An Architecture of a Specification Design Expert System

YUKIHIRO NAKAMURA*, MITSUTERU YUKISHITA*
TOMOTAKA UCHIHASHI** and KIYOSHI OGURI*

At the present stage of processor hardware design, design support technology for lower levels, such as layout design, has reached a practical state. We have studied and developed an upper level CAD system.

We express those specifications in basic terms with object(or entity) descriptions for processes and data, and relation descriptions for the correlation between these entities.

An expert system for computer architecture design has been implemented. With it, the designer is able to use a specification description language in much the same way as a natural language. The system employs 293 rules to analyze the consistency and completeness of the specification. Finally, and RTL behavioral description can be synthesized from the original specification description through a process that employs 113 rules.

We used the above expert system to perform the functional design of a simple pipeline processor which has 127 specification language lines. In this case, 212 SFL statements were generated by the system. In the simple processor, specification analyzing time is about 30 seconds, and synthesizing time is about 2 minutes.

1. Introduction

This paper describes an Architecture of a specification design expert system which translates specifications into a hierarchical behavioral description language, called SFL (Structured Function description Language). We have already developed SFL, its behavioral simulator, and its architecture & logic synthesizer. We have now integrated them into a high level CAD system, called PARTHENON, for practical use [1-5]. Though PARTHENON can be counted among the most advanced tools for raising human interface, it takes a hardware expert to use it effectively. We, therefore, re-directed our work toward developing a technology that would allow design specifications to be described in natural language and behavioral logic to be synthesized automatically. Since, in order to realize this automatic synthesizer, knowledge-processing technology must be introduced to enable the accumulation of design know how in a knowledge base, we have prototyped the system as an expert system [6].

The total system overview and our current stage of development are shown in Fig. 1.

This is a translation of the IPSJ Best Paper Award paper that appeared originaly in Japanese in Transactions of IPSJ, Vol. 30, No. 5 (1989), pp. 564-577.

*NTT Communication Science Laboratories, 2-Chome, Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-02, Japan.
**NTT Network Information Systems Laboratories, 1-2356, Take, Yokosukai-shi, 238-03, Japan.

2. PARTHENON System

We will begin by describing the features of the SFL language, which is at the core of the PARTHENON system. SFL was developed to aid in the design of the hardware functions and behaviors of ASICs composed solely of clock—synchronized circuits. The main features of SFL are as follows: (1) It is not mixed with connection description, but employs only behavioral description, and it provides hierarchical expression of behavioral description. (2) It permits the description of parallel processing operations by adopting a new hardware task concept. And, (3) it is linked with the behavioral simulator, logic synthesizer, and other components of the processing system.

We designed a 32 bit RISC-type processor using PARTHENON in only four days. This processor has 47 instructions which are a subset of DLX (DLX is very similar to MIPS architecture). The size of the circuit synthesized from SFL description (about 1,200 lines) was about 14,000 gates. This processor chip was received from the manufacturer last October, and it functions correctly.

3. Model & Input

PARTHENON is able to perform completely automatic synthesis of logic circuits from SFL behavior description. We, therefore, developed a technique for supporting synthesis from a higher specification descrip-

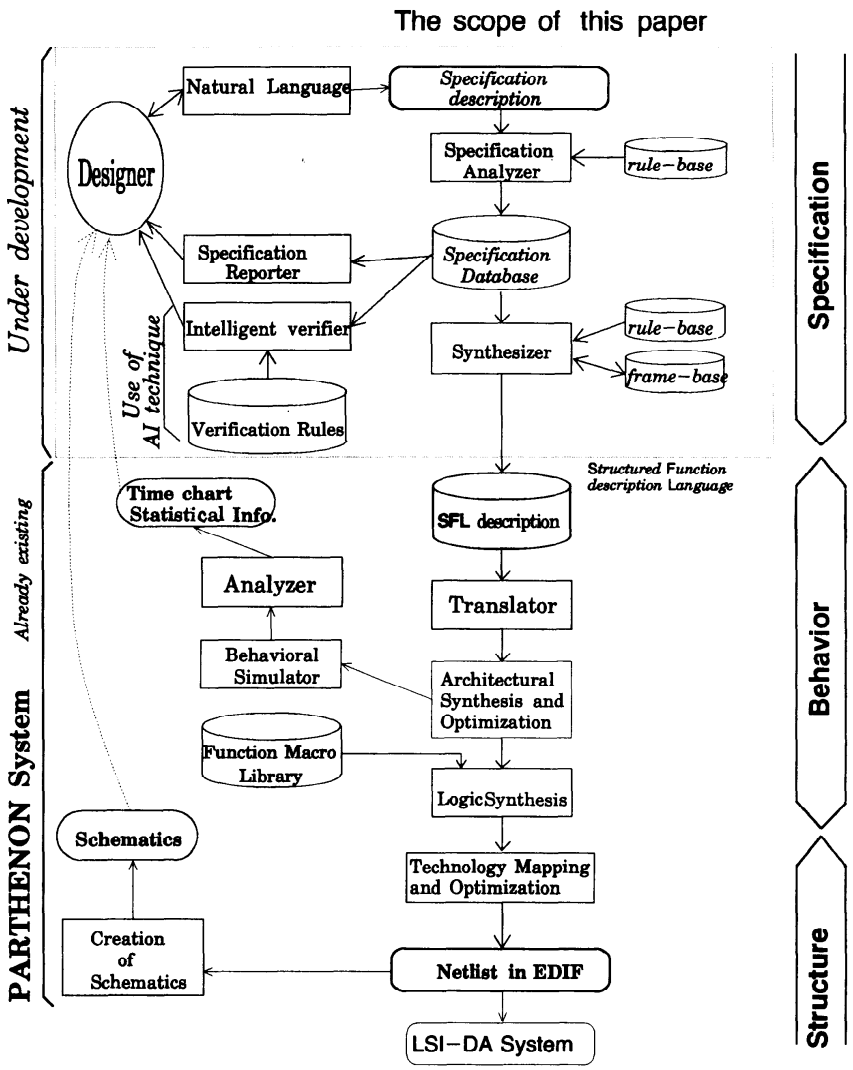

Fig. 1 System Overview.

tion to SFL. Specifications have in the past been described in natural language; This specification expert system can, however, analyze processor structure which is described in natural language. Next, we will explain the basic model of this system.

3.1 Basic Model

We wish to express processor specifications in basic terms using object (or entity) descriptions for processes and data, and relation descriptions for correlation between these entities.

Our reason for choosing this simple model is to allow the expression of specifications with as little ambiguity as possible. This is achieved by expressing processor elements using a limited number of entities and by correlating entities with only those relation descriptions used in the hardware design.

Based on this objective, we conducted an analysis of hardware specifications. As the entities, we use DATA entities to express information manipulated in the hardware, and PROCESS entities to express individual hardware processes. As the relations, we employ STRUCTURE, DATA FLOW, and LOWER CONTROL FLOW (not having time order or condition) to express the inter-relationships between entities. We also provide an upper-relationship to correlate inter-relationships. In expressing the behavior specification level, it is necessary to provide the conditional relationship that

controls the process execution sequence. More specifically, a conditional relationship expressed in terms of "EVENT (if)," "CONDITION (when)," and "TIMING (before/after/concurrent)," does not correlate the entities themselves, but rather the relationship between them. This is why we have introduced an upper-relationship.

We have confirmed that these entities and their inter- and upper-relationships can be used to express almost all hardware specifications.

3.2 Specification Description Language

Our specification language is structured on the above model, and it adopts natural language as its description form. However, because the sole object is hardware specification and because we wanted to make the language structure easier to analyze, we have applied the following restrictions:

(a) The use of particles and auxiliary verbs has been limited,

(b) Only one sentence format is provided per verb to avoid ambiguity in sentence analysis.

This language expresses the inter-relationship between entities using the Japanese syntax: SUBJECT + OBJECT + VERB. The upper-relationship between inter-relationships is expressed using the Japanese syntax: SUBJECT + BASIC SENTENCE + CONJUNCTION + OBJECT + VERB.

Basic sentence:

〈NAME〉は [HA], 〈NAME〉を [WO]〈VERB〉する [SURU].

Complex sentence:

〈NAME〉は [HA], 〈BASICSENTENCE〉〈CONJUNCTION〉, 〈NAME〉を [WO]〈VERB〉する [SURU].

In these sentences, a NAME with the particle "[HA]" signifies the subject, a NAME with the particle "[WO]" signifies the object, and a VERB with the suffix "[SURU]" denotes the predicate. By extension, a BASIC SENTENCE with a conjunction comprises a complex sentence. Reserved words are shown in Table 1.

Inter-relations are expressed in basic sentences, and Upper-relations in complex sentences.

4. Knowledge-Based Analysis

The specification analyzer is a subsystem of the expert system (Fig. 2). It analyzes the consistency and completeness of specifications described using the language characteristics just mentioned. Concretely, if a specification is described in a basic sentence, entities expressing the subject and object are correlated in hardware using the verb as the key. If, on the other hand, a specification is described in a complex sentence, the inter-relationship between the basic component sentences is established using the conjunction as the key. For this analysis, inconsistent and incomplete sentence patterns are chosen in advance and stored in the rule-base. Therefore, sentences can be dynamically checked after they are input by the designer. The check items are STRUCTURE, DATA FLOW, and LOWER CONTROL FLOW in the inter-relationship, and UPPER CONTROL FLOW (having time order and condition) in the upper-relationship (Table 2). Finally, we created a relationship database for specification information which consists of entities and the inter- and upper-relationships between them. We incorporated a reporter which not only reports errors found by the analysis checker, but also immediately provides information (STRUCTURE, DATA FLOW, and CONTROL FLOW) required by the designer in the form of a tree.

Table 1 Corresponding to Relations.

Relation Class	Relation	Verb
Structure	Structure	Consist
	Use	Use
	Attribute	Is
	Part of	Part
Data flow	Refer	Refer
	Produce	Produce
	Maintain	Maintain
	Modify	Modify, Change, Add, etc.
	Transfer	Send, Receive, Transfer
Lower control flow	Start	Start
	Generate	Generate
	Terminate	Terminate
Upper control flow	Order	Before, After
	Condition	If
	Time	When

Table 2 Relationship.

Relation class	Relation	Function
Structure	Structure	Shows hierarchical structure Defines data set
	Utilize	Shows utilization relation
	Attribute	Determines data value
Data flow	Refer	Refers to data
	Produce	Produces data
	Maintain	Maintains data value
	Modify	Changes data value (Equal to changing condition value when the data value is regarded as the condition)
	Transfer	Transfers Data to othe r process
Lower control flow	Start	Starts other process, terminates itself
	Generate	Starts other process, continues working
	Terminate	Terminates other process
Upper control flow	Order	Detemrines time-order in relations
	Condition	Related to condition
	Time	Related to time

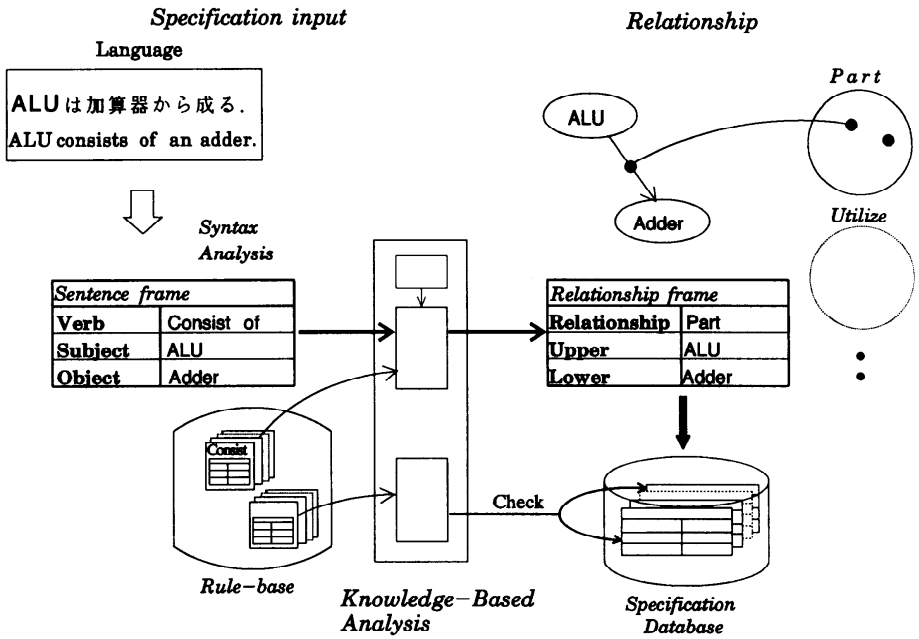

Fig. 2 Specification Analyzer.

4.1 Sentence Meaning Analysis

We adapted KBMS (Knowledge-Base Management System) [7] in the sentence meaning analyzer and analysis checker where processing is based on pattern matching. This system was developed by NTT, and it describes processes using production rules, improving process flow readability and facilitating rules maintenance.

In the analysis checker, check rules are organized into a system to assure that no check is missed. When processes are executed, the rules are applied to frames in the KBMS, and relationship database, which are the final output, are expressed in the frames.

Next, we explain the rule for making structural relationships (see Fig. 3(a)). If there is a frame containing "[KOUSEISURU]" or "[KOUSEISARE]" as the verb, "[HA]" as the particle to indicate the subject, and "[KARA]" to indicate the object, the rule will create an instance frame for a structural relationship which places "?u" (bonded to the subject) into the upper slot, and "?l" (bonded to the object) into the lower slot.

4.2 Analysis Checker

The check items are STRUCTURE, DATA FLOW, and LOWER CONTROL FLOW in the inter-relationship, and UPPER CONTROL FLOW in the upper-relationship. Table 3 shows examples of the approximately 180 rules used to check all of the relationship frames.

We will now use an example to explain a rule for checking structure. Let's say there is a relationship

```
(Defrule (Make_relation make_part1 (:PRIORITY 0))
  (Frame (Sentence ?s
    (Verb ?v|(Member ?v '(KOUSEISARE KOUSEISARERU)))
                    (HA ?u)
    Condition part  (KARA ?l) ))
  -->
  (Create-instance  'Structure nil
                    'Upper  ?u
    Action part     'Lower ?l))
```

Fig. 3(a) A Rule for making relation instance frames.

```
(Defrule (Semantic_check C_PART1 (:PRIORITY 0))
  (Frame (Part ?p1 (Upper ?u) (Lower ?l)))
  (Frame (Part ?p2 (Upper ?l) (Lower ?u)))
  -->
  (Call (Err_pos))
  (Call (Format T "~%Error  recursive"))
  (Call (Err_disp (LIST ?u 'part ?l )))
  (Call (Err_disp (LIST ?l 'part ?u )))
  (Call (Recovery ?ln)) )
```

Fig. 3(b) A Rule for checking relation frames.

frame created from the description "A consists of B" with "A" as its upper structure and "B" as its lower structure. If there is also a relationship frame created from the description "B consists of A" with "B" as its upper structure and "A" as its lower structure, a contradiction will be generated in the structural relationship. A rule such as shown in Fig. 3(b) is used to detect this kind of contradiction.

4.3 Reporter

As explained above, the reporter both reports error information generated by the checker and displays infor-

Table 3 Items of checking rule.

Structure	31rules
Recursive	Structure relation makes loop.
Conflict	More then two processes share one process.
Date flow	**72rules**
Existence	Data not made by process exists.
Maintain & Modify	Data maintained by a process aren't modified, or Data modified by a process aren't maintained.
Relate to process	Data aren't modified by process.
Modify Input/Output	Data modified by a process are used by another process Process receiving data doesn't send it, or process not receiving data sends data.
Name uniqueness	Data transferred from one process to another has unique name.
Lower control flow	**33rules**
Recursive	Process triggers another process which triggers it in return.
No trigger	Process not triggered is terminated.
Process isolation	Process is neither triggered nor terminated.
Upper control flow	**44rules**
Non condition clause	Structure described in condition clause.
Clause recursive	A loop exists in time order.

mation (structure, data flow, and control flow) requested by the designer in a tree structure (Fig. 4). The designer uses this report to search for unintended specifications. If he finds such to exist, he may change the description and re-conduct the analysis. By repeating this process, the designer can set intended specifications.

5. Knowledge-Based Synthesis

Due to improved specifications, more detailed design data is required in the next step. Expressions for this data are adopted at the functional/behavioral level, because it represents the next step in conventional design methods. SFL, a register transfer level (RTL) design language, is employed for the output description of the synthesizer (Fig. 5). Synthesis of RTL logic is carried out in four steps using rules from a relationship database generated by the analyzer.

(1) The synthesizer classifies entities in the specification into SFL elements using the relationship patterns. The correlation of relations (between entities in the specification description language) and elements (in SFL) are shown in Table 4.

(2) The synthesizer creates instances from classes prepared for each SFL element in advance and fills the value described in the specification into the slots of the instance frames.

(3) The synthesizer fills values not described in a specification into the slots using a knowledge function.

(4) Using the information in the instance frames, the synthesizer outputs an SFL description.

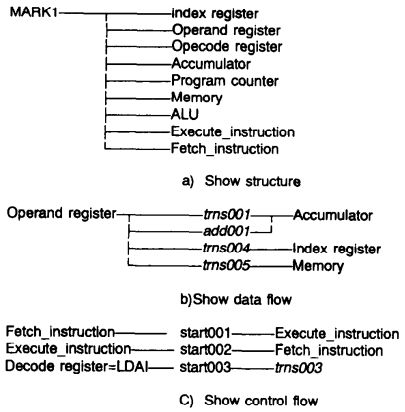

Fig. 4 Example of report executing.

Table 4 SFL Elements.

Relation	Elements	Concept
Structure	Module	Its hardware part consists of some modules and combinational circuit.
	Functional circuit	Its hardware part expresses operation during clock pulses.
	Stage	Expresses an individual operation executed within several clock pulses.
Data flow	Storage	Consists of memories and registers.
	Terminal	Interface between hardware parts.
	Bus	Bi-directional paths.
Control flow	Control terminal	Teminal for controlling stages or modules.
	Signaling line	Control path activated during clock pulses.
	Output terminal of decoders	Controls other combinational circuits, registers, and modules.

5.1 Entities Classification

The rules that correlate entities with SFL elements are based on STRUCTURE, DATA FLOW, and CONTROL FLOW. Processes are classified according to the combination of rules applied. These rules incorporate the characteristics of each hardware element, e.g., how it controls other processes, and the kinds of processes it has on or under the structure level. We will now use an example of a functional circuit to demonstrate the classification concept. This functional circuit has four characteristics:

(1) It maintains data,
(2) it modifies data,
(3) it is used by other processes, and
(4) it does not control other processes.

That is, if a process has the first three relationships but does not control other processes (the fourth relationship), it is identified as a functional circuit. In the specification description example shown in Fig. 6, the

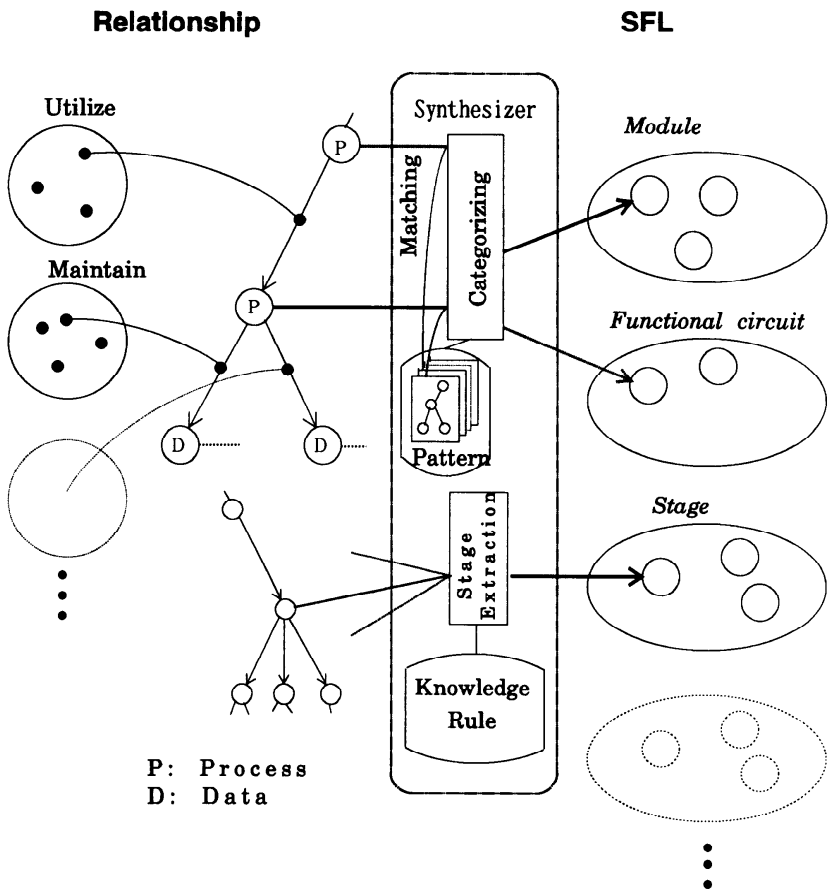

Fig. 5 Concept of Synthesizer.

An ALU consisting of ,ADDW, SUB, AND, OR, and EOR is used by instruction_sets. The ALU modifies the operation result. An ANDW using ALU, index register, accumulator, and memory sends the contents of index register to memory interface,(1) contents of accumulator and a memory data to ALU, and (2) result of ALU operation to accumulator. (3) Memory data sent to accumulator and index register is 16 bits in length. (4) The operation result is 16 bits in length.

Fig. 6 Example of a Specification.

ALU is used through an instruction set (3), it is sent memory data (1), it modifies operation results (2), and does not control other processes (4). Therefore, it is regarded as a functional circuit. Because the ALU is classified as a functional circuit, an instance corresponding to the SFL functional circuit class is created.

5.2 Filling in Slot Values Using Knowledge

When processes fill in slot value, some necessary information of SFL are not written in specification descriptions. So, we solve the problem by using knowledge. In the example of Fig. 6, DATA FLOWS related to the ALU are

Markはpipeline計算機である.
Markは演算器とメモリとプログラムカウンタとアキュムレータとオペコードレジスタとオペランドレジスタとインデックスレジスタと. . . と制御レジスタを持つ.
Markの命令実行サイクルは命令フェッチ部と命令実行部からなる. 命令実行部はJMP, LDN, . . . , CMPを実行する.
:
CMPは, アキュムレータの内容が負である場合に, 制御レジスタの内容をインクリメントし, 制御レジスタに書き込む.

(i) Japanese specification of a pipeline processor

Mark is a pipeline processor.
Mark has ALU, memory, program counter, accumulator, an opecode register, an operand register, an index register, .. , and a control register. Mark's instruction executing cycle consists of instruction fetch unit, and instruction execute unit. The later executes JMP, LDN,..and CMP instructions.
:
If the content of an accumulator is negative, CMP instruction increases the content of control register, and writes the new content into control register.

(ii) English translation

Fig. 7(a) Example of pipeline processor specification.

(1) "ANDW sends the contents of the accumulator and memory data to ALU",

(2) "ANDW sends the result of ALU operation to accumulator",

(3) "memory data of 16 bits in length is sent to ALU", and

```
module MARK {
      :
circuit_class INCREMENTER1 {
    input       IN<13> ;
    output      OUT<13>, CARRYOUT, OVERFLOW ;
    instrin     INCREMENT ;
    instruct_arg INCREMENT (IN) ;
    instruct INCREMENT  OUT = IN + Ob1 ;
}
      :
instruct START
   generate Fetch.TASK1() ;
   stage Fetch {
           state_name STATE1 ;
           first_state STATE1 ;
     state STATE1
       alt { TERMINATE1 : finish ;            ←IF Then
             else : par {                     ←  else
      :
           i_decider.deocde(Memory_IF.I_read(CR).op_code<15:13>).CMP  :
             generate stage2.CMP ;                  ↑ condition
      :                ↑ action
 ┌── stage stage2 {                          ┌──────────────────┐
 │ ┌── state_name  Inst ;                    │ par = parallel   │
 │ │   first_state Inst ;                    │ alt = alternative│
S├ ┴── state Inst par {                      │ any = any        │
t│       any {                               └──────────────────┘
a│ s         :
g│ t         (stage2.CMP & ACC_decoder.decode(ACC).NEGATIVE) : par {
e│ a             CR  := incrementer2.INCREMENT(CR).OUT ;
 │ t             stage2_CR_access = 0x0001 ;   ↑ Object.Task(Parameter).Output
 │ e     }
 │           :
 │       finish ;
 │   }
 └──         :
}
```

Fig. 7(b) Example of synthesized SFL.

(4) "the operation result is 16 bits in length".

(1) is data input to ALU, (2) is data output, (3) and (4) show bit length. Using this information, the terminal frame slot (bitlength, flow), and the term slot and input slot in the circuit_component frame are filled.

6. Prototype

The prototype system, shown in Fig. 1, has already been implemented. It was coded using LISP on an HP Apollo Domain system and contains the following rules and frames:

(a) 293 rules for analyzing the consistency and completeness of specifications,

(b) 113 rules in the rule base (system based on KBMS) for categorizing entities and obtaining information from specifications, and

(c) 16 classes of frames corresponding to the SFL elements.

An example of specification description in a simple pipeline processor is shown in Fig. 7(a), and the SFL behavioral description generated by the specification synthesizer is shown in Fig. 7(b).

7. Experimental Results

In this case study, we used the above expert system to perform the functional design of another simple pipeline processor (an advanced controlled microprocessor) composed of 2,000 gates. The simple pipeline processor has 127 specification language lines, and 24 fundamental instructions. It consists of instruction fetch and execution units. The instruction fetch unit, which is initialized first, fetches one instruction from the memory and activates the execution unit. The execution unit then executes the instruction. Before completing the execution, though, it re-activates the instruction fetch unit, placing the two units in parallel operation. In this case, 212 SFL statements were generated by the system. Whereas, 192 were produced through design by

a human. The main reason for the greater number of SFL steps in the system was an insufficient knowledge-base on parallel control structure. In the simple processor, specification analyzing time is about 30 seconds, and synthesing time is about 2 minutes.

8. Conclusion

An expert system for supporting computer architecture design has been implemented. With it, the designer who is not a hardware expert is able to use a specification description language in much the same way as natural language without SFL knowledge. The system employs 293 rules to analyze the consistency and completeness of the specification. Finally, an RTL behavioral description, expressed in SFL, can be synthesized from the original specification description through a process that employs 113 rules and 16 frames. Rules for complex architectures are presently being implemented in the system.

At present, we only have one synthesized strategy. Therefore, we are now adding strategies to increase processor speed and reduce the number of resources, and to modify synthesizing rules so description can be done as efficiently as by hand.

Acknowledgement

The authors would like to thank Dr. F. Ishino and Dr. T. Kawaoka of NTT Network Information Systems Laboratories for their helpful suggestions and encouragement.

References

1. NAKAMURA, Y., OGURI, K., NAKANISHI, H. and NOMURA, R. An RTL Behaviral Description Based Logic Design CAD System with Synthesis Capability, IFIP CHDL 85 (Aug. 1985), 64–78.
2. OGURI, K., NAKAMURA, Y. and NOMURA, R. Evaluation of Behavior Description Based CAD System Used in Prolog Machine Logic Design, *IEEE ICCAD-86* (Nov. 1986), 116–119.
3. NAKAMURA, Y. An Integrated Logic Design Environment Based on Behavioral Description, *IEEE Trans. on CAD*, **CAD-6**, 3 (1987), 322–336.
4. NAKAMURA, Y. and OGURI, K. An RTL Logic Design Aid for Parallel Control VLSI Processors, *IFIP VLSI 87* (Aug. 1987), 13–28.
5. NAKAMURA, Y., OGURI, K., NAGOYA, A. and NOMURA, R. A Hierarchical Behavioral Description Based CAD System, *EURO ASIC 90* (May 1990), 282–287.
6. YUKISHITA, M. and NAKAMURA, Y. Investigation of Specification Design Expert System, Monograph of Technical Group on Design Automation of Information Processing Society of Japan, 44–1, 1988.
7. HATTORI, F., SHIMIZU, N., TSUCHIYA, H., KUWAHARA, K. and WASANO, T. Knowledge Base Management System (KBMS), Monograph of Technical Group on Knowledge Engineering and AI of Information Processing Society of Japan, 41–6, 1985.