2019 22nd Euromicro Conference on Digital System Design (DSD)

Testing Reliability of Smart Electronic Locks: Analysis and the First Steps Towards

Ondrej Cekan, Jakub Podivinsky, Jakub Lojda, Richard Panek, Martin Krcma, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1361, 1360, 1362, 1360, 1223}
Email: {icekan, ipodivinsky, ilojda, ipanek, ikrcma, kotasek}@fit.vutbr.cz

***Abstract*—This research paper presents an analysis of electronic smart locks and explores the influences of faults on its controller unit. Electronic smart locks often utilize stepper motor as an actuator. Stepper motors, however, need a controller, which is usually implemented in a processor. The aim of our research is to examine the consequences of a failing controller processor. In our previous research, we developed a platform for fault tolerance testing with the ability to monitor the impacts on the mechanical part. We also developed a framework for accelerated testing of fault tolerance properties. The processor can be implemented in an FPGA (Field Programmable Gate Array) in order to be able to emulate HW faults inside the processor. In this paper, the concept of testing a smart lock is presented alongside with the first experimental results utilizing the direct generation of invalid stimuli for the stepper motor. In our research, we found out that random errors probably could not be used for an unauthorized unlock, especially if the lock utilizes a mechanical gearbox. Deeper logic and knowledge of the correct sequence of steps used by the selected motor are needed to perform an attack to unlock the lock. On the other hand, random sequences could cause that lock not to be locked by falsifying the lock request sequence. The second interesting fact is that x% of faults in the valid sequence give the same rotation angle as (100−x)% of faults.**

***Keywords*—Electronic Lock, Stepper Motor, FPGA, Fault Tolerance, Stimuli Generation, Reconfiguration.**

I. Introduction

In today's world, various electronic systems are more and more used and we increasingly meet them in our everyday life. Nowadays, *smart devices* [1] are on the rise, the goal of which is to make our lives more efficient, simpler and more pleasant. The smart device is a device that uses some communication protocol for connecting to another device or network and enables interaction or remote control. As an example smart thermostat can serve, a smart car, smart speaker or a well known smartphone. The smart electronic lock [2] is an example of another smart device which we meet in our everyday life. Connecting electronics with mechanical elements and a remote server brings new possibilities for users to control these locks. The smart electronic locks have many advantages, especially since they can be controlled remotely. The use of electronic locks is widely used in companies where they are combined with the access system and allow, for example, easy adjustment of the access for individual employees to various parts of the company. Electronic locks also bring benefits for home use, whether for short-term rentals of flats or for remote adding access rights to a service technician or a housekeeper.

Electronic systems are exposed to various influences that can cause faults in such devices, especially in various environments with an increased occurrence of charged particles, electrostatic electricity, etc. However, faults in electronic systems can also be induced artificially, usually with ulterior motives. Intentional fault injection uses, for example, bumping attacks which are aimed at data extraction from secure embedded memory, which usually stores critical parts of algorithms, sensitive data and cryptographic keys [3]. As an another example, smart cards are often the target of software or hardware attacks. The most recent attack is based on fault injection which modifies the behavior of the application [4]. In the case of electronic locks, opening a lock by gaining privileges or without privileges is particularly critical. Thanks to unauthorized or accidental unlocking, property can be damaged, financial losses can occur and even human health or life can be in danger. Unauthorized acquiring permission or unauthorized unlocking can be caused by various types of attacks, whether to the server part of the whole system, or directly to the lock installed on the door [5]. One of the ways for unauthorized opening can be a deliberately induced fault, which is not up to now an explored topic that we will deal with in our work.

In our research, we are dealing with designing a fault-tolerant system and evaluating the effects of faults on electro-mechanical systems. We focus primarily on systems which are built on SRAM-based FPGAs. The electronic lock is just another example of such an electro-mechanical system, where the occurrence of a fault can lead to undesirable consequences. In this work, we plan to use gained knowledge and experience and apply previously developed tools to analyze the impact of faults on electronic locks. Electronic locks are usually controlled by an embedded processor that can be implemented in an FPGA which gives us the possibility for a quite easy modification and fault simulation. The result of the research then will not only be the evaluation of the impact of faults, but also the evaluation of the possibility of using general approaches to guarantee fault tolerance.

This paper is organized as follows. Electronic lock analysis is done in Section II. The goals of our research are proposed in Section III. Section IV introduces an evaluation platform for monitoring the impact of faults on electro-mechanical applications and Fault Tolerance ESTimation (FT-EST) framework for the accelerated evaluation of faults impact on electronic design. Simulation tools for stepper motor simulation are the topic of Section V. Section VI presents the use of universal stimuli generator for input stimuli generation. Experiments with stimuli generation for stepper motor are presented in Section VII. Section VIII concludes the paper and presents plans for our future research.

II. Electronic lock analysis

Smart electronic locks are quite complex electronic devices that use many of the latest technologies. The basis of the lock can be summarized in three blocks - control module, I/O module and motor module [6]. The control module is the main core of the entire lock. Typically, a processor is used because it performs a number of computationally intensive operations - peripheral handling, locking decision algorithm or the motor control. The lock is connected via I/O modules with its peripherals, including simple sensors, displays, card readers, or other network communication technologies like Wi-Fi, Bluetooth, GSM, etc. The last but equally important block is the motor module which performs the operation with the mechanical part of the lock.

The motor can be a conventional brushed motor with circular motion that requires a motor drive which operates the motor as an actuator [7]. Another option is to use a different type of motor that already offers precise angle control, linear positioning or speed control. This type may be a servomotor or stepper motor. The servomotor is used in devices where a very precise position plays an important role (e.g. laser cutting machines). In other less precise devices, the stepper motor is sufficient. The stepper motor has to be turned into a known position before it is activated, since it does not offer feedback about its position. It is very often found in electronic locks [8] [9] (we found it also in a car door lock), so we also focus on it in our work.

The stepper motor is a DC electric motor whose full rotation can be divided into several equal steps. The stepper motor is controlled by input pulses (typically square pulses) that precisely rotate the shaft position based on an angle which is given by the number of motor steps. It consists of a cylindrical rotor, a number of stators, a number of yokes, and a set of coils [10]. Different types of stepper motors exist that vary in their internal magnetic field configuration. In our work, we have chosen a conventional bipolar stepper motor with a permanent magnet in its rotor which operates on the attraction or repulsion between the rotor and the stator electromagnets. The cut section of such a motor is shown in Figure 1.

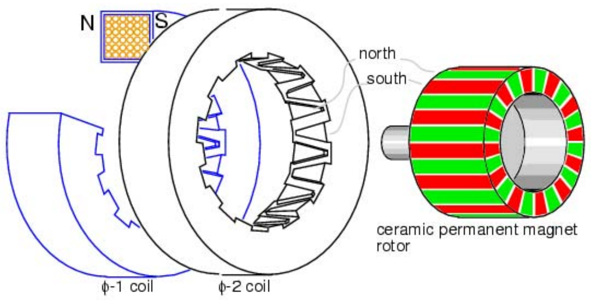


Fig. 1: The cut of stepper motor with a permanent magnet in rotor. [11]

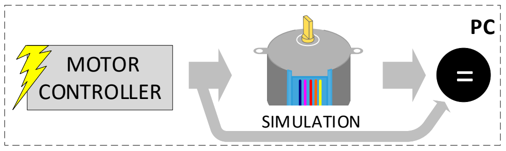
The particular model of this type of stepper motor, that we have chosen, is 28BYJ-48 [12]. It is a small stepper motor operating at 5V. It is the motor equipped with 1/64 transmission gearbox. It has 4 phases with a single step angle of 5.625/64. It is needed to perform 4,096 steps (64 steps without the gearbox) for the full rotation.

III. The Goals of the Research

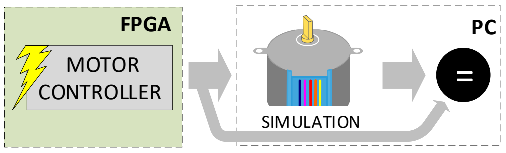
Various tools for fault tolerance evaluation [13], [14], [15] were developed during our previous work. In this work, we propose the use of these tools for the evaluation of the impact of faults on stepper motor, which is the core of an electronic lock. We have identified three main goals that we are going to achieve during our future research:

1) *The evalution of faults injected directly into stepper motor control signals and estimation of the risk of unauthorized unlocking.*
2) *Implement the stepper motor controller with a processor configured into an FPGA and evaluate: a) the impact of faults injected into memory elements of processor and b) possibility of unauthorized unlocking.*
3) *Verify the possibility of using standard fault tolerance techniques for eliminating unauthorized lock unlocking.*

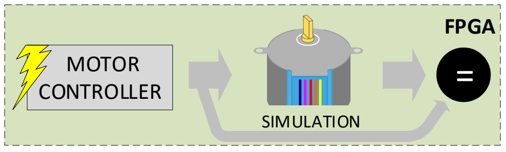
During the solving of the goals we will use three levels of evaluation (Figure 2) with various architectures of component realization and interconnection. Figure 2a shows the architecture, where all components are running on PC which allows us rapid prototyping and evaluation. The problem is that the results are not realistic because there is no real electronic controller into which real faults can be injected. This architecture will be used for monitoring the impact of faults in stepper motor control signals. The second architecture based on the interconnection of a PC and an FPGA is shown in Figure 2b. The electronic controller running on the FPGA allows us to inject a real fault into the FPGA and realistic results are obtained both for electronic output and the behavior of mechanical part. The evaluation is quite slow due to the PC and the FPGA communication. Moreover, communication brings synchronization issues between the PC and the FPGA. The last architecture, shown in Figure 2c, accelerates the evaluation and allows us to perform a more exhaustive evaluation thanks to the full implementation in the FPGA. The problem is that the simulation of a mechanical part is impossible for more complex electro-mechanical systems.



(a) All components running on PC.



(b) The electronic controller running on FPGA and motor simulation performed on PC.



(c) All components running on FPGA.

Fig. 2: Three level of evaluation with various speed and accuracy.

IV. The Evaluation Platform

Two evaluation platforms were developed in our research group. The first one [13] allow us to monitor the impact of faults both on the electronic and mechanical part. This platform corresponds to Figure 2b. The second one [14] is a framework fully implemented in an FPGA which proposes faster evaluation, but monitors just the impact of faults on the electronic part. This platform can be used as a basis of the architecture shown in Figure 2c. These tools are presented in the following sections.

A. *The Functional Verification-based Evaluation Platform*

A previously developed platform for monitoring the impact of faults on electro-mechanical system is presented in [13]. The platform is based on a functional verification technique [16]. The concept and basic components are shown in Figure 3. The electronic control unit is implemented in an FPGA and we use the ML506 evaluation board with Virtex 5 FPGA. The fault injector, mechanical part simulation and software part of the verification environment are running on a computer. The mechanical part communicates with an FPGA through the Ethernet interface. Faults are injected through the JTAG interface directly to the FPGA.

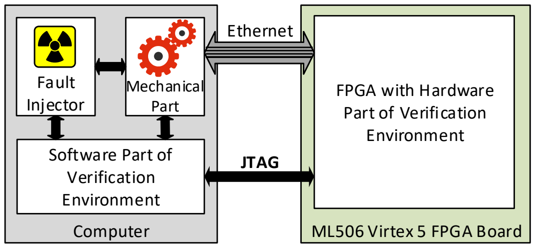

Fault Injector
Mechanical Part
Ethernet
FPGA with Hardware Part of Verification Environment
Software Part of Verification Environment
JTAG
Computer
ML506 Virtex 5 FPGA Board

Fig. 3: The genral architecture of the developed evaluation platform.

The basic concept of the use of functional verification as a tool for monitoring the impact of faults on electro-mechanical system is presented in Figure 4. The difference compared to conventional functional verification is that the Device Under Test (DUT) is moved into an FPGA which allow us to inject faults and monitor its impact. The use of the proposed platform was demonstrated on a robot in maze case study, but this platform is designed to be generally usable with various systems. The versatility of the proposed platform is based especially on the fact that the verification environment is available for an evaluated system, because functional verification is usually used during electronic systems development. Therefore, the verification environment and the reference model (the most important elements dependent on the evaluated system) are available from the previous stage of system development and can be used as a fault tolerance evaluation. The verification scenario generation is usually a part of the verification environment or we can use our previously developed universal generator [15]. An important condition for using the platform is that an electronic controller can be configured into an FPGA. Moving the DUT into the FPGA and proper communication with the rest of verification environment ensure *driver* and *monitor* components. These components are partly universal, but they need to be customized for a particular DUT.

The mechanical part is also an important element, which allows us to monitor the impact of faults not only on electronics, but also on mechanics. It is not important whether it is a real mechanical part or its simulation. It is important that sensors that provide feedback on the behavior of the mechanical part are available. The values provided by these sensors are processed on the verification environment, which check if the system behaves according to its specification. Usually, the use of simulation allows us to speed up testing and is usually cheaper than the use of a real mechanical device.

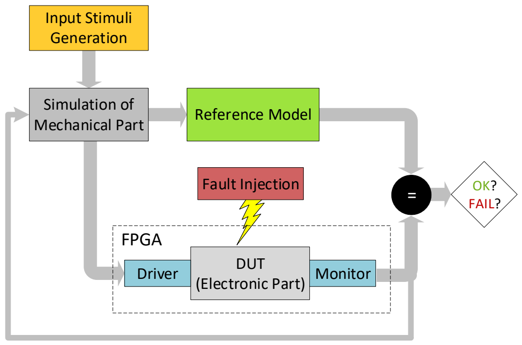


Fig. 4: The general concept of the use of functional verification for monitoring impact of faults.

In this work, we deal with monitoring the impact of faults on the electronic lock, more precisely on the main mechanical element, which is the stepper motor. In this case, the verification environment is modified since no feedback is used when the stepper motor is controlled, only control signals ensuring the stepper motor correct operation are generated. To monitor the behavior of a mechanical part, which is the stepper motor, it is advantageous to use a measurement of the continuous and resulting angle of rotation. The angle measurement can be easily realized both in the case of experiments with a real stepper motor (e.g. a stepper motor can rotate a potentiometer), and in the case of simulation, which gives us this information automatically. The use of the functional verification for monitoring the impact of faults injected into the stepper motor controller implemented in an FPGA is shown in Figure 5.

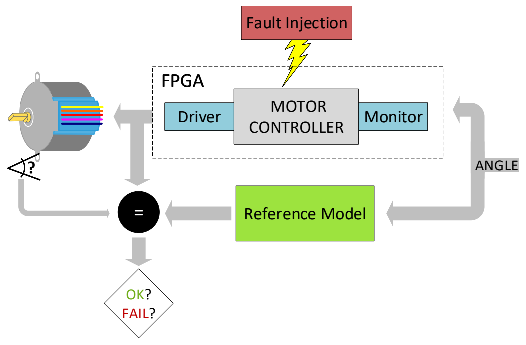


Fig. 5: The use of functional verification for monitoring impact of faults on stepper motor controller.

For the successful implementation of the above mentioned concept of fault tolerance verification, we need the simulation of the mechanical part, which is a stepper motor simulator. We found a suitable stepper motor simulator that is described in Section V.

B. The FT-EST Framework

In our previous research paper [14], we developed a new framework specialized on the acceleration of the evaluation process. The framework also aims at minimizing the user interactions needed to execute a test, as this framework is originally intended to be a part of our fault tolerance design automation toolkit. We call this tool the *Fault Tolerance ESTimation* (FT-EST) framework. Results obtained through the FT-EST can be automatically analyzed using our approach presented in [17]. The FT-EST framework also utilizes the concept of *functional verification*, which means the tested component is instantiated on the actual HW (i.e. a Virtex 5 Xilinx FPGA in our case) and to hold a test, at least two instances of the component must be made: 1) a *golden* unit, which serves reference results and is not a subject to fault injection; and 2) the tested unit, which we call the *Unit Under Test* (UUT). Faults are injected particularly into the area of the UUT occupied on the FPGA. The number of UUTs can be increased in order to run the test in parallel for each unit. The framework is prepared for this functionality, and, thus, the multiple instantiation is achieved through automatic generation and is completely seamless for the user. The framework utilizes several acceleration techniques, for example:

- test of multiple component instances,
- stimuli are generated and outputs are compared on the FPGA to maximize throughput,
- for each test, only the tested UUTs are repaired,
- tests are executed autonomously to minimize latency.

The FT-EST framework consists of the HW part, which is synthesized and run on the FPGA, and also of the SW part, which is executed on the PC and starts and stops the operation of the HW part and also saves the results obtained from the HW into a file. The HW part is written in VHDL and is usually synthesized and uploaded to the FPGA together with the UUT components. The HW part is composed of various componens each of which has its function, making potential changes of the framework much easier. The UUTs are instantiated inside the *Unit Instantiation Area* (UIA). Input stimuli are generated inside the *Input Generation Unit* (IGU), while the outputs are compared in the *Output Compare Unit* (OCU) and detected failures are captured in the *Failure Capture Unit* (FCU). The experimentation on the HW part is autonomously driven by the FSM inside of the *eXperiment Control Unit* (XCU). XCU can be modified to achieve various experimentation flows. The communication with the HW part of the framework can be divided into two main blocks: communication registers are located in the technology-independent *Communication Interface* (CIF), while the *Communication Module* (CM) implements the particular technology-dependent communication. In our case, as we are using Xilinx technology in combination with the JTAG (*Joint Test Action Group*) and *Universal Serial Bus* (USB) interfaces, the CM is utilizing the ChipScope Pro *Integrated Controller* (ICON) core [18] in cooperation with the *Virtual Input/Output* (VIO) cores [19]. The SW part utilizes the *Fault Injector* [20], which was previously developed in our research group. The communication from the SW is achieved through the *ChipScope Tcl Engine Interface* [21], counterparting the CM. The *experimental loop* implements the SEU cycle. The architecture of the framework is shown in Figure 6.

While using the framework, for the exhaustive test of each bit of the bitstream (i.e. the common use case), testing consists of two cycles: 1) during the *test cycle*, all the stimuli in the prescribed order are replayed on the inputs of the UUT, resulting in the full test of the UUT; 2) during the *SEU cycle*, one fault is injected and the impacts of the fault are tested by executing the test cycle. Thus, for the exhaustive testing, the number of SEU cycles corresponds to the number of tested bits of the bitstream.

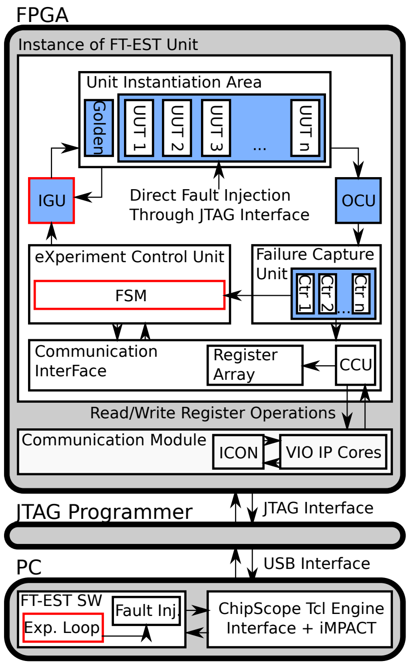


Fig. 6: Simplified architecture of the FT-EST system; the parts highlighted in blue are dynamically and fully automatically generated, while the parts highlighted in red are to be provided by the designer to specify the experiment setup.

In this research, we would like to move the experimentation fully to the FPGA to accelerate the evaluation. However, to accelerate the evaluation of the critical bits representation, HW implementation of the stepper motor simulation must be designed. In such approach, each UUT is assigned one simulator. The OCU monitors the UUTs outputs to distinguish if, or for how long, the stepper motor was in a position that could be considered as *unlocked*. This approach would allow us to significantly lower the time needed for one test, and, thus, result in a possibility to exhaustively evaluate each bit of the bitstream belonging to the implementation of the controller unit. Such approach would also allow us to utilize the additional information obtained through the simulation in the hardening of the system using our FT design automation toolkit, as the in-HW mechanical simulation would allow us to monitor the impacts on the mechanical part even while using the accelerated framework, resulting in a faster evaluation without compromising on accuracy of the results. It should be noted that without the in-HW simulation, the FT design automation toolkit would be fully dependent only on the actual critical bits percentage in order to evaluate a particular design. A possible implementation of HW simulation of the stepper motor is described in Section V-B.

V. The Stepper Motor Simulation

The realization of the simulation is described in the following sections. We need two versions of the simulation: 1) the SW version that could be used with the evaluation platform considering the mechanics which was described in Section IV-A and also 2) the HW implementation that would run on an FPGA for its usage with the FT-EST framework which was described in Section IV-B.

A. Software Implementation

We decided to utilize MATLAB and Simulink [22] for the simulation of a mechanical equipment such as a stepper motor. Moreover, we do not need to detail the stepper motor modelling because the required model is prepared in the Simscape library [23], which is part of the Simulink package. This model is generic for the whole class of stepper motors, therefore, it is necessary to select a particular option which corresponds with a real motor chosen by us. Thus we have to ascertain our stepper-motor parameters for using it. Some of them are in the corresponding datasheet [12] and the rest has to be measured.

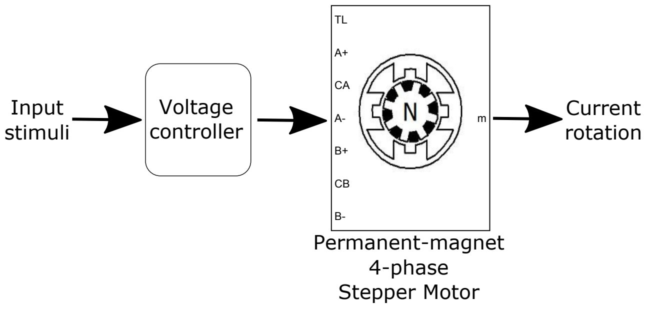


Fig. 7: MATLAB & Simulink: stepper-motor simulation diagram.

It is necessary to model the whole system such as power management and motor control besides the stepper motor itself for a correct simulation. Our model for Simulink is shown in Figure 7. Stimuli for the stepper-motor control, which are generated outside (more in Section VI), are brought to the model input port. The first part of this model is a voltage controller which is responsible for converting logic inputs to the electric impulses which excite the motor coils. An electric power source for the motor is modelled as a constituent of this voltage controller. A subsequent part is the stepper-motor model itself with specific parameters. More accurately, it is the 4-phase stepper motor with a permanent-magnet rotor. The sequence of involving the 4 motor coils to perform one rotation sequence is shown in Table I. the required output from the whole model is the current angle of the motor turning. Fortunately, this is an output from the model of the motor too. For the possibility of a detailed analysis of the motor behaviour, outputs are in the format of a time series which means that data with a current motor rotation are provided by time stamps. As a result, the current motor position in every moment is known.

TABLE I: 8 steps required to perform 1 motor sequence.

STEP	1	2	3	4	5	6	7	8
COIL1	1	1	0	0	0	0	0	1
COIL2	0	1	1	1	0	0	0	0
COIL3	0	0	0	1	1	1	0	0
COIL4	0	0	0	0	0	1	1	1

B. Hardware Implementation

In order to perform more exhaustive experiments with the motor we face a problem with time consumption of the software simulation in MATLAB. Taking several minutes for each simulation run, the time of exploring the entire state space of the motor will be extremely long. As we are going to perform more exhaustive experiments with different parameters, we need to take this into account. To deal with the problem, we decided to use hardware acceleration according to the architecture shown in Figure 2c to speed up the simulation. The motor simulation itself is also supposed to be implemented in the FPGA in order to utilize its performance and also to make communication between the controller and the motor simulation as easy as possible in order to avoid any limitations of the throughput.

The simulation may be implemented in different ways. We may implement a simple processor design into an FPGA and use it to run a simplified simulation as the computation of a set of differential equations which are used in the MATLAB library [23]. This approach is good for its fast development and deployment in the experiments; however, it suffers from a limited performance of the FPGA implemented processor. Another approach may be to design a special computing unit optimized to compute the specific equations with sufficient speed. This approach may prove to be the fastest solution possible. However, the design and the deployment period may be far longer as it requires a special approach to optimize needed mathematical operations and ensure their correct order. The computation precision may also prove to be a problem as floating point arithmetic units are costly to implement and they may slow the whole computation to the level that neglects any reached simulation speed up. In order to keep the performance level we might need to use a fixed point arithmetic which may lead to aforementioned problems with the computation precision.

Another approach we considered is to utilize neural networks as we dispose of efficient FPGA implementation that may be used to optimize the computation using the implementation massive parallelism. In this approach the motor will need to be modeled using a feedback type neural networks as the Hopfield networks or the Boltzmann machines [24]. However, this approach may suffer from an inability to simulate inertia related parameters of the motor. Also, the computation of neural network may also suffer from limited precision due to the usage of the fixed point arithmetic. The feedback design of neural network may also cause the precision limits to accumulate in the iterative computation of the network output and the feedback.

VI. Stimuli Generation

Input stimuli generation is a rather important process for verifying the correct behavior of each designed system. By applying different input stimuli, it is possible to check the correctness of the design and detect possible implementation errors during the development process. Advantageously, the generation of stimuli can be used in the area of fault-tolerant systems for verifying the impact of a fault on system behavior.

In stimuli generation for the stepper motor, we use our previously designed universal stimuli generator which is described in Section VI-A. The application of this generator to the selected stepper motor is described in Section VI-B.

A. Universal Stimuli Generation

Universal stimuli generation (USG) is based on our design which uses two input structures that define a general grammatical system and constraints. Constraints give the grammar more expression power. Consequently, it is possible not only to create stimuli defined by their grammar, but also to dynamically control the entire process of the application of production rules of the grammar during the generation. Together these two input structures create our new grammatical system which is called probabilistic constrained grammar (PCG). It is a context-free grammar where the production rules have defined probability of replacement, and the constraints that can dynamically change this probability during the stimulus generation. More information about this new grammar can be found in our previous work [25]. The input stimulus is composed of production rules of the grammar during the generation process. Each system must have defined its own grammar. In each generation step, the constraints are verified. The constraints limit all possible solutions and only ensure a correct input stimulus composition that is valid in a given combination of generation steps. We have previously applied this universal generator to generate assembly instructions for processors, mazes for a robot controller, software fault tolerance, or generate input values as an expected result for an arithmetic logic unit. Now we use it to generate stepper motor stimuli.

B. Stimuli Generation for Stepper Motor

In the case of stepper motor verification, two types of stimuli can be generated. The first type is the generation of complete verification scenarios to evaluate the impact of fault injection on the controller of the lock. The controller then interprets the input stimuli to specific stepper motor commands. However, since the controller for the lock is not available at this time, we will not deal with this type in this work. The second type is to generate affected stimuli directly into the stepper motor and verify it without the controller. The stimuli thus represent specific input values for the selected stepper motor. In fact, the controller with a fault injection can be replaced by the generator. As a result, we are able to find out how the stepper motor behaves with different values on its input pins using only a stimulus generator. We will deal with this type in the following paragraphs.

We generate input stimuli for the available model of a stepper motor that ispart of the MATLAB and Simulink package. This model requires the voltage level on its inputs for its 4 coils (0V or 5V) changing over time - stepping. For this reason, we have to encode each step into a pattern. Each pattern is composed of five values, where the first value represents the time stamp in which the step occurs. The remaining four values represent the voltage levels on the four coils ($COIL1 - COIL4$). We use logical levels to define the appropriate voltage (logical 0 = 0V, logical 1 = 5V). The resulting pattern of the step of the stepper motor is as follows:

```
TIMESTAMP,COIL1,COIL2,COIL3,COIL4
```

Many steps must be defined (tens to thousands) to turn the stepper motor around its entire axis one or more times. Several steps obtained during the generation process define one input stimulus. For such a proposed step pattern, it is necessary to respectively create a PCG and production rules of the grammar which will generate these steps. Some test scenarios do not require deeper logic and, therefore, individual steps can be stacked randomly without deeper continuity. However, in the case of a valid continuous motor rotation in one direction (or damaging such input by a fault), it is necessary to keep a valid sequence of stepper motor coils. In the case of a 4-phase stepper motor, 8 steps are required. If it is a stepper motor with a gearbox, many more steps are needed.

The PCG for a 4-phase stepper motor is built to be able to generate both a valid stimulus and its various modifications. It also includes a completely random sequence of steps or the possibility of prioritizing a particular logical value. The correct continuity of timestamps and steps for a valid one-way turn ensures constraints that define the valid sequences of production rules that can be selected. Several parameters can be set before creating the resulting PCG:

time	- maximum timestamp value (end of generation).
minstep	- minimum time between steps.
maxstep	- maximum time between steps.
pcoil	- probability of logical 0 or 1 for each coil.

There is no space to show concrete grammar and constraints, so at least we will describe the generation process. The input structures, which contain the definitions of production rules and constraints, are preprocessed first. Certain patterns in the definition of structures are unpacked and supplemented with specific values. The result of preprocessing is a complete PCG. PCG is processed with our stimulus generator which performs the replacement of the leftmost non-terminal of the grammar. After each replacement, the constraints that can affect any grammar rule are verified. The generation process ends when the output string no longer contains any non-terminal symbols that can be replaced. The output of the generation is in our case a text file which forms the input stimulus. The lines of the file represent the steps for the stepper motor. This file is the input stimulus for stepper motor simulation in MATLAB. The process of stimulus generation is shown in Figure 8.

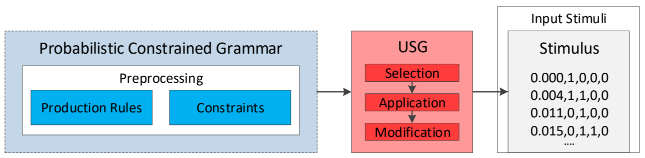


Fig. 8: The process of universal stimuli generation for a stepper motor.

VII. Experiments and Experimental results

As a first step towards meeting the goals of our research, we performed experiments with stimuli generation for direct control of the stepper motor. The aim of these experiments is to verify the possibility of deliberately unlocking or the impossibility of locking the electronic lock. The key part of this verification is the stimuli generator on which the creation of the required test scenarios that lead to this behavior depends. For this reason, the set of probabilistic constrained grammars with the possibility of parametrization have been designed. The step pattern of the stepper motor, from which the stimulus is composed, was shown in the previous section.

The production rules of the grammars have been designed to generate:

1) The valid sequence of steps to rotate the stepper motor in one direction several times around its entire axis.
2) The valid sequence of steps with the percentage representation of faults $(1-100\,\%)$ that will invert a certain number of ones and zeros in the stimulus (it applies to patterns $COIL1-COIL4$).
   For example $25\,\%$ = $\frac{1}{4}$ valid pattern bits is negated, $100\,\%$ = all bits of all patterns are negated (negation of Tab. I).
3) The random sequences of steps with the percentage representation of ones and zeros in the stimulus.
   For example $25\,\%$ of ones and $75\,\%$ of zeros.

We are able to generate the valid sequences which will lead to the fact that the stepper motor performs the proper motion necessary to unlock/lock the electronic lock. We can also generate faulty sequences in order to detect stepper motor behavior in the event of a malfunction.

The available MATLAB stepper motor model has been set to match our real motor selected in Section II. The motor is set without the gearbox to reduce the number of required steps. For the simulation, time stamps of stimuli have been set so that the first step of the motor starts at time 0 and the last step of the motor ends at time 2.56. The interval between motor steps has been set to 0.005. In this setting, there are always 512 steps in the final stimulus. This setting ensures a sufficient number of rotations of the stepper motor around its entire axis (up to 8 times for a valid sequence) and a sufficient time to evaluate the impact of faults. The selected interval between the steps is also large enough to stabilize the motor when the step is changed, because the available stepper motor model also takes these situations into account.

For our experiments, we generated one stimulus with purely valid sequence of steps (FAULT $0\,\%$) and 1000 of random stimuli for each modified sequence. It includes modification for valid sequences with probability of fault (FAULT $1\,\%$, $5\,\%$, $15\,\%$, $25\,\%$, $50\,\%$, $75\,\%$, $85\,\%$, $95\,\%$) and random sequences of steps with different number of ones and zeros (RAND $50\,\%$('1') : $50\,\%$('0'), RAND $25\,\%$('1') : $75\,\%$('0'), RAND $75\,\%$('1') : $25\,\%$('0')). For these stimuli, we examined the angle of rotation of the motor in degrees. Figures 9, 10 and 11 show three different views of the examined rotation of the stepper motor. Suppose we need at least 6 turns around entire axis of the stepper motor to open the lock. This corresponds to 2160 degrees.

Figure 9 shows the maximum motor rotation during the whole simulation. With increasing faults in the valid sequence, the angle of rotation of the motor decreases. This is due to the gradual interruption of continuity in the sequences of steps for proper motor rotation. In the case of fault $50\,\%$, the stepper motor behavior is the same as the random sequence and its maximum rotation is 217 degrees. With increasing faults causing a gradual shift of all values to its negated form in the sequence of steps, the rotation of the motor is increasing again. The effect of faults is symmetrical. It means that fault $25\,\%$ gives the same rotation as fault $75\,\%$, etc.

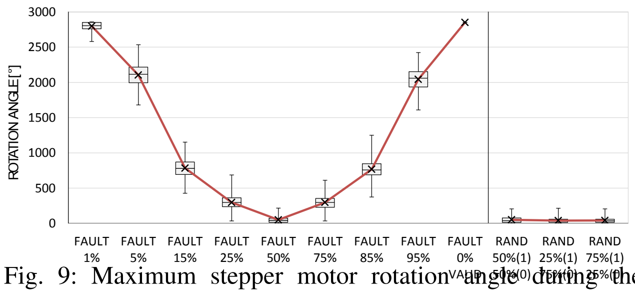


Fig. 9: Maximum stepper motor rotation angle during the whole simulation.

The same behavior described in the previous paragraph can also be seen in Figure 10 which shows the minimum motor rotation during the whole simulation. In the case of fault $50\,\%$ or the random sequence of steps, the minimum rotation of the motor is -237 degrees. It means the stepper motor was rotated in the opposite direction. In other cases, the motor could rotate in the opposite direction due to a fault (negative degree), but eventually began to rotate properly.

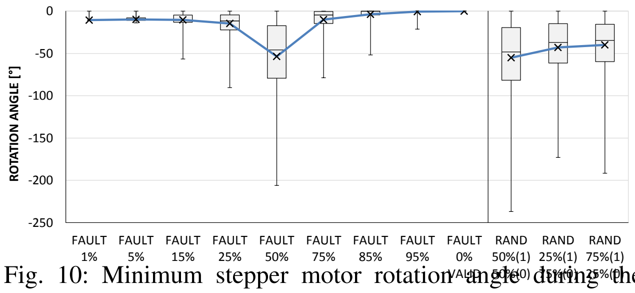


Fig. 10: Minimum stepper motor rotation angle during the whole simulation.

Finally, Figure 11 shows the rotation angle of the stepper motor at the end of the simulation. In the case of fault $50\,\%$ or a random sequence of steps, the motor rotates uncontrolled and alternates the positive and negative directions or stands in place, so there is no significant one-way rotation.

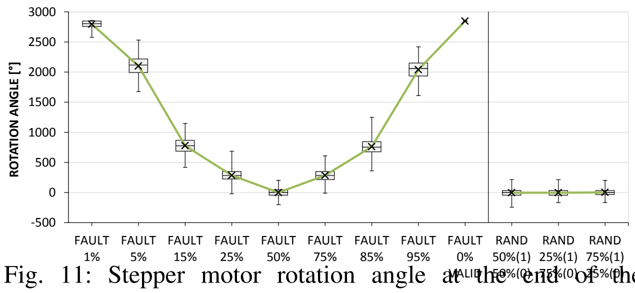


Fig. 11: Stepper motor rotation angle at the end of the simulation.

Experiments have been presented to show that random sequences of steps in our case will not be able to unlock the lock. A deeper logic and knowledge of the correct sequence of steps used by the selected motor are needed to perform the attack to unlock the lock. On the other hand, random sequences could cause that the lock will not be locked by falsifying the lock request sequence. The second interesting fact is that $x\,\%$ of faults in the valid sequence give the same rotation angle as $(100-x)\,\%$ of faults. In the case of faults $(0-15)\,\%$ and faults $(75-100)\,\%$ in a valid sequence, it is possible to unlock the electronic lock in many cases based on the unlock condition that we have defined.

VIII. Conclusions and Future Research

In this work, we presented our first step towards testing the reliability of the smart electronics lock. Our focus is on electronics locks which use the stepper motor as the actuator of the lock. Our research is divided into three phases. In the first phase, the evaluation of the mechanical part is performed purely on SW and simulation basis. The second phase uses the combination of the HW implemented electronics and SW simulation for the evaluation of impact of faults. The last phase is the acceleration of the fault impact evaluation by the FT-EST framework. Using this framework, we are able to accelerate the evaluation by moving the simulation fully into HW. In this paper, we proposed the experimental results obtained through the simulation of the stepper motor in MATLAB. We generated exciting stimuli for the coils of the stepper motor through the universal stimuli generator. This generator is based on a grammar system which allows us to generate both valid and faulty input stimuli. In our experiments, we examined the behavior of the stepper motor by monitoring the rotation angle. From the experiments, we found out that random errors probably could not be used to unauthorized unlock, especially in cases where the lock uses a mechanical gearbox. To unlock the lock, a knowledge of the selected stepper motor is needed, but random step sequences may prevent a door to lock. The interesting fact is that $x\,\%$ of faults in the valid sequence give the same rotation angle as $(100-x)\,\%$ of faults. We plan to continue to fulfill the set goals, i.e. to use both the platforms for monitoring the impacts of faults injected into the control processor implemented into the FPGA.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II), the project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

[1] C. Salzmann, S. Govaerts, W. Halimi, and D. Gillet, "The smart device specification for remote labs," in *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. IEEE, 2015, pp. 199–208.

[2] Y. T. Park, P. Sthapit, and J.-Y. Pyun, "Smart digital door lock for the home automation," in *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE, 2009, pp. 1–6.

[3] S. Skorobogatov, "Flash memory bumpingattacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 158–172.

[4] J.-B. Machemie, C. Mazin, J.-L. Lanet, and J. Cartigny, "Smartcm a smart card fault injection simulator," in *2011 IEEE International Workshop on Information Forensics and Security*. IEEE, 2011, pp. 1–6.

[5] M. Pavelić, Z. Lončarić, M. Vuković, and M. Kušek, "Internet of Things Cyber Security: Smart Door Lock System," in *2018 International Conference on Smart Systems and Technologies (SST)*. IEEE, 2018, pp. 227–232.

[6] Y. T. Park, P. Sthapit, and J. Pyun, "Smart digital door lock for the home automation," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, Jan 2009, pp. 1–6.

[7] C. Doucet, "Electronically activated door lock assembly," Nov. 14 2000, uS Patent 6,145,353.

[8] Z. Fonea, "Electronic lock system," Nov. 14 2000, uS Patent 6,147,622.

[9] G. K. Verma and P. Tripathi, "A digital security system with door lock system using RFID technology," *International Journal of Computer Applications*, vol. 5, no. 11, pp. 6–8, 2010.

[10] G. Saether, "Electric stepper motor," Nov. 29 1994, uS Patent 5,369,324.

[11] All About Circuits, "Introduction to AC Motors — AC Motors — Electronics Textbook," https://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/stepper-motors/, 2006, accessed: 2019-03-26.

[12] Kiatronics®, "28BYJ-48 - 5V Stepper Motor," http://robocraft.ru/files/datasheet/28BYJ-48.pdf, 2015, accessed: 2019-03-26.

[13] J. Podivinsky, O. Cekan, J. Lojda, M. Zachariasova, M. Krcma, and Z. Kotasek, "Functional Verification based Platform for Evaluating Fault Tolerance Properties," *Microprocessors and Microsystems*, vol. 52, pp. 145 – 159, 2017.

[14] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, and Z. Kotasek, "FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation," in *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2018, pp. 244–251.

[15] O. Cekan, J. Podivinsky, and Z. Kotasek, "Random Stimuli Generation Based on a Stochastic Context-Free grammar," in *Proceedings of the 2016 International Conference on Field Programmable Technology*. IEEE Computer Society, 2016, pp. 291–292.

[16] A. Meyer, *Principles of Functional Verification*. Elsevier Science, 2003. [Online]. Available: http://books.google.cz/books?id=qaIiX3hYWL4C

[17] J. Lojda, J. Podivinsky, and Z. Kotasek, "Reliability Indicators for Automatic Design and Analysis of Fault-Tolerant FPGA Systems," in *20th IEEE Latin American Test Symposium (LATS 2019)*. IEEE, 2019, pp. 93–96.

[18] Xilinx Inc., "LogiCORE IP ChipScope Pro Integrated Controller (ICON) Documentation," https://www.xilinx.com/support/documentation/ip_documentation/chipscope_icon/v1_05_a/chipscope_icon.pdf, Jun. 2011, accessed: 2018-02-15.

[19] Xilinx Inc., "ChipScope Pro VIO Documentation," https://www.xilinx.com/support/documentation/ip_documentation/chipscope_vio.pdf, Sep. 2009, accessed: 2018-02-15.

[20] M. Straka, J. Kastil, and Z. Kotasek, "SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems," in *14th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, 2011, pp. 223–230.

[21] Xilinx Inc., "ChipScope Pro 11.4 Software and Cores User Guide," https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf, Dec. 2009, accessed: 2018-02-15.

[22] MathWork®, "MATLAB and Simulink," https://www.mathworks.com/, 2018, accessed: 2019-03-20.

[23] MathWork®, "Stepper motor," https://www.mathworks.com/help/physmod/sps/powersys/ref/steppermotor.html, 2019, accessed: 2019-03-20.

[24] K. Gurney, *An introduction to neural networks*. CRC press, 2014.

[25] O. Cekan, J. Podivinsky, and Z. Kotasek, "Program Generation Through a Probabilistic Constrained Grammar," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Aug 2018, pp. 214–220.