Nuclear Engineering and Technology 48 (2016) 1192–1205

Available online at ScienceDirect

Nuclear Engineering and Technology

journal homepage: www.elsevier.com/locate/net

Original Article

# Field Programmable Gate Array Reliability Analysis Using the Dynamic Flowgraph Methodology

*Phillip McNelles** *and Lixuan Lu*

*Faculty of Energy Systems and Nuclear Science, University of Ontario Institute of Technology (UOIT), 2000 Simcoe Street North, Oshawa, Ontario, L1H 7K4, Canada*

ARTICLE INFO

*Article history:*
Received 16 January 2016
Received in revised form
22 March 2016
Accepted 22 March 2016
Available online 31 March 2016

*Keywords:*
Dynamic Flowgraph Methodology
Field Programmable Gate Array
Instrumentation and Control
Nuclear Power Plant
Reliability

ABSTRACT

Field programmable gate array (FPGA)-based systems are thought to be a practical option to replace certain obsolete instrumentation and control systems in nuclear power plants. An FPGA is a type of integrated circuit, which is programmed after being manufactured. FPGAs have some advantages over other electronic technologies, such as analog circuits, microprocessors, and Programmable Logic Controllers (PLCs), for nuclear instrumentation and control, and safety system applications. However, safety-related issues for FPGA-based systems remain to be verified. Owing to this, modeling FPGA-based systems for safety assessment has now become an important point of research. One potential methodology is the dynamic flowgraph methodology (DFM). It has been used for modeling software/hardware interactions in modern control systems. In this paper, FPGA logic was analyzed using DFM. Four aspects of FPGAs are investigated: the "IEEE 1164 standard," registers (D flip-flops), configurable logic blocks, and an FPGA-based signal compensator. The ModelSim simulations confirmed that DFM was able to accurately model those four FPGA properties, proving that DFM has the potential to be used in the modeling of FPGA-based systems. Furthermore, advantages of DFM over traditional reliability analysis methods and FPGA simulators are presented, along with a discussion of potential issues with using DFM for FPGA-based system modeling.



## 1. Introduction

Field programmable gate arrays (FPGAs) are a type of programmable logic device. FPGAs can be utilized to construct digital logic circuits. These programmable logic devices are programmed by the end user to perform the necessary functions, and certain FPGAs are reprogrammable. FPGAs do not usually include software or operating systems, as the logic functions are programmed (synthesized) onto the chip itself. The programming itself is implemented using hardware description languages (HDLs) [1]. A well-known HDL, named VHDL (very-high-speed integrated circuit HDL), is used in this

* *Corresponding author.*
E-mail address: phillip.mcnelles@gmail.com (P. McNelles).
http://dx.doi.org/10.1016/j.net.2016.03.004

study. FPGAs can perform many of the control functions performed by other electronic logics, such as analog circuits, application-specific integrated circuits, microprocessors, and PLCs. FPGAs can be used in different nuclear instrumentation and control (I&C) systems, provided that they can be proved to satisfy the safety requirements [2,3].

In the nuclear field, many I&C systems that are currently in use in existing nuclear power plants (NPPs) are becoming obsolete. FPGAs are being considered as replacements to those systems. Compared with application-specific integrated circuits (ASIC) and analog circuits, FPGAs can be reprogrammed if needed. Compared with PLCs and microprocessors, FPGAs have been shown to have superior response time and faster processing speed [4,5]. FPGA implementations have taken place in Europe and Asia [6–8], and recently there is increasing interest in these systems in Canada and the USA [9–11].

Very strict safety and quality requirements have been put in place to ensure that the control systems in NPPs function safely. This means that any FPGA-based systems would have to undergo a thorough reliability analysis. To be used in an NPP, an I&C system will have to meet certain qualitative and quantitative reliability requirements; however, these requirements will vary among different regulators. In the case of digital I&C systems, the use of software in the system must also be verified. While FPGAs themselves do not run software, the HDL code is used in configuring the FPGA, which can introduce logic errors into the system. Therefore, both the hardware and HDL logic components of FPGA-based systems must be verified for FPGAs to be used in NPP I&C systems. In this paper, the focus is on the HDL logic, including the Institute of Electrical and Electronics Engineers (IEEE) logic standards, important logical components of the FPGA, and a small test system itself. Regulators may not set specific requirements for FPGA-based systems; however, there is a standard from the International Electrotechnical Commission [12], and guides in the form of International Atomic Energy Agency (IAEA) [13] and Nuclear Regulatory Commission (NUREG) [14] documents that provide guidance on the design and review of FPGA-based systems.

A failure mode and effect analysis (FMEA) at the component and system levels will determine the potential failure modes that can be used as top events in the analysis of FPGA logic. The FPGA logic in these cases will be analyzed to determine the top and initiating events that could lead to failures in FPGA logic subcomponents and in the logic of the system itself. Analysis of FPGA hardware components is beyond the scope of this paper.

There are many reliability analysis techniques in the literature and in industrial practice. The methodology used in this paper is the dynamic flowgraph methodology (DFM). DFM is a dynamic (time-dependent) methodology used to model and analyze digital control systems. In this paper, DFM is shown to be able to validate the logic of FPGA-based systems, including uncovering errors that occur in the logic, and the effect those errors could have on the system or system components. The use of ModelSim simulations adds evidence to the DFM results, to help confirm that DFM can accurately model FPGA system logic. This, in turn, helps validate the use of DFM in the analysis of FPGA-based systems, and allows for DFM to be applied to more in-depth qualitative and quantitative analyses in future research projects.

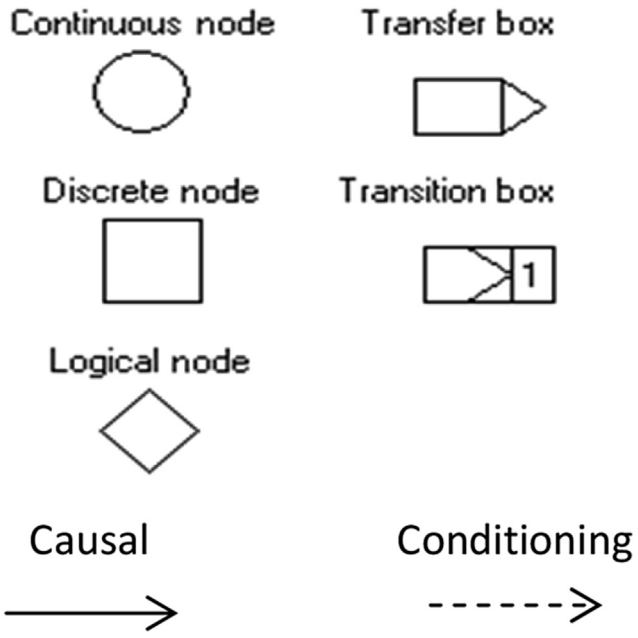


**Fig. 1 – DFM nodes, transfer boxes, and connectors. DFM, dynamic flowgraph methodology.**

A detailed description of DFM and its advantages over other reliability analysis techniques are presented in Section 2. Using this methodology, several important aspects of FPGA systems are modeled and analyzed. Section 3 discusses three important aspects of FPGA-based systems: the IEEE 1164 standard, registers, and configurable logic blocks (CLBs). Afterward, an FPGA-based dynamic signal compensator is presented, and a simplified sample FMEA is discussed. The results of the analyses based on the models created in Sections 3 are presented and discussed in Section 4. Finally, the advantages of DFM for FPGA modeling and potential issues are covered in Section 5.

## 2. Dynamic flowgraph methodology

This section describes the theory and application of DFM. Section 2.1 will provide an overview of DFM. Section 2.2 will discuss the actual DFM model, and Section 2.3 will discuss the main limitation regarding the use of DFM.

### 2.1. *DFM overview*

DFM represents the system being analyzed using a directed graph model. After the model is built, it can be analyzed by the inductive and deductive algorithms built into the methodology [15]. The DFM deductive analysis will return a list of "prime implicants" (PI), which are sets of occurrences that would cause the top event (failure event). They are understood to be the multivalued logic equivalent of minimal cut sets. The

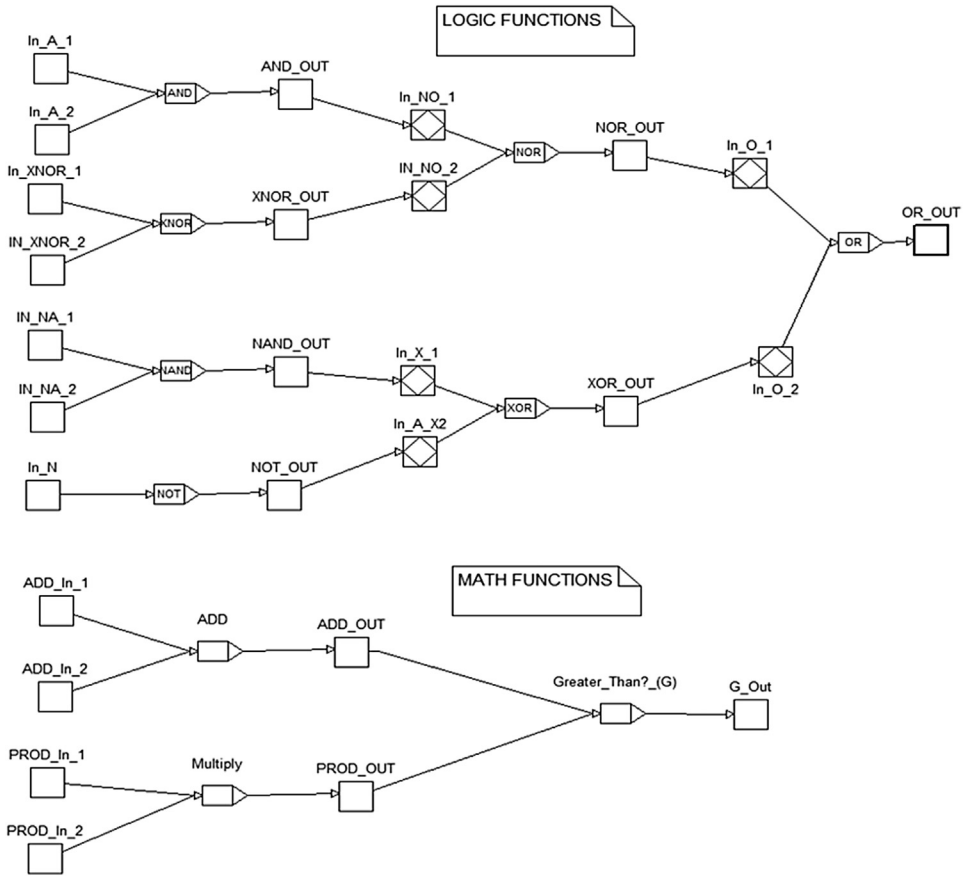


**Fig. 2 – DFM model for logic and mathematical functions. DFM, dynamic flowgraph methodology.**

inductive analysis will return a similar list, although the PIs are referred to as "sequences." DFM also incorporates time-dependent behavior, allowing for both static and dynamic (time-dependent) analyses [15].

DFM was created in order to model both hardware and software components. This makes it a suitable method for modern digital electronics such as FPGAs. It has been shown to be more effective than well-known traditional methods such as FMEA, fault tree, and event tree for modeling the complicated hardware/software/firmware interactions in digital systems [16–18]. Furthermore, DFM also was rated as one of the methodologies for having the most positive features and the fewest negative or uncertain features [16–18]. A literature search of recent publications has consistently shown DFM to be an effective method for reliability analysis and probabilistic risk assessment of those systems [19–22].

### 2.2. *DFM model and calculations*

The actual DFM model consists of a series of process variable nodes, connections, and transition boxes between these nodes to show the relationship between system parameters [15]. The nodes, transition boxes, and connectors are shown in Fig. 1.

In Fig. 1, the nodes are used to represent the parameters, process variables, or components of the system being analyzed. Continuous nodes represent a continuous behavior, discrete nodes represent a discrete behavior, and logical nodes show logic tests on the current state of the system. In the analysis, all nodes work in the same way, but look different in the model to make it easier for the user to understand what is being represented. A process variable node can represent an output value, such as a temperature or voltage output. Each node is then discretized into an arbitrary number of states that the node can take on (e.g., "High Voltage", "Low Voltage," etc.).

A transfer box represents functional relationships between the components of the model (similar to the transfer function of a system). The transition box works in a similar manner, but includes time delays, allowing one to model the time dependence of the system and/or the system components. The user will input decision tables into these boxes, to determine the output of the box, based on the combination of inputs. These decision tables are then combined to make one "critical transition table" when the model is run, and then the PIs are calculated using the DFM algorithms [15]. An example table is given in Section 3.2.

The causal connection shows the input and output of the boxes (the cause-and-effect system behavior). The conditioning connectors indicate the connections between the input and output of functions and determine which function is used.

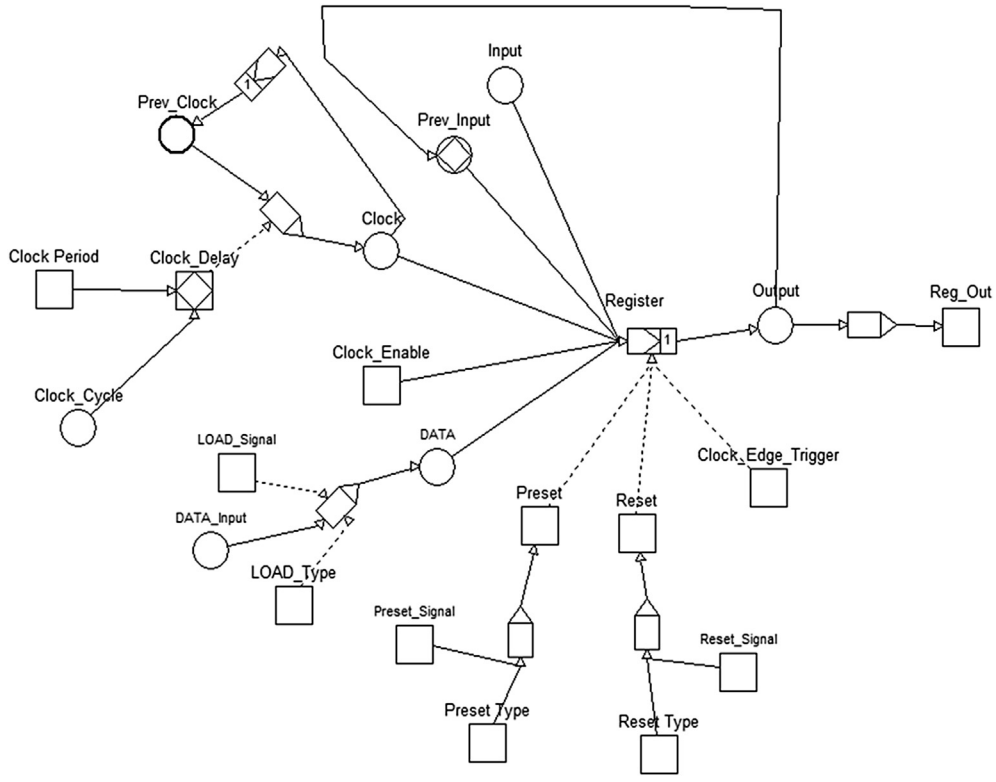


**Fig. 3 – DFM model for FPGA register. DFM, dynamic flowgraph methodology; FPGA, field programmable gate array.**

The basis for solving the actual DFM model is similar to what is used to calculate the minimal cut sets in fault tree analysis (FTA). Each PI is calculated as follows [15]:

$$PI_j = \bigcap_{i=1}^{n} X_i^{(j)} = \prod_{i=1}^{n} X_{i=1}^{(j)} \quad (1)$$

where X represents each node–state–time step combination, referred to as a "literal."

There are several ways to calculate the top event probability, once the PI probability is calculated. The first method is to take the sum of all the PI probabilities, which is the default method in the Dymonda software program. The second method is to employ the "upper bound" approximation using the PI probabilities, which is often used in FTA. However, these two methods are approximations and tend to overestimate the top event probability, as the PIs may not always be mutually exclusive. The exact top event probability can be calculated by converting the PIs to mutually exclusive implicants and then taking the sum of the mutually exclusive implicants [15]. The conversion of PIs to mutually exclusive implicants can be computationally intensive and time consuming, therefore, one of the aforementioned approximations is often used to approximate the top event probability.

**Table 1 – Sample decision table for simplified register.**

| Inputs | | | | Outputs |
|---|---|---|---|---|
| Reset | Enable | Clock | Input | |
| 1 | * | * | * | 0 |
| 0 | 0 | * | * | 1 |
| 0 | 1 | 0 | * | 1 |
| 0 | 1 | 1 | + | 0 |

### 2.3. *DFM limitations*

DFM has certain limitations, with the largest issue being due to the explosion of states or "state explosion" [22]. The inclusion of a large amount of nodes and states can lead to very large and complicated decision tables, creating a combinatorial explosion of states during the analysis. This limits the application of DFM to realistic systems of small- or medium-sized systems, such as the FPGA test system shown in Section 3.4 [23]. State explosion is not limited to DFM, as it is an issue for other dynamic methods, such as Markov models [24].

## 3. Materials and method

In order to prove that DFM is accurately modeling the FPGA logic and system properties, the DFM results must be compared with the results from an established source. To accomplish this, the VHDL code was used to create test systems, as well as testbenches. The testbenches would provide input stimuli in order to test the VHDL test programs. The test programs used synthesizable VHDL codes where possible, which were also used to create the VHDL netlists. The ModelSim simulator program was used to simulate the VHDL code, using the testbenches. The results of the ModelSim

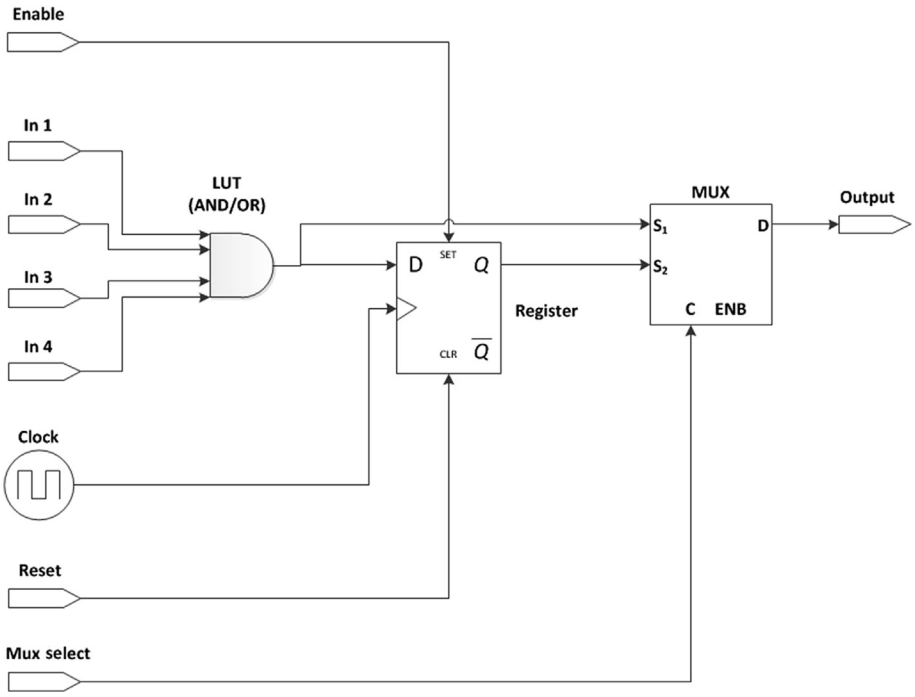


**Fig. 4 – CLB flowgraph with either "AND" gate or "OR" gate LUT. CLB, configurable logic block; CLR, clear; ENB, enable; LUT, look-up table; MUX, multiplexer.**

simulations ("waveforms") were then compared with the results of the DFM analyses, to determine the accuracy of the DFM models.

The three principal aspects of FPGAs considered in this paper are presented in this section. Section 3.1 will discuss the IEEE Standard 1164, while Section 3.2 presents the register modeling. Section 3.3 shows the FPGA logic blocks modeling. An example FPGA test system of a signal compensator is shown in Section 3.4. Lastly, Section 3.5 will present a simplified FMEA that was used to obtain the failure data (top events) for the DFM analysis.

### 3.1. *IEEE Standard 1164*

IEEE 1164 is an important standard when using the VHDL code to program the FPGA. It is a package that is compiled into a library and is often imported into VHDL files. This standard defines important features, including nine-state logic, resolution function, and Boolean functions, such as AND, OR, XOR, and NOT. The Boolean functions AND, OR, XOR, and NOT are defined in tables, and the remaining functions (NOR, XNOR, and NAND) are made using NOT function [25]. The states for the nine-state logic can be found in the literature [25]. DFM is used to model the logic and mathematical functions based on this standard. A DFM model consisting of the logic and mathematical functions is shown in Fig. 2.

In Fig. 2, the logic functions includes AND, OR, XOR, NOT, NOR, NAND, and XNOR functions. The inputs are set to be the logic states given in [25], producing various combinations of outputs. A similar method is taken with the mathematical functions, where two inputs are added (In_Add_1 and In_Add_2), and the sum is compared with the product of the other inputs (In_Prod_1 and In_Prod_2). The output "G_Out" is then the output from the "greater or equal to function" (shown as a "less than" function in the netlist). The ModelSim and DFM results are given in Subsection 4.1.

### 3.2. *Register*

The register is an important component of digital logic and FPGA operation. It is commonly used in electronic systems to store a data value and output it at a certain clock edge. In FPGAs, these registers are used in many kinds of sequential and recursive logics, when the algorithms require results or data from previous time steps [26]. The DFM model for the register is shown in Fig. 3. Here, the "Reset," "Preset," "Input," and "Data" nodes all represent input signals into the register, where the transition box (entitled "Register") performs as a register in an FPGA would. A time delay on "1" is included, in order to hold the data for a time step, similar to an FPGA register. The "Reset" signal will reset the register to "0," while the "Clock Enable" signal will allow the register to store the new "Input" value, and then output the new value after a time delay (based on the "Clock" signal). If the "Clock" signal does not allow the register to update the value, then the previous input value ("Prev_Input") is used instead. This model also includes the "Clock_Period" (if it is longer/shorter than specified), "Clock_Cycle" (if the clock has a duty cycle that is different from 50%), as well as a node for the previous input ("Prev_Input"), as that can have an effect on the output.

The value from the "Output" is fed back into the "Prev_Input", after a delay. The "Clock" node was broken down into four states, 1, 0, +, −, to represent the clock transitions. To allow the "Clock" state to cycle, the "Prev_Clock" node was included, which will store the previous "Clock" signal (after a time delay) and then output the next state to the "Clock". State

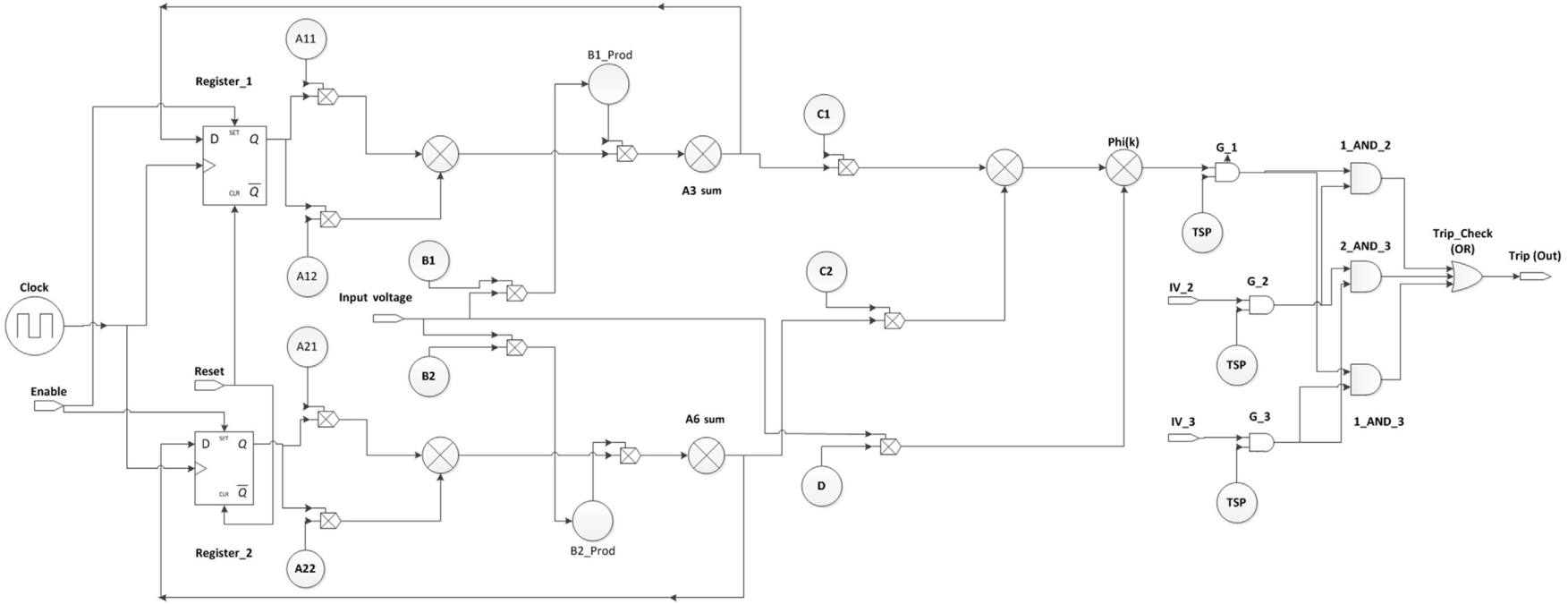


**Fig. 5 – Flowgraph model for FPGA-based platinum detector. FPGA, field programmable gate array.**

**Table 2 – Sample FMEA for FPGA aspects.**

| Aspect | Failure mode | Cause | Effect |
|---|---|---|---|
| IEEE 1164 | Or = 0 | "X" logic<br>"H" logic | Mathematical error |
| | G_Out = 0 | "U" logic | Logic error |
| Register | Output "X" | "X" logic | Memory error |
| CLB | Output "1" | Logical errors | Incorrect CLB value |
| DSC | Spurious Trip | Constant error | System failure |
| | Missed Trip | Input error ("X") | System failure |

CLB, configurable logic blocks; DSC, Dynamic Signal Compensator; FMEA, failure mode and effect analysis; FPGA, field programmable gate array; "H", High; IEEE, Institute of Electrical and Electronics Engineers; "U", Uninitialized; "X", Unknown.

"+" represents the rising edge, while state "–" represents the falling edge. The clock will cycle through the clock states as the time step changes. The "Output" node includes states for "1" and "0," as well as extra states for the outputs for "Reset", "Preset", and "Data". The results are then funneled to the standard "1" and "0" for the "Reg_Out" node. Results for the register model and ModelSim simulations are given in Subsection 4.2. An example of a decision table, similar to the one used in the "Register" transition box is given in Table 1. To save space, it was assumed that the "Previous Input" had a value of "1." It should be noted that the "*" represents a "Don't Care" value.

In Table 1, it is seen in the first row that if the "Reset" signal is received, then the "Output" is reset to "0," regardless of the other inputs. If the "Reset" signal is "0," then the other inputs will become relevant. In the second row, the "Clock Enable" signal is "0," so the register will not update the value, and the "Previous Input" of "1" will be output again. If the "Clock Enable" signal is "1", but the "Clock Signal" is "0" and the clock transitions on a rising clock edge, then the "Previous Output" value will be output again. In the last row, if the "Clock Enable" is "1" and the "Clock Signal" is "+," then the register will update and output the new value of "0."

### 3.3. *FPGA logic blocks*

All FPGAs share certain basic components, namely input/output ports, programmable interconnects, and CLBs. Input/output ports are used to carry data signals to and from the FPGA, while programmable interconnects are used to connect the CLBs together. CLBs are of particular interest, as those logic blocks contain the logic elements needed to perform the desired logic functions. In the most basic form, each CLB will contain a look-up table (LUT), register, and possibly a multiplexer (Mux) that can be used to bypass the register if desired [1]. DFM was used to create two separate logic blocks, one with an "AND" gate LUT, and the other with an "OR" gate LUT. This will show how DFM can model the basic logic elements of an FPGA, encompassing the components discussed in Sections 3.1 and 3.2. A block diagram is presented in Fig. 4.

**Table 3 – Sample implicants for "OR_OUT = 0" and "G_OUT = 0" top events.**

| Implicant 1 (node) | Implicant 1 (state) | Implicant 117 (node) | Implicant 117 (state) |
|---|---|---|---|
| AND_OUT | 1 | *ADD_In_2* | *XX* |
| *IN_NA_1* | *H* | ADD_OUT | XX |
| IN_NA_2 | 1 | | |
| In_A_1 | 1 | | |
| In_A_2 | 1 | | |
| In_N | 1 | | |
| NAND_OUT | 0 | | |
| NOR_OUT | 0 | | |
| NOT_OUT | 0 | | |
| XOR_OUT | 0 | | |

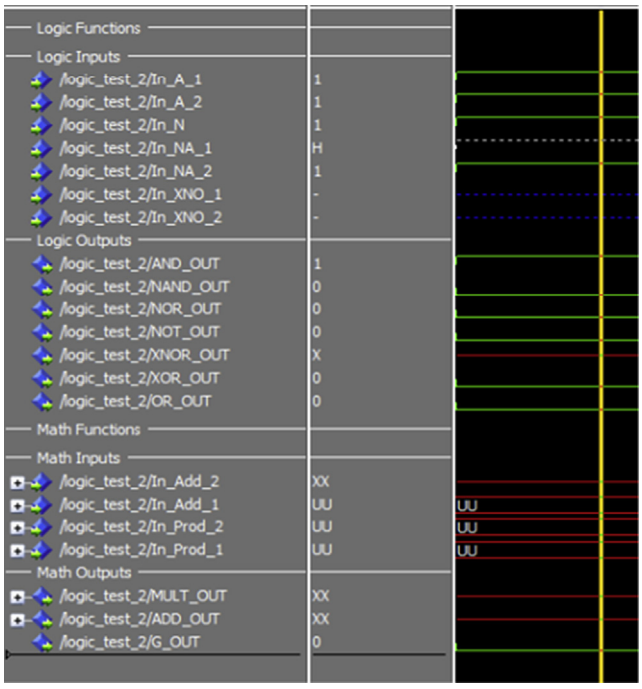


**Fig. 6 – ModelSim results for "OR_OUT" and "G_OUT" top events.**

### 3.4. *Platinum dynamic compensation*

Neutron detectors are used in NPPs to monitor neutron power (flux) inside the reactor core. Several different materials are used, such as platinum, rhodium, vanadium, and cobalt. In a Canada deuterium uranium reactor, platinum detectors are used in safety systems due to their fast (prompt) response. Platinum detectors are composed of multiple isotopes and therefore have multiple decay chains [27,28]. Dynamic signal compensation is used to compensate for the delayed response of the detector, in order to obtain an accurate reading of the current neutron flux. The details of the platinum detectors and signal compensation can be found in [29,30].

$$\Phi_c = 0.89\phi + \frac{0.045\phi}{1+3.9s} + \frac{0.017\phi}{1+30s} + \frac{0.021\phi}{1+250s} + \frac{0.045\phi}{1+2,500s} \quad (2)$$

where $\Phi_c$ refers to the calculated neutron flux and $\phi$ is the actual neutron flux. Eq. (3) is a simplified transfer function for the dynamic signal compensator [31]:

$$\Phi_c = \left[K_1 - \frac{K_2}{1+T_1 s} + \frac{K_3}{1+T_2 s}\right] I \tag{3}$$

where $I$ is the current from the detector; $K_1$, $K_2$, $K_3$ are coefficients; $T_1$, $T_2$ are corresponding time constants; and $s$ is the Laplace transform variable [27,28]. Appending the values for $K_1$, $K_2$, $K_3$, $T_1$, and $T_2$, Eq. (3) becomes as follows:

$$\Phi_c = \left[1.066 - \frac{0.028}{1+30s} + \frac{0.038}{1+2,500s}\right] I \tag{4}$$

Converting Eq. (4) to a state space representation yields the following equations:

$$\dot{\Phi}(t) = \begin{bmatrix} -0.0337 & 0 \\ 1 & 0 \end{bmatrix} \Phi(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} I(t) \tag{5}$$

$$\Phi_c(t) = [\,-0.0142 \quad 0\,]\Phi(t) + [15.99]\, I(t) \tag{6}$$

The state space representation also included the scaling factor, so that an input range of 0–10 V would produce an output range of 0–150% full power (FP). The state space model was then discretized, using the zero-order hold method, and a sampling time of 0.5 seconds, resulting in the following representation:

$$\dot{\Phi}(t) = \begin{bmatrix} 0.9833 & 0 \\ 0.4958 & 1 \end{bmatrix} \Phi(t) + \begin{bmatrix} 0.4958 \\ 0.1243 \end{bmatrix} I(t) \tag{7}$$

$$\Phi_c(t) = [\,-0.0142 \quad 0\,]\Phi(t) + [15.00]\, I(t) \tag{8}$$

The block diagram for the platinum compensator is given in Fig. 5.

In Fig. 5, the nodes "G_1", "G_2", and "G_3" are the logic tests for the input being greater than or equal to the Trip Setpoint(TSP), and "IV_2" and "IV_3" represent the additional inputs. The "Phi(k)" node refers to the total flux calculation, and the nodes named "A," "B," "C," or "D," which also contain numbers (e.g., "A11," "B1," "C1," D″) represent constants in the state space equations in Eqs. (5) and (6). The registers ("Register_1" and "Register_2") store the outputs from Eq. (5) that are needed for the calculation at the next time step. The output of the state space equations is a neutron power, based on the detector current, which is based on the neutron flux. As the neutron flux is very high (generally of the order of $10^{14}$ n/cm$^2$/s), it is often expressed as a percentage of FP. To simplify the calculations, the voltage input was taken in the range of 0–10 V, and the neutron power was scaled for 0–15%, with a TSP set at 12.0%. In reality, the neutron power would have a range of 0–150% FP, and the TSP being set at around 125% FP.

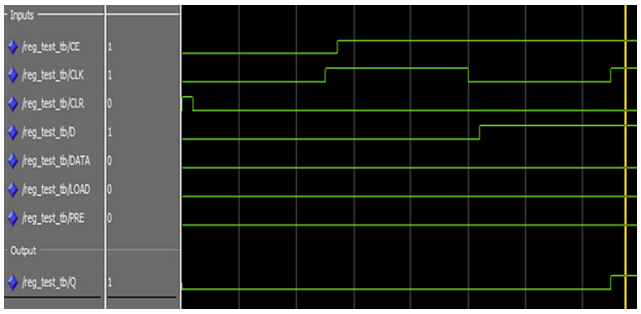


**Fig. 7 – ModelSim results for FPGA register analysis (top event "Output = 1"). FPGA, field programmable gate array.**

It can be seen that the previous values for $\Phi(t)$ are required to calculate the correct flux. The registers (discussed in Section 3.2) are used to store this information, therefore, the FPGA code (and netlist) contains two registers. Additional code was added, for trip logic. The flux value was compared with a TSP, along with two other inputs, to create the two out of three logics. The analysis results for the platinum comparator and trip logic is given in Subsection 4.4.

### 3.5. *Simplified FMEA example*

This section will provide an example of an FMEA for the different FPGA aspects discussed in Subsections 3.1–3.4. These FMEAs produce the top events that are used in some of the DFM analysis results, seen in Section 4. It should be noted that not every DFM analysis considered actual failures, as the aim of the paper was to show the applicability of DFM to modeling of FPGA-based system logic, and as such the models included both correct and erroneous behaviors.

Table 2 presents a simplified FMEA for failures that were considered for the different aspects of FPGA logic. The first is

**Table 4 – Prime implicant for DFM FPGA register analysis (top event "Output = 1").**

| Implicant 26 (node) | Implicant 26 (state) | Implicant 26 (time step) |
|---|---|---|
| *Clock* | ± | −1 |
| Clock_Period | Normal | −1 |
| *Clock_Edge_Trigger* | R_E | −1 |
| *Clock_Enable* | 1 | −1 |
| DATA_Input | No_DATA | −1 |
| *Input* | 1 | −1 |
| Preset_Signal | 0 | −1 |
| Prev_Clock | 0 | −1 |
| Prev_Input | 0 | −1 |
| Preset_Signal | 0 | −1 |
| Reset_Signal | 0 | −1 |

DFM, dynamic flowgraph methodology; FPGA, field programmable gate array.

**Table 5 – Prime implicant for DFM FPGA register analysis (top event "Output = X").**

| Implicant 16 (node) | Implicant 16 (state) | Implicant 16 (time step) |
|---|---|---|
| *Clock* | ± | −1 |
| Clock_Period | Normal | −1 |
| Clock_Edge | R_E | −1 |
| Clock_Enable | 1 | −1 |
| DATA | 0 | −1 |
| DATA_Input | No_DATA | −1 |
| *Input* | X | −1 |
| *Preset* | X | −1 |
| Prev_Clock | 0 | −1 |
| *Reset_Signal* | X | −1 |
| Trigger_Select | Pos | −1 |

DFM, dynamic flowgraph methodology; FPGA, field programmable gate array.

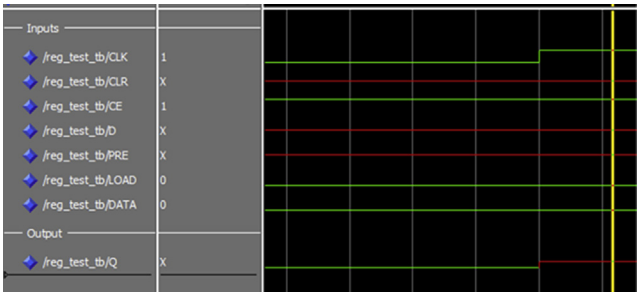


**Fig. 8 – ModelSim results for FPGA register analysis (top event "Output = X"). FPGA, field programmable gate array.**

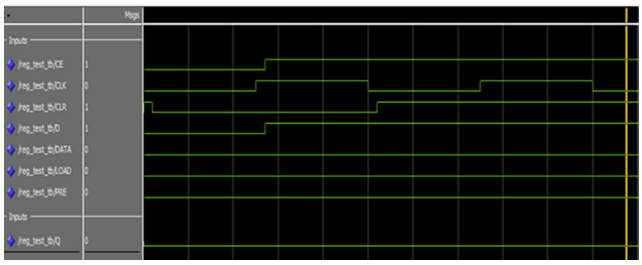

**Fig. 9 – ModelSim results for sequence "Reset = 1".**

the IEEE logic standard, where it is seen that there could be mathematical or logic errors (respectively), due to the presence of certain logic states. The "U" (Uninitialized), "H" (High), or "X" (Unknown) logic states are generally used to represent errors in simulation. These logic states can produce errors in mathematical/logic functions, resulting to the incorrect (in this case) value of "0," due to how the functions are defined in the IEEE 1164 standard [24]. A similar issue is seen with the register. If an "X" logic value is input into the register, it would be stored, and the "X" would be output on the next clock cycle, which could cause a failure at future time steps.

The effects of the failure modes of the IEEE 1164 standard and register are manifested when the larger FPGA aspects are considered. Logic failures on mathematical/logic functions or registers (such as "U" or "X" logic) can affect the entire CLB. In this paper, an erroneous output of "1" (in this example) could be produced, due to the "U" or "X" logic states. This may not occur with the "OR" logic, depending on the other inputs, but for other logic, such as "AND" logic, the output of the CLB could be affected. This all builds up to the whole test system level, where it is seen that a "Missed Trip" occurs due to an "Input Error." This could be due to an error of the input signal itself, or due to errors passed along from the other components (i.e., failure modes at the logic, register, or CLB level cause failures at the whole system level). Lastly, it was seen that "Spurious Trip" could occur not only through errors in the subcomponents, but also through an incorrectly specified constant, used in multiplication.

## 4. Results

This section shows the results for the models created in Section 3. In each case, the entries in the PI tables that are of the most notable are italicized. It should be noted that in certain ModelSim waveforms, the signals "Reset", "Clock Enable", "Clock", and "Preset" (if needed) were shortened to "CLR" (clear), "CE" or "Enable," "CLK," and "PRE," respectively, to save space in the specific waveform graph.

**Table 6 – Sequence for initiating event "Reset = 1."**

| Sequence 1 (node) | Sequence 1 (state) | Sequence 1 (time step) |
|---|---|---|
| Prev_Input | 1 | 0 |
| *Reset_Type* | *Asynch* | 0 |
| *Reset_Signal* | 1 | 0 |
| Reg_Out | 0 | 1 |

### 4.1. *IEEE 1164 standard results*

The results for the DFM analysis and the ModelSim simulations for IEEE 1164 are presented in this section.

Table 3 shows sample implicants for the IEEE 1164 DFM model, with the ModelSim results given in Fig. 6. The top events for both the logic and the mathematical functions were set at "0," and the DFM model was run deductively, to find the PIs. In total, there are 192 PIs for the "OR_OUT" = 0 node and 104 PIs for the "G_OUT" = 0 node. It was seen that the inputs and outputs of the logic model matched up with the inputs and outputs in the ModelSim simulation. The "H" logic in the "In_NA_1" input produced an "X" value at the output of the "XNOR" logic gate. This in turn caused the overall output of the "OR" logic to read "0." It was also seen with the mathematical model, that an unknown state ("XX") in the "In_Add_2" input would cause the "Add_Out" output to also read "XX", forcing the "G_OUT" node to read "0" (false), indicating that an error with one of the inputs could cause the trip signal not to actuate. This is the same as discussed in Subsection 3.5.

### 4.2. *Register results*

The results for the DFM analysis and ModelSim simulations for the register are shown in this subsection. In the ModelSim waveforms, the "Input" node (signal) is denoted as "D," while the "Output" is denoted as "Q." Table 4 and Fig. 7 show the results for a register where the Output = "1."

The simulation in Fig. 7 had the top event "Reg_Out = 1" for one time step. The state "R_E" refers to a rising edge clock trigger, "Normal" clock period means that there is no clock delay, and the "Synch/Asynch" refers to synchronous and asynchronous processes, respectively. The time steps in this DFM model were taken as one time step being equal to one-half of the clock cycle. In Fig. 7 (Implicant 26), it is seen that having an Input of "1," with the "Clock_Enable" signal of "1" and a rising edge clock and trigger will produce the "Output" of "1," when it is not pre-empted by other inputs such as the Reset, Preset, or Load signals. The ModelSim results will not explicitly show values for the "Clock_Edge_Trigger" and will not explicitly state if the "Clock_Period" is correct; it will just show what the value is.

**Table 7 – Prime implicant for DFM FPGA logic block analysis (top event "Logic Block Out = 1").**

| Implicant 9 (node) | Implicant 9 (state) | Implicant 9 (time step) |
|---|---|---|
| *In_1In_2* | 11 | −2 |
| *In_3* | 1 | −2 |
| *In_4* | 1 | −2 |
| AND_Out | 1 | −2 |
| *Clock* | ± | −2 |
| Prev_Clock | 0 | −2 |
| *Clock_Enable* | 1 | −2 |
| *Reset_Signal* | 0 | −2 |
| Clock | 1 | −2 |
| *Mux_S* | 0 | −1 |
| Prev_Clock | + | −1 |
| *Reset_Signal* | 0 | −1 |
| *In_1* | 1 | 0 |
| *In_2* | 1 | 0 |
| *In_3* | 1 | 0 |
| *In_4* | 1 | 0 |

DFM, dynamic flowgraph methodology; FPGA, field programmable gate array.

DFM can also be used to identify possible failure and/or undesired outputs. This can include unknown values ("X"), incorrect outputs, or incorrect clock transitions, as seen in Table 5 and Fig. 8. In Table 5, the top event was set to "Output = X," to simulate an error/failure state with the register. In this case, an "Input" of "X" along with the "+" "Clock" transition on a rising edge clock, caused the "X" state to be passed to the output. The "Input" is not overruled by a "DATA," "Preset," or "Reset" signal (both "Reset" and "Preset" signals are in the error state of "X", so that does not preempt the "Input"). The ModelSim results in Fig. 8 confirm this, where the inputs of "Reset" = "X", "Preset" = "X", "CE" ="1", and "Input" = "X" produce an "Output" of "X" (as discussed in Subsection 3.5) when the clock transitions on a rising edge.

The sequence for an inductive analysis is shown in Table 6. It was taken as an asynchronous reset signal of "1," which should pre-empt any other input combination and result in an output value of "0." As seen in Table 6, only the one PI is produced, and it returns a value of "0," as expected. This is confirmed in the ModelSim simulation shown in Fig. 9 where the simulation shown in Fig. 7 was run again, and this time with the "Reset" signal set to "1." As expected, all three outputs were "0." In this case, if the asynchronous "Reset" (CLR) signal is "1," then the "Output" will be "0," regardless of what the other signals are, so the other signals in Fig. 9 have no effect on the register output.

### 4.3. *Logic block results*

The individual logic block models ("AND" and "OR") were analyzed, with the results shown in this section. The analysis for the "AND" gate CLB was run for two time steps, with the top event set to "Logic_Block_Out ('LB_Out') = 1" at time steps "0" and "−1." This produced 23 PIs; two of them are shown here. In Table 7, it is seen that the four inputs to the LUT ("In_1" … "In_4") are all in state "1," which could produce a value of "1" from an "AND" gate. When the clock transitions ("+") and the "Enable" signal is "1," the "AND" value of "1" is loaded into the register. The register value is then selected by the Mux Select signal ("Mux_S"), which is then output from the logic block at TS = −1 and TS = 0. The corresponding ModelSim results are seen in Fig. 10. It is seen that all four input signals are at a value of "1," the "Enable" is at "1," and the "Reset" is at "0," so when the clock transitions to "1," the "LB_Out" transitions to "1" (due to "AND" logic) and stays there for the next clock cycle. As the "Mux_S" signal is "0," the register is not bypassed.

When discussing the "OR" gate CLB, an inductive analysis of one time step was chosen, with the results shown in Table 8. It was seen that each input was assigned a different logic state. Owing to the "OR" gate and the IEEE 1164 definition, the output of the "OR" gate is a "1." The value is then stored in the register, as the "Enable" signal is "1," "Reset" signal is "0," and the clock transitions on the rising edge ("+"). This value is stored in the register for one time step (stored at TS = 0) due to the "Mux_S" value being "0". The signal stored in the register is then output at the next time step (TS = 1), making the CLB output value equal to "1." The ModelSim results for this simulation are shown in Fig. 11. The "Reset" signal is "0," "Enable" signal is "1," and the "Mux_S" signal is "0"; the register is used again. The inputs this time include several potential error states ("U" and "X"), which eventually resolve to "1" due to "OR" logic when the clock transitions (as discussed in Subsection 3.5).

### 4.4. *Platinum comparator results*

The results for the DFM analysis and ModelSim simulations for the platinum signal compensator are presented here. The models were run for the top events of "Trip" and "Total Flux High," and for "No Trip."

In Table 9, it is seen that all the system components are functioning correctly, except in one instance. The clock transitions "+," the "Reset" signal is "0," and the "Clock Enable" signal is "1," allowing the new (correct) values to be output from the registers. However, it was seen that the value of "D" in the state space model was higher than it should be, causing the value of "Phi" to be above the setpoint. A second input also read high (IV_3), causing the system to trip.

A similar implicant is shown in Table 10; however, in this case the "D" value does not affect the top event. In this case, the "Trip" signal reads "0", in part due to an error with the "Input_Voltage," which has a value of "X." The other nodes/states in the run shown in Table 10 were the same as those in

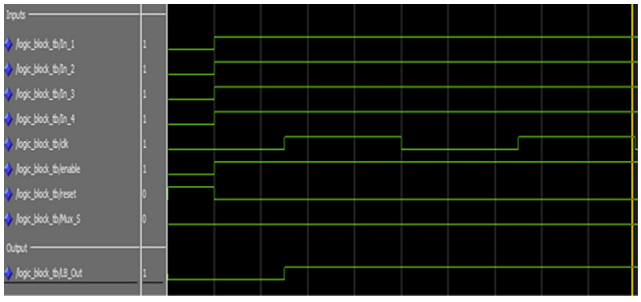


**Fig. 10 – ModelSim results for "AND" logic block "Top Event = 1 at TS = 0 and TS = −1".**

**Table 8 – Sequence for "OR = 1" inductive analysis.**

| Sequence 3 (node) | Sequence 3 (state) | Sequence 3 (time step) |
|---|---|---|
| *In_1* | 1 | 0 |
| *In_2* | 0 | 0 |
| *In_3* | U | 0 |
| *In_4* | X | 0 |
| *Mux_S* | 0 | 0 |
| *Clock_Enable* | 1 | 0 |
| Prev_Clock | + | 0 |
| Reset_Signal | 0 | 0 |
| *Logic_Block_Out* | 0 | 0 |
| *In_1* | 1 | 1 |
| *In_2* | 0 | 1 |
| *In_3* | U | 1 |
| *In_4* | X | 1 |
| *Mux_S* | 0 | 1 |
| *Clock_Enable* | 1 | 1 |
| *Clock* | 1 | 1 |
| *Reset_Signal* | 0 | 1 |
| *Logic_Block_Out* | 1 | 1 |

Table 9, therefore, only the differing states were included in Table 10.

Fig. 12 shows the ModelSim results for Table 9. The high value for "D" causes the flux to read "High" and assist in causing a trip (false alarm), when one of the other channels also denotes a trip. In this case, the ModelSim results do not specifically show the value for "D" (internal signals are not always monitored). However, it is seen in the ModelSim results that the flux value ("Out1") is higher than what it should be, for the given inputs. Fig. 13 gives the simulation results for Table 10, where it shows "No Trip" due to some failure with the input. The "Input_Voltage" node ("In1" in Fig. 13) has an input of "X," which points to some form of input failures, including those from other parts of the system (as discussed in Subsection 3.5). This leads to the value of "Phi" to also be "X," as seen in "Out1." This causes the two out of three logics to return a value of "0," and the "Trip" value is seen as "0" in the figure. This would have the potential to miss a trip, if the value for that input was supposed to be high enough.

## 5. Discussion

This section discusses some of the advantages of DFM discovered in this study when modeling FPGA-based systems.

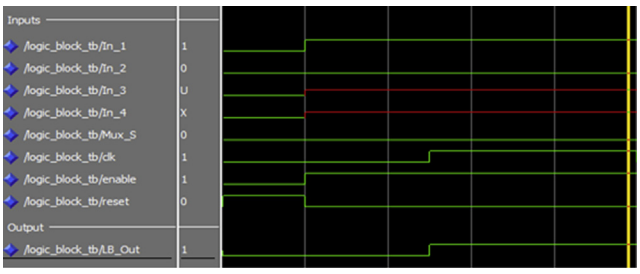


**Fig. 11 – ModelSim results for "OR" logic block inductive analysis.**

**Table 9 – Implicant for "Trip" and "Total Flux High."**

| Implicant 11 (node) | Implicant 11 (state) | Implicant 11 (time step) |
|---|---|---|
| *Clock* | ± | −1 |
| Clock_En_State | 1 | −1 |
| *Clock_En_Sig* | *En_1* | −1 |
| Clock_Period | Normal | −1 |
| *D* | *D_High* | −1 |
| I(k)_2 | I(k)_2_Correct | −1 |
| I(k)_3 | I(k)_3_Correct | −1 |
| Input_Voltage | Correct Voltage | −1 |
| Prev_Clock | 0 | −1 |
| Prev_Input_1 | Correct_Input | −1 |
| Prev_Input_2 | Correct_Input | −1 |
| Register_1_Out | Reg_Input_Correct | −1 |
| Register_2_Out | Reg_Input_Correct | −1 |
| Clock_Period | Normal | 0 |
| *D* | *D_High* | 0 |
| *Phi* | *Total Flux High* | 0 |
| *I(k)_2* | *I(k)_2_Correct* | 0 |
| *I(k)_3* | *I(k)_3_High* | 0 |
| IV_3 | High_Voltage_3 | 0 |
| Input_Voltage | Correct_Voltage | 0 |
| Register_1_Out | Reg_Input_Correct | 0 |
| Register_2_Out | Reg_Input_Correct | |

DFM is compared with static methods, simulation, and formal methods, as presented in the following subsections.

### 5.1. *Advantages of DFM over static methods*

As seen from the analyses in this section, the "Clock" signal is a very important input for FPGA-based systems. The clock states and edge trigger will determine if the data are output through the register(s). However, the clock period and duty cycle can also affect the system.

These effects are seen in the following examples using the platinum comparator. A separate testbench was created to introduce a sine wave into the system. It was seen that the 10-nanosecond clock period will output the calculated neutron power much faster than a 25-nanosecond clock period. The "Odd Cycle" response changes the standard 50% duty cycle to a 75% "1" and 25% "0" clock, with a period of 25 nanoseconds. No effect on the output signal was seen. This is because the register transitions are on clock edges only, so the duty clock cycle does not affect the output value. In the DFM model, the "Clock_Period" node could introduce a clock delay. The DFM model analysis then returned a clock delay for the "Clock_Period" node that was longer than the standard. This means that the DFM model can predict what could cause the clock delay and the effect they could have on the system. While the clock periods are usually very fast (of the order of nanoseconds), long delays can slow down the calculations and lengthen the trip time.

An example of this is seen in Table 11. A clock delay causes the "Clock_Period" to be "Long," i.e., longer than it should be, meaning that it will not undergo transition when it is supposed to. Even though the "Input_voltage" changes to "High," the clock delay disallows the register to output the new value, causing the value for "Phi" to be read as "Normal" and not as "High". In the end, this causes the voting logic to indicate a "No

**Table 10 – Implicant for "No Trip."**

| Implicant 3 (node) | Implicant 3 (state) | Implicant 3 (time step) |
|---|---|---|
| *Input Voltage* | *Input Voltage_X* | −1 |
| D | D_Correct | −1 |
| *I(k)_3* | X | 0 |
| *Phi* | *Total_Flux_X* | 0 |
| D | D_Correct | 0 |

Trip". In real operation, this would lead to a delay in the trip signal, as the trip would occur once the next clock transition occurs. Modeling of the clock signals represents a potential advantage of DFM over traditional (static) reliability analysis methods. The clock is dynamic, as it will change its state ("0" → "1") based on the clock period. In order to properly model the effects of the clock, including clock delays, the analysis must be able to capture these clock effects.

### 5.2. *Advantages of DFM over simulation*

It was seen in Section 7 that DFM is able to model the important logic and components of FPGAs, as well as the FPGA-based test system. The ModelSim simulations were intended to prove that DFM could be used accurately for that purpose. However DFM also possesses certain advantages over simulators. ModelSim would work inductively, giving outputs based on the inputs combinations. However, ModelSim will not identify exactly what combinations of events resulted in those outputs. It will just produce the outputs. DFM, on the other hand, will provide a list of PIs that will breakdown the potential cause of the certain events. Another advantage of DFM is modeling error states. With ModelSim, it is difficult to include the effect of hardware failures, whereas it is more easily accomplished in DFM using additional error nodes/states. ModelSim would provide evidence of logic errors. However DFM can be used for that purpose as well. Lastly, DFM can include probabilities in the model, for use in probabilistic analyses, while ModelSim cannot.

### 5.3. *Comparison of DFM and formal methods*

In this study, DFM was selected to analyze the FPGA logic; however, other methods could also be used, such as formal methods. Formal methods can be defined as "mathematically rigorous techniques and tools for the specification, design,

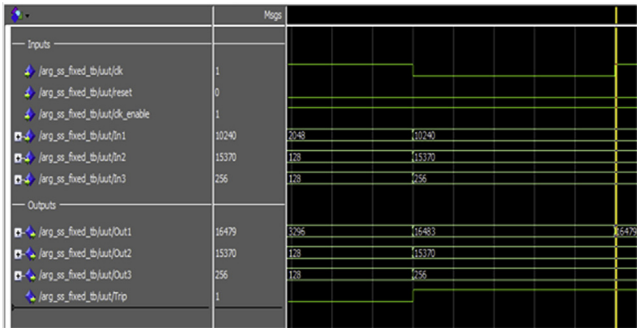


**Fig. 12 – ModelSim results for "Trip" and "Total Flux High."**

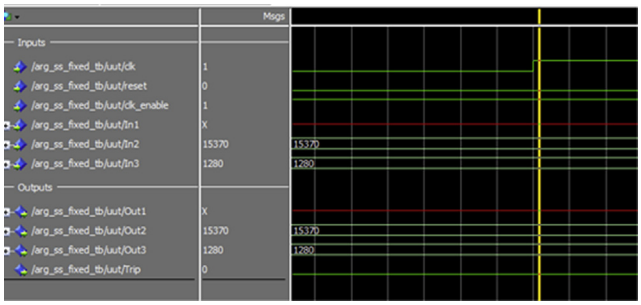


**Fig. 13 – ModelSim results for "No Trip."**

and verification of software and hardware systems" [32]. Formal methods utilize mathematical representations of systems (software and/or hardware), to mathematically verify that the system functions as intended for all input combinations and as such can uncover errors in the software/hardware. DFM and formal methods have some similarities, as they both can be used to analyze the software and hardware components of digital systems, and uncover errors in the system. Additionally, both methods run into issues for very complex, realistic systems, as it may not be feasible to fully analyze a complex system using either method. Formal methods have also seen use in the verification process of FPGA-based systems [33]. However, it should be noted that at the time of this study, formal methods have been used much more widely than DFM.

Significant differences exist between DFM and formal methods. DFM does not rely on the mathematically rigorous formulations that formal methods do; instead it employs user-input decision tables and PI/multivalued logic functions to analyze the system. Although DFM has seen some use in the direct validation of software logic (PIs that do not contain any hardware error states imply a software error) [34], DFM is generally used to analyze the causes of a top event or the

**Table 11 – Prime implicants for "Missed Trip" due to clock delays.**

| Implicant 1 (node) | Implicant 1 (state) | Implicant 1 (time step) |
|---|---|---|
| Clock_En_State | 1 | −1 |
| Clock_En_Sig | En_1 | −1 |
| *Clock_Period* | *Long* | −1 |
| D | Normal | −1 |
| I(k)_2 | Correct Voltage | −1 |
| *I(k)_3* | *High Voltage* | −1 |
| Input Voltage | Correct Voltage | −1 |
| Prev_Input_1 | Correct_Input | −1 |
| Prev_Input_2 | Correct_Input | −1 |
| Register_1_Out | Reg_Input_Correct | −1 |
| Register_2_Out | Reg_Input_Correct | −1 |
| Reset_State | X | 0 |
| Reset_Signal | R_0 | 0 |
| Clock_Period | Long | 0 |
| I(k)_2 | Correct | 0 |
| *I(k)_3* | *High* | 0 |
| *Phi* | *Total_Flux_Normal* | 0 |
| *Input_Voltage* | *High Voltage* | 0 |
| Register_1_Out | Reg_Input_Correct | 0 |
| Register_2_Out | Reg_Input_Correct | 0 |

effects of an initiating event. For example, in DFM if a top event is set to a "Trip," then all possible scenarios, either correct or erroneous, would be solved and stated in the PIs. With DFM, the probabilities of each PI is given, along with the probabilities of the top event, allowing for quantitative analyses, such as calculation of risk importance measures, or the inclusion of DFM results in probabilistic safety assessment. In contrast, formal methods will mathematically verify that the system works for given inputs, but it does not allow for specific top events to be set, in order to determine the minimal cut sets/PIs, and it will not perform probabilistic calculations.

Although DFM and formal methods share some connections for digital system analysis, they generally serve different purposes. Formal methods are used to mathematically verify a system, whereas DFM performs a reliability analysis more similar to that of FTA or Markov methods; however, at the time of this study, direct comparisons of the use of DFM and formal methods were not seen in the literature.

## 6. Conclusion

In this paper, the DFM has been applied to model and analyze important aspects of FPGA-based systems that could find use in nuclear plant safety and control systems. These aspects included the underlying IEEE 1164 standard for VHDL, the use of registers (D flip-flops), logic blocks, and the implementation of an FPGA-based signal compensator and trip logic system. The analysis results were compared using the ModelSim simulations of the VHDL code, which confirmed that DFM was able to correctly model the FPGA logic and properties, making DFM a potential option for the modeling and simulation of FPGAs and FPGA-based systems. The effects of clock delay in the FPGA were examined with DFM, which showed that the clock error could delay the trip, causing a "Missed Trip". The modeling of clock effects shows one potential advantage of DFM for modeling FPGA-based systems, over traditional static methodologies.

## Conflicts of interest

The authors have no conflicts of interest to declare.

## Acknowledgments

The authors would like to thank the Canadian Nuclear Society (CNS) for their support for this research project. The funding for this research is provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES

[1] J. Ranta, The Current State of FPGA Technology in the Nuclear Domain, VTT Technical Research, Finland, 2012.
[2] International Electrotechnical Commission, Nuclear Power Plants—Instrumentation and Control Important for Safety—Software Aspects for Computer-Based Systems Performing Category B or C Functions, 2004. Geneva, Switzerland.
[3] International Electrotechnical Commission, Nuclear Power Plants—Instrumentation and Control Important to Safety—Development of HDL-Programmed Integrated Circuits for Systems Performing Category A Functions, IEC, Geneva (Switzerland), 2012.
[4] J. She, J. Jiang, Potential improvement of CANDU NPP safety margins by shortening the response time of shutdown systems using FPGA based implementation, Nucl. Eng. Des. 244 (2012) 43–51.
[5] J. Naser, Recommended Approaches and Design Criteria for Application of Field Programmable Gate Arrays in Nuclear Plant Instrumentation and Control, Electric Power Research Institute (EPRI), Palo Alto (CA) USA, 2011.
[6] P. McNelles, L. Lu, A Review of the Current State of FPGA Systems in Nuclear Instrumentation and Control, Proceedings of the 21st International Conference on Nuclear Engineering, Chengdu (China), 2013.
[7] J.-J. Lu, H.-P. Chou, K.-W. Wong, Conceptual Design of FPGA-based RPS for the Lungmen Nuclear Power Plant, NPIC & HMIT, Las Vegas (NV), 2010, pp. 944–953.
[8] H. Huang, H. Chou, C. Lin, Design of a FPGA based ABWR feedwater controller, Nucl. Eng. Technol. 44 (2012) 363–368.
[9] J. She, J. Jiang, On the speed of response of an FPGA-based shutdown system in CANDU nuclear power plants, Nucl. Eng. Des. 241 (2011) 2280–2287.
[10] E.S. Bakhmach, A.D. Herasimenko, V.A. Golovyr, V.S. Kharchenko, Yu.V. m Rozen, A.A. Siora, V.V. Sklyar, V.I. Tokarev, S.V. Vinogradskaya, M.A. Yastrebenetsky, FPGA-Based NPP Instrumentation and Control Systems: Development and Safety Assessment, Radiy, Kirovograd, Kharkiv (Ukraine), 2008.
[11] J. Naser, Guidelines on the Use of Field Programmable Gate Arrays (FPGAs) in Nuclear Power Plant I&C Systems, Electric Power Research Institute (EPRI), Palo Alto (CA) USA, 2009.
[12] International Electrotechnical Commission (IEC), 62566, Development of HDL Programmed Integrated Circuits for Systems Performing Category A Functions, IEC, Geneva (Switzerland), 2012.
[13] International Atomic Energy Agency (IAEA), Application of Field Programmable Gate Arrays in Instrumentation and Control Systems of Nuclear Power Plants, IAEA, Vienna (Austria), 2016.
[14] United States Nuclear Regulatory Commission (U.S. NRC), NUREG-7006, Review Guidelines for Field Programmable Gate Arrays in Nuclear Power Plant Safety Systems, U.S. NRC, Washington (DC), 2010.
[15] ASCA Inc, Dymonda 7.0 Software Guide, ASCA Inc., Redondo Beach (CA), 2013.
[16] T. Aldemir, M.P. Stovsky, J. Kirschenbaum, D. Mandelli, P. Bucci, L.A. Mangan, D.W. Miller, X. Sun, E. Ekici, S. Guarro, M. Yau, B. Johnson, C. Eika, S.A. Arndt, Dynamic Reliability Modeling of Digital Instrumentation and Control Systems for Nuclear Reactor Probabilistic Risk Assessments, U.S. Nuclear Regulatory Commission, Washington (DC), 2007.
[17] T. Aldemir, D.W. Miller, M. Stovsk, J. Kirschenbaum, P. Bucci, L.A. Mangan, A. Fentiman, S.A. Arndt, Methodologies for the probabilistic risk assessment of digital reactor protection and control systems, Nucl. Technol. 159 (2007) 167–191.
[18] S. Authen, J.-E. Holmberg, Reliability analysis of digital systems in a probabilistic risk analysis for nuclear power plants, Nucl. Eng. Technol. 44 (2012) 471–482.
[19] C. Garett, S. Guarro, G. Apostolakis, The dynamic flowgraph methodology for assessing the dependability of embedded software systems, IEEE Trans. Syst. Man Cybern 25 (1995) 824–840.

[20] P. McNelles, L. Lu, Lab-Scale Design, Demonstration and Safety Assessment of an FPGA-based Post-Accident Monitoring System for Westinghouse AP1000 Nuclear Power Plants, Proceedings of the 22nd International Conference on Nuclear Engineering, Prague (Czech Republic), 2014.
[21] A. Al-Dabbagh, L. Lu, Reliability modeling of networked control systems using dynamic flowgraph methodology, Reliab. Eng. Syst. Saf. 95 (2010) 1202–1209.
[22] T. Aldemir, S. Guarro, D. Mandelli, J. Kirschenbaum, L.A. Mangan, P. Bucci, M. Yau, E. Ekici, D.W. Miller, X. Sun, S.A. Arndt, Probabilistic risk assessment modeling of digital instrumentation and control using two dynamic methodologies, Reliab. Eng. Syst. Saf. 95 (2010) 1011–1039.
[23] K. Bjorkman, Solving dynamic flowgraph methodology models using binary decision diagrams Reliab, Eng. Syst. Saf. 111 (2013) 206–216.
[24] Organization for Economic Co-Operation and Development – Nuclear Energy Agency (OECD-NEA), Recommendations on Assessing Digital System Reliability in Probabilistic Risk Assessments of Nuclear Power Plants, OECD-NEA, Paris, France, 2009.
[25] Institute for Electrical and Electronics Engineers, IEEE, 1164-1993—IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Stdlogic1164), IEEE Standard [Internet]. [cited 2015 Dec 20]. Available from: https://standards.ieee.org/findstds/standard/1164-1993.html.
[26] Synario Design Automation, VHDL Reference Manual, Synario Design Automation, Redmond (WA), 1997.
[27] A.K. Mishra, S.R. Shimjith, T.U. Bhatt, A.P. Tiwari, Dynamic compensation of vanadium self powered neutron detectors for use in reactor control, IEEE Trans. Nucl. Sci. 60 (2013) 310–318.
[28] A.K. Mishra, S.R. Shimjith, T.U. Bhatt, A.P. Tiwari, Kalman filter-based dynamic compensator for vanadium self powered neutron detectors, IEEE Trans. Nucl. Sci. 61 (2014) 1360–1368.
[29] G.F. Lynch, R.B. Shields, P.G. Coulter, Characterization of platinum self-powered detectors, IEEE Trans. Nucl. Sci. 24 (1977) 692–695.
[30] W.H. Todt, Characteristics of Self-Powered Neutron Detectors Used in Power Reactors, Imaging and Sensing Technology Corporation, Horseheads (NY), 1997.
[31] M. Borairi, Reactor Regulating System (Lecture Notes), Oshawa (ON, Canada), 2014.
[32] R.W. Butler, What is Formal Methods, Langely Formal Methods [Internet]. NASA, 6 August 2001 [cited 2016 Feb 20]. Available from: http://shemesh.larc.nasa.gov/fm/fm-what.html.
[33] H. Yalin, Exploring Formal Verification Methodology for FPGA-based Digital Systems, Sandhia National Laboratories, Alburquerque (NM), 2012.
[34] M. Yau, G. Apostolakis, S. Guarro, The use of prime implicants in dependability of software controlled systems, Reliab. Eng. Syst. Saf. 62 (1998) 23–32.