ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Implementation of an Ultra-Lightweight Block Cipher

Sri Ramudu
M.tech scholar (VLSI),
Department of ECE,
VNRVJIET.

G.Shanthi
Assistant Professor,
Department of ECE,
VNRVJIET.

Abstract:

With the establishment of the AES (Advanced Encryption Standard) the need for new block ciphers has been greatly diminished; for almost all block cipher applications the AES is an excellent and preferred choice. However, despite recent implementation advances, the AES is not suitable for extremely constrained environments such as RFID(Radio Frequency Identification) tags and sensor networks. In this paper we describe an ultra-lightweight block ciphert.

Both security and hardware efficiency have been equally important during the design of the cipher and at 1570 GE, the hardware requirements for this cipher are competitive with today's leading compact stream ciphers. It is one of the most compact encryption methods ever designed and is 2.5 times smaller than AES. The block size is 64 bits and the key size can be 80 bit or 128 bit. It is intended to be used in situations where low-power consumption and high chip efficiency is desired. The International Organization for Standardization and the International Electro technical Commission included this block cipher in the new international standard for lightweight cryptographic methods.

Keywords:
block ciphers, cipher, stream cipher.

I. INTRODUCTION:

One defining trend of this century's IT landscape will be the extensive deployment of tiny computing devices. Not only will these devices feature routinely in consumer items, but they will form an integral part of a pervasive — and unseen — communication infrastructure.

It is already recognized that such deployments bring a range of very particular security risks. Yet at the same time the cryptographic solutions, and particularly the cryptographic primitives, we have at hand are unsatisfactory for extremely resource-constrained environments.In this project we used a new hardware-optimized block cipher that has been carefully designed with area and power constraints uppermost in our mind. Yet, at the same time, we have tried to avoid a compromise in security. In achieving this we have looked back at the pioneering work embodied in the DES and complemented this with features from the AES finalist candidate Serpent which demonstrated excellent performance in hardware.At this point it would be reasonable to ask why we might want to design a new block cipher. After all, it has become an "accepted" fact that stream ciphers are, potentially, more compact. Indeed, renewed efforts to understand the design of compact stream ciphers are underway with the eSTREAM project and several promising proposals offer appealing performance profiles.

But we note couple of reasons why we might want to consider a compact block cipher. First, a block cipher is a versatile primitive and by running a block cipher in counter mode (say) we get a stream cipher. But second, and perhaps more importantly, the art of block cipher design seems to be a little better understood than that of stream ciphers. For instance, while there is a rich theory under-pinning the use of linear feedback shift registers it is not easy to combine these building blocks to give a secure proposal. We suspect that a carefully designed block cipher could be a less risky undertaking than a newly designed stream cipher. Thus, we feel that a block cipher that requires similar hardware resources as a compact stream cipher could be of considerable interest.

It is important to realize that in developing a new block cipher, particularly one with aggressive performance characteristics, we are not just looking for innovative implementation. Rather, the design and implementation of the cipher go hand-in-hand and this has revealed several fundamental limits and inherent contradictions. For instance, a given security level places lower bounds on the block length and key length. Just processing a 64-bit state with 80-bit key places fundamental lower limits on the amount of space we require. We also observe that hardware implementation particularly compact hardware implementation favours repetition. Even minor variations can have an unfortunate effect on the space required for an implementation. Yet, at the same time, the cryptanalyst also favours repetition and seeks mathematical structures that propagate easily across many rounds. How much simple, repetitive structure can we include without compromising its security? The linear cryptanalysis and differential cryptanalysis, the two most significant attacks applicable to symmetric-key block ciphers. Linear cryptanalysis was introduced by Matsui at EUROCRYPT '93 as a theoretical attack on the Data Encryption Standard (DES) and later successfully used in the practical cryptanalysis of DES; differential cryptanalysis was first presented by Biham and Shamir at CRYPTO '90 to attack DES.

Although the early target of both attacks was DES, the wide applicability of both attacks to numerous other block ciphers has solidified the pre-eminence of both cryptanalysis techniques in the consideration of the security of all block ciphers. For example, many of the candidates submitted for the recent Advanced Encryption Standard process undertaken by the National Institute of Standards and Technology were designed using techniques specifically targeted at thwarting linear and differential cryptanalysis. This is evident, for example, in the Rijndael cipher, the encryption algorithm selected to be the new standard.

II. A Basic Substitution-Permutation Network Cipher:

The cipher that we shall use to present the concepts is a basic Substitution-Permutation Network (SPN). We will focus our discussion on a basic cipher, illustrated in Figure below, that takes a 16-bit input block and processes the block by repeating the basic operations of around four times.

Each round consists of (1) substitution, (2) a transposition of the bits (i.e., permutation of the bit positions), and (3) key mixing. This basic structure was presented by Feistel back in 1973 [15] and these basic operations are similar to what is found in DES and many other modern ciphers, including Rijndael. So although, we are considering a somewhat simplified structure, an analysis of the attack of such a cipher presents valuable insight into the security of larger, more practical constructions.

Substitution:

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

Table1: S-box Representation (in hexadecimal)

In our cipher, we break the 16-bit data block into four 4-bit sub-blocks. Each sub-block forms an input to a 4´4 S-box (a substitution with 4 input and 4 output bits), which can be easily implemented with a table lookup of sixteen 4-bit values, indexed by the integer represented by the 4 input bits. The most fundamental property of an S-box is that it is a nonlinear mapping.

Permutation:

The permutation portion of a round is simply the transposition of the bits or the permutation of the bit positions. The permutation of Figure below is given in Table below (where the numbers represent bit positions in the block, with 1 being the leftmost bit and 16being the rightmost bit) and can be simply described as: the output i of S-box j is connected to input j of S-box i. Note that there would be no purpose for a permutation in the last round and, hence, our cipher does not have one.

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Table2: Permutation

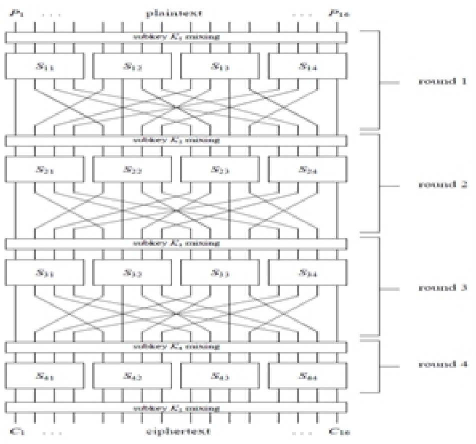


Figure 1. Basic Substitution-Permutation Network (SPN) Cipher

Key Mixing:

To achieve the key mixing, we use a simple bit-wise exclusive-OR between the key bits associated with a round (referred to as a subkey) and the data block input to a round. As well, a subkey is applied following the last round, ensuring that the last layer of substitution cannot be easily ignored by a cryptanalyst that simply works backward through the last round's substitution. Normally, in a cipher, the sub key for a round is derived from the cipher's master key through a process known as the key schedule. In our cipher, we shall assume that all bits of the sub keys are independently generated and unrelated.

Decryption:

In order to decrypt, data is essentially passed backwards through the network. Hence, decryption is also of the form of an SPN as illustrated in Figure 1. However, the mappings used in the S-boxes of the decryption network are the inverse of the mappings in the encryption network (i.e., input becomes output, output becomes input). This implies that in order for an SPN to allow for decryption, all S-boxes must be bijective, that is, a one-to-one mapping with the same number input and output bits. As well, in order for the network to properly decrypt, the sub keys are applied in reverse order and the bits of the sub keys must be moved around according to the permutation, if the SPN is to look similar to Figure 1. Note also that the lack of the permutation after the last round ensures that the decryption network can be the same structure as the encryption network.

III. DESIGN OF BLOCK CIPHER:

Design of a block cipher includes three tasks

• Zero-in to a good S-box, which can render sound non-linearity to the design.

• Define a permutation layer (in connection with the S-box) to achieve good diffusion.

• Design of Key Schedule Algorithm (KSA).
Understanding the existing attacks on a specific type of block cipher design is crucial.
The design process is a cycle.

• Understand basic design principles.

• Define the basic building blocks.

• Understand and analyse the existing attacks.

• Re-define the building blocks.

This process goes on for considerable time and finally we try to aggregate a design which gives us best security trade-off.

We used a SPN based 64-bit block cipher. The cipher name is sriram. Sriram takes 64-bit blocks and encrypts them using a Substitution- Permutation scheme and the cipher has 27 rounds. The Key Scheduling Algorithm (KSA) has two variants. One takes a 96-bit, the other 128-bit user given key. Toplevel description of the Sriram is given in Table below.

```
generateRoundKeys()
for $i := 1$ to 27 do
        addRoundKey(STATE,K_i)
        sBoxLayer(STATE)
        pLayer(STATE)
end for
addRoundKey(STATE,K_32)
```

Algorithm: Top-level description

S-Box Design:

The S-box used in Sriram is a 4 X 4 one. Since the block length is64, this S-box is applied 16-times in parallel on 4-bit nibbles. The S-box is given in Table below. The application of the S-box to the current state is denoted by the procedure sBoxLayer(STATE).

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S(x)	6	3	5	8	1	14	2	11	15	12	9	7	10	0	4	13

Table3: S-Box table

The algebraic equations for Sriram S-box are generated. The input bits are denoted by x0, x1, x2 and x3. The output bits are y0, y1, y2 and y3.

$$y_0 + x_0x_1x_2 + x_0 + x_1x_2x_3 + x_1x_3 + x_2x_3 = 0$$

$$y_1 + x_0x_2x_3 + x_0x_2 + x_0x_3 + x_1x_2x_3 + x_1 + x_3 + 1 = 0$$

$$y_2 + x_0x_1x_3 + x_0x_1 + x_0x_3 + x_1x_2x_3 + x_1x_3 + x_1 + x_2 + 1 = 0$$

$$y_3 + x_0x_1x_3 + x_0x_2x_3 + x_0x_2 + x_0 + x_1 + x_2 + x_3 = 0$$

Permutation Layer Design:

The next component of the design is a permutation layer which permutes the numbers 0 to 63. We have used a simple permutation which allows easy implementation. We also ensured that the permutation layer of Sriram satisfies the following conditions to ensure security and clear analysis.

- The 4 input bits of S-box of next S-layer, comes from 4 different S-boxes of previous layer.
- The 4 output bits of an S-box of previous S-layer, goes to 4 different S-boxes of the next layer.
- The output bits of a group of 4 S-boxes of previous S-layer, spans over all the 16 S-boxes of next S-layer.
- The diffusion number of active S-boxes after n-round is high.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(x)	29	45	61	13	30	46	62	14	31	47	63	15	32	48	0	16
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(x)	33	49	1	17	34	50	2	18	35	51	3	19	36	52	4	20
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(x)	37	53	5	21	38	54	6	22	39	55	7	23	40	56	8	24
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(x)	41	57	9	25	42	58	10	26	43	59	11	27	44	60	12	28

Table4: Permutation table:

The application of the permutation layer to the current state is denoted by the procedure pLayer(STATE).

Key Scheduling Algorithm

We followed the guidelines followed by designers of PRESENT for designing the key scheduling algorithm (KSA) of Sriram. These are as follows:

- Use of round dependent constants for eliminating symmetry. In the case of Sriram, the constants are binary bits of the round number.
- Rotate to get good diffusion.
- Use of two S-boxes for introducing non-linearity. The same S-box (the S-layer) is used.
- We have made it recursive to obtain a small size implementation. The KSA is physically independent of the block cipher and is run only once. Still, a small size KSA is desirable.

Sriram takes a 96-bit or 128-bit user given key and extracts 64-bit round keys. The key is extracted initially and the register `STATE'is updated. The KSA for the 96-bit version is:

$$[k_{95}k_{94}\dots k_1k_0] = [k_{48}k_{47}\dots k_{50}k_{49}] \text{ (left rotate 47 positions)}$$

$$[k_{95}k_{94}k_{93}k_{92}] = S(k_{95}, k_{94}, k_{93}, k_{92}) \text{ (apply S-box to leftmost 4-bit nibble)}$$

$$[k_{35}k_{34}k_{33}k_{32}] = S(k_{35}, k_{34}, k_{33}, k_{32}) \text{ (apply S-box to second-leftmost 4-bit nibble)}$$

$$[k_{37}k_{36}k_{35}k_{34}k_{33}] = [k_{37}k_{36}k_{35}k_{34}k_{33}] \oplus round\_counter\_i.$$

The KSA for the 128-bit version is:

$$[k_{127}k_{126}\dots k_1k_0] = [k_{66}k_{65}\dots k_{68}k_{67}] \text{ (left rotate 67 positions)}$$

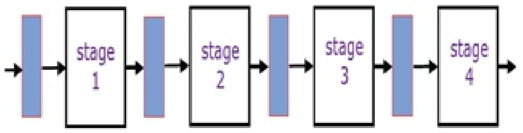
$$[k_{127}k_{126}k_{125}k_{124}] = S(k_{127}, k_{126}, k_{125}, k_{124}) \text{ (apply S-box to leftmost 4-bit nibble)}$$

$$[k_{123}k_{122}k_{121}k_{120}] = S(k_{123}, k_{122}, k_{121}, k_{120}) \text{ (apply S-box to second-leftmost 4-bit nibble)}$$

$$[k_{66}k_{65}k_{64}k_{63}k_{62}] = [k_{66}k_{65}k_{64}k_{63}k_{62}] \oplus round\_counter\_i.$$

Pipelined Architecture:

Pipeline processing is a well-known century old technique employed in industrial production lines. Idea behind pipeline is simple but elegant. The ideal pipeline is:

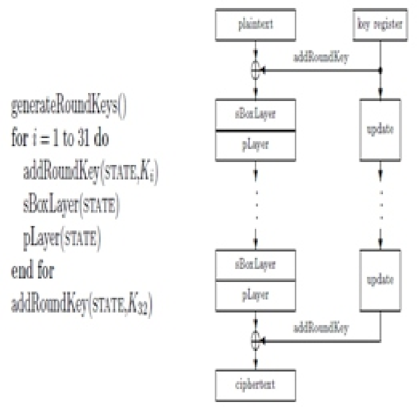


- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal

- The scheduling of an object entering the pipeline is not affected by the objects in other stages

IV.Hardware Optimization Of Block Cipher And Using Pipeline:

While there is a growing body of work on low-cost cryptography, the number of papers dealing with ultra-lightweight ciphers is surprisingly limited. Since our focus is on algorithm design we won't refer to work on low-cost communication and authentication protocols. Some of the most extensive work on compact implementation is currently taking place within the eSTREAM project. As part of that initiative, new stream ciphers suitable for efficient hardware implementation have been proposed. While this work is ongoing, some promising candidates are emerging. equivalents (GE) would be required for the more compact ciphers within the eSTREAM project.



```
generateRoundKeys()
for $i = 1$ to 31 do
    addRoundKey(STATE,$K_i$)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE,$K_{32}$)
```

Fig. 2. A top-level algorithmic description of cipher

Design Issues:

Besides security and efficient implementation, the main goal when designing cipher was simplicity. It is therefore not surprising that similar designs have been considered in other contexts [21] and can even be used as a tutorial for students [20]. In this section we justify the decisions we took during the design of present. First, however, we describe the anticipated application requirements.

Goals and environment of use:

In designing a block cipher suitable for extremely constrained environments, it is important to recognise that we are not building a block cipher that is necessarily suitable for wide-spread use; we already have the AES [35] for this. Instead, we are targeting some very specific applications for which the AES is unsuitable. These will generally conform to the following characteristics.

- The cipher is to be implemented in hardware.

- Applications will only require moderate security levels. Consequently, 80- bit security will be adequate. Note that this is also the position taken for hardware profile stream ciphers submitted to eSTREAM [15].

- After security, the physical space required for an implementation will be the primary consideration. This is closely followed by peak and average power consumption, with the timing requirements being a third important metric.

- In applications that demand the most efficient use of space, the block cipher will often only be implemented as encryption-only. In this way it can be used within challenge-response authentication protocols and, with some careful state management, it could be used for both encryption and decryption of communications to and from the device by using the counter mode [36].

Taking such considerations into account we decided to make present a 64-bit block cipher with an 80-bit key3. Encryption and decryption with present have roughly the same physical requirements. Opting to support both encryption and decryption will result in a lightweight block cipher implementation that is still smaller than an encryption-only AES.

The permutation layer:

When choosing the mixing layer, our focus on hardware efficiency demands a linear layer that can be implemented with a minimum number of processing elements, i.e. transistors. This leads us directly to bit permutations. Given our focus on simplicity, we have chosen a regular bit-permutation and this helps to make a clear security analysis

The S-box:

We use a single 4-bit to 4-bit S-box S : F42 F42in present. This is a direct consequence of our pursuit of hardware efficiency, with the implementation of such an S-box typically being much more compact than that of an 8-bit S-box. Since we use a bit permutation for the linear diffusion layer, AES-like diffusion techniques [12] are not an option for present. Therefore we place some additional conditions on the S-boxes to improve the so-called avalanche of change. More precisely, the S-box for present fulfils the following conditions, where we denote the Fourier coefficient of S by

$$S_b^W(a) = \sum_{x \in \mathbb{F}_2^4} (-1)^{\langle b, S(x)\rangle + \langle a, x\rangle}.$$

1. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed non-zero output difference $\Delta_O \in \mathbb{F}_2^4$ we require

$$\#\{x \in \mathbb{F}_2^4 \,|\, S(x) + S(x + \Delta_I) = \Delta_O\} \leq 4.$$

2. For any fixed non-zero input difference $\Delta_I \in \mathbb{F}_2^4$ and any fixed output difference $\Delta_O \in \mathbb{F}_2^4$ such that $\mathrm{wt}(\Delta_I) = \mathrm{wt}(\Delta_O) = 1$ we have

$$\{x \in \mathbb{F}_2^4 \,|\, S(x) + S(x + \Delta_I) = \Delta_O\} = \emptyset.$$

3. For all non-zero $a \in \mathbb{F}_2^4$ and all non-zero $b \in \mathbb{F}_4$ it holds that $|S_b^W(a)| \leq 8$.

4. For all $a \in \mathbb{F}_2^4$ and all non-zero $b \in \mathbb{F}_4$ such that $\mathrm{wt}(a) = \mathrm{wt}(b) = 1$ it holds that $S_b^W(a) = \pm 4$.

Using a classification of all 4-bit S-boxes that fulfil the above conditions we chose an S-box that is particular well-suited to efficient hardware implementation.

The Block Cipher :

This cipher is an example of an SP-network and consists of 31 rounds. The block length is 64 bits and two key lengths of 80 and 128 bits are supported. Given the applications we have in mind, we recommend the version with 80-bit keys.

This is more than adequate security for the low-security applications typically required in tag-based deployments, but just as importantly, this matches the design goals of hardware-oriented stream ciphers and allows us to make a fairer comparison.

Each of the 31 rounds consists of an xor operation to introduce a round key Ki for 1 ≤ i ≤ 32, where K32 is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box S which is applied 16 times in parallel in each round.

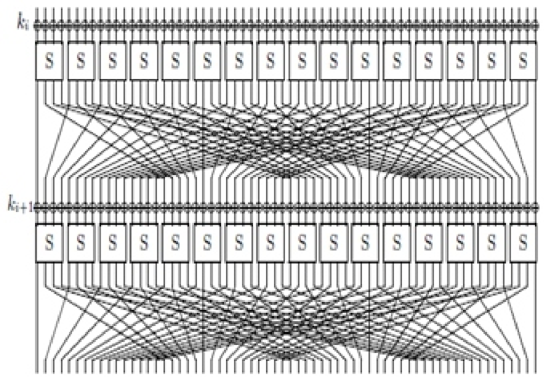


Fig3: The S/P network for present.

The Present can take keys of either 80 or 128 bits. we focussed on the version with 80-bit keys. The user-supplied key is stored in a key register K and represented as k79k78 . . . k0. At round i the 64-bit round key Ki = κ63κ62 . . . κ0 consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that:

$$K_i = \kappa_{63}\kappa_{62}\ldots\kappa_0 = k_{79}k_{78}\ldots k_{16}.$$

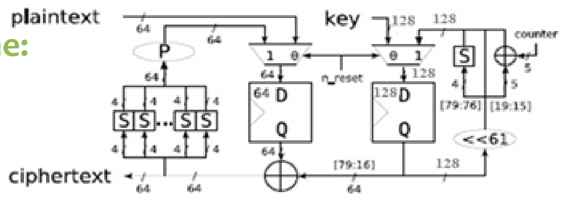
After extracting the round key Ki, the key register K = k79k78 . . . k0 is updated as follows:

1. $[k_{79}k_{78}\ldots k_1k_0] = [k_{18}k_{17}\ldots k_{20}k_{19}]$
2. $[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$
3. $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus \texttt{round\_counter}$

Thus, the key register is rotated by 61 bit positions to the left, the left-most four bits are passed through the present S-box, and the round counter valuei is exclusive-ored with bits k19k18k17k16k15 of K with the least significant bit ofround_counter on the right. The key schedule for 128-bit keys is similar as 80-bit.the Area optimized version of the Present-80 is as follows:

Fig4: The data path of an area-optimized version of present-128

Pipeline:



A 3-stage pipeline has been implemented on Present-80 cipher. The 3-stages are as follows:

- Key mixing
- Substitution layer
- Permutation layer

After every pipeline stage, a 64-bit register is placed to store store the result of their respective stage in each cycle to send it to next stage in pipeline in next cycle.

V.OPTIMIZING PERFORMANCE OF LINEAR-CRYPTANALYSIS WITH PARALLEL COMPUTING:

The ten functions used in Linear Cryptanalysis are as follows:

- **Gen_lat():** For generating linear approximation table.
- **Find_wgt():** For finding input/output mask pair for every output/input mask with maximum weight.
- **Rep_check():** For finding repetition of approximations when finding more than approximations.
- **Permutation():** For traversing through permutation layer.
- **Find_trail() & trail_round():** For matsui's Branch and Bound Algorithm.
- **Encrypt() & pc_pairs_gen():** For forming plain-cipher texts pairs.
- Gen_eq: For generating equation of best approximation.
- **Extract_key:** For extracting key bits from cipher network.

Designing parallel version of Linear-Cryptanalysis as follows:

Determine whether or not the problem is one that can actually be parallelized:

All the functions except find_trail() and trail_round() can be parallelizable and trail_round() can be partially parallelizable.

Identify the program's hotspots:

Know where most of the real work is being done. The majority of scientific and technical programs usually accomplish most of their work in a few places.Profilers and performance analysis tools can help here. Focus on parallelizing the hotspots and ignore those sections of the program that account for little CPU usage.

Analysis:Extract_key() function is a hotspot as it has to iterate all combinations of the key of the active Sboxes with 10000 plain – cipher text pairs.

Identify bottlenecks in the program:

Are there areas that are disproportionately slow, or cause parallelizable work to halt or be deferred? For example, I/O is usually something that slows a program down.May be possible to restructure the program or use a different algorithm to reduce or eliminate unnecessary slow areas.

Analysis: No considerable bottlenecks in the code.

Partitioning:

One of the first steps in designing a parallel program is to break the problem into discrete "chunks" of work that can be distributed to multiple tasks. This is known as decomposition or partitioning. There are two basic ways to partition computational work among parallel tasks: domain decomposition and functional decomposition.

Analysis: Functional decomposition has been applied.

Communications:

Some types of problems can be decomposed and executed in parallel with virtually no need for tasks to share data. For example, imagine an image processing operation where every pixel in a black and white image needs to have its colour reversed. The image data can easily be distributed to multiple tasks that then act independently of each other to do their portion of the work.These types of problems are often called embarrassingly parallel because they are so straight-forward. Very little inter-task communication is required.

Most parallel applications are not quite so simple, and do require tasks to share data with each other. For example, a 3-D heat diffusion problem requires a task to know the temperatures calculated by the tasks that have neighbouring data. Changes to neighbouring data has a direct effect on that task's data.

Analysis: This code is embarrassingly parallel.

Synchronization:

• Barrier

o Usually implies that all tasks are involved

o Each task performs its work until it reaches the barrier. It then stops, or "blocks".

o When the last task reaches the barrier, all tasks are synchronized.

o What happens from here varies. Often, a serial section of work must be done. In other cases, the tasks are automatically released to continue their work.

• Lock / semaphore

o Can involve any number of tasks

o Typically used to serialize (protect) access to global data or a section of code. Only one task at a time may use (own) the lock / semaphore / flag.

o The first task to acquire the lock "sets" it. This task can then safely (serially) access the protected data or code.

o Other tasks can attempt to acquire the lock but must wait until the task that owns the lock releases it.

o Can be blocking or non-blocking

• Synchronous communication operations
oInvolves only those tasks executing a communication operation

o When a task performs a communication operation, some form of coordination is required with the other task(s) participating in the communication. For example, before a task can perform a send operation, it must first receive an acknowledgment from the receiving task that it is OK to send.

o Discussed previously in the Communications section

Analysis: Barrier synchronisation is used for all the functions.

Data Dependencies:

Dependence exists between program statements when the order of statement execution affects the results of the program. A data dependence results from multiple use of the same location(s) in storage by different tasks. Analysis: find_trail() and trail_round() are having data dependencies and recursively depending because of round weights.

Granularity:

In parallel computing, granularity is a qualitative measure of the ratio of computation to communication. Periods of computation are typically separated from periods of communication by synchronization events. Fine-grained and coarse-grained granularities are two types of granularities.
Analysis: Code is embarrassingly parallel. So granularity is negligible.

I/O:

I/O operations are generally regarded as inhibitors to parallelism.I/O operations require an order of magnitude (or greater) amount of time than memory operations.Parallel I/O systems may be immature or not available for all platforms.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Analysis: Input is given only at the beginning. Output is taken from key_extract() after every processing 10000 cipher-plain text pairs for every possible combination of key. It creates considerable overhead if active SBoxes are more in last round.

VII. RESULTS:

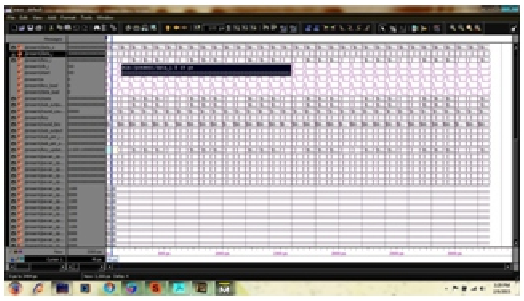


Fig5 : simulation result of 80 bit encryption

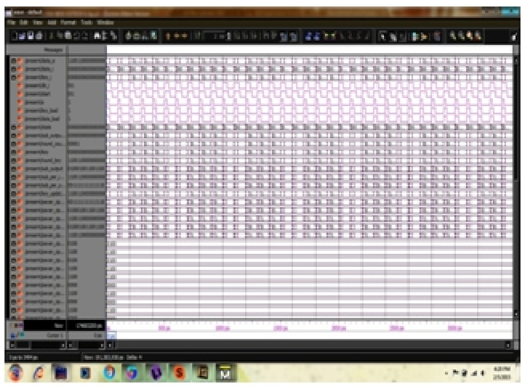
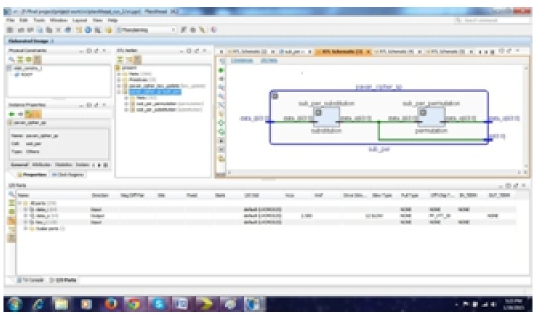
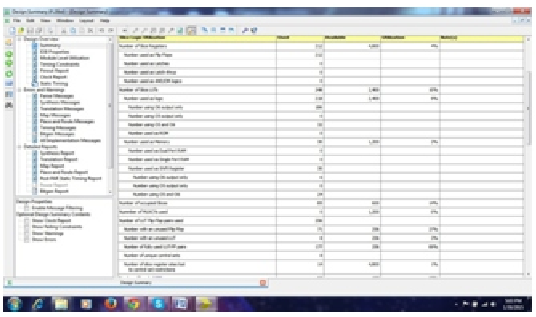
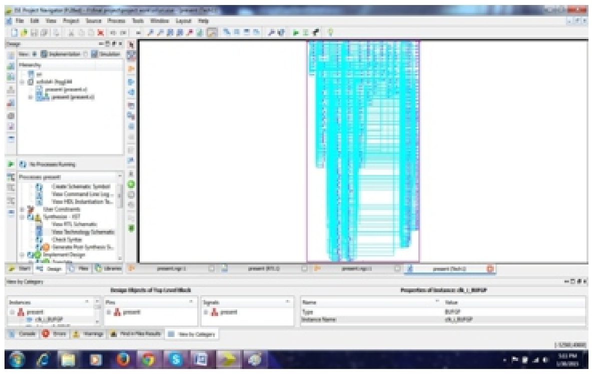


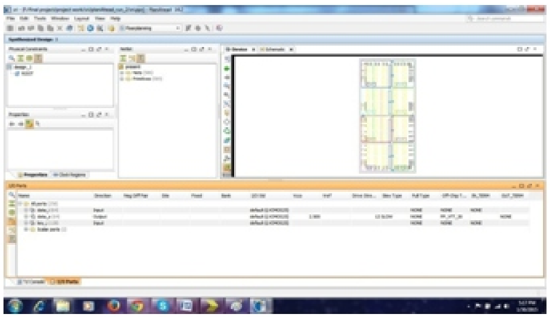
Fig6: simulation result of 128 bit encryption

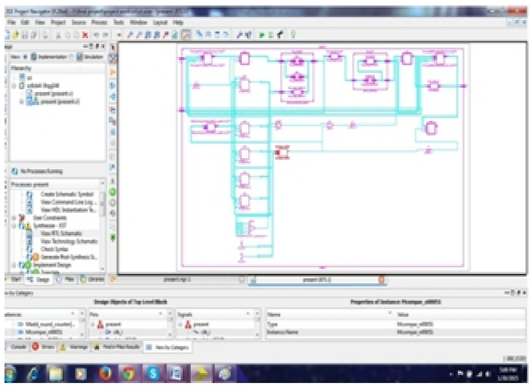
RTL SCHEMATIC











CONCLUSION AND FUTURE SCOPE:

Conclusion:

In this project, a SPN based 64-bit block cipher, which is called as sriram, and has 64-bit block size,27-rounds and takes 96 and 128-bit user given keys has been implemented. Both Encryption and Decryption are implemented and verified and generalized for both the keys. Then hardware optimized version of a SPN based 64-bit light weight block cipher, PRESENT, similar to Sriram cipher, has been implemented in Verilog and simulated. Then pipelined version of PRESENT-80 has been designed and implemented and simulated which gave performance improvement over Normal cipher-80.

Later, the Linear Cryptanalysis has been studied and the complete Linear Cryptanalysis module for block cipher has been implemented and generalized. In Linear Cryptanalysis, using Matsui's branch and bound algorithm for finding best linear approximations and improved performance in time from hours to seconds for four round network. An Application Program Interface (API) that is used to explicitly direct multi-threaded, shared memory parallelism, OpenMP, is used to parallelize Linear Cryptanalysis module, which gave decent improvement in performance but not as much as expected comparing with theoretical values.

Future Scope:

In this paper we have described the new block cipher. Our goal has been an ultra-lightweight cipher that offers a level of security commensurate with 64-bit block size and an 80-bit key. Intriguingly this cipher has implementation requirements similar to many compact stream ciphers. Like all new proposals, the immediate deployment of Present will be discouraged but its analysis will be encouraged strongly.

In Matsui's Branch and Bound algorithm, the active number of S-Boxes will increase with the increase in number of rounds. As they increase, the key extraction process may take large amount of time. So, the algorithms which gives linear approximations with constant or slowly increasing number of active S-Boxes for more rounds with decent probabilities can be designed, which improves time performance of Linear Cryptanalysis.

REFERENCES:

[1] M. Matsui, "Linear Cryptanalysis Method for DES Cipher", Advances in Cryptology- EUROCRYPT '93 (Lecture Notes in Computer Science no. 765),Springer-Verlag, pp. 386-397, 1994.

[2] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems",Journal of Cryptology, vol. 4, no. 1, pp. 3-72, 1991.

[3] National Bureau of Standards, "Data Encryption Standard", Federal InformationProcessing Standard 46, 1977.

[4] M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard",Advances in Cryptology - CRYPTO '94 (Lecture Notes in Computer Scienceno. 839), Springer-Verlag, pp. 1-11, 1994.

[5] E. Biham and A. Shamir, Differential Cryptanalysis of the Data EncryptionStandard, Springer-Verlag, 1993.

[6] National Institute of Standards, Advanced Encryption Standard (AES) web site:www.nist.gov/aes.

[7] J. Daemen and V. Rjimen, "AES Proposal: Rijndael", First Advanced EncryptionStandard (AES) Conference, California, Aug. 1998. (See also [6].)

[8] H.M. Heys and S.E. Tavares, "Substitution-Permutation Networks Resistant toDifferential and Linear Cryptanalysis", Journal of Cryptology, vol. 9, no.1,pp. 1-19, 1996.

[9] L. Keliher, "Linear and Differential Cryptanalysis of SPNs", unpublished.

[10] L. Knudsen, "Block Ciphers: A Survey", State of the Art in Applied Cryptography:Course on Computer Security and Industrial Cryptography (Lecture Notes inComputer Science no. 1528), Springer-Verlag, pp. 18-48, 1998.

[11] D.R. Stinson, Cryptography: Theory and Practice, CRC Press, 1995.

[12] B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C,2nd ed., John Wiley & Sons, 1995.

[13] A. J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of AppliedCryptography, CRC Press, 1997.

[14] W. Stallings, Cryptography and Network Security: Principles and Practices,2nd ed., Prentice Hall, 1999.

[15] H. Feistel, "Cryptography and Computer Privacy", Scientific American, vol. 228,no. 5, pp. 15-23, 1973.

[16] K. Nyberg, "Linear Approximations of Block Ciphers", Advances in Cryptology- EUROCRYPT '94 (Lecture Notes in Computer Science no. 950),Springer-Verlag, pp. 439-444, 1995.

Author details :

Sri Ramudu:

Received his B.TECH degree in Electronics & Communication Engineering from Vardaman College of Engineer-ing &Technology, Hyderabad in the year 2011.He is currently pursuing her Masters in VLSI-System design from VNR Vignana Jyothi College of Engineering& Technology, Hyderabad. His research interests are VLSI implementation of cryptography.

G.Shanthi

Received her B.TECH degree in Electronics & Communication Engineering from JNTUH and obtained her Masters Degree VLSI system design from VNR Vignanajyothi Institute of Engineering and Technology Her area of research include Data communications, Error Detection Codes.