Improving Correctness of Logic Circuit Using Self-Healing Built-In Logic Test Module in FPGA using Dynamic Partial Reconfiguration

Manjith B.C.

Abstract: ***As cloud servers continue to receive more and more applications and data, it starts using ASIC and FPGAs as accelerators for processor intensive applications. Because of the reconfiguration nature of FPGAs, it become a good choice rather than ASIC on cloud. Encryption is an application which happens frequently on cloud and takes lot of processor cycles which can be shifted to hardware accelerator for execution. The security of hardware circuit which is transferred as bitstream to the FPGA board is of major concern for security critical applications. The article proposes a novel logic authentication module that can check the authentication or correctness of selected parts of logic circuit any time after the circuit is burned into FPGA. In the proposed work, Advanced Encryption Standard (AES) circuit parts are checked for correctness by using in-built Secure Hash Algorithm (SHA) and corrected by Dynamic Partial Reconfiguration (DPR). After the bitstream is burned into FPGA, a proposed software module can select a part or entire circuit for checking authentication with multiple sample inputs. Without the proposed method, any error after the circuit creation cannot be identified or corrected. Additionally, small faults in the circuit is corrected by DPR without loading the entire module. Totally 16 Benchmarks were created to check the correctness of the proposed system. Usage of DPR for circuit correction saves about 97% of time compared to reconfiguring the entire module.***

Index Terms: AES, circuit correctness, dynamic partial reconfiguration, SHA, hash code.

I. INTRODUCTION

AES algorithm is being used in every security critical applications. Security to user data in cloud computing is ensured by encryption using AES. Full user virtual machine (VM) encryption and decryption, traffic into and out of VMs while running, encryption as a service in cloud, and so on uses AES [1-4]. Several cloud providers such as Amazon, Microsoft Azure, Google Cloud, CloudSigma, CloudLink, and HP Atalla cloud use encryption for disk encryption, data traffic encryption, key protection, and so on [5–9]. With the increase in user data and application in cloud, dedicated hardware was introduced to reduce processor load. Introduction of Field Programmable Gate Array (FPGA) on cloud allows customization of hardware for different applications. Use of AES hardware as accelerator on FPGA reduces the encryption load on processor. Since FPGAs require lower frequency than processor and parallel execution, less power and high throughput can be obtained [10, 11]. Hardware design is represented in bitstream manner and are burned into FPGAs in cloud to form the hardware whenever the need for an accelerator arise. Bistreams are stored outside the FPGA and are transferred to FPGA on time. Authentication and error detection of bitstream is confirmed by using CRC checking [12]. Fig. 1 shows the general scenario of the bitstream development, transfer and burning on FPGA inside a server in cloud. The bitstream can be directly burned into FPGA received from network or can use an already stored bitstream inside a local memory.

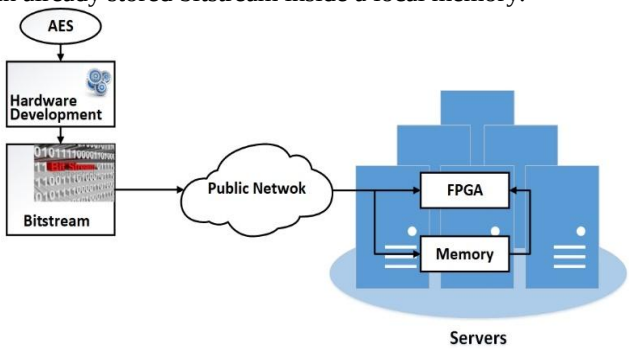


Fig. 1. Overall view of hardware acceleration process

There are many disadvantages on existing authentication checking mechanisms for bitstream.

- It cannot identify the changes or errors caused by environmental changes or attacks on the circuit after the bitstream is burned
- It cannot identify which portion of bitstream or hardware contains error
- If an error in bitstream is found, then an entire copy of new bitstream has to be transferred and loaded which not only affects the working of entire system but also a time consuming process
- An attacker can disable the CRC checking by simply deactivating a flag in bitstream [13]
- There is a chance to change the function with corresponding change of CRC

Checking the authentication of the AES accelerator in cloud is important since it is handling a large amount of user data. Encryption is being used in cloud for data in motion and at rest and encryption as a service. Any change in encryption circuit will affect the security of entire cloud.

Several AES implementations has been done for FPGAs. Optimization for speed and area are done through pipelining, sub-pipelining, loop unrolling, reducing the logic depth etc. [14-20]. Balancing between pipelining stages were done through allocation of registers at proper locations in order to avoid stalling. Farashahi et al. [14] proposed a 2-slow retiming technique that extended the c-slow retiming technique for the throughput improvement of AES

Revised Manuscript Received on July 06, 2019.

Dr. Manjith B.C., Department of Computer Science and Engineering, National Institute of Technology Puducherry, Karaikal

algorithm. Another system was developed with fast AES design using Look Up Tables (LUTs) and it added additional protection for AES core [15]. In the study by Liu et al. [16], logic depth is analyzed and reduced in order to improve throughput. Another study [17] showed different methods for implementing s-box of AES in many proposed designs. The authors used loop unrolling for critical path modification and fully pipelined and sub-pipelined techniques, which allowed the increase in the clock frequency and reduction in critical path. To reduce the area, Lee et al. [18] employed a series of constant binary matrix multiplication instead of Galois field (GF) (28) computation. Good and Benaissa [19] presented two designs for AES feedback mode support. First design assigned LUTs between pipeline cuts to achieve maximum throughput. Second design focused on Key Expansion using which, the key could be changed every clock cycle by proper pipelining. The technique proposed by Hammad et al. [20] splits and rearranges the operation in AES to get optimum area and throughput. The Mix Column operation was split and rearranged with Add Round Key operation. It could achieve a throughput of 39,053 Mbps at 305.1 MHz in XC2V6000-6 [20].

Error in bitstream is checked by implementing CRC in bitstream. Bitstream along with CRC code are send to the board through public/private network. Before burning the bitstream to the board, CRC code is checked for any change in bitstream. In [13], Rajat Subhra Chakraborty shows how CRC can be disabled by an attacker. After modifying the functional units of the AES with his own functional units, an attacker can disable the CRC checking so that it cannot be detected. In the work by Abdellatif et al. [21], bistream protection mechanism is embedded in the static part of FPGA. The security is checked before the bitstream is burned on the board. AES-GCM and RSA are used in Xilinx high end FPGA boards in order to provide protection and authentication of bitstream [22, 23]. In [24], Karam proposes an obfuscation based approach for protecting the bitstream. Swierczynski et.al [25] shows how an encrypted bitstream can be reverse engineered so that an attacker can insert Trojans on to bitstream and goes undetected.

The article proposes a dynamic self-checking authentication module for AES accelerator with hardware support. As of our knowledge, there is no work that checks the authentication of AES circuit after the bitstream is burned. Our aim is to check the authentication of AES hardware after the bitstream is burned into the board for security critical applications. Even if the CRC checking fails or the hardware is altered by atmospheric conditions or by an attacker, the self-checking hardware that is proposing can find it out and correct it. The exact functional unit where the fault occur can be analyzed and can be reconfigured without affecting the execution of other units by partial reconfiguration. Validation of hardware can be done once when the bitstream is burned and can be repeated whenever needed. For any security critical applications, the authentication of hardware can be checked before encryption. Dynamic partial reconfiguration is used to correct any faulty module which saves time and does not affect rest of the circuit.

II. PROPOSED METHOD

A. AES Implementation

AES is a block cipher symmetric encryption algorithm that has a fixed input length of 128 bits and key length of 128/192/256 bits, which operates in 10/12/14 rounds depending on the key length. AES has been accepted since its introduction for its security strength, performance, flexibility, and efficient implementation [26]. AES has mainly four rounds of operation on plain text namely—Byte Substitution (Substitute Byte), Shift Rows, Mix Column, and Add Round Key as shown in Fig. 2. The proposed work uses AES-128 for implementation.

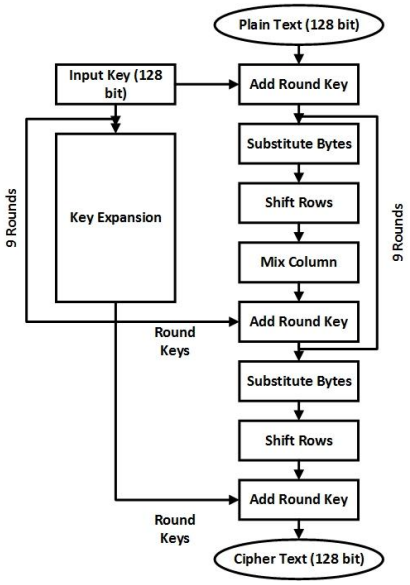


Fig. 2. Original 128-bit Advanced Encryption Standard algorithm.

AES algorithm is implemented and optimized with pipelining, sub-pipelining, loop unrolling and memory partitioning. Registers are inserted at corresponding places in order to make the pipelining balanced. Fig. 3 shows the overall pipelined implementation of AES algorithm. Registers are inserted between rounds in order to save the intermediate results. Fig. 4 shows the proposed sub-pipelined implementation of a single round of AES. Substitute Byte and Shift Rows are merged to balance the pipelining.

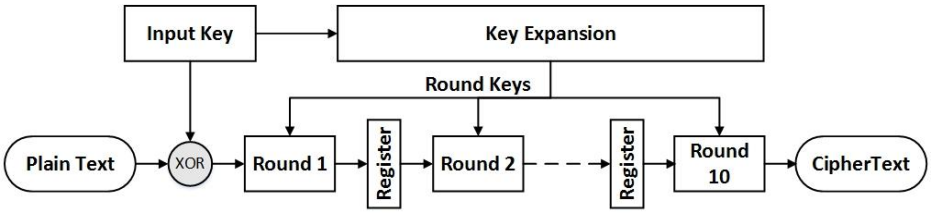


Fig. 3. AES pipelined implementation.

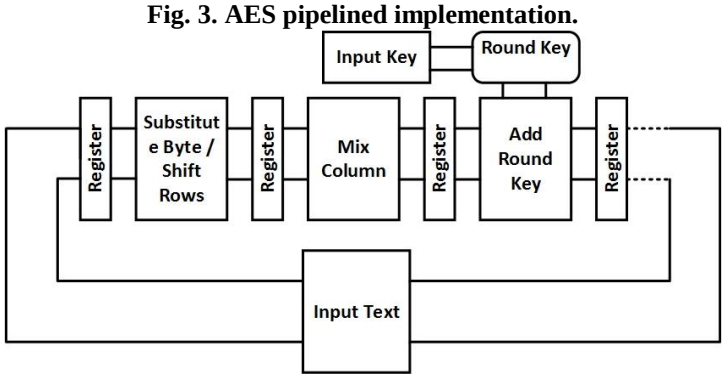


Fig. 4. Proposed one round design of Advanced Encryption Standard.

The latencies measured for different frequencies are shown in Table I.

Table I: Latency values observed with different path delays and frequencies.

Sl. No.	Path delay (ns)	Frequency (MHz)	Latency				
			Substitute Byte	Shift Rows	Mix Column	Add Round Key	Key Expansion
1	1.23	813	0	0	3	0	4
2	2.6	357	0	0	1	0	2
3	2.84	352	0	0	0	0	2
4	4.91	203	0	0	0	0	0

The proposed method uses 1.23ns path delay for implementation. The AES system takes 60 clock cycles at 1.23ns path delay. We have implemented a four-stage and five-stage sub-pipelining for Mix Column and Key Expansion respectively. Since the aim of the article is not AES implementation, further details are avoiding.

B. Proposed logic security core for AES

The proposed logic testing module for AES checks the correctness of AES accelerator after the circuit is created on FPGA board. As desired by the user or application, the accelerator can be tested anytime and any portion of the circuit. Initial validation can be set by checking entire module as well as individual functional units (referring a portion of the circuit). Because of the frequent use of AES accelerator, it can remain on FPGA for quite a long time which can cause tampering of circuit. Since FPGAs are shared in cloud for different applications, frequent checking of circuits must be done for security critical applications. Validation of circuit correctness can be done dynamically and parallel when the accelerator is working using the proposed security core.

Fig. 5 shows the architecture of the proposed hardware and software security checking module. The hardware security module contains connection to different parts of AES accelerator. The software security modules controls the hardware security module by selecting the parts needed for checking, giving input and validating the output. When the AES accelerator is initially loaded into FPGA, the software module will trigger the hardware module for checking the correctness of entire module. The hardware module contains Secure Hash Algorithm (SHA) hardware for validating the correctness of the circuit. If the test is successful after validating entire module, then there is no need to proceed to test individual units. If the initial test fails, the software module triggers checking of individual modules for error. After finding the fault module(s), it will be reconfigured by Dynamic Partial Reconfiguration (DRP).

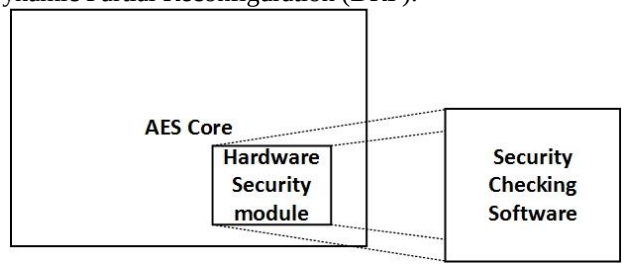


Fig. 5. Proposed in-built hardware authentication module.

Fig. 6 shows the virtual interconnection view of security modules to the AES accelerator. The portions or parts of the circuits are identified as functional modules (Substitute Byte, Shift Rows, Mix Column and Add Round Key) for validation. In order to check each module, it will be connected to the security module. For testing the entire module, the cipher text storing memory is also connected to the security module.

The working of hardware and software functional units are shown in Figure 7. The software security module selects the functional unit than needs to be tested by setting the corresponding parameter. It gives sample input and key to the hardware security module. The hardware module executes the corresponding functional unit using the sample input and key. The result is given to the in-built SHA hardware where a hash code is calculated on it. The hash code obtained will be given back to the software security modules where it is being compared with pre-computed hash code. If the hash code matches, the test is successful and there is no error in that part of the circuit. If the two hash code does not match, then there is error in the particular portion of the circuit and will be reconfigured independently by DPR without affecting rest of the system. If the security core flags success in all functions units and the entire AES module shows error, then only entire module will be reconfigured. Using the proposed method, dynamic checking and correction of logic circuit can be performed without affecting the performance of the overall system.

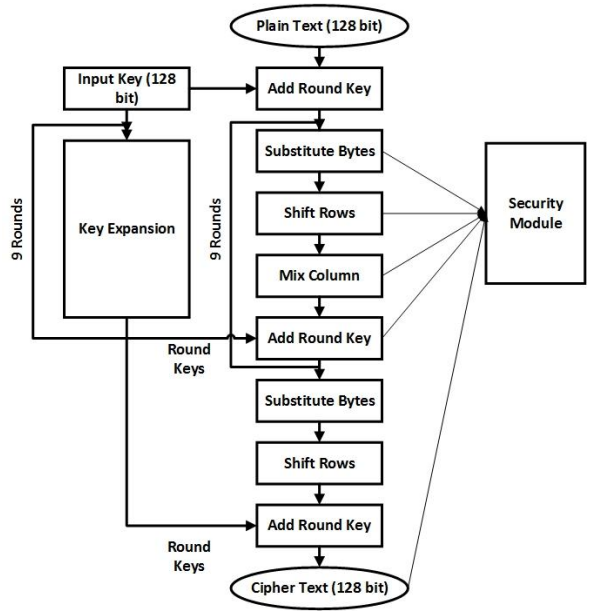


Fig. 6. Overall view of proposed security module on AES.

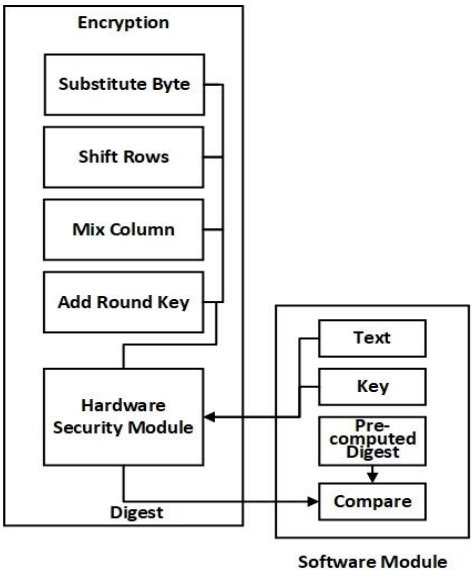


Fig. 7. Internal working of hardware security module and software module.

C. Hardware module

The hardware module for checking the AES core is named as test module. It is internally connected to all functional units (Substitute Byte, Shift Rows, Mix Column and Add Round Key) as shown in Fig. 7. It can select the entire encryption block or any of the functional units for checking authentication. After the bitstream is loaded, initially the entire encryption block is checked for authentication with provided sample input text and key. If there is any error found, separate functional units are checked with additional inputs.

Algorithm 1: Hardware Test Module
Input: txt, key, choice, fn
Output: digest (δ')

1. If choice =1 do steps 2 to 4
2. Set α=encrypt()
3. Set δ'=SHA(α)
4. Return δ'
5. Else if choice =2 do steps 6 to 14
6. Set α = Substitute_Byte()
7. Set δ_1'=SHA(α)
8. Set α = Shift_Rows()
9. Set δ_2'=SHA(α)
10. Set α = Mix_Column()
11. Set δ_3'=SHA(α)
12. Set α = Add_Round_Key()
13. Set δ_4'=SHA(α)
14. Return δ_1', δ_2', δ_3', δ_4'
15. Else if choice = 3
16. If fn = "Substitute_Byte" do steps 17 and 18
17. do steps 6 and 7
18. Return δ_1'
19. Else if fn = "Shift_Rows" do steps 20 and 21
20. Do steps 8 and 9
21. Return δ_2'
22. Else if fn = "Mix_Column" do steps 23 and 24
23. Do steps 10 and 11
24. Return δ_3'
25. Else if fn = "Add_Round_Key" do steps 26 and 27
26. Do steps 12 and 13
27. Return δ_4'
28. Else return -1

Algorithm 2: Software Test module
Input: txt, key, δ, δ1, δ2, δ3, δ4, δmod
δ, δ1, δ2, δ3, δ4, δmod are pre computed message digest for sample input text and key. fn contains the name of the functional unit that needs to be checked. Declare δ'[4] to receive output from test module.

1. If fn = null do steps 2 to 16
2. Set δ'' = test(txt, key, 1, fn)
3. If δ'' = δ
4. "No error in encrypt "
5. Exit
6. Else do steps 7 to 16
7. Set δ' [ ]= test(txt, key, 2, fn)
8. If δ' [0] ≠ δ_1
9. "Error in Substitute Byte "
10. If δ' [1]≠ δ_2
11. "Error in Shift Rows"
12. If δ' [2] ≠ δ_3
13. "Error in Mix Column "
14. If δ' [3] ≠ δ_4
15. "Error in Add Round Key "
16. Exit
17. Else do steps 18 to 23
18. δ'= test (txt, key, 3, fn)
19. If δ' ≠ δ_{mod}
20. Error in module fn
21. Else
22. No error in module fn
23. exit

Algorithms 1 and 2 shows the stepwise procedure for hardware and software modules for the proposed security core respectively. The security core can be set to check the entire circuit or a part of circuit by setting the third parameter as 1, 2 or 3 in test call of software module. For initial authentication of circuit after bitstream burning, the entire circuit is checked with multiple inputs. If there is no error found, the module can start functioning. If any error is found out, each module is verified separately with multiple inputs and correct the faulty module. During working of the circuit, at any point or for any security critical applications, modules can be tested independently without affecting the other modules. If due to some environmental changes or due to some attacks, when the circuit changes, it can be found out and corrected.

SHA-256 (Secure Hash Algorithm -256) is used for checking the authentication of circuit. SHA is embedded in security core. Whenever the circuit needs to be checked, security core gives inputs to the particular hardware module and calculates the corresponding output. That output is given to SHA in the security core for checking the authenticity of the calculated outputs. The process can be repeated for multiple inputs as needed.

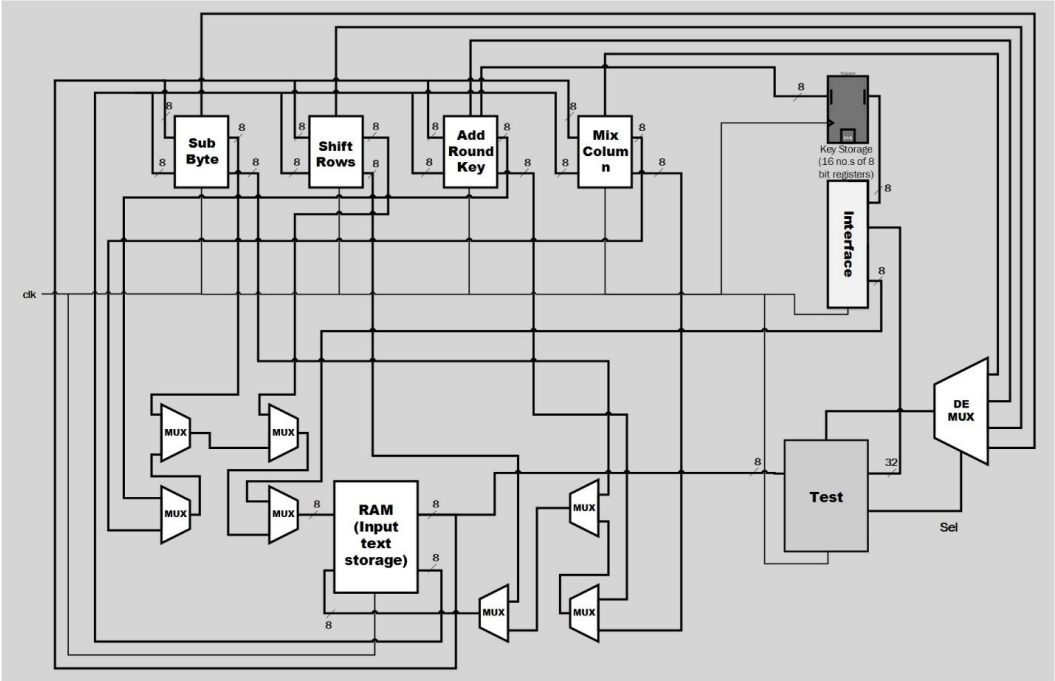


Fig. 8. Proposed hardware architecture of AES core with security core.

Fig. 8 shows the internal connection of security core to AES modules – Substitute Byte, Shift Rows, Mix Column and Add Round Key. Input text is stored inside RAM. Key is stored inside registers. From the interface, the input text and key will be given to RAM and registers for storage. Then the appropriate functions (Substitute Byte, Shift Rows, Mix Column and Add Round Key) are calculated on input text, key and results are send to test module. Sel signal will select the particular module or the entire module for testing. The test module calculates the hash of received text and send the hash value to the software module for checking the correctness. If the hash of values received from any AES functional unit does not match with the pre-computed hash values, the particular functional unit contains error and has to be reloaded.

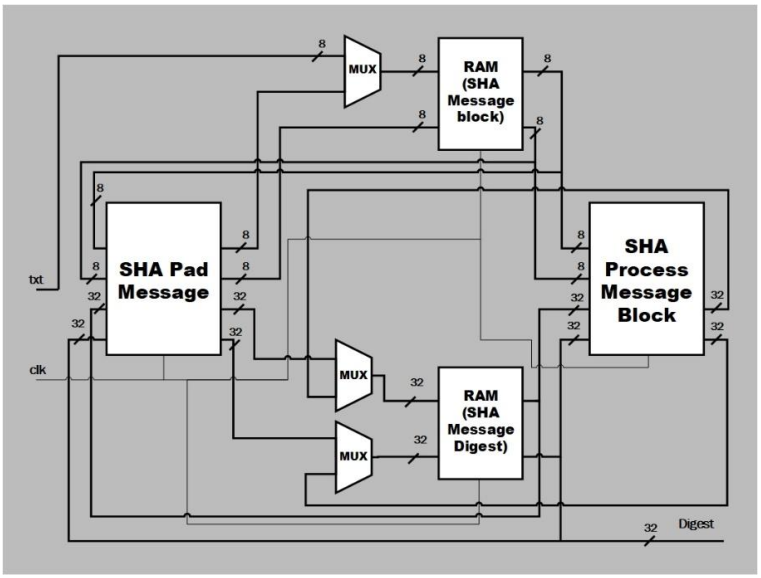


Fig. 9. Proposed internal architecture of security core.

Fig. 9 shows the internal modules and connections of security (test) module. The output received from AES functional units are stored inside RAM named SHA Message block. SHA 256 is implemented for hash computation. Padding of bits is done to the message block in order to convert it to 512 bits. Transformations on message block is done in Process Message Block to get the hash code. The hash code is temporarily stored inside RAM from where it is compared with the pre-computed hash value. Fig. 10 shows screen shot of the RAM module for storing message block and hash code (digest).

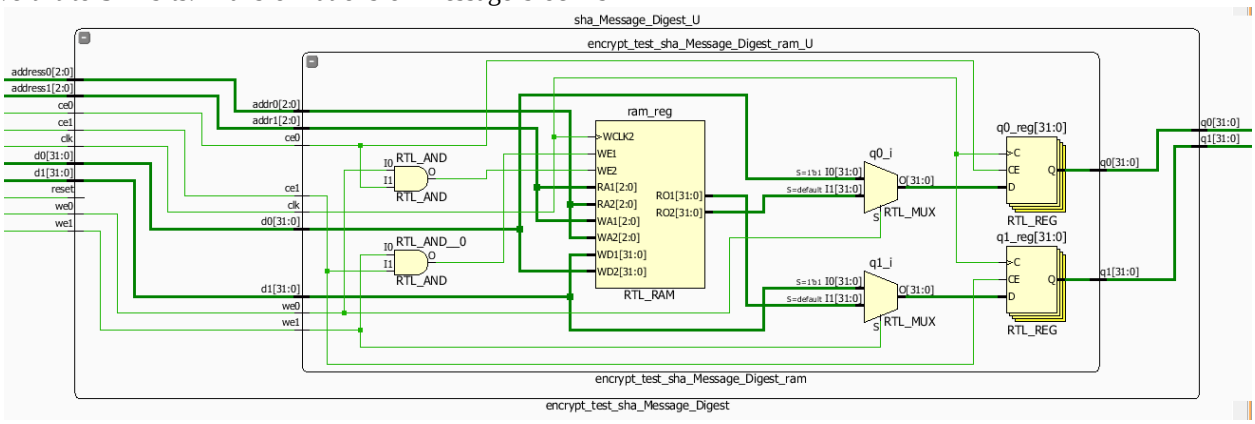


Fig. 10.Message digest calculated by SHA saved inside FPGA RAM.

D. Reconfiguring modules by Dynamic Partial Reconfiguration

When a particular module fails to pass the test, it can be reconfigured at run time without affecting the execution of other modules by Dynamic Partial Reconfiguration (DPR). The modules must be made reconfigurable in order to change unit at run time using partial reconfiguration. Fig. 11 shows the partial reconfiguration architectural view of the proposed system. Four Reconfigurable Partitions (PRs) are created for each module. Blanking and partial bitstreams are stored in local DDR memory. Whenever a module fails to authenticate, it will give a trigger to Partial Reconfiguration Controller (PRC). The PRC then loads the corresponding partial bitstream from DDR3 or from external memory to the RP through ICAP [27].

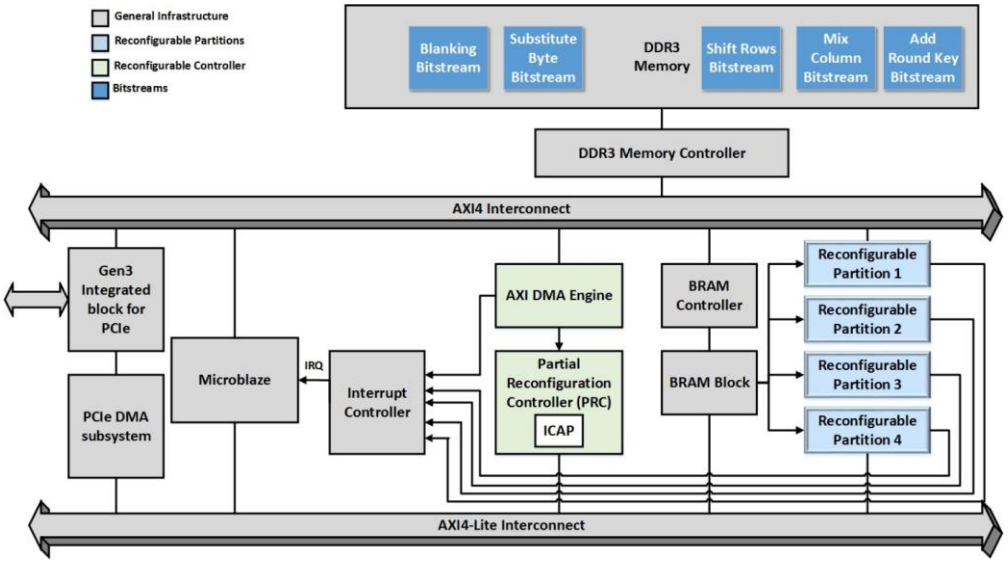


Fig. 11. Proposed model for partial reconfiguration.

III. IMPLEMENTATION AND RESULTS

The proposed scheme is verified, simulated, and implemented using XC7VX690T device. We used Vivado Design Suite—HLx Edition for the implementation [28–30].

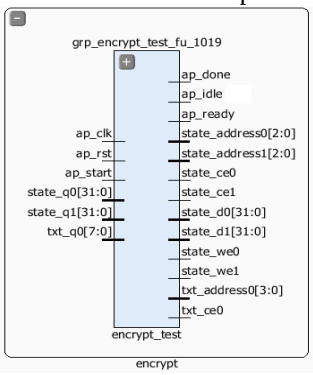


Fig. 12. Overall schematic of test module.

The schematic of test module with input ports are shown in Fig. 12. State port corresponds to SHA digest and txt corresponds to message block. Write enable and chip enable are shown as _we0 and _ce0 respectively. ap_ready pin will be enabled when the block is ready to receive next input, ap_start pin has to be set to start the module and ap_done pin will be enabled when the module finishes the computation and ready to give the output. ap_rst can be used to reset the module.

The SHA Test algorithm is implemented to consume less area as possible in hardware. Tradeoff is made between area and speed. Since the test module is executing only once when the bistream is loaded, the speed is compromised. Further execution of security module can be done in parallel with encryption process and hence it will not affect the performance. Our design takes 330 clock cycles with three samples of inputs for completing the testing of entire AES modules. The hardware utilization for the proposed Test module is shown in Table II and Table II. Table IV shows the comparison of AES hardware module without built-in security module and with built-in security module.

Table II. Hardware utilization of test module.

Modules	Slices	Slice LUTs	Slice Register	F7 MUX	LUT as Logic	LUT FF pairs	BRAM Tile
SHA pad Message	111	266	222	0	266	98	0
SHA process Message Block	319	896	882	32	896	400	1.5
state_ram_U	11	20	0	0	20	0	1
txt_ram_U	58	128	0	0	128	0	0.5
bitlen_ram_U	26	54	0	0	54	0	1
data_ram_U	0	0	0	0	0	0	0.5

Table III: Hardware utilization of SHA process Message Block module.

SHA process Message Block	slices	slice LUTs	Slice Register	F7 MUX	LUT as Logic	LUT FF pairs	BRAM Tile
k_rom_U	2	3	3	0	3	0	0
m_ram_u	71	210	0	0	210	0	1

Table IV. Comparison of hardware utilization of AES encryption without security module and with security module.

	Slices	Slice LUTs	Slice Register	F7 MUX	LUT as Logic	LUT FF pairs	BRAM Tile
without security	2617	9896	2741	3235	9896	2561	0

with security	3142	11260	3845	3267	11260	3059	4.5

With the addition of hardware security module to the AES core, there is an overall 20% increase in resource utilization.

For analyzing the proposed systems, different benchmarks are created for each functional unit and for the entire system. Table V shows the details of the Benchmarks created for testing the proposed system.

Table V. Benchmarks for AES proposed test module

Hardware Functional unit	Benchmarks	Details
Substitute Byte	SBB1	Change of connection to s-box
	SBB2	Change of connection from input register
	SBB3	Change of connection to output register
	SBB4	Change of s-box value
Shift Rows	SRB1	Change of internal shifting connection
	SRB2	Change of connection to output register
Mix Column	MCB1	Missing connection to δ (affine transformation)
	MCB2	Change of connection from input register
	MCB3	Change of connection to output register
	MCB4	Change in function δ
Add Round Key	ARKB1	Missing connection to xor
	ARKB2	Change of connection from input register
	ARKB3	Change of connection to output register
Encryption	ENB1	Missing connection to module
	ENB2	Change of connection from input register
	ENB3	Change of connection to output register

The proposed system is tested with all the benchmarks given in Table 5 for three different input texts and keys. The hamming distance is calculated for the hash code received from software module and that received from hardware test module. The hamming distance shows the number of difference in bits for the hash code computed by the hardware test module and the hash code given by software module during execution. If the hamming distance is greater than zero, it shows the existence of error in the AES accelerator. In short, the existence of error in AES module after the bitstream is burned into FPGA has impact on hamming distance between hash codes. Fig. 13 shows the result of executing the benchmarks for encryption alone to the entire AES module. Fig. 14 (a) to (d) shows the results obtained after applying benchmarks designed for each functional module.

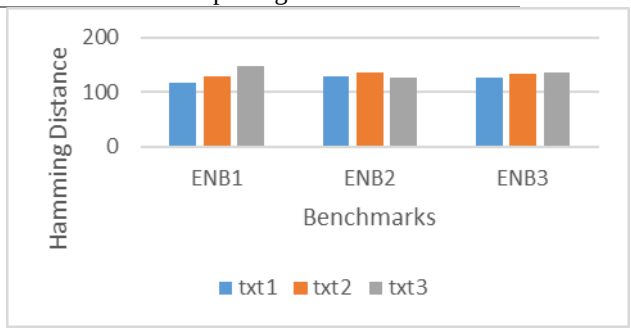
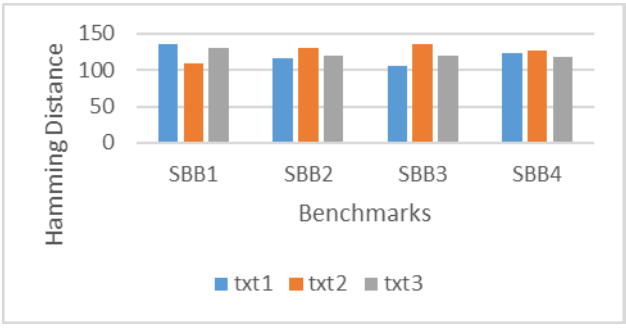
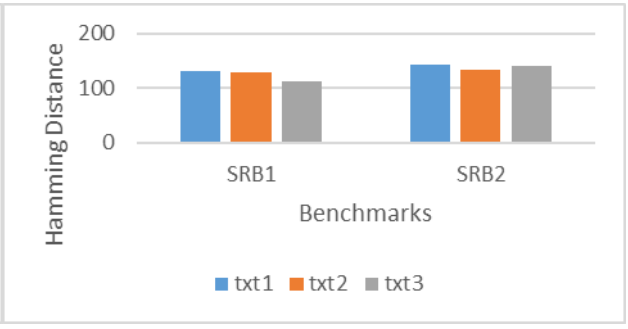


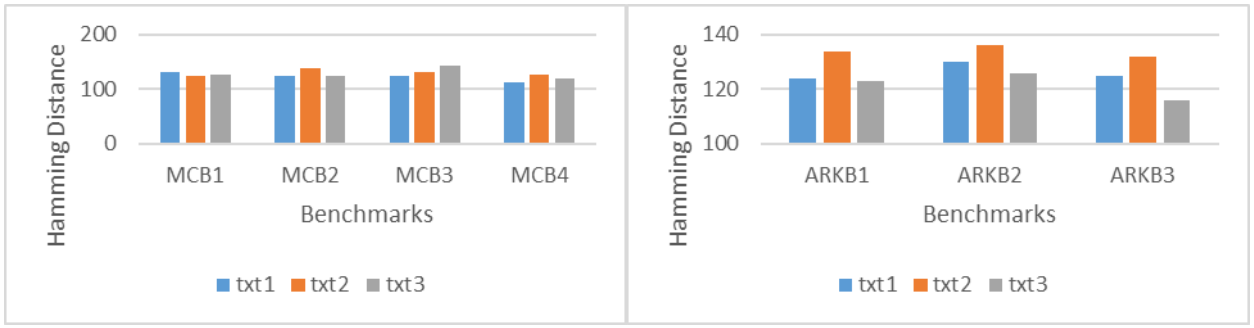
Fig. 13. Hamming distance obtained after the application of benchmark designed for entire encryption module.



(a)



(b)



(c) (d)

Fig. 14. Hamming distance obtained after application of benchmarks on (a) Substitute Byte functional unit (b) Shift Rows functional unit (c) Mix Column functional unit and (d) Add Round Key functional unit.

All the result shows the impact of change in circuit of the AES module on hash codes. In the proposed benchmarks, only small changes are made to analyze the hamming distance. Fig. 15 shows the result of benchmarks for individual functional units applied to the entire functional module. If the hash code of all functional units are same as the pre-computed value, and if the hash code of entire encryption block does not match with the pre-computed value, then entire module will be replaced.

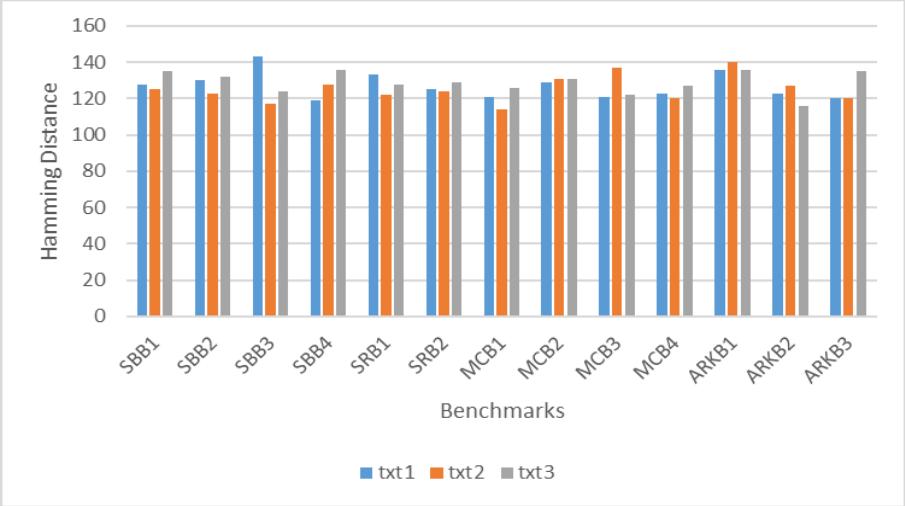


Fig. 15. Effect of application of benchmark for each functional unit on entire encryption module.

The AES accelerator is integrated through PCIe interface [31]. The overall system is designed to have static part and dynamic part. The static part does not change after the system is burned into the device. The dynamic part consists of reconfigurable modules that can be reconfigured dynamically without affecting other parts of the system. The four functional modules of AES comes under the dynamic part of the system. Since the proposed system used PCIe connection for high data transfer, the static part consumes more area. As soon as a change in a module is found out, the corresponding module will be reconfigured using DPR. Fig. 16 shows the screenshot of device after floor planning, routing and placement is done. The marked portion shows the reconfigurable partitions for modules - Substitute Byte, Shift Rows, Mix Column and Add Round Key. The rest of the portions are static part of the system. Table VI shows the bitstream size and reconfiguration time needed for entire module and DPR modules. The partial bitstreams are stored in storage media (SD card in our case). The time needed for reconfiguration is calculated as the sum of time needed for transferring the bitstream from SD card to local memory (DDR), and from there to ICAP configuration cache and finally to the respective configuration memory [32-34].

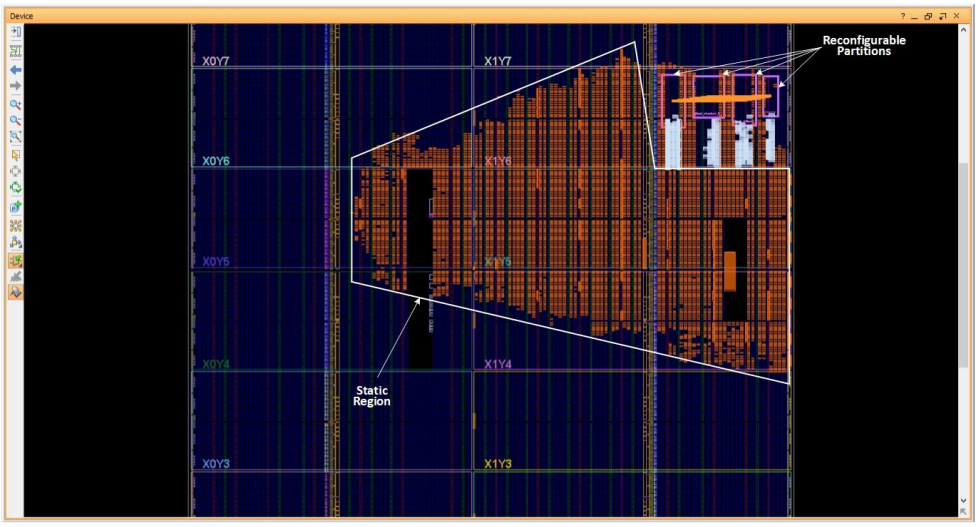


Fig. 16. Chip layout of the proposed scheme on virtex 7(XC7VX690TFFG1761-2) device.

Table VI. Bitstream size of entire AES and partial bitstream size of each module along with corresponding time required for reconfiguration.

Module	Bitstream Size (KB)	Time (ms)
aes	28062	2220.608
Substitute Byte	762	60.311
Shift Rows	818	64.742
Mix Column	869	68.778
Add Round Key	812	64.268

Fig. 17 shows the comparison of bitstream size between the static part of the system and dynamic part of the system. If there is any change in circuit after the bitstream is loaded into FPGA, it is economical in terms of time to reconfigure only the faulty module than to replace the entire system. The bitstream size of entire AES module not only depends on the encryptor module, but also on the interfaces, DMAs and other reconfigurable modules. Fig. 17 clearly shows the amount of bitstream that has to be transferred in order to correct the faulty module which in turn has a great impact on saving reconfiguration time. Fig. 18 shows the percentage of reduction in reconfiguration time if the proposed system is used for partially reconfigure only the modules that has to be changed. If there is fault only in Substitute Byte, then reconfiguring only the specific module saves 97.28 % time compared to reconfiguring the entire module. Similarly, Shift Rows saves 97.08 %, Mix Column save 96.9% and Add Round Key saves 97.11% compared to reconfiguring entire system.

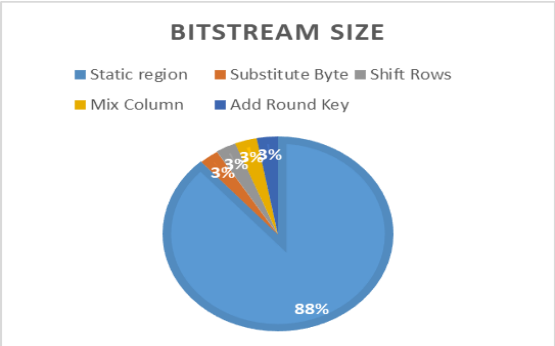


Fig. 17. View of the bitstream size of static region and reconfigurable regions.

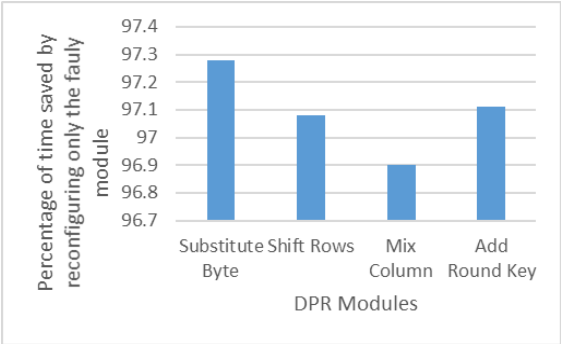


Fig. 18. Percentage of reduction in reconfiguration time of each module compared to reconfiguring entire AES module.

Using the proposed self-checking module and DPR feature, any error in the circuit can be analyzed dynamically without affecting the working of other parts of the design. As the tradeoff is made between speed and area, the proposed security core takes 330 clock cycles for running three inputs to complete the entire modules and takes 20% more area than the hardware without security core. Analysis shows that testing with different benchmarks can uncover even a simple fault in circuit and can correct a module by DPR with 97% less amount of time compared to configuring the entire system.

IV. CONCLUSION

AES is being used in almost all security critical applications. When cloud started to use FPGAs as hardware accelerators, security for hardware circuit became an issue. Checking the authentication of circuit is done only before the bitstream is burned. The article proposed a novel method to check the authentication of AES hardware circuit dynamically after the circuit is created on the board. Tradeoff is made between area and speed. Since the FPGA resources are shared on cloud, area is being reduced with less execution speed for authentication check. The security core can be run along with encryption parallel to check the authentication. Since the proposed method is using hardware-software based security core, it can ensure the security of the hardware even if the bitstream and CRC are tampered. The security core

took an additional 20% more resources than AES core without having security core. Benchmarks showed change of Hamming distance in hash code for faulty circuit which could be corrected by dynamic partial reconfiguration without affecting other parts of the circuit with less amount of reconfiguration time.

REFERENCES

1. J. Teubner, L. Woods, Data Processing on FPGAs, Synthesis Lectures on Data Management 5.2, Morgan & Claypool Publishers, 2013, 1–118.
2. J.D. Bokefode, A.S. Bhise, P.A. Satarkar, D.G. Modani, "Developing a secure cloud storage system for storing IoT data by applying role-based encryption," Procedia Computer Science 89 (2016): 43–50.
3. R. Hossein, S. Elankovan, Z. Md. Ali, A.M. Zin. "Encryption as a service (EaaS) as a solution for cryptography in cloud," Procedia Technology 11 (2013): 1202–1210.
4. R.I. Krutz, R.D. Vines, Cloud Security: A Comprehensive Guide to Secure Cloud Computing, Wiley Publishing, 2010.
5. AWS Storage Services Overview a Look at Storage Services Offered by AWS, Amazon Web Services, December 2016.
6. Protecting Data in Microsoft Azure, White Paper, Microsoft, August 2014
7. Encryption at Rest in Google Cloud Platform Google Cloud Platform Encryption Whitepaper, August 2016.
8. Cloudsigma. https://www.cloudsigma.com/securing-your-data-in-the-cloud-with-encryption/
9. HP Atalla Cloud Encryption Securing Data in the Cloud, Technical White Paper, November 2013.
10. S. Hauck, D. Andre, Reconfigurable Computing: The Theory and Practice of FPGA-based Computation, Vol. 1, Morgan Kaufmann, 2010.
11. S. Kilts, Advanced FPGA Design: Architecture, Implementation, and Optimization, Wiley-IEEE Press, 2007.
12. https://www.altera.com/products/general/devices/stratix-fpgas/about/crc.html
13. Chakraborty, Rajat Subhra, et al. "Hardware Trojan insertion by direct modification of FPGA configuration bitstream." IEEE Design & Test 30.2 (2013): 45-54.
14. R.R. Farashahi, B. Rashidi, S.M. Sayedi, "FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm," Microelectronics Journal 45 (8) (2014): 1014–1025.
15. M-H. Jing, Z-H. Chen, J-H. Chen, Y-H. Chen, "Reconfigurable system for high-speed and diversified AES using FPGA," Microprocessors and Microsystems 31 (2) (2007): 94–102.
16. Q. Liu, Z. Xu, Y. Yuan, "High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion," IET Computers & Digital Techniques 9 (3) (2014): 175–184.
17. A. Soltani, S. Sharifian, "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA," Microprocessors and Microsystems 39 (7) (2015): 480–493.
18. H. Lee, Y. Paik, J. Jun, Y. Han, S.W. Kim, "High-throughput low-area design of AES using constant binary matrix-vector multiplication," Microprocessors and Microsystems 47 (2016): 360–368.
19. T. Good, M. Benaissa, "Pipelined AES on FPGA with support for feedback modes (in a multi-channel environment)," IET Information Security 1 (1) (2007): 1–10.
20. I. Hammad, K. El-Sankary, E. El-Masry, "High-speed AES encryptor with efficient merging techniques," IEEE Embedded Systems Letters 2 (3) (2010): 67–71.
21. Abdellatif, Karim M., Roselyne Chotin-Avot, and Habib Mehrez. "Authenticated encryption on FPGAs from the static part to the reconfigurable part." Microprocessors and Microsystems 38.6 (2014): 526-538.
22. "Using Encryption to Secure a 7 Series FPGA Bitstream" by Kyle Wilkinson, Xilinx XAPP1239 (v1.0) April 15, 2015. (https://www.xilinx.com/support/documentation/application_notes/xapp1239-fpga-bitstream-encryption.pdf)
23. Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream by Kyle Wilkinson, Xilinx XAPP1267 (v1.1) April 13, 2017. (https://www.xilinx.com/support/documentation/application_notes/xapp1267-encryp-efuse-program.pdf)
24. Karam, Robert, et al. "Robust bitstream protection in FPGA-based systems through low-overhead obfuscation." ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on. IEEE, 2016.
25. Swierczynski, Pawel, et al. "FPGA Trojans through detecting and weakening of cryptographic primitives." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34.8 (2015): 1236-1249.
26. J. Daemen, V. Rijmen, The Design of Rijndael: AES—The Advanced Encryption Standard, Springer Science and Business Media, 2013.
27. Vivado Design Suite User Guide Partial Reconfiguration, UG909 (v2016.1) April 6, 2016 (url: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf)
28. Vivado Design Suite Tutorial: High-Level Synthesis (UG871) – Xilinx (url: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug871-vivado-high-level-synthesis-tutorial.pdf).
29. Vivado Design Suite User Guide: Design Flows Overview – Xilinx (url: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug892-vivado-design-flows-overview.pdf).
30. Vivado Design Suite User Guide: Using Constraints (UG903) – Xilinx (url: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf).
31. 7 Series FPGAs Integrated Block for PCI Express v3.0 LogiCORE IP Product Guide Vivado Design Suite PG054 November 19, 2014-Xilinx (url: https://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v3_0/pg054-7series-pcie.pdf)
32. Papadimitriou, Kyprianos, Antonis Anyfantis, and Apostolos Dollas. "Methodology and experimental setup for the determination of system-level dynamic reconfiguration overhead." Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on. IEEE, 2007.
33. Papadimitriou, Kyprianos, Apostolos Dollas, and Scott Hauck. "Performance of partial reconfiguration in FPGA systems: A survey and a cost model." ACM Transactions on Reconfigurable Technology and Systems (TRETS) 4.4 (2011): 36.
34. Papadimitriou, Kyprianos, Antonis Anyfantis, and Apostolos Dollas. "An effective framework to evaluate dynamic partial reconfiguration in FPGA systems." IEEE Transactions on Instrumentation and Measurement 59.6 (2010): 1642-1651.

AUTHOR PROFILE

Dr. Manjith B.C.,(manjithbc@gmail.com), Department of Computer Science and Engineering, National Institute of Technology Puducherry, Karaikal-609609. She is currently working as a faculty on contract at NIT Puducherry. She got her B.Tech degree(Computer Science and Engineering) from Kerala University, ME degree (Computer Science and Engineering) from Anna University and Ph.D. from National Institute of Technology, Tiruchirappalli. Her research interests include reconfigurable computing, FPGA, evolvable hardware and hardware security.

.