Innovative System-level Design Environment Based on FORM for Transport Processing System

Kazushige HIGUCHI and Kazuhiro SHIRAKAWA
NTT Optical Network Systems Laboratories
Y-807C 1-1 Hikari-no-oka, Yokosuka, Kanagawa 239-0847, JAPAN

Abstract

This paper presents a system-level design environment for data transport processing systems. In this environment, designers can easily verify system behavior by formally defining data structures and their related actions, without considering detailed timing. In addition, the verified specification can be translated into synthesizable RTL descriptions by a dedicated RTL generator. Thus, using lower-level EDA tools, actual hardware can be obtained directly from a system-level specification.

1. Introduction

Recently, various kinds of multimedia services on the Broadband Integrated Services Digital Network (B-ISDN) have been introduced, and many more new services will be available in the near future. To keep in step with this situation, transport processing systems must be enhanced continuously.

From the system design point of view, it is very important to define total system behavior as soon as possible. So, system designers often want to concentrate on drawing new ideas and verifying them quickly without worrying about detailed hardware implementation issues such as precise clock timing. To realize such an environment, several rapid prototyping systems have been proposed [1][2][3]. However, they still require a great deal of low-level specifications with which the system designers are not familiar. Thus, even now, system designers often specify system-level behaviors using their own paper-based informal formats, because they cannot use any computer -aided tools.

To help remedy this situation, we proposed a novel system-level specification method called “FORM” (Frame Oriented Representation Method) [4]. The method is well tuned to specifying transport processing systems. Based on FORM, we developed a system-level design environment. The environment allows designers to specify system-level behavior formally through familiar input methods, and verify them using a system-level simulator. In addition, the verified specifications can be translated into synthesizable RTL descriptions by an RTL generator. Thus, actual hardware can be obtained from the system-level specifications using other EDA tools. We believe that this environment bridges the gap between system-level designs and lower-level ones in the telecommunications area.

This paper discusses related work in the next section. Section 3 describes the FORM system overview including features, configurations, and the system model. Section 4 describes the specification editor for the formal specification description tool (FORM_EDIT). Section 5 describes the specification simulator for directly simulating its description (FORM_SIM). The RTL generator (FORM_GEN) for formal specifications is described in Section 6. Section 7 presents the evaluation results of when the FORM system is applied to the specification description of the ATM-AAL5 termination function[5][6].

2. Related Work

To describe a specification of telecommunication systems, various methodologies such as SDL, Estelle, and LOTOS and their tools were proposed [7][8][9][10][11]. They are based on a finite state machine model. Therefore, they can model telecommunication systems and can be used to describe system specifications. The aim, however, is to describe high-level service applications and protocols of telecommunication systems. This makes it difficult to use the specifications for transport processing systems which must process transmission data in real time. So the methodology is not suitable for our requirement.

On the other hand, the Hardware Description Language (HDL) is used when describing a hardware specification. By using HDL, designers can describe high speed and well optimized hardware specifications. However, when describing HDL, designers must be well aware of hardware

and they must consider the activation timing of each action in the description. Thus it is difficult for system designers to describe a specification of a transport system using HDL. Recently, various EDA tools that describe RTL/gate level specification have been proposed. Some EDA tools have several kinds of graphical user interfaces (GUIs) such as hierarchical block diagrams, state diagrams, and truth tables [12]. Since designers can select the entry interface as they like, these tools may be regarded as more user-friendly design entry tools. Even though such tools have sophisticated entry interfaces, they are the same kind as existing ones because, on such tools, the final descriptions of system functions must be broken down into RTL taking into consideration the activation timing of each process. Although behavioral level design tools have been developed and some progress has been made in improving the ease of use, designers must still recognize clock cycles and provide constraints for execution timing [13]. Therefore, these tools are suitable for logic circuit designers, but not yet suitable for system designers to represent system-level specifications.

To solve the above problems we propose a new design method for the transport processing system FORM, by which system designers are able to describe easily transport processing system specifications. Our system is built on FORM and frees system designers from considering the activation timing of each process. The system accomplishes this through describing specifications using only synchronization, the transmission data structure, and the Transport Processing Action (TPA) for the transmission data. Some systems have a similar input form using the frame structure and action for system functions [14][15]. However, the designers must recognize clock cycles and define tool constraints for activation timing. The proposed system can optimize state transitions for controllers and will be useful for logic circuit designers who must design efficient circuits for ASIC. On the other hand, our system aims to easily express system-level specification written by system designers. So in our system, designers can describe specifications without suffering hardware constraints and rapidly achieve a prototyping system such as the FPGA-based emulator to verify the functions [16]. Our system enables system designers to concentrate on considering system-level functions without specifying the hardware implementation in detail.

3. FORM System Overview

3.1 System Characteristics

The characteristics of the FORM system are described below.

a) Formal specification

The transport processing specifications are specified using the transmission data structure and relevant processing functions. Unclear points in the specifications are removed by this description form. In addition, the computer can process the specification, thus facilitating the designing process.

b) TPA definition in partial order

The FORM system does not need a detailed timing design from system designers. System designers only define the activating order of each TPA related to the transmission data structure. The detailed activating timing is defined automatically by the FORM system, and the definition method is described in the next subsection.

c) System-level simulation

This system allows direct simulation of formal specification without conversion to RTL. If a simulation is performed after converting specification descriptions into RTL, it makes it difficult to associate the results with the specification descriptions one-to-one. Therefore, this system function enables clear associations to be formed between the simulation results and the specification descriptions. The correction of mistakes is facilitated within the specification, and this reduces the turn around time for the system design.

d) RTL generation

To link system designers and existing EDA tools, and verify the behavior of specifications on a real system, RTL is generated from formal specifications. FORM_GEN generates synthesizable RTL by determining the activating timing of each action from the rule described in Section 4.2.

e) Multi-platform

To implement this system as a multi-platform tool. We use Tcl/Tk [17] and GNU C++ as implementation tools. Therefore, this system can be used on various operating systems (OSs) on which Tcl/Tk and GNU C++ are available. Only the source code of the FORM_SIM simulation engine requires compiling when moving it on to other operating systems. FORM_EDIT is constructed using only Tcl/Tk, so it can be used on various OSs without any changes or recompiling where Tcl/Tk is available. Formal specification data can also be used in any FORM system on other various OSs such as Microsoft® Windows®95.

FORM systems aim to always provide the same design environment for system designers even if they have several design platforms with different OSs.

f) Client server system

FORM_SIM comprises a user interface and simulation engine. The user interface and simulation engine can work as cooperative independent programs on different computers. These programs will be able to be carried out on different computers. Our system is built based on an implementation strategy suited to a client-server system.

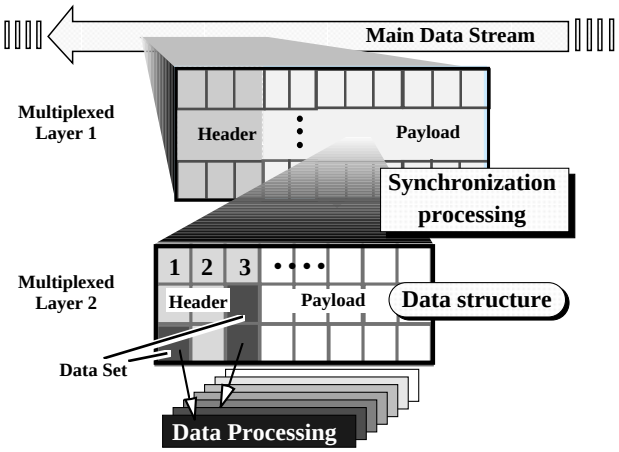


Fig. 1. Transmission Data Structure

3.2 FORM Model

The model used in the FORM system is described below. In this system, we regard the main data stream as transmission data (Fig. 1). The data is transmitted as bit stream data and the transport processing system handled by FORM processes it as eight bit parallel data in one clock cycle. Transmission data have a data format that has a network management/control information part called an overhead or header, a part called a payload that stores the information a user transmits at each layer. Another part called multiplexed structure transmission data is usually stored with similar structured data in the payload. In the header part multiple data sets are defined by the designer. Data sets are a data area where TPA should be performed. Each data set is positioned at a location specified at some distance from the forefront of the transmission data. An ATM cell, as an example of transmission data, has some data sets in the header part such as VPI and VCI.

We divided the processing of each layer into the synchronization processing part and the data processing part (Fig. 2). The synchronization processing part reveals

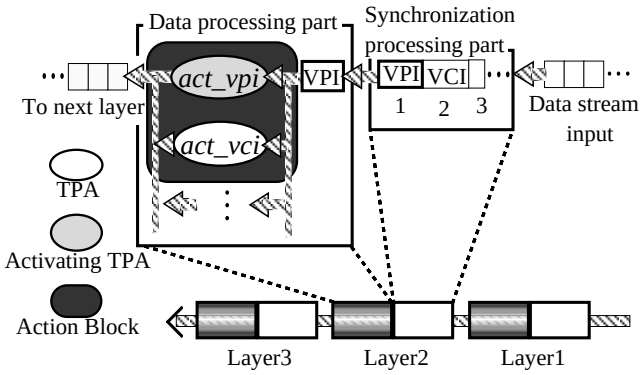


Fig. 2. FORM Model

transmission data from bit stream data. The data processing part contains several TPAs for transmission data, such as the TPA *act_vpi* processes data set VPI. We assume each action is activated by the input of related data sets, such as the TPA *act_vpi* is activated by inputting data set VPI. So the activation timing of actions is implicitly defined through the transmitted data structure. The data processing part comprises multiple TPA modules. Each TPA module processes a specific data set in the transmission data and is activated when the relevant data set is input.

Thus, system designers can design system specifications including timing information without considering the activation timing of each TPA because definitions are created by associating the location of data sets within the transmission data and the TPA modules that processes them. However, some data processes, such as error correction, must be processed first regardless of the position in the transmission data or data relationship. Therefore, system designers must prioritize the processing execution of TPA. The proposed system's grouping prioritizes TPA in the same way into a group called an action block.

3.3 System Configuration

The FORM system comprises the specification editor, the specification simulator, and RTL generator as shown in Fig. 3. The specification editor for preparing a formal description of specifications is called FORM_EDIT. The specification simulator for simulating formal specification descriptions is called FORM_SIM and the RTL generator which generates synthesizable RTL from formal specifications is called FORM_GEN. The user interface of this system is constructed by Tcl/Tk and the processing engine is constructed by GNU C++.

Each tool is used in the following manner. First using

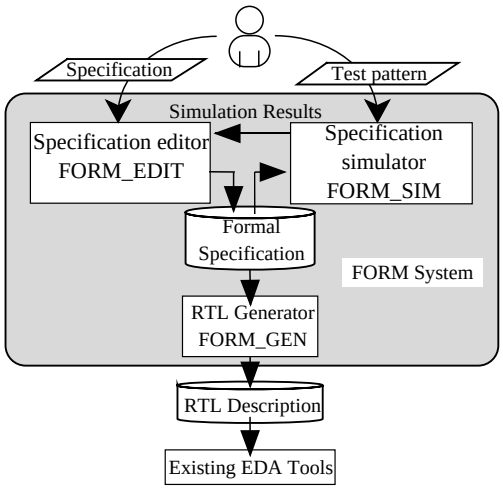


Fig. 3. FORM System Configuration

FORM_EDIT, the system designer defines a multiplexed structure of the transmission data and location of data sets within the transmission data. Then the designer defines the synchronization processing and TPA within the data processing part. After that the specification is verified by simulation. The simulation involves formal specifications which are directly created by FORM_EDIT and the results are displayed in the specification window. The simulation results and formal specification descriptions are tightly coupled. So if some errors are detected, the system designer can quickly correct them. After correcting the mistakes, RTL is generated from formal specifications by the FORM_GEN. A real system can be constructed from the synthesizable RTL with existing EDA tools and the functions of the defined transport system can be adequately verified. This system will allow top-down design from system-level specification for system designers.

4. Specification Editor (FORM_EDIT)

4.1 Configuration

This specification editor comprises the following definitions:

• Multiplexed layers

The layered structure is defined by alternately placing the synchronization processing part and data processing part between the data input and result output.

• Transmission data structure

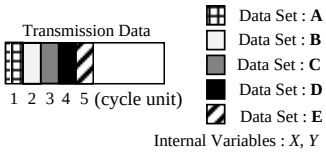
The structure of the transmission data which are processed in each multiplexed layer is defined. Specifically, the total length of the transmission data, header length, and the location and name of data units are defined. Transmissions with variable length data can also be defined.

• Synchronization processing

A method is described for extracting transmission data from the input data or storing transmission data in the transmission data payload in another layer. Because synchronization processing is included as a library in FORM_EDIT, the actual description is completed only by selecting an appropriate synchronization processing module.

• Data processing

Basically system designers write TPAs to specified data sets in the defined transmission data. TPAs are described in accordance with the description format for C Language. The arithmetic operator, bit operation operator, and logic operator in C Language are supported and if-else sentences and while sentences are used as the control structure. For the sake of easy designing, basic data processing such as Cyclic Redundancy Check (CRC) operation and scramble operation are included as libraries. More complicated actions are defined by grouping some TPAs. System designers can also use macro functions in other designs.



Condition Table 1

	B	C	X
1	==0x00	==0x02	NO
2	==0x01	==0x86	ER

Condition Table 2

	B	D	E	Y
1	==0xF0	==0x05	==0x10	YS
2	==0xA1	==0x8F	==0x5F	TE

Condition Table 3

	X	Y	A
1	NO	YS	0x01
2	ER	TE	0x02

Fig. 4. Determination of Processing Timing

The data processing description uses condition tables and a state transition diagram. The activating timing for each TPA in a condition table is evaluated by analyzing the relationship of all variables which are used in descriptions of data processing. So designers can obtain timing constraints for RTL without considering the activation timing of each TPA. Timing determination is explained in detail in the next subsection.

4.2 Determination of Processing Timing

The system processes the functions below for fixing the activation timing of each TPA.

1) Analyzing the relationship of all variables and data set names used in descriptions
2) Determining the order of activation timing of each TPA
3) Inserting a buffering process

An example is described below (Fig. 4).

It is supposed that data set A~E occur in that order. In Condition Table 1, the values of data sets B and C determine value X. Similarly in Condition Table 2, the values of data sets B, D, and E determine variable Y. In Condition Table 3, variable X and variable Y determine the value of data set A. In Table 1 when data set C is input, variable X is determined because data set C is input after data set B. In Table 2 when data set E is input, variable Y is determined because data set E is finally input after other data sets are input. Thus in Table 3, variables X and Y can be determined by inputting data set C and E. Value Y is determined after X because data set E is input after data set C is input. So the value of data set A is determined at the same time as data set E is input. As in the above example, the evaluation timing for each action in a condition table is determined by analyzing the relationship of all variables and data sets.

In the above example the value of data set A is determined by the values of data sets B, C, D, and E that are input after A is input, so buffering is required for the data in data sets A, B, C, D to rewrite the value of data set A. So this system automatically creates a buffering module for four cycle units in an action block when it is needed, as in the

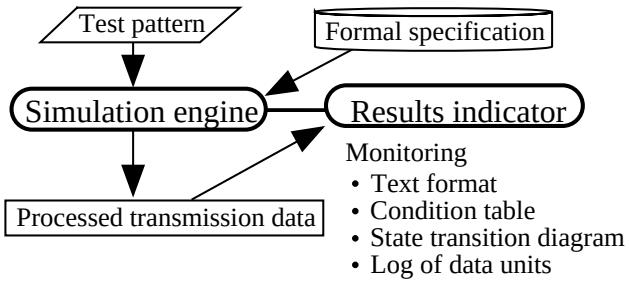


Fig. 5. FORM_SIM Configuration

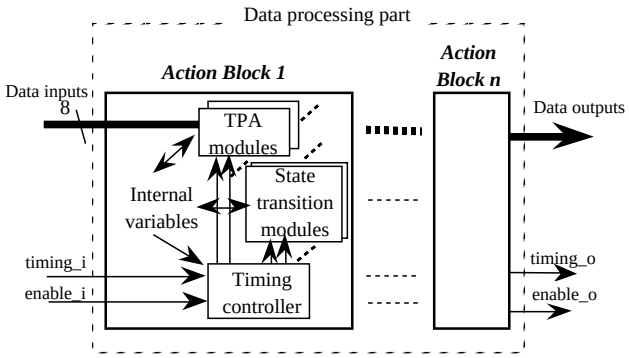


Fig. 6. FORM_GEN Block Diagram

above situation.

5. Specification Simulator (FORM_SIM)

5.1 Configuration

As shown in Fig. 5, the proposed simulation comprises the simulation engine and the results indicator. The simulation engine creates the simulation directly for specification descriptions. The result indicator displays the results output from the simulation engine on the correspondence description of formal specifications. When the simulation is executed, the engine requires formal specifications written using FORM_EDIT and the test pattern data prepared by the user. FORM_SIM reads formal specifications created by FORM_EDIT. Test data is processed based on the specification. As a result of the simulation, processed test data and values of internal variables are given and shown in several ways.

5.2 Simulation Engine

Reading every test data unit (eight-bit parallel data), the simulation engine processes TPA that follow the activation timing information included in the formal specifications. The simulation engine outputs the simulation results and internal data (debugger condition or value of variable) of each TPA that conforms to the result indicator requirements. This simulation engine directly simulates the formal specifications, but does not process line-by-line like a software debugger. In the FORM model each TPA is activated when the relevant transmission data are input. Thus, this simulator exactly reproduces the FORM model action.

5.3 Results indicator

The results indicator displays values of processed data units and values of variables on a specification description. The processing specification description line and the appropriate conditions are dynamically displayed on a condition table used on FORM_EDIT. It also controls the operation of the simulation engine by setting break points in the specification. If an error is detected on the specification during simulation, the corresponding part in the specification can be immediately corrected using FORM_EDIT.

6. RTL Generator (FORM_GEN)

RTL, such as SFL [18] and Verilog-HDL, can be generated from formal specification descriptions created using FORM_EDIT. It predetermines the timing information when each TPA should be performed based on the FORM model as described in Subsection 4.2.

Figure 6 shows the block diagram of the RTL generation. The data processing part of some multiplexed layers comprises several action blocks. The TPA in each block can be performed in parallel. These blocks are connected based on the execution priority order set by the designer. Each variable in the formal specification is converted to control signals in an RTL description by FORM_GEN. Each action block is connected with the same module interface which consists of data and control signals. Data inputs and outputs in the data processing part have an 8-bit parallel width, and timing_i and enable_i are reset and enable signals for the timing controller, respectively. Timing_o and enable_o are reset and enable signals for the timing controller in the next multiplexed layer.

Action blocks comprise TPA modules, state transition modules, and the timing controller. TPA and state transition modules can be directly generated from formal specifications created using FORM_EDIT. The timing controller comprises a counter and decoders that can be automatically generated without designer assistance. Because the time at which each TPA should start can be determined as described in Section 4.2, the timing controller generates timing control signals for TPA and state transition modules by referring to internal variables.

7. Application Example

The results of applying the proposed specification description and simulation environment to the designing of the ATM-AAL5 termination function are shown below. This example is written using FORM_EDIT and checked using FORM_SIM which are implemented on UNIX, Microsoft® Windows®95, and Windows®NT. Their user interfaces and simulation results are completely the same.

The ATM-AAL5 termination function comprises the ATM, AAL5_SAR, and AAL5 layers. The ATM layer corrects errors in the header and rewrites the Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI). The AAL_SAR layer handles packet construction and the AAL5 layer check errors by CRC32 and strips the padding in the payload.

7.1 Specification Definition

A specification definition example is shown below.

1) Multiplexed layer structure definition

The layer structure is defined as synchronization processing and data processing parts for ATM, AAL5_SAR, and AAL5. In this system layers can be easily added or deleted through the GUI.

2) Transmission data structure definition

Table.1 Definition Value of Transmission Data

Transmission data	Total length	Header length	Trailer length
ATM	53	5	0
AAL5_SAR	48	0	0
AAL5	48:63984	0	8

Unit: bytes

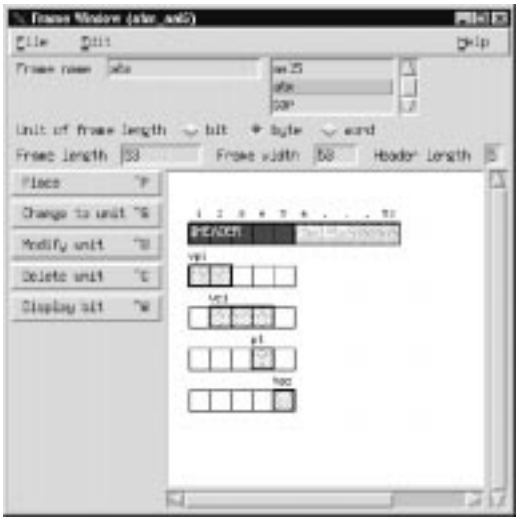


Fig. 7. Definition of Transmission Data Structure

Transmission data structure for ATM, AAL5_SAR, and AAL5 are defined. The settings of the transmission data structure and the actual screen for defining ATM cells are shown (Table 1 and Fig. 7). As shown in Table 1, a value within the range of the frame length comprises several action blocks. The TPA in each block can be performed in parallel. These blocks are connected based on the execution priority order set by the designer. Each variable in the formal specification is converted to control signals in an RTL description by FORM_GEN. Each action block is connected with the same module interface which consists of data and control signals. Data inputs and outputs in the data processing part have an 8-bit parallel width, and timing_i and enable_i are reset and enable signals for the timing controller, respectively. Specification for transmission data and a variable-length payload also becomes available after the total length of the transmission data, header length, and other items are defined. Any desired data area can be selected on the displayed rectangle and its name is defined. It is possible to define information by indicating by the bit, if necessary.

3) Synchronization processing definition

Synchronization processing is the method of extracting and storing transmission data handled in each layer. Specific synchronization processing is prepared as a library. So the actual description is completed only by selecting the desired synchronization processes from the list of functions.

4) Data processing definition

First, the variables to be used in the data processing part are defined. MESSAGE, INT, BIT, and BYTE type variables can be used. MESSAGE type variables are used in control signals and flag signals. In this application example, a BYTE type variable syndrome sends the operation results in the header error check operation module to the errata for error check values. MESSAGE type variables send the synchronization state and error correction state to the state transition diagram as local variables for the ATM layer. Next, a specific data unit is selected. After that processing contents for each data unit are described. At that time, each process is prioritized for execution by the designer.

The rewriting of VPI and VCI is shown below as an example of a condition table (Fig. 8). In this example, the values of the VPI and VCI are rewritten with the values

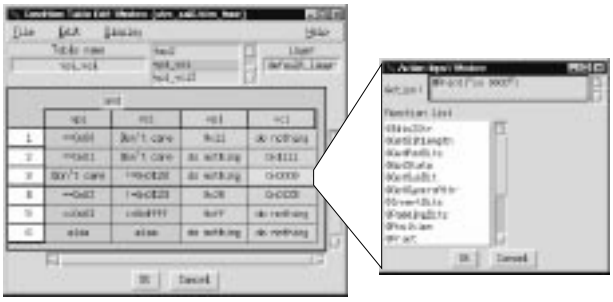


Fig. 8. Specification of Condition Table

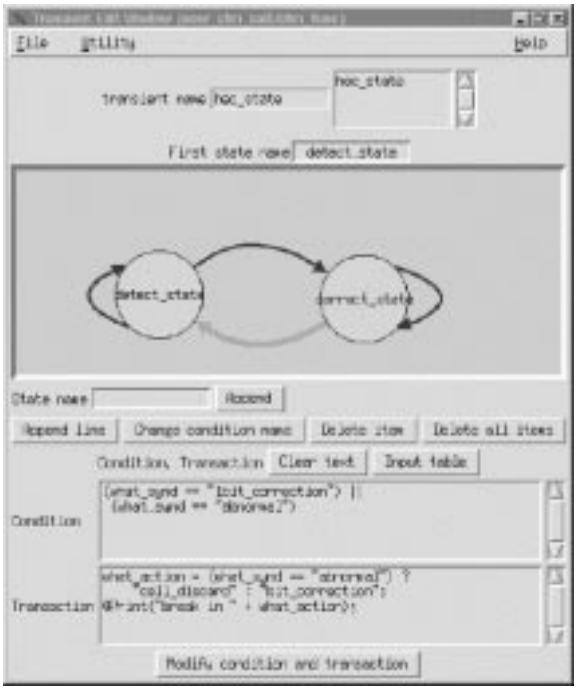


Fig. 9. State Transition Diagram

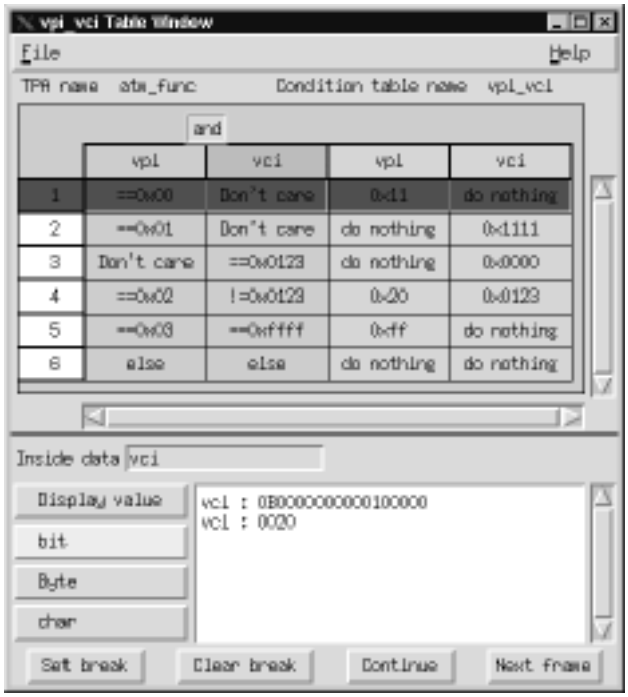


Fig. 10. Monitoring Condition Table

written in the same column depending on the combination of the two values. The cases on the first and second lines are evaluated when the VPI is input because the VCI is "Don't care" for these lines. The cases on the third line and onward are evaluated when input of the VCI is completed because the VPI is "Don't care" or both have values.

For a state transition diagram, the state name is defined first. Next the states of the original and destination points are selected. After that, the transition conditions and the processing to be executed at the transition are described. The transition conditions and processing contents are described in text or in a table format. In this application example, we described the transition between the detection state and the correction state caused by the error correction results for the header part of an ATM cell (Fig. 9). The transition between the detection state and correction state occurs depending on the comparison results of the header error check operation with the errata for error check values.

7.2 Simulation

The specification for the transport processing system comprising the ATM, AAL5_SAR, and AAL5 layers was simulated in this application example. There was a definition error in the specification during the description of the rewriting of the VPI and VCI. It was quickly detected and corrected through observation of the simulation results on the specification description (Fig. 10). The proposed simulator performs various types of monitoring during the simulation. It especially enables observation of results on the specification description interface. As shown in Fig. 10, the simulation step line can be highlighted on the specification description interface. It was thus easy for designers to grasp the simulation status on the step line and to make corrections when an error was detected in a specification description.

We also conducted similar observations on the specification description interface using text and state transition diagrams to confirm whether or not specification descriptions are correct.

7.3 SFL Generation

Three modules, *a_head_c*, *hec_state* ,and *vpi_vci* are generated by FORM_GEN and connected to each other in prioritized order (Fig. 11). Modules *a_head_c* and *vpi_vci*

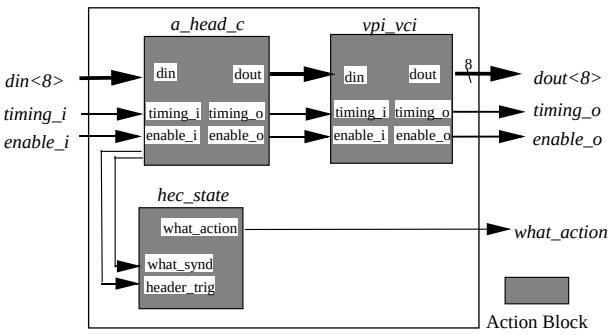


Fig. 11. Block Diagram of Generated Circuit

Table.2 Gate Size of Generated Circuit

Circuit Name	Gate Size (gates)
a_head_c	1333
hec_state	55
vpi_vci	1015

correspond to TPA blocks, which comprise TPA modules and controllers. The controller comprises a six-bit counter and decoder, which are generated by taking into account the timing determination process including the buffering process as described in Subsection 4.2. These modules handle the main bit stream in eight-bit parallel data and are controlled by *timing_i* and *enable_i* signals which are used as counter reset and enable signals. These signals are also generated for the next TPA block module. Module *hec_state* is generated from the state transition diagram. All modules are created as synchronous circuits with a single phase clock. FORM_GEN can generate SFL from which TPA block circuits can be synthesized (Table 2).

8. Conclusion

The FORM system was developed enabling formal specification description of a transport processing system. The specification can be confirmed by direct simulation of the specifications and synthesizable RTL can be generated from it. The characteristics of this system are summed up in the following:

a) Formal specifications of a transport processing system
b) System designing without detailed timing definition
c) Direct simulation of a formal specification
d) Generating RTL from formal specifications on familiar GUIs
e) Multi-platform tool

This system was applied to the functional design of the ATM-AAL5 termination circuit. The simulation results on the specification description were observed and the specification description could be quickly corrected. RTL is generated from the formal specification. In the near future, the performance of generated RTL will be enhanced.

9. Acknowledgement

We gratefully appreciate the suggestion and support of Dr Toshiaki Miyazaki and Akihiro Tsutsui. We would also like to thank the members of High-Speed Network System Research Group of NTT Optical Network Systems Laboratories.

References

[1] K. TOYOSHIMA, et al., “Programmable ATM Adapter: Rapid Prototyping of Call Processing Equipment for ATM Network,” ATM workshop ’97, May 1997.

[2] D. Lewis, D. Galloway, M. van Ierssel, J. Rose, P. Chow, “The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System,” in FPGA ‘97, ACM Symp. on FPGAs, Feb 1997, pp. 53-61.

[3] M. Slimane-Kadi, D. Brasen, G. Saucier, “A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC,” ACM/SIGDA Workshop on Field-Programmable Gate Arrays, 1994.

[4] K. Shirakawa, K. Higuchi, T. Miyazaki, K. Hayashi, and K. Yamada, “FORM: A Frame-Oriented Representation Method for Digital Telecommunication System Design,” ED&TC ’96, pp. 514-520, March 1996.

[5] ITU-T Recommendation I.361:B-ISDN ATM Functional Characteristics, March 1993.

[6] ITU-T Recommendation I.363:B-ISDN ATM Adaptation Layer (AAL) Specification, March 1993.

[7] CCITT: “Functional Specification and Description Language (SDL),” Rec. Z.100, 1992.

[8] ISO, Estelle: A Formal Description Technique Based on an Extended Stated Transition Model, ISO 9074, Jul.1989.

[9] ISO: “Information Processing System - Open System Interconnection -LOTOS - A formal description technique based on the temporal ordering of observational behaviour,” ISO 8807, 1989

[10] A. A. Jerraya and K. O’Brien, “SOLAR: An Intermediate Format for System-Level Design and Specification,” IFIP International Workshop on Hardware/Software Co-Design, Grassau, Germany, Mai 1992.

[11] D. Harel, “Statecharts: A Visual Formalism For Complex Systems,” Science of Computer Programming 8 (1987) 231-274.

[12] M. Donlin, “Graphical-code generators ease path to HDL design,” Computer design, pp. 94-99, Nov. 1996.

[13] N. Kobayashi, K. Wakabayashi, N. Shinohara, H. Tanaka, and T. Kanoh, “Design Experiences with High-Level Synthesis System Cyber-I and Behavioral Description Language BDL,” Proc. Asian PCHDL’94, 5B.2, Oct. 1994.

[14] A. Seawright, U. Holtmann, W. Meyer, B. Pangrle, R. Verbrugghe, and J. Buck, “A System for Compiling and Debugging Structured Data Processing Controllers,” in Proceedings of the European Design Automation Conference 1996, Geneva, Switzerland, September 1996, pp. 86-91

[15] W. Meyer, A. Seawright, and F. Tada, “Design and Synthesis of Array Structured Telecommunication Processing Applications,” DAC ’97, Anaheim, California.

[16] K. Hayashi et al., “Reconfigurable Real-time Signal Transport System Using Custom FPGAs,” Proc. FCCM’95, Napa, USA, Apr. 1995.

[17] SunScript Home Page, World-Wide Web at URL http://sunscript.sun.com/, Sun Microsystems, Inc., 1997

[18] Y. Nakamura, “An integrated logic design environment based on behavioral description,” IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, Vol. CAD-6, No. 3, pp. 322-336 Apr. 1987.