Microprocessor Components and Signal Behavior Modelling in DAME†

by

B. Huber, M. Escalante, D. Caughey, N.J. Dimopoulos,
K.F. Li, D. Li, and E.G. Manning

Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C.
Canada V8W 3P6

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 straight-forward. 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. Knowledge representation using frames is presented, together with some typical rules used for the design process. Actual examples of frame representation, rules to design interfaces between microprocessor and memory components are illustrated.

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 [6,11], computer system configuration [7], geological exploration [4], VLSI design [9,10], computer system design [13], etc.

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) [1,2,3,5] 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.

The envisioned System comprises a library of available components (the knowledge base), the rule base, and the user interface. The rule base uses information from the library in order to choose the appropriate components and to eventually produce a valid design. The library of components incorporates such diverse information as names, signal protocols and timing, packaging and availability, as well as design procedures which may be applicable to the specific component. We have chosen the frame paradigm [12] to organize this diverse information. This was necessitated because of the diversity and repeatability of the information[1]. In addition, several expert system tools provide a well behaved frame development environment[2].

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.

Our basic tenet has been that the interface signals found in various microprocessor families, follow a limited number of well defined protocols for information exchange. This information is given both descriptively and quantitatively as timing diagrams by the manufacturers of the component. We have devised a three-tier representation: events and transitions, control and data protocols, and standard behaviors. Section 2 of this work presents a description language for this representation, together with a parser that can extract timing relationships from the language. Data transfer portion of the interface in the Functional Block Design Level is described in Section 3. Examples of knowledge and rule representations are given in Section 4.

2. Library of Components

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.

2.1 Event Description Language

This section provides an overview of the language used to model the behavior of the various signals associated with the components used in the design of microprocessor-based systems [5].

At the lowest level of the representation, signal transitions are related to events which must precede them and are considered for our purposes as their cause. An event description language has been developed through which, arbitrarily complex events[3] can be described. In addition, causal relations, relating events to transitions, can also be expressed. We use this language in order to encode the information in the timing diagrams provided for by the manufacturers of the components.

A signal associated with a particular device or bus is assigned a *signal_name* which consists of a unique string of characters and will be used to reference the signal. Signals are found to exist at well defined states such as *ASSERTED*, and *NEGATED*. Signals can have more than one logic state and certain states might imply other states.

States can be combined into more specific states using the *or* and *and* operators. The combination of one or more states using operators constitutes a *state_expression*. If a signal changes state, a *transition* results. Combination of states are expressed through *transition_state_expressions* . These are used in order to denote stable states during setup and hold times.

Any change of the state of one or more signals constitutes an *event*. A *transition*, or a *transition_state_expression* are considered as events. The

† This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada under the strategic grant STR0040526 and by the Science Council of British Columbia under Science and Technology Development Fund through grant SCBC #88 243.

[1]There is for example a limited number of different signal protocols that are to be found across the various components with slight variations of name and signal polarity.

[2]Knowledge Craft™ uses *schemata* to structure its data. Knowledge Craft is a trade mark of Carnegie Group Inc.

[3]An event is considered as a collection of signal transitions with imposed timing constraints and precedence.

collection of one or more events, which may or may not occur in a specified order, also is an event.

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.

2.1.1 Parsing and Detector generator

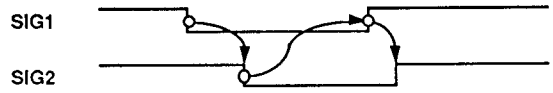
We have developed a parser (using Unix's LEX and YACC tools) which is capable of parsing transition relation expressions that use the grammar described above. Given an event expression or a causal relation, the parser is capable of extracting the timing relationships between the constituent transitions as well as the description (in CUPL) of detectors capable of detecting the described events. At this point, we are capable of generating detectors of events described through associative operators. The structure of such detectors are in the form of non-sequential logic. The parser is currently expanded so that it will be capable of generating detectors of events described by non-associative operators. Such detectors take the form of sequential machines. (Such detectors are for example needed to detect a predefined sequence of transitions obeying well specified precedence timing).

Precedence extraction allows us to graphically display the transition relation expressions in the form of a trace plot, so that relation expressions can be visually verified by the user. An example is shown in Fig. 1. Furthermore, we can extract the relative min/max timings between two transitions. These data are used in the low-level design of logic blocks. Note however, that often the timing between transitions is indeterminate or infinite, and extracting these unusable values may indicate incorrect design intentions or poorly-formed transition relation expressions.

2.2 Control and Data Protocols

At the second level of the representation, collections of limited numbers of causal relations describe basic control or data transfer protocols, using the description language. Examples of such protocols include the two signal handshaking protocol found in many information transfer subsystems.

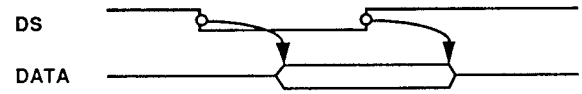
Handshake Template



handshake template

! ASSERTED SIG1 —> ! ASSERTED SIG2 @ T1
! ASSERTED SIG2 —> ! NEGATED SIG1 @ T2
! NEGATED SIG1 —> ! NEGATED SIG2 @ T3

Another typical example is the gating protocol used for the actual transfer of data. The following is a template showing 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.

2.3 Standard Behaviors

At the highest level of the representation, collections of basic protocols as they exist at the second level, constitute Standard Behaviors. In general, a Standard Behavior comprises two parts: the control part which describes the behavior of the control signals participating in this behavior and the information transfer part which describes the actual information transfer.

Examples of Standard Behaviors are data transfer behaviors such as the ones found in processors and which incorporate both a control protocol and the actual transfer protocols necessary for the delivery of addresses and data on a bus. In contrast, the data transfer behavior of static memory components are devoid of the control protocol. This absence of a control protocol illustrates the inability of a memory component to initiate activity.

Protocols as well as Standard Behaviors are depicted as semantic networks (networks of schemata in Knowledge Craft). Template networks of schemata depicting generic protocols and behaviors have been constructed. These templates are customized when they are used in the description of the various components within the component data base to reflect the particular signals and timing constraints involved.

2.4 Modelling Components in DAME

Components incorporate both structure and function specifications. Typical structure specifications include the name of the component, its type, the number and name of its interface signals etc. Function specifications on the other hand, relate to the behavior of its interface signals and they are crucial in the refinement of the design. We have postulated [5] that these signals follow some very typical behavior patterns[4] (Standard Behaviors). The existence of a particular Behavior forces design decisions to be made[5]. These behavior patterns have been identified, and they are described through templates. Some typical templates were introduced in Section 2.2. The existence of particular behavior patterns is denoted through the inclusion of instantiations of the appropriate templates in the function specifications of a component. We have organized the structure and function specifications of a component as a semantic network (network of schemata in Knowledge Craft).

Components are described as semantic networks that include both the structure and function specifications. An example of a partial network of schemata depicting part of the MC68000 is shown in Fig. 2. The hierarchy of the component specification is organized as follows. A component has a name such as MC68000. It is classified as to its type (e.g., Microprocessor, RAM memory, ROM memory, IO peripheral device etc.) Instances of components have their unique names.

Each component has signals associated with it, which will be referenced to the component through the *has-signals* relation. Each signal has a pin number, polarity etc. Each component has behaviors involving classes of signals. Typical behaviors are *data-transfer bus-arbitration* etc. Each of the behaviors relates to the semantic network through a *has-behavior* relation.

Based on the existence of particular Standard Behaviors, modelling aspects of the operation of the interface, one can write few general yet powerful rules that are capable of connecting individual components together.

As an example, we elaborate on the data transferring behavior and its use in designing a simple system comprising of a processor and a memory module.

Data transferring behaviors are distinguished as master or slave Behaviors. A component exhibiting a master behavior can initiate a data transfer (e.g. processor, DMA) while a module exhibiting a slave behavior can only respond to data transfer requests (e.g. memory).

A *data-transfer Behavior* involves three types of signals grouped into buses: the *control-bus*, *address-bus*, and *data-bus*. These groups are identified through the *uses-control-bus*, *uses-data-bus*, and *uses-address-bus* relations. The control and transfer protocols involved are identified through the *has-timing* and *uses-template* relations.

The example depicted in Fig. 2 represents the partial specification of an MC68000 component. The data transferring behavior READ-PROTOCOL-1 of this component includes the two- signal handshaking protocol as well as separate gating protocols for the transfer of the addresses and data. These protocols correspond to the fact that the MC68000 uses an asynchronous non-multiplexed bus.

3. Functional Block Design Level

The Functional Block design level considers pairs of components and identifies the various groups of signals that need to be interfaced. It generates "functional blocks" and identifies the correct signals which are used by these blocks. Each block represents a specific function. The function definition will be used at a later stage to refine design of the block.

Aspects of the Functional Block design level reported in this work, incorporate knowledge necessary for the design of the data transfer portions of the interface. It is capable of identifying the protocol used for the data transfer, and accordingly specifies the correct "functional blocks" for the control, address, and data groups of signals.

As an example, we have used two specific components, namely an MC68000 processor and a MK6116 2K by 8 static RAM and had the corresponding blocks and participating signals identified in two experiments, the first involving a single processor and a single memory component, while the second involved two memory components.

From their *data-transfer* Behaviors, their *address*, *data* and *control buses* are identified, and "buffer blocks" interface the respective address (unidirectional buffers) and data (bidirectional buffers) buses. Because the MC68000 has an asynchronous bus, which is manifested through the handshaking behavior as an

[4]A particular example of such a behavior pattern can be considered is the two signal handshaking protocol and its variants which is widely used for data transfer operations.

[5]For example, the existence of a two signal handshaking behavior in a bus (e.g. the MC68000 asynchronous bus), necessitates the generation of an acknowledge signal within a certain delay. Once such a behavior has been detected, it is easy to synthesize the appropriate block that will generate the required acknowledge.

instance of a *handshake* template, a "delay selection" block is created which will generate the appropriate data transfer acknowledge (DTACK), and chip selection (CE). The following shows the resulting delay block for DTACK :

```
{{
  BINARY-BLOCK-3
    INSTANCE: BINARY-BLOCK
    PURPOSE: DATA-TRANSFER CONTROL GENERATE
    TYPE: DELAY
    DETECT: (MC68000 TIMING-1 EVENT1)
    GENERATE: (MC68000 TIMING-1 TRANSITION2)
    DELAY: (MAX ((DELTA (MIN MK6116 TIMING-6 EVENT1 TIMING-
          6 TRANSITION2 TIMING-6 EVENT1)) MINUS .....)
    DEVICE1: U1
    DEVICE2: U2
    EXTRA-TIMING1: TIMING-7
    EXTRA-TIMING2: NIL
}}
```

The necessary "significant" events are extracted from the description of the control and transfer protocols and the corresponding event detectors are generated automatically. Fig. 1 depicts the description of such an event (this event identifies the onset of a memory access cycle in the MC68000, and it is used in the generation of the appropriate data transfer acknowledge DTACK and the corresponding detector.

4. Knowledge and Rule Representation

We have used Knowledge Craft and OPS-CRL to represent the knowledge necessary for the functional block design level. The cluster of rules which is capable of performing the data transfer design comprises rules which identify the *address*, *data* and *control* buses, the signals participating in them and their behaviors, and rules to structure the appropriate "functional blocks".

A typical rule for connecting the control bus of two components together is shown below:

```
(p create-control-block
 (timing ^schema-name <tim1> ^uses-template handshake-template)
 (data-transfer-control-bus ^schema-name <contr1>
   ^has-timing (member <tim1> <>) ^uses-signal <sigs1>)
 (data-transfer ^has-capability+inv <name1>
   ^uses-control-bus <contr1>)
   (microprocessor ^schema-name <comp-name1> ^instance
   <name1>)
 (timing ^schema-name <tim2> ^uses-template strobe-template)
 (data-transfer-control-bus ^schema-name <contr2>
   ^has-timing (member <tim2> <>) ^uses-signal <sigs2>)
  (data-transfer ^has-capability+inv <name2> ^uses-control-bus
  <contr2>)
 (memory ^schema-name <comp-name2> ^instance <name2>)
   -(block ^type control ^device1 <comp-name1> ^device2 <comp-
   name2>)
 -->
    (cschema (symgen 'block)
          ('instance 'binary-block)
          ('type 'control)
           ('function :new-values '(delay selection))
           ('device1 <comp-name1>) ('device2 <comp-name2>)
           ('signals1 :new-values <sigs1>)
           ('signals2 :new-values <sigs2>)))
```

A typical "functional block" for the control bus between the MC68000 (U1) and MK6116 (U2) is shown below. The block was created using the rule create-control-block above:

```
{{
 BLOCK2
   INSTANCE: BINARY-BLOCK
   TYPE: CONTROL
   FUNCTION: DELAY SELECTION
   DEVICE1: U1
   DEVICE2: U2
   SIGNALS1: MC68000-AS MC68000-RW MC68000-DTACK
        MC68000-LDS MC68000-UDS
   SIGNALS2: MK6116-WR MK6116-OE MK6116-CE
   OUTPUT1: MC68000-DTACK
   OUTPUT2: MK6116-CE MK6116-OE MK6116-WR
   INPUT1: MC68000-UDS MC68000-LDS MC68000-AS MC68000-
         RW
}}
```

The above connection block shows that it is used to connect control signals from devices U1 and U2, where U1 is a MC68000 microprocessor and U2 is a MK6116 static memory.

From U1 the inputs into the connection block are given in INPUT1, while the outputs out of the block are given in OUTPUT1. The connections from the connection block to U2 only consist of outputs given in OUTPUT2.

5. 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 the 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. We have also developed a parser that is capable of interpreting signal timing specifications and used it to create timing diagrams corresponding to the specifications. Actual examples of knowledge representation, and design rules used for interfacing the MC68000 microprocessor and its memory components are illustrated.

We have also implemented a user-friendly interface which is currently used in capturing the relevant information and creating a network of schemata that describes 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 driven, and is implemented as a "work center" in Knowledge Craft. We have started formulating the design rules necessary for the Functional Block Design level.

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 11086*, pp. 1858-1868, 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] Huber, B., K.F. Li, N.J. Dimopoulos, D. Li, R. Burnett, E.Manning, "Modelling Signal Behavior in DAME," *Proceedings of the 1990 International Symposium on Circuits and Systems*, New Orleans, La., Vol. 2, pp. 1497-1500, Apr. 29 - May 3,1990.

[6] 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.

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

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

[9] 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.

[10] 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. 102-118 Mar. 1988.

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

[12] Tanimoto, S., *The Elements of Artificial Intelligence An Introduction Using LISP*, Computer Science Press (1987)

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

```
Partno:          XXXX;
Date:            Wed Jun 13 17:30:30 PDT 1990;
Revision:        01;
Designer:        eros/1/dame/parser/rev2/dame;
Company:         University of Victoria, DAME group;

/**********************************************************************/
/* &(%[--,+~](|asserted RD,+(|asserted UDS,|asserted LDS!)!)          */
/*      ->|asserted DTACK @[--,+~]                                     */
/**********************************************************************/
/*                 ___________                                         */
/*     RD: ____|           |______________________                     */
/*                    ______________________________                   */
/*    UDS: _________|                                                  */
/*                          ________________________                   */
/*    LDS: _______________|                                            */
/*                          ________________________                   */
/*  DTACK: _______________|                                            */
/*                                                                     */
/**********************************************************************/

/* Pins (input and output) */
PIN XX          = !RESET;       /* input */
PIN XX          = event_01;     /* output */
PIN XX          = event_01a;    /* output */
PIN XX          = event_02;     /* output */
PIN XX          = event_02a;    /* output */
PIN XX          = RD;           /* input */
PIN XX          = event_03;     /* output */
PIN XX          = event_04;     /* output */
PIN XX          = UDS;          /* input */
PIN XX          = event_05;     /* output */
PIN XX          = LDS;          /* input */
PIN XX          = event_06;     /* output */
PIN XX          = event_07;     /* output */
PIN XX          = DTACK;        /* output */

/* Declarations and Intermediate Variable Definitions */
clear           = RESET
                # DTACK;

/* Logic Equations */
event_03.D      = !clear & !event_03  & RD
                # !clear &  event_03;
event_05.D      = !clear & !event_05  & UDS
                # !clear &  event_05;
event_06.D      = !clear & !event_06  & LDS
                # !clear &  event_06;
event_04.D      = !clear & !event_04  & ( event_05 + event_06 )
                # !clear &  event_04;
event_02a.D     = !clear & !event_02a & event_03 & !event_04
                # !clear &  event_02a;
event_02.D      = !clear & !event_02  & event_02a & event_04
                # !clear &  event_02;
event_07.D      = !clear & !event_07  & !RD
                # !clear &  event_07;
event_01a.D     = !clear & !event_01a & event_02 & !event_07
                # !clear &  event_01a;
event_01.D      = !clear & !event_01  & event_01a & event_07
                # !clear &  event_01;
DTACK.D         = !RESET & !DTACK     & event_01
                # !RESET &  DTACK;
```

Fig. 1 The rules in the data transfer module produce the specifications for the necessary glue logic needed to handle the various parts of the data transfer operation (e.g. acknowledgement, address transfer, data transfer, selection). The specifications are written in CUPL, and can be exported directly to standard PLA generators to synthesize the appropriate circuits. To this end we have used the Standard Behavior models involved in data transfers for the automatic generation of a processor memory interface in the MC68000 family.

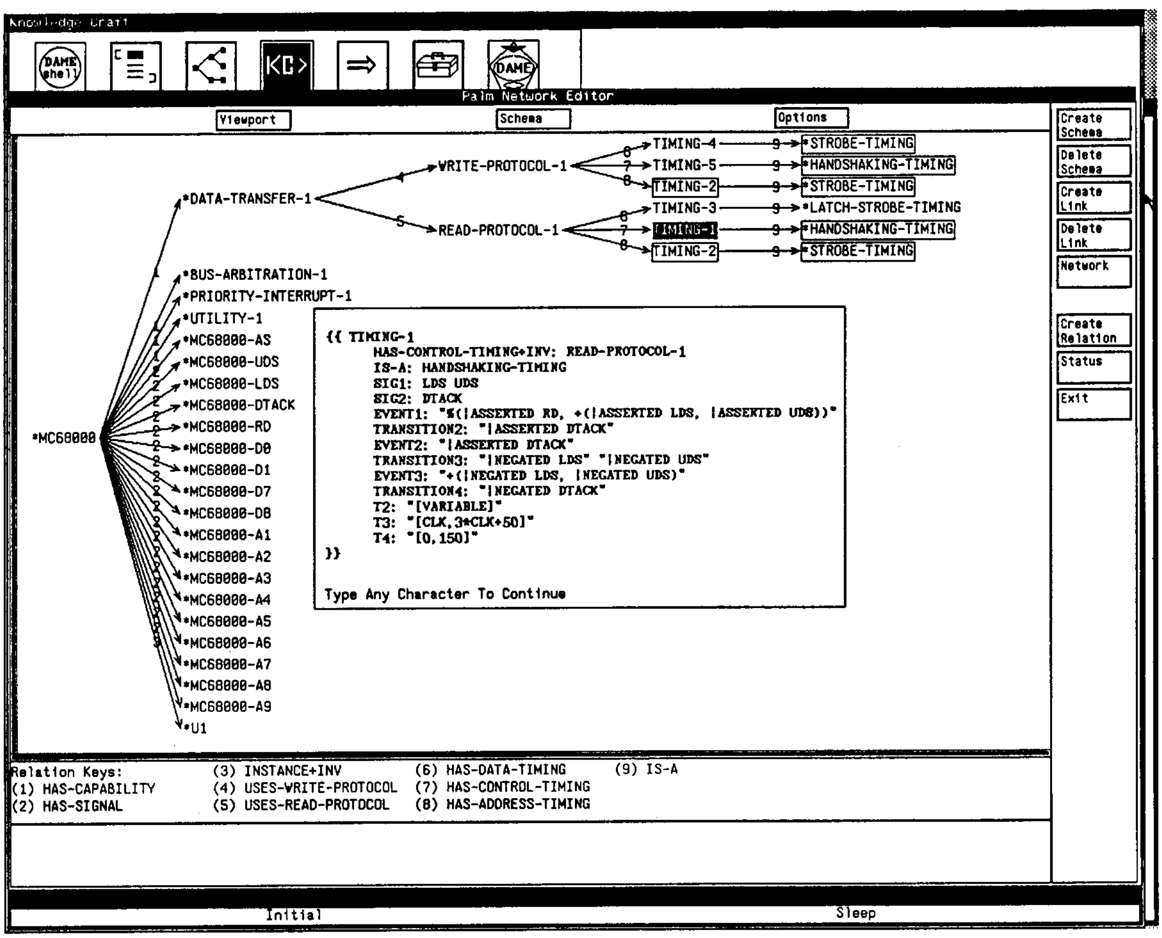


Fig. 2 A Partial Network of Schemata describing the behavior of the MC68000 processor.