electronics

MDPI

Article

10 Clock-Periods Pipelined Implementation of AES-128 Encryption-Decryption Algorithm up to 28 Gbit/s Real Throughput by Xilinx Zynq UltraScale+ MPSoC ZCU102 Platform

Paolo Visconti [1,*], Stefano Capoccia [1], Eugenio Venere [2], Ramiro Velázquez [3] and Roberto de Fazio [1]

[1] Department of Innovation Engineering, University of Salento, 73100 Lecce, Italy; stefano.capoccia@studenti.unisalento.it (S.C.); roberto.defazio@unisalento.it (R.d.F.)

[2] Medinok s.p.a, Via Palazziello, 80040 Napoli, Italy; eugenio.venere@medinok.com

[3] Facultad de Ingeniería, Universidad Panamericana, 20290 Aguascalientes, Mexico; rvelazquez@up.edu.mx

[*] Correspondence: paolo.visconti@unisalento.it; Tel.: +39-0832-29-7334

Received: 30 August 2020; Accepted: 10 October 2020; Published: 13 October 2020

Abstract: The security of communication and computer systems is an increasingly important issue, nowadays pervading all areas of human activity (e.g., credit cards, website encryption, medical data, etc.). Furthermore, the development of high-speed and light-weight implementations of the encryption algorithms is fundamental to improve and widespread their application in low-cost, low-power and portable systems. In this scientific article, a high-speed implementation of the AES-128 algorithm is reported, developed for a short-range and high-frequency communication system, called Wireless Connector; a Xilinx ZCU102 Field Programmable Gate Array (FPGA) platform represents the core of this communication system since manages all the base-band operations, including the encryption/decryption of the data packets. Specifically, a pipelined implementation of the Advanced Encryption Standard (AES) algorithm has been developed, allowing simultaneous processing of distinct rounds on multiple successive plaintext packets for each clock period and thus obtaining higher data throughput. The proposed encryption system supports 220 MHz maximum operating frequency, ensuring encryption and decryption times both equal to only 10 clock periods. Thanks to the pipelined approach and optimized solutions for the Substitute Bytes operation, the proposed implementation can process and provide the encrypted packets each clock period, thus obtaining a maximum data throughput higher than 28 Gbit/s. Also, the simulation results demonstrate that the proposed architecture is very efficient in using hardware resources, requiring only 1631 Configurable Logic Blocks (CLBs) for the encryption block and 3464 CLBs for the decryption one.

Keywords: field programming gate array; encryption; advanced encryption standard; wireless connector; 5G communication; experimental testing

1. Introduction

With the advances of information technology (IT) applications involving sensitive data, the network security is an ever-current topic for business activities; therefore, the development of robust, in computational terms light and efficient encryption algorithms is required for supporting the continuous increase of data volume and throughput in IoT applications, as well as video streaming, real-time communications, mobile transmissions and so forth [1–4]. The Advanced Encryption Standard (AES) remains the preferred encryption standard for governments, banks and high-security systems around the world. This last is the most widespread encryption algorithm, for instance, employed in Gigabit Ethernet, Worldwide Interoperability for Microwave Access (WiMAX) and 5G systems [5–7]. Also, this algorithm can be efficiently implemented both in hardware and software platforms; the software implementations of AES require lower resources but offer lower physical security. On the other hand, the growing demand for high-speed, high-volume secure data transmissions, ensuring at the same time the physical security, strongly requires the hardware implementation of the AES algorithm [8–10].

The Field Programmable Gate Array (FPGA) represents the ideal platform for implementing the ciphering algorithm to ensure the network security; however, the implementation of high throughput encryption/decryption algorithm is a challenge, given the limited resources of the considered platform; therefore, the development of resource-efficient AES encryptor/decryptor on the FPGA platform is crucial, as described above. The (Inv)SubBytes (Sbox) phase significantly affects the performance of the whole encryption algorithm; therefore several approaches are developed for reducing the computational load and resource, such as Composite Field Arithmetic (CFA) and look-up table (LUT) methods [1].

The communication networks are usually arranged into four sections, namely radio access network (RAN), core network, transport network and interconnection network; concerning the network security, the supported network planes (signal, user and management ones) are exposed to different threat typologies. Specifically, the network threats are classified in passive network threats and active network ones; the first ones are aimed to intercept the traffic sent on the network, whereas the second type includes all the attacks in which the execution of commands is involved to disturb the communication.

The introduction of the 5G technology signs the beginning of new concepts also in terms of network security, such as the International Mobile Subscriber Identity (IMSI) encryption while it is transmitted through the network, protecting it from external attacks. Furthermore, the 5G enables the extension of security mechanisms employed in the cellular networks to other wireless networks, ensuring the protection of the smart devices and stored data, a concept called "home network control." In this contest, the standardization is fundamental in order to guarantee full compatibility with the different networks present around the globe.

As well known, the most widespread encryption algorithms for 5G and in general, for high throughput applications, are the AES, SNOW 3G and ZUC ones [11,12]; the AES algorithm is very robust against multiple attacks, such as the brute-force, linear and differential ones. However, the AES algorithm requires hardware acceleration methods to reduce the execution time in the downlink, keeping low the area and energy requirements for mobile devices. On the other hand, the SNOW 3G, updated into a faster version called SNOW-V, complies with all the requirements of 5G, ensuring security, performance and flexibility. It is a word-based synchronous stream cipher, developed by Thomas Johnasson and Patrik Ekdahl at Lund University and works on 32-bit words and 128-bit (or 256-bit) keys; the algorithm is based on the combination of Finite State Machine (FSM) and Linear Feedback Shift Register (LFSR), where this last determines the next state of the FSM. The ZUC is a stream cipher designed by the Data Assurance and Communication Security Research Center (DACAS) of the Chinese Academy of Sciences. The cipher forms the core of the 3GPP mobile standards 128-EEA3 (for encryption) and 128-EIA3 (for message integrity). It was proposed for inclusion in the Long Term Evolution (LTE) or the 4th generation of cellular wireless standards (4G). ZUC is a word-oriented stream cipher, taking a 128-bit initial key and a 128-bit initial vector (IV) as inputs and providing as output a keystream of 32-bit words (called keywords). This keystream can be used for both encryption/decryption phases. The ZUC algorithm has better resistance compared to the SNOW 3G one against specific attacks, such as guess and deterministic ones, also showing a good flexibility in balancing high throughput and consumed area [13].

This research work proposes a high-throughput implementation of the AES-128 algorithm properly designed for a custom, very short-range and high-frequency communication system, called Wireless Connector; in particular, this system was thought for high-throughput data transmission on the frequency range around the 60 GHz between two mobile stations located at short range distance (1–10 m). A demonstrator of the Wireless Connector has been realized employing an FPGA platform for performing the main tasks related to the communication, the base-band elaborations, coding/decoding and the encryption/decryption phases. For these aims, the potentialities of the FPGA, in terms of high-performance, low cost and development time, as well as re-configurability, have been exploited [14,15]. Due to their great flexibility and wide applicability, FPGA platforms are used on a wide range of application fields, such as video and imaging processing, military applications, automotive, electronics for specialized processing and more. They are particularly useful for prototyping Application-Specific Integrated Circuits (ASICs) or processors.

As known, the AES-128 stands out for robustness, efficient occupation of the logical cells for the execution of the various operations and for being relatively computationally light, making it the optimal choice to satisfy all requests of the project [16–18]. For instance, the time required to attack the AES-128 algorithm and then to recover the key, is extremely high ($\propto 2^{126}$ operations); for the US government, it is considered sufficient for documents classified as secret, whereas, for top-secret documents, the AES-192 or AES-256 algorithm is required.

The development of the AES-128 algorithm has been carried out employing the Zynq Ultrascale+MPSoC ZCU102 platform (manufactured by Xilinx, San Jose, CA, USA), based on Zynq Ultrascale+ XCZU9EG-2FFVB1156E MPSoC (Multiprocessor System-on-Chip), which combines a powerful Processing System (PS) and Programmable Logic (PL) Ultrascale architecture into a single device. The proposed implementation of the AES-128 employs multiple elaborations of the incoming data packets for complying with the 3 Gbit/s data rate constraint (with a maximum value of 28 Gbit/s) required by the Wireless Connector application, thus imposing an upper limit on the time interval between successive packets equal to 42.67 ns. The proposed AES-128 implementation employs a pipelined approach in the round-based elaboration of the AES algorithm, consisting of in "assembly-line"-type processing, in which a new plaintext data packet is acquired, as soon as the simultaneous elaboration of the ten rounds on the previous data packets is completed (i.e., the corresponding FPGA logic section is available to be used). In this way, the round's processing on successive plaintext packets is carried out simultaneously during each clock period, with better exploitation of the allocated resources and so reaching higher data throughput.

Also, the developed AES implementation employs a Sbox containing 256 elements of 32 bits each, instead of 8 bits of the standard implementation, thus obtaining the encrypted data packet in a shorter time interval (only 10 clock periods) but requesting a greater area occupation on the FPGA. As demonstrated below, the developed encryption system can support a maximum clock frequency of 220 MHz and is featured by an encryption time of only 10 clock periods; however, it is able to process and provide the encrypted packets each clock period (namely, $4.54\text{ ns} = \frac{1}{220\text{ MHz}}$), thus obtaining a maximum data throughput higher than 28 Gbit/s (i.e., $\frac{128\text{ bit}/\mathit{packet}}{4.54\text{ ns}} = 28.16\text{ Gbit/s}$).

Besides, a fast algorithm for the key expansion has been implemented, based on the combination, by GF operations, of the previous sub-key with the current sub-key modified by the Sbox; in this way, the key expansion operation is completed in only 174.55 ns, obtaining the 44 words from the main key. The results above described have been obtained also thanks to the latest-generation FPGA platform (Xilinx ZCU102 board) used to implement the encryption/decryption system, featured by high performances, large memory capabilities and a wide set of peripherals [19]. Furthermore, the developed encryption/decryption block implements all the control signals required to synchronize its operation with the data generator block and the modem, placed upstream and downstream from it, respectively. Besides, several blocks have been implemented to test the operation of the encryption/decryption block, by deterministically inserting an error inside a known-plaintext data packet and detecting the error in the encrypted/decrypted packet, notified by a proper error signal. Also, a proper mechanism to change the encryption key during the operation of the encryption system has been implemented, resulting in just three packets lost during the replacement process, as described in Section 3.

The following text is organized as follows: the Section 2 includes a literature analysis about high throughput implementation of encryption/decryption algorithms based on FPGA platforms;

furthermore, the demonstrator of the Wireless Connector, based on the Xilinx ZCU102 platform, is described, demonstrating its proper operation. In the Section 3, the VHDL (acronym of VHSIC-Hardware Description Language, VHSIC means Very High Speed Integrated Circuits) blocks to implement and test the encryption and decryption algorithms are carefully described; also, the results related to the performances of the proposed custom AES-128 encryption/decryption algorithm are presented, in terms of encryption/decryption time, resource utilization and complexity. In the Section 4, the discussion on the obtained results are reported, also solving with clock routing issues; furthermore, in this section, the tests of the combined encryption/decryption system, after the loading on the ZCU102 board, are presented. Finally, the comparison of proposed AES-128 implementation with other similar works reported in the scientific literature is reported.

2. Materials and Methods

2.1. Fundamentals of the AES-128 Encryption/Decryption Algorithm

The AES algorithm is a block cipher at the bit level, like the Data Encryption Standard (DES), where each block length is fixed to 128 bits, whereas the key length can be equal to 128, 192 or 256 bits [20]. Each 128-bit data block is partitioned into 16 bytes, mapped on a 4×4 array named state, and each byte of the state corresponds to an element of the Galois Field (GF) with 2^8 cardinality. Based on the key length, the algorithm includes n iterations, called rounds, where n is 10, 12 or 14 when the key length is 128, 192 or 256 bits, respectively. Each round of the encryption process, except for the last one, consists of four operations:

- Substitute Bytes
- Shift Rows
- Mix Columns
- Add Round Key

All the operations are carried out sequentially within each round, except for the initial Add Round Key; in the last round, the Mix Columns operation is not performed (Figure 1).

The Substitute Bytes step is a non-linear transformation, where each byte in the state array is replaced with the entry of a fixed 8-bit Substitution Box (Sbox) implemented as a lookup table with 2^8 words of 8 bits each, used to hide the relationship between the key and the cipher-text. The used Sbox is derived from the multiplicative inverse over $\mathrm{GF}\left(2^8\right)$, combined with an invertible geometric transformation, to avoid attacks based on simple algebraic properties, obtaining a 16×16 bytes table (Figure 2). The permutation is obtained addressing the Sbox locations based on the most significative nibble and the less significative one of the 8-bit input data.

The Shift Rows step operates on the rows of the state array, circularly shifting the bytes in each row by a given offset. The first row is left unchanged, whereas each byte of the second row is shifted one position to the left; likewise, the third and fourth rows are shifted respectively by two and three bytes to the left. The Mix Columns step is a linear transformation mixing the column of the state array; each column, treated as a polynomial over $\mathrm{GF}\left(2^8\right)$, is multiplied, modulo $z^4 + 1$, with a fixed polynomial (i.e., $c(z) = 03z^3 + 01z^2 + 01z + 02$). Both the Shift Rows and the Mix Columns operations are needed to hide the relationship between the cipher-text and the plain text.

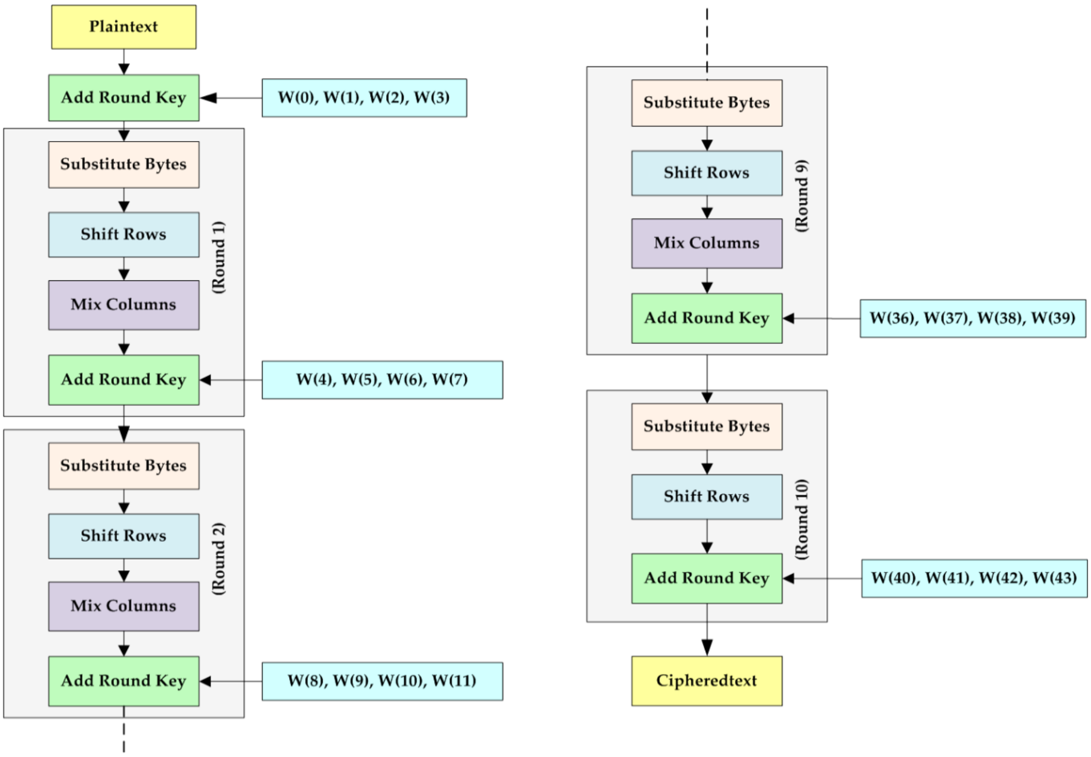
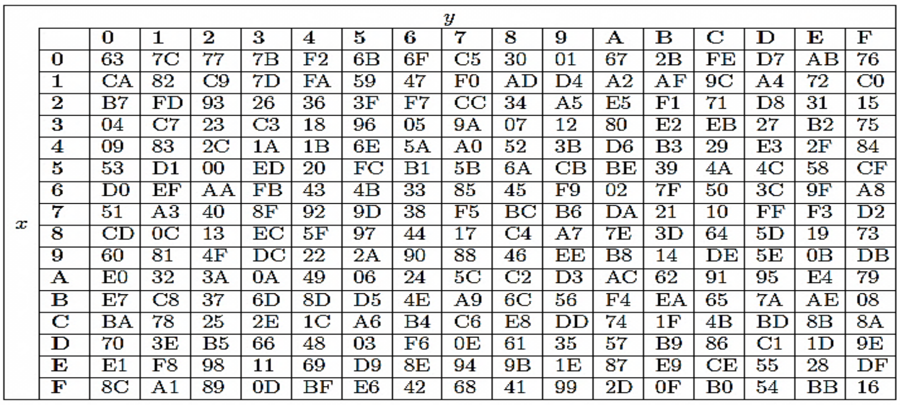


Figure 1. Schematic representation of the Advanced Encryption Standard (AES) encryption algorithm.



		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4B	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 2. Sbox involved in the Substitute Bytes transformation.

In the Add Round Key step, the key is combined with the state array to make the cipher safer; for each round, a subkey is derived from the expansion of the main key, obtaining, at the end of the encryption process, an expanded key of 176 bytes, arranged in a linear array of 44 words (using a key length of 128-bit). After the initialization of the word array ($W[i]$ for $0 \leq i \leq 43$), by inserting the key in the first four words, the other ones are obtained using the following relation:

$$W[i] = W[i-1] \oplus W[i-4]. \tag{1}$$

A particular exception is made for the words with index multiple of four, for which, non-linear relationships, different from bit-to-bit XOR are used:

$$Subword(Rotword(w[i-1])) \oplus Rcon[i/4], \tag{2}$$

where the Subword sub-function replaces each byte of the word, provided as argument of the function, using the Sbox, whereas the Rotword sub-function simply shifts one byte to left; furthermore, the function Rcon[i] is a round constant, represented by the word array $\left[x^{i-1}, \{00\}, \{00\}, \{00\}\right]$, where x^{i-1} is the $(i-1)$-th exponentiation operator in GF(2^8).

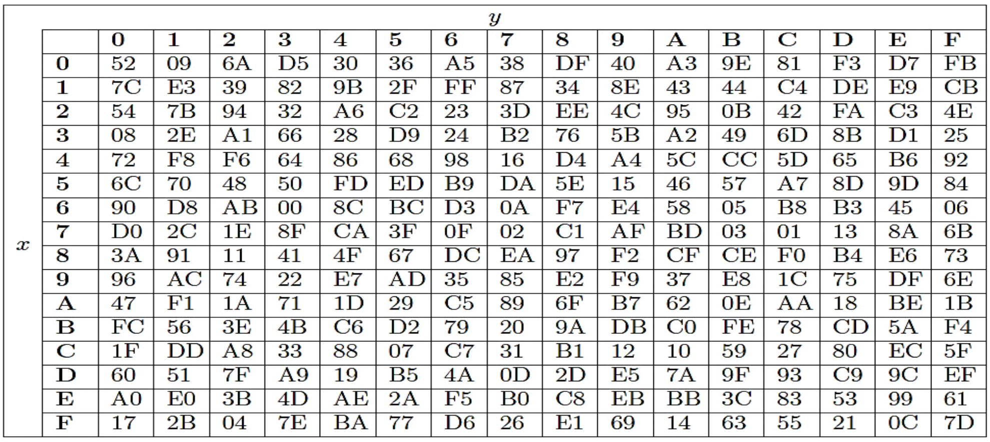
The encryption and decryption procedures employ two different algorithms; nevertheless, each operation in the encryption process corresponds to an inverse equivalent one in the decryption process. However, both are arranged in 10 rounds and both perform the Add Round Key step in the same way. Thus, each round of the decryption process consists of the following operations:

- Inverse Sub Bytes
- Inverse Shift Rows
- Inverse Mix Columns
- Add Round Key

A further difference between the encryption and decryption processes is the order of functions performed within a single round; in the decryption process, the first step is the Inverse Shift Rows, followed by Inverse Sub Bytes, Add Round Key and finally Inverse Mix Columns. In particular, the Inverse Shift Rows cyclically shift to the right by the same offset of the Shift Rows step but in the opposite direction.

To invert the Mix Column operation, the Inverse Mix Columns step employs the corresponding inverse matrix. The 4-byte columns of the state array are multiplied for the inverse 4 × 4 matrix featured by constant entries for producing the output bytes; all operations involved in the matrix multiplication are performed in GF$\left(2^8\right)$ or equivalently by multiplying each column, modulo $z^4 + 1$, with a fixed polynomial $b(z) = 0Bz^3 + 0Dz^2 + 09z + 0E$, where $b(z) = c(z)^{-1} mod\left(z^4 + 1\right)$ *and* $c(z)$ is the polynomial used in the *Mix Columns* step of the encryption.

The Inverse Substitute Bytes function is carried out similarly to the Substitute Bytes but using a different Sbox (Figure 3), obtained applying the inverse affine transformation to the Sbox followed by the multiplicative inverse in GF$\left(2^8\right)$.



									y								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
	0	52	09	6A	D5	30	36	A5	38	DF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
x	**8**	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Figure 3. Sbox involved in Inverse substitute bytes transformation.

Since the encryption and decryption operations are not the same, a significant disadvantage from the implementation point of view is obtained; however, there is an equivalent version of the decryption algorithm, which involves the inverse functions in the same order as the encryption algorithm. In particular, since the Inverse Shift Rows step changes the sequence of the bytes of the state array, leaving the content unchanged, whereas the Inverse SubBytes step changes the content but not the sequence of the bytes, their order has not relevance anymore, thus it can be exchanged. Moreover, the Add Round Key and Inverse Mix Columns transformations, considering the key as a sequence of words, both operate on the state array, column by column; therefore, the Inverse Mix Columns operation can be applied to the phase key $W[i]$, before adding it to the current state array, so obtaining the data packet decryption with the same sequence of operations of the encryption algorithm.

2.2. Implementation of Encrypting/Decrypting Algorithms With FPGA Platforms

In Reference [21], the authors proposed an inexpensive, low area and high throughput hardware implementation of the Advanced Encryption Standard algorithm (AES) for low-cost embedded applications, using a 128-bit key for both encryption and decryption, employing parallel operation in the folded architecture. The hardware selected for the implementation is the Virtex-6 XC6VLX75T FPGA device. In the folded architecture, the 128-bit blocks of input data are divided into four sub-blocks of 32-bit each and all the operations are performed sequentially. Due to the inefficiency of this method, along with the folded architecture, parallel computing is required to speed-up the algorithm execution. The experimental results reveal that the algorithm can achieve a 37.1 Gbit/s data throughput with a maximum clock frequency of 505.5 MHz.

C. Guzmán et al. proposed a hardware implementation of the AES 128-bit algorithm with a pipelined architecture, working on two non-feedback modes of operation, namely Encoded Code-Book (ECB) and Counter (CTR), using a Xilinx Virtex 5 FPGA platform [22]. They compared the two operation modalities in terms of resource utilization, throughput and robustness. The results revealed that the CTR mode is more convenient than ECB one in terms of level security and area efficiency. The proposed architecture reaches a clock frequency of 272.59 MHz corresponding to a throughput of 34.89 Gbit/s.

A triple key AES algorithm is presented in ref. [23]; such a framework requires 128 bits plaintext input and 3 keys for combining the ciphertext. These lasts are combined by a common xor function and, the resulting key is provided along with the plaintext to an "add round key" block, where they are combined by xor function; afterward, the obtained data block and the combined key are sent in input to a 128 AES encryption block for obtaining the cipher data (Figure 4). The proposed algorithm was optimized, thus obtaining 867.34 Mbit/s maximum throughput, with a resource utilization of 3402 Configurable Logic Blocks (CLBs), 27,787 LUTs and 385 Input/Output Blocks (IOBs). By comparing the proposed algorithm with other Xilinx devices, a 15% increase in throughput and a lower processing delay were obtained.

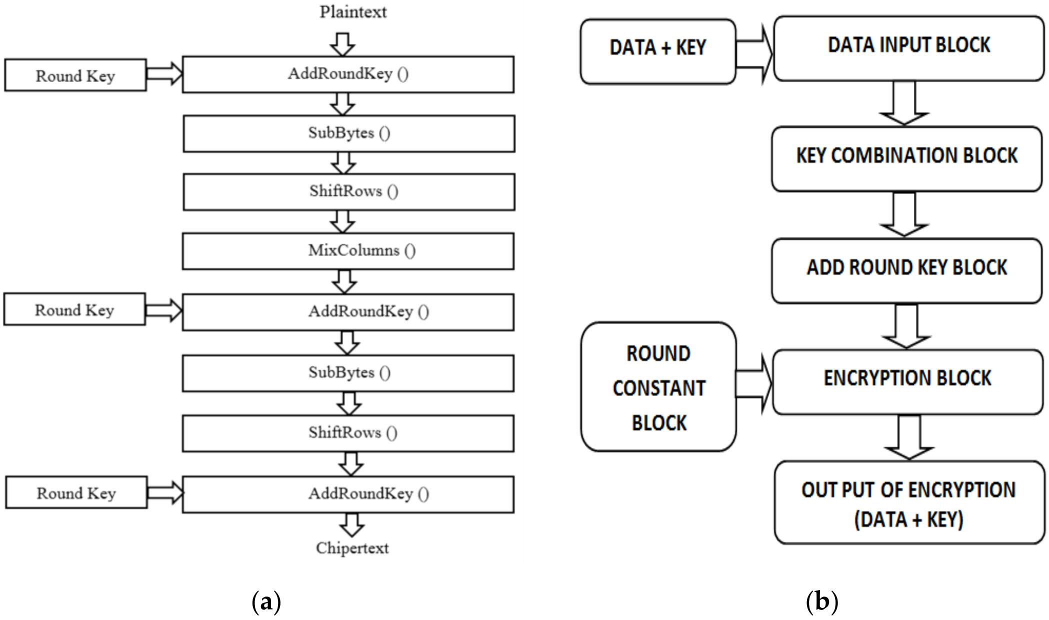


Figure 4. AES encryption round operations (**a**); process flow of the triple-keys AES algorithm proposed in ref. [23] (**b**).

In ref. [24], A. Gopalan et al. described the development and implementation of an AES algorithm on a FPGA platform (Xilinx XC6SLX16), comparing the designed infrastructure with a correspondent software implementation. The developed encryption block required 10 clock cycles for processing each data block, thus corresponding to 100 ns processing time since the clock frequency is equal to 100 MHz. Instead, the decryption block took 11 cycles (i.e., 110 ns), since an initial overhead is required. The two main figures of merit used to evaluate the algorithm performance are the throughput (in Equation (3)) and latency (in Equation (4)):

$$T = \frac{128 * f_{clk}}{block\_per\_cycle} \tag{3}$$

$$Lat = \frac{10 * stages\_per\_round}{f_{clk}}, \tag{4}$$

where *block_per_cycle* is one for a fully-unrolled architecture, whereas it becomes greater than one if the round output is re-used to elaborate the single input; *stages_per_round* is the number of clock cycles to process a single round.

C. P. Fan et al. described a high-speed 128-bit AES encryption module both in sequential and fully pipelined architectures, including a Content Addressable Memory (CAM)-based architecture used to realize pipelined high-speed SubBytes and InvSubBytes blocks, a hardware-sharing solution to carry out a high-speed MixColumns operation and a real-time key generation scheme to realize the AddRoundKey block [25]. The latter generates 128-bit keys for both encryption and decryption processes from the encryption key segmented into four 32-bit blocks and stored into different registers. The last register output (named d register) is dispatched to ROT (shift of bytes), S-box and RCON (XOR operation) blocks. The SubBytes and InvSubBytes operations were implemented by applying a CAM-based architecture, providing as output the data lines value obtained from several register arrays placed both between and inside the AES round computations, according to the matched address line data. Also, the MixColumns and the InvMixColumns operations were carried out by means of the row mapping permutations, based on two corresponding polynomials matrix. To reduce the resource utilization, the InvMixColumns polynomials matrix was decomposed into three different matrices for highlighting the hardware sharing of the two operations; in this way, a high-speed shared circuit for implementing the two transformations was derived. The AES module, implemented on the Xilinx XC2V3000-6 FPGA platform, reached in the sequential architecture a data throughput value up to 0.876 Gbit/s with a clock frequency of 75.3 MHz, both in the encryption and decryption phases. Instead, the proposed fully pipelined AES architecture obtained 28.4 Gbit/s throughput with an operating frequency of 222.2 MHz in the encryption phase.

In ref. [26], the authors proposed high-performance hardware implementation of the Data Encryption Standard (DES) encryption algorithm, with a 16-stage pipelined architecture, operating in CTR mode, on a Xilinx Virtex XCV1000-4 BG560 FPGA platform. In the proposed architecture, an initial delay of 16 clock cycles is required to instantiate the functional block, where are included the key expansion function, the Sbox function and the Pbox function; then, at each clock cycle, fixed-length clusters of data are loaded into this block along with different keys, so allowing the use of multiple keys, one for each of the 16 rounds of the DES algorithm. The major contribution was a parameterizable key scheduling method, where the sub-keys are pre-computed and distributed to the functional blocks of each round; furthermore, a skew core controls the availability overtime of the sub-keys to the different function blocks, delaying their generation by the needed time amount. The results showed that the proposed architecture achieves an encryption rate of 3.87 Gbit/s, guaranteeing a low area utilization with only 6446 CLB slices used.

P. Chodowiec et al. proposed a compact implementation of the 128-bit AES algorithm on the inexpensive Xilinx II XC2S30 FPGA, using a folded architecture and achieving good performance and low area utilization [27]. The folded architecture described is the same reported in ref. [21], nevertheless, the authors have introduced a new approach for implementing MixColumns and InvMixColumns functions using shared logic resources. This architecture requires only 222 CLB slices and 3 blocks of RAM, supporting a maximum throughput of 166 Mbit/s.

In ref. [28], the authors developed and evaluated hardware implementations, based on various FPGA devices, of the DES encryption algorithm, introducing several pipelined architectures that stand out for power consumption, resource utilization and throughput; the most significant ones are an 8-stages pipelined architecture and a 37-stages pipelined architecture. In the first one, two rounds at a time are collapsed into one stage and the output is saved into two intermediate registers of the next stage, up to a total of 8 stages; instead, in the second proposed solution, the authors developed a 37-stages pipelined DES architecture, previously reported in ref. [29] but optimized by reducing the utilization of resources by joining the logical operations by means of a processing block with 4 inputs and 1 output. The second architecture was improved by removing the redundant E (Expansion) and R (Reduction) boxes from the original design. With such modifications, the authors were able to increase the throughput by a 1.1 factor compared to the original design, reaching 40 Gbit/s using a Kintex7 platform. Regarding the proposed 8-stage pipelined implementation, a significant reduction of resources utilization (of a 0.75 factor) and power consumption (of a 0.65 factor) compared to a similar 16-stages pipelined design was demonstrated.

As above described, a LUT-based solution has been used to implement the Sbox in the proposed AES algorithm, which is not an optimal solution for area-limited hardware but offers better performances in terms of data throughput compared to other solutions, for instance, based on combinatorial logic, aimed to minimize the resource utilization, as demonstrated in Reference [30]. In this context, in Reference [31], the author proposed an overview of the different strategies to implement compact Sbox function, based on both polynomial and normal bases. Furthermore, they introduced a compact Sbox implementation based on a multi-level representation of GF operations, obtained properly selecting a particular basis (isomorphism) and making appropriate improvements to the circuital solution. The proposed solution has demonstrated improvements of 20% compared to the most compact Sbox implementation reported in Reference [32]. Besides, T. Good et al. proposed two new FPGA implementations of the AES algorithm [33]; the first one, implemented on Xilinx Spartan-III (XC3S2000) FPGA, relies on fully parallel loop unrolled architecture, reaching a 25 Gbit/s data throughput value. The latter, implemented on Spartan-II (XC2S15) FPGA, is based on state data and LUTs to carry out the AES operations, such as Substitute Bytes and Mix Columns, combined into a single matrix, called “T-box”; this implementation is featured by low area utilization, achieving 2.2 Mbit/s maximum throughput. The Sbox implementations proposed in these works can be applied to our solution to significantly reduce the used hardware resources but probably reducing the maximum throughput, representing the main prerogative of the Wireless Connector communication system.

2.3. Description of the “Wireless Connector” System’s Demonstrator and Relative Communication Tests

In this sub-section, the preliminary demonstrator of the Wireless Connector system is presented, which includes two PEM003 RF radio modules (manufactured by Pasternack, Irvine, CA, highlighted with red box), interfaced with the base-band hardware consisting of Zynq Ultrascale+ MPSoC ZCU102 platform (manufactured by Xilinx, San Jose, CA, USA, highlighted by the yellow box), an ADFMCDAQ2 acquisition board (manufactured by Analog Device, Norwood, MA, USA, highlighted by the purple box), four power splitter combiners (model ZFSCJ-2-4+, manufactured by Mini-Circuits, Brooklyn, NY, USA, highlighted by the green box), four pre-DAC anti-alias low-pass filters (model VLFX-300, manufactured by Mini-Circuits, highlighted by the orange box) and a personal computer for the system management (highlighted by the blue box).

The ADFMCDAQ2 acquisition board includes a dual-ADC (model AD9680, manufactured by Analog Device) featured by 14-bit resolution 1.0 Gsps sample rate and with JESD204B interface and also a 16-bit resolution quad-DAC (model AD9144, manufactured by Analog Device), featured by 2.8 Gsps sample rate and JESD204B interface; furthermore, a clock generator is placed onboard by employing a 14-outputs AD9523-1 low jitter IC (manufactured by Analog Device), along with components for the power management.

The Pasternack’s PEM003 development kit consists of a transmitter (Tx) and a receiver (Rx) module, operating at a frequency band around 60 GHz, supporting complex modulations through a pair of modulation signals I and Q. Each module is equipped with a USB interface, for setting the main parameters by connecting it to a PC but also ensuring its power supply. The baseband I and Q signals are applied to the input of the Tx module or available at the output of the Rx module, through the Micro Coaxial Connector (MCX) placed on the back of each board; the obtained signals are in the differential format (i.e., I+ and I−, Q+ and Q−). The 60 GHz section terminates with two Tx and Rx antennas connected to the UG-385/U flange which acts as an interface with the WR-15 waveguide. A reference design based on an embedded microprocessor system (uBLaze Xilinx) has been used to characterize the ADC/DAC devices. By using the internal logic resources of the FPGA device, the embedded MicroBlaze processor is generated by employing the Vivado/SDK design tool. The drivers for the management of the Ethernet protocol, a UART interface for the information exchange and system management and an external DDR memory for the management of user data are directly connected to the MicroBlaze processor.

The reduced performance of the acquisition board and the logical resources of the FPGA device allow the installation of the Quadrature Phase-Shift Keying (QPSK) modulator-demodulator with low-performance Forward Error Correction (FEC) modules (e.g. Reed-Solomon). In Figure 5, a functional scheme of the system described above is provided, whereas in Figure 6 the realized experimental setup to perform the 60 GHz communication tests is shown, supporting a data-rate up to 3 Gbit/s, constraint previously defined for the whole 5G communication system.

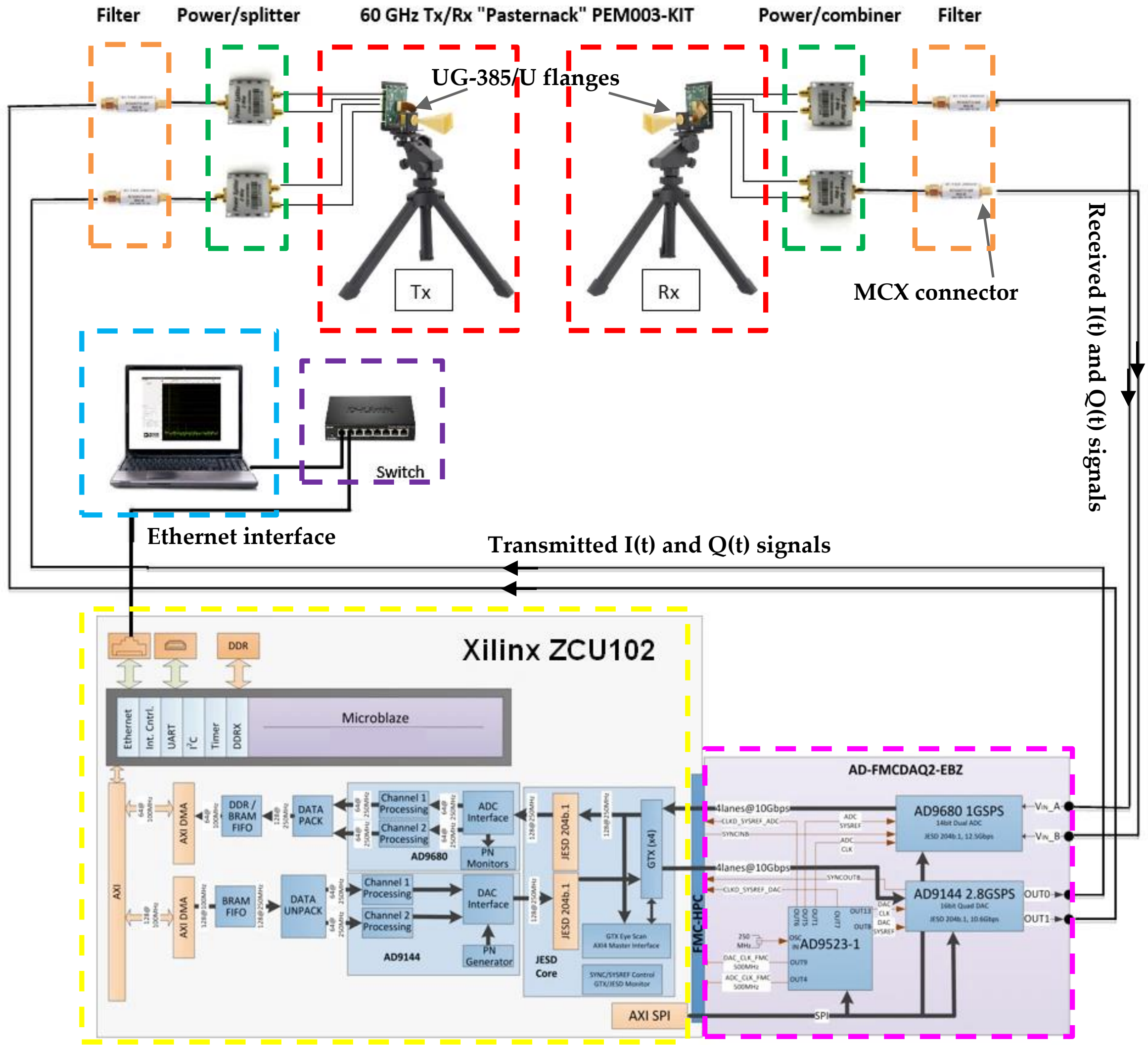


Figure 5. Demonstrator of the “Wireless Connector” system based on the RF-PEM003, ZCU102 and FMCDAQ2 components.

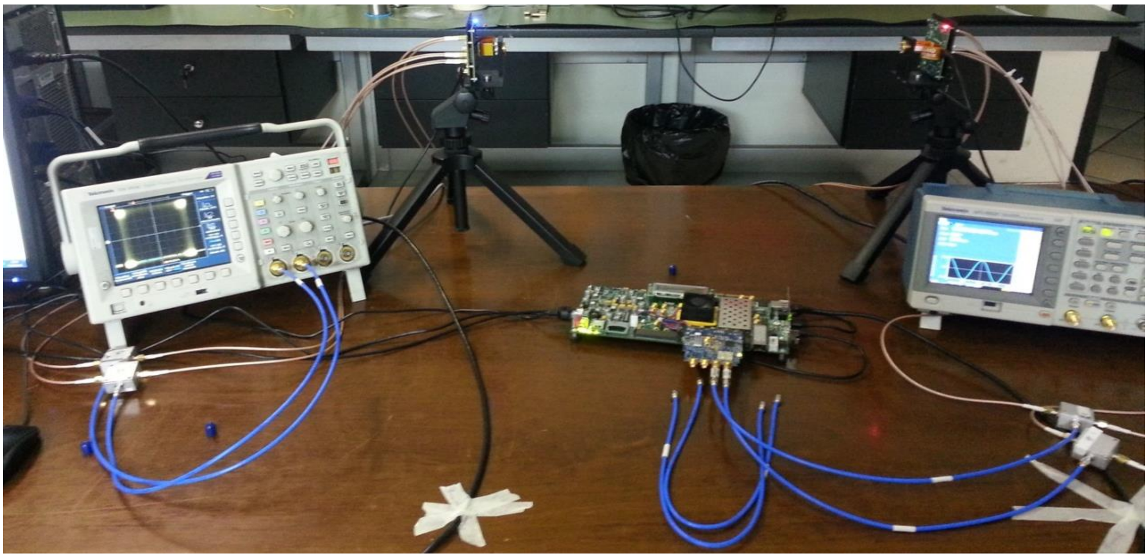


Figure 6. Picture of the experimental setup using the Zynq Ultrascale+MPSoC ZCU102 platform baseband system interconnected to the PEM003 radio system, operating in QPSK modulation.

The QPSK modulation is carried out by generating the I(t) and Q(t) coefficients, to be sent to the two quadrature mixers, where they are mixed with the carrier signal (I(t)) and the latter phase-shifted by 90° (Q(t)), respectively, both produced by the modem block. A representation of the modulated signals can be made on the complex plane, obtaining four symbols which constitute the QPSK constellation (as shown in Figure 7).

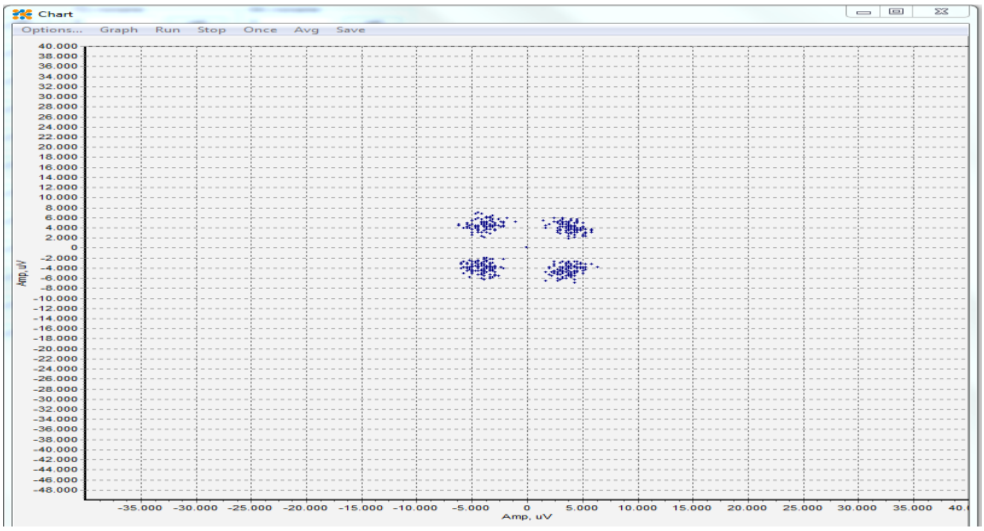


Figure 7. Graphic representation of the QPSK constellation related to the received signals, using the Analog Devices software tool IIO Oscilloscope.

The QPSK demodulation is based on the principle of coherent demodulation, which requires an appropriate reconstruction of the base-band symbols. The ADC component included on the ADFMCDAQ2 acquisition board provides the samples of the received I(t) and Q(t) signals, subsequently processed by First-in First-Out (FIFO) systems for modifying the data-flow on 64-bit registers at a 500 MHz frequency. These data are sent to a threshold decision-maker block for reconstructing the symbols on the receiver side. The constellation of received symbols (Figure 7), with 500 Ms/s symbol rate corresponding to 1 Gbit/s (but extendable up to 3 Gbit/s, as previously reported), has been displayed through a software application (Analog devices IIO Oscilloscope) for extrapolating the transmitted symbol, processed by FPGA and sent to the PC via the UART port.

3. Results

3.1. Description of the VHDL Blocks Implemented for the AES Encryption/Decryption Algorithm

The VDHL block developed for implementing the encryption algorithm is shown in Figure 8 (red box); it accepts the plaintext in input and provides the ciphertext in output, both arranged into 128-bit blocks, via the AXI Stream bus.

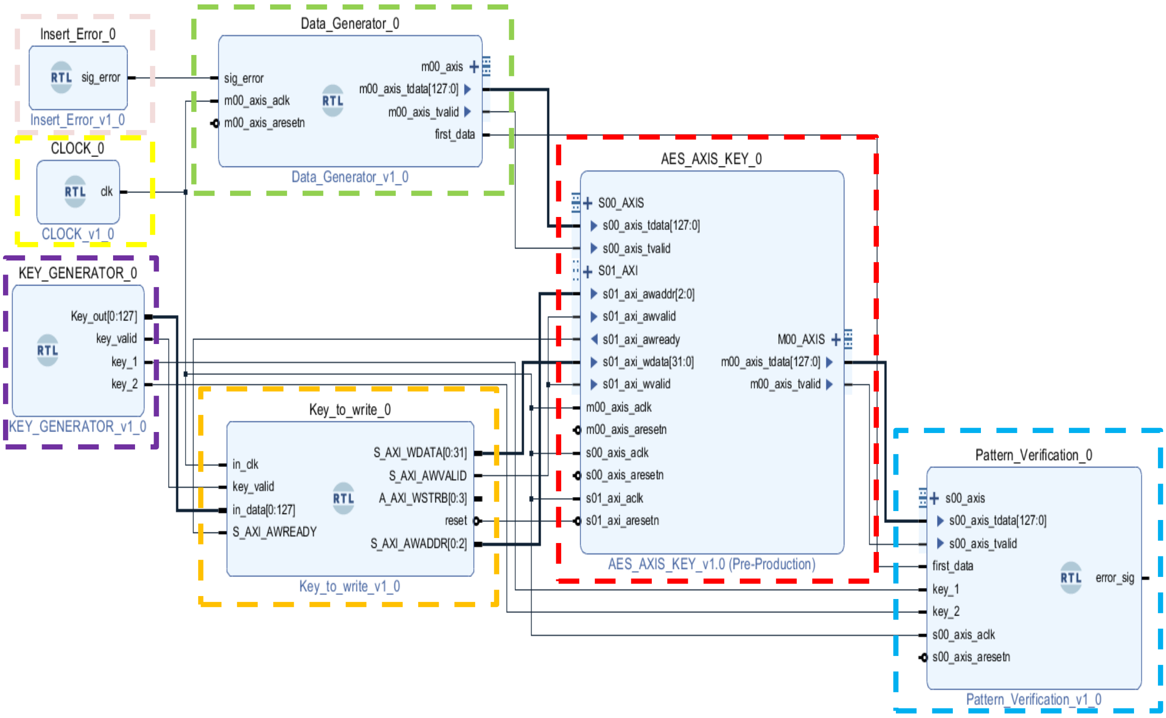
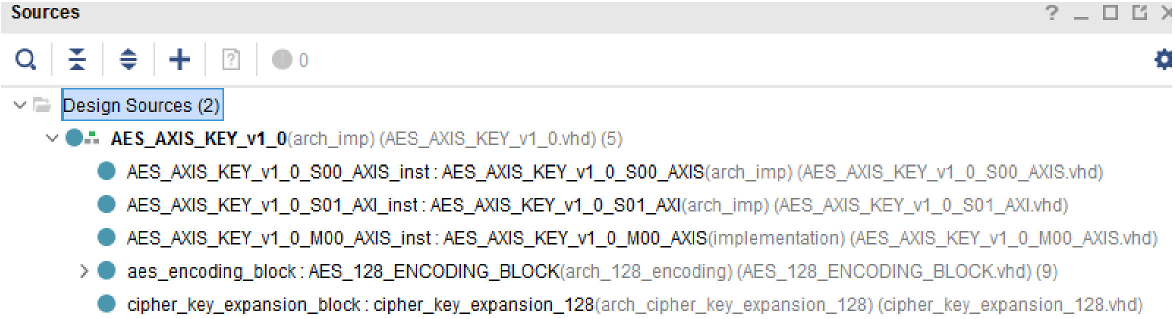


Figure 8. Complete scheme containing all the blocks implemented, both for the encryption algorithm and for testing it.

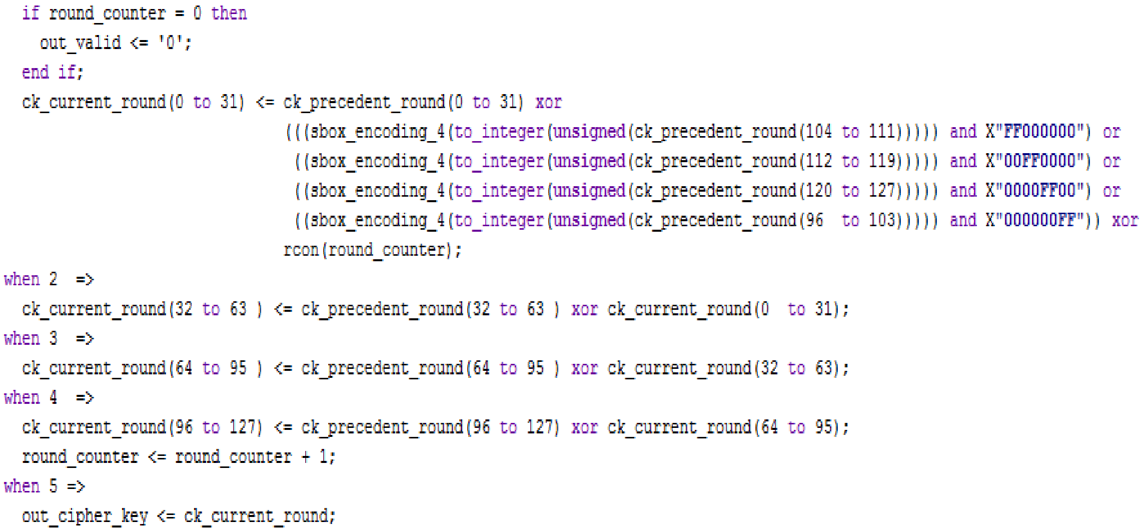
The source files implementing the encryption block are shown in Figure 9; the first four files are, AES_AXIS_KEY_v1, AES_AXIS_KEY_v1_0_S00_AXIS_inst, AES_AXIS_KEY_v1_0_S01_AXI_inst and AES_AXIS_KEY_v1_0_M00_AXIS_inst, related to the implementation of the communication between blocks, through AXI Bus Stream and AXI Lite; instead, the last two files contain the code for performing the encryption algorithm, namely aes_encoding_block and cipher_key_expansion_block. The portion of the firmware, contained in cipher_key_expansion_block, dealing with the expansion of the key is shown in Figure 10, which performs the necessary operations to obtain the 44 words to make up the 10 sub-keys, used during the encryption rounds. Also, the start of the key expansion routine, whenever a new key is validated by the processor, has been implemented using the expansion_key_start signal; in order to obtain all the 44 words of the expanded key, only 174.5 ns are required.

As can be seen, the 44 words, constituting the sub-keys, are obtained by carrying out xor operations between the 32-bit sections of the subkey at the previous round.



```
Design Sources (2)
  AES_AXIS_KEY_v1_0(arch_imp) (AES_AXIS_KEY_v1_0.vhd) (5)
    AES_AXIS_KEY_v1_0_S00_AXIS_inst : AES_AXIS_KEY_v1_0_S00_AXIS(arch_imp) (AES_AXIS_KEY_v1_0_S00_AXIS.vhd)
    AES_AXIS_KEY_v1_0_S01_AXI_inst : AES_AXIS_KEY_v1_0_S01_AXI(arch_imp) (AES_AXIS_KEY_v1_0_S01_AXI.vhd)
    AES_AXIS_KEY_v1_0_M00_AXIS_inst : AES_AXIS_KEY_v1_0_M00_AXIS(implementation) (AES_AXIS_KEY_v1_0_M00_AXIS.vhd)
  > aes_encoding_block : AES_128_ENCODING_BLOCK(arch_128_encoding) (AES_128_ENCODING_BLOCK.vhd) (9)
    cipher_key_expansion_block : cipher_key_expansion_128(arch_cipher_key_expansion_128) (cipher_key_expansion_128.vhd)
```

Figure 9. Source files used to implement the code performing the encryption process.

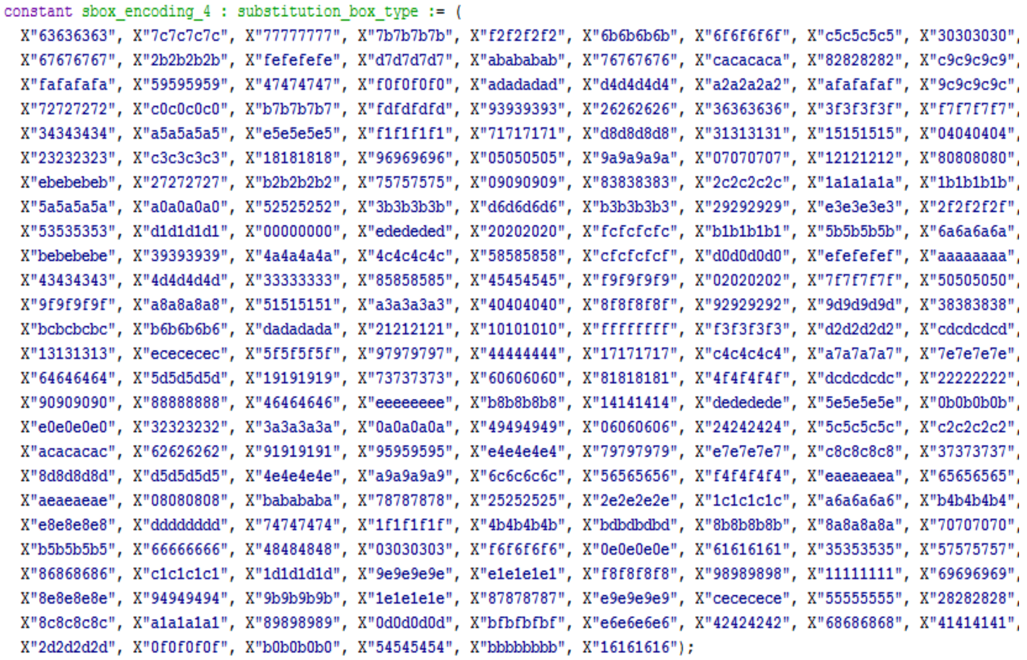


```
  if round_counter = 0 then
    out_valid <= '0';
  end if;
  ck_current_round(0 to 31) <= ck_precedent_round(0 to 31) xor
                              (((sbox_encoding_4(to_integer(unsigned(ck_precedent_round(104 to 111))))) and X"FF000000") or
                               ((sbox_encoding_4(to_integer(unsigned(ck_precedent_round(112 to 119))))) and X"00FF0000") or
                               ((sbox_encoding_4(to_integer(unsigned(ck_precedent_round(120 to 127))))) and X"0000FF00") or
                               ((sbox_encoding_4(to_integer(unsigned(ck_precedent_round(96  to 103))))) and X"000000FF")) xor
                              rcon(round_counter);
when 2  =>
  ck_current_round(32 to 63 ) <= ck_precedent_round(32 to 63 ) xor ck_current_round(0  to 31);
when 3  =>
  ck_current_round(64 to 95 ) <= ck_precedent_round(64 to 95 ) xor ck_current_round(32 to 63);
when 4  =>
  ck_current_round(96 to 127) <= ck_precedent_round(96 to 127) xor ck_current_round(64 to 95);
  round_counter <= round_counter + 1;
when 5 =>
  out_cipher_key <= ck_current_round;
```

Figure 10. Code section used to generate the 10 subkeys, for encrypting the 128-bit plaintext data packets.

A Sbox matrix is employed to expand the key (Figure 11); as mentioned above, each element of the Sbox consists of 32 bits, instead of 8 bits, so allowing the algorithm to perform the related operations and thus obtaining the encrypted data packets, in a shorter temporal interval but with greater resource utilization of the FPGA device. This LUT-based solution was preferred over solutions that implement Sbox through GF operations, as those reported in References [31,33], because the main prerogative of the Wireless Connector is the operating speed rather than hardware resources utilization, given the wide memory capability of the employed FPGA platform; as known, LUT-based Sbox solutions offer better performances in terms of processing time to the detriment of area occupation, as demonstrated in Reference [30], thus affecting the Substitute Bytes step, the most critical operation in the AES algorithm but also the key expansion step in the proposed implementation.

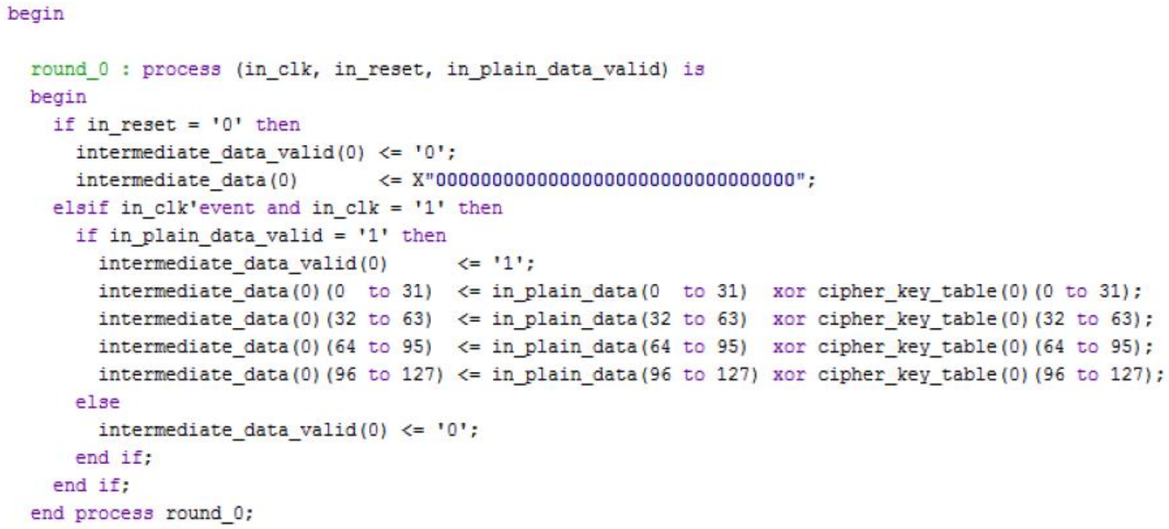
Once the 10 sub-keys are obtained, the algorithm carries out the 10 rounds required by the AES-128 and implemented in the aes_encoding_block source file, which receives in input the sub-keys and the plaintext and carried out the steps required to encrypt the plain text (Figure 1).



```
constant sbox_encoding_4 : substitution_box_type := (
  X"63636363", X"7c7c7c7c", X"77777777", X"7b7b7b7b", X"f2f2f2f2", X"6b6b6b6b", X"6f6f6f6f", X"c5c5c5c5", X"30303030",
  X"67676767", X"2b2b2b2b", X"fefefefe", X"d7d7d7d7", X"abababab", X"76767676", X"cacacaca", X"82828282", X"c9c9c9c9",
  X"fafafafa", X"59595959", X"47474747", X"f0f0f0f0", X"adadadad", X"d4d4d4d4", X"a2a2a2a2", X"afafafaf", X"9c9c9c9c",
  X"72727272", X"c0c0c0c0", X"b7b7b7b7", X"fdfdfdfd", X"93939393", X"26262626", X"36363636", X"3f3f3f3f", X"f7f7f7f7",
  X"34343434", X"a5a5a5a5", X"e5e5e5e5", X"f1f1f1f1", X"71717171", X"d8d8d8d8", X"31313131", X"15151515", X"04040404",
  X"23232323", X"c3c3c3c3", X"18181818", X"96969696", X"05050505", X"9a9a9a9a", X"07070707", X"12121212", X"80808080",
  X"ebebebeb", X"27272727", X"b2b2b2b2", X"75757575", X"09090909", X"83838383", X"2c2c2c2c", X"1a1a1a1a", X"1b1b1b1b",
  X"5a5a5a5a", X"a0a0a0a0", X"52525252", X"3b3b3b3b", X"d6d6d6d6", X"b3b3b3b3", X"29292929", X"e3e3e3e3", X"2f2f2f2f",
  X"53535353", X"d1d1d1d1", X"00000000", X"edededed", X"20202020", X"fcfcfcfc", X"b1b1b1b1", X"5b5b5b5b", X"6a6a6a6a",
  X"bebebebe", X"39393939", X"4a4a4a4a", X"4c4c4c4c", X"58585858", X"cfcfcfcf", X"d0d0d0d0", X"efefefef", X"aaaaaaaa",
  X"43434343", X"4d4d4d4d", X"33333333", X"85858585", X"45454545", X"f9f9f9f9", X"02020202", X"7f7f7f7f", X"50505050",
  X"9f9f9f9f", X"a8a8a8a8", X"51515151", X"a3a3a3a3", X"40404040", X"8f8f8f8f", X"92929292", X"9d9d9d9d", X"38383838",
  X"bcbcbcbc", X"b6b6b6b6", X"dadadada", X"21212121", X"10101010", X"ffffffff", X"f3f3f3f3", X"d2d2d2d2", X"cdcdcdcd",
  X"13131313", X"ecececec", X"5f5f5f5f", X"97979797", X"44444444", X"17171717", X"c4c4c4c4", X"a7a7a7a7", X"7e7e7e7e",
  X"64646464", X"5d5d5d5d", X"19191919", X"73737373", X"60606060", X"81818181", X"4f4f4f4f", X"dcdcdcdc", X"22222222",
  X"90909090", X"88888888", X"46464646", X"eeeeeeee", X"b8b8b8b8", X"14141414", X"dededede", X"5e5e5e5e", X"0b0b0b0b",
  X"e0e0e0e0", X"32323232", X"3a3a3a3a", X"0a0a0a0a", X"49494949", X"06060606", X"24242424", X"5c5c5c5c", X"c2c2c2c2",
  X"acacacac", X"62626262", X"91919191", X"95959595", X"e4e4e4e4", X"79797979", X"e7e7e7e7", X"c8c8c8c8", X"37373737",
  X"8d8d8d8d", X"d5d5d5d5", X"4e4e4e4e", X"a9a9a9a9", X"6c6c6c6c", X"56565656", X"f4f4f4f4", X"eaeaeaea", X"65656565",
  X"aeaeaeae", X"08080808", X"babababa", X"78787878", X"25252525", X"2e2e2e2e", X"1c1c1c1c", X"a6a6a6a6", X"b4b4b4b4",
  X"e8e8e8e8", X"dddddddd", X"74747474", X"1f1f1f1f", X"4b4b4b4b", X"bdbdbdbd", X"8b8b8b8b", X"8a8a8a8a", X"70707070",
  X"b5b5b5b5", X"66666666", X"48484848", X"03030303", X"f6f6f6f6", X"0e0e0e0e", X"61616161", X"35353535", X"57575757",
  X"86868686", X"c1c1c1c1", X"1d1d1d1d", X"9e9e9e9e", X"e1e1e1e1", X"f8f8f8f8", X"98989898", X"11111111", X"69696969",
  X"8e8e8e8e", X"94949494", X"9b9b9b9b", X"1e1e1e1e", X"87878787", X"e9e9e9e9", X"cececece", X"55555555", X"28282828",
  X"8c8c8c8c", X"a1a1a1a1", X"89898989", X"0d0d0d0d", X"bfbfbfbf", X"e6e6e6e6", X"42424242", X"68686868", X"41414141",
  X"2d2d2d2d", X"0f0f0f0f", X"b0b0b0b0", X"54545454", X"bbbbbbbb", X"16161616");
```

Figure 11. S-Box matrix containing 256 elements, each of 32 bit.

In the first round, the xor operation between the plaintext and the cipher_key_table is carried out, which contains the unexpanded encryption key (round_0 in Figure 12).

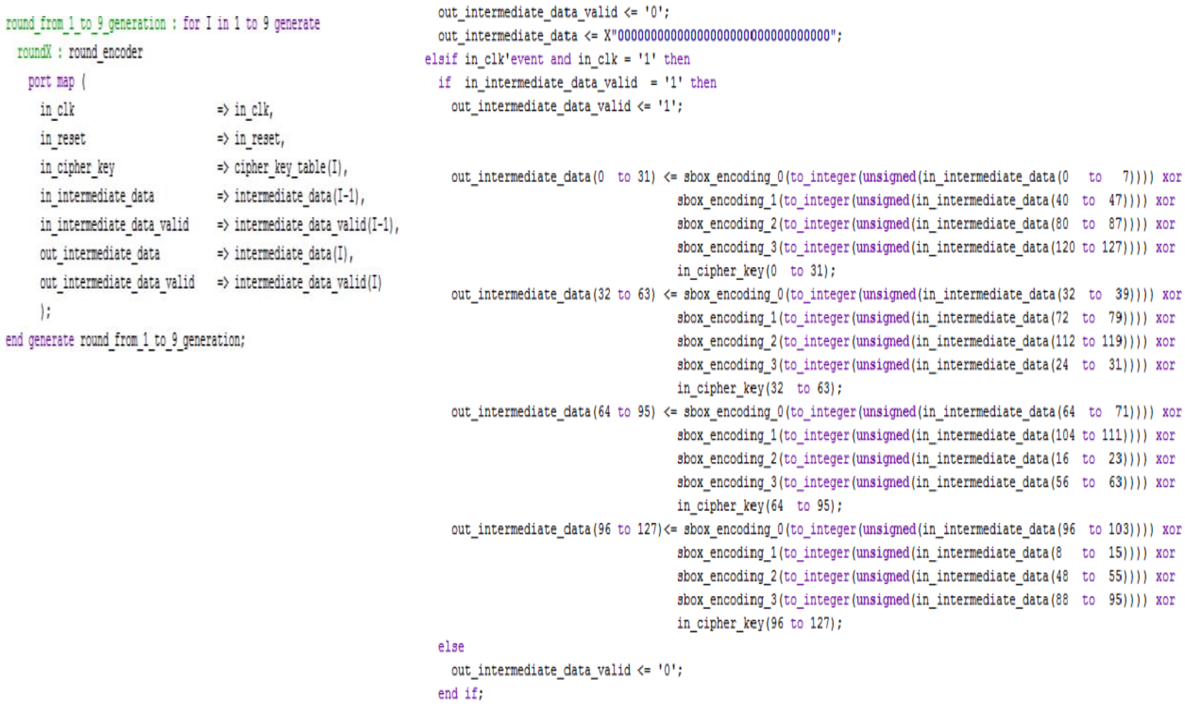


```
begin

  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;
```

Figure 12. Code Section related to the first round (called round_0) of the AES-128 algorithm.

Afterward, the algorithm, using the intermediate data generated by the first round, proceeds with the following 9 rounds required by the AES-128, performing in each round the Substitute Bytes, Shift Rows, Mix Columns and Add Round Key operations (Figure 13a). These operations are iteratively applied to the intermediate data obtained from the previous round, updated until the ninth round (Figure 13b). The data obtained after this iteration, called intermediate_data (9), is provided to round 10 for the last Add Round Key operation and the resulting ciphered data is stored into 128-bit out_cipher_data packet (Figure 14).



```
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;
```

(**a**)

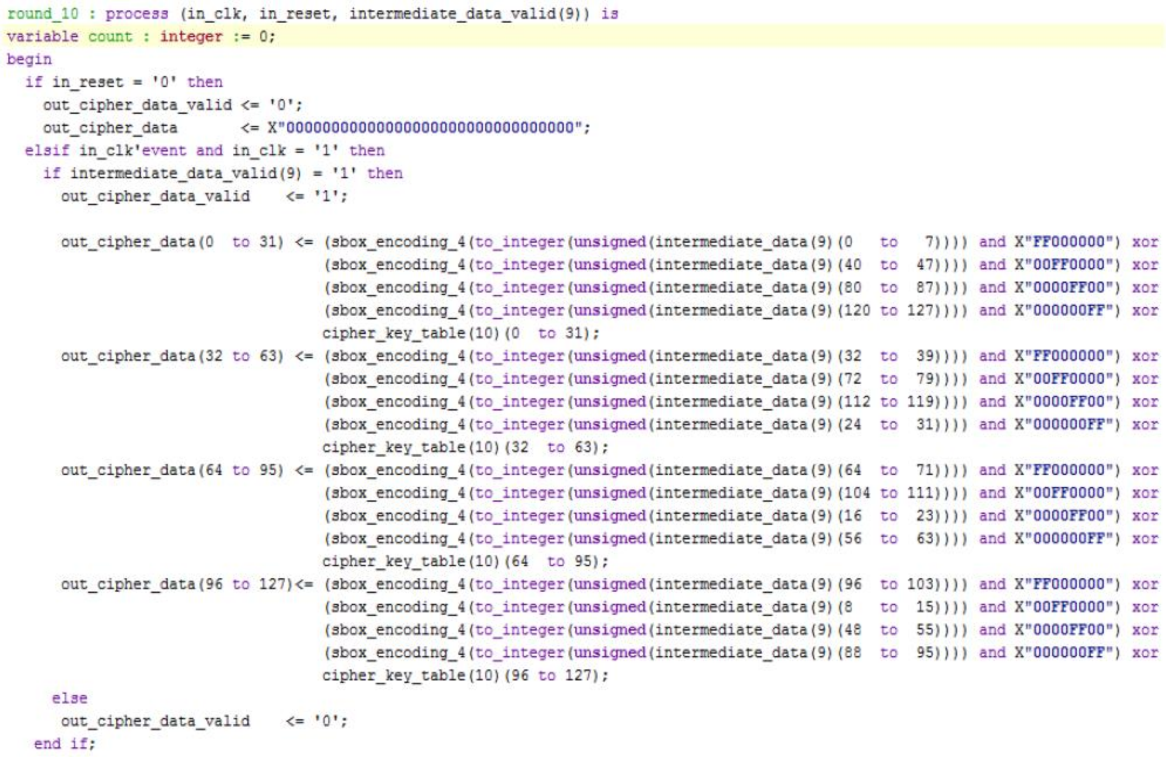
```
  out_intermediate_data_valid <= '0';
  out_intermediate_data <= X"00000000000000000000000000000000";
elsif in_clk'event and in_clk = '1' then
  if  in_intermediate_data_valid  = '1' then
    out_intermediate_data_valid <= '1';


    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;
```

(**b**)

Figure 13. Generation of the 9 intermediate rounds implementing the operation provided by the AES-128 (**a**); code section implementing the operations required by each round (**b**).



```
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';

      out_cipher_data(0  to 31) <= (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(0   to   7)))) and X"FF000000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(40  to  47)))) and X"00FF0000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(80  to  87)))) and X"0000FF00") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(120 to 127)))) and X"000000FF") xor
                                   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)))) and X"FF000000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(72  to  79)))) and X"00FF0000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(112 to 119)))) and X"0000FF00") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(24  to  31)))) and X"000000FF") xor
                                   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)))) and X"FF000000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(104 to 111)))) and X"00FF0000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(16  to  23)))) and X"0000FF00") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(56  to  63)))) and X"000000FF") xor
                                   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)))) and X"FF000000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(8   to  15)))) and X"00FF0000") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(48  to  55)))) and X"0000FF00") xor
                                   (sbox_encoding_4(to_integer(unsigned(intermediate_data(9)(88  to  95)))) and X"000000FF") xor
                                   cipher_key_table(10)(96 to 127);
     else
      out_cipher_data_valid    <= '0';
    end if;
```

Figure 14. Code section related to the Add Round Key carried out in the last round (called round_10) of the AES-128 algorithm.

By saving the results of each round (i.e., intermediate_data(i), i = 1, . . . , 9), a pipelined implementation can be obtained, carrying out simultaneously the 10 rounds on successive data packets, thus allowing to start the processing of a new packet as soon as the round's processing on the previous ones is completed. Therefore, simultaneous processing on multiple packets is performed, thus allowing better exploitation of the used hardware resources, so reaching higher data throughput. As below reported, the proposed AES implementation takes only a clock period to complete the round's processing, allowing to provide an encrypted data packet for each clock cycle.

To test the correct behaviour of the implemented algorithm, a word generator, called Data_Generator (green box in Figure 8), has been included in the tool offered by Vivado IP INTEGRATOR for simulating the presence of the ethernet module, that provides 128-bit data packets at the input of the encryption block, every 42.67 ns, via the AXI Stream bus. Instead, to insert and store the key, an external block called Key_generator (purple box in Figure 8) and a memory block with 4 registers of 32-bit each have been employed, connected via AXI Lite bus, so allowing the user to update the key at any time. A Key_to_write block (orange box in Figure 8) writes the 4 words of the key (32 bits each) in 4 registers, created during the AXI Lite bus implementation phase, asynchronously to the processor, allowing the substitution of the encryption key during the normal operation of the algorithm. Therefore, if the key in the registers is not changed, the algorithm performs the data encryption, otherwise, if it differs from the current key, the expansion_key routine starts and the 10 sub-keys of the new main key are generated.

The switching to a new key is enabled when the processor deems it appropriate by setting a bit of an additional byte transmitted via the AXI Lite bus, stored in an additional 32-bit register, named key_valid. The algorithm queries this bit every 42.7 ns and if it detects that its value is set high, it reads the key stored in the registers and starts the key expansion routine; at the same time, the value of the enabling bit is reset for indicating to decryption block the changing of the encryption key. The developed key substitution mechanism represents an important functionality for the Wireless Connector since a periodic key change is required, for guaranteeing the security of the data exchanged between the two mobile stations constituting the communication system.

The correctness of the encrypted data packets is verified by a Pattern_Verificator block (blue box in Figure 8), connected to the encryption block via the AXI Stream bus; this last simulates the presence of the modem and contains a table with encrypted data packets corresponding to the plaintext data packets provided at the input of the AES_TEST_AXI block by the Data_Generator block. It compares the packets received from the AES_TEST_AXI block with those contained in the table; if the data received is the same as that in its table, the encryption has been successful, otherwise, an error has occurred. To verify the correct operation of the algorithm, an Insert_Error block (pink box in Figure 8) has been implemented to change a bit in the 128-bit plaintext packet, thus verifying the presence of any errors by the Pattern_Verificator. When an error is detected, this last set the error_sig bit in correspondence with the encrypted data packet that does not match the stored ones in the Pattern_Verificator table. The CLOCK block (yellow box in Figure 8) provides the clock to all the blocks with a frequency of 350 MHz. To synchronize the Pattern_Verificator with the encryption block, an impulse is generated to indicate the end of encryption and the availability of a new encrypted data packets at the output of the AES_AXIS_KEY block; this signal is associated to the m00_axis_tvalid pin of the AXI-Strem bus. Besides, two other signals have been implemented, namely a support flag and a signal indicating the packet of the Data_Generator table provided to the input of the encryption block, allowing the Pattern_Verificator to keep track of the packets sent and to associate them to the corresponding entries in its internal table.

Furthermore, an external signal has been defined, called s00_axis_tvalid, which indicates, through an impulse, to the encryption block the availability of the packets at the input, enabling the immediate acceptance of new data packets. To optimize the algorithm and to reduce the execution time, the plaintext packets are acquired on the falling edge of the clock signal, thus allowing to start with the encryption process in advance, so gaining 1.42 ns corresponding to half of the clock period.

The modem, located downstream of the encryption block and simulated by the Pattern_Verificator block, works with 64-bit data packets; therefore, the encrypted packets have to be serialized in 64-bit packets each, determining some latency and timing problems. Therefore, the m00_axis_tready signal, provided by the AXI Stream bus, has been implemented for indicating to the encryption block, when the Pattern_Verificator is available, to accept new packets. When the signal is set, the algorithm accepts the packets in input and performs the encryption process; otherwise, if it is reset, the algorithm stops and waits for the signal to return high. Instead, the m00_axis_tvalid signal indicates to Pattern_Verificator, that a new encrypted data packet is available at the output.

In Figure 15, the temporal trends of the signals involved in the developed encryption algorithm are shown; the s00_axis_tvalid signal is generated on the falling edge of the clock, whereas encryption of plaintext data packets starts on each rising edge of the clock. Also, to consider the case in which the s00_axis_tvalid signal is set to zero, the last two packets are made available using two impulses randomly spaced. Therefore, the algorithm accepts the plaintext data packets to perform encryption and provides the corresponding encrypted data packets after an interval of 28.560 ns. From the above considerations, the developed algorithm can supply 128-bit encrypted data packets every 2.856 ns (equal to the clock period), so obtaining, for a 350 MHz operating frequency, a throughput value of 44.8 Gbit/s.

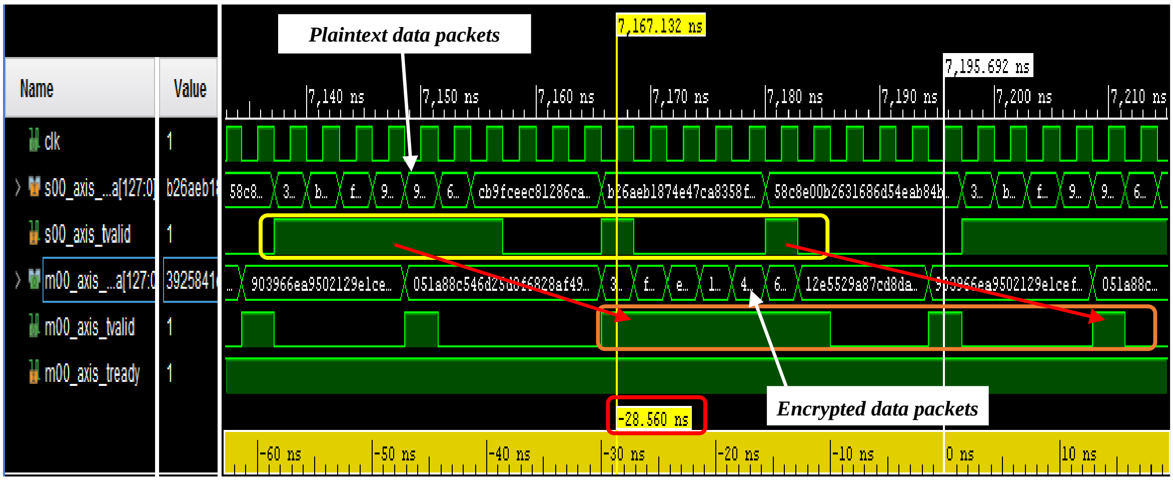


Figure 15. Temporal trends of the signals involved in the encryption algorithm, with the plaintext data packets encrypted on each rising edge of the clock, as indicated by the s00_axis_tvalid signal set to one and the last two packets made available using two impulses spaced randomly over time (yellow box); each encrypted data packet is available at the output after 28.560 ns (10 system clock periods at 350 MHz frequency), as indicated by the m00_axis_tvalid signal (orange box).

The temporal trends related to the expansion of the key are shown in Figure 16, previously stored in the registers in an instant chosen by the user and validated through a signal provided by the processor. During the expansion of the key, which lasts 174.55 ns, the m00_axis_tvalid signal is reset indicating that no valid encrypted packets are provided from the encryption block. The error_sig signal is set in correspondence to the key change because the table related to the new key is considered, whereas the packets are still obtained with the old key; the signal returns to zero as soon as the encrypted packets are obtained through the new key.

The implementation of all the control and synchronization signals, above described, is one of the main contributions provided by the proposed work, fundamental for the correct operation of the whole encryption/decryption system, ensuring correct interoperability of the developed encryption/decryption block with the other sections of the Wireless Connector system.

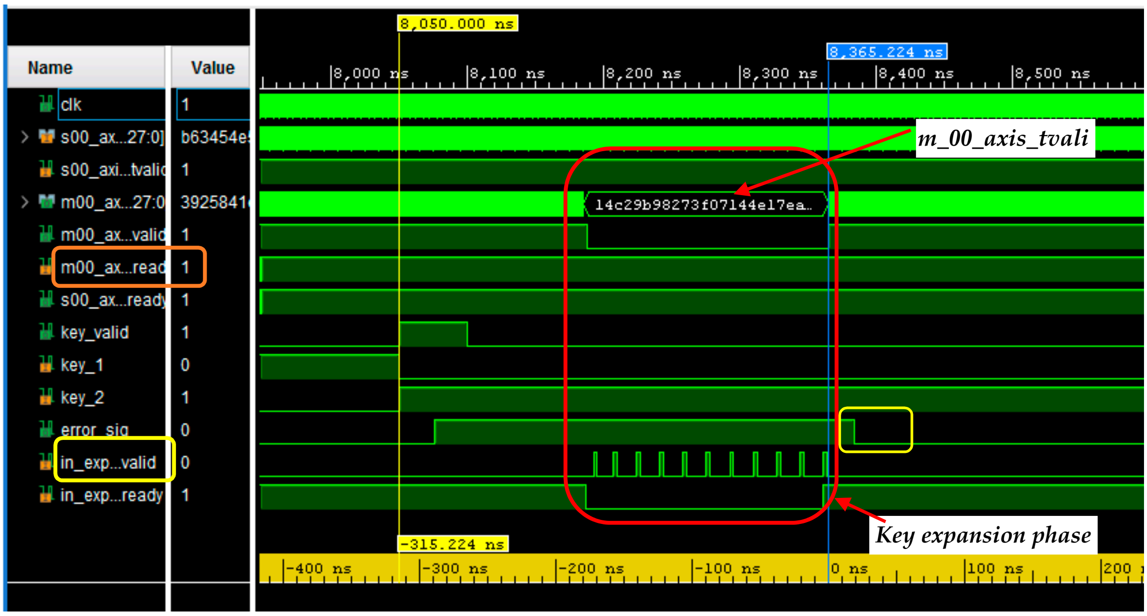


Figure 16. Temporal trends of the key expansion phase; the m00_axis_tvalid (orange box) signal indicates that during this phase there are no valid packets at the output of the encryption block; the signal that error_sig signal (yellow box) returns to zero as soon as the encrypted packets are obtained through the new key.

Afterward, the VHDL block implementing the correspondent decryption algorithm, called AES_128_DEC (red box in Figure 17), has been developed, along with the blocks employed to test it, reproducing an operative scenario similar to that present in the final application.

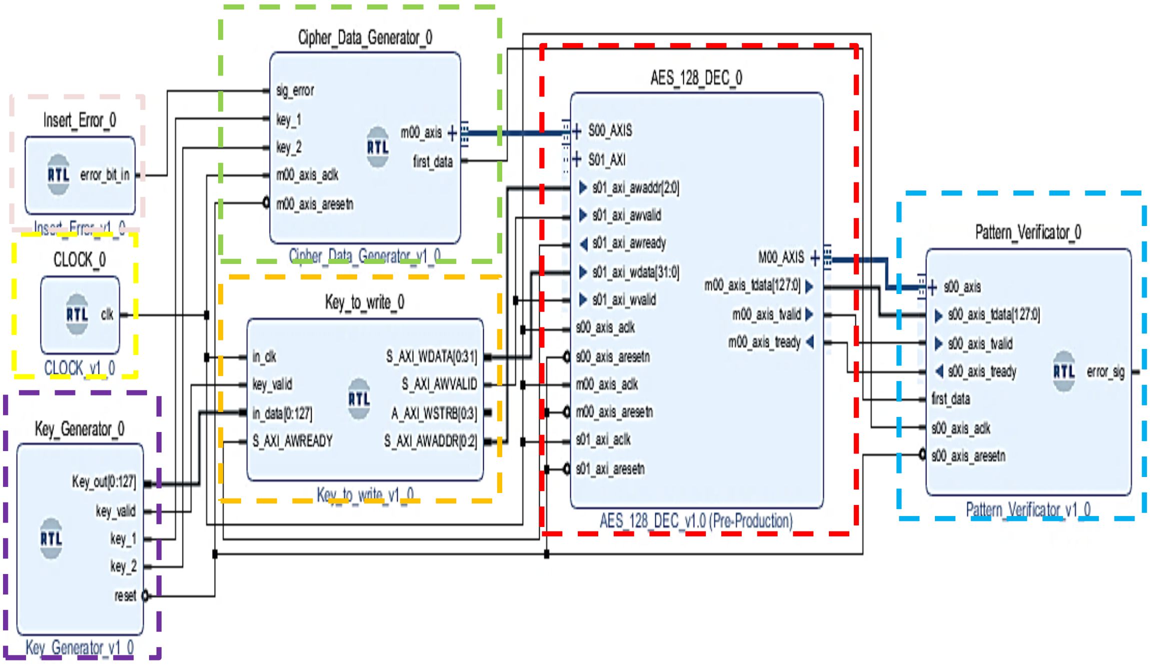
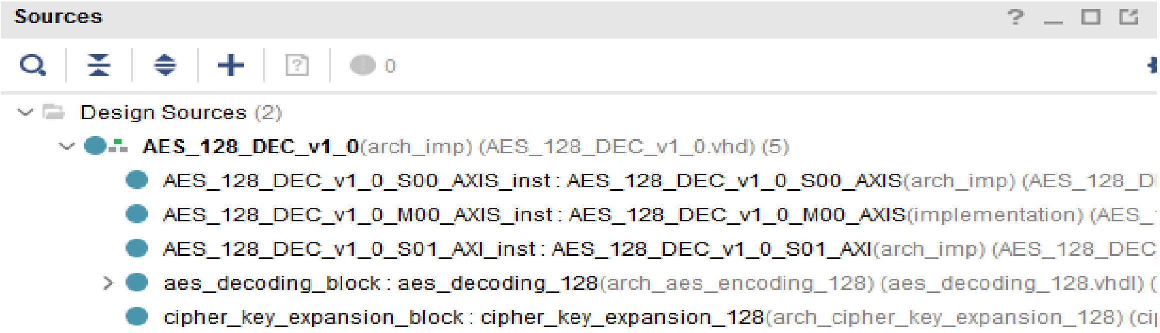


Figure 17. Overall scheme containing the VHDL blocks for both implementing the decryption algorithm and testing it, reproducing the operative scenario of the final application.

The decryption algorithm has been implemented, as well as the encryption algorithm, parallelizing many logical instructions on each rising edge of the clock; similarly to the encryption algorithm, a Sbox matrix was used, consisting of 256 elements each of 32 bits. In Figure 18, the source files used to implement the code performing the decryption are shown; the first four files, AES_128_DEC_v1_0, AES_128_DEC_v1_0_S00_AXIS_inst, AES_128_DEC_v1_0_S01_AXI_inst and AES_128_ DEC_v1_0_M00_AXIS_inst are related to the implementation of the communication between blocks by the means of AXI Bus Stream and AXI Lite. The files containing the code developed to perform the AES-128 decryption algorithm are the last two shown in Figure 18, named aes_decoding_block and cipher_key_expansion_block.

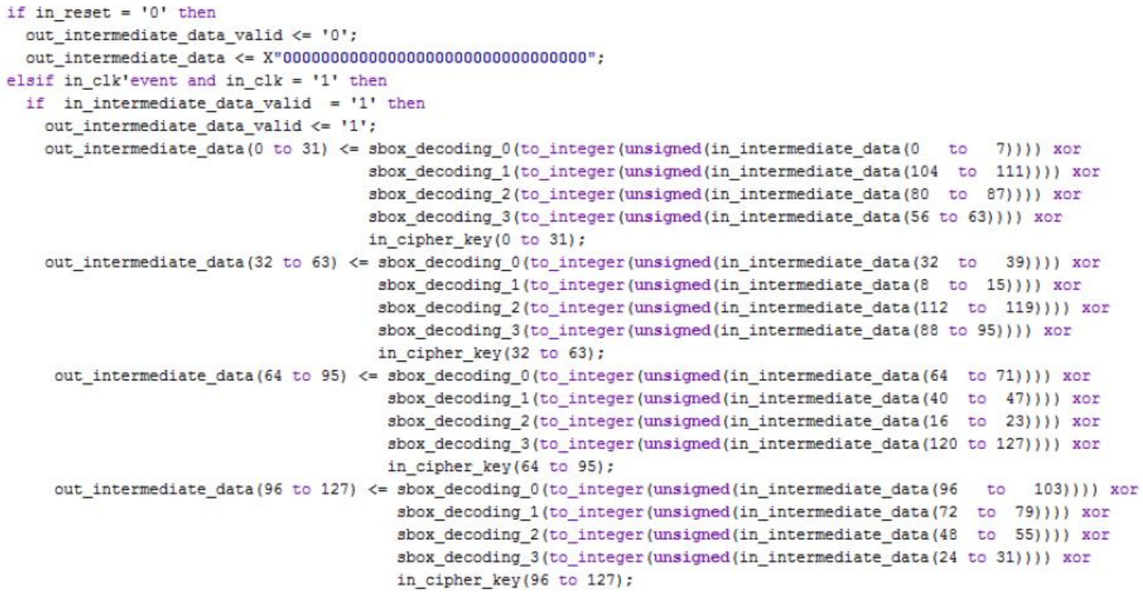


```
Design Sources (2)
  AES_128_DEC_v1_0(arch_imp) (AES_128_DEC_v1_0.vhd) (5)
    AES_128_DEC_v1_0_S00_AXIS_inst : AES_128_DEC_v1_0_S00_AXIS(arch_imp) (AES_128_D
    AES_128_DEC_v1_0_M00_AXIS_inst : AES_128_DEC_v1_0_M00_AXIS(implementation) (AES_
    AES_128_DEC_v1_0_S01_AXI_inst : AES_128_DEC_v1_0_S01_AXI(arch_imp) (AES_128_DEC
  > aes_decoding_block : aes_decoding_128(arch_aes_encoding_128) (aes_decoding_128.vhdl) (
    cipher_key_expansion_block : cipher_key_expansion_128(arch_cipher_key_expansion_128) (ci
```

Figure 18. Source files used to implement the code performing the decryption process.

In the cipher_key_expansion_block file, the code to implement the key expansion has been implemented, to generate the 10 sub-keys employed during the decryption process; to expand the key, the same matrix used for the encryption process is deployed (called sbox_encoding_4). The 10 rounds for obtaining the plaintext data packets are implemented within the aes_decoding_block source file and the rounds are developed in the same way, as done for the encryption operation.

In each round, the InvSubBytes, InvShiftRows and InvMixColumns operations are combined to obtain the plaintext data packets. These operations are carried out by using the four 16 × 16 32-bit matrices, called sbox_decoding_0, sbox_decoding_1, sbox_decoding_2 and sbox_decoding_3, equivalent to operations of the AES decrypting algorithm; in particular, the xor operations between the intermediate data obtained during the different rounds of decryption algorithm and elements of these matrices are carried out (Figure 19). These matrices allow obtaining the plaintext data packets in only 10 clock periods, considerably reducing the necessary time to perform the decryption process; however, greater resource utilization of the FPGA device is required.



```
if in_reset = '0' then
  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';
    out_intermediate_data(0 to 31) <= sbox_decoding_0(to_integer(unsigned(in_intermediate_data(0   to   7)))) xor
                                      sbox_decoding_1(to_integer(unsigned(in_intermediate_data(104  to  111)))) xor
                                      sbox_decoding_2(to_integer(unsigned(in_intermediate_data(80   to  87)))) xor
                                      sbox_decoding_3(to_integer(unsigned(in_intermediate_data(56 to 63)))) xor
                                      in_cipher_key(0 to 31);
    out_intermediate_data(32 to 63) <= sbox_decoding_0(to_integer(unsigned(in_intermediate_data(32  to   39)))) xor
                                       sbox_decoding_1(to_integer(unsigned(in_intermediate_data(8   to  15)))) xor
                                       sbox_decoding_2(to_integer(unsigned(in_intermediate_data(112  to  119)))) xor
                                       sbox_decoding_3(to_integer(unsigned(in_intermediate_data(88 to 95)))) xor
                                       in_cipher_key(32 to 63);
     out_intermediate_data(64 to 95) <= sbox_decoding_0(to_integer(unsigned(in_intermediate_data(64  to 71)))) xor
                                        sbox_decoding_1(to_integer(unsigned(in_intermediate_data(40  to  47)))) xor
                                        sbox_decoding_2(to_integer(unsigned(in_intermediate_data(16  to  23)))) xor
                                        sbox_decoding_3(to_integer(unsigned(in_intermediate_data(120 to 127)))) xor
                                        in_cipher_key(64 to 95);
     out_intermediate_data(96 to 127) <= sbox_decoding_0(to_integer(unsigned(in_intermediate_data(96   to   103)))) xor
                                         sbox_decoding_1(to_integer(unsigned(in_intermediate_data(72  to  79)))) xor
                                         sbox_decoding_2(to_integer(unsigned(in_intermediate_data(48  to  55)))) xor
                                         sbox_decoding_3(to_integer(unsigned(in_intermediate_data(24 to 31)))) xor
                                         in_cipher_key(96 to 127);
```

Figure 19. Code section related to the operation carried out during each round used to obtain the intermediate data and finally, the plaintext.

The developed decryption algorithm provides the plaintext data packets, at the output of the AES_128_DEC block, in just 28.560 ns for 350 MHz clock frequency, thus obtaining the same maximum data-rate as the encryption algorithm (i.e., 44.8 Gbit/s). Similarly to the encryption algorithm, the s00_axis_tvalid signal, provided by the Cipher_Data_Generator block (green box in Figure 17), indicates that the encrypted data packets are available for the decryption; the plaintext data packets provided at the output of the decryption block are reported by the m00_axis_tvalid signal, as depicted in Figure 20.

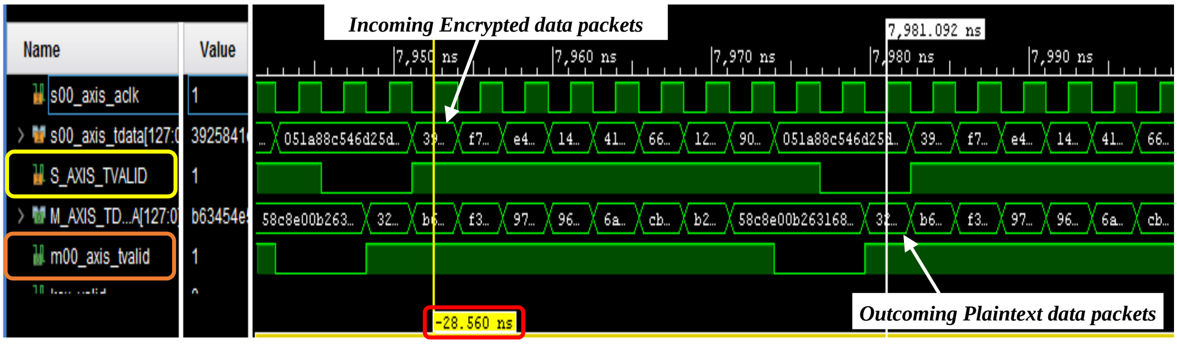


Figure 20. Temporal trends related to the decryption phase; the time interval required to obtain the plaintext data packet from the encrypted data packet is highlighted by time markers applied to s00_axis_tvalid (yellow box) and the m00_axis_tvalid (orange box) signals.

The developed decryption block receives the new key and stores it in four 32-bit registers; the key is validated setting the bit of the key_valid register, used for this purpose. The algorithm checks this bit every 85.6 ns and if it is set, the new key is acquired and the flag bit is reset, thus communicating to the processor that the change of the key has been received; afterward, the expansion key routine starts. During this process, the m00_axis_tvalid signal is reset indicating that no valid decrypted packets are available. This operation requires 205 ns, 177 ns more than the 28.56 ns needed to provide the first valid decrypted packet for the new validated key.

To verify the correctness of the decrypted data packets and to check the sensitivity of the implemented algorithm in detecting errors, the Insert_Error block (pink block in Figure 17) has been implemented, similar to those implemented in the encryption algorithm; the sig_error signal triggers the change a single bit in the input word and verify that the Pattern_verificator block (blue block in Figure 17) detects the error; when it detects the error, the error_sig bit is set in correspondence with the decrypted data packet that does not match with the word set stored in its internal table.

Finally, the s00_axis_tready signal has been configured, for indicating the availability of a decryption block to accept a new encrypted data packet. As discussed above, the implemented algorithm can accept encrypted data packets and thus perform the decryption, on every rising edge of the system clock; therefore, it is always ready to accept new encrypted data packets, consequently, the s00_axis_tready signal is reset only if the m00_axis_tready is reset, namely if the block downward the decrypting block cannot accept decrypted data packets.

3.2. Post-Synthesis Simulation Results: Resources Utilization of the Encryption/Decryption Systems

In this sub-section, the simulations performed to determine the resource utilization on the ZCU102 FPGA platform by the developed AES-128 algorithm are reported. At first, the post-synthesis simulations have been performed on both encryption and decryption blocks, with data packets provided on each rising edge of the 350 MHz clock signal; afterward, the simulation has been performed by modifying the Data_Generator block to provide data packets at the input of the encryption/decryption block every 42.7 ns, thus verifying that the hardware usage remains unchanged.

The resource utilization of FPGA related to the encryption algorithm is reported in Table 1; considering the complete encryption system, in both cases discussed above, the percentages of hardware occupation equal to 5.48% for LUTs and 0.78% for FFs have been obtained. Afterward, the simulation of the encryption system has been performed by removing all the blocks used to verify the correct behavior of the algorithm, leaving only the blocks involved in the encryption algorithm; the hardware resources utilization is 4.76% for the LUT and 0.71% for the FF. A reduction in hardware occupation of 0.72% for LUTs has been obtained compared to the previous case including all the blocks.

Table 1. Hardware resources utilization of the ZCU102 Field Programmable Gate Array (FPGA) platform related to the encryption algorithm; in particular, the complete encryption scheme, including all the blocks to test the correct operation of the algorithm and only the encryption block have been considered.

Simulation	Resource	Utilization	Utilization [%]
Complete encryption system	LUT FF	15,029 4296	5.48 0.78
Encryption block	LUT FF	13,043 3877	4.76 0.71

Considering the decryption algorithm, the post-synthesis simulations have been performed both when the data packets are provided on each rising edge of the 350 MHz clock signal and when the Data_Generator provides encrypted packets every 42.7 ns (i.e., 23.4 MHz packet rate, Table 2); in the first case, the hardware utilization of the FPGA device is 10.62% for LUTs, 0.79% for FFs and 0.25% relative to the Global Buffers (BUFG) used. In the latter case, the use of hardware resources is equal to 10.64% for the LUTs, 0.79% for the FFs and 0.25% for the BUFGs. Finally, the post-synthesis simulation of the decryption system has been performed by removing all the blocks used to verify the correct behavior of the algorithm, leaving only the block involved in the decryption algorithm. This configuration reveals a hardware utilization of 10.11% for the LUTs, 0.71% for the FFs and 0.25% for the Global Buffer, obtaining a reduction in the hardware occupation of 0.53% for the LUTs and 0.08% for the FFs compared to the complete decryption scheme (Table 2).

Table 2. Hardware resource utilization related to the complete decryption scheme, including all the blocks to test the decryption algorithm, both when the encrypted packets are received in input on each rising edge of the 350 MHz clock signal and when they are provided every 42.7 ns (i.e., 23.4 MHz); also, the resource utilization of only the decryption block are reported.

Simulation	Resource	Utilization	Utilization [%]
Complete decryption system (350 MHz packet rate)	LUT FF BUFG	29,111 4339 1	10.62 0.79 0.25
Complete decryption system (23.4 MHz packet rate)	LUT FF BUFG	29,156 4339 1	10.64 0.79 0.25
Decryption block	LUT FF BUFG	27,713 3912 1	10.11 0.71 0.25

As it can be seen from Tables 1 and 2, showing the use of hardware resources for both the encryption and decryption systems, the LUTs used on the FPGA by the latter are 1.94× more, considering the only blocks that perform the decryption and 2.12× more, considering also the blocks needed to test it, compared to the LUTs used by the encryption algorithm; this is due to the implementation of 4 matrices (sbox_decoding_0, sbox_decoding_1, sbox_decoding_2 and sbox_decoding_3) in the decryption algorithm, each containing 32-bit elements deriving from the operations of Inverse SubBytes, Inverse Shift Rows and Inverse Mix Columns. In particular, the greater hardware resources consumption is attributable to the multiplication of the Inverse Mix Columns operation carried out in the decryption block, because involve a large number of values such as 0×09090909, 0×0B0B0B0B, 0×0D0D0D0D, 0×0E0E0E0E; such multiplicative constants require the storing of numerous intermediate values inside the LUT, occupying more hardware resources and consuming more power [34]. For this reason, several strategies were proposed in the scientific literature for reducing resource utilization and power consumption [35,36]. However, since the area occupation requirement is not as stringent as the encryption/decryption speed for the specifications of the developed project, the implementation choice fell on obtaining the data packets in the shortest possible amount of time at the expense of a greater chip's area occupation.

4. Discussion

In this section, the results of the carried out post-implementation simulations on the combined system constituted by the cascade of the encryption system and the decryption one are reported, to verify that the resulting performances are acceptable for the correct operation of the algorithm, once the project is loaded on the FPGA-ZCU102 platform.

4.1. Post-Implementation Simulations: Clock Routing Issues and Overall Performances of the Combined Encryption/Decryption System

The post-implementation simulations represent the closest emulation to downloading a design to a device, providing useful indications related to the functional and timing requirements of the developed system.

After setting the appropriate parameters and using synthesizable blocks, such as the Clocking Wizard, for the system clock and the interface mappable pins on the board, for the clock signal and the error_sig signal provided by the Pattern_Verificator, the post-implementation simulation on the encryption system has been carried; the simulation results indicated a timing problem related to the propagation of the signals within the FPGA-ZCU102 chip. In particular, for a 350 MHz system clock, a Worst Negative Slack (WNS) parameter equal to −1.014 ns has been obtained, indicating excessive delays in the propagation of the digital signals inside the FPGA chip, thus resulting in incorrect scheduling of the performed tasks; therefore, a positive WNS is required, for ensuring the proper operation of the developed encryption/decryption systems.

To overcome this problem, several post-implementation simulations, with a lower system clock frequency, have been carried out, obtaining an improvement of the WNS parameter (Table 3); in particular, by using a system clock frequency of 190 MHz, a WNS value equal to 0 ns was obtained, as well as for 180 MHz operating frequency, a WNS equal to 0.056 ns resulted. Furthermore, to support a greater system clock frequency, it is possible to use the implementation strategies provided by the Vivado tool; therefore, a common strategy suitable for both the encryption and decryption blocks has been chosen, since the final simulations have been carried out on the combined system. The post-implementation simulation has been performed by setting 220 MHz system clock frequency and adopting the Explore strategy, thus obtaining the WNS parameter equal to 0.005 ns for the encryption system and 0.008 ns for the decryption one.

Table 3. Post-implementation simulation results carried out on the encryption system with different clock frequencies to establish the maximum operating frequency with a positive Worst Negative Slack (WNS) parameter.

Clock Frequency [MHz]	Worst Negative Slack [ns]	Total Negative Slack [ns]
180	0.056	0
190	0	0
200	−0.199	−0.353
250	−0.441	−0.895

The area utilization resulting from the post-implementation simulations remains unchanged compared to the results obtained through the post-synthesis simulations, showing, for the encryption system, resource utilization of 5% of LUTs, 1% of FFs, 1% of I/O ports and 1% of BUFGs, as well as for the decryption system, 10% of LUTs, 1% of FFs, 1% of I/O ports and 1% of BUFGs; finally, for both the system, there is a 25% area utilization relative to the IP Clocking Wizard block used to generate the system clock during the post-synthesis and post-implementation simulations. Before performing the post-implementation simulations of the whole system including encryption and decryption blocks, the behavioral simulations with 220 MHz clock frequency have been carried out. In Figure 21, the temporal trends of the signals are shown, obtained providing the plaintext data packets (red box) to the encryption/decryption system every 40.86 ns; this data-rate derives from the clock frequency of 220 MHz, corresponding to 4.54 ns clock period, chosen to comply with the 3 Gbit/s throughput required by the specifications of Wireless Connector system, as calculated below in Equation (5).

$$Data - Rate\ or\ Throughput = \frac{128\ \text{bit}}{9\ \times\ clock\_period} = \frac{128\ \text{bit}}{9\ \times\ 4.54\ \text{ns}} = 3.132\ \text{Gbit/s}. \tag{5}$$

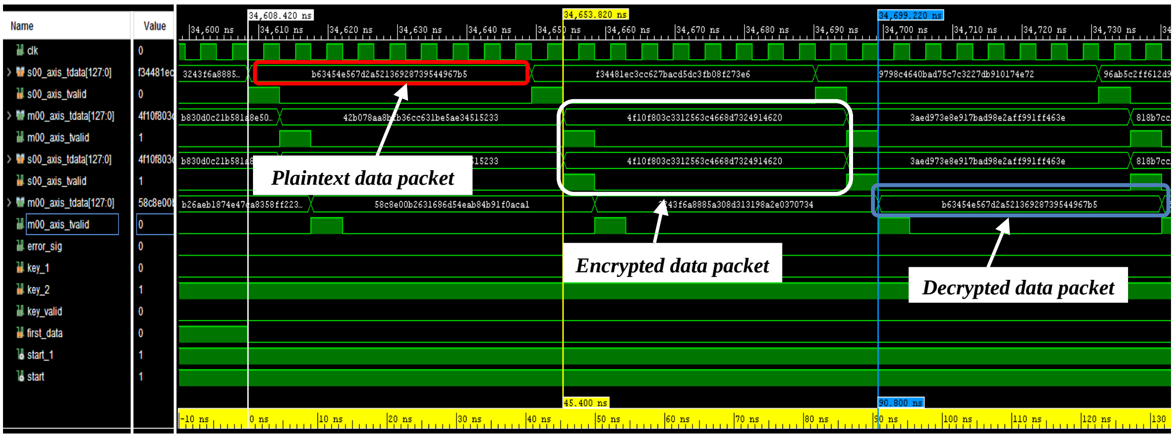


Figure 21. Temporal trends related to the behavioral simulations where plaintext data packets are provided every 9 system clock periods at 220 MHz, corresponding to a data rate of 3.123 Gbit/s.

The encryption of the data packets is performed in 9.5 clock periods (white box); in fact, the encrypted packets are provided at the output of the encryption block on the falling edge of the clock, after exactly 9.5 clock periods and then acquired by the decryption block on the next rising edge. Afterward, the data packet is decrypted in 9.5 clock periods (blue box) and provided at the output of the decryption block after overall 10 clock periods. Therefore, the encryption and decryption operations last 20 clock periods, which are equal to 90.8 ns, considering a system clock frequency of 220 MHz (Figure 21).

In Figure 22, the temporal trends related to the behavioral simulations are shown, with the plaintext data packets provided to the system in each clock period (frequency 220 MHz). The s00_axis_tvalid signal is constantly set, indicating to the receiving block that a new data packet is available at the input, providing encrypted packets on each rising edge of the clock, thus allowing a data rate equal to 28.16 Gbit/s (220 MHz $*$ 128 bit $= 28.16\ \text{Gbit}/s$).

Finally, the post-implementation simulation of the overall system constituted by the cascade of the encryption (red box) and decryption (blue box) blocks has been carried out (Figure 23a). The simulation has been performed by setting the Explore implementation strategy, provided by the Vivado tool. The screenshots of the Project Manager, obtained after the post-implementation simulation, are shown in Figure 23; a positive WNS parameter, equal to 0.056 ns, is obtained (Figure 23b), as well as the hardware utilization of the overall encryption/decryption system is reported in Figure 23c. In particular, the hardware resource utilization was equal to 15% LUTs, 1% FFs, 1% I/O ports, 1% BUFG, as well as a

25% area utilization value relative to the IP Clocking Wizard block, used to generate the system clock, was obtained.

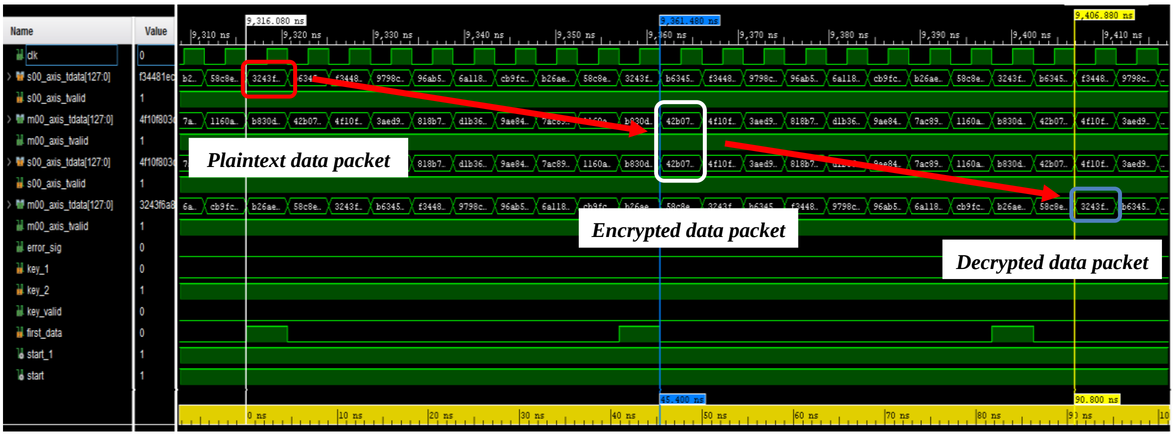
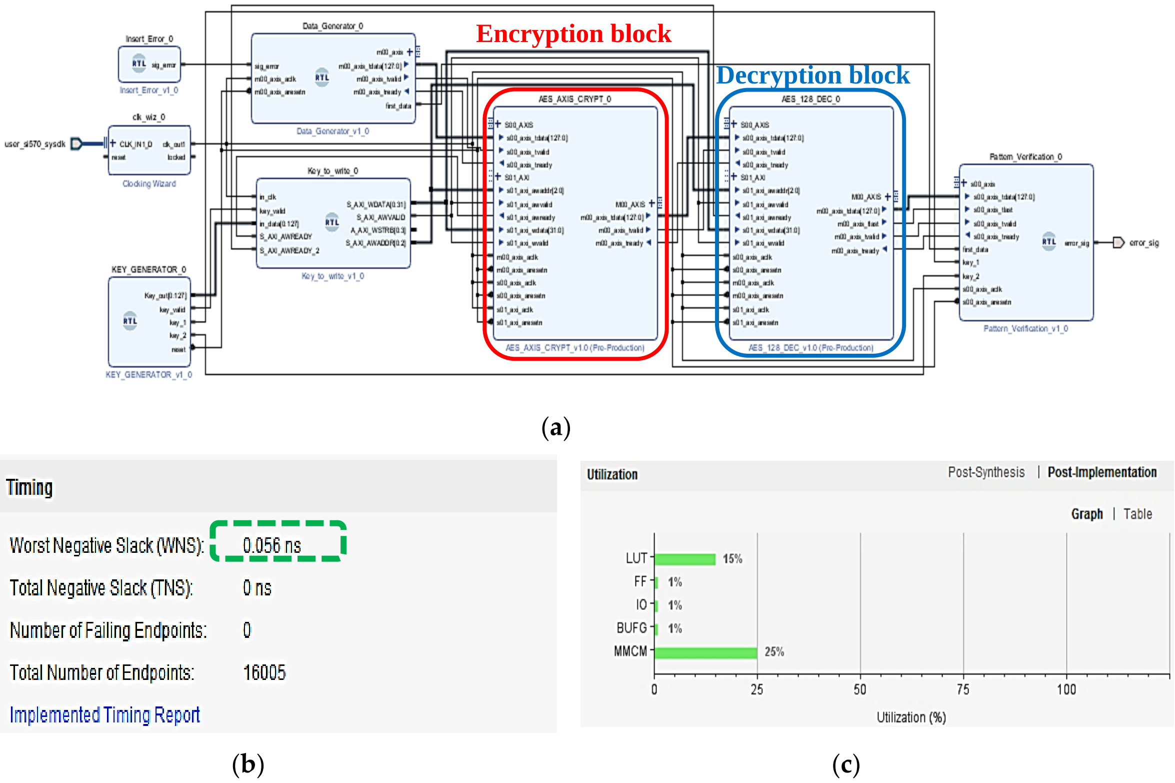


Figure 22. Behavioral simulation with the plaintext data packets (red box) acquired by the system on each rising edge of the clock with frequency 220 MHz and a data rate of 28.16 Gbit/s; the encrypted packets (white box) are obtained after 45.40 ns after receiving them and the decrypted ones (blue box) after 90.80 ns.



(**a**)

Timing

Worst Negative Slack (WNS): 0.056 ns

Total Negative Slack (TNS): 0 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 16005

Implemented Timing Report

(**b**) (**c**)

Figure 23. Block Design including the encryption (red box) and decryption (blue box) blocks connected in cascade (**a**); screenshot of Project Manager, obtained by post-implementation simulation, using a clock frequency of 220 MHz and Explore implementation strategy, showing the positive WNS parameter (green-dashed box) equal to 0.056 ns (**b**) and the hardware usage of the overall system (**c**).

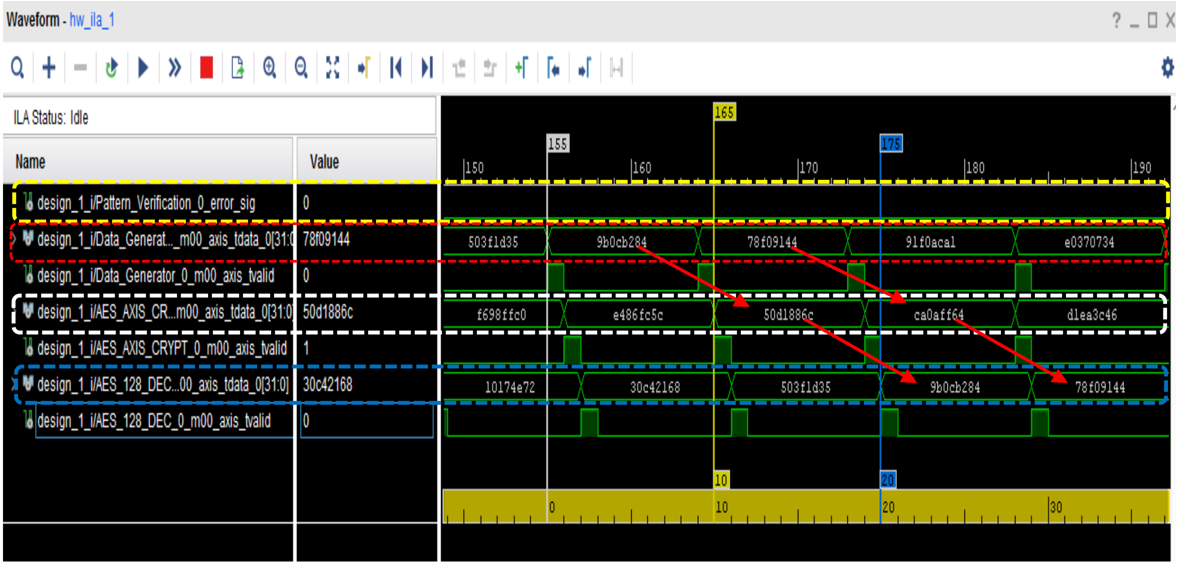
Besides, the estimation of the total on-chip power (sum of the static FPGA power and design power) of the combined encryption/decryption system has been obtained from the post-implementation simulation, providing plaintext data packets each clock period, which is equal to 1.77 W, with 26.7 °C chip temperature, ensuring a thermal margin equal to 73.3 °C (i.e., temperature limit equal to 90 °C).

Furthermore, post-implementation simulations have been carried out on both the encryption and decryption systems individually, so obtaining the total on-chip power consumption equal to 1.17 W and 0.99 W with the chip temperature equal to 26.5 °C and 26.1 °C, respectively. By providing the plaintext data packets in input to the encryption block every 40.86 ns, the post-implementation simulation on the combined encryption/decryption system indicates a power consumption of only 365 mW, with a 25.5 °C chip temperature.

4.2. Testing of the Developed Encryption/Decryption Algorithm on ZCU102 Evaluation Board

After the generation of the bitstream file related to the developed project including the cascade of the encryption and decryption blocks, the file has been loaded on the FPGA-ZCU102 evaluation board. To monitor the interest signals, the IP Integrated Logical Analyzer (IL) has been added to the Block Design; also, to verify the correctness of the decrypted packets, provided by the system constituted by the encryption and decryption blocks connected in cascade, during the test phase, only a single encryption key has been used, initially loaded into four 32-bit registers and subsequently automatically validated; therefore, the error_sig signal produced by the Pattern_Verificator block remains low, thus indicating the errors' absence in the comparison of the packets received by the decryption block and those contained in the Pattern_Verificator table.

The tests carried out on the board confirmed the proper operation of both encryption and decryption algorithms, complying with the operation resulting from the post-implementation simulations reported in the previous paragraph. In Figure 24, the temporal trends related to the complete encryption/decryption system are shown, in which the plaintext data packets, provided every 9.5 clock periods, are accepted by the encryption block (red box) and thus the encrypted packets are delivered to the decryption block (white box), thereby obtaining the decrypted packets downstream (blue box in Figure 24). As expected, the error_sig signal remains low along the observation period, indicating that the processing of the packets is performed correctly, namely the packets leaving the decryption block are equal to those provided at the input to the encryption block.



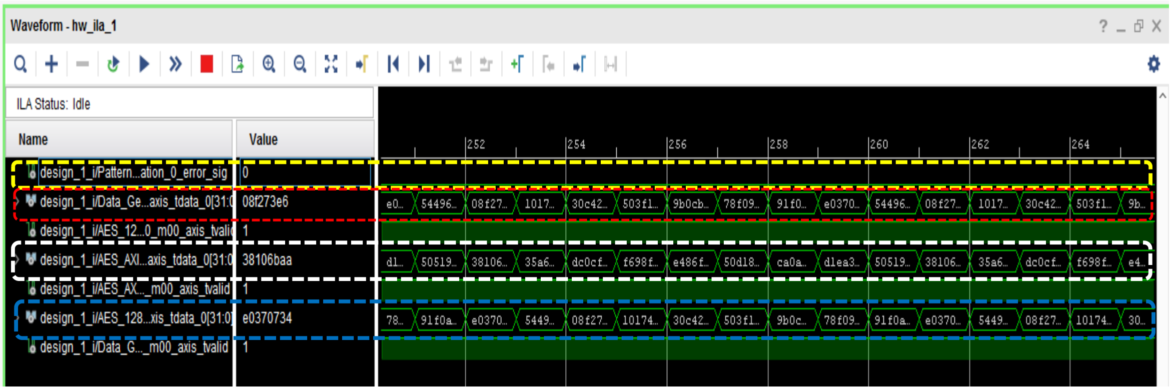
Waveform - hw_ila_1

ILA Status: Idle

Name	Value
design_1_i/Pattern_Verification_0_error_sig	0
design_1_i/Data_Generat..._m00_axis_tdata_0[31:0	78f09144
design_1_i/Data_Generator_0_m00_axis_tvalid	0
design_1_i/AES_AXIS_CR...m00_axis_tdata_0[31:0]	50d1886c
design_1_i/AES_AXIS_CRYPT_0_m00_axis_tvalid	1
design_1_i/AES_128_DEC...00_axis_tdata_0[31:0]	30c42168
design_1_i/AES_128_DEC_0_m00_axis_tvalid	0

Figure 24. Temporal trends with shown the plaintext data packets entering the system (red-dashed box), the encrypted ones delivered by the encryption block to the decryption block (white-dashed box) and finally decrypted packets provided in output by the decryption block (blue-dashed box); as evident, the plaintext data packets provided in input to the system are equal to those provided by the decryption block (as indicated by the red arrow), also demonstrated by error_sig signal, which remains low along the observation period (yellow-dashed box).

In Figure 25, the temporal trends related to the complete encryption/decryption system are shown, in which the plaintext data packets are provided in input on each rising edge of the clock signal; as can be noticed, also, in this case, the error_sig signal remains low, indicating the proper operation of the encryption/decryption system.



Waveform - hw_ila_1

ILA Status: Idle

Name	Value
design_1_i/Pattern...ation_0_error_sig	0
design_1_i/Data_Ge...axis_tdata_0[31:0	08f273e6
design_1_i/AES_12...0_m00_axis_tvalid	1
design_1_i/AES_AXI...axis_tdata_0[31:0	38106baa
design_1_i/AES_AX..._m00_axis_tvalid	1
design_1_i/AES_128...xis_tdata_0[31:0]	e0370734
design_1_i/Data_G..._m00_axis_tvalid	1

252
254
256
258
260
262
264
e0... 54496... 08f27... 1017... 30c42... 503f1... 9b0cb... 78f09... 91f0... e0370... 54496... 08f27... 1017... 30c42... 503f1... 9b...
d1... 50519... 38106... 35a6... dc0cf... f698f... e486f... 50d18... ca0a... d1ea3... 50519... 38106... 35a6... dc0cf... f698f... e4...
78... 91f0a... e0370... 5449... 08f27... 10174... 30c42... 503f1... 9b0c... 78f09... 91f0a... e0370... 5449... 08f27... 10174... 30...

Figure 25. Temporal trends with shown the plaintext data packets provided to the system on each rising edge of the clock; the packets are obtained from the decryption block after 20 clock periods, 10 of which needed for encryption operation and the remaining 10 for the decryption one. The error_sig signal, highlighted in yellow, is low along the whole observation interval, as expected. The packets leaving the decryption block (blue-dashed box) are equal to those entering the encryption block (red-dashed box).

4.3. Comparison of the Proposed AES-128 Implementation with Other Works Reported in the Literature

For the Zynq SoC, just like other FPGA, the PL section is constituted by CLBs arranged according to matrix structure; each CLB contains two slices, each including four LUTs and eight FFs and a configurable switch matrix [37]. Therefore, from the results shown in Tables 1 and 2, the number of CLBs and slices employed by the developed AES-128 encryption and decryption blocks are 1631/3262 and 3464/6928, respectively.

Table 4 reports the comparison between the proposed implementation of AES-128 encryption algorithm with other pipelined implementations previously reported in the scientific literature, similar for operative frequency and supported throughput; also, the platform employed to develop the reported implementations are indicated, since the FPGA technology affects the performance of encryption and decryption. However, the figure of merit chosen for comparing the different implementations is the efficiency, defined as:

$$Efficiency = \frac{Throughput\ [Mbps]}{\#\ of\ used\ slices}. \tag{6}$$

Table 4. Comparison of the proposed AES-128 solution with other FPGA implementations.

Design.	Platform	Frequency [MHz]	Throughput [Gbit/s]	Slices	Efficiency [Mbps/slice]
Zambreno J. et al. [38] (Enc)	XC2V4000	184.1	23.57	16,938	1.39
Fan C.P. et al. [25] (Enc)	XC4VLX200	250.0	32.00	86,806	0.36
Bulens P. et al. [39] (Enc)	Virtex-4	250.0	2.90	1220	2.30
Standaert F. et al. [40] (Enc)	XCV3200E8	145.0	18.56	10,750	1.66
Hodjat A. et al. [41] (Enc)	XC2VP20-7	168.3	21.54	5177	4.16
Kotturi D. et al. [42] (Enc)	XC2VP70-7	232.6	29.77	5408	5.50
Daoud L. et al. [43] (Enc)	XC7Z020	192.0	1.29	431	2.99
Good T. et al. [33] (Enc/Dec)	XC3S2000-5	196.1	23.65	16,693	1.42
Our solution (Enc)	XCZU9EG	220.0	28.16	3262	8.63
Our solution (Enc/Dec)	XCZU9EG	220.0	28.16	10,278	2.74

In particular, this quantity is representative of how efficiently the FPGA hardware resources are used to support a given output throughput.

As evident from the results reported in the following table, the proposed solution can reach high data throughput values (up to 28.16 Gbit/s) but with commensurably lower utilization of the hardware resources compared to other works, thus allowing higher efficiency. Considering the most performing implementation, reported in Reference [42], our solution obtains a maximum data throughput slightly lower (−5.3%) but also employs a lot less FPGA hardware resources (i.e., −39.7%), thus resulting into a higher efficiency value (+56.9%). Also, comparing our solution implementing encryption and decryption operation with those reported in Reference [33], a clear superiority of the former is evident, indicated with a higher efficiency value (+92.9%).

As aforementioned, it must be considered that the comparison shown in the previous table is made between solutions implemented with different platforms for technology, architecture and maximum clock frequency; therefore, the enhanced performances of our solution are also attributable to the advanced features and complex architecture of the used platform but mainly to the implemented solutions aimed to speed up the encryption/decryption process. Such advanced specifications are required to comply with the constraints imposed by the Wireless Connector system, also related to the other functionalities included in the developed communication system. Finally, the platform typology must be considered as a parameter of reported analysis to obtain a fair comparison.

5. Conclusions

In this research work, we have proposed a high-speed implementation of the well-known AES-128 algorithm properly developed for a custom, very short-range and high-frequency communication system, called Wireless Connector; specifically, this last supports high-throughput data transmission on a frequency range around 60 GHz between two mobile stations located at short-range (1–10 m). The core of the communication system is constituted by a Xilinx ZCU102 FPGA platform, which manages all the base-band operations, including the encryption and decryption of the data packets; the prototype of the Wireless Connector was realized, demonstrating its proper operation. In particular, a pipelined approach has been applied to the round-based elaboration typical of the AES algorithm, allowing simultaneous processing of multiple successive plaintext packets each clock period and thus reaching higher data throughput values; furthermore, a 32-bit 16 × 16 Sbox matrix was employed to speed up the Substitute Byte step compared to the classic 8-bit implementation.

Encryption and decryption VHDL blocks have been developed on the Xilinx ZCU102 FPGA platform, carrying out multiple elaborations of the incoming data packets to comply with the 3 Gbit/s data rate, constraint required by the Wireless Connector application. The developed encryption system can operate at a 220 MHz maximum clock frequency, supporting an encryption time of just 10 clock periods. Thanks to the pipelined elaboration, the proposed implementation is able to process and provide the encrypted packets each clock period (namely, $4.54\ \text{ns} = \frac{1}{220\ \text{MHz}}$), reaching a maximum data throughput higher than 28 Gbit/s (i.e., $\frac{128\ \text{bit}/\textit{packet}}{4.54\ \text{ns}} = 28.16\ \text{Gbit/s}$). Similarly, the decrypting system employs just 10 clock period for obtaining the plaintext data packets.

Furthermore, developed AES-128 encryption implementation is featured by higher efficiency (8.63 Mbps/slice) compared to similar solutions operating on the same frequency range, requiring just 1631 CLBs, 13043 LUTs and 3877 FFs. However, the decryption implementation requires higher resource utilization compared to the encryption one (3464 CLBs, 27713 LUTs, 3912 FFs and 1 BUFG), due to the four matrices derived from Inverse SubBytes, Inverse Shift Rows and Inverse Mix Columns operations, each containing 32-bit elements; the greater resource utilization is associated with the Inverse Mix Columns operation, given the multiplicative constants involved in its matrix representation and its LUT-based implementation inside the FPGA, as detailed in the Section 3.2.

Author Contributions: Relatively to the present scientific article: Conceptualization, P.V. and E.V.; methodology, P.V. and E.V.; software, R.d.F.; validation, P.V. and R.V.; investigation, R.d.F. and S.C.; resources, R.d.F. and E.V.; data curation, R.d.F. and S.C.; writing—original draft preparation, P.V., R.d.F. and S.C.; writing—review and

editing, P.V. and R.V.; visualization, R.V.; supervision, R.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Li, L.; Li, S. High throughput AES encryption/decryption with efficient reordering and merging techniques. In *Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Gent, Belgium, 4–6 September 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 1–4.
2. Wei, J.; Han, J.; Cao, S. Satellite IoT Edge Intelligent Computing: A Research on Architecture. *Electronics* **2019**, *8*, 1247. [CrossRef]
3. De Fazio, R.; Cafagna, D.; Marcuccio, G.; Minerba, A.; Visconti, P. A Multi-Source Harvesting System Applied to Sensor-Based Smart Garments for Monitoring Workers' Bio-Physical Parameters in Harsh Environments. *Energies* **2020**, *13*, 2161. [CrossRef]
4. Visconti, P.; de Fazio, R.; Costantini, P.; Miccoli, S.; Cafagna, D. Innovative complete solution for health safety of children unintentionally forgotten in a car: A smart Arduino-based system with user app for remote control. *IET Sci. Meas. Technol.* **2020**, *14*, 665–675. [CrossRef]
5. Rajasekar, P.; Haridas, M. Efficient FPGA implementation of AES 128 bit for IEEE 802.16e mobile WiMax standards. *Circuits Syst.* **2016**, *7*, 371–380. [CrossRef]
6. Denning, D.; Irvine, J.; Harold, N.; Dunn, P.; Devlin, M. An implementation of a gigabit Ethernet AES encryption engine for application processing in SDR. In *Proceedings of the 2004 IEEE 60th Vehicular Technology Conference, VTC2004-Fall. 2004, Los Angeles, CA, USA, 26–29 September 2004*; IEEE: Piscataway, NJ, USA, 2004; Volume 3, pp. 1963–1967.
7. Dey, A.; Nandi, S.; Sarkar, M. Security Measures in IOT based 5G Networks. In *Proceedings of the 2018 3rd International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 15–16 November 2018*; IEEE: Piscataway, NJ, USA, 2018; pp. 561–566.
8. Del-Valle-Soto, C.; Velázquez, R.; Valdivia, L.J.; Giannoccaro, N.I.; Visconti, P. An Energy Model Using Sleeping Algorithms for Wireless Sensor Networks under Proactive and Reactive Protocols: A Performance Evaluation. *Energies* **2020**, *13*, 3024. [CrossRef]
9. Visconti, P.; Sbarro, B.; Primiceri, P.; de Fazio, R.; Ekuakille, A.L. 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. *Int. J. Electron. Telecommun.* **2019**, *65*, 671–678. [CrossRef]
10. Visconti, P.; Sbarro, B.; Primiceri, P. 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. *Int. J. Smart Sens. Intell. Syst.* **2017**, *10*, 793–828. [CrossRef]
11. Long, K.; Leung, V.C.M.; Haijun, Z.; Feng, Z.; Li, Y.; Zhang, Z. *5G for Future Wireless Networks*, 1st ed.; Springer: Beijing, China, 2017; pp. 1–653. ISBN 978-3-319-72822-3.
12. Hejazi, A.; Pu, Y.; Lee, K.-Y. A Design of Wide-Range and Low Phase Noise Linear Transconductance VCO with 193.76 dBc/Hz FoMT for mm-Wave 5G Transceivers. *Electronics* **2020**, *9*, 935. [CrossRef]
13. Ghanim, A.; Alshaikhli, I.; Fakhri, T. Comparative study on 4G/LTE cryptographic algorithms based on different factors. *Int. J. Comput. Sci. Telecommun.* **2014**, *5*, 7–10.
14. Park, J.; Park, Y. Symmetric-Key Cryptographic Routine Detection in Anti-Reverse Engineered Binaries Using Hardware Tracing. *Electronics* **2020**, *9*, 957. [CrossRef]
15. Bellemou, A.M.; García, A.; Castillo, E.; Benblidia, N.; Anane, M.; Álvarez-Bermejo, J.A.; Parrilla, L. Efficient Implementation on Low-Cost SoC-FPGAs of TLSv1.2 Protocol with ECC_AES Support for Secure IoT Coordinators. *Electronics* **2019**, *8*, 1238. [CrossRef]
16. Baldoni, W.M.; Ciliberto, C.; Cattaneo, G.M.P. *Aritmetica, Crittografia e Codici*; Springer: Berlin/Heidelberg, Germany, 2007; ISBN 978-88-470-0456-6.
17. Saggese, G.P.; Mazzeo, A.; Mazzocca, N.; Strollo, A.G.M. An FPGA-Based Performance Analysis of the Unrolling, Tiling and Pipelining of the AES Algorithm. In *Field Programmable Logic and Application*; Cheung, P.Y.K., Constantinides, G.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 292–302.

18. Farooq, U.; Aslam, M.F. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. *J. King Saud Univ. Comput. Inf. Sci.* **2017**, *29*, 295–302. [CrossRef]
19. Xilinx ZCU102 Evaluation Board User Guide. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf (accessed on 16 September 2020).
20. Paar, C.; Pelzl, J. The Advanced Encryption Standard (AES). In *Understanding Cryptography: A Textbook for Students and Practitioners*; Paar, C., Pelzl, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 87–121. ISBN 978-3-642-04101-3.
21. Rahimunnisa, K.; Karthigaikumar, P.; Rasheed, S.; Jayakumar, J.; SureshKumar, S. FPGA implementation of AES algorithm for high throughput using folded parallel architecture. *Secur. Commun. Networks* **2014**, *7*, 2225–2236. [CrossRef]
22. Guzmán, I.; Nieto, R.; Noreña, Á. FPGA implementation of the AES-128 algorithm in non-feedback modes of operation. *DYNA* **2016**, *83*, 37–43. [CrossRef]
23. Noorbasha, F.; Divya, Y.; Poojitha, M.; Navya, K.; Bhavishya, A.; Rao, K.; Kishore, K. FPGA design and implementation of modified AES based encryption and decryption algorithm. *Int. J. Innov. Technol. Explor. Eng.* **2019**, *8*, 132–136.
24. Gopalan, A.; Ganesh, J.; Swathi, M. FPGA-based Message Encryption and Decryption. *Int. J. Innov. Technol. Explor. Eng.* **2015**, *4*, 1225–1232.
25. Fan, C.-P.; Hwang, J.-K. Implementations of high throughput sequential and fully pipelined AES processors on FPGA. In Proceedings of the 2007 International Symposium on Intelligent Signal Processing and Communication Systems, Xiamen, China, 28 November–1 December 2007; pp. 353–356.
26. McLoone, M.; McCanny, J.V. High-performance FPGA implementation of DES using a novel method for implementing the key schedule. *IEE Proc. Circuits Devices Syst.* **2003**, *150*, 373–378. [CrossRef]
27. Chodowiec, P.; Gaj, K. Very compact FPGA implementation of the AES algorithm. In Proceedings of the CHES 2003, Cologne, Germany, 8–10 September 2003; Volume 2779, pp. 319–333.
28. Bani-Hani, R.; Harb, S.; Mhaidat, K.; Taqieddin, E. High-Throughput and Area-Efficient FPGA Implementations of Data Encryption Standard (DES). *Circuits Syst.* **2014**, *5*, 45–56. [CrossRef]
29. Rouvroy, G.; Standaert, F.-X.; Quisquater, J.-J.; Legat, J.-D. Efficient uses of FPGAs for implementations of DES and its experimental linear cryptanalysis. *IEEE Trans. Comput.* **2003**, *52*, 473–482. [CrossRef]
30. Ahmad, N.; Hasan, R.; Jubadi, W.M. Design of AES S-box using combinational logic optimization. In *Proceedings of the 2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA), Penang, Malaysia, 3–6 October 2010*; IEEE: Piscataway, NJ, USA, 2010; pp. 696–699.
31. Canright, D. A Very Compact S-Box for AES. In *Cryptographic Hardware and Embedded Systems–CHES 2005*; Rao, J.R., Sunar, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 441–455.
32. Satoh, A.; Morioka, S.; Takano, K.; Munetoh, S. A Compact Rijndael Hardware Architecture with S-Box Optimization. In *International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 239–254.
33. Good, T.; Benaissa, M. AES on FPGA from the Fastest to the Smallest. In *Cryptographic Hardware and Embedded Systems–CHES 2005*; Rao, J.R., Sunar, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 427–440.
34. Murugeswari, S.; Sridevi, P.; Vanaja, D.; Vanaja, G. Area optimization for reducing circuit complexity in masked AES based on FPGA. *Int. J. Innov. Emerg. Technol.* **2015**, *1*, 1–4.
35. Sutharsan, H.; Thomas, A. Area & Power optimization of AES algorithm using modified mixcolumn with composite S-BOX. *IJRSET* **2016**, *3*, 12–24.
36. Hua, L.; Friggstad, Z. An efficient architecture for the AES mix columns operation. In *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems, Kobe, Japan, 23–26 May 2005*; IEEE: Piscataway, NJ, USA, 2005; Volume 5, pp. 4637–4640.
37. Xilinx UltraScale Architecture Configurable Logic Block User Guide (UG574). Available online: https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf (accessed on 26 August 2020).
38. Zambreno, J.; Nguyen, D.; Choudhary, A. Exploring Area/Delay Tradeoffs in an AES FPGA Implementation. In *Field Programmable Logic and Application*; Becker, J., Platzner, M., Vernalde, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 575–585.

39. Bulens, P.; Standaert, F.-X.; Quisquater, J.-J.; Pellegrin, P.; Rouvroy, G. Implementation of the AES-128 on Virtex-5 FPGAs. In *Progress in Cryptology–AFRICACRYPT 2008*; Vaudenay, S., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 16–26.
40. Standaert, F.-X.; Rouvroy, G.; Quisquater, J.-J.; Legat, J.-D. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In *Cryptographic Hardware and Embedded Systems—CHES 2003*; Walter, C.D., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 334–350.
41. Hodjat, A.; Verbauwhede, I. A 21.54 Gbits/s fully pipelined AES processor on FPGA. In *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, CA, USA, 20–23 April 2004*; IEEE: Piscataway, NJ, USA, 2004; pp. 308–309.
42. Kotturi, D.; Seong-Moo, Y.; Blizzard, J. AES crypto chip utilizing high-speed parallel pipelined architecture. In *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems, Kobe, Japan, 23–26 May 2005*; IEEE: Piscataway, NJ, USA, 2005; Volume 5, pp. 4653–4656.
43. Daoud, L.; Hussein, F.; Rafla, N. Optimization of Advanced Encryption Standard (AES) Using Vivado High Level Synthesis (HLS). In Proceedings of the 34th International Conference on Computers and Their Applications, Honolulu, HI, USA, 18–20 March 2019; Volume 58, pp. 36–44.