Indian Journal of Science and Technology, Vol 10(15), DOI: 10.17485/ijst/2017/v10i15/111162, April 2017
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

High Performance Optimization Function for 32-Bits Microcontrollers in Key Scheduling of the Lightweight Cipher Algorithm CLEFIA

Edwar Jacinto, Holman Montiel* and Fernando Martinez

Technological Faculty, District University Francisco Jose de Caldas, Bogota D.C., Colombia;
ejacintog@udistrital.edu.co, hmontiela@udistrital.edu.co, fmartinezs@udistrital.edu.co

Abstract

Objectives: This paper shows an optimized code for light-weight cipher algorithms, attempting to keep the balance between the use of resources and the communication speed. **Methods/Analysis**: A real performance analysis is applied to the cryptographic algorithm CLEFIA, under the standards by ISO/IEC 29192-2, by means of a code optimization for key scheduling through bit-oriented instructions. It is used the Freescale KL25Z development board for the measure of response times and the structural blocks' execution times for the cipher algorithm. **Findings**: In this paper a bit-level optimization was sought over some operative structures of the algorithm, taking advantage of the 32-bit architecture in the development platform, generating this way a better response time for the application and an increase of the Throughput performance regarding the reference code by SONY. **Novelty/Improvement:** This application was developed so it can be used by many platforms into any electronic application, which requires an encryption process, where the use of a PC is not worthy because of the size and cost.

Keywords: Cipher Algorithm, ISO/IEC 29192, Lightweight Algorithm

1. Introduction

The development of electronic prototypes related to communication processes establish the need for using techniques guaranteeing the information treatment through basic tools and complex mechanisms, offering in this way a security level for specific information[1]. At present can be appreciated a great number of developments based on embedded systems[2,3] solutions applied to low cost and high performance technologies[4], where a variety of developments are implemented, using sturdiness and practicality from the several boards available in the tech market.

These developments need the implementation of standardized cryptographic techniques which allow adding a level of reliability regarding the communication tasks; nevertheless in many occasions, adapting these cipher algorithms carries an analysis[5] and development processes very complex[6], where not always the benefits of the embedded system architecture are not taken advantage of.

In this paper, the intention is to use one of the cipher algorithms standardized by ISO/IEC 29192-2[7], which specifies two suitable block ciphers[8,9] for applications that require light-weight cryptographic implementation. It will be handled the CLEFIA light-weight algorithm optimization with a key-length of 128 bits[9,10].

It is proposed to perform a functional analysis of the cipher algorithm CLEFIA and to take as reference all of the information given by its developer, SONY, to create a code library, which can be executed on 32 bits embedded systems. The optimization of the code is intended to be done through an architecture analysis and the use of bit-oriented instructions, provided by the development compiler of the mbed platforms in the Freescale KL25Z. The mbed compiler provides an interface with C and C ++ languages. The compiler is designed to create,

*Author for correspondence

compile and download projects who can run on a mbed microcontroller. There's no need to install any additional software to create projects, because it is a web app which can be connected from any place and allows he storage of online projects.

2. Proposed Work

The proposal is based on a code library generation which can be used by any 32 bits embedded system, for the key scheduling part (128-bit) used in the CLEFIA encryption process, taking into account the round number that entails the structure, this part provides whitening keys and round keys for the data processing part. The key scheduling part consists of two steps: Generating and expanding as per the Figure 1. This lineal operation references to the round keys associated to permutations and exchanges occurred during the encryption. It seeks to maintain the software efficiency taking advantage of the hardware architecture, based on the premise of the word size or basic structure of the microcontroller used, relaying on an appropriated handle of the rotation and basic logic operations about the type of the variable used during the process and the hardware general capacity of processing.

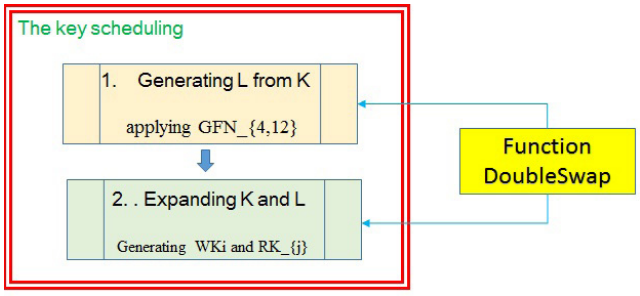

Figure 1. Key schedule – Components.

Figure 2 shows the flowchart of the proposal developed to optimize the code structure of the DoubleSwap function, used by the two operative blocks of the key scheduling.

3. Implementation and Results

3.1 Key Scheduling for CLEFIA

This is an structural part of the algorithm who takes care of the key generation used for the encryption process; all of this based on the DoubleSwap function, which uses a 128 bits input vector divided by 4 parts that are exchanges attempting to increase the algorithm security.

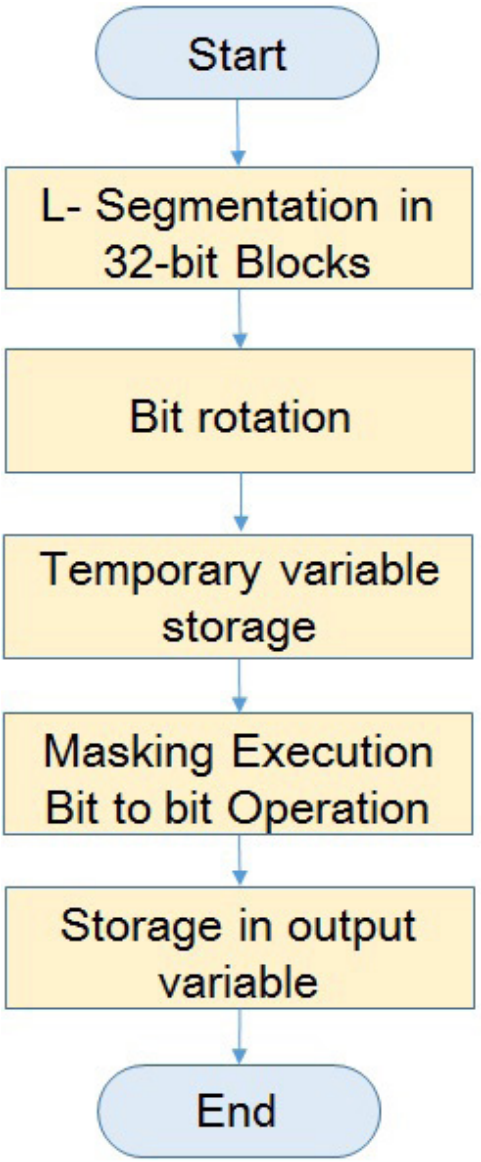

Figure 2. Flowchart code optimization for DoubleSwap Function.

3.1.1 Function DoubleSwap

The DoubleSwap function shown in the Figure 3 which is used in the encryption process is defined by:

$Y = \Sigma(X)$

$Y = X[7\text{-}63] \mid X[121\text{-}127] \mid X[0\text{-}6] \mid X[64\text{-}120]$

Where X [ab] denotes a bit-string cut from the X's a-bit to the X's b-bit. Where the 0 bit is the most significant bit and has a length of 128 bits.

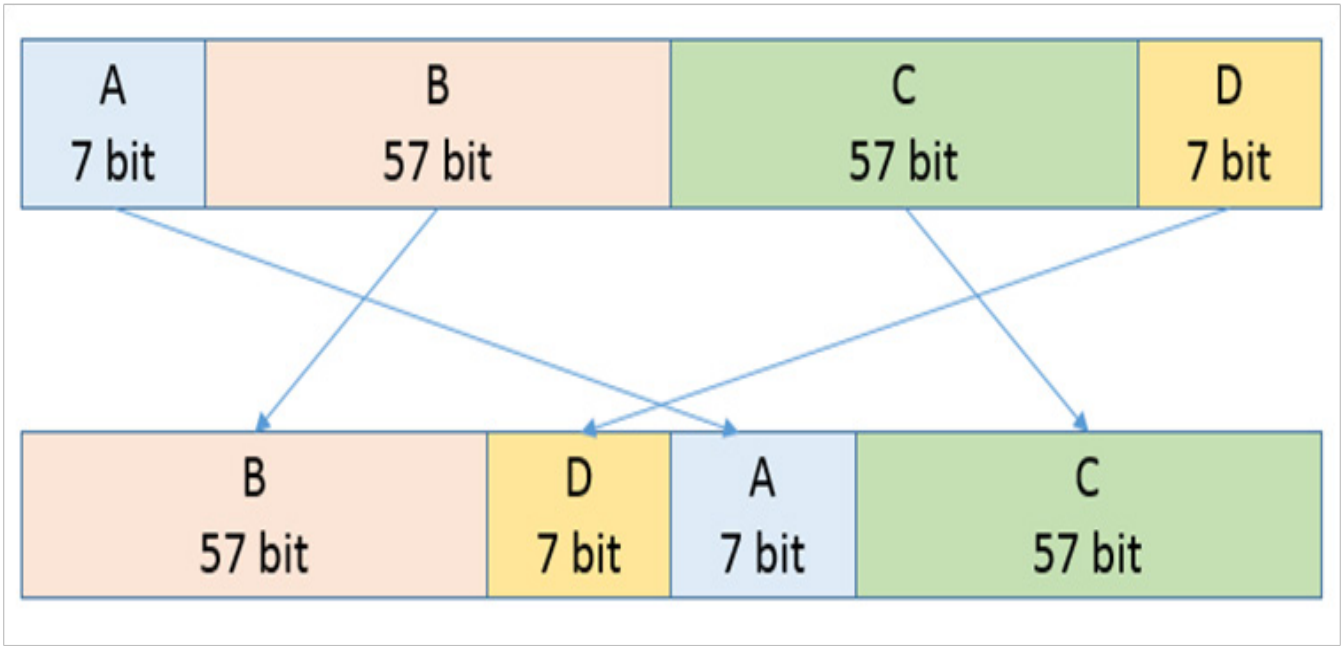

Figure 3. DoubleSwap Function.

Table 1. Round keys by L

WK_0	WK_1	WK_2	WK_3	$\leftarrow K$	
RK_0	RK_1	RK_2	RK_3	$\leftarrow L \oplus$	$(CON_{24}^{(128)} \mid CON_{25}^{(128)} \mid CON_{26}^{(128)} \mid CON_{27}^{(128)})$
RK_4	RK_5	RK_6	RK_7	$\leftarrow \sum L \oplus K \bigoplus$	$(CON_{28}^{(128)} \mid CON_{29}^{(128)} \mid CON_{30}^{(128)} \mid CON_{31}^{(128)})$
RK_8	RK_9	RK_{10}	RK_{11}	$\leftarrow \sum_{11}{}^{\uparrow 2} L \oplus$	$(CON_{32}^{(128)} \mid CON_{33}^{(128)} \mid CON_{34}^{(128)} \mid CON_{35}^{(128)})$
RK_{12}	RK_{13}	RK_{14}	RK_{15}	$\leftarrow \sum_{11}{}^{\uparrow 3} L \oplus K \oplus$	$(CON_{36}^{(128)} \mid CON_{37}^{(128)} \mid CON_{38}^{(128)} \mid CON_{39}^{(128)})$
RK_{16}	RK_{17}	RK_{18}	RK_{19}	$\leftarrow \sum_{11}{}^{\uparrow 4} L \oplus$	$(CON_{40}^{(128)} \mid CON_{41}^{(128)} \mid CON_{42}^{(128)} \mid CON_{43}^{(128)})$
RK_{20}	RK_{21}	RK_{22}	RK_{23}	$\leftarrow \sum_{11}{}^{\uparrow 5} L \oplus K \oplus$	$(CON_{44}^{(128)} \mid CON_{45}^{(128)} \mid CON_{46}^{(128)} \mid CON_{47}^{(128)})$
RK_{24}	RK_{25}	RK_{26}	RK_{27}	$\leftarrow \sum_{11}{}^{\uparrow 6} L \oplus$	$(CON_{48}^{(128)} \mid CON_{49}^{(128)} \mid CON_{50}^{(128)} \mid CON_{51}^{(128)})$
RK_{28}	RK_{29}	RK_{30}	RK_{31}	$\leftarrow \sum_{11}{}^{\uparrow 7} L \oplus K \oplus$	$(CON_{52}^{(128)} \mid CON_{53}^{(128)} \mid CON_{54}^{(128)} \mid CON_{55}^{(128)})$
RK_{32}	RK_{33}	RK_{34}	RK_{35}	$\leftarrow \sum_{11}{}^{\uparrow 8} L \oplus$	$(CON_{56}^{(128)} \mid CON_{57}^{(128)} \mid CON_{58}^{(128)} \mid CON_{59}^{(128)})$

Source. RFC 6114

3.2 General Structure

The CLEFIA cipher system is based on the round keys generation for the encrypted data system, according to this, starting from K as main key, it must be generated an intermediate key named L.

The 128 bits L intermediate key is generated by the function GFN{4,12} (Generalized Feistel Network), that carries 24 32 bits constant values, CON_128 [i] (0 < = i<24), as round keys and K = K0 | K1 | K2 | K3 as the input key. Then, K and L are used to generate WK[i] (0 <i< = 4) and RK[j] (0 < = j <36) constant values CON_128 [i] (24 < = i<60) as in the following steps.

Generating L from K	
Step 1	$L \leftarrow GFN_{4,12}(CON_0^{(128)},\ldots,CON_{23}^{(128)},K0,\ldots,K3)$
Expanding K and L	
Step 2	WK0|WK1|WK2|WK3← K
Step 3	For i = 0 to 8 do: $T \leftarrow L \oplus \left(CON_{24+4i}^{(128)} \middle\| CON_{24+4i+1}^{(128)} \middle\| CON_{24+4i+2}^{(128)} \middle\| CON_{24+4i+3}^{(128)}\right)$ $T \leftarrow L \oplus (CON_{24+4i}^{(128)} \| CON_{24+4i+1}^{(128)} \| CON_{24+4i+2}^{(128)} \| CON_{24+4i+3}^{(128)})$ $L \leftarrow \sum(L)$ if i is odd: T←T⊕K $RK_{4i} \| RK_{4i+1} \| RK_{4i+2} \| RK_{4i+3} \leftarrow T$

The generation of the intermediate key L, is vital for the algorithm execution, because is associated to each one of the respective rounds of the algorithm encryption process. A direct link between the round key value and the process round number is presents. Table 1 shows the relation between the generation of the respective round key and the permutation performed with the DoubleSwap function[11].

3.3 Results and Discussion

The optimization of the DoubleSwap function takes advantage of the 32 bits architecture, mainly focused on bit-oriented instructions linked to the handle of masks which facilitate and reduce execution times per block of operations. The creation of a specific function that take care of the permutation process through the use of temporal arrays and bit rotations will be related directly to the Throughput increase of the general algorithm. A comparative frame emerges from the study proposal given by SONY as developer[9], where it establishes a code structure based on bits rotation on arrays of Char type variables; these arrays have a length of 16 positions of memory where each specific segment has 8 bits. It is used a sequential operation scheme over the memory positions of the permutation array, which entails that only 2 memory positions at the time can be worked on, initiating a continuous process from the position (0 – 1) to (15 – 14).

The performance optimization process for the mentioned function starts from the change of type for the work variable; the next code shows the optimized function. The function works only with three arrays type Integer with only four 32 bits memory positions each one, which are directly handled through comparative masks and bit-oriented instructions, taking full advantage of the length established by the microcontroller architecture, with no need of additional functions for the copy or output data assignation as SONY uses in their reference codes.

DoubleSwap Function Optimized Code

```
Voiddoubleswap(void){
temp_data[0]= L[0]<<7;
temp= L[1] >> 25;
out[0]= temp_data[0] | temp;                    // 1er  block
temp_data[1]= L[1] << 7;
temp_data[3]= L[3] & 0x7f;
out[1]=temp_data[1]|temp_data[3];      // 2do  block
temp = L[2] >> 7;
out[2] = ( temp  | (L[0] & 0xfe000000)); //3er  block
temp_data[2]= L[2] << 25;
temp = L[3] >> 7;
out[3] = temp_data[2]|temp;                    //4to  block

L[0]= out[0];
L[1]= out[1];
L[2]= out[2];
L[3]= out[3];
}
```

This modifications allowed to obtain an increase of the Throughput performance, from 145,45 Kbps to 250 Kbps and a reduction of execution times, from 880 µseg to 512 µseg. These measurements could be done directly on the microcontroller thanks to the use of Timer type elements using the functions start () and stop (), guaranteeing a total control on the validation and verification activities for the key permutation processes and the complete execution of the cipher algorithm.

4. Conclusion

It was obtained a code structure applicable to any 32 bits embedded system, which optimizes the key scheduling, improves the Throughput performance and reduce the execution times for the CLEFIA cipher algorithm structural blocks, proving this way that taking appropriate advantage of the structure potential of a embedded system can accomplish performance improvements of applications in the environments where there is no need to use the computational power of a conventional PC to guarantee the information safety.

5. Acknowledgments

This work was supported by the District University Francisco Jose de Caldas, in part through CIDC and partly by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by District University. The authors thank the research groups ARMOS and DIGITI for the evaluation carried out on prototypes of ideas and strategies.

6. References

1. Aysu A, Gulan E, Shaumont P. Simon says: Break area records of block ciphers on FPGA's. IEEE Embedded Systems Letters. 2014; 6(2):37–40. Crossref
2. Bragadeesh S, Umamakeswari A. Secure data aggregation for Wireless Sensor Network using lightweight cryptography. Indian Journal of Science and Technology. 2016; 9(48):1–6. Crossref
3. Sasi SW, Sivanandam N. A survey on cryptography using optimization algorithms in WSNs. Indian Journal of Science and Technology. 2015 Feb; 8(3):216–21. Crossref
4. Hong D, Sung J, Hong S, Lim J, Lee S, Koo B, Lee C, Chang D, Lee J, Jeong K, Kim H, Kim J, Chee S. HIGHT: A new block cipher suitable for low-resource device. Cryptographic Hardware and Embedded Systems – CHES. Berlin Heidelb, Springer. 2006; 4249:46–59.
5. Isha, Luhach AK. Analysis of lightweight cryptographic solutions for Internet of Things. Indian Journal of Science and Technology. 2016 Jul; 9(28):1–7.
6. Gunasekaran G, Bimal K. Encrypting and decrypting image using computer visualization techniques. ARPN Journal of Engineering and Applied Sciences. 2014 May; 9(5):646–50.
7. International Standard ISO/IEC Information Technology - Security Techniques. Lightweight Cryptography. 2012.
8. Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A, Robshaw MJ, Seurin Y, Vikkelsoe C. Present: An ultra-lightweight block cipher. Berlin Heidelb: Springer; 2007. p. 450–66.
9. Shirai T, Shibutani K, Akishita T, Moriai S, Iwata T. The 128-bit block cipher CLEFIA (Extended Abstract). Berlin Heidelb: Springer; 2007. p.181–95.
10. Internet Engineering Task Force. RFC6114. The 128-bit Block cipher CLEFIA. 2011.
11. Montiel H, Jacinto E, Martínez F. Implementation of light-weight encryption algorithm based on 32-bit embedded systems. International Journal of Applied Engineering Research. 2016; 11(23):11409–13.