

200 00 17 51 - 7

28 MAR 2000

- 1 -

PROCESSOR WITH BIST FUNCTION

BACKGROUND OF THE INVENTION

The present invention relates to the built-in self-testing (BIST) technology that enables rapid testing and highly accurate fault detection of the logical circuits which contain self-testing functions.

The general test method used to detect faults in logical circuits is by entering pre-created test data externally into the circuit to be tested and observing the resulting output response. However, as logical circuits have increased in scale, the volume of test data to be entered has also increased, which in turn has extended the testing time and increased the costs required for the tests. Accordingly, there is an increasing tendency to reduce testing costs by adopting the BIST technology with which logical circuits are to be self-tested using a built-in self-test circuit.

The prior BIST technology is discussed in, for example, the articles entitled 'A Tutorial on Built-In Self-Test' in the March 1993 Issue, pages 73-82, and the June 1993 Issue, pages 69-77, of the "IEEE DESIGN & TEST OF COMPUTER" magazine.

BIST, as shown in Fig. 2, is composed of a test control section, a pattern generator, and a response analyzer. Output signals from the pattern generator are entered into the circuit to be tested, then the resulting response signals are taken into the response analyzer, and the status thereof is monitored to detect faults. Patterns are generated using one of two major

methods. One method is by storing created test data and a testing program into a ROM or the like beforehand and sending the data and the program to the circuit to be tested. For this method, the volumes of data which can be saved in logical circuits are limited since test data must be saved in a memory. Therefore, although the faults for which the test data is intended can be detected, other faults may not be detectable. The other method is by using a pseudo-random number generator. The use of the pseudo-random number generator enables a great number of pseudo-random numbers to be generated and sent to the circuit to be tested. This method using pseudo-random numbers makes a high fault detection ratio difficult to achieve, since if a sequencing circuit is to be tested, all status changes are difficult to create, and since even if a combination circuit is to be tested, undetectable states may occur. In this case, although a circuit for assigning pseudo-random numbers directly to an internal storage device is usually added to enable the sequencing circuit also to be handled as a combination circuit, the overhead of the added circuit makes rapid circuit operation difficult during tests and/or increases the circuit area. An example of combined use of the aforementioned two methods of pattern generation is disclosed in the Japanese Patent Laid-Open No. 5-120052 (1993). Under this method used to perform tests using the instruction execution function of a microprocessor, a testing program is saved in a memory and the data to be used for the program is created by the pseudo-random number generator. In the circuit composition hereof, input of

a test mode specification signal sets the self-test mode and when the testing instructions prestored within the program memory are executed, the test data to be entered into the test target, namely, an arithmetic logical unit (ALU), is created automatically and pseudo-random numbers are normalized according to the particular status of the status flag provided within the circuit. Thus, great volumes of test data can be supplied for the testing instructions. Even with this method, however, since the testing instructions are saved in the program memory, the number of instruction combinations to be tested is limited and this often makes sufficient tests inexecutable for the logical circuit elements whose operation differs according to the order that the instructions appear.

As shown in the foregoing prior examples, the use of the self-test methods poses the problem that test data saving in a memory or the like makes sufficient testing impossible, and the use of the method adopting only a pseudo-random number generator causes overhead associated with circuit addition, that is, a reduced operating speed and an increased circuit area. Even a combination of the aforementioned two methods, namely, the test method using the instruction execution function of a processor, presents the problem that the number of instruction combinations to be tested is limited. To achieve rapid testing and a high fault detection ratio by adopting BIST, the above-mentioned problems must be solved. An object of the present invention is to provide the means that enables operation to be tested at high speed and a great number of instruction

combinations to be tested.

To ensure that the aforementioned object will be achieved, the processors in the present invention are characterized in that they have a function for random generation of testing instructions, and thus enable operation to be tested rapidly and a large number of instruction combinations to be tested.

More specifically, a processor characterized in that rapid operational testing and the testing of a large number of instruction combinations can be accomplished by providing the processor with: a program counter specifying the instructions to be executed by the processor; an instruction register for storing the instructions specified by the program counter; a register for saving data; an arithmetic logical unit adapted to unload instructions from the instruction register, then to read out data from the data storage register in accordance with the unloaded instructions, and to perform arithmetic operations and store arithmetic result data into the data storage register; and a random-number instruction generator for random generation of testing instructions based on the test signals externally entered.

A processor characterized in that rapid operational testing and the testing of a large number of instruction combinations can also be accomplished by providing the processor with: a program counter for specifying the instructions to be executed; a register file having a plurality of registers capable of storing data; an arithmetic logical unit for unloading data from the register file in accordance with

the instructions specified by the program counter, then performing arithmetic operations, and storing arithmetic result data into the data register; a random-number instruction generator for random generation of testing instructions based on the test signals externally entered; a random-number data output means for sending random numbers to the arithmetic logical unit to perform arithmetic operations; and a response analyzer for storing arithmetic results based on random-number data.

In order to fulfill the above-mentioned object, an information-processing system having a program or data storage memory and a processor connected to said memory for data readout or storage can be implemented, in which case, the processor can accomplish rapid operational testing and the testing of a large number of instruction combinations, by further comprising: an address register capable of storing the addresses for accessing the memory; a data register for saving the data which is to be stored into or has been read out from the memory; an arithmetic logical unit by which the data that has been stored into the multiple registers or data register is operated on in accordance with output instructions from the instruction register; an instruction generator for generating instructions randomly in accordance with external signals and storing the instructions into the instruction register; and a response analyzer provided in the arithmetic logical unit to take in data that has been stored into the data register after being executed in accordance with output instructions from the instruction generator.

In order to fulfill the above-mentioned object, an information-processing system having a program or data storage memory and a processor connected to said memory for data readout or storage can be implemented, in which case, the processor can also accomplish rapid operational testing and the testing of a large number of instruction combinations, by further comprising: an instruction register for output of the instructions to be executed; a plurality of registers for storing data; an address register capable of storing the addresses for accessing the memory; a data register for saving the data which is to be stored into or has been read out from the memory; an arithmetic logical unit by which the data that has been stored into the multiple registers or data register is operated on in accordance with output instructions from the instruction register; an instruction generator for generating instructions randomly in accordance with external signals and storing the instructions into the instruction register; and a response analyzer provided in the arithmetic logical unit to take in data that has been stored into the data register after being executed in accordance with output instructions from the instruction generator.

Also, the random-number instruction generators in the aforementioned processors enable a great number of instruction combinations to be tested by either storing various testing instructions into the instruction register, or generating various testing instructions randomly and creating the output signals necessary to send instruction-compliant data from the

register, or selecting instructions in accordance with preset probability data and sending the instructions, or combining these methods.

In addition, the random-number instruction generators in the aforementioned processors enable a great number of instruction combinations to be tested, by possessing a means of saving the instructions to be executed by the particular processor, and a means of selecting instructions randomly from those saved in the instruction saving means.

Furthermore, the random-number instruction generators in the aforementioned processors enable a great number of instruction combinations can be tested, by possessing:

a partial-instruction saving means for saving part or whole of each instruction to be executed,

an instruction-complementary information saving means for storing the complementary information necessary to enable execution of the partial instructions saved within the partial-instruction saving means,

an instruction selection means for selecting partial instructions and the corresponding instruction-complementary information randomly from the partial-instruction saving means and the instruction-complementary information saving means, respectively, and

a means for complementing random-number data in accordance with the selected partial instructions and instruction-complementary information to create the instructions to be executed by the processor.

Furthermore, when the required test mode signal specifies self-testing, a great number of instruction combinations can be tested by sending data from the random-number data output means, instead of that register within the register file that has been specified by the corresponding instruction to send the data, and taking the data in from the response analyzer, instead of that register within the register file that has been specified by the corresponding instruction to store the data.

Furthermore, when the required test mode signal specifies self-testing, the response analyzers of these processors can implement rapid operational testing by taking in output signals from respective program counters, memory access addresses from the processors, and outputs from respective data registers.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a processor block diagram showing an embodiment of the present invention.

Figure 2 shows the configuration that uses prior BIST technology.

Figure 3 lists instructions of the processor shown in Fig. 1.

Figure 4 shows the instruction formats of the processor of Fig. 1.

Figure 5 is a detailed diagram showing the instruction formats of the processor of Fig. 1.

Figure 6 is a block diagram showing an embodiment of the random-number instruction generator in Fig. 1.

Figure 7 shows the contents of the memory which implements the instruction saving means in Fig. 6.

Figure 8 is a block diagram of the linear feedback shift register (LFSR) which generates pseudo-random numbers.

Figure 9 is a block diagram showing another embodiment of the random-number instruction generator in Fig. 1.

Figure 10 shows the contents of the memory which implements the partial-instruction saving means in Fig. 9.

Figure 11 shows the contents of the memory which implements the instruction-complementary information saving means in Fig. 9.

Figure 12 is a block diagram showing a detailed embodiment of the instruction generator in Fig. 9.

Figure 13 is a block diagram showing an embodiment of the random-number data generators in Fig. 1.

Figure 14 is a block diagram showing an embodiment of the register file in Fig. 1.

Figure 15 is a block diagram showing an embodiment of the sequencer in the present invention.

Figure 16 is a block diagram of the multiplex input signature register (MISR).

Figure 17 is a block diagram showing still another embodiment of the random-number instruction generator in Fig. 1.

Figure 18 is a diagram showing a more specific embodiment of the random-number instruction generator in Fig. 13.

Figure 19 is a diagram showing a more specific embodiment

of the instruction adaptation means in Fig. 18.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the present invention are set forth below.

The LSI configuration including a processor is shown in Fig. 1 as an embodiment of the present invention.

In this figure: 101 is a memory interface adapted to exchange signals with a memory via bus 111; 102 a program counter; 103 an instruction cache; 104 a sequencer for controlling the program counter; 105 an instruction register; 106 a decoder for decoding the contents of the instruction register and controlling an arithmetic logical unit and others; 107 a register file consisting of a random-number data generator for random-number data output and a multiplex input signature register (MISR) as a response analyzer; 108 the foregoing arithmetic logical unit (ALU); 109 a memory data register (MDR) for exchanging data with the foregoing memory; 110 a memory address register (MAR) that specifies addresses to the memory interface; 123 a random-number instruction generator for random output of the instructions required for self-testing; 125, 142, and 143 are random-number data generators for random-number data output; 126, 127, 128, and 144 are used as MISRs for response analysis; 129, 130, 131, 132, and 141 are provided as selectors to change the flow of signals during normal mode and during test mode; likewise, 133 is a test mode signal applied from the exterior of the LSI to specify self-testing; 134 an instruction information signal denoting the contents of output instructions

140 from random-number instruction generator 123.

In addition to receiving instruction information signals 134, the random-number data generators in the present embodiment can change the random-number data generating method so that the generators will develop the random-number data outputs matching the instructions sent from random-number instruction generator 123. Output signals 136, 137, 138, and 139 from MISRs 144, 126, 127, and 128, respectively, are sent to the processor exterior.

The processor instructions set forth in the present embodiment are listed in Fig. 3. All basic instructions are inter-register arithmetic operational instructions. Four types of branch instructions are provided: unconditional branch instructions BRA, conditional branch instructions BRACC ("cc" denotes branching conditions), branch-into-subroutine instructions CALL, and return-from-subroutine instructions RTN.

Load instructions LOAD and store instructions STOR are also provided. In addition, the address spaces specifiable in the branch instructions, load instructions, and store instructions, are expressed in a 24-bit pattern.

Instruction formats are shown in Fig. 4. All instructions have a fixed length of 32 bits. The "f" bit in the basic instruction format specifies whether arithmetic results are to be incorporated into the flag. Likewise, the "S1", "S2", and "D" fields specify the first source field, the second source field, and the address of the destination, respectively. Also,

"d" in the branch instruction formats denote the address of the branch destination.

The detailed instruction formats denoting the definitions of the operation (OP) codes in Fig. 4 are shown in Fig. 5. All bits 0 to 7 in the basic instructions take a value of '0', and all bits 4 to 7 in the branch instructions take a value of '0' and bits 8 to 31 are occupied with the address of the branch destination. Bit 16 in the load or store instruction takes a value of '1', and the value of bit 0 identifies LOAD or STOR.

The flow of processing during the normal mode of the processor in the present embodiment is explained next. During basic-instruction processing in normal mode, first the instruction specified by program counter 102 is read out from instruction cache 103 and then set in instruction register 105. Also, instruction signal 115, and flag signal 116 from the ALU108, are sent to sequencer 104 to conduct the control of program counter 102. Next, instruction-specified register data is transferred from register file 107 through buses 118 and 119 to the ALU108, which then operates on the data. Finally, the ALU operational results that have thus been obtained are passed through bus 120 and stored into the instruction-specified register within register file 107.

During branch instruction processing, first the instruction specified by program counter 102 is read out from and set in the instruction cache. Next, for BRA, the address of the branch destination is sent from instruction signal line 115 to sequencer 104 to conduct the control of program counter

102 through bus 112. For BRACC, the address of the branch destination, and flag signal 116 from the ALU108, are sent from instruction signal line 115 to sequencer 104 to conduct the control of program counter 102. For CALL, the contents of program counter 102 are saved in register 0 of the register file, and the address of the branch destination is sent from instruction signal line 115 to sequencer 104 to conduct the control of program counter 102 through bus 112. For RTN, the contents of register 0 within the register file are set in program counter 102.

During load or store instruction processing, first the instruction specified by program counter 102 is read out from and set in the instruction cache. Next, Instruction signal 115, and flag signal 116 from the ALU108, are sent to sequencer 104 to control program counter 102. Next, the instruction-specified register signal is transferred to the MAR110. For STOR, the register signal specified by the S1 field in the instruction is transferred to the MDR109. Next for STOR, address signal 117 specified by the MAR110 and data signal 121 from the MDR109 are sent to memory interface 101, from which the data is then stored into the memory location of the corresponding address. For LOAD, address signal 117 specified by the MAR110 is sent to memory interface 101, then the memory data in the corresponding address is transferred from memory interface 101 to MDR, and the MDR data is transferred to the register specified by the D field in the instruction.

The self-test operation differing from normal mode in the

present embodiment is explained below. First, test mode signal 133 that specifies self-testing is applied from the exterior of the processor. In accordance with test mode signal 133, output instruction 140 from the random-number instruction generator, not the output instruction from the instruction cache, is transferred to instruction register 105 by selector 141. In accordance with the test mode signal 133, selector 130 selects the MISR127 as the output destination for the MAR110 address signal 117. Similarly, in accordance with the test mode signal 133, selector 131 selects the MISR128 as the output destination for the MDR109 address signal 121. In addition, in accordance with the test mode signal 133, the random-number data output from random-number data generator 125, not data signal 122 from the memory interface, is transferred to the MDR109 by selector 132. Furthermore, in accordance with the test mode signal 133, random-number data generators 142 and 143 send data, instead of that register within register file 107 that has been specified by the instruction to do as such, and then the MISR144 takes the data in, instead of that register within the register file that has been specified by the instruction to do as such.

Next, the flow of processing during the self-test mode of the processor in the present embodiment is explained. First, instruction 140 is generated by random-number generator 123 and transferred to instruction register 105. Also, instruction information signal 134 corresponding to instruction 140 is sent from random-number generator 123. Next if the instruction is

a basic instruction, instruction signal 115, and flag signal 116 from the ALU108, are sent to sequencer 104 to control program counter 102. After that, instead of the instruction-specified register signal, random-number data from random-number generators 142 and 143 is transferred from register file 107 through buses 118 and 119 to the ALU108, where the data is then operated on. Finally, the operational results that have thus been obtained are passed through bus 120 and stored into the internal register of MISR, instead of the instruction-specified register within register file 107. The output signal from program counter 102 is also taken into the MISR126.

During branch instruction processing, in the case of BRA, the address of the branch destination is sent from instruction signal line 115 to sequencer 104 to conduct the control of program counter 102 through bus 112. Next, the output signal from program counter 102 is taken into the MISR126. For BRACC, the address of the branch destination, and flag signal 116 from the ALU108, are sent from instruction signal line 115 to sequencer 104 to control program counter 102. Next, the output signal from program counter 102 is taken into the MISR126. For CALL, the contents of program counter 102 are taken into the internal MISR of register file 107, instead of being saved in register 0 within the register file. Also, the address of the branch destination is sent from instruction signal line 115 to sequencer 104 to control program counter 102 through bus 112. For RTN, random-number data outputs from the random-number data generators within the register file, instead of the contents

of register 0 within the register file, are set in program counter 102. Next, the output signal from program counter 102 is taken into the MISR126.

During load or store instruction processing, instruction signal 115, and flag signal 116 from the ALU108, are sent to sequencer 104 to control program counter 102. Next, the output signals from the random-number generators within register file 107, instead of the instruction-specified register signal, are transferred to the MAR110. For STOR, the output signals from the random-number generators within register file 107, instead of the register signal specified by the S1 field in the instruction, are transferred to the MDR109. Next for STOR, address signal 117 specified by the MAR110 is taken into the MISR128, and output data signal 121 from the MDR109 is also taken into the MISR128. For LOAD, address signal 117 specified by the MAR110 is taken into the MISR128, then the output signals from the random-number data generators, instead of the data of memory interface 101, are transferred to the MDR109, and the MDR109 data is taken into the internal MISR of the register file.

In self-testing mode, the successive processing sequence described above is repeated from output of instruction signals from the random-number instruction generator. Fault detection is accomplished by monitoring MISR output signals 136, 137, 138, and 139 during or at the end of the self-tests and then comparing these signals with the fault-free data which has been calculated beforehand. In the event of mismatching, faults are judged to be occurring. Simulated self-tests on the LSI shown in the

present embodiment can be performed using a logical simulator or fault simulator to calculate the fault-free data required for judgment.

As explained above, the present embodiment takes a configuration that enables a great quantity of instructions to be tested for correct instruction execution results by repeating the successive processes of: generating instructions randomly, supplying instruction execution data from the random-number data generators, taking in from MISR the data of the registers, program counter, MAR, and MDR into which instruction execution results are to be incorporated, monitoring MISR output signals, and comparing these signals with the fault-free MISR output data which has been calculated beforehand.

As shown in the present embodiment, provision of a random instruction generating function enables testing under various combinations of the order in which instructions appear. Also, testing efficiency can be improved since the random-number data matching the instructions executed can be created using the instruction information generated by the random-number instruction generator.

An embodiment of the random-number instruction generator is shown in Fig. 6; wherein the random-number instruction generator is composed of instruction saving means 71, which saves testing instructions, and instruction selection means 72, which selects randomly the instructions that have been saved by the instruction saving means.

Operation is started by input of the test mode signal 133 which specifies self-testing. First, information for selecting which of the instructions that have been saved in instruction saving means 71 is sent as selection signal 73 from instruction selection means 72 to instruction saving means 71, and the information is then sent as instruction information signal 134. Next, instruction saving means 71, after having received selection signal 73, sends the corresponding instruction to 140.

Instruction saving means 71 in the embodiment can be realized by using a memory that receives selection signal 73 as an address signal. An example of the memory is shown in Fig. 7. In this memory, addresses are assigned for every 32 bits and the instructions to be generated during self-testing are stored as 32-bit codes based on the instruction formats of Fig. 5. In the example of Fig. 8, an "n" number of instructions are stored. For example, an "ADD 0, R(1), R(19), R(4)" instruction denoted by the 32-bit code "000000000000000010000011001100100" is stored in address "0000", and a "LOAD R(1), R(4)" instruction denoted by the 32-bit code "100000000000000001000010000000100" is stored in address "n".

Instruction selection means 72 in the embodiment can be implemented by adopting the linear feedback shift register (LFSR) generally used as a pseudo-random number generator. The LFSR has its final-stage output of the shift register fed back to the D-flip-flop provided at the initial stage of the shift register, while at the same time using an exclusive logical sum

to provide feedback to the D-flip-flop located at the middle stage. All bit patterns, except in the case that all bits take a value of '0', can be generated in pseudo-random form by taking a feedback configuration as such. An example of 33-bit LFSR is shown in Fig. 8. In this figure, 91 is a D-flip-flop and 900 to 932 are 33-bit pseudo-random number pattern output signals. In the LFSR, one pseudo-random number can be obtained by performing one shift operation on D-flip-flop 91 and then assign the resulting signal data of the D-flip-flop to pseudo-random number pattern output signals 900 to 932. A plurality of pseudo-random number pattern output signals can be obtained by repeating the above-mentioned shift operation as required. Also, when all bits in initial LFSR data take a value of '0', since pseudo-random numbers are not generated, the flip-flop initialized by a combination of signal data, except in the event of all bits taking a value of '0', when the LSI is powered on or the test mode signal changes to the self-test mode signal, will be used as D-flip-flop 91.

The embodiment in Fig. 6 can be constructed by selecting from the LFSR of Fig. 8 the output signals which consist of the bits necessary to specify the desired instructions stored within the memory, then connecting the corresponding output signal lines as the address signal lines to the memory, and sending the address signals keyed to the memory-stored instructions at the rate of 1:1 as the instruction information signals 134 required for the identification of the instructions issued from the random-number instruction generators. Also,

the probability at which instructions are generated by the random-number instruction generators can be set for each instruction by utilizing the fact that prior storage of one instruction in multiple addresses of the memory increases the probability of the instruction being selected repeatedly by the instruction selection means.

Another embodiment of the random-number instruction generators is shown in Fig. 9. The random-number instruction generators consist of: partial-instruction saving means 1002 for saving part or whole of each testing instruction; instruction-complementary information saving means 1003, which saves the information specifying the complementary portion necessary for the incomplete partial instruction saved in the partial-instruction saving means to become executable; instruction selection means 1001 for selecting randomly the instruction-complementary information corresponding to the partial instructions saved in partial-instruction saving means 1002, and to the incomplete partial instruction saved in the instruction-complementary information saving means; and an instruction generator by which the instruction to be executed by the processor is generated by complementing random-number data based on the selected partial instruction and instruction-complementary information.

Operation is started by input of the test mode signal 135 which specifies self-testing. First, information for selecting which of the partial instructions within instruction saving means 1002 is sent as selection signal 1005 from

instruction selection means 1001 to partial-instruction saving means 1002. At the same time, selection signal 1005 is sent to the instruction-complementary information saving means to select the instruction-complementary information corresponding to the selected information. Selection signal 1005 is also sent as instruction information signal 134 to partial-instruction saving means 1002, from which partial instruction 1006 corresponding to selection signal 1005 is then transferred to instruction generator 1004. In addition, instruction-complementary information 1007 corresponding to the selection signal 1005 is sent from instruction-complementary information saving means 1003 to instruction generator 1004. Finally after receiving partial instruction 1006 and instruction-complementary information 1007, instruction generator 1004 generates an executable instruction by assigning complementary random-number data to the missing part in the partial instruction, and then issues the instruction as instruction 140.

The partial-instruction saving means 1002 and instruction-complementary information saving means 1003 in the embodiment can be realized by using the memories that receive selection signals 1005 as address signals. An example of a memory for implementing the partial-instruction saving means is shown in Fig. 10. In this memory, addresses are assigned for every 32 bits and the instructions to be generated during self-testing are stored as 32-bit codes based on the instruction formats of Fig. 5. At this time, of the entire instruction code

to be saved, only the bits to be complemented by assigning random-number data are set to '0'. In the example of Fig. 10, an "m" number of instructions are stored. For example, a "SUB 0, ?, ?, ?" instruction denoted by the 32-bit code "00000000000000010000000000000000" is stored in address "0001", and a "BRA ?" instruction denoted by the 32-bit code "00100000000000000000000000000000" is stored in address "0004", where "?" in the instructions signifies that field in each instruction code that is to be filled in with random-number data. For example, in the case of address 0001, "?" means that bits 17 to 31 in the instruction code will be filled in with random-number data, and the corresponding bits are all saved as '0s'. An example of a memory for implementing the instruction-complementary information saving means is shown in Fig. 11. In this memory, addresses are assigned for every 32 bits and instruction-complementary information is expressed in a 32-bit pattern. Also, the partial instructions that have been saved in the memory of Fig. 10 and the instruction-complementary information corresponding to each saved partial instruction are stored to match in address. The format of the instruction-complementary information is such that under correspondence to the partial instructions saved in the partial-instruction saving means, the bits in each partial instruction that are to be complemented by assigning random-number data will be set to '1' and all other bits will be set to '0'. In the example of Fig. 11, for address 0001, bits 17 to 31 corresponding to the partial instruction in Fig. 10 take a value of '1' and all other

bits take a value of '0'. Thus, 32-bit code "00000000000000001111111111111111" is stored in the corresponding address. Similarly, for address 0004, bits 8 to 31 denoting the branch destination address of a branch instruction take a value of '1' and bits 0 to 7 take a value of '0'. Thus, 32-bit code "00000000111111111111111111111111" is stored in the corresponding address.

A detailed embodiment of instruction generator 1004 is shown in Fig. 12. The instruction generator consists of an LFSR1301, which starts operating as a 32-bit pseudo-random number pattern generator when test mode signal 133 for specifying self-tests is entered, 32-bit AND calculator 1302, which receives two sets of 32-bit data and then after calculating logical products for each bit, takes out the results, and 32-bit OR calculator 1303, which receives two sets of 32-bit data and then after calculating logical sums for each bit, takes out the results.

During operation, logical products between the bits in instruction-complementary information 1007 given from the instruction-complementary information saving means, and between the bits in the pseudo-random number data given from the LFSR1301, are calculated first and 32-bit data is created as an output by assigning random-number data only to the bits that require complementing. Next, logical sums between the bits in the 32-bit data and partial instruction 1006 given from the partial-instruction saving means are calculated and instruction 140 is created as an output by assigning

random-number data to the bits that require complementing.

Instruction selection means 1001 can be implemented using the LFSR shown in Fig. 8. Output signals equal to the number of bits necessary to specify the contents of both the partial-instruction saving memory and the instruction-complementary information saving memory are selected from the LFSR outputs and the selected signal lines are then connected as address signal lines. The memory that have received the address signals send the corresponding address data as partial instruction 1006 and instruction-complementary information 1007. If the address data is sent as instruction information signal 134, output instructions from the random-number instruction generators can be identified. In the configuration of the above-described embodiment, for improved efficiency, the number of partial instructions stored into the memory should be made equal to the number represented by the exponential power of 2, as with the embodiment of Fig. 6. Instead, however, the LFSR outputs can also be normalized into the "m" number of partial instructions prestored within the memory. Also, the probability of instructions being generated by the random-number instruction generators can be set for each instruction by utilizing the fact that prior storage of one instruction in multiple addresses of the memory increases the probability of the instruction being selected repeatedly by the instruction selection means.

Compared with that of Fig. 6, the embodiment shown in Fig. 9 enables a great number of instruction combinations to be

tested since part of each instruction can be complemented by assigning random-number data to generate instructions.

A detailed embodiment of the random-number data generators used in the embodiment of Fig. 1 is shown in Fig. 13. These random-number data generators consist of: 32-bit random-number generators 1401 to 1403; selector 1404 for selecting one set of output data from the random-number generators and then sending random-number data 1406; and decoder 1405 which can receive instruction information signal 134, then judge whether the random-number generators are to send random-number data, and if the data is to be sent, send to the selector not only the information which specifies the random-number generator output corresponding to the instruction information signal, but also the signal which specifies the operation to be performed by the random-number generator specified by the corresponding information.

Random-number generators 1401 to 1403 send the data of the data formats necessary to enable the execution of the test items assumed for the instructions identified by the instruction information corresponding to instruction information 134.

During processing, first when instruction information signal 134 from random-number generator 123 is entered, decoder 1405 will judge whether the random-number data generator is to send random-number data, and if the data is to be sent, the signal that specifies the random-number generator sending the random-number data corresponding to the instruction information will be sent to selector 1404 and the decoder will

specify random-number data output operation to one of the random-number generators 1401-1403 which send the random-number data corresponding to the instruction information. Next, output random-number data from the random-number generator which has been specified from decoder 1405 will be sent as the random-number data that matches instruction information signal 134. In the case of the random-number generators in the present embodiment, random-number data of the data formats necessary to enable the execution of the test items assumed for the instructions identified by the instruction information corresponding to instruction information 134 can be sent as the random-number data matching the corresponding instruction.

The random-number generator for sending the random-number data matching an instruction information signal can be easily realized by using the combination circuit which uses LFSR and its output as an input. For example, if a part with fixed bit data is required as the data part for testing an instruction of "a", this can be performed by adding an LFSR output masking circuit. The load instruction in the embodiment of Fig. 1 is shown as a specific example. LOAD R(S1), R(D) is the load instruction by which the memory data addressed by the register data specified by S1 will be transferred to the register specified by D. In the case of this instruction, when, instead of the register data specified by S1, output data from a random-number generator is used, since the length of address spaces is limited to 24 bits, the random-number generator output

data must also be expressed in a 24-bit pattern. The random-number generator for sending the random-number data matching this load instruction is shown in Fig. 18. This random-number generator is composed of an LFSR1902, which sends 32-bit pseudo-random numbers, and combination logical circuit 1903, which performs 8-bit logical multiplications.

During processing, first when operating instruction signal 1901 from decoder 1405 is entered, LFSR will send a 32-bit pseudo-random number pattern. Next, the higher-order eight-bit signals in the 32-bit pseudo-random number pattern will be sent to logical multiplication circuit 1903, which will then perform logical multiplications on the signals whose eight bits are all '0s' and thus send the data 1905 whose eight bits all take a value of '0'. Finally, data consisting of a combination of data 1905 and the lower-order 24-bit data 1904 in the 32-bit pseudo-random number pattern, will be sent as the 32-bit data 1906 matching the load instruction.

Also, when data that has been obtained by adjusting the probability of '0s' and '1s' being generated for each bit is required as the data conforming to instruction "b" of the data format necessary to execute the test items assumed for this instruction, a random-number generator can be constructed by designing a combination circuit utilizing the fact that: when the logical sum between multiple output bits of the LFSR is taken as its output, the generation probability of '1s' will increase, and when the logical product is taken as its output, the generation probability of '0s' will increase.

In the case that a set of bit patterns are provided as the data matching an instruction of "c", when a pattern that has been selected randomly from the pattern set is to be used as the data matching the "c" instruction, this can be implemented by using LFSR and the memory in which the bit patterns have been saved, and then using the output of the LSFR to generate memory data readout addresses.

Another embodiment of the random-number data generators used in Fig. 1 is shown in Fig. 17.

These random-number data generators consist of: decoder 1805 which can receive instruction information signal 134, then judge whether the random-number generators are to send random-number data, and if the data is to be sent, not only send the signal which specifies random-number data output operation to an LFSR1807, but also send information 1808 which selects the random-number data matching the instruction information; the above-mentioned LFSR1807 for output of 32-bit pseudo-random number data; instruction adaptation means 1801 to 1803 for sending the LFSR output 32-bit data to selector 1804 after converting the data into the random-number data adapted to the instruction identified by instruction information signal 134; and the above-mentioned selector 1804 by which the random-number data adapted to the instruction information is selected from the outputs of the instruction adaptation means in accordance with selection information 1808 and then sent to the next stage.

During processing, first when instruction information

signal 134 from random-number generator 123 is entered, decoder 1805 will judge whether the random-number data generator is to send random-number data, and if the data is to be sent, information that specifies the instruction adaptation means sending the random-number data corresponding to the instruction information will be sent to selector 1804 and the decoder will send pseudo-random number data output instruction signal 1809 to the LFSR1807. Next, the LFSR1807 that has received the instruction will send pseudo-random number data to instruction adaptation means 1801 to 1803. The instruction adaptation means, after having obtained the pseudo-random number data from the LFSR, will convert the data into the random-number data adapted to the instruction and send the random-number data to selector 1804. At this selector, the pseudo-random number data adapted to instruction information signal 134 will then be selected from the output pseudo-random number data of the instruction adaptation means in accordance with selection information 1808 and sent to the next stage as the random-number data 1806 adapted to the instruction identified by instruction information signal 134.

The load instruction of the processor in the embodiment of Fig. 1 is shown as a specific example of the instruction adaptation means in the present embodiment. LOAD R(S1), R(D) is the load instruction by which the memory data addressed by the register data specified by S1 will be transferred to the register specified by D. In the case of this instruction, when, instead of the register data specified by S1, output data from

a random-number generator is used to change the corresponding address randomly and conduct the instruction execution test, since the length of address spaces is limited to 24 bits, the random-number generator output data must also be expressed in a 24-bit pattern. The instruction adaptation means for converting the LFSR1807 output pseudo-random number data into the format matching the load instruction is shown in Fig. 19. The instruction adaptation means is composed of combination logical circuit 2001, which performs 8-bit logical multiplications, and through-circuit 2002, which sends 24-bit signals intact without performing any modifications.

A detailed embodiment of the register file in the embodiment of Fig. 1 is shown in Fig. 14. This register file consists of: 32 registers having two reading ports and one writing port and saving data; output selectors 1500 and 1501, which change the test mode signal to a random-number data generator output signal, instead of a register output signal, and send the random-number data generator output signal to a data bus; an MISR1505 taking in 32-bit data as a response analyzer; and an input selector by which the register write data entered from bus 120 will be sent to the above MISR via the test mode signal.

In normal operation mode, the register write data entered from bus 120 is stored into the register specified by an instruction. Also, the instruction-selected register data is sent to buses 118 and 119. In self-testing mode, the register write data entered from bus 120 is sent to the MISR1505 by the input selector and taken into the MISR. Also, the

instruction-selected register data is changed into the output data of random-number data generators 1503 and 1504, from which the data is then sent to buses 118 and 119.

An example of embodiment of the configuration where, instead of one or more storage devices within sequencer 104, the random-number data output means sends signals in the case that in the embodiment of Fig. 1, the test mode signal specifies the self-testing mode, is shown in Fig. 15. The sequencer is a sequencing circuit that consists of combination circuit block 1601 and an "i" number of storage devices 1604 to 1606. When test mode signal 133 specifies self-testing, selectors 1602 and 1603 send the output signals 1607 and 1608 of the random-number data generator 1609, instead of storage devices 1605 and 1606. Output data from the random-number data generator can be obtained by assigning two bits from the output of the LFSR shown in Fig. 8. In the present embodiment, although the two storage devices are subject to selection by the selector, the signal output configuration is not restricted hereby and can include one or more storage devices. Introduction of the configuration shown in the present embodiment enables testing with diversified program counter data, even if the random-number instruction generators have generated non-branch instructions.

The configuration of the MISR in the embodiment of Fig. 1 is shown in Fig. 16. The MISR is generally used as a response analyzer in the BIST function block, and the response analyzer is configured so that exclusive logical sums can be used to apply

signals to each bit of the LFSR. In Fig. 16, 17 is a D-flip-flop and 1700 to 1732 are data input signal lines taking a 33-bit pattern. The signals required for data acquisition are coupled to the data input signal lines. Data acquisition is accomplished by taking into each D-flip-flop the exclusive logical sums between the D-flip-flop outputs and the data to be acquired during shifting. Shifting for data acquisition is repeated to compress acquired data.

According to the present invention, testing under a great number of combinations of instruction generation sequences is possible by providing the function that generates testing instructions randomly. Also, since the random-number data matching the instructions executed can be created using the instruction information generated by the random-number instruction generator, testing efficiency can be improved and thus a high fault detection ratio can be achieved. In addition, rapid testing is possible by adopting hardware as the random-number instruction generator and the random-number data output means.

According to the present invention, a high fault detection ratio can be achieved and rapid testing is possible.



WHAT IS CLAIMED IS:

1. A processor comprising:

a program counter for specifying the instructions to be executed,

an instruction register capable of storing the instructions specified by said program counter,

a register into which data is to be stored,

an arithmetic logical unit adapted to unload instructions from said instruction register, then to read out data from said data storage register in accordance with the unloaded instructions, and to perform arithmetic operations and store arithmetic result data into the data storage register, and

a random-number instruction generator for random generation of testing instructions based on the test signals externally entered.

2. A processor according to in Claim 1 above, wherein said random-number instruction generator generates testing instructions in accordance with said test signals and stores the testing instructions into said instruction register.

3. A processor according to in Claim 1 above, wherein said random-number instruction generator generates testing instructions randomly and also generates the output signals for sending appropriate data from said data storage register in accordance with the particular testing instruction.



- 34 -

4. A processor according to in Claims 1 to 3 above, wherein said random-number instruction generator possesses a plurality of instructions and sends said instructions based on the conditions established beforehand.

5. A processor comprising:

a program counter for specifying the instructions to be executed,

a register file having a plurality of registers for storing data,

an arithmetic logical unit for unloading data from said register file in accordance with the instructions specified by said program counter, then performing arithmetic operations, and storing arithmetic result data into the data register,

a random-number instruction generator for random generation of testing instructions based on the test signals externally entered,

a random-number data output means for sending random numbers to said arithmetic logical unit to perform arithmetic operations, and

a response analyzer capable of storing arithmetic results based on said random-number data are to be stored.

6. A processor according to in Claim 5 above, wherein said response analyzer fetches output signals from the program counter in accordance with the test signals.



- 35 -

7. A processor according to in either of Claims 1 to 6 above, wherein said random-number instruction generator has a means for saving the instructions that the processor executes, and a means for selecting instructions randomly from said instruction saving means.

8. A processor according to in either of Claims 1 to 7 above, wherein said random-number instruction generator further comprises:

a partial-instruction saving means for saving part or whole of each instruction to be executed,

an instruction-complementary information saving means for storing the complementary information necessary to enable execution of the partial instructions saved within said partial-instruction saving means,

an instruction selection means for selecting partial instructions and the corresponding instruction-complementary information randomly from said partial-instruction saving means and said instruction-complementary information saving means, respectively, and

a means for complementing random-number data in accordance with the selected partial instructions and instruction-complementary information to create the instructions to be executed by the processor.

9. An information-processing system having a program or data storage memory and a processor connected to said memory for data



- 36 -

readout or storage; wherein said processor further comprises
an instruction register for output of the instructions to
be executed,

a plurality of registers for storing data,

an address register capable of storing the addresses for
accessing the memory,

a data register for saving the data which is to be stored
into or has been read out from the memory,

an arithmetic logical unit by which the data that has been
stored into said multiple registers or data register is operated
on in accordance with output instructions from the instruction
register,

an instruction generator for generating instructions
randomly in accordance with external signals and saving the
instructions in the instruction register, and

a response analyzer provided in the arithmetic logical unit
to take in data that has been stored into the data register after
being executed in accordance with output instructions from said
instruction generator.