〈Technical Note〉

DEVELOPMENT OF RPS TRIP LOGIC BASED ON PLD TECHNOLOGY

JONG GYUN CHOI* and DONG YOUNG LEE

Korea Atomic Energy Research Institute

1045 Daedeok-daero, Yuseong-gu, Daejeon, Republic of Korea 305-353

*Corresponding author. E-mail : choijg@kaeri.re.kr

*Received January 18, 2011**Accepted for Publication August 25, 2011*

The majority of instrumentation and control (I&C) systems in today's nuclear power plants (NPPs) are based on analog technology. Thus, most existing I&C systems now face obsolescence problems. Existing NPPs have difficulty in repairing and replacing devices and boards during maintenance because manufacturers no longer produce the analog devices and boards used in the implemented I&C systems. Therefore, existing NPPs are replacing the obsolete analog I&C systems with advanced digital systems. New NPPs are also adopting digital I&C systems because the economic efficiencies and usability of the systems are higher than the analog I&C systems.

Digital I&C systems are based on two technologies: a microprocessor based system in which software programs manage the required functions and a programmable logic device (PLD) based system in which programmable logic devices, such as field programmable gate arrays, manage the required functions. PLD based systems provide higher levels of performance compared with microprocessor based systems because PLD systems can process the data in parallel while microprocessor based systems process the data sequentially.

In this research, a bistable trip logic in a reactor protection system (RPS) was developed using very high speed integrated circuits hardware description language (VHDL), which is a hardware description language used in electronic design to describe the behavior of the digital system. Functional verifications were also performed in order to verify that the bistable trip logic was designed correctly and satisfied the required specifications. For the functional verification, a random testing technique was adopted to generate test inputs for the bistable trip logic.

KEYWORDS : Programmable Logic Controller, Digital Safety System, Programmable Logic Device, Field Programmable Gate Array

1. INTRODUCTION

Before the 1970s, the instrumentation and control (I&C) functions in nuclear power plants (NPPs) were developed by assembling elementary logic and analog components such as printed circuits, relays, transistors, resistors, and solid state devices. The system based on analog technology is easily understandable and testable, while also being very easy to maintain because only failed components must be replaced. However, it is very difficult to implement complex functions in systems using analog technology and the system should be redesigned to add new functionalities. In addition to this, in recent times, manufacturers have become reluctant to supply analog devices and boards that are used in I&C systems because the cost of designing and maintaining analog devices and boards constantly increases although the system that uses analog technology still only has limited functionality [1].

In the 1980s, the I&C functions in NPPs were imple-

mented using software in microprocessor-based digital systems. The programmable logic controller (PLC) was used for the reactor protection system (RPS) in the late 1980s, and it was reliable and technically proven in many industrial fields. Westinghouse developed a digital protection system, named the Q series. This was a hybrid protection system that is composed of analog and digital circuits. Westinghouse also developed a digital I&C system package, the Eagle Family 21. The Eagle Family 21 was applied to the Sizewell B NPP. AECL developed a programmable digital comparator (PDC) and applied it to the CANDU-6 NPP. Siemens developed a protection system, named Teleperm XS, and it was applied to the Parks NPP in Hungary. In South Korea, the safety grade PLC named POSAFE-Q was developed through the Korea Nuclear Instrumentation and Control System (KNICS) project [2].

Recently, there have been several attempts to implement I&C functions in NPPs using PLD technology, be-

cause PLD technology has been widely used by aerospace, military, and other highly safety critical applications [3-5]. RPC “Raidy” in the Ukraine developed the FPGA based I&C platform for safety applications in NPPs [6]; also, CS Innovation in the USA developed an advanced logic system (ALS) platform that is a FPGA based safety system. The Wolf Creek Nuclear Operating Corporation adopted the ALS platform for use in the primary steam and feed water isolation system in 2009 [7]. The FPGA implementation of one channel in a CANDU shutdown system number was performed in Canada. Furthermore, the implemented system was validated in a hardware-in-the-loop system using a training simulator [8].

The PLD-based system offers many benefits over microprocessor-based systems. Only the necessary I&C functions can be directly implemented in the PLD component without complex system software, which makes the qualification effort easier. It also provides higher performance compared with microprocessor based systems because it processes the data in parallel while microprocessor based systems process the data sequentially. Furthermore, obsolescence can be mitigated using the portable register transfer level (RTL) design which supports targeting new technologies.

In this research, the bistable trip logic in the RPS was developed using very high speed integrated circuits hardware description language (VHDL) for the application of the PLD technology in safety I&C systems in NPPs. VHDL is a hardware description language used in electronic design to describe the behavior of the digital system. The two primary applications of VHDL are in the field of programmable logic devices (CPLD and FPGA) and in the field of ASICs. The bistable trip logic has been designed in the RTL using a synchronous finite state machine modeling technique. Then, the bistable trip logic was synthesized for implementation into the FPGA. The functional verification was also performed in order to

verify that the bistable trip logic has been designed correctly and that it satisfies the specified requirements. For functional verification, the random testing technique was adopted to locate bugs in the developed bistable trip logic. During the random test of the bistable trip logic, various code coverage metrics were assessed to verify that the bistable trip logic was sufficiently exercised. These metrics include the statement, branch, condition, toggle, and finite state machine coverage.

2. MODEL OF FINITE STATE MACHINES

2.1 Finite State Machine

A finite state machine (FSM) is a mathematical abstraction model used to design digital logic. It is a behavior model composed of a finite number of states, transitions between those states, and actions. The operation of an FSM begins from one of the states (called the initial state), goes through the transitions depending on the input to different states and can end in any available state. In a digital circuit, an FSM can be built using logic gates and flip flops within a programmable logic device. More specifically, a hardware implementation requires a sequential logic block to store state variables and determine the state transition, and a second combinational logic block that determines the output of the FSM as shown in Figure 1 [9]. The combinational logic has two inputs: one being the present state (*pr_state*) and the other the external input. It also has two outputs, the next state (*nx_state*) and the external output. The sequential logic has three inputs (clock, reset, and *nx_state*) and one output (*pr_state*). The output of the combinational logic depends solely on the current inputs. The output of the sequential logic depends not only on the current inputs but also on the previous state.

Fig. 1. Finite State Machine and its Hardware Implementation.

A state transition diagram is an abstract diagrammatic representation of a finite state machine as shown in Figure 2. The classic form of the state transition diagram is a directed graph that includes states, inputs, and outputs [10]. It uses a circle to represent each state. The directed arc between states represents the transitions from one state to another. An arc is labeled with a combination of input values that allow the transition to occur. The outputs are denoted in the circle. When there is no transition condition on the arc, it indicates an unconditional transition that causes the transition to the next state regardless of the input values. In a state transition diagram, all possible combinations of the input values should be accounted for in each state, and no combination should be repeated on more than one arc from a given state.

2.2 VHDL and Code Template

A hardware description language (HDL) is a programming language for the formal description of electronic circuits and, more specially, digital logic. It can describe the circuit's operation, its design, and its organization; it also performs tests to verify its operation through simulation. HDL is analogous to software programming languages, but it has one significant difference. Many programming languages are inherently procedural with limited syntactical and semantic support to handle concurrency; however, HDL is a concurrent programming language in its ability to model multiple parallel processes that automatically execute independent of each other [11].

The two most widely used HDLs are VHDL and Verilog. Once the code has been written in VHDL or Verilog, it can be used to implement the circuit in a programmable logic device (PLD) such as CPLD and

FPGA using the following steps: (1) Write the HDL code, (2) Synthesis, (3) Translation, (4) Mapping, (5) Place and route, (6) Bit stream generation, and (7) Configuration. The synthesis in Step (2) is the process of converting the code into a gate level netlist using a synthesis tool. In Step (3), the translation is the process that prepares the synthesized design for use within the PLD. The result of this step is stored in a file in a tool specific binary format that describes the logic design in terms of design primitives such as logic gates, latches, and flip flops. The mapping in Step (4) is the process that assigns the design primitives to the specific physical elements in the PLD. During the place and route step, the design blocks created during the mapping process are assigned a specific location in the PLD and interconnected with each other. The output obtained from the place and route is converted into a bit file and downloaded to the target device.

From a VHDL perspective, the combinational logic and sequential logic in a finite state machine can be written with a **PROCESS** that is the most general construct in the VHDL for describing the behavior of the hardware. It is characterized by the presence of **IF**, **WAIT**, **CASE**, and **LOOP** commands, and by a sensitivity list containing all inputs of the logic block. The statements in the **PROCESS** block are executed whenever the value of any signal in the sensitivity list changes.

Figure 3 shows an example of the code template written in VHDL for implementation in the FSM. The reset and clock signal appear in the sensitivity list of the **PROCESS** block to describe the sequential logic. When reset is asserted, the *pr_state* will be set to the logic's initial state. Otherwise, at the appropriate clock edge, the flip flops will store the *nx_state*, thus transferring it to the

Fig. 2. State Transition Diagram.

```

ENTITY <ent_name> IS
  PORT ( input:      IN    <data_type> ;
         reset, clock: IN    STD_LOGIC ;
         output:     OUT   <data_type>);
END <ent_name>;

ARCHITECTURE <arch_name> OF <ent_name> IS
  TYPE states IS (state0, state1, state2, state3, ...);
  SIGNAL pr_state, nx_state: states ;
  SIGNAL temp: <data_type> ;
BEGIN
  ----- Sequential logic -----
  PROCESS (reset, clock)
  BEGIN
    IF (reset = '1') THEN
      pr_state <= state0 ;
    ELSIF (clock'EVENT AND clock = '1') THEN
      out_put <= temp ;
      pr_state <= nx_state ;
    END IF;
  END PROCESS ;

  ----- Combinational logic -----
  PROCESS (pr_state)
  BEGIN
    CASE pr_state IS
      WHEN state0 =>
        Temp <= <value> ;
        IF (condition) THEN nx_state <= state1 ;
        ...
      END IF ;
      WHEN state1 =>
        Temp <= <value> ;
        IF (condition) THEN nx_state <= state2 ;
        ...
      END IF;
      ...
    END CASE ;
  END PROCESS ;

  -----
END <arch_name> ;

```

Fig. 3. VHDL Code Template of a Finite State Machine.

PROCESS block of the combinational logic. The **PROCESS** block for the combinational logic assigns the output value and determines the next state of the logic depending on the external inputs and present state [9].

2.3 Fractional Number Representation

While many applications deal with integer data, there are increasing numbers of applications that must also consider fractional numeric data. Many such applications involve digital signal processing in which time-varying analog signals are sampled from the field, converted to a digital representation, and subject to numerical operations. For example, the bistable trip logic in the RPS continuously monitors the analog signals from the field and generates a trip initiation signal whenever the monitored values reach predetermined values. Therefore, bistable trip logic must consider the fractional numeric data.

Unfortunately, VHDL provides a real number data

type for simulations only, not for synthesis. This means that the logic design requires a method for the fractional numeric data and its arithmetic operation. There are two methods for this: fixed point and floating point. The fixed point method is often used for computations because the floating point method significantly increases the design size and complexity. In this work, the fixed point representation method has been adopted for fractional numbers and their arithmetic.

In the fixed point method, the fractional numbers can be approximated using a pair of integers (n , e): the mantissa and the exponent. The pair represents the fractional number as follows [12]:

$$a = n2^{-e}.$$

The exponent e can be considered as the number of digits that must be moved into n in order to place the binary point. For example, when the mantissa n is an 8-bit un-

Fig. 4. Fixed Setpoint Trip Configuration.

signed binary number, 01100100, and the exponent is 3, the fractional number is given by:

$$01100.100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} = 12.5_{10}.$$

The addition and subtraction of the two fixed point numbers, $a = n2^{-r}$ and $b = m2^{-r}$, are given by:

$$\begin{aligned} a + b &= n2^{-r} + m2^{-r} = (n + m)2^{-r}, \\ a - b &= n2^{-r} - m2^{-r} = (n - m)2^{-r}. \end{aligned}$$

3. DEVELOPMENT OF BISTABLE TRIP LOGIC IN RPS WITH VHDL

The RPS protects the reactor core and pressurized coolant system by continuously monitoring selected system parameters and initiating protective action if any input parameter reaches a predetermined unsafe level. The system is designed to prevent accidents. Each parameter is monitored using four independent RPS channels designated A through D. The same parameter must indicate an unsafe condition on at least two of the four channels before the RPS initiates the protective action [13].

The RPS consists of measurement channels, bistable logics, coincidence logics, and the circuits required to ensure a rapid and reliable reactor trip if the same parameter in two or more channels reaches a predefined limiting safety system setting. The measurement channels consist of sensors and signal conditioning equipment whose purpose is to convert the parameters being measured into signals usable to bistables. The signals provided directly to the bistables from the measurement channels are in the form of analog voltages. The bistables compare these analog voltages to the trip levels and generate trip signals. Pretrip alarms are initiated below the trip setpoint value in order to provide audible and visual indications of the approaching trip condition.

3.1 Types of Bistable Trip Setpoint

There are three types of trip setpoint: the fixed setpoint, the rate limited variable setpoint, and the directional variable setpoint. With the fixed setpoint rising trip configuration as shown in Figure 4, the process input signal is normally less than the trip setpoint. As the process input signal rises through the pretrip or trip setpoint, the bistable generates the pretrip or trip signal. When the bistable is tripped, it decreases the trip setpoint by the amount of the hysteresis. When the bistable is untripped, it restores the trip setpoint. With the fixed setpoint falling trip configuration, the process input signal is normally greater than the pretrip or trip setpoint. As the process input signal decreases through the pretrip or trip setpoint, the bistable generates the pretrip or trip signal. When the bistable is tripped, it increases the trip setpoint by the amount of the hysteresis; when the bistable is untripped, it restores the trip setpoint.

The bistable operation with a rate limited variable setpoint (VSP) is identical to that of the bistable with a fixed setpoint. The bistable generates a pretrip or trip signal when the process input signal reaches the level of the trip or pretrip setpoint that exists at that time. However, the setpoint value will change depending on the rising or falling of the process input signal as shown in Figure 5. The trip setpoint will track the process input signal using the difference of the step adjustment value if the process input signal changes at a rate that is less than the maximum preset rate. The trip setpoint rises or falls at a fixed maximum preset rate when the process input signal changes with a rate greater than the maximum preset rate.

The bistable operation with a directional VSP falling trip configuration is identical to that of the bistable operation with a fixed setpoint. The bistable will generate a pretrip or trip signal when the process input signal reaches the level of the trip or pretrip setpoint that exists at that time. However, the setpoint value will change depending

Fig. 5. Rate Limited Variable Setpoint Configuration.

Fig. 6. Directional Variable Setpoint Trip Configuration.

on the process input signal and external manual reset as shown in Figure 6. The trip setpoint will track the process input signal using the difference of the step adjustment value if the process input signal rises away from the trip. When the process input signal falls toward the trip, the setpoint will be fixed, but can be decreased by the amount of the preset step adjustment when the external manual setpoint reset signal is asserted.

3.2 State Transition Diagram for the Bistable Trip Logic

The state transition diagram representing the fixed setpoint trip logic is simple and has two states: **UNTRIP** and **TRIP**. As shown in Figure 7, when the reset signal is asserted, the logic will be set to the initial **UNTRIP** state.

The state of the logic with a rising trip is the **TRIP** state in the next clock cycle if the process input, PI , is greater than or equal to the preset trip setpoint, T_{SP} .

The logic will transit to the **UNTRIP** state when the logic is in the **TRIP** state and the process input is less than the preset trip setpoint minus the preset trip hysteresis, $T_{SP} - T_{HYS}$.

The logic state with a falling trip will transit to the **TRIP** state in the next clock cycle if the process input is less than or equal to the preset trip setpoint, T_{SP} . The logic will transit to the **UNTRIP** state when the logic is in the **TRIP** state and the process input is greater than the preset trip setpoint plus the preset trip hysteresis, $T_{SP} + T_{HYS}$.

As shown in Figure 8, the bistable logic with the rate limited VSP trip can be designed with two logic blocks: a setpoint calculation and a trip decision logic. The state transition diagram of the trip decision logic is identical to that of the fixed setpoint trip logic as shown in Figure 7. The state transition diagram of the setpoint calculation logic is shown in Figure 9.

Fig. 7. State Transition Diagram of the Fixed Setpoint Trip.

Fig. 8. Block Diagram of the Rate Limited VSP Trip Logic and Directional VSP Trip Logic.

Fig. 9. State Transition Diagram of the Setpoint Calculation Logic for Rate Limited VSP Trip.

The setpoint calculation logic for a rising trip has four states: **FOLLOW**, **MAX RATE**, **MIN POINT**, and **HOLD**. The **FOLLOW** state is the state in which the setpoint tracks the process input using the difference of

the amount of the step adjustment, $STEP_ADJ$, when the process input is greater than the minimum preset setpoint minus the step adjustment, $MIN_T_SP - STEP_ADJ$, and the slope of the process input is less than the maximum

Fig. 10. State Transition Diagram of the Setpoint Calculation Logic for a Directional VSP Falling Trip.

preset rate, MAX_RATE . The **MAX RATE** state is the state in which the setpoint rises at a maximum preset rate when the process input changes at a rate greater than the maximum preset rate. The **MIN POINT** state is the state in which the setpoint is fixed at a minimum preset setpoint when the process input is less than the value, $MIN_T_SP - STEP_ADJ$. The **HOLD** state is the state in which the setpoint is fixed at the previous setpoint when the trip signal is asserted.

The setpoint calculation logic with a falling trip is similar to that with a rising trip. The logic has four states: **FOLLOW**, **MAX RATE**, **MAX POINT**, and **HOLD**. The **FOLLOW** state is the state in which the setpoint tracks the process input using the difference of the amount of the step adjustment, $STEP_ADJ$, when the process input is less than the maximum preset setpoint minus the step adjustment, $MAX_T_SP - STEP_ADJ$, and the slope of the process input is greater than the maximum preset rate, MAX_RATE . The **MAX RATE** state is the state in which the setpoint falls at the maximum preset rate when the process input changes at a rate less than the maximum preset rate. The **MAX POINT** state is the state in which the setpoint is fixed at the maximum preset setpoint when the process input is greater than the value, $MIN_T_SP + STEP_ADJ$. The **HOLD** state is the state in which the setpoint is fixed at the previous setpoint when the trip signal is asserted.

A bistable logic with a directional VSP falling trip can be designed using two logic blocks: the setpoint calculation and the trip decision logic. The state transition diagram of the trip decision logic is identical to that of

the fixed setpoint trip logic shown in Figure 7.

The state transition diagram of the setpoint calculation logic is shown in Figure 10. The logic has five states: **FOLLOW**, **MIN POINT**, **MAX POINT**, **HOLD**, and **STEP**. The **FOLLOW** state is the state in which the setpoint tracks the process input using the difference of the amount of the step adjustment, $STEP_ADJ$, when the process input is less than the maximum preset setpoint plus the step adjustment, $MAX_T_SP + STEP_ADJ$, and the slope of the process input is greater than the zero value. The **MIN POINT** state is the state in which the setpoint is fixed at a minimum preset trip setpoint, MIN_T_SP , when the process input is less than the minimum preset setpoint plus the step adjustment, $MIN_T_SP + STEP_ADJ$. The **MAX POINT** state is the state in which the setpoint is fixed at the maximum preset setpoint when the process input is greater than the value, $MAX_T_SP + STEP_ADJ$. The **HOLD** state is the state in which the setpoint is fixed at the previous setpoint when the process input is greater than the minimum preset trip setpoint plus the step adjustment, $MIN_T_SP + STEP_ADJ$, and the slope of the process input is less than or equal to the zero value. The **STEP** state is the state in which the trip setpoint decreases by the amount of the preset step adjustment when the external manual setpoint reset signal is asserted.

3.3 Synthesis of the Bistable Trip Logic

The code of the bistable trip logic can be written based on VHDL code template described in Figure 3 and the state transition diagram modeled in Section 3.2. The written

Fig. 11. Synthesized Result of the Bistable Logic with a Fixed Setpoint Rising Trip.

code of the bistable trip logic can also be synthesized using the various synthesis tools for implementing the code into the programmable logic device.

In this work, the software tool Libero™ (provided by Actel) has been adopted for writing and synthesizing the code. Figure 11 shows the synthesized results of the bistable logic with a fixed setpoint rising trip. The logic consists of two flip flops, two multiplexors, four invertors, and four comparators.

4. FUNCTIONAL VERIFICATION

The primary purpose of the functional verification is to ensure that the design implements the intended functionality. In this work, the black box approach has been adopted for the functional verification of the bistable trip logic. With a black box approach, the functional verification is performed without knowledge of the actual design implementation. All verifications were accomplished using the available interfaces, without direct access to the internal state of the design and without knowledge of its structure and implementation.

4.1 Testbench and Random Test

For the functional verification, the testbench was designed using SystemVerilog which supports object

oriented programming technique. As shown in Figure 12, SystemVerilog allows the development of modular components when designing a testbench; it also supports the automatic creation of random test cases using a random test technique [14]. In this work, the QuestaSim™ software verification tool provided by Mentor Graphics was used for the functional verification because this tool supports SystemVerilog.

The testbench contains a generator, scoreboard, checker, driver, and monitor. The testbench is connected to the Design Under Test (DUT), which is the bistable trip logic described in Section 3 with signals. The DUT inputs are driven by a driver that separates the random test inputs received from the generator into individual signals. The monitor receives the individual signals from the DUT and transmits them into a data format that is used by the checker. The generator automatically produces random test inputs for the DUT and sends these to the driver and scoreboard. A bistable trip logic with a fixed setpoint and a rate limited VSP requires the process variable as input, whereas the bistable trip logic with a directional VSP requires the process variable and external manual setpoint reset as inputs. The generator produces random values for the process variable by randomizing the time interval and the slope of the process variable during the time interval. It also produces random values for the external manual setpoint reset by randomizing the time interval

Fig. 12. Testbench Configuration.

Fig. 13. Random Testing Results of the Bistable Rising Trip Logic with a Fixed Setpoint.

and setpoint reset value during the time interval. The scoreboard predicts the results of the bistable trip logic. The checker compares the results from the monitor with those from the scoreboard to determine whether the test is passed or not.

The verification task is terminated when the test satisfies the 100% code coverage criteria including the statement coverage, branch coverage, condition coverage, expression coverage, toggle coverage, and finite state machine coverage.

4.2 Test Results of Bistable Trip Logic

Figure 13 shows part of the testing result of the bistable rising trip logic using a fixed setpoint. The logic generates the trip signal and decreases the trip setpoint by the amount of the trip hysteresis when the process variable is greater

than or equal to the preset trip setpoint. The logic generates the untrip signal and increases the trip setpoint by the amount of the trip hysteresis when the process variable is less than the trip setpoint.

Figure 14 shows part of the testing result of the bistable rising trip logic using the rate limited VSP. The logic generates the pretrip or trip signal when the process input reaches the level of the pretrip or trip setpoint that exists at that time. However, the setpoint value changes depending on the rising or falling of the process input. The trip setpoint tracks the process input using the step adjustment value difference if the process input changes with a rate less than that is than the maximum preset rate. The trip setpoint rises or falls at a fixed maximum preset rate when the process input signal changes at a rate greater than the maximum preset rate.

Figure 15 shows part of the testing result of the bistable

Fig. 14. Random Testing Results of the Bistable Rising Trip Logic using the Rate Limited VSP.

Fig. 15. Random Testing Results of the Bistable Falling Trip Logic using the Directional VSP.

falling with a directional VSP. The logic generates the pretrip or trip signal when the process input reaches the level of the pretrip or trip setpoint that exists at that time. However, the setpoint value changes depending on the process input and external manual reset. The trip setpoint tracks the process input using the difference of the step adjustment value if the process input rises away from the trip. When the process input falls toward the trip, the setpoint will be fixed, but can be decreased by the amount of the preset step adjustment when the external manual setpoint reset is asserted.

5. CONCLUSIONS

In this work, the bistable trip logic was designed in RPS using a VHDL for the application of the PLD technology in safety I&C systems in NPPs. The bistable

trip logic was designed in the RTL using the synchronous finite state machine modeling technique. Then, the bistable trip logic was synthesized for implementation in an FPGA. The functional verification was also performed to verify that the bistable trip logic was designed correctly satisfying the requirements. For the functional verification, a random testing technique was developed to generate random test inputs for the bistable trip logic. During the random testing of the bistable trip logic, the various code coverage metrics were assessed in order to verify that the bistable trip logic was sufficiently exercised. This included statement, branch, condition, toggle, and finite state machine coverage.

PLD technology has many advantages such as performance, simple architecture, and easy qualification. The synthesized results demonstrated that it is possible to design and implement the logic using PLD technology that has the complexity required by the safety I&C systems. This work also demonstrated that one verification issue related

to PLD based safety I&C systems can be solved using the random testing technique.

ACKNOWLEDGMENT

This work was supported by the Nuclear Technology Development Program of the Korea Institute of Energy Technology Evaluation and Planning (KETEP) with a grant funded by the Korean Ministry of Knowledge Economy (2010161010001F).

REFERENCES

- [1] Patrick Salaün, Frederic Dumas, Thuy Nguyen and Claude Esmenjaud, "FPGA/ASIC: A promising technology for future of I&C Systems in power industry," *6th American Nuclear Society International Topical Meeting on NPIC& HMIT*, Knoxville, Tennessee, USA, Apr. 5-9, 2009.
- [2] Kee Choon Kwon and Myeong Soo Lee, "Technical Review on the Localized Digital Instrumentation and Control Systems," *Nuclear Engineering and Technology*, Vol. 41, No. 4, pp. 447-454 (2009).
- [3] Patrizio Piasentin, "FPGA: a solution for High-Rel and long life time application," *1st Workshop on the Applications of Field-Programmable Gate Arrays in Nuclear Power Plants*, Chatou, France, Oct. 8-10, 2008.
- [4] Claude Esmenjaud, "Drivers and barriers for use of FPGA technology in safety I&C systems in NPPs, *1st Workshop on the Applications of Field-Programmable Gate Arrays in Nuclear Power Plants*, Chatou, France, Oct. 8-10, 2008.
- [5] David Dangla, "FPGA for space applications," *1st Workshop on the Applications of Field-Programmable Gate Arrays in Nuclear Power Plants*, Chatou, France, Oct. 8-10, 2008.
- [6] Vyacheslav Kharchenko, "Experience of RPC <<Radiy>> is designing, manufacturing and implementation of FPGA-based NPP I & C systems," *1st Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants*, Chatou, France, Oct. 8-10, 2008.
- [7] Bernard F. Dittman, "Regulatory Experience with a FPGA-based Digital I&C Review," *2nd Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants*, Kirovograd, Ukraine, Sept. 29-Oct. 1, 2009.
- [8] Jingke She and Jin Jiang, "Application of FPGA to Shut-down system No. 1 in CANDU", *6th American Nuclear Society International Topical Meeting on NPIC and HMIT*, Knoxville, Tennessee, USA, Apr. 5-9, 2009.
- [9] Volnei A. Pedroni, *Circuit Design with VHDL*, p. 160, MIT Press, London, England (2004).
- [10] Peter J. Ashenden, *Digital Design: An Embedded Systems Approach Using VHDL*, p. 194, Elsevier Inc., New York, USA (2008).
- [11] Wikipedia, http://en.wikipedia.org/wiki/Hardware_description_language.
- [12] Israel Koren, *Computer Arithmetic Algorithm*, 2nd ed., A K Peters, Natick, USA (2002).
- [13] Seop Hur and Dong Hoon Kim, "A New Automatic Periodic Test Method for the Digital Reactor Protection System," *6th American Nuclear Society International Topical Meeting on NPIC and HMIT*, Knoxville, Tennessee, USA, Apr. 5-9, 2009.
- [14] Chris Spear, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, 2nd ed., p. 161, Springer, New York, USA (2008).