Design and Implementation of Uninterrupted FPGA using Spare Modules

Abhishek L Narayanan [1], Deepak. V[2], Guruprasath.S [3], Caroline Jebakumari S[4]
Electronics and Communication Engineering, Easwari Engineering College, Ramapuram, Chennai.

Received: January 31, 2019 Accepted: March 13, 2019

ABSTRACT: *FPGA technology is used in all kinds of high-speed devices now, which also need so many demands in Quick configuration of FPGA at runtime is the latest need. Task Scheduling in FPGAs has been increasingly used in modern real-time systems which has to complete the task before the deadline. The existing system aims at tackling such a problem with a self-aware approach. The security module checks the course of the proceedings at each intermittent point of a schedule and on detecting a malicious environment heals the scenario and takes precautions to prevent similar malfunctions in future. In the proposed system, Spare modules are utilized for scheduling the process more accurately and timely. The advantage of spare modules is the multiple-choice multi-tasking spare support during scheduling process. The spare modules are interfaced in four directions so if there is any Trojan malfunction in process, the spare modules will provide scheduling help by generating temporary storage space, spare data, or delay clock etc. which obviously improve the performance of the FPGA configurations.*

Key Words: : *FPGA, Scheduling, Sparemodule, Multitasking, Malfunction.*

I. INTRODUCTION

FPGAs havebeen increased rapidly over past few years. They find use in varied sections of electronics which are used in everyday life. FPGAs are used in high speed, low latency networks for trade purposes. FPGAsare most preferable when it comes to implementation of parallelism. FPGA has high logic handling capability, which is the reason for it being used for complex, high computing designs. FPGAs can be used in bizarre working conditions as it has dynamic reconfigurable setup which adds to its flexibility. The reconfiguration of FPGAs can be done quickly at the run time which comes in handy in critical systems like nuclear reactors and automotive. Since the processes are running against a deadline it is necessary to ensure that the outputs are correct and accurate. Once the deadline is over, it may lead to fatal scenario.

FPGAs are programmed predominantly in HDL which is used for structural and behavioral description. These programs are obtained from third party vendors which may not be certified sources all of the time. Some of the vendors might provide code which may contain malicious codes. These codes may seem harmless during the reception and stay dormant. During the run time of the program, the inactive Trojans may be activated. They maycause interruption of the process. The scheduling is affected causing jeopardy in the working of the FPGAs. In our proposed system, we intend to use spare modules in order to bypass the harm induced by the harmful codes. These spare modules are assertion of the concept of redundancy programming. It helps in achieving a fault tolerant design in FPGAs which is major requirement for uninterrupted functioning.

In order to devise a scheme in which the FPGA is fault tolerant and functions in proper way, self-repair is used. Self-repairing isa mechanism drawn from living organisms. Though the organism has been damaged, it slowly heals with time by creating new cells instead of dead cells. A similar technique is used in the FPGA in order to heal the damage taken. The spare modules are used to serve as the functioning module at the time of fault occurrence. For the spare module to take the place of the affected code we need to use a separate scheduler for them.

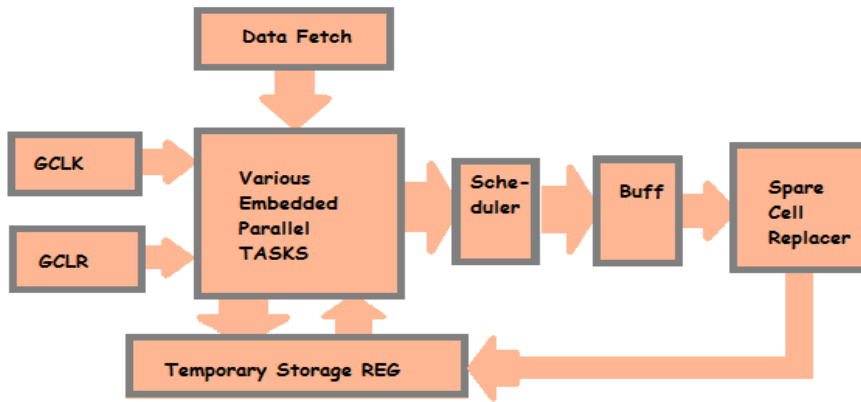

Figure 1 Block Diagram

II. RELATED WORK

Self-aware module is used in the current FPGAs in order to mitigate the effect of malicious codes. It involves using Observe-Design-Act (ODA) paradigm. Till now no self-aware method exists in which both scheduling and simultaneous healing can be performed. The method does not provide with a means that can prevent catastrophic scenario. The security module checks the course of the proceedings at each intermittent point of a schedule and on detecting a malicious environment heals the scenario and takes precautions to prevent similar malfunctions in future.

The FPGA is dynamically reconfigured at a given time lapse in order to rectify the error. Upon detection of the threat, the malfunctioning code is replaced with another harmless code. The working can be understood by using illustration. If the number of process to be run is 14 in the time frame, a threat may cause only 10 instructions to be executed. The missed-out instructions are executed in the next time frame with the help of dynamic reconfiguration

III. ANALYSIS OF THREAT

3.1 Normal Scenario

Consider a system that runs a few tasks and the constraint is that all these tasks must be completed within a certain unit of time. In this case these tasks are segregated and are grouped into time slots that are spread over the entire time range providing a clear-cut demarcation regarding the run time of these separate tasks. When the system executes in normal scenario, the tasks are completed in a perfectly ordered manner and complete according to the time specified for each of the tasks. This ensures that all the tasks are completed inside the time frame.

3.2 Threat Scenario

The ideology that a system always runs in a perfect state can only be assumed in case of a theoretical scenario. In practicality, all systems tend to be influenced by some threat or trojan, during the course of execution. This means that the system may undergo threats during the process. These threats tend to stop the ongoing tasks until such a time that these trojans or threats are neutralized. Also, these threats are very difficult to detect and can take a certain amount of time to be countered. Thus, it introduces a period of time which remains void that is system does not execute any of the tasks. Since the system is used in applications whose primary intention is time constraint, these threats prove to be a highly unfavorable factor in practical scenarios.

3.3ProposedMethodology

The proposed method aims at countering the interruption in the system caused due to the threats. The process of countering includes an addition of a spare module into the system. This spare module acts as a buffer to main module. If the system continues to work in the normal case scenario, there would be no requirement of the spare module as the process will execute uninterrupted. If and when a threat is encountered, the spare module comes to work. The spare module places itself in the system and runs the tasks that are scheduled to be run by the main module. This allows the main frame module a certain amount of time to detect and neutralize the threat using a self- repairing mechanism. This self-repairing mechanism is done by introduction of delay in order to counter the threat. Thus, in this proposed process both, the threat is neutralized and the system continues to run the tasks using the spare module. This proves to be

highly advantageous as it provides uninterrupted services which remains the priority in its practical applications.

IV. SCHEDULING

4.1 Time constraint scheduling:

Time constrained scheduling is indispensable part of design targeted towards applications related to real time applications of FPGA. Time-constrained scheduling algorithms can be implemented by using three different techniques namely; Iterative Refinement, Constructive heuristics, Mathematical programming

4.2 Task scheduler

Task Scheduler worksupon the principle of Finite state machine. TheFSM provides a mathematical model for computation. In the case of the FPGA, we can create finite number of states each of which depicts a particular process. For the purpose of simulation, we make use of four parallel processes. Hence the modules of various task performing systems won't get interfere with each other and can working a synchronized manner with respect to global clock.

```
Fsm_mode: process(mode)
begin
case mode is
when "01"   => fsm_mode <= "1010";
when "10"   => fsm_mode <= "1110";
when "11"   => fsm_mode <= "1000";
when others => fsm_mode <= "1111";
end case;
```

Figure 2 FSM Module

4.3 Multi-tasking

The ability to do runtime configuration and read back of FPGA's allow investigating the problems of implementing realistic multitasking. The FPGA is designed to support multitasking of many applications. It is possible to simultaneously run multiple processes in real life system. It can put to use to create a parallel running system consisting of Traffic light controlling system, Motor controls, Speakers. These are scheduled and run based upon Finite State Machine coding algorithm.

V. SIMULATION RESULTS AND DISCUSSION

Simulation

Initially the VHDL code is written for the various embedded tasks. Using VHDL an encoder, a pulse counter and reference signal are implemented. Then the data scheduler is written and compiled. The Data scheduler is based upon Finite State Machine module. Once all of the processes are compiled and debugged, the integration process is done. In this VHDL File all of the above-mentioned processes are integrated and then it is compiled. The Testbench is written in which the clock and the inputs are given for the simulation. The testbench output waveform is generated. The working of the spare module comes into the picture upon being faced with an error. the attack scenario can be replicated by introducing a binary error pulse using the pulse counter in the HDL coding. The occurrence of fault is valued at 0.01. The binary error causes the data scheduler to be powered up which then activates the spare module. In our case the reference signal and data generator of the data scheduler acts as the spare module. Whenever there is an error occurring the reference signal (or) data generator is called in based upon the type of data that is required, using the data scheduler, which provides with the required type of analog pulse signal (or) digital binary data correspondingly. Using these analog and digital signals as spare the error is averted. In the meantime, the code is repaired by means of introducing a delay in the erroneous code. Once the error is passed the operation is back to normalcy.

Data Scheduler

The Data Scheduler is fired up upon encountering the error. The error is introduced for simulation purpose by forcing a 1 in data scheduler.

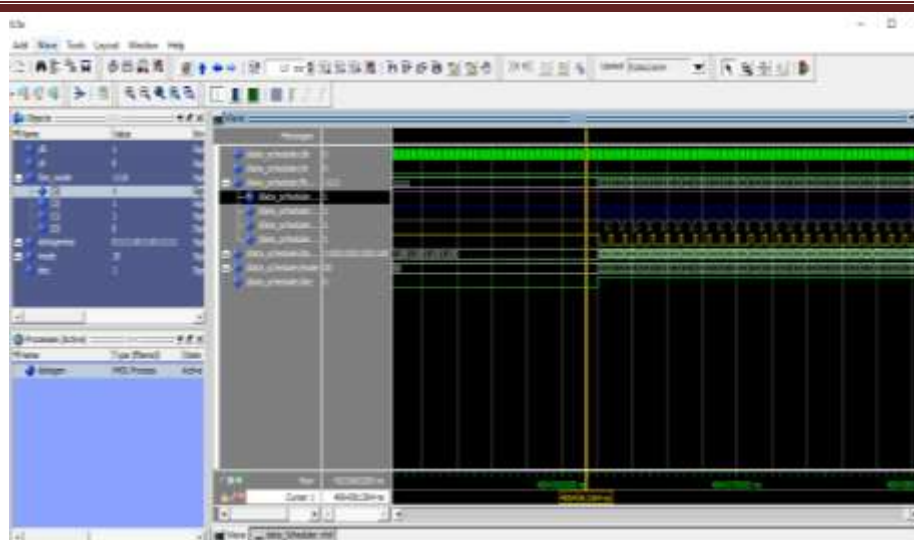

Figure 2 Data Scheduler Waveform

Testbench

The testbench setup is used to generate all the possible outcomes of the parallel processes that are running

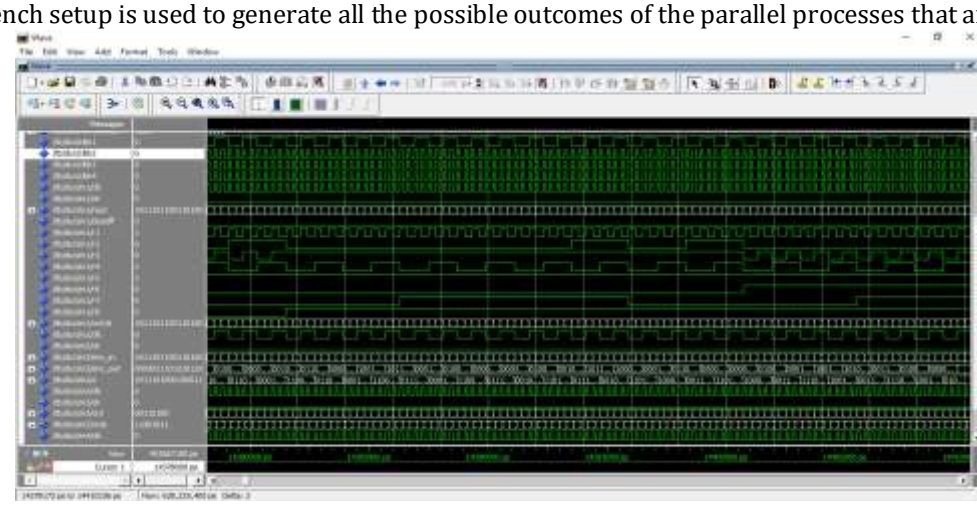

Figure 3 Testbench waveform

Encoder:

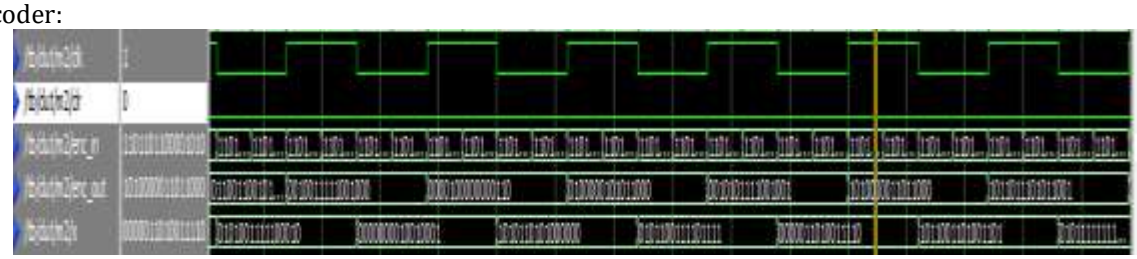

Figure 4 Encoder Waveform

Up & down pulse Counter:

Figure 5 Up & down pulse counter waveform

Reference Signal:

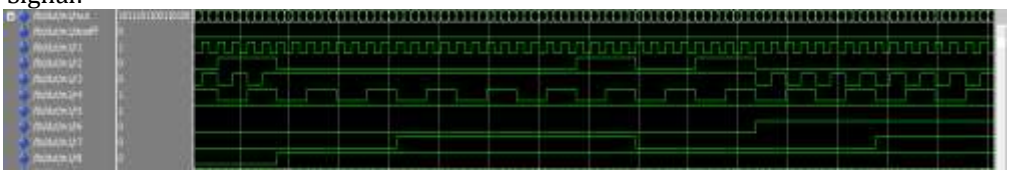

Figure 6 Reference Signal Waveform

Implementation

Once the system level testing has been carried out with the help of the ModelSim, the code is dumped into the FPGA kit with the help of Xilinx ISE. Upon implementation of the proposed methodology, an uninterrupted parallel running FPGA is obtained. Using the Arduino UNO as the interface three different components are scheduled and run without any error. The FPGA ensures parallel running of Traffic light controller, DC Motor and Buzzer System based upon the scheduling. The continuous opening and closing of a switch will lead to the kit produced scheduling time to be implemented for the hardware components that are connected.

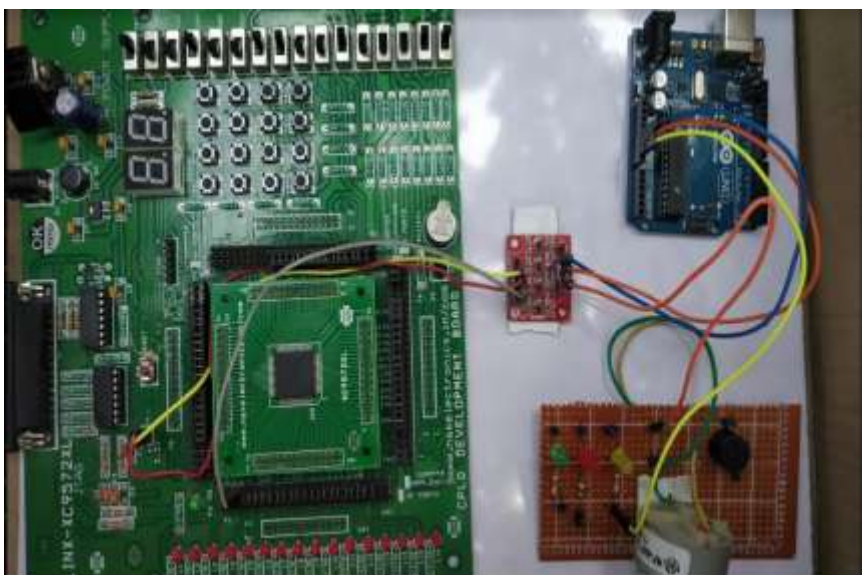

Figure 7Hardware Implementation

VI. CONCLUSION AND FUTURE WORK

By means of implemented the proposed system, the malfunctioning of the process can be averted entirely. The spare modules are put to optimum use to ensure that there is no interruption in the running of the FPGA. With FPGAs being put to use in critical systems like avionics to nuclear reactors, our proposed system can be implemented to avoid any catastrophic events. The efficient interconnection for spare modules should be also considered and achieved. Since the spare modules have a simple architecture, it has very low power consumption. Thus, future work will consider the efficient interconnection of spare modules and the hardware optimizations.

VII. REFERENCE

1. Junxiu Liu, Jim Harkin, Liam P. Maguire, Liam J. McDaid, John J. Wade (2018) 'SPANNER: A Self-Repairing Spiking Neural Network Hardware Architecture', IEEE Transactions on Neural Networks and Learning Systems Vol 29 Issue 4, pp 1287-1300
2. Junya Kaida, Takuji Hieda, Ittetsu Taniguchi, Hiroyuki Tomiyama, Yuko Hara-Azumi, Koji Inoue (2012) 'Task mapping Scheduling for Embedded Many-core SoCs', International SoC Design Conference pp 204-207
3. Krishnendu Guha, Sangeet Saha, Amlan Chakrabarti (2018) 'SHIRT (Self-Healing Intelligent Real Time) Scheduling for Secure Embedded Task Processing', 31th International Conference on VLSI Design, pp 463 - 464
4. Lu Hao, Xiaopeng Yang, Shangkun Hu (2016) 'Task scheduling of improved time shifting based on genetic algorithm for phased array radar', IEEE 13th International Conference on Signal Processing, pp 1655 - 1660
5. Magda A. Silvério Miyashiro, Maurício G. V. Ferreira (2013), 'Phase Cyclical Process Requirements for the Development of Embedded Systems',
6. III Brazilian Symposium on Computing Systems Engineering, pp 155-156

7. K. Guha, D. Saha, A. Chakrabarti (2017), 'Self-Aware SoC security to Counteract Delay Inducing Hardware Trojans at Runtime', VLSID 2017, pp. 417-422.
8. Sangeet Saha, Arnab Sarkar, Amlan Chakrabarti (2015), 'Scheduling dynamic hard real-time task sets on fully and partially reconfigurable platforms', IEEE Embedded Systems Letters, Vol. 7, no. 1, pp. 23-26.
9. Keng-Mao Cho, Chun-Hung Liang, Jun-Ying Huang, Chu-Shing Yang (2011) 'Design and implementation of a general-purpose power saving scheduling algorithm for embedded systems', IEEE International Conference on Signal Processing, Communications and Computing, pp 1-5
10. Swarup Bhunia , Michael S. Hsiao ,Mainak Banga , Seetharam Narasimhan(2014) 'Hardware Trojan Attacks: Threat Analysis and Countermeasures' , Proceedings of the IEEE, Volume: 102 , Issue: 8,pp 1229 - 1247