Hierarchical Design Synthesis: The M1 System

by

Anurag P. Gupta, Daniel P. Siewiorek

EDRC18-09-89

Hierarchical Design Synthesis

The M1 System

Anurag P. Gupta and Daniel P. Siewiorek

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
412-268-6665
anurag@maxwell.ece.cmu.edu

4 November 1988

This research was funded by the National Science Foundation grant DMC-8405136 to the DEMETER project and the Engineering Design Research Center, Carnegie Mellon University, an NSF engineering research center supported by grant CDR-8522616. The views presented in this report are solely those of the authors.

Abstract

Rapid advances in semiconductor technology are motivating the development of computer-aided design tools to assist human designers at the higher levels of the design process. In particular, there is a need for tools that aid system-level design synthesis - the process of producing a detailed artifact satisfying high level specifications given a set of components. A knowledge-based approach that uses a hierarchical design synthesis methodology is described. M1 is a knowledge-based system that implements the approach for the domain of single board computer design. A set of experiments which indicate M1's capabilities and the effectiveness of the approach in organizing and utilizing knowledge are presented.

1 Introduction

Advances in integrated circuit fabrication technologies have increased the complexity of design. To cope with complexity, engineers use computer-aided design (CAD) tools that automate the lower-level design tasks such as placement, routing and layout generation. However, as complexity continues to increase, there is a strong motivation to raise the level at which the designer interacts with CAD tools by automating higher levels of the design process. Hence, much research has been devoted to automating the process of design synthesis.

The design synthesis process maps abstract behavior and structural specifications into detailed hardware implementations. In addition to reducing the complexity seen by human designers, automating the design synthesis process reduces product development time, reduces errors, allows production of inexpensive customized designs, and aids in the documentation and distribution of design expertise. Synthesis occurs at several levels because designs can be described in varying amounts of detail. Researchers [21] [9] have used the level of abstraction of the design to classify various synthesis tasks. For example, *logic synthesis* maps a set of logic equations into gates; *register-transfer synthesis* maps an algorithm, control-flow behavior or register-transfer behavior into a register-transfer structure. Several automated systems for these tasks have been developed and are gaining acceptance in industry [20] [14]. Influenced by the combination of maturing logic synthesis systems and continuing increase in design complexity, interest is gradually shifting from logic synthesis to register-transfer synthesis. Extrapolating this trend, it is reasonable to expect that automation of design synthesis will continue to move toward higher levels of abstraction. Already, some interest has been shown in *system-level synthesis* that maps an abstract description of a system into a set of components, which could either be existing artifacts, or in turn be synthesized by a lower-level synthesis system.

System-level synthesis differs from lower-level synthesis tasks in that it is a loosely defined task. There is no equivalent of a hardware description language nor a set of boolean equations to clearly define the requirements for or *specify* the artifact being designed. For example, a personal computer system is specified by the name of the processor, its clock speed, co-processor name (if any), amount and type of memory, number and type of input-output ports *etc,* without detailing the interfacing of these components. Also, there is no complete and well-defined theory for relating behavior and structure at the system level. To use a domain-theory existing at lower levels, one would have to characterize the artifact using lower level primitives resulting in the explosion of detail. The combination of these factors makes an algorithmic formulation infeasible. Instead, a knowledge-based approach is used, as knowledge can be culled from designers who routinely address system-level synthesis.

Knowledge-based systems (KBS) have been successfully applied to a variety of CAD tasks that have defied algorithmic formulation. The following characteristics of the design synthesis task make a KBS suitable:

- KBS present a good way to model the strong data-driven nature of intelligent action. This is especially so in ill-structured domains, like design synthesis, where the experts are unable to provide an algorithm for solving the problem but rather describe the solution in terms of situations that can arise and the responses that are appropriate for these situations.
- As a design synthesis system has to track new technology and design styles, new knowledge has to be continually added. KBS allow new knowledge to be easily added accounting for new situations without disturbing the rest of the system.

Several design synthesis KBS's have been developed for a wide range of domains, including analog circuit design [11], air cylinder design [7] and preliminary structural design of buildings [12]. In the digital system design domain, KBS include XCON/R1 [13] that configures a VAX for a partially specified order using off-the-shelve components and VEXED [15, 18], an interactive knowledge-based consultant and editor that assists the user to find an NMOS implementation of logic designs.

This paper describes M1, the synthesis module of the MICON system [1, 2] that designs small computer systems. M1 addresses a particular instance of the system-level synthesis problem - designing a single board computer (SBC) to satisfy high level input specifications using existing components. Input specifications describe the functionality required, *e.g.* type of micro-processor, clock-speed, amount and type of memory, amount and type of input-output devices *etc*, as well as global constraints such as board size, board cost and power dissipation. Existing components are parts listed in manufacturer's databooks (*e.g.* MC6809, MC68A50, PD62256, 1000 ohm resistor, 25-pin connector *etc*), custom-designed parts and structures (*e.g.* ASICs, error correcting code generator and checker, *etc*).

KBS are composed of two major elements: the problem-solver (or architecture) and the knowledge-base. The problem-solver encompasses a methodology for solving a class of problems. The knowledge-base contains knowledge specific to the task domain that is used by the problem-solver to find the solution. The problem-solver in M1 utilizes a hierarchical design synthesis (HDS) methodology. The HDS methodology is based on the general problem-solving strategies of *problem decomposition* and *hierarchical abstraction*; the methodology is hence applicable to a wide range of problems.

Several methodologies for design synthesis have been proposed, some of which can be found in [6] [19]. Few of these are easily extensible beyond the domain for which they were developed[1]. The HDS methodology, though similar in many respects to existing methodologies, is largely domain-independent [4]. HDS also supports automated knowledge-acquisition (KA). KA is the transfer and transformation of problem-solving expertise from some knowledge-source, usually human-experts, to a performance program. As the performance of a KBS depends strongly on its knowledge, KA becomes the bottleneck in the development of most systems. Usually, a knowledge engineer, familiar with the structure and representation of the KBS, culls knowledge from a domain-expert and puts it into the system. To facilitate knowledge growth, there is a strong motivation for automating KA. The HDS methodology provides a framework for CGEN [3], the KA tool for M1, that has been successfully used to capture design expertise.

[1] An exception is the methodology for VEXED [15] which has been extended to mechanical design [16].

The general **HDS methodology** is motivated and described in Section 2, followed by its extension for the SBC **domain. M1, based on the** HDS methodology for the SBC domain, is described in Section 4. Finally, some experimental results are presented.

2 Hierarchical design synthesis

This section first describes the general design synthesis problem. Abstraction and problem decomposition are motivated as strategies for solving this problem. Finally, the HDS methodology based on these strategies is described.

2.1 The design synthesis problem

Design synthesis is the process of producing a detailed artifact that satisfies some high level behavioral and structural specifications. This paper addresses the automation of a particular instance of design synthesis where a complete artifact is formed by integrating several primitive components. For example, in the single board computer (SBC) design domain, design synthesis requires selecting parts from databooks and interconnecting them suitably.

Design synthesis is a complex problem due to several reasons.

- The design space (the set of possible designs) is very large. There is a combinatorial explosion of design possibilities.
- There is interaction between steps in the design process.
- There is no good evaluator for partially designed artifacts. One cannot tell from a fragment of a design whether that fragment is part of a satisfactory complete solution.
- Human designers do not use a formalized theory in the synthesis process. However, an automated system requires a well-defined methodology.

2.2 Abstraction and Problem-decomposition strategies

A brute-force search of the large design space not suitable as it would take an unreasonable amount of time. Knowledge of the important aspects of the problem can be used to construct an abstracted representation of the design space. Abstraction reduces the combinatorics of a large design space and make the search quicker.

Another strategy for coping with a complex problem is to decompose it into a set of smaller and easier sub-problems, solve each sub-problem and integrate the partial results into a solution for the total problem. Designers have advocated this approach for design of such diverse things as computer programs, machines and buildings. However, problem-decomposition must be used judicially as the penalty for handling sub-problems interactions may outweigh the advantages.

The design synthesis problem is decomposed into:

- selecting components that satisfy the behavioral specifications for the artifact - a search for function.
- integrating the selected components into the artifact - a search for structure.

In addition to reducing complexity, the decomposition is appropriate as issues involved in both searches are significantly different. Also, handling sub-problem interaction is simple as it is limited to conveying the result of the search of function to the search for structure.

The search for function occurs along a *functional hierarchy* that is formed using abstraction and problem-decomposition. The nodes of the functional hierarchy are termed *parts.* The leaf nodes correspond to the primitive components and are termed *physical parts.* All other nodes correspond to a functionality and are termed *abstract parts.* Nodes at which an abstract part is decomposed into several successor parts are termed *AND nodes.* Nodes at which an abstract part is formed by functional abstraction of the successor parts are termed *OR nodes.*

An example functional hierarchy for the SBC domain is showed in Figure 1. The single board computer (SBC_0) is decomposed into the processor (PROC_0) sub-system, the memory (MEM_0) sub-system, the to (IO_0) sub-system, the address decoder (ADDR_DEC_0), *etc.* The processor physical parts *(e.g.* MC6800, MC6809, MC6502) have been abstracted to processor-family specific parts (e.g. 68XX_PROC_0, 80386_PROC_0) that are in turn abstracted to the generic PROCJ). The io sub-system is decomposed into serial input-output (SIO_0) and]3arallel input-output (PIOJ)). The rest of Figure 1 can be interpreted in a similar manner.

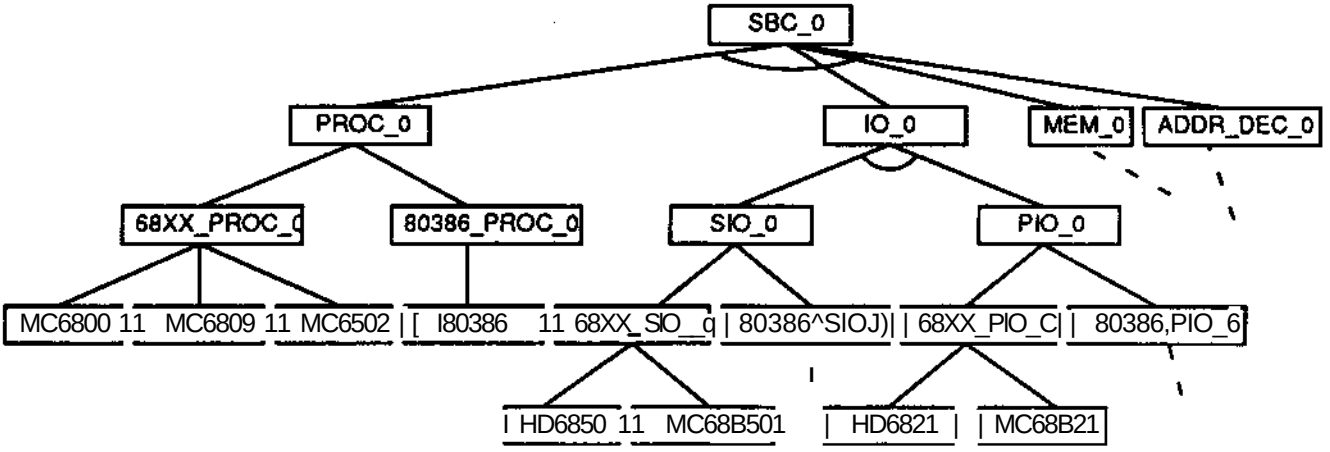

Figure 1: Design hierarchy in single board computer design domain

The functional hierarchy has several benefits:

- The total number of possibilities explored in the search for function is reduced.
- The complexity of each decision made in the search for function is reduced.
- The combinatoric explosion of knowledge is reduced by sharing it between abstract parts.

These are discussed in greater detail in [10].

The search for structure must integrate the selected successor part with the abstract part. Typically, given a successor part and a particular design situation, a unique solution for the search for structure exists. For example, given that a MC6850 has been selected for 68XX_SIO_0 and an RS232 port interface is required, a single structure that integrates the MC6850 and the RS232 port to the 68XX_SIO_0 exists; another unique structure exists if an RS422 port interface is required instead. Thus, the search for structure essentially requires recognizing the design situation and using the appropriate structure - it is *deductive.* On the other hand, the search for function involves *choosing* a successor based on some criterion. If there is a case where more than one structure is satisfactory, more information must be added to the design situation to select one of them. Alternatively, both structures could be abstracted to separate parts and the search for function could select one of them based on

some criterion[2]. Thus, the restriction of having a unique solution for the search for structure is not a limitation of the framework; on the contrary, it aids in distinguishing the nature of the two searches and also is the basis for the knowledge-acquisition framework (discussed in detail in Section 4.3).

2.3 A hierarchical design synthesis methodology

The design synthesis process can be viewed as successive top-down refinement of an abstract part at the root of the functional hierarchy into a set of integrated physical parts that are the leaves of the functional hierarchy. The problem-solver traverses down the functional hierarchy, one level and one abstract part at a time, until it reaches the physical parts. Each traversal step, termed a *design cycle,* involves a search for function followed by a search for structure. Thus, the problem-solver essentially does alternate search for function and structure until the entire design has been refined into physical parts. The design cycle is discussed in greater detail after some basic terminology is introduced.

A *part model* is used to organize the information associated with each part. In this model, each part has of the following entities:

Characteristics: represent the properties of the part and are (attribute-name, attribute-value) pairs. Both abstract and physical parts can have any number of characteristics. For example, a MC6809 has characteristics (MAX_CLOCK_SPEED, 1000000) and (POWERJ5ISSIPATION, 1000);

Specifications: represent the behavioral and structural requirements that the design of the part is to satisfy. They are (attribute-name, attribute-value) pairs but unlike characteristics, the value is filled in at run-time depending on the user-input specifications and the evolving state of the design. For example, MIN_CLOCK_SPEED is a specification for the generic processor part PROC_0 that specified by the user at run-time. Clearly, only abstract parts can have specifications.

Links: represent the relationships between the part and its predecessors in the functional hierarchy. They are (part-name, link-type) pairs. As the functional hierarchy is defined prior to the execution of the design synthesis problem-solver, the links are static. Both abstract and physical parts can have any number of links. Multiple links mean that the part can be used to satisfy the functionality for any of the predecessor parts. For example, MC6809 has a link (68XX_PROC_0, OR-link).

Sub-problems interact and a mechanism for handling the interactions is needed. Interactions are essentially communication of (attribute-name, attribute-value) pairs, termed *reports,* from one sub-problem solution to another. If the design of one abstract part A_1 depends on another abstract part A_2, appropriate reports may be generated during the design of A_2 that can be suitably used in the design of A_v For example, the design of MEM_0 depends on the design of PROCJD because parts used to design memory need to satisfy certain processor-memory timing relations. During the design of PROC_0, a report MAX_ACCESS_TIME with suitable value[3] is generated. This report is used in the design of MEMJ) to ensure that the processor-memory timing relationship is satisfied.

The design cycle refines an abstract part, say A_{jf} into its successor parts in the functional hierarchy. The sequence of steps in the design cycle are illustrated in Figure 2. Each of the steps are explained

[2] If neither alternatives is possible, both structures are equivalent in all known respects and a designer needs to know just one[1]

[3] Computed using other timing information (e.g. processor-clock-speed, processor-setup-time).

next.

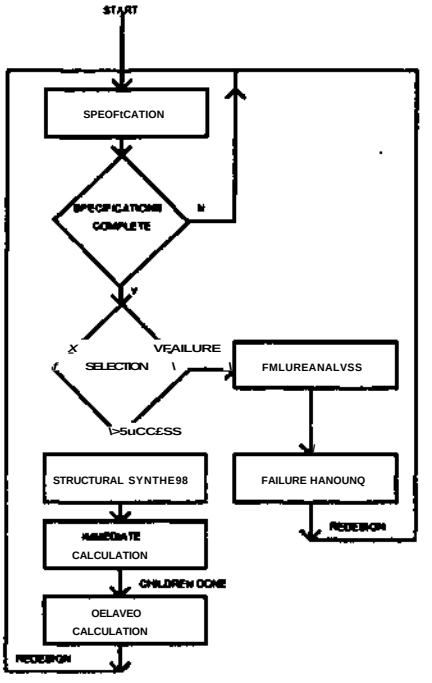

Figure 2: The design cycle

Specification (d_{spec}) The specifications associated with A_i are filled (i.e. a value is assigned to them). Any of the following three mechanisms (or a combination of them) can be used:

- *Query.* The user is prompted with an appropriate question and the response is used to fill in the specification.
- *Translation:* The specifications and characteristics of the predecessor of A_i are used to fill in the specification.
- *Formulation:* The reports are used to fill in the specification.

The design can proceed only when all the specifications associated with A_i are filled[4].

Selection (d_{select}) At an OR node, one of the successor parts of Aj is chosen. No action is taken at an AND node as all the successor parts of A_j are required in the design. This step corresponds to the search for function.

Structural synthesis (d_{Gasc} + $d_{template}$)
At an OR node, the selected part is integrated into Aj and thus integrated into the design. At an AND node, all the successor parts are integrated with Aj. This step corresponds to the search for structure. In Section 3, this step is divided into two distinct sub-steps.

Calculation (d^fj The design of Aj generates reports or information to be used elsewhere in the design. This step has two sub-steps:

- *Immediate* calculation: Reports are generated immediately after structural synthesis.

[4] ***Don't care*** **values can be assigned to specifications that are not relevant in a particular design situation.**

- *Delayed* calculation: Structural synthesis may bring in several abstract parts into the design and the reports to be generated by Aj may depend on reports generated by the design of those abstract parts. Hence, generation of some reports is delayed until these abstract parts have been completely designed *(i.e.* have completed their respective design cycles).

Failure analysis — When none of the successor parts satisfy the specifications for A_{jt} selection at an OR node fails. The cause of the failure is analyzed to create *failure reports* in this step.

Failure handling — Remedial action for the failure during design is taken in this step. The action would involve redesigning $A_{\{}$ and possibly other abstract parts in the design.

The *design-state* characterizes the state of the problem-solver at any time. The total design-state consists of all parts organized hierarchically, their characteristics and specifications and the global reports. However, while executing a design cycle for a particular abstract part $A_{\{(}$ only a subset of the total design-state needs to be considered thereby reducing the complexity seen by the problem-solver. The design-state for $A_{\{}$ includes:

- specifications and characteristics of A|
- specifications and characteristics of the children of $A_{\{}$
- all reports

Thus, specifications and characteristics of Aj are like local variables with their scope limited to the portion of the design involving the parent and children of $A_{\{;}$ reports are like global variables.

3 The single board computer domain

This section considers issues in the SBC domain that influence the HDS methodology.

3.1 Abstract parts and physical parts

In the SBC domain, physical parts also have pins associated with them, in addition to characteristics that describe physical and timing properties. Abstract parts are formed by generalizing/removing some characteristics of successor physical parts. However, to make the functional abstraction meaningful and useful, the pins on the physical parts must also be abstracted by forming bus-pins on abstract parts. The grouping together of several pins that have some common identity to form a bus is a popular concept in digital system design. For example, control-bus is a group of pins with different names but the same function in a broad sense - providing control to some part of the design; data-bus and the address-bus are groups of pins that differ by index. Mechanisms for pin-abstraction are discussed in further detail in [10].

Several parts may be used as merely glue logic or support circuitry. For example, resistors, capacitors, crystals and logic gates are used while integrating several parts into the design. Typically, human designers have a different perspective towards these parts. So they are represented in separate hierarchies. Some support circuitry hierarchies are shown in Figure 3. Any part in these hierarchies may be linked into the design during the structural synthesis step. For example, during structural synthesis of a part with an open-collector output, a resistor abstract part may be brought into the design from a separate hierarchy.

Another interesting issue is that some physical parts have multiple functional units. For example, the

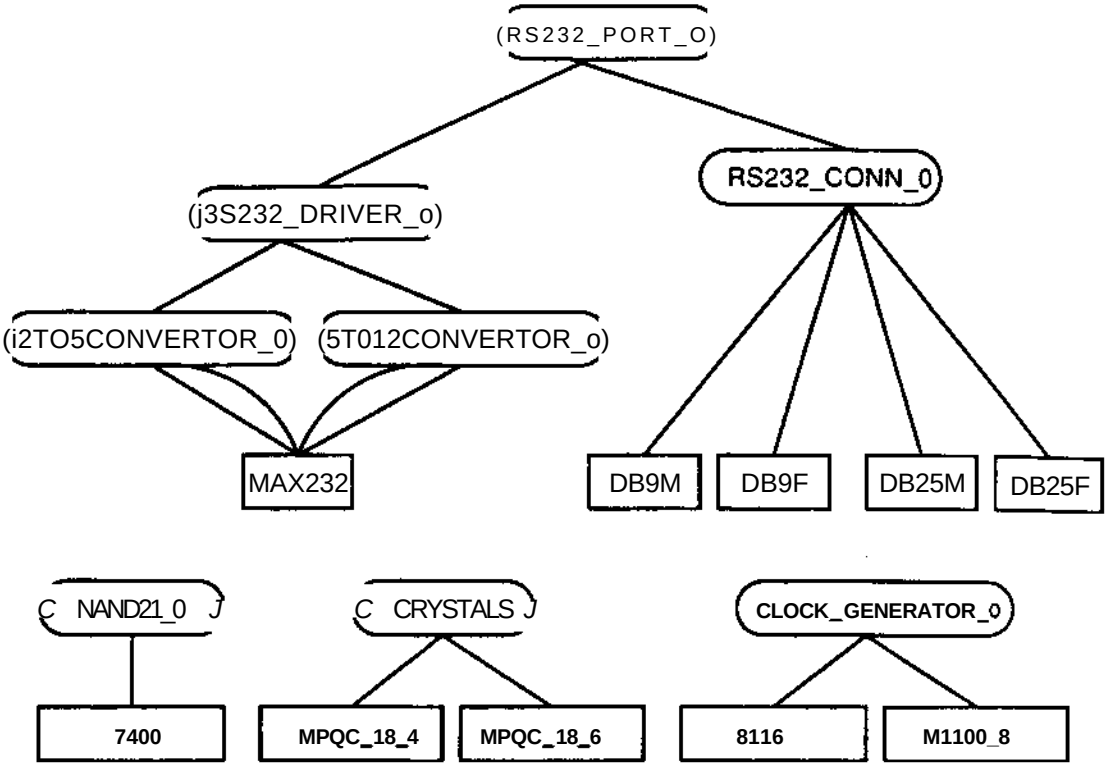

Figure 3: Support circuitry hierarchies

MC6821 has two PIO ports, the 8155 has 256 byte SRAM and three PIO ports. An additional field called *functional unit number* is added to links in the part model. These parts have several links, each with a different functional unit number. If such a part has been brought into the design, its other functions could be used if found suitable in other abstract part designs. Using existing functional units is always preferred since it reduces the total number of parts in the design. Hence, in the design cycle, a check is made for unused functional units prior to instantiating a new part.

3.2 Part selection

The part selection step at an OR node picks one of the successor parts. A simple agenda is used

1. All successor parts are candidates.
2. The number of instances of each candidate needed in the design is computed. For example, depending upon the size of memory required and the amount of memory on a chip, a different number of chips may be needed.
3. Candidates whose characteristics do not meet the abstract part's specifications are rejected. For example, the memory chip whose characteristic ACCESS_TIME is not less than the specification MAX__ACCESS_TIME for MEM_0 is rejected.
4. If no candidates remain, selection has failed. If just one candidate remains, it is selected, if more than one candidate remains, either the user may pick one or an internal weighted objective function is used to seiect one.

The design synthesis process has to satisfy certain ceilings on global resources like board area. cost and power dissipation. The objective function biases the selection to satisfy these requirements Each candidate consumes certain portions of the global resources. The function assigns a normalized penalty for each candidate based on the amount it uses. Also, the penalty is weighted by the criticality of the

resource - if the design is running out of a particular resource, its criticality increases. The objective function is given by:

$$\textit{Penalty points} = \sum_{\textit{Global resources}} \textit{Criticality} \times \textit{Normalized penalty}$$

$$= \sum_{\sim \textit{Global}\ \wedge \textit{sources}}^{1} \frac{\textit{Amount already used}}{\textit{Total amount}} \times \frac{\textit{Amount used by part}}{\textit{Amount left}}$$

The candidate with the least penalty points is chosen.

3.3 Structural synthesis

Structural synthesis instantiates and integrates the successor part(s) with the abstract part in a suitable manner. The number of successor parts to be instantiated depends on specifications that are filled only at run-time. For example, the number of 68XX_SIO_0 to be brought into the design depends upon the number of SIO ports desired by the user; the number of SRAM physical parts to instantiated depends upon the amount of memory required by the user. As the specifications cover a large range of values, the amount of integration knowledge can explode. Fortunately, in the SBC domain, the successor parts can be viewed as constituting a *structure* and the knowledge of forming the structure js independent from the knowledge of integrating this structure into the design. For example, the SRAM chips can be viewed as constituting an array structure and the knowledge of how a variable number of SRAM chips form an array is independent from the knowledge of how the array structure is integrated into the memory sub-system. This leads to the division of the structural synthesis step into two separate sub-steps:

- *Part expansion (d^^:* A variable number of successor parts are instantiated and integrated into the structure.
- *Structural design ($d_{templ ate}$):* The successor parts in the structure are integrated with the abstract part.

In part expansion, depending upon the specifications for the abstract part, different structures may be formed. Some common structures in the SBC domain are:

- *Independent* Each instance of the successor part is independent of the other. For example, each 68XX_SIO_0 instantiated is independently integrated to SIO_0.
- *Array.* The successor parts are organized as an array with variable dimensions. $d_{template}$ is used for the boundary parts in the array only. Integration of the other parts of the array is decoupled from $d_{tempJat0}$ and also from the dimensions of the array. For example, memory chips are organized as an array with dimensions depending upon the data-bus width and the amount of memory required; the address-bus pins and chip-select signals connect across rows while the data-bus pins connect across columns.

At an OR node Aj, structural design integrates the selected successor part in the structure with A_r At an AND node A_{jp} structural design instantiates all successor parts and integrates them with A_r Structural design is done using *templates.* A template is a chunk of knowledge about structural design for a particular abstract part in a particular design situation. An example template detailing the integration of a 6850 SIO physical part to a 68XX_SIO_0 abstract part is shown in Figure 4. Salient features of a template are:

- The pins on the abstract part constitute a functional boundary for the template. Integrating the parts in the template with this boundary effectively integrates them into the rest of the design. In Figure 4, the pins on the 68XX_SIO___0 constitute the functional boundary.
- The selected successor part is the major functional component (only for OR nodes). The 6850 is the major functional component in Figure 4.

- Support circuitry may be required to complete the design. The RS232_DRIVER_0 and the CLOCK_GENERATOR_0 are support circuitry in Figure 4.
- Invocation conditions for each template are unique and consist of the specifications of the abstract part, reports and the selected successor part in the structure. There may be several templates for an abstract part, albeit with different invocation conditions. Invocation conditions for the template in Figure 4 include the specification that a RS232 port is required for the 68XX_SIO_0. Another template would be used if a RS422 port was required instead.

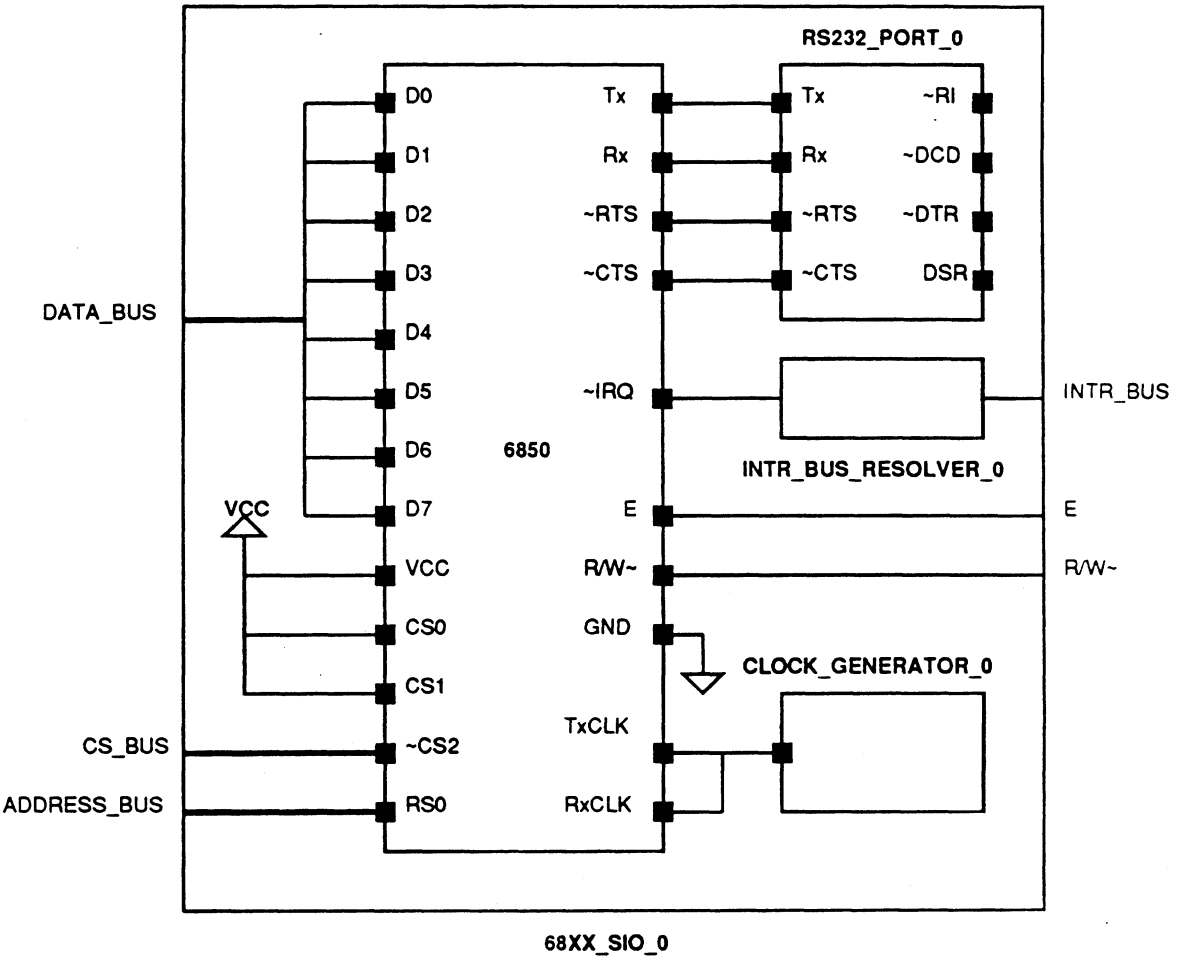

Figure 4: Template showing integration of 6850 to 68XX_SIO_0

Chunking the knowledge into a template results in a representation that is efficient for both storage and utilization. The increased granularity of the knowledge representation also means lesser generality. There is a trade-off between template granularity and generality. However, since problems have been successively decomposed into sub-problems, the granularity is not too large. Also, templates at higher abstraction levels in the functional hierarchy are quite general.

Templates essentially represent knowledge of how the problem-solver must map from one level in the functional hierarchy to a lower level. Designs are created by combinations of templates as the problem-solver traverses the hierarchy. Thus, given a fixed functional hierarchy and a set of templates, the designs that a system can produce are enumerable *a priori* (except for the variability allowed by part-expansion). Hence, in order to cover a wide range of designs, a large number of templates would be needed.

4 The M1 system

M1 is a knowledge-based system based on the HDS methodology for designing single board computers. M1 has been implemented in OPS83 [8].

4.1 Architecture

The control for monolithic system can get unmanageably complicated. The multi-layered control approach recognizes the meta-problem of deciding which action to apply next, and builds a separate problem-solver for the meta-problem [17]. M1 has three distinct layers of control based on the HDS methodology [10]:

Top layer — Decisions about which abstract part's design is to be attempted next is made at this level. Designing of abstract parts with incomplete specifications is blocked. The design each abstract part is attempted in turn until all abstract parts have been completely designed. Decisions to execute external analysis tools are made at this level. For example, control is passed ASSURE [5], which analyzes the design for reliability metrics and makes suggestions for enhancing system reliability, at this level.

Middle layer — Sequencing of the design cycle for each abstract part is done at this level. For each step, a certain knowledge-base partition (described in next subsection) is activated.

Bottom layer — Decisions about which rule in a particular knowledge-base partition is to be fired is made at this level. The control is provided by a simple conflict resolution strategy.

In addition to simplifying the implementation of the control structure, the multi-layered approach also allows for easy modification of control at any level. For example, adding another step to the design cycle would just influence the middle layer; passing control to an external expert like ASSURE would influence just the top layer.

4.2 Knowledge-base

The knowledge about part models, *i.e.* characteristics, specifications and links for all parts (see Section 2.3), is stored in a central database for the MICON system [2]. The knowledge that M1 reasons with is present in its knowledge-base. M1's knowledge-base is divided into disjoint partitions (or contexts), each corresponding to a particular sub-task. Each rule resides in one and only one partition. Strict separation of knowledge reduces the interactions in the rule-base (which usually hinder the growth of a KBS) and facilitates knowledge-acquisition.

Each of the following knowledge-base partitions in M1 correspond to steps in the design cycle:

- **Specification knowledge**
- **Selection knowledge**
- Structural synthesis

- Part expansion knowledge
- Structural design knowledge

- Calculation knowledge (Immediate and Delayed)
- Failure analysis knowledge
- Failure handling knowledge[5]

An additional partition, *Architecture and support knowledge,* provides M1's control structure and other functions like database interaction and user input-output handling.

4.3 Knowledge acquisition

Knowledge about part models is stored in a database and is added using an interactive data-entry program. Knowledge in MVs KB is captured using the automated KA tool, CGEN. CGEN addresses only some partitions in the KB whose knowledge is expected to grow as Mrs design space increases.

Structural design, calculation and specification knowledge are organized around templates and are acquired in one session. Consider the situation where M1 has sufficient knowledge to proceed till the ^*template* $^{ste}P^{forabs}$^act part A_{jf} but it does not have a suitable template. M1 recognizes this situation and requests new knowledge from the domain-expert. The domain-expert uses CGEN to add suitable structural design knowledge. Now, the design-state for the d_{cah} step that follows the $d_{template}$ step is exactly the same. Also, if the template uses any abstract part, design cycles for them would be spawned. The design-state for the d_{spec} step for those abstract parts would be the same too. Thus, calculation knowledge for $A_{\{}$ and specification knowledge for abstract parts appearing in the template for Aj are acquired together with the structural design knowledge in a single training session using CGEN. In this manner, M1 now has enough knowledge to proceed to the $d_{tempfate}$ step for the successors of A_{r} and the KA loop could be repeated again. The capability of M1 to trigger KA helps to maintain self-consistent KB as new knowledge is input only when it is found missing. This is possible due the requirement of uniqueness of design-state (pre-conditions) for structural design or templates.

Selection knowledge depends on the specifications of an abstract part and characteristics of its successors and is acquired independently. CGEN, the KA tool, is discussed in greater detail in [3].

5 Experimental results

This section describes some experimental results for M1 in the SBC domain. An M1 session that produces a complete 6809 based SBC is presented as an example. An indication of the design space of M1 is given using a few tables. MVs capabilities and limitations are discussed using examples. Finally, experimental results that validate the HDS methodology are given.

[5]The knowledge in this partition is skeletal as the emphasis in this phase of development was on amassing synthesis knowledge

5.1 Design example

A session with M1 that produces a 6809 microprocessor based SBC design is given in Figure 5. M1 boots up by attempting to design SBCJ), the root node of the functional hierarchy. The user is queried for various global requirements. As SBCJ) is an AND node, the selection and part expansion steps take no action. Structural design integrates the various sub-systems *e.g.* processor, memory, io, address decoder, *etc.* The design cycle of SBC_0 is blocked while it waits for the successor abstract parts to be designed. As the system does not have any explicit sequencing of design activities, any sub-system design may be activated next. Here, M1 activates the IO_0 design which starts its design cycle. Some specifications for IO_0 are collected by prompting the user with questions. However, as IO_0 cannot be designed until certain timing attributes *(e.g.* max_accessjime) have been specified which in turn need certain reports generated by the PROCJ3 design, its design cycle is blocked. Next, M1 goes through the design of MEM_0 in a manner similar to IO_0. Eventually, M1 activates the PROC_0 design. Specifications for PROCJ) are obtained from the user. In the selection step, M1 gets all successors of PROCJ) and all but 68XX_PROC_0 are rejected as they are not suitable for the specified processor name. 68XX_PROC_0 is instantiated in the part expansion step. After structural design integrates 68XX_PROC_0 with PROCJ), the PROCJ) design is blocked waiting for the 68XX_PROC_0 to be designed. Next, the 68XX_PROC_0 design is activated. M1 continues to go further in this manner until the 68XX_PROC_0 design is completed. Finally, the PROCJ) design is activated and the delayed calculation step is completed. M1 repeats this process until the SBCJ) design is completed.

Performance measures for this design session in M1 are given in Table 1.

5.2 Design space

The design space for a system describes the set of designs that it could possibly produce. M1 's design space is rapidly expanding as new knowledge is being added to the system. The Tables 2 through 6 give an indication M1's current design space. M1 can produce many designs for four micro-processor families. One of the designs, based on a Motorola 68008, was constructed and successfully executed at the designed clock rate. CGEN was used to build MVs knowledge-base and more details on the size and growth of the KB can be found in [3].

5.3 Capabilities and limitations

M1 attempts to meet both the board cost and board area constraints. In the selection step of the design cycle, the weighted objective function makes an implicit cost-area tradeoff.

This experiment illustrates the cost-area tradeoff capability in designing a ROM array. The choice of physical parts available for this design are given in Figure 6[6]. Three ROM chips of different bit organization satisfy the timing constraints. A different number of each chip would be used in the design depending upon the amount of ROM memory required by the user. Each choice would incur a cost and area expenditure as graphed in Figure 6. M1's selection function *monitors the* global resources available and biases the choice in favor of smaller consumption of the scarce resource. For this experiment, design states with varying global resource allocation and usage were hand-created, as exploring all the

[6]**The design space has been simplified to include three chips from the same manufacturer. The** prices **for** chips are *real world* **prices acquired from a vendor's catalog.**

```
%m1 -b MICON
 ***  Ml : Single Board Computer Synthesis Tool  ***
.....
Activating SBC_0 design
     Specification
          Total amound of board area < sq. inches > : 32
          Total amount of board cost <nearest dollar > : 5000
          Total amount of powe^dissipation allowed < xtW > : 10000
     Specifications Complete returns Y
     Resource check returns NONE SUITABLE
     Selection
          AND node : no action
     Part expansion
          AND node : no action
     Structural design : AND node
     Immediate calculation
Blocking SBC_0 design : waiting for children to be done
Activating IO_0 design
     Specification
     Is PIO needed < Y N > : Y
     Is SIO needed < Y N > : Y
     Is TIMER needed < Y N > : N
     Specifications Complete returns N
Blocking IO_0 design : incomplete specifications
Activating MEM_0 design
     Specification
     Is DRAM needed < Y N >. : N
     Is SRAM needed < Y N >> : Y
     Is ROM needed < Y N > : Y
     Specifications Complete returns N
Blocking MEM_0 design : incomplete specifications
.....
Activating PROC_0 design
     Specification
     Name of Processor Family : ?
     Processor name : 6809
     Minimum clock speed < Hz > : 1000000
     Specifications Complete returns Y
     Resource check returns NONE SUITABLE
     Selection
          Rejecting 68008_PROC_0 due to mismatching PROC_NAME
          Rejecting 32016_PROC_0 due to mismatching PROC_NAME
          Rejecting 68XX0_PROCJ) due to mismatching PROCJNAME
          Rejecting 80386_PROC_0 due to mismatching PROC_NAME
          Selected 68XX_PROC_0
          Structure : independent
     Part expansion
     Structural design : OR node
     Immediate calculation
Blocking PROC_0 design : waiting for children to be done
Activating 68XX_PROC_0 design
.....
Done 68XXJPROCJ) design
```

```
Activating PROC_0 design
     Delayed calculation
Done PROC_0 design
.....
        Number of SRAM BYTES needed  : 10000
        Name of SRAM chip to use  : ?

        How many SIO units are needed : 1

        External interface for this port < RS-232 >  : RS-232
        SIO baud rate < Hz >  : 4800
        Should interrupts be maskable < Y N >  : Y
        Priority level for interrupts  : 0
        External port size < 3 5 9 >  : 5
        Type of port < DCE DTE >  : DTE
        External connector polarity < MALE FEMALE >  : MALE
        External connector size < 9 25 >  : 25
        Protocol supported < SYNC / ASYNC / BOTH > : ASYNC
        Should this SIO generate interrupts < Y N >  : Y
        Enter a tag for this SIO  : SIO_1
        Baud switch < FIXED HW_SWITCH SW_SWITCH >  : HW_SWITCH
...
Activating SBC_0 design
     Delayed calculation
Done SBC_0 design

Do you want a netlist [y/n] : jyl
     With abstract part connections [y/n] : I n I
```

Figure 5: Execution trace of M1 for 6809 based design

Performance measures for M1		
Rules fired	Top layer	248
	Middle layer	606
	Bottom layer	2634
	Total	3488
Run time (sees)	Real time	2197
(Microvax)	User time	267
	System time	15
	Database time	1911
Parts count	Physical parts	28
	Abstract parts	72
	Total	100
Nets count	Physical part nets	149
	Total nets	252

Table 1: Performance measures of M1 for 6809 based design

Design space for Processor sub-system	
Bus Family	Component
68XX	MC6809P
	MC68A09
	MC68B09
	MC6809ES
68008	MC68008L8
	MC68008L10
68XX0	MC68010L8
80386	IAPX80386-16
	IAPX80386-20
	IAPX80386-25

Table 2: Processor summary

Design space for Memory sub-system	
Type	Component
ROM	TMS2732A-20JL
	TMS2764-20JL
	TMS27128-20JL
	TMS27256-20JL
	TMS27256-12JL
SRAM	62256
	PD43256-10C

Table 3: Memory summary

possibilities in an actual design would be tedious. The results of the experiment are summarized in Table 7.

The lines at the bottom of Figure 6 show the best choice under extreme conditions where either cost or area is scarce. When the amount of ROM memory desired is either 4K, 8K, 12K, 16K or 28K, the same chip minimizes both cost and area. For 20K and 24K, the best choice depends upon which resource, cost or area, is more scarce. When cost is scarce ($W_c \gg W_a$)[7], the 2764 chip should be selected. When area is scarce ($W_c \ll W_a$), the 27128 chip should be selected. Table 7 shows that M1 makes the choice correctly validating the objective function used in M1.

Another capability of M1 is maximizing performance while satisfying the global constraint on cost and area. For example, M1 will add wait-states to peripherals instead of using costlier chips to meet a global cost constraint. The experiment illustrating this capability of M1 can be found in [10].

[7] W_C and W_a denote criticality of cost and area respectively.

Design space for 10 sub-system		
Type	Family	Component
Serial IO	68XX	MC6850P
		MC68A50
		MC68B50
	68008	MC6850P
	68XX0	MC6850P
	80386	I8274
		18251A
Parallel IO	68XX	MC6821P
	68008	MC6821P
	68XX0	SN74LS245
Timer	68XX	MC68B40
	80386	8254A2

Table 4: IO summary

Design space for external interface for Serial IO port				
Protocol	Mode	Port Size	Connector	Driver
RS232	DTE	3 wire		
		5 wire	9 pin	MAX232
		9 wire	or	or
	DCE	3 wire	25 pin	1488/89
		5 wire		
		9 wire		

Table 5: SIO interface summary

System data	
Total **Part** models	382
Total rules	1050

Table 6: Summary of system data

A limitation of M1 is that it is a local optimizer due to its approach of dividing the task into simpler sub-tasks and solving them independently. All abstract part designs interact with each other as they use the same global resources of board area, cost and power dissipation. Although, the objective selection function used in each design considers the global design state, the decision is still made locally and pathological cases where M1 would overlook a suitable design can be found in [10]. It should be pointed out that ASSURE, the external reliability expert, seeks and achieve a global optimum since it has the whole design to critique.

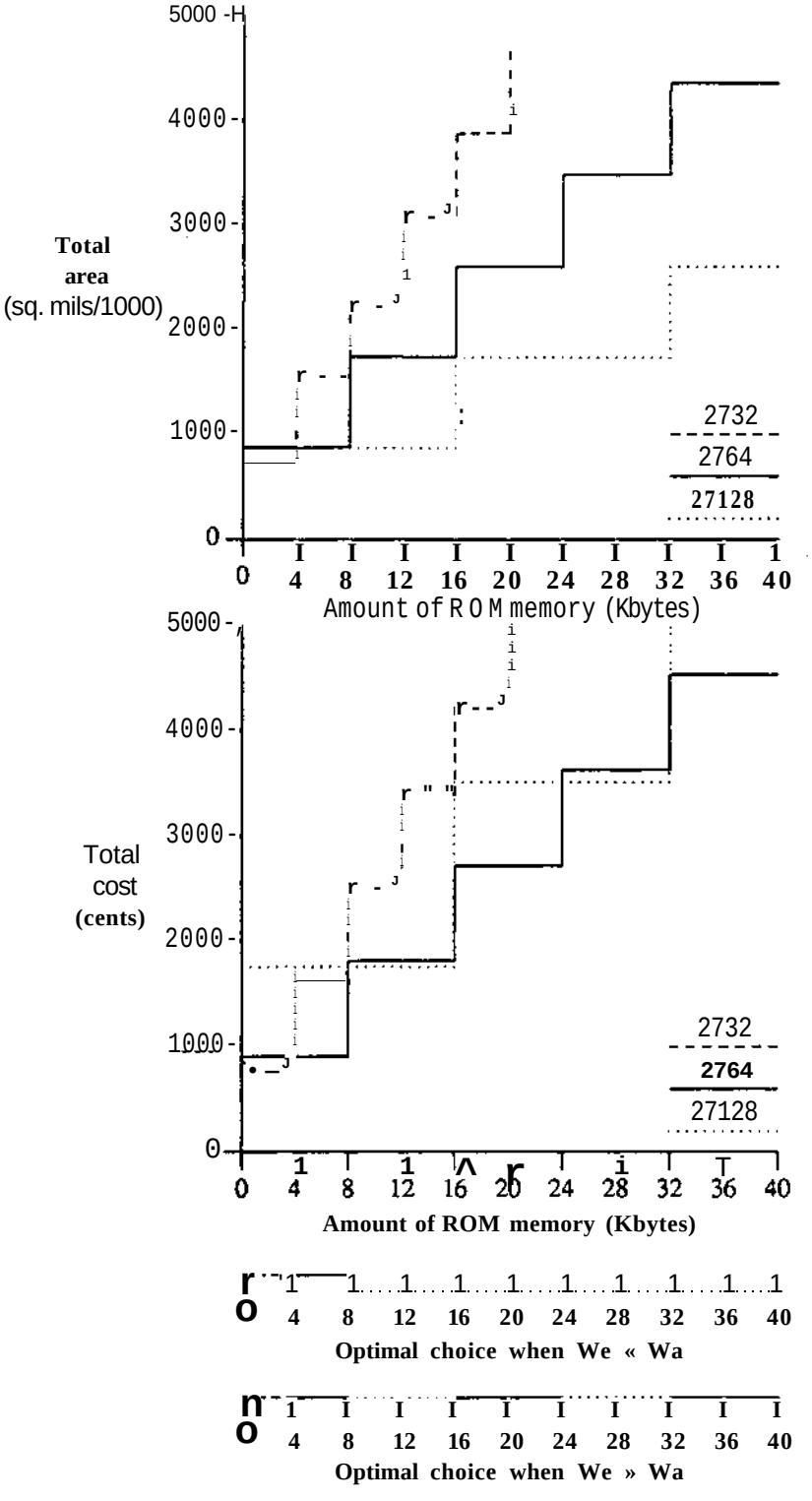

Figure 6: Design space of ROM chips

Amount of ROM memory	Cost left	Area left	Chip chosen by M1
4K	-	-	2732
8K	-	-	2764
12K	-	-	27128
16K	-	-	27128
20 K	HIGH	HIGH	2764
	HIGH	LOW	27128
	LOW	HIGH	2764
24 K	HIGH	HIGH	2764
	HIGH	LOW	27128
	LOW	HIGH	2764
28 K	-	-	27128

The total board cost and area were kept fixed at 100000 units for all cases.
HIGH : 80000 units, LOW : 5000 units

Table 7: ROM chip selection results for M1

5.4 Validating the methodology

Some experiments were conducted to validate the HDS methodology by comparing it with an approach, labeled X, at the other extreme that would treat the entire design as a flat structure. The HDS methodology reduces the complexity seen by the problem-solver at any stage of the design process. Complexity was measured as number of parts and connections. The complexity seen by M1 is the number of parts and connections in a template. For X, complexity is the number of parts and connections in the complete design. The average complexity for M1 was found to be a substantially less than that for X.

Also, the HDS methodology makes it possible to share knowledge across several designs. The development of a 6809 based SBC design was used as an example. Knowledge was measured as the number of part and connection information in M1's knowledge-base. The solid and dotted lines in Figure 7 correspond to M1 and X respectively. Figure 7 shows that knowledge grows much more slowly for M1 (as compared to X) when the design space covered increases. A detailed discussion of these results can be found in [10].

6 Summary

A methodology to support design synthesis at higher levels has been presented. M1, a KBS based on the methodology, can design single board computers from high level specifications using parts stored in a database. Experiments with M1 have demonstrated the methodology to be effective for utilizing and organizing design knowledge. M1 has been successfully integrated with an automated knowledge-acquisition tool CGEN. CGEN was used to capture knowledge for four micro-processor families

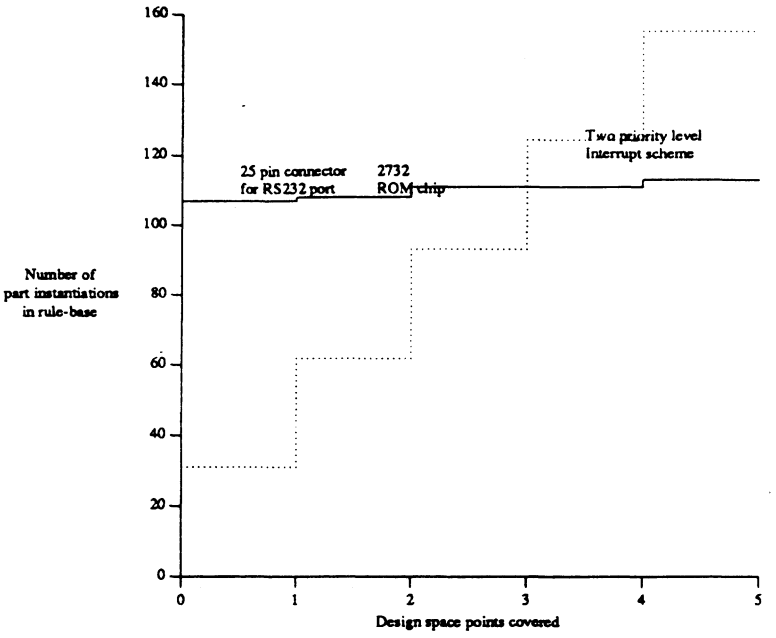

Figure 7: Experimental plot for growth in knowledge measured as parts indicating the effectiveness of the approach for supporting knowledge-acquisition.

M1's knowledge-base is presently being enhanced by using CGEN to teach it about new components and structures. M1 is being deployed as part of the MICON system to several IC and computer manufacturers to test its effectiveness in the real world.

7 Acknowledgements

Bill Birmingham contributed greatly to this work while he was developing CGEN, a program tightly coupled with M1.

References

[I] William P. Birmingham, Audrey Brennan, Anurag P. Gupta and Daniel P. Siewiorek.
MICON : A Single Board Computer Synthesis Tool.
IEEE Circuits and Devices Magazine, January, 1988.

[2] William P. Birmingham, Anurag P. Gupta and Daniel P. Siewiorek.
The MICON System for Computer Design.
In *Submitted to The 26th Design Automation Conference.* IEEE and ACM-SIGDA, IEEE Computer Society, 1989.

[3] William P. Birmingham and Daniel P. Siewiorek.
Capturing Designer Expertise - The CGEN System.
In *Submitted to The 26th Design Automation Conference.* IEEE and ACM-SIGDA, IEEE Computer Society, 1989.

[4] William P. Birmingham, Anurag P. Gupta and Daniel P. Siewiorek.
A General Synthesis Engine - Making MICON Domain-Independent.
In *Submitted to The 26th Design Automation Conference.* IEEE and ACM-SIGDA, IEEE Computer Society, 1989.

[5] Audrey A. Brennan.
Automatic Synthesis for Reliability.
Master's thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, Jan., 1988.

[6] Forrest D. Brewer and Daniel D. Gajski.
An Expert-System Paradigm for Design.
In *Proceedings of the 23rd Design Automation Conference,* pages 62-68. IEEE and ACM-SIGDA, 1986.

[7] David C. Brown and B. Chandrasekaran.
Knowledge and Control for a Mechanical Design Expert System.
Computer 19(7):92-100, July, 1986.

[8] Charles L. Forgy.
The OPS83 Report.
Technical Report CMU-CS-84-133, Department of Computer Science, Carnegie Mellon University, May, 1984.

[9] Daniel D. Gajski and Robert H. Kuhn.
Guest Editors' Introduction: New VLSI Tools.
Computer 16(12):11-14, December, 1983.

[10] Anurag P. Gupta.
A Hierarchical Problem Solving Architecture for Design Synthesis of Single Board Computers.
Master's thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, Feb., 1988.

[II] Ramesh Harjani, Rob A. Rutenbar and L. Richard Carley.
A Prototype Framework for Knowledge-Based Analog Circuit Synthesis.
In *Proceedings of the 24th Design Automation Conference.* IEEE and ACM-SIGDA, 1987

[12] M.L Maher and S.J. Fenves.
HI-RISE : An Expert System for the Preliminary Structural Design of High Rise Buildings.
In John S. Gero (editor), *Knowledge Engineering in Compouter-Aided Design,* pages 125-140 September, 1984.

[13] John McDermott.
R1: A Rule-Based Configurer of Computer Systems.
Artificial Intelligence 0(19):39-88,1982.

[14] Michael C. McFarland, Alice C. Parker and Raul Camposano.
Tutorial on High-Level Synthesis.
In *Proceedings of the 25th Design Automation Conference,* pages 330-336. IEEE and ACM-SIGDA, IEEE Computer Society, 1988.

[15] Tom M. Mitchell, Louis I. Steinberg and Jeffrey S. Shulman.
A Knowledge-Based Approach to Design.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-7(5):502-509, September, 1985.

[16] Noshir A. Langrana, Tom M. Mitchell and N. Ramachandran.
Progress Toward a Knowledge-Based Aid for Mechanical Design.
In *Symposium on Integrated and Intelligent Manufacturing.* ASME, 1986.

[17] MarkStefik.
Planning and Meta-Planning (MOLGEN: Part 2).
Artificial Intelligence (16):141-170,1981.

[18] Louis I. Steinberg.
Design as Refinement Plus Constraint Propagation: The VEXED Experience.
In *Proceedings of AAAI-87.* Morgan Kaufmann Publishers, 1987.

[19] Gerald Jay Sussman.
Electrical Design : A problem for Artificial Intelligence Research.
In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 894-900. 1980.

[20] Louise Trevillyan.
An Overview of Logic Synthesis Systems.
In *Proceedings of the 24th Design Automation Conference,* pages 166-172. IEEE and ACM-SIGDA, IEEE Computer Society, 1987.

[21] Robert A. Walker and Donald E. Thomas.
A Model of Design Representation and Synthesis.
In *Proceedings of the 22nd Design Automation Conference,* pages 453-459. IEEE and ACM-SIGDA, IEEE Computer Society, 1985.