# AES CryptoCore

**Catalina Sarbu**
**Catalinasarbu.0595@gmail.com**

**Mihai Alexandru Olaru**
**Alexmhiolaru96@gmail.com**

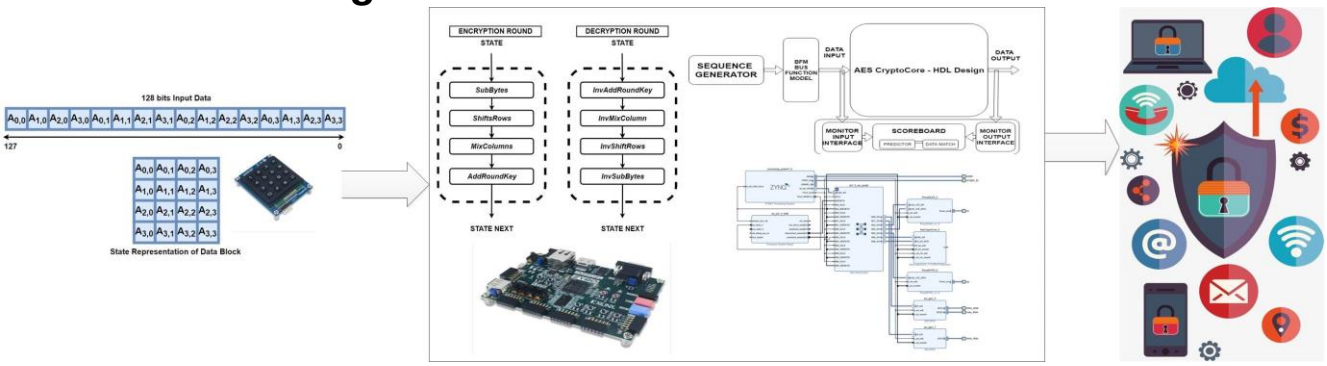
**Submitted for the 2019 Digilent Design Contest Europe**

**07.05.2019**

**Advisor: Dan Marius Dobrea**

**University ”Gheorghe Asachi” din Iasi**
**Iasi, Romania**

# Product Marketing Sheet



# Introduction

NIST (US NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY) called for a new Advanced Encryption Standard. The process for selection of this algorithm was open for the community. The following requirements for all AES submission were mandatory:

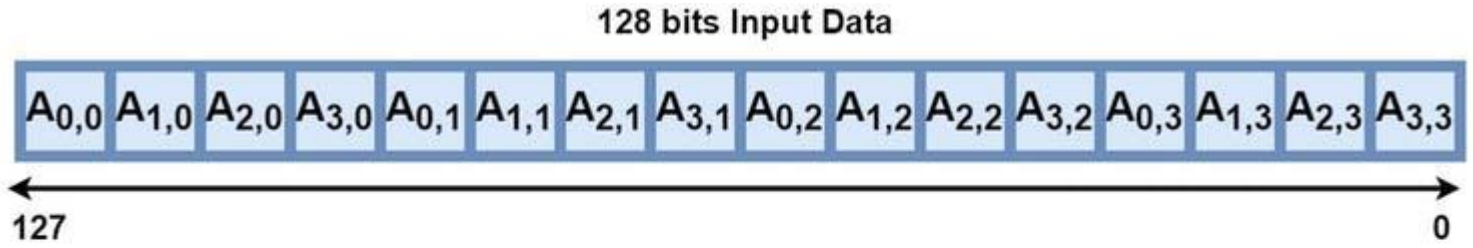
- block cypher with a 128-bit block size
- three key lengths must be supported: 128, 192 and 256 bit
- security
- efficiency in software and hardware

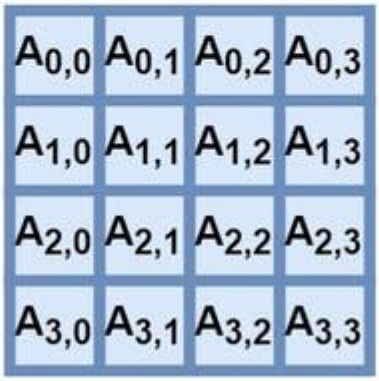
NIST declared the block cipher Rijndael as the new AES and published it as a final standard in 2001.

AES is a symmetric-key block cipher. It is an iterative cipher, which means that bots decryption and encryption consist of multiple iterations of the same basic round function.

In each round, a different key is generated according to the rounds index. The number of rounds depends on the size of the key. For this project, the number of rounds is equal with 10 because of the key length is 128 bits.

The input can be represented as a regtangular array of bytes, with four rows. The number of columns is equal to the block length divided by 32. Similarly, the key is mapped onto a two-dimensional cipher key.




State Representation of Data Block

State Represantation

**Abstract**

This project want to provide a security environment for your hard disk and your stored data. The hard drive will be encrypted with a password of up to 8 digits provided by the user. An enhanced fully pipelined an area efficient XTS-AES(**Advanced Encryption Standard**) mode design using one AES core is our sollution for this problem.

**Objectives**

The objectives of our project, AES CryptoCores, are encrypting and decrypting data using a user key.

**Features-in-Brief**

New hard disks save the already encrypted data and after a malfunction the data can be found only encrypted and the decryption would involve a difficult process that sometimes can not be achieved. By saving the unencrypted data first, this could help us a lot in the case presented above.

**Project Summary**

This system requires two steps of implementation. First, the system must be able to communicate with a hard disk and take the data from it. Also, the system need to respond of the commands of a Master. In our case, the test platform will have a CPU which will be the traffic generator. The CPU will generate Read and Write Requests for tests. The traffic will be generated using C program. The Second part of the implementation is to incorporate the AES core into the Zybo FPGA through the data flow, also the data provided by the C program will be stored in other form on hard disk.

For the security of information, the user shall introduce a public key for the AES algorithm using a keypad. After the key is correct introduced, the AES Core will start to decode the data from the HDD. After decoding, data is saved in the system memory by the DMA. On the Write operation, AES core will encrypt the data.

This kind of data protection, can be used by the companies which gives laptops to developers which are working from home. Also, the AES core can be integrated in different structure to encrypt data (CAN communication in automotive), because the main proposal is to offer real-time hardware encryption.

### Digilent Products Required

Digilent Zybo
Digilent Pmod KYPD
Digilent Pmod BT2
Digilent Pmod OLED

### Tools Required

Xilinx Vivado Design Suite

### Design Status

Our project is incomplete, we think we would need a few more days to complete. We did not abandon the design but just a series of events that impacted the work on the project and that's why we stayed a bit behind, we consider that only time prevents the completion of the project.

# Background

### Why This Project?

AES is a very popular algorithm that attracted our attention due to the fact that we found it interesting to use it for encryption and decryption of the hard drive. It is easy to use and understand both hardware and software and has allowed us to play our hard disk and gain experience in this area.

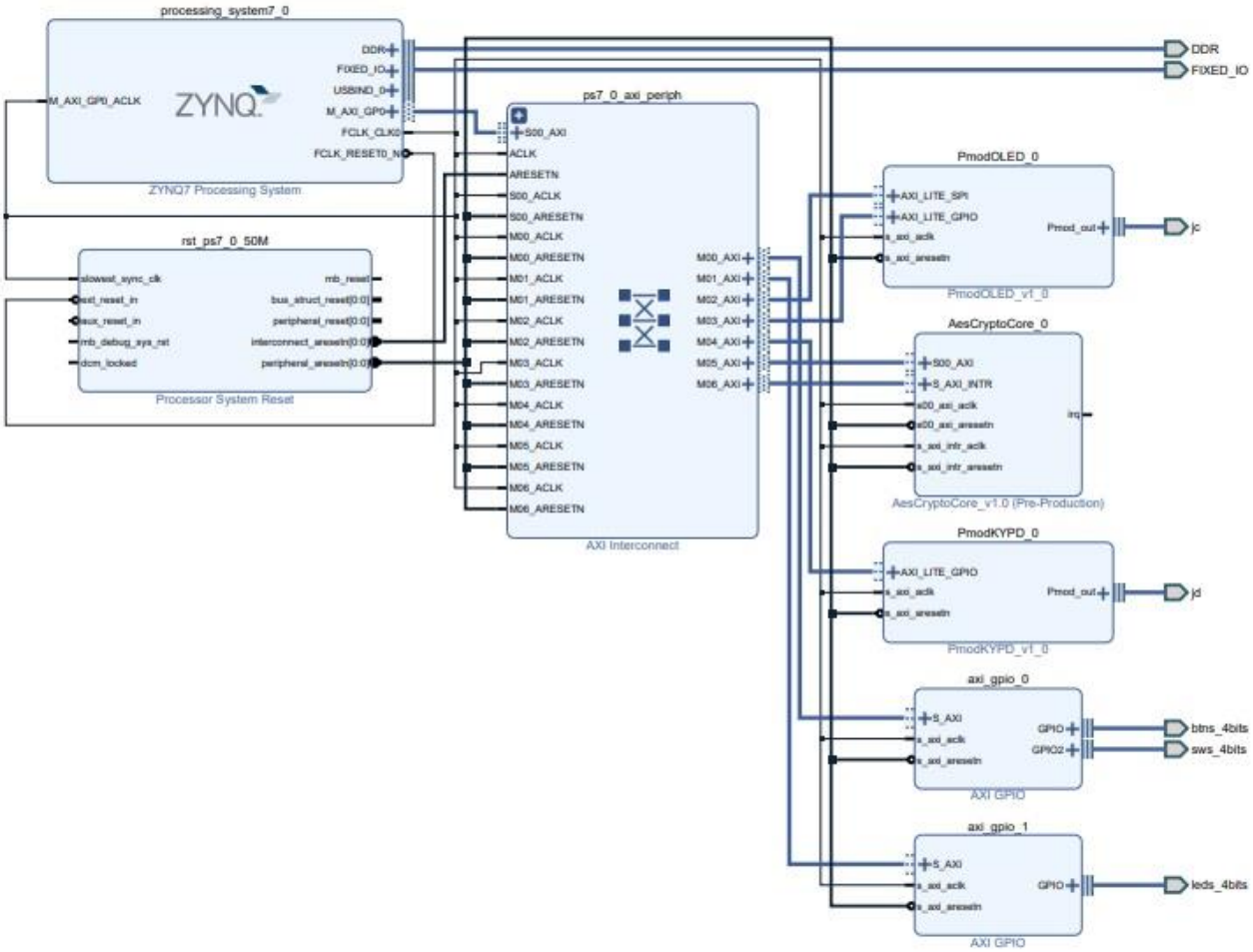
### Reference Material

# Design

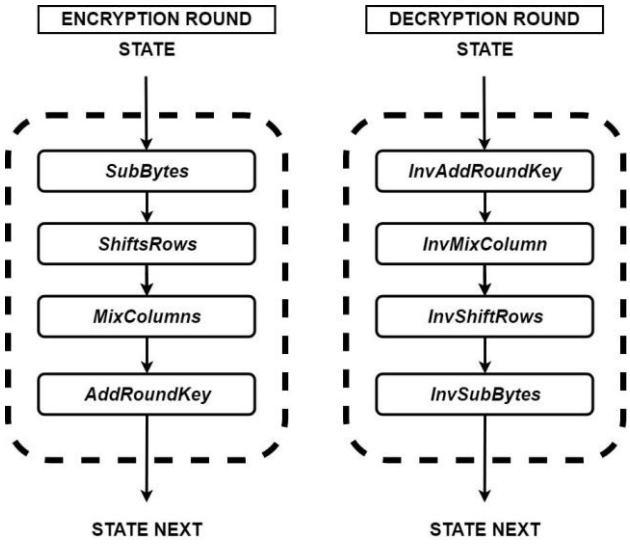
### Features and Specifications

The number of Pipeline Levels can be increased to receive a high speed encryption.

## Design Overview



## Detailed Design Description




### S-Box - Presentation

The AES algorithm operates on a 4x4 array of bytes which is called a state. The state are 4 transformation which are namely the AddRoundKey, SubByte, ShiftRow and MixColumn.

The SubByte state is the only one which is non-linear substitution where each byte in the state array is replaced with another from a lookup table called an S-Box.

The design criteria for the S-box are :

- Non-Linearity -> Correlation : The maximum input-output correlation amplitude must be as small as possible
- Non-Linearity -> Difference propagation probability
- Algebraic complexity : The expression in GF(2^8) must be complex.

The S-box is defined by the following function in GF(2^8) :

g : a -> b = a^-1

a^-1 is the inverse element : a * a^-1 = 1

SubByte : Multiplicative Inversion -> Affine Transformation

1. *Inverse*: Let $c = a^{-1}$, the multiplicative inverse in $GF(2^8)$ (except if $a = 0$ then $c = 0$).
2. *Affine Transformation*: Then the output is $s = M\,c \oplus b$, with the constant bit matrix $M$ and byte $b$ shown below:

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

where bit #7 is the most significant, with all bit operations modulo 2.

InvSubByte : Inverse Multiplicative Transformation -> Multiplicative Inversion in GF(2^8)
The Affine Transformation can be represented in matrix form:

$$AT(a) = \begin{pmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \qquad (1.1)$$

$$AT^{-1}(a) = \begin{pmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{pmatrix} \times \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad (1.2)$$

The last version of S-Box has precomputed values stored in ROM memory and the input value is wired to the ROM's address bus. This implementation is expensive in terms of hardware. Another way is an arithmetic implementation of S-Box Module. The basic idea of the logic-only implementation is to notice that inversion in GF(2^8) can be decomposed into a sequence of operation in GF(2^4). (including addition, multiplication and inversion). Similarly, from GF(2^4) -> GF(2^2), from GF(2^2) -> GF(2). The operation in GF(2) can be implemented using simple XOR Gate(addition) and AND gate(multiplication).

Matrix Multiplication can be changed to a logical XOR Transformation.
Any arbitrary polynomial can be represented by bx+c where b is upper half term and c is the lower half term. Using this idea, the logical equations for the addition, squaring, multiplication and inversion can be derived.

## For FPGA implementation :

**Cryptographic Engineering : Cap 10.6 -> Subbytes and InvSubBytes**
**Rijndael_SBox_FPGA : Edwin NC Mui**

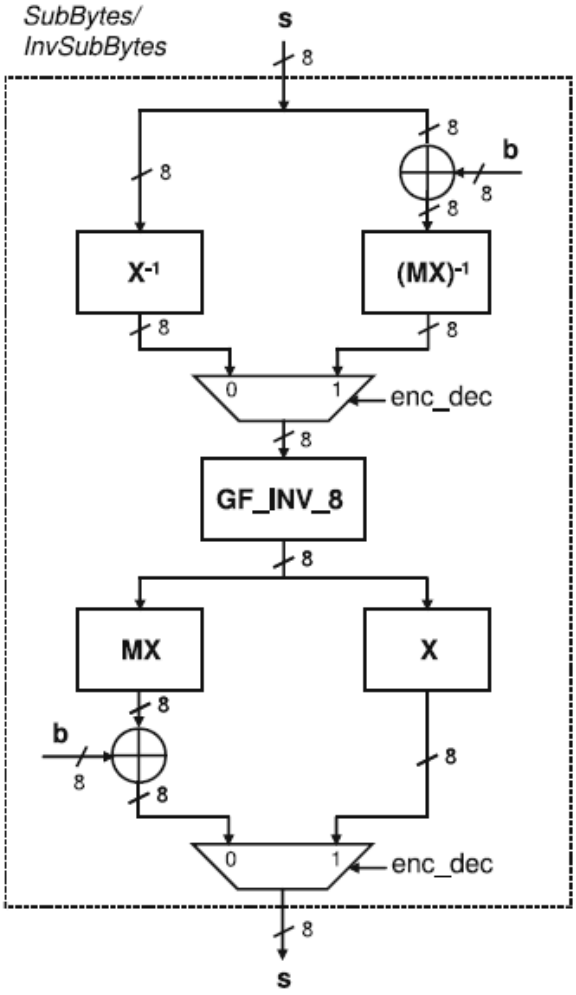



**Fig. 10.28** Implementation of *SubBytes* and *InvSubBytes* using logic only, according to [3, 4]. The notation follows conventions introduced in [3]. *enc_dec* is a select signal equal to 0 for encryption and 1 for decryption. $X$ is an $8 \times 8$ basis conversion matrix, $M$ is an $8 \times 8$ matrix, and $b$ is an $8 \times 1$ bit vector, where $y' = M \cdot y + b$, with $b = \{63\}$, is an equation describing the affine transformation of *SubBytes*.

This layer provides difussion, so simply modifying a bit in a byte must influence the calculation of the other bytes. The combination of the ShiftRows and MixColumn layer makes it possible that after only three rounds every byte of the state matrix depends on all 16 plaintext bytes.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

MIxColumn Layer

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0E\ 0B\ 0D\ 09 \\ 09\ 0E\ 0B\ 0D \\ 0D\ 09\ 0E\ 0B \\ 0B\ 0D\ 09\ 0E \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

InvMixColumn Layer

SHIFTROWS Layer

The ShiftRows transformation cyclically shifts the seconds row of the state matrix by three bytes to the right, the third row by two bytes to the right and the fourth row by one byte to the right. The first row is not changed in this layer.

AddRounds Layer

The key addition layer consists in a simple XOR operation between the data from the current round and the internal key generated. The XOR operation is equal to addition in the Galois field GF(2).

## Discussion

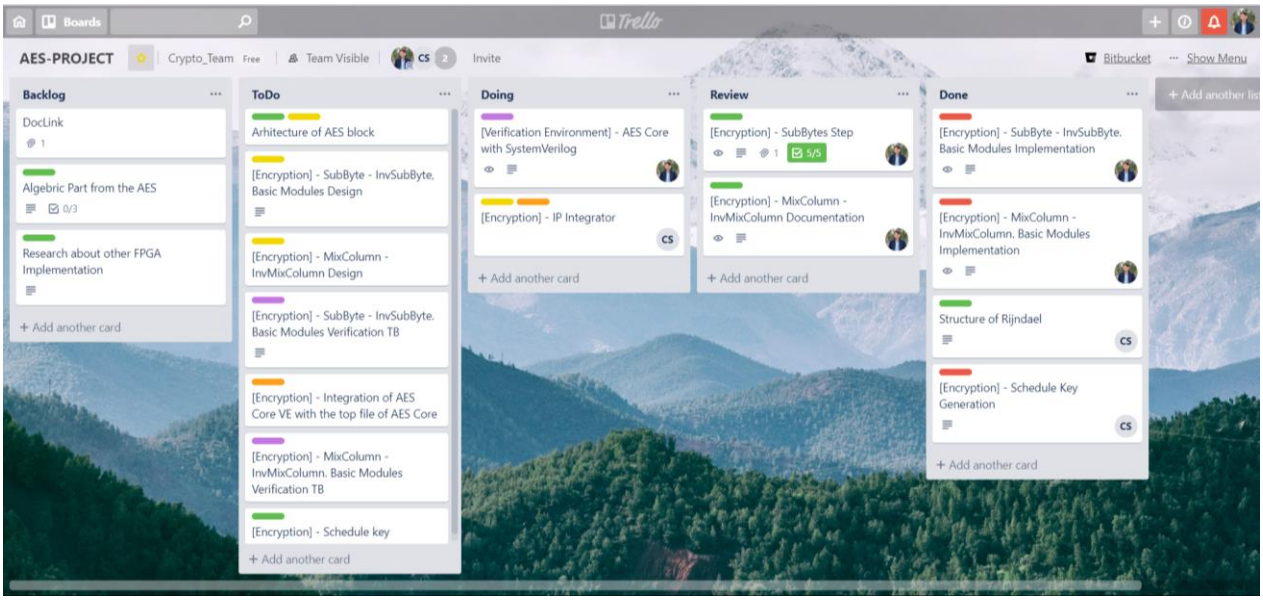
### Problems Encountered

Since the project has many features to be implemented, some limitation had to be set. The test platform will have implemented the communication part, the base function (Read, Write Operation) and implemente AES steps.

The difficult part was to integrate all the modular and configuration of each IP. The main challenge was to adapt the implementation of the non-synthesized code into a synthesizable code. Synthesizable code writing implies some implementation "rules" for the design to go through synthesis and implementation on an FPGA. Another challenge was the debug part, as being 3 IPs of the project, to discover where the problem is, the errors were eliminated one at a time.

The AES Core will be developed like an individual module which should be able to be integrated like a bridge in a flux of data.

### Engineering Resources Used




### Marketability

The AES project can be easily integrated in a commercial product because the algorithm is efficient for hardware implementation. The dedicated hardware can be used in different areas of interests such as communication protocols: UART, SPI, WIFI, Bluetooth. An innovation which we can be bring is the automotive subject because in our days, the cars use modern technologies and data traffic inside the car should be secured by encryption and decryption. For example for CAN bus communication or Ethernet protocols.

Also, the IoT Industry can be another zone of interests.

### Community Feedback

Insert text here.

## References

[1] Çetin Kaya Koç. *Cryptographic Engineering*. Springer, 2008.
[2] Christof Paar. *Understanding Cryptography.*Springer, 2010.
[3] Vincent Rijmen.*The Design of Rijndael AES – The Advanced Ecryption Standard.* Spriger 2002

# Appendix A: System Verilog Code Sample

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Mihai Olaru
//
// Create Date: 02/18/2019 02:52:17 PM
// File Name: AES Class, implemeting High Level Algorithm
// Project Name: AES - Crypto
// Description: Key Schedule Generator
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



class agent_KeySchedule extends BaseUnit;

      virtual key_schedule_intf intf;
      AesCore aes;


      extern function new(
                                    virtual key_schedule_intf p_intf,
                                    string name,
                                    int id,
                                    const ref AesCore aes_core);
      extern task run();



endclass: agent_KeySchedule

function agent_KeySchedule :: new(
                                                virtual key_schedule_intf p_intf,
                                                string name,
                                                int id,
                                                const ref AesCore aes_core);
      super.new(name,id);
      this.aes           = aes_core;
      intf = p_intf;

endfunction : new

task agent_KeySchedule :: run();

      word128 internal_key = 0;
      for (int i = 0; i <= `NO_OF_ROUNDS; i++) begin
```

```
            intf.key_in       = internal_key;
            intf.encrypt      = `ENCRIPT;
            intf.round_index = i;
            #1;
            $display("[%0t] %s %0d Internal key Generated %0h", $time,
super.name,i, internal_key);
            internal_key = intf.key_out;
      end


endtask:run
```

# Appendix B: Verilog RTL Code Sample

```
//
// Create Date: 02/18/2019 02:52:17 PM
// Design Name: MixColumns
// Module Name: MixColumns
// Project Name: AES - Crypto
// Target Devices:
// Tool Versions:
// Description: MixColumn Transformation. Difusion Layer in AES
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////
module MixColumns( b0, b1, b2, b3, a0, a1, a2, a3, c0, c1, c2, c3);

      input    [7:0] b0, b1, b2, b3;
      output   [7:0] a0, a1, a2, a3;
      output   [7:0] c0, c1, c2, c3;

      wire   [7:0] x0, x1, x2, x3;
      wire   [7:0] x0_out, x1_out, x2_out, x3_out;
      wire     [7:0] z0, z1;

      assign x0 = b3 ^ b0;
      assign x1 = b1 ^ b0;
      assign x2 = b2 ^ b1;
      assign x3 = b3 ^ b2;

      x_times xtime_inst0 (x0, x0_out);
      x_times xtime_inst1 (x1, x1_out);
```

```
    x_times xtime_inst2 (x2, x2_out);
    x_times xtime_inst3 (x3, x3_out);

    assign a0 = (x0_out ^ b1) ^ x3;
    assign a1 = (x1_out ^ b0) ^ x3;
    assign a2 = (x2_out ^ b3) ^ x1;
    assign a3 = (x3_out ^ b2) ^ x1;

    x_time_square xtime_sq0(a3^a1, z0);
    x_time_square xtime_sq1(a2^a0, z1);

    assign c0 = z1 ^ a0;
    assign c1 = z0 ^ a1;
    assign c2 = z1 ^ a2;
    assign c3 = z0 ^ a3;

endmodule
```