ANALYSIS AND IMPLEMENTATION OF STATISTICAL CIPHER FEEDBACK MODE AND OPTIMIZED CIPHER FEEDBACK MODE

FANG YANG

ANALYSIS AND IMPLEMENTATION OF STATISTICAL CIPHER FEEDBACK MODE AND OPTIMIZED CIPHER FEEDBACK MODE

by

Fang Yang, B.Eng.

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

January 2004

St. John's Newfoundland Canada

Your file Votre référence
ISBN: 0-494-09932-1
Our file Notre référence
ISBN: 0-494-09932-1

Dedication

To My Dearest Father, Mother, Sister, and Cheng

Abstract

In this thesis, two recently proposed modes of operation for block ciphers, referred to as statistical cipher feedback (SCFB) mode and optimized cipher feedback (OCFB) mode, are investigated. Both cipher modes can achieve the capability of self-synchronization to recover from bit slips or insertions in the communication channel automatically. Compared to CFB mode, both cipher modes can obtain higher efficiency with modest buffer size and reasonable latency. Hence, both modes can be applied to high-speed digital hardware implementation and they have been identified as being suitable for physical layer security for applications such as SONET/SDH.

In this thesis, both modes are implemented in software and hardware. In particular, the hardware structure and method of hardware implementation are investigated. Parallel and serial transfers are applied to the hardware implementation of SCFB mode and OCFB mode, respectively. Very High Speed Integrated Circuit Description Language (VHDL) and LSI design with 0.18 CMOS technology supported by Canadian Microelectronics Corporations (CMC) are used in the process of hardware implementation. The hardware structures of both modes are synthesized by using Synopsis tools provided by CMC as well.

In addition, the performances of both modes are analyzed with respect to characteristics such as the theoretical efficiency, synchronization recovery delay, and the error propagation factor. Furthermore, the relationship between efficiency, probability of

queue overflow, and buffer size is investigated. It is definite that both of modes can obtain higher efficiency than cipher feedback mode. OCFB mode performs marginally better with respect to error propagation and synchronization recovery delay in some circumstances. SCFB mode is able to achieve higher efficiency with a given buffer size and probability of buffer overflow in an efficient hardware implementation. Similarly, for a given efficiency and buffer size, SCFB mode has a lower probability of buffer overflow than OCFB mode. In fact, while for SCFB it is possible to guarantee no overflow with 50% efficiency and a buffer size equal to the block size, it is not possible to guarantee no overflow for efficiencies that are much less than 50% for OCFB mode. These results imply that SCFB is a mode more suitable for high speed physical layer security than OCFB mode.

Acknowledgement

I sincerely thank my supervisor Dr. Howard Heys for his guidance and support throughout my master's study. Numerous meetings and discussion with you are the important step to complete my study. Your precious attitude to questions and patience impresses me very much.

This is also a chance to thank members of the Computer Engineering Research Laboratories (CERL) that provided a friendly and resourceful environment for my research. Thanks especially to *Reza Shahidi* who helped me to solve many computer problems I came across. As well, I would like to thank my friends, *Lu Xiao*, *Janaka Deepakumara*, *Zhiwei An*, *Padmini Vellore*, and many friends whose names are not mentioned.

I genuinely thank my father, *Hongkui Yang*, mother, *Guoyu Fu*, and sister, *Li Yang*, for their continuous encouragement and support. In these two years of my Master's study, my dearest sister had gotten cancer and did not have a chance to say goodbye to me in order to let me concentrate on my studies. My parents had endured anguish on the loss of my sister and kept it secret from me for one year. I can't imagine I could finish the research if I had known this sad news at the beginning. I wish my sister could know I so love her and miss her.

I also have to honestly thank my dearest husband, *Cheng Li*, for all his love, care, encouragement, and suggestion. He is always my strongest supporter. When I feel tired, he gives his shoulder to let me have a rest. When I meet some problems, he is my best advisor. When I feel depressed, he encourages me and gives me strength. Words can't express my gratitude to your love. I am so rich with your care and love. I dedicate my thesis to you.

Table of Contents

List of Figures

List of Tables

Notation and List of Abbreviations

n	: The length of sync pattern
B	: The length of a block
k	: The length of data bit between the previous sync pattern and the next sync pattern in ciphertext
M	: The size of queue
d	: The number of data in the queue
R	: The rate of incoming data of plaintext queue
R'	: The rate of outgoing data of plaintext queue
Re	: The operating rate of the block cipher
T	: The collection time of one block of plaintext data
T'	: The leaving time of one block of plaintext data
Te	: The operating time of one block of data for the block cipher
m	: The number of data less than or equal to the length of a block
η	: The theoretical efficiency
$p(k)$	: The probability of k
$E(k)$	: The expectation of k
μ	: The expect synchronization cycle size
eff	: The practical efficiency
QoS	: Quality of Service
DES	: Data Encryption Standard
AES	: Advanced encryption Standard
CAD	: Computer Aided Design

ASIC	: Application-Specific Integrated Circuit
SCFB	: Statistical Cipher Feedback
OCFB	: Optimized Cipher Feedback
NIST	: National Institute of Standards and Technology
XOR	: Exclusive-or
IV	: Initialization Vector
ECB	: Electronic Code Book
CBC	: Cipher Block Chaining
CFB	: Cipher Feedback
OFB	: Output Feedback
CTR	: Counter mode
SV	: State Vector
PCBC	: Propagation Cipher Feedback
VLSI	: Very Large Scale Integration
DRC	: Design Rule Checking
HDL	: Hardware Description Language
VHSIC	: Very High Speed Integrated Circuit
VHDL	: VHSIC Hardware Description Language
IC	: Integrated Circuit
CMC	: Canadian Microelectronics Corporation
SPLD	: Simple Programmable Logic Devices
CPLD	: Complex Programmable Logic Devices
FPGA	: Field Programmable Gate Arrays
FPIC	: Field Programmable InterConnect
DFT	: Design For Testability

PDP	: Physical Design Planner
LVS	: Layout-Versus-Schematic
PQ	: Plaintext queue
CQ	: Ciphertext queue
IVQ	: IV queue
SRD	: Synchronization Recovery Delay
EPF	: Error Propagation Factor

Chapter 1
Introduction

Information security is an old science which can be traced back several centuries. The sender tries to hide information in order that it can reach the receiver safely without being recognized by the enemy. In many ways, cryptography can be seen as a war without the smoke of gunpowder.

The main task of information security is to ensure the security of information during the process of transmission. Three key criteria mentioned in information security are confidentiality, integrity, and authentication. The confidentiality of the information means that only authorized access is allowed and unauthorized access is prevented. It guarantees that the information is hidden from the unauthorized users. The integrity of information ensures that the information transferred is original and not modified. The authentication of information ensures that the information is not processed during the transmission and both of the transmitter and the receiver are correct. It prevents the storage and the processing of information and a third party masquerading as one of the two parties.

The use of cryptographic systems offers the highest level of security together with maximum flexibility. A cryptographic system includes an encryption system and a decryption system. The encryption system utilizes algorithms and keys to convert the original meaningful data into the nonsense data. If the modified information is obtained

or accessed by an unauthorized user, it cannot be figured out without the knowledge of algorithms and keys. In reality, the algorithms are usually published, but the keys are kept secret. The security of keys decides the security of information. Hence, it is no doubt that a cryptographic system can offer the information security by correct management and implementation.

1.1 Motivation

Today, the development and application of high quality and high-speed networks makes bandwidth capability and data confidentiality more and more important. Suitable modes of operation not only protect the data but also have the ability to maximize the use of network bandwidth. This thesis will focus on the studies of two recently proposed modes of operation, referred to as statistical cipher feedback (SCFB) mode and optimized cipher feedback (OCFB) mode, which can not only obtain the ability of self-synchronization, but also have high efficiency compared to cipher feedback (CFB) mode. The ability to implement these modes for high-speed networks is also investigated.

1.2 Objective of the Thesis

The objective of the thesis firstly focuses on the hardware implementation of SCFB mode and OCFB mode. Hardware structure and the implementation method are discussed. The relative hardware characteristics with respect to the buffer size, probability of queue overflow, and the implementation efficiency are investigated.

The second objective of the thesis is to performance analysis on theoretical efficiency, error propagation delay, synchronization recovery delay and the relationships between buffer size, efficiency, and buffer overflow.

By the analysis and comparison of hardware and performance, the conclusion of which mode is more suitable for high speed networks can be drawn.

1.3 Thesis Outline

This thesis includes seven chapters. Chapter 1 is the introduction part. Chapter 2 provides the background related to the research. In this chapter, conventional modes of operation for the Data Encryption Standard (DES) [1] are discussed and the performances are analyzed in detail. Advanced Encryption Standard (AES) [2] algorithm is also provided as it is currently the most important block cipher.

Chapter 3 explains the environment for the design and the use of Computer Aided (CAD) tools used for hardware implementation in detail. This chapter also describes the processes of how to make a chip or board from an idea or an algorithm.

Chapter 4 introduces the algorithm of SCFB mode and considers the implementation of SCFB mode in hardware. The hardware characteristics with respect to the requirement on buffer size, the complexity, and timing analysis are discussed.

Chapter 5 introduces the algorithm of OCFB mode and considers the implementation of OCFB mode in hardware. Again, the hardware characteristics with

respect to the requirement on buffer size, the complexity, and timing analysis are discussed.

Chapter 6 analyzes the performance of SCFB mode and OCFB mode with respect to theoretical efficiency, synchronization recover delay, error propagation factor, and the relationship among buffer size, full-queue efficiency, and probability of overflow. Comparisons of these properties between SCFB mode and OCFB mode are then provided.

Finally chapter 7 will draw a conclusion for this thesis and suggest some future work.

Chapter 2
Background of Research

This chapter introduces the background material for the research and provides a view of previous work. In general, an operational mode is needed to realize an encryption/decryption system. The mode of operation which is selected has a great influence on the security and the efficiency of system implementation. Therefore, it is significant to study the modes of operation.

Besides the introduction of the modes of operation, the Rijndael algorithm which was announced as the Advanced Encryption Standard (AES) by the National Institute of Standards and Technology (NIST) recently is explained in this chapter.

2.1 Symmetric Encryption Systems

Conventional encryption, also referred to as symmetric encryption, is an encryption technology which uses the same key at the transmitter and receiver to encrypt and decrypt the message. Figure 2.1 illustrates the conventional encryption process [3]. The encryption system converts the original meaningful data, referred to as plaintext, into the nonsense data, referred to as ciphertext. The process of conversion combines the secret key shared by the transmitter and the receiver with a certain algorithm to produce the ciphertext. The secret key is independent of the plaintext. The ciphertext is sent into

the communication channel and collected by the receiver. The receiver then uses a decryption algorithm with the same key to decrypt the ciphertext to recover the plaintext.

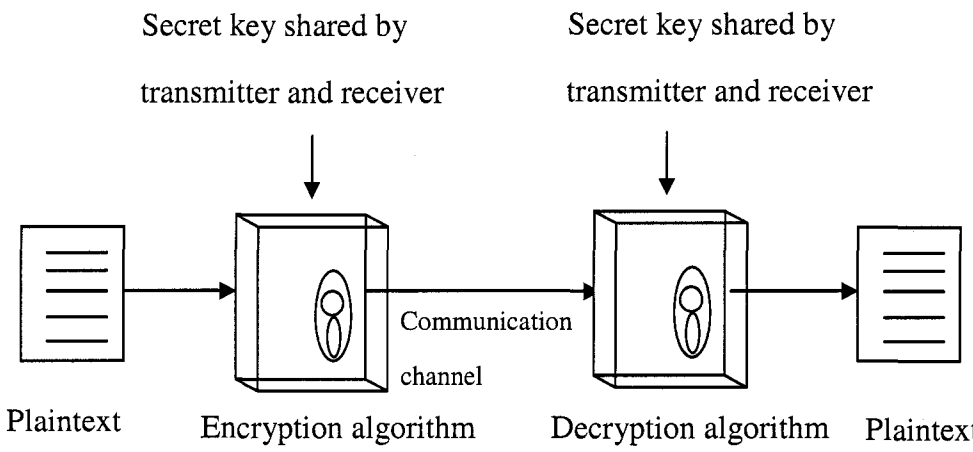


Figure 2.1 Conventional encryption models

2.2 Stream Ciphers and Block Ciphers

Block ciphers and stream ciphers are two important classes of encryption algorithms. A block cipher is a scheme which encrypts a fixed length of plaintext as a whole to produce the same length of ciphertext. The same plaintext produces the same output if the same key is provided. The benefit of the usage of fixed length of the block is its ease of implementation in software. Furthermore, it enables the incorporation of the encryption scheme into existing protocols or hardware components. A bit error in a block will cause a whole block error in the recovered plaintext.

In contrast to the block cipher, a stream cipher [4] is an algorithm in which plaintext is encrypted bit-by-bit or symbol-by-symbol to produce the corresponding ciphertext. A stream cipher usually generates a pseudo random keystream to exclusive-or

(XOR) with plaintext to produce ciphertext. The output of a stream cipher is varied relative to the plaintext depending on the pseudo random keystream during the encryption process. Stream ciphers have relatively simple circuits and faster encryption speed in hardware compared with the block ciphers. Therefore, a stream cipher is suitable for high-speed networks or the physical layer in a communication channel. In addition, stream ciphers can have the significant property of no error propagation. A single bit of ciphertext error results in a single bit of plaintext error. This property makes stream cipher suitable for systems with high error probabilities in transmission. If a ciphertext bit is lost in transmission, a stream cipher will cause complete nonsense data for the rest of the recovered plaintext unless special measures are taken. Hence, stream ciphers need the ability of resynchronization through either a special signalling channel or the method of self-synchronization. This research will discuss the modes of operation which configure block ciphers as stream ciphers capable of self-synchronization.

Bit slips are defined as the loss of a bit or bits in the process of data transmission. It is possible for the data transmitted to pass by several network switches and then arrive at the destination. Clock differences between network nodes, could cause bit slips or insertions.The ability to recover from bit slips and insertions is inevitable to consider when the modes of operation are discussed.

There is a class of stream cipher, referred to as self-synchronizing stream ciphers, which can recover from bit slips or errors automatically. It looks for a sync pattern in the ciphertext to extract an Initialization Vector (IV) to synchronize the encryption and decryption system. This works because the ciphertext is shared by the encryption and

decryption system. Self-synchronizing makes the cryptographic system more efficiently use bandwidth than the stream ciphers which require an extra signalling channel to transfer IV to synchronize the encryption and decryption system periodically because it does not need additional bandwidth for synchronization purposes.

Interestingly, there are several modes of operation that can configure a block cipher as a stream cipher to produce a pseudo random keystream and attain the ability of self-synchronization. Before we discuss self-synchronization explicitly, let us introduce the conventional modes of operation.

2.3 Conventional Modes of Operation

There are four operational modes for block ciphers that were published in December 1980 [4]. They are listed as follows:

1. Electronic codebook mode (ECB)
2. Cipher block chaining mode (CBC)
3. Cipher feedback mode (CFB)
4. Output feedback mode (OFB)

Electronic Code Book (ECB) [3] is a mode in which each block of plaintext produces a corresponding ciphertext value according to the key. In other words a plaintext always has the same ciphertext given the same key. When data is applied to ECB mode, data is separated into blocks and then each block is encrypted independently. Figure 2.2 illustrates ECB mode [3]. ECB mode is not fitted for a system with small block sizes because the repetition possibility of the block becomes high and that will cause a decreasing of the security. That situation may be improved by the addition of

random pad bits in the block. Another method to increase the security is to enlarge the block size. A large size block has the ability to prevent a codebook attack since it contains enough unique entropy. However, a bit error will cause a whole block of data errors.

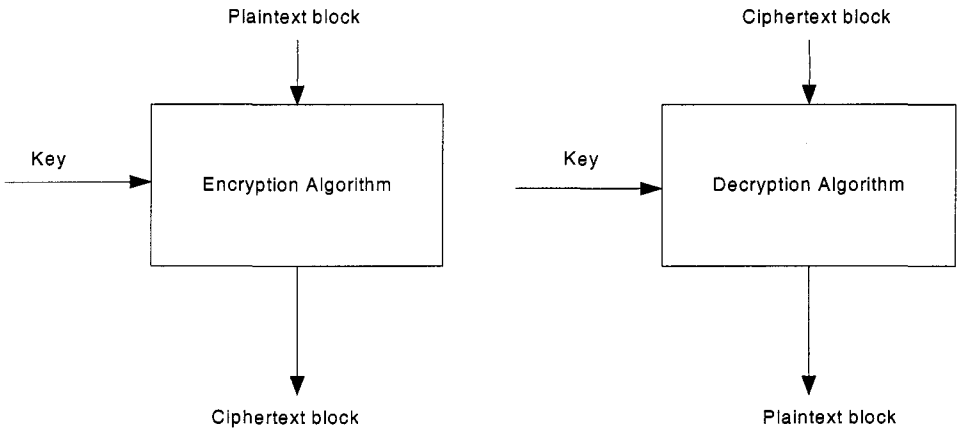


Figure 2.2 ECB mode

Cipher Block Chaining (CBC) [3] is the mode in which each plaintext block XORs with the previous ciphertext block, and then is encrypted with the key to produce the next ciphertext. The first ciphertext block is provided by an IV. Figure 2.3 illustrates CBC mode. In the figure, P_J represents the current plaintext with bits, P represents the recovered plaintext, E represents encryption algorithm, E_{-1} represents decryption algorithm, B represents block size, and C_0 represents the first ciphertext which is usually given by IV. C_{J-1} represents the previous ciphertext, and C_J represents the current ciphertext. In the figure, $\oplus$ represents the bitwise XOR of a block. The chaining structure makes the current ciphertext block entirely dependent on the previous ciphertext block. The same ciphertext block for a given plaintext can be obtained only if the same key and

the previous ciphertext block is the same. For CBC mode, a bit slip will cause the whole block and the remaining blocks to be random with respect to the plaintext.

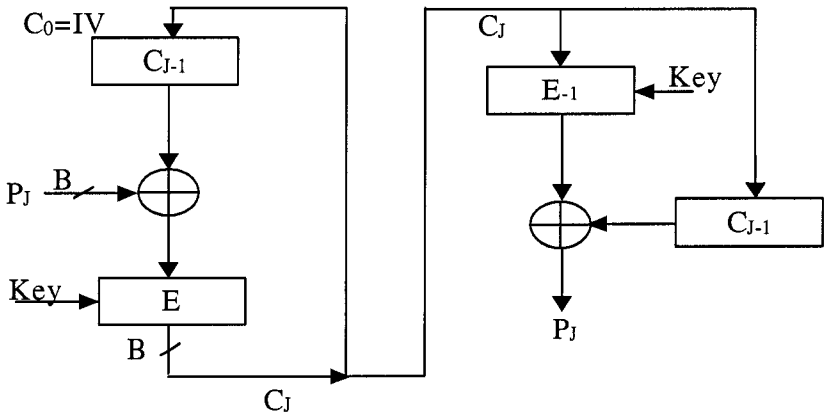


Figure 2.3 CBC mode

Cipher Feedback (CFB) [3] mode utilizes a pseudo-random keystream which is generated by a block cipher to encrypt plaintext. Because the encryption of plaintext is bit by bit, CFB mode can fall into the class of stream ciphers. The significant characteristics of CFB mode is that it feedbacks the ciphertext into a shift register at the input of the block cipher to produce the next key stream block. CFB mode is illustrated in Figure 2.4. In this figure, m and B are the feedback and block size and m could be B or less than B, and E represents the block cipher with a block size of B.

Similar to CBC mode, an IV as the initial input of the block cipher is provided to CFB mode to guarantee the same start on both the transmitter and the receiver. The value of m greatly influences the properties of CFB mode. When $m = 1$, CFB mode has the ability to recover from bit slips or insertions. When $m > 1$ and a single bit slip occurs, the input to the block cipher at the receiver will become misaligned and resynchronization

will not occur. CFB mode can be categorized as a self-synchronizing cipher when $m = 1$. However, it is very inefficient from the view of implementation [5]. How to increase the efficiency becomes an interesting topic. One single bit error in the communication channel will cause the recovered plaintext bit to be in error and the next whole block of B recovered plaintext bits to be corrupted.

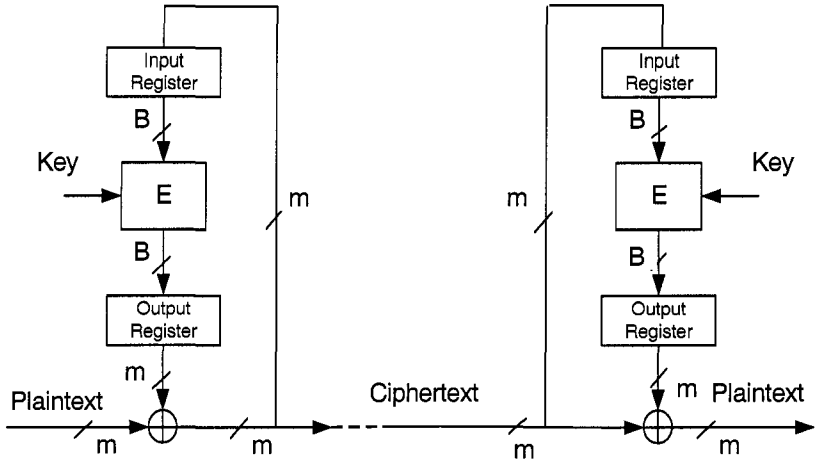


Figure 2.4 CFB mode

Output Feedback (OFB) mode [3] can be used to construct a stream cipher by making use of a block cipher as a pseudo-random generator. As with CFB mode, it uses an IV as the input of block cipher initially. It then takes the previous output of the block cipher (not the previous ciphertext) as the next input to the block cipher to produce the next key stream block. OFB mode is illustrated in Figure 2.5.

Of all operational modes, OFB mode offers minimal error propagation. A bit error will cause only one bit error because the generation of the key only has a relationship with the output of the block cipher rather than the ciphertext. It can be implemented with

high data throughput as well by performing the XOR of plaintext with keystream in blocks of B bits. However, it does not have the ability to resynchronize. To ensure that the communication system is working properly, OFB mode needs an extra signalling channel to periodically transfer an IV from the transmitter to the receiver to recover from any synchronization loss that may occur due to bit slips.

OFB mode has a subtle security problem. Because the output of the block cipher is fed back as the input of the block cipher and the fixed length of the block size is used, it is possible to cause the repetitive usage of the keystream. That means the cryptanalyst can figure all subsequent messages transferred from the repeated parts, if the system is accessed by cryptanalyst and the cryptanalyst finds the repetition cycle of data. Thus, the security of OFB mode is decreased.

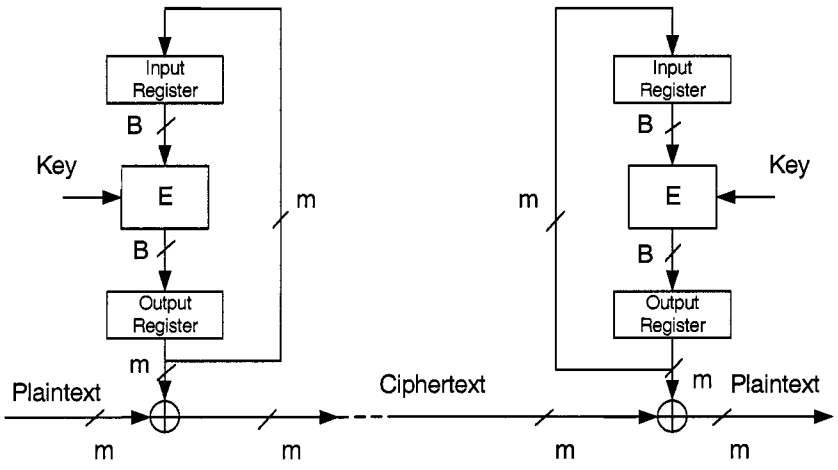


Figure 2.5 OFB mode

2.4 Other Modes of Operation

The conventional modes of operation discussed above were recommended to be used with DES [1]. With concerns of the security of DES, Triple-DES and AES have taken its place. This brings some new modes of operation and applications.

Counter mode [6] also turns a block cipher into a stream cipher. It generates the next keystream block by encrypting successive values of a "counter". The counter can be any simple function, which produces a sequence that is guaranteed not to repeat for a long time, although an actual counter is the simplest and most popular. In counter mode, the state vector is simply a number the same size as the block of the block cipher. To encrypt any block, the number is incremented, then the incremented number is encrypted, the output of that encryption is XORed to the plaintext, and the result of the XOR is the ciphertext. The problem of the OFB mode having cycles of unpredictable (and potentially short) periods in some cases is solved. Since the counter doesn't cycle until it was stepped through all 2^B numbers (assuming block size B). Obviously, counter mode should not be used with block ciphers whose blocks are so short that there is a risk of running a full cycle on one key. Hence, counter mode has to be applied on 64-bits or larger blocks. Counter mode has similar error propagation characteristics to OFB.

Many other modes of operation besides counter mode are modified from the conventional modes of operation and counter mode, such as cipher-chain-cipher mode, propagation cipher feedback (PCBC) mode, CFB64 mode, etc [6].

2.5 Advanced Encryption Standard (AES)

AES [2] was developed by NIST to replace DES and protect sensitive government information well into the twenty-first century. Among five finalists, the Rijndael algorithm won out and has become the proposed AES algorithm.

Before describing the cipher Rijndael algorithm, there are several parameters that need to be explained. State describes the intermediate data and is expressed as a byte array which has four rows and *Nb* columns. *Nb* is the block size divided by 32. The cipher key is expanded with the key schedule to generate round keys, $w[i,j]$. The number of rounds, *Nr,* is decided by the length of the cipher key.

The cipher Rijndael consists of:

- An initial round key addition
- *Nr*-1 intermediate rounds
- A final round.

Initially a round key is added in to enhance the system security. In the intermediate *Nr*-1 rounds, each round is composed of four different transformations to realize confusion, diffusion, and key mixing [3]. The final round is slightly different from the previous *Nr*-1 rounds.

Figure 2.6 [3] illustrates the structure of AES using the 128-bit block as an example. During the encryption and decryption, the State array is initialized as the plaintext and modified at each stage of the transformations. The 128-bit cipher key is arranged into the matrix of bytes to be expanded into 44 words of key schedule for 10 rounds and each round takes 128 bits as the round key from the key expansion. In the

diagram, the inverse substitute bytes, the inverse shift rows, and the inverse mix columns are the inverse of the corresponding transformations. The details on the inverse transformations and other length of the cipher key are described in [2] and [3].

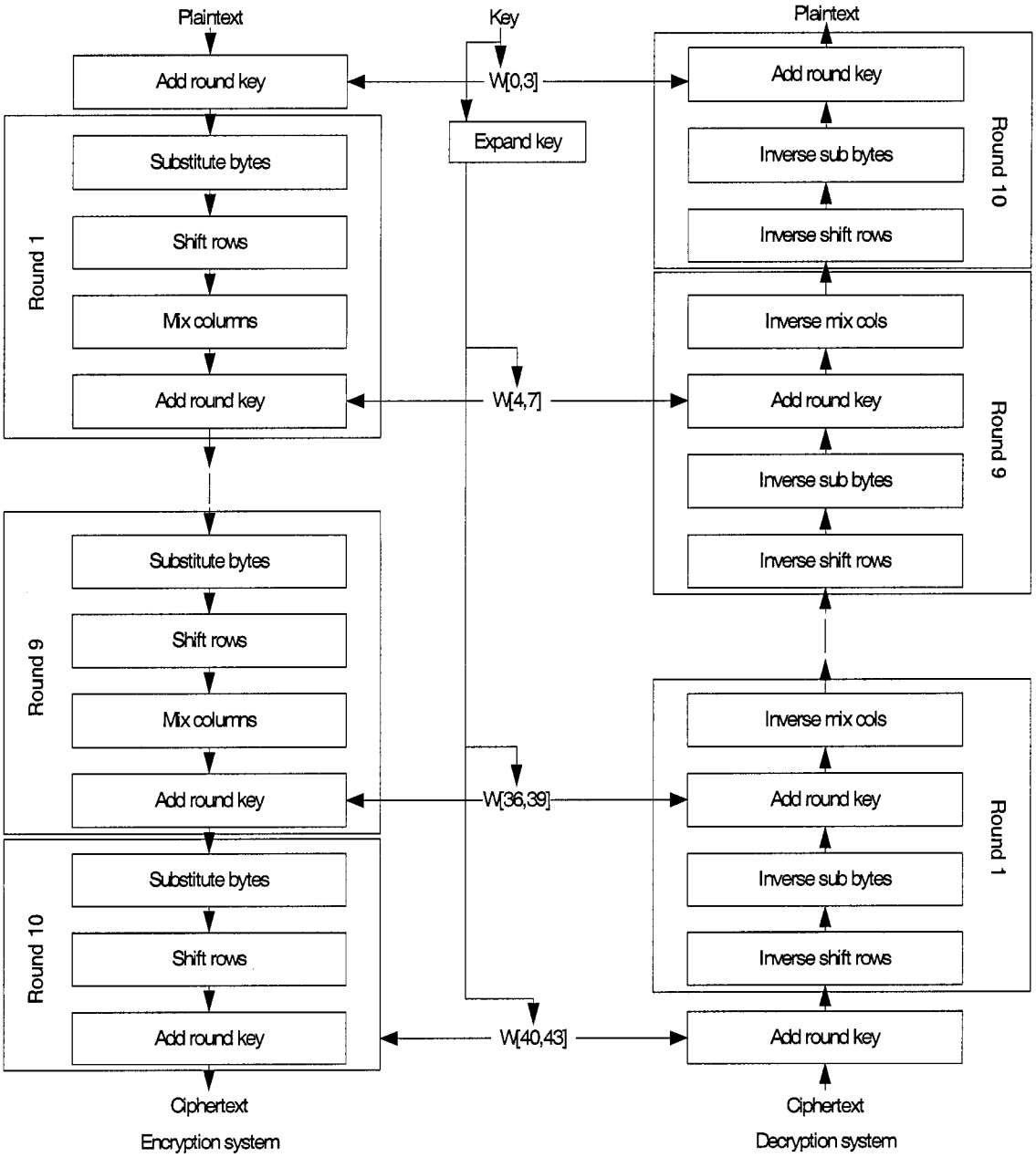


Figure 2.6 AES encryption and decryption

The functions of four transformations on the Sate are briefly explained here.

- Substitute bytes: This transformation uses a nonlinear substitution table called an S-box to substitute each State byte-by-byte.
- Shift rows: This transformation is a simple permutation according to the rules that the first row does not shift, the second row performs circular left shift by 1-byte, the third row performs circular left shift by 2-bytes, and the fourth row performs circular left shift by 3-byte.
- Mix columns: This is a linear transformation on each column of State over GF (2^8) to generate new columns.
- Add round key: This transformation XORs the current block with the round key.

Chapter 3
Design and Implementation Environment

An algorithm or an abstract idea can be implemented by software and hardware through a design process, and made into a system or a chip. A design has to depend on current particular technology, especially for hardware design. With systems or chips becoming more and more complex and the sizes of systems becoming smaller and smaller, a hardware technology, referred to as Very Large Scale Integration (VLSI), has become very popular. Computer Aided Design (CAD) tools are utilized in the design and implementation. This chapter will give some basic background to these issues.

3.1 Software vs. Hardware Implementation

Software implementation is prevalently used to check the correctness and the feasibility of a system or an algorithm due to its flexibility, ease of use, relatively low cost, and relatively short implementation time compared to hardware implementation.

However, hardware implementation plays an important role in the system implementation due to its concurrent characteristics and the capability of satisfying speed that system requires. In hardware each part of a system can work concurrently such that the system efficiency can be improved. Hence, hardware implementation becomes more attractive for high speed applications such as broadband communication networks.

3.2 Design Methodology

Unlike software, hardware implementation is a complicated process. Figure 3.1 illustrates the design process of full ASIC from an idea to a chip or a board [8]. There are seven steps from an initial design idea to the final hardware implementation. Before it is passed to the next step, the result is checked to guarantee correctness of the transformation. In the end, a stream file which describes mask layer information for a circuit is generated for chip fabrication after Design Rule Checking (DRC) is completed.

During the design process a Hardware Description Language (HDL) is used widely. Among many HDLs, the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) is very popular in the research and industry domains. In our work we have made use of design methodology which is prevalently used in the process of a system design focused on the digital Integrated Circuit (IC) design flow recommended by Canadian Microelectronics Corporation (CMC) [8].

Instead of trying to implement the design of a large system all at once, a divide-and-conquer strategy is taken in a top-down design process. Top-down design is referred to as recursive partitioning of a system into its subcomponents until all subcomponents become manageable design parts. By "manageable design parts" is meant that the components designed can be found in a library provided. Figure 3.2 outlines the recursive partitioning in a top-down design process. In the figure, the shadowed sub-components represent the manageable parts by hardware mapping in a library [8].

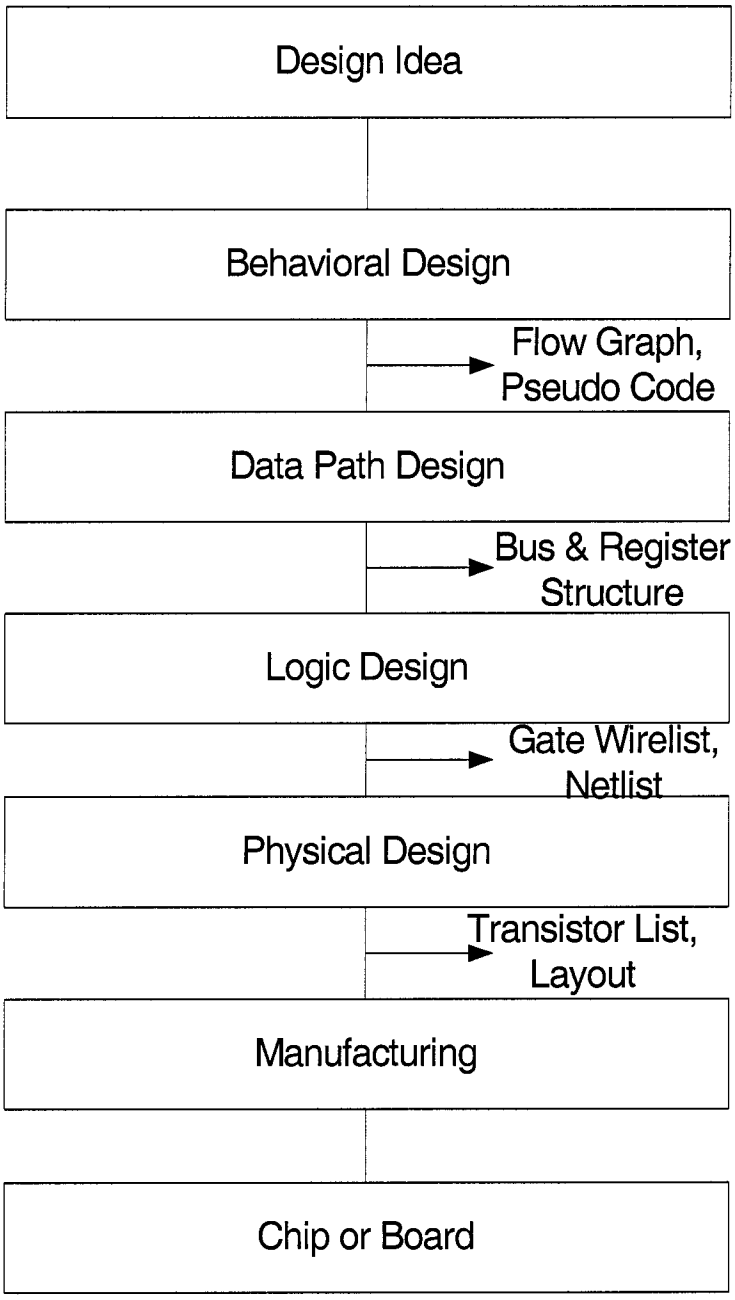


Figure 3.1 A digital system design process

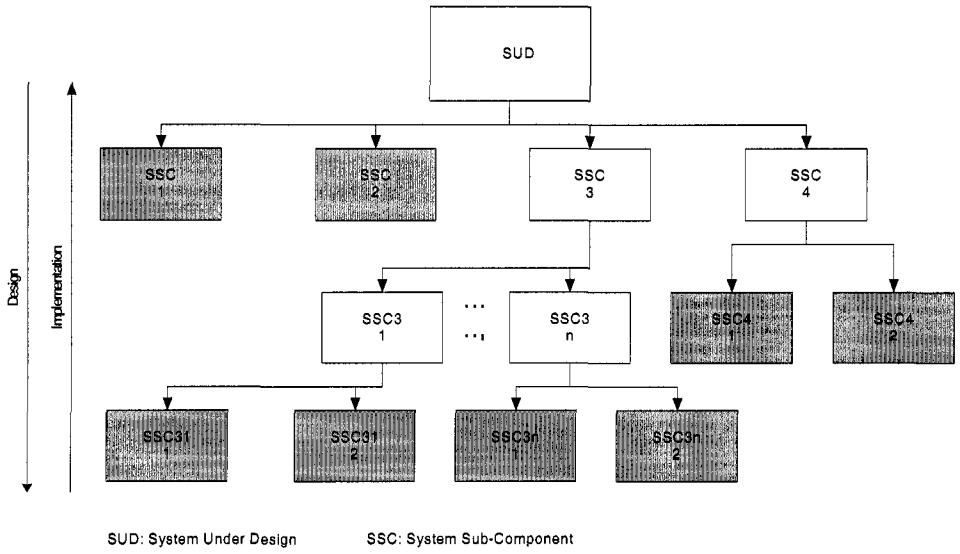


Figure 3.2 Top-down design and bottom-up implementation

When the top-down design process is completed, a partition tree is available. Then, the bottom-up implementation process begins. During this process, hardware components corresponding to the leaves of the tree are recursively bound until the system is completed.

3.3 Design Flow, Functional Test and Verification

Figure 3.3 shows the digital system design flow using the Deep-Sub-Micron (DSM) technology recommended by CMC. From the figure, it can be seen that the design flow is divided into two parts. The first four steps, which are referred to as front-end design part, use the VHDL language and Synopsys CAD tools and the remaining five steps, which compose the back-end design part, use Verilog and Cadence CAD tools. In the front-end design, a design idea is converted to a gate-level netlist. The gate-level netlist is then passed to the back-end design part for placing and routing.

Step one uses the VHDL language to transform the design idea to Register Transfer Level (RTL) codes and verify the functionality of RTL code. The top-down design and bottom-up implementation are the main strategies in this process.

Step two accepts RTL codes and synthesizes them. The synthesis turns VHDL code automatically into gate-level code depending on the current libraries. This process may bring in many unnecessary circuits because of the limitations of the CAD tools [9]. The result could be unoptimizable. The situation may get worse when synthesizing a large design. However, for small and simple designs, it is efficient and trustworthy to use CAD tools for an excellent job. It is recommended in the Synopsys documentation that large designs not be imported directly to the synthesis tool. A hierarchical bottom-up approach should be used instead. Importing large designs leads to crashing the synthesis tool and in some cases may result in an unoptimized design [8]. In this research each module of the design is designed, synthesized and tested separately, based on this bottom up implementation approach.

In the synthesis step, each component is analyzed and elaborated using the Synopsys Design Analyzer tool. Higher-level components are then built up when the synthesis results of all the bottom components are saved in the database or the work library. After the complete design is successfully imported, it is constrained based on the designer's performance objectives. In most cases, the constraints include input/output (I/O) pads specification, scan style definition, output load definition, and clock definition. If all pre-set constraints are met, the constrained design is then synthesized into gates.

Otherwise, the RTL code needs to be modified, simulated and then synthesized until all constraint requirements are met.

Step three is the scan insertion for the design test by the standard scan-based Design for Testability (DFT) techniques.

Step four is the gate-level simulation. This simulation is different from the RTL simulation in step one which ensures the design is functionally correct. Timing is not considered at that time because hardware timing information, which is tightly associated with the targeted technology and is defined in the library, is not available to the design yet. Hence, this step is the gate-level simulation with timing information considered for the design.

Floor planning, which is step five, is to create a floor plan for the design.

Step six, placement, is to use forward-annotated timing information from Synopsys tools to perform core cells placement.

Clock tree generation is step seven to add clock buffer cells and nets to create a balanced clock tree .

Step eight is routing and timing verification, which verify the gate-level circuit generated from the step previous.

The last step is stream file. This step verifies the placed and routed design and fixes minor violations. The stream output is used for chip fabrication.

This concludes the complete digital system design process based on CMC recommended design flow.

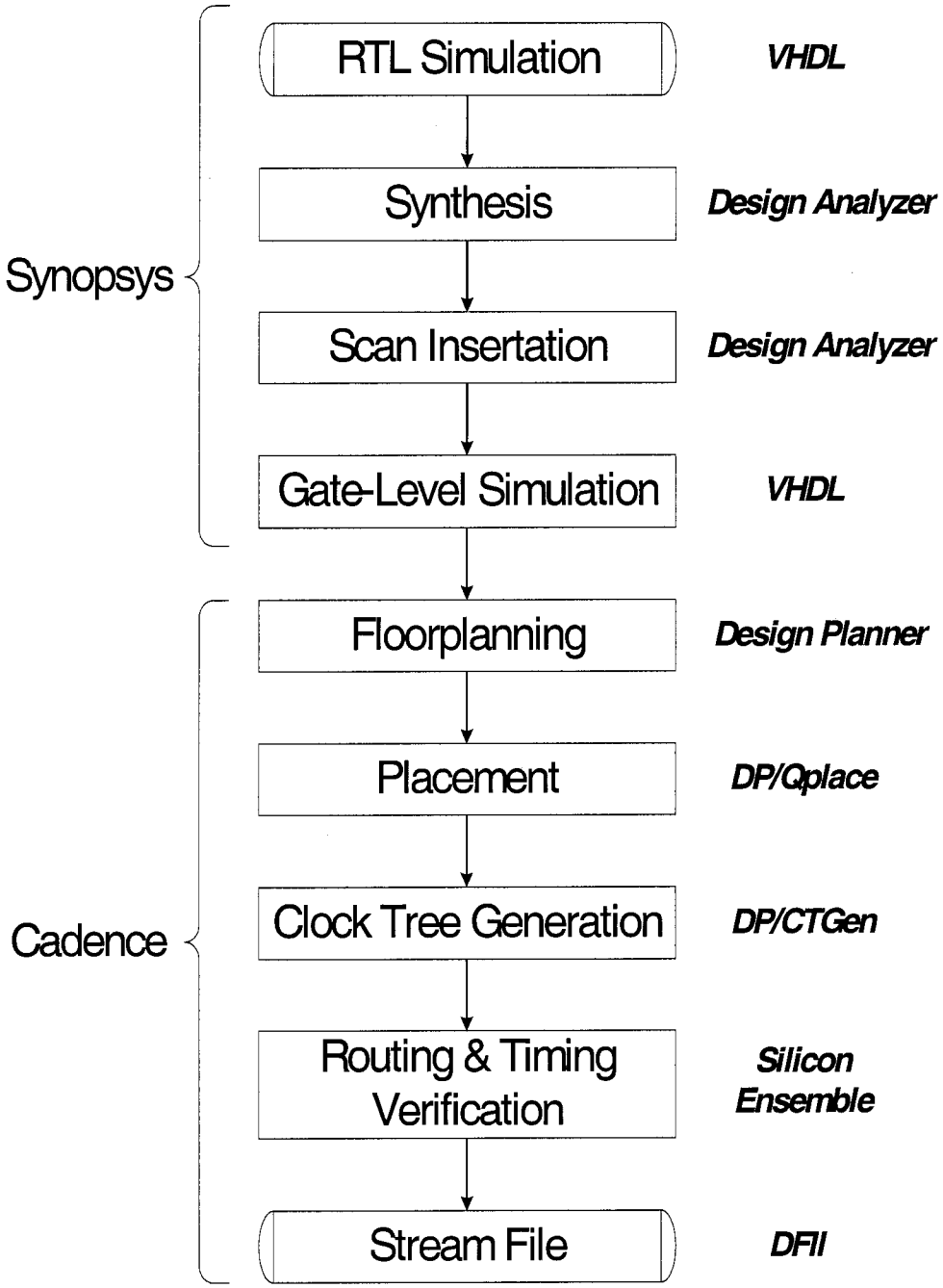


Figure 3.3 Design flow recommended by CMC

Chapter 4
Statistical Cipher Feedback (SCFB) Mode

In this chapter, statistical cipher feedback (SCFB) mode [10] is investigated. The algorithm of the SCFB mode was first described in [10] and the name was given in [5] because the cipher feedback is statistical which depends on the frequency of recognizing sync pattern. SCFB mode is a form of stream cipher which can utilize a block cipher to produce a keystream to XOR with plaintext data. Compared with conventional block cipher modes, SCFB mode can achieve self-synchronization with high efficiency, reasonable latency and modest buffer sizes. Hence, SCFB mode has the capability to recover from bit slips in the communication channel.

Firstly, the working theory of SCFB mode is introduced. Then the top-down design and the bottom-up implementation are provided. The performance analysis of SCFB mode will be discussed in Chapter 6.

4.1 Introduction to SCFB Mode

In Chapter 2, it was shown that CFB mode with $m = 1$ is an inefficient mode with the property of self-synchronization. Hence, how to improve system efficiency and to keep the property of self-synchronization becomes a research direction. To save communications bandwidth, one way to control synchronizations of the encryption system and the decryption system is to check for a sync pattern in the ciphertext data

because the encryption system and the decryption system can obtain the same ciphertext. The sketch of SCFB mode is shown in Figure 4.1 where *E* represents the block cipher and the *input register* is used to store data as the input of the block cipher [10]. The *scan block* is used to scan ciphertext to find a sync pattern and collect the IV after a sync pattern is found. If the sync pattern is not recognized and the status of the system is not in the collection of new IV, the switch is put into the position *A* and SCFB mode can be thought of as OFB mode because its block cipher uses the previous output of the block cipher as the next input of block cipher to produce a block of keystream. If the sync pattern occurs, the switch is put into position *B* while the new IV is collected from ciphertext. After the collection of IV is completed, a new IV has been loaded into the *input register* to synchronize the system. During the collection of IV, SCFB mode is considered as CFB mode because its block cipher uses ciphertext as the next input to produce a block of keystream.

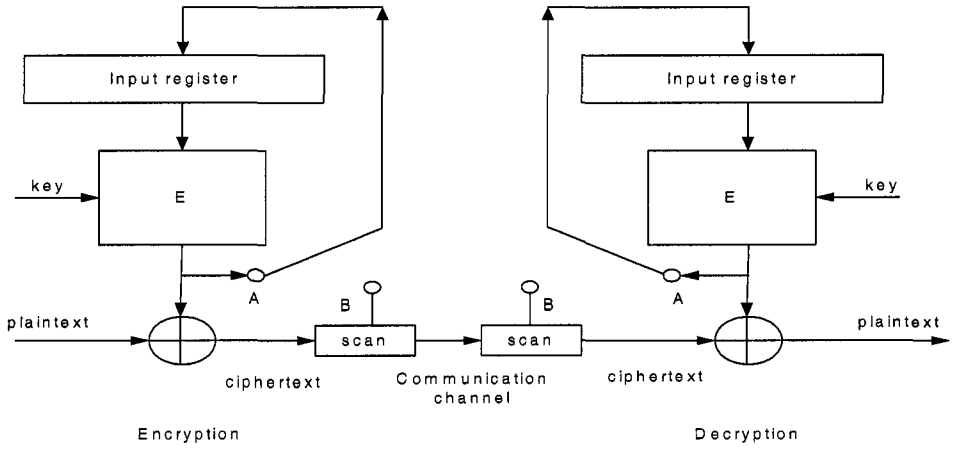


Figure 4.1 SCFB system

Hence, SCFB mode is a combination of CFB mode and OFB mode. SCFB mode conquers the deficiency of OFB mode by turning into CFB mode following the detecting of a sync pattern in the ciphertext to provide the capability of self-synchronization. As well, the efficiency of SCFB mode is improved significantly compared to the conventional CFB mode since SCFB mode works as OFB mode most of time. From this figure it can also be know that the decryption system has the same structure as the encryption system except the plaintext instead of the ciphertext is shifted into the *input register.*

4.2 Implementation of SCFB Mode

4.2.1 Software Implementation

To precisely describe the operation of SCFB mode, the flowchart of SCFB mode is shown in Figure 4.2. In the figure, $X_0 \ldots X_{B-1}$ and $Y_0 \ldots Y_{B-1}$ represent plaintext bits and ciphertext bits, respectively. Furthermore, $W_0 \ldots W_{n-1}$ represents an n-bit window that is used to compare with a sync pattern and $Q_0 \ldots Q_{n-1}$ represents the n-bit sync pattern. $E_k(\cdot)$ represents the block cipher with the key k and block size B. $Z_0 \ldots Z_{B-1}$ is a register to collect the B-bit IV. The flag, *loading_IV*, is used to indicate whether or not the collecting of IV is underway. The flag, *new_IV*, indicates whether or not the collection of the new IV is completed.

From the flow chart, it is clear that block cipher operation is triggered by either the case that the sync pattern is found or the case that the encryption of B bits of plaintext is completed. If the sync pattern is found, the system starts to collect the B-bit new IV to

save in the Z register and waives checking for the sync pattern until a new IV is ready. All plaintext bits following the new IV are encrypted as a new OFB keystream. If the sync pattern is not found and the encryption of a B-bit plaintext is finished, the system works in OFB mode to trigger $E_k(\cdot)$ to produce a new B-bit block of keystream and continues to check for the sync pattern in ciphertext.

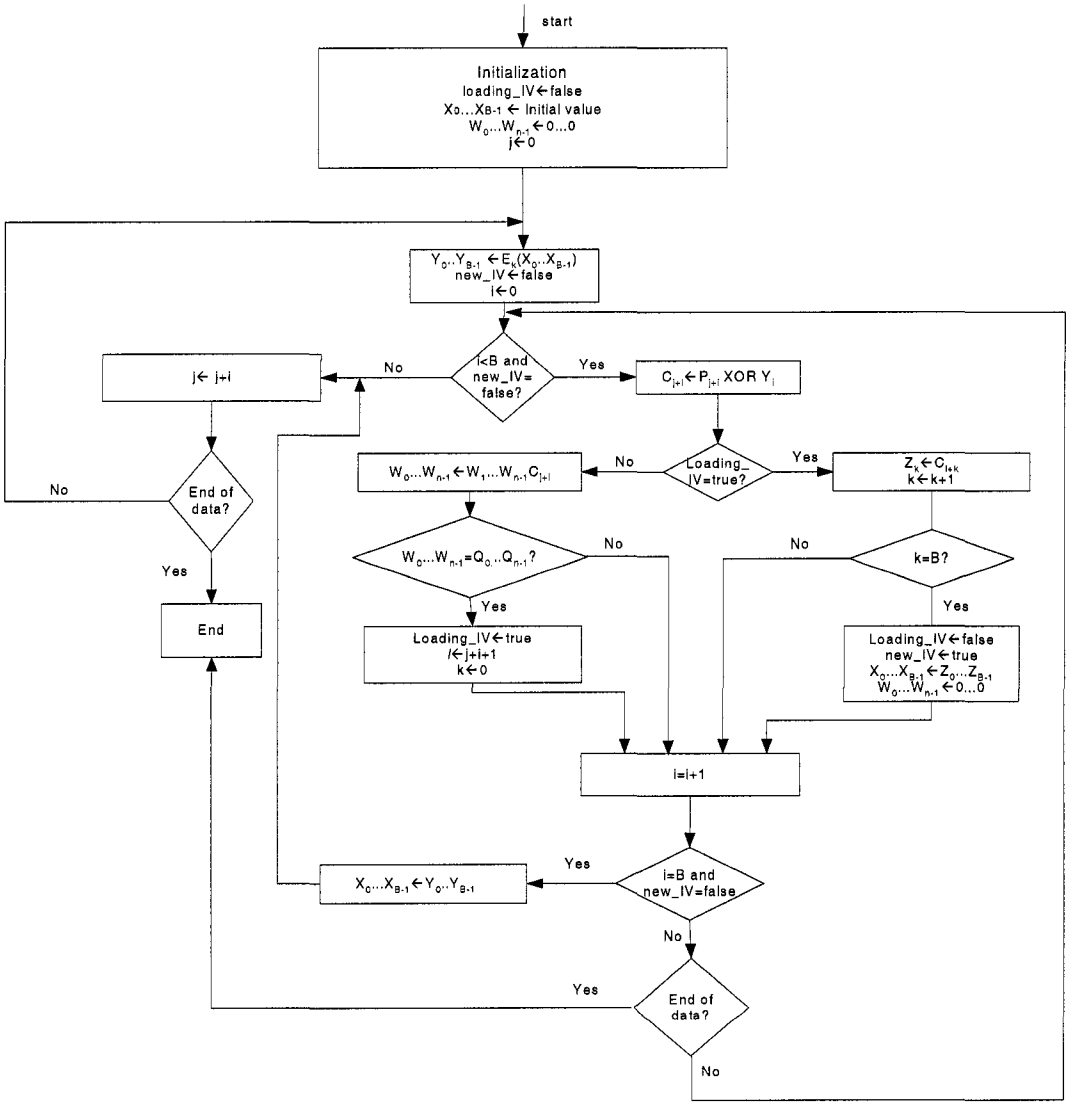


Figure 4.2 Flow chart of SCFB system

Although a description of software can clarify the algorithm of SCFB mode, the property of sequential logic gives a limitation on explaining and simulating how the system works efficiently. The system will become more efficient only if components in the system are used simultaneously and only a fewer components remain idle at any time. That is the fundamental reason why the hardware implementation is investigated in this thesis.

4.2.2 Hardware Implementation

4.2.2.1 Top-down Hardware Design of SCFB Mode

The completed system of SCFB mode consists of the encryption system and the decryption system as shown in Figure 4.3.

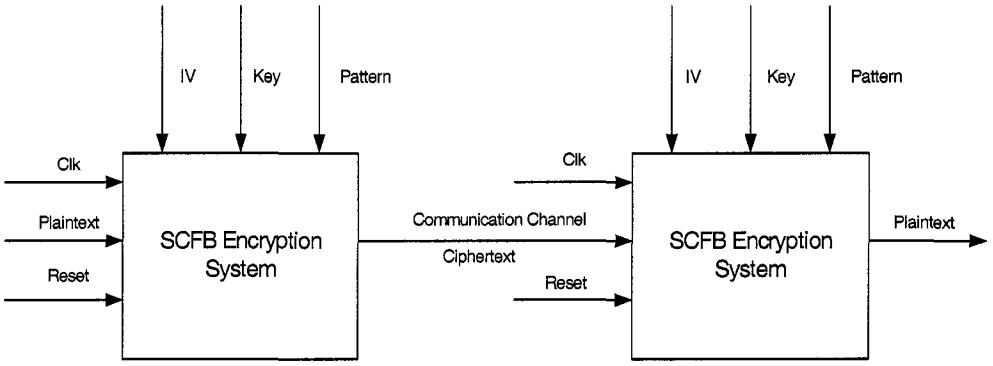


Figure 4.3 General diagram of SCFB system

In the figure, *IV*, *key* and *pattern* are known by both the encryption system and the decryption systems. The port *IV* provides an initial IV to the inputs of block cipher. The *key* port gives a key to the block cipher. The *pattern* port provides a sync pattern to the system. Plaintext data is encrypted by the encryption system and sent to the decryption

system through a communication channel. The decryption system will then decrypt the encrypted data to recover the plaintext data. The input and output ports of the encryption system are displayed in Figure 4.4.

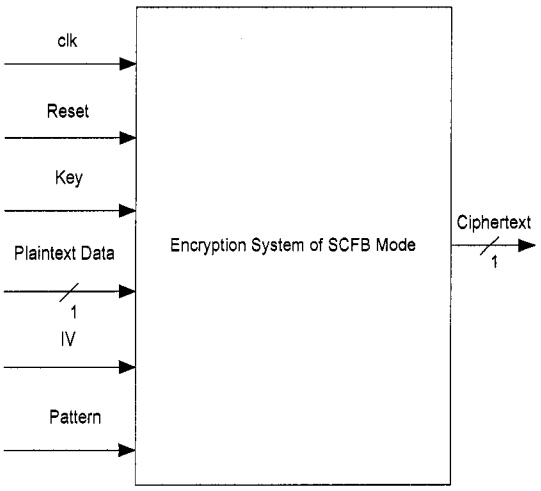


Figure 4.4 I/O ports of the encryption system

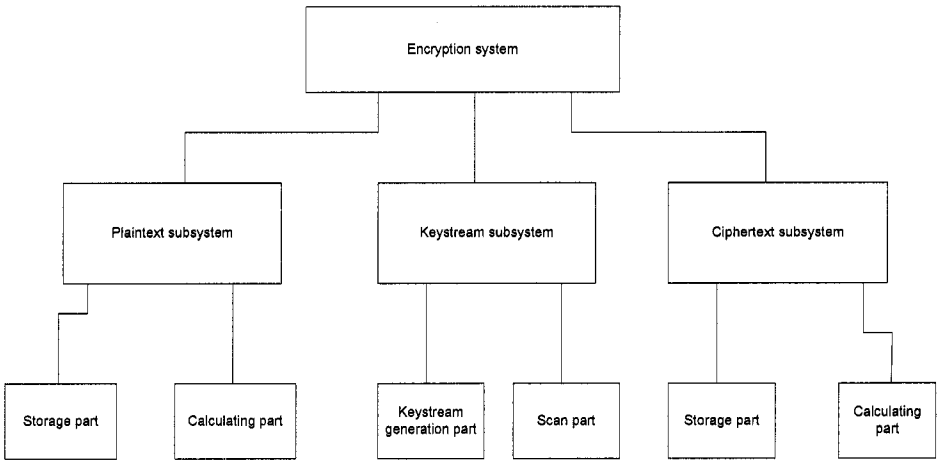


Figure 4.5 Block diagram of the encryption system

The encryption system can be divided into three subsystems according to different functions: plaintext subsystem, keystream subsystem and ciphertext subsystem as shown

in Figure 4.5. In practice, a plaintext buffer and a ciphertext buffer are required in order to provide the elasticity needed to ensure that the incoming and outgoing data speeds are the same even while the processing of data inside the system is not constant.

The plaintext subsystem is used to collect plaintext data and send data out after the collection of B bits of data is completed. Because the plaintext subsystem needs to store incoming data in a queue which is named plaintext queue (PQ), there is a storage part in the figure. The calculating part is used to calculate the queue position at which incoming data should be placed. The keystream subsystem is used to generate the keystream and scan the ciphertext for the sync pattern to synchronize the encryption and the decryption systems. Therefore, the keystream subsystem has two functions: the keystream generation part produces the keystream using a secure block cipher, and the scan part recognizes a sync pattern. If the sync pattern occurs, the scan part will collect a block of ciphertext as the new IV to send to the keystream generation part as the new IV. If the sync pattern does not occur, the keystream subsystem uses the previous output of the block cipher as the input to produce keystream and the scan part continues to scan for the sync pattern. The ciphertext subsystem stores ciphertext data and sends ciphertext data to communication channel. The ciphertext subsystem has the same queue structure as the plaintext subsystem and hence, requires storage (referred to as the ciphertext queue (CQ)) and the calculating part to manage the queue data positions.

4.2.2.2 Parallel Transfer vs. Serial Transfer

Parallel transfer and serial transfer are two methods to transfer data bits from the PQ to CQ. In parallel transfer the incoming plaintext data bits are not sent to XOR with

the keystream until there are enough data bits in the PQ and the block cipher finishes the production of a block of keystream. In the case which SCFB mode works as OFB mode and the sync pattern is not recognized, the PQ collects B data bits depending on the frequency of clock and then sends the whole block of data to XOR with B bits of keystream at a time. In the case that the sync pattern is found but the collection of the new IV is not completed, the PQ collects data bits until it has the number of bits needed by the new IV and then sends the exactly needed data out to XOR with a partial block of keystream to produce a partial block of ciphertext.

In contrast to parallel transfer, serial transfer sends plaintext data out bit by bit to XOR with keystream bits and the CQ receives the ciphertext data bit by bit. Serial transfer generally requires a simpler circuit than parallel transfer. However, unlike parallel transfer which has no clock limitation and can obtain high efficiency, serial transfer has clock limitation which constrains the system efficiency. In this thesis, to investigate the tradeoffs between these two methods, parallel transfer and serial transfer are applied to SCFB mode and OCFB mode (which will be discussed in Chapter 5), respectively.

4.2.2.3 Implementation Structure of an SCFB System

According to the discussion above, a general figure for the encryption system is drawn in Figure 4.6 [5]. While the plaintext data is being collected, a keystream block of B bits is generated by the block cipher. The input of the block cipher can be either the output of the block cipher or the IV from the ciphertext depending on whether n bit sync pattern is recognized. After a block of keystream is ready and the collection of B bits of

plaintext data is finished when the sync pattern is not found, the B bits of the keystream will be XORed with B bits of the plaintext data to produce the same length of ciphertext which is then stored into the CQ. When the sync pattern is recognized, the number of plaintext bits needed by the portion of new IV not in the same encrypted block as the sync pattern is collected to XOR with the keystream and then sent to CQ. The CQ will send data bit by bit into the communication channel at a constant rate. In the figure, the low case d refers to the number of bits transferred out of PQ which could be less than or equal to B bits. The queue which collects ciphertexts to compare with the n-bit sync pattern in the figure is named new IV queue (IVQ) and is used to provide new IV for the system. The block cipher described as E in this figure adopts the AES algorithm with the 128-bit block length which was implemented by NIST and the results are published in the implementation of SCFB mode.

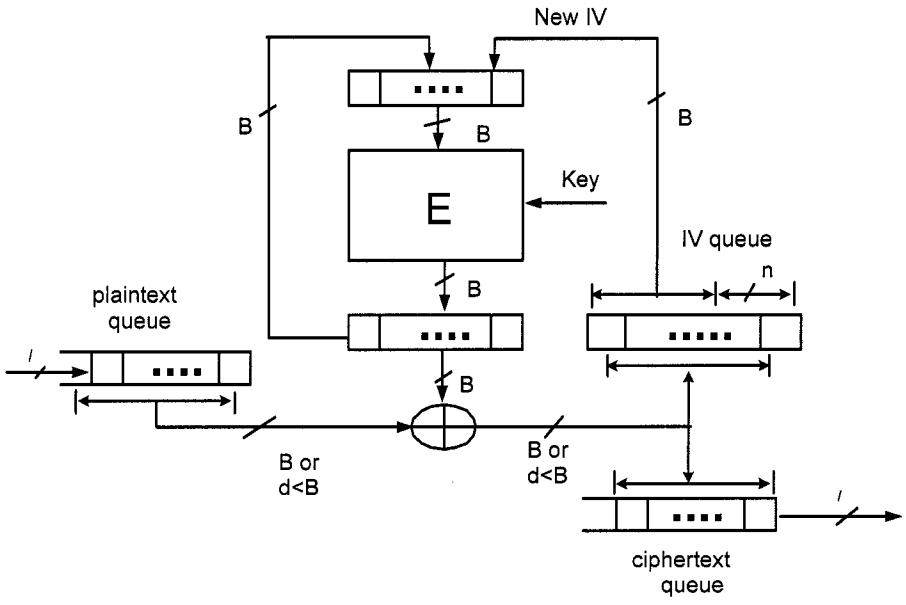


Figure 4.6 Structure of the encryption system

The decryption system has a structure similar to the encryption system except that the position of the checking the sync pattern occurs on the input (i.e., ciphertext) side rather than the output (i.e., recovered plaintext) side. The input switching of the block cipher from the output of the block cipher to the new IV is again dependent on the detection of the sync pattern.

4.2.2.4 Discussion on Queuing

Initially the PQ is empty and the CQ is full with arbitrary data. When the PQ is collecting data at a fixed rate, the CQ is sending data out bit by bit at the same fixed rate. Because the rate of incoming data to the PQ is exactly the same as the rate of departing data from the CQ, the CQ becomes empty when the PQ fills up. When the PQ sends a whole block of data to XOR with the keystream to produce a block of ciphertext, the block of ciphertext data is put in the CQ. Hence, the PQ empties and the CQ fills up. This process represents the elastic property of the queues. If resynchronization occurs frequently in a short time, it will cause the PQ overflow. To avoid the overflow, the size of the PQ has to be large enough to reduce the probability of overflow to as small as possible.

The size of queue has an influence on the delay when data passes through the system. If k represents the number of bits in the PQ and M represents the size of PQ which is the same as the size of CQ, the CQ should have $(M - k)$ bits because the incoming speed of the PQ is identical with the outgoing speed of the CQ when no sync pattern is found. The delay through the system is $k + (M - k) = M$ bits [5]. To minimize the delay, the buffer size M should be as small as possible. However, M has to be large

enough to save the incoming data when the block cipher gets delayed producing the keystream due to a burst of resynchronizations. *M* should be greater than or equal to *B* because PQ continues to collect data while the system collects all *B* bits of IV after the sync pattern is recognized. It is possible that the last bit of IV could happen anywhere within a block of ciphertext and there is a scenario where only part of the block needs to be XORed since all bits following the last bit of IV have to be encrypted by the new block of keystream. Although this partial plaintext block is XORed with the keystream as soon as possible, it still gives some delay to the system. Hence, *M* should not be less than *B* bits so that it has enough space to store the data and does not have data overflow [5].

4.2.2.5 Bottom-up Hardware Implementation

In this section, the plaintext subsystem, ciphertext subsystem and keystream subsystem implementation are discussed in detail. The encryption system is then built up. (The decryption system of SCFB mode has a similar structure to the encryption system.)

In this implementation, the Rijndael algorithm with a 128 bit block size is used as the core algorithm of the block cipher. To mitigate buffer overflow, the sizes of PQ and CQ are 256 bits. The sync pattern is chosen as "10000000" with a length of 8 bits.

- Plaintext Subsystem

The main task of the plaintext subsystem is to collect plaintext data and send data to XOR with the keystream when there is enough data in the queue. There are two scenarios for the PQ. One is that the PQ collects *B* bits data and then sends them out when the sync pattern does not occur. When the sync pattern is found, the keystream

subsystem will take the data following the sync pattern as the first part of IV and then the scan part of the keystream subsystem calculates how many bits beyond a block boundary are needed to finish the collection of all B bits of IV. The value calculated is then delivered to the plaintext subsystem. After the value is received, the plaintext subsystem compares it with the number of bits in the queue. If the number is greater than the value received, the plaintext subsystem sends the exact number of bits needed by the keystream subsystem to XOR with the keystream to produce a partial ciphertext block as the rest of IV. If the number is less than the value, the plaintext subsystem will wait until there are enough data bits in the queue and then send them to XOR with the keystream.

The plaintext subsystem includes two main parts which are the storage part and the calculation part as shown in Figure 4.7. There are two parts included in the storage part: buffers and pointers. From the figure, it can be seen that there are two buffers. The upper buffer with the size of 256 bits stores incoming data and the lower buffer saves the 128-bit data extracted from the upper buffer to be ready to XOR with the keystream. The part between the upper buffer and the lower buffer is the pointer part which extracts 128 bits of data from the 256 bit queue to store into the lower buffer according to the result of the calculation part. Although there is a scenario that only a partial block of plaintext is needed when the sync pattern is recognized, the PQ part always provides 128 bits to XOR keystream. The task of extracting the exact number of valid data bits is left to the ciphertext subsystem and the scan part of the keystream subsystem. The extracted data is conceptually removed from the 256 bit queue by adjusting the appropriate pointer.

Because the addresses of the 128 bits of data extracted are contiguous, there are 128 MUXes to select 128 bits from the upper buffer.

The calculation part includes two parts: calculation and comparison. The calculation part is in charge of the number of bits in the upper buffer and sends the number calculated to the pointer part of the queue. There are two cases for the calculation. If the sync pattern is not found, every 128-bit data is sent out from the plaintext subsystem and 128 is subtracted from the *upcounter* which is used to count the number of bits for the upper buffer as shown in the figure. If the sync pattern is found, the number subtracted from the *upcounter* is varied from 1 to 128 depending on where a sync pattern occurs. Thus, the pointers move according to the results of the calculation part. The comparison part is used to compare the number of bits in the upper buffer with the number of bits in a partial block which the new IV needs if the sync pattern is recognized. If the sync pattern does not occur, the comparison part is used to control the time when a block plaintext should be sent out from the lower queue. In other words, only if there are enough bits in the plaintext queue and the keystream is ready, is it the time to send out the plaintext data and the flag, *ready*, is set to '1' to indicate that data is ready.

- Ciphertext Subsystem

The diagram of ciphertext block is illustrated in Figure 4.8. The task of the ciphertext subsystem is to receive the incoming block of ciphertext data and put them into the corresponding positions in the CQ. Again, there are two scenarios. If the sync pattern is not found, it accepts 128 bits of data. However, if sync pattern is found, it only accepts

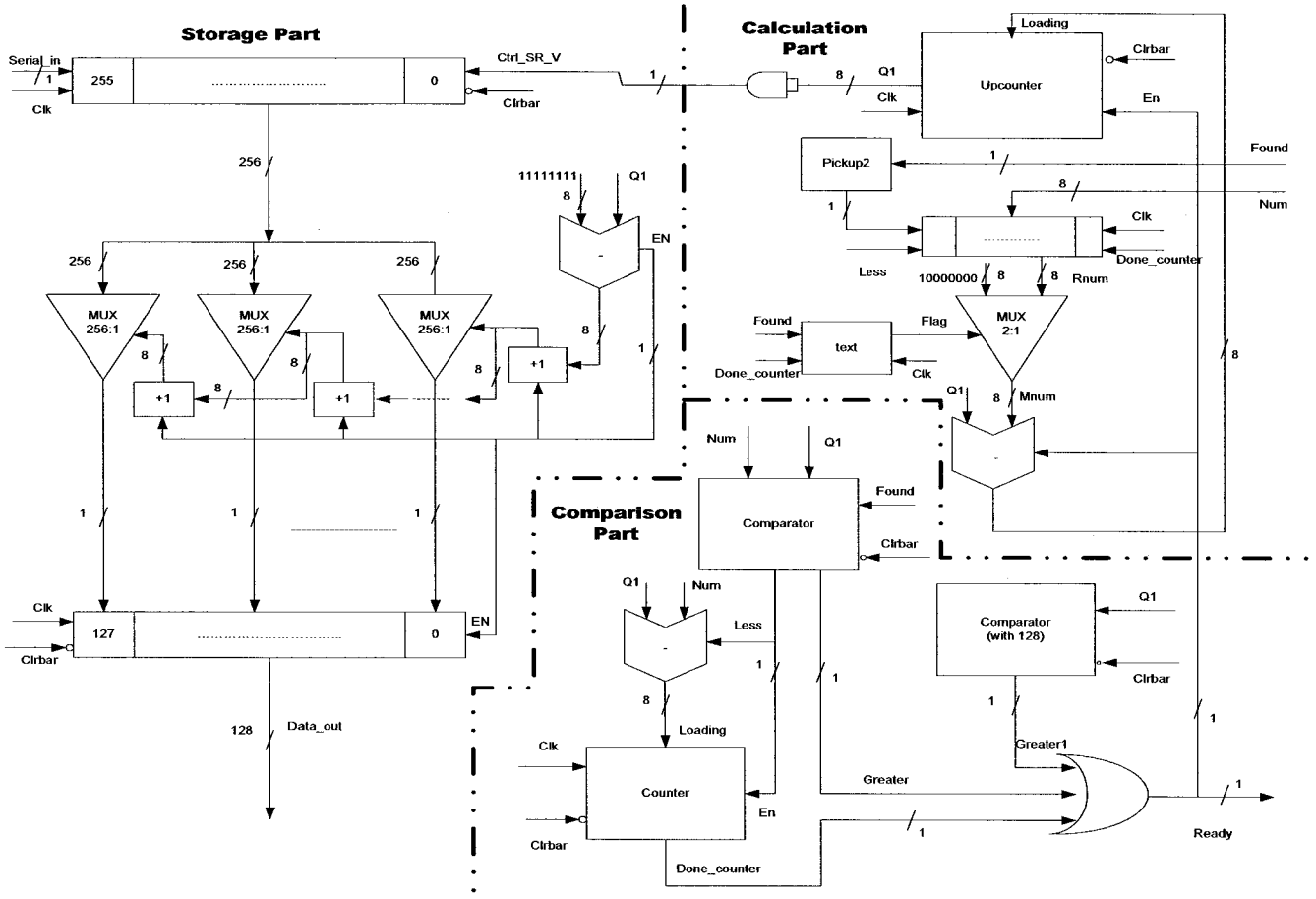


Figure 4.7 Hardware structure of the plaintext subsystem

the number of bits needed. Obviously, there are two parts included in the ciphertext subsystem. One is the 256-bit buffer which receives 128 or less bits of ciphertext to put in the corresponding positions according to the pointer and sends data out bit by bit. The other part is used to calculate the position in the ciphertext buffer for the incoming 128 or less bits of ciphertext .

From the figure, the *ctrl_de* signal stands for the first pointer to point to the position into which the incoming ciphertext bits are moved. Because the addresses of the incoming ciphertext data are continuous, only the address of the first pointer needs to be calculated. The *ready_data* signal indicates when the pointer value is valid for the CQ. When part of the block of ciphertext data is coming, the queue part only accepts the exact number needed and the pointer has to move to the corresponding position.

From the discussion above, it can be seen that the ciphertext part, *Demux*, has the function that accepts 128 or less bits data and distributes each data bit into the corresponding position and shifts data out every clock period.

- Key Subsystem

The key subsystem includes the key generation part shown in Figure 4.9 and the scan part shown in Figure 4.11. The key generation part accepts either the previous output of the block cipher or the new IV from the scan part as the input of the block cipher which is decided by the recognition of the sync pattern. There are two registers, upper *Reg* and lower *Reg*. The upper *Reg*, initialized by the IV, stores the input of the block cipher and the lower *Reg* stores the output of the block cipher to XOR with the plaintext data. The block cipher shown as E block in the Figure 4.9 adopts to the

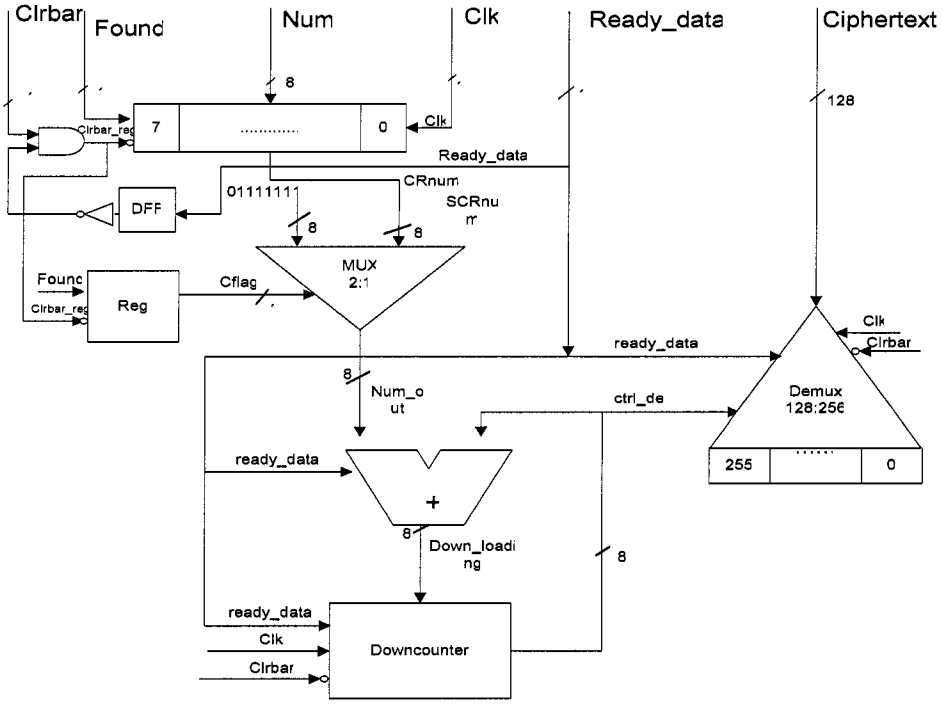


Figure 4.8 Hardware structure of the ciphertext subsystem

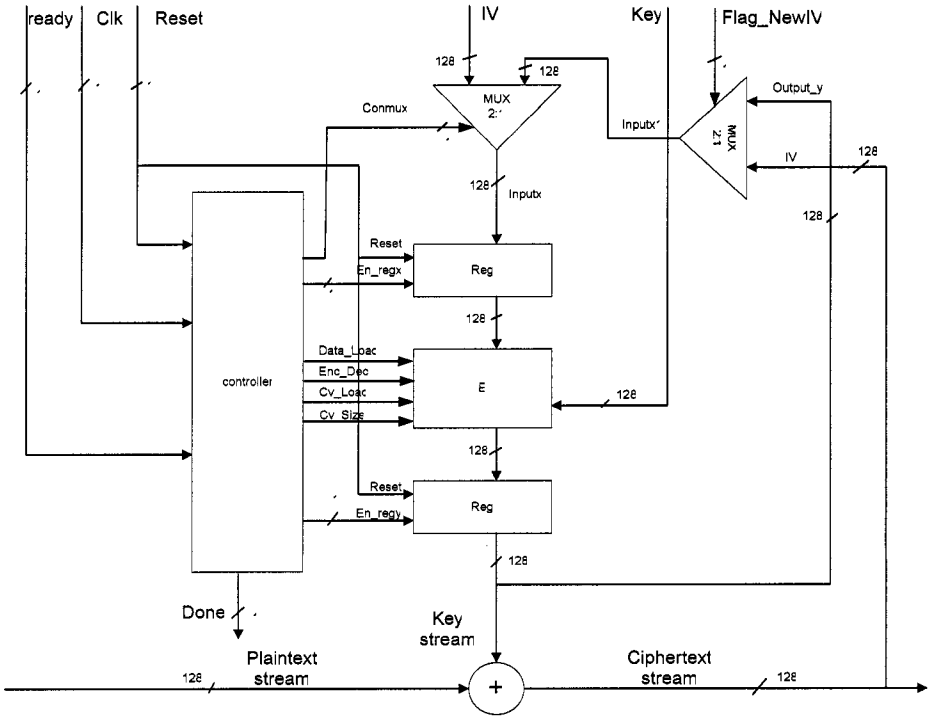


Figure 4.9 Hardware structure of key generation part

Rijndael algorithm to be used to generate the pseudo-random keystream. The implementation of Rijndael algorithm is published by NIST. The encryption of Rijndael algorithm causes some delay which is related with the number of rounds involved in encryption.

The controller controls the enable pins of the upper *Reg* and the lower *Reg* to decide the loading time of a new block of data. Its state machine is illustrated in Figure 4.10. Combined with the Figure 4.9, it is clear that the controller controls the signals of *En_regx*, E*n_regy*, *Data_Load*, *Cv_Load*, *Cv_size*, and *Conmux*. The signal *En_regx* is the enable signal of the upper *Reg*, *En_regy* is the enable signal of the lower *Reg*, *Conmux* is the signal to control the selection of the input of the block cipher, and the signals of *Data_Load*, *Cv_Load*, and *Cv_size* are provided to the Rijndael algorithm of the block cipher.

When started, the system is reset and the controller goes into the state of *RST* to initialize the parameters. After reset changes to '0' from '1', the state of the controller goes to *Gen_key1*, which generates a new block of keystream, on the rising edge of the system clock. When the signal of *CipherDone*, which stands for whether the generation of the new block of the key is completed or not, is set to '1', the controller changes to the state of *Taken_key1*, which means that the new block of the keystream is ready and can be used to XOR with the new block of the plaintext. In this state, the controller sets the output signal, *Done*, to '1' to indicate the encryption system that the keystream is ready. The encryption system is waiting for the completion of the collection of the block of the plaintext. After the collection of the plaintext is finished, a block of keystream XORs

with *B* bits or less than *B* bits plaintext data at a time which depends on whether or not the sync pattern is recognized and the encryption system sets the signal, *Ready_data*, to '1' from '0'. The change of the signal *Ready_data* makes the state turn into *More_time* from *Taken_key1*. This state gives one more clock cycle time to the controller. Because the parallel comparators are applied to find the sync pattern in the ciphertext, the finding can be finished in one clock cycle. Hence, in the state of *More_time*, the signal, *flag_NewIV*, can be decided. Then the controller turns into the state of *Load_input*, which sets the control signal, *Conmux,* to '1' and loads the new input of the block cipher. After the new input is provided to the block cipher, the new round of the controller of the keystream is started.

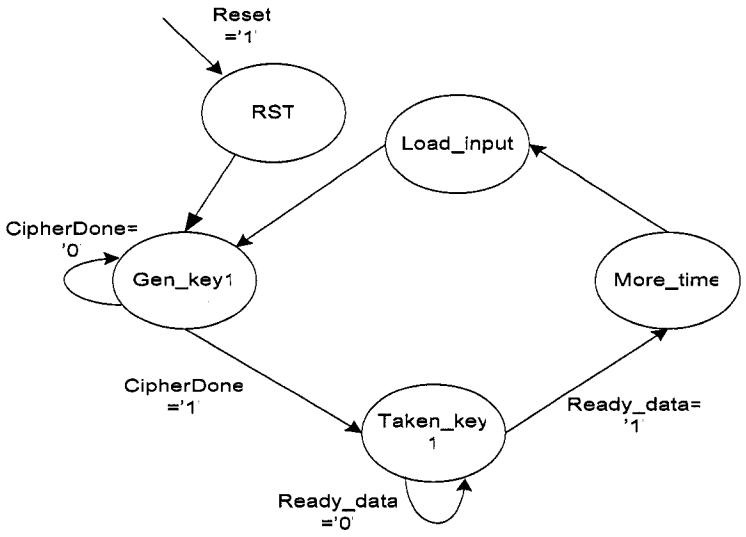


Figure 4.10 State machine of the controller of the keystream subsystem

The scan part [11] is the most important and complicated part in the whole implementation of the encryption system as shown in Figure 4.11. As mentioned above, the scan part has two tasks. One is to check for the sync pattern and the other is to collect

the new IV when the sync pattern is found. Before implementation, there are several issues which need to be considered.

- How to check for the sync pattern?

Because the data is encrypted block-by-block, it is necessary to consider the situation of the borders of the blocks since there is a possibility that the sync pattern is split across blocks. Copying the last n-1 bits of the previous comparison block to the next first n-1 bits of the current comparison block is a useful approach to solve this problem. That is why the queue of the sync pattern has n-1 more bits than the block size B. The number of B comparators is needed to check the (B+n-1) bits of data at once.

- How to collect the new IV after the sync pattern is recognized?

It is impossible to fulfill the collection in one block. Hence, how to calculate the number of bits remaining in the new IV past a block boundary is an issue. The data following the new IV should be encrypted by the key which is generated by the new IV. Therefore for the collection of the second part of the new IV, the PQ needs only collected to the number of bits that the new IV needs to XOR with the keystream to generate the rest of the new IV. The comparators compare the data with the sync pattern and the controller is used to control when the result of the comparison is sent to the encoder because SCFB mode does not check the sync pattern during the collection of the new IV. After the flag, "*found*", changes from '0' to '1', the control part closes the output of the comparators and keeps the previous value to the encoder. If the sync pattern is found, the encoder can translate the results of the comparison into the position where the sync pattern is.

According to the pointer of the encoder, the number that the new IV needs for the rest can be calculated by use of the adders. This value is sent to the plaintext subsystem and the ciphertext subsystem.

- How to extract the new IV from the queue if the sync pattern is found?

The two components of "MUX 2:1" on the right side of the figure are used to point to the correct positions. The upper "MUX 2:1" points out the position of the data extracted from the queue and the lower "MUX 2:1" points out the position where the bits extracted are going to. The "MUX 135:1" outputs the data pointed to and the component, *SDemux*, accepts the data from 128 "MUX 135:1" to form the new IV according to the pointer of the position. When the collecting of the new IV is completed, a flag in the "MUX 2:1" indicates the new IV done.

- Encryption System

An encryption system is formed by putting the plaintext subsystem, the keystream subsystem and the ciphertext subsystem together. The encryption of the system needs to wait until the keystream and the 128 bits of plaintext data (or the number needed) are ready. When the ready signal of the keystream subsystem, *Done*, and the ready signal of the plaintext subsystem, *Ready*, are set to '1', the system starts to send plaintext data to XOR with the keystream to generate a block of ciphertext data and sets the flag, *ready_data*, to '1'. This indicates that the current ciphertext data are valid ciphertext data. The ciphertext subsystem can place it in the queue.

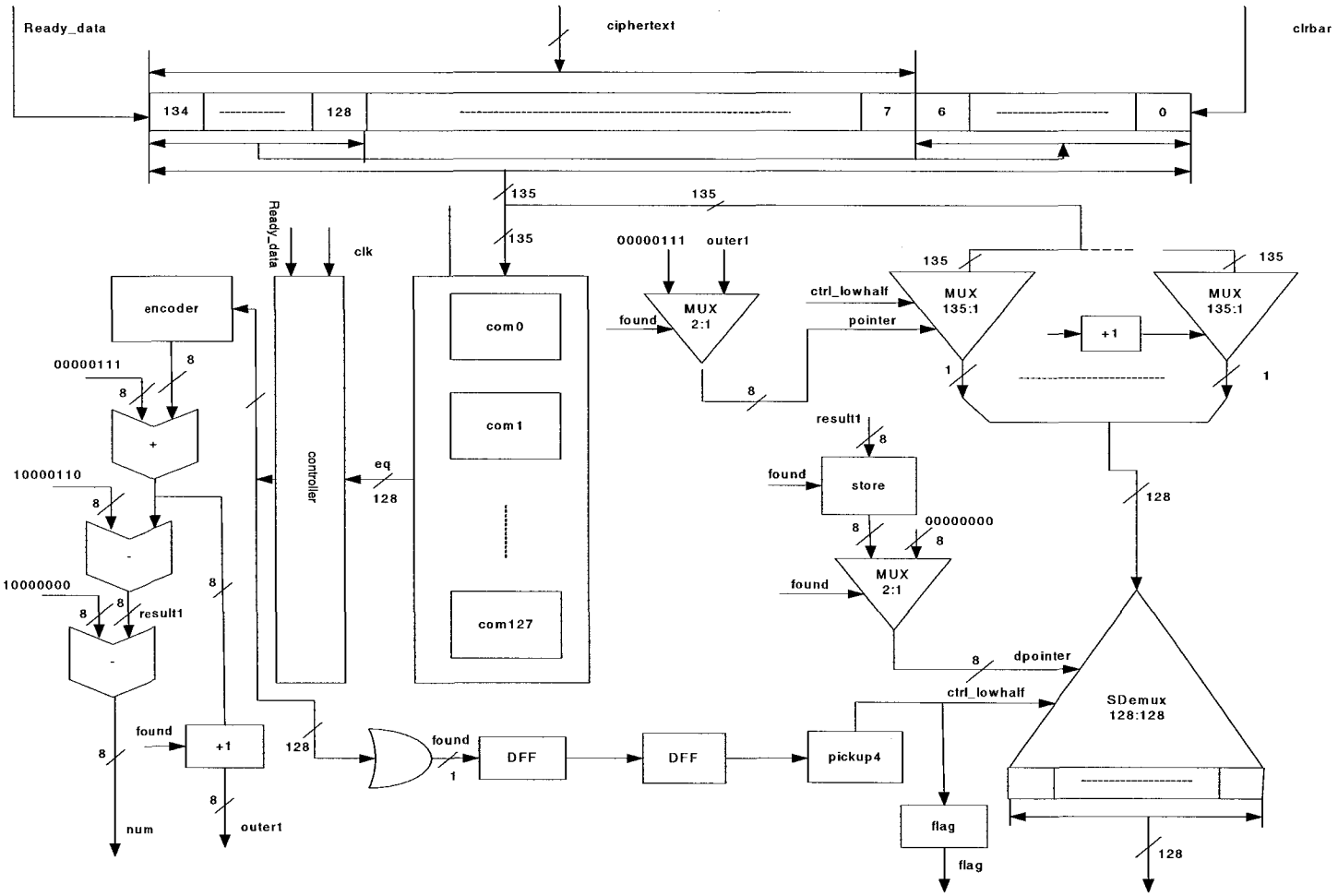


Figure 4.11 Hardware structure of the scan part of the keystream subsystem

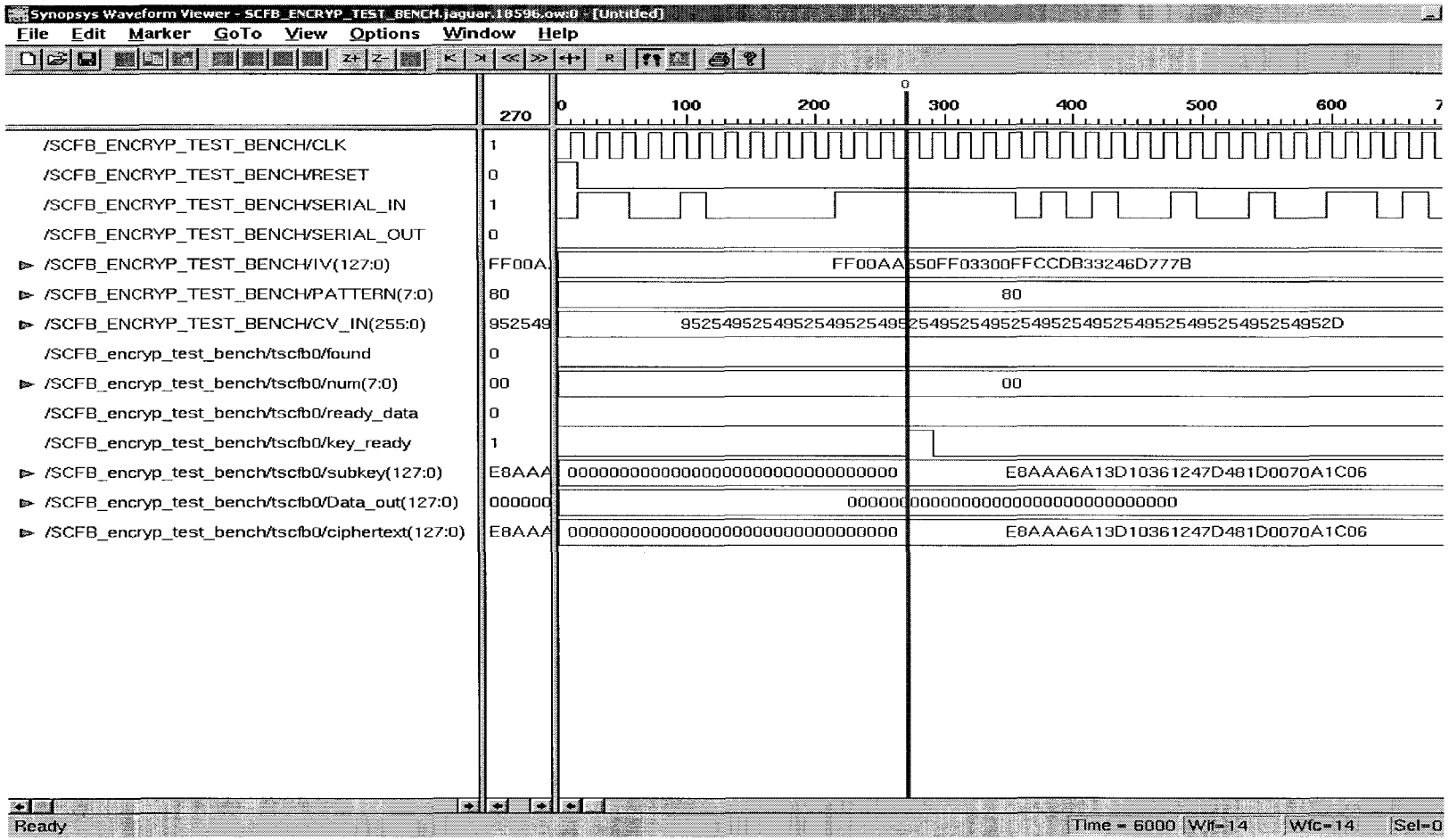


Figure 4.12 Simulation waveform of the encryption system

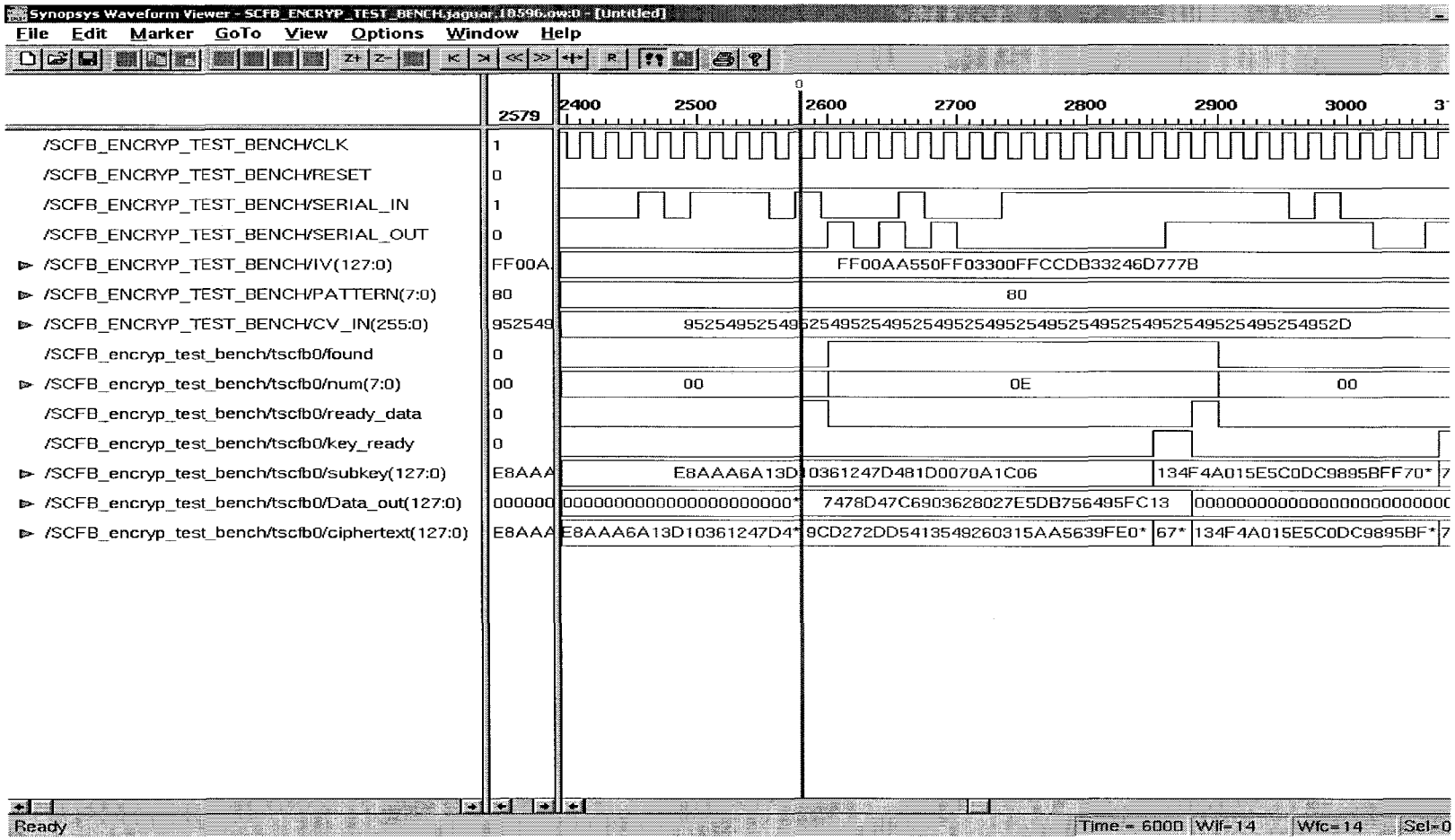


Figure 4.13 Simulation waveform of the encryption system

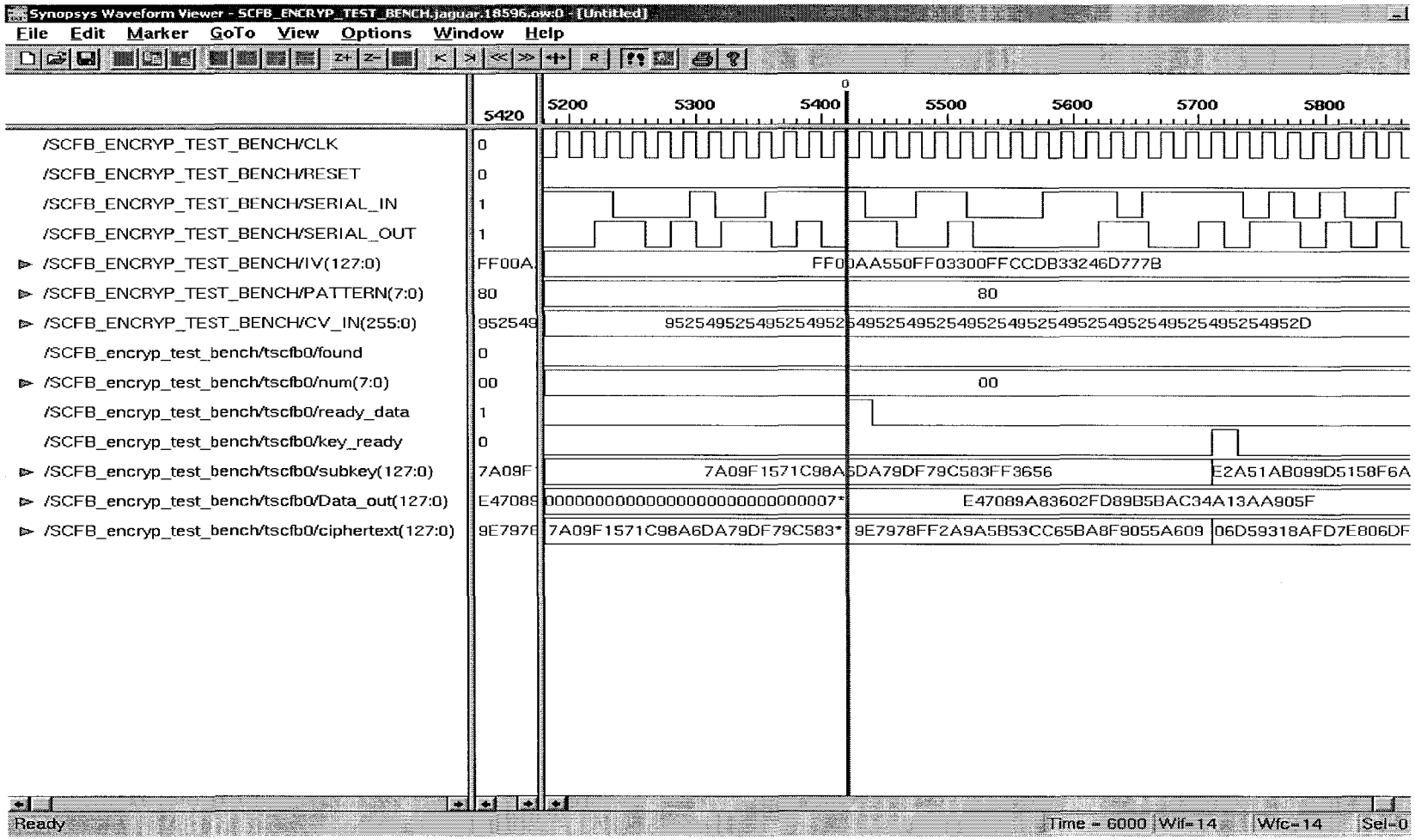


Figure 4.14 Simulation waveform of the encryption system

The encryption system, coded using VHDL and simulated by the Synopsys tool [8] and the waveforms are shown in Figure 4.12, Figure 4.13, and Figure 4.14. The Figure 4.12 displays that the first block of keystream is ready and the ready flag, *key_ready*,becomes '1'. *Data_out* (127:0) in the figure stands for a block of plaintext data sent out to XOR with the keystream. In this figure the collection of 128 bits of plaintext data is not ready. The Figure 4.13 indicates that the first 128 bits of plaintext data is ready and the new block of ciphertext data is generated. The encryption system sets the flag, ready_data, to '1'. The sync pattern is found after the ready_data flag is set to '1'. The system calculates instantly the number needed by the second part of new IV which is shown as 0E in hexadecimal. The finding of the sync pattern makes the keystream subsystem restart to generate the new block of keystream. After several clock cycles, the key_data becomes '1' and 15 bits of plaintext data is ready. It produces the new 15-bit ciphertext data and the ready_data flag becomes '1' again. The new IV is sent to the block cipher as the new input to produce the new block of keystream. The collection of the new block of plaintext data is ready in the Figure 4.14. The new block of ciphertext data is produced and no sync pattern is found in this round. More waveforms of simulation results of the encryption system are shown in Appendix A

- Decryption System

The decryption system has the same structure as the encryption system except the plaintext instead of the ciphertext is shifted in the last bit of the upper *Reg*. The simulation waveforms of the decryption system are shown in Appendix A.

4.2.2.6 Test Methodology

To test the full encryption system, a system was built up by combining the encryption system and the decryption system. Random bit patterns generated by C++ code were used as the plaintext input and saved to an input file. The random plaintext is encrypted by the encryption system and then sent to the decryption system. By decryption, the ciphertext are turned into recovered plaintext. Comparing the plaintext recovered with the original input, we can know whether or not the system working is correct. One special case of the inputs and outputs of the system is shown in Appendix A to make results easy to check is to let all of the input be '1's at the encryption system. The first 128 bits of the output of the decryption system in the Appendix A are the initial bits in the CQ of the decryption system. In this implementation, the CQ of the decryption system is initialized as all '0'.

4.2.2.7 Complexity of Hardware Implementation

The hardware implementation of SCFB mode utilizes the Synopsys tool based on 0.18 μm CMOS technology to perform the front-end synthesis of the design. The hardware complexity as shown in Table 4.1 is reported by the design analyzer of the Synopsis tool with the constraint of the system clock of 10 ns. With the system clock rate, no slack is generated during the process of synthesis. No slack means that the design circuit can satisfy the required speed. The synthesized result of the Rijndael algorithm comes from [12]. When the circuit is synthesized it gets a report indicating a number of different gates, timing and a total overall area. One common way to estimate the circuit size is to use the number of equivalent 2-input NAND gates as a metric of the circuit size.

The area of synthesized circuits, which is in square microns (μm^2), is converted to the gate count by using the two-input NAND gate which has the area of 12.197 μm^2 as a basis for comparison.

The table shows that estimated total gates of the encryption system is 1255644, out of which the Rijndael algorithm needs 612834 gates. Hence, the whole keystream subsystem including the key generation and the scan part occupies 60% of the hardware complexity of the system.

4.2.2.8 Discussion of Other Structures

There are other structures that are suitable to implement the SCFB system. One approach would be to remove the PQ and the CQ and thus remove elasticity in the flow of data through the system. A block cipher output must then be generated within one bit time of incoming data. This would clearly minimize delay through the system as the block cipher runs at the rate of B times of link rate. The system efficiency becomes $1/B$, which is the same as for the CFB mode. Hence, this approach has little value [5].

4.3 Conclusion

This chapter introduces the concepts of SCFB mode and describes the structure of a hardware implementation of an SCFB system. In the hardware implementation of SCFB mode, parallel transfer is applied to obtain high working efficiency. The hardware aspects of SCFB mode such as the queuing requirement, the relationship between queue sizes, and the data delay during the transmission from the plaintext queue to the ciphertext queue are discussed briefly. These will be further discussed in Chapter 6.

For a hardware implementation, SCFB system has the ability of self-synchronization and can obtain much higher efficiency than CFB mode. Compared to Rijndael algorithm, the system doubles in hardware complexity. We conclude that SCFB systems are well suited to high-speed digital hardware implementation.

Component Name	Combinational Area	Noncombinational Area	Total Area	Total Gate
Plaintext subsystem	2,237,383	89,662	2,327,046	190,788
Keystream generation part without Rijndael	11,298	13,221	24,519	2,010
Scan part of keystream subsystem	1,512,222	148,470	1,660,693	136,155
Rijndael algorithm				612,834
Ciphertext subsystem	3,759,299	68,801	3,828,101	313,856
Encryption system				1,255,644

Table 4.1 Hardware complexity of the encryption system of SCFB mode

Chapter 5
Optimized Cipher Feedback (OCFB) Mode

In this chapter, optimized cipher feedback (OCFB) mode [13] is investigated. As with the SCFB mode, OCFB mode can configure the block cipher as a stream cipher by using the output of the block cipher as the keystream to XOR with the plaintext to produce a block of ciphertext.

Compared with CFB mode, OCFB mode attains higher efficiency and the ability of self-synchronization by checking for a sync pattern in the process of producing ciphertext. OCFB mode is quite similar to SCFB mode except OCFB mode keeps checking for the sync pattern all the time even during the IV collection phase. These differences influence the method of implementation and the properties of the system.

This chapter describes the working theory first and then the top-down design is given. Finally it provides the hardware implementation of OCFB mode in detail. In the next chapter, the discussion on the performance analysis of OCFB mode will be combined with performance analysis of SCFB mode.

As we discussed in Chapter 4, the serial transfer from the PQ to the CQ is adopted in the hardware implementation of OCFB mode instead of the parallel transfer of our SCFB mode implementation in order to analyze the influences given by the different methods of implementation. As we shall see, the serial transfer reduces the hardware

complexity but brings a complicated timing relationship to the implementation of OCFB mode which limits its operating efficiency.

5.1 Introduction of OCFB mode

The motivation of OCFB mode is to optimize CFB mode by improving the efficiency while still achieving the ability of self-synchronization. The approach is illustrated in Figure 5.1. OCFB mode is optimized by buffering all output bits of block cipher into the shift register *SR2* as keystream to produce ciphertext. The ciphertext is sent out to the communication channel and the shift register *SR1* as the input of the block cipher simultaneously. A counter, named *shiftcounter*, is used to trigger execution of the block cipher after enough bits are collected in *SR1*. However, synchronization would be destroyed due to bit slips or insertions in a communication channel because the counters of the encryption system and the decryption system would lose synchrony relative to the ciphertext stream. Hence, resynchronization has to be done for both sides of counters. The only way to obtain resynchronization is to check for a sync pattern in the ciphertext because the ciphertext can be obtained for both the encryption and the decryption. The encryption system and the decryption system in the figure compare the current content of *SR1* with the sync pattern on each clock cycle. The counters are not reset until the sync pattern is found. This causes the counters of the encryption system and the decryption system to obtain synchronization again.

Unlike SCFB mode, OCFB mode continues to check for the sync pattern in all of the ciphertext bits even when IV is collecting. Here we define IV to be the next B bits following the last bit of the sync pattern. This gives the OCFB mode more opportunity to

resynchronize. However, for a hardware implementation we shall see that the result is that it is more likely to have a buffer overflow resulting in a decrease of the system performance.

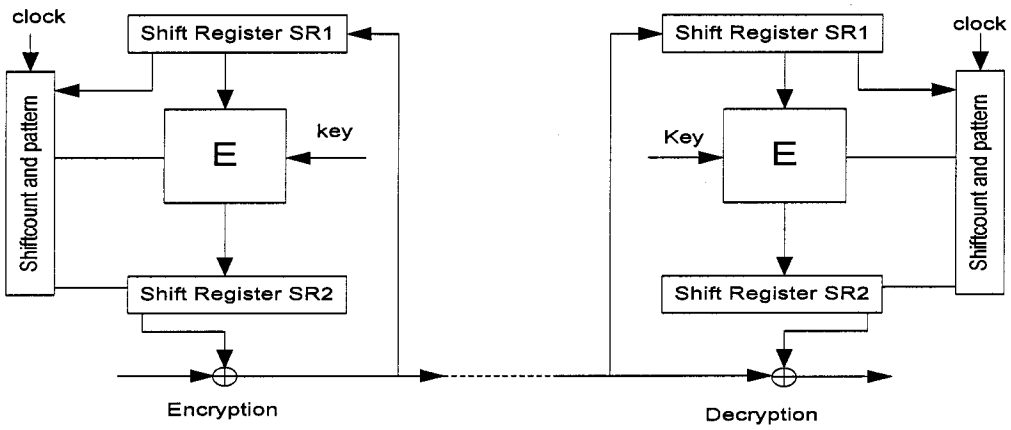


Figure 5.1 OCFB system

5.2 Implementation of OCFB mode

5.2.1 Software Implementation

To precisely illustrate the operation of OCFB mode, the flowchart of OCFB mode is shown in Figure 5.2. In the flowchart, B represents the block length and *shiftcount* represents the counter with $\log_2 B$ bits which is used to keep track of which bit is being XORed with plaintext. $SR1_i$ and $SR2_i$ represent the i-th bit of *SR1* and *SR2,* respectively. The notation $SR1_0...SR1_{B-1} \leftarrow SR1_{1...}SR1_{B-1}C_{j+i}$ is used to indicate the shifting of *SR1* from the higher bit position to the lower bit position and the highest position is substituted by the ciphertext bit. The IV that is known by both the encryption system and the decryption system is loaded into *SR1* after the start of the systems. The pattern represents the sync

pattern which is used to resynchronize the system. The block cipher E_k encrypts B bits of data to produce the same length of output as keystream and is stored in *SR2*. We assume the E_k represents 128-bit Rijndael with 128 bit key.

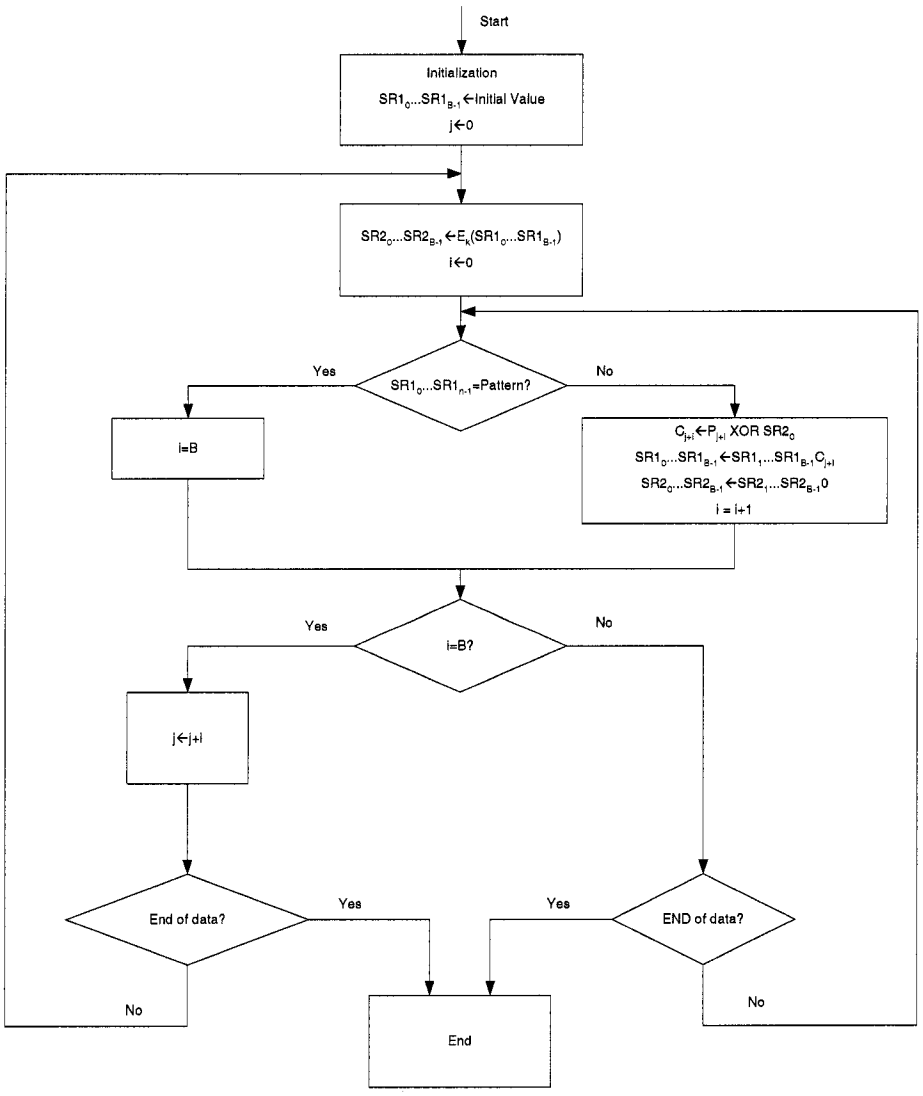


Figure 5.2 Flow chart of OCFB system

After the system is started, an IV is loaded into *SR1* and the system encrypts IV to produce the first block of keystream. The symbol *i* representing the *shiftcounter* is cleared to '0'. The system starts to compare the 1st n bits of the current content of *SR1* with the

sync pattern. If it is equal to the sync pattern, the system sets i to B and triggers E_k to encrypt B-bit *SR1* and save the B-bit output into register *SR2*. If the content of *SR1* is not equal to the sync pattern, i will add one. If i does not reach B, *SR1* and *SR2* shift one bit position and the ciphertext generated by the previous *SR2* bit is moved into the last bit of *SR1*. Meanwhile, the ciphertext is sent out to the communication channel. This process is repeated until all of the data is encrypted.

The software implementation of the OCFB mode is simple. Because software executes sequentially and does not take advantage of concurrency to gain efficiency, hardware implementation realizes the full value of an OCFB system. The advantage of a hardware implementation is that components can run concurrently thereby minimizing the idle time for each part to improve the system efficiency.

5.2.2 Hardware Implementation

5.2.2.1 Top-down Hardware Design of the OCFB Mode

From the description of the OCFB mode, it is clear that six data input ports and one output port are required as shown in Figure 5.3. The *clk1* port provides the system clock. The *Reset* port is used to reset the system. The *Key* port provides the primary cipher key for the Rijndael algorithm. The *Plaintext data* port serially collects the incoming data bits. The *IV* port provides the first IV to the system. The encryption system and the decryption system initially have the same IV and primary key to give the system the same starting point.

The encryption system can be divided into three functional subsystems which have different tasks: the plaintext subsystem, the keystream subsystem, and the ciphertext

subsystem as shown in Figure 5.4. The plaintext subsystem is in charge of collection, storage and sending out plaintext data. The keystream subsystem prepares for the keystream bits and controls the synchronization between the keystream and the corresponding plaintext. The ciphertext subsystem takes charge of collection of ciphertext bits and sends them to the communication channel.

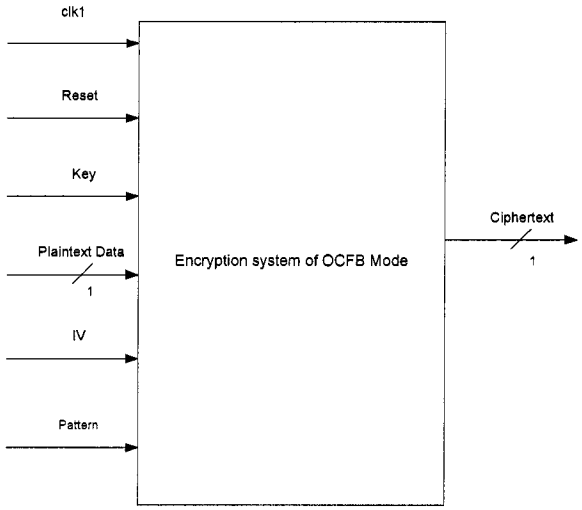


Figure 5.3 Port relationships of the encryption system

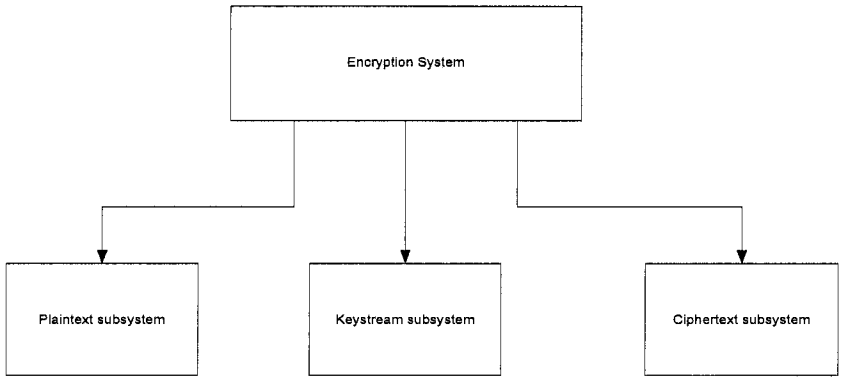


Figure 5.4 Block diagram of the encryption system

The OCFB system needs PQ and CQ to store the data bits in order to ensure that the incoming and outgoing data speed are the same even while the processing of data

inside the system is not at a constant rate. The queues provide the storage for the incoming and outgoing data while a new block of keystream is being generated because the production of the new keystream block requires more time than one clock cycle and data may have to wait until the new block of the keystream is ready.

The keystream subsystem uses a register *SR1* to store the input of the block cipher and a register *SR2* to store the output of the block cipher. A counter, called *shiftcounter,* should be used to count the amount of shifting and a *comparator* is needed to check for a sync pattern in the ciphertext.

The general sketch of the encryption system of OCFB mode is illustrated in Figure 5.5. In the figure, B stands for the length of a cipher block and n stands for the length of the synchronization pattern. R is the rate of data coming into the PQ and R' is the rate the plaintext bits that leave from the PQ. R_e is used to represent the rate of encryption of the block cipher. It makes sense that the outgoing rate of keystream bits is equal to R' since the keystream bits directly XOR with the outgoing plaintext data. As well, the incoming bit rate of the CQ has to be the rate of outgoing from PQ, R', otherwise it will cause the loss of data or duplication. In order that data comes into the system and leaves the system at a uniform rate, the outgoing rate of data in the CQ should be equal to R. To reduce the possibility of PQ overflow as much as possible, R' must be greater than R to compensate for delay in producing keystream blocks due to the block cipher process when resynchronization occurs.

In the figure, there are three clocks, *clk1*, *clk2*, and *clk3*, to control the running speeds of the data transfer and the block cipher. Among these three clocks, the *clk1* is the

fastest clock and it can be the system clock in the implementation. The *clk2* and the *clk3* can be derived from the *clk1*. Since the data collection of PQ is based on the *clk2*, the rate of incoming plaintext data of PQ, *R*, is directly equal to the frequency of the *clk2*. As well, *R'* is equal to the frequency of the *clk1* and R_e is equal to the frequency of the *clk3*. The system efficiency can be controlled by adjustment of these three clock frequencies.

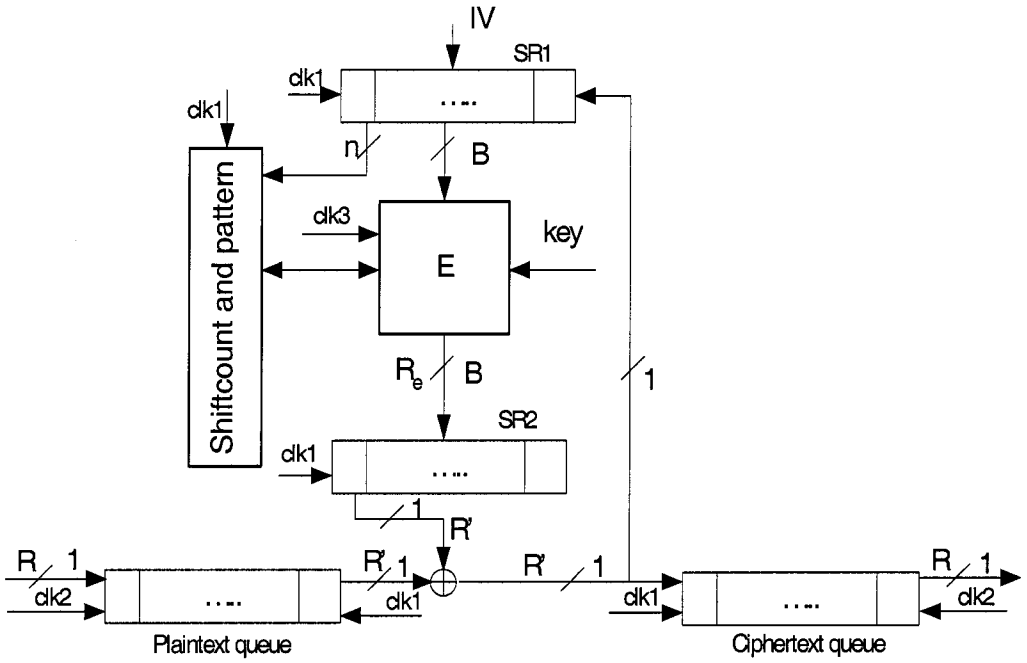


Figure 5.5 Structure of the encryption system

In each subsystem, the plaintext subsystem, the keystream subsystem, and the ciphertext subsystem, there are two different rates. The plaintext subsystem collects incoming data at the rate *R* and sends data out at rate *R'*. The ciphertext subsystem has the reverse situation of the plaintext subsystem. For the keystream subsystem, the interfaces of the keystream subsystem use the rate *R'* to keep the same pace with the interfaces of the plaintext subsystem and the ciphertext subsystem. The core component

of the keystream subsystem, the block cipher, adopts the rate R_e to control the running time of the block cipher in order to reduce the idle time of the keystream subsystem to improve the system efficiency. Hence, there are two conversions of the rate in the keystream subsystem. One is from *SR1* to the block cipher and the other is from the block cipher to *SR2*.

5.2.2.2 Discussion on Queuing

Both PQ and CQ have the same buffer size. While bits are coming into PQ, bits are leaving from CQ at exactly the same rate, like SCFB mode. During the period that the block cipher encrypts to produce the new block of keystream, the incoming plaintext bits are stored into the queue until the block of keystream is available. After the keystream is ready, PQ starts to shift bits out until the counter counts to the maximum or the sync pattern is found. The block cipher is then triggered to generate the new keystream block. The input of PQ therefore accepts bits at a uniform rate, but the output of the PQ shifts bit by bit only when the keystream is available.

CQ has exactly the reverse situation as PQ. When the keystream is available, the CQ starts to accept the incoming data bit by bit. While the block cipher encrypts to generate the new keystream, CQ stops collecting data until the keystream is ready. However, the output of CQ keeps the uniform rate to send bits out. The CQ is initialized with arbitrary data so that it is in the full state initially when the PQ is empty. When PQ fills up, CQ goes to empty. The situation of overflow of the PQ is exactly the same as the situation of underflow of the CQ. Hence, the overflow condition of PQ only needs to be considered.

The size of the queue has an influence on the delay of data passing through the system. Let k represent the number of bits in PQ and M represent the size of PQ and CQ. The delay of data passing through is $k + (M - k) = M$ bit times. To minimize the delay, the buffer size M should be as small as possible. However, M has to be large enough to save the incoming data when the block cipher produces the keystream. Because OCFB mode checks for the sync pattern all the time, bits fill up in PQ easily causing overflow if resynchronizations occur frequently. (This manifests as an underflow in CQ.) Loss of data bits will then occur. Whether or not PQ overflows is related to the occurring frequency of the sync pattern, the size of M, the time spent by keystream generation, and the incoming rate of plaintext bits. Of these four factors, the last three can be decided when the system is set up. However, the first factor relates to the size of the sync pattern and the time between sync patterns is given approximately as the geometric probability distribution [5]. The relationship between the probability of overflow and the buffer size will be discussed in Chapter 6. In order to minimize the probability of overflow, PQ has to be large enough and outgoing data rate leaving the PQ has to be greater than the incoming data rate. Hence, there is an important relationship among the rate R_e of the block cipher encryption, the rate R' of bits removed from PQ, the rate R of bits coming into PQ and the buffer size.

5.2.2.3 Discussion on Timing Characteristics of Implementation

Clearly, there are two working threads in the OCFB system. One is the generation of the keystream and the other is data movement in the queues. In order to obtain higher system efficiency, the encryption of the keystream and the data transfer in the queue have

to work concurrently and shorten the idle time of each subsystem. The different rates, R', R, and R_e, introduce the different times, T', T, and T_e. T' represents the time spent sending one block (i.e. B bits) of plaintext data out of PQ, T is the time collecting one block of plaintext data in PQ, and T_e is the time generating one block of keystream.

In this thesis for ease of implementation 50% efficiency is examined. 50% efficiency implies that for every B bits entering the queue, $2B$ bits are removed from the queue. Unlike SCFB mode, 50% efficiency does not have special meaning for OCFB mode. 50% efficiency is chosen because it allows the system to generate clocks that are integer multiples of the system clock *clk1*. This results in the following equations where R, R' and R_e are in units of bits / second:

$$\text{Efficiency} = R / R_e = 50\%$$

$$R' = 2 * R$$

$$R' = R_e.$$

From the three equations above, it is easy to deduce the equations as below:

$$T' = T / 2$$

$$T' = T_e$$

$$R_e = B / T_e$$

From the equations above, it is seen that the generating time of one block of keystream, T_e, is the same as the leaving time of one block of plaintext data, T'. T_e and T' are equal to half of T. This implies that the generation of one block of keystream and the XORing of one block of plaintext have to be completed in the time of T / 2 for 50% efficiency.

The encryption processing of OCFB system can be described as follows. After the system is reset, the keystream subsystem starts to produce a new block of keystream and the plaintext subsystem begins to collect the incoming data. Since the collecting speed of the plaintext data is two times slower than the generation speed of one block of keystream, it can be imagined that $B / 2$ bits would be in PQ when the generation of one block of keystream is finished. Then PQ starts to send data out to XOR with keystream generated bit by bit after the keystream is ready. During B bits of keystream XOR with the B bits of plaintext data bit by bit, PQ still collects incoming data at the rate of R. This can happen only when resynchronization does not occur. If resynchronization occurs, only part of the B bits in PQ XOR with the key and the rest of the bits will stay in the queue to wait until the new keystream block is generated. It is clear that bits will fill up in the queue and might cause overflow if resynchronization occurs frequently. Enlarging the queue size may be a better way to decrease the probability of overflow, but it will suffer from longer delay.

Figure 5.6 illustrates the timing relationships between the data flow associated with PQ, CQ and the keystream. This figure also shows some detailed considerations for implementation. In the diagram, $clk2 = 2 * clk1$ since $R' = 2 * R$. The up / down arrows indicate that the components are triggered by the rising/falling edge of clock. The lower case s stands for shifting and the lower case w for writing. Because a component in the hardware implementation cannot resolve two signals simultaneously, the rising and falling edges are selected to avoid two signals affecting one component at the same time. Writing into the PQ is controlled by *clk2*. The rising and falling edges of *clk2* are always

on the rising edge of *clk1*. Hence, there is no limitation on the selecting of the rising or the falling edge for the writing of the PQ. However, the control of the shifting of PQ has to use the falling edge since writing has already occupied the rising edge. CQ has the same situation except writing is using the falling edge of *clk1* and shifting is using the rising edge of *clk2*. Because a plaintext bit is shifted out on the falling edge of *clk1*, the keystream has to keep the same pace as the plaintext bit to guarantee the synchronization. Hence, the interfaces of the keystream part have to use the falling edge of *clk1*. The writing of CQ triggered by falling edge cannot write the latest bit but the previous one because the generations of plaintext and keystream depend on the falling edge of *clk1*. Therefore, the data of CQ is delayed one *clk1* cycle.

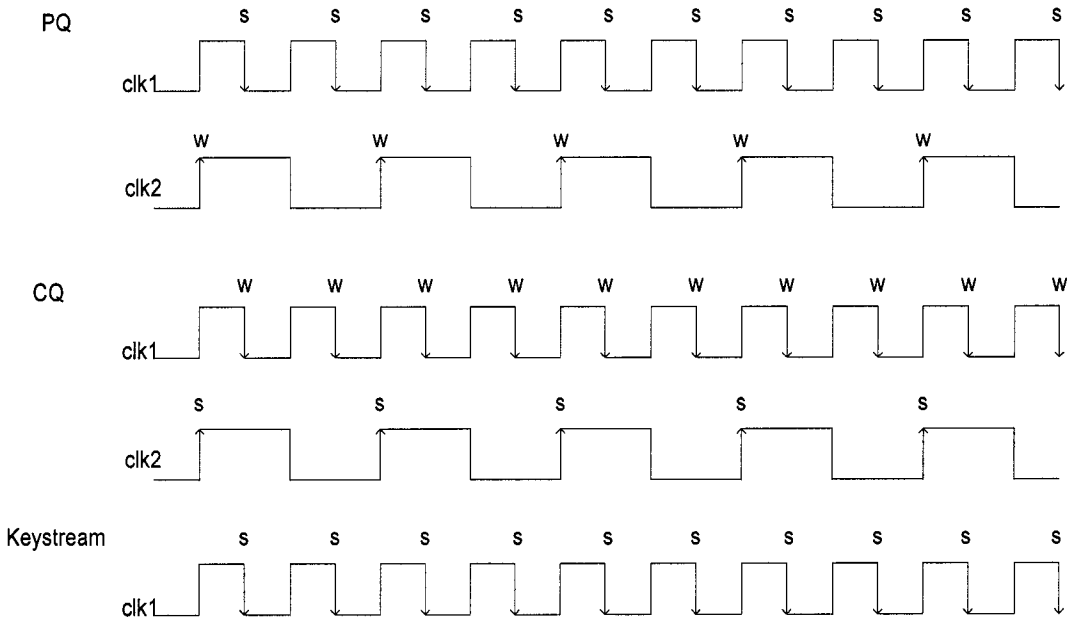


Figure 5.6 Timing relationships between PQ, CQ, keystream

5.2.2.4 Bottom-up Hardware Implementation

In this part, the plaintext subsystem, the ciphertext subsystem and the keystream subsystem are implemented individually as foundational subsystems and then encryption system is built up from them. (The decryption system of OCFB mode has the same structure as the encryption system.)

Some conditions are assumed before the implementation. In this implementation the Rijndael algorithm is used as the core algorithm of the block cipher. Alternatively, triple DES, DES or other algorithms which are thought to be secure can be used as the block cipher. For Rijndael, the length of each block is 128 bits. Hence, the sizes of *SR1* and *SR2 must* be 128 bits as well. To decrease the delay as data passes through the system and to make the possibility of overflow / underflow low, the sizes of PQ and CQ are chosen as 256 bits. The sync pattern is chosen as "10000000" and the length is 8 bits.

- Plaintext Subsystem

As mentioned above, PQ is a 256 bit queue and collects data bits while the keystream subsystem produces a new block of keystream. After the new block of keystream is ready, the PQ starts to send bits out. The writing of PQ is triggered by the rising edge of *clk2* and shifting out is triggered by the falling edge of *clk1* under the condition that the valid signal, named *val*, is '1'. The PQ is illustrated in Figure 5.7.

The implementation of PQ should include the following interface parts:

a. The *val* signal indicates whether the output of PQ is valid or not.

b. One 8-bit write pointer points to the correct position for writing.

c. The signal *clk2* is generated for *clk1* and is twice *clk1* because of 50% efficiency.

d. One data input port and one data output port

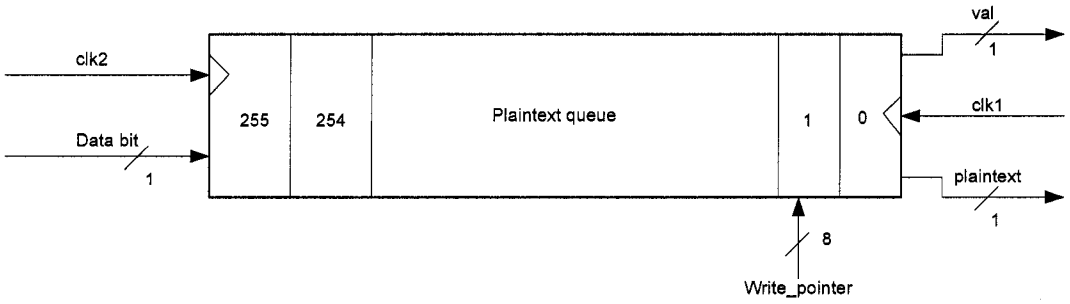


Figure 5.7 Structure of the plaintext queue

- Ciphertext Subsystem

The ciphertext queue has the reverse situation of PQ and is shown in Figure 5.8. It collects data bits according to the falling edge of *clk1* when the *val* signal is '1' and stops collecting bits when the *val* is '0'. The output of the CQ is timed by the rising edge of *clk2*. The write pointer points to the position for the incoming data bits. The CQ starts from the full state with arbitrary data. To make checking easier, the initial value of CQ is set to all of '1's. Hence, the write pointer starts from 255, increases by one at the falling edge of *clk1* when *val* is '1', and decreases by one on each rising edge of *clk2*.

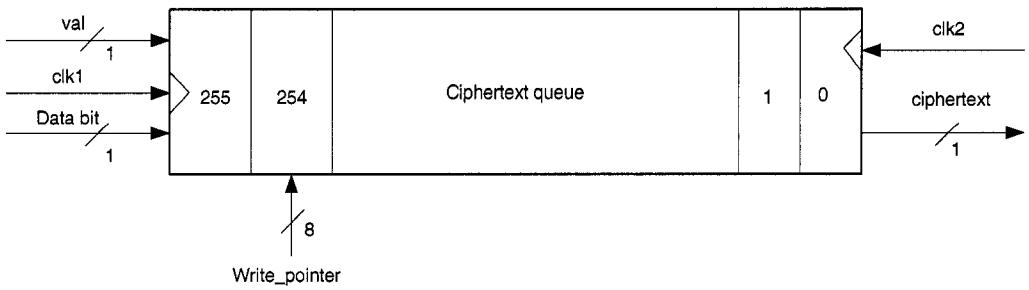


Figure 5.8 Structure of the ciphertext queue

- Keystream Subsystem

The keystream block consists of two 128-bit registers *SR1* and *SR2*, the core block cipher, and a *shiftcounter* which includes one 8-bit comparator and a counter, as shown in Figure 5.9.

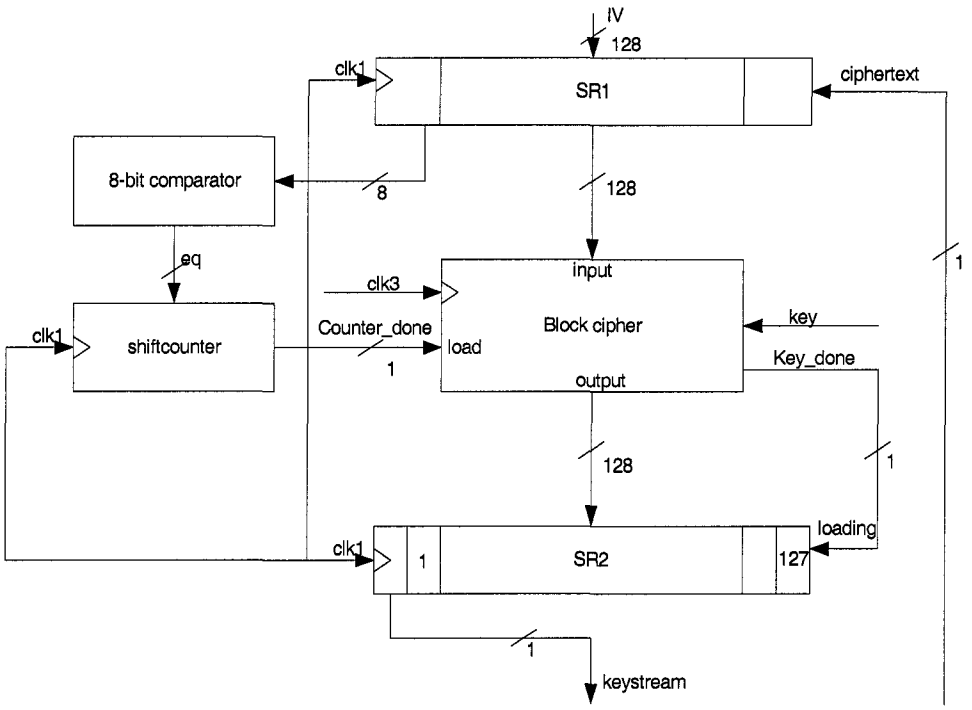


Figure 5.9 Hardware structure of the keystream subsystem

This part mainly takes charge of the generation of keystream. The 8-bit *comparator* compares the first 8 bits of *SR1* with the sync pattern until the sync pattern is found. The *eq* signal in the figure is the output of 8-bit *comparator*. It sets the counter to the maximum when *eq* is equal to '1'. The counter then sends a done signal out to the core algorithm of the block cipher. This *counter_done* signal triggers the block cipher to load the 128-bit data of *SR1* as the input of the block cipher. Rijndael, the core algorithm of the block cipher, controlled by *clk3* is in charge of the generation of the keystream.

The time spent by the Rijndael algorithm is mainly constrained by the method of implementation of the algorithm and the current hardware technology. The frequency of *clk3* is derived from the system clock, *clk1,* and is equal to *g* * *clk1* where *g* is an integer and decided by system efficiency, the time used by the generation of one block of keystream and the time used by collection of one block of plaintext data. After the new keystream is produced, the *key_done* signal is set to '1'. Due to the connection between *key_done* signal and the loading port directly, *SR2* loads the output of the block cipher asynchronously and then shifts data by the falling edge of *clk1*. The data shifted out from *SR2* is XORed with the plaintext bit to produce the ciphertext bit.

- Encryption System

The plaintext subsystem, the ciphertext subsystem, and the keystream subsystem together make up an encryption system. After the system is reset, PQ starts to collect data, CQ starts to shift data out and the block cipher is triggered to generate the keystream block. Simulation waveforms are shown in Figure 5.10, Figure 5.11, Figure 5.12, Figure 5.13, and Figure 5.14. PQ is initialized to all '0' and CQ is initialized to all '1'. Because the input data of PQ is always given to '1', it is easy to check data movement in PQ.

Figure 5.10 shows the movement of data in both PQ and CQ after the system is started. Figure 5.11 shows that the generation of the keystream is done. The keystream generated is then XORed with plaintext to produce ciphertext until the counter counts to 128 in Figure 5.12. The *eq* signal turns to '1' in Figure 5.13. It is implied that the current process is stopped and the new key will be generated by the block cipher. The new keystream is generated in Figure 5.14. More waveforms are shown in Appendix B.

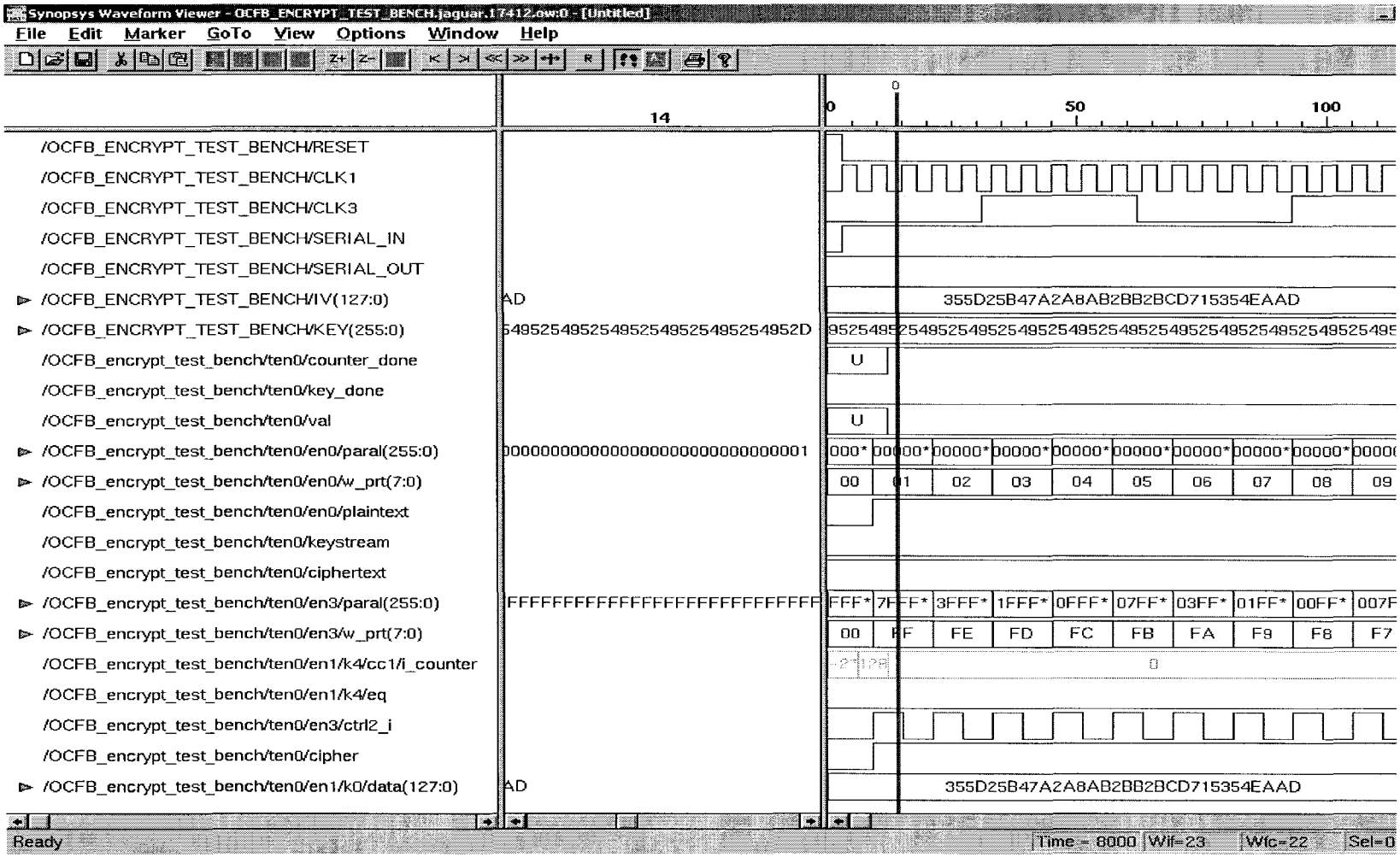


Figure 5.10 Simulation waveform of the encryption system

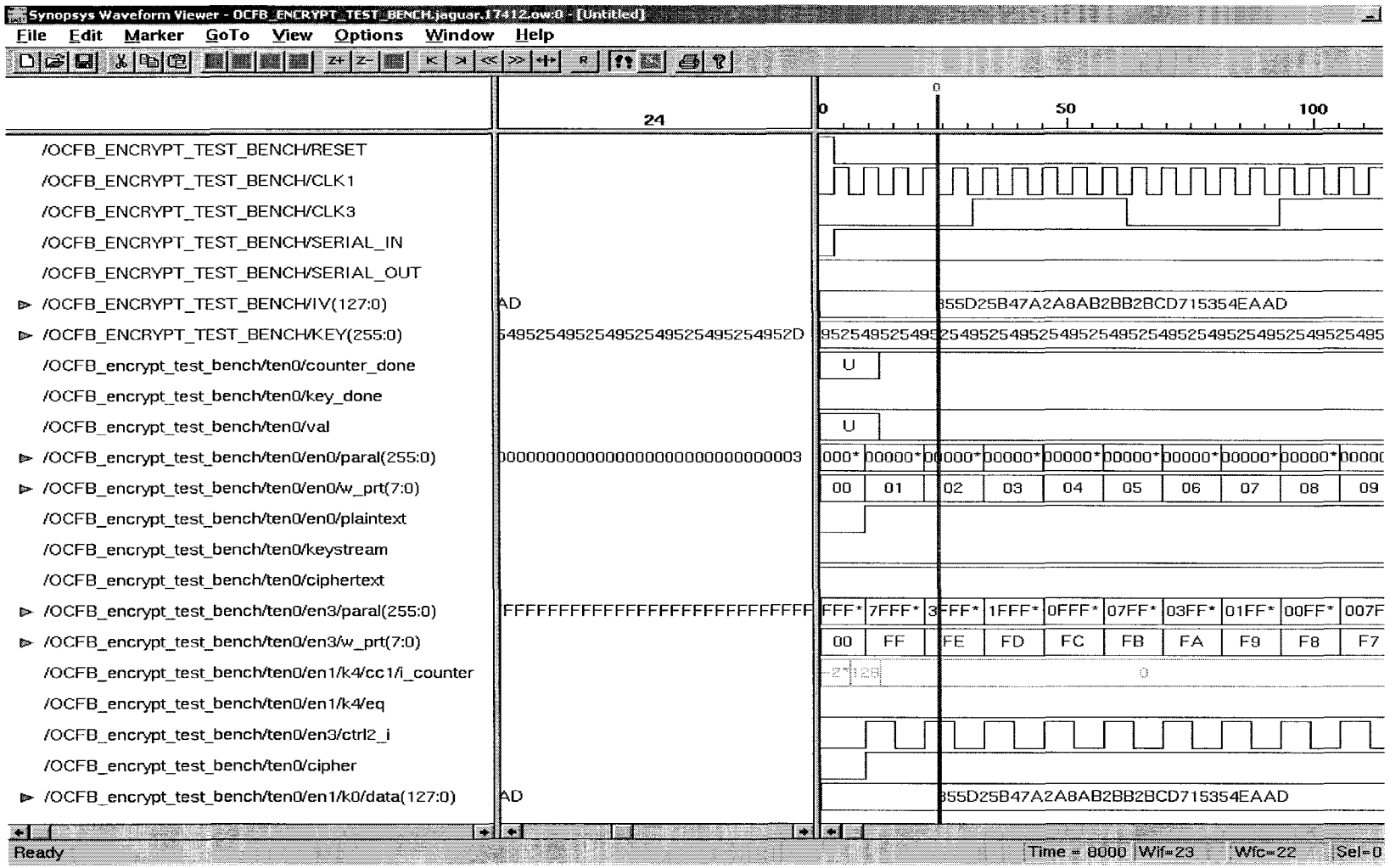


Figure 5.11 Simulation waveform of the encryption system

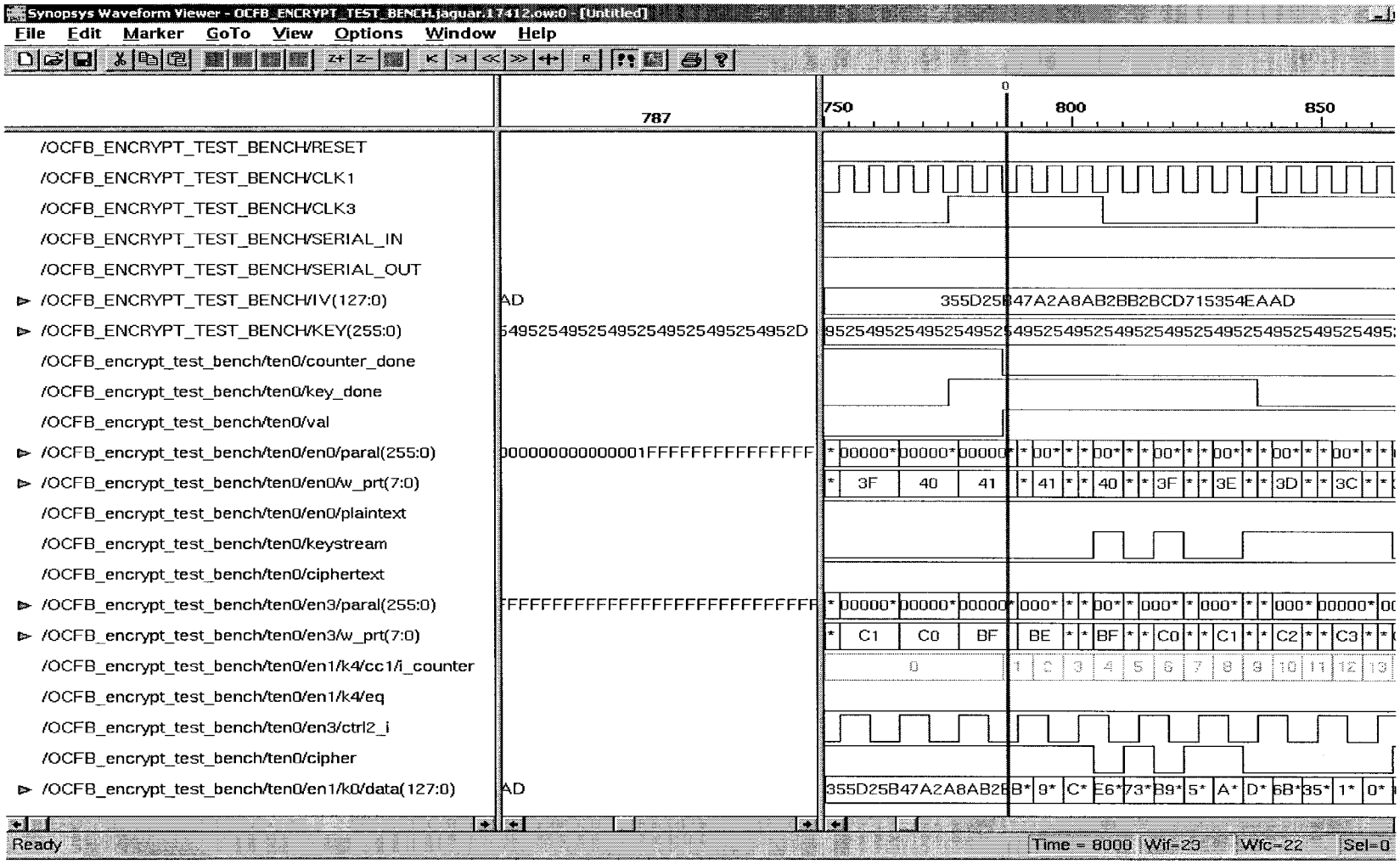


Figure 5.12 Simulation waveform of the encryption system

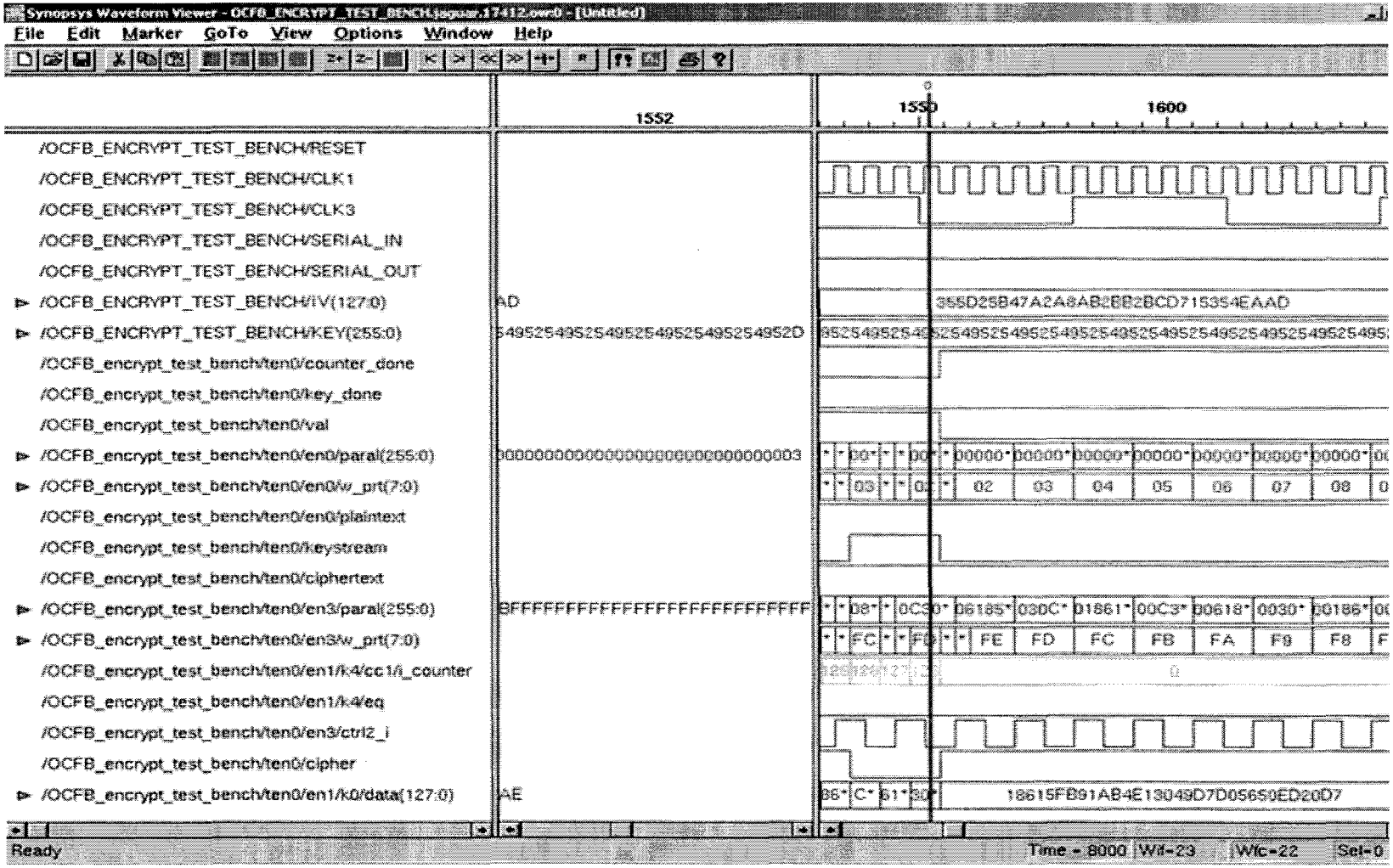


Figure 5.13 Simulation waveform of the encryption system

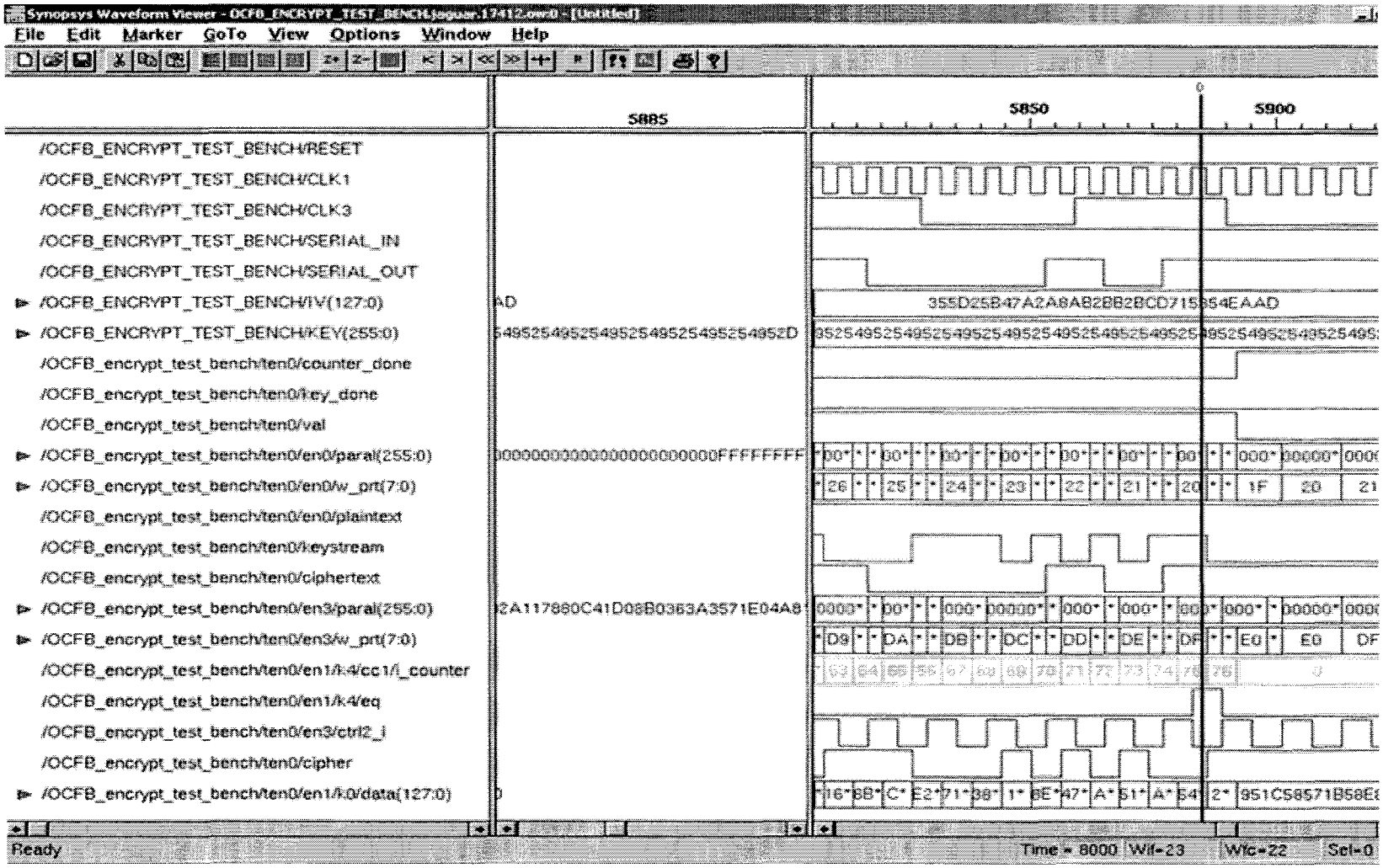


Figure 5.14 Simulation waveform of the encryption system

- Decryption System

The decryption system has the same structure as the encryption system except the plaintext rather than ciphertext is shifted into the last bit of *SR1*. The simulation waveforms of the decryption system are shown in Appendix B.

5.2.2.5 Test Methodology

After implementation, it is very important to prove that the implementation is correct. We can use ANSI C / C++ programming to generate random plaintext to store in an input file. A test bench is able to take data from the input file. After encryption, ciphertext data is saved in an output file. The test bench of the decryption system automatically reads data from the output file as input data. It will store the output of the decryption system in another output file. Therefore, the content of the output file of the decryption system can be compared with the input file of encryption system to check for the correctness of implementation.

The result of the decryption of the OCFB mode is shown in Appendix B. It demonstrates that the structure and the method used to implement OCFB mode are correct.

5.2.2.6 Complexity of Hardware Implementation

The hardware implementation of the OCFB mode is implemented using VHDL and synthesized using Synopsis with 0.18 μm CMOS technology. The hardware complexity is collected by design analyzer of the Synopsis tool with the constraint of the system clock frequency of 10 *ns*. The area of synthesized circuits, which is in square

microns (μm^2), is converted to the gate count by using the two-input NAND gate which has the area of 12.197 μm^2 as a basis for comparison.

The resulting synthesized Rijndael algorithm comes from [12]. From Table 5.1, it is obvious that the implementation of Rijndael algorithm occupies over 90% area in an encryption system when the OCFB mode is implemented by serial transfer.

Component Name	Combinational Area	Noncombinational Area	Total Area	Total Gate
Plaintext subsystem	265167	158	265167	21740
Keystream subsystem without Rijndael	20575	28799	49375	4048
Rijndael algorithm				612834
Ciphertext subsystem	264045	1122	265326	21753
Encryption system				660376

Table 5.1 Hardware complexity of the encryption system of OCFB mode

5.2.2.7 Discussion of Other Structures

There are other structures that are suitable for the implementation of the OCFB system. One approach would be to remove the PQ and the CQ and let the incoming rate of data and outgoing rate of data be consistent. A block cipher output must then be generated within one bit time. This would clearly minimize the delay through the system.

However, the block cipher part of the system is in an idle state for most of time. This implies that the block cipher runs at the rate of B times the data rate. The system efficiency becomes $1/B$, which is the same as CFB mode. Hence, this approach has little value.

5.3 Conclusion

This chapter introduces the concepts of OCFB mode and investigates the hardware structure of an OCFB system. In the hardware implementation of OCFB mode, serial transfer is applied to the hardware implementation of OCFB system to simplify the hardware structure. For the investigation of hardware implementation, it is shown that it is practical to achieve the ability of self-synchronization from bit slips or insertions in communication channel and still obtain higher efficiency than CFB mode (50% in our implementation). Hence, OCFB system can be used for high speed network applications.

Chapter 6
Performance Analysis of SCFB and OCFB Modes

This chapter analyzes the performance of SCFB mode and OCFB mode with respect to theoretical efficiency, synchronization recovery delay, error propagation factor, full-queue efficiency, practical system efficiency, and the relationships between efficiency, buffer size, and the probability of buffer overflow [5]. The characteristics of SCFB mode and OCFB mode then are compared.

6.1 Basic Parameters of Performance Analysis

In this section, the concepts of basic metrics of performance analysis are introduced. These metrics indicate the system abilities of efficiency, synchronization recovery, and error recovery.

- Theoretical efficiency

  Theoretical efficiency is defined as [5]:

$$\eta = \lim_{D \to \infty} \frac{D/B}{E\{\#\text{block cipher operation for } D \text{ bits}\}}. \tag{6-1}$$

Here the denominator is the expected number of block cipher operations required for the encryption of D bits. The numerator represents the number of blocks

corresponding to the encryption of D bits. The theoretical efficiency represents a rate at which the stream cipher can encrypt compared with the rate of the block cipher.

For OFB mode, η can be 1 because all B output bits of the block cipher can be used in the stream cipher keystream. For CFB mode, if it is guaranteed to resynchronize from individual bit slips, CFB must have $m = 1$. Then $\eta = 1/B << 1$. That is the reason why the CFB mode is a very inefficient mode and why an investigation of SCFB mode and OCFB mode is of interest.

- Synchronization Recovery Delay (SRD)

The SRD is defined as the expected number of bits between the synchronization loss and resynchronization. It is a measure of the recovery speed from the sync loss. It is worth noting that the SRD does not include the lost bits and there is no explicit specification on the number of bits lost in the slip [5].

- Error Propagation Factor (EPF)

The EPF measures the bit errors on the output of the decryption when a bit error occurs in the communication channel. It is a metric to examine the influence of a bit error on the data recovered. It is defined as the bit error rate of the plaintext recovered by the decryption system divided by a bit error rate in the communication channel [5].

6.2 Performance Analysis of SCFB Mode

The performance analysis on SCFB mode is presented in [5]. However in order to provide the background and to compare it with OCFB mode, it is repeated below.

6.2.1 Theoretical efficiency

The theoretical efficiency is used to indicate that the rate at which the block cipher must operate to avoid data growing without bounds in PQ. For SCFB mode, the ciphertext bits in the communication channel can be categorized as shown in Figure 6.1. In the figure, n represents the n bit sync pattern, B represents the B bit IV and k represents the number of bits following IV until the sync pattern occurs, which is referred to as OFB block. A synchronization cycle consists of the data bits from the beginning of the sync pattern to the beginning of the next sync pattern. Hence, the size of the synchronization cycle is equal to $n+B+k$. Because k is the amount of data before the sync pattern is found, k is a random variable and decided by a probability distribution dependent on the sync pattern used (e.g. 1111..11, or 1000..00, etc.). Assuming that 0 and 1 have equal probability in ciphertext and each n-bit sequence is independent, then the distribution of k is geometric and the probability of a particular sync pattern is $1/2^n$.

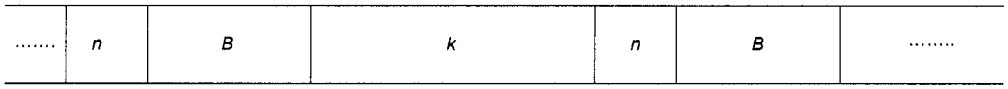


Figure 6.1 Synchronization cycle

Strictly, the distribution of k is related to the sync pattern used (e.g. 11...11, or 10...00, etc) and not the geometric distribution because each n-bit sequence is not independent but overlaps n-1 bits of adjacent sequences when checking for the sync pattern. However we use the geometric distribution for k as an approximation [5].

Hence, the probability of a particular n-bit sync pattern is $1/2^n$ and the probability distribution of k is

$$p(k) = (1-1/2^n)^k \cdot 1/2^n . \tag{6-2}$$

As a result, the expected value of k is

$$E(k) = 2^n - 1. \tag{6-3}$$

and the second moment of k is

$$E\{k^2\} = 2^{2n+1} - 3 \cdot 2^n + 1. \tag{6-4}$$

The expected synchronization cycle size is

$$\mu = n + B + 2^n - 1. \tag{6-5}$$

It is always true that $n+B+k = \alpha B+\delta$ where α and δ are integers and $\delta < B$. Hence, the block cipher has to run $(\alpha + 1)$ block encryptions to produce enough keystream to encrypt $\alpha B+\delta$ plaintext bits and the running rate of the block cipher should be greater than the rate of straight block encryption. From the equation (6-1), we can get

$$\eta = \frac{E\{\text{sync cycle size}\}/B}{E\{\#\text{block cipher operations per sync cycle}\}}. \tag{6- 6}$$

This leads to

$$\eta = \frac{\mu / B}{\sum_{k=0}^{\infty} p(k) \cdot \lceil (k+n+B)/B \rceil}. \tag{6- 7}$$

By deduction, we can get that η becomes a function of n and B [5]:

$$\eta = \frac{\mu / B}{2 + \frac{(1-1/2^n)^{B-n+1}}{1-(1-1/2^n)^B}}. \qquad (6\text{-}8)$$

Figure 6.2 shows the relationship between sync pattern size n and theoretical efficiency η with 64, 128, and 256 bits block. It is obvious that all the theoretical efficiencies are greater than 50%. This is because at least one full block is used in each synchronization cycle since B bits are associated with the IV. Therefore, the theoretical efficiency of SCFB mode is much better than CFB mode. With the increases of the sync pattern size, the theoretical efficiency gets larger. For larger n, the stream cipher can be run at a rate very close to the rate of straight block encryption as the theoretical efficiency approaches 1. Because large n has lower the occurring probability than small n, most bits in the synchronization cycle belong to OFB block. This causes the SCFB mode with large n mainly running as OFB mode. Hence, SCFB mode with larger n can attain much higher theoretical efficiency. The graph also demonstrates that the theoretical efficiency is lower when block length B is larger. The efficiencies are still much higher than conventional CFB mode and are close to 1.

6.2.2 SRD

SRD of SCFB mode is tested by experiment and the result is shown in Figure 6.3 which is plotted as the logarithm base-2 of the SRD. From the figure, it can be seen that SRD is approximately 2^n for large n ($n \geq 10$). For small n ($n \leq 4$), there is a situation that the probability of a slip occurring near the end of the OFB block is much higher than the

case of $n \geq 4$. Hence, it is more likely that the receiver will interpret the next valid sync pattern bits as part of the false IV and will ignore them. As a result, resynchronization will be delayed until the next proper sync pattern. If misinterpretation happens several times, it will cause higher SRD. Unfortunately as can be seen from the figure, this phenomenon is prevalent for small n. The experimental results are obtained under the condition that the sync pattern chosen is of the form of 100...00, 10^9 bits are encrypted and the probability of a bit slip rate is 10^{-5}. The Rijndael algorithm is used as the block cipher with a block length of 128 bits.

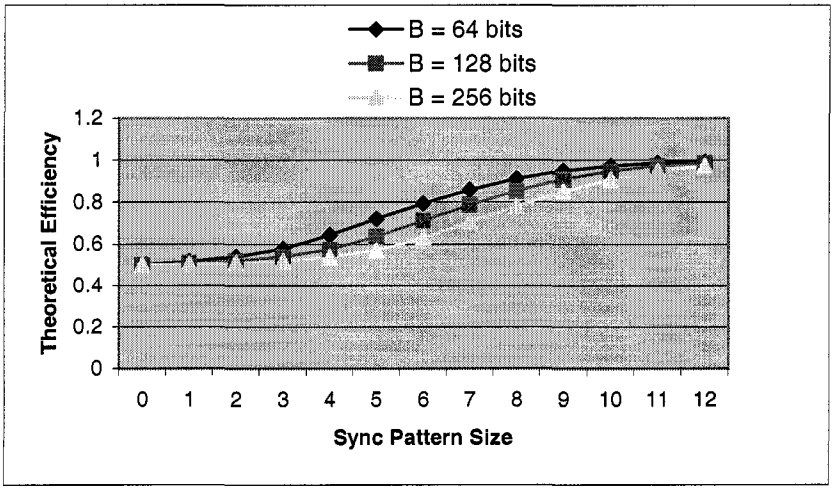


Figure 6.2 Theoretical efficiency vs. sync pattern size

6.2.3 EPF

In this section, the characteristics of EPF are examined. Figure 6.4 illustrates the experimental results which assume that the error probability is 10^{-5}, 10^9 bits are encrypted in the experiment and errors are randomly generated in the communication channel. It is clear that EPF is a function of n with fixed block length. The position in which an error

occurs in the synchronization cycle has a great influence on EPF [5]. The two basic cases are:

- If the error happens in $n+B$ bits of sync/IV block, the synchronization will be lost and $(n+B+k) / 2$ bits are the expected error.
- If the error happens in k bits of the OFB block and no false sync occurs, only one bit error can be caused at output of decryption.

When the value of n is small, there is a higher possibility that the error will happen in $n+B$ bits necessary to cause missing of the sync pattern or the incorrect IV. That will cause EPF to be higher when n is small. As n is getting larger, most bits in the synchronization cycle belong to OFB block and most errors happen in the OFB block, which makes EPF relatively lower. EPF of SCFB mode is on the same order as for CFB mode (EPF = $B / 2$) and is higher than that of OFB mode (EPF = 1).

6.2.4 Practical Efficiency of SCFB Mode

The theoretical efficiency is an ideal efficiency and represents the upper bound of efficiency [5]. In reality, the addition of queuing and the method of implementation have a great influence on system efficiency. "Full-queue efficiency" represents the system efficiency while PQ has data to process. It is the efficiency at which the system operates at its peak rate. Assume α is the full queue efficiency and R is the rate at which bits enter the PQ in bits per unit time, the system will remove bits from the queue in blocks of B bits at a rate of $(1/\alpha) \times R/B$ blocks per second if there are more than B bits in the queue. 50% efficiency implies that for every B bits entering the queue, $2B$ bits are removed from the queue to XOR with the keystream.

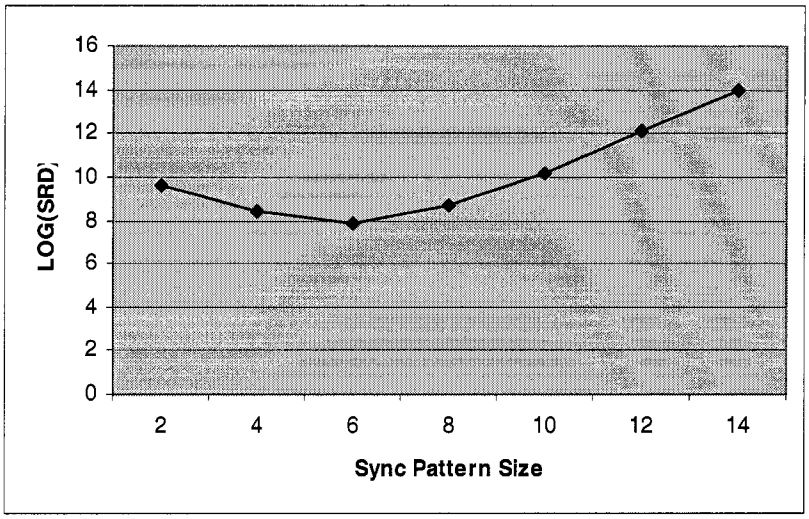


Figure 6.3 Synchronization recovery delay vs. sync pattern size with B = 128

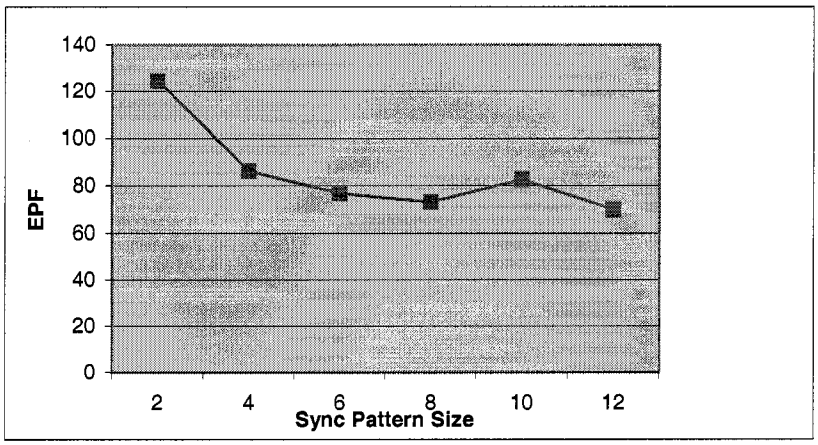


Figure 6.4 Error propagation factor vs. sync. pattern size with B = 128

"Average efficiency" is greater than full-queue efficiency. Because the average efficiency represents the average rate at which system operates including the period that the PQ has fewer than *B* bits and the system has to wait and the period that the PQ has more than *B* bits. If the system is stable, the full-queue efficiency must be less than the average efficiency and the average efficiency must be less than the theoretical efficiency [5].

6.2.5 Relationship Between Buffer Size and Overflow Probability

In this section, the requirement on buffer size in the practical implementation of SCFB mode is investigated experimentally. The experiment is based on the Rijndael algorithm and the 8-bit sync pattern of "10000000". Figure 6.5 shows that the probability of overflow is a function of buffer size with 78.1%, 84.4%, and 90.6% full-queue efficiency, respectively. In theory, the system can attain the efficiency of 91.1% but it is clearly seen that the system will suffer from significant buffer overflow even for a buffer size up to 512 bits.

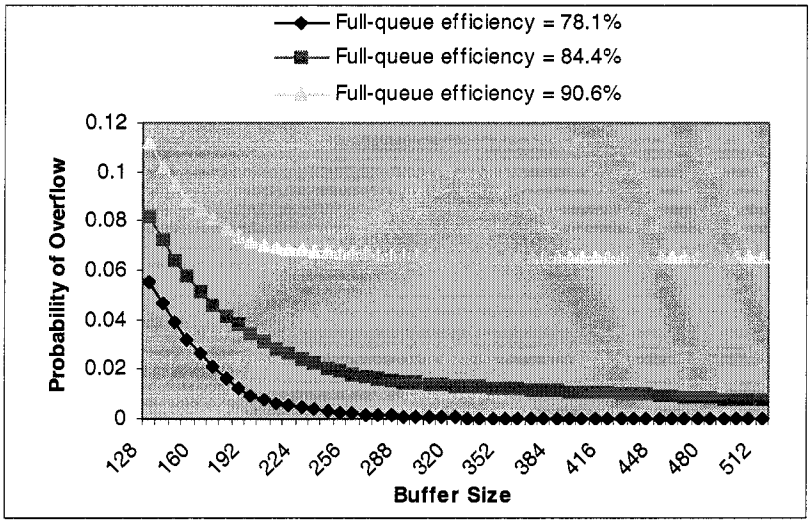


Figure 6.5 Probability of overflow vs. buffer size with Rijndael

It is noteworthy that the system will not overflow for a buffer size of B with 50% efficiency. With this efficiency the block cipher is being run at two times the rate of incoming bits of PQ. Hence, even for the worst case that block cipher needs two complete encryptions to finish the encryption of $B + l$ $(n \leq l \leq B)$ bits when sync pattern is found. Hence, it is guaranteed that no overflow will occur for 50% efficiency with B bit buffers and delay exactly B bits times.

6.2.6 Relationship Between Encryption Efficiency and Overflow Probability

Figure 6.6 shows that the probability of overflow is a function of the full-queue efficiency with fixed buffer size. The probability of overflow increases dramatically with the increase of the full-queue efficiency. In the figure, the buffer size is fixed at 192, 224, or 256 bits and the block cipher uses the Rijndael algorithm. As expected the probability of overflow will get higher as the buffer size is smaller.

6.3 Performance Analysis of OCFB Mode

6.3.1 Theoretical Efficiency

In this thesis, the efficiency of OCFB mode does not adopt the definition in [13]. Because the efficiency of OCFB mode needs do comparison with the result of SCFB mode, I adopted the definition of the efficiency that [5] provided. The differences of these two efficiencies lie on that the efficiency in [13] is absolute efficiency and the efficiency in [5] is comparative efficiency.

For OCFB mode, the ciphertext transmitted in the communication channel can be categorized as illustrated in Figure 6.7. In the figure, n represents the size of the sync pattern and k represents the length of data without the sync pattern. A synchronization cycle is referred to as a set of bits from the beginning of the sync pattern to the beginning of the next sync pattern. Because OCFB mode checks for the sync pattern at anytime, there is no B-bit new IV block in the synchronization cycle as in SCFB mode. Hence, a synchronization cycle consists of $n + k$ bits.

Because k is the amount of data before the sync pattern is found, k is a random variable decided by the probability distribution dependent on the sync pattern used (e.g. 1111..11, or 1000..00, etc.). Assuming that 0 and 1 have equal probability in ciphertext and each n-bit sequence is independent, then the distribution of k is geometric and was already given in the equation (6-2). The corresponding expect value of k and the second moment of k have given in the equation (6-3) and (6-4). The probability of a particular sync pattern is $1/2^n$.

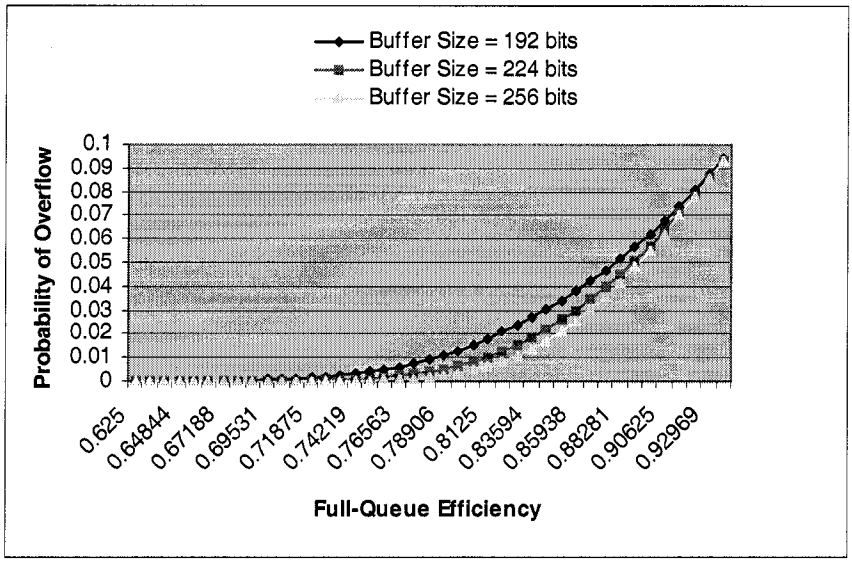


Figure 6.6 Probability of overflow vs. efficiency with Rijndael

.....	n	k	n	k	.....

Figure 6.7 Synchronization cycle of OCFB mode

Strictly, the distribution of k is not the geometric distribution because each n-bit sequence is not independent but overlaps n-1 bits of adjacent sequences when checking

for the sync pattern. However we use the geometric distribution for k as an approximation. Hence, the expected synchronization cycle size μ is:

$$\mu = n + 2^n - 1 \tag{6-9}$$

From the basic definition of theoretical efficiency (6-1), the equation below can be derived

$$\eta = \frac{E\{\text{sync cycle size}\}/B}{E\{\#\text{block cipher operations per sync cycle}\}}. \tag{6-10}$$

It is easy to deduce that:

$$\eta = \frac{\mu/B}{\sum_{k=0}^{\infty} p(k) \cdot \lceil (k+n)/B \rceil} \tag{6-11}$$

Following the approach in [5], the following equation can be deduced

$$\begin{aligned} \eta &= \frac{\mu/B}{1+\frac{(1-1/2^n)^{B-n+1}}{1-(1-1/2^n)^B}} \\ &= \frac{(n+2^n-1)/B}{1+\frac{(1-1/2^n)^{B-n+1}}{1-(1-1/2^n)^B}} \end{aligned} \tag{6-12}$$

From equation (6-12), it can be seen that theoretical efficiency is converted to a function of n and B. This equation is plotted in Figure 6.8 with the block size of 64, 128, and 256 bits. It is obvious that the theoretical efficiency increases with the increase of the sync pattern size. For large n, the efficiency can approach 100% implying that OCFB

mode can obtain high efficiency. However, for small n, the efficiency is close to 0, which results from the system continuously resynchronizing.

If the size of the sync pattern is larger than 4 whenever B is 64, 128 or 256 the theoretical efficiency is definitely higher than the efficiency of CFB mode. For n = 8, the theoretical efficiencies are 89.1%, 79.6%, 64.4% for 64, 128, and 256 bit blocks, respectively. This is much better than the efficiency of CFB mode which is 1/128 and is approaching the efficiency of 100% for OFB mode.

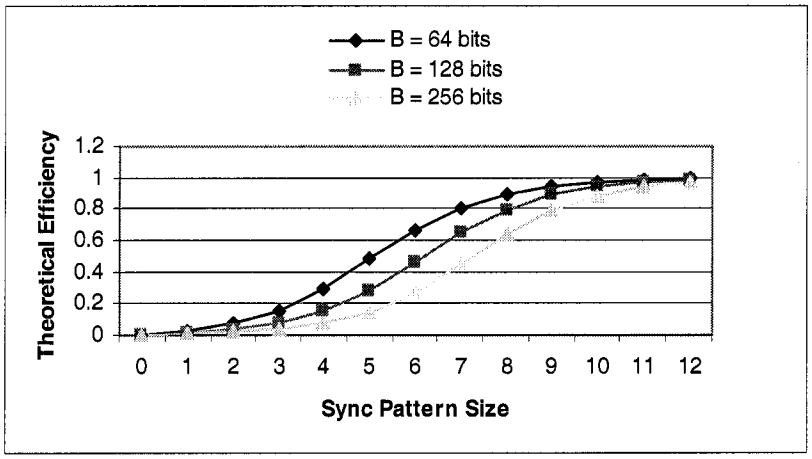


Figure 6.8 Theoretical efficiency vs. sync pattern size

6.3.2 SRD

The resynchronization properties of OCFB mode are examined experimentally. As described above, SRD is used to measure the speed of recovery from bit slips. The relationship between SRD and sync pattern size is illustrated in Figure 6.9 using a plot of the logarithm base-2 of SRD.

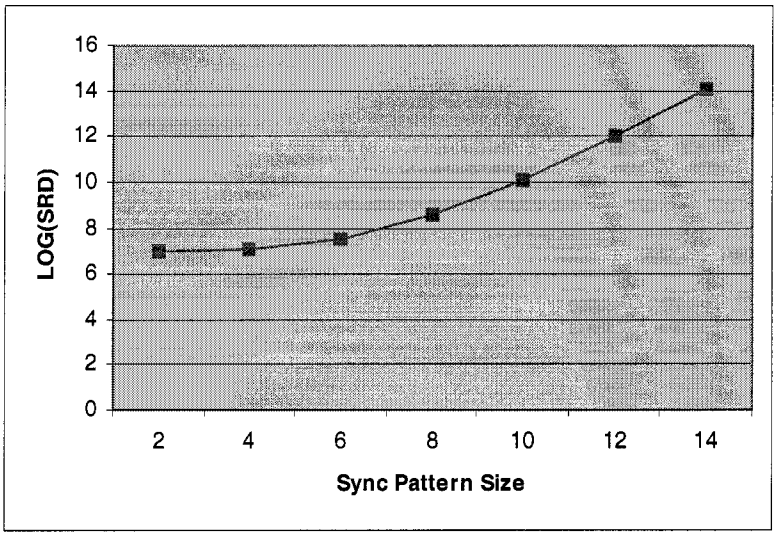


Figure 6.9 Sync Recovery Delay vs. Sync Pattern Size (B=128)

The simulation is run under the condition that the probability of a bit slip is 10^{-5} and 10^9 bits are encrypted. It is assumed that there are no multiple bit slips. In the figure, the SRD is increased exponentially with the increase of n. SRD is approximately 2^n when $n \geq 10$. When $n < 10$, SRD is larger than 2^n. This is because the sync pattern with small n has higher frequency in ciphertext data than large n. The probability that bit slips occur in the sync pattern to cause missed synchronization is much higher than the cases with large n. But compared with SCFB mode, SRD of OCFB mode with small n is lower.

6.3.3 EPF

It is important to analyze the variety of situations where errors occur before the effect of error propagation factor (EPF) to the communication system is analyzed. There are two likely situations according to the effect of error.

- If the error happens in sync pattern, it will cause missing synchronization until the next correct sync pattern occurs. The average number of bit errors are $(n+k)/2$.
- If the error happens not in sync pattern but no false sync pattern is generated, it will cause only one bit error at the receiver.

The error propagation factor is examined experimentally and the results are shown in Figure 6.10. The simulation results are obtained with $p_e = 10^{-5}$. In this figure, EPF becomes a function of n when $B = 128$. There are two main factors to affect the value of EPF. One is the probability of the sync pattern occurring in ciphertext and the other is the position where error occurs in the synchronization cycle. The probability of the sync pattern represents the recovering capability of the system from an error in the sync pattern. The position where error occurs influences on the number of error bits according to the discussion above. When the value of n is small, there is a higher possibility that the error happens in n-bit sync pattern to cause missing of the sync pattern. However, it has stronger capability on recovering from error because the probability of the sync pattern with small n is higher than that of the sync pattern with larger n. Due to the frequent resynchronization for the system with small n, EPF is reduced. As n is getting larger, the decreasing of the probability of the sync pattern reduces the capability of the system resynchronization, although the probability that error occurs in the sync pattern is also decreased. The result is EPF is generally about $B / 2 = 64$.

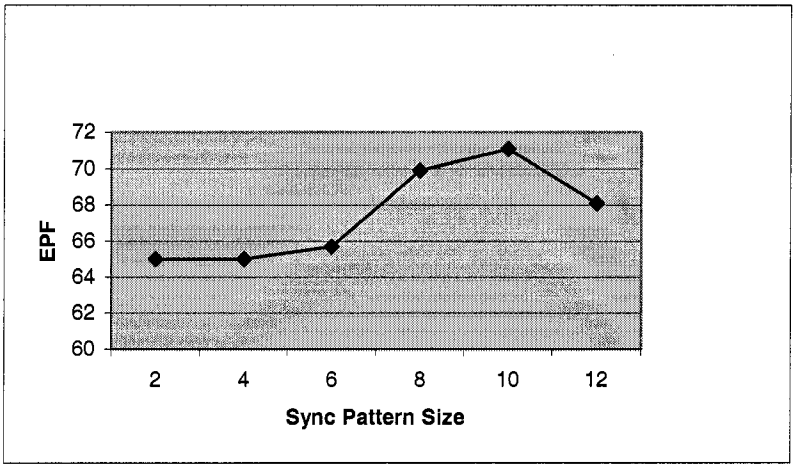


Figure 6.10 Error propagation factor vs. sync pattern size with B = 128

6.3.4 Practical Efficiency of OCFB Mode

Practical efficiency of OCFB mode can be represented by eff = R / R_e where R is the rate of data coming into PQ and R_e is the rate of encryption of the block cipher. The ratio between R and R_e can be adjusted to attain high efficiency. Since R_e is constrained by current technology, R is constrained by R_e. That means that the maximum throughput is constrained by the rate that a block encryption performs. Because of the serial transfer from PQ to CQ, the data rate R' leaving from the PQ needs to be considered. However data leaving PQ does not have regular data transfer rate because bits might be moved for some periods and bit transfer might stall for other periods. On average, data is moved from the PQ to the CQ at a rate less than R'. It makes sense that $R' = R_e$ and $R' > R$. It can be deducted that $R_e > R$. Efficiency might approach 1 if all of outputs of the block cipher can be used to XOR a block of plaintext. However, if it is guaranteed that resynchronization can be regained from bit slips or bit errors, the partial block of the output of the block cipher would be used in the block following the sync pattern.

6.3.5 Relationship Between Buffer Size and Probability of Overflow

The relationship between buffer size and the probability of overflow is investigated based on full-queue efficiency of 78.1%, 84.4%, and 90.6%. The experimental results with Rijndael based on sync pattern "10000000" are shown in Figure 6.11. It is quite straightforward to note that the probability of overflow is lower when the buffer size is larger. It is noteworthy that OCFB mode requires much more buffer size than SCFB mode. From the figure, it can be seen that it is easy to overflow with the high efficiency even with the larger buffer size. In order to achieve the smallest probability of data loss, modest efficiency and larger buffer sizes are recommended for OCFB mode. For lower probability of overflow, OCFB mode still can obtain relatively high efficiency compared with CFB mode.

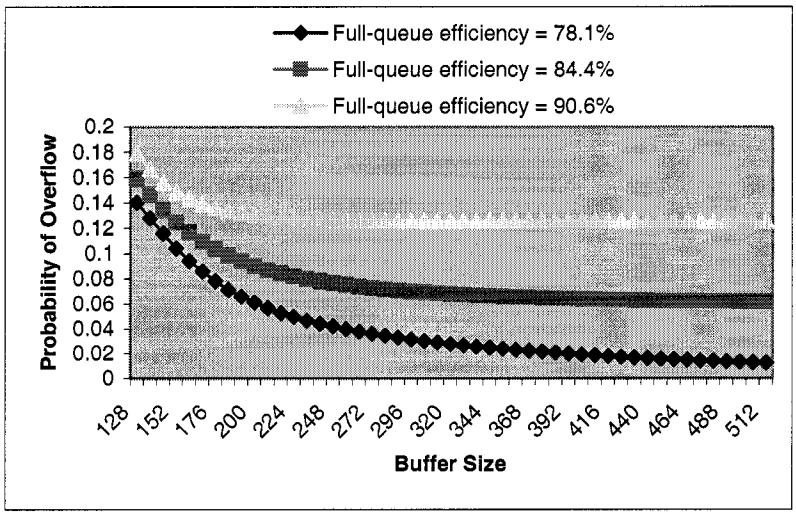


Figure 6.11 Probability of overflow vs. buffer size with Rijndael

6.3.6 Relationship Between Encryption Efficiency and Probability of Overflow

The relationship based on Rijndael between encryption efficiency and probability of overflow is illustrated in Figure 6.12. From this figure, it can be seen that the probability of overflow approaches 0 when efficiency is below 50% whether the buffer size is 192, or 224, or 256. Hence, OCFB mode with high efficiency requires much more buffer size in order to minimize the probability of overflow. The probability of overflow appears to exponentially follow the increase of efficiency. As expected, large buffer size will provide lower probability of overflow for a given efficiency.

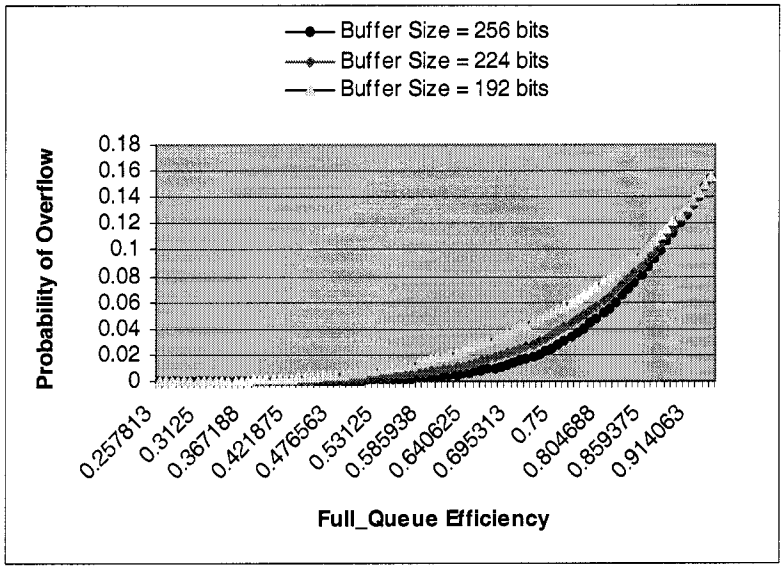


Figure 6.12 Probability of overflow vs. efficiency with B = 128

6.4 Performance Comparison between SCFB Mode and OCFB Mode

6.4.1 Theoretical efficiency

It is known that both SCFB mode and OCFB mode can achieve high efficiency as shown in Figure 6.2 and Figure 6.8. However, the difference between them is obvious. SCFB mode achieves at least 50% theoretical efficiency and highest efficiency can arrive to 100% , but the theoretical efficiency of OCFB mode can vary from 0% to 100%. This difference is decided by the characteristics of these two modes. SCFB mode does not check for the sync pattern in IV after the sync pattern is found. This makes SCFB mode at least use one full block, B bits, in one synchronization cycle, $n+B+k$. OCFB mode checks each n bits for the sync pattern without exception, even in IV. The frequency of resynchronization occurring greatly influences the theoretical efficiency of OCFB mode. If the sync pattern occurs frequently, the system is busy resynchronizing. This makes the theoretical efficiency of OCFB mode decreased and close to 0% for small value of n. However, it is possible to obtain high efficiency for OCFB mode by increasing the size of the sync pattern.

6.4.2 SRD

SCFB mode and OCFB mode have a similar trend shown in Figure 6.13 in SRD when the sync pattern size n is increased. However, SCFB mode has much higher SRD than OCFB mode when $n \leq 6$. This indicates that OCFB mode recovers more quickly from the loss of synchronization because OCFB mode checks all ciphertext for the sync pattern. As mentioned before, the possibility that a bit slip occurs in the first $n+B$ bits of a

sync cycle is higher when n is small than when n is large. This will lead to either missed resynchronization or wrong IV at receiver. Because SCFB mode does not check the sync pattern in IV block after the sync pattern is recognized, this phenomenon results in SCFB mode needing a longer time to recover for small n.

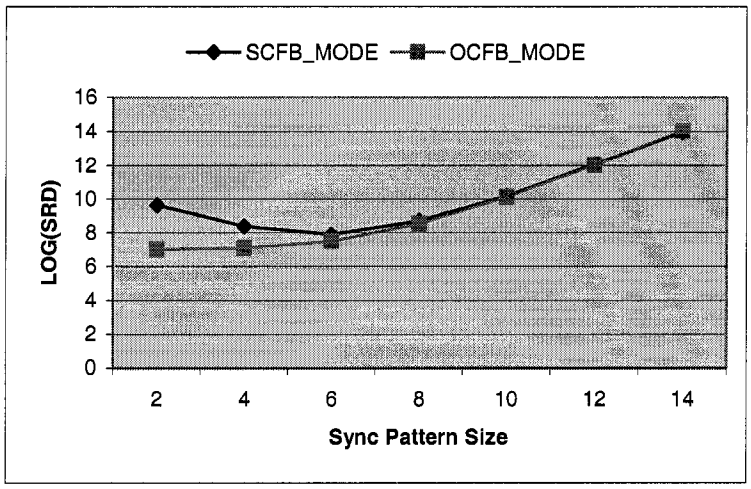


Figure 6.13 Sync recovery delay vs. sync pattern size with B = 128

6.4.3 EPF

From Figure 6.14, EPF of OCFB mode is better than EPF of SCFB mode whether the sync pattern size is small or large. This can be explained by the property of OCFB mode which checks sync pattern in all ciphertext whether or not it is the first $n+B$ bits of a sync cycle. This property gives OCFB mode a fast resynchronization. Another result from Figure 6.14 is that when n is small SCFB mode has a much higher EPF. This is because false synchronization due to bit errors is prevalent when n is small and SCFB will take much longer to resynchronize in these circumstances. Hence, the influence of errors on OCFB mode is smaller than on SCFB mode.

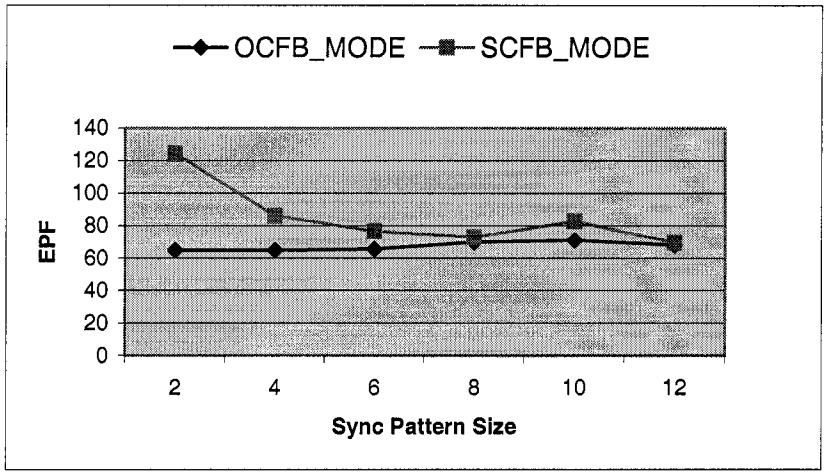


Figure 6.14 Error propagation factor vs. sync pattern size with B = 128

6.4.4 Relationships Between Probability of Overflow and Buffer Size

To clearly compare the relationship between probability of overflow and buffer size of SCFB mode and OCFB mode, Figure 6.15, Figure 6.16, and Figure 6.17, Figure 6.18, are presented.

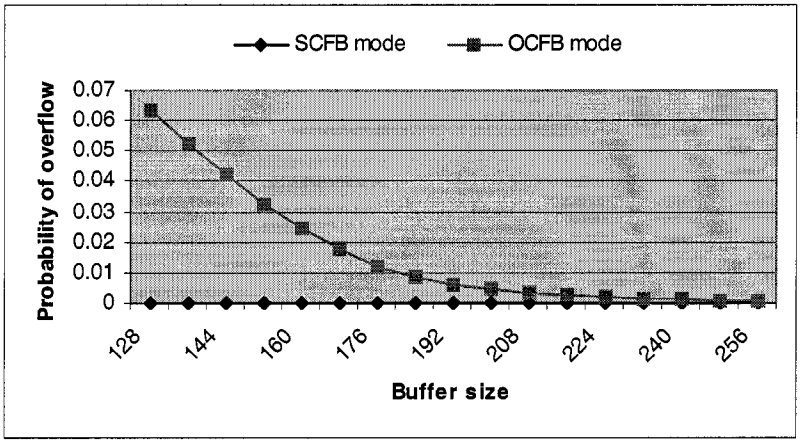


Figure 6.15 Probability of overflow vs. buffer size with full-queue efficiency = 50%

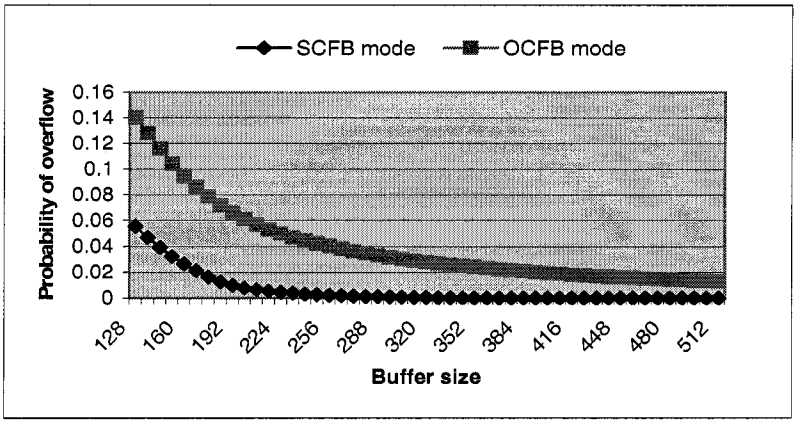


Figure 6.16 Probability of overflow vs. buffer size with full-queue efficiency = 78.10%

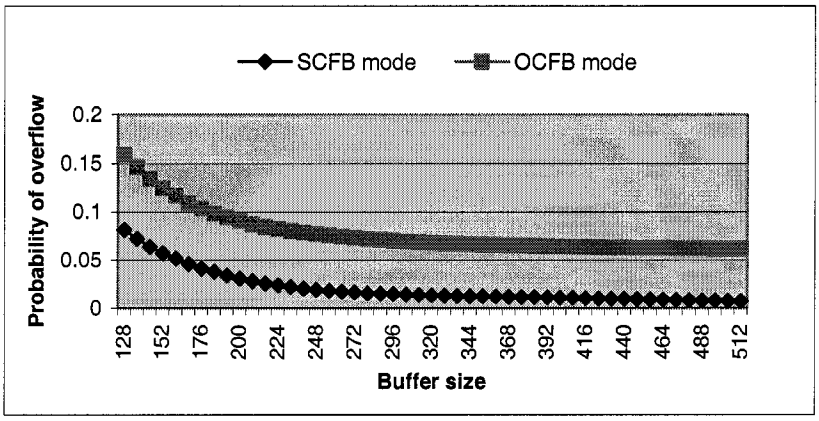


Figure 6.17 Probability of overflow vs. buffer size with full-queue efficiency = 84.40%

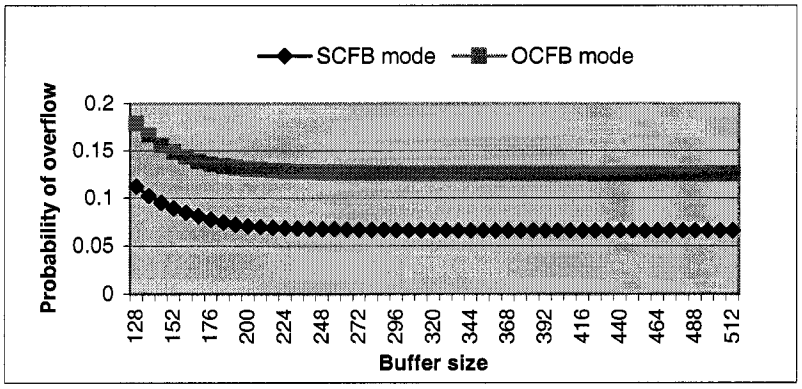


Figure 6.18 Probability of overflow vs. buffer size with full-queue efficiency = 90.60%

As explained previously for SCFB mode, Figure 6.15 tells us that 50% efficiency with B bit buffer size guarantees the system does not have any buffer overflow. It can also be seen from Figure 6.15 that OCFB mode suffers from higher probability of overflow than SCFB mode when full-queue efficiency is 50% and buffer size is 128 bits. The probability of buffer overflow is decreased with the increase of buffer size. When buffer size is 256 bits, the probability of buffer overflow is close to 0.

For the implementation of this thesis, 256 bit buffer size is applied although for SCFB mode at 50%, 128 bits would have been sufficient. There are several considerations. One consideration is that the system efficiency higher than 50% may be implemented. It requires larger buffer size to avoid buffer overflow. The second consideration is that OCFB mode requires more buffer size than SCFB mode under the same condition of efficiency and low buffer overflow. Hence, the selection of 256 bit buffer size is quite reasonable although for SCFB mode only a buffer of 128 bits is required.

Figure 6.16, Figure 6.17, and Figure 6.18 further presents that SCFB mode requires significantly less buffer space than OCFB mode for the same probability of overflow. OCFB mode needs more buffer size because of its tendency towards more frequent resynchronization.

6.4.5 Relationship Between Probability of Overflow and Efficiency

Again, because of the frequent resynchronization of OCFB mode, when the probability of overflow is the same and the buffer size is fixed, the OCFB efficiency is significantly less than SCFB mode as shown in Figure 6.19, Figure 6.20, and Figure 6.21.

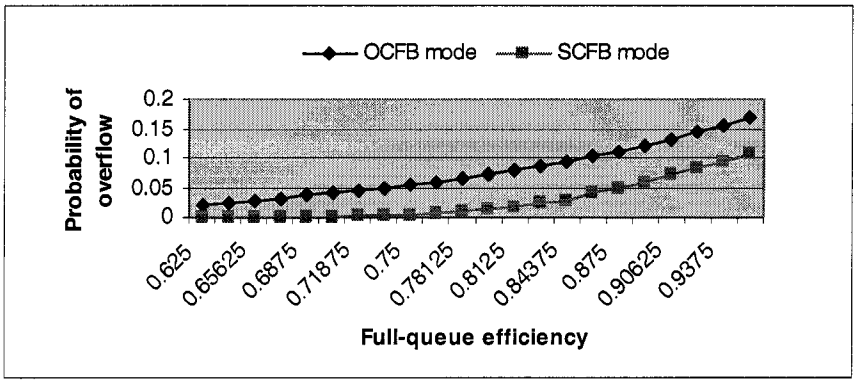


Figure 6.19 Probability of overflow vs. full-queue efficiency with B = 192

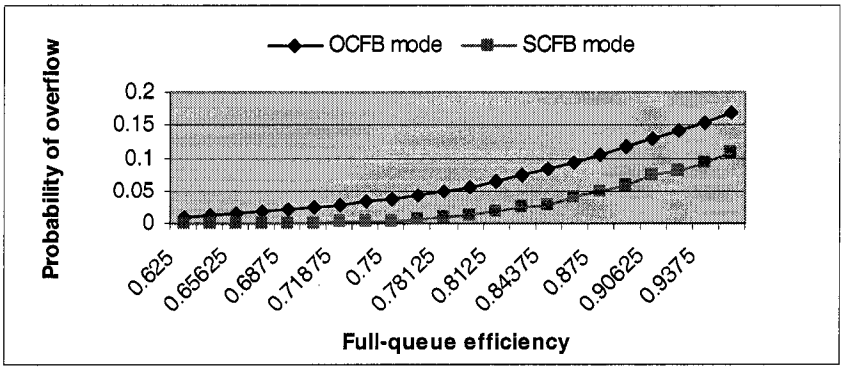


Figure 6.20 Probability of overflow vs. full-queue efficiency with B = 224

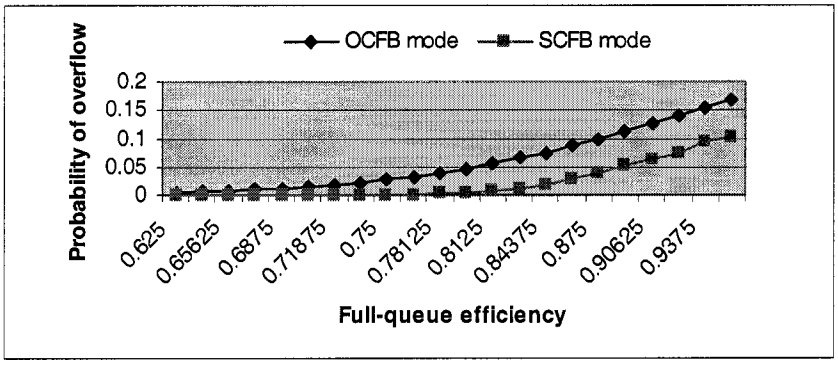


Figure 6.21 Probability of overflow vs. full-queue efficiency with B = 256

6.5 Conclusion

This chapter analyzes the performance of SCFB and OCFB modes with respect to the theoretical efficiency, the synchronization recovery delay, and the error propagation

factor. In addition, the relationships between efficiency, probability of buffer overflow, and buffer size, are investigated. It is definite that both modes can obtain higher efficiency than CFB mode. From the above analysis, it is concluded that OCFB mode can obtain better error propagation and synchronization recovery delay than SCFB mode. However, SCFB mode is able to achieve higher efficiency with a given buffer size and lower probability of buffer overflow than OCFB mode. Furthermore, SCFB mode can guarantee that the system with 50% efficiency and B bit buffer size would have no buffer overflow. OCFB cannot guarantee no overflow for any level of efficiency.

Hence, SCFB mode is more suitable for high speed physical layer security than OCFB mode.

Chapter 7
Conclusions and Future Work

7.1 Summary of the Research

This thesis investigates two modes of operation, Statistical Cipher Feedback (SCFB) mode and Optimized Cipher Feedback (OCFB) mode. These two modes can configure a block cipher to operate as a stream cipher and are categorized as self-synchronizing stream ciphers.

SCFB mode improves efficiency over CFB mode by turning into OFB mode and working as OFB mode most of time. SCFB mode also achieves the ability of self-synchronization by checking for the sync pattern in ciphertext and turning into CFB mode to periodically obtain the IV after the sync pattern is found. SCFB mode is implemented in software to analyze the characteristics of synchronization recovery delay, error propagation factor and the relationship between buffer size, probability of overflow, and full-queue efficiency. In addition, design and hardware implementation of SCFB mode was considered by using the Synopsis tool with 0.18 μm CMOS technology to study the timing issues and hardware properties. In order to compare the performances of SCFB mode and OCFB mode and the influences of the different methods of hardware implementation on SCFB mode and OCFB mode, our SCFB mode implementation adopts a parallel transfer method to encrypt a block of data and then store it into the ciphertext queue. For simplicity, the implementation of SCFB mode in this thesis only

has approximately 10% efficiency. The key generation part is sitting in an idle state most of time due to the usage of the same clock in the system to drive the block cipher and data transfer. This situation can be changed by using different clocks to drive the block cipher and the data transfer into and out of the system. Since, for a higher efficiency, the block cipher clocking must be slower, it is simple to derive a slow block cipher clock from the data link clock, thereby increasing efficiency. Thus the time of generating the keystream can be extended which will shorten the idle time and improve the efficiency of whole system. Parallel transfer would allow implementation of efficiency > 50%. The penalty with parallel transfer, however, is that a large amount of hardware is required.

OCFB mode utilizes all of the output of the block cipher to XOR plaintext data to produce ciphertext to attain high efficiency and recognizes the sync pattern in ciphertext to synchronize the encryption system and the decryption system. The analysis of OCFB mode adopts the same analysis method as SCFB mode. OCFB mode has been implemented by software to study the performances of SRD, EPF, and the relationships between buffer size, full-queue efficiency and probability of overflow. Further OCFB mode has been designed and simulated in hardware by using the Synopsis tool with 0.18 μm CMOS technology. The method of hardware implementation on OCFB mode uses the serial transfer from the PQ to the CQ. Serial transfer has relatively simple hardware components but suffers from difficulties on the timing relationship between key generation part, PQ, and CQ. The method of coordinating these three clock frequencies to improve the system efficiency becomes an important part in the implementation of OCFB mode. In this study, OCFB mode with 50% efficiency is implemented in hardware.

The analysis of the performances of OCFB mode and SCFB mode reveal that under the same probability of overflow SCFB mode requires significantly less buffer size compared with OCFB mode. With the same buffer size and probability of overflow, SCFB mode can achieve higher efficiency than OCFB mode. However, due to the characteristic that SCFB mode does not check for the sync pattern in the B-bit IV collection phase, SCFB mode has the relatively large SRD compared with OCFB mode. As well, SCFB mode has a marginally higher EPF than OCFB mode. SCFB mode can obtain at least 50% theoretical efficiency without any buffer overflow and up to close to 100% efficiency with some buffer overflow. OCFB mode can achieve the efficiency from 0 to approximately 100% but always suffers from some buffer overflow.

The parallel implementation method of SCFB mode provides the simple timing relationship and the lower requirement on buffer size but has complicated hardware implementation. Serial implementation method of OCFB mode simplifies hardware structure but increases the complexity on timing issues and constrains efficiency to no higher than 50%.

From the analysis and comparison of SCFB mode and OCFB mode, it can be concluded that SCFB mode and OCFB mode are quite similar modes except that OCFB mode checks the sync pattern all the time without any exception while SCFB mode does not check the sync pattern during the collection of the B-bit IV. Another difference between these two modes is that OCFB mode always accepts ciphertext as the input of the block cipher but SCFB mode turns into CFB mode from OFB mode according to the occurrence of the sync pattern. This property of SCFB mode gives an attacker an

opportunity due to the birthday paradox because there is a possibility that IV is the same as one of the outputs of block cipher when SCFB mode works as OFB mode. However, this possibility is virtually negligible for a large block size such as 128 bits [5].

7.2 Suggestion for Future Work

A number of directions can be taken for future work. SCFB mode can be implemented using serial transfer and the difference from parallel transfer can be compared in the future. The hardware implementation efficiency of SCFB mode can be improved by giving different clocks to drive data transfer and the block cipher. OCFB mode can be implemented using parallel transfer and the difference from serial transfer can be compared in the future. The hardware structures of SCFB mode and OCFB mode can be optimized and the area utilization can be improved by adding more constraints. Further hardware synthesis for these two modes can be done to fulfill the work of placing and routing in a real VLSI device. FPGAs technologies today provide flexible design, cost effective, reprogrammed capability compared with traditional fixed-logic ASICs. These features make FPGAs technology become key system-level technology. SCFB mode and OCFB mode can be implemented in FPGAs. Test on the real chip is required as the important step for the chip design. As well, the current used mode in reality and other new modes can be compared with SCFB mode and OCFB mode to select a better mode which can be applied for physical layer of high speed network.

References

[1] National Institute of Standards and Technology, "Data Encryption Standard (DES)", *Federal Information Processing Standard 46, 1997*

[2] National Institute of Standards and Technology, AES web site: csrc.nist.gov/encryption/aes

[3] W. Stallings, *Cryptography and Network Security*, Prentice Hall press, second edition, 2003

[4] A.J. Menezes, P.van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997

[5] H. M. Heys, "Analysis of the Statistical Cipher Feedback Mode of Block Ciphers," *IEEE Transactions on Computer*, vol.52, no. 1, pp. 77-92, Jan. 2003

[6] O. Jung, S. Kuhn, C. Ruland, and K. Wollenweber, "Enhanced Modes of Operation for the Encryption in High-Speed Networks and their Impact on QoS," *ACISP 2001*, pp. 344-359

[7] J. Pitman, *Probability*, Springer-Verlag press, 1993

[8] Canadian Microelectronics Corporation, *Tutorial on CMC's Digital IC Design Flow*, October 2002, Document ICI-096

[9] Y. EI-Sayed, *Performance Analysis, Design and Reliability of the Balanced Gamma Network*, Ph.D. thesis, Memorial University, 1999

[10] O.Jung and C. Ruland, "Encryption with Statistical Self-Synchronization in Synchronous Broadband Networks," *Cryptographic Hardware and*

Embedded Systems – CHES'99, Lecture Notes in Computer Science 1717, Springer-Verlag, pp. 340-352, 1999

[11] F. Yang, H. M. Heys, "Implementation of Statistical Cipher Feedback Mode", *NECEC conference*, Dec. 2002

[12] T. Ichikawa, T. Kasuya, and M. Matsui, "Hardware evaluation of the AES finalists," in Proc. 3rd *AES Candidate Conference*, New York, pp. 279—285, Apr. 2000

[13] A.Alkassar, A. Geraldy, *B.* Pfitzmann, A-*R.* Sadeghi, "Optimized Self-Synchronizing Mode of Operation," *Fast Software Encryption Workshop – FSE 2001*, Yokohama, Japan, Apr. 2001.

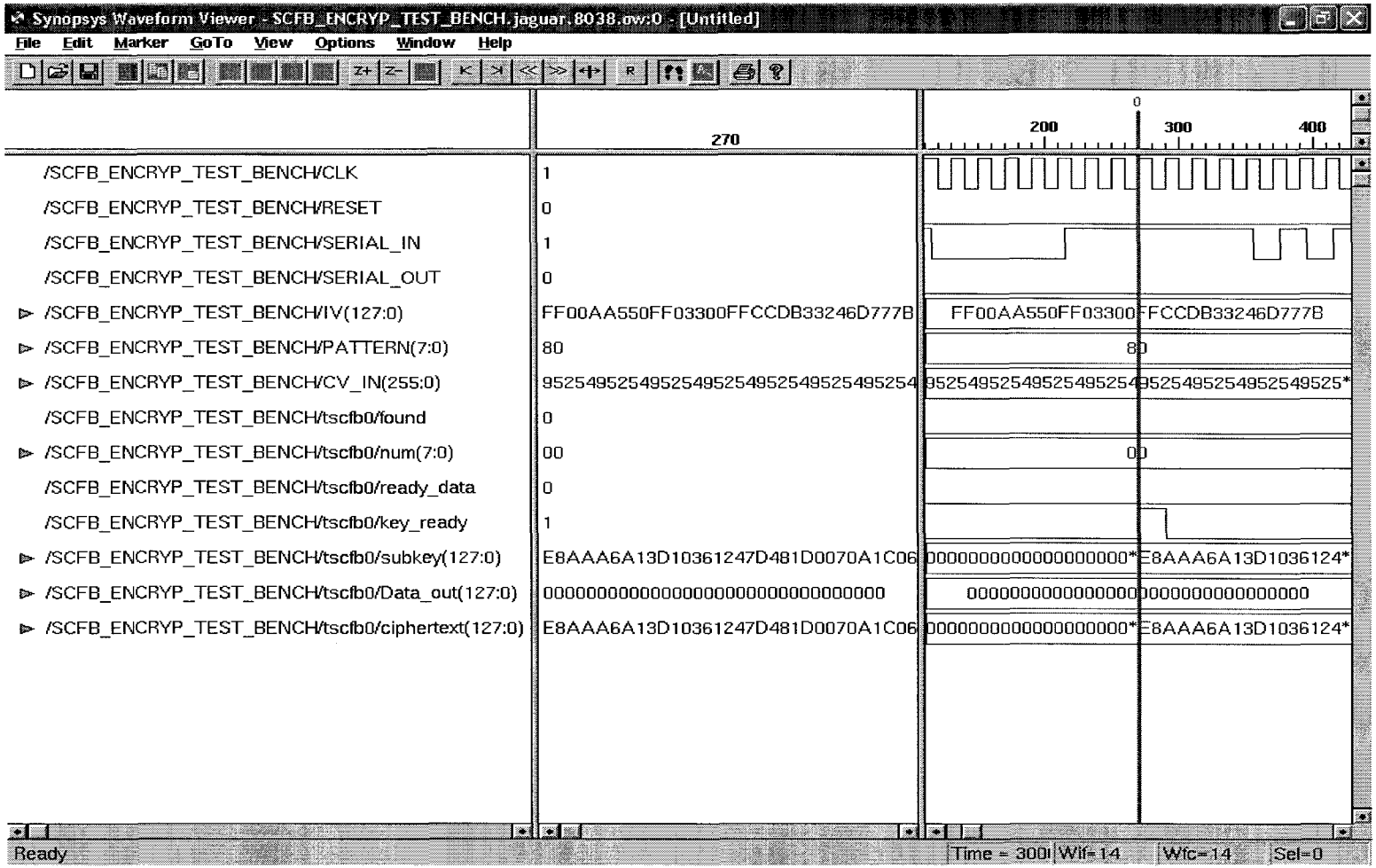
Appendix A

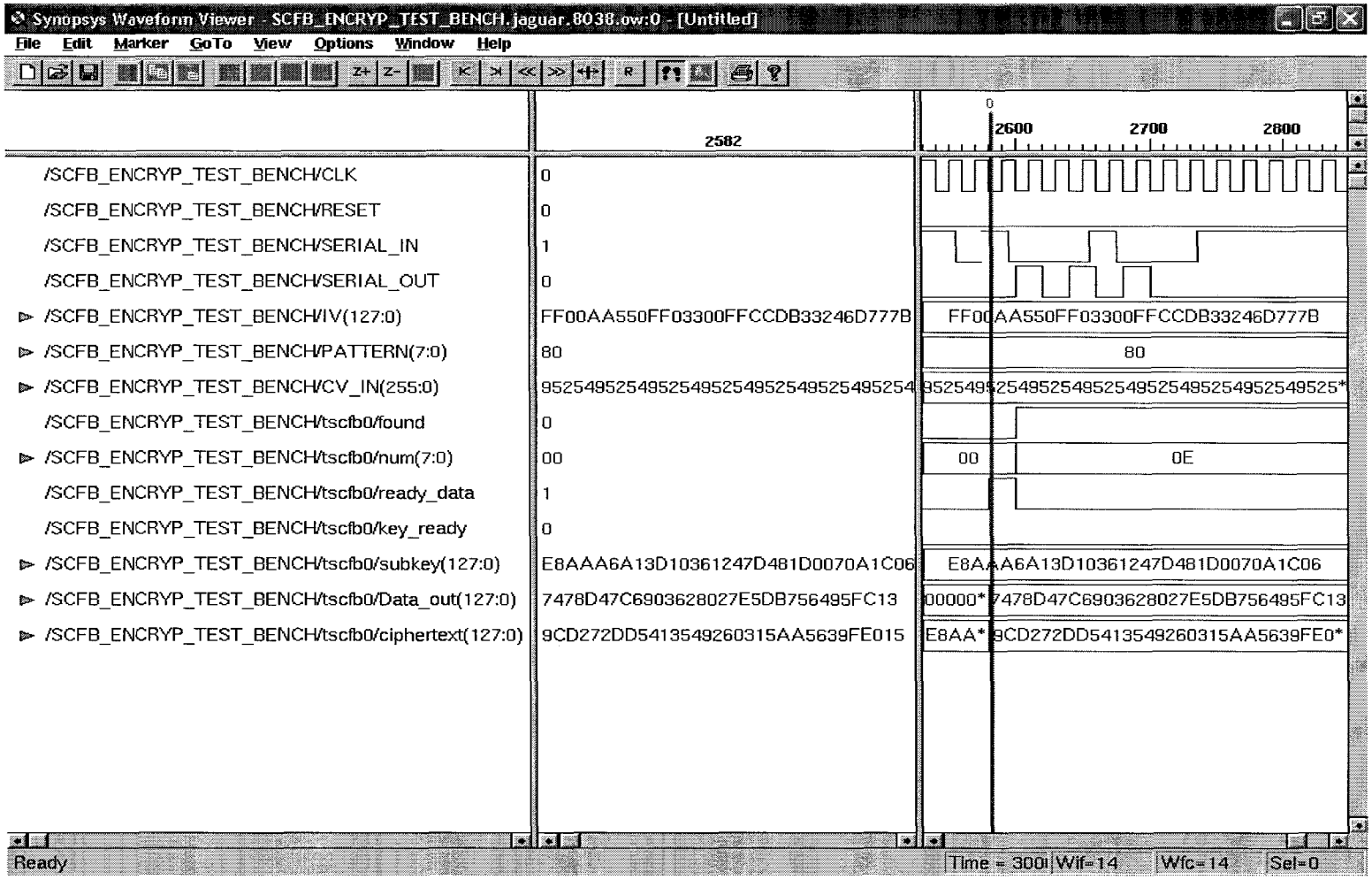
Waveforms of the Hardware Implementation of SCFB Mode

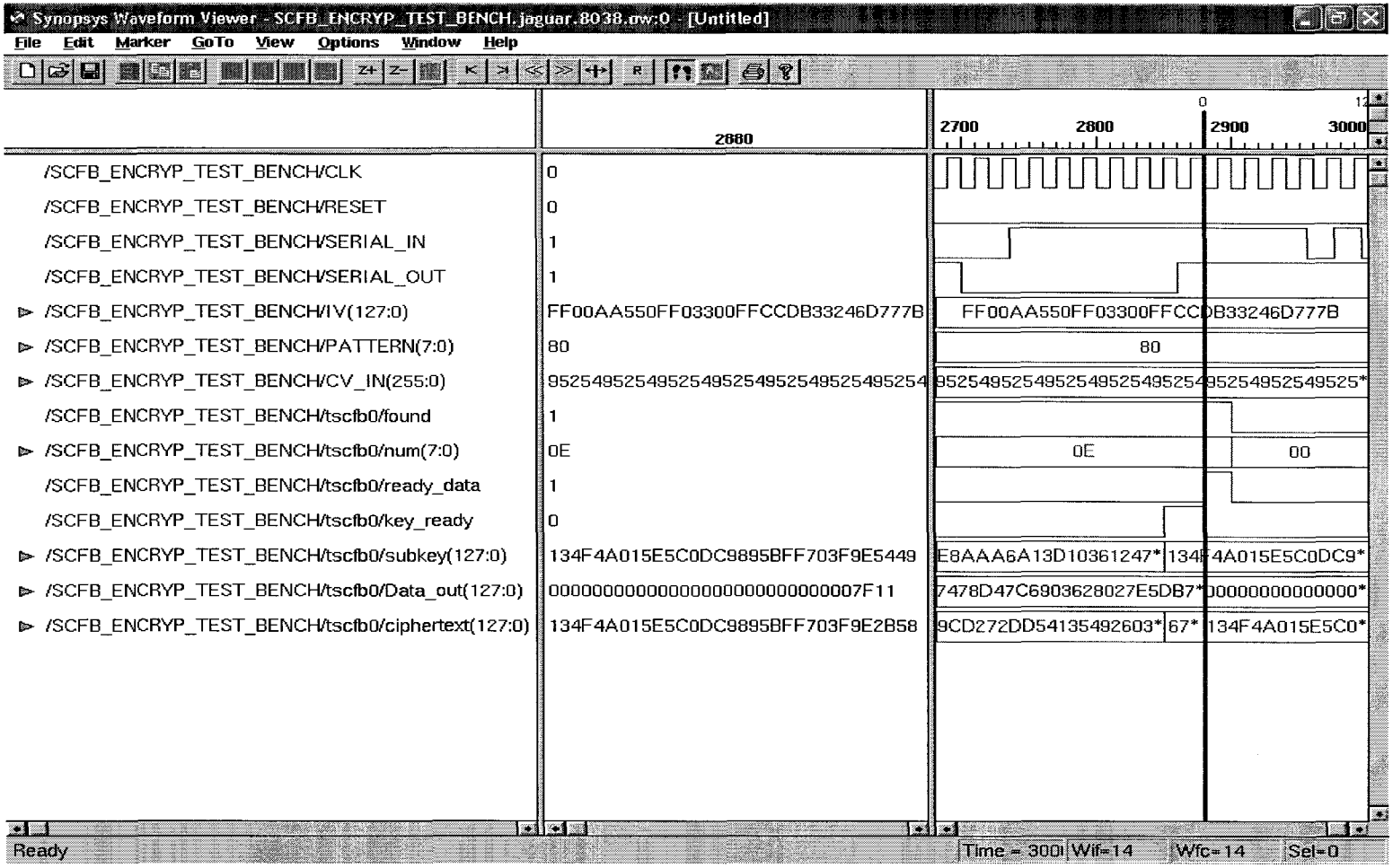
The signals used in the waveforms are described below.

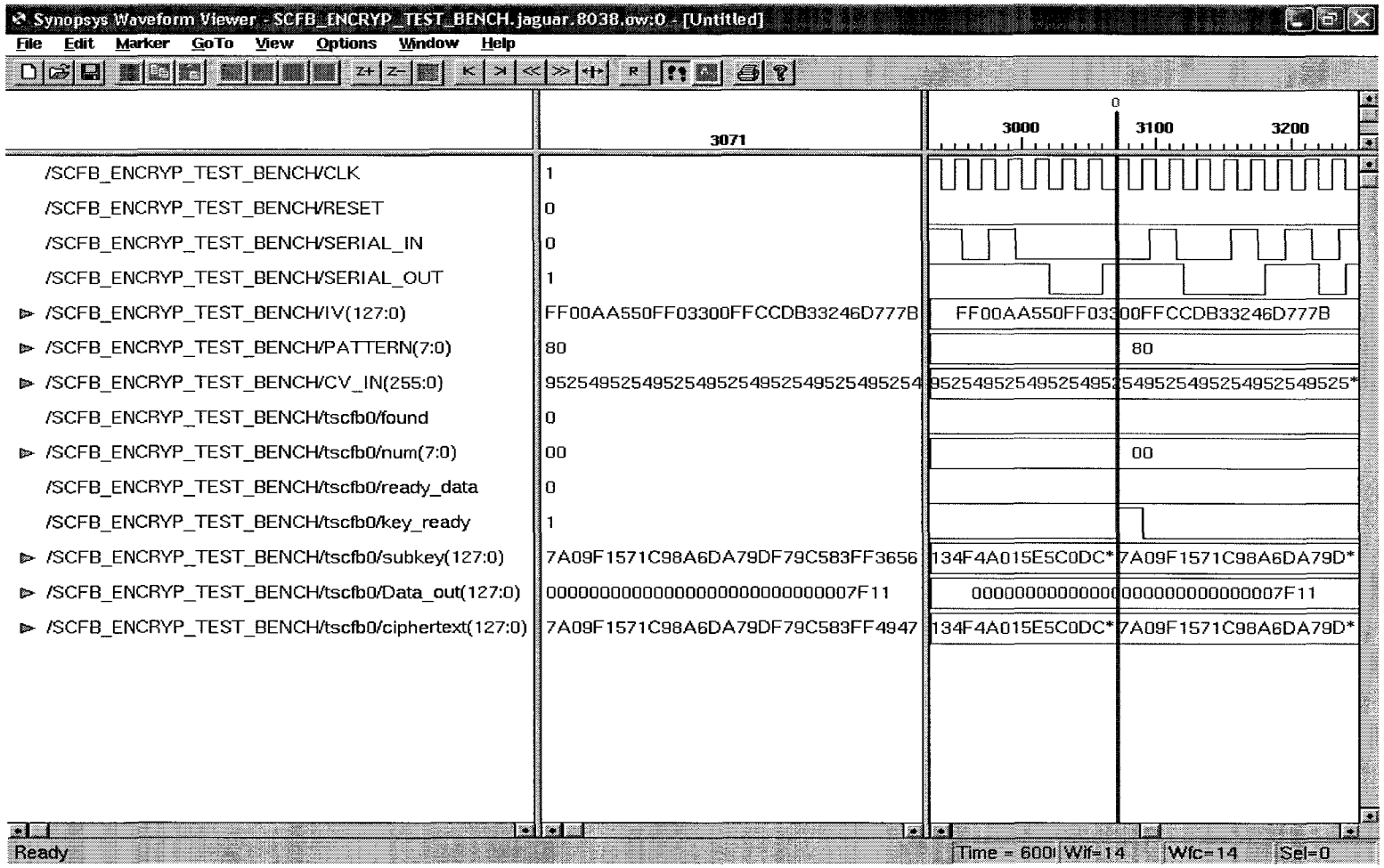
Name of Signal	Function
/SCFB_ENCRYP_TEST_BENCH/RESET	Reset the system
/SCFB_ENCRYP_TEST_BENCH/CLK	System clock
/SCFB_ENCRYP_TEST_BENCH/SERIAL_IN	Data bit incoming into PQ
/SCFB_ENCRYP_TEST_BENCH/SERIAL_OUT	Data bit outgoing from CQ
/SCFB_ENCRYP_TEST_BENCH/IV	Initialization Vector
/SCFB_ENCRYP_TEST_BENCH/PATTERN	Sync pattern
/SCFB_ENCRYP_TEST_BENCH/CV_IN	Initial key
/SCFB_ENCRYP_TEST_BENCH/tscfb0/found	The flag that the sync patter is found or not
/SCFB_ENCRYP_TEST_BENCH/ tscfb0/num	The number needed by the new IV
/SCFB_ENCRYP_TEST_BENCH/ tscfb0/ready_data	The flag to sign that the new block of ciphertext data is ready
/SCFB_ENCRYP_TEST_BENCH/ tscfb0/key_ready	The flag that the new block of keystream is ready
/SCFB_ENCRYP_TEST_BENCH/ tscfb0/subkey	The current keystream
/SCFB_ENCRYP_TEST_BENCH/ tscfb0/Data_out	The current plaintext data sent out by the plaintext subsystem

The waveforms of the encryption system

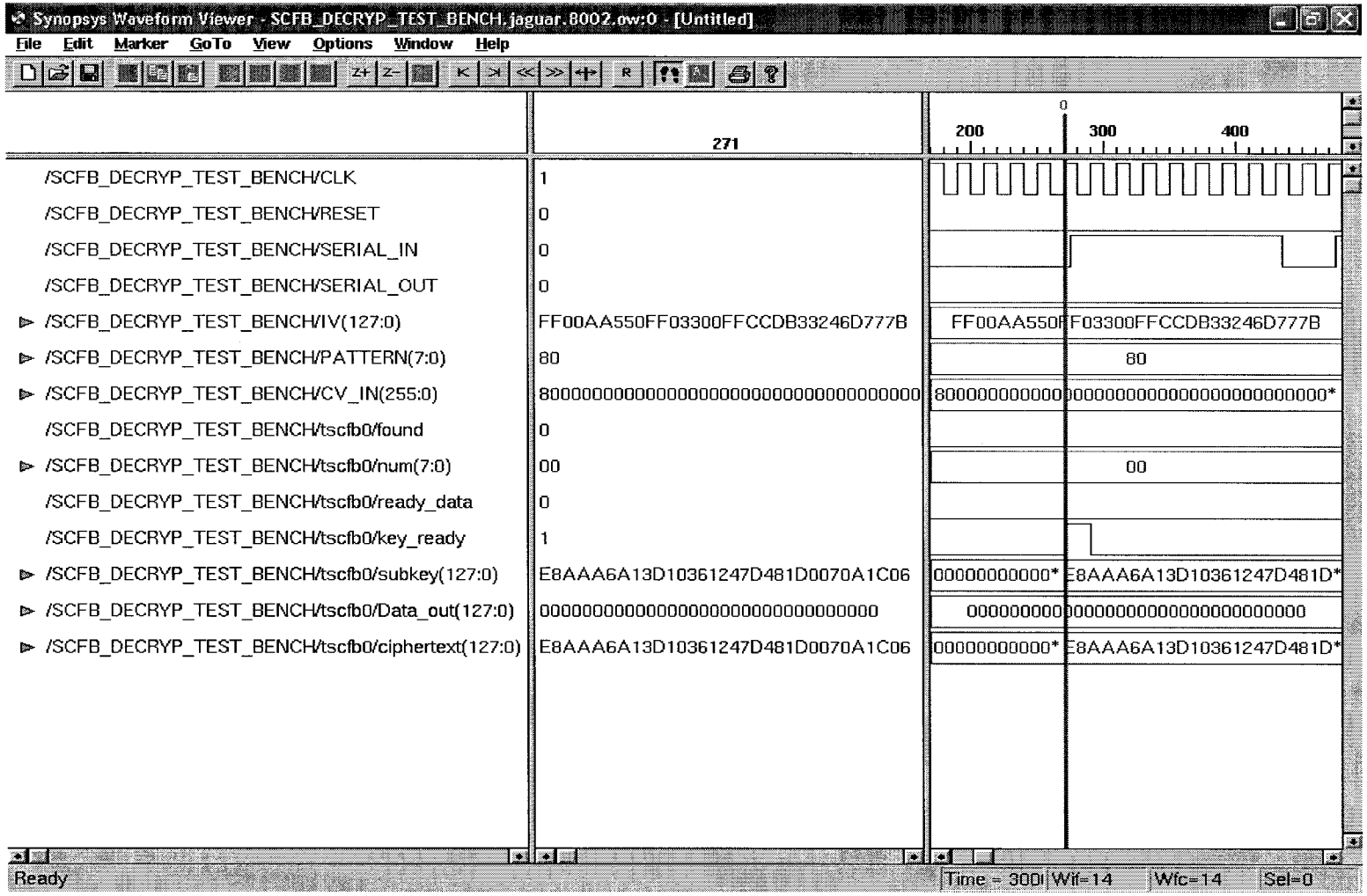

Synopsys Waveform Viewer - SCFB_ENCRYP_TEST_BENCH.jaguar.8038.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
270
0
200
300
400
/SCFB_ENCRYP_TEST_BENCH/CLK
/SCFB_ENCRYP_TEST_BENCH/RESET
/SCFB_ENCRYP_TEST_BENCH/SERIAL_IN
/SCFB_ENCRYP_TEST_BENCH/SERIAL_OUT
/SCFB_ENCRYP_TEST_BENCH/IV(127:0)
/SCFB_ENCRYP_TEST_BENCH/PATTERN(7:0)
/SCFB_ENCRYP_TEST_BENCH/CV_IN(255:0)
/SCFB_ENCRYP_TEST_BENCH/tscfb0/found
/SCFB_ENCRYP_TEST_BENCH/tscfb0/num(7:0)
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ready_data
/SCFB_ENCRYP_TEST_BENCH/tscfb0/key_ready
/SCFB_ENCRYP_TEST_BENCH/tscfb0/subkey(127:0)
/SCFB_ENCRYP_TEST_BENCH/tscfb0/Data_out(127:0)
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ciphertext(127:0)
1
0
1
0
FF00AA550FF03300FFCCDB33246D777B
80
E8AAA6A13D10361247D481D0070A1C06
00
0
1
Ready
Time = 300
Wf=1:4
Wfc=1:4
Sel=0

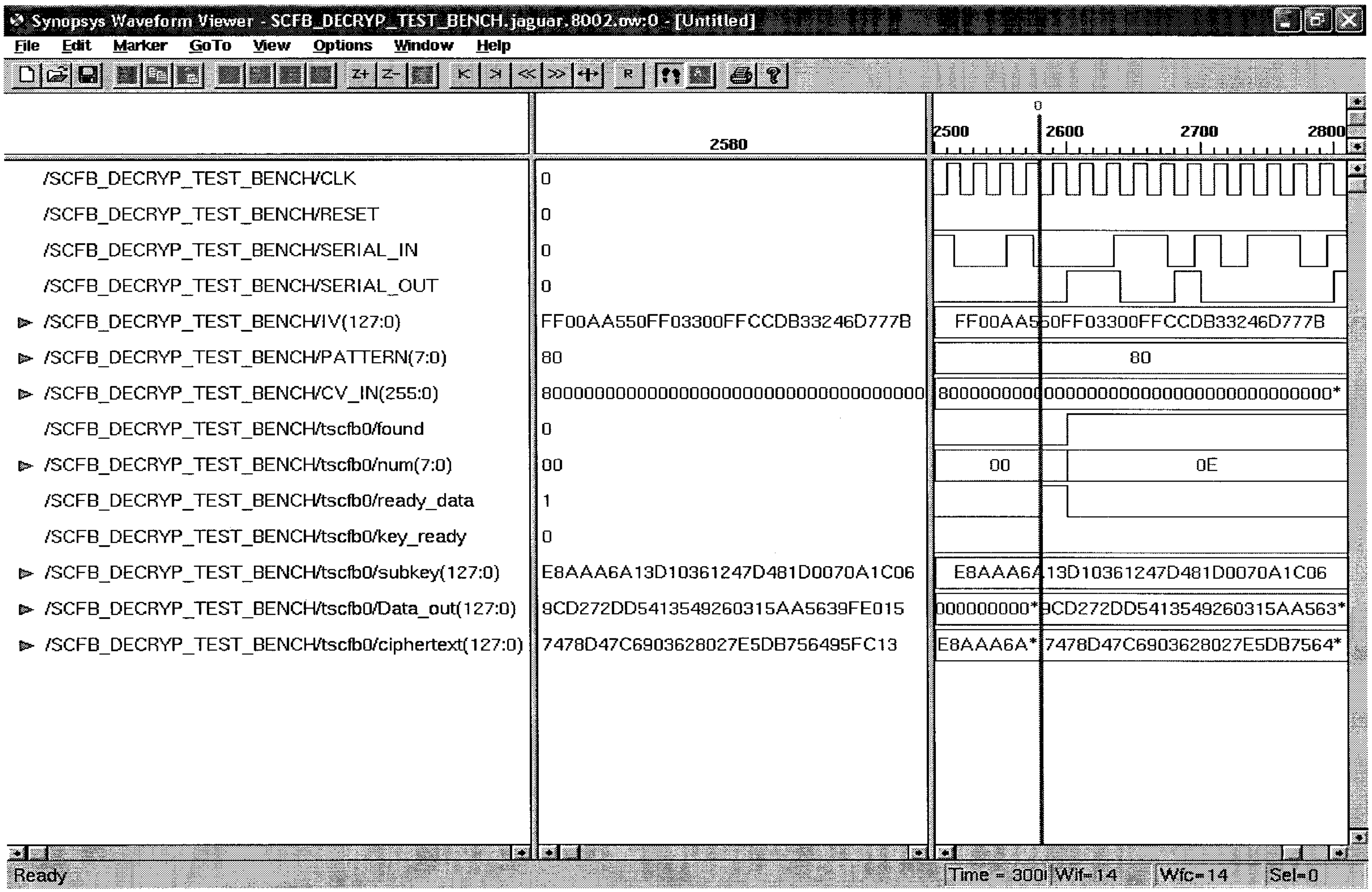

Synopsys Waveform Viewer - SCFB_ENCRYP_TEST_BENCH.jaguar.8038.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
2582
2600
2700
2800
/SCFB_ENCRYP_TEST_BENCH/CLK 0
/SCFB_ENCRYP_TEST_BENCH/RESET 0
/SCFB_ENCRYP_TEST_BENCH/SERIAL_IN 1
/SCFB_ENCRYP_TEST_BENCH/SERIAL_OUT 0
/SCFB_ENCRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B
/SCFB_ENCRYP_TEST_BENCH/PATTERN(7:0) 80
/SCFB_ENCRYP_TEST_BENCH/CV_IN(255:0) 9525495254952549525495254952549525495254
/SCFB_ENCRYP_TEST_BENCH/tscfb0/found 0
/SCFB_ENCRYP_TEST_BENCH/tscfb0/num(7:0) 00
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ready_data 1
/SCFB_ENCRYP_TEST_BENCH/tscfb0/key_ready 0
/SCFB_ENCRYP_TEST_BENCH/tscfb0/subkey(127:0) E8AAA6A13D10361247D481D0070A1C06
/SCFB_ENCRYP_TEST_BENCH/tscfb0/Data_out(127:0) 7478D47C6903628027E5DB756495FC13
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ciphertext(127:0) 9CD272DD5413549260315AA5639FE015
FF00AA550FF03300FFCCDB33246D777B
80
00
0E
E8AAA6A13D10361247D481D0070A1C06
00000*
7478D47C6903628027E5DB756495FC13
E8AA*
9CD272DD5413549260315AA5639FE0*
Ready
Time = 300i
Wif=14
Wfc=14
Sel=0

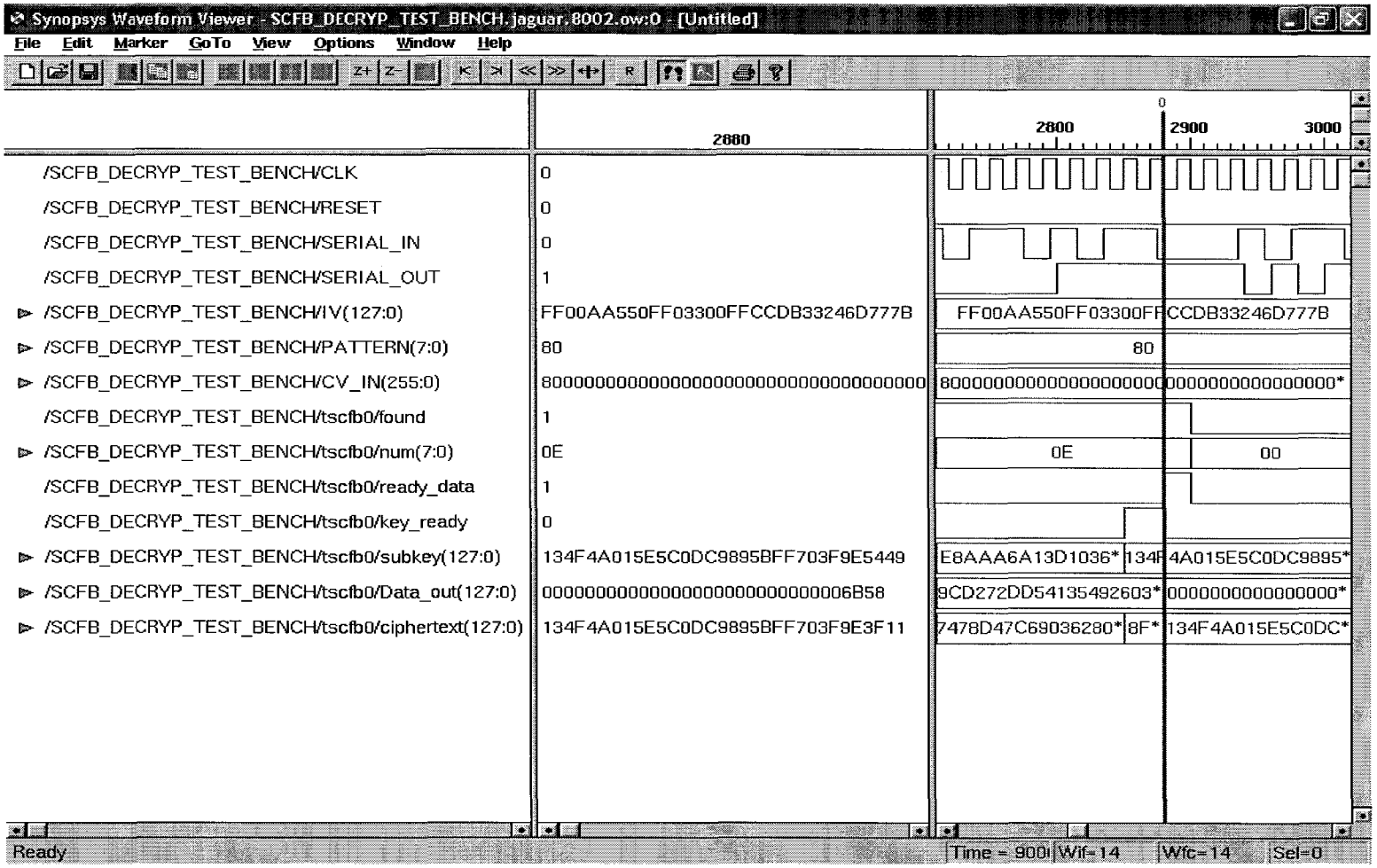

Synopsys Waveform Viewer - SCFB_ENCRYP_TEST_BENCH.jaguar.8038.awv:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
2680
2700 2800 2900 3000
/SCFB_ENCRYP_TEST_BENCH/CLK 0
/SCFB_ENCRYP_TEST_BENCH/RESET 0
/SCFB_ENCRYP_TEST_BENCH/SERIAL_IN 1
/SCFB_ENCRYP_TEST_BENCH/SERIAL_OUT 1
/SCFB_ENCRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B FF00AA550FF03300FFCCDB33246D777B
/SCFB_ENCRYP_TEST_BENCH/PATTERN(7:0) 80 80
/SCFB_ENCRYP_TEST_BENCH/CV_IN(255:0) 952549525495254952549525495254952549525495254 952549525495254952549525495254952549525495254*
/SCFB_ENCRYP_TEST_BENCH/tscfb0/found 1
/SCFB_ENCRYP_TEST_BENCH/tscfb0/num(7:0) 0E 0E 00
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ready_data 1
/SCFB_ENCRYP_TEST_BENCH/tscfb0/key_ready 0
/SCFB_ENCRYP_TEST_BENCH/tscfb0/subkey(127:0) 134F4A015E5C0DC9895BFF703F9E5449 E8AAA6A13D10361247* 134F4A015E5C0DC9*
/SCFB_ENCRYP_TEST_BENCH/tscfb0/Data_out(127:0) 00000000000000000000000000007F11 7478D47C6803628027E5DB7* 00000000000000*
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ciphertext(127:0) 134F4A015E5C0DC9895BFF703F9E2B58 9CD272DD54135492603* 67* 134F4A015E5C0*
Ready
Time = 3001 Wf=14 Wfc=14 Sel=0

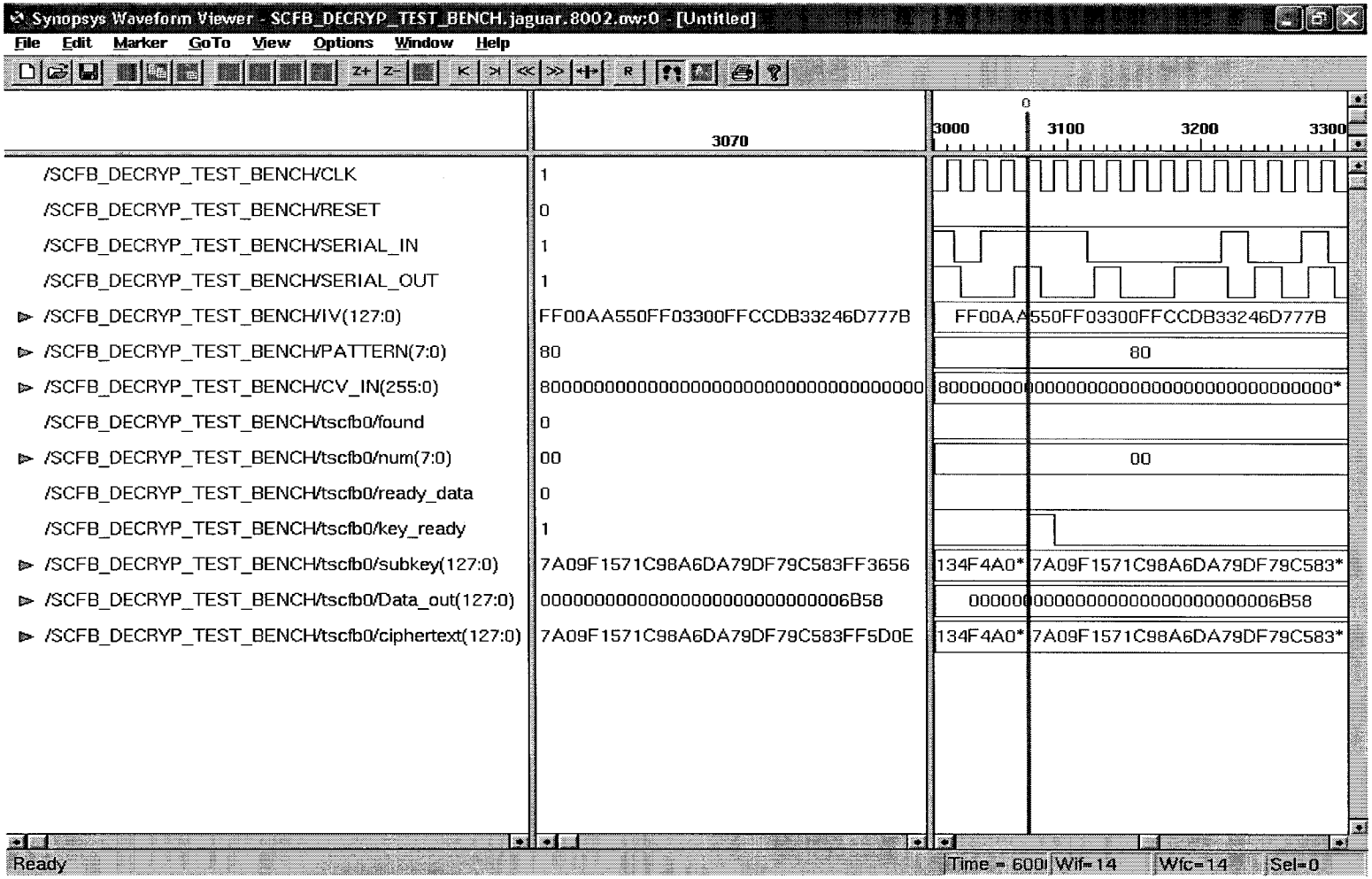

Synopsys Waveform Viewer - SCFB_ENCRYP_TEST_BENCH.jaguar.8038.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
3071
3000
3100
3200
/SCFB_ENCRYP_TEST_BENCH/CLK 1
/SCFB_ENCRYP_TEST_BENCH/RESET 0
/SCFB_ENCRYP_TEST_BENCH/SERIAL_IN 0
/SCFB_ENCRYP_TEST_BENCH/SERIAL_OUT 1
/SCFB_ENCRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B
/SCFB_ENCRYP_TEST_BENCH/PATTERN(7:0) 80
/SCFB_ENCRYP_TEST_BENCH/CV_IN(255:0)
/SCFB_ENCRYP_TEST_BENCH/tscfb0/found 0
/SCFB_ENCRYP_TEST_BENCH/tscfb0/num(7:0) 00
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ready_data 0
/SCFB_ENCRYP_TEST_BENCH/tscfb0/key_ready 1
/SCFB_ENCRYP_TEST_BENCH/tscfb0/subkey(127:0) 7A09F1571C98A6DA79DF79C583FF3656
/SCFB_ENCRYP_TEST_BENCH/tscfb0/Data_out(127:0) 0000000000000000000000000007F11
/SCFB_ENCRYP_TEST_BENCH/tscfb0/ciphertext(127:0) 7A09F1571C98A6DA79DF79C583FF4947
FF00AA550FF03300FFCCDB33246D777B
80
00
134F4A015E5C0DC*
7A09F1571C98A6DA79D*
0000000000000000000000000007F11
134F4A015E5C0DC*
7A09F1571C98A6DA79D*
Ready
Time = 600l Wlf=14
Wfc=14
Sel=0

The waveforms of the decryption system


Synopsys Waveform Viewer - SCFB_DECRYP_TEST_BENCH.jaguar.8002.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
271
200
300
400
/SCFB_DECRYP_TEST_BENCH/CLK 1
/SCFB_DECRYP_TEST_BENCH/RESET 0
/SCFB_DECRYP_TEST_BENCH/SERIAL_IN 0
/SCFB_DECRYP_TEST_BENCH/SERIAL_OUT 0
/SCFB_DECRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B
/SCFB_DECRYP_TEST_BENCH/PATTERN(7:0) 80
/SCFB_DECRYP_TEST_BENCH/CV_IN(255:0) 8000000000000000000000000000000000000000
/SCFB_DECRYP_TEST_BENCH/tscfb0/found 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/num(7:0) 00
/SCFB_DECRYP_TEST_BENCH/tscfb0/ready_data 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/key_ready 1
/SCFB_DECRYP_TEST_BENCH/tscfb0/subkey(127:0) E8AAA6A13D10361247D481D0070A1C06
/SCFB_DECRYP_TEST_BENCH/tscfb0/Data_out(127:0) 00000000000000000000000000000000
/SCFB_DECRYP_TEST_BENCH/tscfb0/ciphertext(127:0) E8AAA6A13D10361247D481D0070A1C06
FF00AA550F F03300FFCCDB33246D777B
80
8000000000000000000000000000000000000000*
00
00000000000* E8AAA6A13D10361247D481D*
00000000000000000000000000000000
00000000000* E8AAA6A13D10361247D481D*
Ready
Time = 300i Wif=14 Wfc=14 Sel=0


Synopsys Waveform Viewer - SCFB_DECRYP_TEST_BENCH.jaguar.8002.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
2580
2500 2600 2700 2800
/SCFB_DECRYP_TEST_BENCH/CLK 0
/SCFB_DECRYP_TEST_BENCH/RESET 0
/SCFB_DECRYP_TEST_BENCH/SERIAL_IN 0
/SCFB_DECRYP_TEST_BENCH/SERIAL_OUT 0
/SCFB_DECRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B
/SCFB_DECRYP_TEST_BENCH/PATTERN(7:0) 80
/SCFB_DECRYP_TEST_BENCH/CV_IN(255:0) 8000000000000000000000000000000000000
/SCFB_DECRYP_TEST_BENCH/tscfb0/found 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/num(7:0) 00
/SCFB_DECRYP_TEST_BENCH/tscfb0/ready_data 1
/SCFB_DECRYP_TEST_BENCH/tscfb0/key_ready 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/subkey(127:0) E8AAA6A13D10361247D481D0070A1C06
/SCFB_DECRYP_TEST_BENCH/tscfb0/Data_out(127:0) 9CD272DD5413549260315AA5639FE015
/SCFB_DECRYP_TEST_BENCH/tscfb0/ciphertext(127:0) 7478D47C6903628027E5DB756495FC13
FF00AA550FF03300FFCCDB33246D777B
80
800000000 000000000000000000000000000*
00 0E
E8AAA6A13D10361247D481D0070A1C06
000000000* 9CD272DD5413549260315AA563*
E8AAA6A* 7478D47C6903628027E5DB7564*
Ready
Time = 300 Wif=14 Wfc=14 Sel=0


Synopsys Waveform Viewer - SCFB_DECRYP_TEST_BENCH.jaguar.8002.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
2680
2800
2900
3000
/SCFB_DECRYP_TEST_BENCH/CLK
/SCFB_DECRYP_TEST_BENCH/RESET
/SCFB_DECRYP_TEST_BENCH/SERIAL_IN
/SCFB_DECRYP_TEST_BENCH/SERIAL_OUT
/SCFB_DECRYP_TEST_BENCH/IV(127:0)
/SCFB_DECRYP_TEST_BENCH/PATTERN(7:0)
/SCFB_DECRYP_TEST_BENCH/CV_IN(255:0)
/SCFB_DECRYP_TEST_BENCH/tscfb0/found
/SCFB_DECRYP_TEST_BENCH/tscfb0/num(7:0)
/SCFB_DECRYP_TEST_BENCH/tscfb0/ready_data
/SCFB_DECRYP_TEST_BENCH/tscfb0/key_ready
/SCFB_DECRYP_TEST_BENCH/tscfb0/subkey(127:0)
/SCFB_DECRYP_TEST_BENCH/tscfb0/Data_out(127:0)
/SCFB_DECRYP_TEST_BENCH/tscfb0/ciphertext(127:0)
0
0
0
1
FF00AA550FF03300FFCCDB33246D777B
80
80000000000000000000000000000000
1
0E
1
0
134F4A015E5C0DC9895BFF703F9E5449
000000000000000000000000006B58
134F4A015E5C0DC9895BFF703F9E3F11
FF00AA550FF03300FFCCDB33246D777B
80
0E
00
E8AAA6A13D1036*
134F4A015E5C0DC9895*
9CD272DD54135492603*
0000000000000000*
7478D47C69036280*
8F*
134F4A015E5C0DC*
Ready
Time = 900
Wf=14
Wfc=14
Sel=0


Synopsys Waveform Viewer - SCFB_DECRYP_TEST_BENCH.jaguar.8002.aw:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
3070
3000
3100
3200
3300
/SCFB_DECRYP_TEST_BENCH/CLK 1
/SCFB_DECRYP_TEST_BENCH/RESET 0
/SCFB_DECRYP_TEST_BENCH/SERIAL_IN 1
/SCFB_DECRYP_TEST_BENCH/SERIAL_OUT 1
/SCFB_DECRYP_TEST_BENCH/IV(127:0) FF00AA550FF03300FFCCDB33246D777B
/SCFB_DECRYP_TEST_BENCH/PATTERN(7:0) 80
/SCFB_DECRYP_TEST_BENCH/CV_IN(255:0) 80000000000000000000000000000000000000000000000000000000000000000
/SCFB_DECRYP_TEST_BENCH/tscfb0/found 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/num(7:0) 00
/SCFB_DECRYP_TEST_BENCH/tscfb0/ready_data 0
/SCFB_DECRYP_TEST_BENCH/tscfb0/key_ready 1
/SCFB_DECRYP_TEST_BENCH/tscfb0/subkey(127:0) 7A09F1571C98A6DA79DF79C583FF3656
/SCFB_DECRYP_TEST_BENCH/tscfb0/Data_out(127:0) 00000000000000000000000000006B58
/SCFB_DECRYP_TEST_BENCH/tscfb0/ciphertext(127:0) 7A09F1571C98A6DA79DF79C583FF5D0E
FF00AA550FF03300FFCCDB33246D777B
80
80000000000000000000000000000000*
00
134F4A0* 7A09F1571C98A6DA79DF79C583*
00000000000000000000000000006B58
134F4A0* 7A09F1571C98A6DA79DF79C583*
Ready
Time = 6001 Wif=14 Wfc=14 Sel=0

The plaintext data in the input file of the encryption system:

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

The plaintext data recovered in the output file of the decryption system:

00000000000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000

00000000000000000000000000001111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

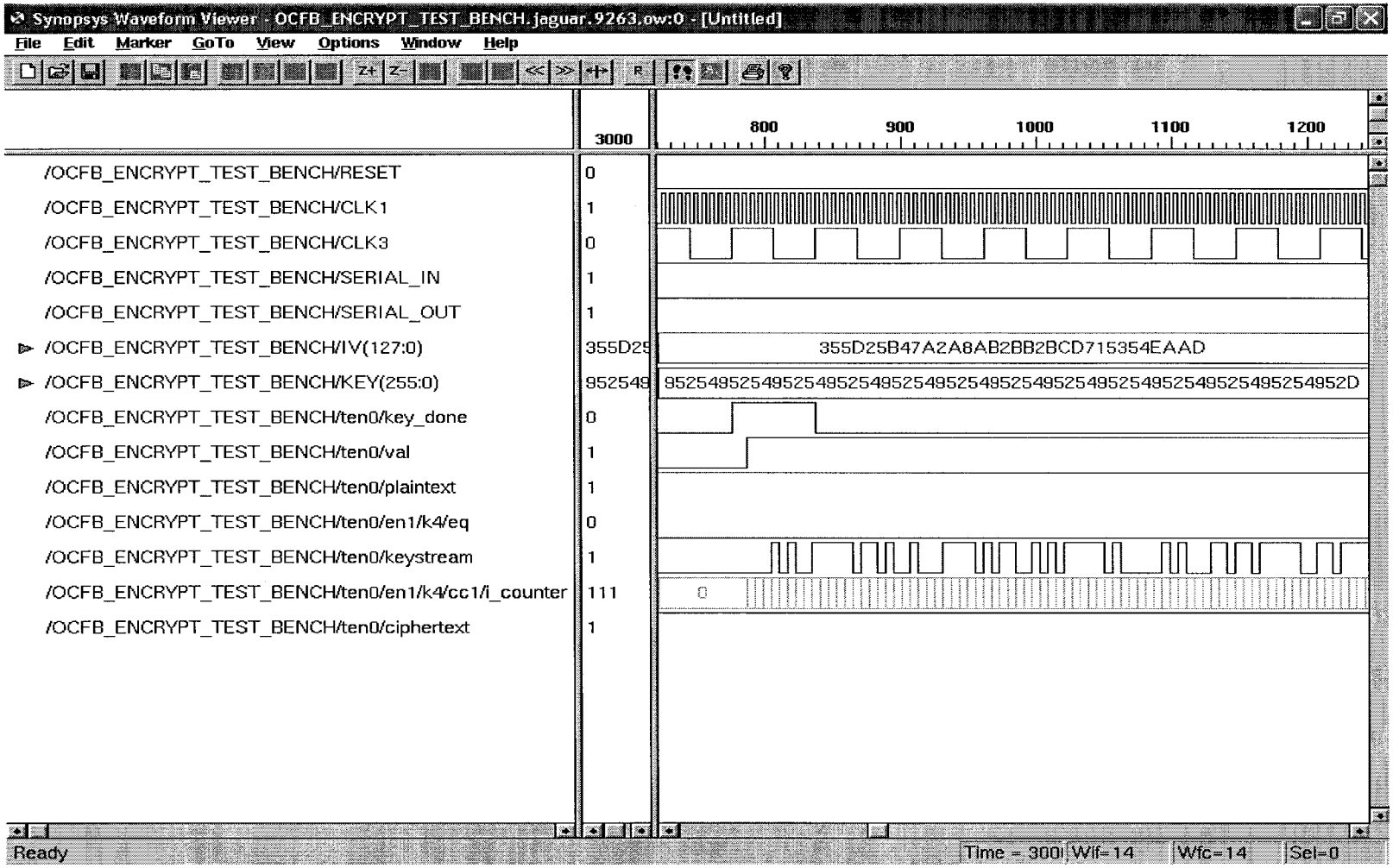
Appendix B

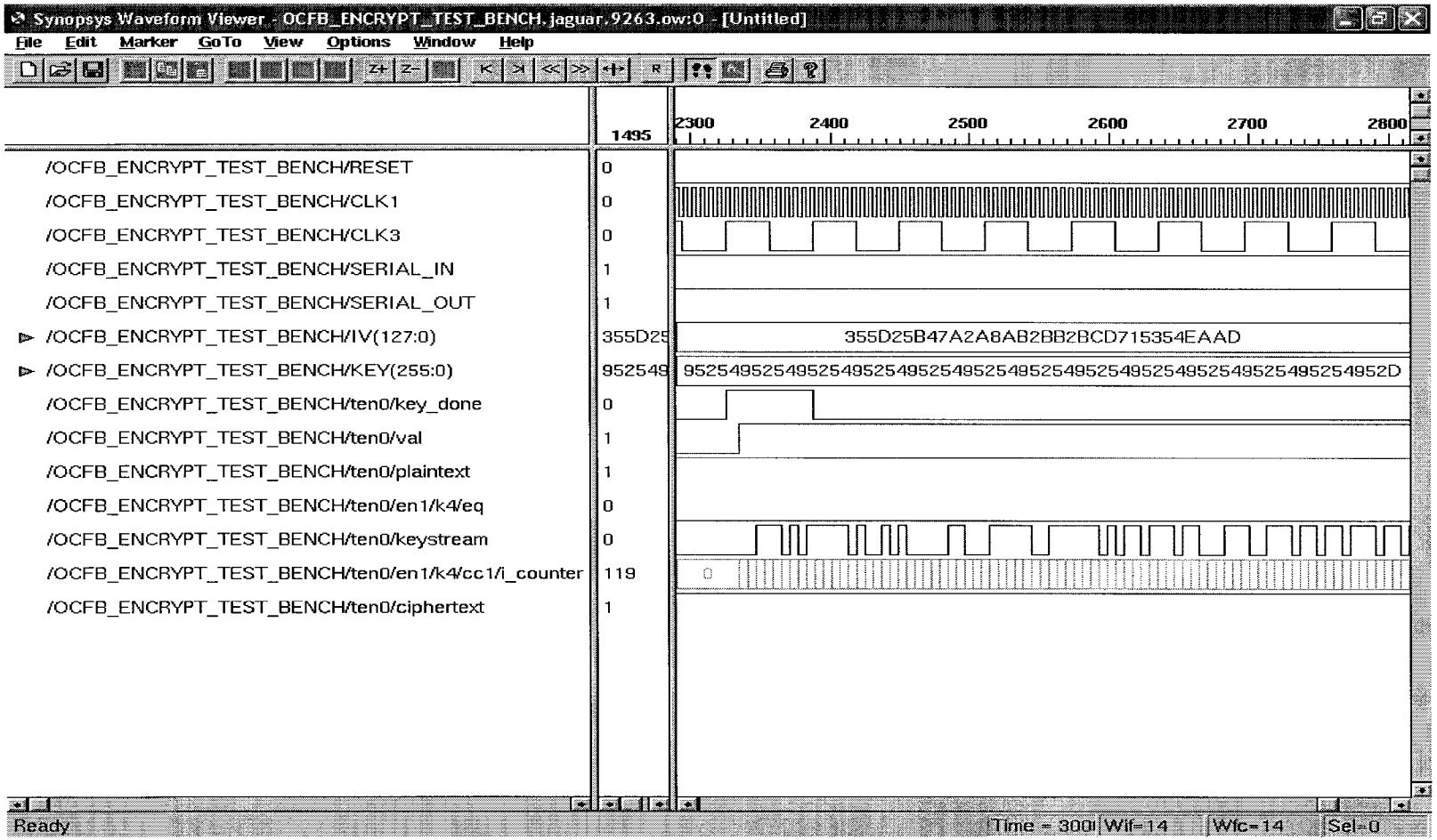
Waveforms of the Hardware Implementation of OCFB Mode

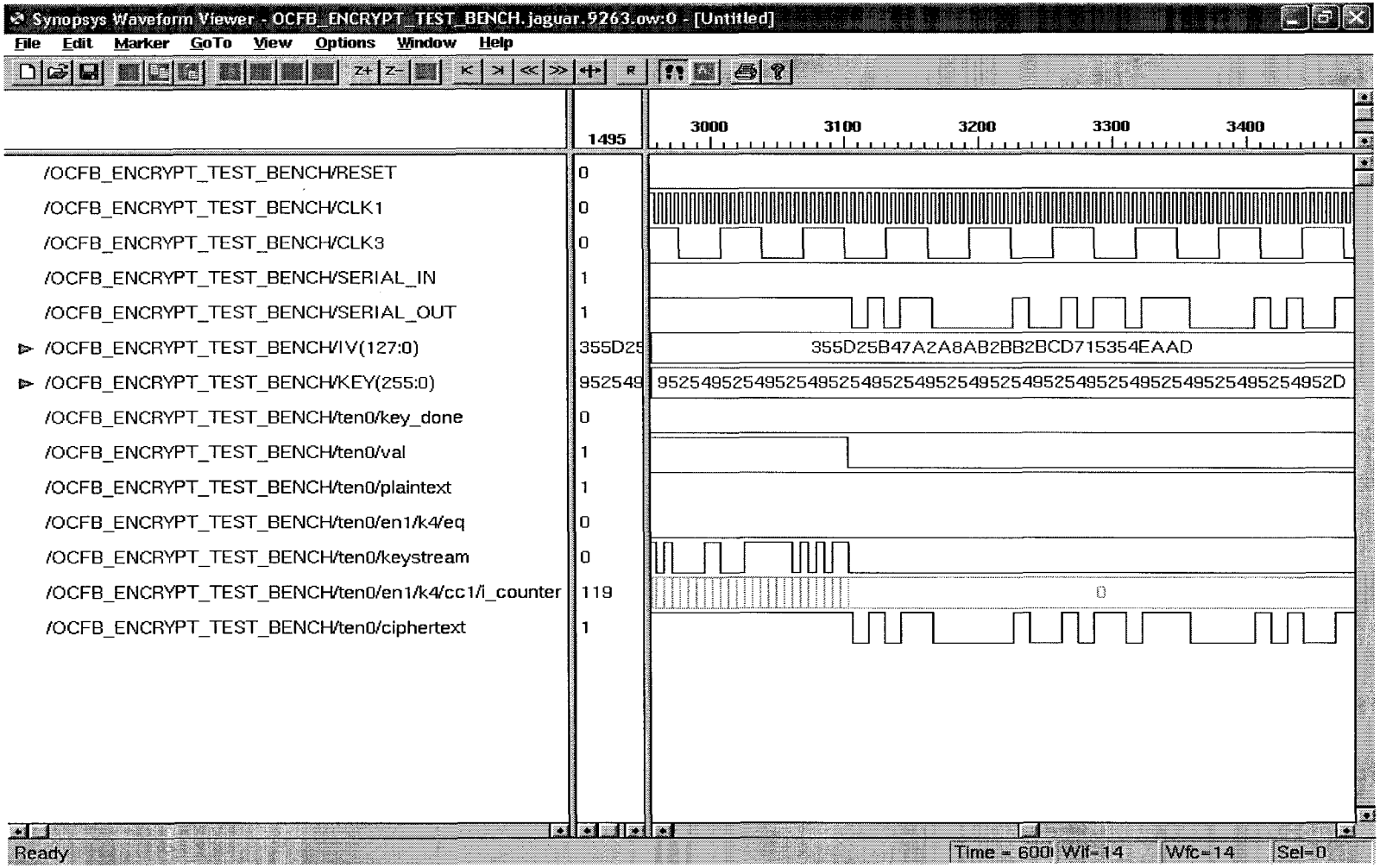
The signals used in the waveforms are described below:

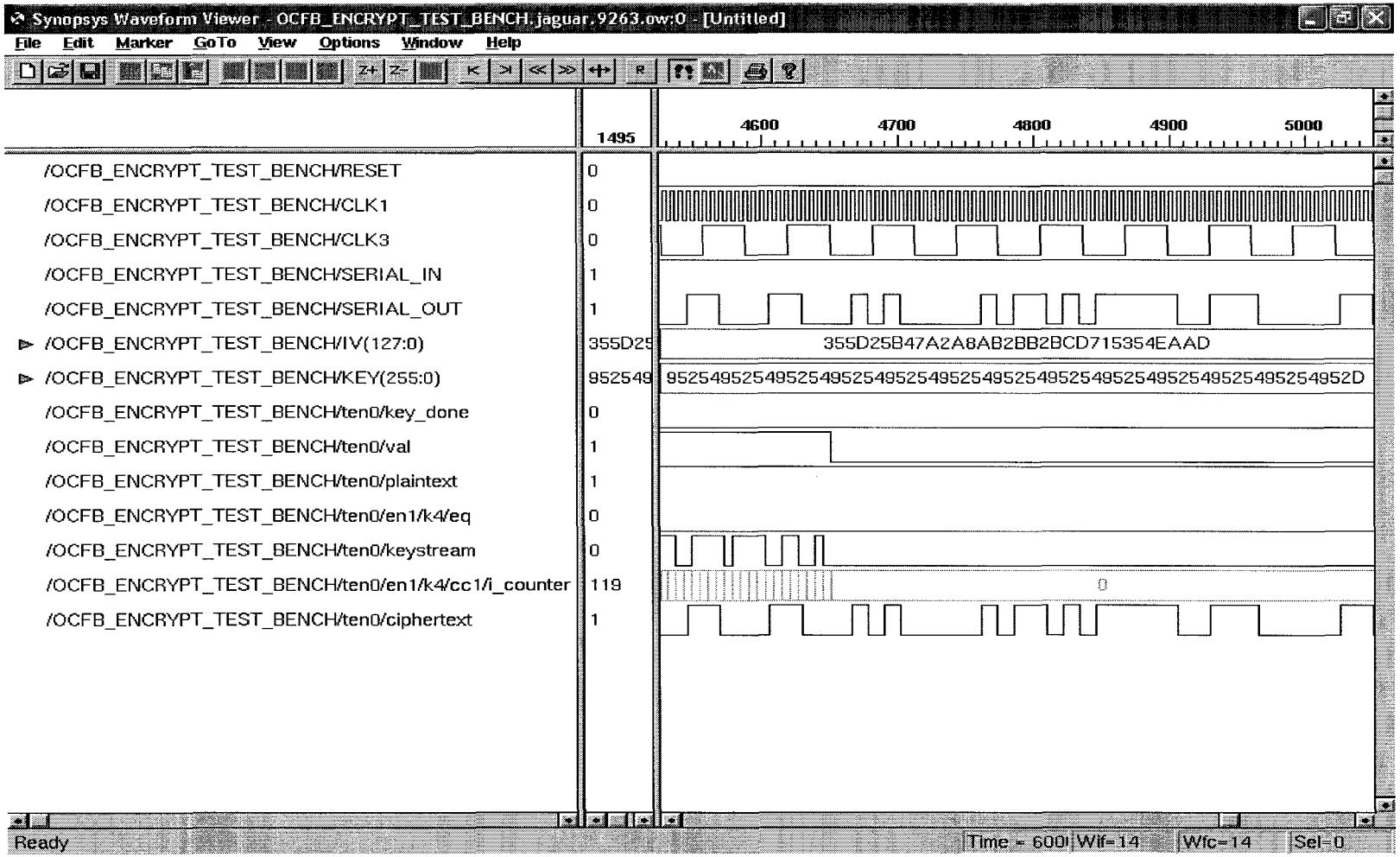
Name of Signal	Function
/OCFB_ENCRYPT_TEST_BENCH/RESET	Reset the system
/OCFB_ENCRYPT_TEST_BENCH/CLK1	System clock
/OCFB_ENCRYPT_TEST_BENCH/ ten0/en0/CLK2	Clock for Leaving from PQ and incoming to CQ
/OCFB_ENCRYPT_TEST_BENCH/CLK3	Clock for block cipher
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_IN	Data bit incoming into PQ
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_OUT	Data bit outgoing from CQ
/OCFB_ENCRYPT_TEST_BENCH/IV	Initialization Vector
/OCFB_ENCRYPT_TEST_BENCH/KEY	Initial key
/OCFB_ENCRYPT_TEST_BENCH/ ten0/key_done	Done signal for encryption of block cipher
/OCFB_ENCRYPT_TEST_BENCH/ ten0/val	Validation of data
/OCFB_ENCRYPT_TEST_BENCH/ ten0/plaintext	Plaintext bit outgoing from PQ
/OCFB_ENCRYPT_TEST_BENCH/ ten0/en1/k4/eq	Flag for sync pattern
/OCFB_ENCRYPT_TEST_BENCH/ ten0/keystream	Keystream bits
/OCFB_ENCRYPT_TEST_BENCH/ ten0/en1/k4/cc1/i_counter	Counter

The waveforms of the encryption system

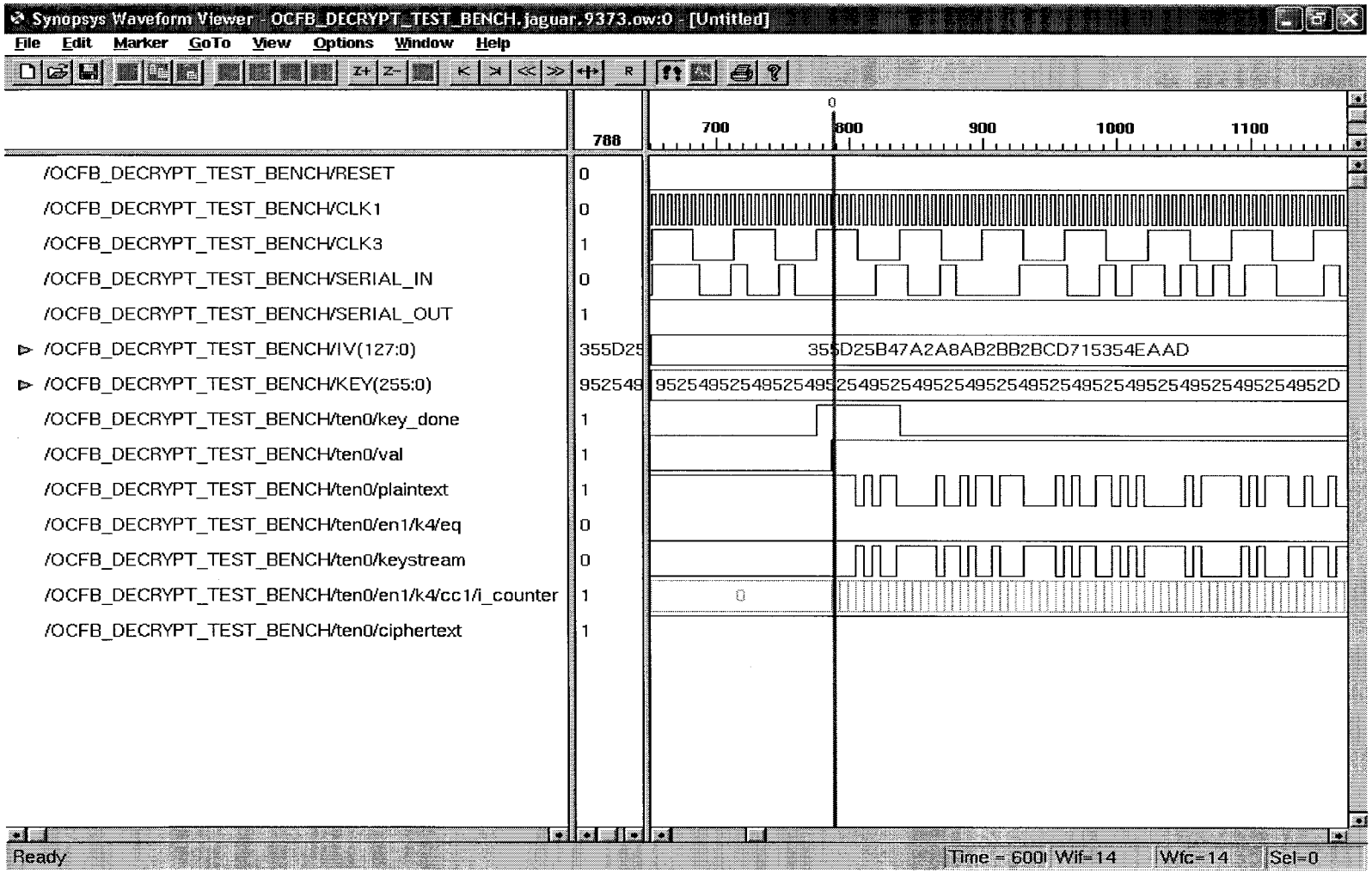

Synopsys Waveform Viewer - OCFB_ENCRYPT_TEST_BENCH.jaguar.9263.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
3000
800 900 1000 1100 1200
/OCFB_ENCRYPT_TEST_BENCH/RESET 0
/OCFB_ENCRYPT_TEST_BENCH/CLK1 1
/OCFB_ENCRYPT_TEST_BENCH/CLK3 0
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_IN 1
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_OUT 1
/OCFB_ENCRYPT_TEST_BENCH/IV(127:0) 355D25
355D25B47A2A8AB2BB2BCD715354EAAD
/OCFB_ENCRYPT_TEST_BENCH/KEY(255:0) 952549
/OCFB_ENCRYPT_TEST_BENCH/ten0/key_done 0
/OCFB_ENCRYPT_TEST_BENCH/ten0/val 1
/OCFB_ENCRYPT_TEST_BENCH/ten0/plaintext 1
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/eq 0
/OCFB_ENCRYPT_TEST_BENCH/ten0/keystream 1
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter 111
/OCFB_ENCRYPT_TEST_BENCH/ten0/ciphertext 1
Ready
Time = 3001
Wif=14
Wfc=14
Sel=0

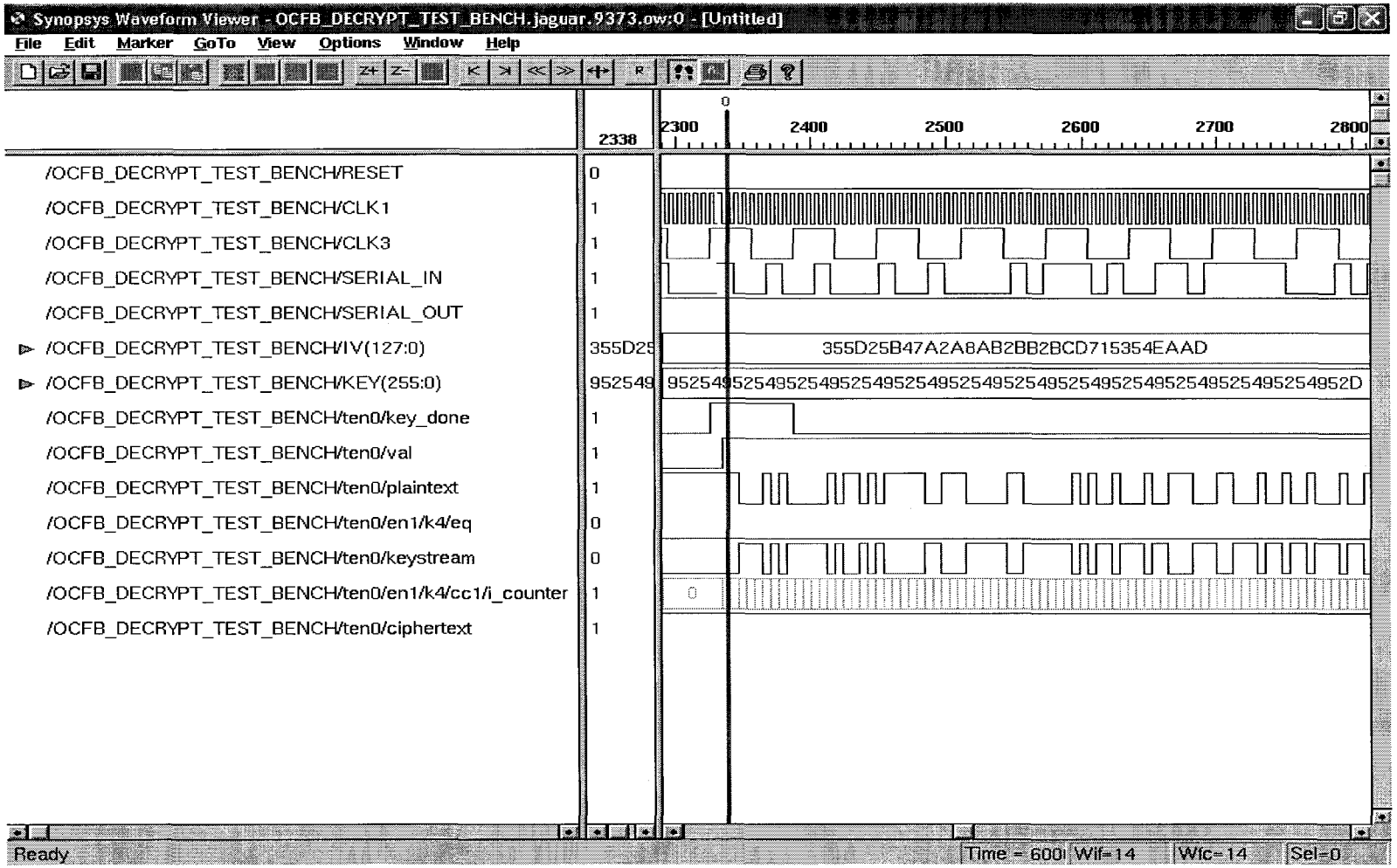

Synopsys Waveform Viewer - OCFB_ENCRYPT_TEST_BENCH.jaguar.9263.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
1495
2300 2400 2500 2600 2700 2800
/OCFB_ENCRYPT_TEST_BENCH/RESET
/OCFB_ENCRYPT_TEST_BENCH/CLK1
/OCFB_ENCRYPT_TEST_BENCH/CLK3
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_IN
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_ENCRYPT_TEST_BENCH/IV(127:0)
/OCFB_ENCRYPT_TEST_BENCH/KEY(255:0)
/OCFB_ENCRYPT_TEST_BENCH/ten0/key_done
/OCFB_ENCRYPT_TEST_BENCH/ten0/val
/OCFB_ENCRYPT_TEST_BENCH/ten0/plaintext
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_ENCRYPT_TEST_BENCH/ten0/keystream
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_ENCRYPT_TEST_BENCH/ten0/ciphertext
0
0
0
1
1
355D25
355D25B47A2A8AB2BB2BCD715354EAAD
952549
0
1
1
0
0
119
1
Ready
Time = 300i
Wif=14
Wfc=14
Sel=0

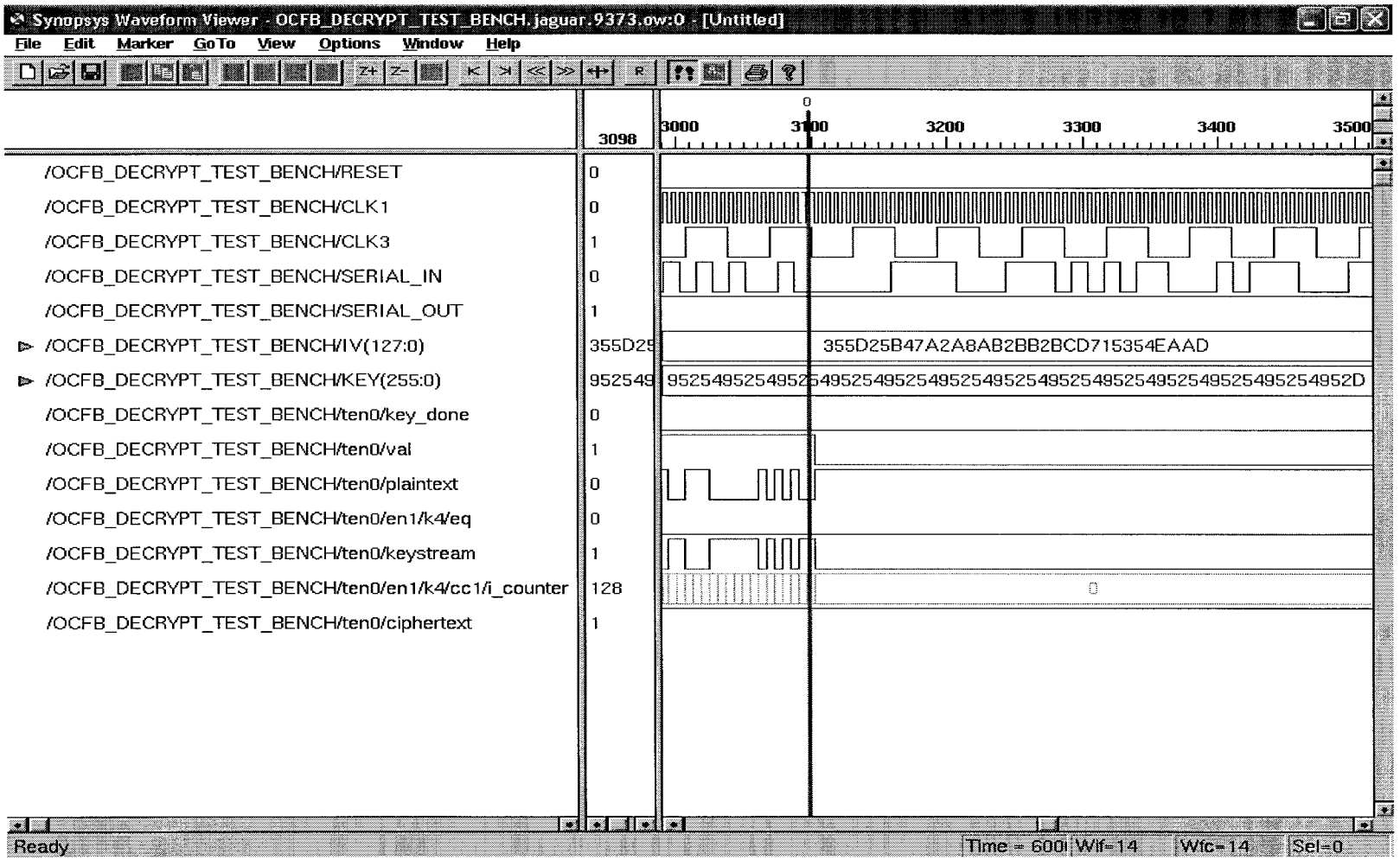

Synopsys Waveform Viewer - OCFB_ENCRYPT_TEST_BENCH.jaguar.9263.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
1495
3000
3100
3200
3300
3400
/OCFB_ENCRYPT_TEST_BENCH/RESET
/OCFB_ENCRYPT_TEST_BENCH/CLK1
/OCFB_ENCRYPT_TEST_BENCH/CLK3
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_IN
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_ENCRYPT_TEST_BENCH/IV(127:0)
/OCFB_ENCRYPT_TEST_BENCH/KEY(255:0)
/OCFB_ENCRYPT_TEST_BENCH/ten0/key_done
/OCFB_ENCRYPT_TEST_BENCH/ten0/val
/OCFB_ENCRYPT_TEST_BENCH/ten0/plaintext
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_ENCRYPT_TEST_BENCH/ten0/keystream
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_ENCRYPT_TEST_BENCH/ten0/ciphertext
0
0
0
1
1
355D25
355D25B47A2A8AB2BB2BCD715354EAAD
952549
952549525495254952548525495254952549525495254952549525495254952549525495254952D
0
1
1
0
0
119
0
1
Ready
Time = 600
Wf=14
Wfc=14
Sel=0

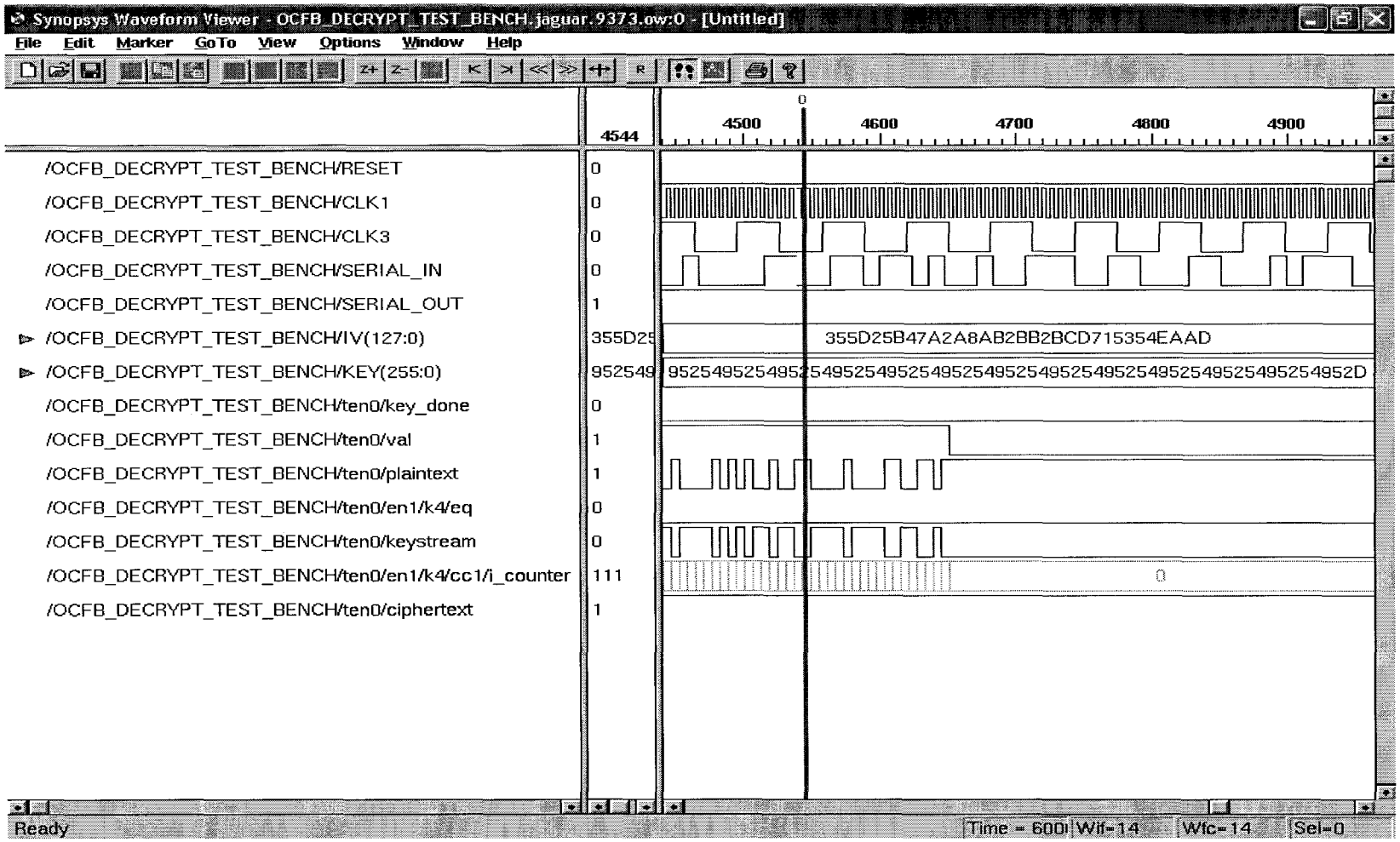

Synopsys Waveform Viewer - OCFB_ENCRYPT_TEST_BENCH.jaguar.9263.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
1495
4600
4700
4800
4900
5000
/OCFB_ENCRYPT_TEST_BENCH/RESET
/OCFB_ENCRYPT_TEST_BENCH/CLK1
/OCFB_ENCRYPT_TEST_BENCH/CLK3
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_IN
/OCFB_ENCRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_ENCRYPT_TEST_BENCH/IV(127:0)
/OCFB_ENCRYPT_TEST_BENCH/KEY(255:0)
/OCFB_ENCRYPT_TEST_BENCH/ten0/key_done
/OCFB_ENCRYPT_TEST_BENCH/ten0/val
/OCFB_ENCRYPT_TEST_BENCH/ten0/plaintext
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_ENCRYPT_TEST_BENCH/ten0/keystream
/OCFB_ENCRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_ENCRYPT_TEST_BENCH/ten0/ciphertext
0
0
0
1
1
355D25
355D25B47A2A8AB2BB2BCD715354EAAD
952549
952549525495254952549525495254952549525495254952549525495254952D
0
1
1
0
0
119
0
1
Ready
Time = 600
Wlf=14
Wfc=14
Sel=0

The waveforms of the decryption system


Synopsys Waveform Viewer - OCFB_DECRYPT_TEST_BENCH.jaguar.9373.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
788
700
800
900
1000
1100
/OCFB_DECRYPT_TEST_BENCH/RESET
/OCFB_DECRYPT_TEST_BENCH/CLK1
/OCFB_DECRYPT_TEST_BENCH/CLK3
/OCFB_DECRYPT_TEST_BENCH/SERIAL_IN
/OCFB_DECRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_DECRYPT_TEST_BENCH/IV(127:0)
/OCFB_DECRYPT_TEST_BENCH/KEY(255:0)
/OCFB_DECRYPT_TEST_BENCH/ten0/key_done
/OCFB_DECRYPT_TEST_BENCH/ten0/val
/OCFB_DECRYPT_TEST_BENCH/ten0/plaintext
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_DECRYPT_TEST_BENCH/ten0/keystream
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_DECRYPT_TEST_BENCH/ten0/ciphertext
355D25B47A2A8AB2BB2BCD715354EAAD
Ready
Time = 600l Wif=14
Wfc=14
Sel=0


Synopsys Waveform Viewer - OCFB_DECRYPT_TEST_BENCH.jaguar.9373.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
2338
2300 2400 2500 2600 2700 2800
/OCFB_DECRYPT_TEST_BENCH/RESET 0
/OCFB_DECRYPT_TEST_BENCH/CLK1 1
/OCFB_DECRYPT_TEST_BENCH/CLK3 1
/OCFB_DECRYPT_TEST_BENCH/SERIAL_IN 1
/OCFB_DECRYPT_TEST_BENCH/SERIAL_OUT 1
/OCFB_DECRYPT_TEST_BENCH/IV(127:0) 355D25 355D25B47A2A8AB2BB2BCD715354EAAD
/OCFB_DECRYPT_TEST_BENCH/KEY(255:0) 952549
/OCFB_DECRYPT_TEST_BENCH/ten0/key_done 1
/OCFB_DECRYPT_TEST_BENCH/ten0/val 1
/OCFB_DECRYPT_TEST_BENCH/ten0/plaintext 1
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/eq 0
/OCFB_DECRYPT_TEST_BENCH/ten0/keystream 0
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter 1
/OCFB_DECRYPT_TEST_BENCH/ten0/ciphertext 1
Ready
Time = 600 Wf=14 Wfc=14 Sel=0


Synopsys Waveform Viewer - OCFB_DECRYPT_TEST_BENCH.jaguar.9373.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
3098
3000
3100
3200
3300
3400
3500
/OCFB_DECRYPT_TEST_BENCH/RESET
/OCFB_DECRYPT_TEST_BENCH/CLK1
/OCFB_DECRYPT_TEST_BENCH/CLK3
/OCFB_DECRYPT_TEST_BENCH/SERIAL_IN
/OCFB_DECRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_DECRYPT_TEST_BENCH/IV(127:0)
/OCFB_DECRYPT_TEST_BENCH/KEY(255:0)
/OCFB_DECRYPT_TEST_BENCH/ten0/key_done
/OCFB_DECRYPT_TEST_BENCH/ten0/val
/OCFB_DECRYPT_TEST_BENCH/ten0/plaintext
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_DECRYPT_TEST_BENCH/ten0/keystream
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_DECRYPT_TEST_BENCH/ten0/ciphertext
0
0
1
0
1
355D25
355D25B47A2A8AB2BB2BCD715354EAAD
952549
128
1
Ready
Time = 600
Wf=14
Wfc=14
Sel=0


Synopsys Waveform Viewer - OCFB_DECRYPT_TEST_BENCH.jaguar.9373.ow:0 - [Untitled]
File Edit Marker GoTo View Options Window Help
4544
4500
4600
4700
4800
4900
/OCFB_DECRYPT_TEST_BENCH/RESET
/OCFB_DECRYPT_TEST_BENCH/CLK1
/OCFB_DECRYPT_TEST_BENCH/CLK3
/OCFB_DECRYPT_TEST_BENCH/SERIAL_IN
/OCFB_DECRYPT_TEST_BENCH/SERIAL_OUT
/OCFB_DECRYPT_TEST_BENCH/IV(127:0)
/OCFB_DECRYPT_TEST_BENCH/KEY(255:0)
/OCFB_DECRYPT_TEST_BENCH/ten0/key_done
/OCFB_DECRYPT_TEST_BENCH/ten0/val
/OCFB_DECRYPT_TEST_BENCH/ten0/plaintext
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/eq
/OCFB_DECRYPT_TEST_BENCH/ten0/keystream
/OCFB_DECRYPT_TEST_BENCH/ten0/en1/k4/cc1/i_counter
/OCFB_DECRYPT_TEST_BENCH/ten0/ciphertext
0
0
0
0
1
355D25
355D25B47A2A8AB2BB2BCD715354EAAD
952549
111
1
Ready
Time = 600
Wf=14
Wfc=14
Sel=0

The plaintext data in the input file of the encryption system:

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

The plaintext data recovered in the output file of the decryption system:

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111

11111111111111111111111111111111111111111111111111