MODELLING SIGNAL BEHAVIOR IN DAME: A RULE BASED DESIGNER OF MICROPROCESSOR BASED SYSTEMS§

B. T. Huber, D. Caughey, K. F. Li, N. J. Dimopoulos,
D. Li, R. Burnett, and E. G. Manning
Department of Electrical and Computer Engineering
University of Victoria, Victoria, B.C., Canada V8W 2Y2

ABSTRACT

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) is an expert system that will be capable of configuring and designing a customized microprocessor system from original specifications.

We have postulated 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 straightforward. Our investigation into the modelling of signal behavior confirmed this premise.

In this work, we present the notation used to model signals in DAME. A notation is developed to allow complete specification of the static and temporal behavior of signals and their relation to other signals.

1. 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.

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. Examples can be drawn from several diverse fields such as patient care [5,10], computer system configuration [6], geological exploration [4], VLSI design [8,9], computer system design [11], etc.

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) [1,2,3] is an expert system that will be capable of configuring and designing a customized microprocessor system from original specifications such as type and application, environment, communication and computational requirements as well as economic criteria.

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.

DAME organizes the design process into a hierarchy consisting of 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. The gross system architecture is established during the Configuration phase. The system is divided into subsystems which are interconnected to produce the gross system architecture. The Behavior Description phase defines the capabilities of the subsystems produced by the Configuration phase. During Functional Block Design phase, the capabilities of the subsystems are mapped directly to the available components or combinations thereof.

During the Implementation and Integration phase the modules obtained during the Functional Block Design phase are connected together to produce the final system. Undefined functions which do not directly correspond to a hardware component, are synthesized using random logic.

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 requirement at the particular abstraction level. The signal and the signal timing for a particular component are two of the objects represented in DAME. In this work, the notation for the representation of the signals and their timing is described. The notation developed is then used to illustrate some typical handshaking protocols widely used in microprocessor-based systems. Actual examples modelling signals for the MC68000 microprocessor are also given.

§ Supported in part by the Natural Sciences and Engineering Research Council of Canada under the strategic grant #STR0040526 and the B.C. Science Council under grant SCBC #88 243

2 Signal Description

A model is needed to encapsulate the specifications of microprocessor components, into a data base that can be used by DAME, currently implemented in Knowledge Craft™†.

The data describing the component must include information required at each design phase. Examples of such information are: component name, manufacturer, cost, temperature range and reliability, speed of operation, package types available, power requirements and consumption, device pin definition, and especially timing and interface specification.

Note that not all information must be accessible at all levels of the design phases. However, it should be available when needed. The aim of this work is to develop a notation to describe the pin definition, timing and interface information.

The electrical, temporal, and logical behavior of a signal is predetermined by the specification given by the manufacturers. Each signal modelled is associated with a particular physical port (usually a pin on the component) and will be referred to by a symbolic name. This name must be unique to each pin on a component, but pins from different components may have the same name. The extension of the pin name with the component number will result to a unique name in the design.

With the device in operation, each signal pin will attain different electrical and logic states such as enabled, asserted or valid at different times. The notation to describe the signals should include all possible non redundant input and output states. A bidirectional pin can attain both the input state and the output state at different times. Open collector type output pins should also be covered with the notation. The state notation must cover all these possibilities without introducing inconsistencies or impossible states.

Any change of the state of one or more signals constitutes an event. An example of an event is the change of an output pin from tristate to enabled, or an input pin that changes from invalid input to valid input. A simple state transition event can be restricted by the state of other signals or by other events that precede or succeed it. This allows arbitrarily complicated events to be generated. For example, for a certain event to occur, three different events must occur in a specified order, and only when all three events occur in the correct order, is the complete event considered to have occurred. Because of the temporal restrictions placed on the signals, timing information must be embedded with the event for the specification to be complete.

If a transition can only occur if certain events precede it, a causal relation results which specifies the prerequisite event for a signal transition and its timing. A set of causal relations can be used to specify the complete timing characteristics of a set of interacting signals.

In our notation, causal relations are denoted by *causal_expressions*. The event on the left hand side of the expression is considered to be the prerequisite of the transition, denoted on the right hand side of the expression. *causal_expressions* are incorporated in the frames describing the timing behavior of the signals. We are collecting standard behaviors (such as the *two_signal_handshaking* and its variations, the *three_signal_handshaking* etc.) and making them available as behavior templates to be used in describing the behavior of any specific set of signals in a given component. Associated with the behavior templates, we are incorporating primitive design rules that will be used for the design of the glue logic necessary for the integration of the components into a working system.

3 Notation for Interface Signals

This section defines the syntax used to model the behavior of the various signals associated with the components used in the design of microprocessor-based systems.

3.1 Signal Names

A signal associated with a particular device or bus is assigned a name which consists of a unique string of characters.

signal_name:
    string_of_characters

This symbolic name will be used to reference the signal. No two signals are

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

CH2868-8/90/0000-1497$1.00 © 1990 IEEE

allowed to have the same name unless they are the same physical signal. If the same signal on different devices has different symbolic names, then the signal can be referenced using any one of the names.

3.2 Signal States

Signals are found to exist at well defined states. Each state corresponds to a particular physical condition of the signal in question. Most states will have well defined voltages associated with them. A signal may be an INPUT, or an OUTPUT, or both, signifying the direction of the information flow with respect to the device to which the signal is associated. Below, we present a short description of the various states a signal can attain.

3.2.1 VALIDI or INVALIDI

If a signal is an INPUT, there will usually be a timing restriction given on when the correct input is required. This state is called the VALIDI state, which stands for VALID INPUT. Any other time when the correct input is not required, the signal will be in the INVALIDI state.

3.2.2 ENABLED or TRISTATED

If a signal is an OUTPUT, it can be ENABLED, which means the pin is driving the signal to some voltage, or it can be TRISTATED which is equivalent to having the pin disconnected.

3.2.3 VALIDO or INVALIDO

A signal can be ENABLED, but it may not be correct or valid. This signal condition is called INVALIDO state for INVALID OUTPUT. If the signal is ENABLED and valid, the signal is in the VALIDO state for VALID OUTPUT. For example the address lines on a MC68000 will normally be enabled, but they will only be VALIDO (correct and usable by an address decoder) during the time when AS (address strobe) is at a low (0 volt) level. Any other time the address lines will be INVALIDO.

3.2.4 ASSERTED or NEGATED

If a signal is VALIDO or VALIDI it will be at one of two voltage levels (usually 0 volts or 5 volts). The two voltage levels are described by the two states ASSERTED or NEGATED. If a signal is defined as asserted low, ASSERTED implies the signal is at a low voltage level (usually 0 volts) while NEGATED implies the signal is at a high voltage level (usually 5 volts). For example the AS signal on a MC68000 is asserted low.

3.2.5 UNKNOWN_ASSERTED

If a signal is VALIDO, it is also ENABLED and either ASSERTED or NEGATED. If it is not known if the signal is ASSERTED or NEGATED, the signal is in an unknown state called UNKNOWN_ASSERTED (UNKNOWN_ASS).

3.2.6 STATE DEFINITIONS

The signal states are defined below:

signal_state: ASSERTED | NEGATED | UNKNOWN_ASS | VALIDO | INVALIDO | VALIDI | INVALIDI | ENABLED | TRISTATED | INPUT | OUTPUT

Signals can have more than one logic state. Certain states imply other states. For example a signal can be ENABLED and ASSERTED. This implies that the signal is also in the states of OUTPUT and VALIDO. The above signal states can occur in the following combinations:

ASSERTED	—> (VALIDI | VALIDO)
NEGATED	—> (VALIDI | VALIDO)
UNKNOWN_ASS	—> (VALIDI | VALIDO)
VALIDO	—> OUTPUT and ENABLED and (ASSERTED | NEGATED | UNKNOWN_ASS)
INVALIDO	—> OUTPUT
VALIDI	—> INPUT and (ASSERTED | NEGATED | UNKNOWN_ASS)
INVALIDI	—> INPUT
ENABLED	—> OUTPUT and (VALIDO | INVALIDO)
TRISTATED	—> OUTPUT and INVALIDO
INPUT	—> (INVALIDI | VALIDI)
OUTPUT	—> (ENABLED | TRISTATE)

In the above notation the | symbol implies that any one of the states separated by the | symbol can occur, but that all states separated by 'and' will occur if the state on the left of the —> symbol occurs.

The INPUT and OUTPUT states are presented in the above definition for the purpose of classifying a signal's attainable states. In the actual implementation for the definition of a signal and its syntax, the states INPUT and OUTPUT are not allowed in the specification.

3.3 State Expressions

The state of a signal is expressed by giving the signal state followed by the signal name:

state:
signal_state signal_name

States can be combined into more specific states using operators. The combination of one or more states using operators constitutes a state expression.

state_expression:
state
or_state_expression
and_state_expression

There are 2 operators to combine states:

The or operator '+' is used to represent the OR between two or more states: e.g. +(STATE1, STATE2). The resulting expression is true iff one or more of the arguments is true. The order of the arguments is unimportant. The or state expression becomes true at the instant when the first argument of the OR becomes true.

The and operator '^' is used to represent the AND between two or more states: e.g. ^(STATE1, STATE2). The resulting expression is true iff all of arguments are true. The order of the arguments is unimportant. The and state expression becomes true at the instant when the last argument of the AND becomes true.

state_list:
state_expression , state_expression
state_list , state_expression

or_state expression:
+(state_list)

and_state_expression:
^(state_list)

3.4 Transitions

If a signal changes state, a transition results. The symbol to represent the transition is '!'. The '!' symbol is surrounded by the old and the new states. For example: (NEGATED ! ASSERTED SIG1) represents the transition that occurs when SIG1 changes from NEGATED to ASSERTED. If the first state is missing, it is assumed that the signal changes state from the opposite state to the given state. For example (! ASSERTED SIG1) is equivalent to (NEGATED ! ASSERTED SIG1).

A transition can therefore be expressed as follows:

transition_expression:
! state
signal_state ! state

3.5 Transitions and States

At several instances, one needs to specify setup and hold times for a group of signals so that they will be stable during the transition of another signal. This is important in the description of complex timing patterns such as the ones occuring during a *read-modify-write* cycle. This can be expressed through a combination of a transition and state expression in the following manner.

transition_state_expression:
 # (transition_expression , state_expression)
 # (state_expression , transition_expression)
 # [min , max](transition_expression , state_expression)
 # [min , max](state_expression , transition_expression)

The significance of the above definition, is that one expects a certain state combination (denoted through a *state_expression*) to be stable coinciding with the specified transition, or to be stable within a time interval around the specified transition.

3.6 Events

A transition, or a combination of state and a transition, as defined above, is an event. The collection of one or more events which may or may not occur in a specified order also is an event:

event expression:
transition expression
transition_state_expression
or_event_expression
and_event_expression
non_associative_and_event_expression

There are 3 operators to combine events into more restricted events:

The or operator '+' is used to represent the OR between two ore more events: e.g. +(EVENT1, EVENT2) The resulting expression is true iff one or more of the events is true. The order of the arguments is unimportant. The time of occurrence of the event is at the instant when the first argument of the OR becomes true.

The and operator '^' is used to represent the AND between two ore more events: e.g. ^(EVENT1, EVENT2). The resulting expression is true iff all of arguments are true. The order of the arguments is unimportant. The time of occurrence of the event is at the instant when the last argument of the AND becomes true. A timing value can optionally be associated with the associative AND by preceding the list of events with a number. To be true the events in the AND must occur in the specified time interval. e.g. (^20(EVENT1, EVENT2)) implies that EVENT1 must occur within 20nsec of EVENT2.

The operator '&' is used represent the non associative AND between two events: e.g. &(EVENT1, EVENT2). The resulting event is true iff both events are true and occur in the specified order. The time of occurrence of the resulting event is at the instant when the last event is true. Timing information can optionally be associated with the non associative events in the form of timing values. The meaning is that the second event in the sequence must occur within the timing interval specified by the timing values: e.g. & [+10,+~] (EVENT1, EVENT2) means that EVENT2 must occur at least 10nsec after EVENT1.

event_list:
- *event_expression , event_expression*
- *event_list , event_expression*

or_event expression:
- *+(event_list)*

and_event_expression:
- *^(event_list)*
- *^number (event_list)*

non_associative_and_event_expression
- *&(event_expression , event_expression)*
- *&[min , max] (event_expression , event_expression)*

min:
- *number*
- *—*

max:
- *number*
- *+~*

The detection of an event is then used to specify the state transition of a given signal. This is expressed through the *causal_relation* construct which is defined below.

3.7 Causal Relations

A causal relation is defined as an *event_expression* that causes a transition to occur at a certain time by the use of the '—>' operator:

causal_expression:
- *event_expression —>transition_expression*
- *event_expression —>transition_expression @[min,max]*
- *event_expression —>transition_expression @ number*

3.8 Parsing

We have developed a parser (based on unix's YACC and LEX tools) that is capable of parsing the language defined above. We are currently capable of extracting transition precedence from descriptions signal behavior defined as per the discussion above and displying this information as a timing diagram. An example is shown in Figure 1, together with the user interface which we are currently developing.

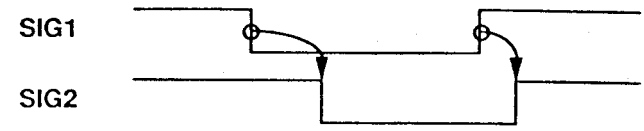
4 Templates for System Interface Signals

The following are some examples of the templates developed to represent some common signal interfacing that can be found in microprocessor systems. Each signal can either be an actual signal or each edge in a signal can represent a different event. Note that the following signals are assumed ASSERTED low.

Also presented are the timing relationships for some of the signals of the MC68000 microprocessor using the syntax of section 3 and the example templates introduced here. The timing values given are for a 12.5 MHz MC68000, according to Motorola specification [7].

A typical signal handshaking sequence occurs when one signal follows another as in the figure below:

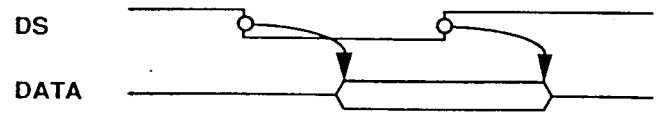
SIG2 follows SIG1



SIG2 follows SIG1

! ASSERTED SIG1 —> ! ASSERTED SIG2 @ T1
! NEGATED SIG1 —> ! NEGATED SIG2 @ T2

An example of the SIG2 follows SIG1 template is the timing of the LDS/UDS strobes and the data bus for the MC68000 microprocessor.



DS / DATA TIMING (READ)

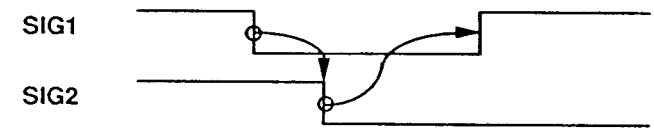
! ASSERTED LDS —> ! VALIDI D0-D7 @ [-~,(50 + T6)]
! NEGATED LDS —> ! INVALIDI D0-D7 @ [0,20]

! ASSERTED UDS —> ! VALIDI D8-D15 @ [-~,(50 + T6)]
! NEGATED UDS —> ! INVALIDI D8-D15 @ [0,20]

In the above equation, T6 is DTACK asserted delay from LDS / UDS signal in the LDS/UDS/DTACK timing.

Another typical template is shown below:

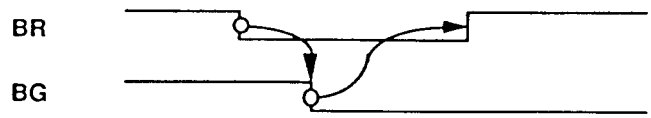
EVENT2 follows and triggers SIG1



EVENT2 follows and triggers SIG1

! ASSERTED SIG1 —> ! ASSERTED SIG2 @ T1
! ASSERTED SIG2 —> ! NEGATED SIG1 @ T2

This template can be used to describe the handshaking between the BR and BG signals for the MC68000 microprocessor:



ACTIVE BG follows and triggers BR

! ASSERTED BR —> ! ASSERTED BG @ [1.5CLK,3.5CLK]
! ASSERTED BG —> ! NEGATED BR @ [0,+~]

In the above notation BG will go asserted 1.5 to 3.5 clock cycles after BR goes asserted, and BR in turn can go negated any time after BG goes asserted.

6 Conclusion

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. Notations and templates for typical timing specification are presented with examples shown for the MC68000. We used objects and relations to capture both the static and temporal behavior of these signals. We implemented these objects by using *schemata* as defined in Knowledge Craft [3]. We have also developed a parser that is capable of interpreting signal timing specifications and used to create timing diagrams corresponding to the specifications.

Currently, we are implementing a user-friendly interface that will be used in capturing the relevant information and creating a network of schemata that will describe the properties of a component. Our approach is to use templates of networks of schemata describing partial properties and incorporate those templates into the final network. Our user interface is window based, mouse drivers and is implemented as a "work center" in knowledge craft. An example of this interface is given in Figure 1.

References

[1] 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.

[2] 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, Canada, May 1986.

[3] Dimopoulos, N.J., K.F. Li, and E.G. Manning, "DAME: A Rule Based Designer of Microprocessor Based Systems," *Proceedings of the 2nd International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, vol. 1, pp.486-492, Tullahoma, Tennessee, June 6-9, 1989.

[4] Duda, R. O., P. E. Hart, K. Konolige and R. Reboh, "A computer-Based Consultant for Mineral Exploration" *Technical Report*, SRI International, Sep. 1979.

[5] 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.

[6] McDermott, J., "R1: A Rule-Based Configurer of Computer Systems" *Artificial Intelligence* , vol. 19, pp. 39-88, Sept. 1982.

[7] *Motorola Microprocessors Data Manual*, Motorola Inc., Austin, Texas, 1985.

[8] 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.

[9] 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.

[10] Shortliffe, E. H., *Computer-Based Medical Consultation: MYCIN*, Elsevier, New York, 1976.

[11] Uehara, T., "A Knowledge-Based Logic Design System" *IEEE Design & Test* , vol. 2 no. 5 pp. 27-34 Oct. 1985.

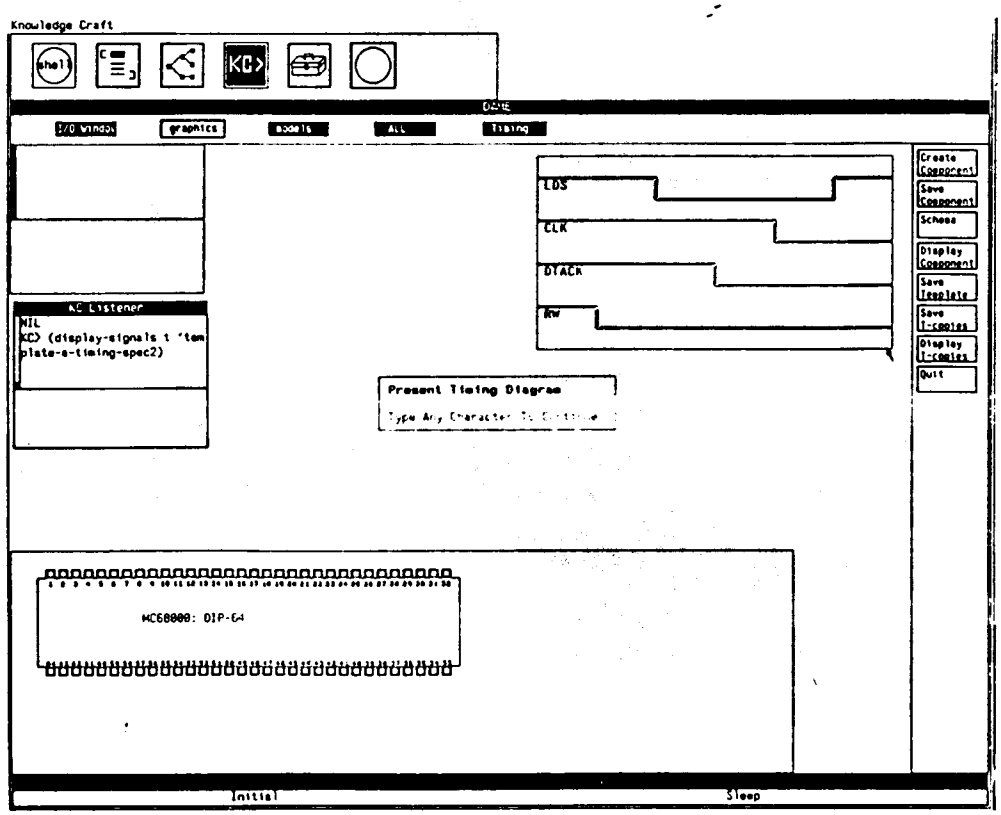


Figure 1. The DAME interface window with timing specification output.

```
Causal Relation for the timing depicted in the figure
&(&[20,+~](
            ^(
               &(
                  &(|NEGATED RW,|ASSERTED LDS),|NEGATED CLK),
                    |ASSERTED DTACK
                ),
               |NEGATED CLK
             ),
             |NEGATED CLK
           )
                              ->     |NEGATED LDS @ [0,50]
```