DAME: A RULE BASED DESIGNER OF MICROPROCESSOR BASED SYSTEMS§

Nikitas J. Dimopoulos, Kin F. Li, Eric G. Manning
Department of Electrical and Computer Engineering
University of Victoria
Victoria, B. C.

ABSTRACT

In this work, we present the overall structure of DAME, which is an expert system capable of configuring and designing customized microprocessor based systems from original specifications.

DAME [3, 4] , is organized as a hierarchy of design levels, each one of which refines the design provided by the previous level, by following established practices in the field of hardware design.

In this work, we shall present the general structure of DAME as well as our approach in modelling the behavior of the various signals associated with the components used for the design of microprocessor based systems.

Introduction

In many Systems' design problems, the lack of comprehensive theory of system integration and design choices, has led to a more or less empirical set of rules, which an experienced designer can draw upon in order to give an optimum solution to a given problem. Examples can be drawn from several diverse fields such as patient care, computer system configuration, geological exploration, VLSI Design, computer system design etc.

Knowledge-Based systems have recently proliferated in several fields of human endeavor. These systems play the dual role of categorizing and codifying expert knowledge, and then using this knowledge in order to solve time consuming and/or challenging problems.

Initially, expert systems were developed that were capable of analyzing a certain set of facts and suggesting plausible explanations or interdependencies. The fields of expertise were drawn from such diverse fields as medicine (e.g. MYCIN[16], EMERGE [9] etc.) analytical chemistry (DENDRAL[10]) and oil field exploration (PROPECTOR [5]).

Recently though, several attempts have been made of producing systems capable of synthesis. The most celebrated example is R1 [11] which is capable of configuring VAX computers. Given a customer's order, R1 determines if the order is consistent and complete, otherwise it is modified to meet the completeness and consistency criteria, and it produces a set of diagrams depicting the components included, their arrangement and interconnection.

Several examples of systems capable of automated logic design are also cited. Uehara [17], describes a knowledge-based synthesizer which transforms a technology independent functional design of a system, to a technology dependent gate level design.

VEXED [13] provides interactive aid to the user in designing a circuit. Initially the circuit is represented as a module whose functional specifications are entered by the user. Then VEXED together with the user refines the module into submodules which themselves are further refined into subsubmodules until the design is finished. VEXED+ [14] provides focusing of attention and selects the best refinement rule that meets the current design goal.

SOCRATES[2] optimizes combinational logic for a specified target technology; it performs substitutions of combinations of gates with equivalent simplified configurations, thus reducing the area and time complexity of the design.

ELF [15] is a prototype generator for wire routing applications. Given an application environment such as routing planes, cell expansion strategy etc. ELF is capable of generating a router specifically tailored to the application environment.

Additionally, several expert system development environments have been constructed [8] that help the collection of knowledge and the effective construction and debugging of the resulting production system. The first expert systems produced (such as MYCIN) were written in LISP. The Inference engine of MYCIN was isolated and made available (termed EMYCIN[12]) so that coupled with knowledge of a particular field would produce a distinct production system. The sequence of OPS [1,7] languages produced at Carnegie Mellon, combines a forward chaining reasoning mechanism together with an efficient pattern matching (the Rete algorithm [6]) to produce a good environment for Expert System construction. Subsequent generations of tools such Knowledge Craft encapsulate several environments (such as OPS-CRL, PROLOG, the Palm Editor etc.) plus a user friendly interface for increased productivity and application control.

The aim of this work, is to establish the framework for an expert system capable of configuring and designing a customized microprocessor based system from original specifications such as type and application, environment, communication and computational requirements as well as economical criteria.

DAME (Design Automation of Microprocessor based systems, using an Expert systems approach) will be capable of interpreting the design specifications, make appropriate choices of system configuration and components, and finally produce a complete design of the specified system.

We postulate that such an expert system, can be easily constructed, since most of the interfaces used by the various microprocessors and related peripherals are standardized. Thus, once the gross structure of the design and the modules comprising it have been chosen, their interconnection is fairly straight-forward.

In this work, we will specifically discuss the overall structure of DAME comprising several hierarchical levels, and we shall discuss our approach in modelling the various signals found in the components that are used in the design process.

Structure

In designing a microprocessor based system, the designer goes through a formal design process incorporating the following phases: (1) Design Specification (2) Configuration (3) Behavior Description (4) Functional Block Design (5) Implementation and Integration.

During the Design Specification phase, the system responsibilities, design constraints, and system environment are established. This phase involves considerable consultation between the designer and the customer, so as the various design criteria could be clarified.

The gross system architecture, is established during the Configuration Phase. During this phase, the system is divided into subsystems which are finally interconnected to produce the gross system architecture. The subsystems known and used by the Configuration phase are Memory, Processor, I/O and Bus.

The Behavior Description phase, defines the capabilities of the subsystems produced by the Configuration phase. For example, it is during this phase that the choice of 16-bit over an 8-bit processor is made, the size of the memory, the number and type of I/O channels are established, as well as the decision on the type of standard to be followed for the System Bus is made.

During the Functional Block design phase, the capabilities (i.e. functions) of the Subsystems, are further refined into simpler functions known to map directly into available components or

§ Supported in part by the Natural Sciences and Engineering Research Council of Canada under the strategic grant #STR0040526

combinations thereof. For example, in designing the memory module, (the size and speed of which have been specified during the Behavior Description phase) the Functional Block design phase choses the type of memory (e.g dynamic or static), determines the requirements of the address decoding module, and finally determines whether refreshing and/or error correcting modules are necessary. It also specifies the characteristics of these modules.

During the Implementation and Integration phase, the modules obtained during the Functional Block design phase are connected together to produce the final system. Functions specified during the Functional Block design phase which do not directly correspond to a hardware component, are synthesized, at this point by using random logic.

In DAME, the formal design process as described above, is followed. Thus, DAME has a hierarchical structure with levels that correspond exactly to the phases of the formal design process as described above. The overall structure of DAME is given in Fig. 1.

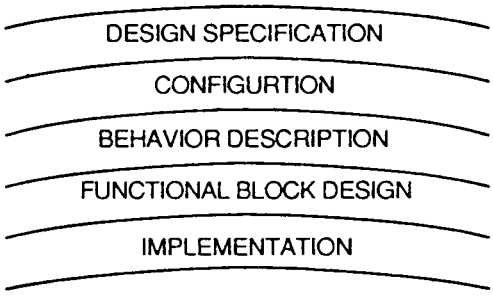

Figure 1. The Hierarchical Structure of DAME.

Each hierarchical level represents an abstraction of the given design problem. As the levels are transversed, the abstraction of the design is refined, until at the last level the complete design is formed.

Each hierarchical level manipulates objects which represent the system's concept of the design requirements at the particular abstraction of the level. These objects are refined by the current hierarchical level and the resulting objects are communicated to the next level which repeats the process. Each level may itself be partitioned in more sublevels which in turn compute particular requirements within a given level.

The types of objects that are known to a particular level are the following:

(1) Concept Classes (CC). These describe abstract sets of objects possessing similar properties. Examples of Concept Classes are Computer Systems, Processors, Memory, Signals, Signal Timing etc.

(2) Individual Objects (IO) are particular representations of specific Concept Classes, in the sense that obtain terminal values for the properties described in a general way within the Concept Class. Examples of Individual Objects are MC68000 (a member of the CC 16-bit processors, which itself is a member of the CC processors). Also, a unibus structure is a member of the CC Computer Systems.
Individual Objects and Concept Classes are shared between hierarchical levels so as a common language between levels is established.

(3) Relation Descriptors (RD) characterize relations between Concept Classes and Individual Objects. Examples of such relations are *IS_A* which describes membership in a class, *HAS_TIMING_SPECS* which relates timing specifications of a signal to the signal itself, *IS_CONNECTED_TO* which expresses the pin connectivity, and through which the whole design will be expressed.

(4) Facts (F) constitute "knowledge units" which embody descriptive information that is used to make inferences. Facts are inferred, or are incorporated within each of the levels (and they are pertinent only to that particular level). For example, the fact *A decoupling capacitor must be connected between VCC and GND pins of all the dynamic RAM chips* is found in the system integration and implementation level, and the objects *decoupling capacitor, dynamic RAM* etc. are known only in this level. On the other hand the fact *The chosen processor is a 16-bit processor*, is passed to the Functional Block design level from the Behavior Description level where it was inferred by a rule of the form: *If large addressing capabilities, then choose a 16-bit processor.*

(5) Production Rules (PR) describe the static and dynamic behavior of the objects of the design. They are used in each level in order to refine the abstraction of the design.

We are using Knowledge Craft™† as our implementation platform. Knowledge Craft supports "schemata" through which the objects and relations outlined above, can be expressed in a hierarchical fashion. Relations allow schema slots, holding particular information to be inherited in a controlled way from object to object in the Hierarchy. We shall be describing in detail in the following section the objects and relations necessary to describe certain signal classes and their timing. We shall also give their implementation by using schemata.

Describing Signals

We consider signals to be associated with a particular physical port (in the case of a component, a pin) to have a name, to follow a certain type of logic (i.e. to be active either high or low) and to be related to other signals. Through these relations, one understands the behavior of a particular signal especially its timing behavior. For example, the address strobe of the MC68000 is associated with pin# 6 in the DIP package, it has the name AS, it is active low, it is an output signal (relative to the component) and it is related with the data transfer acknowledge, in the sense that the assertion of the data transfer acknowledge forces the negation of the address strobe after a predetermined time interval. The complete specification of the timing interdependence of these two signals is given in Fig. 2.

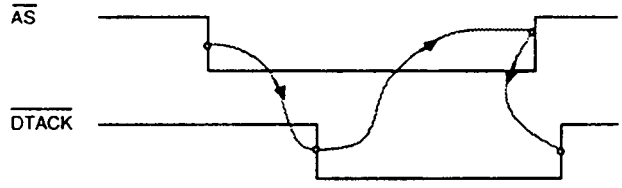

Figure 2. The timing interdependence of the $\overline{AS}$ and $\overline{DTACK}$ signals of the MC68000.

In modeling signals such as the $\overline{AS}$ and $\overline{DTACK}$ described above, we have constructed the following objects.

SIGNALS This is an object class that acts as the template of the objects to be described. It incorporates slots for the pin#, the type of logic a particular signal is following, and the direction of the signal.

INDIVIDUAL_SIGNALS These are objects describing the individual signals comprising the class of SIGNALS for a particular component. INDIVIDUAL_SIGNALS are related to the class of SIGNALS through an IS-A relation through which they inherit all the slots of the parent class SIGNALS. These slots are then filled individually with pertinent values. Examples of such objects are $\overline{AS}$, $\overline{DTACK}$ etc.

TIMING_SPECS is a class of objects that acts as the template for the objects that describe the timing specifications of the individual signals.

† Knowledge Craft is a trade mark of Carnegie Group Inc.

INDIVIDUAL_SIGNAL_TIMING_SPECS These are the objects that describe the timing specifications of individual signals. They are related to the class TIMING_SPECS through an IS-A relation, and they inherit all slots found in the class TIMING_SPECS. They are also related to the INDIVIDUAL_SIGNALS objects through the IS_TIMING_SPECS_OF relation.

We have also defined and used the following relations

HAS_TIMING_SPECS and its inverse IS_TIMING_SPECS_OF. This relation is used to associate the object describing an individual signal to the object that describes the signal's timing specifications. This relation has a transitivity of one, requiring that the signals influencing each other to be directly related, and it does not convey any slots during inheritance. That is the timing specifications of a signal must be directly related to the signal itself.

The following four relations

LO_HI_TRIGGERED_BY_LO_HI
LO_HI_TRIGGERED_BY_HI_LO
HI_LO_TRIGGERED_BY_HI_LO
HI_LO_TRIGGERED_BY_LO_HI

convey the name of the signal whose transition forces the indicated transition to the domain signal. Thus the low to high transition of the $\overline{AS}$ resulting from a previous high to low transition of the $\overline{DTACK}$ is described by relating the timing specifications object of the $\overline{AS}$ with the $\overline{DTACK}$ signal through the LO_HI_TRIGGERED_BY_HI_LO relation. These relations have a transitivity of one, requiring that the signals influencing each other to be directly related, and they do not convey any slots during inheritance.

The above mentioned objects and relations have been implemented as "schemata" in a Knowledge Craft environment. Their complete descriptions can be found in TABLE I, while the network describing the $\overline{AS}$ and $\overline{DTACK}$ signals and their timing relations can be found in Fig. 3.

Conclusions

In this work, we provided the framework of DAME which is an expert system capable of configuring and designing customized microprocessor based systems from original specifications.

DAME, is organized as a hierarchy of design levels, each one of which refines the design provided by the previous level, by following established practices in the field of hardware design.

We have postulated that the use of an expert system at this level of design activity is achievable, since the design methodology is well established, and the interfaces for most of the components used are standardized.

We presented our approach in modelling the behavior of the various signals associated with the components used for the design of microprocessor based systems. Specifically, we presented the modelling of two signals, namely the Address Strobe and the Data Transfer Acknowledge, as defined for the MC68000 microprocessor. We used objects and relations to capture both their static and temporal behavior. We implemented these objects by using *schemata* as defined in Knowledge Craft, and we presented their implementation.

The signals chosen for modelling are typical in that they implement the two signal handshaking protocol widely used by many microprocessors and their associated peripherals. We are in the process of modelling more complex behaving signals. Such an example is the Bus Request, Bus Grant, Bus Grant Acknowledge set, also used by the MC68000 and constituting a typical example of the implementation of a three signal arbitration protocol.

REFERENCES

1. Brownston L., R. Farrell, E. Kant and N. Martin, *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison Wesley, 1985.
2. de Geus, A. J. and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic" *IEEE Design & Test* , pp. 22-32 Aug. 1985.
3. Dimopoulos, N. J. and H. C. Lee, "Experiments in Designing with DAME: Design Automation of Microprocessor Based Systems using and Expert Systems Approach" *Proceedings of International Computer Symposium 1986* , pp. 1858-1867, Tainan, Taiwan, Dec. 1986.
4. Dimopoulos, N. J., C. H. Lee and N. Galatis, "DAME: Automated Design of Microprocessor based Systems, an Expert Systems Approach" *Proceedings of the Canadian Conference on Industrial Computer Systems* , pp. 20-1/ 20-7, Montreal, May 1986.
5. Duda, R. O., P. E. Hart, K. Konolige and R. Reboh, "A computer-Based Consultant for Mineral Exploration" *Technical Report*, SRI International, Sep. 1979.
6. Forgy , C. L., "Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem" *Artificial Intelligence* , vol. 19, pp. 17-38, Sept. 1982.
7. Forgy, C. L., "OPS5 User's Manual" Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, Jul. 1981.
8. Gevarter, W. B., "The Nature and Evaluation of Commercial Expert System Building Tools" *IEEE Computer* , pp. 24-41, May 1987.
9. Hudson, D. L., and T. Estrin, "EMERGE- A Data-driven Medical Decision Making Aid" *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 87-91, Jan. 1984.
10. Lindsay, R., B. G. Buchanan, E. A. Feigenbaum, J. Lederberg, *Applications of Artificial Intelligence for Chemical Inference: The DENDRAL Project* , McGraw-Hill Book Company, New York, 1980.
11. McDermott, J., "R1: A Rule-Based Configurer of Computer Systems" *Artificial Intelligence* , vol. 19, pp. 39-88, Sept. 1982.
12. Melle, W. Van, A. C. Scott, J. S. Benett and M. A. Peairs, "The EMYCIN manual" *Technical Report*, Heuristica Programming Project, Stanford University, 1981.
13. Mitchell, T. M., L. I. Steinberg and J. S. Shulman, "A Knowledge-Based Approach to Design" *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp.502-510, Sep. 1985.
14. Norton, S. W. and K. M Kelly, "Learning Preference Rules for a VLSI Design Problem-Solver" *Proceedings of the fourth Conference on Artificial Intelligence Applications* , pp. 152-158 Mar. 1988.
15. Setliff, D. E. and R. A. Rutenbar, "Knowledge -Based Synthesis of Custom VLSI Physical Design Tools: First Steps" *Proceedings of the fourth Conference on Artificial Intelligence Applications* , pp. 102-108 Mar. 1988.
16. Shortliffe, E. H., *Computer-Based Medical Consultation: MYCIN*, Elsevier, New York, 1976.
17. Uehara, T., "A Knowledge-Based Logic Design System" *IEEE Design & Test* , vol. 2 no. 5 pp. 27-34 Oct. 1985.

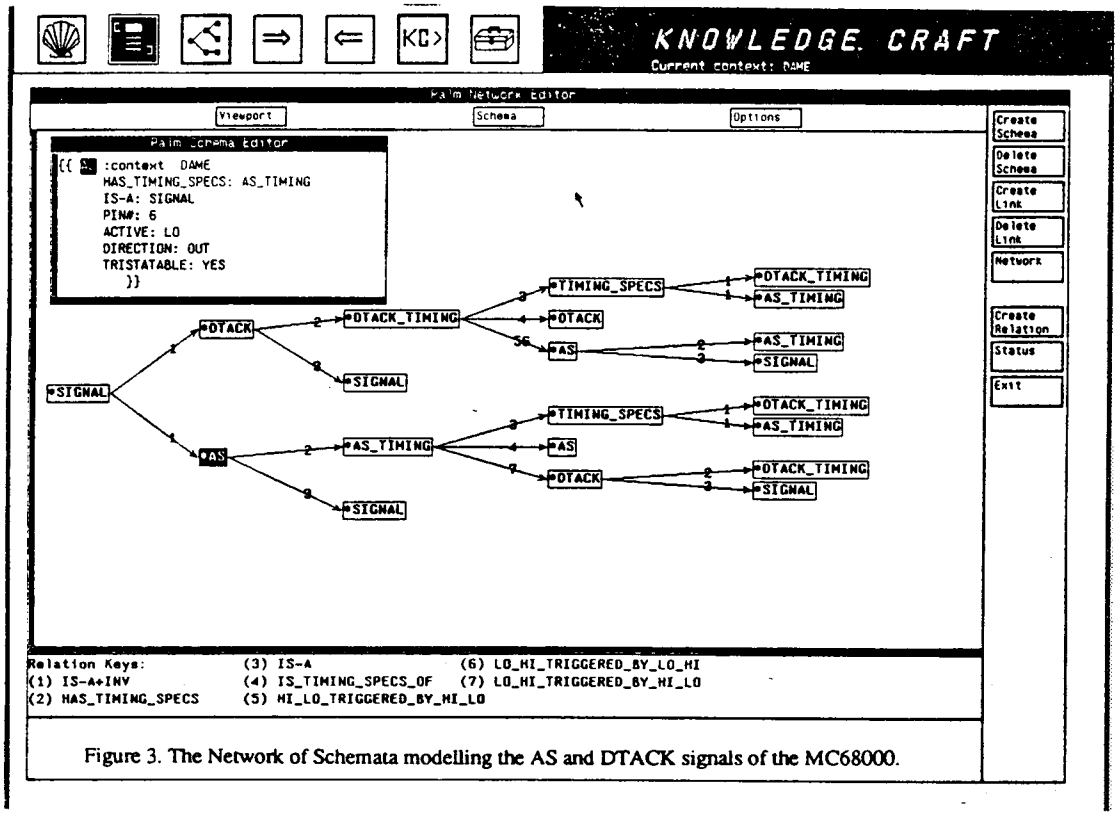

Figure 3. The Network of Schemata modelling the AS and DTACK signals of the MC68000.

TABLE I. Schemata Implementing Some of the Objects and Relations

{{ SIGNAL
IS-A+INV: AS DTACK
PIN#:
ACTIVE:
DIRECTION:
TRISTATABLE:}}

{{ AS
HAS_TIMING_SPECS: AS_TIMING
IS-A: SIGNAL
PIN#: 6
ACTIVE: LO
DIRECTION: OUT
TRISTATABLE: YES}}

{{ AS_TIMING
IS-A: TIMING_SPECS
IS_TIMING_SPECS_OF: AS
LO_HI_TRIGGERED_BY_HI_LO: DTACK
LO-HI-DELAY: (' (COND ((<= (GET-VALUE ' DTACK_TIMING ' HI-LO) 2) 2)
(T (+ (GET-VALUE ' DTACK_TIMING ' HI-LO) 2))))

{{ DTACK
HAS_TIMING_SPECS: DTACK_TIMING
IS-A: SIGNAL
PIN#: 10
ACTIVE: LO
DIRECTION: IN}}

{{ DTACK_TIMING
IS-A: TIMING_SPECS
IS_TIMING_SPECS_OF: DTACK
HI_LO_TRIGGERED_BY_HI_LO: AS
LO_HI_TRIGGERED_BY_LO_HI: AS
HI-LO-DELAY: <X>
LO-HI-DELAY: 0}}

{{ TIMING_SPECS
IS-A+INV: DTACK_TIMING AS_TIMING
HI-LO-DELAY:
LO-HI-DELAY:}}

{{ LO_HI_TRIGGERED_BY_HI_LO
IS-A: RELATION
DOMAIN: (TYPE IS-A TIMING_SPECS)
RANGE: (SCHEMA (TYPE IS-A SIGNAL))
TRANSITIVITY: (STEP LO_HI_TRIGGERED_BY_HI_LO T)
INVERSE: HI_LO_TRIGGERS_LO_HI_TRANSITION_OF}}

{{ HI_LO_TRIGGERED_BY_HI_LO
IS-A: RELATION
DOMAIN: (TYPE IS-A TIMING_SPECS)
RANGE: (SCHEMA (TYPE IS-A SIGNAL))
TRANSITIVITY: (STEP HI_LO_TRIGGERED_BY_HI_LO T)
INVERSE: HI_LO_TRIGGERS_HI_LO_TRANSITION_OF}}

{{ HI_LO_TRIGGERED_BY_LO_HI
IS-A: RELATION
DOMAIN: (TYPE IS-A TIMING_SPECS)
RANGE: (SCHEMA (TYPE IS-A SIGNAL))
TRANSITIVITY: (STEP HI_LO_TRIGGERED_BY_LO_HI T)
INVERSE: LO_HI_TRIGGERS_HI_LO_TRANSITION_OF}}

{{ HAS_TIMING_SPECS
IS-A: RELATION
DOMAIN: (TYPE IS-A SIGNAL)
RANGE: (SCHEMA (TYPE IS-A TIMING_SPECS))
TRANSITIVITY: (STEP HAS_TIMING_SPECS T)
INVERSE: IS_TIMING_SPECS_OF}}

{{ IS_TIMING_SPECS_OF
IS-A: RELATION
DOMAIN: (TYPE IS-A TIMING_SPECS)
RANGE: (SCHEMA (TYPE IS-A SIGNAL))
TRANSITIVITY: (STEP IS_TIMING_SPECS_OF T)
INVERSE: HAS_TIMING_SPECS}}