The Implementor Subsystem in DAME: Using OASIS to Complete the Design Automation of Microprocessor-based Systems

Marco A. Escalante, Nikitas J. Dimopoulos, D. Michael Miller,
Kin F. Li and Eric G. Manning
Faculty of Engineering
P.O. Box 3055
University of Victoria
Victoria, B.C. V8W 3P6

Abstract

This paper discusses the implementor subsystem of DAME, a microprocessor-based-systems designer. DAME produces designs from system specifications such as type of application, performance, processing requirements, etc. The main responsibility of the implementor subsystem is to translate DAME's functional design specification of the necessary interface logic into a VLSI implementation. The basic design steps followed by DAME as well as the implementor methodology are presented, and a protocol-conversion interface example is used to illustrate the procedure. The implementor automatically converts the design specification to LogicIII and the OASIS environment then carries out the VLSI design.

1 Introduction

There is a strong motivation for the automation of the hardware design process [1, 2]. Semiconductor technology is evolving rapidly, producing high-end chips that allow the designer to build highly sophisticated systems with off-the-shelf components. Designers are confronted with a proliferation of different and highly complex components which must be integrated in the new systems. The designer must keep up-to-date with the existence of such modules and their interfacing in order to design the state-of-the-art products the market demands. DAME[1] (Design Automation of Microprocessor-based systems using an Expert system approach) attempts to minimize the impact of the large number of available components and their everchanging functionality and specifications to the design process through reasoning based on technology (and function) independent characteristics. In this sense, adaptation to the constantly forthcoming new technologies would be rather smooth. DAME is capable of configuring and designing a customized microprocessor system from system specifications.

DAME goes through a formal design process following a top-down approach which traverses the following phases [3]:

1. Design specification stage, where the system responsibilities, design constraints and system environment are established.
2. Configuration phase, where the gross architecture (processor, memory, I/O) is determined;
3. Behavioral description phase, which specifies the capabilities of the subsystem produced in the configuration phase;
4. Functional block design phase, where the capabilities of the subsystems are mapped into available components;
5. Integration phase, in which the modules are interconnected and the required interfaces designed; and
6. Implementation phase, which creates a final hardware realization of the system.

This paper presents DAME's implementor, a subsystem that transforms the behavioral description of an interface generated at the end of the integration phase into a final VLSI layout. This work is divided into five sections. Section 2 discusses some related work. Section 3 illustrates DAME's representation design methodology through an example, with emphasis on the integration stage. Section 4 describes the implementor subsystem in more detail. We have targeted the implementation towards the

[1]This research was supported by the Natural Sciences and Engineering Council of Canada through a strategic grant.

OASIS[2] (Open Architecture Silicon Implementation Software) environment. Finally in the last section we discuss some further work.

2 Related Work.

Hardware synthesis is the process of mapping an input specification of a hardware design into a hardware implementation using a set of hardware primitives [4].

Most of the effort in digital hardware design has focused on High-Level synthesis, which takes a register-transfer level specification in the behavioral domain to produce a structure [5]. However some work has been done that addresses the hardware design process from the system level as discussed in the subsequent.

MAPLE (Microprocessor ApPLications Expert) is an expert system prototype which takes the role of an expert consultant in the field of hardware design [6]. A MAPLE consultation consists of an interview in which the user is asked for the system hardware requirements and constraints, a design stage in which MAPLE uses its knowledge to build up a system fulfilling the requirements and constraints, and a report stage in which the design documentation is generated. MAPLE follows the case-based reasoning paradigm [7].

The Micon system (MIcrocomputer CONfigurer) is an integrated collection of programs which automatically synthesizes a single-board computer from system specifications using standard microprocessor technology [8]. The designer subsystem M1 accepts the system specification (in the form of processor and memory type, number and type of I/O devices, and system constraints such as cost and performance), selects the closest components in the database that match the specifications, and makes the interconnection to build up the system. Components are embodied in *templates* which may comprise several chips but possess a common interconnnection structure. The final implementation is completed by a group of CAD tools that perform placement, routing, and PCB fabrication. Micon solves the interface problem by defining templates with a common interface structure so that no extra glue logic needs to be added[3].

KDMS [9] is a hybrid system which uses a knowledge-based expert system together with algorithmic procedures to implement a microprocessor-based digital system starting from a system specification. Not only does KDMS produce the hardware design in the form of an EDIF (Electronic Design Interface Format) netlist file but also it writes the required control program using a retargetable compiler. The component database adopts a frame structure organized in a tree hierarchy in which the leaves represent primitive functional units that map directly to physical components. KDMS top-down design methodology transforms the original specification given as a collection of top-level frames in the database successively into children frames using frame construction information until a final logical design in terms of primitive frames is reached. KMDS solves the interface problem by placing the interface information into the frame construction slots. Finally the logical design is bound to physical devices with information contained in two device libraries.

DAME [3] views the final design as a collection of communicating modules which implement the desired functionality. Modules, most of which map directly to physical chips, have capabilities whose behaviors follow a limited number of standard protocols. The presence of a particular protocol dictates the design solution as produced by DAME. In contrast with the other systems, DAME's atomic elements are the protocols that rule the communication between modules and not the modules themselves. In this sense DAME reasons about the design at a deeper level. Consequently only a few powerful rules that recognize protocols, as opposed to rules addressing specific modules, are necessary to create a working design. In addition, the introduction of new components does not require the alteration of the design rules as long as their behavior can be described in terms of the stored protocols.

3 A Design Example.

In order to illustrate DAME's methodology we shall use the problem of designing a bus-based system comprising a number of devices sharing a common bus [10] as shown in Figure 1.

A protocol specifies the sequence of actions that assures the correct intercommunication between devices. Devices can be classified according to their role as masters and slaves. A master is the module that can initiate a transaction, while the slaves only participate in the transaction after being requested to do so. A bus normally includes an arbitration protocol through which a single master can be determined at any given time.

Although there are relatively few protocols [11], their instantiations differ depending on the components chosen. In this example we shall use knowl-

[2] Developed by the Microelectronics Center of North Carolina.

[3] Actually, the glue logic is integrated (hardwired) into the templates.

edge pertaining to the protocols used by the bus and individual devices to design the arbiter and the appropriate interfaces. We shall restrict our example to the arbitration part of the design. The arbiter must guarantee that at most one master will be in control of the bus at any time.

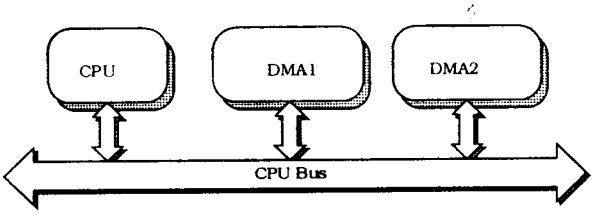

Figure 1: Multi-master system

There are several arbitration protocols. Two of the most commonly used are the two-signal and the three-signal arbitration protocols. A timing diagram of a two-signal bus arbitration protocol is shown in Figure 2. It exhibits a fully-responsive handshake between REQ* and ACK*[4]. A potential master asserts its REQ* signal whenever it needs to transfer data through the bus, and negates REQ* at the end of the transaction. On the other side, the arbiter asserts ACK* to signal the availability of the bus, and releases ACK* to acknowledge the end of the transaction.

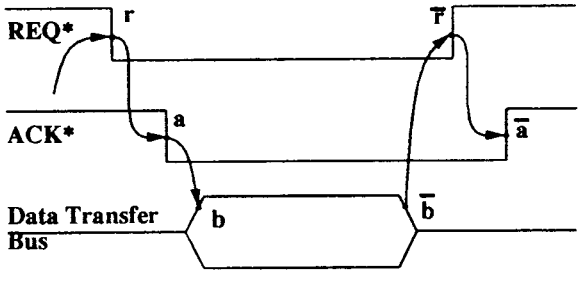

Figure 2: Timing diagram of the two-signal bus arbitration protocol

In a three-signal arbitration protocol, an additional signal is used to indicate the current status of the bus. This makes it possible to overlap the arbitration cycle with the last bus utilization cycle, with a consequent increase of efficiency [10].

In DAME, protocols are represented by action graphs. Actions are the elementary operations whose sequence defines the protocol [12]. Let A be the set of actions in the protocol. We define the relation P (*precedes*) between two actions $a, b \in A$ such that $(a, b) \in P$ iff b occurs after a . A protocol can be described by a labelled digraph (A, P, t), where A is the set of vertices, P is the set of edges, and t is a labelling function that assigns to each edge a pair (t_{min}, t_{max}) corresponding to the minimum and maximum times between the actions connected by the edge.

The action graph that prototypes the two-signal bus arbitration protocol is shown in Figure 3. There are three pairs of actions:

- $(r, \overline{r})$ corresponds to the request and release of the bus
- $(a, \overline{a})$ corresponds to the grant of the bus and acknowledge of the end of the transaction
- $(b, \overline{b})$ corresponds to the start and end of the use of the bus

Actions are represented in terms of transitions of the participating signals [13].

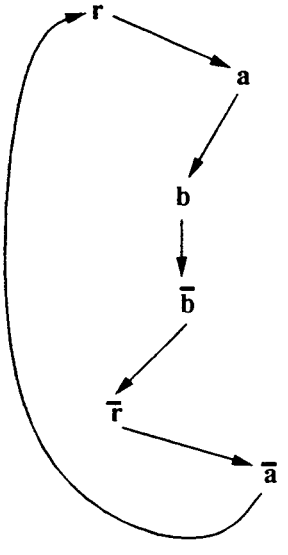

Figure 3: Action graph for the two-signal bus arbitration protocol

Components are characterized by their capabilities and are represented as networks of frames. Figure 4 shows a partial network describing the MC68000 emphasizing the bus arbitration capability. Components are described in terms of their interfacing capabilities, such as data transfer, interrupt, bus arbitration, etc. Capabilities are described by standard protocols. Action slots in the protocol subtrees contain the particular information about the signals that implement the protocol. The bus arbitration designer submodule uses this information to instantiate the design. as discussed in the following.

[4] We follow the convention that a negative-logic or asserted-low signal is denoted by suffixing "*" to its name.

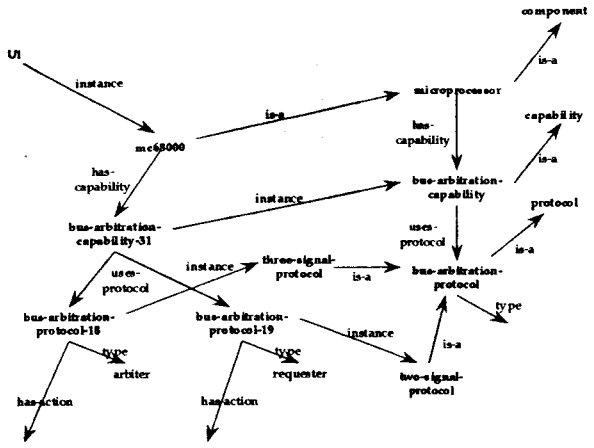

Figure 4: Partial semantic network of the MC68000

One possible configuration of the arbiter in a multi-master system is a daisy chain structure, shown in Figure 5. The arbiter generates a unique grant token that propagates through the requesters. The closest requester to the arbiter with a pending request has the highest priority and it is allowed to capture the token. In this example, the arbiter follows a three-signal bus arbitration protocol while the requesters may use a two-signal or a three-signal protocol. Therefore an interface for protocol conversion may be required.

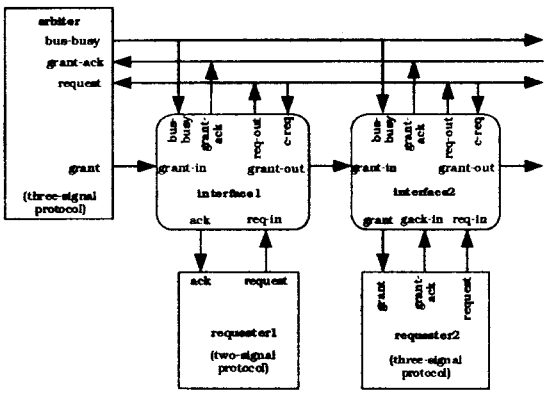

Figure 5: Daisy-chain bus arbitration structure

An initially empty interface block is created between the bus and each potential master. The interface blocks implement the token propagation and capture required by the daisy chain, as well as the effect of conversion between the two- and three-signal protocols involved.

DAME recognizes several bus arbitration structures such as daisy-chain, independent request, etc. For each structure, a group of rules is activated which configures the interface block according to the arbitration protocols of the participating components. After the interface block is configured, the implementor module implements the specified functionality using a particular technology.

One example of a designed interface between the Intel 8257 DMA chip and the VMEbus is shown in Figure 6.

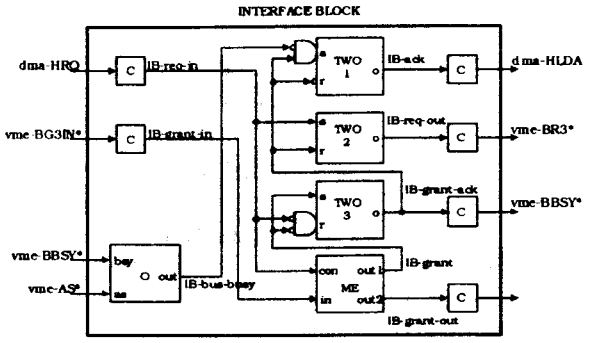

Figure 6: Block diagram of DAME's bus arbitration designed interface between the Intel 8257 DMA device and the VMEbus

The primitive blocks that comprise the interface can be classified into:

Block Type	Examples
Combinational blocks	Connect blocks (C)
Finite state machines (FSM)	Detect blocks (D), Two-state machines (TWO), Mutual exclusion blocks (ME), Special blocks (Bus observer (O))

These primitive blocks are represented as networks of frames whose slots contain the signal information particular to the specific design. Frames are structured in a hierarchy similar to the component database, in which templates at the top level encapsulate the general information of the blocks (e.g., its functionality) while instances contain the specifics of the blocks (e.g. signal names).

The template corresponding to the two-state asynchronous sequential machines (ASM) described by the TWO blocks in figure 6 is presented below. This template contains not only the empty slots corresponding to the essential information required to fully characterize an instance but also a functional description of the block. In the case of a state machine, the functional description consists of the set of inputs I, the set of outputs O, the set of states S, the output relation $OT : S \times I \rightarrow O$, the next state relation $TT : S \times I \rightarrow S$, and the initial state S_0. For these

machines the input and output sets are comprised of signal events [14]. The blank slots, such as the input and output events, must be filled out in an instance with the corresponding particular information.

```
{{ TWO-STATE-ASM
        IS-A: ASM
        INPUTS: RESET-INPUT SET-INPUT
        OUTPUTS: RESET-OUTPUT SET-OUTPUT
        STATES: STATE0 STATE1
        INITIAL-STATE: STATE0
        OUTPUT-TABLE:
           ( ((STATE0) RESET-OUTPUT)
             ((STATE1) SET-OUTPUT))
        TRANSITION-TABLE:
           ( ((STATE0 RESET-INPUT) STATE1)
             ((STATE1 SET-INPUT) STATE0))
        FUNCTION:
        RESET-INPUT:
        SET-INPUT:
        RESET-OUTPUT:
        SET-OUTPUT:
 }}
```

The frame shown below forms part of the designed interface in Figure 6, corresponding to the TWO3 asynchronous sequential machine (ASM), an instance of the generic TWO-STATE-ASM block. The input and output slots contain event expressions [13] that need to be converted into signals by the translator as discussed later.

```
{{ TWO-STATE-ASM-3
   INSTANCE: TWO-STATE-ASM
   FUNCTION: GRANT-ACK
   RESET-INPUT:
    (^ (! NEGATED INTERFACE-BLOCK-1-REQ-IN)
       (! NEGATED INTERFACE-BLOCK-1-GRANT))
   SET-INPUT:
    (! ASSERTED INTERFACE-BLOCK-1-GRANT)
   RESET-OUTPUT:
    (! NEGATED INTERFACE-BLOCK-1-GRANT-ACK)
   SET-OUTPUT:
    (! ASSERTED INTERFACE-BLOCK-1-GRANT-ACK)
}}
```

4 Implementor.

As discussed in the previous section, DAME generates a functional description of the design consisting of semantic networks specifying the required interface blocks in terms of instantiations of templates. These instantiations correspond to combinational logic and sequential machines. A further step is required such that this design can be implemented using a particular technology. The implementor subsystem accepts a functional description of the design and creates the final layout. The block diagram of the implementor is presented in Figure 7. We can see two stages. Firstly the semantic network representing the design is taken by the translator to produce a description suitable for a VLSI CAD tool. Finally the VLSI tool goes through the logic design, placement, and routing to complete the design process.

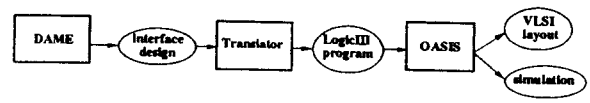

Figure 7: Block diagram of the implementor subsystem

OASIS has been chosen as the VLSI tool while OASIS's scmos2.0 is the target technology. The LogicIII language is used to describe the structure and functionality of synchronous digital designs in OASIS [15]. We have written a translator module which transforms DAME's functional specifications into LogicIII programs.

In the sequel, we shall describe the translation process using connector blocks and two-state ASM's as examples of both combinational blocks and finite-state machines to illustrate our approach.

4.1 Connector blocks.

Connector blocks are single-input single-output combinational blocks which precondition an input signal or buffer an output signal. The implementor takes into account the attributes of the input and output signals and selects the appropriate connector block in LogicIII, i.e. an inverter or a buffer.

An example of the functional description of a connector block is presented in the frame below. U7-BG3IN and INTERFACE-BLOCK-1-GRANT-IN[5] are the input and the output of CONNECTOR-BLOCK-1 respectively.

```
{{ CONNECT-BLOCK-1
    INSTANCE: CONNECT-BLOCK
    HAS-BLOCK+INV: INTERFACE-BLOCK-1
    INPUT: U7-BG3IN
    OUTPUT: INTERFACE-BLOCK-1-GRANT-IN }}
```

The implied behavior in a connector block is that the output follows the input signal (possibly with a delay). CONNECT-BLOCK-1 is recognized by

[5]INTERFACE-BLOCK-1-GRANT-IN is an signal internal to the interface.

the translator as an inverter because internal signals (e.g., INTERFACE-BLOCK-1-GRANT-IN) use positive logic while U7-BG3IN is an active-low signal as it can be seen from the frame shown below.

```
{{ U7-BG3IN
    IS-A: SIGNAL
    HAS-SIGNAL+INV: VME-BUS
    ACTIVE: LOW
    TYPE: BINARY
    NAME: BG3IN
    PIN-NUMBER: 24
    I-O:  OUTPUT }}
```

The translator takes this information and instantiates the corresponding inverter module in LogicIII with values *in* = U7-BG3IN and *out* = INTERFACE-BLOCK-1-GRANT-IN:

```
NET_MODULE inverter
  (in: INPUT; out: OUTPUT;);
BEGIN
  i1(in, out);
END.
```

LogicIII defines structural or NET modules and functional or LOGIC modules. NET modules are used to describe circuits as interconnections of simpler modules. For example the inverter NET module consists only of the OASIS inverter primitive *i1*. LOGIC modules allow us to describe a circuit not by its topology but by its behavior. It is important to note that there is no one-to-one correspondance between the complexity of DAME's functional description and the resulting OASIS representation. A Pascal-like language is used to express module behavior as shown in the next section.

4.2 Two-state ASM.

Two-state asynchronous sequential machines were used in the specification of the interface block described in section 3. Such machines have two inputs, set and reset, and one output, initially set to zero. The transition table was shown in section 3. The rationale behind the use of the two-state ASM's as design primitives lies in the fact that they are structured to formulate the onset and the negation of the various control signals that govern the protocol.

The two-state ASM primitive block is expanded in the implementation as a two-state core (TWO) and two event detectors (D) at its inputs, as depicted in Figure 8. Event detectors are the circuits which condition and synchronize the signals which represent the setting and resetting events which in turn are used

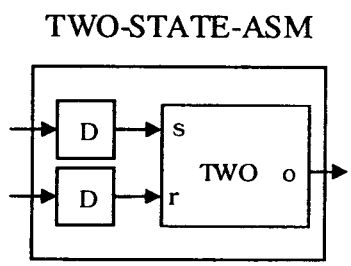

Figure 8: Implementation of the two-state ASM.

to drive the two-state ASM. The TWO block and the event detectors are implemented as finite state machines.

The LogicIII module corresponding to the TWO core is described using by the following LOGIC module.

```
LOGIC_MODULE two_state_asm
  (set_in, reset_in : INPUT; out : OUTPUT;);
VAR
  St : ARRAY [0..0] of STATE;
  state0, state1 : INTEGER;
BEGIN
  (* State Definition *)
  state0:= 0;
  state1:= 1;
  (* Transition Table *)
  CASE
    St = state0 : BEGIN
      IF set_in THEN
        BEGIN St:= state1; out:= 1; END
      ELSE
        BEGIN St:= state0; out:= 0; END
    END;
    St = state1 : BEGIN
      IF reset_in THEN
        BEGIN St:= state0; out:= 0; END
      ELSE
        BEGIN St:= state1; out:= 1; END;
    END;
  END (*CASE*);
```

State variables are used to describe the internal states of a finite state machine, and are implemented as an array of D flip-flops. In this LogicIII description of the TWO block there are two states corresponding to the two binary values of the state variable *St*. The transition table is defined in the CASE statement. Similar LOGIC modules exist for the event detectors and other sequential machines.

4.3 Translator.

Although DAME's primitives (e.g. the two-state machine described in section 3) are not bound to a particular implementation discipline, at the implementation stage it is necessary to select one discipline[6].

Because Logic III supports a synchronous methodology, we have identified the set of implementation primitives necessary in a synchronous realization (some of which were discussed in previous subsections) and have written the corresponding LogicIII modules. The translator's responsibility is to convert the interface blocks instantiated during the integration phase into the appropriate implementation primitives and their corresponding LogicIII modules (see figure 7).

One block in DAME's interface description may be expanded into several implementation primitives. As discussed earlier, the TWO-STATE-ASM's are decomposed into input event detectors and core sequential blocks. For example, the TWO-STATE-ASM-3 frame presented in section 3 is decomposed into the following implementation primitives: one two-state core block, two inverters and two event detectors.

```
{{ OASIS-TWO-STATE-ASM-3
    INSTANCE: OASIS-TWO-STATE-ASM
    SET_IN: OASIS-SINGLE-DETECTOR-3-OUTPUT
    RESET_IN: OASIS-AND-DETECTOR-1-OUTPUT
    OUT: INTERFACE-BLOCK-1-GRANT-ACK }}

{{ OASIS-INVERTER-1
    INSTANCE: OASIS-INVERTER
    IN: INTERFACE-BLOCK-1-REQ-IN
    OUT: NOT-INTERFACE-BLOCK-1-REQ-IN }}

{{ OASIS-INVERTER-2
    INSTANCE: OASIS-INVERTER
    IN: INTERFACE-BLOCK-1-GRANT
    OUT: NOT-INTERFACE-BLOCK-1-GRANT }}

{{ OASIS-SINGLE-DETECTOR-3
    INSTANCE: OASIS-SINGLE-DETECTOR
    IN: INTERFACE-BLOCK-1-GRANT
    OUT: OASIS-SINGLE-DETECTOR-3-OUTPUT }}

{{ OASIS-AND-DETECTOR-1
    INSTANCE: OASIS-AND-DETECTOR
    IN1: NOT-INTERFACE-BLOCK-1-REQ-IN
    IN2: NOT-INTERFACE-BLOCK-1-GRANT
    OUT: OASIS-AND-DETECTOR-1-OUTPUT }}
```

[6]The most common choices are: synchronous, self-timed, speed-independent, or asynchronous designs.

Finally each of these modules is substituted by a logic module in LogicIII. For example, the OASIS-TWO-STATE-ASM-3 frame is converted into the following instantiation of the LOGIC module discussed in section 4.2.

```
two_state_asm(
  DETECTOR_3_OUTPUT,
  AND_DETECTOR_1_OUTPUT,
  INTERFACE_BLOCK_1_GRANT_ACK);
```

OASIS accepts the LogicIII description and proceeds to carry out logic minimization, factorization/decomposition, cell allocation, placement and routing, to produce a VLSI layout. Figure 9 shows the layout obtained from the interface depicted in Figure 6.

5 Conclusions.

Currently, DAME is an experimental system which incorporates the models, rules and protocols pertaining to arbitration and data transfer. DAME is implemented in KNOWLEDGE CRAFT™[7]. KNOWLEDGE CRAFT was chosen because of its versatility in defining relations, its use of frames, and its user interface. We have used DAME to produce bus arbitration designs for systems incorporating a standard bus (e.g. VME) and several daisy-chained masters. We have also produced memory subsystem designs for different types of processors (including the MC680x0 and Intel 80x86 families).

DAME, as it has been developed, is a proof of concept and an experimental vehicle for testing our ideas. Interfacing DAME's output to existing VLSI CAD tools is of paramount importance to show that DAME's designs can be easily fabricated and simulated. Up to this moment the OASIS environment has provided us with a stable environment through which we have been able to implement DAME's designs. In addition, we are currently considering VHDL [16] as a means to enlarge the number of implementation environments DAME could use.

References

[1] W. M. van Cleemput and H. Ofek, "Design automation for digital systems," *Computer*, pp. 114–122, Oct. 1984.

[2] A. C. Parker, "Automated synthesis of digital systems," *IEEE Design & Test*, pp. 75–81, Nov. 1984.

[7]Knowledge Craft is a trademark of Carnegie Group Inc.

[3] N. J. Dimopoulos, K. F. Li, and E. G. Manning, "DAME: A rule based designer of microprocessor based systems," in *Proc. of the 2nd Intl. Conf. on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, (Tullahoma, Tennessee), pp. 486–492, 1989.

[4] A. C. Parker, "Automated synthesis of digital systems," *IEEE Design & Test*, pp. 75–81, Nov. 1984.

[5] R. Camposano, "From behavior to structure: High-level synthesis," *IEEE Design & Test of Computers*, pp. 8–19, Oct. 1990.

[6] M. F. Smith and J. A. Bowen, "Knowledge and experience-based systems for analysis and design of microprocessor applications," *Microprocessors and Microsystems*, vol. 6, pp. 515–518, Dec. 1982.

[7] J. L. Kolodner and C. K. Riesbeck, *Experience, Memory, and Reasoning.* Hillsdale, NJ: Lawrence Erlbaum Assoc., Inc., 1986.

[8] W. P. Birmingham, A. P. Gupta, and D. P. Siewiorek, "The Micon system for computer design," *IEEE Micro*, vol. 9, pp. 61–67, Oct. 1989.

[9] Y.-H. Kuo, L. Kung, C.-C. Tzeng, G.-H. Jeng, and W.-K. Chia, "KMDS: An expert system for integrated hardware/software design of microprocessor-based digital systems," *IEEE Micro*, vol. 11, pp. 32–92, Aug. 1991.

[10] M. Escalante, N. J. Dimopoulos, B. Huber, K. F. Li, D. Li, and E. G. Manning, "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.

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

[12] D. del Corso, H. Kirmann, and J. D. Nicoud, *Microcomputer buses and links.* London: Academic Press, 1986.

[13] B. Huber, M. Escalante, D. Caughey, N. J. Dimopoulos, K. F. Li, D. Li, and E. G. Manning, "Microprocessor components and signal behavior modelling in DAME," in *Proc. of the Canadian Conference on Electrical and Computer Engineering*, pp. 19.4.1–19.4.4, Sept. 1990.

[14] T.-A. Chu, "On the models for designing VLSI asynchronous digital systems," *INTEGRATION, the VLSI journal*, no. 4, pp. 99–113, 1986.

[15] Microelectronics Center of North Carolina, *OASIS Users' Reference Guide*, 1989. Chapter 2.

[16] J. R. Armstrong, *Chip-level modeling with VHDL.* Englewood Cliffs: Prentice Hall, 1989.

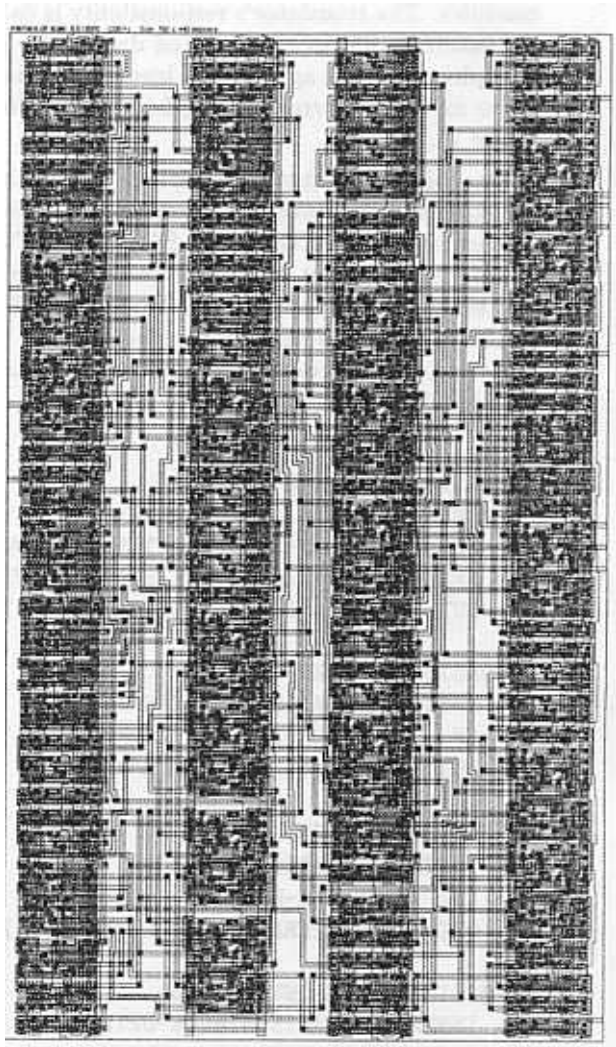

Figure 9: The VLSI layout of the designed interface