GENERIC DESIGN RULES FOR THE DESIGN OF MICROPROCESSOR BASED SYSTEMS IN DAME: BUS ARBITRATION SUBSYSTEM†

M. Escalante, N. J. Dimopoulos, B. Huber, K.F. Li, D. Li and E.G. Manning
Department of Electrical and Computer Engineering
University of Victoria
CANADA

ABSTRACT
In this work we present an overview of the design philosophy followed in DAME. We are postulating that systems are composed of interconnected components whose properties are represented as collections of instantiations of standard templates. This gives us the opportunity to design with the reference to these standards, and as a result we employ very few and generic rules. A specific example of the design of the bus arbitration subsystem is presented to illustrate our approach.

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. 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 computer system configuration [9, 2], geological exploration [4], VLSI design [10 , 11], etc.
DAME (**D**esign **A**utomation of **M**icroprocessor-based systems, using an **E**xpert system approach) [7 , 8] is an expert system capable of configuring and designing a customized microprocessor system from original specifications.
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 comprises a library of available components, 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.
We have chosen the frame paradigm [13] 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.

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

[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.

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.
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 capabilities [8]. 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. The abstract designs obtained thus, are then instantiated according to the individual components used.
In this work, we are presenting our method for the specific case of designing the bus arbitration subsystem. Thus, in section 2 we are presenting the basic principles of a bus operation, in section 3 we are presenting the design of the abstract interfaces while section 4 discusses the structure of the knowledge base, and present an example of the design obtained by our system. Finally we conclude in section 5.

2. The Bus concept.
Working computer systems comprise several interconnected modules that include processors, memory and I/O. These modules communicate through private or shared communication links. These links are termed "buses", and a system, depending on its structure, may include several.
Buses may be specialized. The private CPU bus is optimized to provide efficient communication with coprocessors and/or cache. The system bus provides access to peripherals, system memory and may be shared by a number of masters. Buses may be physically limited within a module or span several modules.
The introduction of numerous processor families and associated peripherals, has necessitated the standardization of bus designs so that subsystems can be easily designed and integrated [3].
Modern buses incorporate advanced protocols and technology so that efficient bus utilization and optimum transfer rates are achieved [5].

2.1 Masters and Slaves
Modules utilizing a bus are characterized as *masters* and/or *slaves* as well as *requestors* or *responders*.
Bus masters can obtain the use of the bus and initiate data transfers, while bus slaves, , participate in a transaction if requested by a master. A requestor requests services, while a responder responds to such requests.

2.2 Bus Lines

Buses are collections of pathways carrying both information and control. In addition, well defined protocols manage the orderly access and information transfer.

Bus pathways are divided into the following general categories. A bus may contain some or all of the pathway categories. These are:

Address lines. These lines allow the current master of the bus to select one of the slave modules in order to establish a communication path. Some buses support separate memory and I/O addressing.

Data lines. These are the lines that carry the actual information. The width of the data path is a fundamental parameter of a bus. Some buses include extra lines for error checking. In some buses, data and addresses are time multiplexed over the same pathways.

Control lines. These specify the direction, width and bytes that will be valid in the data transfer, the type of address and possibly the type of protocol.

Status lines. These carry error codes or status information.

Arbitration lines. The function of these lines is to prevent simultaneous use of the bus by two masters, and to schedule the requests from multiple masters.

Interrupt lines. These lines allow slaves to request service from a particular master.

Serial lines. It is becoming common practice to include one or more serial lines for local networking and diagnosis.

ID lines. Several buses now include lines that permit the assignment of unique addresses to daughterboards.

Timing lines. Implement the transfer protocol: they include strobes, syncs, and clocks.

2.3 Bus protocols.

The method used by the bus for signaling the valid use of a bus is called bus protocol. A classification scheme of the bus protocols by Stone [12] is the following:

Synchronous. The modules involved in a transaction use a common clock to time all the signals involved. Setup time, hold time, and signal skew are important in a synchronous design. This approach is simple to understand, implement, and test. However when slow devices are connected to the bus, the performance is degraded.

Asynchronous. These protocols are self-timed. As a result, fast and slow devices can share the bus without any special hardware. However the interface is more complicated.

Although the trend in recent years has favored asynchronous buses [14 ,1], some of the latest high-performance designs are synchronous (NuBus, etc.) [5].

Semisynchronous. This hybrid protocol combines the synchronous and asynchronous protocols. In this case only if the slave cannot respond fast enough, it will assert a WAIT signal; otherwise the operation is synchronous.

3.0 Bus Arbitration Interface Design

In the subsequent, we shall address the problem of designing a bus based system comprising a number of masters sharing a common bus.

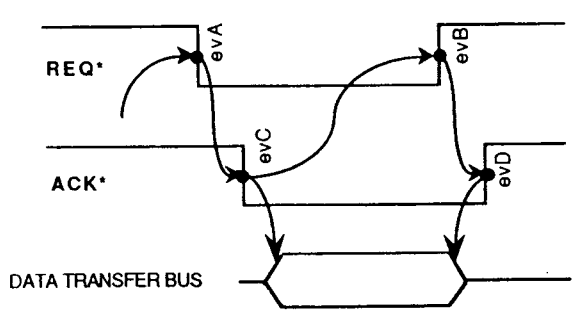

Figure 1. 2-signal Bus Arbitration Protocol (Requestor part).

Normally, a bus provides a protocol through which a single master module can be determined at any time. An arbiter adhering to the bus protocol is attached to the bus either as a dedicated module or it is distributed among the participating modules.

Modules capable of requesting the bus adhere to the requestor part of the protocol, while the arbiter adheres to the responder part of the protocol.

There are but a few choices of such protocols. They are distinguished as two or three signal arbitration protocols. Arbitration structures can be centralized or distributed.

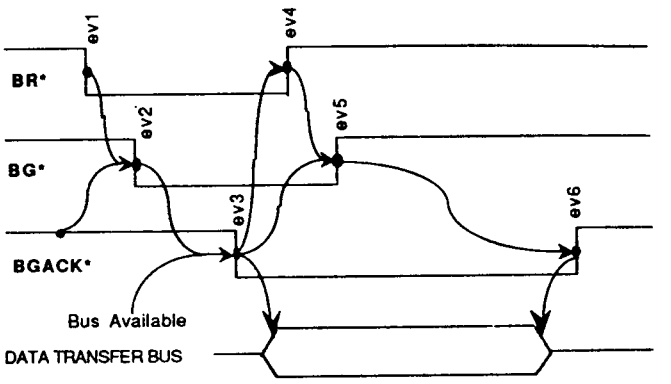

Figure 2. 3-signal Bus Arbitration Protocol (responder part e.g. MC68000).

Figures 1. and 2. illustrate the 2-signal and 3-signal arbitration protocols through their timing diagrams.

The 2-signal requestor protocol is a fully responsive asynchronous handshake protocol [6] between REQ* and ACK*. REQ* is an output signal indicating, when asserted, a request to use the bus, and when negated the end of the current bus use. ACK* is an input signal that when asserted informs the device that the bus is available, and when negated that the transaction is terminated.

In the corresponding 2-signal responder part, the same REQ* and ACK* signals are present, but in their dual rôle. REQ* is an input to the arbiter indicating, when asserted, that there is at least one requesting device, while the ACK* is an output indicating when asserted that the bus is available for the next master to use.

The 3-signal Bus Arbitration protocol [6] is more involved. The input signal BR* indicates, when asserted, that an external device is requesting the bus. The arbiter responds to this request by asserting BG* if BGACK* is not active (which indicates that the previous fully responsive asynchronous transaction has ended). The external arbitration mechanism then has to decide which device is to take the bus. Once one device is selected, it must release BR* and it has to wait until the bus is available (BAV* asserted) before starting using the bus. The device informs that it is the owner of the bus by asserting BGACK*. As it was mentioned before, BG* and BGACK* follow a fully responsive asynchronous handshake protocol. Finally the selected device signals the end of its transaction by negating BGACK*.

The power of the 3-signal arbitration protocol lies to the fact that the determination of the new master can proceed while the previous master is still using the bus, speeding thus the transactions on the bus.

Modules utilize interface blocks specifically tailored to the protocol of the bus used. As an example, we shall present the structure of a system, where several master modules share a bus employing a 3-signal arbitration protocol. We shall assume that the bus is driven by a 3-signal arbiter, and that the modules are arranged in a daisy chain reflecting their relative priorities.

This example reflects the structure of systems which incorporate a VME bus, an MC680XX processor as the lowest priority master and bus arbiter and several DMA devices.

3.1 A 3-signal multi-master system example.

2- and 3-signal modules are interfaced in a daisy chained fashion to a 3-signal arbitration bus. The general structure of the system is given in Fig. 3.

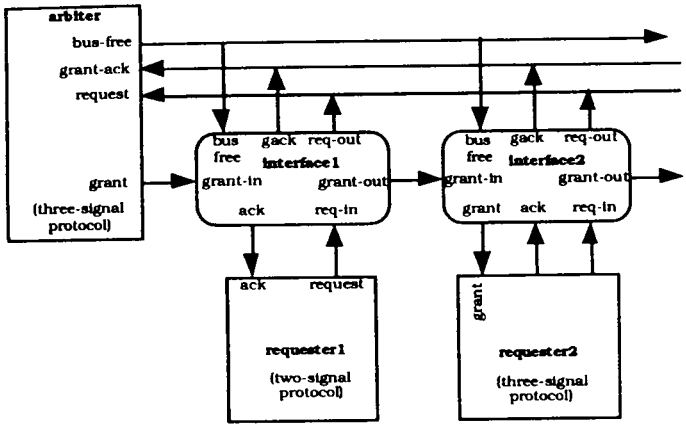

Figure 3. The structure of a multi-master system.

The primary responsibility of an interface module, is to propagate the grant in a daisy chain fashion and to acknowledge the grant when it is received and while the request from their corresponding master is asserted.

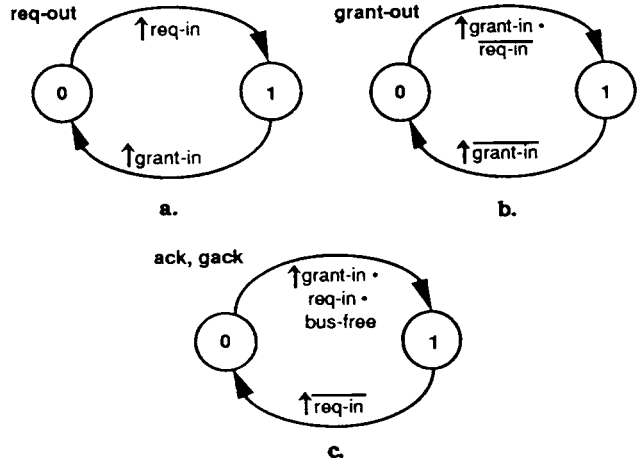

Figure 4. The sequential machines comprising Interface 1.

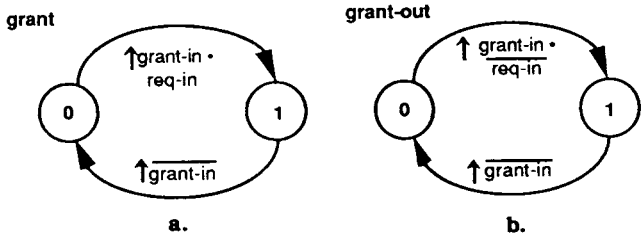

Figure 5. The sequential machines comprising Interface 2 (ACK and REQ-IN are wired directly to GACK and REQ-OUT respectively).

The sequential machines necessary for the generation of the correct signals are given in Figs. 4. and 5. These being Moore machines , have states which are associated directly with the state of the signal they control. State 0 indicates a negated signal, state 1 indicates an asserted signal.

As it can be seen (Fig. 4a), the bus request (REQ-OUT) is asserted by the interface in case that the corresponding module is requesting (REQ-IN asserted), the grant is propagated if the corresponding module is not requesting (Fig. 4b) while the grant is "arrested" by the interface block and acknowledged in case that the module is requesting.

4.0 Representation

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

As it has been described in earlier works [7, 8], there exist but a limited number of protocols, and the protocol subtrees incorporated in the semantic networks of the component are instantiations of these well defined protocols.

Figure 6. presents a partial semantic network of the description of the MC68000 component with the emphasis on the bus arbitration protocol. The bus arbitration capability comprises both a requestor and a responder protocols. These protocols are related through IS-A relations to the protocol templates that define them.

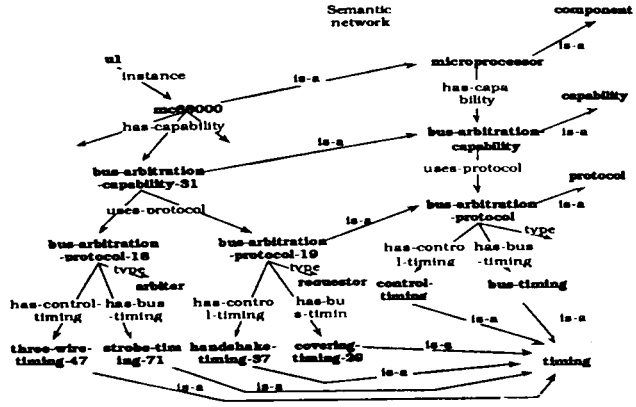

Figure 6. The partial network of schemata describing the MC68000 component.

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 [8] 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 clusters of rules dealing with the design of the arbitration subsystem. Figure 7 presents such a cluster of rules.

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 participating components.

Interface blocks are produced complete with the description of their function and their interconnection.

Subsequent layers (implementation layer) are responsible of implementing the functionality of these blocks in a particular technology.

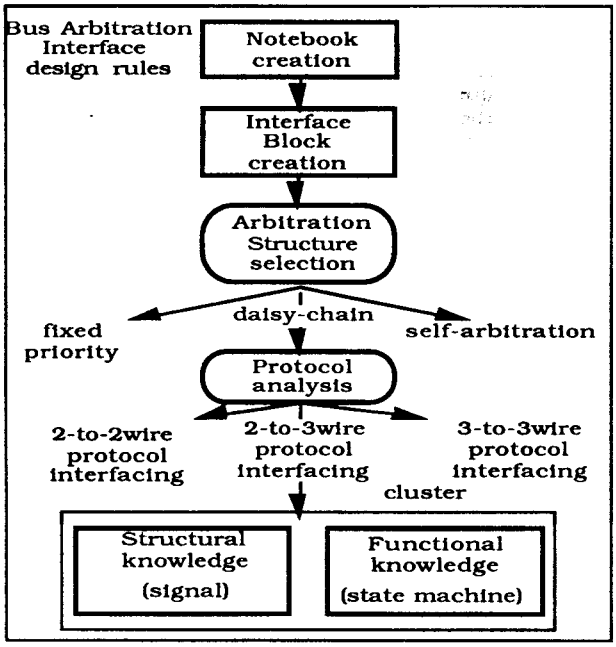

Figure 7. The structure of the cluster of rules that accomplish the Bus Arbitration Subsystem Design.

4.2 Notebooks

Since clusters of rules focus into specific aspects of the design process, they access only pertinent information from the network of schemata. Each cluster of rules employs a private notebook in which it copies the pertinent parts of the networks of schemata of the participating components ignoring the remaining schemata. The power of inheritance is employed here in order to efficiently access the necessary information from the library.

Knowledge Craft™ provides inheritance implicitly in the schemata and through the CRL™ language.

4.3 Arbitration Subsystem Design Example

As an example, we present the design produced by DAME for a system incorporating the VME bus, an MC68000 processor acting as the lowest priority master as well as driving the arbitration of the bus, and three DMA devices arranged as a daisy chain.

The overall structure of the system is identical to the one shown in Fig. 3 while Fig. 8 shows some of the resulting schemata describing the produced interface blocks, their functions and interconnections.

Needless to say that this example exercised one of the clusters namely the one dealing with the daisy chain design. Other clusters deal with priority allocation (both static and rotating) for both 2 and 3-signal protocols.

5.0 Conclusions

In this work, we presented our design approach in DAME. In particular, we concentrated in discussing the structure of a bus arbitration subsystem and presented the rule base that accomplishes the same design.

REFERENCES

1. *IEEE Standard for a Versatile Backplane Bus: VMEbus.* Wiley-Interscience, USA, 1988..
2. Birmingham, W.P., Gupta, A.P., and Siewiorek, D.P. The MICON System for Computer Design. *IEEE Micro 9*, 5 (October 1989), 61-67.
3. Conte, G. and del Corso, D. *Multi-Microprocessor Systems for Real-Time Applications*, D. Reidel, Dodrecht (1985).
4. Duda, R.O., Hart, P.E., Konolige, K., and Reboh, R. A computer-Based Consultant for Mineral Exploration. Tech. Rept. SRI International, September, 1979.
5. Gustavson, D.B. Computer Buses - A Tutorial. *IEEE Micro 4*, 4 (August 1984), 7-22.
6. Hill, F.J. and Peterson, G.R. *Digital Systems: Hardware Organization and Design*, John Wiley and Sons, New York (1987).
7. 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 International Symposium on Circuits and Systems*, IEEE, New Orleans, May 1990.
8. 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. *Proceedings of theCanadian Conference on Electrical and Computer Engineering*, Ottawa, Sep 1990, pp. 19.4.1-19.4.4.
9. McDermott, J. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence 19*(Sept 1982), 39-88.
10. Norton, S.W. and Kelly, K.M. Learning Preference Rules for a VLSI Design Problem-Solver. In *Proceedings of the fourth Conference on Artificial Intelligence Applications*, Mar 1988., pp. 152-158.
11. Setliff, D.E. and Rutenbar, R.A. Knowledge-Based Synthesis of Custom VLSI Physical Design Tools: First Steps. In *Proceedings of the fourth Conference on Artificial Intelligence Applications*, Mar. 1988, pp. 102-118.
12. Stone, H. *Microcomputer Interfacing*, Addison-Wesley, Massachussets (1982).
13. Tanimoto, S. *The Elements of Artificial Intelligence An Introduction Using LISP*, Computer Science Press (1987).
14. Taub, D.M. Arbitration and Control Acquisition in the Proposed IEEE 896 Futurebus. *IEEE Micro 4*, 4 (August 1984), 28-41.

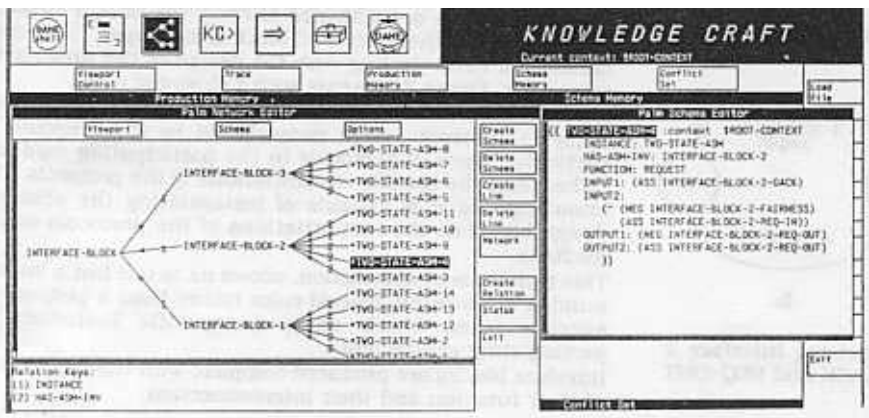

Figure 8. Schemata describing the resulting interface blocks.