AES High-Level SystemC Modeling using Aspect Oriented Programming Approach

Hassen Mestiri
Prince Sattam bin Abdulaziz University, College of Computer Engineering and Sciences, Department of Computer Engineering, Alkharj, Saudi Arabia
and
Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse, Tunisia
and
Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Tunisia
h.mestiri@psau.edu.sa

Imen Barraj
METS Research Group, Electrical Engineering Department, National Engineers School of Sfax
University of Sfax, Tunisia
and
Higher Institute of Computer Science and Multimedia of Gabes
University of Gabes, Tunisia
imen.barraj@gmail.com

Mohsen Machhout
Electronics and Micro-Electronics Laboratory
Faculty of Sciences of Monastir
University of Monastir
Monastir, Tunisia
mohsen.machhout@fsm.rnu.tn

Abstract-The increasing complexity of the cryptographic modeling and security simulation of the Advanced Encryption Standard (AES) necessitate fast modeling and simulation security environment. The SystemC language is used in Electronic System Level (ESL) that allows cryptographic models to achieve high security and modeling simulation speed. Yet, the use of SystemC in the security simulation requires modifications of the original code which increases the modeling complexity. The Aspect-Oriented Programming (AOP) can be used in the cryptographic modeling and security simulations without any code modification. In this paper, a new AES SystemC model using the AOP technique is presented. A functional verification environment is proposed to test the functionality of the AES SystemC AOP model, the impact of AOP on simulation time, and the size of the executable files. The design of the AES model is developed with the weaving of all modules by AspectC++ which is an AOP language. The Simulation results show the efficiency of the proposed AES model and the uses of the AOP technique do not have a significant impact on simulation time or on the size of the executable file.

Keywords-security; cryptographic; AES; SystemC; AOP; high-level

I. INTRODUCTION

Cryptographic systems are implemented in embedded systems to protect secret information. Those systems store the encryption key in conjunction with the cryptographic algorithm execution [1-3]. The modeling complexity of cryptographic systems is increasing more than the verification capability and the design of the developers [4]. SystemC is used for the modeling and verification of complex systems. It has been considered as a suitable language for developing cryptographic models. Yet, its uses need to modify the original SystemC code in order to weave any cryptographic modules [5, 6]. To avoid modifying the cryptographic algorithms code under test, a new technique is used: Aspect Oriented Programming (AOP). It consists of weaving a module into the original code without any modification [7, 8]. Until now, a few cryptographic models and verification security environments using SystemC and AOP have been presented. In [9], the authors presented a security SystemC benchmark for high-level synthesis. They developed a security benchmark suite in a behavioral language supported by all major HLS vendors with different types of trojan hardware which produce different effects. A security threat analysis using SystemC to simulate the power attacks is presented in [10]. The authors developed a new approach to simulate power attacks in early stages. The authors also developed a power consumption graphical interface to analyze the security of the elliptic curve cryptography and RSA encryption against simple power attacks and differential power attacks respectively. The authors in [7] proposed a new fault injection system based on aspect-oriented programming (AspectC++). This system allows an automated injection of errors without modifying the original code. The authors showed that their system provides automatisms to generate the test environment of embedded systems.

Corresponding author: Hassen Mestiri

In this paper, we present a new AES SystemC model using the AOP technique. We summarize our contributions as:

- AspectC++ has been used as an AOP technique to avoid modifying the cryptographic algorithm code.
- A new cryptographic AES SystemC model using the AOP technique is proposed.
- A functional verification environment is proposed to test the functionality of the AES SystemC AOP model.
- The proposed AES SystemC-AOP model has been simulated in order to estimate the impact of AOP on simulation time and on the size of the executable files. Through the simulations, it was shown that the efficiency of the proposed AES model and the uses of AOP technique do not have a significant impact on simulation time and on the size of the executable file.

II. ASPECT ORIENTED PROGRAMMING

AOP is a programming technique based on the principle of the separation of concerns [7, 8]. In the AOP, an application is made up of classes and aspects. The transversal code linked to non-functional concerns is put into modules in the form of aspects. The aspects will then be weaved into the functional code in order to generate a complete application. Aspects are used to implement technical functionalities that are found to be dispersed in the code of an application. An aspect consists of two parts: Pointcut and advice code.

- Pointcut allows defining the place where the code of the transverse functionality of the associated aspect will be applied. This is defined by one or more Join Points. Each Join Point represents a point in which the code of a transverse functionality will be inserted. There are two types of Join Points: call and execute.
- An advice code is a part of the code that will be inserted in the places defined by the Join Points. It denotes the way in which cross functionality will be weaving.

An aspect can include several advice codes at the same time where each advice is associated with a cut. There are three types of advice codes:

- before: the code will be executed before the cut.
- after: the code will be executed after the cut.
- around: part of the code will be executed before the cut and another will be executed after.

To weave a new feature like an aspect into the code of an application, junction points must be defined in the primary code to indicate where the aspect should act. Figure 1 shows how the aspect code of the new functionality is weaving into the application code. The AOP technique is widely used in the field of testing embedded programs written in C ++ and Java. In our work, in order to avoid the modifications of the source code when inserting the codes relating to the AES model and the analysis of the cryptographic system, the use of the AOP technique to design a new AES model is proposed. Since the AOP technique allows the separation of transversal concerns, we can obtain a new distribution of the cryptographic model. This distribution consists of class and aspects which present the codes relating to the modules to be inserted.

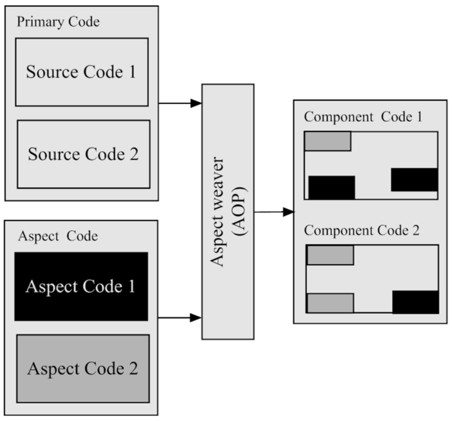


Fig. 1. Weaving aspect code into source code.

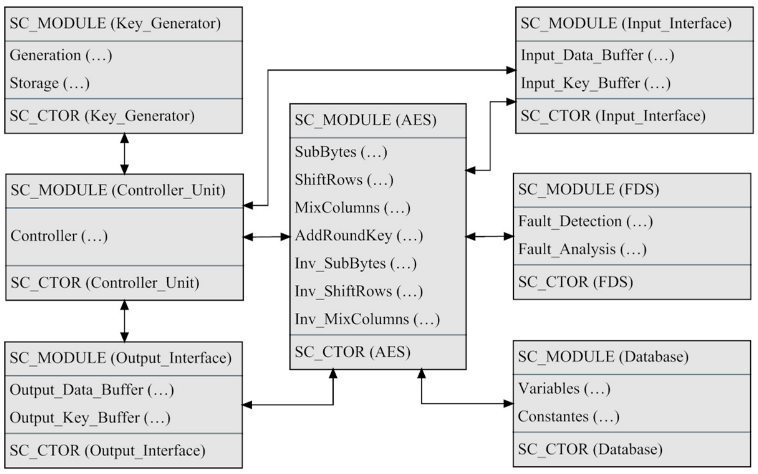


Fig. 2. Diagram of the AES SystemC AOP design.

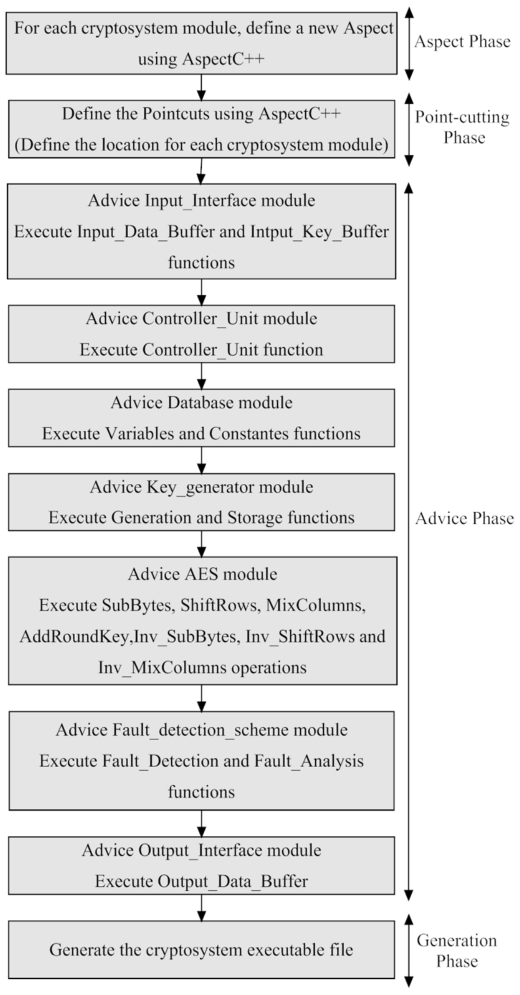

For each cryptosystem module, define a new Aspect using AspectC++
Aspect Phase
Define the Pointcuts using AspectC++ (Define the location for each cryptosystem module)
Point-cutting Phase
Advice Input_Interface module Execute Input_Data_Buffer and Intput_Key_Buffer functions
Advice Controller_Unit module Execute Controller_Unit function
Advice Database module Execute Variables and Constantes functions
Advice Key_generator module Execute Generation and Storage functions
Advice AES module Execute SubBytes, ShiftRows, MixColumns, AddRoundKey,Inv_SubBytes, Inv_ShiftRows and Inv_MixColumns operations
Advice Fault_detection_scheme module Execute Fault_Detection and Fault_Analysis functions
Advice Output_Interface module Execute Output_Data_Buffer
Advice Phase
Generate the cryptosystem executable file
Generation Phase

Fig. 3. Flowchart of the AES cryptosystem aspect.

III. SYSTEMC AES MODELING

The goal of this section is to design an AES model in SystemC using the AOP approach. For this purpose, the cryptographic operations performed by the AES cryptosystem were divided into several modules. Figure 2 presents a detailed description of the proposed AES model. The diagram of the AES SystemC model presents the different cryptographic modules as well as their interconnections using AspectC++ which is an AOP language. The AES SystemC AOP model is described in 7 modules:

- AES
- Key_generator
- Controller_Unit
- Input_Interface
- Output_Interface
- Fault_Detection_Scheme
- Database

Figure 3 presents the flowchart of the aspect used by the AES cryptosystem. The aspect code is woven by the aspectC++ weaver into original code. The SystemC-AOP flow has four main phases. The Aspect phase consists to create a new aspect for each AES cryptosystem module using AspectC++. The point-cutting phase consists to define the weaving location for each module. The third phase is to insert all modules in the pointcuts and then execute all functions:

- The Input_Interface module reorganizes the input flow according to a standard format defined by the communications protocol. It executes the Input_Data_Buffer and Intput_Key_Buffer functions.
- The Controller_Unit module is a state controller that allows the synchronization of all modules of AES cryptographic model.
- The Database module stores all variables and constants used in the encryption and decryption process. It processes two functions: Variables and Constants.
- The AES module is the main module, it performs the encryption and decryption processes via 7 operations: SubBytes, ShiftRows, ShiftRows, AddRoundKey, Inv_SubBytes, Inv_ShiftRows and Inv_MixColumns.
- The Fault Detection sSheme (FDS) module is developed to protect the AES module against fault attacks. It executes Fault_Detection and Fault_Analysis functions.
- The Output_Interface module reorganizes the encrypted stream according to a standard format defined by the communications protocol. It executes the Output_Data_Buffer function.

In the fourth phase, after the weaving process, the resulting source code can be compiled by SystemC and AspectC++ compilers and the resulted executable file is therefore generated.

IV. FUNCTIONAL VERIFICATION ENVIRONMENT

In this section, the proposed functional verification environment developed to validate the AES model is presented. This environment is inspired by the transaction-based verification environment in the SystemC. The AES model verification environment includes a reference model that presents a high level design description. This model is executed in parallel with the proposed model in order to compare the output results during the simulation. The reference model often models the interaction between modules at the transactional level rather than at the signal level. In this case, this module is written in TLM. Figure 4 shows the proposed environment architecture. As presented by Figure 4, the testbench SCV is a module that randomly generates the encryption keys and plaintext for the proposed and the reference models. The transactor receives the stimuli from the testbench SCV and applies them to the inputs of the proposed model. At the end of each transaction, the outputs are compared and the results are generated by a comparator module.

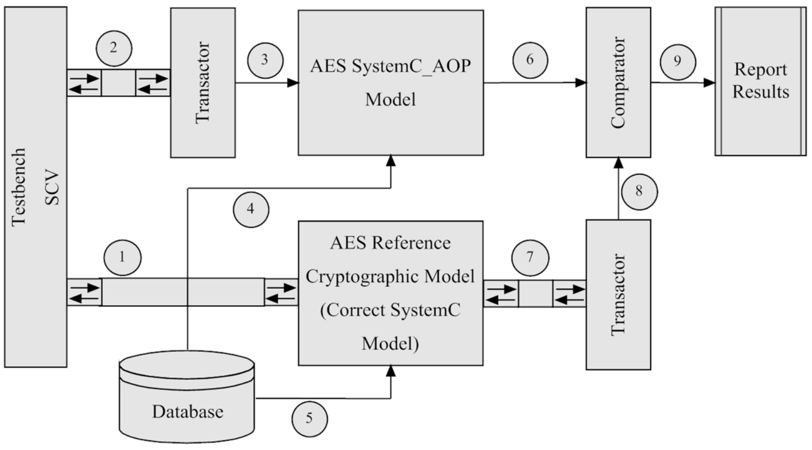


Fig. 4. Functional verification environment.

(1) + (2): The stimuli are generated randomly by the testbench SCV towards the two cryptographic models: proposed and reference models.

(3): The transactions are converted into signals by the transactor module to be adapted to the inputs of the non-TLM model (proposed model).

(4) + (5): Both models use the Database.

(6) + (8): The AES SystemC based AOP outputs and the transactor outputs are the inputs of the comparator module.

(7): The TLM output of reference model

(8): The transactions converted into signals by the transactor to be adapted to the outputs of the non-TLM model.

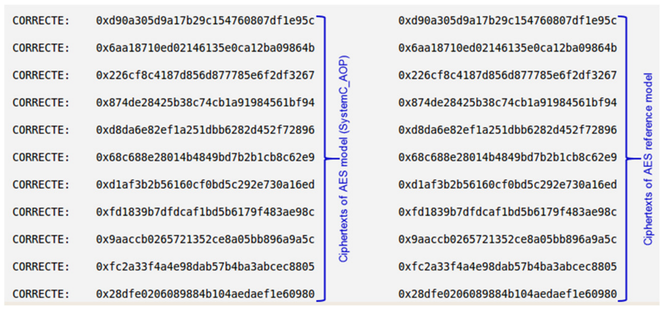
(9): The comparison report results are generated.

V. RESULTS AND DISCUSSION

This section is devoted to the evaluation of the proposed SystemC-AOP model as well as to the analysis of the impact of using SystemC and AspectC++ at the ESL level on the cryptographic design. First, we will perform a series of simulations to validate the proposed SystemC-AOP model. This validation is performed using the functional verification environment where the proposed model is compared with the reference model. Secondly, we will analyze the impact of using the AOP technique on the simulation time and on the size of the executable file of cryptographic model. The modeling was carried out with SystemC 2.3.2 language and AspectC++ 2.2, and the validation of the AES SystemC AOP model was carried out on a PC equipped with the Ubuntu 18.4 operating system, an Intel Core I3-4010U 1.7GHz microprocessor, a RAM of 6GB, and the gcc 7.2.0 compiler.

A. AES SystemC AOP Model Validation

A series of simulations will be performed in order to validate the proposed SystemC-AOP model. The validation is performed using the functional verification environment. The keys and data randomly generated by the testbench module are distributed to the reference AES model and the proposed AES model through the transactor module. The two models are simulated with the same simulation model and using the same test conditions. The verification results show that the proposed AES SystemC model works perfectly and it provides the same results as the reference model. Figure 5 shows an extract of the report simulation results.



```
CORRECTE:    0xd90a305d9a17b29c154760807df1e95c        0xd90a305d9a17b29c154760807df1e95c
CORRECTE:    0x6aa18710ed02146135e0ca12ba09864b        0x6aa18710ed02146135e0ca12ba09864b
CORRECTE:    0x226cf8c4187d856d877785e6f2df3267        0x226cf8c4187d856d877785e6f2df3267
CORRECTE:    0x874de28425b38c74cb1a91984561bf94        0x874de28425b38c74cb1a91984561bf94
CORRECTE:    0xd8da6e82ef1a251dbb6282d452f72896        0xd8da6e82ef1a251dbb6282d452f72896
CORRECTE:    0x68c688e28014b4849bd7b2b1cb8c62e9        0x68c688e28014b4849bd7b2b1cb8c62e9
CORRECTE:    0xd1af3b2b56160cf0bd5c292e730a16ed        0xd1af3b2b56160cf0bd5c292e730a16ed
CORRECTE:    0xfd1839b7dfdcaf1bd5b6179f483ae98c        0xfd1839b7dfdcaf1bd5b6179f483ae98c
CORRECTE:    0x9aaccb0265721352ce8a05bb896a9a5c        0x9aaccb0265721352ce8a05bb896a9a5c
CORRECTE:    0xfc2a33f4a4e98dab57b4ba3abcec8805        0xfc2a33f4a4e98dab57b4ba3abcec8805
CORRECTE:    0x28dfe0206089884b104aedaef1e60980        0x28dfe0206089884b104aedaef1e60980
```

Ciphertexts of AES model (SystemC_AOP)
Ciphertexts of AES reference model

Fig. 5. AES SystemC AOP: Report simulation results.

B. Impact of AOP on Simulation Time

To analyze the impact of AOP on simulation time, a series of simulations were performed using the developed environment. This process consists of measuring kernel time (kTime) and user time (uTime) using two fault detection schemes. Note that kTime and uTime depend on the number of Join Points as well as the number of modules to be inserted via the AOP technique. Table I shows the simulation results in terms of kTime and uTime using two scenarios (SystemC and SystemC_AOP). We can see that the margin of error of the measurement tool (Linux commands) that was used is very low and does not affect the accuracy of the simulation results. These results show that the weaving of Key_generator, Controller_unit, Database, Input_interface, Output_interface and FDS modules into the SystemC model, using the AOP technique, does not affect simulation times. Another very important parameter for analyzing the impact of AOP is the size of the resulting executable file. Table II presents the size of the executable files generated by AspectC ++ and the SystemC kernel for the two scenarios.

The simulation results show that even when the proposed model is modeled using two languages (SystemC and AspectC ++) and that the weaving of all modules is carried out via the

AOP technique, the size of the executable files remains almost unchanged. This shows that the AOP does not have a significant impact on the size of the executable file.

TABLE I. SIMULATION TIME ANALYSIS WITH AND WITHOUT AOP

AES (SystemC AOP)	kTime (s)		uTime (s)	
	SystemC	SystemC_AOP	SystemC	SystemC_AOP
AES model protected by FDS [1]	0.039	0.041	2.133	2.135
AES model protected by FDS [11]	0.036	0.037	2.065	2.070
AES model protected by FDS [12]	0.041	0.038	2.259	2.265

TABLE II. SIMULATION OF THE EXECUTABLE FILE WITH AND WITHOUT AOP

AES (SystemC AOP)	SystemC	SystemC_AOP
AES model protected by FDS [1]	1.226Mo	1.219Mo
AES model protected by FDS [11]	1.305Mo	1.309Mo
AES model protected by FDS [12]	1.346Mo	1.350Mo

VI. CONCLUSION

In this paper a new AES SystemC model using the AOP technique at the Electronic System Level was presented. A functional verification environment was proposed to test the functionality of the AES SystemC AOP model, the impact of AOP on simulation time and on size of the executable files. The simulation results show the efficiency of the proposed AES model and the weaving of AES modules using the AspectC++ does not have a significant impact on simulation time and on the size of the executable file.

ACKNOWLEDGMENT

This publication was supported by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University, Alkharj, Saudi Arabia.

REFERENCES

[1] H. Mestiri, N. Benhadjyoussef, and M. Machhout, “Fault Attacks Resistant AES Hardware Implementation,” in *2019 IEEE International Conference on Design Test of Integrated Micro Nano-Systems (DTS)*, Gammarth-Tunis, Tunisia, Apr. 2019, https://doi.org/10.1109/DTSS.2019.8914979.

[2] J. Zhang, N. Wu, F. Zhou, F. Ge, and X. Zhang, “Securing the AES Cryptographic Circuit Against Both Power and Fault Attacks,” *Journal of Electrical Engineering & Technology*, vol. 14, no. 5, pp. 2171–2180, Sep. 2019, https://doi.org/10.1007/s42835-019-00226-6.

[3] E. S. I. Harba, “Secure Data Encryption Through a Combination of AES, RSA and HMAC,” *Engineering, Technology & Applied Science Research*, vol. 7, no. 4, pp. 1781–1785, Aug. 2017, https://doi.org/10.48084/etasr.1272.

[4] A. Alamer and B. Soh, “Design and Implementation of a Statistical Testing Framework for a Lightweight Stream Cipher,” *Engineering, Technology & Applied Science Research*, vol. 10, no. 1, pp. 5132–5141, Feb. 2020, https://doi.org/10.48084/etasr.3250.

[5] B. Lin and F. Xie, “SCBench: A benchmark design suite for SystemC verification and validation,” in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju, South Korea, Jan. 2018, pp. 440–445, https://doi.org/10.1109/ASPDAC.2018.8297363.

[6] S. Aygün, L. Kouhalvandi, B. Örs, and E. O. Güneş, “Karatsuba Ofman Multiplication implementation on SystemC for Diffie-Hellman Key Exchange algorithm,” in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Tehran, Iran, Dec. 2017, pp. 0641–0645, https://doi.org/10.1109/KBEI.2017.8324878.

[7] U. T. Gabor, C. von Egidy, and O. Spinczyk, “Interface Injection with AspectC++ in Embedded Systems,” in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, Hangzhou, China, Jan. 2019, pp. 131–138, https://doi.org/10.1109/HASE.2019.00028.

[8] C. Borchert and O. Spinczyk, “Hardening an L4 Microkernel Against Soft Errors by Aspect-Oriented Programming and Whole-Program Analysis,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 2, pp. 37–43, Jan. 2016, https://doi.org/10.1145/2883591.2883600.

[9] N. Veeranna and B. C. Schafer, “S3CBench: Synthesizable Security SystemC Benchmarks for High-Level Synthesis,” *Journal of Hardware and Systems Security*, vol. 1, no. 2, pp. 103–113, Jun. 2017, https://doi.org/10.1007/s41635-017-0014-1.

[10] J. Treus and P. Herber, “Early Analysis of Security Threats by Modeling and Simulating Power Attacks in SystemC,” in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, Antwerp, Belgium, May 2020, pp. 1–5, https://doi.org/10.1109/VTC2020-Spring48590.2020.9129426.

[11] M. Bedoui, H. Mestiri, B. Bouallegue, and M. Machhout, “A reliable fault detection scheme for the AES hardware implementation,” in *2016 International Symposium on Signal, Image, Video and Communications (ISIVC)*, Tunis, Tunisia, Nov. 2016, pp. 47–52, https://doi.org/10.1109/ISIVC.2016.7893960.

[12] M. Bedoui, H. Mestiri, B. Bouallegue, M. Marzougui, M. Qayyum, and M. Machhout, “An improved and efficient countermeasure against fault attacks for AES,” in *2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*, Abha, Saudi Arabia, Mar. 2017, pp. 209–212, https://doi.org/10.1109/Anti-Cybercrime.2017.7905292.