Abstract

Security and privacy of Integrated circuits (IC) is biggest challenge in front of semiconductor industry. Particularly, Intellectual Properties (IP) and IC's used in critical applications such as defence, banking, and network security are targeted by hackers. The title of our project "Design Methodology for Secure Test and IP Protection" points towards two different aspects regarding security concerns and they are "Secure Test" and "IP Protection". Our approach targets security measures for both these aspects at the same time. Testing of IC's during fabrication is extremely important with increasing number of transistor per chip when fabrication technology has gone down to twenty two nanometres. "Design for Test" (DFT) is the most popular and widely used method of testing. But it has proven to be a big security threat for IC's. Under "Secure Test", we have investigated how traditional DFT structures (scan chain) can be used to retrieve secret and proprietary information and based on this we are proposing countermeasures to improve security, without sacrificing testability. With our approach testing is possible only for the test vectors holding security key and any attempt to make use of scan chain without security key will result in randomise response from the chip. Intellectual properties are used as an integral part of many system-on-chip designs. The privacy of IP's has become headache for IP vendors. The use of third party foundries has increased rapidly to lower the overall manufacturing cost. This may result in unauthorised excess production or mask theft. To control this, we provide "IP Protection" technique by which only authorised users who hold the rights for IP can make use of it. Our approach provide variable key technique in which new key is generated on the fly in every new iteration and the user will not be able to use the IP without knowing the variable key sequence. We have designed a technique which fulfils both these concerns. We have shown, by utilizing single add-on-hardware we can provide IP protection as well as the existing testing method can be made secure. Both these aspects can be achieved with less area and power overheads.

Acknowledgements

I would like to sincerely thank Dr. Jimson Mathew for all the motivation, support and advice he has given me regarding the project. The guidance and freedom I received from him helped me in many ways and to learn a lot of things from this project.

I would like to thank Professor Dhiraj Pradhan for his advice and support in the project.

My heartfelt thanks to my friend Pranav Gadre who helped me in understanding TetraMax tool and Scan insertion flow.

Last but not the least I would also like to thank my parents, my classmates and my friends in Bristol for their support and encouragement throughout my MSc course.

Table of Contents

List of Figures:

List of Tables:

Chapter 1. Introduction

Security is one of the major concerns in the semiconductor industry. Designs are attacked by hackers to steal the secret information or to inject malicious data which alters their functioning. The Crypto chips are used to implement encryption and decryption algorithms at hardware level. Such chips are often attacked by hackers to crack the encryption key. Along with crypto chips, Intellectual Properties (IP) and smart cards are attacked by hackers to steal the secret information from the hardware. Lot of research is going on to detect such malicious attacks on the chip and to provide countermeasures against them. These attacks are made with various techniques such as power analysis, timing analysis, fault injection, spike attack, glitch attack, and optical attack.

Semiconductor manufacturing needs huge capital investment. Earlier all companies had their own fabrication labs. Therefore after designing, IC's were fabricated in their own company. To put down the cost, outsourcing fabrication from third party companies has rapidly grown. Qualcomm became the first fables company in 2007 [15]. This outsourcing gives exposure to mask theft or unauthorised excess production as every foundry is not trustworthy. Due to this privacy has become major concern for IP vendors. After lot of research and analysis, US defence science has pointed out that masks can be stolen by industrial and military spies [16]. Production of counterfeit IC's is done by using reverse engineering techniques or by breaking down IP privacy. This has become a huge issue in IC design community [17]-[20]. Sometimes chip design houses buy IP from IP vendors and after making minor modification, sell them as their own product [18]. Such un-trusted foundries make extra illegal production of IP's and sell it under different brand name [18]. In such scenarios it becomes important for IP vendors to have mechanism so that only licensed users should be able to use the IP.

The modern VLSI designs are becoming more and more complex to achieve maximum throughput from the chip. In doing so, the size of the chip is kept as small as possible by accumulating maximum number of transistors in a small area. This may result in faulty chips after fabrication. Because of this, the testing of chips is very critical. The testing engineers should use highly reliable and quick methods for testing. For this, the engineers should get good enough access to the chip for thorough verification. DFT is the most widely used process in industry for testing. It provides maximum controllability and observability over the circuit. High controllability allow engineers to put design in any desired state and observability provides the benefit of analysing different states of the design at any point of time. Therefore DFT (Scan chain) is the method which allows engineers to achieve very high percentage of fault coverage in a quick and reliable manner.

DFT (scan design) has become a security threat for intellectual properties and crypto chips. DFT properties such as controllability and observability over the chip, opens the back door for hackers to steal the information from the chip. DFT is proven to be a high risk for designs, which has been proved by an attack on DES crypto chip [1]. The secret encryption key can be easily revealed by using small number of plain texts. It is also quite easy to uncover the Intellectual property designs using DFT as it may give access to some important registers in the design. Therefore DFT has become a big concern for designers and testers.

DFT testing cannot be compensated for security reasons. If testing is not done properly, companies may end up manufacturing faulty chips which may cause a big loss. So it becomes very important to strengthen the existing hardware against such malicious attacks. We have considered the Advanced Encryption Standard (AES) as a bench mark circuit, announced by

National Institute of Standards and Technology. It was originally published as Rijndael [2]. Since AES is widely used in the cryptographic field, it always has some security threat. In this research we have analysed the AES design algorithm thoroughly to understand the loop holes and security threats, to design. To strengthen the design, we are presenting a secured and scan testable design method which aims to stratify both security and testability of the AES design. Our proposed method also provides IP protection with the use of variable key technique which makes reverse engineering very difficult.

After doing thorough research and analysis over these security threats we are proposing a new design solution for security concerns. We have proposed a new design method which provides counter solution for both "Secure Test" and "IP Protection" at the same time.

1.1 Aims and Objectives:

The basic aim of this project is to research and explore a better design methodology for secure testable hardware and IP Protection. We consider AES as a benchmark circuit. The designing focus is to implement efficient AES design, which is completely testable and invulnerable to security threats. Also, it should provide firm protection for IP against reverse engineering and IP piracy. Our focus is to investigate a flexible add-on circuit that can be smoothly integrated with the existing design flow. It should provide access for only licensed users and also should prevent scan based attacks without affecting the chip fault coverage and performance. The following are the objectives of this research project:

- Implement DFT (scan chain) based design for Advanced Encryption Standards.
- Analyse scan chain based attacks on advanced encryption standard hardware.
- Design a circuit which improves security without compromising its testability.
- Design method which provides solution for IP protection.
- Analyse power, area and simulation time overheads.

1.2 Outline of the Thesis:

We start with the background where we have explained different attacks made on the hardware. Scan chain based side channel attacks are described in detail. It is followed by AES algorithm which we are implementing. Then we have explained the necessity of DFT in detail. The next section describes, how encryption key can be revealed by using scan chain structure. Then previous work section gives idea regarding previously proposed methods for "secure test" and "IP Protection". After the Background section we describe our proposed approach in detail. Then we describe the overall design flow which is divided in to four phases. In the Implementation section, AES implementation is described at block level. Implementation is followed by detailed result and analysis.

Chapter 2. BACKGROUND

2.1 Security and Side channel attacks

Security has become a very important aspect for modern designs. It has become very crucial with the invention of crypto chips. Crypto chips are employed to perform encryption and decryption algorithms which are embedded in to hardware. Many researchers have proved that these chips are highly vulnerable to side channel attacks. These attacks can be made with various techniques such as power analysis, timing analysis and fault injection. If these attacks are not considered seriously, strong encryptions algorithms which may take several years to get cracked can be cracked in weeks, days or even in several hours. To steal the information or to break the algorithm it is important for hacker to have some access to attack the hardware. Theoretically he should be able to find the weakness in the algorithm. Such loop holes or weaknesses are known as side channel.

2.1.1 Side Channel

Side channel paves the way for accessing the chip by the attacker. Power and timing analysis are mandatory for any hardware. But these are considered to be side channels as they can leak the vital information to the attacker by creating a side channel. Some side channel can provide the knowledge of functioning of the chip e.g. the knowledge of algorithm in case of AES. But power and timing attacks can be considered as the black box attack. Without knowing the internal functionality, these can reveal the hidden information. Scan based attacks are also very common. The scan circuitry creates a back door for side channel attack. We have discussed these attacks in short.

2.1.2 Power analysis

Power dissipation in any circuit can’t be avoided. So in any circuit there will be power loss. By measurement of these power losses in hardware, the attacker could find the secret information. One of the methods of analysing hardware using power losses is described in [3]. In this method it is shown how the power analysis of a smart card can be measured. This is one of the common side channel attacks.

2.1.3 Timing Analysis

In timing analysis, the amount of time required to perform private key related operations is measured. With this information attackers try to steal the secret key. These attacks are inexpensive and just need a known cipher text. The method by which the crypto algorithm can be broken is discussed in [4]. Sometimes it becomes important to have new protocols and algorithms for preventing the timing attacks.

2.1.4 Fault Attacks

Fault attacks are performed on crypto hardware to yield faulty cipher text from which the secret key is calculated. These attacks can be termed as physical attacks. There are two purposes of these attacks.

- First is to induce error in the hardware.
- To modify the functionality of the hardware.

There are various types of these attacks.

- Spike attacks
- Glitch attacks
- Eddy current attacks which are also known as optical attacks.

In [5] these attacks are described in detail. These attacks are obvious because they don't need any physical contact with the hardware. There are many more error injections methods but these are the most common attacks as mentioned.

2.1.5 Scan Chain based attacks

The scan chain has become a big threat for security. Because of its properties, it provides a back door for side channel attacks. It has been proven as a potent side channel for attackers. In [2] it is described in detail that using hardly three plain texts the key of the Data Encryption Standard (DES) can be revealed easily.

The scan chain based attacks can be mainly categorised in to two types [7].The basic categorisation is made according to the method used by the attackers to apply the stimuli.

- Scan based observability attacks
- Scan based controllability attacks.

2.1.5.1 Scan based observability attacks:

A scan based observability attack depends on the hacker's ability to use scan chains for taking snapshots of the internal states at any time. It is as shown in Figure 1.

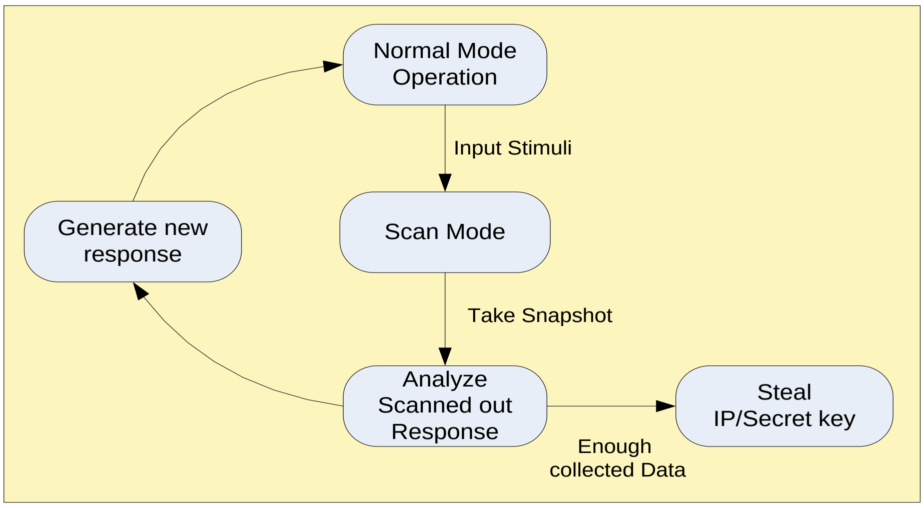

Figure 1: Scan based observability attack

The hacker begins the attack by observing the position of the critical registers in the scan chain. Initially some known vector is applied as input to the scan chain. Then the chip is allowed to run in the functional mode until the target register is supposed to have the data in it. At this point, chip is put into test mode and the response in the scan chain is shifted out. Then chip 'reset' is applied and new vector that will generate new response for target register is applied as input. The chip is again put into the functional mode for specific number of cycles and then set into test mode. The new response is shifted out and the result with the previous response is analysed.

2.1.5.2 Scan based controllability attacks:

A scan based controllability attack is as shown in Figure 2. It takes a different approach to apply stimuli to the circuit under test (CUT).

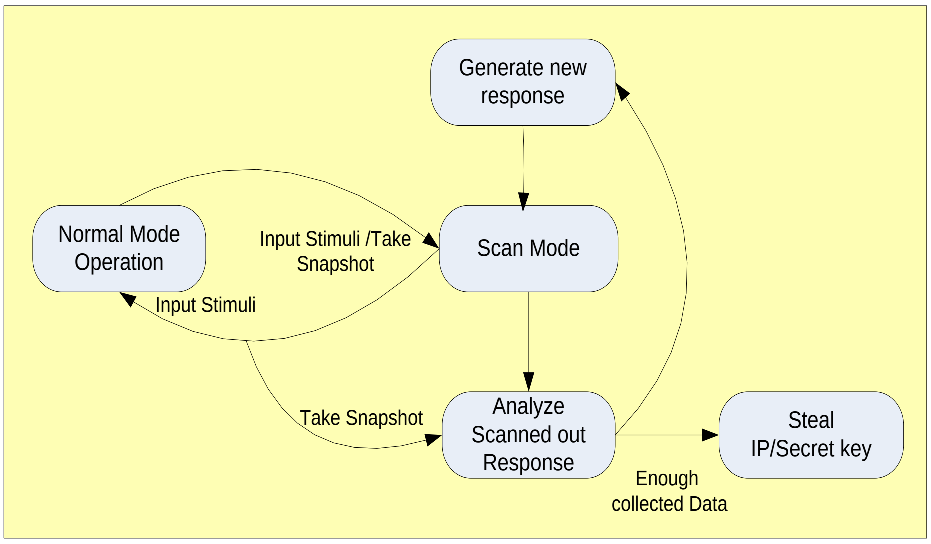

Figure 2 : Scan based controllability attack

It begins by applying the stimuli directly to the scan chain in test mode. In order to mount the successful attack, the hacker should determine the position of any critical register which is similar to scan based observability attacks. Once the hacker locates the target register, he can load the register with any desired data using test mode. Then the chip can be switched to the functional mode with the vector which hacker has already scanned in. Finally the chip is switched back to test mode which allows hacker to observe desired information at scan out port.

In this research we will focus on the Scan based side channel attacks. In DFT section we will see in detail the architecture of Scan based testing and how scan chains provide the internal states of the design. The further sections will describe how the scan chain can be used to reveal the secret information. We will also take a look at the Low Cost Secured Scan (LCSS) [7] which will make the existing scan chain more secure. We consider AES as a bench mark circuit. AES has always been prone to side channel attacks. We aim to make the existing AES algorithm more secure with the DFT testing feature. So it becomes important to know about the AES architecture.

2.2 Advanced Encryption Standard

The National institute of Standards and Technology (NITS) announced Advanced Encryption Standards (AES) [2] in 2001, which was originally published by Rijndael. AES algorithm is a symmetric key block cipher. Symmetric key means sender and receiver uses the same key for encryption and deception. Block cipher means the encryption operation is performed on group of bits. AES uses a block cipher of 128 bits. And key size can be 128, 192 or 256. AES operates on a 4X4 array of bytes which is called a State. The encryption process basically consists of 4 transformations:

- Sub Bytes
- Shift Rows
- Mix Columns
- Add Round Key

2.2.1 SUB Bytes

SUB Byte operation is basically a non linear byte substitution which is performed on every byte of a State. It is performed using a hardware called S- Box. This is very important part of AES. There are around 16 S- Boxes. Each of these S- Boxes works on 8 bits. Hence it provides 128 (16 x 8) bit transformations.

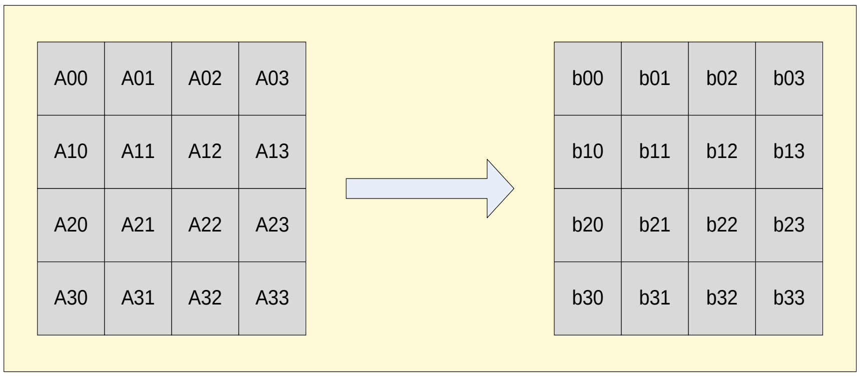

Figure 3 : Sub Byte Transformation

In this transformation, the multiplicative inverse of input is considered. In the above figure, the array to the right is the output. Figure 4 shows how the operation is performed on input data.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 4 : Substitution Equation

2.2.2 Shift Rows

This is quite a simple transformation. In this process the last three rows of the state are shifted. Figure 5 gives the complete idea of how the rows are shifted and the output of the process is obtained.

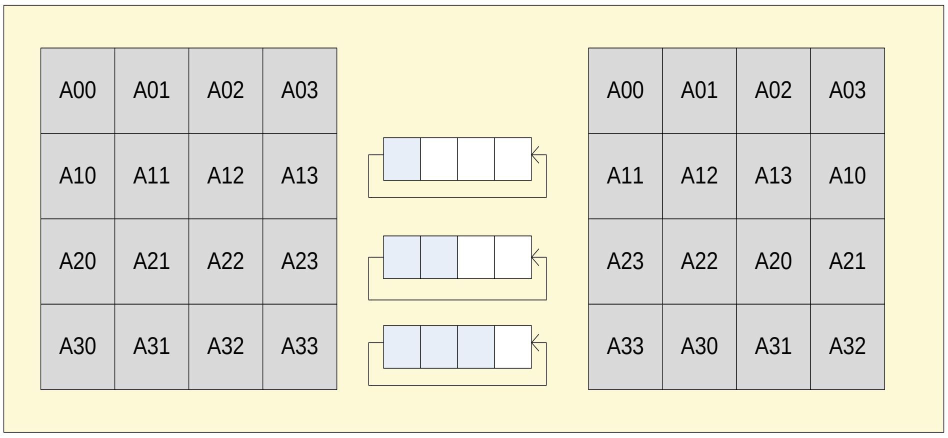

Figure 5 : Shift Rows

The diagram clearly shows that second row is shifted to the left by one. Third row is shifted to left by two and the fourth row is shifted to left by three places.

2.2.3 Mix Column

We have seen that the operation is performed on a state and not on the individual bytes. But in this transformation, all the four input bytes of a state undergo the operation. The following figure describes the operation.

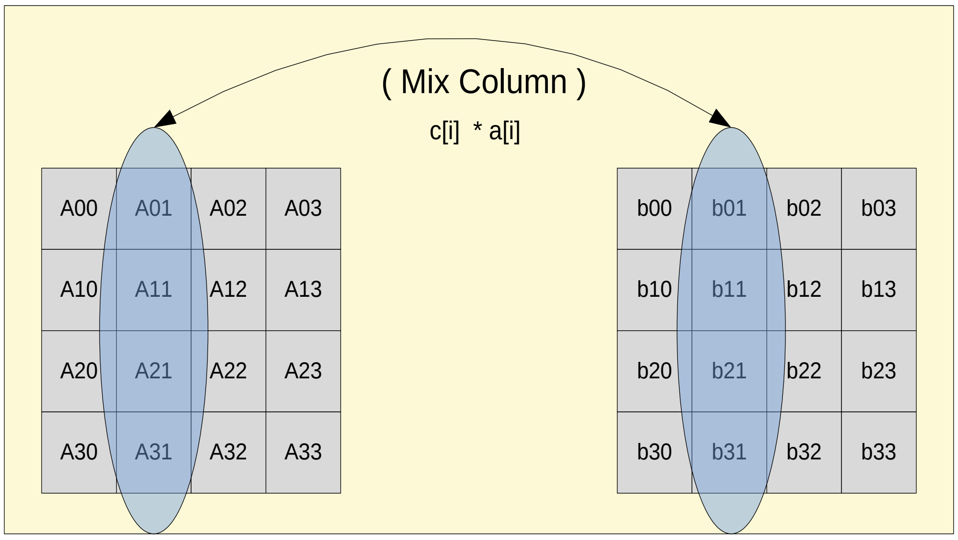

Figure 6 : Mix Column

Here the input matrix is considered as 4 x 1 i.e. the whole column. No change is made in the positions of the elements or a column, in this operation.

2.2.4 Add Round Key

In this method, the transformation is by ‘XORing’ a cell in the state with the SUB key. SUB key is generated by the key scheduler, it is 128 bits long. For the operation, this key can be considered as an array of 4 x 4. The following figure shows the add-round key operation.

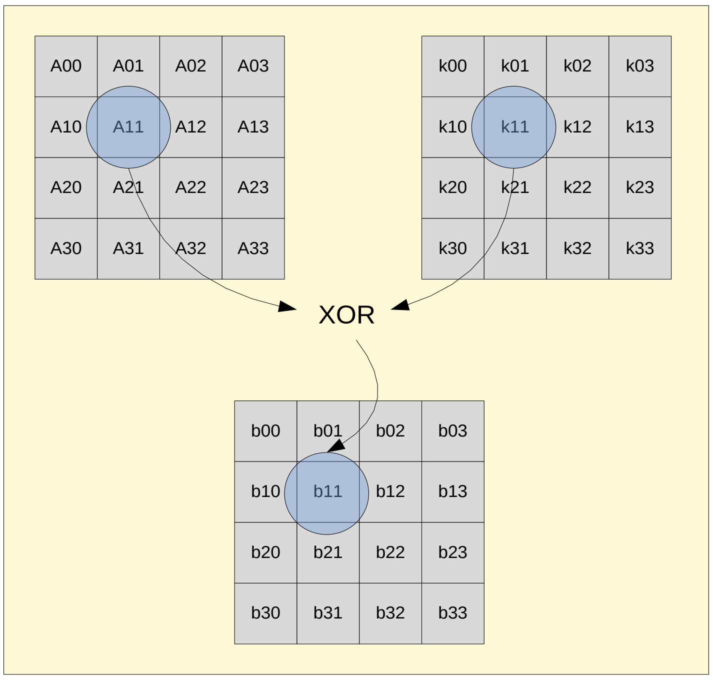

Figure 7 : Add Round Key

Figure 8 shows the flowchart for AES. Mix-column stage is not performed on last text.

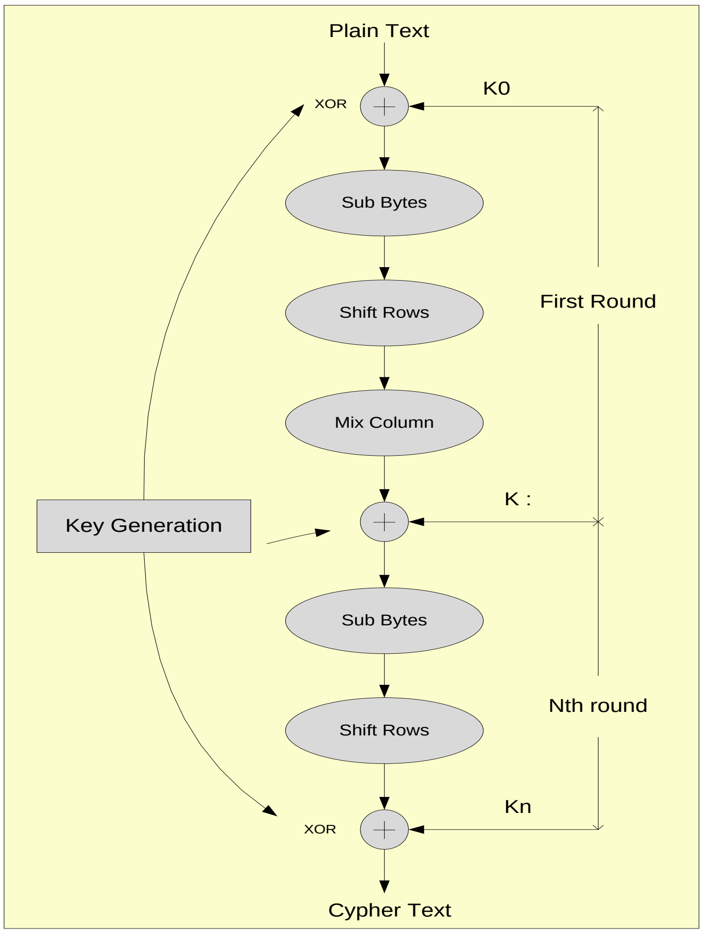

Figure 8 : AES algorithm flowchart

2.3 **Design for Test**

In the past, testability could be easily ignored when typical designs were restricted to a few thousand gates. Earlier testing engineers were given the designs only after complete implementation or on vendors' demand to run the test program before device production. But now a day's, for successful ASIC designs and manufacturing, the demand of DFT is increasing, which is incorporated in the process. A designer's approach will always be to presume that testability will have a negative impact on the performance of the design and also that it creates a lot of power and area overheads. But testability provides additional design constraints. These constraints are unimaginable if completely ignored until the design is fully complete. If testability is considered as a part of the design, it will add a value to the device throughout the manufacturing process and system life.

2.3.1 Test and Testability

There is a major difference between verification and test. In verification, the correctness of the design is tested whereas; test is used to verify the correctness of the manufactured hardware. Testing is applied to each and every individual manufactured device. This testing is completely responsible for the quality of design. So, testing and testability are highly essential phases of the design cycle. DFT provides the best test patterns, which can be used to test the hardware thoroughly with minimum effort. Including DFT in the design at an earlier phase of the design cycle ensures maximum testability with minimum efforts. Including DFT in the design affects the hardware as it adds additional logic which increases area and power. It increases the silicon required to implement the design but the benefits due to DFT cannot be truly identified until overall cycle time, cost of testing hardware and end systems are analysed.

One of the most important parts of DFT is Fault Simulation. This technique basically enables the tester to test whether a particular pattern is able to test the fault completely or not. Faults can occur in the design during the circuit fabrication phase. A fault can be defined as a physical defect in the chip. Usually a fault can be created because of broken wires or short circuited wires. This means that depending on the nature of the fault a particular signal will retain either logic '1' or logic '0' value. These faults are called as Stuck-at Faults. The response of the faulty circuit is compared with the response of the fault free circuit. If there is a mismatch between the responses of the two circuits, then it confirms that a particular pattern detects the fault.

By detecting faults at all the nodes in the circuit, fault simulator determines the fault coverage of the test pattern. The fault coverage can be defined as the percentage of total number faults detected from the total number of faults tested. If the fault coverage is very high then the pattern can separate the faulty and the fault free circuits clearly. Once undetected faults are analysed, other patterns are generated to cover those faults. If the fault coverage of the given set of patterns is very high, the probability of manufacturing fault free circuits increases.

2.3.1.1 Test- Time Cost

Testing cost is a very simple calculation as it is directly related to time. Highly commercial testers cost between $2 million to $3 million. The overall test cost including operator cost, plant and tester depreciation falls in the range of $1 to $2 per test second. Designs with millions of gates will have large number of patterns plus individual pattern lengths will be quiet high. For quality testing, test patterns are run at various temperatures and different power supplies. Inefficient patterns will have a very serious impact on the complex ASIC designs.

2.3.1.2 Time -to- Market

Various surveys indicate that test insertion and test pattern generation takes 40 percent of the total design cycle. The following figure shows the relation between fault grade and the time taken to generate effective test patterns. It also shows the comparison between the test strategy with and without DFT.

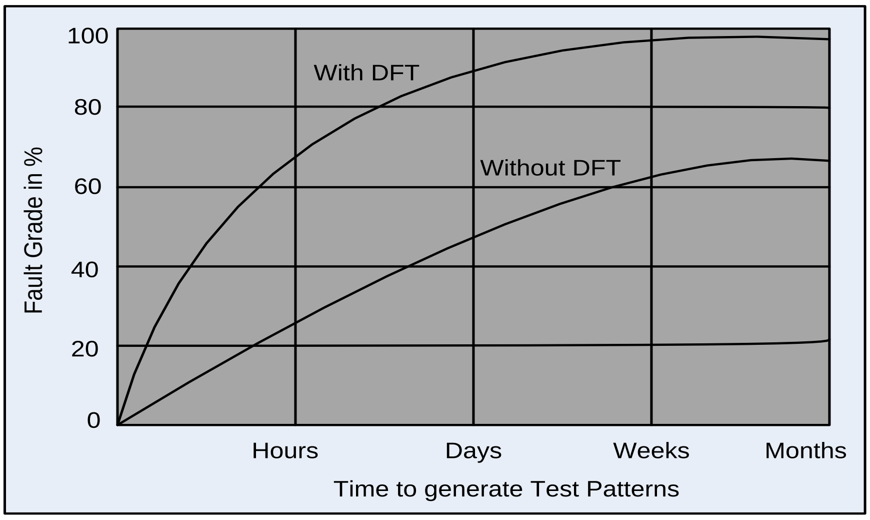

Figure 9: Fault grade verses development time

The basic motive behind incorporating DFT is to achieve very high fault coverage with optimum test patterns in less time. DFT helps to reduce time per cycle.

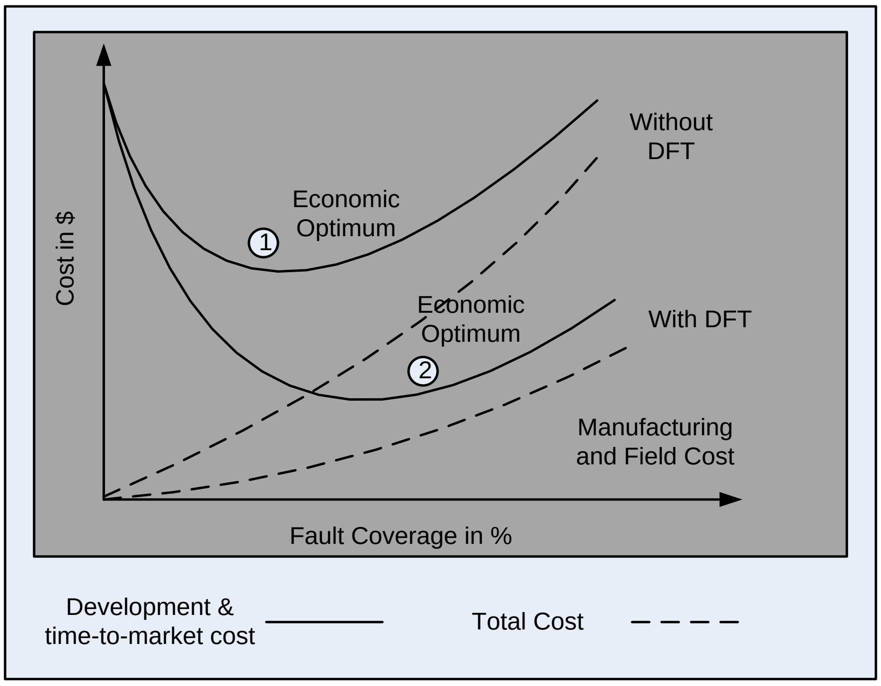

Figure 10 : Economic trade-off for a testable design

Figure 10 explains the trade-off between manufacturing and maintenances cost and time to market. Plot 1 in the figure explains the scenario where time to the market has given priority. The emphasis is given to the time and not to the quality of testing. The outcome of this is extra manufacturing and maintenance cost. Plot 2 shows that even if it takes a bit longer to come to the market, total cost is less. From the waveforms, we can easily make out that the devices with DFT are more suited for long term manufacturing and take less field maintenance cost.

Another advantage of DFT is ease in debugging, DFT reduces debugging time. ASIC designers consider some assumptions for system requirements. It is quite possible that designs may fail for certain system environments. In such circumstances debugging becomes crucial. FFT provides quality testing as the designs are built in such a way that they are highly controllable and observable. If these two features are not incorporated in the design, debugging becomes very difficult to accomplish.

These points highlight the importance of DFT in the ASIC manufacturing process. In further section we will take a detailed look at the elements used in DFT and the type of faults covered.

2.4 **Fault Models:**

A fault can be defined as a physical defect. Different conditions have to be considered to derive the tests to detect a fault. Fault abstraction helps in reducing the number of conditions. Based on the conditions, fault models can be distinguished in various types:

- Stuck-at fault model
- Toggle fault model
- Transition fault model
- Path delay fault model

2.4.1.1 Stuck-at fault model:

Stuck-at fault models are used to model the faults which occur when one or more ports of a gate get stuck to the power supply or ground. When the port gets stuck to the power supply it is considered as stuck-at-1 fault and when port gets stuck with the ground then the fault is considered as stuck-at-0 fault. A gate with ‘n’ input ports can have 2(n+1) faults.

Figure 11 shows a 2 input ‘AND’ gate with expected number of stuck-at-faults.

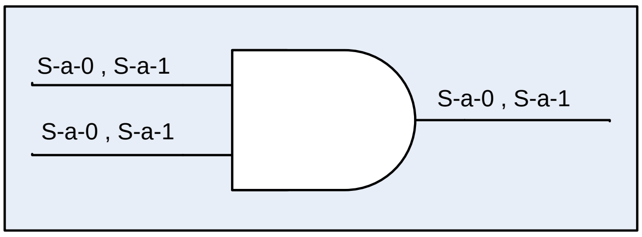

Figure 11: AND gate showing predicted faults

Table 1: Port A stuck-at-1

A	B	Y	Y(with fault)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	1

Table 2 :Port A stuck-at-0

A	B	Y	Y(with fault)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	0

Table 3 : Port B stuck-at-1

A	B	Y	Y(with fault)
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Table 4 : Port B stuck-at-0

A	B	Y	Y(with fault)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	0

Table 5 : Port Y stuck-at-1

A	B	Y	Y(with fault)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

Table 6 : Port Y stuck-at-0

A	B	Y	Y(with fault)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	0

The Stuck-at-fault model is the most popular model. It detects most of the faults. It can also detect bridging faults.

2.4.1.2 Toggle fault model:

Toggle fault model is used to cover the faults which check whether each node is toggling accurately. It checks if a node can be driven to both logic 0 and logic 1. It normally takes less overhead than the Stuck-at-fault model, it is also much faster.

2.4.1.3 Transition fault model:

This fault model is based on delays. It checks whether input changes get reflected on the output port within the allocated time or not. The following figures explain the passing and failing situations of AND gate simulation.

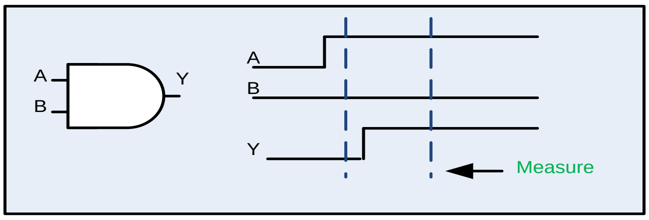

Figure 12 : And gate transition delay measurement

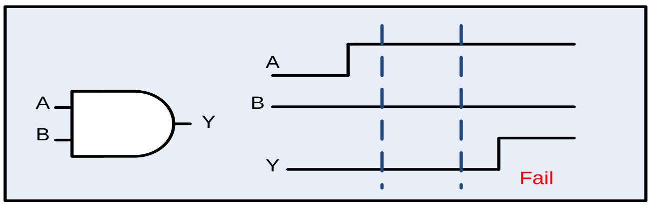

Figure 13: And gate transition fault fail situation

These faults involve two cycles, ‘Capture’ and ‘Launch’. The transition time between ‘launch’ and ‘capture’ is used to detect the fault. These models are useful to partially detect the conducting transistors.

2.4.1.4 Path Delay Fault Model:

These fault models are used to model the defects in the circuit path. A combinational path is modelled to check delays. It needs to have definitions for entry points (launch points) and end points (capture point). These faults test the lumped time delay caused by a particular path. Total delay in the path which stack up is summed together to calculate overall path delay. These models are useful to detect partially conducting transistors and diffusions.

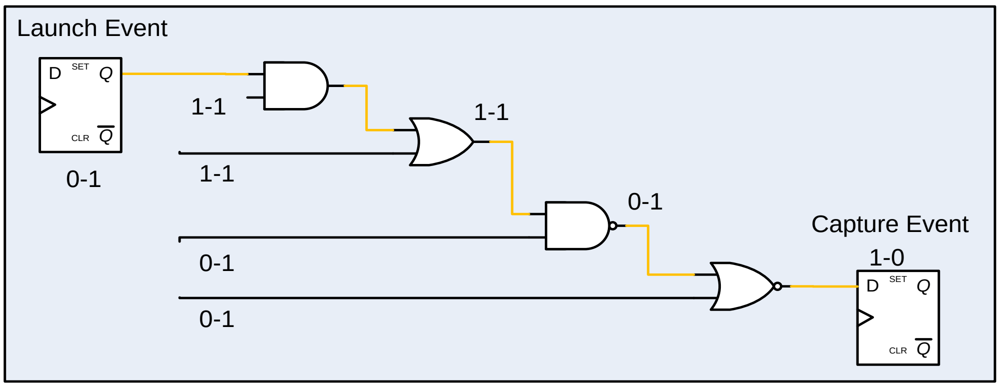

Figure 14: Path delay Fault Model

2.5 **Scan Chain:**

In the above section we have seen the necessity of DFT and the basic faults models which are used to test the chip. To model these faults in the DFT, normal flip-flops are replaced by scan flip-flops and a formation of scan chains. In this section we will look into the details of the scan chain. The following diagram shows a general design consisting of combinational and non combinational logic.

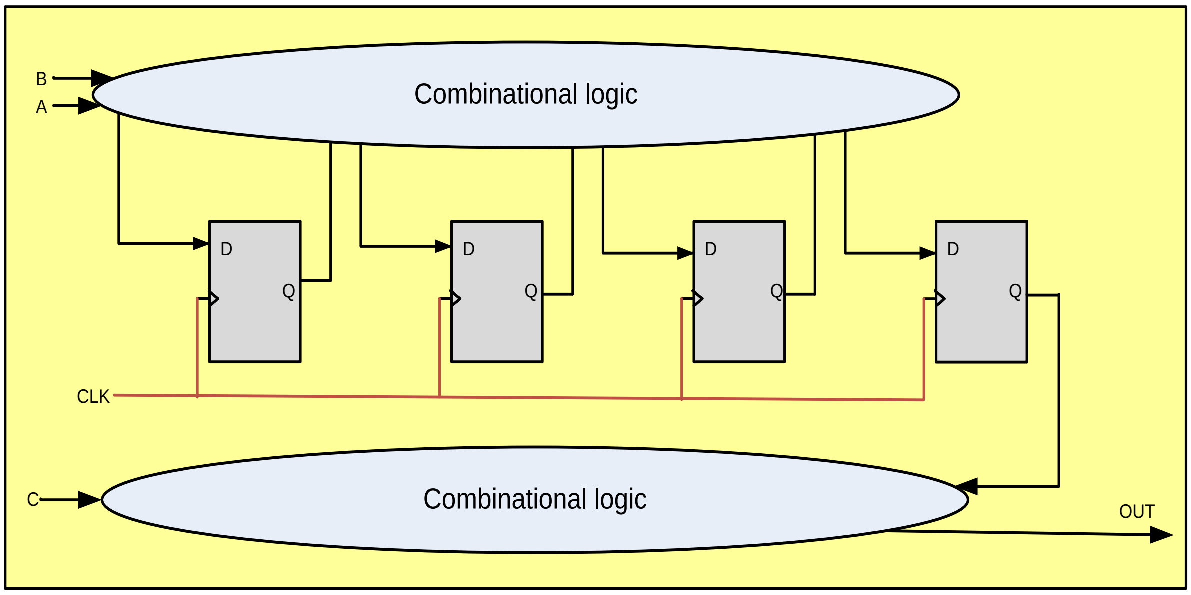

Figure 15 : Before Scan insertion

We can see that the combinational and non combinational elements are connected to form logic. There are primary inputs (A, B, C) and outputs (Out) as shown in the diagram. For a tester, this design is a black box. The tester can only see primary inputs and outputs. So from the tester's point of view there are a few drawbacks in the design.

- There is no access to the internal circuitry.
- It is difficult to initialize design to a known state.
- There is no access to observe the behaviour of the internal circuitry.

For a tester, any given design is not controllable or observable. To achieve the best fault coverage, controllability and observability of the design has to be improved.

2.5.1.1 Controllability

In order to make the testing more reliable and manageable, test engineers must have a method that would provide them access to the internal signals of the chip, i.e. it should provide better controllability. Controllability allows us to place the circuit under test (CUT) into any configuration and apply any stimuli during testing.

2.5.1.2 Observability

Even though testing engineers are given access to control the internal signals, it is important for them to check the response of the stimuli applied. Observability facilitates us to see the state of circuit at any point of time. It allows us to see the response of internal registers and various phases the design is going through. These drawbacks can be eliminated by using scan flip-flops and the formation of scan chains. The following diagram shows the orientation of the same circuit after insertion of a scan chain.

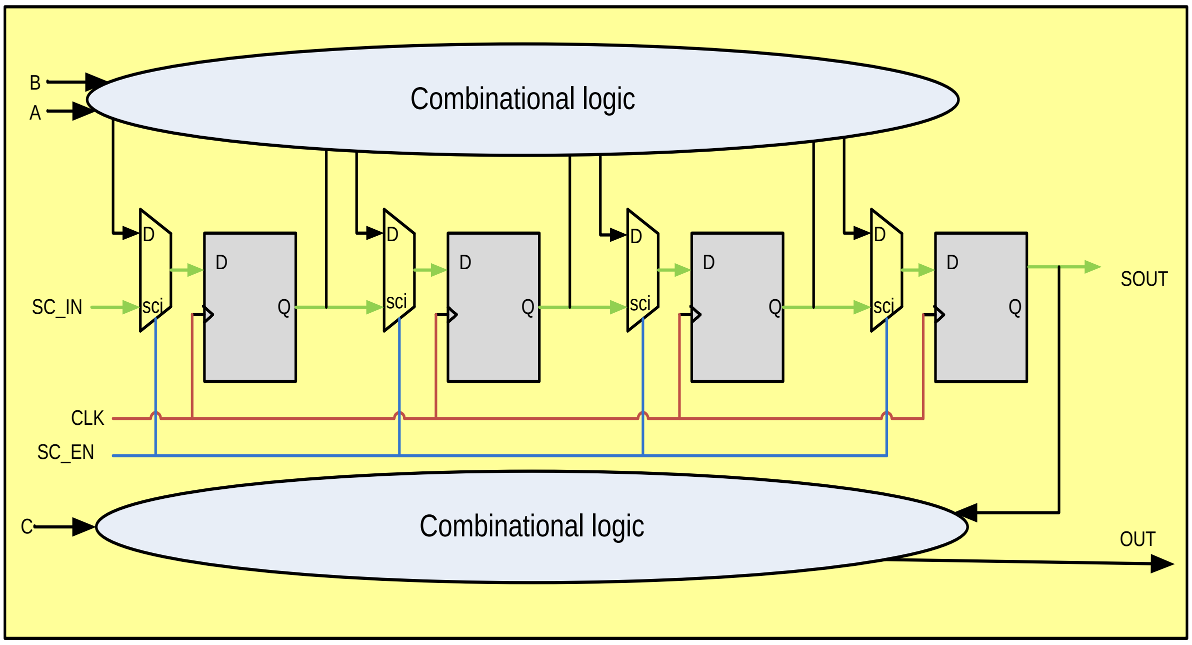

Figure 16 : After Scan insertion

With scan chain insertion, there is some additional hardware which is to be added in the existing circuit. Two additional inputs (SC-IN, SC_EN) and one additional output (SOUT) are added to the circuit. Scan flip-flop is nothing but an addition of a multiplexer at the input of a flip-flop. The additional input pin SC_EN acts as a select line for the multiplexer. With the addition of a multiplexer, every flip-flop has two inputs - regular 'D' and an extra input "sci". D input is used to capture the data from combinational logic; 'sci' input is used to form a chain of flip flops. With this addition, combinational logic in the design can be directly connected to the inputs of the flip-flops. As all flip flops are connected in series, the data present in any particular register can be easily shifted out and can be observed at the SOUT pin. Scan chain provides access to the combinational logic. With this, the internal state of any element can be observed at SOUT pin. In a similar way, the internal state of the design can be changed. Any internal signal can be controlled by inserting data through the SC_IN pin. So testing engineers can manipulate the values that are input (controllable) and observe the output (observable) of each block.

The test vectors are generated in such a way that they cover various faults in the design. Then test vectors are shifted in through the scan chain in multiple cycles and the response is captured and shifted out. The vectors are run in two processes, "Shift" and "Load". During "Load" the response from the combinational logic is captured in the sequential elements and during "Shift" the data is shifted out through the chain. Effectively sequential logic becomes the combinational logic during the test.

Creating the test patterns that achieve high fault coverage is a much easier task for testing engineers, as they have access to combinational as well as sequential logic in the design. This process speeds up the testing process by a significant margin and makes testing more reliable. Because of these benefits, even though there are area overheads, DFT is most popular and widely used method for testing complex ASIC designs.

Due its own qualities, DFT opens back door for hackers. By giving full access to the chip to modify internal states and provision to see the response, DFT makes the attacker's job very easy. In the next section we will see how the secret key can be revealed using scan chains.

2.6 **Discover a Secret Key**

B Yang [8] has shown how an AES 'encryption key' can be discovered from a hardware implementation. The following section describes key retrieval technique, with reference to this work.

The main characteristic of a scan based attack is to enable 'scan in' and 'scan out' operations at any point, particularly when the 'encryption key' is stored in internal registers. However, scan shift operations are restricted to 'scan test' mode, while data is processed during the functional mode. So the security hazard depends on the opportunity to switch from test mode to functional mode and vice-versa.

AES algorithm is used to encrypt 128-bit data using a 128 bit key. In both pipelined and iterative architectures, one round hardware is used. Figure 17 shows the block level representation of the round hardware. In first clock cycle, the plain text input is selected and the result after pre-round and round one of operation is stored in the register R as shown in Figure 17. The result of the pre-round is not registered as it is a simple bitwise XOR operation performed on the input data. Position 'a' in Figure 17 shows pre round.

2.6.1 Iterative method

While performing this operation for multiple iterations, the output of register R is taken as the input for the next round operation. The output of the register R is fed back to the S- boxes for next iteration at position 'b'. The AES round operation is performed ten times with different round keys taken each time.

2.6.2 Pipelined method

In case of pipelined architecture, the hardware for one round is replicated ten times. The output of the register R in the current round is applied to the S-box of the next round. The value of the register after the tenth round is the cipher text.

The round keys are either generated on fly or they are pre-computed. The RAM is used to store the pre-computed round keys. The data flow in AES is quite complex, but the control logic is very much straight forward. 'Round keys' or the 'secret encryption key' used by user is the main target for hackers. If this information is stored in the register, and these registers are part of the scan chain, then it becomes very easy to scan out the secret information from the AES chip.

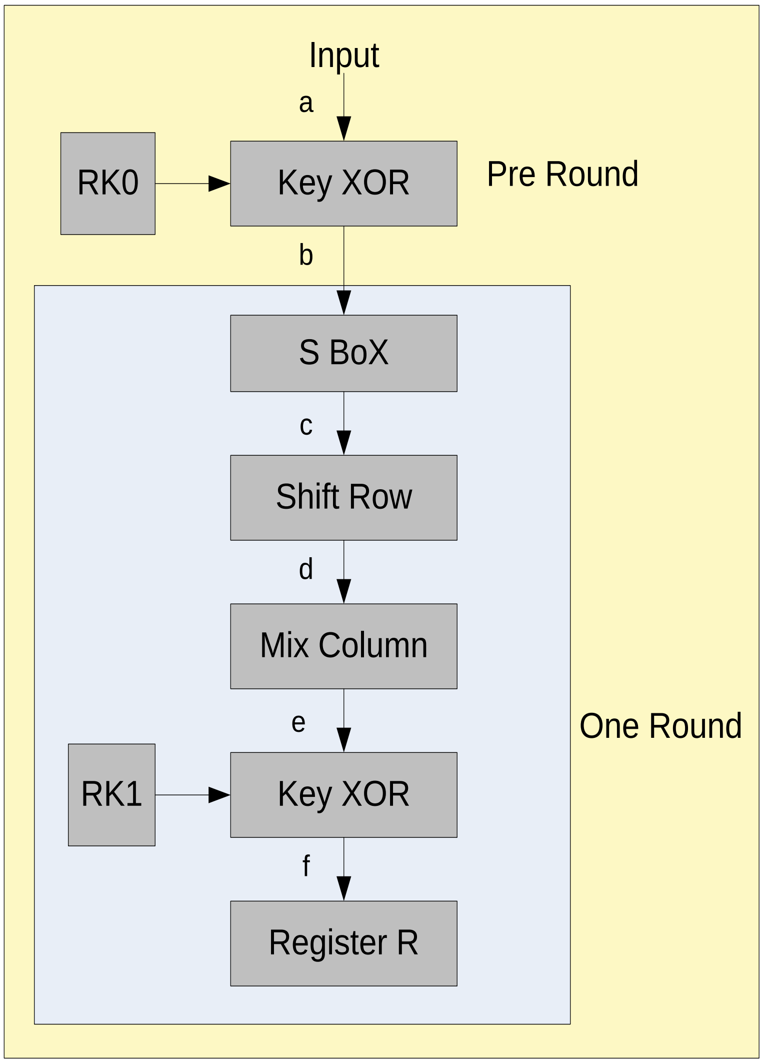

Figure 17 : Round operations on AES algorithm

Key can be retrieved in two major steps. First let's understand the structure of a scan chain. It is very important to know the location or the register holding the key. Once the structure is known, patterns are applied to recover the round key.

2.6.3 Determine the Scan structure

The intermediate cipher text after the pre-round and first round is stored in the register R. Even though register is a part of a scan chain, the attacker doesn't know which bits in the bit stream are forming the register R. By switching AES circuit to functional mode and testing mode this information can be decoded. Consider the SUB byte figure shown in AES algorithm. If we change byte a11, at point 'a' in figure 17, the output of the S box will change at position 'c'. Position'd' shows the output of the shift row operation. Position 'e' shows the output of mix column operation and finally at position 'f' we have the desired output of register R. The exact location of this register can be determined as follows.

1. Reset the chip and run in functional mode. The plain text will be processed in pre-round and round one. The intermediate result will be stored in register R. Now by

switching the chip in testing mode the one can scan out the bit stream. This is the pattern1.

2. By repeating the same steps for different plain texts we can collect different 256 patterns.

By comparing the values in different patterns corresponding to the flip-flops in register R, the exact location of the register R can be determined. In the worst case, one will have to go through all the 256 patterns to locate the exact position. But in [8] it has been shown in detail how to locate register with minimum combinations.

2.6.4 Recover the round key

As shown in figure 17, pre-round and round 1 are performed in one cycle and on two round keys - RK0 and RK1. One can chose the plain text and observe the corresponding word in register R. By repeating this step several times one can determine all the bytes in key RK0.

1. Apply bit pattern 2t (0<t<127) greater than 0 and less than 127 and run AES chip in functional mode for one clock cycle, shift to the test mode. Shift out the data and obtain the pattern f1.
2. Reset the chip and apply the pattern 2k+1 (0<k<127). Repeat the procedure in step 1 and obtain the pattern f2.
3. If number of 1's in f1 XOR f2 is 9, 12, 23 or 24 b11 can be determined, it is explained in detail in [8].
4. Determine the round key byte.

This process is repeated until all the bytes in the round key are known. The equations and the table of data comparison in [8] give a detailed understanding of this process.

In next section we will look at the some of the methods already proposed.

2.7 **Previous Work**

Several papers have been published to strengthen the existing scan architecture and to implement key authorization technique which will keep a check on the side channel attacks, made using scan chains. The main idea behind most of the proposals is to alter the chip response and make the attackers life tough. In case of IP piracy, it is important that IP vendors should have control over the use of their IP. To serve this purpose many ideas have been proposed. But one thing has to be understood that none of the approaches target both ‘Secure Test’ and ‘IP Protection’ together. We will discuss some of the proposed ideas.

Many approaches have been proposed for IP protection, some of the approaches are really interesting and useful. Locking of combinational logic is one the proposed methods [21]. In this approach, the main idea is the activation of hardware by insertion of unlocking key. This key is changed for every single chip. Unlocking key is provided only for the licensed users. Harpoon-Obfuscation at Netlist level is proposed in [22]. This method aims at Netlist level modification to implement security. In this approach, every chip needs to be activated by a specific input sequence. If activation doesn’t occur, response of the hardware changes randomly.

As scan chain reveals the internal states of the circuit, one can stop this by blocking this access. This is done in Built in Self Test (BIST). BIST can provide the testing by avoiding any access to hardware. One can easily stop the unauthorised access to the chip and can avoid any injection of malicious data in the chip using BIST. This is recommended for the security of the chip, also in [6]. But BIST can reduce the fault coverage marginally, compared to Scan based testing. Also, the testing circuit is embedded into the chip in the case of BIST. In case of faulty testing circuit there can be a huge mess with the overall testing procedure .So, for security reasons, it cannot be the ultimate solution [14]. Another approach is scan chain scrambling [9] and lock and key security [10]. In both these proposals the basic idea is to divide the existing scan chain in sub-chains. In scan chain scrambling, when the data is shifted out, the chains are randomly connected. One cannot understand the pattern unless and until there is any idea about the scan interconnects. In lock and key technique, sub-chains are connected in corresponding scan in and scan outs randomly when an unauthorised user tries to access the information. This can be checked with the authorised key. These approaches are quite effective but random scan chain connections cannot be implemented easily. Specific hardware need to be designed to do so, which is quite complicated and can cause the area overhead [11]. Sengar et al. [12] have proposed an idea of a model consisting of fluffed scan chains. In this approach, inverters are inserted in the scan chains for protection. The design follows conventional scan chains operation without the need of additional test keys hence can save the clock cycles. The architecture can also be tested the same way with scan chains and with additional NOT gates. However, despite the presence of sufficient number of inverters their position can be determined simply by shifting out the patterns after resetting all the flip flops in the scan chain. Thus, the internal state can be identified and security can be broken [13]. A secure scan approach proposed by Lee et al. [7] integrates a test key in the scan vectors .The dummy flip flops are added into the design which are used to hold this key. This key is verified during the scan operation. So, to have access one should know this key. If any unauthorized person tries to access the chain, the key verifier detects the attack and the response of the scan chain is randomly alerted.

2.7.1 Low Cost Secure Scan Approach

. .

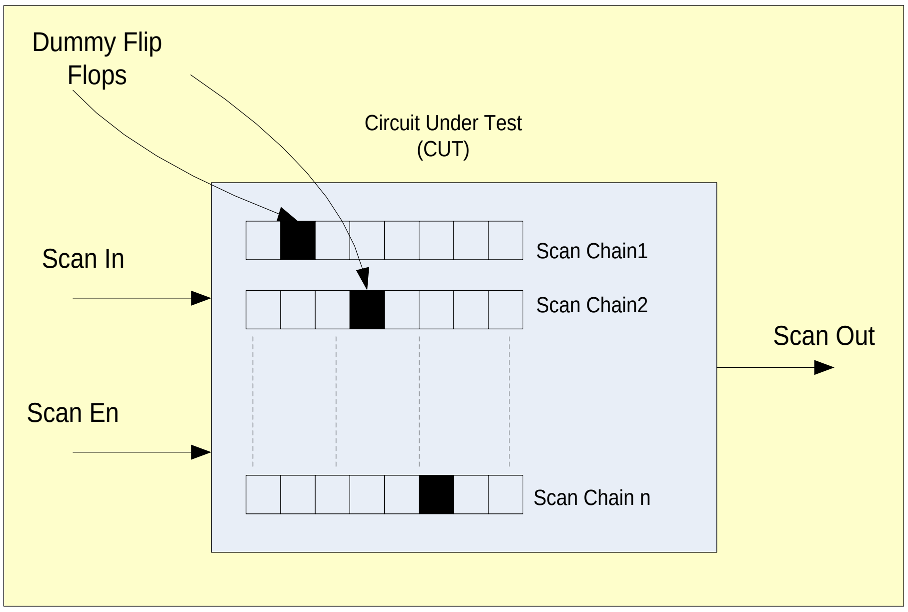

Figure 18 : General view of scan test with dummy flip flops

The above figure shows the general view of scan test. The dummy flip flops shown hold the secret key. The state of the scan chain depends on the inserted secret test key. The two states of the scan chain can be defined as secure and non secure. With the help of integrated test key, the scan vectors can be verified in the hardware, by a trustworthy source. This is a secure state of chains. In this case, the trustworthy source can get the desired response from the chain. When different test key is inserted along with the vectors, the response of the scan chain will be randomly changed. This can effectively prevent the reverse engineering of the critical data. This state can be defined as the insecure state of the chain. Attacks can be prevented because the attempt made to correlate the data received from the chain will be unsuccessful due to the randomization of the output data.

The LCCS architecture is as shown in Figure 19. The same test key is going to be used for every vector. So, the dedicated flops can be used to hold the key. These extra flip flops are inserted in the existing scan chain. They are similar to the rest of the scan flops; the only difference is that they don’t have any connection with the combinational logic. The number of dummy flip flops inserted depends upon the desired level of security as the number of flip flops determines the size of the test key.

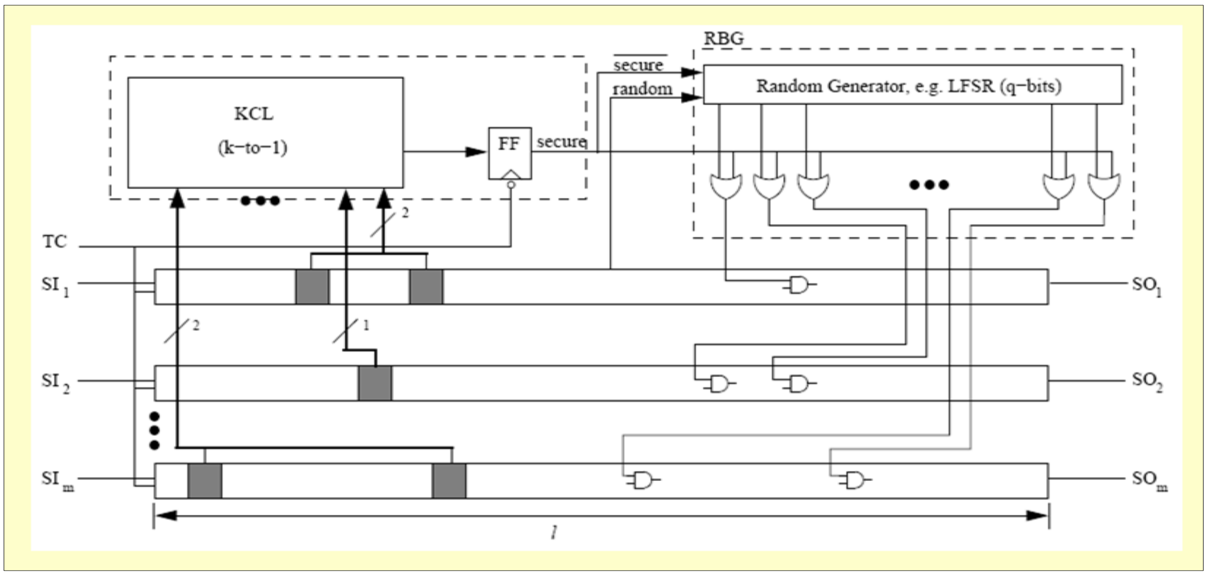

Figure 19 : Low cost secure scan design [7]

The key can be inserted even before the distribution of scan chains. Care has to be taken while distributing these dummy flip flops along the scan chain. All the flip flops are checked simultaneously. The data in these flops is checked using the key checking logic (KCL). The KCL unit as shown in the figure takes a k-bit input from all the dummy flops and gives a single bit output. The key verification is performed and accordingly the output of the block is asserted or disserted. The output of the KCL is negative edge sensitive with respect to the scan enable pin. So as soon as the CUT switches from the test mode to functional mode, it clocks the output. The output of this block informs the further logic whether the access is authorized or unauthorized.

The next block in the LCSS is the Random Bit Generator (RBG). This consists of a Linear Feedback Shift Register (LFSR). As shown in the figure, the output of the random response generator block passes through an array of OR gates. One input of these OR gates are directly coming from the output of KCL block. Therefore, in case of unauthorised access the output of the random sequence is inserted in the chain and hence the output of the scan chain will be altered. The Random logic block is formed by both, the OR gate array and LFSR.

Since dummy flip-flops are used to check the key, these must be before the random response network. If the property is violated, then any key information that is trying to pass a gate with random response will be altered by considering it as unauthorised access. In this way the LCSS can prevent any unauthorised access to the chip and can prevent the attacks made to steal the secret information.

Chapter 3. Proposed Architecture

As the project title suggests, there are two aspects to our implementation "Secure Test" and "IP Protection". In the previous chapter we discussed the many proposed methods which speak about Secure Test. Some of the works mentioned, show the security measures taken for IP protection. But none of the proposed methods include both Secure Test and IP protection.

Novelty: *The novelty in our approach is that we are targeting two scenarios, "Secure test" and "IP protection" simultaneously. Our proposed method provides both Secure Test and IP protection using the same hardware without much area and power overheads.*

3.1 Secure Test:

This approach is a bit similar to the previously proposed method "Low Cost Secure Scan" which is described in the previous chapter. The block diagram of our approach for secure test is shown in the figure below.

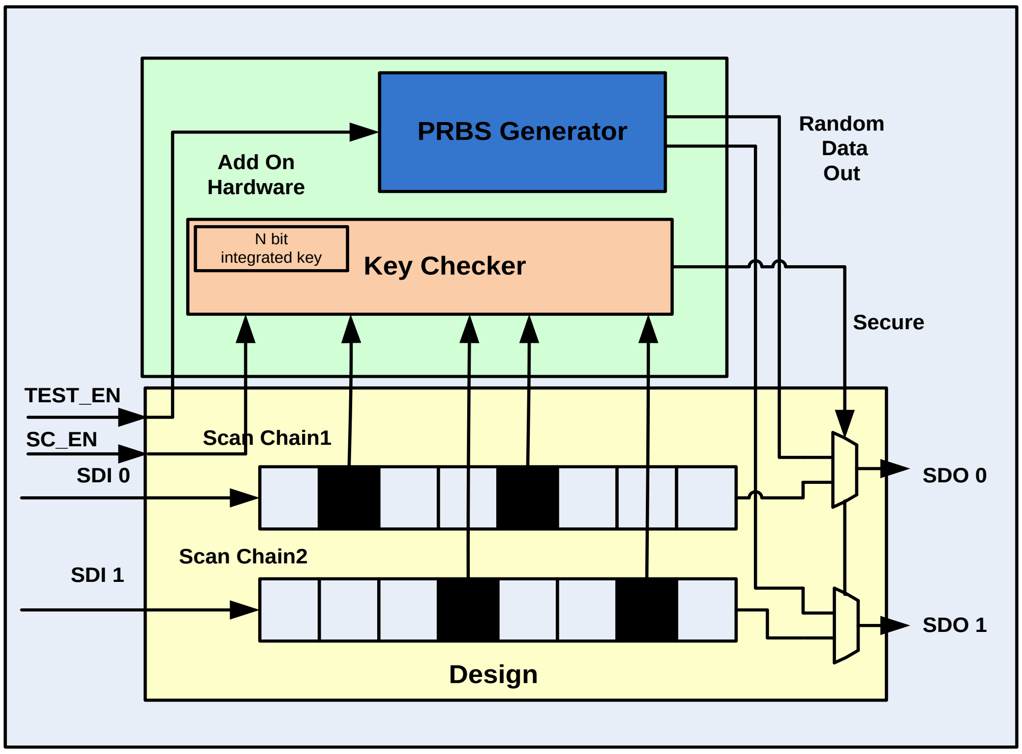

Figure 20 : Hardware Block for Secure Test

Figure 20 shows the design with inserted scan chain. To make the existing design secure, we have to make changes in the existing hardware. These changes are done at Netlist level. We break the existing scan chain and insert the Dummy flip flops randomly in the design.

Each dummy flip-flop is used to hold a single bit of secret key. For 'n' bit key we need to insert 'n' dummy flip flops. The add-on hardware shown in the figure consists of a 'key checker' and 'Pseudo Random Sequence Generator'. At the output of the scan chain one multiplexer is inserted which receives two inputs, one from random data generator and one from regular scan chain.

TEST_EN:

This input pin differentiates between Functional Mode and DFT mode.

TEST_EN = 1 -> DFT Mode, TEST_EN = 0 -> Functional Mode.

SC_EN:

This input pin is necessary for DFT mode and is used by scan flip-flops. Use of this input pin is mentioned under DFT section before. This pin is also used by 'Key checker' unit in the add-on hardware. This pin is de-asserted for every load operation. 'Key checker' checks the key only when SC_EN pin is de-asserted.

SDI0, SD1, SDO0, and SDO1:

These are inputs and outputs of the respective scan chains as illustrated in the figure 20.

Pseudo Random Sequence Generator:

This is a random generator which generates random number on every clock cycle when TEST_EN pin is asserted.

Key checker:

Key checker block holds hardcoded integrated secret key. It receives the input from all the dummy flops which is 'n' bit wide, depending on the size of the key.

3.1.1.1 Process Sequence:

1. Take the scan inserted Netlist.
2. Break the scan chain and insert the dummy flip-flops.
3. Connect dummy flip-flop to form shift register for IP security.
4. Integrate the add-on security hardware with the Netlist.
5. Hard code desired key in the key checker.
6. Generate the ATPG patterns with integrated key.
7. Simulate the patterns to check the desired outcome.

3.1.1.2 Function:

During DFT mode, the data is shifted into the scan chain and the response is checked at the output. The data shifting happens in "Load" and "Shift" cycles. During "Load", internal data from the combinational logic is taken into the chain and during "Shift" data is shifted out. The key checker reads the input from dummy flip-flops during every 'Load' operation. This input is verified with the secret key which is hardcoded in the 'key checker' block. In case of mismatch the "Secure" output from this block gets asserted. This output acts as a select line for the multiplexer connected at the output of the scan chain. This multiplexer receives two data inputs, one from scan chain and another from PRBS generator. When "Secure" pin is asserted, random data generated by PRBS goes to the output of the chain. By observing the simulation waveforms shown in "Result and Analysis" section, it is easier to understand the actual behaviour.

We integrate the secure key in the patterns generated for DFT. The patterns are generated in such a way that during "Load" operation dummy flip-flops hold the key. So with these patterns, chip can be tested completely with high fault coverage. With this add-on circuit DFT is possible only for the patterns with integrated key.

With this add-on circuit any unauthorised access to the chip can be detected. In such cases, output response from the chip changes randomly which makes any prediction very difficult. To make use of the chip for malicious purpose, the hacker must know the following things:

- Size of the secret key.
- Position of the dummy flops.
- The seed used for Pseudo Random Generator.

Without any prior knowledge of the above, hacking is almost impossible. It is impossible to get hold of these as response from the chip will be changing randomly. Also since these modifications are done at the Netlist level, it makes hacking at RTL level impossible.

The size of the security key can be chosen according to the security level that needs to be implemented. More number of dummy flops can be inserted to have a bigger key. The random generator can be made bigger to generate highly random pattern.

In the result and analysis section, we have shown the DFT simulation waveforms for both the passing and failing scenarios. When patterns with integrated key are used for the simulation, it passes without any error. When the patterns that do not have keys are simulated, unauthorised access is detected and simulation fails showing multiple errors. Detailed analysis shows increase in the simulation time which is because of dummy flips-flops.

In next section we will see how IP security can be obtained by reusing the same hardware and without making any changes in the current flow.

3.2 **IP Protection:**

The basic idea used for IP protection is the use of a new security key for new iterations. The use of a variable key is becoming popular as it is an effective way for providing protection. Figure 21 shows the block diagram for IP protection. As we are using the same hardware for both the modes, block diagram looks similar to the "Secure test" mode with few interconnect changes. This mode of operation comes into picture when design is in the functional mode.

The following are the fundamental changes in the add-on circuit compared to DFT mode.

- Dummy Flip-flops forms the shift register.
- PRBS is used as an internal key generator.
- Key generator checks for new key in every iteration instead of hardcoded key.
- SD0 (scan chain input) pin used as an input pin for shift register.
- PRBS generates new number only on specific condition and not on every clock.

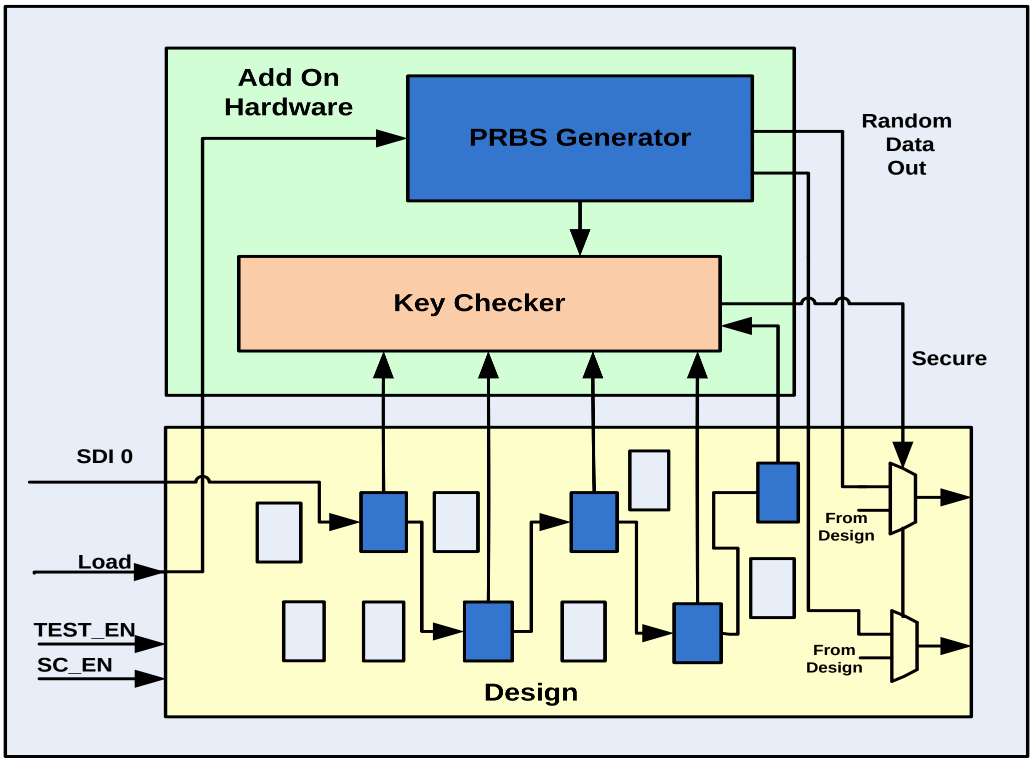

Figure 21 : IP Protection

3.2.1 Shift Register using Dummy flip-flops:

In functional mode, logically there is no use of dummy flip-flops as they are not a part of any functional logic. So these can be reused to provide protection in the functional mode. Every scan flip-flop has two input pins, 'D' input for functional mode and 'SC_IN' for scan mode. This has been described in detail in the DFT section. For rest of the flip-flops, 'D' input is connected to the logic and for dummy flip-flops this pin is floating as they are not attached to any logic. When we insert the dummy flip-flops, we connect their 'D' input pin in such a way that they form a shift register. The scan output pin (SD0) is connected to the 'D' input of the next dummy flip-flop and so on, forming a shift register. This is clearly shown in the figure 21.

3.2.2 Pseudo Random Sequence Generator:

Pseudo random generator generates random numbers which are considered as a key. This is generated on a specific event only. We have designed Advanced Encryption Standard hardware which has "Load" input pin which gets asserted indicating new iteration. Random generator generates new number for every assertion of this pin. So, effectively we have a new key during each iteration.

3.2.3 Key Shifting Protocol:

We have shift registers which can be used to hold the key. The new key has to be injected in for new iteration. For new iteration we shift in the key in the shift register on consecutive clock cycles. For 'n' bit key the data should be injected in on 'n' consecutive cycles. After 'n' cycles shift register hold the security key.

3.2.4 Function:

When the TEST_EN pin is de-asserted, the chip is supposed to be in functional mode. As described earlier we have variable keys getting generated inside the chip. The same key has to be injected in as per the key shifting protocol. Key checker compares both the keys and in case of mismatch it asserts its output pin "Secure" which acts as a select line for the multiplexer provided in the output path. In case of DFT mode, a multiplexer is provided on scan chain output. In case of IP security the multiplexers can be provided in the path of any functional outputs. The two inputs for the multiplexer are from random generator and the actual functional output. In case of mismatch random data is pumped out of the chip.

This method provides high level of IP security as random response makes reverse engineering very difficult. We are providing variable key approach which makes the key determination or prediction very tough. The IP vendor can provide key sequence to the authorised and genuine users only. So even in case of IP stealing or unauthorised IP production only the user having key sequence can use the chip.

For malicious access hacker should know following things.

- The variable key sequence.
- The protocol to shift in the key.
- The seed used for Pseudo Random Generator.

Without the prior knowledge of the above, reverse engineering or illegal use of IP is impossible. The key sequence can be made really complex and also the protocol to shift in the key can be varied to improve the security even further.

In the Result and Analysis section, we have provided waveform simulation for both passing and failing conditions. It clearly shows that if the genuine variable keys are shifted in, the simulation passes without any error and in case of false key injection, simulation fails as output response randomly changes.

Chapter 4. IMPLEMENTATION

4.1 Design Flow:

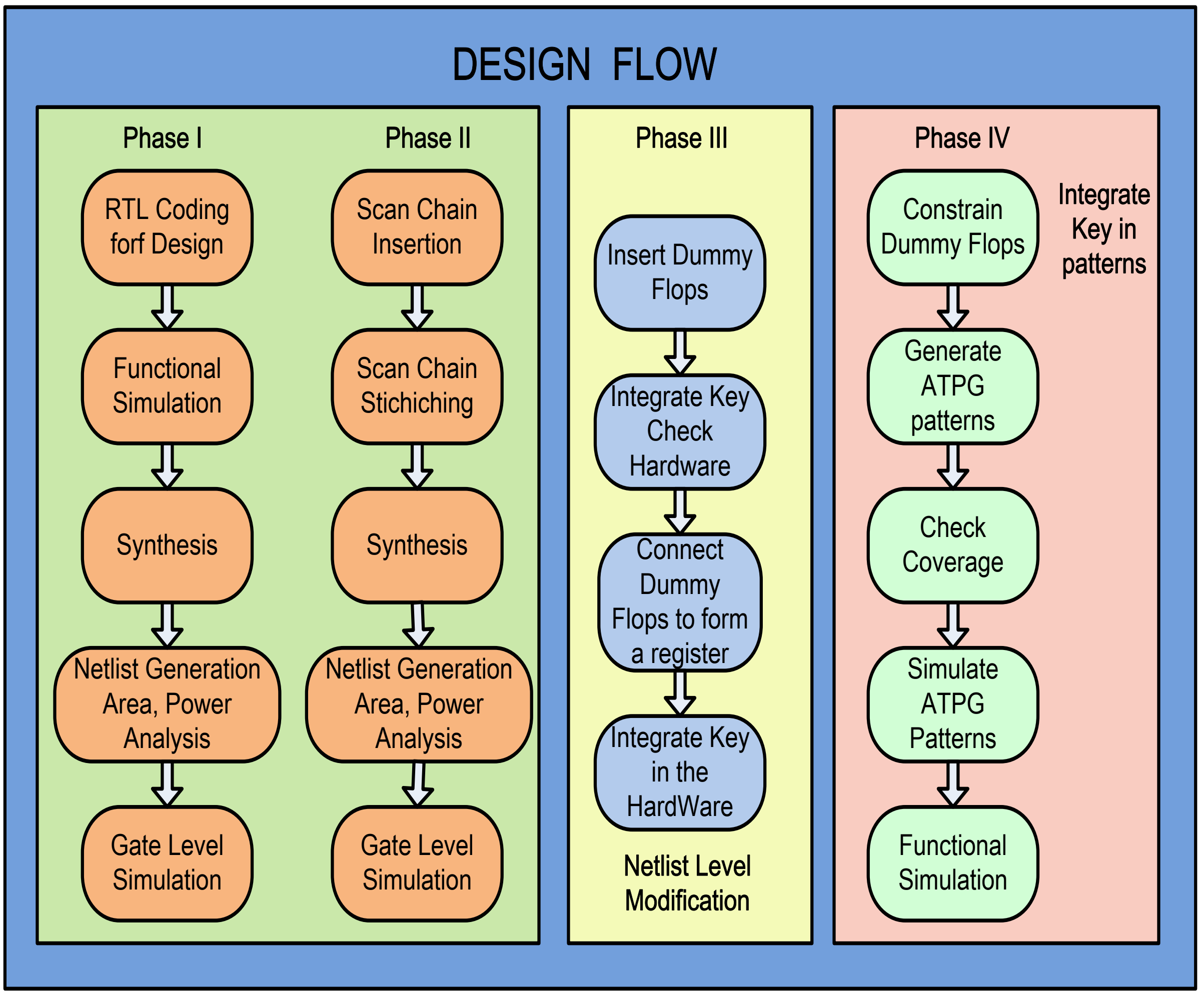

Figure 22 : Design Flow

The above figure shows the detailed design flow. The overall design is divided into four phases. It is very much similar to the basic ASIC design flow. The overall flow is divided according to the level of abstraction required. The first two phases are shown together because combining them gives the actual hardware, which is then made secure in phase three.

4.1.1 Phase I

Phase I includes the design specification, RTL design, functional simulation, synthesis, power analysis, and gate level simulation.

Design Specification: The AES is chosen as a benchmark circuit. The architecture is defined to meet the standard encryption levels defined by AES algorithm. The overall design is broken into the several modules according to their functionality. The state machine is design to achieve the desired control logic. Then the design integration of modules and interfaces to connect the different modules is defined.

RTL design: The Verilog hardware description language is used to implement the design. The RTL design is made to meet the design specifications. The detailed block level description is given in the next section.

Functional Simulation: IUS Cadence simulator from Cadence is used for the functional simulation. Automated test bench is designed in Verilog, which tests the design in both encryption and decryption mode.

Synthesis: Synthesis of the RTL is done using Design Compiler from Synopsis. This process includes the development of synthesis scripts and Netlist generation. We use the UMC cell library with 180 nanometre technology. After Netlist generation, area and power analysis is done. The detailed analysis is provided in the results and analysis section.

Gate Level Simulation: This includes the functional simulation of the Netlist. The test bench used for the RTL design is used to verify the Netlist simulation also. This process proves the correct hardware generation which satisfies design specification.

4.1.2 Phase II

Phase II is the DFT strategy. This includes scan chain insertion, scan chain stitching, synthesis and gate level simulation.

Scan chain insertion and Stitching: In this process the synthesis is performed on the generated Netlist. Flip-flops in the design are replaced by scan flip-flops and stitched together to form the scan chain. The extra ports required for DFT are also added to the Netlist. Then through the synthesis, a new Netlist is generated which includes DFT elements. Area and power analysis at this stage gives an idea about overheads caused by extra hardware.

Gate Level Simulation: This is necessary to verify the functional behaviour of the design after scan chain insertion. This includes the functional simulation of the Netlist with scan chain. This process proves that there is no change in the design functionality after design changes.

4.1.3 Phase III

It is the most crucial part of the design flow. In this phase we actually make the hardware changes which are required to make the secured design. This includes insertion of dummy flip-flops in the design, integrating key checking hardware to the design and connecting inserted dummy flip-flops to form a shift register. In this process the existing scan chain is broken and dummy flip-flops are inserted in the design. All these changes are done at the Netlist level. In further sections these Netlist level modifications are shown in detail.

4.1.4 Phase IV

It is the final phase which includes the pattern generation, fault coverage, DFT simulations and functional simulation.

ATPG pattern generation:

In this process the test patterns are generated which are used for testing the hardware during manufacturing. For pattern generation Tetra Max tool from Synopsis is used.

Fault Coverage analysis:

The detailed fault coverage analysis for the design is done. The effectiveness of the generated patterns is checked through the covered and uncovered faults. Pattern generation and fault coverage analysis is continued until desired coverage is obtained. The detailed coverage reports are provided in the result and analysis section.

DFT and Functional Simulation:

The generated patterns are tested through DFT simulation. This ensures the correct generation of patterns and guaranties the fault coverage for actual test. The DFT simulation is done trough special test bench generated by Tetra Max tool. Functional simulation after integrating security hardware ensures there is no effect on the function of hardware.

In the next section we will look into the details of processes involved in all the phases.

4.2 Design Specification and RTL:

AES is the topmost module which is used to use with the external world. It consists of following modules:

- Sbox
- Mix_column
- Shift_row
- Key_expander
- Control logic

In the next section we will see complete data flow and detail implementation of each block.

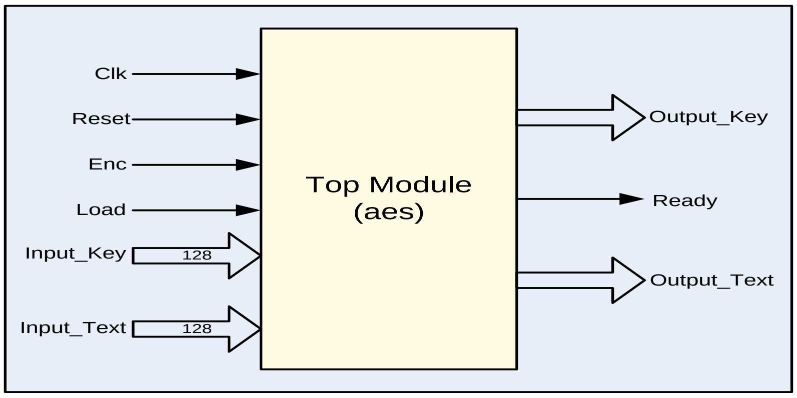

Figure 23 : AES top module interface

The above diagram shows the input/output interface of the topmost module. The module takes 128 bit key and gives an encrypted output text and key. The following table shows the description of the input and outputs.

Table 7 : I/O list of AES top module

Port	Size(in number of bits)	Direction	Description
Clk	1	Input	Clock input
Reset	1	Input	Synchronous , Active high reset signal

Enc	1	Input	Enc=1-> Encryption
			Enc=0 -> Decryption

Load	1	Input	This is a pulse signal indicates start of new transaction
Input_key	128	Input	Input key used for encryption
Input_text	128	Input	Input test to be encrypted
Output_key	128	Output	Encrypted key
Output_text	128	Output	Encrypted text
Ready	1	Output	Indicates end of iteration

.

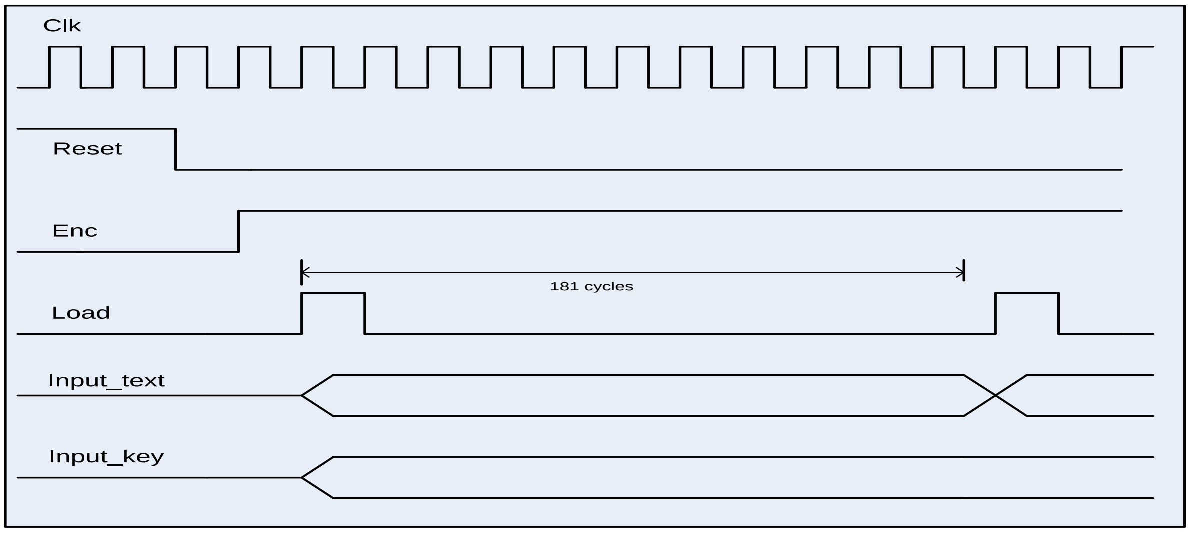

Figure 24 : Input sequence

Figure 24 shows the input sequence with respect to the clock. On the application of reset all, the registers in the design get loaded with the reset value and state machine enters default state. ‘Enc’ input is used to indicate whether current transaction is for encryption or decryption. This signal can either be asserted or disserted, but should maintain the same state throughout the whole process. ‘Load’ signal indicates the start of a new transaction and as shown in the diagram it is a pulse signal which asserts for only one clock cycle. The input text and input key has to be driven on the same clock cycle when ‘Load’ is getting asserted. Input key remains the same for rest of the transaction but input text changes for every new transaction. We can observe in the waveform that input text changes along with the new 'Load'.

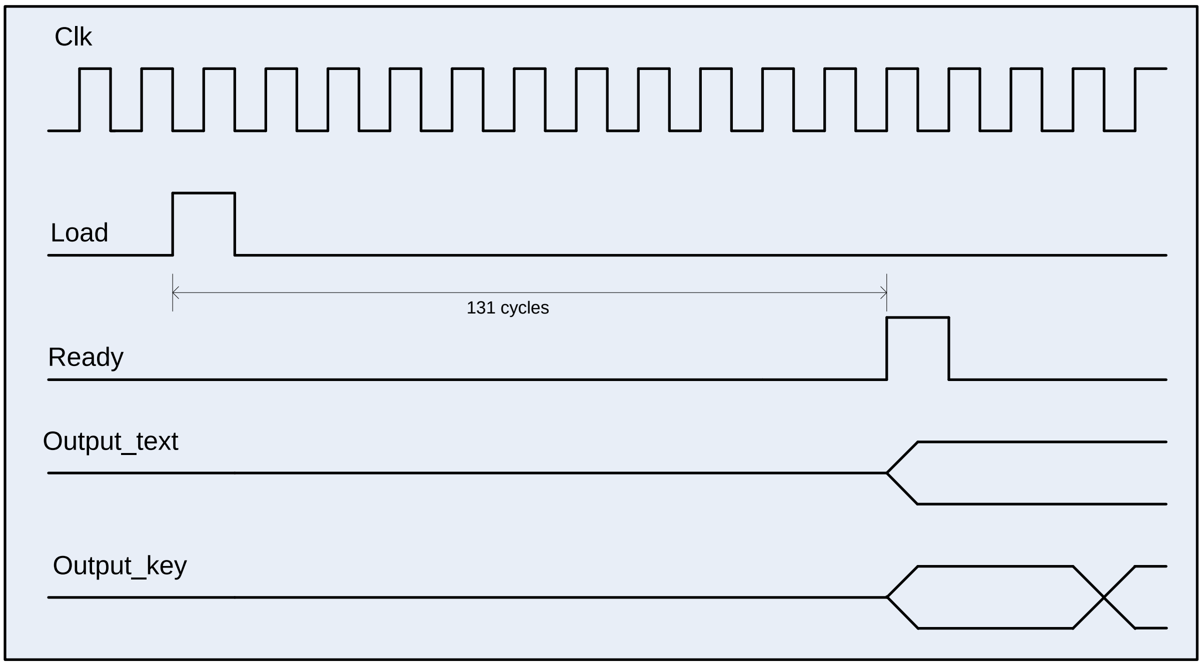

Figure 25 : Output sequence

The above diagram describes the output sequence. Ready signal indicates that the encrypted text and the key are ready. This signal is a pulse and will be asserted for one clock cycle. Output text and key are available on the same clock cycle when 'Ready' signal gets asserted. Also, we can observe that once new data enters hardware, the output is obtained after 131 cycles. So the hardware latency from input to output is 131 clock cycles. This changes if the number of s-boxes used in the design changes. The latency shown in the diagram is for the design using 4 s-boxes.

4.2.1 Data Flow:

Figure 26 explains the complete data flow of AES design. It consists of 4 s-box modules, control logic and state machine, Shift_row, Mix_column and key expander block, XOR logic block and output formation block.

As we can make out from the figure 28, input text and input key both are128 bits. They pass through XOR logic and then fed to the s-box. This design uses four s-boxes. Each s-box is used to process 32 bits of input text. This data gets concatenated to form the 128 bits input for shift-rows block. Shift-row block takes 128 bits inputs and forms input for the next block which in mix column. Mix column block forms the final encrypted text. Key expander block works in parallel with S-box, Mix_column and Shift_row block. This block forms the final encrypted key. Output formation block is used to register final outputs. Control logic and state machine are used to control complete data flow. This is the most critical block in the design.

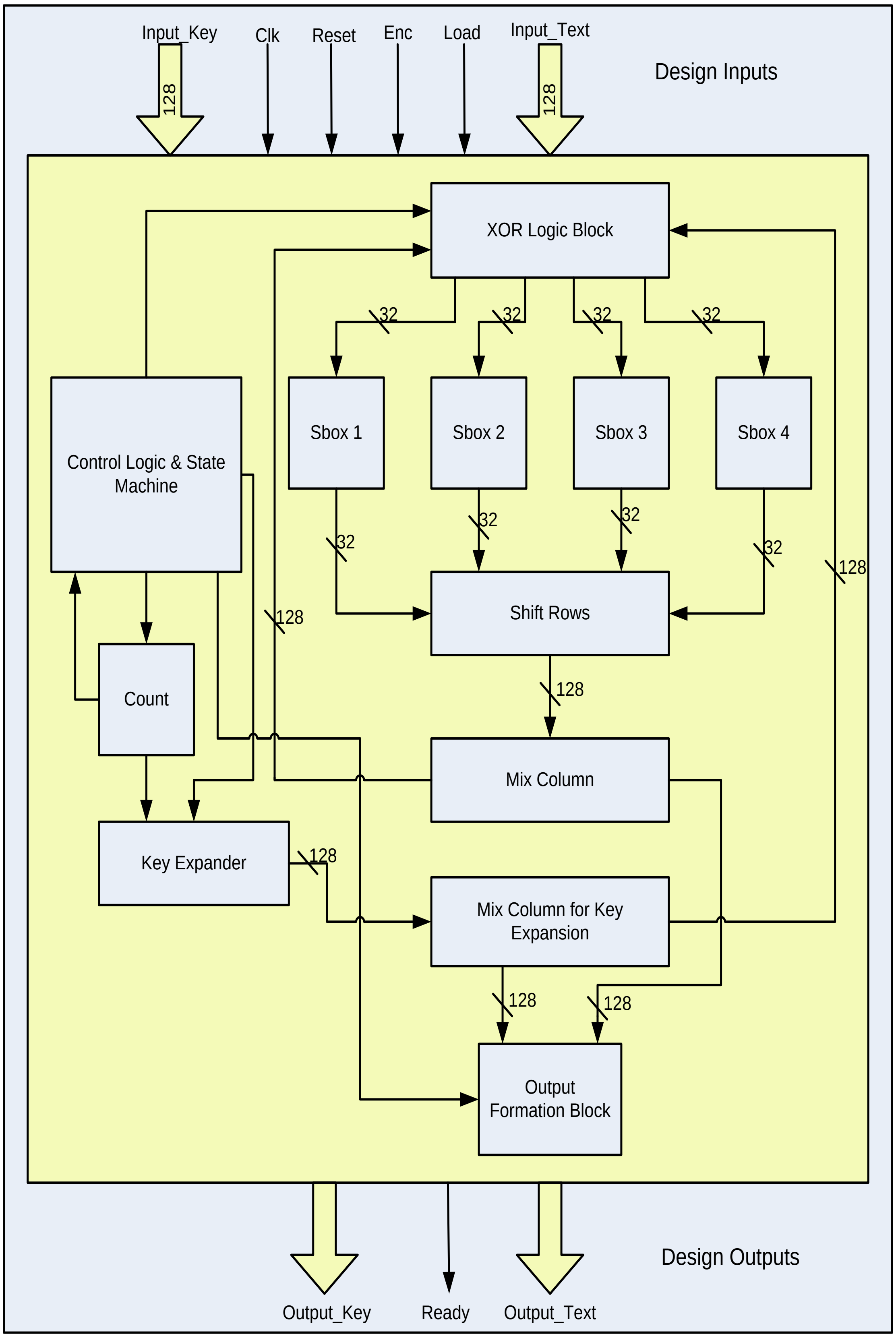

Figure 26 : AES Data Flow

4.2.2 Control Logic and State Machine:

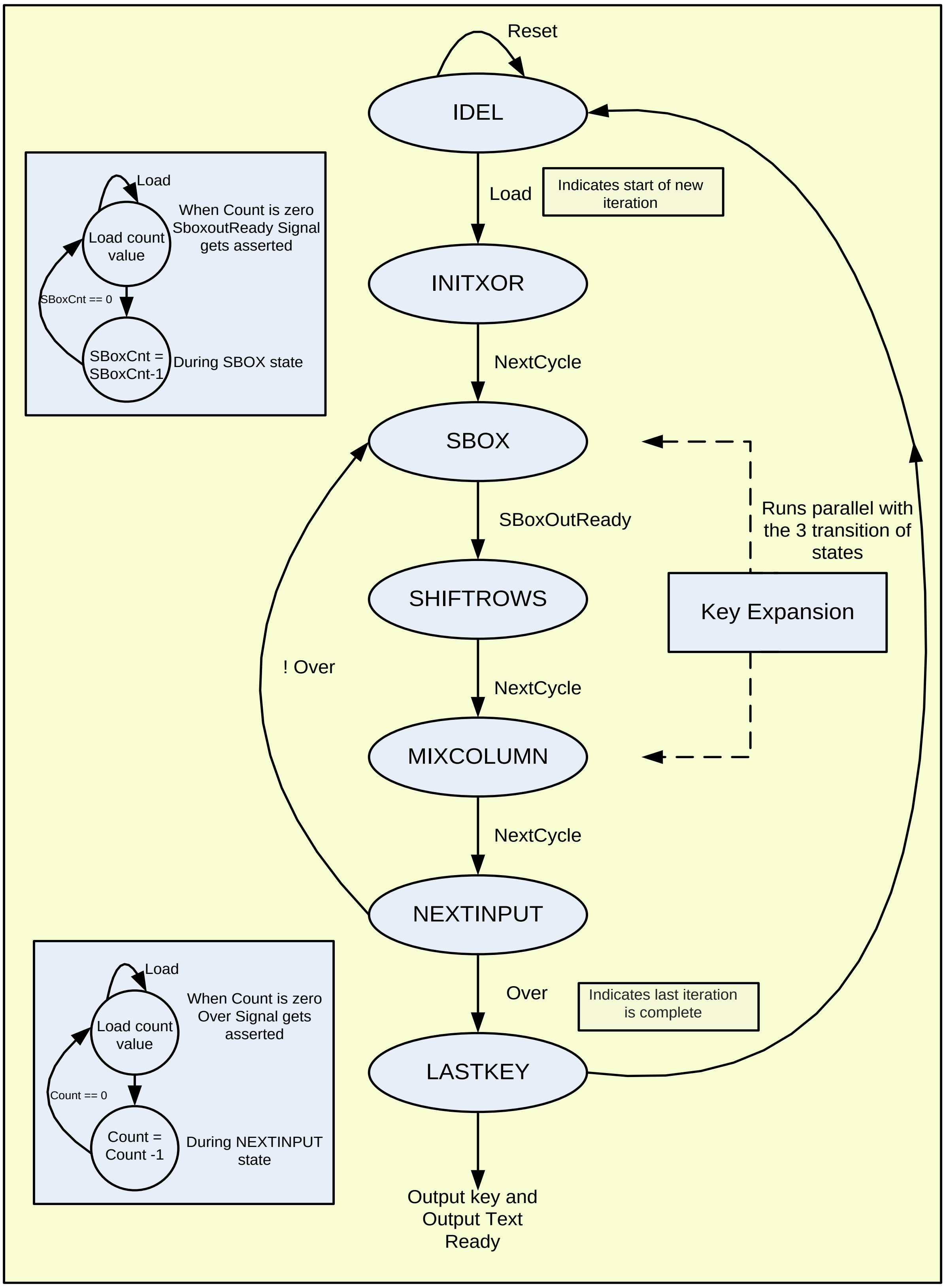

Figure 27 : State Machine

The above schematic describes the state machine designed in the code. The state machine handles the main control of overall process. The advantage of using the state machine in the

RTL design is that it makes the overall process very easy to understand. The segregation of the design units becomes very convenient. Modifications required to make a design with a number of s-boxes is possible by just changing state transition logic and varying the use of number of s-boxes.

As shown in the diagram, state machine consists of seven states which include IDLE, INITXOR, SBOX, SHIFTROWS, MIXCOLUME, NEXTINPUT, LASTKEY. Also there are some control signals which decides the transitions of states which are Load, SboxOut and Over. Key expansion unit runs in parallel, with three states, SBOX, SHIFTROWS and MIXCOLUME. The square on the top left of the diagram shows the use of a counter which keeps the track of number of cycles utilised in the SBOX process. As we need to have seven states, the register holding the state is three bits wide.

4.2.2.1 State machine:

IDLE: As the name suggests, this state is maintained until the design is in idle state. On the application of Reset, state machine enters IDLE state. This state gets maintained unless the Load signal gets asserted. Once Load gets asserted state machine enters INITXOR state. When state machine is in LASTKEY state, on the next clock cycle it comes back to the IDLE state. This is also the default state for the state machine.

INITXOR: State machine maintains this state just for one cycle. XOR operation is performed between input text and the input key for the new iteration. This is done only once. Later, the internally generated key is used to perform an XOR operation and on the next cycle it transits to SBOX state without the need of any control signal.

SBOX: This state of the state machine consumes more clock cycles than any other functional states. The number of clock cycles consumed by this state is dependent on the number of s-boxes used in the design. One s-box can perform operation on 32 bits at a time. Therefore if we are using 4 s-box designs, the state machine remains in this state for four clock cycles. For 8 s-box design this state will consume two clock cycles as s-box operation will be performed on 64 bits at a time. This can be controlled with the SboxCnt that is shown in the top left corner of figure. When Load signal is asserted, desired value of the SboxCnt gets loaded into the register. When state machine is in the SBOX state, SboxCnt gets decremented. Once the SboxCnt reaches the zero value SBoxoutReady signal gets asserted. On the assertion of SBoxOutReady signal, state machine transits to SHIFTROWS.

SHIFTROWS: This state consumes only one clock cycle. During this state rows are shifted in the formatted input text. This process utilizes only combinational elements. Therefore, one clock cycle is sufficient for the operation. In one cycle, state machine transits to the next MIXCOLUMN.

MIXCOLUMN: This state consumes only one clock cycle. During this state mixing of column operation is performed on the formatted input text. This process utilizes combinational elements. Therefore one clock cycle is sufficient for the operation. In one cycle, the state machine transits to NEXTINPUT.

NEXTINPUT: When state machine exits this state, design should be ready with new key and formatted input text for the next iteration. Before next iteration starts, XOR operation has to be performed on the new text and new key. Formatted input comes from mix column block and new key comes from key expander block. These two blocks work in parallel, which may take different number of clock cycles. Therefore synchronization has to be maintained in the process. This state actually provides this synchronization. It provides wait cycles according to the design requirements. The total operation takes several iterations to form encrypted output. This state maintains the count, which keeps track of number of iterations elapsed. This count is shown in the bottom left corner of the figure. Zero value in the register holding count denotes the last iteration. At this moment, Over signal gets asserted which decides the transition of state machine. If Over signal is asserted then state machine transits to the last state, LASTKEY. Otherwise state machine transits to the SBOX state for next iteration.

LASTKEY: This is the last state in the state transitions. This state is used to write the final results in to the output registers Output_text and Output_key. In this state Ready signal also gets asserted which indicates that the output text and output key are encrypted and ready for use. This state consumes one clock cycle. Then state machine transits to the IDLE state to wait for next iteration.

4.2.2.2 Control Signals:

SBoxCnt: This is a three bit down-counter. It loads the desired value on the assertion of Load signal. The counter decrements when the state machine is in the SBOX state. The counter loading value is decided by the number of s-boxes used in the design. This counter is also used to decide the inputs of the Sbox block. According to the count, 128 bit input text divides to form the chunks of data and gets applied to the Sbox block. e.g. – For 2 s-box design counter value should be seven. For 4 s-box design counter value should be sixteen.

SBoxOutReady: This signal gets asserted when SBoxCnt counter reaches zero value. This signal indicates that the operation Sbox block is over. When this signal gets asserted state machine transits from the SBOX state.

Count: This is a 4 bit down counter. This counter used to keep track of the number of iterations. When load signal gets asserted, Count gets loaded with the desired value. When state machine is in the NEXTINPUT state, counter decrements by one.

Over: This signal indicates the end of iteration. This signal gets asserted when counter Count reaches zero value. This signal decides the transition of a state machine from NEXTINPUT state. If this signal is asserted, the then state machine transits to LASTKET state otherwise goes back to SBOX state.

4.2.3 S-box Module:

As shown in the diagram, s-box is a simple combinational logic consisting of two LUT's and output MUX logic. The formation of inputs for the s-box unit is done through Input MUX logic. State_in_reg, SBoxCnt and Sbox_out_reg are situated in the top module which plays an important role to form input and outputs for s-box.

Table 8: Sbox I/O list

Port	Size(in number of bits)	Direction	Description
s_in	8	Input	Input for s-box
Enc	1	Input	Enc=1-> Encryption, Enc=0 -> Decryption
s_out	8	Output	Combinational output from s-box

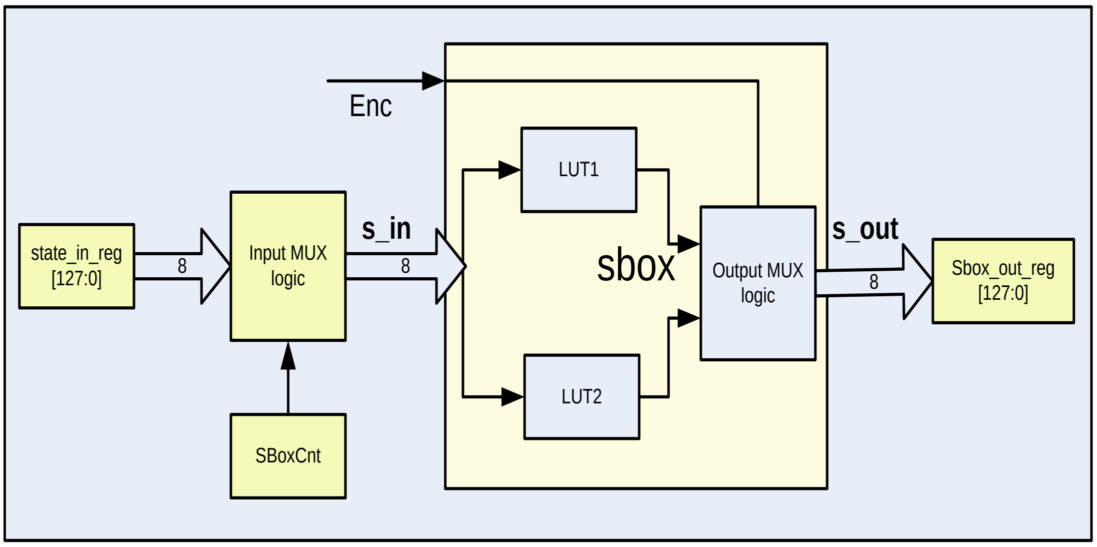

Figure 28 : Sbox block diagram

4.2.3.1 Input MUX Logic:

S-box block perform operations on 8 bits at a time. There are 128 bit inputs which pass through s-box operation. Input MUX Logic obtains the input from 128 bit state_in_register. We have four s-boxes in total, where each s-box operates on 32 bits. Every s-box has to get four chunks of 8 bit input data. This input data formation is controlled through Input MUX Logic in accordance with the SBoxCnt counter. Every input takes one clock cycle, so in four clock cycles we have the complete 128 bit output.

4.2.3.2 LUT's and Output MUX Logic:

The actual s-box operation is performed through LUT's. There are two LUT's each for encryption and decryption. Each LUT contains hard coded values for all the combinations of 8 bit input. That is, in total there are 256 hardcoded values in each LUT. 'Enc' signal acts as a select line for Output MUX Logic. On the basis of logic state of Enc signal, s-box output is written in to SBoxOut register from respective LUT. The output gets written in to the register from four s-boxes in four clock cycles depending on the particular input which is controlled through SBoxCnt counter.

4.2.4 Shift-rows Module:

Shift_row is a combinational block consisting of two LUT's and output MUX Logic. This block takes the input from the 128 bit state_sr_in register and writes output into 128 bits state_sr_reg. Both the registers are situated in the topmost module. Logic for this block is simple compared to the rest of blocks. But it consumes a lot of combinational logic as it has two LUT's which contains hardcoded values for 2^128 input combinations.

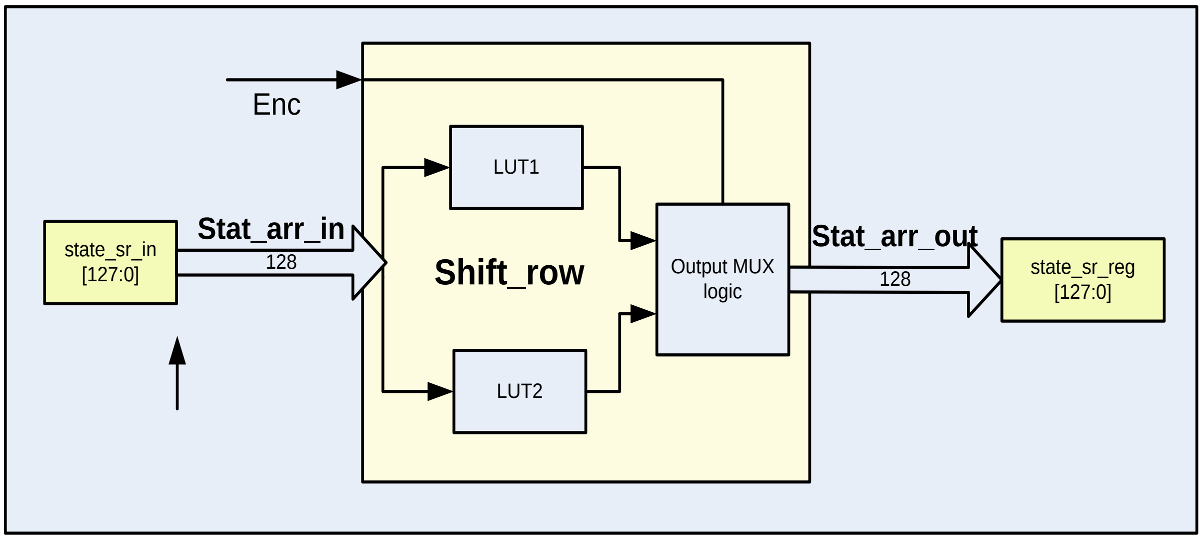

Figure 29: shift_rows block diagram

Table 9 : shift_rows I/O list

Port	Size(in number of bits)	Direction	Description
Stat_arr_in	128	Input	Input for shift_rows

Enc	1	Input	Enc=1-> Encryption
			Enc=0 -> Decryption

Stat_arr_out	128	Output	Combinational output from shift_rows

4.2.4.1 LUT's and Output MUX Logic:

There are two LUT's each used for either encryption or decryption operation. Enc input acts as a select line for Output MUX Logic. The output is written into the state_sr_reg register. The whole operation consumes only one clock cycle, which is necessary to register the output.

4.2.5 Mix_column Module:

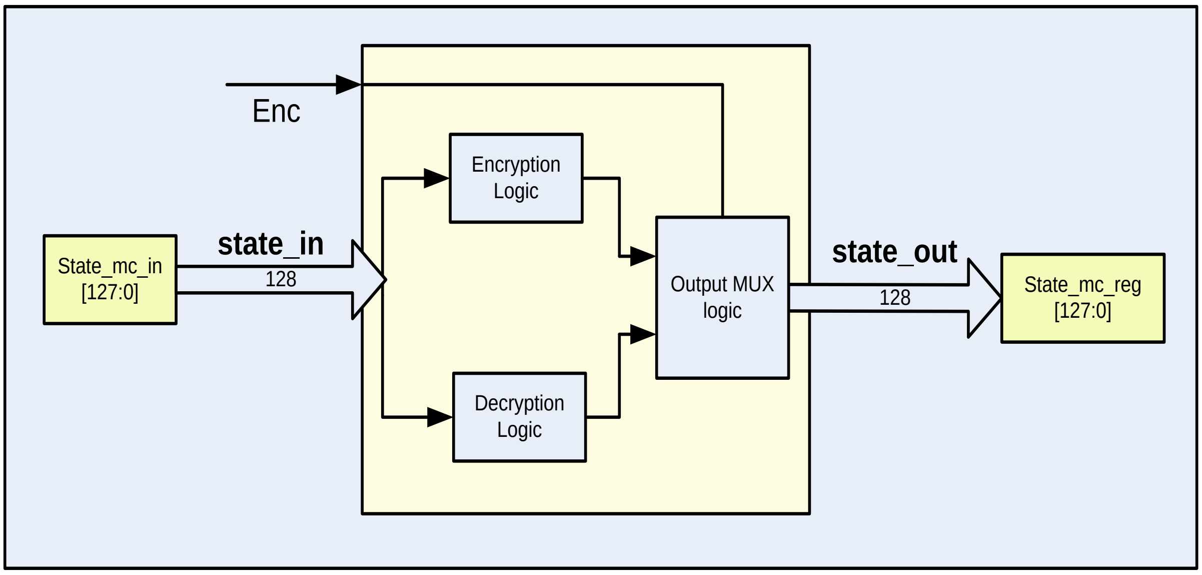

Figure 30: Mix_column Block diagram

Table 10 : Mix_column I/O list

Port	Size(in number of bits)	Direction	Description
state_in	128	Input	Input for Mix_column

Enc	1	Input	Enc=1-> Encryption
			Enc=0 -> Decryption

state_out	128	Output	Combinational output from s Mix_column

This is one of the most crucial blocks, as the main encryption/decryption is performed by this block. This is done with the help of encryption and decryption block. Output MUX logic decides the final output on the status of Enc input. The input is taken from 128 bits state_mc_in register and written into 128 bits state_mc_reg register. Both the registers are located in the topmost module.

4.2.5.1 Encryption/ Decryption Block:

These blocks consist of only combinational logic. The actual operation includes lot of arithmetic operations like addition, multiplication and XOR logic. This is performed by using number of functions which operates on different chunks of 128 bits input data. The function covers the equations for encryption and decryption described in the AES algorithm, described in the previous section.

4.2.5.2 Output MUX Logic:

This block takes the inputs from encryption and decryption blocks. Enc input acts as a select line for Output MUX Logic. According to the logic state of Enc input, the output is written into the state_mc_reg register.

4.2.6 Key_expander:

Key expander is the only block which performs operation on the input key. Table 11 shows the input and output list for this block. Figure 31 shows the block level implementation of key expander block. The output of key expander block goes to the mix column block which forms the key for next iteration. This block is consisting of both sequential and combinational logic.

Table 11 : Key_expander I/O list

Port	Size(in number of bits)	Direction	Description
Clk	1	Input	Clock input
Reset	1	Input	Synchronous , Active high reset signal

Enc	1	Input	Enc=1-> Encryption
			Enc=0 -> Decryption

Load	1	Input	This is a pulse signal indicates start of new transaction
Current_State	3	Input	State machine indicates the current state of a state machine
Keyin_c0	32	Input	32 bits of Input key[127:96]
Keyin_c1	32	Input	32 bits of Input key[95:64]
Keyin_c2	32	Input	32 bits of Input key[63:32]
Keyin_c3	32	Input	32 bits of Input key[31:0]
Keyout_c0	32	Output	32 bits of encrypted key [127:96]
Keyout_c1	32	Output	32 bits of encrypted key [95:64]
Keyout_c2	32	Output	32 bits of encrypted key [63:32]
Keyout_c3	32	Output	32 bits of encrypted key [31:0]

It takes 128 bits of the key input through four different ports, where each port takes 32 bits input of the key. Keyin_c0, Keyin_c1**,** Keyin_c2, Keyin_c3 are the port used for the key input**.** In the same way the 128 bit key is broken into four chunks and given to Keyout_c0, Keyout_c1, Keyout_c2, and Keyout_c3 respectively. The breaking of key into four chunks is for logic reutilisation and optimisation.

As shown in the figure 31, the overall operation is executed through two blocks control logic and key expansion logic block. Key expansion logic runs in parallel with the three blocks s-box, shift_rows and mix column.

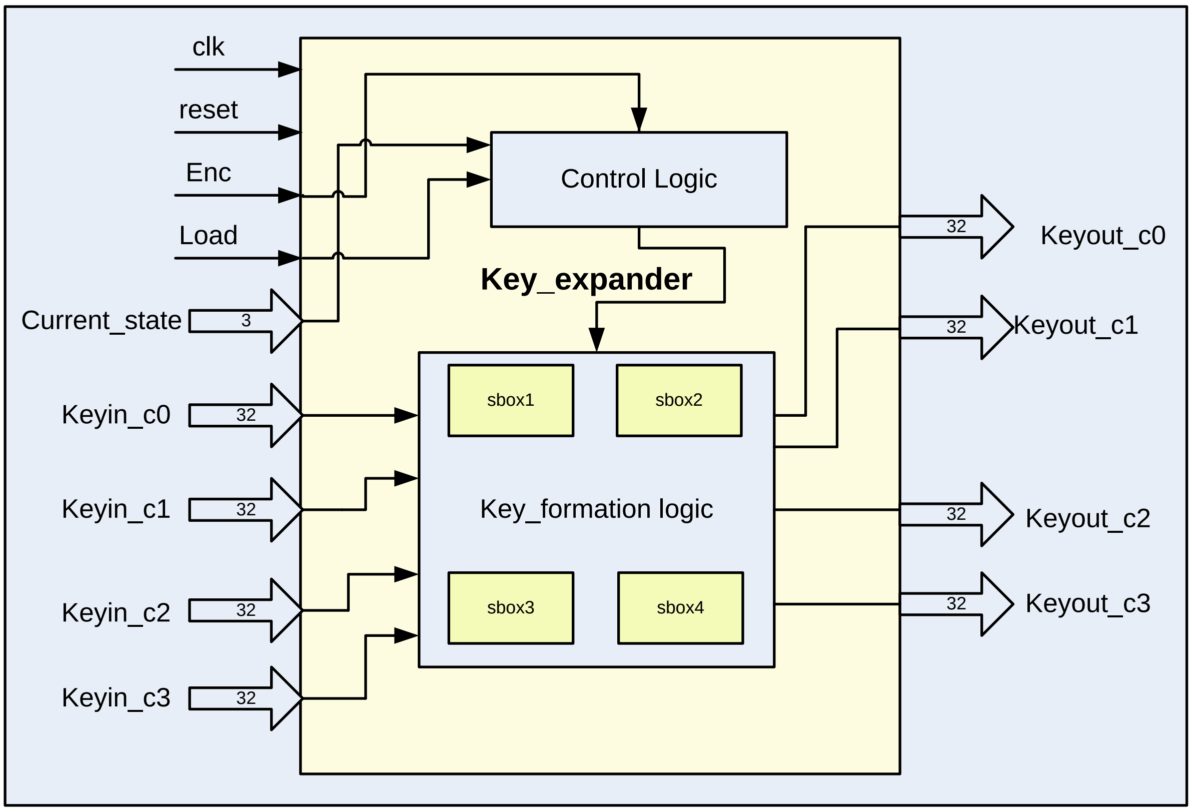

Figure 31 : Key_expander block diagram

4.2.6.1 Control Logic:

The key expansion logic needs to be controlled through state machine, as specific values have to be loaded into the registers according to the state machine transition. The input signal Current_state indicates to the control logic in which state the state machine currently in. Accordingly the state control signals for the key expansion block are derived. The Load signal is used to load the initial key into the internal resister before the start of new iteration. The internal register key_rot is used to form the input of s-boxes inside the key expansion logic. The contents of this registers vary according to the logic state of ‘Enc’ input. If the ‘Enc’ input is at logic one then the ‘key_rot’ register gets loaded with the contents required for encryption and if the ‘Enc’ input is at logic zero then the key_rot register gets loaded with the contents required for decryption.

The register key_rcon is used for the XOR operation performed on the input key. Its value changes for every iteration which is controlled by the main counter available in the topmost module. The contents of the key_rcon register are also dependent on the logic state of ‘Enc’ input. The value in this register gets updated from LUT where different values are hardcoded. The LUT contains eleven different values for different iterations, separately for encryption and decryption.

4.2.6.2 Key Expansion Logic:

As shown in the diagram, this block is majorly consisting of four s-boxes. These are different from the s-boxes used for the input text operation. This block basically receives 128 bit key from four different inputs, each 32 bits wide. These four different inputs are used to form one single 32 bits value. This is done through some arithmetic and XOR operations. This utilises four extra registers k_c1, k_c2, k_c3 and k_c4. These registers are used to hold the intermediate values during arithmetic operations.

The input of the s-boxes is formed using XOR operation between k_c1, k_c2, k_c3 and k_c4 registers and the contents of k_rot register. The outcome of this operation is stored in a 32 bit register and given to four s-boxes. The function of the s-box is absolutely the same as explained in the previous section.

The XOR operation is performed on the output from the four s-boxes and 32 bit result is obtained. The 32 bit result is stored in the internal register. By performing arithmetic and XOR operations according to the process shown in the AES algorithm description, four 32 bits key outputs are obtained. These are driven on the kout_c0, keyoutc1, keyoutc2, keyoutc3 ports respectively.

4.2.6.3 Mix Column:

This block functions in the same way as the mix_column used for the input text encryption. It has already been mentioned in the previous section.

This block is used for 128 bit output key from the four 32 bits keys given by key_expander block. The four outputs from the key_expander block are concatenated together to form 128 bits input for mix_column. The output from the mix_column block is stored in the state_mc register.

This output is used to form the key for next iteration. This output is used to perform XOR operation with the encrypted text to perform further iterations.

4.2.6.4 Output Formation Logic:

This block carries logic to the register final output text and final output key. Registering output text is a very simple and straight forward exercise. When the state machine is in the LASTKEY state, the output from the mix_column involved in the input text encryption is directly registered as the final output. Registering output key is bit tricky compared to registering output text. As per AES algorithm, mix_column operation is not performed on the key involved in the final iteration. On the basis of “Count” Counter value, the final output is registered directly from key_expansion block.

The registered output is driven on to the respective ports. Along with the output_text and output_key, Ready output is also driven through output formation logic. This output specifies that output_key and output_text are ready to use for the next block or external world. Figure25 shows how these are driven with respect to the clock.

4.3 Simulation and Verification:

This section explains about the functional simulation performed on both RTL and gate level design.

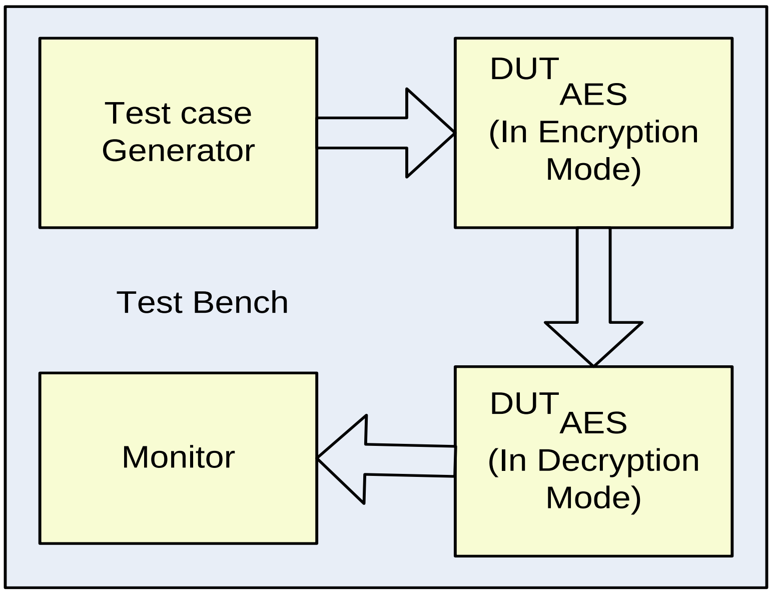

Figure 32 : Verification Environment

The above diagram describes the structure of test environment. The same environment is used to verify functional as well as gate level simulation. The same testing environment helps in evaluating the correctness of generated hardware.

The AES design works in both encryption and decryption modes. Both the modes are verified simultaneously. As shown in the figure, there are two designs under tests where one configured in encryption mode and another in decryption mode. Test case generator generates different sets of inputs which are driven into AES in encryption mode. This encrypted output driven into the AES in decryption mode. Monitor collects the response from the AES under decrypted mode and results are verified. Monitor also checks the encrypted outputs separately to ensure this mode functioning properly. Effectively monitor should receive the same data driven into AES for encryption. With this, in same environment we are able to test both the design behaviours.

As shown in the design flow verification is performed in phase I, II and IV. For the verification done in Phase II the scan chain inputs and outputs has to be added in the testing environment. Phase IV verification is different than other simulations. Netlist in Phase IV consist of additional hardware added for security. So along with every set of input, variable security key must be shifted in to the design. This requires extra logic to be added in the test bench. This extra logic carries generation of security key and the task to shift in this security key. In case of key mismatch simulation fails as AES output changes randomly.

The Result and analysis section contains snapshots of simulation waveforms for RTL and gate level simulation. Another set of waveforms explains the simulation with security key insertion. The waveforms explaining both failing and passing scenarios are shown through waveforms.

4.4 Synthesis:

Synthesis is performed in Phase I and Phase II. The basic difference in both the process is that in Phase I synthesis is performed on RTL to generate the netlist. Phase II reads the netlist from phase I and performs operations related to design fort test.

4.4.1 Synthesis steps for Phase I:

- Read RTL.
- Read Library.
- Elaborate Design.
  The design is read and mapped to the library elements.
- Compile.
- Generate Netlist.
- Generate Area and Power reports.

4.4.2 Synthesis steps for Phase II:

- Read netlist.
- Create and add DFT related ports
  TEST_EN-> test mode pin, SCAN_EN-> scan enable pin, SC_IN -> Scan data in pin.
- Set DFT constraints.
  This includes setting of DFT ports. The detailed information about DFT ports and their active state have to be specified to the tool.
  This includes Setting scan configuration, Setting desired number of scan chains, Set scan style, Define scan enable, scan clock, test mode, scan in and scan out ports and map them to actual ports, Set DFT clock period.
- Create Test Protocol
  Test protocol is generated on the basis of DFT constraints. This creates timing information and DFT ports log for the tool which is used to trace the scan chain and DRC rules.
- Check DFT DRC
  In this all the flip-flops are check for DFT functionality. It gives the detailed analysis of which flip-flops can be used to form scan chain and which flip-flops fail to pass the DFT rules. This is checked on the basis of clock and reset signal connected to the flop. It is possible that because of some complicated logic some flip-flops are not traceable through scan chain. DRC reports all such flip-flops. The counter measures have to be performed to make such flip-flops scanable.
- Compile.
- Insert Scan
  This is a process where normal flips-flops which pass the DRC rules are replaced by scanable flip-flops.
- Stitch Scan Chain
  This process stitches all the scan flip-flops to form the scan chain.
- Check Post DFT DRC
- Generate Netlist.
- Create Test Protocol File
- Generate Area and Power reports.

All the analysis reports are mentioned in the Result and analysis section.

4.5 **Netlist Level Modifications:**

This is a part of Phase III which is the core part of the design flow. The changes made in this phase are independent of the RTL and difficult to analyse from the netlist.

The process can be simplified by breaking into following steps.

- **Analyse the critical registers in the chain**.
  This may include the key registers in the design which may hold critical information. Or these can be most widely used registers in the design. Trace these registers in the netlist and identify their connection in the scan chain.
- **Break the scan chain to insert dummy flops**
  Insertion of dummy flip-flop is nothing but the insertion of extra library cell for scan flip-flop. It is better to insert dummy flip-flops just before or after the critical register. This provides more security as critical register path immediately passes through dummy flip-flop. Desired number of dummy flip-flops inserted at various positions. The care has to be taken that scan chain should not break any ware in the design otherwise it creates error which is really hard to debug.
- **Formation of Shift registers**.
  The inserted dummy flip-flops are connected to form the shift register. The output of one dummy cell gets connected to the 'D' input of another dummy cell.
- **Integration of add-on hardware**.
  The add-on hardware carrying 'key checking' and 'random generation logic is integrated with the design. This integration requites instantiation of this module in the design and mapping its input/output ports to the desired logic. The multiplexers are added at the outputs of scan chain for randomising scan data and additional multiplexers are added to the functional outputs to randomising chip response in functional mode. The connections of these multiplexers is critical task as it carries connections from add-on hardware as well as design internal signals and must be done with care otherwise can create errors which are tough to debug.

The following diagrams show the structure of hardware before and after inserting dummy flip-flop.

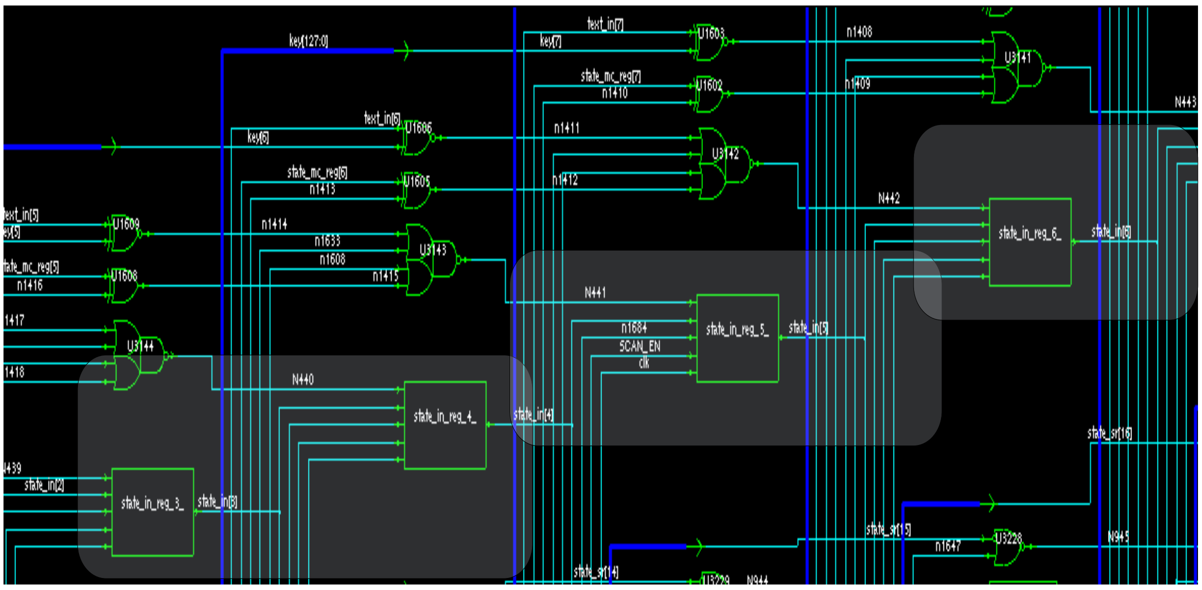

Figure 33 : Scan Chain

The above diagram shows the structure of scan chain before inserting dummy flip-flop. Registers state_in_reg3, state_in_reg4, state_in_reg5, state_in_reg6 are highlighted in the above figure are consecutive scan cells in the scan chain.

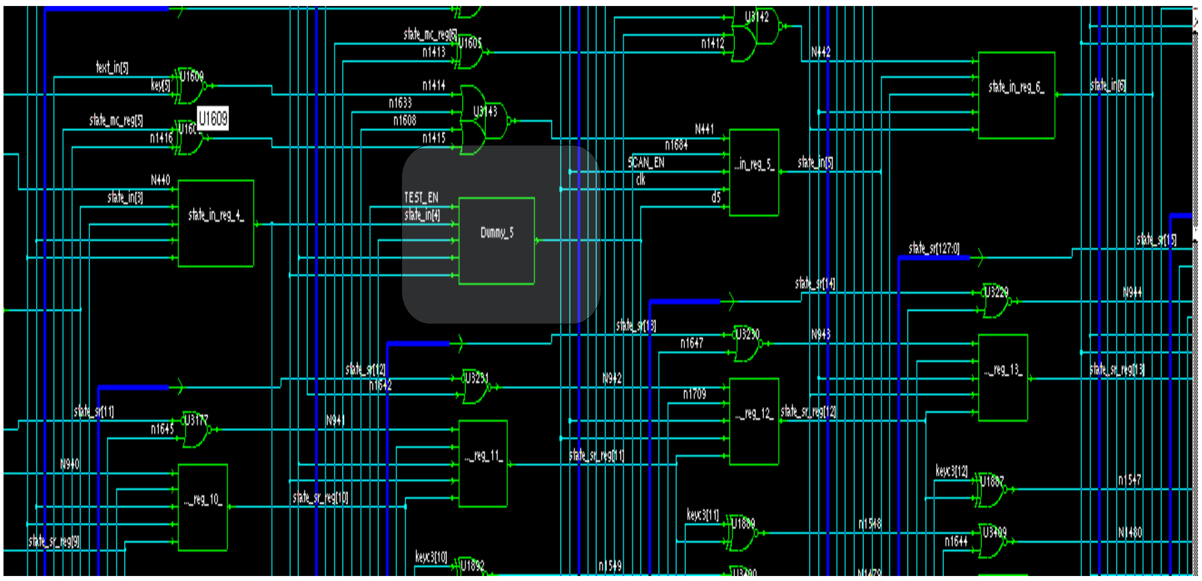

Figure 34 : Dummy flip-flop insertion

The above diagram shows the structure of scan chain after insertion of dummy flip-flop. The scan chain is broken in between state_in_reg4, state_in_reg5 and dummy cell inserted. The highlighted cell is dummy flip-flop. The Output of state_in_reg4 goes to the scan input of dummy cell and its output goes to the sc_in input of state_in_reg5.

4.6 **Test Pattern generation and Simulation:**

This is the final part of the design flow which is the part of Phase IV. This process includes the pattern generation and simulation of these patterns.

4.6.1 Pattern Generation:

Patterns are generated with secures key integrated into them. This can be achieved by constraining dummy flip-flops during pattern generation. Constraining dummy flip-flop ensures the logic value hold by them during “Load” operation. The tool should be given information about position of dummy flip-flops and their desired value during “Load”.

There are two types of vector that are generated for dft.

WGL format – These patterns are in Waveform Generation Language (WGL) format. These patterns are used to test the design on Automatic Test Equipment (ATE). These vectors hold the information in terms of 1’s and 0’s which specifies the activity happening input and output ports during application of test pattern to the chip.

Verilog Format – These patterns are generated in verilog format or in the memory format which can be read through verilog test bench. Tool generates two types of test pattern serial and parallel. Serial patterns check faults in detail but take longer time for simulation than parallel patterns. Parallel vectors take less time for simulation and can be used to check any errors in the pattern generation in quicker time. The basic difference between serial and parallel vectors is the stimulus application to the design.

4.6.2 Pattern Simulation:

The tool generates verilog test bench along with the verilog patterns. These are automated test benches which displays error on the consol in case of any failure .This test bench can be used to run DFT simulations. The scan netlist is used as a design under test and patterns are applied to it. In case of correct pattern generation all pattern simulation passes without any error. In case of failure the scan chain should be traced back for debugging.

We are simulating patterns with integrated key and patterns without integrated key. The simulation of patterns without integrated key fails as key checker circuit detects the mismatch and scan chain output changes randomly. So test bench fires multiple errors for failure patterns. Simulation of patterns with integrated key runs successfully. Both the simulation waveforms are shown in the result and analysis section.

The WGL and verilog patterns are different in nature but similar in terms of activities happen on input and output port. So the successful simulation of verilog pattern ensures the effectiveness of WGL patterns on the Actual tester (Automatic Test Equipment).

Chapter 5. RESULTS AND ANALYSIS

We have designed three different AES designs with different number of S-boxes. S-box uses the maximum logic amongst all functional blocks. Therefore, designs with different number of S-boxes show variation in terms of simulation time and area overheads. This section includes results and analysis of every phase required in the design implementation. It includes simulation waveforms and synthesis results with area and power consumption.

5.1 Simulation Waveforms for Phase I and II:

Following figure 35 shows the waveform of functional simulation of RTL design.

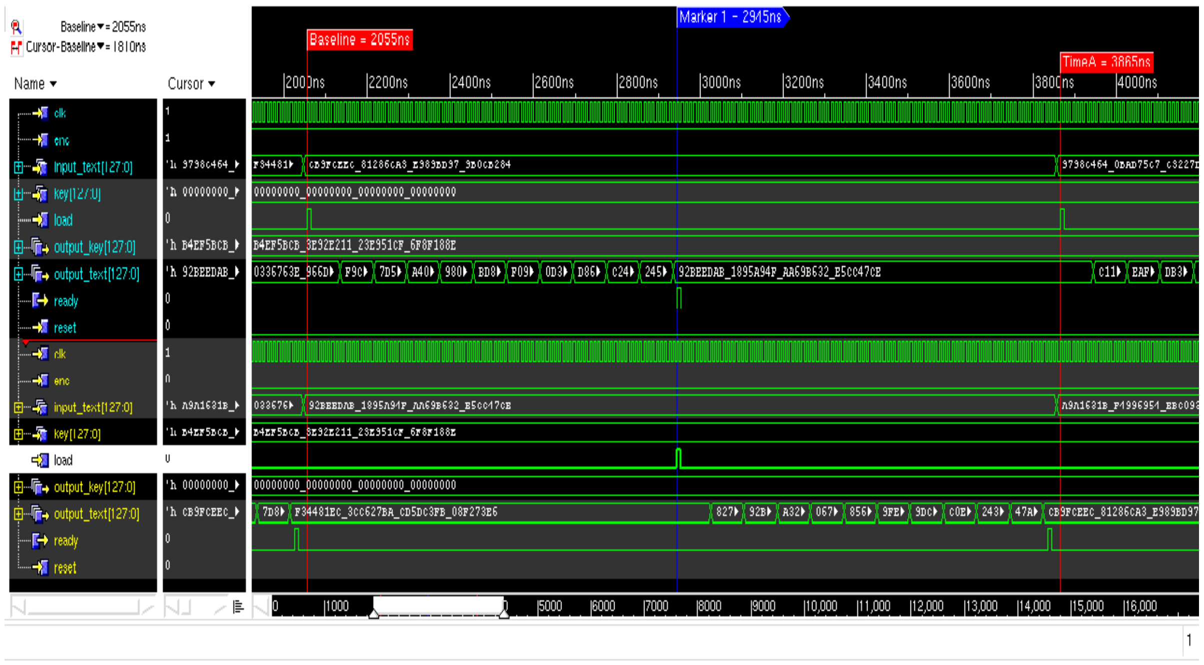

Figure 35 : RTL simulation

The above waveforms show the simulation of two AES modules, one in encryption (signals in blue colour) mode and another in decryption (signals in yellow colour).

As mentioned in the implementation section, the encrypted data is an input to the AES for decryption. The ready signal from first AES is fed as the ‘Load’ input to the next AES. The blue marker in the above diagram points to the ‘ready’ signal of the first AES and the Load signal of another AES. Both are asserted at the same time. Left Red marker points to the Load of first AES. The difference between Blue marker and Left Red marker is 131 clock cycles. This is the input to output latency in the design. Right marker indicates next Load for first AES. The delay between the two loads is 181 cycles. This can be varied from the test bench. Encrypted text is getting driven on “Output_text” port along with the ready signal in the same clock.

Gate level simulation waveforms are very much similar with addition of gate level signals. These waveforms are added in to the appendix section.

5.2 Synthesis Results:

In this section we present the synthesis results obtained during Phase I and Phase II. It includes the area and power results of design before and after insertion of scan chain.

Following results include the area, power and timing information in phase I. These results are without scan chain insertion.

Individual Modules:

Table 12 shows the area consumption of individual modules in the AES.

Table 12 : Area consumption by individual AES modules

Area in um square	Sbox	Shift_row	Mix_column	Key_expander
Combinational Area	34475.187569	1651.520020	34830.009571	54580.265940
Non combinational Area	0.000000	0.000000	0.000000	24772.480469
Cell Area	34475.187500	1651.520020	34830.007812	79352.750000
Design Area	34475.187569	1651.520020	34830.009571	79352.746408
Total Number of Nets	1305	257	1306	4562

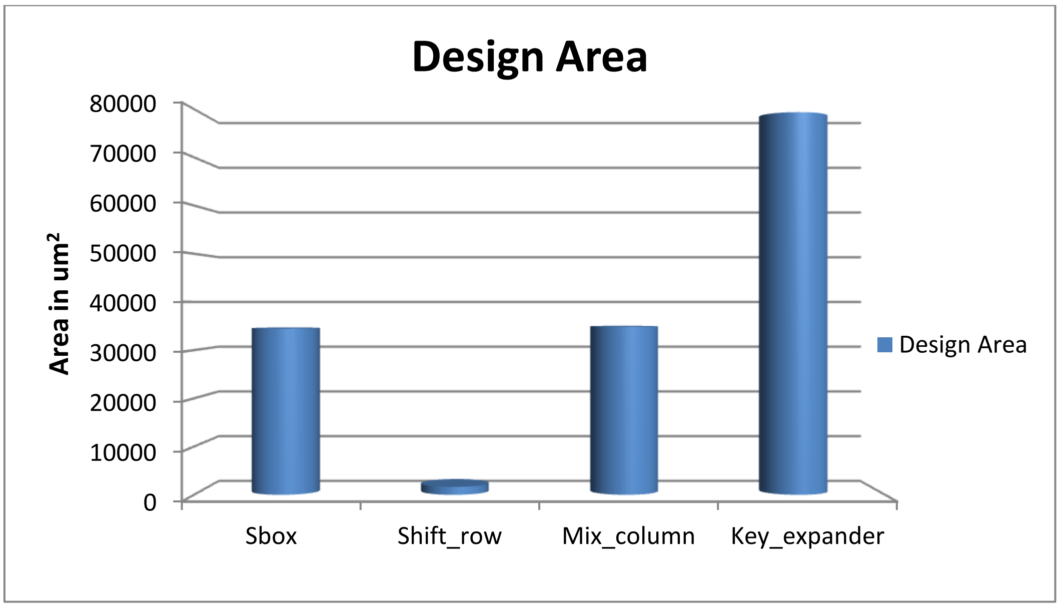

Figure 36 : Design Area occupied by different AES modules

Figure 36 shows the area consumption of each AES module in micrometre square. According to the chart 'key_expander' unit occupies the maximum area, but there are 4 s-boxes inside 'key_expander' module. There are 4, 8 or 16 extra S-boxes in the top module according to the AES design made. So when we consider design as a whole, the area occupied by S-box modules is much larger as compared to any other module.

Area /Power without Scan:

Table 13 shows the overall area occupied by overall AES design in micrometer square. This area calculation is before insertion of scan chain.

Table 13 : Area without Scan

Area in um square	4-S box Design	8- Sbox Design	16-Sbox Design
Combinational Area	190826.211227	229188.149912	306266.880266
Non combinational Area	84917.003445	86316.901474	86720.025517
Cell Area	275743.218750	315505.062500	392986.906250
Design Area	275743.214672	315505.051386	392986.905783
Total Number of Nets	12829	15833	21971

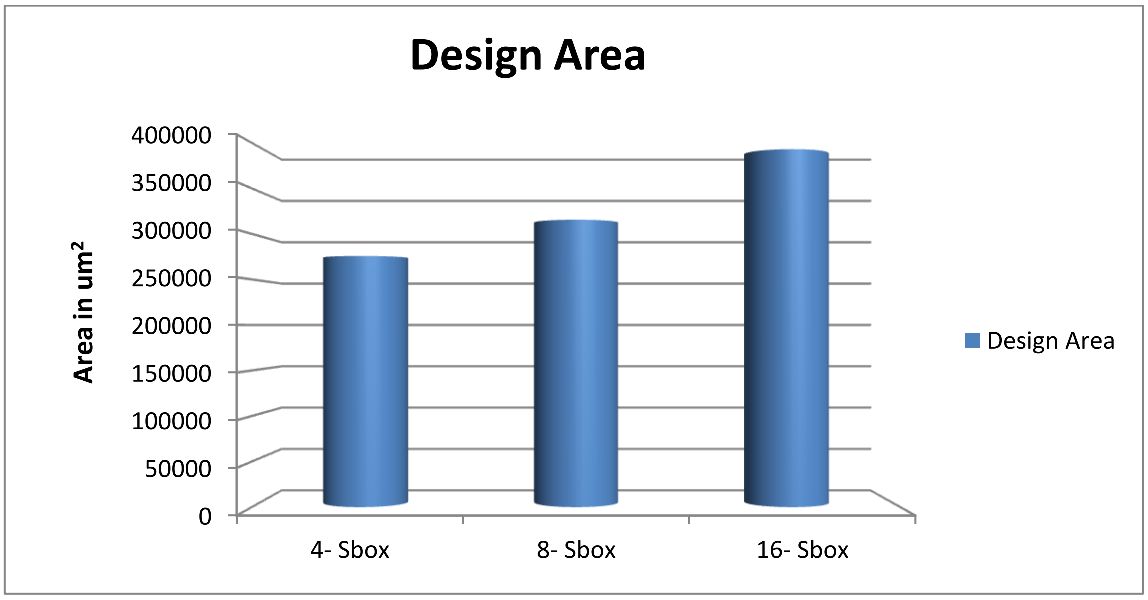

Figure 37 : Total Design Area without Scan

Figure 37 shows the total dynamic area is increasing significantly with the increasing number for S-boxes. From Table 13 we can see there is not much increase in the area occupied by sequential element but combinational area is increasing with bigger margins as S-box utilises only combinational logic.

Table 14 shows power consumptions of AES designs with different number of S-boxes, without scan chain.

Table 14: Power without Scan

Power in mW	4-S box Design	8- Sbox Design	16-Sbox Design
Cell Internal Power	10.7557	10.9813	10.7722
Net Switching Power	2.3310	2.5417	2.6311
Total Dynamic Power	13.0867	13.5230	13.8034
Cell Leakage Power	1.5635	1.8001	2.2606

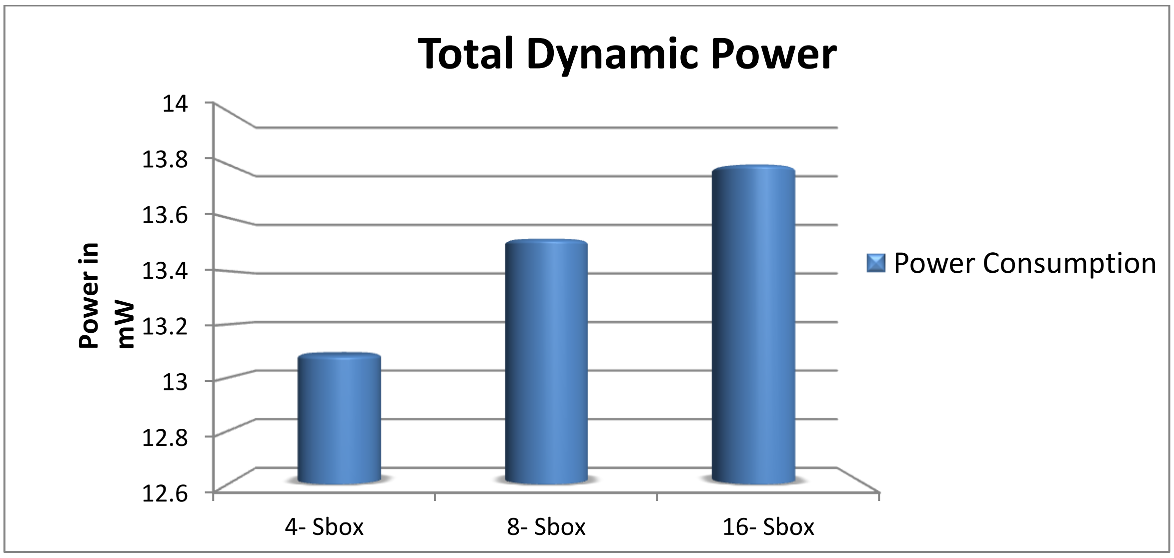

Figure 38 : Dynamic Power without Scan

From Figure 38, we can conclude power increases with the increasing number of S-boxes but there is no drastic increase in the power. Total power increase from 4 S-box design to 16 S-box design, is hardly 0.8mW. This is because sequential elements have not increased much, which consumes more power than combinational logic.

Area /Power with Scan:

From Table 15, we can see that there is not much increase in the combinational area for scan design compared to non-scan design, but is occupied by the sequential elements is increasing with more margins. After insertion of scan design, every sequential element is replaced with a scan sequential element. This actually adds one multiplexer in front of every sequential element which results in more area consumption. Also we can observe uniform sequential area increase in designs with different number of S-boxes compared to their own results for area without scan design.

Table 15: Area consumption With Scan Design

Area in um square	4-S box Design	8- S box Design	16-Sbox Design
Combinational Area	191003.623228	229391.365913	306450.743267
Non combinational Area	110141.832439	111941.723465	112367.489967
Cell Area	301145.468750	341333.093750	418818.218750
Design Area	301145.455667	341333.089378	418818.233234
Total Number of Nets	12847	15855	21992

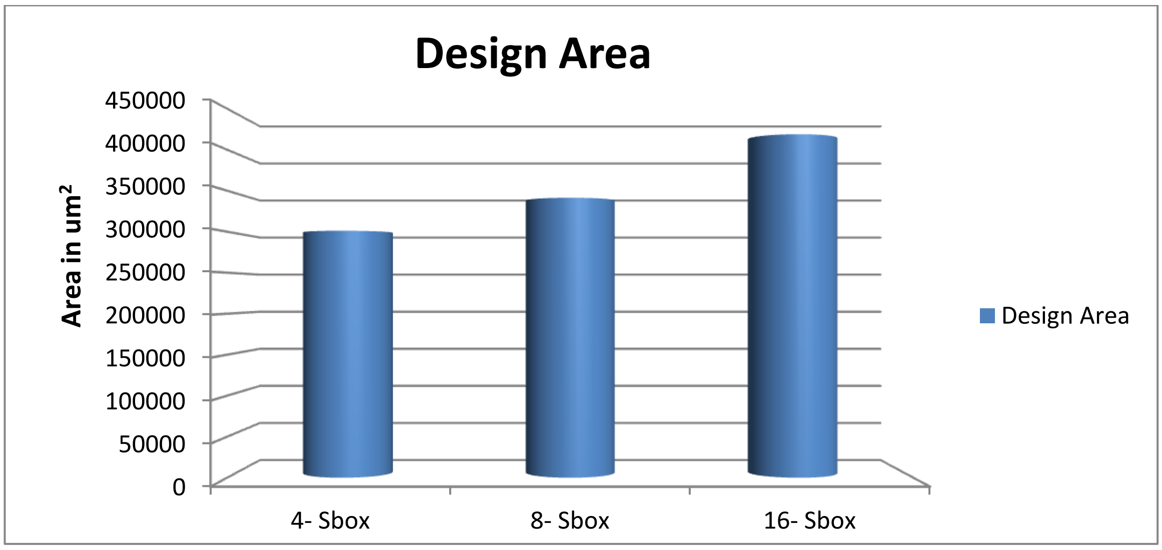

Figure 39: Total Design Area with Scan Design

Table 16 shows the power consumption by AES design after scan chain insertion. Power consumption is more, compared to AES design without scan. There is a significant power increase in the design, but it is necessary to have an error free design. The power difference in different S-box designs before scan chain insertion is very nominal. After scan chain insertion, this difference increases marginally. This is because the sequential elements used in the scan will consume more power than normal sequential element.

Table 16 : Power consumption With Scan design

Power in mW	4-S box Design	8- Sbox Design	16-Sbox Design
Cell Internal Power	15.1194	15.6199	16.7442
Net Switching Power	2.8460	3.1596	3.0728
Total Dynamic Power	17.9653	18.7795	19.8170
Cell Leakage Power	1.6929	1.9342	2.4049

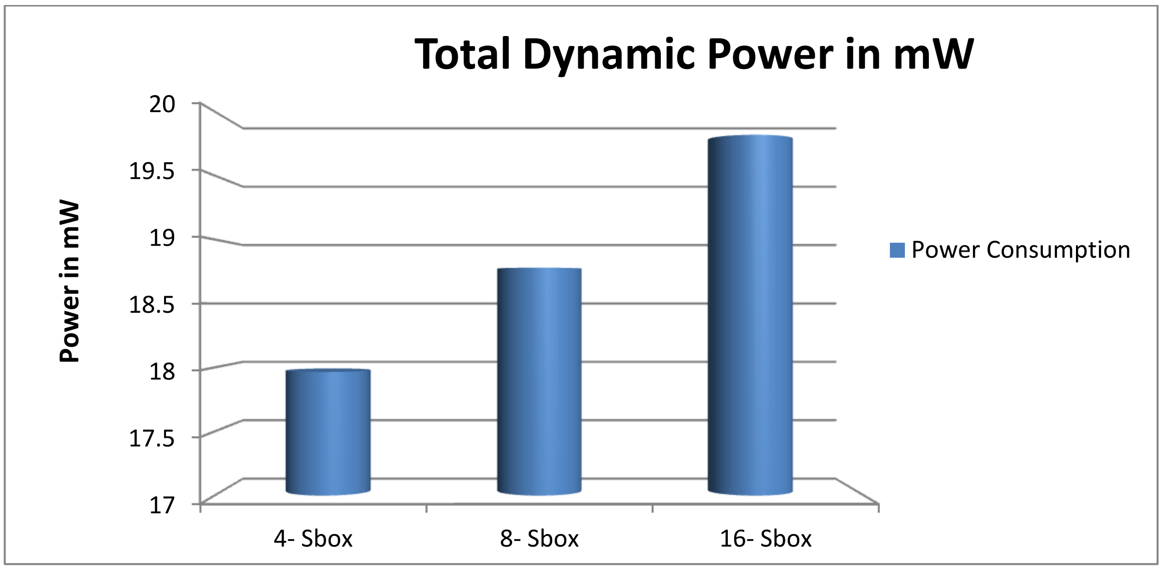

Figure 40 : Total Dynamic Power with Scan design

5.3 ATPG Simulation:

This section includes the results obtained in phase II. It includes the DFT simulation diagrams in case of authorised and unauthorised access. Fault coverage's obtained by DFT patterns and the comparison of simulation time during DFT simulation is shown in the tabular format for different AES designs.

Table 17 shows the simulation time taken by parallel and serial Verilog vectors for DFT simulations for insecure design.

Table 17: Simulation Time for Insecure design

Simulation Time in nS	4-S box Design	8- Sbox Design	16-Sbox Design
Parallel Vectors	38790	48190	62240
Serial Vectors	11081270	14064750	18961340
Total Number Of Patterns	775	963	1244

Table 18 shows the simulation time taken by parallel and serial Verilog vectors for DFT simulations for secure design.

Table 18 : Simulation Time for Secure design

Simulation Time in nS	4-S box Design	8- Sbox Design	16-Sbox Design
Parallel Vectors	39590	48940	62990
Serial Vectors	1137311	14361920	19290590
Total Number Of Patterns	790	978	1259

From Table 17 and 18 we can see that the time for simulation of parallel vectors is much less than the simulation time for serial vectors. Secure design takes longer for simulation, for both serial and parallel vectors and also has more number of patterns to simulate. This is because it has the additional hardware which adds additional faults. To cover these faults more number of patterns is required.

5.4 DFT Simulations

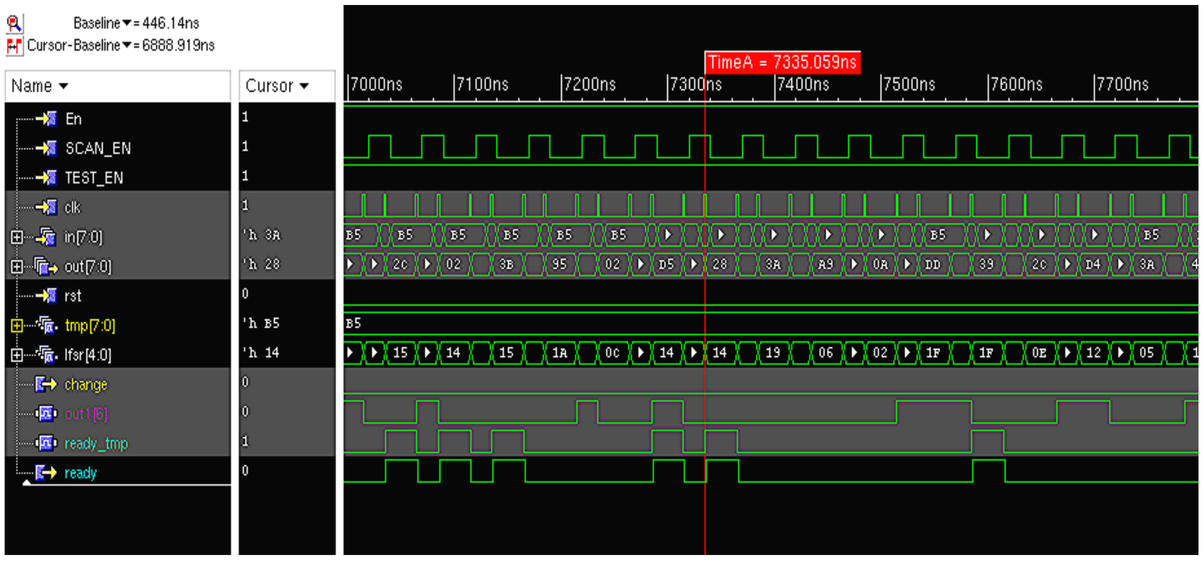

Figure 41: DFT Simulation for authorised access

Figure 41 shows the simulation of an authorised pattern. These patterns hold the integrated key. Signal “tmp” (shown in yellow) gets updated with the key read from dummy flip-flops. Its value is constant throughout simulation. Signal “change” (shown in yellow) is de-asserted because there is no mismatch in the key, “ready” signal is the output of the scan chain, “ready_tmp” and “out [6]” are the inputs for the multiplexer at the output of scan chain and “change” signal is controlling the select line for the multiplexer. The “ready_tmp” signal carries the actual output of scan chain and “out [6]” signal carries the random data. From the waveforms, we can clearly see that output “ready” and “ready_tmp” are exactly matching. This shows that the access is authorised and correct data goes out of the chip.

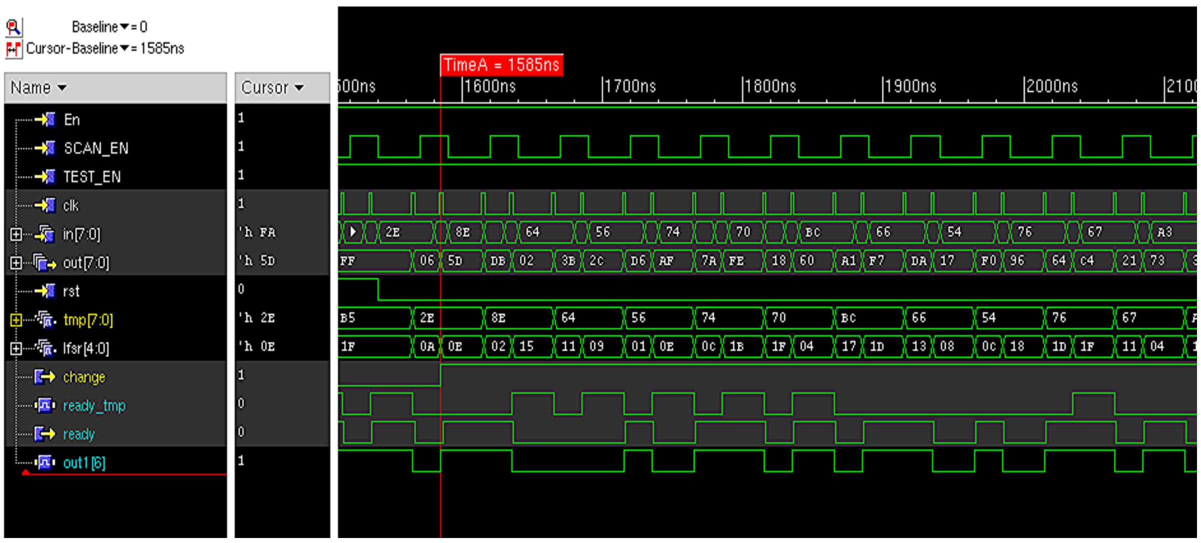

Figure 42: DFT Simulation for Unauthorised access

This simulation is performed using the patterns which don't hold the key. As a result, key mismatch occurs and random data gets driven out of the chip. From the simulation, we can clearly see that the value of "tmp" register is changing continuously because dummy flip-flops are getting driven by wrong data. Circuit detects this mismatch and asserts the "change" signal. Marker A shows the assertion of change signal. We can clearly see that the data going out of the chip is random as output "ready" and "out [6]" random data generated are exactly matching.

5.5 Coverage Analysis:

Table 19 : Coverage Analysis

	4-S box Design	8- S box Design	16-Sbox Design
Total faults	94316	116256	160412
Test coverage	99.00%	99.00%	99.00%
Fault coverage	95.32%	96.17%	97.28%
ATPG effectiveness	99.04%	99.03%	99.02%

We can see from the Table 19 that Test coverage obtained for all the designs is 99 % which is considered to be a good for fault free chip production. There is difference between fault coverage obtained for different AES designs. This is because as hardware increases, number of faults in the design increase. ATPG coverage can be less even though fault coverage is high.

5.6 Functional Simulation with authorised and unauthorised access:

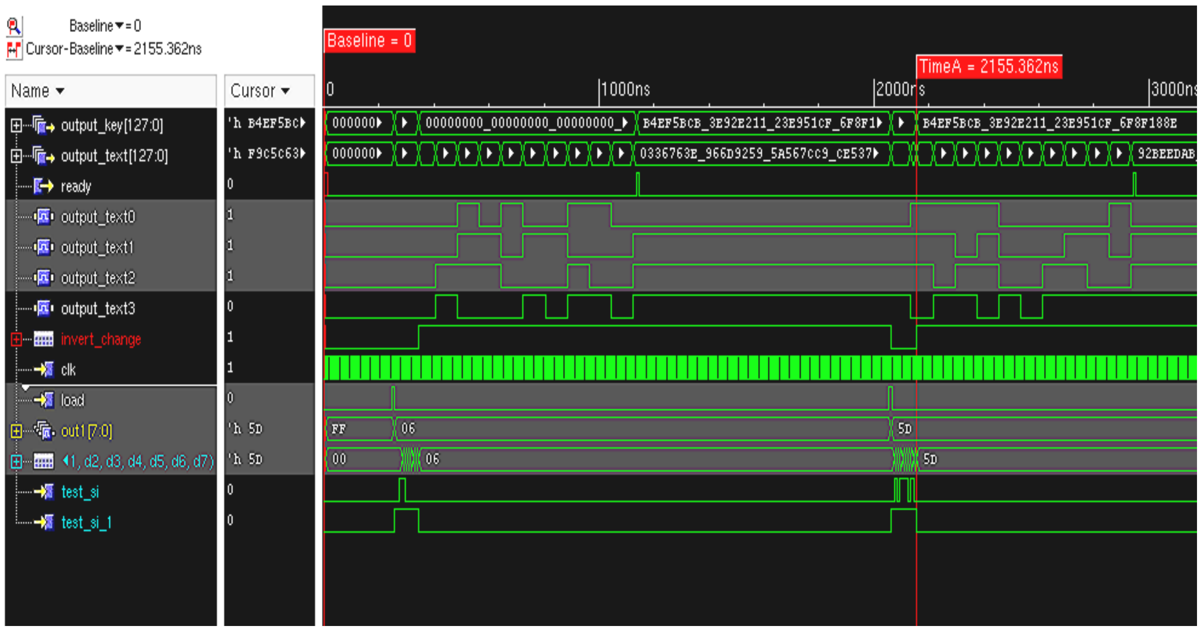

Figure 43 : Functional Simulation for Authorised access

Authorised Access:

The above waveforms (Figure 43) show the functional simulation with authorised access. The "test_si" input (shown in Blue colour) used to drive the key into the chip. Registers d1, d2, d3, d4, d5, d6 and d7 forms the shift register. The output from all registers is concatenated and shown in the waveforms (third last signal) to see the shifted key properly. "Out" signal shows the variable key generated internally to the hardware. "Load" signal indicates start of new iteration and on every new load new key is getting generated. In this simulation the same key is shifted in therefore there is no key mismatch. Therefore "invert_change" signal is getting asserted. This signal shows the inverted behaviour compared to the signal ("change") used in the DFT simulation. Therefore in case of mismatch it gets de-asserted and in case of match it is asserted. The output text0, output text1, output text2, output text3 signals are going out through the multiplexer. These outputs are driven randomly in case of mismatch.

Unauthorised Access:

Figure 44 shows the simulation for unauthorised access. From the waveforms, it can be clearly seen that the key shifted into the chip and key generated internally in the chip are completely different. Therefore mismatch gets detected and "invert_change" signal is de-asserted. In this case the above mentioned output response of above mentioned output signals varies randomly. Both the simulations are performed at gate level.

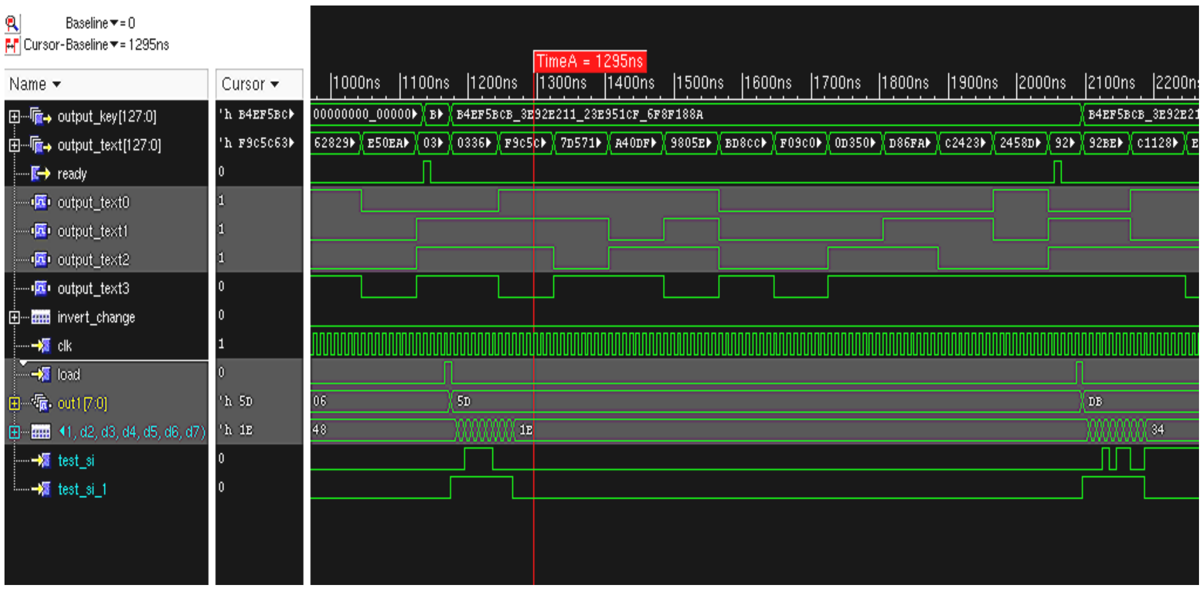

Figure 44 : Functional Simulation for Unauthorised access

5.7 Design Layout:

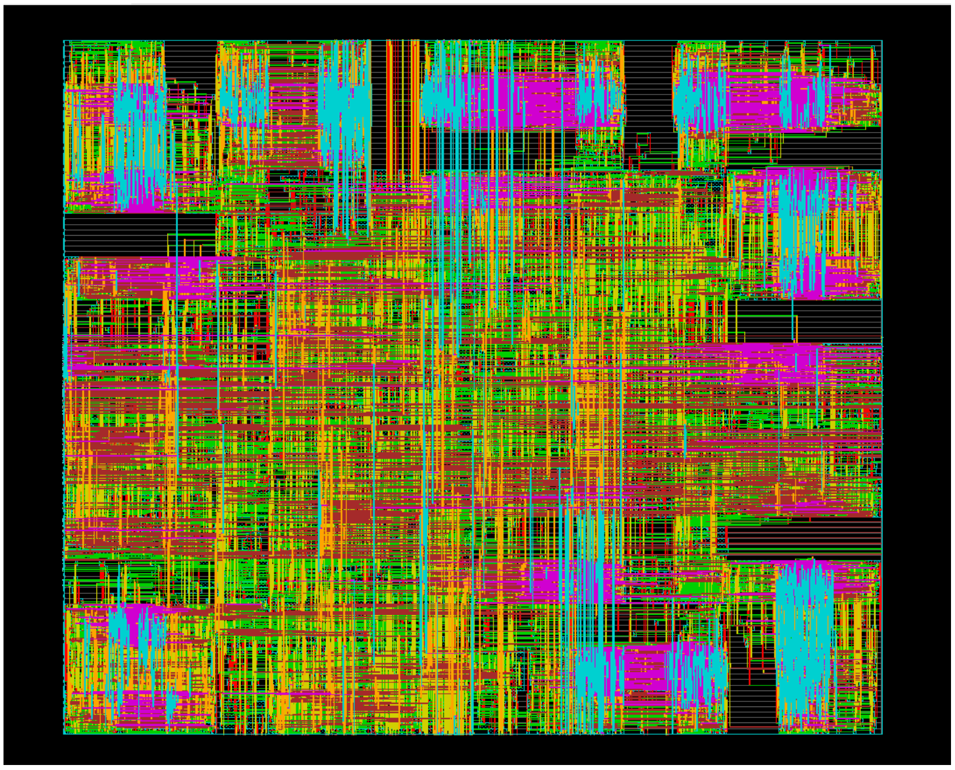

Figure 45 : Design Layout

The above layout is done using "Artisan 180nm" library in "encounter" tool from "Synopsis". The total core area is 426592.051 µm square and total chip area is 427388.506 µm square.

Chapter 6. Critical Analysis

The main objective of the project is to propose a method which provides “Secure Test” and “IP Protection”. For implementation we have chosen AES design which is a popular cryptography technique and is used in several applications. As it holds the critical information, it is often attacked by hackers. As discussed in chapter 3, DFT is the most popular method for testing, but it has proven to be a major security threat over the years. In “Secure Test” we have shown how existing DFT method can be made secured without affecting its testability. For IP protection, we have shown that the IP vender can keep check on the user by using our approach. Thus IP piracy can be controlled.

6.1 Contribution:

- I have developed three different AES designs with variable s-boxes. (4-sbox, 8sbox and 16 sbox)
- The designs were verified for their functionality thoroughly and have been synthesised to generate netlist.
- By using “Design Compiler” the designs were synthesised and made scan testable (DFT).
- The testing patterns were also generated achieving very high fault coverage.
- I designed add-on circuit which can be smoothly integrated with the existing design and makes the design more secure.
- **By generating ATPG test patterns using “TetraMax” I showed, through simulation, that the DFT testing has made the design more robust and only patterns holding secret key can be used to test the circuit**.
- **Functional simulation is also done to show that the implemented variable key technique is an effective way for IP protection**.
- In both the simulations, I have tested passing and failing scenarios. Failing scenarios demonstrates the design behaviour in case of unauthorised access.
- The analysis done for area and power confirmed that the integrated ‘security ad-on’ circuitry does not create much area and power overheads.

6.2 Critical Evaluation of Work Done:

After analysing the work done for the project, all the objectives that had been set have been achieved. We found the novel approach which provides an efficient way for IP protection against IP piracy and at the same time it provides solution for security without sacrificing its testability. There are many methods proposed targeting “Secure Test” and “IP Protection”. But all approaches talk either about “Secure Test” or “IP Protection. Our method gives solution for both “Secure Test” and “IP Protection” at the same time with single add-on circuit.

We are submitting a paper on the basis of work done during this project.
“Pranav Yeolekar, J. Mathew, D.K. Pradhan, “**A New Design Methodology for secure Scan test with IP Protection**” *13th IEEE International Symposium on Quality Electronic Design (ISQED), 2012 (under preparation)”*

Chapter 7. Conclusion and Future Work

7.1 Conclusion:

To meet the set aims and objective we have designed an add-on circuitry which can be smoothly inserted in any design flow and also provide “Secure Test” and “IP Protection” at the same time.

- We have presented a secure scan solution that can be used to prevent scan-based side-channel attacks. By integrating proposed add-on circuitry into the scan insertion flow, the complexity of determining secret information significantly increases. While doing so we have managed to keep the testability of the chip intact. DFT the most popular testing method, provide back door for side channel attacks and creates security threat. With our approach, chip testing is possible only with the test patterns having integrated “secured key”. Our add-on circuitry can detect any unauthorised access to the chip and alters the chip response randomly, which makes any sort of prediction/attack very tough.
- The same add-on circuit also provides IP Protection in the functional mode. We have used a variable key technique to provide IP Protection, which generates random keys during each iteration, helping to avoid unauthorized access to the chip. This helps IP vendors to keep a check on the IP users. The sequence of the variable key will be unknown to any unauthorised user. Therefore even in case of IP piracy, the user will not be able to make use of the chip, unless and until the information of variable keys sequence from IP vendors is given. Also the injection of these keys at particular instance is governed by IP vendors and is passed only to the licensed users.
- We have shown through the results that the security can be achieved without compensating the testability and also IP’s can be protected with very less area and power over heads.
- Our approach provides solution for both the security concerns “Secure Test” and “IP Protection” at the same time which is not implemented before.

7.2 Future Work:

- With our approach, modifications at the Netlist level are mandatory. This is a tough job and takes significant amount of time. An error introduces during these modifications can create issue which will be extremely tough to debug and will create delay in design cycle. This can be avoided by atomising the process. The integration of the circuit and insertion of dummy flops can be automated by making powerful scripts.
- The add-on circuit can be made advanced further to detect the hardware torsions injected during fabrication process.
- Design method can be extended to detect scan based time-bombs.

Chapter 8. Bibliography

[1] B. Yang; K. Wu; R. Karri, “Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard,” International Test Conference, pp 339-344, 2004.

[2] “National Institute of Standards and Technology, Advanced Encryption Standard (AES),” Federal Information Processing Standards Publication 197, 2001.

[3] E. Trichina; T. Korkishko; K. Hee Lee, "Small size, Low power, Side Channel Immune AES coprocessor: Design and Synthesis results," in Advanced Encryption Standard – 4th International Conference, Germany, ed: H. Dobbertin; V. Rijmen; A. Sowa, 2004

[4] P. Kocher, Timing attacks on implementations of diffie-bellman, RSA, DSS, and other systems. pp.104-113, CRYPTO, 1996

[5]J. Blomer; J.P. Seifert, “Fault based cryptanalysis of the advanced encryption standard (AES)”, Financial Cryptography, pp 162-181, 2003

[6]National Bureau of Standards, Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication FIPS PUB 140-1, 1994

[7] J. Lee, M. Tehranipoor, J. Plusquellic, A Low-Cost Solution for Protecting IPs Against Side-Channel Scan-Based Attacks, *Proc. of IEEE VLSI Test Symposium, 2006*

[8] B. Yang, K. Wu, and R. Karri. ”Secure scan: A design-for-test architecture for crypto chips.” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 2006*

[9] D. Hely, M.-L. Flottes, F. Bancel, B. Rouzeyre, and N. Berard. ”Scan design and secure chip,” *10th IEEE International On-Line Testing Symposium (IOLTS04)*

[10] J. Lee, M. Tehranipoor, C. Patel, J. Plusquellic, Securing Design Against Scan-Based Side-Channel Attacks, *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 4, Oct.-Dec. 2007

[11] Unni Chandran and Dan Zhao, SS-KTC: A High-Testability Low-Overhead ScanArchitecture with Multi-Level Security Integration, *Proc. of IEEE VLSI Test Symposium, 2007*

[12] Gaurav Sengar, Debdeep Mukhopadhyay and Dipanwita Roy Chowdhury, Secured Flipped Scan Chain Model for Crypto-Architecture, *IEEE Trans. on*

[13] H. Fujiwara and M. E. J. Obien, Secure and testable scan design using extended de Bruijn graph, *15th Asia and South Pacific Design Automation Conference, Jan. 2010*

[14] R. Karri, K. Wu, P. Mishra, and Y. Kim, “Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1509.1517, Dec. 2002.

[15] M. LaPedus, Qualcomm cracks top-10 in chip rankings. EE Times, August 23, 2007.

[16] Defense Science Board (DSB) study on High Performance Microchip Supply. 2005-02-HPMS Report Final.pdf

[17] E. Castillo, U. Meyer-Baese, A. Garcia, L. Parilla, and A. Lloris,“IPP@HDL: Efficient intellectual property protection scheme for IP cores,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 578–590, May 2007.

[18] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, “Constraint-based watermarking techniques for design IP protection,” *IEEE Trans. Comput.- Aided Design Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252,Oct. 2001.

[19] F. Koushanfar and G. Qu, “Hardware metering,” in *Proc. ACM/IEEE Des.Autom. Conf.*, 2001, pp. 490–493. D. C. Musker, “Protecting and exploiting intellectual property in electronics,” in *Proc. IBC Conf.*, 1998. [Online].

[21]Jarrod A. Roy, Farinaz Koushanfar and Igor L. Marko The University of Michigan, Department of EECS, 2260 Hayward Ave ,MI 48109-2121 Rice University, ECE and CS Departments, 6100 South Main, Houston, TX 77005

[22] HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection by Rajat Subhra Chakraborty, *Student Member, IEEE*, and Swarup Bhunia, *Senior Member, IEEE*

Chapter 9. Appendices

9.1 Part of Code:

```
module
AES(clk,reset,load,enc,key,input_text,outp
ut_text,output_key,ready);

input          clk,reset,load,enc;
input  [127:0]  key,input_text;
wire   [31:0]   keyc0,keyc1,keyc2,keyc3;
wire   [127:0]  key_nxt,key_mc,key_last;
reg    [7:0]    s1_in,s2_in,s3_in,s4_in;
// SBOX input
wire   [7:0]    s1_out,s2_out,s3_out,s4_out;
// SBOX output
reg    [127:0]  sboxout;
reg [1:0]SboxCnt;
reg Over;

// state definitions
parameter IDLE        = 3'b000;
parameter INITXOR     = 3'b001;
parameter SBOX        = 3'b010;
parameter SHIFTROWS   = 3'b011;
parameter MIXCOLUMN   = 3'b100;
parameter NEXTINPUT   = 3'b101;
parameter LASTKEY     = 3'b110;

//FSM

always @ ( CurrentState or SboxOutReady
or Over or load)begin //{
  case (CurrentState) //{
    IDLE : begin//{
      if(load)
           NextState = INITXOR;
      else
         NextState = IDLE;
    end//}

    INITXOR : begin//{
          NextState = SBOX;
    end //}
    SBOX :  begin//{
      if(SboxOutReady)
            NextState = SHIFTROWS;
      else
           NextState = SBOX;
    end //}

    SHIFTROWS : begin//{
      NextState = MIXCOLUMN;
    end //}

    MIXCOLUMN : begin//{
      NextState = NEXTINPUT;
    end //}

    NEXTINPUT: begin//{
      if(Over)
        NextState = LASTKEY;
      else
            NextState = SBOX;
    end //}

    LASTKEY: begin//{
       NextState = IDLE;
    end //}

  default : NextState = IDLE;
  endcase//}

end //}

//assign States
always @ (posedge clk) begin //{
  if (reset)
    CurrentState <= IDLE;
  else
    CurrentState <= NextState;
end //}


// FORM Input

always @ (posedge clk) begin //{
  if (reset)
    text_in <= 128'd0;
  else if(load)
    text_in <= input_text;
end //}
```

```
always @ (posedge clk) begin //{
  if (reset)
    state_in <= 128'd0;
  else if(CurrentState == INITXOR)
    state_in <= text_in ^ key;
  else if(CurrentState == NEXTINPUT)
    state_in <= state_mc_reg ^ key_nxt;
end //}

// Main Counter
always @ (posedge clk) begin //{
  if (reset)
    count <= 4'b1011;
  else if(count == 4'b0000)
    count <= 4'b1011;
  else if(CurrentState == SHIFTROWS)
    count <= count - 1'b1;
end //}

// Over Signal
always @ (posedge clk) begin //{
  if (reset)
    Over <= 1'b0;
  else if(count == 4'b0000)
    Over <= 1'b1;
  else
    Over <= 1'b0;
end //}

// Conditions for SBOX
always @ (posedge clk) begin //{
  if (reset)
    SboxCnt <= 2'b11;
  else if((CurrentState == SBOX)  &
(!SboxOutReady))
    SboxCnt <= SboxCnt - 1'b1;
  else
    SboxCnt <= 2'b11;
end //}

always @ (posedge clk) begin //{
  if (reset)
    SboxOutReady <= 1'b0;
  else if(SboxCnt == 2'b00)
    SboxOutReady <= 1'b1;
  else
    SboxOutReady <= 1'b0;
end //}

//forming inputs to sbox
always @(posedge clk) begin //{
        if (reset) begin //{
           sboxout[127:0] <= 128'd0;
               s1_in <= 8'h00;
           s2_in <= 8'h00;
               s3_in <= 8'h00;
               s4_in <= 8'h00;
        end //}
            else begin
              case(SboxCnt) //begin {
               3'b011: begin //{
                        s1_in <=  state_in[127:120]
;
                        s2_in <=  state_in[95:88]   ;
                        s3_in <=  state_in[63:56]   ;
                        s4_in <=  state_in[31:24]   ;

                        sboxout[103:96]<=s1_out;
                        sboxout[71:64] <=s2_out;
                        sboxout[39:32] <=s3_out;
            sboxout[7:0]   <=s4_out;

               end //}
               3'b010: begin //{
                        s1_in <= state_in[119:112];
                        s2_in <= state_in[87:80]    ;
                        s3_in <= state_in[55:48]    ;
                        s4_in <= state_in[23:16]    ;

                        sboxout[127:120]<=s1_out;
                        sboxout[95:88]  <=s2_out;
                        sboxout[63:56]  <=s3_out;
                        sboxout[31:24]  <=s4_out;

               end //}
               3'b001: begin //{
                        s1_in <= state_in[111:104]
                        s2_in <= state_in[79:72]    ;
                        s3_in <= state_in[47:40]    ;
                        s4_in <= state_in[15:8]     ;

                        sboxout[119:112]<=s1_out;
                        sboxout[87:80]  <=s2_out;
                        sboxout[55:48]  <=s3_out;
                        sboxout[23:16]  <=s4_out;

               end //}
               3'b000: begin //{
                        s1_in <= state_in[103:96]  ;
```

```
                    s2_in <= state_in[71:64]   ;
                    s3_in <= state_in[39:32]   ;
                    s4_in <= state_in[7:0]     ;

                    sboxout[111:104]<=s1_out;
                    sboxout[79:72]  <=s2_out;
                    sboxout[47:40]  <=s3_out;
                    sboxout[15:8]   <=s4_out;

                end //}
              endcase //}
          end //}
end //}

S_box sbox5 (
                .enc(enc),
             .s_in(s1_in),
                .s_out(s1_out)
             );

S_box sbox6 (
                .enc(enc),
             .s_in(s2_in),
                .s_out(s2_out)
             );

S_box sbox7 (
                .enc(enc),
             .s_in(s3_in),
                .s_out(s3_out)
             );

S_box sbox8 (
                .enc(enc),
             .s_in(s4_in),
                .s_out(s4_out)
             );

// Shift rows

assign state_sr_in = sboxout ;

shift_row AES_sr(
                   .enc(enc),
                      .stat_arr_in(state_sr_in),
                      .stat_arr_out(state_sr)
                   );

always @ (posedge clk) begin //{
  if (reset)
    state_sr_reg <= 128'd0;
  else if(CurrentState == SHIFTROWS)
    state_sr_reg <= state_sr;
end //}

// Mix Column

mix_column AES_mc(
                 .enc(enc),
                    .state_in(state_mc_in),
                    .state_out(state_mc)
                 );

assign state_mc_in[127:0] =
state_sr_reg[127:0];

always @ (posedge clk) begin //{
  if (reset)
    state_mc_reg <= 128'd0;
  else if(CurrentState == MIXCOLUMN)
    state_mc_reg <= state_mc;
end //}


// KEY FORMATION

key_expander AES_key(
                   .clk(clk),
                      .enc(enc),
                      .reset(reset),
                      .count(count),
                      .state(CurrentState),
                      .load(load),

.keyin_c0({key[127:120],key[119:112],ke
y[111:104],key[103:96]}),

.keyin_c1({key[95:88],key[87:80],key[79:
72],key[71:64]}),

.keyin_c2({key[63:56],key[55:48],key[47:
40],key[39:32]}),

.keyin_c3({key[31:24],key[23:16],key[15:
8],key[7:0]}),
                      .keyout_c0(keyc0),
                      .keyout_c1(keyc1),
                      .keyout_c2(keyc2),
                   .keyout_c3(keyc3)
                  );
```

```
mix_column invcol_key (
                    .enc(enc),

.state_in({keyc0,keyc1,keyc2,keyc3}),
                       .state_out(key_mc)
                    );

assign key_nxt[127:120]  =  enc ?
keyc0[31:24] : key_mc[127:120];
assign key_nxt[119:112]  =  enc ?
keyc0[23:16] : key_mc[119:112];
assign key_nxt[111:104]  =  enc ?
keyc0[15:8]  : key_mc[111:104];
assign key_nxt[103:96]   =  enc ?
keyc0[7:0]   : key_mc[103:96] ;
assign key_nxt[95:88]    =  enc ?
keyc1[31:24] : key_mc[95:88]  ;
assign key_nxt[87:80]    =  enc ?
keyc1[23:16] : key_mc[87:80]  ;
assign key_nxt[79:72]    =  enc ?
keyc1[15:8]  : key_mc[79:72]  ;
assign key_nxt[71:64]    =  enc ?
keyc1[7:0]   : key_mc[71:64]  ;
assign key_nxt[63:56]    =  enc ?
keyc2[31:24] : key_mc[63:56]  ;
assign key_nxt[55:48]    =  enc ?
keyc2[23:16] : key_mc[55:48]  ;
assign key_nxt[47:40]    =  enc ?
keyc2[15:8]  : key_mc[47:40]  ;
assign key_nxt[39:32]    =  enc ?
keyc2[7:0]   : key_mc[39:32]  ;
assign key_nxt[31:24]    =  enc ?
keyc3[31:24] : key_mc[31:24]  ;
assign key_nxt[23:16]    =  enc ?
keyc3[23:16] : key_mc[23:16]  ;
assign key_nxt[15:8]     =  enc ?
keyc3[15:8]  : key_mc[15:8]   ;
assign key_nxt[7:0]      =  enc ? keyc3[7:0]
: key_mc[7:0]    ;


// Final Outputs

always @ (posedge clk) begin //{
  if (reset)
    ready <= 1'b0;
  else if (count == 4'b0000)
   ready <= 1'b1;
  else
   ready <= 1'b0;
end //}

always @ (posedge clk) begin //{
  if (reset) begin //{
    output_key <= 128'd0;
  end//}
  else if((count == 4'b0000)) begin //{
    output_key <= key_nxt;
  end//}
end //}

always @ (posedge clk) begin //{
  if (reset) begin //{
    output_text <= 128'd0;
  end//}
  else if ((CurrentState == SHIFTROWS))
begin //{
    output_text <= state_sr_reg ^ key_last;
  end//}
end //}

endmodule
```

9.2 Part of netlist after modifications

The following library cells show the insertion of Dummy flip-flop at netlist level.

```
HDSDEPQ1 state_mc_reg_reg_9_ ( .D(N1070), .SD(state_mc_reg[8]), .CEB(n1670),
       .SE(SCAN_EN), .CK(clk), .Q(state_mc_reg[9]) );
  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  HDSDEPQ1 state_mc_reg_reg_10_ ( .D(N1071), .SD(state_mc_reg[9]), .CEB(n1670),
       .SE(SCAN_EN), .CK(clk), .Q(state_mc_reg[10]) );
  HDSDEPQ1 Dummy_0 ( .D(dd0), .SD(state_mc_reg[10]), .CEB(den), .SE(SCAN_EN),
.CK(clk), .Q(d0));
  HDSDEPQ1 state_mc_reg_reg_11_ ( .D(N1072), .SD(d0), .CEB(n1670), .SE(SCAN_EN),
.CK(clk), .Q(state_mc_reg[11]) );
 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  HDSDEPQ1 state_mc_reg_reg_12_ ( .D(N1073), .SD(state_mc_reg[11]), .CEB(n1670),
.SE(SCAN_EN), .CK(clk), .Q(state_mc_reg[12]) );
  HDSDEPQ1 state_mc_reg_reg_13_ ( .D(N1074), .SD(state_mc_reg[12]), .CEB(n1670),
.SE(SCAN_EN), .CK(clk), .Q(state_mc_reg[13]) );
```

9.3 Paper to be submitted:

Pranav Yeolekar, J. Mathew, D.K. Pradhan, "**A New Design Methodology for secure scan test with IP Protection**" *13th IEEE International Symposium on Quality Electronic Design (ISQED), 2012 (under preparation)*