at a given level in a diagram one level down, enabling for a better readability of the design, an important goal of top-down design. A single state can also be refined into several *concurrent* sub-diagrams which will be executed in parallel. The actionblocks permit the specification of sets of actions at different places. Placed on transitions, they enable the specification of actions to be undertaken when the respective transition is triggered, placed at the entry respectively exit of a state, they are triggered for all inbound and outbound transitions of the corresponding state. In order to support *data-path* constructs, they can also be placed inside a state or as an independent unit in a sub-diagram: the action block is asynchronously evaluated as long as the respective state or sub-diagram is active.

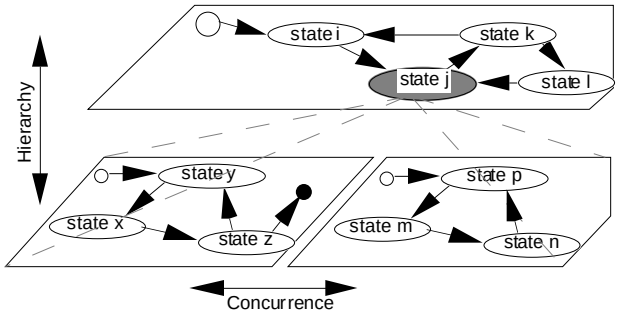

Figure 2. Concurrent and hierarchic finite state machines

Extended Timing Diagrams ETD
Recently, the ETD (Extended Timing Diagrams) formalism has been presented [10], extending traditional timing diagrams - which represent signal wave forms at the I/O pins - by adding timing constrains and action blocks with the respective triggering conditions. This tool (or formalism) supports hierarchy and concurrence as well.

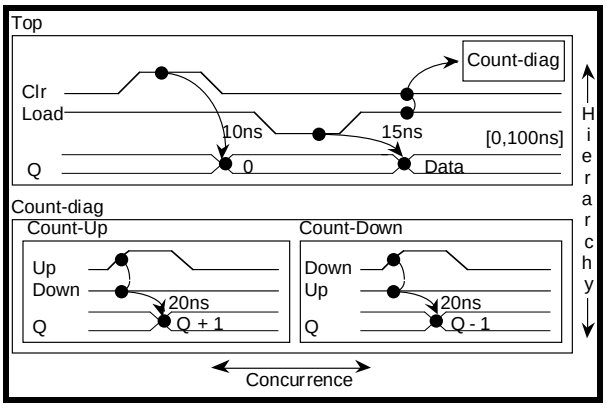

Figure 3: Extended Timing Diagrams ETD

Extended Truth-Tables ETT
Combinatorial logic can be specified conveniently using truth-tables. Again, the basic formalism has been extended such that clocked elements are also covered: Mealy and Moore machines can also be specified.

Utilities
Beside the model specification tools, the user disposes of a set of utilities. A *stimuli editor* permits the specification of one or more processes stimulating the model designed during simulation, both at high-level and - after code generation for a corresponding test bench - at HDL level. A more advanced version - the *test bench editor* - allows also the specification

of the desired results, and the generation of the code reporting differences. A *logic set editor* enables the user to define the logic value set and the corresponding logic functions and resolution functions.

4. STORAGE AND REUSE OF MODULES

MODES uses a specific knowledge base which constitutes a repository of all the information that will be required, i.e. modelling rules and previously instantiated designs, to create new behavioural models without having to start from scratch. The *module manager* uses the knowledge base to provide the user with a global design control system. On one hand, such a system provides a simple save/restore capability to allow the user to complete the design in multiple sessions. On the other hand, it provides access to whole or parts of previously completed designs to allow extensive reuse. To do so, the specifications are stored according to a predefined schema which defines the different types of models and functions and their attributes (Fig. 4).

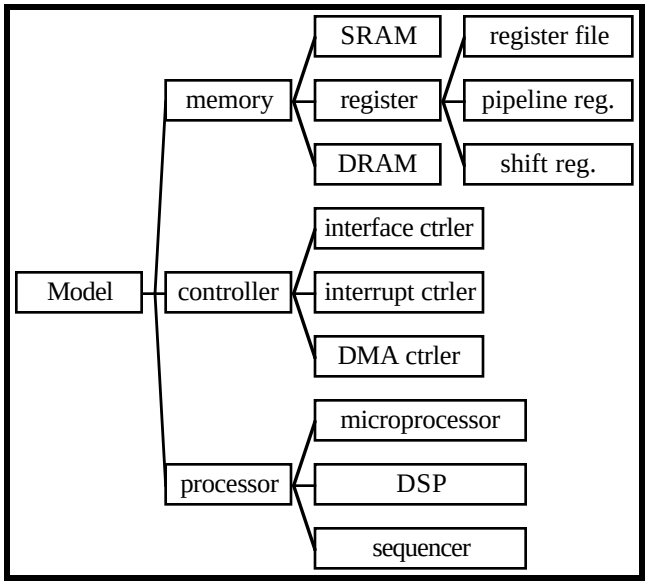

Figure 4: Schema definition (partial)

5. INTERNAL REPRESENTATION, CONSISTENCY CHECKS, VERIFICATION

Each one of the editors presented previously represents and stores a specification according to its own format. It must not understand how the other editors are working. This is essential to permit future additions of new editors as necessary. On the other hand, MODES provides several editors to allow the user to cope with complex designs by selecting the appropriate tool. Therefore, the validation process inside each specific editor is not sufficient to validate the global specification constituted by the union of all individual specifications.

In order to overcome this problem, all individual specifications are merged by the *model builder* into a *Common Specification Intermediary Format* (CSIF) [12]. CSIF allows to represent all specifications in a unique format such that the capturing process is separated from other processes which may directly use the merged specifications. Examples of such processes are: extraction of global properties, consistency verification, HDL translation. Having a single format at that stage allows to develop tools performing the latter processes independently of the input editors.

The representation of the specification in CSIF keeps all information related to the structure of the specification (i.e. hierarchy, concurrence). It therefore respects the way the user has entered the specifications. This is very important to allow the reuse of the specifications.

6. HIGH-LEVEL SIMULATION, ANIMATION

In order to verify as soon as possible the correct functionality of a design, a simulation must be possible at the level of abstraction used before for the specification. Thus a simulation and animation tool is integrated into MODES.

The simulation process runs top-down through the hierarchy. At each level, concurrent modules - sub-diagrams for behavioural editors or blocks for the schematic editor - are evaluated separately according to their input signals' sensitivity list. For certain states, evaluations are iterated such that immediate, asynchronous changes can also be handled. The high-level simulator performs a dead-lock check such that infinite simulation loops can be avoided. Timing constraint checks (set-up, hold, min. pukse width) are applied and violations reported.

The user is given the possibility to visualise the results of the simulations in two different forms: either i) in the well known form of timing diagrams, displaying both the values of the various variables and the activation of the FSM's states or ii) in the form of high-lighted states in the original specifications. Due to the merging method used, this approach can easily be extended to ETDs and ETTs, permitting simulation and animation in the respective specification environments.

7. OUTPUT CODE GENERATION

For the integration of the modules providing from the behavioural formalisms mentioned, we take profit from the fact that all of them have several common points: they are driven by controlling units (FSM, Mealy and Moore machines, sequences of events), can fire blocks of actions which then can - among others - represent combinatorial logic and, finally, integrate hierarchy and concurrence features. We therefore can limit us to the generation of VHDL code for the most general of the formats, concurrent and hierarchical, annotated FSMs.

The modules are combined according to the structure expressed by the user, using i.e. a block or schematic editor. As a result, we get a single model which can be translated into the desired output HDL code.

The translation into behavioural VHDL code for simulation is straightforward and the functional mapping unambiguous thanks to the evaluation scheme used. The controlling FSMs are translated into a first set of processes, the data-path elements, and the timing checks, are put in additional processes.

For synthesis, the situation is much different: i) every

commercially available synthesiser just supports a (different) subset of VHDL, ii) synthesisers prefer or even need synchronous designs, iii) after statements and timing checks cannot be handled, etc. In order to keep control over hardware synthesised for the controlling FSMs, a synchronous implementation has been chosen while the data-path elements can be integrated in asynchronous hardware.

8. GLOBAL CONTROL ENVIRONMENT

Due to the complexity of the MODES system it appeared soon that the system would by constituted of several tools working in close co-operation. In order to evaluate rapidly the interdependencies of different tools, we have elaborated the Global Control Environment (GCE) tool [11].

The GCE tool aims to build rapidly prototypes of the whole system with a minimum of coding. It also provides different mechanisms that enforce the construction of the MODES system in a framework fashion.

The prototyping approach proposed by the GCE tool is based on the fact that the user interface of the whole CAD system is described through the use of an interpreted language.

9. RESULTS AND OPEN ISSUES

A sub-set of MODES have been commercialised in the SPeeDCHART product from Speed Electronic SA. The coupling with existing synthesisers (Compass, Synopsys) provides the user with a complete top-down design methodology.

A very important feature of MODES lies in the module manager and the model builder. However, only a basic implementation of them has been done so far. For example, an efficient way to specify and apply modelling rules to guide the user is still to be studied further.

10. CONCLUSIONS

The aim of the MODES project is the realisation of a high-level specification system for the automated generation of behavioural models of blocks of hardware, thus allowing the reduction of both the time and the modelling experience required for the generation of behavioural models. Starting from the captured specifications expressed in the form of annotated and extended state machines, truth-tables and timing diagrams, a common internal representation is generated which first can be simulated, using a high-level simulation tool, and then translated either into code for simulation and documentation at HDL level or in more particular sub-set HDL code for hardware synthesis using one of the commercially available synthesis tools.

Furthermore, some facilities for verification and consistency checks as well as for test bench generation at the HDL level have been integrated.

A sub-set version of MODES has been successfully commercialised [Dar93], encouraging us to continue in our work, focusing or interest on module storage and reuse, the internal representation (formalism) and consistency check and verification.

ACKNOWLEDGEMENTS

This work, funded by the Swiss commission for the encouragement of scientific research CERS/KWF under project numbers 2081.2, 2260.2, 2499.1 and 2609.1, has been realised in collaboration with P. Ukelo, D. van den Heuvel, K. Bettaieb, S. Dart and R. Zinszner.

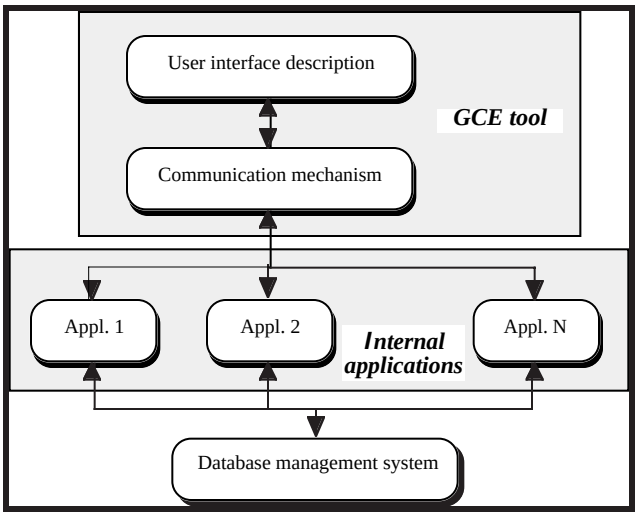

Figure. 5. GCE tool in a CAD system environment

REFERENCES

[1] D. Gajski, "Design process beyond ASICs", keynote speech, Proc. EDAC-EuroASIC, Paris, Feb 1993, pp3-4.

[2] D. Harel, "StateCharts: A Visual Formalism for Complex Systems", Science of Computer Programming 8, Elseiver Science Publishers, 1987, pp231-274.

[3] D. Harel et al., "StateMate: A Working Environment for the Development of Complex Reactive Systems", IEEE Transactions on Software Engineering, Vol. 16, No. 4, April 1990, pp403-414.

[4] F. Vahid et al., "SpecCharts: A Language for System Level Synthesis", in "Computer Hardware Description Languages and their Applications", Editors D. Borrione and R. Waxman, Elsevier, 1991, pp165-174.

[5] K.D. Mueller et al., " An approach to Computer-Aided Specification", IEEE Journal on Solid-State Circuits, Vol. 25 No. 2, April 1990, pp335-345.

[6] J.P. Calvez, "Embedded Real-Time Systems", John Wiley & Sons, Chichester UK, 1993.

[7] H.P. Amann et al., "The MODES modelling expert system", Proc. EURO-VHDL'91, Stockholm, Sept. 1991, pp192-195.

[8] D.O. van den Heuvel et al., "High-Level behavioral Modelling using BEMCharts", Proc. ECCTD, Davos, , Aug./Sept. 93, pp211-216.

[9] S. Dart, "SPeeDCHART-VHDL reference manual V2.2.0", Speed Electronic, Neuchâtel, July 1993.

[10] Ph. Moeschler et al., "High-Level Modeling using Extended Timing Diagrams", Proc. Euro-VHDL'93, Hamburg, Sept. 1993, pp.

[11] Ch. Munk et al., "The MODES Global Control Environment: A tool for Rapid Prototyping", Proc. EuroDAC'93, Hamburg, Sept. 1993, pp338-343.

[12] Ch. Munk et al., "The MODES Common Specification Intermediate Format", submitted to EuroDAC'94.