VHDL Implementation of Hybrid Block Cipher Method (SRC)

Ashwaq T. Hishem*, Najwa M. Hassen* & Ekhlas M. Farhan**

Received on: 26/4/2009
Accepted on:5/11/2009

Abstract

This paper discusses the hardware design of the hybrid block cipher method that combines the RC6 cipher and the Serpent cipher.
The block size is 128 bits, and the key can be any length up to 256 bytes. This algorithm is designed to take advantage of the powerful, which is supported by RC6 and Serpent encryption algorithms with overcoming their weaknesses, resulting in a much improved security/performance tradeoff over existing ciphers. The discussion addresses hardware design and VHDL implementation of the key expansion algorithm and the encryption algorithm.

Keywords: SRC, RC6, Serpent, Cryptography, VLSI design.

تنفيذ بلغة VHDL لطريقة التشفير الهجينة (SRC)

الخلاصة

هذا البحث يناقش التصميم المادي لطريقة التشفير الهجينة التي تجمع التشفبر RC6 و التشفير Serpent . حجم الكتلة هو 128 bits والمفتاح يمكن ان يصل طوله الى 256 bytes. هـذه الخوارزمية صممت لتاخذ فائدة القوة المدعومة بواسطة خوارزميات التشفير RC6 و Serpent مع تجاوز ضعفهما وقد كانت النتيجة تحسين اكثر للامنية /الاداء المتوازن على ما موجود من تشفيرات. لقد تم مناقشة التصميم و التنفيذ المادي باستخدام لغة VHDL لخوارزميـة توسـيع المفتاح وخوارزمية التشفير.

1 Introduction

Symmetric-key block ciphers have long been used as a fundamental cryptographic element for providing information security. Although they are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudo random number generators, message authentication, protocols, stream ciphers, and hash functions.

There are many symmetric-key block ciphers, which offer different levels of security, flexibility, and efficiency. Among the many symmetric-key block ciphers currently available, some (such as DES, FEAL, Blowfish, IDEA, RC6, Twofish, Mars, Rijndeal, and Serpent) have received the greatest practical interest [1].
The SRC proposed by [2] is a hybrid block cipher that adopts Serpent and RC6 ciphers. The block size is 128

* Control and Systems Engineering Department ,University of Technology/Baghdad
**Electrical and Electronics Engineering Department, University of Technology /Baghdad

https://doi.org/10.30684/etj.28.5.8
2412-0758/University of Technology-Iraq, Baghdad, Iraq

bits, and the key can be any length up to 256 bytes. The algorithm is designed to take advantage of RC6 and Serpent features with overcoming their weaknesses, resulting in a much improved security/performance tradeoff over existing ciphers. As a result, this algorithm offers better security than RC6 and Serpent algorithms. Also, it is compact, fast, simple and easy to understand. It adapts key depended permutation and combined different algebraic group which is adapted from RC6 algorithm. Serpent algorithm is very secure but the best Serpent software implementation will be 3-4 times slower than other candidates, including Mars, Twofish, RC6, and Rijndael. Thus, even a non-conservative version of Serpent with only sixteen rounds would be considerably slower than these other algorithms. On the other hand the RC6 algorithm is the fastest algorithm compared with other candidates, so that the SRC algorithm would combine the Serpent and RC6 algorithms to obtain a fast and more secure algorithm [2].

2 Ciphers Algorithms

In this section a brief description of each algorithms is presented with their main features.

Serpent Algorithm

Serpent encrypts a 128-bit plaintext ***P*** to a 128-bit ciphertext ***C*** in **r** rounds under the control of **r+1** subkey K_0, …, K_r. The user key can be any length between 64 and 256 bits [3]. Short keys with less than 256 bits are mapped to full-length keys of 256 bits by appending one "1" bit and as many "0" bits are required to make up 256 bits [3]. The cipher itself is an SP-network and consists of:

An initial permutation IP.

32 rounds, each consisting of a key mixing operation, a pass S-boxes, and (in all but the last round) a linear transformation. In the last round, this linear transformation is replaced by additional key mixing operation.

A final permutation FP.

Each round function R_i **{i=0,…, 31}** uses only a single replicated S-box. For example, R_0 uses S_0, 32 copies of which are applied in parallel. Thus the first copy of S_0 takes bits 0, 1, 2 and 3 of B_0 Å K_0 as its input and returns as output the first four bits of an intermediate vector; the next copy of S_0 input bits 4-7 of B_0 Å K_0 and returns the next four bits of the intermediate vector, and so on. The intermediate vector is then transformed using the linear transformation, giving B_1. Similarly, R_1 uses 32 copes of S_1 in parallel on B_1 Å K_1 and transforms their output using the linear transformation, giving B_2.

In the last round R_{31}, S_{31} is applied on B_{31}Å K_{31}, and XOR the result with K_{32} rather than applying the linear transformation.

2.2 The RC6 Algorithm

RC6 is a fully parameterized family of encryption algorithms [4]. A version of RC6 is more accurately specified as RC6-w/r/b where the word size is w bits, encryption consists of a nonnegative number of rounds r, and b denotes the length of the encryption key in bytes. Since the AES submission is targeted at w = 32 and r = 20, hence when any other value of w or r is intended in the text, the parameter values will be specified as RC6-w/r. Of particular relevance to the AES effort will be the versions of RC6 with 16-, 24-, and 32-byte keys. Fig. 1 shows the detail of one round of RC6 cipher.

For all variants, RC6-w/r/b operates on units of four w-bit words using the following six basic operations. The

base-two logarithm of w will be denoted by lg*w*.
a + b integer addition modulo 2^w
a - b integer subtraction modulo 2^w
a ⊕ b bitwise exclusive-OR of w-bit words
a × b integer multiplication modulo 2^w
a<<<b rotate the w-bit word a to the left by the amount Given by the least significant lg*w* bits of b
a>>>b rotate the w-bit word to the right by the amount given by the least significant lg*w* bits of b.

2.3 The SRC Algorithm

Like RC6 and Serpent, SRC is a fully parameterized family of encryption algorithms. However, the SRC algorithm overcame the weaknesses of RC6 and Serpent algorithms. SRC is a block cipher that encrypts data in 16-byte blocks. The algorithm consists of two parts: a key-expansion part and a data-encryption part as shown in Fig 2. Key expansion will be adapts from Serpent algorithm. SRC has 32 rounds. Each round consists of a key-dependent permutation and substitution. The substitution will be adapted from Serpent and key-dependent permutation from RC6.

The SRC algorithm uses an integer multiplication, rotation and shift as additional primitive operations which are greatly increases the diffusion achieved per round, allowing for greater security, fewer rounds, and increased throughput. As a result the new SRC has much faster diffusion than Serpent.

3 Hardware Design

The VHSIC Hardware Description Language is an industry standard language used to describe hardware.
The following are some of the building blocks of VHDL.
1. Entity: All designs are expressed in terms of entities. An entity is the most basic building block in a design. It specifies the name of the entity, ports of the entity, and entity-related information. All designs are created using one or more entity.
2. Architecture: All entities that can be simulated have an architecture description. The architecture describes the behavior of the entity.
3. Configuration: A configuration statement is used to bind a component instance to an entity-architecture pair.
4. Package: A package is a collection of commonly used data types and subprograms used like a parts list for a design.

Using these building blocks we have implemented the SRC block cipher to demonstrate the improvement in time, as shown in the waveforms at the end [5].
The iterated approach to implementing the algorithm is used in the design. Considering the algorithm steps, one step is performed per clock period, with the output of the previous step being used as the input to the next step. Data is only placed on the output after the required number of algorithm rounds has been completed. The design involves encryption module and SRC encryption module interconnected as depicted by Fig. 3.

3.1 Encryption Module Design

The complete description of the hardware proposal is presented in VHDL language. Data in each encryption iteration pass into two units: Serpent' S-box and RC6 round (see Fig. 2). The two units are connected directly and the data is flow under the control of the control unit signals that delivered depending on state machine scheme. The algorithm in Fig. 4 shows the state machine in VHDL code.
After reset input has been strobe, the state is starting to count from zero. At

the first state the key expansion module produces the keys and then delivers them one by one at the next steps.

Meanwhile the SRC block is also reset and starts to receive the keys provided by the key expansion module. First, the data is XORed with the first key then it is substituted in the Serpent's S-boxes. Before result is provided to the RC6 round the round counter is checked, if it reached the 32 round, then the Serpent output will provide the final output after XORing it with the last key. If round is less than 32, then data will applied to the RC6 round and return to the Serpent's S-box again.

3.2 Key Expansion Design

In this paper, the hardware implementation of the key schedule is not addressed. The key expansion algorithm was programmed in a non-synthesizable code of VHDL for simulation purpose. The key for each round in Serpent is different from the key of RC6 round (one 128-bit key for Serpent round vs. two 32-bit key for RC6 round). As described in [2], the round key for the SRC is 128-bit and it is used directly to the Serpent's S-box but for the RC6 only the first 64 bits is used. The derivation of Serpent round key and RC6 round keys is shown in Fig. 5.

4 Simulation and Synthesis Results

The Simulation result was obtained using *ModelSim 5.4* environment in conjunction with *Project Navigator (ISE 4.1)*. Table 1 below lists six different Data/Key samples with their resultant outputs. The time diagram of the first sample is shown in Figure 6.

The synthesis results represent the Iterative design specifications (area and speed) of the SRC encryption module. The *timing and delay specification* is summarized by the synthesis report as follow:

Number of ?5%
External
GCLKIOBs:
Number of 512 75%
External IOBs
Number of 12288 6%
SLICEs:
Number of ?5%
GCLKs:
Speed Grade: -4
Target device: xcv1000-4fg680
Minimum period: 51.604ns (Maximum Frequency: 19.378MHz)
Minimum input arrival time before clock: 51.221ns
Maximum output required time after clock: 13.152ns

While the *Device utilization summary* as provided by the place and route report is shown below:

For the purpose of comparison, regarding the hardware, we find that Serpent is achieving a throughput of 5.04 Gbit/sec, versus 2.40 Gbit/sec for RC6 [6]. An NSA study of ASIC costs predicts 8.03 Gbit/sec for Serpent versus 2.171 for RC6 [7].

Conclusions

Cryptographic algorithms are more efficiently implemented in custom hardware than in software running on general-purpose processors. this paper addresses hardware implementation approaches for the SRC block cipher algorithm and describes the design and performance testing of algorithm. This implementation of the algorithm and encryption processors in VHDL allow for efficient implementation in both FPGA and ASIC mediums. Also optimizations and changes can be made quickly and easily. This allows for a high degree of scalability and controllability of the parallel architecture. The synthesis results

present an estimated value of maximum speed achieved by the design as well as the occupied size of the target device. Compared to software implementation, hardware implementations provide more physical security as well as higher speed. This implementation will be useful in wireless security like military communication and mobile telephony where there is a greater emphasis on the speed of communication.

References

[1] Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. M. Matyas, L. O'Connor, M. Peyravian, D. Safford and N. Zunic, "Mars a candidate cipher for AES", First Advanced Encryption Standard (AES) Conference, Ventura, CA, 1998.

[2] Ashwaq T. Hashim "SRC hybrid Block Cipher method", IJCCCE, Vol.6, No.1, 2006.

[3] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," First Advanced Encryption Standard (AES) Conference, Ventura, CA, 1998.

[4] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6TM Block Cipher," First Advanced Encryption Standard (AES) Conference, Ventura, CA, 1998.

[5] Krishnamurthy G.N , V. Ramaswamy , Leela G.H and Ashalatha M.E," Blow-CAST-Fish: A New 64-bit Block Cipher", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, p. 282,April 2008.

[6] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", The Third Advanced Encryption Standard Candidate Conference. National Institute of Standards and Technology, New York, New York, USA, 13-27.

[7] B. Weeks, M Bean, T. Rozylowicz, C. Ficke, "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms", Proceedings of 3rd, AES conference, New York, pages 286-304. April 2000

Table (1) Data/Key samples with their resultant outputs

Test No.	Signal	Value
1	Key	00000000000000000000000000000000 00000000000000000000000000000000
	Input data	00000000000000000000000000000000
	Output data	**185E66C02BCC78ACBA18D06384F671F1**
2	Key	00000000000000000000000000000000 00000000000000000000000000000000
	Input data	**FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF**
	Output data	**D77E09762153C2C552C22668B619BED5**
3	Key	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
	Input data	00000000000000000000000000000000
	Output data	**B9EC50C4297734C3892AF36CEA7D645F**
4	Key	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
	Input data	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
	Output data	**CBA45DEE1D4AB3582F82DA7AD3AEC305**
5	Key	00000000000000000000000000000000 00000000000000000000000000000000
	Input data	**0123456789ABCDEF0123456789ABCDEF**
	Output data	**8697BE0C55A5F18B4CF0E5B673CAEB64**
6	Key	**0123456789ABCDEF0123456789ABCDEF** **0123456789ABCDEF0123456789ABCDEF**
	Input data	00000000000000000000000000000000
	Output data	**FEDC229F744E63FC3B9449C0DD3BBFF4**

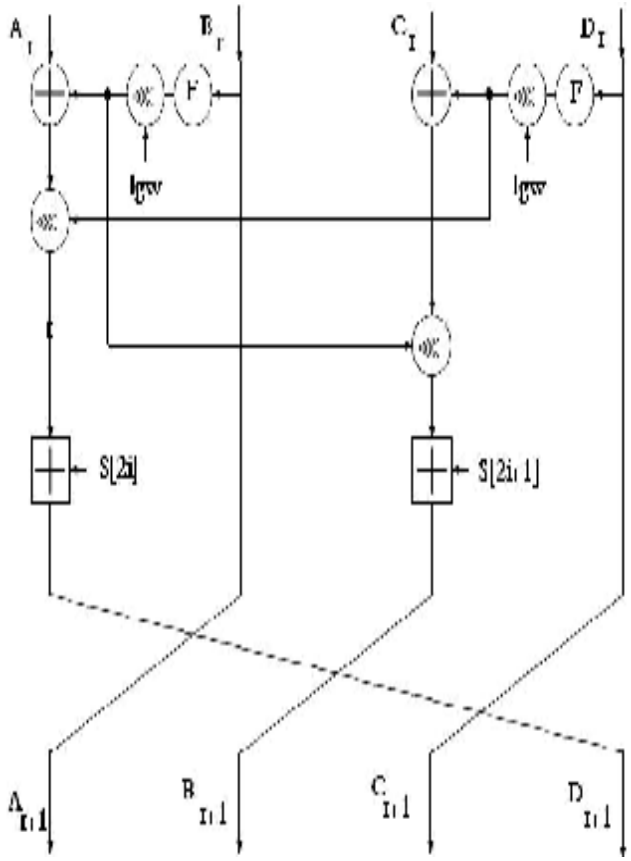

Figure (1) RC6 round

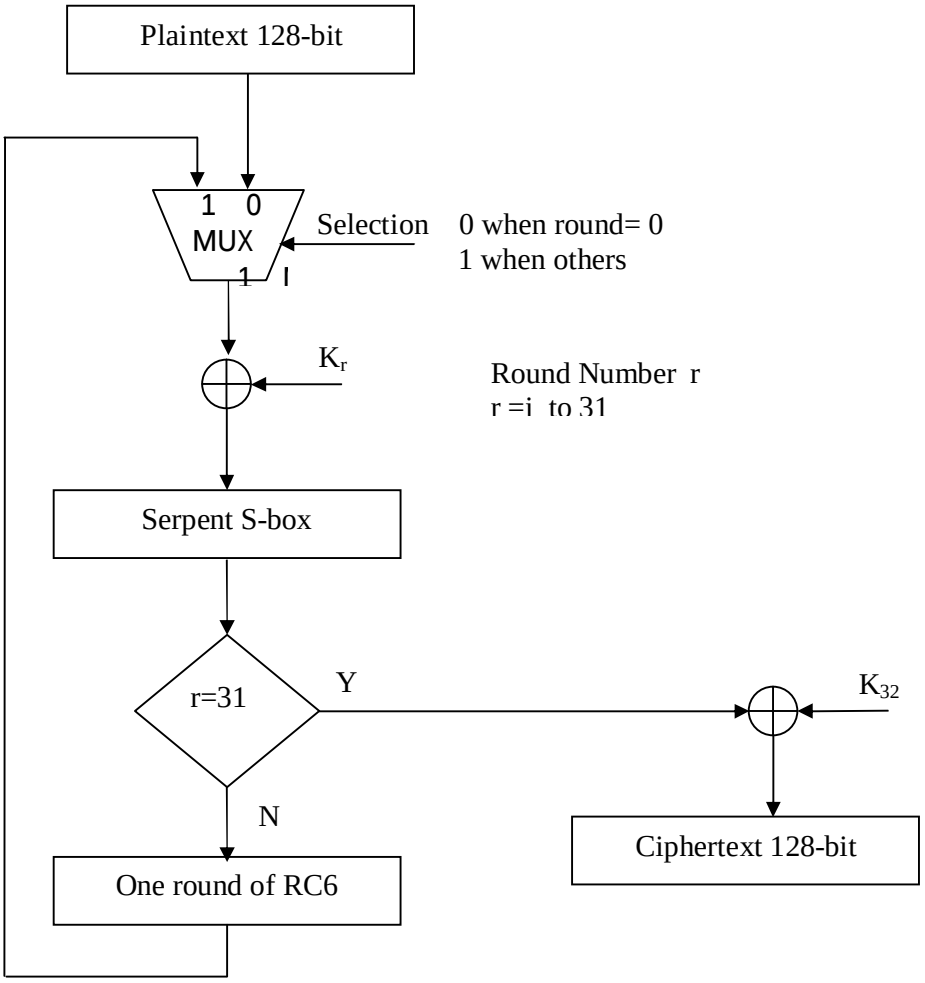

Figure (2) SRC Algorithm

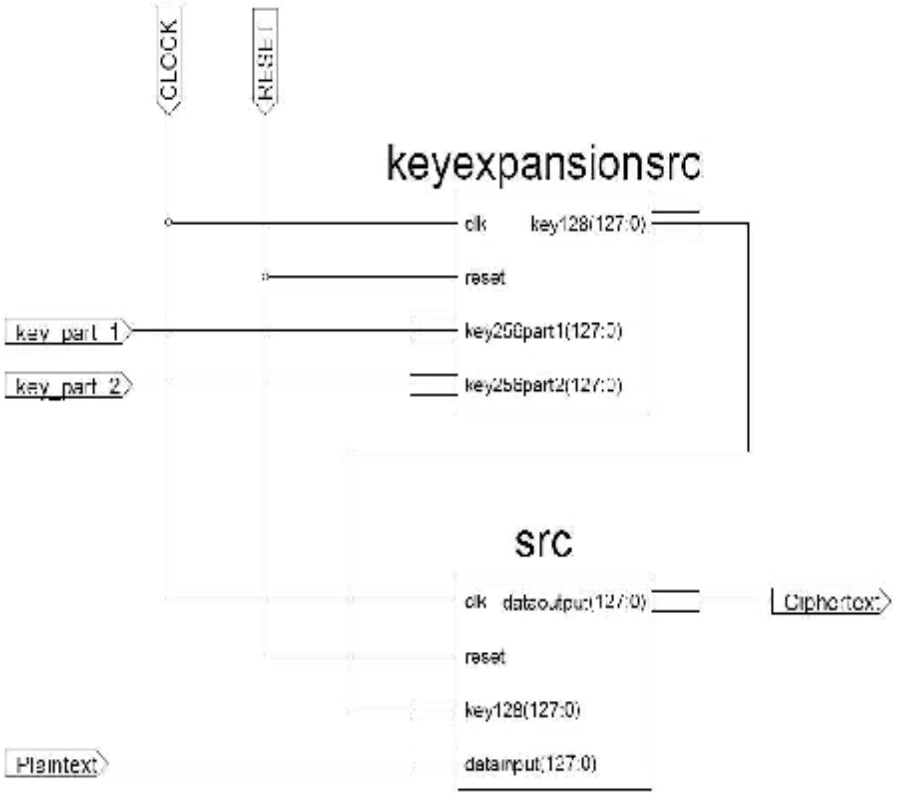

Figure (3) Interconnection of SRC encryption module and key expansion module

```
if reset ='1' then
     state<="00";j<=0;
else
     case state is
     when "00" =>
     s_datainput<= datainput;
     state<= "01";  j<=j+1;
     dataoutput<=(others=> 'Z');
     when "01" =>
     s_datainput<= rc6out;
     j<=j+1;
     dataoutput<=(others=> 'Z');
     if j=32 then
          state<= "10";
     end if;
     when "10" =>
     dataoutput<=serpentout xor key128;
     state<="11";
     when others=>
     dataoutput<=(others=> 'Z');
     end case;
end if;
```

Figure (4) State Machine that control the data flow for the 32 iteration

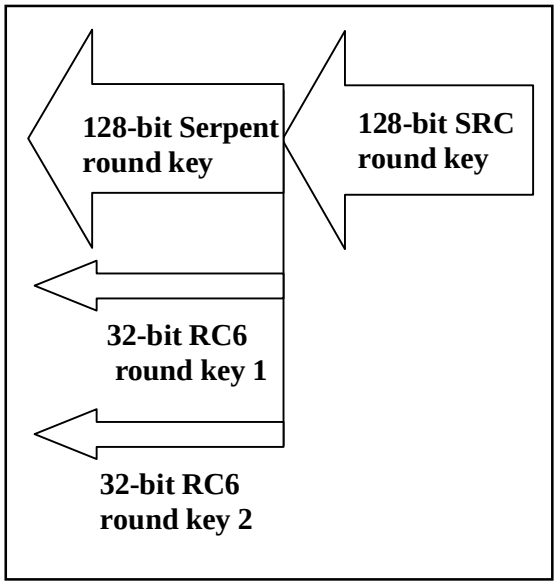

Figure (5) Derivation of RC6 round key and Serpent round key from SRC round key

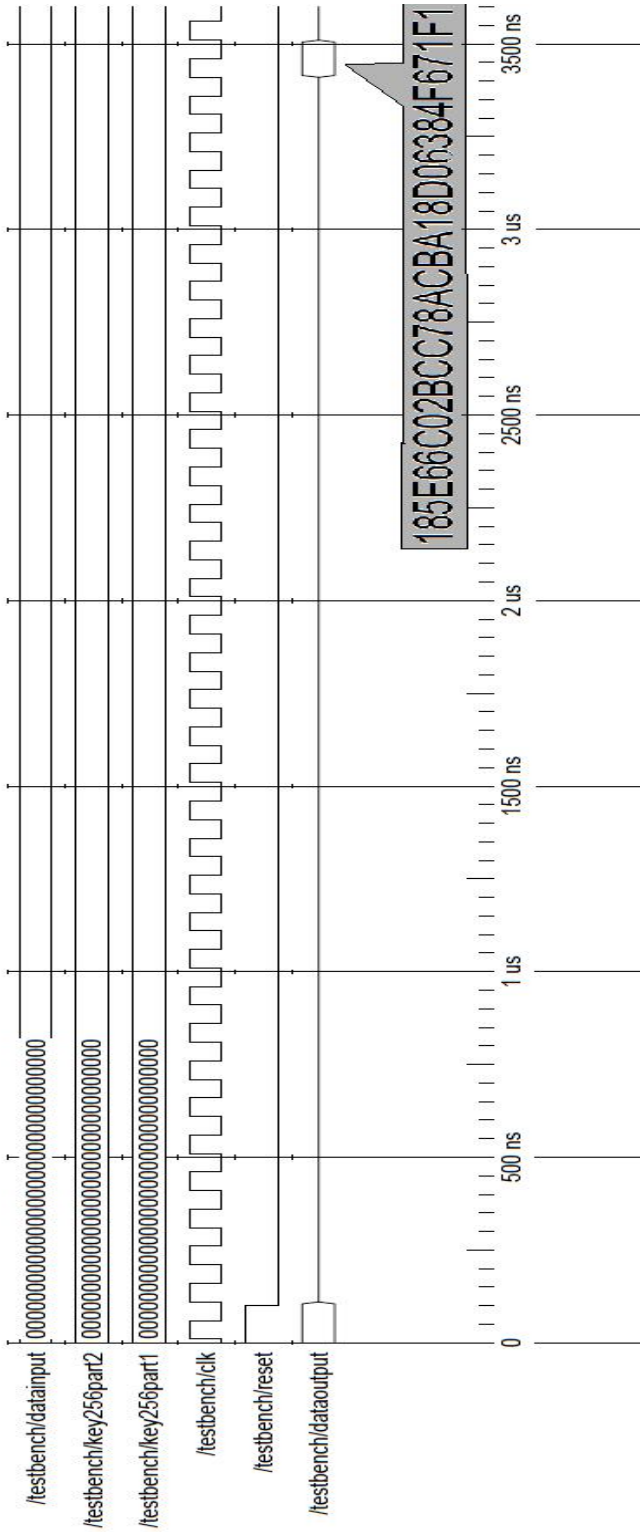

Figure (6) Time diagram for all zero input data and all zero key