BUS ARBITRATION MODELLING AND DESIGN IN DAME: AN EXPERT MICROPROCESSOR-BASED-SYSTEMS-DESIGNER[§]

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

Department of Electrical & Computer Engineering
University of Victoria
Box 3055, Victoria, B.C., Canada V8W 3P6

ABSTRACT

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) is an expert system 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. In this work, we present as a design example the bus arbitration interface to illustrate the design process in DAME. Knowledge and data representations are shown, together with a sample design for the VMEbus.

1. Introduction.

Knowledge-Based Systems (KBSs) 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 [13], geological exploration [9], etc.

Computer system design and synthesis is a very complex task that involves a large search space, requires problem-dependent decision making, and is a designer-dependent process. Since the early 1980's, several KBSs for computer systems have been developed. DAA [14] and ASP [1] are prime examples of KBS for hardware synthesis at the register transfer level. In addition, silicon compilers that use a combination of algorithmic and knowledge base approaches have become available [21].

Systems have been developed to perform microprocessor hardware design at the component level which produce a component list from input specifications, but do not provide information on their connectivity [19,22]. At the system architectural level, R1 was developed to configure computer systems but no design knowledge was involved [16]. CMU's Micon is a system synthesis tool able to assist the designers in configuring single board system from commercially available components according to customers requirements [2].

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) [5,6,8,7,11,12] is an expert system that will be capable of configuring and designing a customized microprocessor system from original specifications such as type, application and environment, communication, and computational requirements. Employing deep reasoning, DAME attempts to exploit the general design methodologies using generic communication and interfacing protocols, and to adjust and fine-tune the details according to the specific components' timing requirements and their interfacing properties. The objective is to configure the interfacing of different components intelligently once they are selected; in order to accomplish this, it employs rules which operate on the abstract properties of a component rather than their instantiations.

DAME organizes the design process into a hierarchy [5,6,7] consisting of the following phases: (1) Design Specification; (2) Configuration; (3) Behavior Description; (4) Functional Block Design; (5) Implementation and Integration.

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. Objects found at each level of the hierarchy represent the system's concept of the design requirement at the corresponding abstraction level.

Our basic tenet for interconnection of components 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. The components' descriptions include references to these basic protocols for information exchange, which are instantiated for each component. The inclusion of the references to these basic protocols in the description of our components provides a powerful model in that we are able to carry out the design process by employing a limited number of general rules which are specific to the protocols used.

§ This research is supported by the Natural Sciences and Engineering Research Council of Canada through a strategic grant.

Section 2 describes the general interface problem, and the bus arbitration protocols in particular. Action graphs representing protocols are discussed in section 3. Representations of the data and knowledge in DAME are presented in section 4, while a VMEbus design example is given in section 5. DAME is currently being implemented in Knowledge Craft™ on a Sun™ workstation.

2. The Interface Design Problem.

A protocol specifies the sequence of actions that assures the correct intercommunication between components. One example is the bus arbitration operation in a multi-master system. The actions or elementary operations of the protocol are associated with a state or a change of state in a boolean variable or signal [4].

Although there are relatively few protocols [23], the different possible mappings into signals are countless. When two components using different protocols are to be connected, it is necessary to design an interface that converts and maps the actions in these protocols to one another. We base the design of the interface upon the identification of the protocols that govern the information transfer in the component, and the protocols' instantiations for the participating components.

A multiple-master bus is a communication structure through which several units perform independent information transfers. The units can be classified according to their roles as masters and slaves. A commander is the module that initiates operation, while the other units[1] that participate in it are called responders. A master module can act as a commander, while a slave module can only be a responder. Since the bus is a single resource, only one master, called the current master (CM), is allowed to take over the bus at a given time. Arbitration is the process that grants the bus to a unique CM.

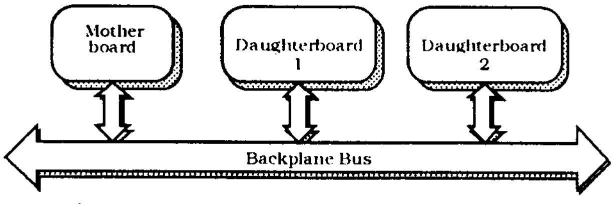

Figure 1. Multi-board multi-master system.

2.1 Bus arbitration protocols.

In the subsequent we shall address the problem of designing a bus-based system comprising a number of masters sharing a common bus as shown in figure 1. A bus provides a protocol through which a single master can be determined at any time. Modules capable of requesting the bus adhere to the requester part of the protocol, while the arbiter adheres to the responder part of the protocol. There are but a few choices of arbitration protocols. We consider the two-signal and three-signal bus arbitration protocols.

[1] In the broadcast mode, several responders can accept a piece of information from the commander.

The two-signal protocol, as shown in figure 2, is a fully-responsive asynchronous handshake protocol between REQ* and ACK*. A requester asserts this signal to request the bus, and negates it at the end of the use of the bus. ACK* informs, when asserted, that the bus is granted to the requester, and when negated, acknowledges the end of the operation.

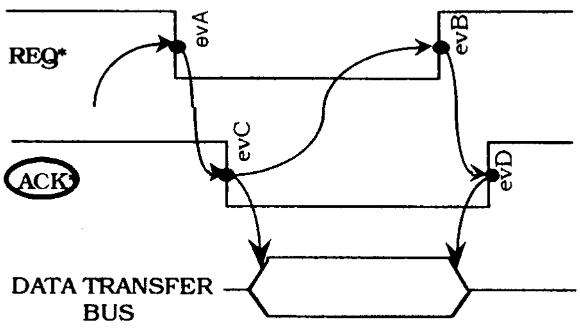

Figure 2. Two-signal bus arbitration protocol.

The three-signal protocol is more involved (see figure 3). A device requests the bus by asserting BR*. The arbiter responds by asserting BG* if BGACK* is not active. There is a fully-responsive asynchronous handshake between BG* and BGACK*. When the device receives the grant, it must release BR* and has to wait until the bus is available before starting using the bus, allowing the previous master to end its last operation. The device informs that it will become the CM by asserting BGACK*. Finally the CM signals the end of its transaction by negating BGACK*.

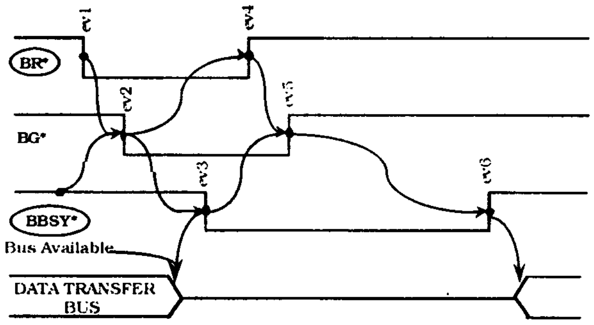

Figure 3. Three-signal bus arbitration protocol.

The power of the three-signal protocol lies to the fact that the determination of the new master can proceed while the previous master is performing its last transaction, speeding thus the transactions on the bus.

3. Protocol Graph.

An action is an event monitored by the protocol, or a change of state effected by the protocol. The sequence of actions that define a protocol has been represented using a state transition graph[24], or a marked Petri net[18]. In DAME, we use an action graph to represent the protocol as discussed in the following.

Let A be the set of actions in the protocol. We define the relation P (for precedes) for any two actions a, b ∈ A, (a, b) ∈ P *iff* action b occurs after action a. We use a weighted directed graph (A, P, t) [15] to represent the protocol, where A is the set of vertices of the graph, P is the set of edges, and t is a function on P that associates a weight (t_{min}, t_{max}), with the minimum and maximum timing between actions, to the edges. For self-timed circuits, the weight is (0, ∞).

Actions are mapped into boolean signals that are either asserted or negated (true or false). In some cases, signals are allowed to be disabled so that a group of signals can share a single wire. The state of a signal can be defined using BNF notation as follows:

state::= enabled | disabled.
enabled::= ASSERTED | NEGATED.
disabled::= TRI-STATED | OPEN-COLLECTOR.

A boolean signal can have only enabled states. A tri-stated signal can be enabled or disabled, while an "open-collector" signal (i.e., open-collector in bipolar technology and open drain in MOS technology) has only the states ASSERTED and OPEN-COLLECTOR.

A signal can be represented as follows, where the name of the signal is a string of characters:

signal::= state signal-name.
signal-name::= $\{CHAR\}^+$.

When a signal changes state, a transition occurs. We denote the transition of a signal from state 1 to state 2 as:

transition-exp::= [state-1] **!** state-2 signal-name.

When state 1 is the opposite of state 2, state 1 can be omitted (i.e., ! ASSERTED READ is equivalent to NEGATED ! ASSERTED READ). In some cases an action depends on transitions in several signals. An event expression describes transitions in various signals.

event-exp::= transition-exp | and-event-exp |
    or-event-exp | state-trans-exp.
and-event-exp::= (**^** event-exp $\{$, event-exp$\}^+$).
or-event-exp::= (**+** event-exp $\{$, event-exp$\}^+$).
state-trans-exp::= (**#** transition-exp state).

A more thorough overview of the description of signal behavior can be found in [11,12]. This notation allows us to relate actions to signals.

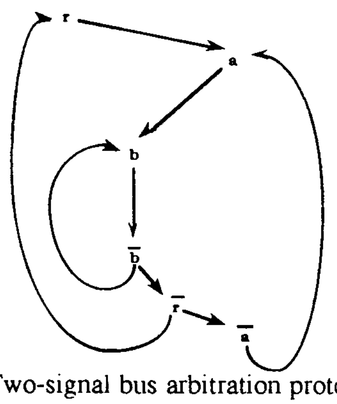

Figure 4. Two-signal bus arbitration protocol graph.

3.1 Bus arbitration protocol graphs.

In the context of the definition of an action as an event, the negation of an action is understood to correspond to the negation of the associated event. Thus, if an action **a** corresponds to a certain transition of a given signal, the negation of **a**, denoted by $\overline{\mathbf{a}}$, is defined as the opposite transition of the same signal.

The two-signal bus arbitration timing diagram of figure 2 can be described by the graph in figure 4. The REQ* signal is associated to the action pair (**r**, $\overline{\mathbf{r}}$). Action **r** corresponds to ! ASSERTED REQ*, and action $\overline{\mathbf{r}}$ corresponds to ! NEGATED REQ*. Similarly, actions **a** and $\overline{\mathbf{a}}$ are associated to signal ACK*. Actions **b** and $\overline{\mathbf{b}}$ indicate the actual use of the bus by the requester.

The three-signal protocol is modelled by the graph shown in figure 5. Signals BR*, BG* and BBSY* encode in their transitions the action pairs (**R**, $\overline{\mathbf{R}}$), (**G**, $\overline{\mathbf{G}}$), and (**GA**, $\overline{\mathbf{GA}}$) respectively.

In the above action graphs, there must exist at least one simple circuit that contains an action and its complement. That guarantees that the signal associated with a particular action will eventually return to its initial state (return-to-zero condition).

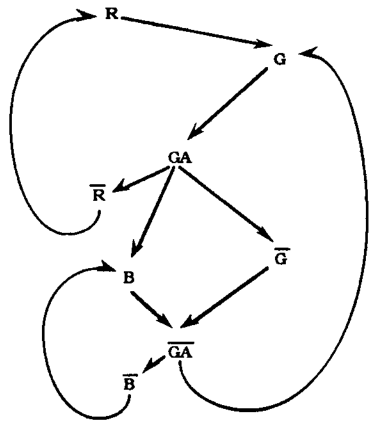

Figure 5. Three-signal bus arbitration protocol graph.

The action graphs can also be viewed as data flow graphs. Thus, when an action takes place, a token is placed on all its outgoing edges. An action cannot happen unless all its incident edges have tokens. The initial state is represented by placing tokens on certain edges. The initial marking from figure 3 calls for tokens on the edges from $\overline{\mathbf{R}}$ to **R**, from $\overline{\mathbf{B}}$ to **B**, and from $\overline{\mathbf{GA}}$ to **G**.

The interface design procedure involving two protocols can be stated as the problem of finding a graph that incorporates both protocols in which the precedences between all the edges are satisfied. In the general case there may be several solutions. Figure 6 shows a merged graph from the protocols depicted in figures 6 and 7. In this case, the requester follows a two-signal bus arbitration protocol, and it is connected to an arbiter that uses a three-signal bus arbitration protocol.

In figure 6, input actions to the devices (requester and arbiter) are encircled, and non-encircled actions are signaled by the devices. The square blocks mark the actions that inform about the status of the bus. $\overline{\mathbf{B}}$ represents the end of the use of the bus by the CM, while **B** is the negation of $\overline{\mathbf{B}}$, that includes the use of the bus by the

CM, or the idling of the bus. The interface must generate the encircled actions. Before presenting a basic block for the designed interface, we introduce the representation of the bus arbitration interface in the designer module of DAME.

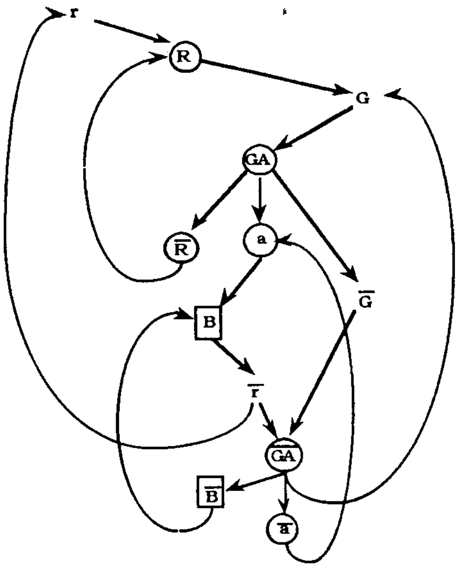

Figure 6. Interface of a two-signal protocol (requester) with a three-signal protocol (arbiter).

One configuration of a multi-master system in a daisy-chained fashion includes an arbiter that generates the unique grant, and several requesters that pass the grant through a daisy chain. The closest requester to the arbiter has the highest priority and it can take the grant before any of the other requesters. In figure 7, the arbiter follows a three-signal protocol. Requesters may use a two-signal protocol so that an interface for protocol conversion is necessitated.

Interface 1 shown in figure 7 captures the structural information of the combined protocol graph of figure 6. An action and its complement are encoded into one line. For instance, the action pairs of the two-signal protocol ($\mathbf{r}, \bar{\mathbf{r}}$) and ($\mathbf{a}, \bar{\mathbf{a}}$) are described by **req-in** and **ack** respectively. Similarly for the three-signal sub-graph the pairs ($\mathbf{R}, \bar{\mathbf{R}}$), ($\mathbf{GA}, \overline{\mathbf{GA}}$) and ($\mathbf{B}, \bar{\mathbf{B}}$) are represented by **req-out**, **grant-ack** and **bus-busy** respectively. The grant pair ($\mathbf{G}, \bar{\mathbf{G}}$) is covered in the following section. These lines are converted into single signals within the interface.

3.2 Daisy chain model.

The arbitration process belongs to the general exclusive access problem for resource allocation, the allocated resource being the bus. The selection of the CM in a daisy-chained fashion corresponds to a distributed arbiter structure. Therefore the arbitration takes place in the interfaces. Basically the grant given out by the arbiter is received by a requester that has to decide if it will take the token or pass it through the daisy chain. The internal **grant** in the interface signals the first event, while the **grant-out** is connected to the next requester in the chain.

The asynchronous nature of the arbitration makes it vulnerable to the synchronization problem [17], in which in lieu to the fact that physical systems require a finite time to respond, two tautochronous requests to an arbiter may yield a metastable state in which the output is unpredictable. Although this unstable state will decay, there is no upper bound for that moment. There are circuits that minimize the likelihood of the metastable state and avoid the hazards that arise in this kind of situation [20] (i.e., emitting two grants to two different requesters that will take over the bus, producing a collision that may even damage the hardware).

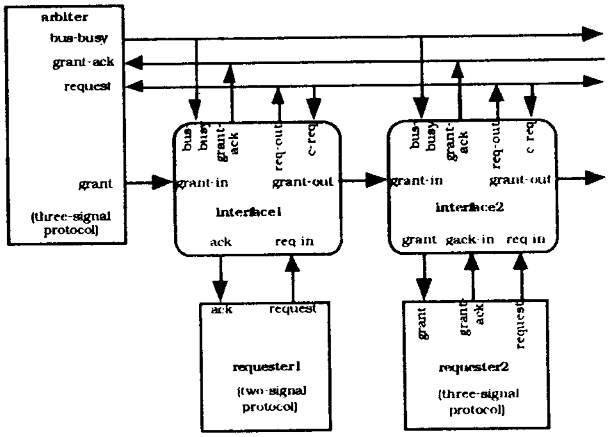

Figure 7. Bus arbitration model for a multi-master system with a daisy-chain scheme.

The basic element in those circuits is the Mutual Exclusion block [3], shown in figure 8 for the multi-master system depicted in figure 7. Even if the **req-in** and **grant-in** transitions occur simultaneously, eventually only one of the two outputs will be asserted. If **req-in** occurs earlier than **grant-in**, the requester is granted the bus via the **grant** signal, otherwise the grant is propagated through the daisy chain to the lower-priority requesters.

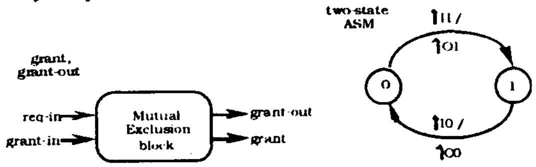

Figure 8. Basic blocks in DAME: Mutual Exclusion element and Two-state ASM.

A bare daisy-chain scheme of distributed arbitration has the problem of live-lock, in which the lower priority devices can starve because of the possibility that the higher priority devices take turn in keeping the resource busy ad infinitum. We have incorporated a fair design in the daisy-chain structure as described in [4]. In that fairness scheme, a requester that has just released the bus cannot start another request until all pending requests have been served. For this purpose the interface includes an input line **c-req** to monitor the requests from other devices.

3.3 Two-state ASM.

The two-state asynchronous state machine (ASM) is a basic building block used in our interface design. The Mealy machine representation of the two-state ASM in figure 8 indicates that a change in state from 0 to 1 occurs when the input event !I1 takes place, resulting in an output event !O1. While in state 1, a event !I0 will reset the state machine to the initial state 0, causing a event output !O0.

If !O1 and !O0 represent a pair of actions (**a**, $\bar{\mathbf{a}}$), !I1 marks the moment in which all the actions that must precede **a** have occurred and !I0 does the same for $\bar{\mathbf{a}}$. In this manner, the designer uses the two-state ASM's to generate the encircled actions in figure 6. Figure 9 shows the resulting sequential machines that constitute interface 1 in figure 7.

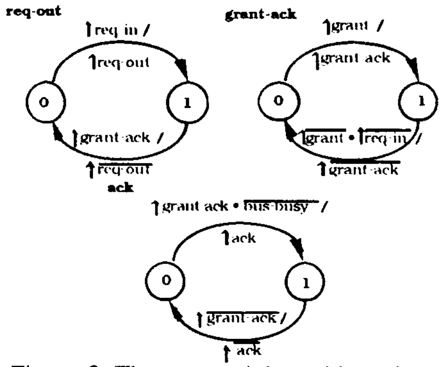

Figure 9. The sequential machines that define part of interface 1.

A description of the function for the two-state ASM using the behavioral language in SILOS II™ (SBL) has been used for the verification of our design. SILOS II permits the description of digital circuits at different levels, hence a top-down approach of the design becomes natural. We have incorporated the interface design in DAME as we shall discuss in the subsequent.

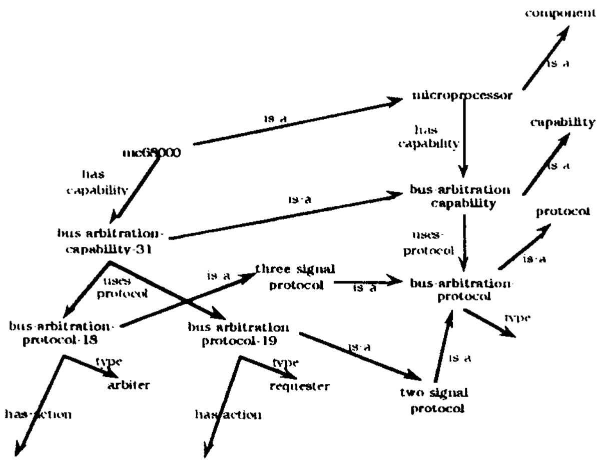

Figure 10. Partial semantic network describing the MC68000 microprocessor.

4. Representation.

Components are represented as networks of schemata. These networks are in the form of a tree that incorporates in sub-trees the description of the participating protocols.

As it has been described in earlier works [11,12,10], there exist but a limited number of protocols, and the protocol sub-trees incorporated in the semantic networks of the component are instantiations of these well defined protocols. Figure 10 presents a partial semantic network describing the MC68000 microprocessor with emphasis on the bus arbitration protocol. The bus arbitration capability comprises both a requester and a responder protocols. These protocols are related through the IS-A relations to the protocol templates that define them.

4.1 Design Knowledge Base.

The design knowledge base is structured in clusters of rules pertaining to specific aspects of the design. The design process is organized in a hierarchical manner [7] and proceeds through a continuous refinement of the structures arrived at the previous layers of the hierarchy. Clusters of rules at particular layers are activated to carry the design forward. For example, in the functional block design layer, there exist cluster of rules dealing with the design of the arbitration subsystem. Figure 11 presents such a cluster of rules.

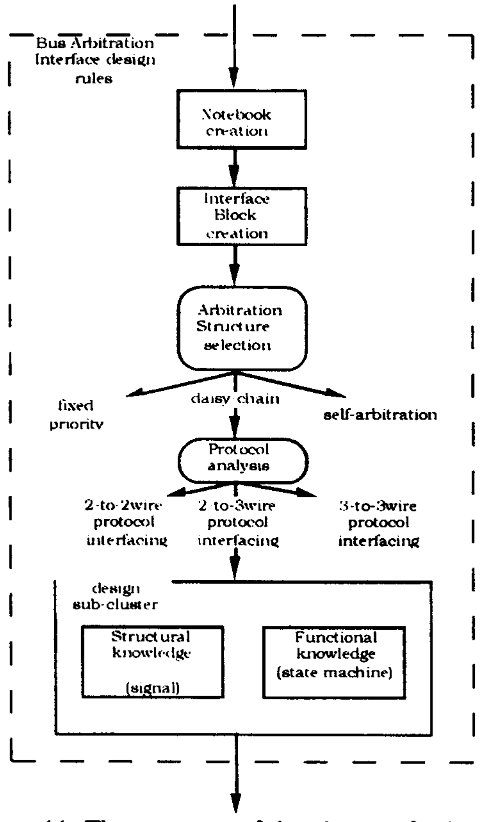
Bus Arbitration Interface design rules
Notebook creation
Interface Block creation
Arbitration Structure selection
fixed priority
daisy chain
self-arbitration
Protocol analysis
2-to-2wire protocol interfacing
2-to-3wire protocol interfacing
3-to-3wire protocol interfacing
design sub-cluster
Structural knowledge (signal)
Functional knowledge (state machine)

Figure 11. The structure of the cluster of rules that accomplishes the Bus Arbitration Subsystem Design.

Consistent with the design philosophy of DAME, these rules apply to abstract design necessitated by the presence of particular types of protocols in the participating modules rather than the specific instantiations of the protocols. The knowledge base is capable of instantiating the abstract design based on the instantiations of the protocols in the modules.

This technique of abstraction allows us to use but a limited number of powerful general rules rather than a plethora of specific rules which apply to specific instances of components. Interface blocks are produced complete with the description of their function and their interconnection. The subsequent implementation layers are responsible of implementing the functionality of these blocks in a particular technology.

5. Arbitration Subsystem Design Example.

As an example we present the design produced by the DAME designer for a DMA device in a multi-board system using the VMEbus standard in Single Level Arbitration mode. In this mode the arbiter drives BG3IN* at slot 1; BG3IN* and BG3OUT* are the bussed lines for the daisy chain. All the requesters share lines BR3* to initiate a request, and BBSY* to acknowledge the grant and to take over the bus. The status of the bus is obtained from monitoring lines BBSY* and AS* in the bus. The DMA device (Intel 8257) uses a two-signal bus arbitration protocol with HRQ as the request signal and HLDA as the acknowledge signal.

Figure 12 shows the description of one of the two-state ASM comprising interface 1.

```
{{ TWO-STATE-ASM-2
  INSTANCE: TWO-STATE-ASM
  HAS-BLOCK+INV: IB-1
  ACTION: GRANT-ACK
  INPUT0: (^ (! NEGATED IB-1-REQ-IN)
             (! NEGATED IB-1-GRANT))
  INPUT1: (! ASSERTED IB-1-GRANT)
  OUTPUT0: (! NEGATED IB-1-GRANT-ACK)
  OUTPUT1: (! ASSERTED IB-1-GRANT-ACK)}}
```

Figure 12. Frame describing the two-state ASM for **grant-ack**.

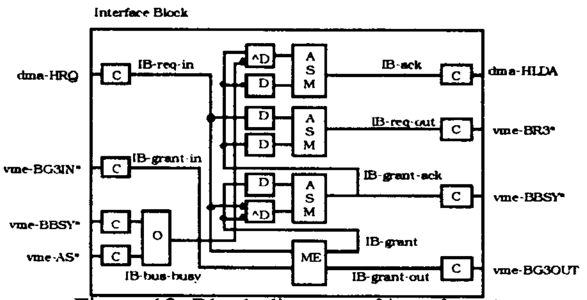

Figure 13. Block diagram of interface 1.

Beside the two-state ASM and the Mutual Exclusion (ME) block that were described earlier, other blocks that are included in interface 1 (see figure 13) are connect blocks (C), that translate the logic level and technology of a single signal, and detector blocks (D), which are digital monostable circuits that detect transitions. The and-detector (^D) is able to detect when two transitions have occurred, as defined by an and-event-expression. Finally, the observer O monitors the bussed signals to infer the availability of the data transfer bus.

The final VME bus arbitration design was implemented manually from the design blocks produced by the DAME bus arbitration design subsystem, by following conventional digital design procedures. Connection blocks were transformed into buffers. State transition diagrams were developed for the asynchronous state machines required in this design, which were in turn minimized and converted to a combinatorial logic implementation. A circuit diagram of the implemented circuit is shown in figure 14. This Figure also shows the general behavior of the two state machines required.

The final design was implemented using a XILINX 3020 programmable gate array. It was also tested by transferring the finished design to the SILOS logic simulator. The test vectors applied to the circuit consisted of all the expected input conditions. Under all conditions the circuit performed as expected. The worst case propagation delay from BG3IN* to HLDA was measured at 273 nsec.

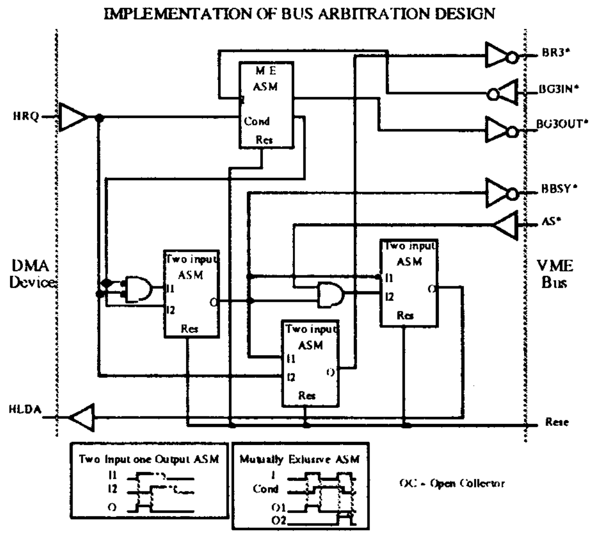

Figure 14. Circuit schematic for the implementation of the designed interface.

Conclusions.

In this work we presented the framework of the bus arbitration sub-system in DAME, an expert designer of microprocessor-based systems. In particular, we provided the action graphs used to represent the two-signal and three-signal bus arbitration protocols, and the interface procedure used for the two protocols. It was shown how the design rules are clustered pertaining to specific aspects of the design. The design example presented produced an implementation of the interface for the VME bus and a DMA device.

References.

[1] Baldwin, D., *A Model for Automatic Design of Digital Circuits*, Tech. Rep. 188, Dept. of Comp. Science, University of Rochester, July 1986.

[2] Birmingham, W.P., Gupta, A.P., and Siewiorek, D.P., "The MICON System for Computer Design," *IEEE Micro*, pp. 61-67, October 1989.

[3] Bochmann, G.V., "Hardware Specification with Temporal Logic: An Example," *IEEE Trans. on Comp.*, vol. C-31, no. 3, pp. 223-230, March 1982.

[4] Del Corso, D., Kirmann, H., and Nicoud, J.D., *Microcomputer Buses and Links*. Academic Press, London, 1986.

[5] Dimopoulos, N.J., Lee, H.C., and Galatis, N. "DAME: Automated Design of Microprocessor-based Systems, an Expert Systems Approach," In *Proc. of the Canadian Conf. on Ind. Comp. Systems*, Montreal, Canada, pp. 20-1/20-7, May 1986.

[6] Dimopoulos, N.J. and Lee, C.H. "Experiments in Designing with DAME: Design Automation of Microprocessor Based Systems using an Expert Systems Approach," In *Proc. of the Intl Computer Symposium 1986*, CCICS, Tainan, Taiwan, pp. 1858-1867, Dec. 1986.

[7] Dimopoulos, N.J., Li, K.F., and Manning, E.G. "DAME: A Rule Based Designer of Microprocessor Based Systems," In *Proc. of the 2nd Intl Conf. on Ind.& Eng. Appl. of AI & Expert Systems*, Tullahoma, Tennessee, pp. 486-492, June 6-9 1989.

[8] Dimopoulos, N.J., Huber, B., Li, K.F., Caughey, D., Escalante, M., Li, D., Burnett, R., and Manning, E. "Modelling Components in DAME," In *Proc. of the 3rd Intl Conf. on Ind. & Eng. Appl. of AI and Expert Systems*, Charleston, South Carolina, pp. 716-725, July 15-18 1990.

[9] Duda, R.O., Hart, P.E., Konolige, K., and Reboh, R., *A Computer-Based Consultant for Mineral Exploration*, Techn. Report, SRI International, September 1979.

[10] Escalante, M., Dimopoulos, N.J., Huber, B., Li, K.F., Li, D., and Manning, E.G. "Generic Design Rules for the Design of Microprocessor Based Systems in DAME: Bus Arbitration Subsystem," In *Proc. of the 1991 IEEE Intl Symp. on Circuit and Systems*, Singapore, pp. 3166-3169, June 11-14 1991.

[11] Huber, B., Li, K.F., Dimopoulos, N.J., Li, D., Burnett, R., and Manning, E.G. "Modelling Signal Behavior in DAME," In *Proceedings of the1990 Intl Symp. on Circuits and Systems*, New Orleans, Louisiana, pp. 1497-1500, April 29-May 3 1990.

[12] Huber, B., Escalante, M., Caughey, D., Dimopoulos, N.J., Li, K.F., Li, D., and Manning, E.G. "Microprocessor Components and Signal Behavior Modelling in DAME," In *Proc. of the Canadian Conf. on Electrical and Comp. Eng.*, Ottawa, pp. 19.4.1-19.4.4, Sept. 1990.

[13] Hudson, D.L. and Estrin, T., "EMERGE: A Data-driven Medical Decision Making Aid," *IEEE Trans. on Patt. Anal. and Mach. Intelligence*, vol. PAMI-6, pp. 87-91, January 1984.

[14] Kowalski, T.J., *The VLSI Design Automation Assistant: A Knowledge-Based Expert System*, Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA, April 1984.

[15] Liu, C.L., *Elements of Discrete Mathematics*, McGraw-Hill, New York, Computer Science series, 2nd. ed. Edition, 1985.

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

[17] Mead, C. and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachussetts, 1980.

[18] Ramamoorthy, C.V. and Ho, G.S., "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Trans. on Softw. Eng.*, vol. SE-6, no. 5, pp. 440-449, Sept. 1980.

[19] Ronald, C.G., *PECOS - An Expert Hardware Synthesis Systems*, Technical Report, US Army Research Office, Triangle Park, NC, 1985.

[20] Seitz, C.L., "Ideas about Arbiters," *Lambda*, pp. 10-14, First Quarter 1980.

[21] Setliff, D.E. and Rutenbar, R.A. "Knowledge-Based Synthesis of Custom VLSI Physical Design Tools: First Steps," In *Proc. of the 4th Conf. on AI Appl.*, pp. 102-118, March 1988.

[22] Smith, M.F. and Bowen, J.A., "Knowledge and Experience-based Systems for Analysis and Design of Microprocessor Applications Hardware," *Microprocessors and Microsystems*, vol. 6, no. 10, pp. 515-518, Dec. 1982.

[23] Stone, H.S., *Microcomputer Interfacing*, Addison-Wesley, Reading, Massachussetts, 1982.

[24] Vanbekbergen, P., Catthoor, F., van Meerbergen, J., and de Man, H., *Race-Free Time-Optimised Synthesis of Asynchronous Interface Circuits*, 1989, IMEC Lab., Leuven, Belgium and Philips Research Labs, Eindhoven, the Netherlands.