International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 5, October 2021, pp. 4221~4232
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i5.pp4221-4232

High-performance AES-128 algorithm implementation by FPGA-based SoC for 5G communications

Paolo Visconti[1], Ramiro Velazquez[2], Stefano Capoccia[3], Roberto de Fazio[4]
[1,3,4]Department of Innovation Engineering, University of Salento, Lecce, Italy
[2]Facultad de Ingeniería, Universidad Panamericana, Aguascalientes, Mexico

Article Info

Article history:

Received Oct 14, 2020
Revised Jan 14, 2021
Accepted Feb 4, 2021

Keywords:

5G communications
Advanced encryption standard
Field programmable gate array
VHDL
Xilinx ZCU102 platform

ABSTRACT

In this research work, a fast and lightweight AES-128 cypher based on the Xilinx ZCU102 FPGA board is presented, suitable for 5G communications. In particular, both encryption and decryption algorithms have been developed using a pipelined approach, so enabling the simultaneous processing of the rounds on multiple data packets at each clock cycle. Both the encryption and decryption systems support an operative frequency up to 220 MHz, reaching 28.16 Gbit/s maximum data throughput; besides, the encryption and decryption phases last both only ten clock periods. To guarantee the interoperability of the developed encryption/decryption system with the other sections of the 5G communication apparatus, synchronization and control signals have been integrated. The encryption system uses only 1631 CLBs, whereas the decryption one only 3464 CLBs, ascribable, mainly, to the *Inverse Mix Columns* step. The developed cypher shows higher efficiency (8.63 Mbps/slice) than similar solutions present in literature.

Corresponding Author:

P. Visconti
Department of Innovation Engineering, University of Salento
Road to Monteroni, Ecotekne Campus, Lecce, 73100, Italy
Email: paolo.visconti@unisalento.it

1. INTRODUCTION

Network security is the set of preventive measures, both hardware, and software, to protect files, drivers, and directories from unauthorized access, modification, destruction, and to provide *secure communication* between a sender and a receiver. The three primary security goals for *secure transmission* are *confidentiality*, *integrity*, and *authentication*; the *confidentiality* is ensured by the cryptography, which is a procedure of encoding a *plaintext* into an unintelligible format (encryption), known as ciphertext, through the use of a key, and allowing only the receiver to retrieve the original data by means of an identical key or a different one (decryption). Hence, a cryptographic process is mainly based on two components: a cryptographic algorithm and at least one secret key. The cryptographic systems are classified into two categories, based on how the keys are used, namely symmetric key cryptography (SKC), known as private key cryptography, and asymmetric key cryptography (AKC), known as public-key cryptography. In the first one, the two users have the same private key, employed to encrypt and decrypt the data. In contrast, in the latter, the senders and receivers use two keys; a public key employed for encoding the plaintext, known to both entities, and a private key for decoding the ciphertext. The best-known symmetric encryption algorithms include the already outdated data encryption standard (DES) and its successor, namely the advanced encryption standard (AES) [1]-[3]; instead, the asymmetric encryption algorithm most widespread is the rivest shamir adleman (RSA) [4].

With the evolution of 5G technology, new architecture, technologies, traffic strategies have been introduced, opening new challenges to be faced for ensuring security [5]-[8]. Specifically, several typologies of security are required, given the wide range of applications involved in 5G technologies, such as enhanced mobile broadband, massive and critical machine-type communication. For instance, internet of things (IoT) applications, featured by open and shared network architectures, cloud computing, virtualization, and polling technology require the development of a secure air interface to prevent information leakage [9]-[14]. Furthermore, access to a huge amount of nodes, with low latency, and high throughput is the prerogative of 5G, also considering nodes with limited computational resources, and power consumption [15]; therefore, lightweight and reliable security algorithms are required to support these applications [16]-[18]. The 5G systems use protection mechanisms for signal and user plane traffic, through encryption and integrity protection mechanisms [19]-[21]. Therefore, several reliable and well-known encryption algorithms from 4G systems have been extended to the 5G ones, such as SNOW 3G, AES-counter mode (CTR), and ZUC, as well as for integrity protection, AES-CMAC, SNOW 3G, and ZUC are commonly used [22].

The AES is the most widespread symmetric cypher, chosen in 2001 by the US National Institute of standards and technology (NIST), as a mandatory solution in several commercial and industrial systems. It operates on 128-bit data blocks, similarly to Rijndael cypher, available in three versions according to the keys' length, namely 128-bit, 192-bit, and 256-bit, and developable both in hardware and software [23]. The ciphertext is obtained through a given number of rounds, depending on the key length; for AES-128, ten rounds are required, each round, except the first, includes three layers, namely the *key addition layer* (corresponding to *AddRoundKey* operation), *substitution layer* (corresponding to *SubtitutionByte* operation), and the *diffusion layer* (corresponding to *ShiftRows* and *MixColumns* operations). Exceptions are made for the *AddRoundKey* function in the first round and the absence of the *MixColumns* operation in the last round.

The field programmable gate array (FPGA) platforms are employed in several applications, like video and imaging processing [24]-[26], military and medical applications [27]-[32], automotive, electronics for high-speed processing [33], [34], and more, thanks to their great reconfigurability and processing capacity. Furthermore, the FPGAs are widely applied to communication applications, specifically, to efficiently develop hardware implementations of encryption/decryption algorithms, exploiting the high performances and quick development time, and parallel implementation offered by these platforms [35]-[39]. Several cryptographic implementations, from fully pipelined to low-cost and low power consumption architectures, have been proposed in the scientific literature. Specifically, in [40], the authors proposed two efficient implementations of the encrypt-only AES-128 algorithm on the XC7Z010clq225-3 FPGA device, involving partial loop unrolling and multi-stage pipelining solutions; both developed implementations consume about 455 mW of dynamic power. Furthermore, Guzmàn *et al.* presented an FPGA-based implementation of the AES-128 algorithm, on Xilinx Virtex 5, operating in electronic codebook (ECB) and counter (CTR) modes. In the two methods, the implementation obtains a data throughput equal to 34.89 Gbit/s for encryption and 25.5 Gbit/s for the decryption. In [41], the authors introduced a high-throughput AES-128 implementation on the FPGA platform, based on the content addressable memory (CAM) scheme and high-efficiency SubBytes block. The proposed design supports a 75.3 MHz maximum operational frequency with a 32 Gbit/s throughput value. Zodpe *et al.* developed a new generation scheme for Sbox and initial key, applicable to AES implementations, by using the pseudo noise (PN) sequence generator [42]. The proposed solution was tested on several FPGA platforms, demonstrating significant improvements in data throughput values. Furthermore, the authors in [43] introduced a secret multi-dimensional symmetric encryption algorithm, based on substitution-permutation network and supporting high-speed processing in six dimensions; a parallel encryption structure, employing a 128-bit block for each dimension, has been implemented, so allowing to manage large data volumes.

In the present manuscript, an FPGA high-throughput AES-128 cypher is presented, thought for 5G communications. Specifically, a pipelined framework has been used to implement the AES-128 encryption/decryption system, enabling parallel elaboration of multiple data packets every clock cycle, and thus higher throughput; the proposed implementation employs a 32 bit 16 x 16 Sbox for speeding up the processing time of the AES-128 rounds. The simulations and on-field tests demonstrated that an operating frequency up to 220 MHz is supported by both encryption and decryption blocks, allowing encryption/decryption time of just ten clock cycle, and accepting and providing data packets each clock cycle, resulting in a maximum throughput over 28 Gbit/s (namely, $128\frac{bit}{packet} \times 220MHz = 28.16\,GHz$). A Zynq Ultrascale+MPSoC ZCU102 board (produced by Xilinx Inc.), using Zynq Ultrascale+ XCZU9EG-2FFVB1156E multiprocessor system-on-chip (MPSoC), has been employed for developing the encryption/decryption blocks, and support a maximum operating frequency equal to 350 MHz [44].

In addition, a fast key expansion algorithm has been implemented, combining, through GF functions, the previous sub-key with the current sub-key transformed by the Sbox; therefore, the key

expansion step lasts just 174.55 ns for deriving the 44 words from the encryption key. Besides, the proposed VHSIC-very high-speed circuits-hardware description language (VHDL) encryption/decryption system includes several control signals for synchronizing it with the data generator block and the modulator/demodulator, respectively; also, different blocks have been added for testing the developed encryption/decryption block, by placing in a deterministic way an error into the incoming data packet and checking the correctness of the resulting encrypted/decrypted packet, indicated by a suitable error signal. A suitable mechanism has been developed to substitute the encryption key in every instant of the encryption process, resulting in the loss of just three data packets in each substitution process. The simulation results and on-field tests have demonstrated the proper operation of both encryption and decryption blocks and higher efficiency in the utilization of hardware resources (i.e. 8.63 Mbps/slices) than similar implementations present in the scientific literature.

The rest of the paper is organized as follows: the section 2 reports the design and implementation of the proposed encryption and decryption systems and all the VHDL sections developed to verify their operation. Section 3 presents the results of post-implementation and post-synthesis simulations carried out on the cascade system of the cypher and decryptor. Finally, in the fourth section, the performances of the proposed encryption/decryption system are discussed, comparing them with similar implementations reported in the literature.

2. RESEARCH METHOD

The *Xilinx* Vivado Design Suite has been used for developing the proposed encryption/decryption system, exploiting the wide range of tools offered to the designers. The block diagram related to the encryption system is shown in Figure 1, along with all the blocks for testing its correct operation; the *AXI Stream* bus provides the 128-bit plaintext data packets in input and ciphered packets in output from the cypher. The function of all the implemented blocks is following described:

- *Insert Error* block to insert an error into a packet included in the internal table of the *Data Generator* block (blue box in Figure 1).
- *Clock generator* block to generate the 220 MHz system clock to all blocks (purple box in Figure1);
- *Key_Generator* block for providing the encryption key to the *Key_to_write* block, which loads the key into the memory registers by AXI Lite bus; also, this block changes the encryption key.
- *Data Generator* provides the 128-bit plaintext data packets to the *AES_AXIS_KEY* encryption section (yellow box in Figure 1).
- *Key_to_write* block for loading the encryption key into the memory registers, through the AXI Lite bus, implementing all the controls required for this operation (orange box in Figure 1).
- *AES_AXIS_KEY* block encrypts the incoming plaintext data packets, taking in input the 128-bit plaintext data packets, the encryption key, the clock signal, and the synchronization signals used to manage the loading of the new encryption key into the first in first out (FIFO) registers (highlighted in red in Figure 1).
- *Pattern Verificator* for checking the compliance of the encrypted packets with the input plaintext data, notifying an error signal eventually if an error is detected (grey box in Figure 1).

The first step performed *AES_AXIS_KEY* block concerning the key expansion, aimed to obtain the 11 subkeys used in the 10 rounds constituting the AES algorithm. The developed implementation uses a 16 X 16 Sbox constituted by 32-bit elements, unlike the 8-bit of the classic implementation, thus speeding up the different operations involving it, but with higher resource utilization. A LUT-based implementation has been preferred over combinatorial-based solutions for Sbox, since the main goal of the proposed encryption/decryption implementation is the data throughput rather that the utilization of hardware resources, given the large memory capabilities offered by the FPGA platform; in fact, Sbox solutions based on LUT offer better performances from a processing time point of view at the expense of area occupation. The key expansion phase starts with the validation of the new key, generating the *expansion_key_start* signal, once the new key is rightly loaded from the registers; the whole key expansion operation lasts 174.55 ns to obtain the 44 words from the encryption key. The pseudo-code related to the key_expansion step is the following:

- for round_counter<11
- if round count=0 then out_valid=0 end
- current_key[0-31]=precedent_key[0-31] xor Sbox_4(precedent_key[96-103] or Sbox_4(precedent_key[104-111] or Sbox_4(precedent_key[112-119] or Sbox_4(precedent_key[120-127] xor rcon(round_count) end
- current_key[32-63]= precedent_key[32-63] xor current_key[0-31] end
- current_key[64-95]= precedent_key[64-95] xor current_key[32-63] end
- current_key[96-127]= precedent_key[32-63] xor current_key[64-95], round_count++ end

– return current_key end
– round_counter=0;
– end for

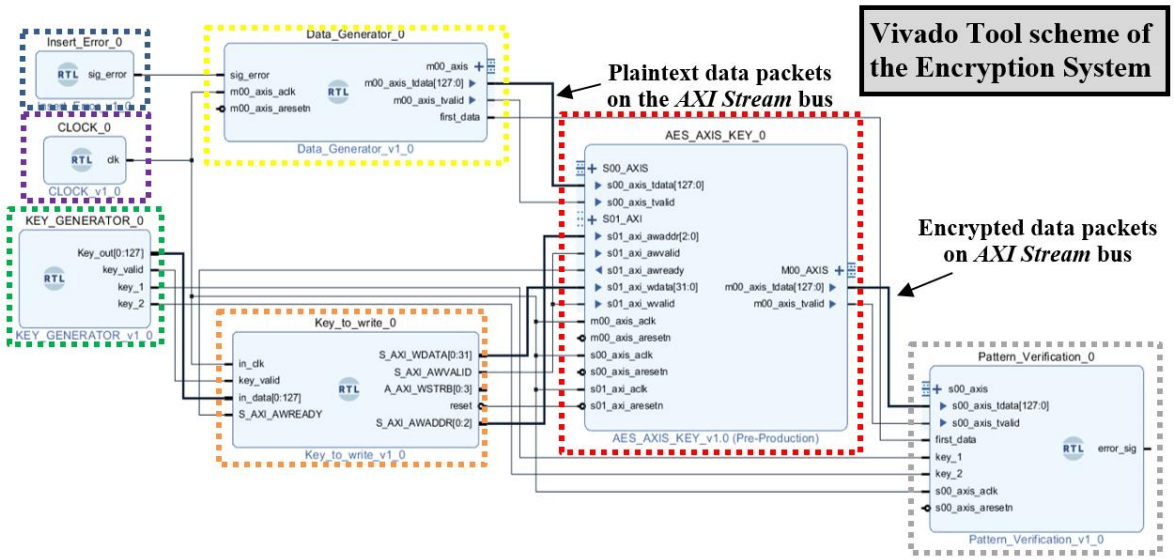


Figure 1. Block diagram of the proposed encryption system, obtained by the Xilinx *Vivado* design tool

This operation consists of 11 iterations, in each of which the key is validated, at first, and then the four words for each sub-key are obtained combining the previous sub-keys with the current key transformed by the Sbox; the *rcon* method is a round constant added to the result of the two sub-functions. Once the 11 sub-keys are obtained, the implemented algorithm generates the encrypted data by carrying out the ten iterations (the ten rounds) required by the AES 128 standard; in the first round (round_0), the plaintext data packets (*in_plain_data*) are combined through xor operation with the *cipher_key_table* which contains the key to be used not yet expanded; red box in Figure 2(a). After obtaining the intermediate data in the first round, the following nine rounds are carried out, in each of which the *SubstituteBytes*, *ShiftRow*, *MixColumns*, *AddRoundKey* operations are performed. The code section that recalls the nine rounds, developed in the *AES_AXIS_KEY* block, is shown in Figure 2(b). However, each round operates on the words resulted from the previous round. Thus, the new intermediate data is updated up to the last round (round 9), which provides the data for the round 10, where the last *Add Round Key* function is carried out; in Figure 3(a), the operations involved in each round of AES-128 algorithm are shown. After the ninth round, the *Add_round_key* function is applied to the last intermediate data, called *intermediate_data(9)* Figure 3(b).

The *Pattern_Verificator* block has been equipped with a second table, allowing us to use another key; when a different key is used for the encryption, this event is received by the two signals *key_1* and *key_2*, which indicates to the *Pattern_verificator* which table refers. These two signals are generated from the *Key_Generator* block and supplied directly to the *Pattern_verificator* block. This last is synchronized with the encryption block by a pulse on the *m00_axis_tvalid* pin provided at the encryption end, signalling the availability of the following data packet.

Also, two flag signals have been added, namely a synchronization flag and a signal to indicate the packet *Data_Generator* table provided to *AES_AXIS_KEY* block, enabling to the *Pattern_Verificator* the tracking of incoming packets, checking the corresponding ones inside its internal table. In addition, as mentioned above, after writing in the register and resetting the *key_valid* bit, the algorithm performs the key expansion operation, lasting 174.5 ns; during this phase, the change of the sub-keys causes the loss of three packets; these incorrect packets are reported externally through a pin called *invalid_packets* which goes high when wrong encrypted packets occur at the *AES_AXIS_KEY* block output and return low when the packets are encrypted correctly. An *error_sig* signal, provided by *Pattern_verificator*, indicates errors in encrypted data packets through a low level in correspondence to the wrong data packet.

The key substitution system is an essential feature for every communication system because a periodic key substitution is needed for ensuring the security of the transmitted data. Considering the *AES_AXIS_KEY* block, it accepts, by *Data_Generator*, a synchronization signal, called *s00_axis_tvalid*, to

signal the presence of the next data packet on the AXI bus; also, the encryption block accepts another synchronization signal from *Pattern_verificator*, called *m00_axis_tready*, for indicating when this last is available to receive a new data packet. Moreover, the *s00_axis_tready* signal has been configured, which indicates when the encryption block is ready to accept a new plaintext data packet; this signal is reset, only when the *m00_axis_tready* signal is reset. If the *Pattern_verificator* notifies its unavailability to load a new encrypted data packet, bringing so the *m00_axis_tready* signal to a low logic level, the encryption block, communicates to the *Data_Generator* its unavailability to accept new plaintext data packets, bringing so the *s00_axis_tready* signal to a low logical level. One of the primary contributions of the proposed framework is constituted by the control and synchronization mechanisms, above described, essential for the operation of the entire encryption/decryption system, guaranteeing its compatibility with the other functional blocks included in the communication system.

```
round_0 : process (in_clk, in_reset, in_plain_data_valid) is
begin
  if in_reset = '0' then
    intermediate_data_valid(0) <= '0';
    intermediate_data(0)       <= X"00000000000000000000000000000000";
  elsif in_clk'event and in_clk = '1' then
    if in_plain_data_valid = '1' then
      intermediate_data_valid(0)      <= '1';
      intermediate_data(0)(0  to 31)  <= in_plain_data(0  to 31)  xor cipher_key_table(0)(0 to 31);
      intermediate_data(0)(32 to 63)  <= in_plain_data(32 to 63)  xor cipher_key_table(0)(32 to 63);
      intermediate_data(0)(64 to 95)  <= in_plain_data(64 to 95)  xor cipher_key_table(0)(64 to 95);
      intermediate_data(0)(96 to 127) <= in_plain_data(96 to 127) xor cipher_key_table(0)(96 to 127);
    else
      intermediate_data_valid(0) <= '0';
    end if;
  end if;
end process round_0;
```

Add Round Key operation

(a)

```
round_from_1_to_9_generation : for I in 1 to 9 generate
  roundX : round_encoder
    port map (
      in_clk                    => in_clk,
      in_reset                  => in_reset,
      in_cipher_key             => cipher_key_table(I),
      in_intermediate_data      => intermediate_data(I-1),
      in_intermediate_data_valid => intermediate_data_valid(I-1),
      out_intermediate_data     => intermediate_data(I),
      out_intermediate_data_valid => intermediate_data_valid(I)
      );
end generate round_from_1_to_9_generation;
```

(b)

Figure 2. This figure are, (a) VHDL implementation of the first round (round_0) of AES algorithm, where the Add Round Key operation on the plaintext data packet is carried out; (b) code section associated with the nine intermediate rounds that carry out the operations required by the AES algorithm

```
  out_intermediate_data_valid <= '0';
  out_intermediate_data <= X"00000000000000000000000000000000";
elsif in_clk'event and in_clk = '1' then
  if  in_intermediate_data_valid  = '1' then
    out_intermediate_data_valid <= '1';
```

Intermediate Result of the Round

```
  out_intermediate_data(0  to 31) <= sbox_encoding_0(to_integer(unsigned(in_intermediate_data(0   to   7)))) xor
                                     sbox_encoding_1(to_integer(unsigned(in_intermediate_data(40  to  47)))) xor
                                     sbox_encoding_2(to_integer(unsigned(in_intermediate_data(80  to  87)))) xor
                                     sbox_encoding_3(to_integer(unsigned(in_intermediate_data(120 to 127)))) xor
                                     in_cipher_key(0  to 31);
  out_intermediate_data(32 to 63) <= sbox_encoding_0(to_integer(unsigned(in_intermediate_data(32  to  39)))) xor
                                     sbox_encoding_1(to_integer(unsigned(in_intermediate_data(72  to  79)))) xor
                                     sbox_encoding_2(to_integer(unsigned(in_intermediate_data(112 to 119)))) xor
                                     sbox_encoding_3(to_integer(unsigned(in_intermediate_data(24  to  31)))) xor
                                     in_cipher_key(32  to 63);
  out_intermediate_data(64 to 95) <= sbox_encoding_0(to_integer(unsigned(in_intermediate_data(64  to  71)))) xor
                                     sbox_encoding_1(to_integer(unsigned(in_intermediate_data(104 to 111)))) xor
                                     sbox_encoding_2(to_integer(unsigned(in_intermediate_data(16  to  23)))) xor
                                     sbox_encoding_3(to_integer(unsigned(in_intermediate_data(56  to  63)))) xor
                                     in_cipher_key(64  to 95);
  out_intermediate_data(96 to 127)<= sbox_encoding_0(to_integer(unsigned(in_intermediate_data(96  to 103)))) xor
                                     sbox_encoding_1(to_integer(unsigned(in_intermediate_data(8   to  15)))) xor
                                     sbox_encoding_2(to_integer(unsigned(in_intermediate_data(48  to  55)))) xor
                                     sbox_encoding_3(to_integer(unsigned(in_intermediate_data(88  to  95)))) xor
                                     in_cipher_key(96 to 127);
  else
    out_intermediate_data_valid <= '0';
  end if;
```

(a)

```
round_10 : process (in_clk, in_reset, intermediate_data_valid(9)) is
variable count : integer := 0;
begin
  if in_reset = '0' then
    out_cipher_data_valid <= '0';
    out_cipher_data       <= X"00000000000000000000000000000000";
  elsif in_clk'event and in_clk = '1' then
    if intermediate_data_valid(9) = '1' then
      out_cipher_data_valid    <= '1';
```

Encrypted packet

```
      out_cipher_data(0  to 31) <= (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(0   to   7))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(40  to  47))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(80  to  87))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(120 to 127))))
                                    cipher_key_table(10)(0  to 31);
      out_cipher_data(32 to 63) <= (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(32  to  39))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(72  to  79))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(112 to 119))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(24  to  31))))
                                    cipher_key_table(10)(32  to 63);
      out_cipher_data(64 to 95) <= (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(64  to  71))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(104 to 111))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(16  to  23))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(56  to  63))))
                                    cipher_key_table(10)(64  to 95);
      out_cipher_data(96 to 127)<= (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(96  to 103))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(8   to  15))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(48  to  55))))
                                    (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(88  to  95))))
                                    cipher_key_table(10)(96 to 127);
    else
      out_cipher_data_valid    <= '0';
  end if;
```

(b)

Figure 3. This figure are, (a) Operations involved in the first nine rounds of AES-128 algorithm, where the intermediate result of the round (*out_intermediate_data*) is stored for the next one; (b) generation of the encrypted data packet (*out_cipher_data*) from intermediate data produced from round 9 (*intermediate_data(9)*, blue box)

By storing the intermediate results obtained from each round (i.e. $intermediate\_data(i)$, $i = 1, \ldots .9$), a pipelined strategy has been implemented, performing the ten rounds on consecutive data packets, at the same time, thus enabling the elaboration of a new packet only when the processing on the previous packets is concluded. In this way, parallel elaboration on successive packets is obtained, thus improving the usage of hardware resources, and enabling higher data throughput. As aforementioned, the proposed AES implementation is featured by a round's processing time equal to only a clock cycle, providing an encrypted data packet every clock period. Moreover, the implementation of the AES-128 decryption algorithm has been carried out, similarly to the encryption system, parallelizing many logical operations on each clock period; specifically, a 16X16 *State* matrix was employed, also in this case, containing 32-bits elements and not of 8 bits as in the case of the standard implementation. The proposed implementation starts with key expansion, carried out with the same Sbox employed for the encryption process. The description process consists of ten rounds, involving the correspondent inverse operations to encryption, viz *InvShiftRows*, *InvSubBytes*, and *InvMixColumns*. The inverse functions are all obtained using matrix implementations represented by four 16 X 16 matrices with 32-bit elements (named *sbox_decoding_0*, *sbox_decoding_1*, *sbox_decoding_2*, and *sbox_decoding_3*), combined by *xor* operation to derive the intermediate data of the different rounds. This solution allows to reduce the time duration of the decryption operation to just ten clock cycles; nevertheless, this implementation requires more FPGA hardware resources. In order to test the decryption implementation, several VHDL blocks have been employed, with functionalities similar to those used for the encryption system.

3. RESULTS AND DISCUSSION

3.1. Behavioral simulations of the cascade system including the encryption and decryption sections

Although the Xilinx ZCU102 platform is featured by 350 MHz maximum operating frequency, post-implementation simulations produced a negative worst negative slack (WNS) for clock frequencies higher than 220 MHz, indicating clock signal propagation issues. To overcome this problem, the clock frequency has been reduced to 220 MHz and employing the *Explore* strategy offered by the Vivado tool, thus resulting in a WNS value of 0.005 ns, related to the encryption task, and 0.008 ns for the decryption one. The behavioural simulations on the system composed of the encryption and decryption sections connected in cascade have been performed, using a 220 MHz clock frequency as shown in Figures 4 and 5. In Figure 4, the waveforms related to the encryption/decryption process are shown, obtained by feeding the encryption system with plaintext packets every 40.86 ns, corresponding to a date-rate of about 3 Gbit/s ($\frac{128bit}{40.86\ ns} = 3.132\ Gbit/s$). As evident, the encrypted packets are processed by the encryption block output after ten clock periods (i.e. $10 \times 4.54\ ns = 45.4\ ns$), and loaded by the decryption section on the following clock rising edge; this last is decrypted and provided in output to the decryption system after only ten clock cycle. Hence, the whole encryption/decryption process lasts only 20 clock periods equal to 90.8 ns (for 220 MHz clock frequency). As evident, the control and synchronizing signals have been implemented for ensuring the interoperability between the developed encryption/decryption system and the different components integrated into the communication system.

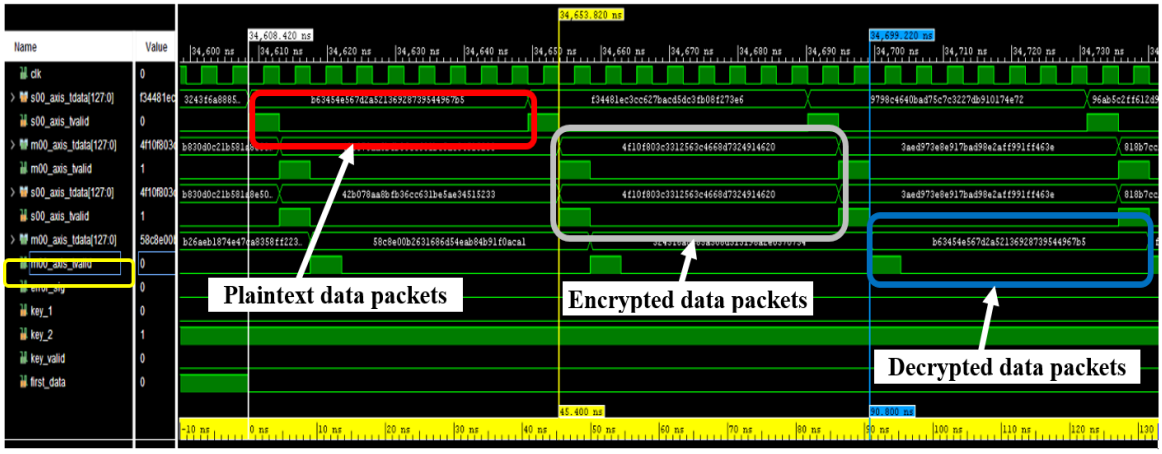


Figure 4. Temporal trends of the waveforms related to the encryption/decryption providing the plaintext data packets every 40.86 ns (data-rate 3.123 Gbit/s)

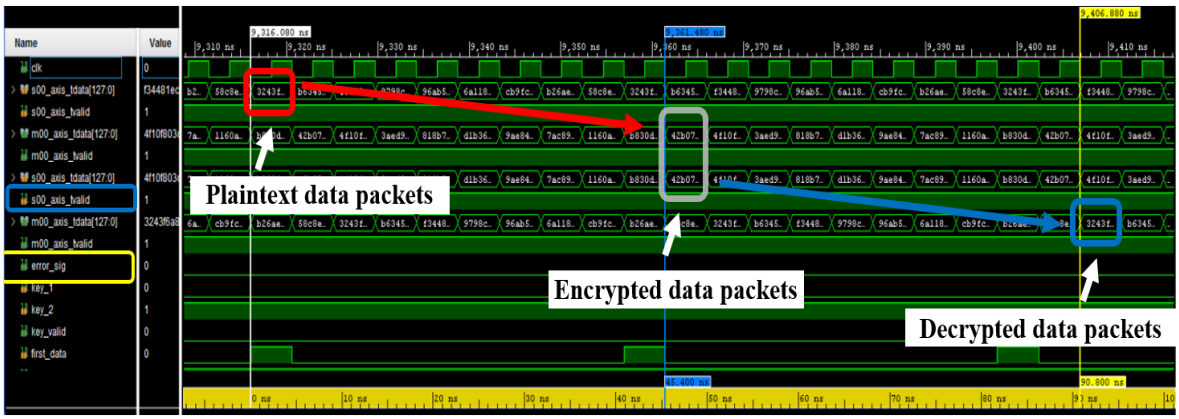


Figure 5. Temporal trends of the waveforms related to the encryption/decryption providing the plaintext data packets on every rising edge of the 220 MHz clock signal (every 4.54 ns)

Afterwards, the behavioural simulation is carried out, providing plaintext data packets to the input of the encryption block, on the rising edges of the clock signal. The time interval required for the encryption and decryption remains 90.8 ns. The combined system can elaborate and provide encrypted data on the rising edges of the synchronization signal, obtaining 28.16 Gbit/s data-rate ($220\ MHz\ \times 128\ bit = 28.16\ Gbit/s$); this is indicated by the status of *s00_axis_tvalid* signal, generated by the *AES_AXIS_KEY* block, which remains to a high level for the whole operating time of the system (blue box in Figure 5), index of the continuous availability of the encryption block to receive a new plaintext data packet. Also, the *error_sig*, provided by *Pattern Verificator*, indicates the correct operation of the designed encryption/decryption section, since it remains at a low level for all the operation time, indicating that the decrypted data packets are identical to the corresponding ones provided to the encryptor input (yellow box in Figures 4 and 5).

3.2. Post-synthesis and post-implementation simulations of the encryption and decryption systems

In this section, the real resource utilization of the FPGA-ZCU102 device related to the developed AES-128 implementation is derived by post-synthesis and post-implementation simulations. Therefore, the post-synthesis simulations are carried out on both encryption and decryption sections providing data packets on each clock period and with a 220 MHz clock frequency; the simulation results of the FPGA area utilization are summarized in Table 1. Subsequently, the simulation was repeated providing, every 40.86 ns, the data packets to the encryption/decryption block input, obtaining the same resource utilization. Besides, the post-synthesis simulation was performed on the encryption section, after the removal of the blocks added to test the developed implementation, leaving just the blocks required by the encryption process. In this case, the resources needed for the system synthesis are 4.76% of lookup table (LUT) and 0.71% of flip flop (FF) respect to the overall resources of Zynq Ultrascale+ XCZU9EG-2FFVB1156E MPSoC, corresponding to 1631 configurable logic blocks (CLBs); therefore, a reduction of 0.72 % has been obtained for LUT utilization and 0.07% for FF compared to the complete scheme. The post-synthesis simulations on the decryption system were carried out, providing the encrypted data packets both on the clock rising edges and every 40.86 ns, obtaining the same resource utilization as shown in Table 1.

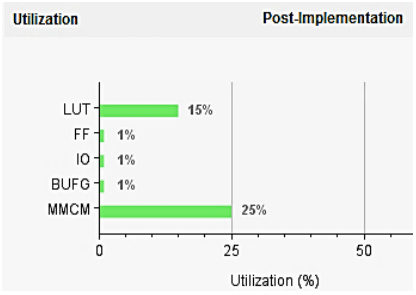
Similarly, the simulation has been repeated, after the removal of all the blocks not involved in the decryption; in this condition, 10.11% of LUTs, 0.71% of FF, and 0.25% of global buffer (BUFG) have been used, corresponding to 3464 CLBs, thus obtaining a reduction of 0.53% for LUTs and 0.08% for FF, compared the complete decryption system. It is evident that the decipherer requires more FPGA resources than the cypher, attributable to the four matrix implementations of *Inverse SubBytes*, *Inverse ShiftRows*, and *Inverse MixColumns* functions. Anyway, this latter corresponds most of the used hardware resources, due to the inverse matrix elements and their related implementation, based on LUT [45].

Besides, the post-implementation simulations were carried out for the encryption, decryption systems, and the one consisting of both blocks' cascade to check that the parameters resulting from the simulation have acceptable values to ensure the regular operation of the algorithm. The simulations demonstrated that for a clock frequency higher than 220 MHz, a positive worst negative slack (WNS) value could not be obtained, thus indicating clock routing issues. Specifically, for 220 MHz operative frequency, and setting the *Explorer* implementation strategy, the resulting WNS parameter value was equal to 0.005 ns and 0.008 ns relatively to the encryption and decryption sections, respectively. Also, the encryption and

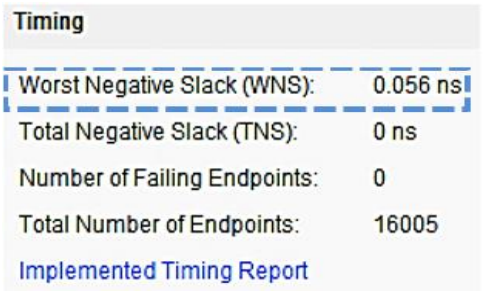
decryption sections arranged into a cascade configuration was tested by the post-implementation simulation. In particular, the occupation of the FPGA's resources turned out to be 15% LUTs 1% FFs 1% I/O ports 1% BUFG besides a further 25% area utilization was obtained ascribable to the *IP Clocking Wizard* section for generating the main clock as shown in Figure 6(a); also, the WSN value of the combined system was equal to 0.056 ns (dashed blue box in Figure 6(b)). The total-on-chip power, defined as the sum of device and design power consumptions, for the encryption and decryption system arranged into a cascade configuration, is equal to 1.768 W (red dashed box in Figure 6(c)), with 26.7 °C junction temperature, ensuring 73.3 °C of thermal margin, providing the data packets at each clock cycle.

Table 1. Summarizing table with reported utilization of the FPGA resources corresponding to the encryption and decryption sections produced by the post-synthesis simulations

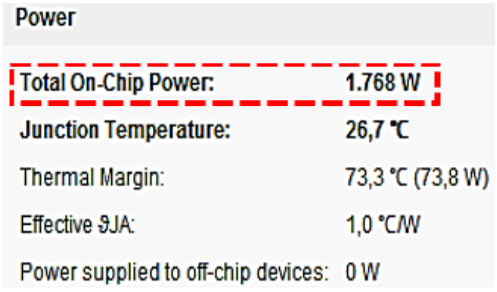
Simulation	Resource	Estimation	Available	Utilization [%]
Encryption system	LUT	15029	274080	5.48
	FF	4296	548160	0.78
	CLB	1879	600000	0.31
Encryption block	LUT	13043	274080	4.76
	FF	3877	548160	0.71
	CLB	1631	600000	0.27
Decryption system	LUT	29156	274080	10.64
	FF	4339	548160	0.79
	BUFG	1	404	0.25
	CLB	3642	600000	0.61
Decryption block	LUT	27713	274080	10.11
	FF	3912	548160	0.71
	BUFG	1	404	0.25
	CLB	3464	600000	0.58



(a)



(b)



(c)

Figure 6. Pictures captured by the *Project Manager*, after the *post-implementation* simulation, employing a 220 MHz clock frequency and *Explore* strategy for the implementation: (a) resource utilization section, (b) timing section, and (c) power section

By comparing our solution with another pipelined AES-128 implementation, reported in [46], on a comparable FPGA platform, the first shows a higher efficiency (8.63 Mbps/slices), compared to the latter (2.99 Mbps/slices), indicating better exploitation of hardware resources to obtain a given data throughput.

Besides, considering the high-throughput AES implementation reported in [47], our encryption system gets a slightly lower maximum data throughput (-5.3%), but also uses fewer FPGA's resources (namely, -39.7%), thus reaching a higher value of hardware resource utilization efficiency (+56.9%). By comparing our encryption/decryption solution with the implementation reported in [48], our system shows a considerably higher efficiency (+ 92.9%).

Afterwards, the bitstream file of the designed system, consisting of the encryption and decryption sections arranged into a cascade configuration, was generated and, then, loaded on the FPGA-ZCU102 board. In this way, the encrypted data packets, provided in output by the encryption block, are immediately loaded by the decryption block, which processes them in only ten clock periods; the whole encryption/decryption operation lasts only 20 clock periods. Also, the IP integrated logical (IL) analyzer has been added to the *Block Design*, to monitor the interest signals. The system was successfully tested for both the aforementioned operative conditions, namely furnishing the plaintext data packets every 40.86 ns (i.e. 3.13 Gbit/s) and every clock cycle (i.e. 28.16 Gbit/s).

4. CONCLUSION

The proposed research work presents a high-speed and lightweight implementation of AES-128 cypher for 5G communication systems, on a Xilinx ZCU102 FPGA platform. A pipelined framework has been employed, both for the encryption and decryption tasks, so enabling the contemporary elaboration of several data packets in the same clock cycle. A maximum working frequency of 220 MHz was obtained to ensure a positive WNS value in the post-implementation simulations, thus reaching 28.16 Gbit/s maximum data rate (i.e. $220\,MHz \times 128\,bit$); the encryption and decryption times last both just ten clock periods. Some control and synchronization signals have been implemented to ensure the interoperability of the proposed encryption/decryption section with the other ones included in the communication system. The hardware resources used by the encryption system was only 1631 CLBs, as well as the decryption one employs 3464 CLBs, mainly due to the LUT-based *Inverse MixColums* operation. The encryption system shows a higher efficiency (8.63 Mbps/slices) compared to other similar implementations present in the scientific literature.

REFERENCES

[1] C. Paar and J. Pelzl, "The Advanced Encryption Standard (AES)," in *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer, Berlin, Heidelberg, 2010, pp. 87-121, doi: 10.1007/978-3-642-04101-3_4.

[2] L. Li and S. Li, "High throughput AES encryption/decryption with efficient reordering and merging techniques," *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, Belgium, 2017, pp. 1-4, doi: 10.23919/FPL.2017.8056803.

[3] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Efficient uses of FPGAs for implementations of DES and its experimental linear cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 473-482, Apr. 2003, doi: 10.1109/TC.2003.1190588.

[4] C. A. Henk and T. Van., "RSA Based Systems," in *Fundamentals of Cryptology: A Professional Reference and Interactive Tutorial*, Springer, Boston, US, 2000, vol. 528, pp. 147-211, doi: 10.1007/0-306-47053-5_9.

[5] X. Ji, K. Huang, L. Jin, H. Tang, C. Liu, Z. Zhong *et al*, "Overview of 5G security technology," *Science China Information Sciences*, vol. 61, no. 8, pp. 1-26, Jul. 2018, doi: 10.1007/s11432-017-9426-4.

[6] A. G. Sulaiman and I. F. Alshaikhli, "Comparative study on 4G/LTE cryptographic algorithms based on different factors," *International Journal of Computer Science and Telecommunications*, vol. 5, no. 7, pp. 7-10, 2014.

[7] H. Huang, J. Xia, and S. Boumaiza, "Novel Parallel-Processing-Based Hardware Implementation of Baseband Digital Predistorters for Linearizing Wideband 5G Transmitters," *IEEE Transactions on Microwave Theory and Techniques*, pp. 1-11, May 2020, doi: 10.1109/TMTT.2020.2993236.

[8] C. Shen, D. Lee, C. Ku, M. Lin, K. Lu and S. Tan, "A Programmable and FPGA-accelerated GTP Offloading Engine for Mobile Edge Computing in 5G Networks," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Paris, France, 2019, pp. 1021-1022, doi: 10.1109/INFCOMW.2019.8845143.

[9] P. Visconti, B. Sbarro, and P. Primiceri, "A ST X-Nucleo-based telemetry unit for detection and WiFi transmission of competition car sensors data: firmware development, sensors testing and real-time data analysis," *International Journal of Smart Sensing and Intelligent System*, vol. 10, no. 4, pp. 793-828, 2017, doi: 10.21307/ijssis-2018-019.

[10] P. Visconti, B. Sbarro, P. Primiceri, R. de Fazio, and A. L. Ekuakille, "Design and Testing of a Telemetry System Based on STM X-Nucleo Board for Detection and Wireless Transmission of Sensors Data Applied to a Single-Seat Formula SAE Car," *International Journal of Electronic and Telecommunication*, vol. 65, no. 4, pp. 671-678, Oct. 2019, doi: 10.24425/ijet.2019.130248.

[11] A. Gopalan, J. Ganesh, and M. Swathi, "FPGA-based Message Encryption and Decryption," in *Interantional Journal of Innovative Technology and Exploring Engineering*, May 2015, pp. 1225-1232.

[12] P. Visconti, P. Primiceri, and C. Orlando, "Solar Powered Wireless Monitoring System of Environmental Conditions for Early Flood Prediction or Optimized Irrigation in Agriculture," *ARPN Journal of Engineering and Applied Sciences*, vol. 11, no. 7, pp. 4623-4632, Nov. 2016.
[13] P. Visconti, P. Costantini, C. Orlando, A. Lay-Ekuakille, and G. Cavalera, "Software solution implemented on hardware system to manage and drive multiple bi-axial solar trackers by PC in photovoltaic solar plants," *Measurement*, vol. 76, pp. 80-92, Dec. 2015, doi: 10.1016/j.measurement.2015.08.024.
[14] I. Ahmed and F. I. Alam, "Integrity verification for an optimized cloud architecture," *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, vol. 18, no. 1, 2020, doi: 10.12928/telkomnika.v18i1.14105.
[15] S. D. Putra, M. Yudhiprawira, S. Sutikno, Y. Kurniawan, and A. S. Ahmad, "Power analysis attack against encryption devices: a comprehensive analysis of AES, DES, and BC3," *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, vol. 17, no. 3, pp. 1282-1289, Jun. 2019, doi: 10.12928/telkomnika.v17i3.9384.
[16] C. Del-Valle-Soto, R. Velázquez, L. J. Valdivia, N. I. Giannoccaro, and P. Visconti, "An Energy Model Using Sleeping Algorithms for Wireless Sensor Networks under Proactive and Reactive Protocols: A Performance Evaluation," *Energies*, vol. 13, no. 11, pp. 1-31, Jan. 2020, doi: 10.3390/en13113024.
[17] P. Visconti, N. I. Giannoccaro, R. de Fazio, S. Strazzella, and D. Cafagna, "IoT-oriented software platform applied to sensors-based farming facility with smartphone farmer app," *Bulletin of Electronic Engineering and Informatics (BEEI)*, vol. 9, no. 3, pp. 1095-1105, Jun. 2020, doi: 10.11591/eei.v9i3.2177.
[18] S. D. Putra, A. S. Ahmad, S. Sutikno, Y. Kurniawan, and A. D. W. Sumari, "Revealing AES Encryption Device Key on 328P Microcontrollers with Differential Power Analysis," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 6, pp. 5144-5152, Dec. 2018, doi: 10.11591/ijece.v8i6.pp5144-5152.
[19] "5G security -trustworthy 5G system|Whitepaper," *Ericsson.com*, Mar. 26, 2018. https://www.ericsson.com/en/reports-and-papers/white-papers/5g-security---enabling-a-trustworthy-5g-system (accessed Sep. 10, 2020).
[20] A. Ahmad and S. Ismail, "User Selective Encryption Method for Securing MANETs," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 5, pp. 3103-3111, 2018, doi: 10.11591/ijece.v8i5.pp3103-3111.
[21] R. Bani-Hani, S. Harb, K. Mhaidat, and E. Taqieddin, "High-Throughput and Area-Efficient FPGA Implementations of Data Encryption Standard (DES)," *Circuits and Systems*, vol. 5, pp. 45-56, Mar. 2014, doi: 10.4236/cs.2014.53007.
[22] K. Long, V. C. M. Leung, Z. Haijun, Z. Feng, Y. Li, and Z. Zhang, Eds., *5G for Future Wireless Networks*, 1st ed. Beijing, China: Springer International Publishing, 2017.
[23] M. N. Hieu, D. H. Ngoc, C. H. Ngoc, T. D. Phuong, and M. T. Cong, "New primitives of controlled elements F2/4 for block ciphers," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 5, pp. 5470-5478, Oct. 2020, doi: 10.11591/ijece.v10i5.pp5470-5478.
[24] J. Ordoñez and G. Sahonero-Alvarez, "A 3D Convolution Accelerator Implemented on FPGA Using SDSoC," in *2019 International Engineering, Sciences and Technology Conference*, Oct. 2019, pp. 677-681, 2019, doi: 10.1109/IESTEC46403.2019.00126.
[25] M. Véstias, R. Duarte, J. Sousa, and H. Neto, "Fast Convolutional Neural Networks in Low Density FPGAs Using Zero-Skipping and Weight Pruning," *Electronics*, vol. 8, pp. 1-24, Nov. 2019, doi: 10.3390/electronics8111321.
[26] A. O. Mulani and P. B. Mane, "Watermarking and Cryptography Based Image Authentication on Reconfigurable Platform," vol. 6, no. 2, pp. 181-187, Jun. 2017, doi: 10.11591/EEI.V6I2.651.
[27] M. Turcanìk, "FPGA - Disruptive Technologies for Military Applications," in 2012 *International conference of scientific paper*, pp. 1-6, 2012.
[28] S. Putra, H. S. Sheshadri, and V. Lokesha, "A Naïve Visual Cryptographic Algorithm for the Transfer of a Compressed Medical Images," *Int. J. Recent Contributions Eng. Sci. IT*, 2015, doi: 10.3991/ijes.v3i4.5190.
[29] M. E. Hameed, M. M. Ibrahim, N. A. Manap, and M. L. Attiah, "Comparative study of several operation modes of AES algorithm for encryption ECG biomedical signal," *International Journal of Electric and Computer Engineering*, vol. 9, no. 6, pp. 4850-4859, Dec. 2019, doi: 10.11591/ijece.v9i6.pp4850-4859.
[30] S. Shivaputra, H. S. Sheshadri, and V. Lokesha, "A Naïve Visual Cryptographic Algorithm for the Transfer of Compressed Medical Images," *Bulletin of Electrical Engineering and Informatics*, vol. 5, no. 3, pp. 347-365 , Sep. 2016, doi: 10.11591/eei.v5i3.544.
[31] R. Srividya and B. Ramesh, "Implementation of AES using biometric," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 5, pp. 4266-4276, Oct. 2019, doi: 10.11591/ijece.v9i5.pp4266-4276.
[32] M. Hameed, M. M. Ibrahim, N. A. Manap, and A. A. Mohammed, "An enhanced lossless compression with cryptography hybrid mechanism for ECG biomedical signal monitoring," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 3, pp. 3235-3243, Jun. 2020, doi: 10.11591/ijece.v10i3.pp3235-3243.
[33] M. Gabrick, R. Nicholson, F. Winters, B. Young, and J. Patton, "FPGA Considerations for Automotive Applications," SAE International, Warrendale, PA, SAE Technical Paper 2006-01-0368, Apr. 2006, doi: 10.4271/2006-01-0368.
[34] X. Di, H.-G. Yang, Y. Jia, Z. Huang, and N. Mao, "Exploring Efficient Acceleration Architecture for Winograd-Transformed Transposed Convolution of GANs on FPGAs," *Electronics*, vol. 9, pp. 1-21, Feb. 2020, doi: 10.3390/electronics9020286.

[35] G. Renuka, V. U. Shree, and P. C. S. Reddy, "Comparison of AES and DES Algorithms Implemented on Virtex-6 FPGA and Microblaze Soft Core Processor," *International Journal of Electric and Computer Engineering (IJECE)*, vol. 8, no. 5, pp. 37-43, Oct. 2018, doi: 10.11591/ijece.v8i5.pp3544-3549.
[36] G. C. Cardarilli *et al.*, "Efficient FPGA implementation of high speed digital delay for wideband beamforming using parallel architectures," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 8, no. 2, pp. 422-427, Jun. 2019, doi: 10.11591/eei.v8i2.1483.
[37] A. T. Hashim, A. M. Hasan, and H. M. Abbas, "Design and implementation of proposed 320 bit RC6-cascaded encryption/decryption cores on altera FPGA," *International Journal of Electrical Computer Engineering (IJECE)*, vol. 10, no. 6, pp. 6370-6379, Dec. 2020, doi: 10.11591/ijece.v10i6.pp6370-6379.
[38] Y. Guo, H. Zheng, J. Wang, S. Xiao, G. Li, and Z. Yu, "A Low-Cost and High-Throughput Virtual-Channel Router with Arbitration Optimization," in *2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, Chengdu, China, Nov. 2019, pp. 75-76, doi: 10.1109/ICTA48799.2019.9012817.
[39] F. Noorbasha *et al.*, "FPGA design and implementation of modified AES based encryption and decryption algorithm," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, pp. 132-136, Apr. 2019.
[40] S. M. Soliman, B. Magdy, and M. A. Abd El Ghany, "Efficient implementation of the AES algorithm for security applications," in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, Seattle, WA, USA, Sep. 2016, pp. 206-210, doi: 10.1109/SOCC.2016.7905466.
[41] C.-P. Fan and J.-K. Hwang, "Implementations of high throughput sequential and fully pipelined AES processors on FPGA," in *2007 International Symposium on Intelligent Signal Processing and Communication Systems*, Xiamen, China, Nov. 2007, pp. 353-356, doi: 10.1109/ISPACS.2007.4445896.
[42] H. Zodpe and A. Sapkal, "An efficient AES implementation using FPGA with enhanced security features," *Journal of King Saud University - Engineering Sciences*, vol. 32, no. 2, pp. 115-122, Feb. 2020, doi: 10.1016/j.jksues.2018.07.002.
[43] O. A. Dawood, O. I. Hammadi, K. Shaker, and M. Khalaf, "Multi-dimensional cubic symmetric block cipher algorithm for encrypting big data," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 9, no. 6, pp. 2569-2577, Dec. 2020, doi: 10.11591/eei.v9i6.2475.
[44] Xilinx, "Zynq UltraScale+ MPSoC ZCU102, ZCU102 Evaluation Board User Guide," 2019. https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf (accessed Jun. 11, 2020).
[45] S. Murugeswari, P. Sridevi, D. Vanaja, and G. Vanaja, "Area optimization for reducing circuit complexity in masked AES based on FPGA," *International Journal of Innovative and Emerging Technologies*, vol. 1, no. 1, pp. 1-4, May 2015.
[46] L. Daoud, F. Hussein, and N. Rafla, "Optimization of Advanced Encryption Standard (AES) Using Vivado High Level Synthesis (HLS)," in *CATA*, 2019, vol. 58, pp. 36-44, doi: 10.29007/X3TX.
[47] D. Kotturi, Seong-Moo Yoo, and J. Blizzard, "AES crypto chip utilizing high-speed parallel pipelined architecture," in *2005 IEEE International Symposium on Circuits and Systems*, vol. 5, May 2005, pp. 4653-4656, doi: 10.1109/ISCAS.2005.1465670.
[48] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest," in *Cryptographic Hardware and Embedded Systems - CHES 2005*, Berlin, Heidelberg, 2005, pp. 427-440, doi: 10.1007/11545262_31.