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

Trade-OFFS For Threshold Implementations Illustrated on AES

Syed Kareem Uddin
M.Tech (VLSI Design),
Department of ECE,
VIF College of Engineering and Technology,
Hyderabad, T.S, India.

Imthiazunnisa Begum
Assistant Professor & HOD,
Department of ECE,
VIF College of Engineering and Technology,
Hyderabad, T.S, India.

Abstract:
Embedded cryptographic devices are vulnerable to power analysis attacks. Threshold Implementations provide provable security against first-order power analysis attacks for hardware and software implementations. Like masking, the approach relies on secret sharing but it differs in the implementation of logic functions. While masking can fail to provide protection due to glitches in the circuit, Threshold Implementations rely on few assumptions about the hardware and are fully compatible with standard design flows. We investigate two important properties of Threshold Implementations in detail and point out interesting trade-offs between circuit area and randomness requirements. We propose two new Threshold Implementations of AES that, starting from a common previously published implementation, illustrate possible trade-offs. We provide concrete ASIC implementation results for all three designs using the same library, and we evaluate the practical security of all three designs on the same FPGA platform. Our analysis allows us to directly compare the security provided by the different trade-offs, and to quantify the associated hardware cost.

I. INTRODUCTION:

The Advanced Encryption Standard (AES) is an encryption standard chosen by the National Institute of Standards and Technology (NIST) in 2001, which has its origin in the Rijndael block cipher. Several studies in the area had identified the nonlinear Sub Bytes transformation as the major bottleneck in achieving both small area and high speed VLSI AES implementations. This brief presents a methodology that is based on a pure combinatorial circuitry.

In which, the Galois inverse of elements in is computed prior using the composite field arithmetic (CFA). To date, there are several successful composite field constructions reported for AES S-box implementations. Summarizing from the previous works, the smallest composite field AES S-box is attributed to can right. However, the issue of critical path was not addressed in Can right's work. A short critical path is highly desirable in VLSI architectures, as it enables deep sub-pipelining for an increased performance in the clock frequency. On the other hand, the works of Zhang and Parham and contributed an AES S-box with the shortest critical path to date However, their work requires a larger area compared to Can right's. The remainder of this paper is organized as follows. In some details on the AES algorithm are discussed. In we explain our approach to minimize the area of the S-box and compare our new solution with the S-box of Satoh.

The current work improves on the compact implementation of in the following ways. Many choices of representation (isomorphism's) were compared, and the most compact turns out to use a normal basis for each subfield (uses a polynomial basis for each subfield). And while used the "greedy algorithm" to reduce the number of gates in the bit matrices required in changing representations, here each bit matrix is fully optimized, resulting in the minimum number of gates. These various refinements result in an S-box circuit that is 20% smaller, a significant improvement. The AES algorithm, also called the Rijndael algorithm, is a symmetric encryption algorithm; meaning encryption and decryption are performed by essentially the same steps.

It is a block cipher, where the data is encrypted / decrypted in blocks of 128 bits. (The original Randal algorithm allows other block sizes, but the Standard only permits 128-bit blocks.) Each data block is modified by several “rounds” of processing, where each round involves four steps. Three different key sizes are allowed: 128 bits, 192 bits, or 256 bits, and the corresponding number of rounds for each is 10 rounds, 12 rounds, or 14 rounds, respectively. From the original key, a different “round key” is computed for each of these rounds. For simplicity, the discussion below will use a key length of 128 bits and hence 10 rounds. There are several different modes in which AES can be used. For some of these, such as Cipher Block Chaining (CBC), the result of encrypting one block is used in encrypting the next. These are called feedback modes, and the feedback effectively precludes pipelining (simultaneous processing of several blocks in the “pipeline”). Other modes, such as the “Electronic Code Book” mode or “Counter” modes do not require feedback. These no feedback modes may be ipelined for greater throughput.

But for hardware implementations of AES, there is one drawback of the table look-up approach to the S-box function: each copy of the table requires 256 bytes of storage, along with the circuitry to address the table and fetch the results. Each of the 16 bytes in a block can go through the S-box function independently, the byte substitution step. This then effectively requires 16 copies of the S-box table for one round. To fully pipeline the encryption would entail “unrolling” the loop of 10 rounds into 10 sequential copies of the round calculation. This would require 160 copies of the S-box table, a significant allocation of hardware resources. In contrast, this work describes a direct calculation of the S-box function using sub-field arithmetic, similar to while the calculation is complicated to describe, the advantage is that the circuitry required to implement this in hardware is relatively simple, in terms of the number of logic gates required.

This type of S-box implementation is significantly smaller (less area) than the table it replaces, especially with the optimizations in this work. Furthermore, when chip area is limited, this compact implementation may allow parallelism in each round and/or unrolling of the round loop, for a significant gain in speed.

II. COMPACT HIGH-THROUGHPUT AES S-BOXES

Performance on Area:

Chip area is determined by the logic blocks, interconnections and the I/O pads. Routing area, area of diffusion, transistor size, and parasitic transistor capacitance are some of the important factors that affect the area of the device. Routing area is the most demanding factor of all, taking up to 30% of the design time and a large percentage of the layout area. Using the technology mapping approach, the routing area can be estimated by using two parameters available at the mapping stage; one is the fanout count of a gate, and the other is the "overlap of fan in level intervals”. Minimizing switching capacitance can reduce the size of the transistors. There are some techniques that can reduce the routing area such as the use of more metal layers routing interconnects and new technology to reduce λ size. Reducing λ can also reduce the area of diffusion and transistor size. The area of a circuit has a direct influence on the yield of the manufacturing process. Yield is defined as the number of chips that are defect-free in a batch of manufactured chips. The following is the yield formula to calculate the original yield of the memory array:

δ is the defect density
A is the area of the RAM array
α is some clustering factor of the defects
From the equation above we know that the smaller the chip area, the higher the yield. A low yield would mean a high production cost, which in turn would increase the selling cost of the chip.

Performance on Power:
There are three sources that cause power dissipation in a CMOS circuit:
• Dynamic power dissipation due to switching current
• Dynamic power dissipation due to short-circuit current
• Static power dissipation

Switching Current:
When the p-channel transistor charges the output capacitive load, CL, the current through the transistor is CL (dV/dt). The power dissipation is thus CLV(dV/dt) for one-half the period of the input. The power dissipated in the p-channel transistor is thus total switching power dissipation PD=CLVDD^2f

Short-Circuit Current:
Another source of power dissipation is during the transition of “0” to “1” or “1” to “0”, and both p-channel and n-channel transistors are on for a short period of time. This results in a short current pulse from VDD to GND that causes a short-circuit power dissipation. The short-circuit power dissipation is given by , In general, the transistor size of the p-channel and the n-channel are not the same to achieve the same rising time and falling time. The short-circuit current is also typically less than 20% of the switching current.

Static Power Dissipation:
Considering a CMOS gate, as shown in Figure 2.6, when the p-channel is biased “ON,” the n-channel will be “OFF”. On the other hand, when the n-channel is “ON,” the p-channel will be “OFF.” Since one of the transistors is always “OFF,” there should be no DC current from VDD to GND. However, there is a small leakage current between the diffusion and the substrate to cause the static dissipation.
Is= reverse saturation current
V = diode voltage
q= electronic charge (e= 1.6x10^-19 C)
k= Boltzmann’s constant (1.38X10^-23 J/K)
T = temperature.

The static power dissipation is produced by the leakage current and the supply voltage. Then the total static power dissipation is obtained from n = number of devices. Typical static power dissipation due to leakage for an inverter operating at 5 volts is between 1 and 2 nanowatts. The static power dissipation is generally negligible due to the low range compared to dynamic power dissipation.

III. ALGORITHM DESIGN:

The Rijndael Algorithm:
The Rijndael AES is a secret-key (symmetric) block cipher crypto-system which encrypts (or decrypts) one block of data at a time. The encryption algorithm accepts one data block (or plain text) and the key and produces the encrypted data block (the input and output data blocks are of identical size). The decryption algorithm accepts one encrypted data block and the key and outputs the plain text. Both encryption and decryption use the same secret key. Internally, the AES encryption algorithm can be partitioned into two processes, performed in parallel: encryption and key schedule. In the case where the AES encryption process is executed by a dedicated device (or crypto processor), these two processes can be viewed as the data path and the ntrol-path of the complete AES crypto processor. The decryption algorithm is similarly partitioned into the decryption and inverse key schedule processes. Encryption and decryption are mathematically inverse, as are key schedule and inverse key schedule.

AES Algorithm:
The AES is a symmetric block cipher, which encrypts 128-bit input blocks (internally stored in a 4 X 4 byte state matrix) and accepts key length of 128, 192, and 256 bits. It has an iterative structure: the operations computed at each round are fixed and predetermined, whereas the number of iterations depends on the key length (e.g., 10 rounds for a 128-bit key). All operations are byte-oriented, which makes AES a good choice for 8-bit architectures, although it can be also implemented in wider designs with higher

performance. A generic round is made of the following operations: 1. a nonlinear byte substitution (Sub Bytes) by means of substitution tables (S-Boxes) or online computations; 2. a data shuffling phase (Shift Rows) operating on rows; 3. linear multiplication in binary extension fields operating on the columns of the state (Mix Columns); 4. the addition with a round-dependent key. All rounds are identical, except for the last one, which lacks the linear multiplication for sake of symmetry, thus simplifying the decryption process. The specifications of the algorithm and of each component of the cipher are well known and fully detailed in the literature. The interested reader is invited to refer to for further details. The AES consists of two parts, the data procedure and the key schedule. The data procedure is the main body of the encryption (decryption) and consists of four operations, (Inv)Sub Bytes, (Inv)Shift Rows, (Inv)Mix Columns, and (Inv)Add Round Key. During encryption, these four operations are executed in a specific order—Add Round Key, a number of rounds, and then the final round.

The number of rounds is 10, 12, or 14, respectively, for a key size of 128 bits, 192 bits, or 256 bits. Each round is comprised of the four operations and the final round has Sub Bytes, Shift Rows, and Add Round Key. The decryption flow is simply the reverse of the encryption, and each operation is the inverse of the corresponding one in encryption. In the data procedure, the 16-byte (128-bit) data block is rearranged as a 4 X 4 matrix, called state S, where si denotes the ith byte of the data block. In this context, S denotes the input of an operation and T denotes the output. AES is operated in two fields, GF(2) and GF(2^8). In GF(2^8), addition is denoted by , and multiplication is denoted by EX-OR operation. Similarly, the two symbols, + and x, denote addition and multiplication in GF(2^8).

Sub byte and Inv sub byte Transformation:

The Sub Byte transformation is computed by taking the multiplicative inverse in GF (28) followed by an affine transformation. For its reverse, the Inv Sub Byte transformation, the inverse affine transformation is applied first prior to computing the multiplicative inverse. The step Involved for both transformations is shown below.

Sub Byte:

1. Multiplicative Inversion in GF (28)
2. Affine Transformation

Inv Sub Byte:

1. Inverse Affine Transformation
2. Multiplicative Inversion in GF(28)

The AT and AT-1 are the Affine Transformation and its inverse while the vector a is the multiplicative inverse of the input byte from the state array. From here, it is observed that both the Sub Byte and the Inv Sub Byte transformation involve a multiplicative inversion operation. Thus, both transformations may actually share the same multiplicative inversion module in a combined architecture. Switching between Sub Byte and Inv Sub Byte is just a matter of changing the value of INV. INV is set to 0 for Sub Byte while 1 is set when Inv Sub Byte operation is desired.

IV. IMPLEMENTATIONS OF THE S-BOX

One of the most common and straight forward implementation of the S-Box for the Sub Byte operation which was done in previous work was to have the pre-computed values stored in a ROM based lookup table. In this implementation, all 256 values are stored in a ROM and the input byte would be wired to the ROM's address bus. However, this method suffers from an unbreakable delay since ROMs have a fixed access time for its read and write operation. Furthermore, such implementation is expensive in terms of hardware. A more refined way of implementing the S-Box is to use combinational logic. This S-Box has the advantage of having small area occupancy, in addition to be capable of being pipelined for increased performance in clock frequency. The S-Box architecture discussed in this paper is based on the combinational logic implementation.

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

S-BOX CONSTRUCTION METHODOLOGY:

This section illustrates the steps involved in constructing the multiplicative inverse module for the S-Box using composite field arithmetic. Since both the Sub Byte and Inv Sub Byte Transformation are similar other than their operations which involve the Affine Transformation and its inverse, therefore only the implementation of the Sub Byte operation will be discussed in this paper. The multiplicative inverse computation will first be covered and the affine transformation will then follow to complete the methodology involved for constructing the S-Box for the Sub Byte operation. For the Inv Sub Byte operation, the reader can reuse multiplicative inversion module and combine it with the Inverse Affine Transformation, as shown. The individual bits in a byte representing a GF (28) element can be viewed as coefficients to each power term in the GF(28) polynomial. For instance, {10001011}2 is representing the polynomial.

q7 + q3 + q + 1 in GF(28).

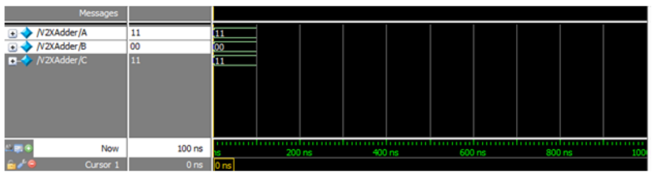
From it is stated that any arbitrary polynomial can be represented as bx + c, given an irreducible polynomial of x 2 + Ax + B.

SIMULATION IMPLEMENTATION GENERAL

Snapshot is nothing but every moment of the application while running. It gives the clear elaborated of application. It will be useful for the new user to understand for the future steps.

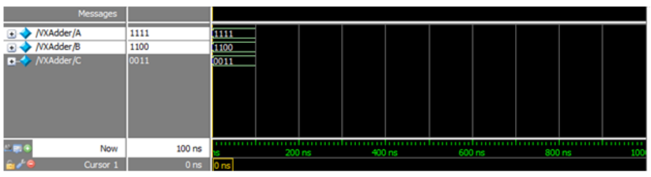
VARIOUS SNAPSHOTS

2 BIT ADDERS:



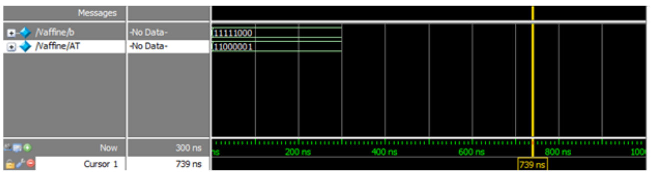
```
Messages
/N2XAdder/A    11
/N2XAdder/B    00
/N2XAdder/C    11
Now            100 ns
Cursor 1       0 ns
```

4 BIT ADDERS:



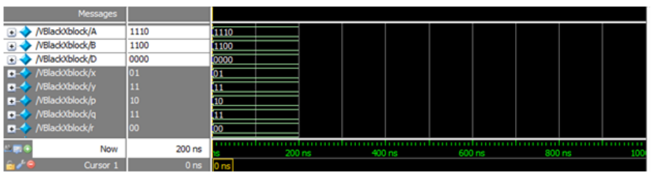
```
Messages
/NXAdder/A     1111
/NXAdder/B     1100
/NXAdder/C     0011
Now            100 ns
Cursor 1       0 ns
```

AFFINE TRANSFORM:



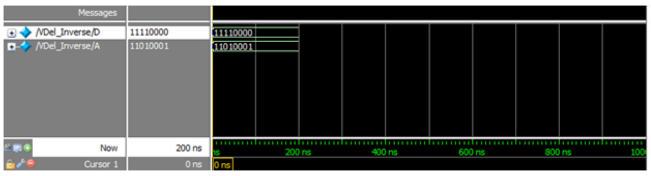
```
Messages
/Naffine/b     -No Data-    11111000
/Naffine/AT    -No Data-    11000001
Now            300 ns
Cursor 1       739 ns
```

BLACKXBLOCK:



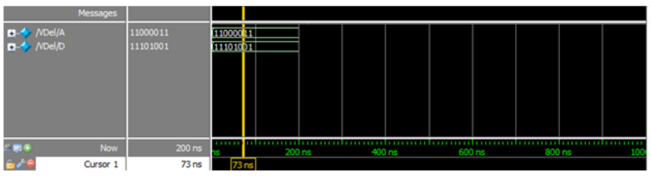
```
Messages
/NBlackXblock/A    1110
/NBlackXblock/B    1100
/NBlackXblock/D    0000
/NBlackXblock/x    01
/NBlackXblock/y    11
/NBlackXblock/p    10
/NBlackXblock/q    11
/NBlackXblock/r    00
Now                200 ns
Cursor 1           0 ns
```

DEL INVERSE:



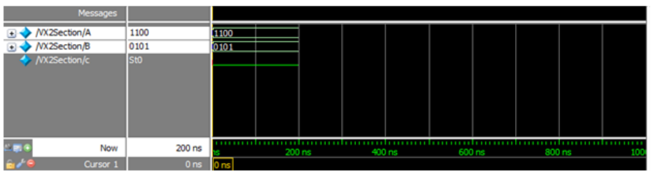
```
Messages
/NDel_Inverse/D    11110000
/NDel_Inverse/A    11010001
Now                200 ns
Cursor 1           0 ns
```

DEL:



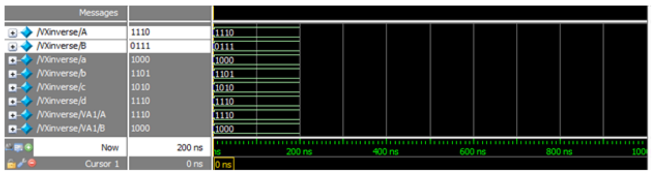
```
Messages
/NDel/A     11000011
/NDel/D     11101001
Now         200 ns
Cursor 1    73 ns
```

X2 SECTION:



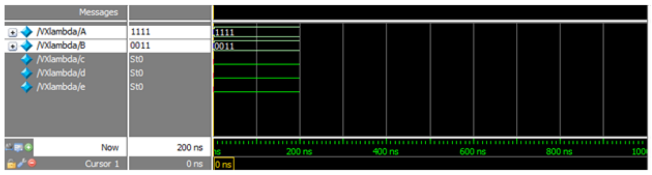
```
Messages
/NX2Section/A    1100
/NX2Section/B    0101
/NX2Section/c    St0
Now              200 ns
Cursor 1         0 ns
```

X INVERSE:



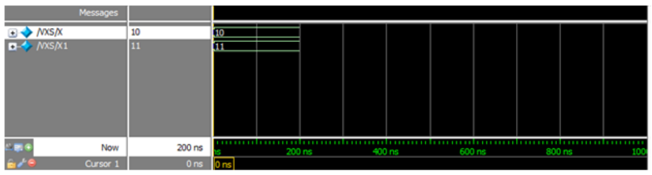
```
Messages
/Xinverse/A      1110
/Xinverse/B      0111
/Xinverse/a      1000
/Xinverse/b      1101
/Xinverse/c      1010
/Xinverse/d      1110
/Xinverse/VA1/A  1110
/Xinverse/VA1/B  1000
Now     200 ns
Cursor 1  0 ns
```

XLAMBADA:



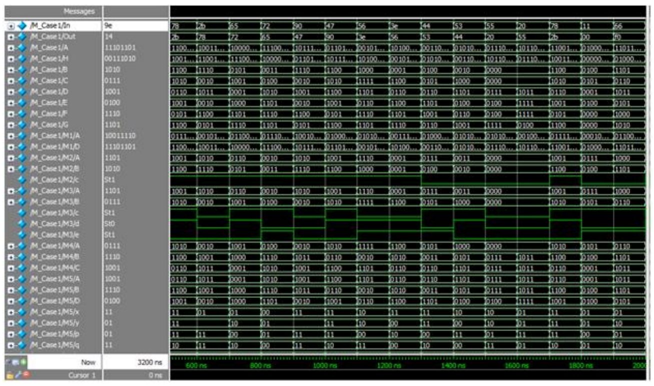
```
Messages
/Xlambda/A  1111
/Xlambda/B  0011
/Xlambda/c  St0
/Xlambda/d  St0
/Xlambda/e  St0
Now     200 ns
Cursor 1  0 ns
```

XA:

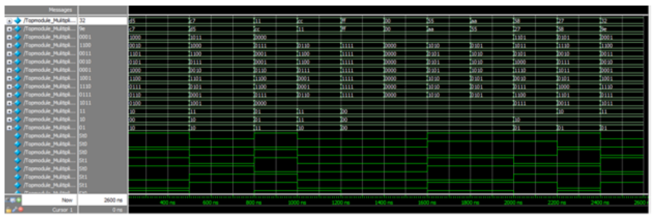


```
Messages
/XS/X   10
/XS/X1  11
Now     200 ns
Cursor 1  0 ns
```

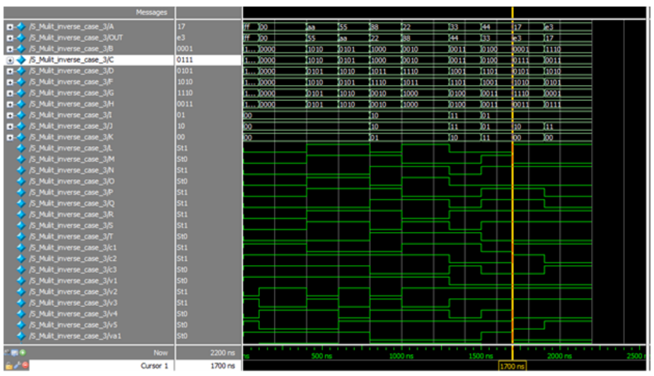
RAW IMPLEMETATION:



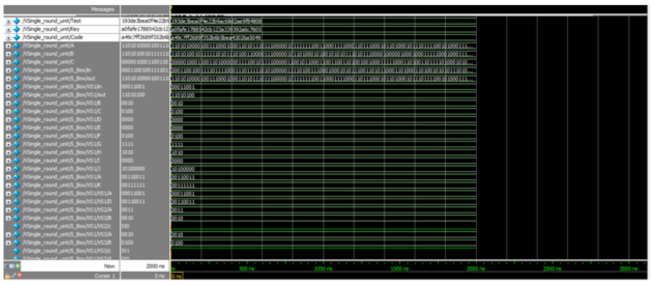
ADJUSTED IMPLEMETATION 2:



NIMBLE IMPLEMETATION:

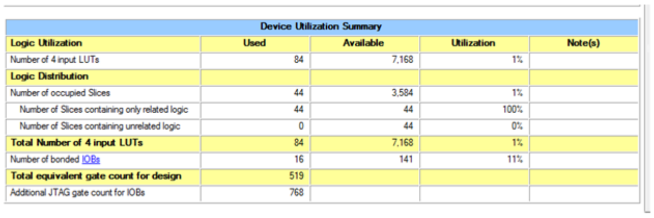


AES ENCRYPTION:



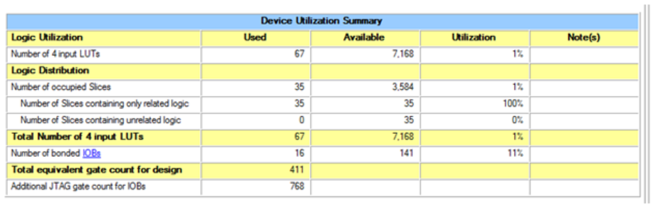
SYSTHESIS REPORT:

RAW IMPLEMENTATION:



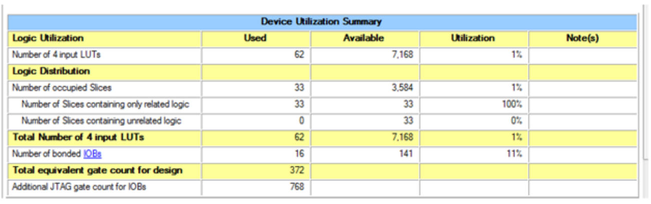
Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	84	7,168	1%	
Logic Distribution				
Number of occupied Slices	44	3,584	1%	
Number of Slices containing only related logic	44	44	100%	
Number of Slices containing unrelated logic	0	44	0%	
Total Number of 4 input LUTs	84	7,168	1%	
Number of bonded IOBs	16	141	11%	
Total equivalent gate count for design	519			
Additional JTAG gate count for IOBs	768			

ADJUSTED IMPLEMENTATION:



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	67	7,168	1%	
Logic Distribution				
Number of occupied Slices	35	3,584	1%	
Number of Slices containing only related logic	35	35	100%	
Number of Slices containing unrelated logic	0	35	0%	
Total Number of 4 input LUTs	67	7,168	1%	
Number of bonded IOBs	16	141	11%	
Total equivalent gate count for design	411			
Additional JTAG gate count for IOBs	768			

NIMBLE IMPLEMETATION:



Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	62	7,168	1%	
Logic Distribution				
Number of occupied Slices	33	3,584	1%	
Number of Slices containing only related logic	33	33	100%	
Number of Slices containing unrelated logic	0	33	0%	
Total Number of 4 input LUTs	62	7,168	1%	
Number of bonded IOBs	16	141	11%	
Total equivalent gate count for design	372			
Additional JTAG gate count for IOBs	768			

V. FUTURE SCOPE:

The detailed study on composite field construction for the S-box function in AES was presented. The major contribution of our work was the derivation of a new composite field AES S-box that achieves an optimally balanced construction in terms of area of implementation and critical path, compared to the previous studies. Furthermore, we had explored all of the possible isomorphic mapping for each of the composite field construction and employed a new CSE algorithm to derive the most optimum isomorphic and inverse isomorphic mapping with affine transformation. The best architecture obtained (i.e., Case III) possesses a total of 36 AND gates and 96 XOR gates with critical path of 4 AND gates and 20 XORs. Furthermore, we have found that there is a substantial gain in our CFAAES S-box in achieving a high throughput FPGA implementation.

VI. CONCLUSION:

The detailed study on composite field construction for the S-box function in AES was presented. The major contribution of our work was the derivation of a new composite field AES S-box that achieves an optimally balanced construction in terms of area of implementation and critical path, compared to the previous studies. Furthermore, we had explored all of the possible isomorphic mapping for each of the composite field construction and employed a new CSE algorithm to derive the most optimum isomorphic and inverse isomorphic mapping with affine transformation. The best architecture obtained (i.e., Case III) possesses a total of 36 AND gates and 96 XOR gates with critical path of 4 AND gates and 20 XORs. Furthermore, we have found that there is a substantial gain in our CFAAES S-box in achieving a high throughput FPGA implementation.

REFERENCES:

[1] S. Mathew, F. Sheikh, A. Agarwal, M. Kounavis, S. Hsu, H. Kaul, M. Anders, and R. Krishnamurthy, “53 Gbps native GF (24)2 composite- field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors,” in Proc. IEEE Symp. VLSI Circuits (VLSIC), 2010, pp. 169–170.

[2] V. Rijmen, “Efficient implementation of the Rijndael S-box,” 2000. [Online]. Available: http://ftp.comms.scitech.susx.ac.uk/fft/crypto/rijndael-sbox.pdf

[3] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, “Efficient rijndael encryption implementation with composite field arithmetic,” in Proc. CHES, 2001, pp. 171–184.

[4] J.Wolkerstorfer, E. Oswald, and M. Limburger, “An ASIC implementation of the AES S-boxes,” in Proc. RSA Conf., 2002, pp. 67–78.

[5] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A compact Rijndael hardware architecture with S-box optimization,” in Proc. ASIACRYPT, Dec. 2000, pp. 239–245.

[6] N. Men tens, L. Bateman, B. Greenland, and I. Verbauwhede, “A systematic evaluation of compact hardware implementations for the Rijndael S-box,” in Proc. Topics Cryptology (CT-RSA), 2005, vol. 3376/ 2005, pp. 323–333.

[7] D. Canright, “A very compact Rijndael S-box,” Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-MA-04-001, 2005.

[8] X. Zhang and K. K. Parhi, “On the optimum constructions of composite field for the AES algorithm,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 53, no. 10, pp. 1153–1157, Oct. 2006.