LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

FUNCHARD-An Expert System for Translating Functional Specifications to Hardware Structures

Computation Structures Group Memo 231
October 1983

Willie Y-P Lim
Thomas S. Wanuga

This research was supported by the Department of Energy under grant no. DE-AC02-79ER10473, that National Science Foundation under grant no. MCS-7915255, and the National Science Foundation Fellowship Program.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

FUNCHARD – An Expert System for Translating Functional Specifications to Hardware Structures

Willie Y-P. Lim
Thomas S. Wanuga

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139.

Abstract

A rule-based expert system for generating a network of hardware modules that meet some high level system specification is proposed as a component of a VLSI design system called FUNCHARD. The high level system specification is in the form of a data flow graph. Rules are used to determine what the underlying architecture should be, what modules should be used, how they are to be controlled, and how they are to be connected. All the necessary secondary hardware modules that are needed to support the operation of the main hardware modules are generated through the activation of more rules. The output of FUNCHARD is a hardware structure that implements the system specification.

1. Introduction

The design of a VLSI chip transcends many levels. At the top is the conceptual level where the designer thinks of the chip in terms of its high level functionality. The view at this level is generally vague, informal and too imprecise to be directly used by a practical computer aided design system for generating VLSI chips. At the other extreme, is the integrated circuit mask of the chip. The amount of details required for the mask is enormous, especially for VLSI and wafer scale integration. It is impractical for human beings to generate the masks manually. Such a task is well suited for the computer. As software technology progresses, the tasks delegated either partially or totally to the computer is moving closer and closer to the top level. We have seen the progression of VLSI tools from the layout level to higher level tools like silicon compilers [9] and the Design Automation Assistant [3]. In this paper we propose a higher level tool, that translates high level

---

1. This research was supported by the Department of Energy under grant number DE-AC02-79ER10473, the National Science Foundation under grant number MCS-7915255, and the National Science Foundation Fellowship Program.

FUNCHARD – An Expert System for Translating Functional Specifications to Hardware Structures

Willie Y-P. Lim
Thomas S. Wanuga

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139.

Abstract

A rule-based expert system for generating a network of hardware modules that meet some high level system specification is proposed as a component of a VLSI design system called FUNCHARD. The high level system specification is in the form of a data flow graph. Rules are used to determine what the underlying architecture should be, what modules should be used, how they are to be controlled, and how they are to be connected. All the necessary secondary hardware modules that are needed to support the operation of the main hardware modules are generated through the activation of more rules. The output of FUNCHARD is a hardware structure that implements the system specification.

1. Introduction

The design of a VLSI chip transcends many levels. At the top is the conceptual level where the designer thinks of the chip in terms of its high level functionality. The view at this level is generally vague, informal and too imprecise to be directly used by a practical computer aided design system for generating VLSI chips. At the other extreme, is the integrated circuit mask of the chip. The amount of details required for the mask is enormous, especially for VLSI and wafer scale integration. It is impractical for human beings to generate the masks manually. Such a task is well suited for the computer. As software technology progresses, the tasks delegated either partially or totally to the computer is moving closer and closer to the top level. We have seen the progression of VLSI tools from the layout level to higher level tools like silicon compilers [9] and the Design Automation Assistant [3]. In this paper we propose a higher level tool, that translates high level

---

1. This research was supported by the Department of Energy under grant number DE-AC02-79ER10473, the National Science Foundation under grant number MCS-7915255, and the National Science Foundation Fellowship Program.

functional specifications to hardware structures.

For a large system, the process of converting a high level specification to its hardware equivalent is very tedious. It is desirable to have an automatic or semi-automatic system for doing part if not all of this work. Some work has been done in this area [1, 2, 11, 12]. Most of this work has been based on analytical methods. The knowledge for translating the initial specification to hardware structures is built into these system. We feel such systems are too inflexible to be useful. Their hardware building blocks have specifications that state how they are built and how they are to be used. These specifications can basically be thought of as rules describing how the blocks perform their functions and how they should be used. Each new building block has to be accompanied by a new set of such rules. This is very much the current practice. The functionality specifications in the manufacturer's data books are examples of such rules. These specifications are important in that they are used for matching a conceptual design specification to an interconnection of building blocks, be these obtained "off the shelf" or through custom design. Hence if this type of knowledge is built into the system, the addition of a new building block will be difficult. However if the system is built as a rule based expert system, then it can also be built with the capability for adding new rules and hence support the addition of new hardware building blocks. Such a system can thus keep up with changes in technology.

FUNCHARD is a rule-based expert system that generates hardware structures from high level behavioral specifications. The behavioral specification is entered in a high level hardware specification, while the hardware structure generated is a network of hardware modules plus some additional control information. FUNCHARD is a tool that allows the hardware engineer to explore various hardware structures that implements a behavioral specification. If necessary, the hardware designer can improve on the FUNCHARD output. This process is much faster than the manual one, allowing many more iterations of the design cycle.

The role played by FUNCHARD in a VLSI CAD system is described in Section 2. We then discuss in Section 3, how hardware structures are generated by FUNCHARD. This is followed by an example in Section 4 and the conclusions of the paper in Section 5.

2. System Overview of FUNCHARD

The structure of a full-blown FUNCHARD system is shown in Figure 1. The system (input to FUNCHARD) to be designed is specified in a high level hardware specification language. A high level translator is used to generate a data flow graph from the specification. With the graph as input to the hardware structure generator, a hardware structure is obtained. The hardware structure generator can be extended to include new rules for selecting the design methodology to be used. Since a design methodology is a discipline, it can be specified as a set of rules. The methodologies supported can range from the totally

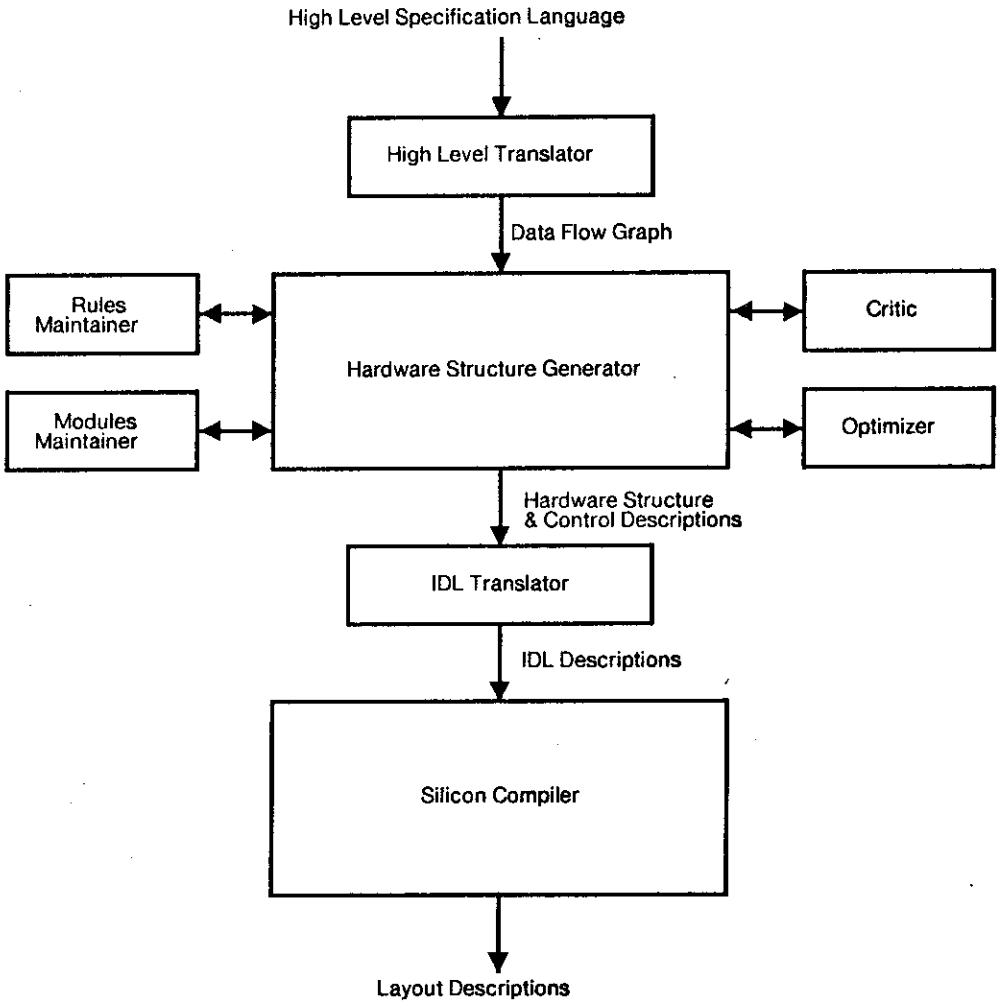

Figure 1. The FUNCHARD VLSI CAD System

asynchronous discipline to the totally synchronous discipline. The particular methodology selected will influence the choice of hardware modules and how they are to be connected.

For FUNCHARD to be practical, it has to generate reasonable hardware structures. It thus has to have some knowledge of how to trade off between performance and the cost of hardware. To minimize the hardware cost, some cost evaluation function must be provided. Associated with each module, some cost index must be given to reflect the relative cost of the modules. For example, one would expect an ALU to have a higher cost than say a tri-state buffer. Hence the cost minimization process is not one of counting the number of modules but rather the minimization of the total cost of the whole hardware structure including the support modules and interconnections.

In determining the speed of a system, each module has to be given some performance index that represents its speed. The overall speed of the system is based on how the modules are connected and how

many modules can operate concurrently. FUNCHARD should be provided with the knowledge for detecting concurrency, as well as maximizing and minimizing it within a system through reorganization of the system hardware structure. It also needs to know when hardware modules, especially support modules, can be shared among the seed modules without affecting the degree of parallelism that already exists.

It is unreasonable to expect FUNCHARD to be a one pass system with high level specifications entering at one end and a practical hardware structure coming out at the other. FUNCHARD will have to go through many iterations before arriving at a reasonable hardware structure. A critic and an optimizer are needed for this. The critic can be a human being or a program that examines the intermediate FUNCHARD output and provides criticisms of the output. It can also provide suggestions on where improvements can be made. Similarly the optimizer can be a human being or a program that is frequently called upon by FUNCHARD to optimize the intermediate FUNCHARD output.

To cope with a changing technology, some mechanism must be provided for updating the rules and modules used by FUNCHARD. Thus a rules maintainer and a modules maintainer are needed. When a module is added, the modules maintainer should check to see that it does not already exist. It should check to see what operations it can perform and whether it can be used as a support module; updating the knowledge base accordingly. Checks should be made to see that whatever support modules that are invoked by the newly added module are in the knowledge base. If not, the support modules should be added too. In the case of module deletion, the module maintainer should check that the module is not used as a support module by any other module. Addition of new modules may include the addition of new rules that specify how the modules are to be used. Thus the modules maintainer and rules maintainer may need to work together when the knowledge base is being updated.

The output of FUNCHARD contains a hardware structure and its control specification. Using an Intermediate Design Language (IDL) translator, the FUNCHARD output can be converted to an IDL description. This description in turn becomes an input to a silicon compiler which converts it to a chip layout description. The silicon compiler to be used is by itself a complex system. It might very well be another rule-based system like Palladio [10].

The system shown in Figure 1 can further be augmented with facilities for generating descriptions for human consumption. The hardware structure generator may generate summary descriptions that will present to the human an accurate and yet sufficiently high level view of the hardware structure being generated. We feel that the work of Patil [5] will be useful here. The use of "landmarks" seem particularly apt for this purpose. The main seed modules can be used as landmarks for transcending the multi-level description of the hardware structure. Thus one may start by giving a view in terms of the main seed modules and eventually focus on the subnetwork that is centered around a particular seed module. In the first view the support

modules are not mentioned, but in the second view the seed module as well as all the support modules in the subnetwork have to be described.

3. Hardware Structure Generation

The complete FUNCHARD system described above is a large project. For this paper, we limit our discussion to the hardware structure generator subset of FUNCHARD. The input to the hardware structure generator is a data flow graph composed of operators (i.e. the nodes) and the connecting arcs through which data flows. The operators perform simple operations like addition, subtraction, and multiplication. Hardware modules that perform these operations are termed *operation* modules. Other kinds of modules are needed to support the operation modules in performing their functions. These modules are termed *support* modules and include clocks, tri-state buffers, multiplexors, and demultiplexors.

There are two kinds of support modules – invoked and inferred support modules. Invoked support modules are those secondary modules that are directly invoked when an operation module is used. For example when an ALU is used, an additional module that performs some control function may have to be invoked to generate the operation codes for the ALU. The knowledge on which support module to invoke is kept in a frame [4] for the operation module. Inferred support modules, on the other hand, are not automatically invoked when an operation module is used, but rather are inferred from the structure and functionality of the network of modules that is being generated. For example, if more than a pair of inputs has to be presented to a two input operation module, the need for one or more multiplexor modules arises. Thus the multiplexor module is an inferred support module for this case. Control modules that are needed to generate various control signals like the select signals for the multiplexors are also inferred support modules.

The process of generating the hardware structure that meets some high level functional specification can be thought of as a plan generation process. Sacerdoti's work [6, 7] can be extended in the following manner to cover this domain. Some rules are used to map the data flow operators to operation modules. These may be termed architecture selection rules which provide biases in the choice of the modules. Hence through the execution of the rules, a set of operation modules are obtained to form the core of the hardware structure to be generated. These modules are basically "seeds" from which the interconnection structure grows. As the support modules are put in place, subnetworks that are centered around these seed modules are formed. Starting with the top level goal of connecting the seeds, FUNCHARD focuses on each seed and uses it as a subgoal. It checks to see if there are any support modules that are needed. If so, each support modules that is needed becomes a subgoal and the process is repeated. When all the invoked support modules are in place, FUNCHARD then looks for inferred support modules by trying to connect the seed module to its support

modules and the resultant subnetworks together. After all these "data" related modules are in place, the control modules are generated and the control signal lines connected.

The two main parts of the hardware structure generator, namely seed module generation and module interconnection, are discussed below.

3.1 Seed Module Generation

Before a hardware structure can be generated, some high level decisions have to be made. These decisions basically determine what the underlying architecture should be. They also provide some guidance and biases for selecting the seed and support modules. For simplicity, we assume that the systems are to be implemented as synchronous systems. Data flow graphs represent an extreme case in system implementation in the sense that the maximum number of hardware modules (one for each operator node in the graphs) is needed. Thus they are expensive to implement directly. They do however provide a lot of concurrency. Except for very trivial cases, almost all practical implementations of hardware systems based on data flow graphs must involve sharing of some hardware modules, i.e. very frequently the mapping of operator nodes to hardware modules is not one to one but rather many to one. This is done to reduce the cost of hardware, but usually results in a loss of concurrency.

The high level decisions discussed above come into play in the following manner. For each operator type (e.g. add, multiply, subtract), there is a list of candidate hardware modules that will implement it. All seed modules to be selected must be from the list of candidate hardware modules. Rules are then invoked to reduce the number of candidates. This will reduce the size of the set of candidates. However this still gives a one to many mapping for operator nodes to hardware modules. To obtain the many to one mapping for operator nodes to hardware modules, more decisions have to be made. These decisions have to do with how modules are to be shared and help to select the seed modules from the list of candidate hardware modules. To do this, each hardware module that is identified as a likely candidate for implementing an operator node must suggest itself as a seed module. This then gives us a set of hardware modules that will implement all the operator nodes in the data flow graph, i.e. a set of seed modules. However it must be noted that FUNCHARD should not always try to get a single module to replace the system since this can only happen if the system has the same (or subset of the) functionality as that module.

For seed module generation, some amount of user interaction is required. The user may be asked for things related to the architecture like what kind of interconnection structure should be used, e.g. bus or routing networks. It could also query the user for the type of performance expected, the amount of area available, and the power limitations for the chip. This may help in deciding whether a ripple carry adder or a

carry look-ahead adder should be used. The specifications of the system can be changed by the user during these interactive sessions as more information is obtained. Consider the situation where the user determines that the hardware structure generated by FUNCHARD have too many modules and that some parts of the system can run a bit slower without degrading the overall system performance. The user can feed back this kind of knowledge to FUNCHARD through the interactive sessions and have FUNCHARD regenerate the hardware structure, using the additional knowledge.

To handle the situation where incomplete knowledge in the domain, numerical weights might be associated with the hardware modules for setting priorities and biases. Through these weights, FUNCHARD might be told which hardware modules to pick for a given operator node or to set a priority in terms of choice of modules for the operator node. This in some sense models what the human designer is doing. Very often, the human designer has a preference for certain types of hardware modules for implementing a group of operator nodes. As more insight is gained, the preference may change. The weights are changed accordingly and propagated to the seed modules derived from the candidate hardware modules. These certainty factors may be used to set preferences in the selection of competing seed modules to implement a group of operator nodes.

3.2 Module Interconnection

Two kinds of rules are involved in module interconnections. There are the general rules that are not specific to a particular module. A rules specifying that one of the ports to be connected must be an output port and another must be an input port unless bidirectional ports are used is an example of this kind of rules. Other examples of such rules are those that are used for inferring when a multiplexor or demultiplexor is needed or when connections can can be connected in a hard-wire OR configuration.

The other class of rules are module specific and have to be given for each module. These rules represent the specialized knowledge that is needed for connecting the modules and for mapping arcs in the data flow graph to connections in the hardware structure. They may specify for example which input port to use for an arc that is incident on the operator node implemented by the module, or, based on the connection generated what values the control signals (e.g. opcodes, port select) should have. The sequence of control signal applications also have to be specified so that the right inputs/outputs are routed to the right modules and the right operators are performed with the inputs. There is a set of rules used by FUNCHARD for doing this. However the sequence generated is dependent on the hardware structure and hence is generated as the structure is being built. This control sequence is actually a specification of one or more control modules that are needed for the hardware structure. For most cases, these control modules will be state machines

implementing the control sequence.

4. An Example

Figure 2 shows a very simple data flow graph with two operators – an add and a multiply operator. There are three inputs – A, B, and C; and an output – D; to the graph. The arc C' connects the output of the add operator to an input of the multiply operator. We will go through some of the transformations that take place in generating the hardware structure. The sequence of transformations illustrates how a subnetwork centered around a seed module is generated.

4.1 Seed Module Generation

We have implemented part of the seed generation part of FUNCHARD with a production rule system based on MYCIN [8]. The MYCIN system was chosen because the seed generation process seemed to relate well to the MYCIN model, and because such a system was made available for our use. In the rest of this section we will discuss some of our experiences in implementing this part of FUNCHARD specifically for the simple ALU example, and we will briefly trace through the seed module generation as it occurs our system.

Rules are invoked to generate candidate hardware modules for each operator in the data flow operator graph. For our example, the PLUS node generates ADDER and ALU candidates, and the TIMES node generates three candidates – a PARALLEL-MULTIPLIER, a SERIAL-MULTIPLIER, and an ALU. Seed modules begin to establish themselves according to what the base architecture is thought to be like, and what the other modules in the graph will be like. At the current time, the base architecture is determined by asking the user for the values of parameters such as POWER, SPEED, and SIZE. Clearly, these parameters cannot be inferred by the system, and thus must be input by the user. However, in the future we would like the user to enter specific physical constraint values, and then let FUNCHARD infer the relative values of parameters

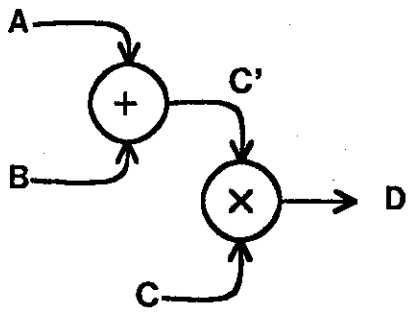

Figure 2. A Data Flow Graph Specification

such as POWER, SPEED, and SIZE. Weights, or CFs (Certainty Factors) in MYCIN terminology, also play an important part in determining which seed modules eventually will be chosen. Depending on the values of the base architecture parameters, the seed modules begin to propose themselves as being the best choice for the actual seed module. The more likely seed modules propose themselves with higher CFs and the seed module that is finally chosen is the one with the eventual highest CF.

A few of the seed-proposing rules are shown and explained below:

```
(DEFRULE rule-small-alu-seed
         (NOTDEFINITE seed-hardware-module)
         (DEFIS candidate-hardware-modules alu)
         (SAME size small)
         ==>
         (seed-hardware-module alu 0.9))
```

The above rule says that an ALU as a candidate module will very strongly (with CF = 0.9) propose itself when it is known that there are strict requirements on the size of the chip. Of course, it would be nice to come along later and realize that if there was some extra available area (due to area savings that had just been discovered somewhere else for example), then the CF that the ALU should be used here should be lowered. Unfortunately, this cannot be done with the system as it exists now. This is a generally hard problem that occurs in many places with FUNCHARD.

```
(DEFRULE rule-slow-alu-seed
         (NOTDEFINITE seed-hardware-module)
         (DEFIS candidate-hardware-modules alu)
         (SAME speed slow)
         ==>
         (seed-hardware-module alu 0.6))
```

This rule says that an ALU as a candidate module will somewhat strongly (with CF = 0.6) propose itself as the seed module when there are not strict requirements on the speed of this operator. The idea behind this rule is that if there is not a strict speed requirement, perhaps we can save some area by combining this operator with another one through sharing the same ALU.

```
(DEFRULE rule-adjacent-arithmetic-alu-seed
         (NOTDEFINITE seed-hardware-module)
         (DEFIS candidate-hardware-modules alu)
         (NOTSAME speed fast)
         (input-operation-is-arithmetic name)
         ==>
         (seed-hardware-module alu 0.9))
```

The above rule says that an ALU as a candidate module will very strongly (with CF = 0.9) propose itself as the seed module when one of the operators that feeds it as an input is an arithmetic (plus, minus, times, or divide) operator, and there is no strict speed requirement.

```
(DEFRULE rule-serial-multiplier-seed
        (NOTKNOWN seed-hardware-module)
        (DEFIS candidate-hardware-modules serial-multiplier)
        (NOTSAME size large)
        ==>
        (seed-hardware-module serial-multiplier 0.8))
```

The above rule says that a SERIAL-MULTIPLIER as a candidate module will strongly (with CF = 0.8) propose itself as the seed module when there is no tight size constraint.

In our example, the PLUS node proposes only an ALU, since we set the base architecture parameters SIZE and SPEED to SMALL and SLOW respectively. By the RULE-SERIAL-MULTIPLIER-SEED rule and the RULE-ADJACENT-ARITHMETIC-SEED rule, the TIMES node proposes a SERIAL-MULTIPLIER and an ALU , respectively.

Eventually, the seed modules all propose themselves, and the ones with the largest CFs are chosen. In the ALU example, this means that the TIMES node has the ALU (CF = 0.9) chosen over the SERIAL-MULTIPLIER (with CF = 0.8). Note that nowhere in our current system is it explicitly shown that the ALU used by the PLUS node and the TIMES node is actually the same one. In a more complete system this would certainly have to be known. As each seed becomes known, it is added to the seed graph. When all the seeds become known, the final output is displayed.

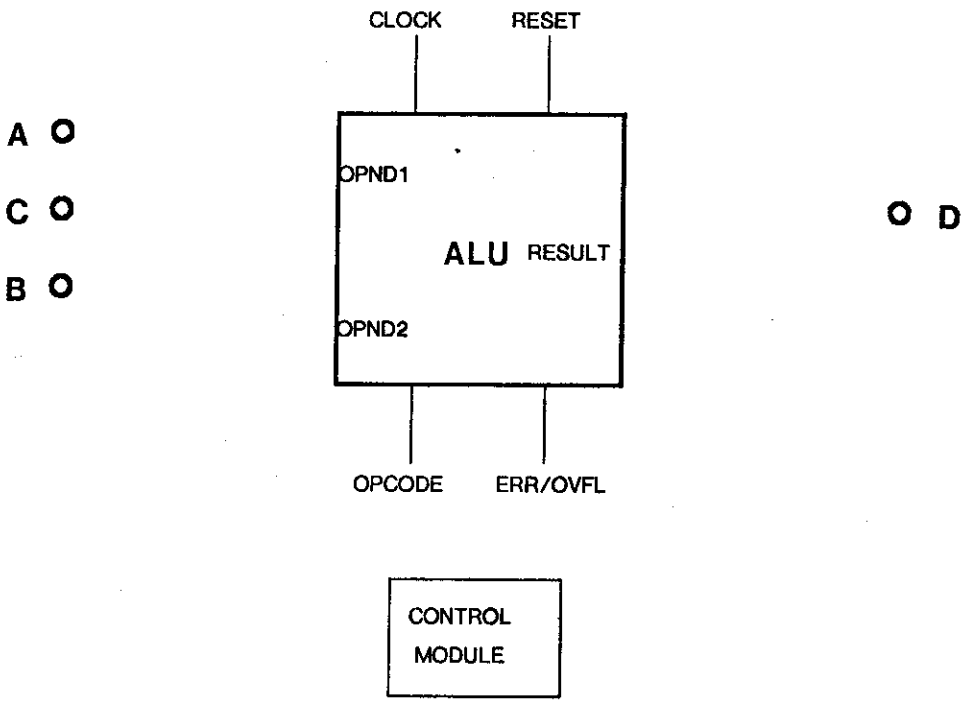

Figure 3. An ALU with an Invoked Control Module

4.2 Support Module, Connection, and Control Module Generation

The information contained in the frame for the ALU is shown below:

```
FRAME NAME:      ALU
OPERATORS:       ADD, MULTIPLY, SUBTRACT, DIVIDE
SUPPORT:         OPCODE GENERATOR (TYPE CONTROL), CLOCK (TYPE CLOCK)
PORTS:           OPND1 (DATA INPUT), OPND2 (DATA INPUT), RESULT (DATA OUTPUT),
                 OPCODE (CONTROL INPUT), RESET (CONTROL INPUT),
                 CLOCK (CONTROL INPUT), ERR/OVFL (CONTROL OUTPUT)
FUNCTIONALITY:   AF0) IF OPERATOR='ADD' THEN OPCODE=01 AND
                      RESULT=ADD(OPND1,OPND2),
                 AF1) IF OPERATOR='MULTIPLY' THEN OPCODE=02 AND
                      RESULT=MULTIPLY(OPND1,OPND2),
                 AF2) IF OPERATOR='SUBTRACT' THEN OPCODE=03 AND
                      RESULT=SUBTRACT(OPND1,OPND2),
                 AF3) IF OPERATOR='DIVIDE' THEN OPCODE=04 AND
                      RESULT=DIVIDE(OPND1,OPND2).
CONN. RULES:     AC0) IF ARC IS TOP INPUT ARC THEN CONNECT PORT AT THE ARC'S
                      HEAD TO OPND1,
                 AC1) IF ARC IS BOTTOM INPUT ARC THEN CONNECT PORT AT THE ARC'S
                      HEAD TO OPND2,
                 AC2) IF ARC IS OUTPUT ARC THEN CONNECT PORT AT THE ARC'S
                      TAIL TO RESULT
```

It contains a list of operators that it can perform. The list must include the add and multiply operators, otherwise the ALU module will not be selected. The frame also contains descriptions of the inputs and outputs of the ALU. In this case there are two data inputs – OPND1 and OPND2 – and one data output, RESULT. There is also an additional inputs and outputs for the opcodes, the clock input, and an error/overflow indicator will also be specified in the frame. These are the control inputs and outputs and all inputs and outputs are tagged as either data or control inputs and outputs. Two support modules have to be invoked. They are the system clock and a control module for driving the opcode input and reset input, as well as for sensing the error/overflow indicator. The functionality of the ALU is specified as a set of rules in the FUNCTIONALITY slot (rules AF0 to AF3). These rules relate the opcode values to the operators and the data inputs to the data output.

Figure 3 shows the ALU with an invoked support module.FUNCHARD attempts to connect the ports A, B, C, and D to the data ports of the ALU. It first recognizes that the add operation has to be done before the multiply operation. It also recognizes that the 3 inputs A, B, and C have to be connected to the two data inputs of the ALU. Thus one multiplexor is needed to multiplex two of the inputs to one of the ALU data inputs. Since the output of the add operation is an input to the multiply operation and that the add and multiply operations are performed by the same ALU, an additional input has to be connected to the ALU data inputs. A pair of pseudo input and output ports are created to handle the connection for C'. Let C" and C"' be the pseudo input and output ports, respectively. The connection between C" and C"' represents the arc C'. There are thus a total of 4 inputs that have to be connected to the ALU data inputs. Hence an additional multiplexor is needed. Also, as the data output of the ALU is used twice, once as an input to the

ALU and another time as an output to the port D, a demultiplexor is required at the data output of the ALU. Each of the multiplexor and demultiplexor has a control input for selecting the input or output to be used.

More rules have to be invoked to connect the multiplexors to the ALU and the ports as shown in Figure 4. The outputs of the multiplexors are connected to the ALU data inputs while the data output of the ALU is connected to the input of the demultiplexor. FUNCHARD has to connect the inputs A, B, C, and C" to the multiplexor inputs. Since the ports A and B carry operands to the add operator, the two inputs must be connected to different multiplexors. Similarly for C and C". Now the outputs of the demultiplexor have to be connected to the output ports D and C"'. This is done by connecting one output port of the demultiplexor to D and the other to C"'. Realizing that C" and C"' are pseudo ports, they can be omitted.

Having established the data path connections, the control signal lines and modules have to be put in. The sequence of events that has to happen is shown in Figure 5. For the add operation, the inputs A and B have to be available at the data inputs to the ALU at the same time. Thus the select signals for the multiplexors have to switch in the right input ports before the ALU can do the add operation. The select signals can be applied at the same time or one after the other. In any case they must be applied before the ALU is activated. The opcode for the ALU is then applied and the ALU is activated. After some time, in the absence of an error/overflow indication, the ALU output is valid and has to be switched to the right output of

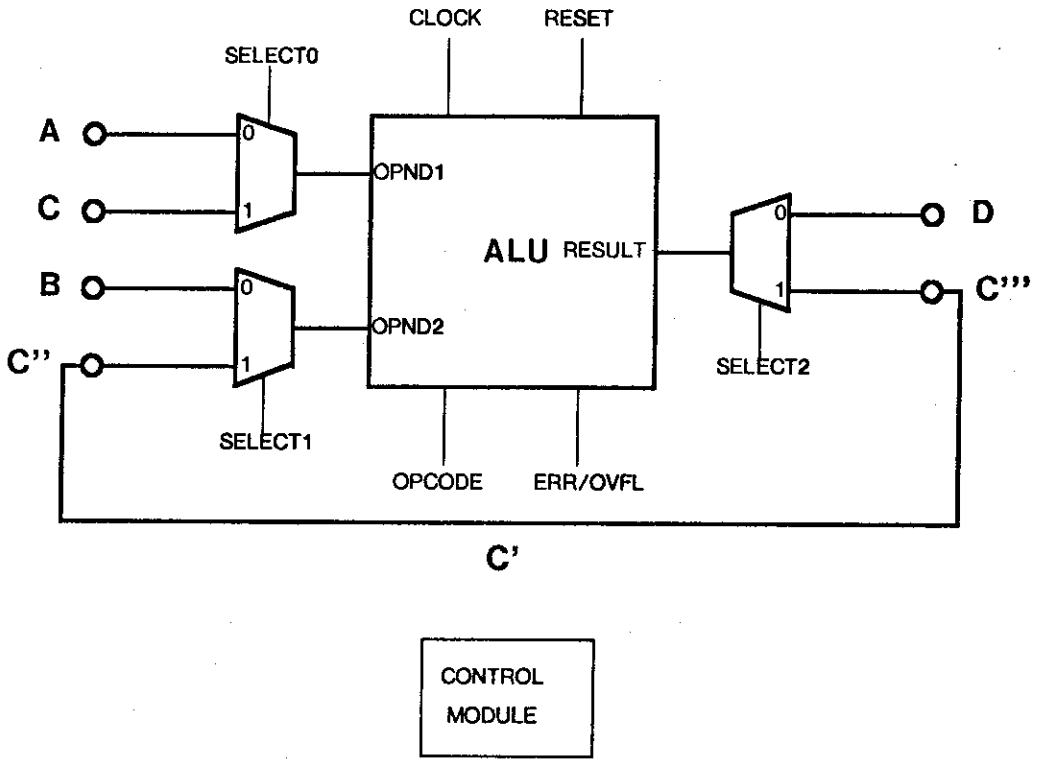

Figure 4. A Complete Subnetwork of Data Related Modules

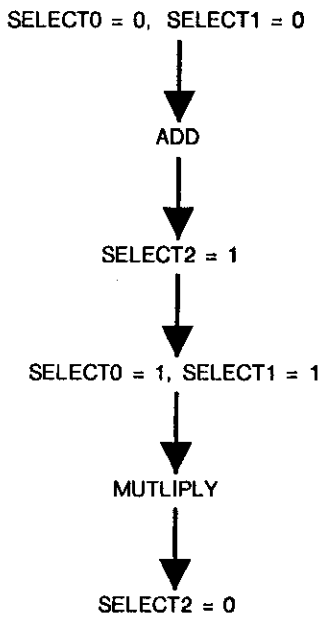

Figure 5. Control Sequence for the ALU Example

the demultiplexor. In this case the C"' output has to be selected. This is then repeated for switching in the inputs C and C", which is followed by activating the ALU with the multiply opcode. The ALU output is then switched to the D output. If there is an error/overflow indication, a special mechanism has to be given to handle it. This can be specified either in the ALU frame or by the user when specifying the system. The sequence of events and the error/overflow handling mechanism basically specifies what the control module must do. For this case, the control module is a state machine. Again rules are used to generate the internal structure of the control module. These rules are used to determine the number of inputs and outputs needed for the state machine, the number of states required, the outputs that are produced for each of the states, and when state transitions should occur.

5. Conclusions

We have proposed a viable rule-based expert system for translating high level system specifications to hardware structures. Such an expert system is important in the design of VLSI systems as it will speed up the design process. This is because the mechanical and time-consuming process of hardware structure generation can now be delegated to the computer. The hardware designer can instead concentrate on the more important issues that require creativity and human judgment. FUNCHARD also enables more alternatives to be studied in the evaluation of implementation structures for the system. This is particularly important in VLSI where the systems designed are so complicated that an optimal solution would be expensive and in

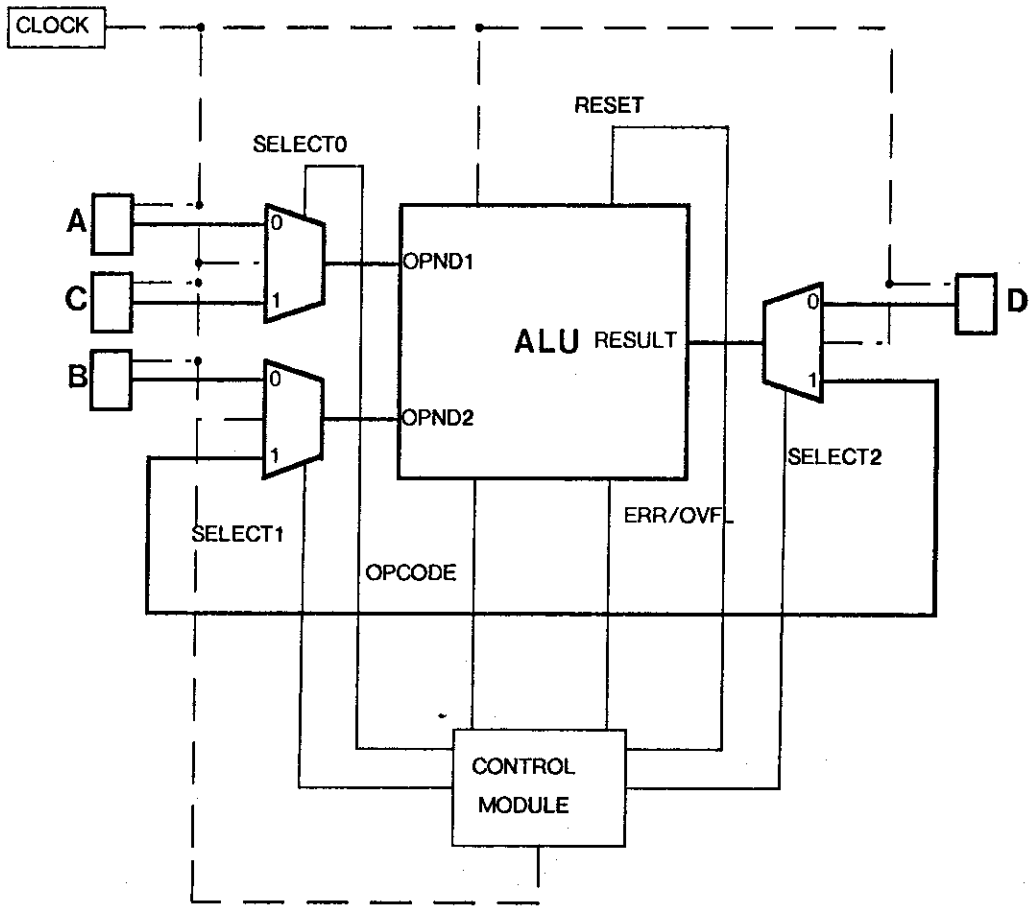

Figure 6. The Complete Subnetwork

some case impossible. By studying the alternative solutions an acceptable and practical solution can be obtained with relative ease.

6. Acknowledgments

The authors would like to thank Ramesh Patil for his suggestions and comments on this work.

7. References

[1] Barbacci, M.R., *Automated Exploration of the Design Space for Register-Transfer (RT) Systems*, University Microfilms International, Ann Arbor, Michigan, 1973.

[2] Barbacci, M.R. and Siewiorek, "Some Aspects of the Symbolic Manipulation of Computer Descriptions", CMU Department of Computer Science, July 1974.

[3] Kowalksi, T.J. and Thomas, D.E., "The VLSI Design Automation Assistant: Learning to Walk", *Proceedings of the IEEE 1983 International Symposium on Circuits and Systems*, Newport, California, pp. 186-190.

[4] Minsky, M., "A Framework for Representing Knowledge", *MIT AI Laboratory Memo No. 306*, June 1974.

[5] Patil, R. S., "Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis," *Technical Report no. MIT/LCS/TR-267*, Laboratory for Computer Science, Massachusetts Institute of Technology, October 1981.

[6] Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, vol. 5, no. 2, Summer 1974, pp. 115-135.

[7] Sacerdoti, E.D., "The Nonlinear Nature of Plans," *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, vol. 1, Tbilisi, Georgia, USSR, September 1975, pp. 206-214.

[8] Shortliffe, E.H., "Consultation System," in Chapter 3 of Shortliffe, Computer-Based Medical Consultations: MYCIN, Artificial Intelligence Series, Book 2, Elsevier Computer Science Library, 1976.

[9] Siskind, J.M., Southard, J.R. and Crouch, K.W., "Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions", In the *Proceedings of the 1982 Conference on Advanced Research in VLSI*, Paul Penfield (Editor), Artech House Inc., Dedham, Massachusetts, January, 1982, pp 28-40.

[10] Stefik, M., Bobrow, D.G., Bell, A., Brown, H., Conway, L., and Tong, C., "The Partitioning of Concerns in Digital System Design", In the *Proceedings of the 1982 Conference on Advanced Research in VLSI*, Paul Penfield (Editor), Artech House Inc., Dedham, Massachusetts, January, 1982, pp. 43-52.

[11] Snow, E.A., *Automation of Module Set Independent Register-Transfer Level Design*, University Microfilms International, Ann Arbor, Michigan, 1978.

[12] Snow, E.A., Siewiorek, D.P., and Thomas, D.E., "A Technology-Relative Computer-Aided Design System: Abstract Representations, Transformations, and Design Tradeoffs", *Proceedings of the 15th Design Automation Conference*, Las Vegas, Nevada, 1978, pp. 220-226.