The MICON System for Single Board Computer Design

by

N. Balram. W. Birmingham, S. Brady, R. Tremain, and D. Siewiorek

EDRC-18-03-87

Published

Proceedings of Applications of AI to Engineering Problems
Reading, England . April 1986

The M1C0N System for Single Board Computer Design

Nikhil Balram
William P. Birmingham
Sean Brady
Robert Tremain
Daniel P. Siewiorek

Department of Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213
USA

Abstract

The MICON (M]cro-processor QQNftgurer) system is a knowledge-based design aid, written in OPS5. for the domain of single board computers. A prototype of the system has produced working hardware. From a set of high level requirements, MICON is able to produce a customized board. A high level specification is necessarily vague with regards to detailed implementation issues, in order to produce a detailed implementation of the design, MICON must continuously transform and refine the representation of the design objects across three levels of representation. The design is refined using domain specific knowledge. Domain knowledge is partitioned into five categories to facilitate future expansion of the system. The inference engine of MICON is based on a simple design model consisting of three steps: specification, selection and integration.

1. Introduction

The MICON[1] system is a knowledge-based design aid, written in OPS5 (3), for the domain of single board computers. A prototype of the system has produced working hardware. From a set of high level requirements, MICON is able to produce a customized board. The high level requirements capture salient features of a board needed to fit a particular application. However a high level specification is necessarily vague with regards to detailed implementation issues. In order to produce a detailed implementation of the design, MICON must continuously transform and refine the representation of the design objects at the highest abstraction level into objects which can be constructed at the lowest abstraction level.

In producing designs for the single board computer domain, MICON uses a technique of template refinement. A set of functional templates have been developed which represent subsystems of a single board computer system. Functional templates are devoid of particular component information, but contain important abstracted features of the subsystems, allowing a rough initial structure to be developed from the high level requirements. From the functional template level the design is brought to the specific logic level, where the full logic structure necessary to realize the abstracted features of the functional template level are developed* The final level is the specific hardware level, where specific chips are bound to the structure used in the levels above. By using several abstraction levels, well defined partitions in the knowledge base are established, easing the burden of adding new knowledge to allow MICON to design with new chips. Besides the structural knowledge, there are four types of knowledge in MICON's knowledge base: selection, refinement, problem-solving and component. Selection knowledge allows MICON to choose the correct components for refinement of a template. Refinement knowledge describes how to refine the templates for specific applications. Problem-solving knowledge describes when and how refinement knowledge should be applied. Component knowledge is a store of knowledge on the salient features of the design components used by Micon.

At every design decision point, there are many options available. The absence of a well formed set of transformations to guide the refinement process and incomplete domain knowledge, combine to make the design process an ill-structured task. This is typical of the tasks handled by other knowledge based tools. Three recent works of this type attacked various aspects of integrated circuit synthesis. TALIB [5] automates the mask layout phase of the IC design process. WEAVER [4] is an expert channel/switchbox router. Ulysses [2] is an environment for the design of analog integrated circuits. Other related work includes a system for the redesign of circuits *[7]* and the configuration of super-mini computer systems [6]. All of these systems used domain specific knowledge to accomplish their tasks.

Domain knowledge is exploited in two ways in MICON. First, the techniques of refinement are all based on very specific domain actions. Hence, a large number of rules are necessary m order to produce good designs. Second, the ordering of tasks is determined utilizing domain-specific knowledge. By delaying the execution of those design tasks that require information generated by other design tasks, unnecessary search is eliminated.

MICON has evolved from a prototype system. The prototype system contained many of the features m the present MICON. While the prototype produced a working Z80[1] board, it capabilities were limited. The limitation of the prototype rest partially with an inadequate representation scheme. In order to provide context for the representation scheme presently used, a brief overview of the prototype system will be presented. After the overview, a detailed discussion of MICON's architecture and representation techniques is presented.

2. Prototype MICON

The Prototype MICON (P-MICON) was completed by the end of 1983. P-MICON generated a schematic for a Z-80 based single board computer. Several man-months later, a working board was produced.

2.1. P-MICON structure

The design process used by P-MICON was based on the customization of a large template for a given set of input parameters. A template represents the general interconnection structure between the computer's subsystems. The 'subsystems are: Processor, Memory, and Input/Output (I/O) peripherals. Template refinement is based upon a simple design model:

<u>Specification:</u> A set of requirements for each subsystem is given by the user. For example, the amount of memory needed; the type of processor; and the types of I/O devices.

<u>Selection:</u> Components are selected from a parts database by matching their characteristics with the requirements of the user. In some cases, selection is accomplished by a simple name match; other situations* require weighing various features of a component against the requirements (e.g. power vs. speed).

<u>Instantiation:</u> The chosen components are inserted into the appropriate position in the template. During instantiation, support components are also added. Support components are not specified by the user, but are necessary for proper system operation. Examples of such components are: buffers, drivers and the system clock.

P-MICON strictly ordered tasks during refinement Each step above was completed for each subsystem before the next step began. For example, the specification and selection steps for memory, processor and I/O were performed before any instantiation steps. The completed design was output in the form of a net-list.

2.2. P-MICON Deficiencies

Upon completing the construction of the Z-80 board, several deficiencies of the system were discovered. The deficiencies are best explained after consideration of P-MICON's assumptions. The first assumption was that enough domain knowledge was available for P-MICON to make an assertion about the design without having to retract it later as the design progressed and more information

[1]Z8O it *r»gistered trademark of ZItoffCotp.

became available. The second assumption was that each of the design tasks could be considered nearty-independently of other design tasks based on the straightforward nature of the domain.

Experience has shown that the first assumption is generally valid, except in certain situations which exhibit bin-packing problem characteristics. The second assumption is valid only for simple designs. Many of the assertions made by the prototype system must be done in the absence of full design state information. This problem is particularly evident in resolving constraint violations. For example, the prototype system assigns a maximum amount of board area for implementing the system. If a subsystem requires more space than it was given, the prototype finds the problem unsolvabie even if another subsystem consumed less area than it was given.

Another limitation of P-MICON is its limited degree of design freedom stemming from too large a template. Since each template is oriented towards a particular processor and the components which support it, it is very difficult to design in components that do not belong to the microprocessor's component family.

The results of these shortcomings are: difficulty in adding new design knowledge to the system, inability to produce more sophisticated designs, and unnecessary search. The remaining sections will discuss MICON's solutions to these problems.

3. MICON Representation Levels

In order to remove the aforementioned representation difficulties, two new levels of hierarchy were added to MICON's structural representation. The present representation assists MICON's design activities as it progresses through a design by reflecting the natural hierarchy in the task. Details of components and their interconnections are made available only when necessary to complete some task, making attention focusing much easier. This also results in better run time performance of the system.

The present representation scheme provides clean partitions in the types of structures allowed by MICON at any level. In so doing, the types of actions the system can perform on the structure are also clearly defined. This, combined with a well-defined system architecture, results in a knowledge base that is easily expandable. In this way MICON avoids being trapped in the same limited framework as P-MICON was. Deciding where new knowledge should be properly added to MICON's knowledge-base, and deciding exactly how to add the knowledge, is an easier task than with P-MICON.

A description of the representation levels of MICON follows. Each level adds more detail to the developing design. The first two representation levels are oriented towards functional and logical design. The last level has provisions for electrical characteristics of the system.

3.1. Level 1 - Functional Logic

This is the highest and most general level of abstraction. At the functional logic level a single-board computer system is divided into the following subsystems: Processor, Memory, I/O, Bus, and Address decoder. This decomposition is finer-grained than P-MICON.

All signals included in the design at this level are generic signals, typical of any design, and independent of the actual components chosen. Thus, all signals are referred to by their functional names rather than by the specific names assigned by individual manufacturers.

3.2. Level 2 •- Specific Logic

At this level the design is still partitioned into the same subsystems as in Level 1, but is now processor specific. Since the processor has been selected, all interconnections are referred to by the name given by the processor. To illustrate the difference between Levels 1 and 2, a sample design, with a Z80 processor, a MC68000[2] family i/O chip and 64K SRAM is considered. As shown in Figure 1, the Level 1 generic template for this design is typical of all designs. At Level 2 the generic signals have been changed into Z80 nomenclature, as depicted in Figure 2. Some signals are merely renamed and others are modified (by adding gates) to have the functionality the Z80 requires. For example, the generic signal *lOreq* (see Fig 1) has now been renamed *Int* (see Fig 2).

Each of the major subsystems are now fully expanded logically at this level. Continuing with the example, the memory macro fa *macro is an abstract representation tor a set of items that exhibit certain specified properties]* now •contains the 64K SRAM arranged as required, and the I/O subsystem macro contains the desired I/O chips with its interface to the system fully described logically. The I/O subsystem is by far the most complicated subsystem to represent The problem of finding a general representation (in the form of a general macro that fits all I/O devices with the same functionality, as was done with memory) is made very difficult because of the lack of conformity, in terms of signal names and functions, between chips from different manufacturers. Even I/O devices with the same functionality are not similar. The problem is the difficulty of connecting a processor belonging to one family (Z80 in the example being considered) to I/O chip(s) of another family (MC68000 in the sample design). The solution is to create an interface for the I/O subsystem that the processor could treat as being an interface to its own family devices. This solution is based upon the observation that within a family, all the I/O devices (SIO, PIO, etc..) have an identical interface with a processor of the same family, so that a single macro could be defined to represent each family of I/O devices. The family macros defined at present are:

1.Z80

2. MC68000

3.MC6800[2]

4. NS32032[3]

The following steps are followed to incorporate an individual I/O device at Level 2:

1. The processors' family I/O macro is created (in the example a Z80 I/O

---

[2]MC68000 and MC6800 are registered trademarks of Motorola Inc.

[3]NS32032 is a reg«tered trademark of National Semiconductor Corp.

macro is created, as shown in Figure 3).

2. The family I/O macro, corresponding to the family of the I/O chip being included in the design, is created (in the example the I/O chip belongs to the MC68000 family, hence a MC68000 family I/O macro is instantiated, see Figure 4).

3. The I/O chip's family macro is connected inside the processor's family macro (in the example the MC68000 I/O macro is connected to the interior of the Z80 I/O macro, as shown in Figure 5). Essentially, this step creates an interface between two different chip families, making them compatible for the same design.

4. Finally, the actual I/O device (MC68681 in the example) at Level 2 is placed inside its own family I/O macro (MC68000 family in the example), thus creating a device with a processor-specific interface. Refer to Fig 6.

The above sequence of steps is followed as required for each I/O device being included in the design. Once a device has the outer processor specific macro, it can be connected, along with other support chips, to form an I/O subsystem.

This method of representing the I/O systems provides a great deal of flexibility and expandability. New I/O devices can be added to Micon's library by simply adding the family I/O macro and a set of rules specifying the way of constructing interfaces to macros already part of Micon's library. No changes have to be made in the existing knowledge base.

3.3. Level 3 •• Specific Hardware

After Level 2 is fully developed, a logic design exists, with all required interconnections between different subsystems. At Level 3 electrical properties and interfaces to external buses are represented. Specifically:

1. Connections to external buses are represented.

2. Discrete logic gates are represented where needed for the purpose of producing consistent logic signal assertion levels.

3. Discrete electrical components (resistors and capacitors) are represented.

4. MICON Architecture

To generate a design, MICON refines the design through the three levels of abstraction mentioned previously. MICON has an opportunistic approach to refinement. MICON will refine portions of the design whenever an opportunity (a known design situation) appears. When an opportunity is developed, it generally leads to further refinement opportunities. This process continues throughout the design process-

In determining what design action to take for a particular design situation, MICON relies on match and limited means-ends analysis. Match knowledge** used to represent specific refinement opportunities and the method in which to develop the opportunity. When match knowledge is insufficient for a situation, MICON determines which action to perform based on the current state of the design and the goal of the current task. Since only simple metrics of determining the current state of the design are incorporated into the system (e.g. current component is not known, incompatible component types) the means-ends analysis has limited application.

The methods mentioned above form the basic MICON problem-solving architecture. The architecture allows MICON to design single board computers by defining a set of tasks for design refinement and by defining the preconditions of a task's invocation. The architecture is organized around the simple design model used in the prototype: specftcation, selection, and instantiation.

The two central features of MICON's architecture are the knowledge partitions and the problem-solving techniques. These are discussed in the following sections.

4.1. Knowledge Partitions

The knowledge-base is partitioned into five areas:

- Structural relationships
- Refinement knowledge
- Selection knowledge
- Component knowledge.
- Problem-solving knowledge

Structural knowledge describes interconnections between MICON's subsystems. The interconnections are based on the abstracted functionality of signals running between the subsystems. For example, the processor-memory structure is defined by the address bus, data bus, and generalized control signals running between them. Presently, structural knowledge covers processor, memory, I/O subsystems, an address decoder, reliability related devices such as error detection units, and coprocessors. The relationships are general enough to cover any new subsystems MICON might later be extended to incorporate. This knowledge exists tn the form of rules. An example of the tasks involved with constructing the structural relationships at Level 1 are:

- Instantiate - I/O
- Instantiate - Processor
- Connect - control - signals
- Connect * decoder - to - I/O - subsystem
- Connect - I/O - subsystem - to - address - bus.

The design representation built from structural knowledge forms a map for controlling further design since it expresses general relations between subsystems* but is not yet processor or component specific.

Refinement knowledge describes how to refine the abstracted structural relationships for a given desrfjn. This is the richest set of knowledge in the system. There are two parts to refinement knowledge: mtra-subsystem knowledge and inter-subsystem knowledge.

Intra-subsystetn knowledge provides the detailed design knowledge required to construct actual subsystems, such as I/O. Utilized primarily during refinement from Level 1 to Level 2, this knowledge includes the processor and component specific rules needed for actual design situations. Taking I/O as an example: there exist specific rules for each processor to accomodnte the specialized interrupt handling mechanisms. These rules cover the refinement of devices in the same family as well as non-processor-family devices. Non-processor-family devices require a supplementary set of knowledge that describes how to design hardware to interface between device families. An example of the design tasks required to interconnect an I/O subsystem is:

- Instantiate - interrupt - control - logic
- Instantiate - I/O - device.

Ouring the execution of these tasks, appropriate intra-subsystem knowledge is invoked to perform the design.

Inter-subsystem knowledge contains the knowledge necessary to interconnect the subsystems. While the design of the interior of most subsystems is performed by component specific rules used to establish connections, the connection of major subsystems is done by matching signal names. Level 1 dictates the subsystems between which connections are to be made, via functionally named signals. Level 2 representation decomposes the functional signals based on individual processor requirements. While most of the design is handled by matching signal names, this knowledge partition contains additional rules to resolve the inevitable exceptions. For example, processors lacking the separate Read and Write signals expected by MICON have these signals created.

Component knowledge describes various attributes of the hardware components MICON designs with. These attributes include input/output signals, power, area, etc. Most of the component knowledge is represented as a set of facts about the component, however this is not always sufficient. For example, most I/O devices, except for commodity chips, are constructed to match some manufacturer's processor family. However, there are always some minor exceptions that require small modifications in order to match a family interface and protocol specification. Thus, in some cases axiomatic knowledge is supplemented with device specific rules to override the default values. The other form in which component information is found is as a set of data objects. This would include the detailed hardware information (e.g. the activity of an asserted signal, pin numbers, power requirements, etc.) required for actual implementation.

Problem-solving knowledge is the design knowledge needed to direct and sequence the problem-solving strategy. This knowledge determines when design tasks should be invoked and provides a strategy for developing a design. For example, this knowledge prioritizes pending design tasks depending on a given design situation. Additionally, demon rules monitor the design for violation of constraints such as board space, total cost, etc.

These well defined partitions help to implement abstractions and guide the further development of MICON. A well crafted framework exists for adding new components and design techniques, by adding knowledge within appropriate partitions without affecting the rest of the system.

4.2. Problem-Solving

MICON utilizes a combination of means-ends analysis and match methods. In conjunction, explicit tasks help to direct design activity; invocation and sequencing of the tasks is done by problem-solving knowledge. As in P·MICON, the inference engine is based upon the following steps: specification, selection, and instantiation. These steps are referred to as contexts in MICON. Within each context many design tasks are performed. Generally, design tasks are scheduled using an agenda mechanism; the conditions of invocation are based on the state of the design. However, there are cases when MICON has to reorder the tasks dynamically to handle an exceptional design case. Exceptions arise when a constraint is violated and it is necessary to take immediate corrective action. The ability to handle constraint violations is the major difference between P-MICON and MICON inference engines. Examples of design tasks used in MICON were given in the previous section.

Frequently, constraints are viofated during the instantiation context requiring backtracking to the selection context for choosing a different component mix. In backtracking, portions of the design must be removed and those portions of the design related to the removed portion must be found and marked as potentially inconsistent. This process generally involves search through the design already constructed. Since backtracking can occur often, a means of reducing .this search is necessary. To facilitate backtracking, dependencies are recorded during the design process. Dependencies in MICON can be thought of as a set of pointers from a design object to its related object at the next highest abstraction level. They are formed during refinement. Dependencies are maintained for both components and signals. By maintaining this information, removing a portion of a design and determining related parts of the design are simple. P-MICON*s assumption of nearly-independent subproWems is relaxed in MICON by explicitly reasoning with subproblem interactions through dependency information.

5. Summary

MICON has evolved into a sophisticated system for single board computer design. The system utilizes three representation levels: functional logic, specific logic and specific hardware. Its knowledge base is partitioned into five types of knowledge: selection, structural, refinement, problem-solving, and component. The inference engine of MICON is based on a simple design model, consisting of three steps: specification, selection, and integration.

At present the first two representation levels, and the design tasks necessary to generate them, have been implemented. The third level is in the process of being implemented. Future plans include a knowledge-acquisition tool to assist in developing a more complete rule base.

6. Acknowledgements

The authors would like to thank members of the DEMETER project at CMU, in particular Veerendra Rao, Laurence Goodby, Dario Giuse, and Janaki Akella for their technical advice.

References

[1] William P. Birmingham, Daniel P. Siewiorek.
MICON: A Knowledge Based Single Board Computer Designer.
In *Proceedings of the 21st Design Automation Conference.* IEEE and ACM-SIGOA, IEEE Computer Society, 1984.

[2] M. Bushnell, S. Director.
ULYSSES- An Expert System Based VLSI Design Environment
International Symposium on Circuits and Systems :632-646,1985.

[3] C.L.Forgy.
OPSS User's Manual.
Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, 1981.

[4] R. Joobbani, D.P. Siewiorek.
WEAVER: A Knowledge-Based Routing Expert.
In *Proceedings of the 22nd Design Automation Conference.* IEEE and ACM-SIGDA, IEEE Computer Society, 1985.

[5] Jin H. Kim.
Use *of Domain Knowledge in Computer Aid for IC Cell Layout Design.*
PhD thesis, Carnegie-Mellon University, Department of Electrical and Computer Engineering, 1985.

[6] J. McDermott.
R1: A Rule-based Configurer of Computer Systems.
Technical Report CMU-CS-80-119, Department of Computer Science, Carnegie-Mellon University, 1980.

[7] T:M. Mitchell, L.I. Steinberg, S Kedar-Cabelli, V.E. Kelly, J. Shulman, T.Weinrich.
An Intelligent Aid for Circuit Redesign.
In *Proceedings of the National Conference on Artifical Intelligence.* AAAI, William Kaufmann Inc., 1983.

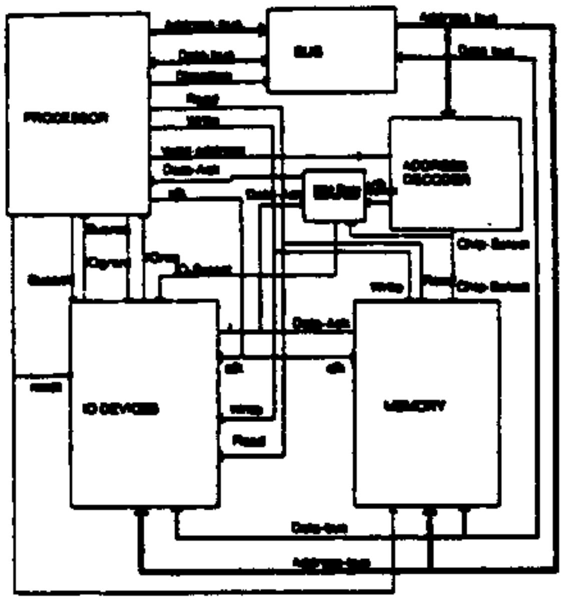

Figure 7*1: Level 1 Template.

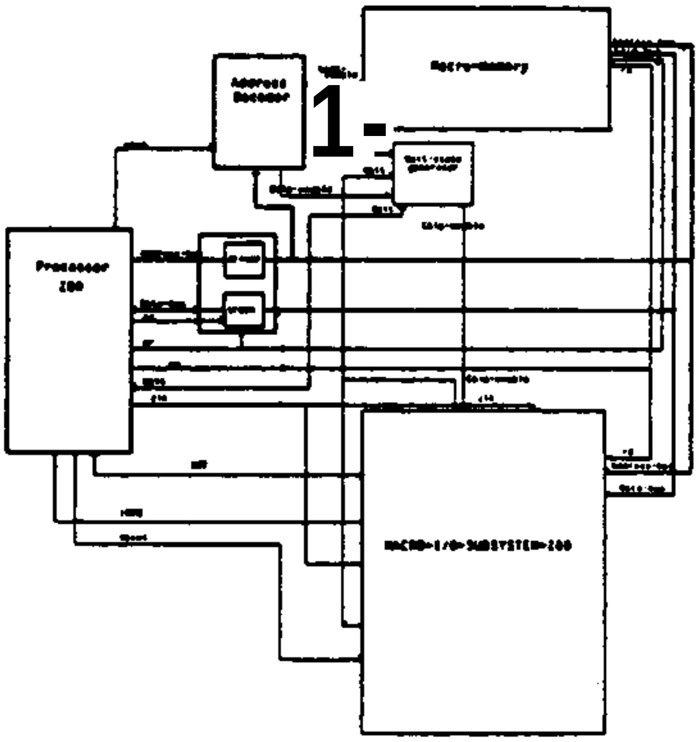

Figure 7-2: Level 2 sample design.

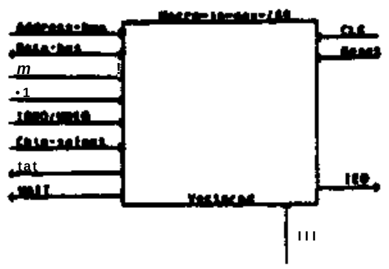

Figure 7-3: Level 2 Z80 **I/O macro.**

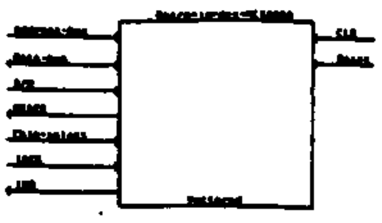

Figure 7-4: Level 2 MC68000 I/O macro.

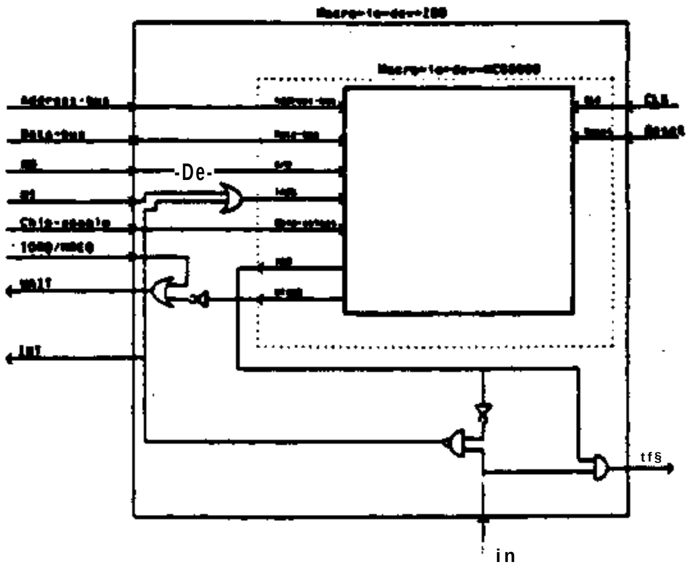

Figure 7-5: MC68000 macro with Z80 interface.

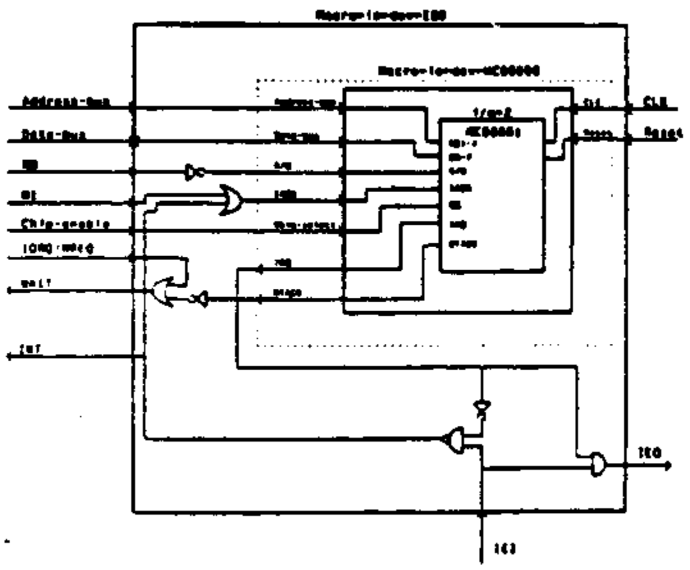

Figure 7-6: MC68681 with Z80 interface.