CHAPTER 6

PREVENTION OF AES AGAINST DPA ATTACK USING MODIFIED NEW FULLY PIPELINED TECHNIQUE

6.1 OVERVIEW

A fast and area efficient composite field implementation of the byte substitution phase is designed using an optimum number of pipeline stages against DPA for the prevention of AES implemented in FPGA. By the word "differential", the DPA takes the differences of power consumptions among different samples with different operations or with different data processed Paul Kocher et al (1999); Sokolov et al (2005) and Hiren Patel & Rusty Baldwin (2013). For Differential Power Analysis, an attacker does not need information about the analyzed hardware nor about the points in time during processed of information. Using DPA, the adversary doesn't have to know implementation details of the attacked device. If Smart Cards are unprotected against DPA, it is possible to reveal the secret key by measuring and analyzing the power consumption.

6.2 PIPELINED AES ARCHITECTURE

For AES with 128 bit inputs, key and cipher text in a 4x4 matrix. This matrix is known as State matrix as shown in the figure 6.1. From this state matrix each conversion in every round is achieved in a 32 bit pipeline manner.
Each value inside the matrix is 8bit. As we know out of the four modules, shift rows, mix columns and add round key all are linear conversions except the s-box substitution. In AES algorithm is key schedule and round transformation are the two main processes.

Key scheduling consist of two modules: expansion of the key and round key selection. Key expansion means mapping $N_k$ bits first key to the so-called expanded key, while the round key selection select $N_b$ bits of rounds key from the expanded key unit.

Round key conversion involves four modules by Byte substitution, Byte Rotation, Mixcolumn and addround key. In the round transformation, the Byte Rotation, Mixcolumn and addround key all are the four transformation units and all are linear transformation except the Bytesub.

6.2.1 The Procedure for New Algorithms

The inputs of original plain text and the first key, intermediate inputs and outputs of round transformation, as well as the output of code text in the AES algorithm are all stored in the state matrices, which are processed in one byte or one word. Thus, the original 128-bit data should be segmented in order to take operations at least bits. We design some peripheral controllers.
in the new algorithm, so that the information transmission and processing can be implemented on each column of the state matrix (32bit). This means that the information should be well packed and put into further operations. In this new algorithm, in substitute byte each byte is replaced with its corresponding byte using composite field which means GF(2^8) is replaced with the mapping of GF((2^3)^3)^2 which leads to reduction in the resource area.

Figure 6.2 Byte segmentation and replacement process

The independent and reversible bytes substitution operation of S-box is taken as an example. First, the state matrix is divided into four columns. Then the byte replacement is achieved by the operation of composite Field shown as Figure 6.2.

6.2.2 S-Box Construction Methodology

The steps involved in constructing the multiplicative inverse module for the s-box using composite field arithmetic is explained below. Since both the Subbyte and inverse subbyte conversions are similar except the operation which involve the affine transformation and its inverse, hence their implementation of the subbyte operation is discussed in the following steps.
The bits are considered individually in a byte representing a $\text{GF}(2^8)$ element can be viewed as coefficients to each power term in the $\text{GF}(2^8)$ polynomial. For example $\{10001011\}_2$ can be expressed a polynomial $q^7+q^3+q+1$ in $\text{GF}(2^8)$.

Multiplicative inverse can be evaluated using the equation given below.

$$(bx+c)^{-1} = b(b^2B+bcA+c^2)^{-1}x+(c+bA)(b^2B+bcA+c^2)^{-1}$$

The above equation includes mathematical operations like multiplication, addition, squaring and multiplication inversion in $\text{GF}(2^4)$ operations in Galois Field. Each of above operators could be transformed into individual blocks when constructing the circuit for calculating the multiplicative inverse. The above simplified equation, from which the multiplicative inverse circuit $\text{GF}(2^8)$ that could be produced is as shown in Figure 6.3.

![Figure 6.3 Multiplicative Inversion Module for the S-Box](image-url)
Isomorphic mapping and Inverse isomorphic mapping

The multiplicative inverse computation would be done by decomposing the more complex GF(2^8) to lower order fields of GF(2), GF(2^2) and GF((2^2)^2). The following irreducible polynomials are used in order to achieve the above equation.

\[
\begin{align*}
\text{GF (2)} & \quad \longrightarrow \quad \text{GF(2): } x^2 + x + 1 \\
\text{GF ((2^2)^2)} & \quad \longrightarrow \quad \text{GF (2^2): } x^2 + x + \varphi \\
\text{GF (((2^2)^2)^2)} & \quad \longrightarrow \quad \text{GF ((2^2)^2): } x^2 + x + \lambda
\end{align*}
\]

Where \( \varphi = \{10\}_2 \) and \( \lambda = \{1100\}_2 \).

Calculation of the multiplicative inverse in the composite field could not be directly applied to an element which is based on GF (2^8). That element has to be mapped to its composite field representation via an isomorphic function denoted by \( \iota \). Similarly, after obtaining the multiplicative inversion, the result must also be mapped back from its composite field representation to its equivalent in GF(2^8) through the inverse isomorphic function.

The squaring mathematical operation is done using the following steps in GF(2^4).

\textbf{Squaring in GF(2^4)}

\[
\begin{align*}
K_3 &= q_3 \\
K_2 &= q_3 \oplus q_2
\end{align*}
\]
\[ K_1 = q_2 \oplus q_1 \]
\[ K_0 = q_3 \oplus q_1 \oplus q_0 \]

The following equation is realized using the hardware logic diagram shown in the Figure 6.4.

**Figure 6.4 Hardware Diagram for Squarer in GF(2^4)**

The four bits are represented as \( q_3 \ q_2 \ q_1 \ q_0 \) and all the mathematical operations are performed respectively as shown in the equation.

The multiplication operation with a constant \( \lambda \) in the Galois Field GF(2^4) is calculated using the equations as follows:

**Multiplication with constant, \( \lambda \)**

\[ K_3 = q_2 \oplus q_0 \]
\[ K_2 = q_3 \oplus q_2 \oplus q_1 \oplus q_0. \]
\[ K_1 = q_3, \]
\[ K_0 = q_1 \]
The above equations are realized using the hardware logic diagram shown in the Figure 6.5.

The hardware logic diagram for the multiplication operation in GF(2^4) is shown in the following Figure 6.6. The resulting value is also four bit representation. The four bit is divided into two 2 bits and multiplication is done in GF(2^2) separately and later they are combined to get 4 bit value.

Instead of doing all these mathematical operations in GF(2^8), it is first split into Galois Field GF(2^4) and later split into Galois Field GF((2^2)^2) to do the operations so that the resources used for the operations are minimized and reduction in area is achieved. This is the main reason for doing conversion in substitute byte in composite field.
**GF ($2^4$) Multiplication**

![Diagram of multiplication in GF($2^4$)](attachment:image)

**Figure 6.6 Hardware implementation of multiplication in GF($2^4$)**

The hardware implementation of multiplication in Galois Field GF($2^4$) shown in the Figure 6.6.

The multiplication of two numbers in the Galois Field GF($2^4$) for all possible hexadecimal values are done and pre calculated and shown in the following Table 6.1.
Table 6.1 Pre-computed multiplication result of 2 elements in GF $(2^4)$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>D</th>
<th>e</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>D</td>
<td>e</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>A</td>
<td>b</td>
<td>9</td>
<td>c</td>
<td>E</td>
<td>f</td>
<td>d</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>C</td>
<td>F</td>
<td>d</td>
<td>e</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>B</td>
<td>9</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>C</td>
<td>6</td>
<td>2</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>F</td>
<td>3</td>
<td>7</td>
<td>d</td>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>a</td>
<td>F</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>d</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>c</td>
<td>1</td>
<td>4</td>
<td>b</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>b</td>
<td>D</td>
<td>E</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>c</td>
<td>a</td>
<td>9</td>
<td>F</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>E</td>
<td>A</td>
<td>D</td>
<td>3</td>
<td>4</td>
<td>f</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>c</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td>c</td>
<td>4</td>
<td>B</td>
<td>3</td>
<td>7</td>
<td>f</td>
<td>d</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>E</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>9</td>
<td>e</td>
<td>7</td>
<td>F</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>C</td>
<td>b</td>
<td>2</td>
<td>a</td>
<td>3</td>
<td>4</td>
<td>D</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>a</td>
<td>f</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>c</td>
<td>6</td>
<td>1</td>
<td>B</td>
<td>e</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>d</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>b</td>
<td>d</td>
<td>6</td>
<td>7</td>
<td>C</td>
<td>a</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>f</td>
<td>e</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>c</td>
<td>4</td>
<td>8</td>
<td>D</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>A</td>
<td>2</td>
<td>e</td>
<td>b</td>
<td>7</td>
<td>f</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>d</td>
<td>6</td>
<td>B</td>
<td>9</td>
<td>4</td>
<td>f</td>
<td>2</td>
<td>e</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>A</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>e</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td>B</td>
<td>2</td>
<td>c</td>
<td>a</td>
<td>4</td>
<td>d</td>
<td>3</td>
<td>f</td>
<td>1</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>f</td>
<td>5</td>
<td>A</td>
<td>1</td>
<td>E</td>
<td>4</td>
<td>b</td>
<td>2</td>
<td>D</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>C</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

From Table 6.1, the results for multiplication with constant $\lambda$ and squaring operation in GF $(2^4)$ can also be obtained.

**GF $(2^4)$ Multiplication**

$$K_1=q_1w_1 \oplus q_0w_0 \oplus q_1w_0$$

$$K_0 = q_1w_1 \oplus q_0w_0$$

The hardware realization of multiplication in GF$(2^4)$ is shown in the Figure 6.7.
The hardware realization of multiplication of a number with a constant in GF(2^2) is shown in the Figure 6.8.

**Multiplication with constant φ**

\[ K_1 = q_1 \oplus q_0 \]

\[ K_0 = q_1 \]

**Figure 6.8 Hardware implementation of multiplication with constant φ**

**Multiplicative inversion in GF(2^4)**

The formula to compute the multiplicative inverse of q (where q is an element of GF(2^4)) such that \( q^{-1} = \{q_3^{-1}, q_2^{-1}, q_1^{-1}, q_0^{-1}\} \). The inverses of the individual bits can be computed from the equation below and the pre-computed values are shown in the figure 6.9.
$q_3^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_0 \oplus q_2$

$q_2^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_0 \oplus q_2q_1$

$q_1^{-1} = q_3 \oplus q_3q_2q_1 \oplus q_3q_1q_0 \oplus q_2 \oplus q_2q_0 \oplus q_1$

$q_0^{-1} = q_3q_2q_1 \oplus q_3q_2q_0 \oplus q_3q_1q_0 \oplus q_3q_0 \oplus q_2 \oplus q_2q_1 \oplus q_2q_1q_0 \oplus q_1 \oplus q_0$

<table>
<thead>
<tr>
<th>Q</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q^{-1}$</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>F</td>
<td>C</td>
<td>9</td>
<td>b</td>
<td>a</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>e</td>
<td>D</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 6.9 Pre-computed results of the multiplicative inverse operation in GF($2^4$)

6.3 PIPELINING TECHNOLOGY FOR HIGH SPEED

Pipelining technology used in AES is mainly meant for increasing the speed of the cryptosystem. The following block diagram explains how the pipelining technology is performed in the every round transformation as shown in the Figure 6.10 below.

Figure 6.10 Pipelining Technology in Round Transformation
The operation of the above four groups of information could be realized in pipelining technology. In the pipelining process, the 128-bit data is further divided into four consecutive 32-bit packets that take round transformation autonomously. The 128 bits which are not processed is stored in the 128 bit register. Then the 128 bit is divided into four consecutive 32 bit units. The round conversion on that 32 bit unit operates independently. When the operation is completed the 32 bit units are later joined to form a 128 bit and again feedback to 128 bit register. The functions of various parts of the structure shown above are explained as follows:

→ The first round of encryption

The four packets of consecutive 32-bit plaintext (128 bits) have been placed into the correlative registers. Then, another four packets of consecutive 32-bit initial key (128 bits) have been put into other registers by the control of the enable the clock signal. Moreover, this unit should be combined with the original plaintext and initial key by using the XOR operators.

→ Round Transformation in the intermediate steps

A round transformation mainly realizes the function of SubBytes and MixColumns with 32-bit columns. Independently process the four packets of round transformation. Then by using XOR operators the results of MixColumns and the 32-bit keys sourced from Key expansions are joined. Here, the round transformation is a module with 64 input ports (32-bit plaintext+32-bit key) and 32 output ports. The function of SubByte is realized by composite field. The implementation of MixColumn is mainly based on the mathematical analysis in the Galois field GF(2^8). Only the multiplication module and the 32-bit XOR module of each processing unit (one column) are needed to be designed, because the elements of the multiplication and
addition in Galois field are commutative and associative. Then the function MixColumn can be achieved.

\[ \rightarrow \text{The process of the last round} \]

The final round is a 128-bit processor. After nine rounds of operations included ShiftRows, SubByte and MixColumns, the intermediate 128-bit encrypted data will be used in XOR operation with the final expanded key(4*32bit), which is provided by the key expansion module. The output of final round in the processor is the desired 128-bit codetext. Similarly, the codetext is divided into four packets of 32bit data by an external enable signal.

6.4 FULLY PIPELINING ARCHITECTURE

Figure 6.11 shows the architecture of proposed full pipelined AES processor. In this figure architecture is composed of ten AES functional blocks and key expansion circuits. The inter-pipelined and outer-pipelined schemes are utilized for implementations. In the inter-pipelined scheme, the register arrays are assigned among the operational circuits of SubBytes, ShiftRows, MixColumns and AddRoundKey.

The SubBytes block is further divided into three pipelined phases. Therefore, the AES i functional blocks, for i=1,2, 3, ..., 10 are segmented into five inter-pipelined phases. In the outer pipelined scheme, several register arrays are added between each AES round computation. Thus, the latency delay of the proposed full-pipelined AES processor is 51 clock cycles. The throughput of this architecture is 128 bits per clock cycle.
6.4.1 Encryption – Pipeline Design

The AES algorithm encryption for a pipeline design is presented in Figure 6.12 one for each round of the encryption process this design consists of ten stages.

Figure 6.12 AES algorithm encryption-Pipeline Design

The input plain-text is received at each clock cycle through input register (INPUT- REG). Round keys (RKs) are provided to each stage
permanently. At each clock cycle, data is shifted to next stage and in the same way; output code-text will appear at each clock cycle after whole pipeline has been filled. The internal design of each round is quite similar as one explained in last section for sequential design. The difference is that, the rounds are now replicated and, therefore, this design occupies more resources. It is important to note that in pipeline design, ROMs used in the sequential design, are replaced with BlockRAMs (BRAMs). BRAMs are memory modules available in Virtex and VirtexE family of devices. They consume less area than configuring LUTs as ROMs. The said FPGAs families contain more than 280 BRAMs and this is the reason why this family is well suited for the implementation of a pipeline design. The sequential design occupies 16 ROMs, but pipeline design uses 8 BRAMs for each round, hence a total of 80 BRAM are needed. At more area consumption, throughput of the design has been increased to approximately 10 times too. Since internal round design for sequential and pipeline architectures are identical, here we only provide details for key scheduling of our pipeline design.

6.4.2 Key Schedule

For a pipeline approach, all round keys must be available at the same time for each one of the ten AES rounds. Therefore, the same key generator (KEY-GEN), which is used in case of sequential design, has to be replicated for each round as shown in Figure 6.13.

![Figure 6.13 Key schedule for encryption- pipeline design](image-url)
The user key is accepted at beginning and then round keys (RKs) are generated in subsequent stages. The only difference between key generator for a sequential design described is use of BRAMs instead of ROMs. As a dual port BRAM can be configured to two single port BRAMs, therefore, in a single KEY-GEN, instead of four ROMs, two BRAMs are used, hence a total of 20 BRAMs.

6.5 DPA OF A FPGA IMPLEMENTATION

The first step of the DPA attack is to find the point of the measurements

- The highest seven spikes show the end of seven EC point doubling operations.
- The first one corresponds to the end of the first EC doubling operation. This spike shows the ending of the second operation which is $Q \rightarrow 2P$ and this step is executed independent from the key bits.
- Hence our choice for the measurement point is the second update of $Q$ after the second EC point doubling (step 3)
- We use the transitions between the previous value of $Q$ $2p$, and the new value at our target point, $4p$ or $6P$ according to the value of $k$ as the power consumption predictions.

6.5.1 A DPA using Simulated Data

- Dynamic power consumptions is predicted using Behavioural HDL simulations.
- It allows simulating attacks in an early stage of the design flow. After each AES execution, we did not reset the chip. At the
beginning of an AES execution, the state still contained some value which is related to the previous AES execution, produced a simulated power consumption file. We have chosen N random plaintexts and one fixed, but random key. After each first clock cycle, the simulator has written the total number of bit changes between the previous and the current values of the state to those file. Hence, the simulator has produced a file which contains an N x 1 matrix.

### 6.6 SIMULATION RESULTS

In AES, the final round sub key is recovered with large amount of power traces by the attackers. Our method shows that the power consumption is almost constant as shown in figure 6.11 and even after 17,230 power traces the correct key was not successfully traced. This simulation result is achieved using microwind software where it is possible to measure the power consumed by the different operations involved in AES encryption algorithm.

The realization of pipelined architecture of high-throughput 128 bits AES cipher processor in Vertex III FPGA by new high-speed and hardware sharing functional blocks are shown in Figure 6.14a (Hardware Setup) and figure 6.14b (VHDL Simulation). It can be shown that the speed of the AES can be increased while reducing the delay achieved by introducing pipelining not only in the in between operations of AES but also in every round of AES. Hence, it is mentioned as fully pipelined process. The hardware implementation shows that power traces are almost constant and hence, the hacker is unable to find out the exact key matching point. When the hacker’s key matches with the exact key a large spike will occur and he will come to the conclusion that he has found the original key. Since, the power traces are constant, the hacker won’t be able to find the coincidence of his
value with the original value. By this way we can prevent the AES from the Differential Power Analysis attack.

Figure 6.14(a) Fully Pipelined combinational AES (Hardware setup)

Figure 6.14(b) Fully pipelined combinational AES (VHDL Simulation)
By using the decryption program operated on 8-bit portion of the code(cipher) text for 256 times i.e. for all possible combination the correlation value does not seem to be differentiated from the correct key guess.

The memory complexity is dramatically reduced using the Content-Addressable Memory (CAM) compared to the SRAM based S-box and Inverse S-box look-up tables. The new hardware sharing architecture is applied to implement the proposed high-speed secure encryption. The resource utilized is given in Table 6.2. The description of the measured parameter for the proposed method is compared with conventional methods shown in Table 6.2.

**Figure 6.15 Synthesized result of full pipelined AES**

A comparison of synthesized results of AES using the device XC6VLX75T-2FF784(i) Polynomial based masking in S-Box, (ii)Cipher text generated by AES by introducing pseudo random generator and (iii) Introducing full pipeline in the AES using Content Addressable Memory in S-
box and Inverse S-box which result in the increase in the speed of the encryption process shown in the Table 6.2.

<table>
<thead>
<tr>
<th>Designs</th>
<th>Clk Frequency (MHz)</th>
<th>Area (No. Of slices reg)</th>
<th>Area (LUT)</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES Full pipeline</td>
<td>393.499</td>
<td>3440</td>
<td>3283</td>
<td>2.541</td>
</tr>
<tr>
<td>AES_cipher</td>
<td>277.277</td>
<td>410</td>
<td>700</td>
<td>3.607</td>
</tr>
<tr>
<td>AES_Polynomial</td>
<td>321.849</td>
<td>5088</td>
<td>16112</td>
<td>3.107</td>
</tr>
</tbody>
</table>

The bar graph for the above table is shown below.

![Bar graph for Comparison of Polynomial and Pipelined](image)

**Figure 6.16** Bar graph for Comparison of Polynomial and Pipelined

### 6.7 PROPOSED WORK

In this work, an effective and more powerful enhanced DPA method to protect the secret key from an AES hardware implementation is
presented. The combinational circuit design implemented for AES is highly inaccessible for power analysis attack. The result executed shows reduced resource utilization of minimum 50% and the increased number of traces. The power traces are captured and the key extraction is also done.

In general, it is clear that the DPA is possible because of the power radiated or leaked during the process by the registers used. The attack in the first round of AES is not possible because of the use of XOR gate [combinational circuits] between plain text and key before loading them into the register for successive processing. Instead of register, randomization is added the processing [Key generation and encrypting] has been implemented using combinational circuit to minimize the power leakage due to state change in registers. The proposed architecture consists of random delay combinational circuit design for initial and final round processing, preventing the uniform power fluctuation.

![Sequential Circuit](Image)

**Figure 6.17 Sequential Circuit**

In AES attack program, the 128-bit cipher text message is split into byte long blocks. AES decryption algorithm operates on each byte individually and key guess is done for each 8-bit portion of the round key.
The relationship between the hamming distance of the bits in the data registers before and after 10th round of encryption and power fluctuation has been used to decode the key/data.

![Combinational Circuit](image)

**Figure 6.18 Combinational Circuit**

In AES algorithm the sequential circuit used in the circuit can be modified to combinational logic circuit, so that processing does not change the output for the given input combination. By removing clock signal, the chance for the hacker to grasp the information is entirely avoided thereby preventing the Differential Power Analysis attack.

By replacing sequential circuit to combinational circuit the power variations in the circuit become constant and no chance of identifying the instant of data transfer. The experimental results for power variations are obtained using for existing and proposed methods using micro-wind software are shown as in Figures 6.19 and 6.20 respectively. Similarly the simulation results for initial and final round of the AES coding and complete power utilization have been given in corresponding figures respectively.
The power traces of the circuit is observed using microwind software and power traces of combined circuit and power traces of last round of AES is shown in the figure 6.21 and 6.22 respectively.
The power traces are observed using the mirowind software by changing all the sequential circuit into combinational circuit in the steps involved in AES as shown in the figures 6.18-6.21. We can observe that the power consumed is constant and hence no chance for the hacker to identify the transition of information.
The novel hardware sharing architecture is applied to implement the proposed high-speed secure encryption. The resource utilized is given in Table 6.3.

**Table 6.3 Comparison of Device Utilization**

<table>
<thead>
<tr>
<th>Author</th>
<th>Device</th>
<th>Device Utilization</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premier Researchers</td>
<td>Spartan3</td>
<td>2154</td>
<td>Parallel 802.11i</td>
</tr>
<tr>
<td>Premier Researchers</td>
<td>Spartan3</td>
<td>5605</td>
<td>Sequential</td>
</tr>
<tr>
<td>Proposed Method</td>
<td>Vertex3</td>
<td>260</td>
<td>Combinational</td>
</tr>
</tbody>
</table>

### 6.8 SUMMARY

An effective and more powerful enhanced DPA method to protect the secret key from an AES hardware implementation is presented. The implemented combinational circuit design for AES is highly inaccessible for power analysis attack. The implementation result shows reduced resource utilization of minimum 50% and the increased number of traces. The power traces are captured and the key extraction is also done. This proposed work presents the architecture of a fully pipelined AES encryption processor on a single chip FPGA. By using loop unrolling and inner-round-outer fully pipelined technique a maximum throughput is achieved.