IASys: An Integrated Approach to Sytem-Level Synthesis

Grace McNally, Janeen Deang and Daniel P. Siewiorek

EDRC 18-55-95

IASys: An Integrated Approach to System-Level Synthesis

Grace McNally, Janeen Deang, Dan Siewiorek

Departement of Electrical and Computer Engineering
and EDRC
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

October 5, 1995

This work has been supported in part by the Engineering Design Research Center, an NSF Engineering Research Center.

IASys: An Integrated Approach for System-Level Synthesis

Abstract

IASys is an integrated approach to system-level synthesis which allows a complete computer system to be specified using a set of high-level building blocks rather than behavioral level specifications. By reusing domain-specific design knowledge, alternative hardware and software configurations can be rapidly specified and synthesized. Thirty-two different embedded computer systems were synthesized generating a design space with variations of 45% in cost, 20% in power, 120% in area, 50% in program size, and 2000% in performance.

1 Introduction

The task of designing a complete computer system, which includes not only the application software, but the underlying operating system, hardware platform, as well as the interconnects between different components, has become increasingly challenging in recent years. To deal with the increased complexity and shortened design cycle, various synthesis systems have been developed to aid the synthesis of cost-effective systems. Automated synthesis attempts to map a set of input specifications to a hardware or software implementation. The specification can be at the system level, the behavioral level, or the implementation level. System level specification is the most abstract of these, which is a description that contains expert knowledge about the intended use of a system, its components, and their interconnections from input to output. This is followed by behavioral level specification, which details the intended behavior of a system, such as VHDL code for hardware and high-level programming languages for software. Implementation level specification is the least abstract and is concerned with the physical implementation of the final system, whether in logic components and circuits for hardware, or executable code for software. Depending on the level of specification, synthesis systems can be classified as performing system-level, behavioral level, or implementation level synthesis.

This paper presents IASys, which performs system-level synthesis of both hardware and software within a single environment. Through a combination of design reuse, abstraction, and integration

of design tools, IASys shortens the design cycle, reduces the complexity of the design task, and eases the creation of cost-effective systems. To specify a complete system the designer only needs to answer a series of queries from IASys regarding the functionalities and performance of the system. By incorporating low-level hardware and software design knowledge into its framework, IASys frees the system designer from having to generate the implementation details of the final system. Instead, the IASys user can concentrate on defining requirements for the system and exploring different design tradeoffs. Designs can be synthesized on the order of minutes which allows the system designer to rapidly explore the design space to select a cost effective design.

Section 2 summarizes existing approaches to hardware and software synthesis, followed by Section 3 which presents the IASys architecture, including the user specification format, knowledge representation, synthesis engine, and the code generation stage for software. The design space of a small computer system is presented in Section 4, and finally Section 5 concludes the paper with results from the thirty two different configurations of the computer system which were synthesized using IASys

2 Background

Design automation systems can be classified by the level of synthesis they perform, as shown in Table 1.

	Hardware Synthesis	Software Synthesis
System Level	• Configurers • System-Level Synthesis	• Application Generators • Program synthesizers
Behavioral Level	• Algorithmic-Level Synthesis	•VHLL
Implementation Level	• Logic-Level Synthesis	• Compilers • Assemblers

Table 1 Classification of synthesis systems

The higher the level of specification, the more intuitive and less detailed the user specification, and more of the design process is performed by the synthesis system, corresponding to a higher

degree of automation. However, high levels of automation is often accomplished by codifying domain-specific design knowledge, which unfortunately limits the *degree of generality* of the synthesis system. Often design automation tools represent a tradeoff between the degree of automation and degree of generality.

Systems which perform system-level synthesis correspond to the highest level of automation and are often based on sets of rules which codify expert design knowledge, but at the same time the type of systems they can synthesize is limited. Examples of such hardware synthesis systems include the Rl/XCON [7] computer configuration management system, and COSMOS [9], Fidelity [10], and MICON [8] which perform system level synthesis based on a set of rules about the structural and constraint information about components and interconnects. Program synthesizers[13],[14] and application generators[15] are examples of software synthesis systems which automate system level design, capable of generating a software program from a user specification which indicates the functions the resulting system has to perform, rather than a detailed code specification for the resulting system.

Synthesis tools which perform behavioral-level synthesis are less domain-specific, but requires the user to input a more detailed behavioral specification. Algorithmic-level hardware synthesis systems such as CAMAD [4], MIMOLA [5], and the System Architect's Workbench [6] transforms a behavioral level specifications written in a procedural language into a register-transfer level specification, which can then be implemented in hardware at a later stage. VHLL (very high-level language) systems such as SETL[12] and database query languages automates the behavioral specification stage for software programs. Though the process of writing code for behavioral-level tools is time-consuming, the tradeoff is that these tools can be used to generate a wide variety of applications and are not restricted to narrow application domains.

At the lowest level are systems which perform synthesis at the implementation level. Logic level synthesis tools discussed in [1], [2], and [3] assist the expert designer by automating the selection of components. Using one-to-one mapping of boolean equations to an interconnected gate network, these synthesis tools refine each specification to a component in the target technology. Assemblers and compilers automate the implementation of a software program from source code

to assembly code and represent the first generation of automatic programming systems documented in the ACM in 1958[11].

IASys is a unified approach to system level synthesis which combines both the properties of a program synthesizer and hardware system level synthesis tool. Though both hardware and software synthesis systems which synthesize from system level specifications exist, IASys represents the first system-level synthesis tool which can generate both hardware and software systems from system-level specification. Currently IASys has design knowledge for the domain of wearable computers, which are small, embedded systems that are used as mobile personal assistants for such tasks as document browsing and navigation. Because its knowledge representation is declarative, IASys can be easily tailored to other domains.

3 IASys Architecture

Figure 1 shows the IASys architecture, with a common synthesis engine FIDELITY [10] shared between the hardware and software synthesis process.

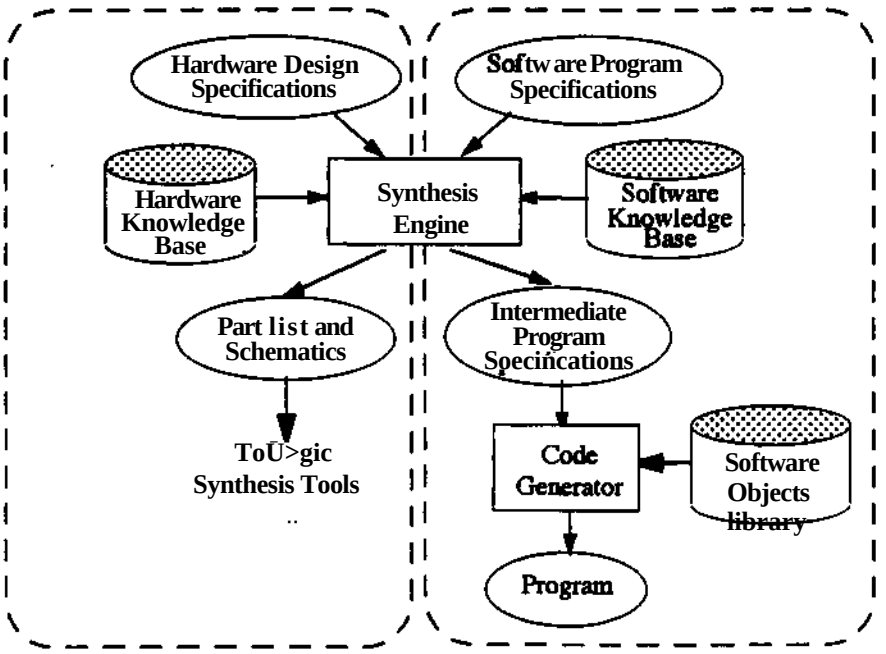


Figure 1: Overview of IASys Architecture

FIDELITY is based on MICON[8], a knowledge-based system which was designed to generate hardware. During design the synthesis engine alternates through a series of searches to select the appropriate hardware or software objects from its design libraries which make up the final system. To design hardware the IASys synthesis engine accesses domain-specific knowledge stored in its hardware knowledge base to guide the search and generate a partslist, netlist, and other requirements to be used with logic synthesis tools at a later stage. To design software, the synthesis engine accesses its software knowledge base to generate an intermediate program specification, which is used by the code generator to assemble the appropriate software modules from its software object library to produce the final software program.

The hardware synthesis process of IASys closely follows the MICON design paradigm, as detailed in [8]. The rest of this paper will highlight how the hardware design paradigms can be extended to software so that the single synthesis engine can be used to design both hardware and software. Table 2 lists by section the original hardware terminologies used in MICON, and the corresponding terminology adapted for software synthesis using IASys.

Sections	Hardware	Software
User Specification (Section 3.1)	• Design Specification	• Program Specification
Knowledge Representation (Section 3.2)	• Hardware Knowledge Base • Hardware Components • Hardware Design Templates • Wire Connections	• Software Knowledge Base • Software Objects • Software Design Templates • Object Messages
Design Synthesis (Section 3.3)	• Search for Function • Search for Structure	• Search for Function • Search for Structure
Implementation (Section 3.4)	• Logic Synthesis • Logic Design Library • Netlist and Schematics • Schematics	• Code Generation • Software Objects Library • Intermediate Program Specifications •Program Flow File

Table 2 Hardware and Software terms in IASys

A running example will be used in the following sections to illustrate the flow of each of the hardware and software design processes. For the hardware synthesis process, the knowledge base is assumed to contain information about various off-the-shelf microprocessors, and the appropriate processor meeting the specified speed requirements will be incorporated into a hardware design. For the software design process, an application will be synthesized which performs simple document browsing tasks. Figure 2 shows a series of frames which are displayed by the application during a typical browsing sequence.

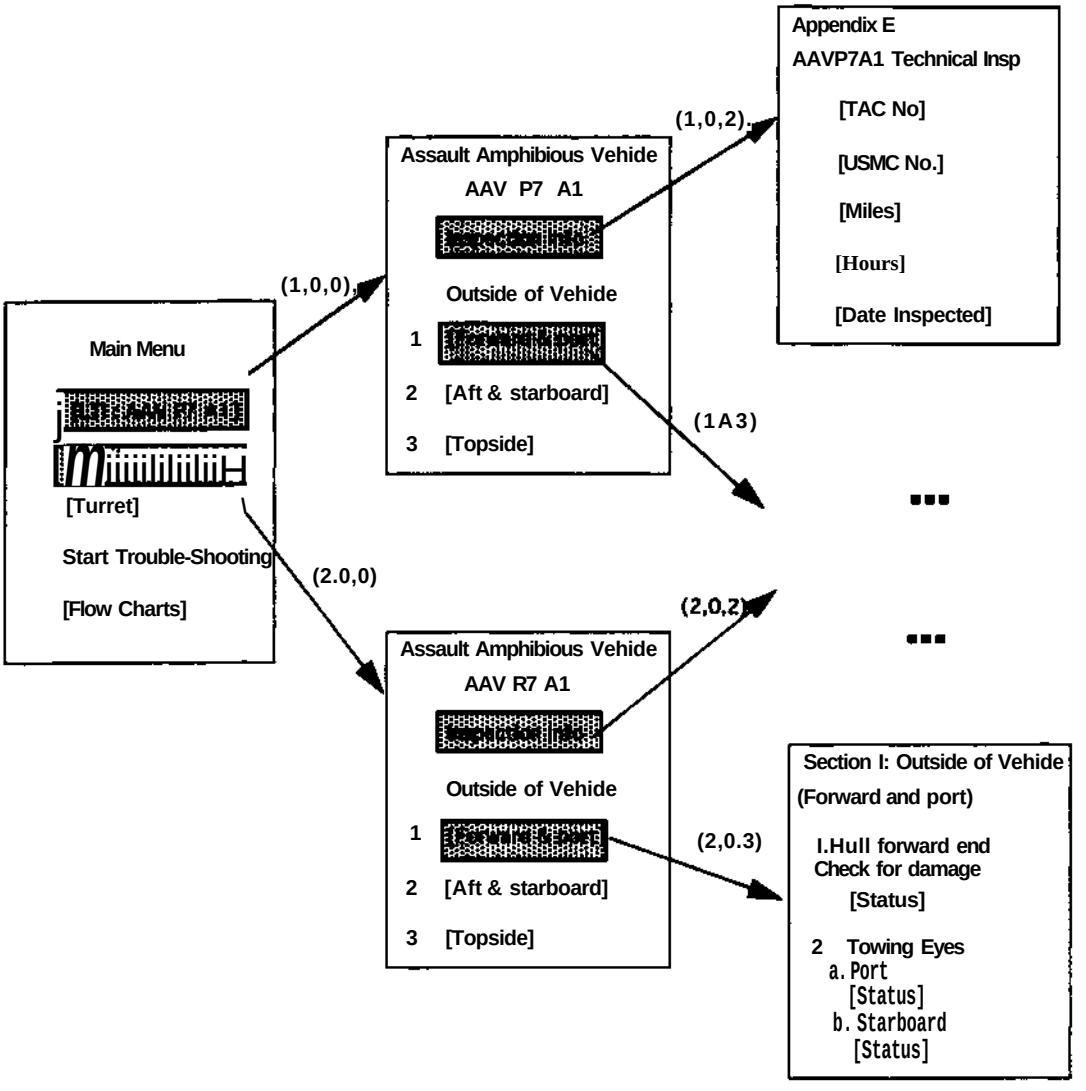


Figure 2 : Application User Interface

From the end-user's point of view, the document consists of a set of sections, which in turn consists of a set of pages, and these pages are further subdivided into a collection of frames. Within each frame are static text and active fields, which are the text enclosed in brackets in Figure 2. Each document subsection may be viewed one frame at a time, and the end-user interacts with the system by clicking on a mouse to select the available fields to navigate from frame to frame. In Figure 2 the end user has clicked on the highlighted fields and the frames which correspond to the Frame ID, which are links embedded in the fields are subsequently displayed.

To implement such a system, the system designer configures the document browser application by answering queries from the design synthesis engine. The software knowledge base is assumed to contain information about a set of data managers which retrieves text and bitmaps to display from a document database. The data managers are categorized in the software knowledge base by the way they access document information. Depending on whether they manage the fields, frames, or the multi-level tables which store the location to the frames to be displayed on the screen, the data managers are called FocusMgr, FrameMgr, or DocSysMgr (Document System Manager), respectively.

There are two kinds of DocSysMgr's which the system designer can choose from during design, depending on how extensive the document browsing system is to be. A DocTableDirMgr can handle more extensive document systems, using Frame ID's consisting of 3 indexes to access multi-level tables that store the frame locations, where a DocTableMgr use Frame ID's consisting of only 2 indexes to access location tables. The first entry in a 3-index Frame ID consists of a Section Number, followed by its Page Number, followed by its Frame Number, while a 2-index Frame ID omits the Section Number and can only be used to index frames located in the same section. Figure 2 displays a sequence of frames with 3-index Frame ID's, divided into sections 1 and 2 reachable from the Main Menu frame. Within each section the frames can be further broken down by their page numbers, though in this example no frames only 1 page is used. Within the same page frames can have different frame numbers. For example, Appendix E with Frame ID (1,0,2) in Figure 2 resides in Section 1, Page 0, Frame 2 of the document system. A design with DocTableDirMgr implemented would be able to handle all of the Frame ID's as shown in Figure 2, while a design with DocTableMgr can only handle Frame ID's with two lower indexes in the

Frame ID valid, fixing the section number in the Frame ID, and would only be able to reference the frames in either the top subtree with Frame ID's (l,x, x), or the bottom subtree with Frame ID's (2,x,x) in Figure 2.

3.1 User Specification

IASys acquires the user specification interactively through a series of queries. A sample input specification for the document browser application is shown in Figure 3.

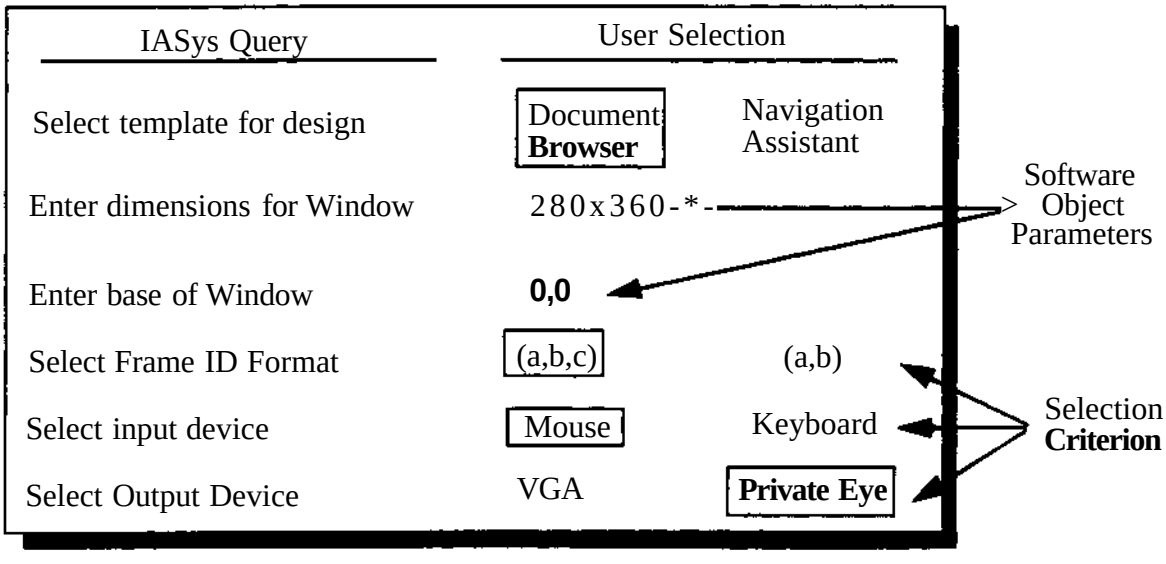


Figure 3 : IASys input specification

At the start of the specification session, the user is prompted to select a design template. In Figure 3 the user selects the Document Browser design template. The user is then prompted with increasingly detailed questions regarding the design, such as the size and location of the text window which will be used in the application, the format of the Frame ID, with (a,b,c) corresponding to a 3-index format, and (a,b) the 2-index format, and the type of input and output devices. IASys uses this information to refine the abstract design template into an implementation.

As Figure 3 shows, there are two types of user queries. *Selection criterions* such as the type of input and output devices are used by the synthesis engine to select the appropriate hardware com-

ponents or software objects which make up the final design, while *software object parameters* are used only by the IASys software synthesis process to set values for software object initialization parameters.

3.2 Knowledge Representation

IASys uses knowledge stored in its knowledge base to decide which and how to assemble the components for a particular software or hardware design. Each component is represented in the knowledge base as a set of facts, which specify the properties that describe the component, and a set of rules, which states how the facts about a component are to be used during design. Figure 4 shows the facts and rules representation in the IASys knowledge base for the Intel 188 13Mhz processor component on the left, and the software components DocTableDirMgr (Document Table Directory Manager) on the right, which is a specialization of the DocSysMgr.

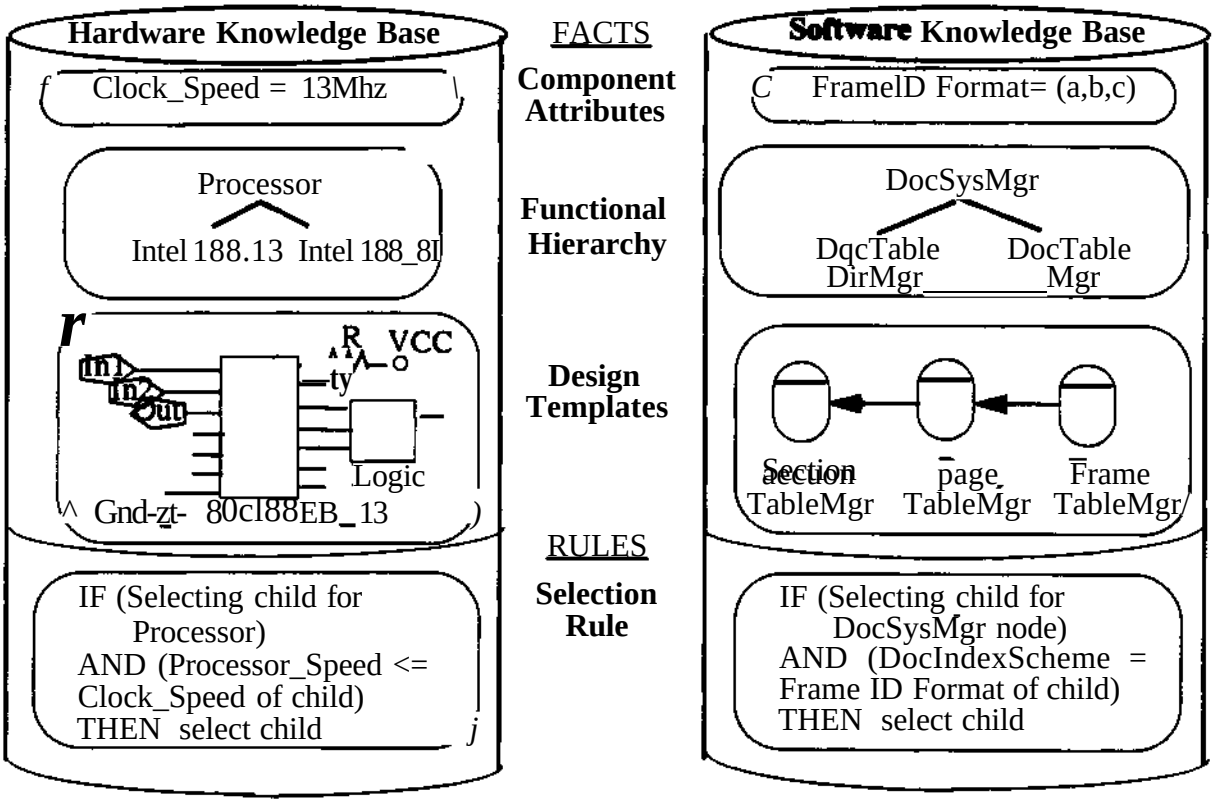


Figure 4 : IASys Knowledge Representation

As Figure 4 shows, the facts regarding a component consist of its *component attributes,* which are attribute-value pairs that contain the name of a particular attribute and its corresponding value, how it is organized in the *functional hierarchy,* and how it should be connected with other components as specified by *design templates.* A component's representation also include the *selection rules* which specify the conditions under which a component or a particular template should be selected, based on the facts about a component. The knowledge organization is discussed in detail in [8].

Functional hierarchies are central to reducing the size and complexity of the design space. As mentioned previously, components are grouped by function and organized into a directed acyclic graph (DAG) called the functional hierarchy. The nodes in the functional hierarchy represent hardware or software components which are abstractions of their leaf or child nodes and are characterized by their functions and constraints. Components which have similar functions are grouped into the same functional block, which correspond to higher level nodes. Actual hardware off-the-shelf parts or software objects are represented by nodes at the very bottom of the hierarchy. For example, in Figure 4, the Intel 188_13 and Intel 188_8 components are functionally equivalent and are grouped into the same processor functional block, while the software components DocTableDirMgr and DocTableMgr are grouped as DocSysMgr functional block.

The specification of how a functional block may be refined to its less abstract components, and how these components need to be connected with one another are stored in templates. Templates capture fragments of design information and allow hardware and software reuse at the design level. For hardware design templates the lines between components represent wire connections between hardware components, as shown by the hardware design template for the Intel 188 13Mhz processor in Figure 4. In IASys software components communicate with one another using a message-passing paradigm, and lines connecting the software components in a software design template correspond to messages being sent from one object to another. The software design template which corresponds to the refinement from DocSysMgr to DocTableDirMgr is shown in Figure 4, which specifies how the different document table mangers need to interact to process a document browsing task. Note that in this template there is a table manager corresponding to a particular index in a 3-index Frame ID. The FrameTableMgr object sends messages to the

PageTableMgr object, which in turn sends messages to the SectionTableMgr object.

3.3 Design Synthesis

During design the synthesis engine performs a top-down traversal of the functional hierarchy. At each node IASys attempts to select a child node which satisfies the requirements of the design, called search-for-function (SFF), and then integrates the child node into the design by selecting the appropriate template, called search-for-structure (SFS). By alternating between these two searches, IASys gradually refines abstract components into actual hardware or software objects to be used. Figure 5 shows one iteration through SFF and SFS for integrating an Intel 188 13Mhz processor into a hardware design on the left hand side, and for integrating the software objects composing the DocTableDirMgr functional block into a program on the right hand side.

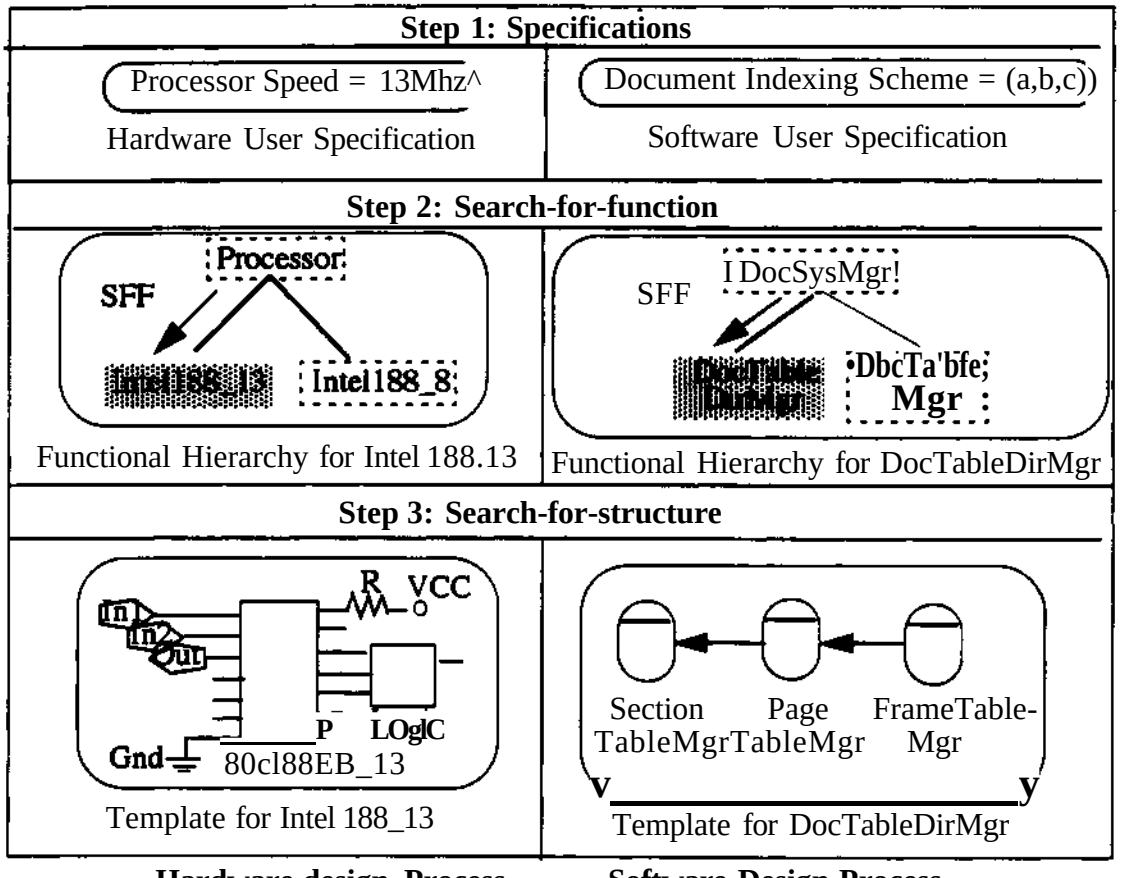


Figure 5 : Partial Design Synthesis Step

After a series of iterations, the synthesis engine generates a set of parts list and schematics, which can then be used as input into logic synthesis tools if designing hardware, or they can be used as a set of intermediate program files for the code-generation stage to generate a software program. Because the synthesis engine was designed to produce hardware, it interfaces with standard logic synthesis tools such as the Design Architect from Mentor Graphics at this stage to generate the final hardware implementation. However, to produce an executable software program, the output of the synthesis stage needs to be further processed by the code generation stage.

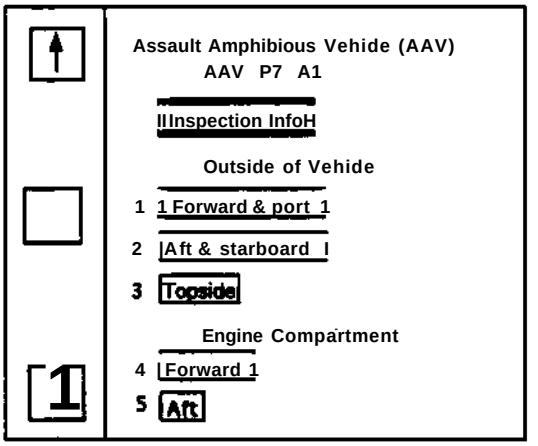
3.4 Code Generation Stage

The output of the design synthesis stage consists of a user specification file, and a set of schematics which the code generator uses as a program control flow diagram. The user specification file records all the user responses to queries during the IASys design stage, and the program control flow diagram specifies how the software objects interact with each other in terms of the messages sent to one another. The code generator produces code for the final program by accessing its library of software objects, as shown in Figure 1. Each software object in the code generator library consists of an object specification file, which the code generator fills in with parameter values as required by the design, and a code skeleton which uses the values from the object specification file. The code generator may tailor the software object to different designs by manipulating the object specification files, thereby reusing source code. To generate the final program, the code generator links in the information from the intermediate program specification provided by the synthesis stage to select the appropriate set of software objects from its library, and to fill in parameters values for these software objects.

4 Case Study

To gauge the effectiveness of IASys for rapidly synthesizing complete computer systems so that design tradeoffs can be easily evaluated, thirty-two different designs of a highly configurable embedded computer system were generated. The designs were based on VuMan 2[16], a small, embedded computer system which displays maps, pictures, or textual information to the user. The VuMan 2 hardware is shown on the left side of Figure 6, while the application interface is shown

on the right side. Designed for ease of use for the maintenance and repair worker, who often needs access to manuals and checklists, the entire VuMan 2 system is light enough to be worn on the belt



VuMan 2 Software User Interface

Figure 6 : VuMan 2 Computer System

The application which runs on VuMan 2 is similar to the document browsing program used in the running example, with a DocTableDirMgr implemented which can handle three levels of Frame ID indexes. The user inputs commands by clicking a button mouse on different parts of a frame which is viewed on a heads-up display. For flexibility VuMan 2 applications and document databases reside on removable PCMCIA flash memory cards so different applications can be easily swapped into the VuMan 2 system.

5 Results and Conclusions

As Figure 7 shows, several features on VuMan 2 can be varied to generate different system configurations.

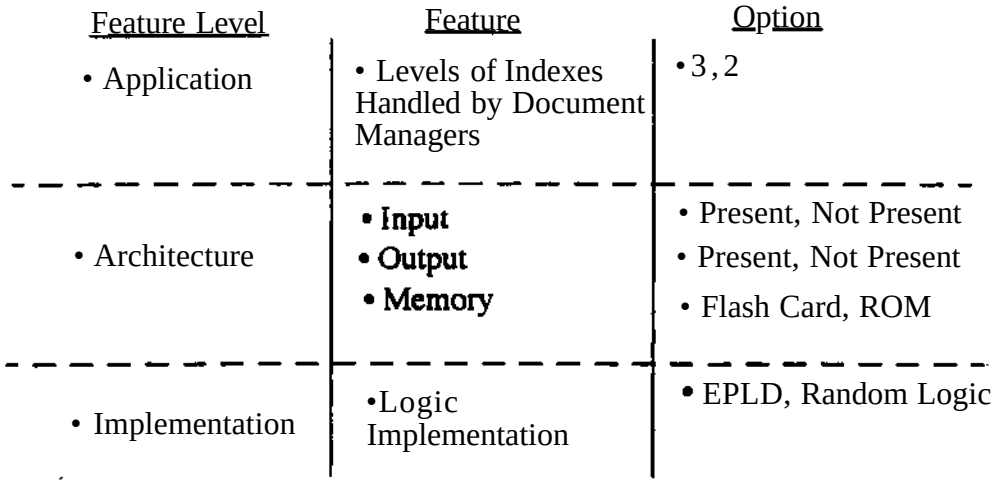


Figure 7 : VuMan 2 Configurable Features

These features span the spectrum from software to hardware, and can be grouped into application level, which is independent of the hardware platform, architecture level which impact both hardware and software, and implementation level which impact mostly hardware design parameters, and is transparent to software. At the application level the system can consist of document managers that handle two or three indexes for the Frame ID. At the architecture level the system can consist of an input device such as a mouse, an output device such as a heads-up display; and the document database may be placed on removable PCMCIA flash cards, or permanently installed ROM's. At the implementation level, the logic circuitry may be implemented in EPLD's (Erasable Programmable Logic Devices) or as random logic. To assess the impact of each of these features, thirty-two different systems were synthesized using IASys. The cost, power, and area information for the base hardware systems were generated. The synthesized software was ran on an Intel 486/33 PC based simulator to acquire program size and performance data. Both the hardware and software design information, as shown in Appendix A, were used to evaluate design tradeoffs. Each design is distinguished by the feature vector as shown in Figure 8[1].

1. Though designs without input or output devices are not fully functional, they are generated for this study to fully explore the design space.

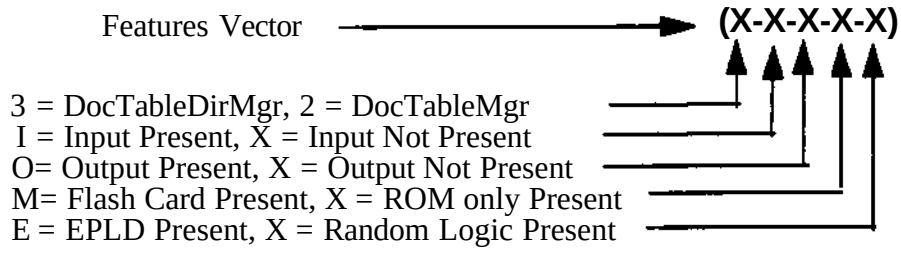


Figure 8 : Design Features Vector

To assess the impact of the system features on design parameters of cost, power, area, program size, and performance, the designs were partitioned according to different system features.The Manager Levels group partitioned designs into those which can handle only two Frame ID indexes, and those which can handle three; the Memory group partitioned designs into those which stored the document system on ROM's and those which stored them on a flash card, and the Input, Output, and EPLD groups partitioned the designs into those with and without input devices, with and without output devices, and those designed with EPLD or random logic, respectively. All of the designs were synthesized in under 10 minutes on a Sparc 10 Workstation. The average difference between the partitions for each group were then calculated for each design parameter, and converted to a percentage of the total difference resulting from varying all of the system features, as shown in Table 3. Cost, power, and area take into account the hardware portions of the system only, while program size and performance measurements are taken by running the programs on an Intel 486/33 PC-based simulator platform. The performance is based on the average number of clock ticks elapsed after running a typical user workload on the application.

	EPLD	Input	Output	Memory	Mgr Levels
Cost	34.38	3.47	0.99	61.16	0.00
Power	32.26	16.24	39.17	12.33	0.00
Area	16.28	8.08	8.32	67.32	0.00
Program Size	0.00	0.40	98.38	0.38	0.85
Perfor-mance	0.00	1.80	14.95	80.15	3.09

Table 3 : Relative Impact of Features on Design Parameters

Figure 9 shows the impact of each of these features on the embedded system in a bar graph, grouping features based on whether they are implementation, architecture, or application level features.

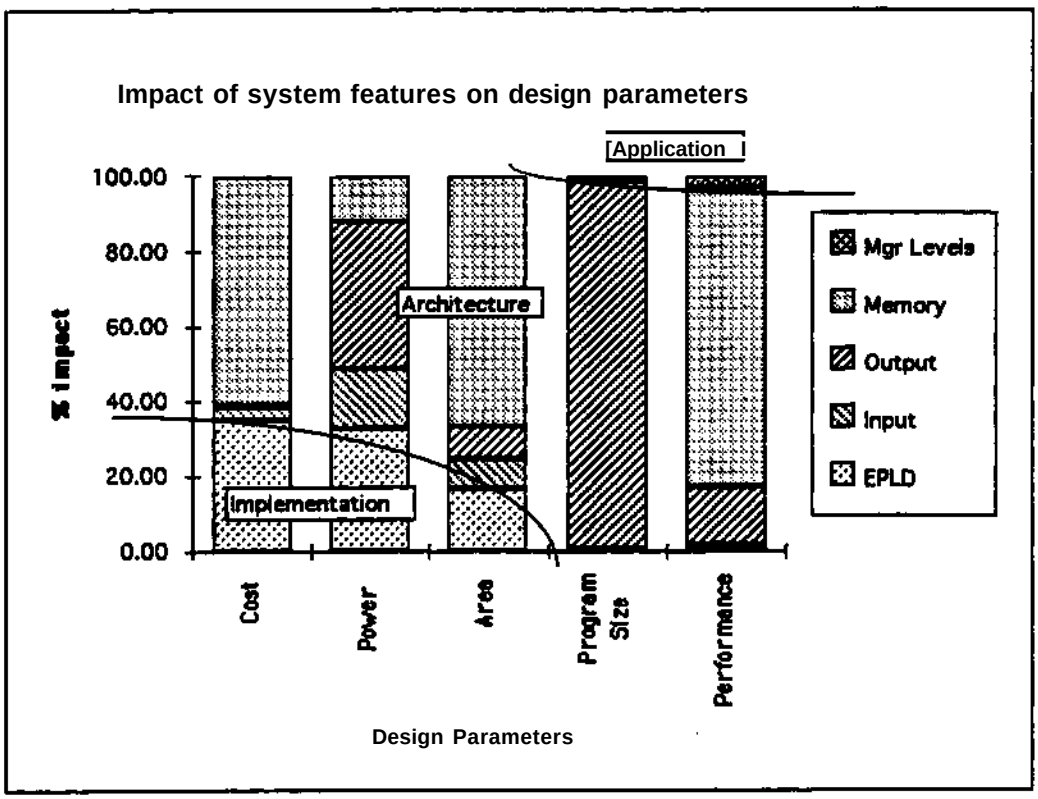


Figure 9 tlmpact of System Features

The implementation level feature impacted only the hardware design parameters, the architecture level features appear to have the most impact in both the hardware and software design parameters, dominated by memory and/or output options, while the application level feature only minimally impacted software design parameters, though they also impact the functionality of the system which is a parameter not evaluated in this study. The dominating system features for each design parameter implies the primary areas to focus design efforts for minimizing that particular design parameter. The cost, area, and performance design parameters are dominated by the memory configuration, impacted 61%, 67%, and 80%, respectively. In the case of using of flash memory the system cost was lowered by 20% to 30% and the area by 50% to 100%, but accounted for a factor of 5 to 15 in decreasing performance. Because the performance data was taken on a simulator platform, the effects of whether the logic circuitry was implemented using and EPLD instead

IASys: An Integrated Approach to Sytem-Level Synthesis

Grace McNally, Janeen Deang and Daniel P. Siewiorek

EDRC 18-55-95

IASys: An Integrated Approach to System-Level Synthesis

Grace McNally, Janeen Deang, Dan Siewiorek

Departement of Electrical and Computer Engineering
and EDRC
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

October 5, 1995

This work has been supported in part by the Engineering Design Research Center, an NSF Engineering Research Center.

IASys: An Integrated Approach for System-Level Synthesis

Abstract

IASys is an integrated approach to system-level synthesis which allows a complete computer system to be specified using a set of high-level building blocks rather than behavioral level specifications. By reusing domain-specific design knowledge, alternative hardware and software configurations can be rapidly specified and synthesized. Thirty-two different embedded computer systems were synthesized generating a design space with variations of 45% in cost, 20% in power, 120% in area, 50% in program size, and 2000% in performance.

1 Introduction

The task of designing a complete computer system, which includes not only the application software, but the underlying operating system, hardware platform, as well as the interconnects between different components, has become increasingly challenging in recent years. To deal with the increased complexity and shortened design cycle, various synthesis systems have been developed to aid the synthesis of cost-effective systems. Automated synthesis attempts to map a set of input specifications to a hardware or software implementation. The specification can be at the system level, the behavioral level, or the implementation level. System level specification is the most abstract of these, which is a description that contains expert knowledge about the intended use of a system, its components, and their interconnections from input to output. This is followed by behavioral level specification, which details the intended behavior of a system, such as VHDL code for hardware and high-level programming languages for software. Implementation level specification is the least abstract and is concerned with the physical implementation of the final system, whether in logic components and circuits for hardware, or executable code for software. Depending on the level of specification, synthesis systems can be classified as performing system-level, behavioral level, or implementation level synthesis.

This paper presents IASys, which performs system-level synthesis of both hardware and software within a single environment. Through a combination of design reuse, abstraction, and integration

of design tools, IASys shortens the design cycle, reduces the complexity of the design task, and eases the creation of cost-effective systems. To specify a complete system the designer only needs to answer a series of queries from IASys regarding the functionalities and performance of the system. By incorporating low-level hardware and software design knowledge into its framework, IASys frees the system designer from having to generate the implementation details of the final system. Instead, the IASys user can concentrate on defining requirements for the system and exploring different design tradeoffs. Designs can be synthesized on the order of minutes which allows the system designer to rapidly explore the design space to select a cost effective design.

Section 2 summarizes existing approaches to hardware and software synthesis, followed by Section 3 which presents the IASys architecture, including the user specification format, knowledge representation, synthesis engine, and the code generation stage for software. The design space of a small computer system is presented in Section 4, and finally Section 5 concludes the paper with results from the thirty two different configurations of the computer system which were synthesized using IASys

2 Background

Design automation systems can be classified by the level of synthesis they perform, as shown in Table 1.

	Hardware Synthesis	Software Synthesis
System Level	• Configurers • System-Level Synthesis	• Application Generators • Program synthesizers
Behavioral Level	• Algorithmic-Level Synthesis	•VHLL
Implementation Level	• Logic-Level Synthesis	• Compilers • Assemblers

Table 1 Classification of synthesis systems

The higher the level of specification, the more intuitive and less detailed the user specification, and more of the design process is performed by the synthesis system, corresponding to a higher

degree of automation. However, high levels of automation is often accomplished by codifying domain-specific design knowledge, which unfortunately limits the *degree of generality* of the synthesis system. Often design automation tools represent a tradeoff between the degree of automation and degree of generality.

Systems which perform system-level synthesis correspond to the highest level of automation and are often based on sets of rules which codify expert design knowledge, but at the same time the type of systems they can synthesize is limited. Examples of such hardware synthesis systems include the Rl/XCON [7] computer configuration management system, and COSMOS [9], Fidelity [10], and MICON [8] which perform system level synthesis based on a set of rules about the structural and constraint information about components and interconnects. Program synthesizers[13],[14] and application generators[15] are examples of software synthesis systems which automate system level design, capable of generating a software program from a user specification which indicates the functions the resulting system has to perform, rather than a detailed code specification for the resulting system.

Synthesis tools which perform behavioral-level synthesis are less domain-specific, but requires the user to input a more detailed behavioral specification. Algorithmic-level hardware synthesis systems such as CAMAD [4], MIMOLA [5], and the System Architect's Workbench [6] transforms a behavioral level specifications written in a procedural language into a register-transfer level specification, which can then be implemented in hardware at a later stage. VHLL (very high-level language) systems such as SETL[12] and database query languages automates the behavioral specification stage for software programs. Though the process of writing code for behavioral-level tools is time-consuming, the tradeoff is that these tools can be used to generate a wide variety of applications and are not restricted to narrow application domains.

At the lowest level are systems which perform synthesis at the implementation level. Logic level synthesis tools discussed in [1], [2], and [3] assist the expert designer by automating the selection of components. Using one-to-one mapping of boolean equations to an interconnected gate network, these synthesis tools refine each specification to a component in the target technology. Assemblers and compilers automate the implementation of a software program from source code

to assembly code and represent the first generation of automatic programming systems documented in the ACM in 1958[11].

IASys is a unified approach to system level synthesis which combines both the properties of a program synthesizer and hardware system level synthesis tool. Though both hardware and software synthesis systems which synthesize from system level specifications exist, IASys represents the first system-level synthesis tool which can generate both hardware and software systems from system-level specification. Currently IASys has design knowledge for the domain of wearable computers, which are small, embedded systems that are used as mobile personal assistants for such tasks as document browsing and navigation. Because its knowledge representation is declarative, IASys can be easily tailored to other domains.

3 IASys Architecture

Figure 1 shows the IASys architecture, with a common synthesis engine FIDELITY [10] shared between the hardware and software synthesis process.

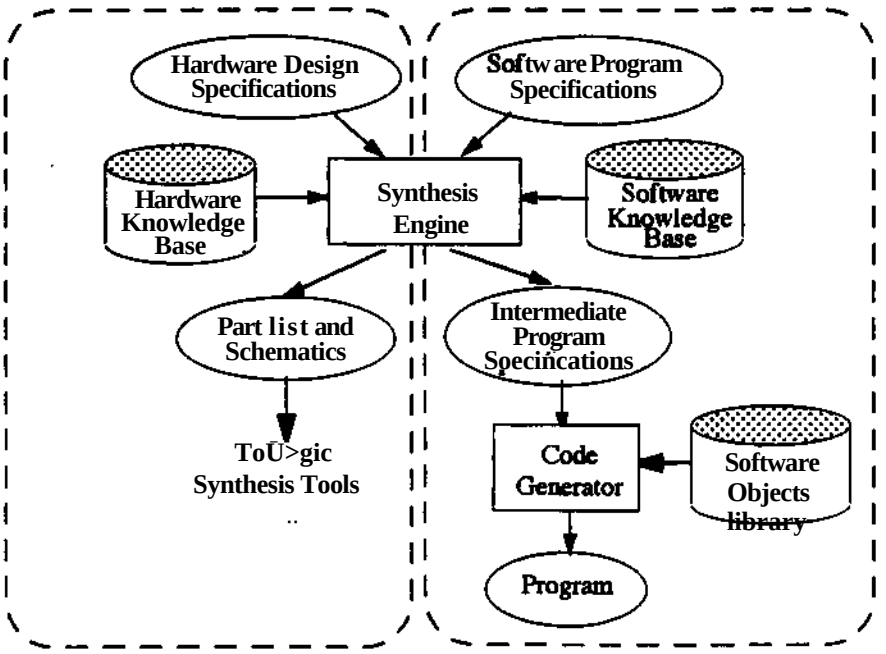


Figure 1: Overview of IASys Architecture

FIDELITY is based on MICON[8], a knowledge-based system which was designed to generate hardware. During design the synthesis engine alternates through a series of searches to select the appropriate hardware or software objects from its design libraries which make up the final system. To design hardware the IASys synthesis engine accesses domain-specific knowledge stored in its hardware knowledge base to guide the search and generate a partslist, netlist, and other requirements to be used with logic synthesis tools at a later stage. To design software, the synthesis engine accesses its software knowledge base to generate an intermediate program specification, which is used by the code generator to assemble the appropriate software modules from its software object library to produce the final software program.

The hardware synthesis process of IASys closely follows the MICON design paradigm, as detailed in [8]. The rest of this paper will highlight how the hardware design paradigms can be extended to software so that the single synthesis engine can be used to design both hardware and software. Table 2 lists by section the original hardware terminologies used in MICON, and the corresponding terminology adapted for software synthesis using IASys.

Sections	Hardware	Software
User Specification (Section 3.1)	• Design Specification	• Program Specification
Knowledge Representation (Section 3.2)	• Hardware Knowledge Base • Hardware Components • Hardware Design Templates • Wire Connections	• Software Knowledge Base • Software Objects • Software Design Templates • Object Messages
Design Synthesis (Section 3.3)	• Search for Function • Search for Structure	• Search for Function • Search for Structure
Implementation (Section 3.4)	• Logic Synthesis • Logic Design Library • Netlist and Schematics • Schematics	• Code Generation • Software Objects Library • Intermediate Program Specifications •Program Flow File

Table 2 Hardware and Software terms in IASys

A running example will be used in the following sections to illustrate the flow of each of the hardware and software design processes. For the hardware synthesis process, the knowledge base is assumed to contain information about various off-the-shelf microprocessors, and the appropriate processor meeting the specified speed requirements will be incorporated into a hardware design. For the software design process, an application will be synthesized which performs simple document browsing tasks. Figure 2 shows a series of frames which are displayed by the application during a typical browsing sequence.

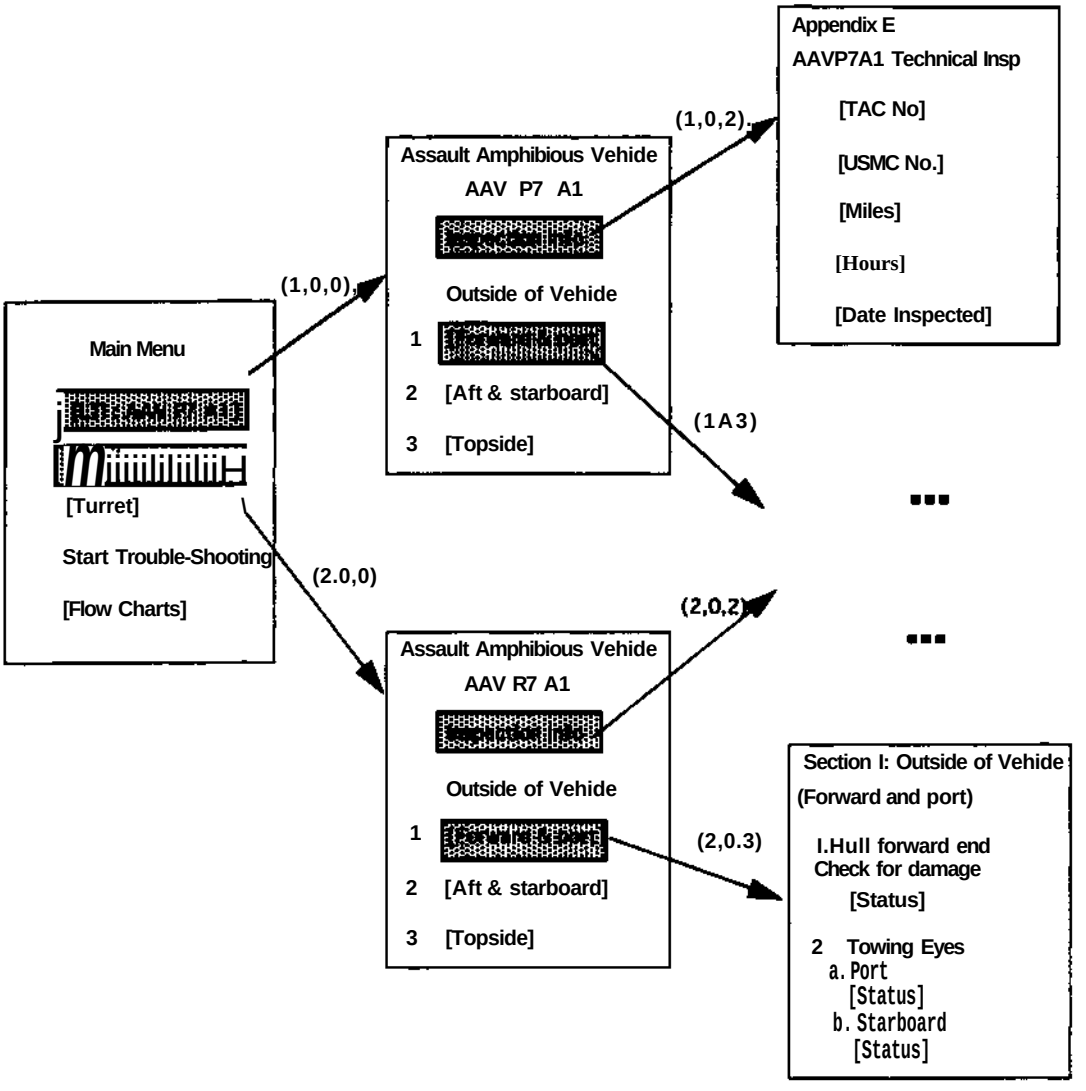


Figure 2 : Application User Interface

From the end-user's point of view, the document consists of a set of sections, which in turn consists of a set of pages, and these pages are further subdivided into a collection of frames. Within each frame are static text and active fields, which are the text enclosed in brackets in Figure 2. Each document subsection may be viewed one frame at a time, and the end-user interacts with the system by clicking on a mouse to select the available fields to navigate from frame to frame. In Figure 2 the end user has clicked on the highlighted fields and the frames which correspond to the Frame ID, which are links embedded in the fields are subsequently displayed.

To implement such a system, the system designer configures the document browser application by answering queries from the design synthesis engine. The software knowledge base is assumed to contain information about a set of data managers which retrieves text and bitmaps to display from a document database. The data managers are categorized in the software knowledge base by the way they access document information. Depending on whether they manage the fields, frames, or the multi-level tables which store the location to the frames to be displayed on the screen, the data managers are called FocusMgr, FrameMgr, or DocSysMgr (Document System Manager), respectively.

There are two kinds of DocSysMgr's which the system designer can choose from during design, depending on how extensive the document browsing system is to be. A DocTableDirMgr can handle more extensive document systems, using Frame ID's consisting of 3 indexes to access multi-level tables that store the frame locations, where a DocTableMgr use Frame ID's consisting of only 2 indexes to access location tables. The first entry in a 3-index Frame ID consists of a Section Number, followed by its Page Number, followed by its Frame Number, while a 2-index Frame ID omits the Section Number and can only be used to index frames located in the same section. Figure 2 displays a sequence of frames with 3-index Frame ID's, divided into sections 1 and 2 reachable from the Main Menu frame. Within each section the frames can be further broken down by their page numbers, though in this example no frames only 1 page is used. Within the same page frames can have different frame numbers. For example, Appendix E with Frame ID (1,0,2) in Figure 2 resides in Section 1, Page 0, Frame 2 of the document system. A design with DocTableDirMgr implemented would be able to handle all of the Frame ID's as shown in Figure 2, while a design with DocTableMgr can only handle Frame ID's with two lower indexes in the

Frame ID valid, fixing the section number in the Frame ID, and would only be able to reference the frames in either the top subtree with Frame ID's (l,x, x), or the bottom subtree with Frame ID's (2,x,x) in Figure 2.

3.1 User Specification

IASys acquires the user specification interactively through a series of queries. A sample input specification for the document browser application is shown in Figure 3.

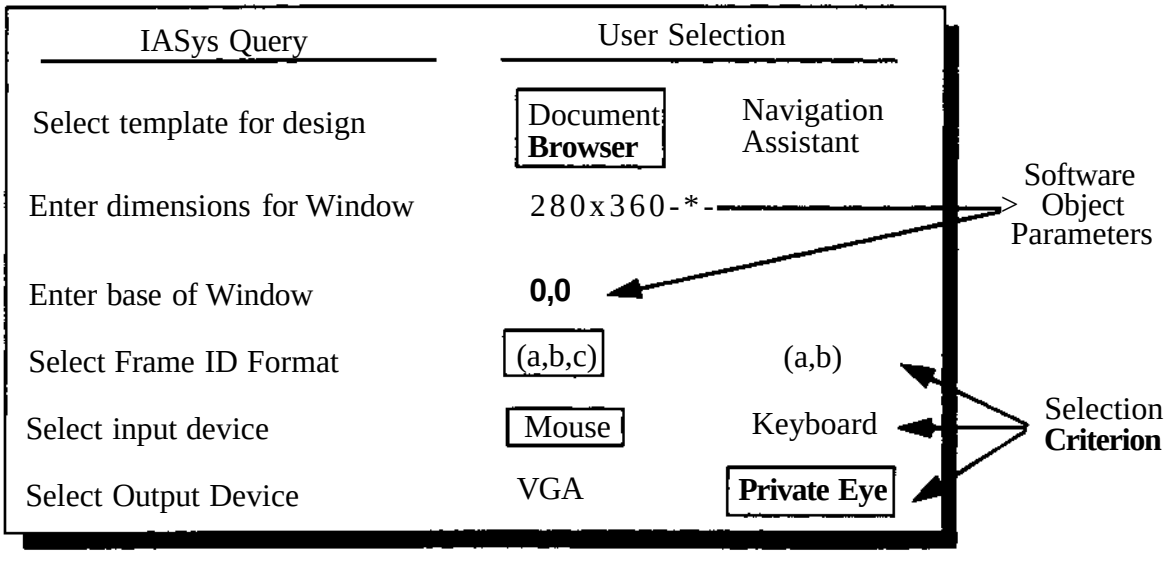


Figure 3 : IASys input specification

At the start of the specification session, the user is prompted to select a design template. In Figure 3 the user selects the Document Browser design template. The user is then prompted with increasingly detailed questions regarding the design, such as the size and location of the text window which will be used in the application, the format of the Frame ID, with (a,b,c) corresponding to a 3-index format, and (a,b) the 2-index format, and the type of input and output devices. IASys uses this information to refine the abstract design template into an implementation.

As Figure 3 shows, there are two types of user queries. *Selection criterions* such as the type of input and output devices are used by the synthesis engine to select the appropriate hardware com-

ponents or software objects which make up the final design, while *software object parameters* are used only by the IASys software synthesis process to set values for software object initialization parameters.

3.2 Knowledge Representation

IASys uses knowledge stored in its knowledge base to decide which and how to assemble the components for a particular software or hardware design. Each component is represented in the knowledge base as a set of facts, which specify the properties that describe the component, and a set of rules, which states how the facts about a component are to be used during design. Figure 4 shows the facts and rules representation in the IASys knowledge base for the Intel 188 13Mhz processor component on the left, and the software components DocTableDirMgr (Document Table Directory Manager) on the right, which is a specialization of the DocSysMgr.

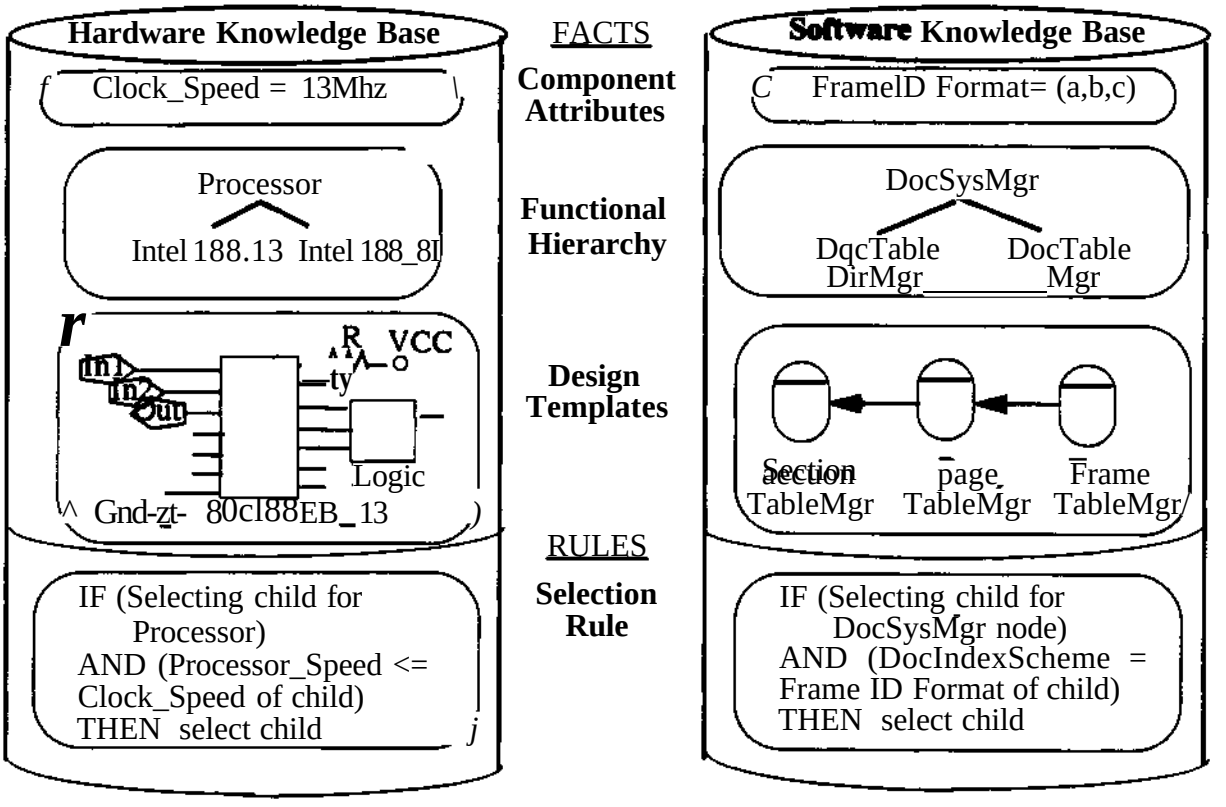


Figure 4 : IASys Knowledge Representation

As Figure 4 shows, the facts regarding a component consist of its *component attributes,* which are attribute-value pairs that contain the name of a particular attribute and its corresponding value, how it is organized in the *functional hierarchy,* and how it should be connected with other components as specified by *design templates.* A component's representation also include the *selection rules* which specify the conditions under which a component or a particular template should be selected, based on the facts about a component. The knowledge organization is discussed in detail in [8].

Functional hierarchies are central to reducing the size and complexity of the design space. As mentioned previously, components are grouped by function and organized into a directed acyclic graph (DAG) called the functional hierarchy. The nodes in the functional hierarchy represent hardware or software components which are abstractions of their leaf or child nodes and are characterized by their functions and constraints. Components which have similar functions are grouped into the same functional block, which correspond to higher level nodes. Actual hardware off-the-shelf parts or software objects are represented by nodes at the very bottom of the hierarchy. For example, in Figure 4, the Intel 188_13 and Intel 188_8 components are functionally equivalent and are grouped into the same processor functional block, while the software components DocTableDirMgr and DocTableMgr are grouped as DocSysMgr functional block.

The specification of how a functional block may be refined to its less abstract components, and how these components need to be connected with one another are stored in templates. Templates capture fragments of design information and allow hardware and software reuse at the design level. For hardware design templates the lines between components represent wire connections between hardware components, as shown by the hardware design template for the Intel 188 13Mhz processor in Figure 4. In IASys software components communicate with one another using a message-passing paradigm, and lines connecting the software components in a software design template correspond to messages being sent from one object to another. The software design template which corresponds to the refinement from DocSysMgr to DocTableDirMgr is shown in Figure 4, which specifies how the different document table mangers need to interact to process a document browsing task. Note that in this template there is a table manager corresponding to a particular index in a 3-index Frame ID. The FrameTableMgr object sends messages to the

PageTableMgr object, which in turn sends messages to the SectionTableMgr object.

3.3 Design Synthesis

During design the synthesis engine performs a top-down traversal of the functional hierarchy. At each node IASys attempts to select a child node which satisfies the requirements of the design, called search-for-function (SFF), and then integrates the child node into the design by selecting the appropriate template, called search-for-structure (SFS). By alternating between these two searches, IASys gradually refines abstract components into actual hardware or software objects to be used. Figure 5 shows one iteration through SFF and SFS for integrating an Intel 188 13Mhz processor into a hardware design on the left hand side, and for integrating the software objects composing the DocTableDirMgr functional block into a program on the right hand side.

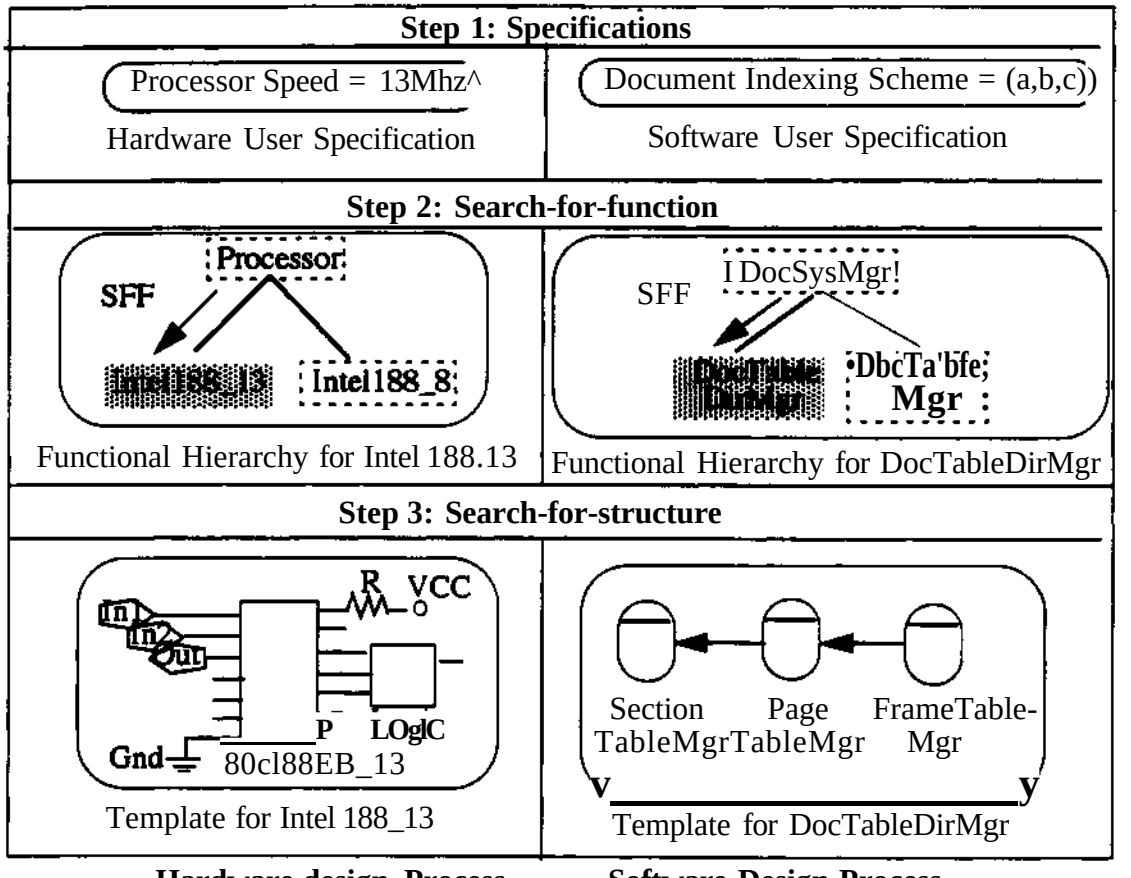


Figure 5 : Partial Design Synthesis Step

After a series of iterations, the synthesis engine generates a set of parts list and schematics, which can then be used as input into logic synthesis tools if designing hardware, or they can be used as a set of intermediate program files for the code-generation stage to generate a software program. Because the synthesis engine was designed to produce hardware, it interfaces with standard logic synthesis tools such as the Design Architect from Mentor Graphics at this stage to generate the final hardware implementation. However, to produce an executable software program, the output of the synthesis stage needs to be further processed by the code generation stage.

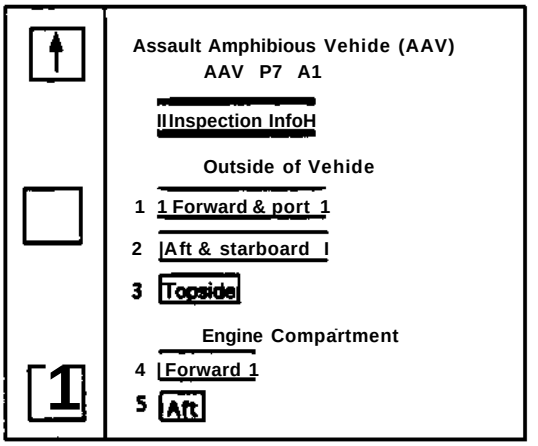
3.4 Code Generation Stage

The output of the design synthesis stage consists of a user specification file, and a set of schematics which the code generator uses as a program control flow diagram. The user specification file records all the user responses to queries during the IASys design stage, and the program control flow diagram specifies how the software objects interact with each other in terms of the messages sent to one another. The code generator produces code for the final program by accessing its library of software objects, as shown in Figure 1. Each software object in the code generator library consists of an object specification file, which the code generator fills in with parameter values as required by the design, and a code skeleton which uses the values from the object specification file. The code generator may tailor the software object to different designs by manipulating the object specification files, thereby reusing source code. To generate the final program, the code generator links in the information from the intermediate program specification provided by the synthesis stage to select the appropriate set of software objects from its library, and to fill in parameters values for these software objects.

4 Case Study

To gauge the effectiveness of IASys for rapidly synthesizing complete computer systems so that design tradeoffs can be easily evaluated, thirty-two different designs of a highly configurable embedded computer system were generated. The designs were based on VuMan 2[16], a small, embedded computer system which displays maps, pictures, or textual information to the user. The VuMan 2 hardware is shown on the left side of Figure 6, while the application interface is shown

on the right side. Designed for ease of use for the maintenance and repair worker, who often needs access to manuals and checklists, the entire VuMan 2 system is light enough to be worn on the belt



VuMan 2 Software User Interface

Figure 6 : VuMan 2 Computer System

The application which runs on VuMan 2 is similar to the document browsing program used in the running example, with a DocTableDirMgr implemented which can handle three levels of Frame ID indexes. The user inputs commands by clicking a button mouse on different parts of a frame which is viewed on a heads-up display. For flexibility VuMan 2 applications and document databases reside on removable PCMCIA flash memory cards so different applications can be easily swapped into the VuMan 2 system.

5 Results and Conclusions

As Figure 7 shows, several features on VuMan 2 can be varied to generate different system configurations.

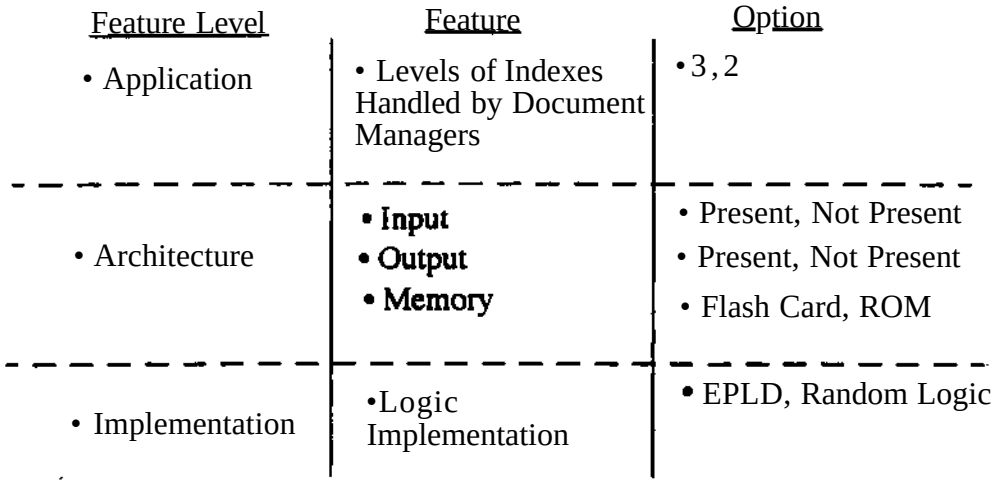


Figure 7 : VuMan 2 Configurable Features

These features span the spectrum from software to hardware, and can be grouped into application level, which is independent of the hardware platform, architecture level which impact both hardware and software, and implementation level which impact mostly hardware design parameters, and is transparent to software. At the application level the system can consist of document managers that handle two or three indexes for the Frame ID. At the architecture level the system can consist of an input device such as a mouse, an output device such as a heads-up display; and the document database may be placed on removable PCMCIA flash cards, or permanently installed ROM's. At the implementation level, the logic circuitry may be implemented in EPLD's (Erasable Programmable Logic Devices) or as random logic. To assess the impact of each of these features, thirty-two different systems were synthesized using IASys. The cost, power, and area information for the base hardware systems were generated. The synthesized software was ran on an Intel 486/33 PC based simulator to acquire program size and performance data. Both the hardware and software design information, as shown in Appendix A, were used to evaluate design tradeoffs. Each design is distinguished by the feature vector as shown in Figure 8[1].

1. Though designs without input or output devices are not fully functional, they are generated for this study to fully explore the design space.

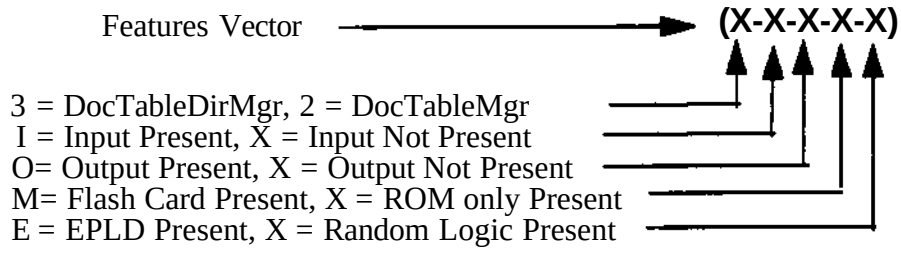


Figure 8 : Design Features Vector

To assess the impact of the system features on design parameters of cost, power, area, program size, and performance, the designs were partitioned according to different system features.The Manager Levels group partitioned designs into those which can handle only two Frame ID indexes, and those which can handle three; the Memory group partitioned designs into those which stored the document system on ROM's and those which stored them on a flash card, and the Input, Output, and EPLD groups partitioned the designs into those with and without input devices, with and without output devices, and those designed with EPLD or random logic, respectively. All of the designs were synthesized in under 10 minutes on a Sparc 10 Workstation. The average difference between the partitions for each group were then calculated for each design parameter, and converted to a percentage of the total difference resulting from varying all of the system features, as shown in Table 3. Cost, power, and area take into account the hardware portions of the system only, while program size and performance measurements are taken by running the programs on an Intel 486/33 PC-based simulator platform. The performance is based on the average number of clock ticks elapsed after running a typical user workload on the application.

	EPLD	Input	Output	Memory	Mgr Levels
Cost	34.38	3.47	0.99	61.16	0.00
Power	32.26	16.24	39.17	12.33	0.00
Area	16.28	8.08	8.32	67.32	0.00
Program Size	0.00	0.40	98.38	0.38	0.85
Perfor-mance	0.00	1.80	14.95	80.15	3.09

Table 3 : Relative Impact of Features on Design Parameters

Figure 9 shows the impact of each of these features on the embedded system in a bar graph, grouping features based on whether they are implementation, architecture, or application level features.

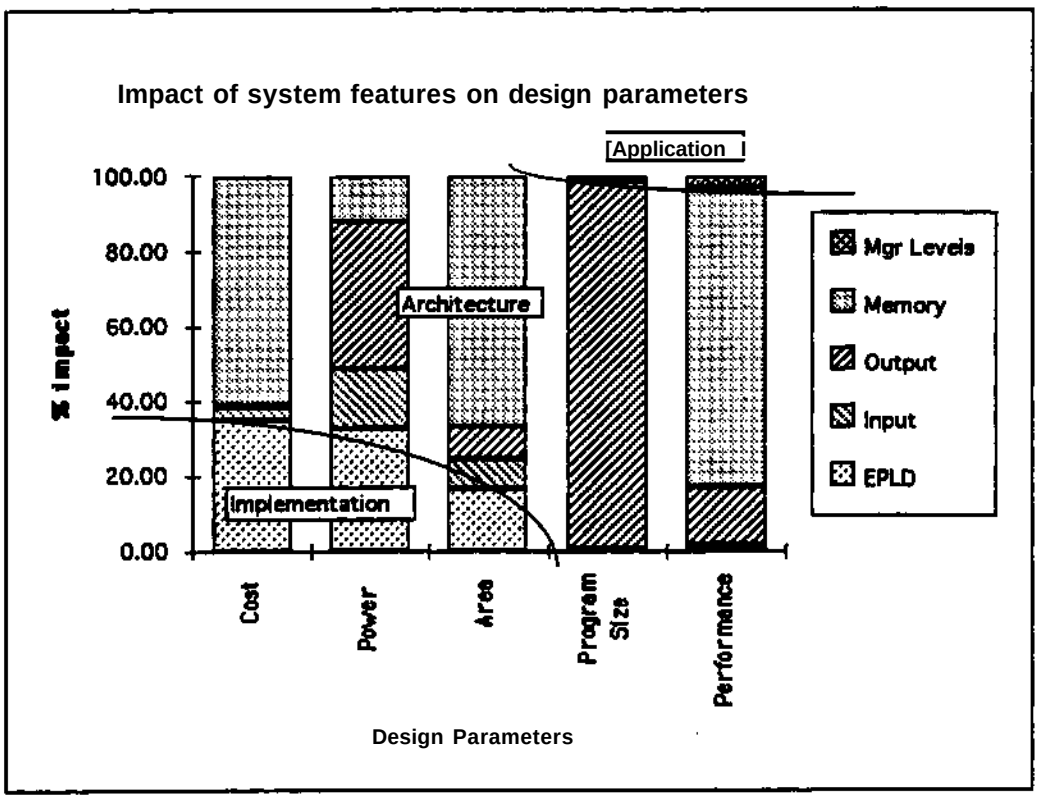


Figure 9 tlmpact of System Features

The implementation level feature impacted only the hardware design parameters, the architecture level features appear to have the most impact in both the hardware and software design parameters, dominated by memory and/or output options, while the application level feature only minimally impacted software design parameters, though they also impact the functionality of the system which is a parameter not evaluated in this study. The dominating system features for each design parameter implies the primary areas to focus design efforts for minimizing that particular design parameter. The cost, area, and performance design parameters are dominated by the memory configuration, impacted 61%, 67%, and 80%, respectively. In the case of using of flash memory the system cost was lowered by 20% to 30% and the area by 50% to 100%, but accounted for a factor of 5 to 15 in decreasing performance. Because the performance data was taken on a simulator platform, the effects of whether the logic circuitry was implemented using and EPLD instead

of random logic cannot be determined[1]. The impacts on power consumption are equally shared between output and logic implementation features, at 39% and 32%, respectively. Thus to minimize power consumption the system designer should carefully choose the output device and minimize the amount of random logic gates. The impact on program size is dominated over 98% by the software output device drivers. Therefore the system designer may consider implementing graphics routines using hardware.

6 Acknowledgments

The research reported in this paper is supported by the ARPA/Tri-Series-sponsored Rapid Prototyping of Application-Specific Signal Processors (RASSP) program, as executed by the Martin Marietta Corporation.

1. Because all the logic circuitry is used as glue logic between the CPU and other devices, the performance is independent of the logic circuit implementation as long as the delay of the logic circuitry does not exceed the CPU clock at 13Mhz.

7 References

[I] D. Brand, R.F Damiano, L.P. van Ginneken, and A.D. Drumm. In the Driver's Seat of Boole-Dozer. In *Proc. of 1994 IEEE International Conference on Computer Design,* Oct. 1994, pp 518-521.

[2] B. Mitra, L. Ramachnadran, S.Rajam, and G. Rajagopalas. CLSS- A Workfiench for Control Logic Synthesis. In *Proc. of 4th CSI/IEEE International Symposium on VLSI Design,* Jan. 1991, pp 219-224.

[3] H. Sato, M. Yamazaki, and M. Fugita. ZEPHCAD and FLORA: Logic Synthesis for Control and Datapath. In *Proc. of 1994 IEEE International Conference on Computer Design,* Oct. 1994, pp 527-530.

[4] Z. Peng and K. Kuchcinski. Automated transformations of algorithms into register-transfer level implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 13, No. 2, Feb. 1994, pp 150-166.

[5] N. Hendrich, J. Lohse, and R. Rauscher. Silicon Compilation and Rapid Prototyping of Microprogrammed VLSI-Circuits with MIMOLA and SOLO 1400. 18th EUROMICRO Symposium on Microprocessing and Microprogramming, *Microprocessing and Microprogramming,* Vol. 35, No. 1-5, Sept. 1992, pp 287-294.

[6] D. E. Thomas, et. al. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench.* Kluwer Academic Publishers, 1990.

[7] J. McDermott. Rl fXCON') at age 12: Lessons from an elementary school achiever. *Artificial Intelligence,* Vol. 59, No. 1-2, Feb. 1993, pp 241-247.

[8] W.P. Birmingham, A.P. Gupta, and D.P. Siewiorek. *Automating the Design of Computer Systems The MCON Project,* Jones and Bartlett Publishers, 1992.

[9] T.B. Ismail,M. Abid, and A. Jerraya. COSMOS: a codesign approach for communication systems. In *Proc. of the 3rd International Workshop on Hardware! Software Codesign,* Sept. 1994, pp 17-24.

[10] The FIDELITY Design Synthesis System User's Manual. Version 0.9 (alpha). Omniview, Inc., Pittsburgh, PA. Jan 1994.

[II] Rich and Waters. "Automatic Programming: Myths and Prospects." In *IEEE Computer,*

August 1988. pp. 4-51.

[12] J.T. Schwartz et al., *Programming with Sets: An Introduction to SETL,* Springer Verlag, New York, 1986.

[13] Charles Rich and Richard C. Waters. *Artificial Intelligence and Software Engineering.* Morgan Kaufmann Publishers, Inc. CA, 1986.

[14] Michael R. Lowry and Robert D. McCartney. *Automating Software Design.* AAAI Press. Menlo Park, CA, 1991.

[15] Ellis Horowitz et al., "A Survey of Application Generators." In IEEE Software, Jan. 1985. pp. 40-53.

[16] A. Smailagic, D.P. Siewiorek. 'The VuMan 2 Wearable Computer". *IEEE Design and Test of Computers,* September 1993.

Appendix A

				IASys Designs Generated				
	2-x-x-x-x	3-x-x-x-x	2-i-x-x-x	3-i-x-x-x	2-x-o-x-x	3-x-o-x-x	2-i-o-x-x	3-i-o-x-x
Cost($)	882.28	882.28	901.99	901.99	887.73	887.73	906.14	906.14
Power(W)	11.61	11.61	12.29	12.29	12.89	12.89	13.58	13.58
Area (in2)	45.26	45.26	50.12	50.12	50.11	50.11	54.88	54.88
Program Size(Bytes)	37212.00	37356.00	37116.00	37276.00	55500.00	55660.00	55436.00	55596.00
Performance(Clocks)	5.00	5.00	5.00	5.00	17.00	18.00	19.00	19.00
	2-x-x-m-x	3-x-x-m-x	2-i-x-m-x	3-i-x-m-x	2-x-o-m-x	3-x-o-m-x	2-i-o-m-x	3-i-o-m-x
Cost($)	692.68	692.68	712.39	712.39	698.13	698.13	716.54	716.54
Power(W)	12.03	12.03	12.72	12.72	13.31	13.31	14.00	14.00
Area (in2)	26.14	26.14	30.99	30.99	30.99	30.99	35.76	35.76
Program Size(Bytes)	37260.00	37420.00	37196.00	37356.00	55580.00	55740.00	55500.00	55660.00
Performance(Clocks)	80.00	86.00	78.00	82.00	96.00	100.00	92.00	101.00
	2-x-x-x-e	3-x-x-x-e	2-i-x-x-e	3-i-x-x-e	2-x-o-x-e	3-x-o-x-e	2-i-o-x-e	3-i-o-x-e
Cost($)	999.78	999.78	1002.88	1002.88	1001.71	1001.71	1003.81	1003.81
Power(W)	12.11	12.11	12.11	12.11	12.48	12.48	12.48	12.48
Area (in2)	46.59	46.59	46.79	46.79	46.94	46.94	46.94	46.94
Program Size(Bytes)	37212.00	37356.00	37116.00	37276.00	55500.00	55660.00	55436.00	55596.00
Performance(Clocks)	5.00	5.00	5.00	5.00	17.00	18.00	19.00	19.00
	2-x-x-m-e	3-x-x-x-e	2-i-x-m-e	3-i-x-m-e	2-x-o-m-e	3-x-o-m-e	2-i-o-m-e	3-i-o-m-e
Cost($)	809.08	809.08	812.18	812.18	811.01	811.01	812.81	812.81
Power(W)	12.01	12.01	12.01	12.01	12.38	12.38	12.38	12.38
Area (in2)	24.79	24.79	24.99	24.99	25.14	25.14	25.14	25.14
Program Size(Bytes)	37260.00	37420.00	37196.00	37356.00	55580.00	55740.00	55500.00	55660.00
Performance(Clocks)	80.00	86.00	78.00	82.00	96.00	100.00	92.00	101.00

7 References

[I] D. Brand, R.F Damiano, L.P. van Ginneken, and A.D. Drumm. In the Driver's Seat of Boole-Dozer. In *Proc. of 1994 IEEE International Conference on Computer Design,* Oct. 1994, pp 518-521.

[2] B. Mitra, L. Ramachnadran, S.Rajam, and G. Rajagopalas. CLSS- A Workfiench for Control Logic Synthesis. In *Proc. of 4th CSI/IEEE International Symposium on VLSI Design,* Jan. 1991, pp 219-224.

[3] H. Sato, M. Yamazaki, and M. Fugita. ZEPHCAD and FLORA: Logic Synthesis for Control and Datapath. In *Proc. of 1994 IEEE International Conference on Computer Design,* Oct. 1994, pp 527-530.

[4] Z. Peng and K. Kuchcinski. Automated transformations of algorithms into register-transfer level implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 13, No. 2, Feb. 1994, pp 150-166.

[5] N. Hendrich, J. Lohse, and R. Rauscher. Silicon Compilation and Rapid Prototyping of Microprogrammed VLSI-Circuits with MIMOLA and SOLO 1400. 18th EUROMICRO Symposium on Microprocessing and Microprogramming, *Microprocessing and Microprogramming,* Vol. 35, No. 1-5, Sept. 1992, pp 287-294.

[6] D. E. Thomas, et. al. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench.* Kluwer Academic Publishers, 1990.

[7] J. McDermott. Rl fXCON') at age 12: Lessons from an elementary school achiever. *Artificial Intelligence,* Vol. 59, No. 1-2, Feb. 1993, pp 241-247.

[8] W.P. Birmingham, A.P. Gupta, and D.P. Siewiorek. *Automating the Design of Computer Systems The MCON Project,* Jones and Bartlett Publishers, 1992.

[9] T.B. Ismail,M. Abid, and A. Jerraya. COSMOS: a codesign approach for communication systems. In *Proc. of the 3rd International Workshop on Hardware! Software Codesign,* Sept. 1994, pp 17-24.

[10] The FIDELITY Design Synthesis System User's Manual. Version 0.9 (alpha). Omniview, Inc., Pittsburgh, PA. Jan 1994.

[II] Rich and Waters. "Automatic Programming: Myths and Prospects." In *IEEE Computer,*

August 1988. pp. 4-51.

[12] J.T. Schwartz et al., *Programming with Sets: An Introduction to SETL,* Springer Verlag, New York, 1986.

[13] Charles Rich and Richard C. Waters. *Artificial Intelligence and Software Engineering.* Morgan Kaufmann Publishers, Inc. CA, 1986.

[14] Michael R. Lowry and Robert D. McCartney. *Automating Software Design.* AAAI Press. Menlo Park, CA, 1991.

[15] Ellis Horowitz et al., "A Survey of Application Generators." In IEEE Software, Jan. 1985. pp. 40-53.

[16] A. Smailagic, D.P. Siewiorek. 'The VuMan 2 Wearable Computer". *IEEE Design and Test of Computers,* September 1993.

Appendix A

				IASys Designs Generated				
	2-x-x-x-x	3-x-x-x-x	2-i-x-x-x	3-i-x-x-x	2-x-o-x-x	3-x-o-x-x	2-i-o-x-x	3-i-o-x-x
Cost($)	882.28	882.28	901.99	901.99	887.73	887.73	906.14	906.14
Power(W)	11.61	11.61	12.29	12.29	12.89	12.89	13.58	13.58
Area (in2)	45.26	45.26	50.12	50.12	50.11	50.11	54.88	54.88
Program Size(Bytes)	37212.00	37356.00	37116.00	37276.00	55500.00	55660.00	55436.00	55596.00
Performance(Clocks)	5.00	5.00	5.00	5.00	17.00	18.00	19.00	19.00
	2-x-x-m-x	3-x-x-m-x	2-i-x-m-x	3-i-x-m-x	2-x-o-m-x	3-x-o-m-x	2-i-o-m-x	3-i-o-m-x
Cost($)	692.68	692.68	712.39	712.39	698.13	698.13	716.54	716.54
Power(W)	12.03	12.03	12.72	12.72	13.31	13.31	14.00	14.00
Area (in2)	26.14	26.14	30.99	30.99	30.99	30.99	35.76	35.76
Program Size(Bytes)	37260.00	37420.00	37196.00	37356.00	55580.00	55740.00	55500.00	55660.00
Performance(Clocks)	80.00	86.00	78.00	82.00	96.00	100.00	92.00	101.00
	2-x-x-x-e	3-x-x-x-e	2-i-x-x-e	3-i-x-x-e	2-x-o-x-e	3-x-o-x-e	2-i-o-x-e	3-i-o-x-e
Cost($)	999.78	999.78	1002.88	1002.88	1001.71	1001.71	1003.81	1003.81
Power(W)	12.11	12.11	12.11	12.11	12.48	12.48	12.48	12.48
Area (in2)	46.59	46.59	46.79	46.79	46.94	46.94	46.94	46.94
Program Size(Bytes)	37212.00	37356.00	37116.00	37276.00	55500.00	55660.00	55436.00	55596.00
Performance(Clocks)	5.00	5.00	5.00	5.00	17.00	18.00	19.00	19.00
	2-x-x-m-e	3-x-x-x-e	2-i-x-m-e	3-i-x-m-e	2-x-o-m-e	3-x-o-m-e	2-i-o-m-e	3-i-o-m-e
Cost($)	809.08	809.08	812.18	812.18	811.01	811.01	812.81	812.81
Power(W)	12.01	12.01	12.01	12.01	12.38	12.38	12.38	12.38
Area (in2)	24.79	24.79	24.99	24.99	25.14	25.14	25.14	25.14
Program Size(Bytes)	37260.00	37420.00	37196.00	37356.00	55580.00	55740.00	55500.00	55660.00
Performance(Clocks)	80.00	86.00	78.00	82.00	96.00	100.00	92.00	101.00