International Journal of Difference Equations (IJDE).
ISSN 0973-6069 Volume 18, Number 1 (2023), pp. 73-89

https://www.ripublication.com/ijde.htm

Hardware/Software co-design for CLEFIA Lightweight Encryption Algorithm Implemented on a Programmable System-On-Chip

Laura Quiroga C., Holman Montiel A. and Fredy Martínez S.

Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Abstract

Currently, the increase in the number of technological devices that are constantly connected to the Internet; makes it necessary to use security techniques that guarantee the integrity and confidentiality of the information. This leads to the search for portable solutions that can be implemented in low-cost electronic devices with reduced processing capacity compared to a conventional PC. The versatility of some embedded systems that structurally have reconfigurable hardware favors the implementation of cryptographic processes that increase the protection of such information. This work describes the implementation of the CLEFIA lightweight encryption algorithm on a PSoC® 5LP with ARM Cortex®-M3 processor; through a hardware/software co-design analysis and detailed knowledge of the functional structure of the cryptographic algorithm. Universal Digital Block Arrays (UDB) of the PSoC are used to optimize and improve the response times associated with the operation of the respective algorithm. A detailed description of the most relevant elements at the time of carrying out the hardware implementation on the PSoC is made. The validation of the correct operation of the proposed solution is carried out by comparing the response times associated with the execution only in software versus the proposed solution; making use of the hardware/software co-design technique in which the CPU is freed from the computational load corresponding to the CLEFIA exchange function.

Keywords: Lightweight cryptography; CLEFIA; hardware/software co-design; System on chip.

1. INTRODUCTION

According to the World Population Prospects report prepared by the United Nations in

2019, the total population is 7.7 million inhabitants ([1]) of which approximately 4.5 million are Internet users ([2]). Due to this and to the fact that access to the network is public, the volume of data generated is in a high state of vulnerability, which is why the integrity of information, and its confidentiality has become one of the priorities of organizations and individuals.

In turn, with the incursion of the Internet of Things (IoT) in areas such as health ([3]), transportation, logistics, environment, manufacturing, and agriculture, among others ([4]); the connection between devices allows data to be collected and information to be exchanged from various sources and fields; since today a large part of the activities have been automated to improve processes, increase efficiency and the quality of life of those who use these devices. On the other hand, one of the challenges that IoT shows is information security; since sometimes the platforms do not have the necessary protocols to identify and neutralize the latent threats to which the data is exposed ([5],[6]).

Consequently, to increase information security levels, one of the methods implemented is cryptography, which has been used since ancient times to protect significant data by encrypting and decrypting messages ([7]). However, conventional cryptography presents some limitations in the processing requirements which demand large memory size, high processing time, high performance, and energy consumption. ([8]). Thus, lightweight cryptography emerges as an alternative to meet the demands present in devices that have limited resources, and its application is aimed at hardware and software platforms ([9],[10]).

Therefore, the options devised for the protection of information are booming and the solutions presented; seek through light cryptography to increase the required security levels such as confidentiality, data integrity, authentication, availability, and privacy ([11],[12]). Among the research developed, one implementation focuses on the physical layer of IoT; where the construction of a scheme for device authentication and generation of communication keys employing frequency hopping was elaborated ([13]). There have been implementations that make use of a hash function with a sponge construction approach, to increase security in the transmission of information over vehicular ad-hoc networks (VANET); which are used in intelligent transportation systems for the exchange of information between vehicles and vehicular infrastructure. Through the use of OBU (On Board Unit) devices that collect data to be sent to the RSU (Road-Side Unit) that aim to monitor and manage the status and traffic on the road. ([14]).

Likewise, systems have been proposed to increase the security of D2D communication in a 5G IoT environment; where D2D communication is a device-to-device connection without an intermediate node that has three procedures: device detection, connection configuration, and data transmission. However, these processes present a high degree of security vulnerability since there is no procedure for device validation, nor does it use encryption for message confidentiality; therefore, cryptographic algorithms have been applied such as digital signature, Diffie-Hellman key exchange algorithm, and AEAD lightweight encryption ([15]).

In other applications, security has been evaluated for IoT devices, wearables, and RFID tags; where PRESENT, AES, ECDH, DH, and RSA algorithms were analyzed by

comparing their robustness and performance. After the hardware implementation and cryptographic analysis, to find the one that presents an optimal performance according to the requirements of the device ([16]). In addition, the NVLC (New Variant Lightweight Cryptography Algorithm) symmetric block cipher algorithm was used in RFID identification tags and sensors; to increase the encryption complexity and information security through a substitution-permutation network within the algorithm. ([17]).

On the other hand, through the development and evaluation of the cryptographic hash function (CHF) combined with the OTK (One-Time Key) algorithm, the effective encryption of the garbage collection memory is obtained; supporting the NMAPA (Next Memory Address Occupation Attack) recognized as one of the riskiest cyber-attacks of today ([18]).

In contrast, from the wide variety of existing algorithms, CLEFIA is selected as the focus of the research since it is a lightweight block cipher algorithm whose block length is 128 bits and a key length of 128, 192 or 256 bits can be used ([19]). This is a cryptographic mechanism that provides security for devices with limited capabilities, providing a high-level of security since it has high resistance to at least 19 known attacks and high performance ([20]).

For this reason, this document presents the use of the CLEFIA lightweight encryption algorithm on a PSoC embedded system; taking advantage of the mixed development characteristics that allow programming its hardware and software components with the purpose of reducing the output times in the information encryption and decryption processes. Section II describes the basic elements necessary for the development of the hardware implementation on PSoC and the structural description of the CLEFIA encryption algorithm. Section III describes the implementation of the co-design with emphasis on the requirements necessary for the integration of operational modules created in Verilog. Section IV presents the results obtained, and finally, the conclusions and future work are presented in Section V.

2. METHODOLOGY

A. SoC architecture: Cypress PSoC

The implementation is developed on a PSoC microcontroller that is a highly configurable chip for embedded control design; as shown in Figure 1, it is composed of a bus (microprocessor, system memory, programming, debug, and test subsystem), system resources, and an analog and digital system that allows a high level of integration in a wide variety of applications; thanks to the customization and flexible routing of peripheral functions.

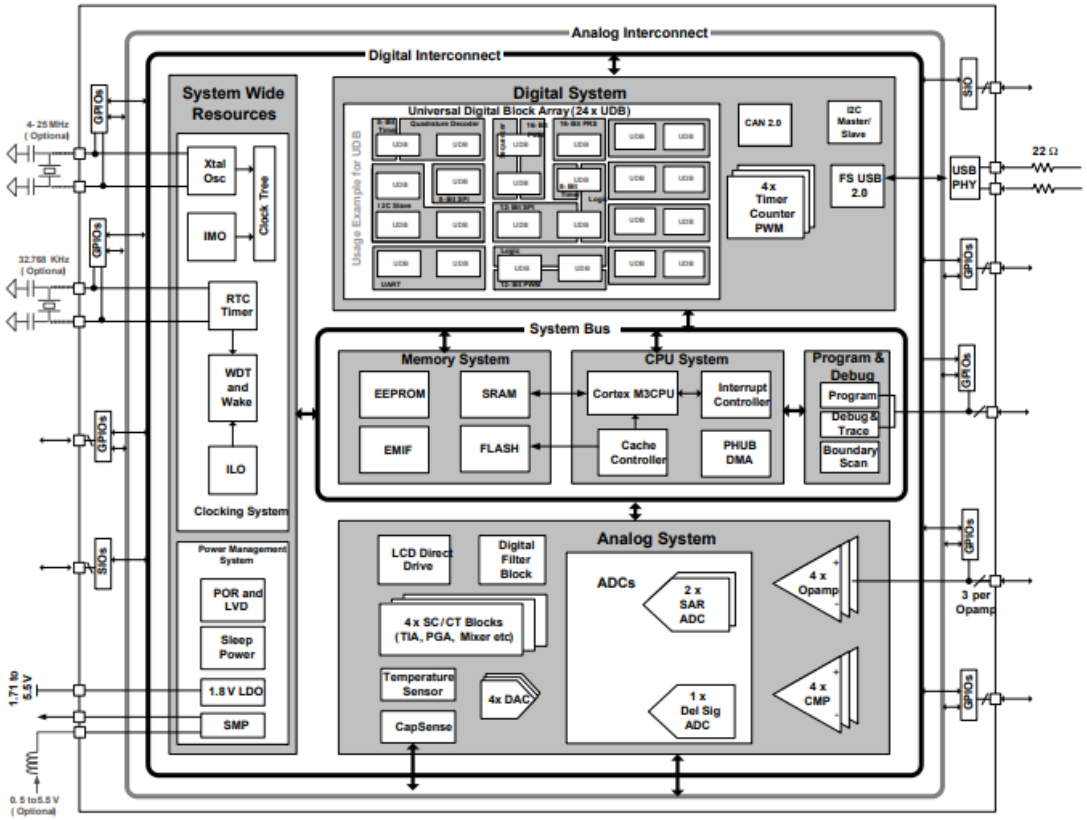

Figure 1: Simplified Block Diagram PSoC ([21]).

Moreover, this architecture contributes to the reduction of card space, design time, system cost, and power consumption. It improves performance through the ARM Cortex-M3 Core which is a low-power 32-bit CPU ([21]).

Additionally, the microprocessor employs the SysTick timer, which is a falling counter that is part of the Cortex M3 of the PSoC 5LP, with a 24-bit resolution that is synchronized with the system clock. The counter generates an interrupt when the given time elapses and the counter is reloaded; the interrupt is available as part of the Nested Vectored Interrupt Controller (NVIC) for CPU service and can be employed for the main purpose of timing control in user code ([22]). See Table 1.

Table 1: Systick Timing Functions.

Function	void CySysTickInit(void)	void CySysTickEnable(void)	void CySysTickSetReload(uint32 value)	uint32 CySysTickGetValue(void)	void CySysTickClear(void)
Description	Initializes the callback addresses with pointers to null and associates the SysTick system vector with the function responsible for calling the callback, sets the timer to generate an interrupt every 1 ms.	Enables the SysTick timer and its interruption.	Sets the value at which the counter is placed at startup and after reaching zero.	Gets the current value of the SysTick counter.	Removes the SysTick counter for a well-defined start. This function is used if you change the SysTick setting in the reload value or timer clock source. The function is called as part of the execution of CySysTickStart().
Parameters	None	None	Counter reset value. Valid range. [0x0-0x00FFFFFFFF].	None	None
Return value	None	None	None	Returns the value of the SysTick counter.	None

Side Effects and Restrictions	Removes the SysTick count indicator if it was set. The interrupt interval of 1 ms is set according to the frequency determined by PSoC Creator at compilation. If the system clock frequency is changed at runtime, the CyDelayFreq() function must be called with the appropriate parameter to ensure the actual frequency used in the SysTick reload value calculation.	Removes the SysTick count indicator if it was set.	None	None	None

A fundamental part in the development and implementation process of the proposed solution is based on the unique characteristics of the PSoC® 5LP microcontroller; since this microcontroller, in addition to the mixed hardware it has, has a configurable matrix of Universal Digital Block Arrays (UDB). These blocks, being distributed in a programmable interconnection matrix, allow a flexible reprogramming supported on a routing interconnection bus, thus facilitating the integration of elements or codes developed in Verilog. See Fig. 2.

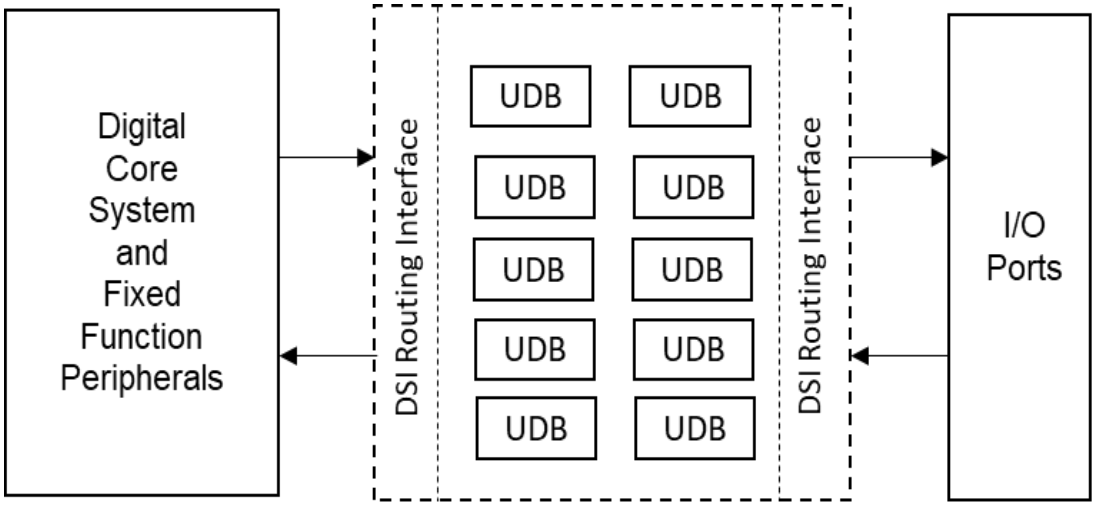

Figure 2: Digital Programmable Architecture of the PSoC ([23]).

B. Cryptographic algorithm structure

In accordance with ISO/IEC 29192-2, which standardizes the CLEFIA algorithm, the structure of the algorithm is presented in three parts: the building blocks, the data processing, and the encryption system.

- CLEFIA building blocks
- Generalized Feistel network GFN_{d,r}

Initially, the function GFN_{d,r} is established, which acts as the main structure for the algorithm, later a function is defined for data processing and another for processing the encryption system. The algorithm employs a four (4) branch generalized Feistel network that is dedicated to the processing function and encryption for 128-bit keys.
The branch of the Feistel network (d) and the round (r) used in CLEFIA is called GNF_{d,r}. For (d) 32-bit pairs, the inputs are defined as Zi, the outputs Vi (0 <= i < d) and dr/2 the 32-bit round keys RK_{i} ($0 \leq i < dr/2$), GFN_{d,r} (d = 4.8) is defined as follow in the Table 2.

Table 2: Generalized Four-Branch Network ([24])

Four-Branch Network GFN_ {4, r}	
ACTIVITY	DESCRIPTION
Set function	GFN_ {4, r} (RK_ {0},..., RK_{2r-1}, Z0, Z1, Z2, Z3)
Determine inputs	32-bit round key RK_ {0},..., RK_{2r-1},
	32-bit data Z0, Z1, Z2, Z3
Determine outputs	32-bit data V0, V1, V2, V3
Step One	A0 | A1 | A2 | A3 < - Z0 | Z1 | Z2 | Z3
Step Two	For i = 0 to r-1 do:
	A1 <- A1 XOR F0 (RK_{2i}, A0)

	A3 <- A3 XOR F1 (RK_{2i+1}, A2)
	A0 | A1 | A2 | A3 <- A1 | A2 | A3 | A0
Step Three	V0 | V1 | V2 | V3 <- A3 | A0 | A1 | A2

In turn, the inverse function for the four-branch network GFNINV_{4,r} is achieved by modifying the order of RK_{i} and the direction of rotation of the words in steps two and three in GFN_{4,r}.

- **Functions F**

The algorithm employs two functions F, F0 and F1 which are used in GFN_{4,r} and are defined as follow in the Table 3 and Table 4.

Table 3: F0 Function ([24])

F0 Function	
ACTIVITY	DESCRIPTION
Set function	F0 (RK,z)
Determine inputs	32-bit round key RK, 32-bit data z
Determine outputs	32-bit data v
Step One	A <- RK XOR z
Step Two	A = A0 | A1 | A2 | A3 where Ai is an 8-bit data,
	A0 <- S0 (A0),
	A1 <- S1 (A1),
	A2 <- S0 (A2),
	A3 <- S1 (A3)
Step Three	v = v0 | v1 | v2 | v3, where vi is an 8-bit data,
	v <- M0 trans ((A0, A1, A2, A3))

Table 4: F1 Function ([24])

F1 Function	
ACTIVITY	DESCRIPTION
Set function	F1 (RK,z)
Determine inputs	Round key 32-bit RK, 32-bit data z
Determine outputs	32-bit data v
Step One	A<- RK XOR z
Step Two	A = A0 | A1 | A2 | A3 where Ai is an 8-bit data,
	A0 <- S1 (A0),
	A1 <- S0 (A1),
	A2 <- S1 (A2),
	A3 <- S0 (A3)
Step Three	v = v0 | v1 | v2 | v3, where vi is an 8-bit data,
	v <- M1 trans ((A0, A1, A2, A3))

On the other hand, each F function uses two S-boxes which are used in different order and additionally employs a different diffusion matrix.

- **S Boxes**

The CLEFIA algorithm applies two classes of 8-bit nonlinear S-boxes, where S0 is based on four four-bit S-boxes and S1 is based on the inverse function GF (2^8). In the S-boxes the output values are exposed in hexadecimal form, so in an input containing 8 bits for the S-box, the upper four bits point to the row and the lower four bits to the column. See Table 5.

Table 5: Hexadecimal Values S1 Box ([24])

S1 Box																
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	6c	da	c3	e9	4e	9d	0a	3d	b8	36	b4	38	13	34	0c	d9
1.	bf	74	94	8f	b7	9c	e5	dc	9e	7	49	4f	98	2c	b0	93
2.	12	eb	cd	b3	92	e7	41	60	e3	21	27	3b	e6	19	d2	0e
3.	91	11	c7	3f	2ª	8e	a1	bc	2b	c8	c5	0f	5b	f3	87	8b
4.	fb	f5	de	20	c6	a7	84	ce	d8	65	51	c9	a4	ef	43	53
5.	25	5d	9b	31	e8	3e	0d	d7	80	ff	69	8ª	ba	0b	73	5c
6.	6e	54	15	62	f6	35	30	52	a3	16	d3	28	32	fa	aa	5e
7.	cf	ea	ed	78	33	58	9	7b	63	c0	c1	46	1e	df	a9	99
8.	55	4	c4	86	39	77	82	ec	40	18	90	97	59	dd	83	1f
9.	9ª	37	6	24	64	7c	a5	56	48	8	85	d0	61	26	ca	6f
a.	7e	6ª	b6	71	a0	70	5	d1	45	8c	23	1c	f0	ee	89	Ad
b.	7ª	4b	c2	2f	db	5ª	4d	76	67	17	2d	f4	cb	b1	4ª	a8
c.	b5	22	47	3ª	d5	10	4c	72	cc	0	f9	e0	fd	e2	fe	Ae
d.	f8	5f	ab	f1	1b	42	81	d6	be	44	29	a6	57	b9	af	f2
e.	d4	75	66	bb	68	9f	50	2	1	3c	7f	8d	1ª	88	bd	ac
f.	f7	e4	79	96	a2	fc	6d	b2	6b	3	e1	2e	7d	14	95	1d

- **Diffusion matrixes**

The product between the diffusion matrices M0 or M1 and a vector A are elaborated with the method presented below in Tables 6 and 7.

Table 6: M0 Matrix ([24])

Diffusion matrix M0
v = M0 trans ((A0, A1, A2, A3))
v0 = XOR A0 (0x02 * A1) XOR (0x04 * A2) XOR (0x06 * A3)
v1 = (0x02 * A0) XOR A1 XOR (0x06 *A2) XOR (0x04 * A3)
v2 = (0x04 * A0) XOR (0x06 * A1) XOR A2 XOR (0x02 * A3)
v3 = (0x06 * A0) XOR (0x04 * A1) XOR (0x02 * A2) XOR A3

Table 7: M1 Matrix ([24])

Diffusion matrix M1
v = M1 trans ((A0, A1, A2, A3))
v0 = XOR A0 (0x08 * A1) XOR (0x02 * A2) XOR (0x0a * A3)
v1 = (0x08 * A0) XOR A1 XOR (0x0a * A2) XOR (0x02 * A3)
v2 = (0x02 * A0) XOR (0x0a * A1) XOR A2 XOR (0x08 * A3)
v3 = (0x0a * A0) XOR (0x02 * A1) XOR (0x08 * A2) A3 XOR

It should be noted that the asterisk symbol (*) expressed in the equations indicates the product in GF (2^8) defined through the polynomial b^8 + b^4 + b^3 + b^2 + 1. The constants 0x02, 0x04, 0x06, 0x08, and 0x0a are represented in hexadecimal form of finite field polynomials.

- **CLEFIA data processing**

The data processing algorithm for encryption and decryption is based on the generalized Feistel network GFN_{4,r}.

- **Encryption and Decryption**

For the encryption and decryption process, the 128-bit plaintext (P) and ciphertext (C) are determined, which are divided into 32-bit blocks resulting in P = P0 | P1 | P2 | P3 and C = C0 | C1 | C2 | C3, values to which a 32-bit round key (W) based on RK_{i} and provided by the encryption system is subsequently applied. See Tables 8 and 9.

Table 8: Encryption Function ([24])

Encr Encryption Function	
ACTIVITY	DESCRIPTION
Step One	A0 | A1 | A2 | A3 <- P0 | (P1 XOR WK0) | P2 | (P3 XOR WK1)
Step Two	A0 | A1 | A2 | A3
	<- GFN_ {4, r} (RK_ {0},..., RK_ {2r-1}, A0, A1, A2, A3)
Step Three	C0 | C1 | C2 | C3 <- A0 | (A1 XOR WK2) | A2 | (A3 XOR WK3)

Table 9: Decryption Function ([24])

Decr Decryption Function	
ACTIVITY	DESCRIPTION
Step One	A0 | A1 | A2 | A3 <- C0 | (C1 XOR WK2) | C2 | (C3 XOR WK3)
Step Two	A0 | A1 | A2 | A3
	<- GFNINV_ {4, r} (RK_ {0},..., RK_ {2r-1}, A0, A1, A2, A3)
Step Three	P0 | P1 | P2 | P3 <- A0 | (A1 XOR WK0) | A2 | (A3 XOR WK1)

- **Number of Rounds**

The algorithm uses several rounds r of 18, 22 and 26 for 128-bit, 192-bit and 256-bit keys respectively, likewise, the number of RK_{i} obey the key length and the data processing requires 36, 44 and 52 round keys corresponding to 128-bit, 192-bit and 256-bit keys.

- **Encryption System**

CLEFIA in its encryption system applies 128-bit, 192-bit and 256-bit keys, in turn, Wki outputs (0 <= i <4) and RK_ {j} round keys (0 <= j <2r) for data processing.

- **DoubleSwap function**

The DoubleSwap function for the encryption system is defined as follows:

$$V = Sigma(Z) \tag{1.1}$$

$$V = Z[7-63] \mid Z[121-127] \mid Z[0-6] \mid Z[64-120] \tag{1.2}$$

Given the above, Z [ab] indicates a bit string cut from bit a to bit b of Z where zero is the most significant bit.

The encryption system generates round keys where K is the initial key and L an intermediate key, for this, initially, the intermediate 128-bit key L is generated from K employing the function GFN {4,12}, formed by 24 32-bit constant values where CON_128 [i] (0 <= i < 24) is determined as round keys and K = K0 | K1 | K2 | K3 as initial key. Subsequently, K and L are intended to generate Wki (0 <i <= 4) and RK_ {j} (0 <= j <36) 36 32-bit constant values CON_128 [i] (24 <= i <60), the steps are illustrated below in Table 10.

Table 10: Generation and Expansion of L and K ([24])

Generation of Key L from K	
ACTIVITY	DESCRIPTION
Step One	L ← GFN_ {4,12} (CON_128 [0],..., CON_128 [23], K0,..., K3)
Expansion of L and K	
Step Two	WK0 | WK1 | WK2 | WK3 ← K
Step Three	For i = 0 to 8 do:
	A ← L XOR (CON_128 [24 + 4i] | CON_128 [24 + 4i + 1])
	| CON_128 [24 + 4i + 2] | CON_128 [24 + 4i + 3])
	L ← Sigma (L)
	If i is odd: A ←A XOR K
	RK_{4i} | RK_ {4i + 1} | RK_ {4i + 2} | RK_ {4i + 3} ← A

3. IMPLEMENTATION CO-DESIGN

Implementing a co-design technique involves full knowledge of architecture and modular structure of the cryptographic algorithm ([25],[26]). Detailed planning of each

one of the operative blocks must be carried out; to determine at which moments, they will be executed and how the actions of each element involved in the subsystems used will be synchronized. The design process starts by elaborating a UDB component with Verilog; these components are formed by uncommitted PLDs within the internal architecture of the microcontroller, see Fig. 3. In general, each UDB is formed by a data path module, control, and status registers, and a 7-bit counter module. The high degree of adaptability of the created elements allows the development of multiple work logics; allowing not only to have direct communication with the CPU; but also facilitating, if necessary, communication processes with the different peripherals or hardware components of the PSOC ([27]).

art Page | PLayer.v | main.c

```
13  `include "cypress.v"
14  //`#end` -- edit above this line, do not edit this line
15  // Generated on 04/21/2020 at 01:44
16  // Component: PLayer
17  module PLayer (
18      input   clock
19  );
20      parameter param_19 = 0;
21
22  //`#start body` -- edit after this line, do not edit this line
23
24      //8 buses de 8 bits para las conexiones entre los registros de control y status
25      wire [63:0] D;
26      wire [63:0] P_Layer;
27      wire [63:0] P_Layer_i;
28      wire [63:0] Data_out;
29      //bits de control para realizar el P layer o el P layer inverso (bit 0)
30      wire [7:0]  Control;
31
32      assign P_Layer   = {    D[63],D[59],D[55],D[51],D[47],D[43],D[39],D[35],
33                              D[31],D[27],D[23],D[19],D[15],D[11],D[7], D[3],
34                              D[62],D[58],D[54],D[50],D[46],D[42],D[38],D[34],
35                              D[30],D[26],D[22],D[18],D[14],D[10],D[6], D[2],
36                              D[61],D[57],D[53],D[49],D[45],D[41],D[37],D[33],
37                              D[29],D[25],D[21],D[17],D[13],D[9], D[5], D[1],
38                              D[60],D[56],D[52],D[48],D[44],D[40],D[36],D[32],
39                              D[28],D[24],D[20],D[16],D[12],D[8], D[4], D[0]      };
40
```

Figure 3: Structural definition of the element created in Verilog.

For this implementation case, a completely independent operating block was created, which is called whenever necessary within the functional structure of the cryptographic algorithm. The communication process between the CPU and the UDB is carried out directly. To transmit information from the CPU to the UDB, the Control Register is used; the reading or writing speed of this register is generally defined by the system bus clock. There may be specific cases in which it can work in synchronization mode or pulse mode; it all depends on the functional requirements of the end user ([28],[29]). For the transmission of information between the UDB and the CPU, the Status Register is used. This register allows monitoring the changes of the hardware signal states. A

remarkable function of the Status Register is that it can make use of maskable interrupts. The size associated with the above registers is 8 bits, see Fig. 4.

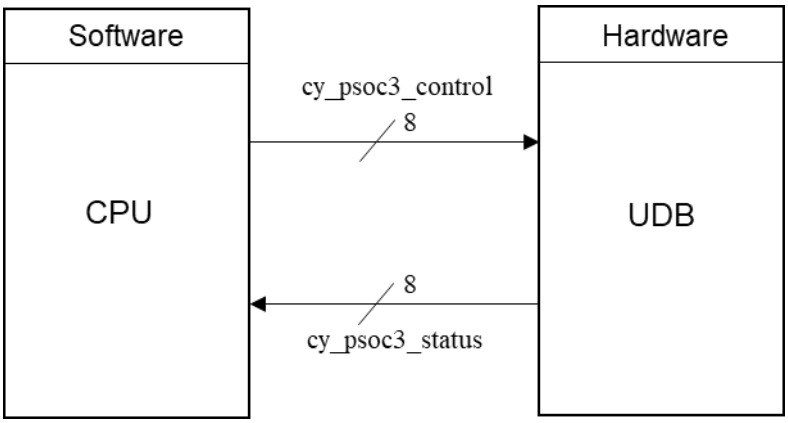

Figure 4: Interconnection buses used.

When instantiating the communication registers in Verilog, the operational requirements must be considered according to the respective process. In the case of the control register, this was worked in a basic way without optional parameters; only the following were used: cy_init_value, which is responsible for giving the initial value of the register during configuration, and cy_force_order, which is a Boolean flag that determines the importance of the order of the bits in the register; in this case, it must be true. For the status register; the option explained above is set for cy_force_order and in the requirement cy_md_select, the transparent mode is defined for each of the bits in the register.

As the registers used are of a specific size of 8 bits, they must be called as many times as necessary to complete the required size; for this case, the optimization will be performed on the exchange function of the cryptographic algorithm CLEFIA with a length of 128 bits. The module must be called 16 times; considering the respective order of the packets and data bus, synchronized with the respective system bus clock. An example of the structure used can be seen in Fig. 5.

```
cy_psoc3_control #(.cy_init_value (8'b00000000), .cy_force_order(`TRUE)) //Default mode
ByteIn1(      // name of the control register
    .control(D[15:8]),  //
    .clock(clock)
);
//componentes PSoC registros de salida (status registers)
cy_psoc3_status #(.cy_force_order(`TRUE), .cy_md_select(8'b00000000)) //Default mode
ByteOut0(      // name of the control register
    .status( Data_out[7:0]),  // output bus [7:0] 'outp'
    .clock(clock)
);
```

Figure 5: Example of communication register configuration.

4. RESULTS AND ANALYSIS

The initial validation of the correct operation of the CLEFIA cryptographic algorithm is initially carried out by adapting the operating structure provided by SONY ([19]). The manufacturer provides an operating structure based on 8-bit processors; to validate and carry out the optimization and improvement process, the code structure must be adapted for 32-bit processors. The implementation was carried out on the PSoC® 5LP development system with an ARM Cortex®-M3 processor; initially, the test vectors provided by the manufacturer were used as a reference. For the key we used ffeeddcc bbaa9988 77665544 33221100 and as plain text 00010203 04050607 08090ª0b 0c0d0e0f; obtaining as cipher text result DE2BF2FD 9B74AACD F1298555 459494FD.

The process of verifying the effectiveness of the CLEFIA algorithm encryption was checked by sending the final ciphertext vector through the serial communication port of the PSOC. In addition to the resulting vector, the time spent in the encryption process was measured. This measurement was obtained using the internal timers and their SysTick functions. See Fig. 6.

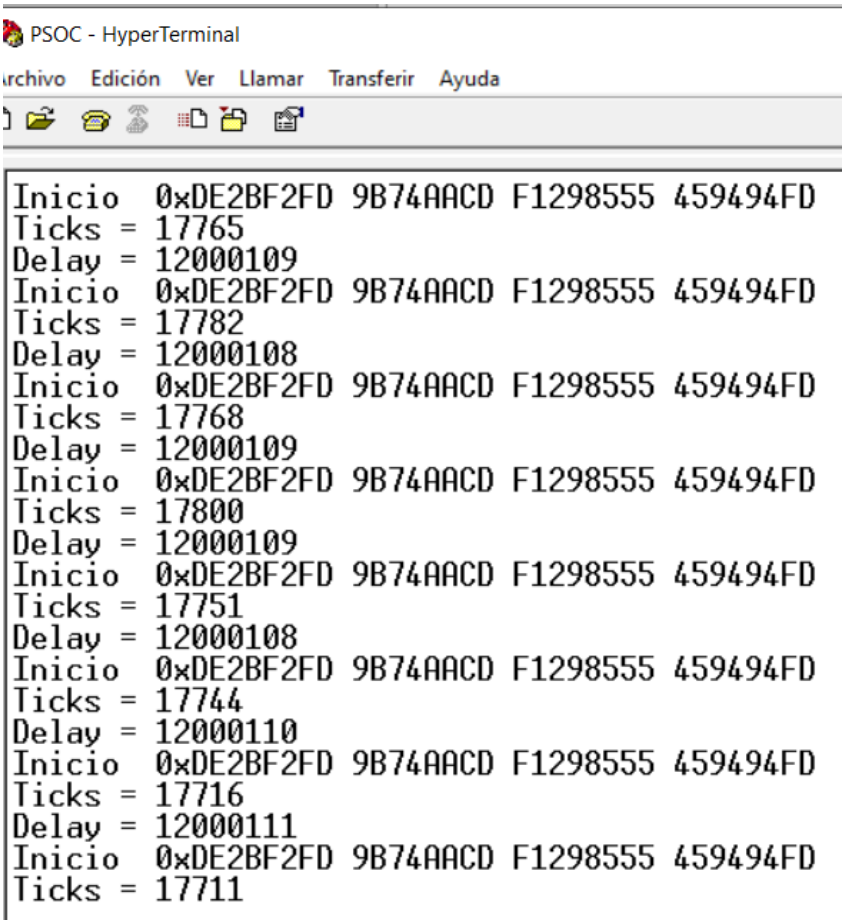

Figure 6: Full algorithm timing test.

The average time taken for the complete execution of the algorithm in software structure was 740 microseconds. Once the structural modification of the hardware component was performed using the Swap function block with UDB; the total execution time measurement was 234,43 microseconds. Thus, demonstrating a decrease in the processing time of 68.32% compared to the complete software implementation. Concerning the flash memory, an improvement of 42.80% of program memory was obtained based on the optimization performed in the implementation on the same device.

5. CONCLUSIONS AND FUTURE WORK

This document presents clearly and simply the key concepts that must be considered for the development of embedded cryptography applications. Thanks to the routing flexibility of the UDBs in the PSoC architecture, it is possible to create a fully functional design of the CLEFIA algorithm. All this is to take advantage of the processing potential of this 32-bit embedded system. It is thus demonstrated that by handling and knowing the complete structure of the cryptographic algorithm; it was possible to obtain a considerable decrease in the encryption time compared to a total software solution.

The deepening of the internal architecture of embedded systems allows the development of low-cost functional applications applicable to different environments of the productive sector. It should be emphasized that for the case of the selected device; data paths become the core for solutions with UDB; allowing gradual scalability in terms of the number of bits that may be required in a task associated with the word size of the processor used.

As future work, it is intended to develop specific hardware applications that allow digital signatures. Similarly, it is intended to perform the optimization for the implementation, validation, and comparison of lightweight encryption algorithms standardized by ISO/IEC 29192-2:2019; which are: PRESENT, CLEFIA, and LEA.

REFERENCES

[1]. United Nations, Department of Economic and Social Affairs, Population Division "World Population Prospects 2019: Highlights", pp. 141, 2019.

[2]. We Are Social Inc. and Hootsuite Inc., "Digital 2020 - Global Digital Overview," 2020.

[3]. B. D. Deebak, F. Al-turjman, M. Aloqaily, and O. Alfandi, "An Authentic-Based Privacy Preservation Protocol for Smart e-Healthcare Systems in IoT," IEEE Access, vol. 7, pp. 135632–135649, 2019.

[4]. V. Rao and K. V. Prema, "A review on lightweight cryptography for Internet of Things based applications," Journal of Ambient Intelligence and Humanized Computing, vol. 12, no. 9, pp. 8835–8857, 2021.

[5]. V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures," in IEEE Access, vol. 7, pp. 82721–82743, 2019.

[6]. S. K. Mousavi, A. Ghaffari, S. Besharat, and H. Afshari, "Security of internet of things based on cryptographic algorithms: a survey," Wireless Networks, vol. 27, no. 2, pp. 1515–1555, 2021.
[7]. W. Willems and I. Gutiérrez, Una introducción a la criptografía de clave pública, Ediciones Uninorte. Barranquilla, Colombia, 2010.
[8]. W. J. Buchanan, S. Li, and R. Asif, "Lightweight cryptography methods," Journal Cyber Security Technology, vol. 1, no. 3–4, pp. 187–201, 2017.
[9]. S. Sadkhan and A. Salman, "A survey on lightweight-cryptography status and future challenges," 2018 International Conference on Advance of Sustainable Engineering and its Application (ICASEA), pp. 105–108, 2018.
[10]. J. Zhang, D. Zhou, X. Qiu, and R. Lai, "Hardware / Software Partitioning Algorithm under Multi-Constraints for the Optimization of Power Consumption," Current Trends in Computer Science and Mechanical Automation, vol. 2, pp. 578–589, 2017.
[11]. V. A. Thakor, M. Razzaque, and M. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," in IEEE Access, vol.9, pp. 28177–28193, 2021.
[12]. Y. Harbi, Z. Aliouat, A. Refoufi, and S. Harous, "Recent Security Trends in Internet of Things: A Comprehensive Survey," in IEEE Access, vol. 9, pp. 113292–113314, 2021.
[13]. Y. Jiang, A. Hu, and J. Huang, "A lightweight physical-layer based security strategy for Internet of things," Cluster Computing, vol. 22, pp. 12971–12983, 2019.
[14]. P. Manickam, K. Shankar, E. Perumal, M. Ilayaraja, and K. Sathesh, "Secure Data Transmission Through Reliable Vehicles in VANET Using Optimal Lightweight Cryptography," in Cybersecurity and Secure Information Systems. Advanced Sciences and Technologies for Security Applications, Springer, Cham, pp. 193–204, 2019.
[15] . B. Seok, J. C. S. Sicato, T. Erzhena, C. Xuan, Y. Pan, and J. H. Park, "Secure D2D communication for 5G IoT network based on lightweight cryptography," Applied Sciences, vol. 10, no. 1, pp. 1–14, 2020.
[16]. T. Kumar, V. Sahula, and D. Kumawat, "Energy Efficient Lightweight Cryptography Algorithms for IoT Devices," IETE Journal Research, pp. 1722-1735, 2019.
[17]. S. Abd Al-Rahman, A. Sagheer, and O. Dawood, "NVLC: New Variant Lightweight Cryptography Algorithm for Internet of Things," 2018 1st Annual International Conference on Information and Science (AiCIS), pp. 176–181, 2018.
[18]. M. Khalifa, F. Algarni, M. Ayoub, A. Ullah, and K. Aloufi, "A lightweight cryptography (LWC) framework to secure memory heap in Internet of Things," Alexandria Engineering Journal, vol. 60, no. 1, pp. 1489–1497, 2021.
[19]. Sony Corporation, "Sony Global – CLEFIA: The 128-bit Blockcipher," 2007.
[20]. C. Coronel, "Comparación del rendimiento y nivel de seguridad en algoritmos criptográficos ligeros PRESENT, CLEFIA, KECCAK y HIGHT: Una revisión sistemática.," 2018.

[21]. Cypress Semiconductor Corporation, “PSoC ® 5LP: CY8C58LP Family Datasheet Programmable System-on-Chip (PSoC®) General Description,” 2019.
[22]. Cypress Semiconductor Corporation, “PSoC ® CreatorTM System Reference Guide,” 2014.
[23]. Cypress Semiconductor Corporation, “PSoC ® CreatorTM System Component Author Guide,” 2016.
[24]. M. Katagi and S. Moriai, “The 128- Bit Blockcipher CLEFIA.” pp. 1–33, 2011.
[25]. A. Iguider, K. Bousselam, O. Elissati, M. Chami and A. En-Nouaary, “Heuristic algorithms for multi-criteria hardware/software partitioning in embedded systems codesign,” Computers & Electrical Engineering, vol. 84, 2020.
[26]. F. Boutekkouk, “Embedded systems codesign under artificial intelligence perspective: a review,” International Journal of Ad Hoc and Ubiquituos Computing, vol. 32, no. 4, 2019.
[27]. H. Liang, Z. Wang, D. Roy, S. Dey, S. Chakraborty and Q. Zhu, “Security-driven codesign with weakly-hard constraints for real-time embedded systems,” 2019 IEEE 37th International Conference on Computer Design (ICCD), pp. 217-226, 2019.
[28]. N. Govil and S. Roy, “High performanceand low cost implementation of fast Fourier Transform Algorithm based on Hardware Software Co-design,” 2014 IEEE Region 10 Symposium, pp. 403-407, 2014.
[29]. M. Azzaz, A. Maali, R. Kaibou, I. Kakouche, M. Saad and H. Hamil, “FPGA HW/SW codesign approach for real-time image processing using HLS,” 2020 1st International Conference on Communications, Control Systems and Signal Processing (CCSSP), pp. 169-174, 2020.