Fişa suspiciunii de plagiat / Sheet of plagiarism's suspicion	Indexat la: 49/06

Opera suspicionată (OS) Suspicious work	Opera autentică (OA) Authentic work

OS	MANG, E., MANG, I. An FPGA-based implementation of the key transformation procedure in the MARS algorithm. *Journal of Computer Science and Control Systems*. **3**(1), 2010, 119-122.
OA	* * *, MARS Encryption, Disponibil la: http :// www . tropsoft . com / strongenc / mars . htm.

Incidenţa minimă a suspiciunii / Minimum incidence of suspicion	
p.119:01s – p.119:04s	p.02:33 – p.02:35
p.119:19s – p.119:31s	p.02:35 – p.02:45
p.119:42d - p.119:47d	p.02:45 – p.02:47

An FPGA-based implementation of the key transformation procedure in the MARS algorithm

MANG Erica, MANG Ioan
University of Oradea, Romania,
Department of Computer Science, Faculty of Faculty of Electrical Engineering and Information Technology,
Universitatii St.1, 410087 Oradea, E-Mail1: emang@dec-it.eu, imang@decnet.ro

Abstract *– MARS is a block cipher designed by IBM as a candidate algorithm for the Advanced Encryption Standard (AES). It has been selected as one of the five finalists in the AES competition. It was designed to meet and exceed the requirements for a standard for shared-key encryption in the next few decades. The main theme behind the design of MARS is to get the best security/performance tradeoff by utilizing the strongest tools and techniques available today for designing block ciphers. As a result, MARS provides a very high level of security, combined with much better performance than other existing ciphers. In this paper we present a hardware implementation of key expansion procedure using VHDL from Xilinx Software package on FPGA board.*

Keywords: *AES, MARS, key expansion, cryptography, FSM*

I. INTRODUCTION

MARS has a very high level of security and is much more efficient than many older algorithms. The algorithm is resistant to all known shortcut attacks. Its design takes advantage of the powerful capabilities of modern computers to allow a much higher level of performance than can be obtained from less optimized algorithms such as DES. MARS is unique in that it combines virtually every design technique known to cryptographers in one algorithm. But it does so in a way that is easy to analyze so that the properties of the algorithm are more completely understood, decreasing the probability that any loopholes exist which might weaken the algorithm.

MARS is a shared-key block cipher, with a block size of 128 bits and a variable key size, ranging from 128 to over 400 bits. The MARS cipher uses a variety of operations on 32-bit words; it combines exclusive-ors, additions, subtractions (used to "mix together" data values and key values), table look-ups (MARS uses a single table of 512 32-bit words, called the S-box. Sometimes the S-box is viewed as two tables, each of 256 entries), multiplications (there are used "native multiplications" modulo 232, in conjunction with data-dependent rotations, to obtain very high security), and both fixed and data-dependent rotations (can be performed quickly in software and hardware and very effective against linear and differential cryptanalysis) [4]. MARS is designed to take advantage of the powerful operations supported in today's computers, resulting in a much improved security/performance tradeoff over existing ciphers.

MARS is symmetric of encryption and decryption. MARS is design to be as secure against chosen ciphertext attacks as against chosen plaintext attacks. This dictates making the cipher very symmetric, so the last half of the rounds are almost a "mirror image" of the first half.

MARS is design to work with 32 bit words At the current state of the technology, this choice provides a good tradeoff between the ability to run the algorithm on computers that are available today, and the ability to take advantage of larger word-size in future architectures.

MARS used Type-3 Feistel network. Since MARS has a block length of 128 bits and word-size of 32 bits, it follows that each block consists of four words. Among the various network-structures that are capable of handling four words in a block, it seems that a type-3 Feistel network provides the best tradeoff between speed, strength and suitability for analysis. A type-3 Feistel network consists of many rounds, where in each round one data word (and a few key words) is used to modify all the other data words. Compared with a type-1 Feistel network (where in each round one data word is used to modify one other data word), this construct provides much better diffusion properties with only a slightly added cost. Hence, fewer rounds can be used to achieve the same strength. Additionally, a type-3 Feistel network has advantages over structures in which several data words are used "at once" to modify other data words, in that these structures are typically much harder to analyze (and hence, much more prone to design errors). The reason is that in such structures the analysis must take into account all the possible combinations of values for the input data words, which quickly leads to unmanageable complexity.

The MARS cipher consists of a mixture of two very different structures, so that an attack that works against one portion of the algorithm wouldn't be able to affect the other. This mixed structure is likely to provide better resistance against new, as yet undiscovered attacks that might be used in the future.

II. THE MARS ALGORITHM

MARS takes as input (and produces as output) four 32-bit data words. The cipher itself is word-oriented, in that all the internal operations are performed on 32-bit words, and hence the internal structure is endian-neutral (i.e., the same code works on both little-endian and big-endian machines). When the input (or output) of the cipher is a byte stream, we use little endian byte ordering to interpret each four bytes as one 32-bit word.

The general structure of the cipher is depicted in Figure 1 [4]. The cipher consists of a "cryptographic core" of keyed transformation, which is wrapped with two layers providing rapid key avalanche. The first phase provides rapid mixing and key avalanche, to frustrate chosen-plaintext at-tacks, and to make it harder to "strip out" rounds of the cryptographic core in linear and differential attacks. It consists of addition of key words to the data words, followed by eight rounds of S-box based, unkeyed type-3 Feistel mixing (in "forward mode"). The second phase is the "cryptographic core" of the cipher, consisting of sixteen rounds of keyed type-3 Feistel transformation. To ensure that encryption and decryption have the same strength, we perform the first eight rounds in "forward mode" while the last eight rounds are performed in "backwards mode". The last phase again provides rapid mixing and key avalanche, this time to protect against chosen-ciphertext attacks. This phase is essentially the inverse of the first phase, consisting of eight rounds of the same type-3 Feistel mixing as in the first phase (except in "backwards mode"), followed by subtraction of key words from the data words. MARS is IBM submission to AES. Below we describe the cipher in details. In this description we use the following notations: is an array of 4 32-bit data words [3]. Initially D contains the plaintext words, and at the end of the encryption process it contains the ciphertext words.

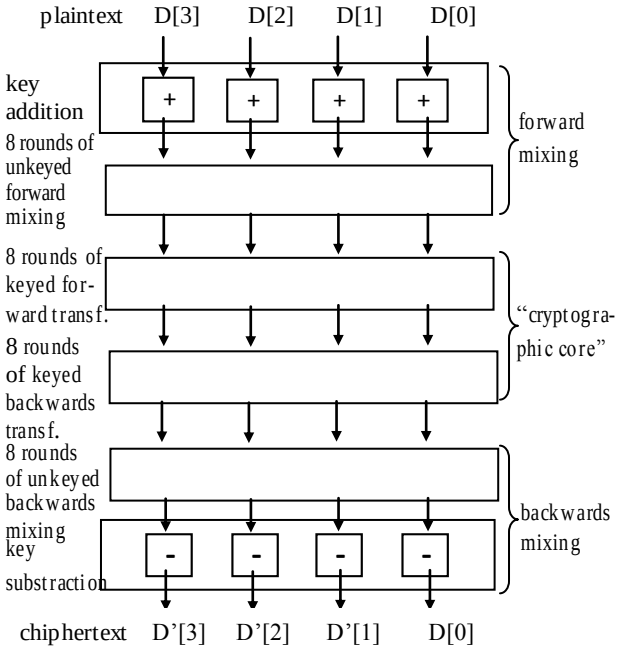

Figure 1. High-level structure of the chiper

III. KEY EXPANSION PROCEDURE

The key expansion procedure expands a given key array k [], consisting of n 32-bit words (where n is any number between 4 and 14) into an array K [] of 40 words. The original key is not required to have any structure (in particular, the key does not include any parity bits). In addition, the key expansion procedure also guarantees that the key words that are used for multiplication in the encryption procedure have the following properties: (a) The two lowest bits in a key word that is used for multiplication are set to 1; (b) None of these key words contains either ten consecutive 0's or ten consecutive 1's.

The MARS key expansion procedure expands the user-supplied key ranging from 4 to 14 words into a 40-word key for use in the encryption/decryption operation. The key expansion procedure consists of three steps:

1. the original key material is copied into a temporary table T[] of 15 words, followed by the number of words n, and 0's.
   $T[0\ldots n-1] = k[0\ldots n-1]$, $T[n] = n$; $T[n+1\ldots 14] = 0$
2. the following process is repeated four times, where each iteration computes the next ten words of the expanded key:
   a. The array T[] is transformed using the following linear formula:
      for i=0,…,14, $T[i]=T[i]\oplus((T[i-7 \bmod 15]\oplus T[i-2 \bmod 15]) <<< 3)\oplus(4i+j)$
      where j is the iteration number;
   b. the array T[] is stired using four rounds of type-1 Feistel network. Specifically, the operation is repeted four times
      $T[i]=(T[i] + S[\text{low 9 bits of } T[i-1 \bmod 15]]) <<< 9$, i = 0,1,…,14
   c. 10 of the words in T[] are taken and reordered into the next ten words of the expanded key array, K[]. This is done by setting
      $K[10j + i] = T[4i \bmod 15]$, i = 0,1,…, 9
      where j is the iteration number
3. Finally, the sixteen words which are used in the cipher for multiplication (K[5],K[7],…,K[35]) are modified to have the two properties from above. Each of this words is processed as follows:
   a. the two lowest bits of K[i] are recorded by setting $j = K[i]\wedge 3$, and then considered the word with these 2 bits set to 1, $w=K[i]\vee 3$
   b. a mask M of the bits in w is constructed, which belongs to asequence of 10 or more consecutive 0's or 1's. Namely, M_l if and only if w_l belongs to a sequence of ten consecutive 0's or 1's. Then we reset to 0 the 1's in M which correspond to the "end-points of runs of 0's or 1's in w", and also the two lowest bits and the highest bit in M. More precisely, the i'th bit of M is reset to 0 if i<2, i=31, or if the i'th bit of w differs from either the (i+1)'th or the (i-1)'th bits.
   c. Next a fixed four-word table B is used to "fix w", where the four entries in B are chosen so

that they (and their cyclic shifts) do not contain any seven consecutive 0's or ten consecutive 1's. The 2 recorded bits j (from Step (a)) are used to select an entry from B, and the lowest five bits of K[i-1] to rotate this entry, p = B[j]<<<(lowest 5 bits of K[i-1]).

d. Finally, we xor the pattern p into w under the control of the mask M, and store the result in K[i]: K[i] = w⊕(p∧M). Since the lowest 2 bits of M are 0's, then the lowest 2 bits of K[i] will be 1's. Also, the choice of B guarantees that K[i] will meet not have a sequence of ten consecutive 0's or 1's.

The key expansion procedure guarantees that the key words which are used for multiplication do not have any obvious weaknesses. This procedure keeps these words "random", in the sense that no single word has probability much larger than in the uniform distribution. An exhaustive search confirmed that no 20-bit pattern occurs in these words with probability of more than $1{:}23x2^{-20}$. Similarly, no 10-bit pattern appears with probability larger than $1{:}06x2^{-10}$. Due to the structure of the key expansion procedure, the performance of MARS is essentially independent of the key-length used [1].

IV. HARDWARE IMPLEMENTATION

For the hardware implementation we used a FSM (Finite State Machine) to implement the functionality of key expansion module. The general architecture of an FSM consists of combinational block of next state logic, state registers, and combinational output logic.

Finite state machine must be initialized by means of an explicit reset signal. Otherwise, there is no reliable way to get the VHDL and gate level representation of the FSM into the same known state, and thus no way to verify their equivalence. The description of a finite state machine consists of a process, synchronized on a clock edge (clk).

The module from figure 2 is described using an FSM. There are many ways to describe a finite state machine in VHDL. We use a process containing a case statement [7]. The state of the machine is stored in a state variable, and the possible states are represented with a user-defined enumeration type. The type declaration gives symbolic names to each of the states, but say nothing about their hardware implementation.

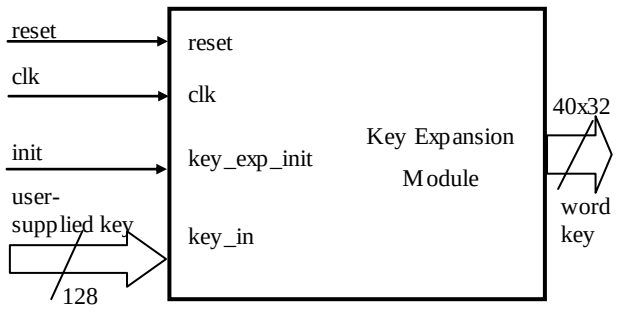

Figure 2. The detailed structure of key expansion module

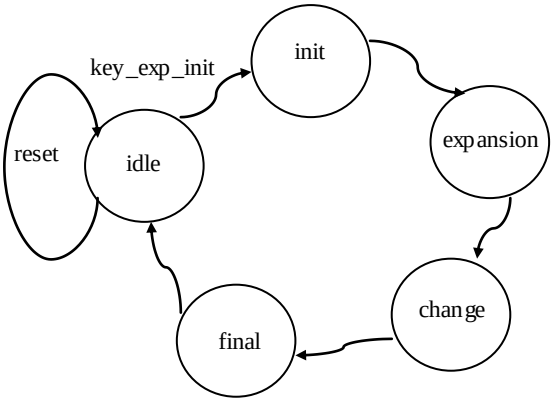

Figure 3. The state transition diagram for key expansion module

Finite state machine must be initialized by means of an explicit reset signal. Otherwise, there is no reliable way to get the VHDL and gate level representation of the FSM into the same known state, and thus no way to verify their equivalence. The description of a finite state machine consists of a process, synchronized on a clock edge (clk).

The state transition diagram for encryption command structure is shown in figure 3. The possible states are: *idle*,*init*,*expansion*,*change*,*final*. All state transitions occur on a rising edge of a global master clock (*clk*). Some of the transitions depends on signals such us *key_exp_init*. *Idle* is a waiting state, the initialization state for the finite state machine. On the first rising edge of the clock, after the *key_exp_init* signal is set *init* becomes current state. In this state the variable are initialize. On the next rising edge of the master clock *expantion* becomes current state and the computing process begin. On the next rising edge the *change* becomes current state. The FSM leaves this state and the next state is *final*. After this the FSM reach again the *idle* state, waiting for another task. *Reset* and *key_exp init* are external asyncronous signals, first for general reset and the second for starting the key expantion process.

The architecture for this command structure consists of processes. In VHDL a process contains sequential statements. While each process executes its statements in sequence, multiple processes interact with each other concurrently.

The architecture for subkeys generation module consists of some of the following processes: *sync_p*, *async_p*. The FSM can be describe using VHDL as below:

```
async_p: process (clk, current_s, key_exp_init)
begin
case current_s is
   when idle_s =>
            if key_exp_init='1' then
                  next_s <=initializare;
            else
            next_s <=idle_s;
        end if;
   when initializare =>
```

```
                    next_s <=expandare;
  when expandare =>
                    next_s <=modificare;
    when modificare =>
                    next_s <=final;
    when final =>
                next_s <=idle_s;
end case;
end process;
```

A. *Design Implementation Summary*

```
Design Summary
--------------
  Number of errors: 0
  Number of warnings: 0
  Number of CLB slices: 131
  Number of function generators: 5
  Number of flip flops or latches: 261
  Number of external Iobs: 130
  Number of Bufgs: 1
  Number of Bufgpads: 1
Total equivalent gate count for design: 2118
-----------------------------------------------------
Device utilization summary:
  Number of External GCLKIOBs 1 out of 4     25%
  Number of External IOBs     130 out of 404 32%
  Number of SLICEs            131 out of 12288  1%
  Number of GCLKs             1 out of 4     25%
-----------------------------------------------------
Timing constraint: Default period analysis
 1810 items analyzed, 0 timing errors detected.
 Minimum period is  18.468ns.
-----------------------------------------------------
Timing constraint: Default net enumeration
 268 items analyzed, 0 timing errors detected.
 Maximum net delay is  12.963ns.
-----------------------------------------------------
Timing summary:
---------------
Timing errors: 0  Score: 0
Constraints cover 1810 paths, 268 nets, and 536
connections (100.0% coverage)
Design statistics:
  Minimum period:  18.468ns (Maximum frequency:
54.148MHz)
  Maximum net delay:  12.963ns
-----------------------------------------------------
```

The MARS key expansion procedure expands the input 128-bit key into a 1280-bit key. First a linear-key expansion occurs following by stirring the key-words based on an S-box. Both processes involves simple operations performed repeatedly. However, the final stage of modifying the multiplication key-words involves string-matching operations that are relatively expensive functions. String-matching is an expensive operation compared with the rest of the operations required by MARS. A compact implementation of string-matching introduces high latency while a high-performance implementation increases the area requirements dramatically. The implementation of key-sheduling in [2] gives a number of 2275 # of slices.

Even if all studies show that in hardware implementation MARS is the slowest one compared with the other four AES candidates [2], [5], it is still faster than DES. MARS has a very high level of security and is much more efficient than many older algorithms.

V. CONCLUSIONS

The main strength of MARS is its robustness [4]. This was the main design goal, and MARS contains more “fail stop” mechanisms. Due to the heterogeneous structure and the large variety of “strong operations” in MARS, even a major advance in the cryptanalysis of any one of its components is very unlikely to lead to a significant attack against the overall cipher [6].

MARS is also a very fast cipher in common use environments (i.e., in software). The large number of fail-stop mechanisms in MARS makes its hardware implementation more involved, but it is still very small and cheap to implement in hardware, and is suitable to any real-life environment [4].

FPGA devices are a highly promising alternative for implementing private-key cryptographic algorithms. Compared with software-based implementations, FPGA implementations can achieve superior performance [2]. Security issues also make FPGA implementations more advantageous than software-based solutions. Hardware cryptographic devices can be securely encapsulated to prevent any modification of the implemented algorithm.

In this paper we present a FPGA implementation of key expansion procedure using Virtex devices.

REFERENCES

[1] Burwick, Coppersmith, D'Avignon, Gennaro, Halevi, Jutla, Matyas Jr., O'Connor, Peyravian, Safford, Zunic, ``MARS - a candidate cipher for AES'' http://www.research.ibm.com/security/mars.html

[2] A. Dandalis, V. K. Prasanna, J. D. P. Rolim - A Comparative Study of Performance of AES Final Candidates Using FPGAs, Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems, 2000, Pag. 125 – 140, ISBN:3-540-41455-X

[3] MARS - a candidate cipher for AES, Carolynn Burwick, Don Coppersmith, Edward D’Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla,Stephen M. Matyas Jr., Luke O’Connor, Mohammad Peyravian, David Safford, Nevenko Zunic, IBM Corporation, Revised, September, 22 1999

[4] IBM MARS Team, MARS and the AES Selection Criteria, May 15, 2000

[5] Federal Register; Department of Commerce: National Institute of Standards and Technology. http://www.nist.gov/aes

[6] T. Wollinger, J. Guajardo, C. Paar - Security on FPGAs: State of the Art Implementations and Attacks ACM Special Issue Security and Embedded Systems Vol. No. 3,

[7] Xilinx Coorporation. Virtex Series FPGAs