國 立 交 通 大 學

電機資訊學院 電子與光電學程

碩 士 論 文

可測試性設計與內建自我測試技術於
有線等效保密智財之應用與研究

A Case Study on DFT and BIST Design for a Wired Equivalent Privacy IP

研 究 生：林隆裕

指導教授：李崇仁 教授

中華民國 九十三 年 八 月

可測試性設計與內建自我測試技術於有線等效保密智財之應用與研究

A Case Study on DFT and BIST Design for a Wired Equivalent Privacy IP

研究生　：林隆裕　　　Student: Lung-Yu Lin

指導教授：李崇仁 教授　　Advisor: Prof. Chung-Len Lee

國 立 交 通 大 學

電機資訊學院 電子與光電學程

碩 士 論 文

A Thesis
Submitted to Degree Program of Electrical Engineering Computer Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Electronics and Electro-Optical Engineering
August 2004
Hsinchu, Taiwan, Republic of China

中華民國　九十三　年　八　月

可測試性設計與內建自我測試技術於有線等效保密智財之應用與研究

研究生:林隆裕　　　　教授:李崇仁博士

國立交通大學

電機資訊學院 電子與光電學程（研究所）碩士班

摘　要

隨著半導體工藝的進步，一顆單晶片上擁有數百萬邏輯閘的時代已經來臨。也因此，從事可重複使用的智財（IP）設計在當前的 SoC 設計上已經是一種趨勢。然而，在面對如此龐大的電路，尤其是內部的獨立模組，該如何有效且完整的測試著實是一個很大的問題。在本論文中，吾人透過實作來證明“內建自我測試技術”（BIST）在當前的智財設計中有相當的重要性，並倡議設計者在從事智財設計時應該要將“內建自我測試技術”列入設計的考量。

本論文從事的實作有：首先設計出一個符合 IEEE 802.11 標準的“有線等效保密”（WEP）智財；再針對此智財運用各種設計上的技巧，設計出一個可以滿足工業規格的內建自我測試模組，來進行相關的測試與分析。最後，再拿這個內建自我測試模組與一般業界常用的傳統測試方法做分析比較，以期將本論文的實作結果提供給將來從事“內建自我測試技術”研究的讀者做為參考。

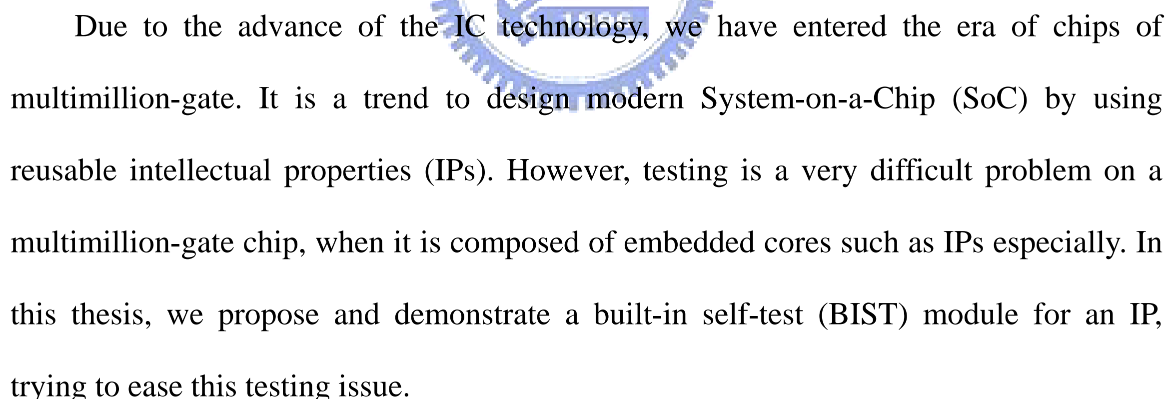
A Case Study on DFT and BIST Design for a Wired Equivalent Privacy IP

Student: Lung-Yu Lin   Advisor: Prof. Chung-Len Lee

Degree Program of Electrical Engineering Computer Science

National Chiao Tung University

Abstract



Due to the advance of the IC technology, we have entered the era of chips of multimillion-gate. It is a trend to design modern System-on-a-Chip (SoC) by using reusable intellectual properties (IPs). However, testing is a very difficult problem on a multimillion-gate chip, when it is composed of embedded cores such as IPs especially. In this thesis, we propose and demonstrate a built-in self-test (BIST) module for an IP, trying to ease this testing issue.

First, we design a reusable “wired equivalent privacy” (WEP) IP based on the IEEE 802.11 standard. Then, in order to test and qualify this IP, we employ the BIST strategy, accommodating some design skills, used in the IP design, to test the IP. Analysis on this strategy has been done to compare it with the traditional testing technique such as scan in terms of fault coverage and hardware overhead. The experience and information obtained will serve as valuable reference to those who engage in BIST research and practice.

// 誌　謝

謹在此獻上最誠摯的謝意，感謝指導教授李崇仁老師在論文研究上提供寶貴的建議與細心指導，以及在英文寫作與投影片報告上的指導，而得以順利完成學業。老師的教學、研究及嚴謹負責的態度也給予我一個好的典範。

其次，我要感謝中華大學陳竹一老師對於論文研究上所提供的寶貴意見，及實驗室裡的夥伴們，因為有你們，激盪出我更多的想法也讓我重溫學生時代的酸甜苦辣。

另外，還要感謝世紀民生科技在我的論文研究上所提供的支持與協助。

最後，謹將本論文獻給我的父母、親愛的老婆及兩個淘氣的小寶貝，感謝你們一路上默默的支持與付出，謝謝！

林隆裕

謹誌於 新竹交大

九十三年　八月

Contents

Chinese abstract I

English abstract II

Acknowledgments III

Contents IV

List of Figures VII

List of Tables IX

Acronyms X

Chapter 1 Introduction 1

1.1 Reusable Intellectual Property 1

1.2 Built-In Self Test Technique for VLSI Design 2

1.3 Purpose of This Thesis 3

1.4 Outline of This Thesis 4

Chapter 2 The Wired Equivalent Privacy (WEP) IP 5

2.1 Wired Equivalent Privacy 5

2.1.1 Introduction 5

2.1.2 WEP Theory of Operation 5

2.1.3 WEP Frame Body Expansion 8

2.1.4 RC4 PRNG Algorithm 9

2.2 Designing a Reusable WEP IP 11

2.2.1 Design Flow 11

2.2.2 Specification 13

2.2.3 Architecture 15

2.2.4 Simulation Results 18

2.2.5 Synthesis Results 22

2.3 Summary 22
Chapter 3 Traditional Testing Methods for WEP 23
3.1 Functional Testing 23
3.1.1 Flow of Function Pattern Generation 24
3.1.2 Testing for Embedded Memory 26
3.1.3 Fault Coverage Results 28
3.2 Test Method by Scan-Chain 29
3.2.1 ATPG with a WEP Hard-Macro 29
3.2.2 ATPG on WEP IP without Memory Wrapper 31
3.2.3 ATPG on WEP IP with Memory Wrapper 32
3.3 Summary 33
Chapter 4 A Built-In Self Test Design for WEP IP 35
4.1 Features 35
4.2 Description of the BIST for WEP IP 35
4.2.1 Architecture 35
4.2.2 Testing Method 38
4.2.3 Testing Flow 40
4.2.4 Embedded 14N March C+ for Embedded Memory 43
4.2.5 The Golden Checksum by CRC-16 44
4.3 Other Special Skills 46
4.3.1 Power Down Mode 46
4.3.2 Speed Up Mode 47
4.3.3 Internal Multiplexers 48
4.3.4 Self-Test of BIST for WEP IP 49
4.4 Experiment Reports 52

4.4.1 Synthesis Results ……………………..……………..……………52

4.4.2 Fault Coverage Results ……………..…….………..………53

4.5 Analysis and Discussion of BIST for WEP IP ………………………54

4.5.1 The Fault Coverage of the Sub-Modules ……………...……54

4.5.2 Improvements of the Fault Coverage ………………………55

4.6 BIST vs. Traditional Testing Methods for WEP IP ………..……..……57

4.7 Summary ……………………………………..……………..………59

Chapter 5 Conclusions ………………………………………………………61

References ………………………………………………………………………..63

Vita ……………………………………………………………………………………..64

List of Figures

Figure 1.1 An example of complex SoC 1
Figure 1.2 Basic BIST architecture 3
Figure 2.1 A confidential data channel 6
Figure 2.2 WEP encipherment block diagram 7
Figure 2.3 WEP decipherment block diagram 7
Figure 2.4 Format of Seed = { Secret Key , Initialization Vector } 8
Figure 2.5 Construction of expanded WEP Frame Body 8
Figure 2.6 RC4 PRNG Engine and S-Box 9
Figure 2.7 RC4 seeding equation 10
Figure 2.8 RC4 Generation Key Sequence Equation 11
Figure 2.9 The design flow of the WEP IP with BIST 12
Figure 2.10 The Block diagram of WEP IP 15
Figure 2.11 The FSM of WEP_RX 16
Figure 2.12 The FSM of WEP_PRNG 17
Figure 2.13 The WEP encryption procedure 18
Figure 2.14 The encryption chart 19
Figure 2.15 The simulation waveform of encipherment 20
Figure 2.16 The WEP decrypt procedure 20
Figure 2.17 The decrypt chart 21
Figure 2.18 The simulation waveform of decipherment 21
Figure 3.1 The WEP IP simulation waveform of functional vectors 24
Figure 3.2 A test vector example without compression 25
Figure 3.3 Local scan-chain in a Hard-Macro IP 30
Figure 3.4 ATPG on WEP IP without Memory Wrapper 31
Figure 3.5 Memory Wrapper in ATPG 32
Figure 4.1 The block diagram of the WEP IP with the BIST module 36
Figure 4.2 The WEP IP data flow in the normal operation mode 37
Figure 4.3 The WEP IP data flow in the BIST operation mode 37
Figure 4.4 The multiplexers of the WEP BIST 38
Figure 4.5 The WEP IP simulation waveform the during BIST mode 40
Figure 4.6 The Major FSM of the WEP BIST Controller 41
Figure 4.7 The sub-FSM of Encrypt Mode (a.) and Decrypt Mode (b.) 42
Figure 4.8 WEP BIST Controller's FSM vs. WEP RX's FSM 43
Figure 4.9 WEP BIST Clock Controller circuit and waveform 47
Figure 4.10 The internal faults on a normal design 48
Figure 4.11 The circuit with the internal multiplexer 49

Figure 4.12 The undetectable faults on normal input ports ……………………..…50
Figure 4.13 The data flow in the bypass mode ……………………………………51
Figure 4.14 Fault Coverage vs. Gate Count and Testing Time ……………………56
Figure 4.15 Fault Coverage vs. BIST Area Overhead ……………….....………..…57
Figure 4.16 Comparison diagrams of the fault coverage and area overhead ……..58
Figure 4.17 Comparison diagrams of the total gate count and test cycles ……...……58

List of Tables

Table 2.1 The pin description of WEP IP 14
Table 2.2 The synthesis results of the WEP IP 22
Table 3.1 WEP IP fault coverage report with functional test vectors 28
Table 3.2 Fault coverage report of the Hard-Macro WEP IP 30
Table 3.3 Fault coverage report of the WEP IP without Memory Wrapper 32
Table 3.4 Fault coverage report of the WEP IP with Memory Wrapper 33
Table 4.1 The input vs. output mapping table in the bypass mode. 51
Table 4.2 The synthesis results of the WEP IP with BIST 52
Table 4.3 The fault coverage report of WEP IP with BIST 53
Table 4.4 Fault coverage and area report of sub-modules 54
Table 4.5 Improvements of the fault coverage 56
Table 4.6 WEP BIST vs. ATPG vs. Functional Testing 57

Acronyms

AF	:	Address Decode Fault
ATE	:	Automatic Test Equipment
ATPG	:	Automatic Test-Pattern Generation
BIST	:	Built-In Self Test
CUT	:	Circuit under Test
FSM	:	Finite State Machine
HDL	:	Hardware Description Language
ICV	:	Integrity Check Value defined in IEEE 802.11 standard
IP	:	Intellectual Property
IV	:	Initialization Vector defined in IEEE 802.11 standard
LFSR	:	Linear Feedback Shift Register
MBIST	:	Memory Built-In Self Test
NRZI	:	Non-Return to Zero, Inverted
PI	:	Parallel Inputs in ATPG
PO	:	Parallel Outputs in ATPG
PRNG	:	Pseudo-Random Number Generator
RTL	:	Register Transfer Level
RZI	:	Return to Zero, Inverted
SAF	:	Stuck-at Fault
S-Box	:	Substitution Box
SoC	:	System-on-a-Chip
WEP	:	Wire Equivalent Privacy defined in IEEE 802.11 standard
WEP_TOP		The top module of our WEP IP design with BIST module
WEP_IP	:	The top module of our WEP IP design without BIST module
WEP_CORE	:	The sub-module includes all digital logic of WEP_IP.
WEP_RX	:	The sub-module manages the input signals of WEP_CORE.
WEP_PRNG	:	The sub-module generates pseudo-random number the in WEP_CORE.
WEP_ICV	:	The sub-module manages the ICV in WEP_CORE.
WEP_BIST	:	The BIST sub-module in WEP_TOP to generate the test vectors to WEP_IP.
wepbtclk	:	WEP BIST Clock Controller. A sub-module in WEP_TOP.
wepbc	:	WEP BIST Controller. A sub-module in WEP_BIST.
wepcrc16	:	WEP CRC-16 Checker. A sub-module in WEP_BIST

Chapter 1 Introduction

1.1 Reusable Intellectual Property

In 1965, Gordon Moore noted that the number of transistors on a chip doubled every 18 to 24 months. Now, advance in silicon technology allows us to build chips consisting of tens of millions of transistors. The ability to leverage valuable intellectual property (IP) through design reuse will be the invariable cornerstone of any effective attack on the productivity issue. Reusable IP is essential to achieving the engineering quality and the timely completion of multimillion-gate ICs [1].

Figure 1.1 is an example of complex *System-on-a-Chip* (SoC) design. This design is the USB2.0-to-IEEE802.11 bridge controller. We can find many reusable IPs in this design, such as a microprocessor, a USB2.0 SIE, a USB2.0 PHY, memories, the IEEE802.11 controller, and a WEP IP.

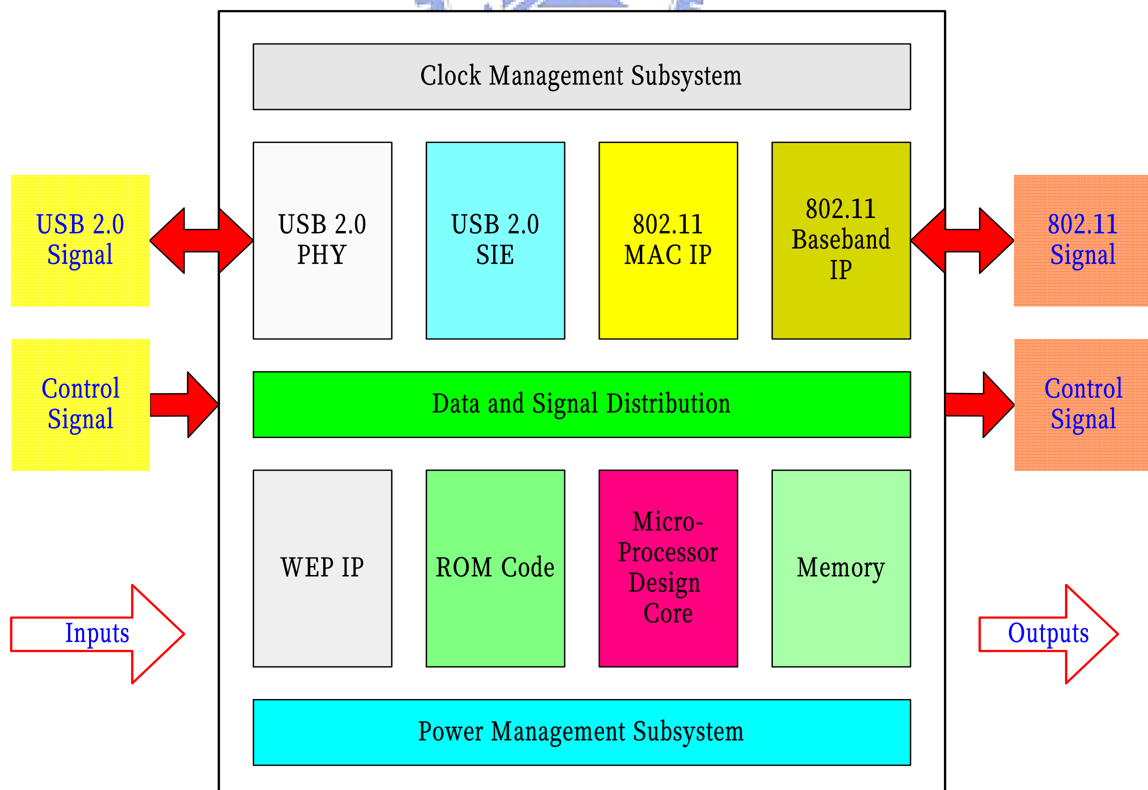


Figure 1.1 An example of complex SoC

1.2 Built-In Self Test Technique for VLSI Design

Testing has become a more difficult problem, due to increasing chip complexity over the past two to three decades. Especially, embedded core functions have become common in VLSI devices with the embedded nature of these functions, making them more difficult to test. In addition, more expensive test equipments are required to handle larger number of I/O pins, higher operating frequencies, and larger sets of test vectors, typically associated with the more complex VLSI device. Finally, due to the growing complexity of VLSI devices, the ability to provide some level of fault diagnosis during manufacturing testing is needed to assist *failure mode analysis* (FMA) for yield enhancement and repair procedures. *Built-In Self Test* (BIST) is considered to be one of the primary solutions to these predicted and growing testing problems.

The basic idea of BIST is to design a circuit so that the circuit can test itself and determine whether it is fault-free or faulty. This typically requires additional circuitry and functionality to be incorporated into the circuit to facilitate the self-testing. This additional functionality must be capable of generating test patterns as well as providing a mechanism to determine if the output responses of the CUT to the test patterns agree to that of the fault-free circuit.

A representative architecture of the BIST is illustrated in the block diagram of Figure 1.2. This BIST architecture includes two essential functions as well as two additional functions that are necessary to facilitate execution of the self-testing feature while in the system. These two essential functions include the *test pattern generator* (TPG) and the *output response analyzer* (ORA). While the TPG produces a sequence of patterns for testing the CUT, the ORA compacts the output responses of the CUT into some type of Pass/Fail indication. The other two functions needed for system-level use of the BIST include the BIST controller and the input multiplexers.

Aside from the normal system I/O pins, the incorporation of BIST may also require additional I/O pins for activating the BIST sequence (the BIST Start signal), reporting the results of the BIST (the Pass/Fail indication), and an optional indication (BIST Done) that the BIST sequence is complete and that the BIST results are valid to be read to determine the fault-free/faulty status of the CUT [2].

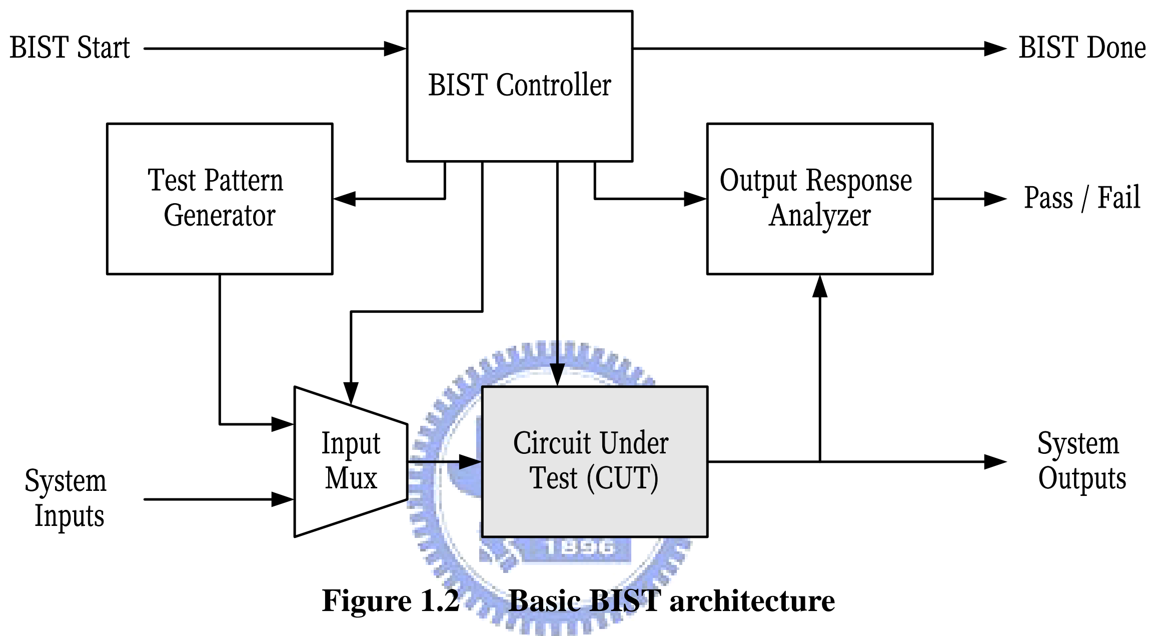


Figure 1.2 Basic BIST architecture

1.3 Purpose of This Thesis

Because of the exponentially growing in wireless communications, many VLSI designers invest much time and resource in creating these components in wireless communications. However, at the same time, the eavesdropping is a familiar problem to users of any types of wireless communications. For this reason, there are many algorithms and architectures about encryption and decryption in wireless technology.

In this thesis, we will design a reusable *Wire Equivalent Privacy* (WEP) IP for the secrecy requirement. The specification of WEP is defined in the IEEE 802.11 wireless standard. In order to guarantee the quality and ease to be reused, we will not only create a

reusable WEP component but also design a built-in self test (BIST) circuit for this IP. We hope to devise a smart method to obtain the quality fault coverage with less effort.

1.4 Outline of This Thesis

This thesis is organized as follows. Chapter 2 presents the IEEE 802.11 Wire Equivalent Privacy standard and describes the architecture of the WEP IP. Chapter 3 presents the traditional testing methods for the WEP IP. Chapter 4 presents the new BIST design in the WEP IP. In chapter 5, conclusions are given.

Chapter 2 The Wired Equivalent Privacy (WEP) IP

2.1 Wired Equivalent Privacy [3]

2.1.1 Introduction

Eavesdropping is a familiar problem to users of any types of wireless technology. IEEE 802.11 specifies an optional privacy algorithm “Wire Equivalent Privacy” that is designed to satisfy the goal of wired LAN “equivalent” privacy. The algorithm is not designed for ultimate security but rather to be “at least as secure as a wire”.

The WEP algorithm has the following properties:

1. It is reasonably strong because WEP allows for the changing of the key and frequent changing of the *initialization vector* (IV) to against the brute-force attack.
2. WEP is self-synchronizing for each message to reduce the packet loss rates.
3. The WEP algorithm is efficient and may be implemented in either hardware or software
4. The WEP algorithm may be exportable from the USA.
5. The implementation and use of WEP is an IEEE 802.11 option.

2.1.2 WEP Theory of Operation

Data that are not enciphered are called *plaintext* (denoted by P) and data that are enciphered are called *ciphertext* (denoted by C). The process of disguising (binary) data in order to hide their information content is called *encryption* (denoted by *E)* and the process of turning ciphertext back into plaintext is called *decryption* (denoted by *D*).

The encryption function *E* operates on *P* to produce *C:*

$$E_K(P) = C \qquad (1)$$

In the reverse process, the decryption function *D* operates on *C* to produce *P:*

$$D_K(C) = P \tag{2}$$

WEP is a symmetric algorithm in which the same key is used for encipherment and decipherment. We can get the following equation:

$$D_K(E_K(P)) = D_K(C) = P \tag{3}$$

The procedure of encryption and decryption is depicted in Figure 2.1.

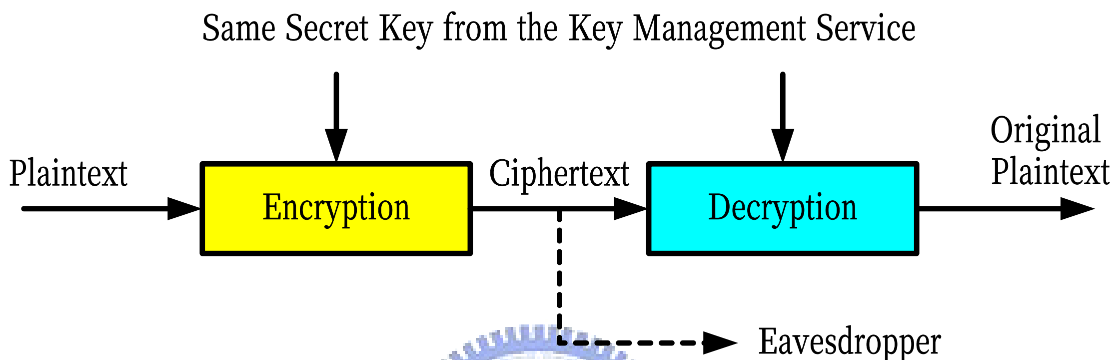


Figure 2.1 A confidential data channel

Referring to Figure 2.2, the secret key is concatenated with an initialization vector (IV) and the resulting *seed* is input to a *pseudo-random number generator* (PRNG). WEP uses the RC4 PRNG algorithm as defined in Section 2.1.4. The PRNG outputs a *key sequence k* of pseudorandom octets equal in length to the number of data octets that are to be transmitted plus 4. The key sequence is used to protect the *integrity check value* (ICV) as well as the data. The WEP ICV is 32 bits. The WEP Integrity Check algorithm is CRC-32, as defined with the following equation:

$$G(X)=X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X^{1}+1 \tag{4}$$

The encrypted ICV can guarantee the completeness of message and protect the ciphertext from malicious distortion.

WEP combines key sequence k with plaintext using bitwise XOR to generate ciphertext. The output of the process is a message containing the IV and ciphertext.

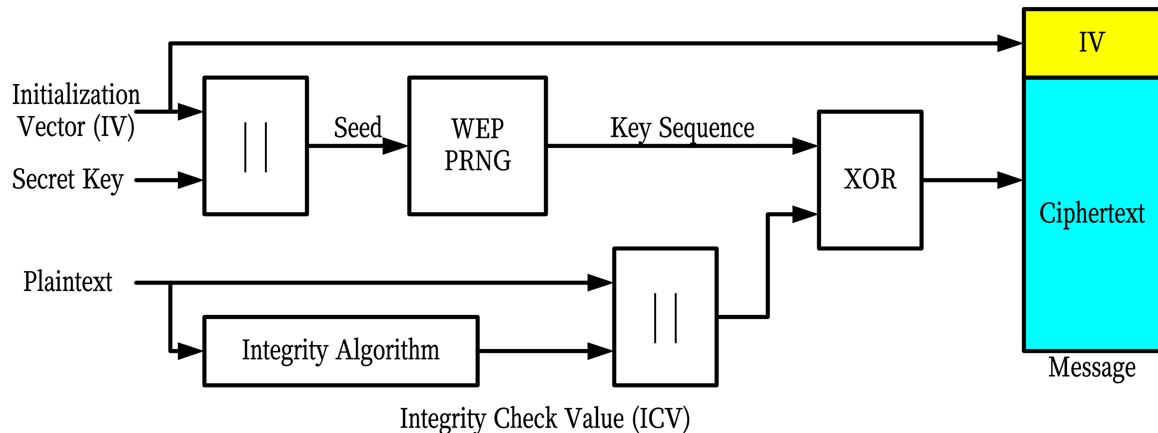


Figure 2.2 WEP encipherment block diagram

Referring to Figure 2.3, decipherment begins with the arrival of a message. The IV of the incoming message shall be combined with the same secret key to generate the same seed as encipherment, as depicted in Figure 2.4. The same seed and the same PRNG algorithm generate the same key sequence to decipher the incoming encrypted message. Combining the ciphertext with the proper key sequence yields the original plaintext and ICV. The same integrity check algorithm is used to generate the ICV' from the recovered plaintext. Correct decipherment shall be verified by comparing the output ICV' to the ICV transmitted with the message. If ICV' is not equal to ICV, the received message is in error and an error indication is sent to the management.

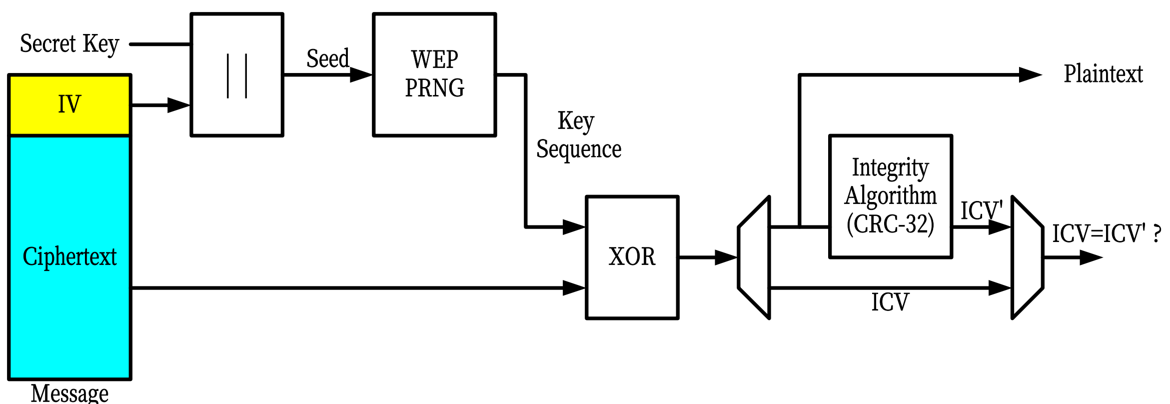


Figure 2.3 WEP decipherment block diagram

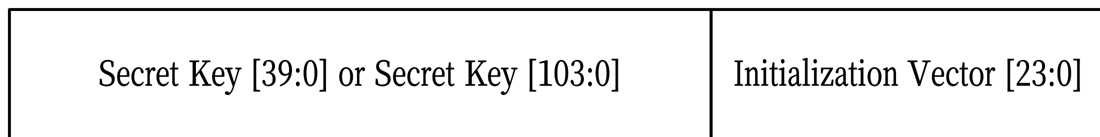


Figure 2.4 Format of the Seed = { Secret Key , Initialization Vector }

2.1.3 WEP Frame Body Expansion

Figure 2.5 shows the encrypted Frame Body in general frame form as constructed by the WEP algorithm.

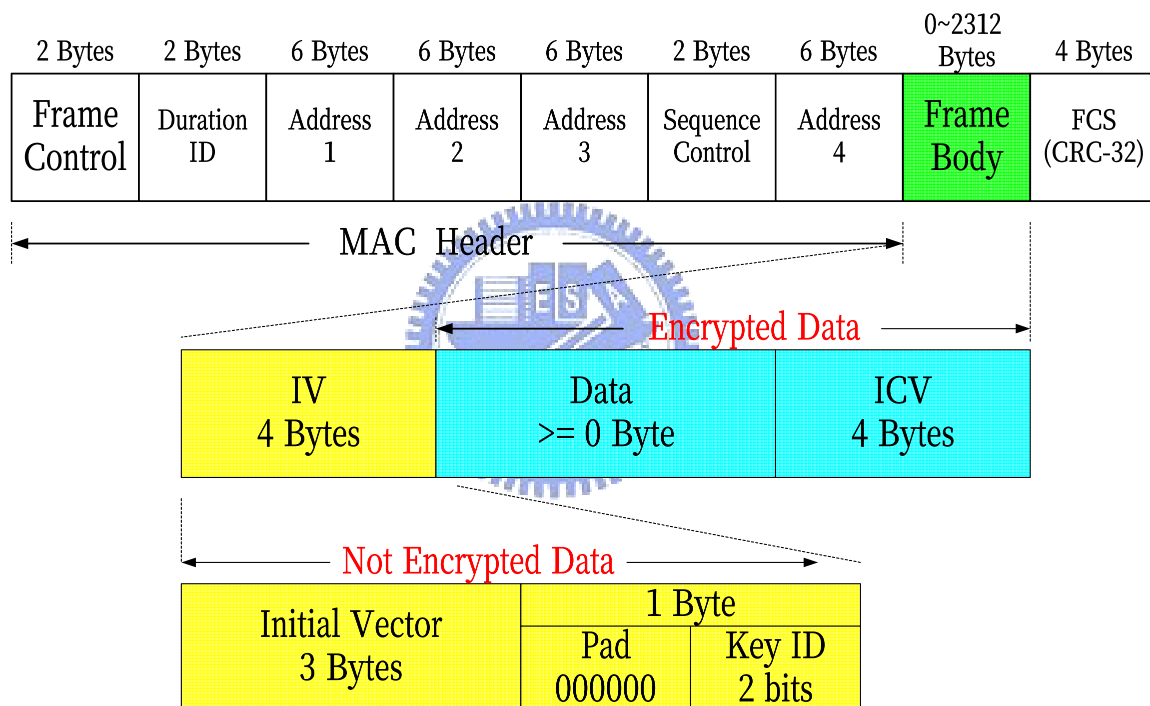


Figure 2.5 Construction of expanded WEP Frame Body

The encipherment process has expanded the original Frame Body by 8 octets, 4 for the IV field and 4 for the ICV. The ICV is calculated on the data field only. The IV field shall contain three subfields: a three-octet field that contains the initialization vector, a 2-bit key ID field, and a 6-bit pad field. The initialization vector is a part of seed which was described in the early part of this chapter. The contents of the pad subfield shall be zero.

The Key ID subfield contents select one of four possible secret key values for use in decrypting this Frame Body.

2.1.4 RC4 PRNG Algorithm [4]

WEP uses the "RC4 PRNG algorithm" which was originally from RSA Data Security, Inc. The RC4 PRNG is a stream cipher symmetric key algorithm. The transmitter and the receiver have same secret key sequence. Referring to Figure 2.6, a 256-byte RAM that is named substitution boxes (denoted by S-Box) contents 256 independent number in random order. The RC4 PRNG engine is like a stirrer to stir the random number in the S-Box.

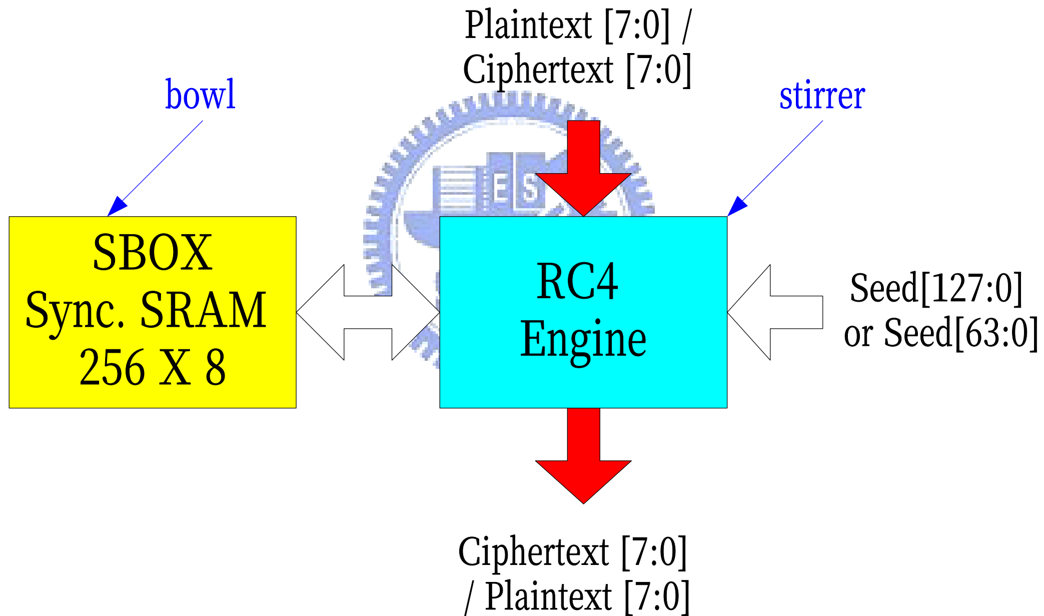


Figure 2.6 RC4 PRNG Engine and S-Box

There are two major steps in RC4 PRNG to initiate the S-Box then generate the key sequence. The first step is called "RC4 Seeding". In this step, the RC4 PRNG engine uses the incoming seed to stir the order of number in S-Box.

The algorithm of the "RC4 Seeding" is defined in following Figure 2.7:

Defined:

Key_Length = (bit number of valid Seed) / 8

Key [x] = Seed [x * 8 + 7 : x] ; where 0 ≦ x ≦ 15

S [y] is the octets in the S-Box at address y ;   where 0 ≦ y ≦ 255

RC4 Seeding:

```
For ( I = 0 ; I < 256 ; I = I + 1 )
    S [ I ] = I;
For ( I = J = 0 ; I < 256 ; I = I + 1 )
{
    J = J + S [ I ] + Key [ ( I   modulo   Key_Length ) ] ;
    SWAP ( S [ I ] , S [ J ] );
}
I = J = 0;
```

Figure 2.7 RC4 Seeding equation

The second step is "Generate Key Sequence". The RC4 PRNG engine continuously generates 8-bit key sequence of pseudorandom octets equal in length to the number of data octets that are to be transmitted plus 4. In the encipherment process, WEP combines key sequence with plaintext using bitwise XOR to generate ciphertext. In the decipherment process, the plaintext is recovered from the same key sequence and incoming ciphertext.

When an octet key sequence is generated, the RC4 PRNG engine stirs the order of number in S-Box again.

The algorithm of the "RC4 Generate Key-stream" is defined in following Figure 2.8:

RC4 Generate Key Sequence :

I = (I + 1) modulo 256;

J = (J + S [I]) modulo 256;

SWAP (S [I], S [J]);

Key Sequence [7 : 0] = S [(S [I] + S [J]) modulo 256];

Figure 2.8 RC4 Generate Key Sequence equation

2.2 Designing a Reusable WEP IP

In the second part of Chapter 2, our target is to design a reusable WEP IP based on IEEE 802.11 standard and RC4 algorithm described in the early part of this chapter.

2.2.1 Design Flow

Before to deign a "Build-In Self Test (BIST)" circuit for WEP IP, we must design a reusable WEP IP based on standards. To finish a reusable WEP IP, there are some major steps of design shown in Figure 2.9. In this figure, the EDA tools used for developing the WEP IP are also shown beside the blocks of steps with blue text [5, 6, 7, 8].

The first step is "specification definition". Every design needs a target specification to point out the features and the applications. The detail specification of WEP IP is defined in Section 2.2.2.

The second step is "RTL coding of WEP IP". In this step, the architecture and the block diagram of "WEP IP" are depicted. Because the WEP IP is a synchronous and synthesizable design, we create the "Register Transfer Level" (denoted by *RTL*) code in the "Verilog Hardware Description Language (HDL)".

The third step is "RTL simulation of WEP IP". There are two simulation cases to

verify the correctness of the WEP encipherment and the decipherment. These simulation environments and patterns are also created in Verilog HDL. If there are any mismatch or incorrectness, we will return to step 2 to modify the RTL code and to fix the issue. Please refer to Section 2.2.4 for the waveforms of simulations.

After all simulations are passed successfully, the RTL code will be synthesized to gate level netlist with a target process. We use the tsmc 0.35um process in this paper. In this step, we also do the timing and area optimization with synthesis EDA tools. Please refer to Section 2.2.5 for the synthesis constraints and results.

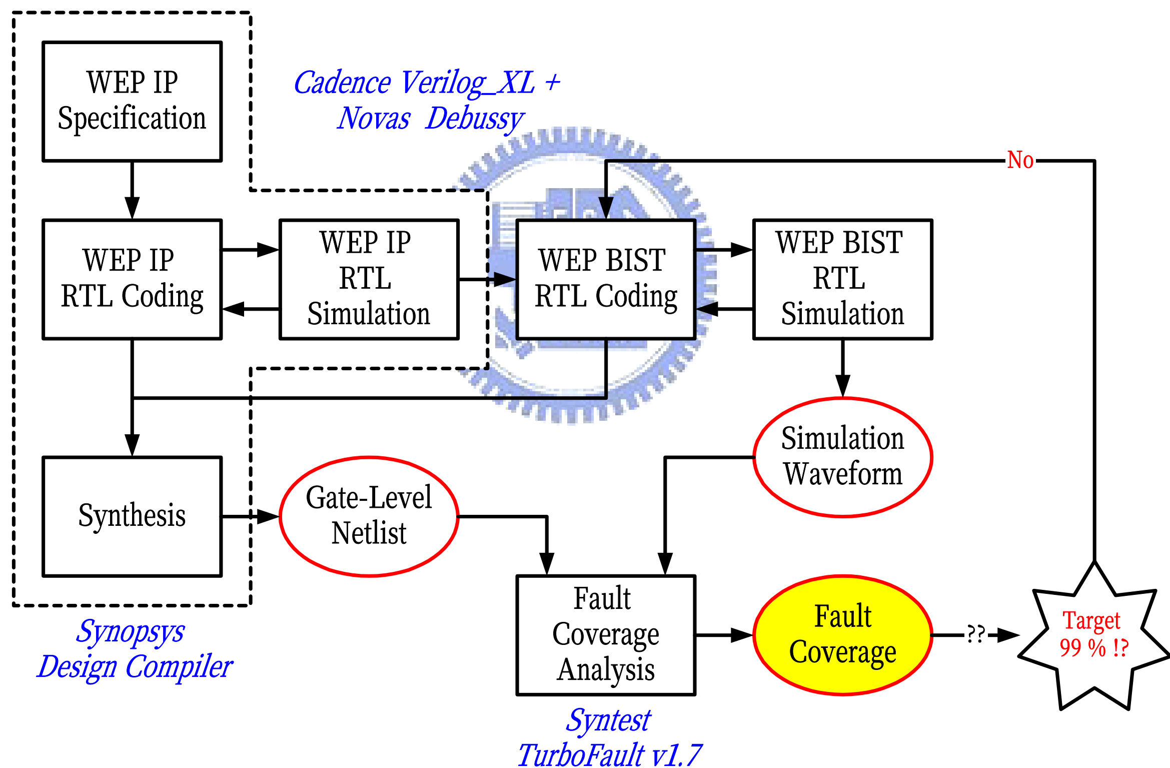


Figure 2.9 The Design Flow of the WEP IP with BIST

When the gate level netlist of WEP IP is ready, we begin to design a BIST circuit for this IP. The same design flows described below are also followed, for us to create the BIST module for the WEP IP. Please refer to Chapter 4 for the detail description of the BIST

design for the WEP IP.

1. Define the architecture and create the RTL code of BIST.
2. Pass the simulations of the full IP including WEP IP and BIST.
3. Synthesize and transfer all RTL codes to gate level netlist.
4. Simulate and verify the “Fault Coverage” of WEP IP with the gate level netlist and simulation patterns. The BIST module has to be re-structured and re-deigned until we get an approved “Fault Coverage”.

2.2.2 Specification

This “WEP IP” is a reusable intellectual proprietary (IP) for IEEE 802.11a/b wireless network decrypting and encrypting function. Because of the embedded PRNG and CRC-32, this module can be a platform for “Design for Test” verification of NCTU VLSI Testing Lab.

The features of this WEP IP are described in below:

- A reusable independent Wired Equivalent Privacy (WEP) decryption and encryption module for IEEE 802.11 a/b.
- Fully compatible IEEE 802.11 a/b WEP specification.
  - Embedded RC4 decrypt and encrypt algorithm.
  - Supply both 64 and 128 secret keys.
  - Built-in one 256 bytes synchronous SRAM for S-Box.
- A half-duplex module. Only decrypt mode or encrypt mode can be enabled at a time.
- The maximal operating frequency is 125MHz with 0.35um processing.
- The maximal latency to encrypt or decrypt is 6 system clocks. In other words, the maximal through rate is 166.67 Mbps.
- Provide a high performance Built-in Self Test module in this IP.

In order to verify this IP with a single chip, we specify a few pin count assignment as defined in Table 2.1. The last three pins, “BIST_Mode”, “BIST_Good”, and “BIST_End” are defined for BIST mode. Please refer to chapter 4 for BIST mode.

Name	I/O	Function Description
Clock	I	System Clock, The maximal working frequency is 125MHz.
Reset	I	System Reset, Active High, The active width must larger than two clock periods.
Mode_Sel	I	Mode select signal; 1 : Encrypt Mode, 0 : Decrypt Mode
Din[7:0]	I	In decrypt mode, Din[7:0] are the secret key, ciphertext, and the encrypted ICV data bus input. In encrypt mode, Din[7:0] are the secret key, and plaintext input.
Din_STB	I	Input data (Din[7:0]) valid strobe pulse. Active High
Key_Sel	I	When “Key_Sel” is active, the “Din[7:0]” is secret key input.
Data_Sel	I	When “Data_Sel” is active, the “Din[7:0]” is ciphertext input in decrypt mode or plaintext input in encrypt mode.
ICV_Sel	I	This signal is only active in decrypt mode when the ICV data is valid on Din[7:0].
Dout[7:0]	O	In decrypt mode, Dout[7:0] are plaintext output. In encrypt mode, Dout[7:0] are ciphertext and the encrypted ICV output.
Dout_STB	O	Output data (Dout[7:0]) valid strobe pulse.
SBox_RDY	O	This signal is active when the RC4 seeding is done and the S-Box is ready to generate the key sequence.
WEP_ICVo	O	In decrypt mode, “WEP_ICVo” is the flag to show the ICV comparing result. In encrypt mode, the Dout[7:0] are the encrypted ICV output when “WEP_ICVo” is active.
WEP_ERR	O	This signal is active when the WEP procedure is stopped with any error. The signal will be cleared automatically while the procedure is restarted.
BIST_Mode	I	Set “BIST_Mode” to enable the BIST mode.
BIST_Good	O	The BIST result, “1” is passed and “0” is failed.
BIST_End	O	The BIST procedure is done when this signal is set.

Table 2.1 The pin description of WEP IP

2.2.3 Architecture

Referring to Figure 2.10 "The Block Diagram of WEP IP", the top level is named "WEP_IP". All input signals are at the left side, and all output signals are at the right side.

There are two major blocks in "WEP_IP". The green block is a synchronous SRAM 256 X 8. This is an unsynthesisable behavioral model. Being the S-Box of WEP, this module stores the secret keystream for WEP encryption or decryption. The behavior model is only for simulation. This module is impossible to be synthesized by the synthesis tools, such as "Synopsys Design Compiler" at the present stage. In the real chip implementation, we can use the "Memory Compiler" EDA tools provided by foundries or custom design circuit and layout to generate this memory macro.

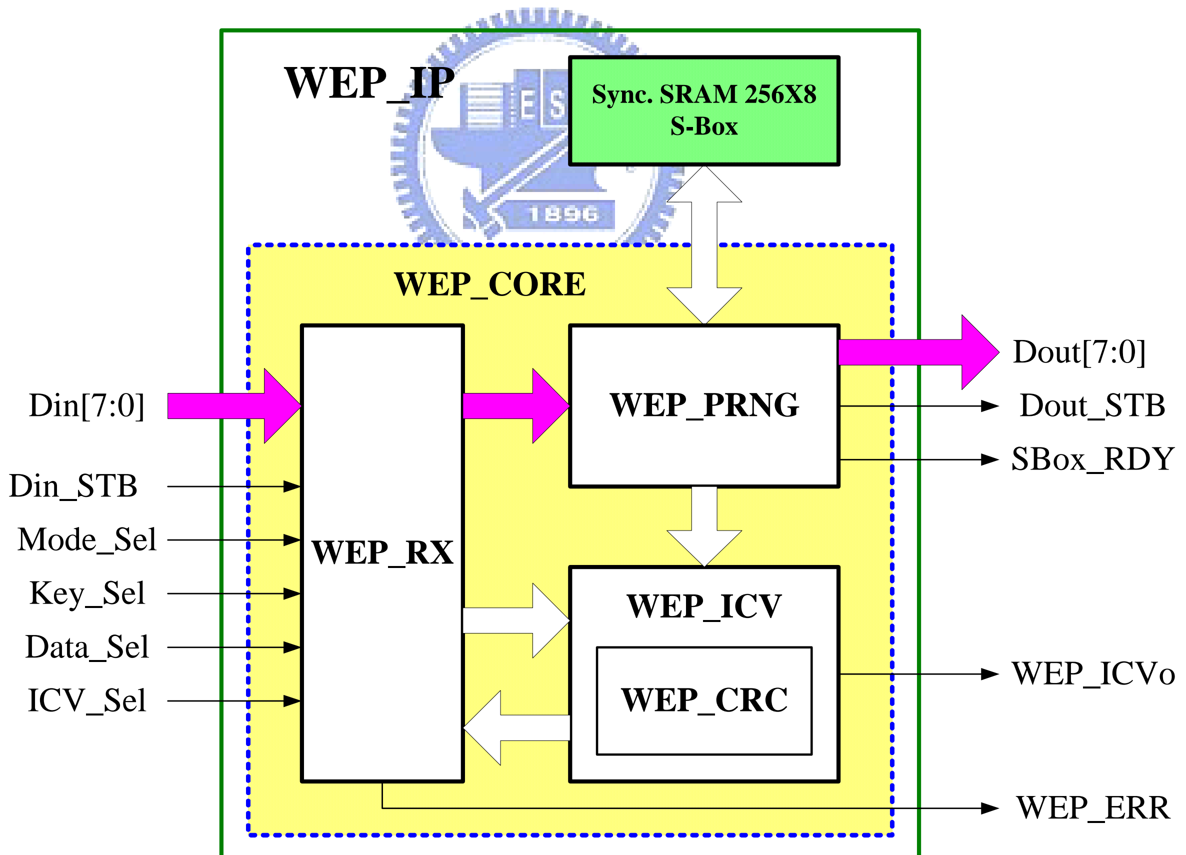


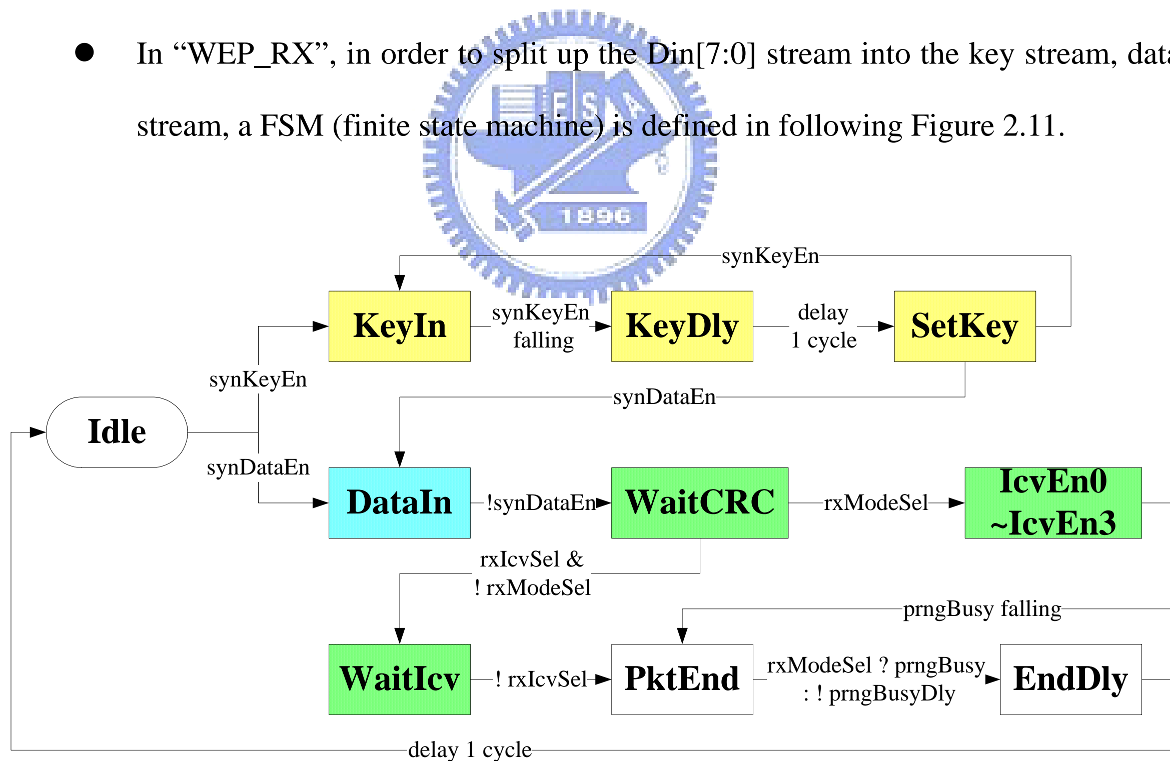
Figure 2.10 The Block Diagram of WEP IP

In "WEP_IP", the yellow block is called "WEP_CORE". This module is the top level

of all synthesizable logic circuits. All input and output signals are also connected with this module.

We divide “WEP_CORE” into three sub-modules by functions. The first one is named “WEP_RX”. “WEP_RX” is entrance module of “WEP_CORE”. It manages three major functions described as follows:

- “WEP_RX” synchronizes all external input signals with the internal clock and transfers them to other sub-modules to avoid all unexpected timing violation in RTL design, such as metastability violation.
- “WEP_RX” also collects the 64/128 bits secret key from the Din[7:0] bus. It will set a “secret key ready” signal to inform “WEP_PRNG” when received a correct secret key set.
- In “WEP_RX”, in order to split up the Din[7:0] stream into the key stream, data stream, a FSM (finite state machine) is defined in following Figure 2.11.



Yellow Blocks : Key Stream Input
Green Blocks : ICV Stream Input
Blue Blocks : Data Stream Input
White Blocks : Control Block

Figure 2.11 The FSM of WEP_RX

The second sub-module in “WEP_CORE” is named “WEP_PRNG”. As implied in the name, “WEP_PRNG” is the kernel to generate a pseudo-random number with the RC4 algorithm as defined in Section 2.1.4.

The main FSM of “WEP_PRNG” is defined in Figure 2.12 below.

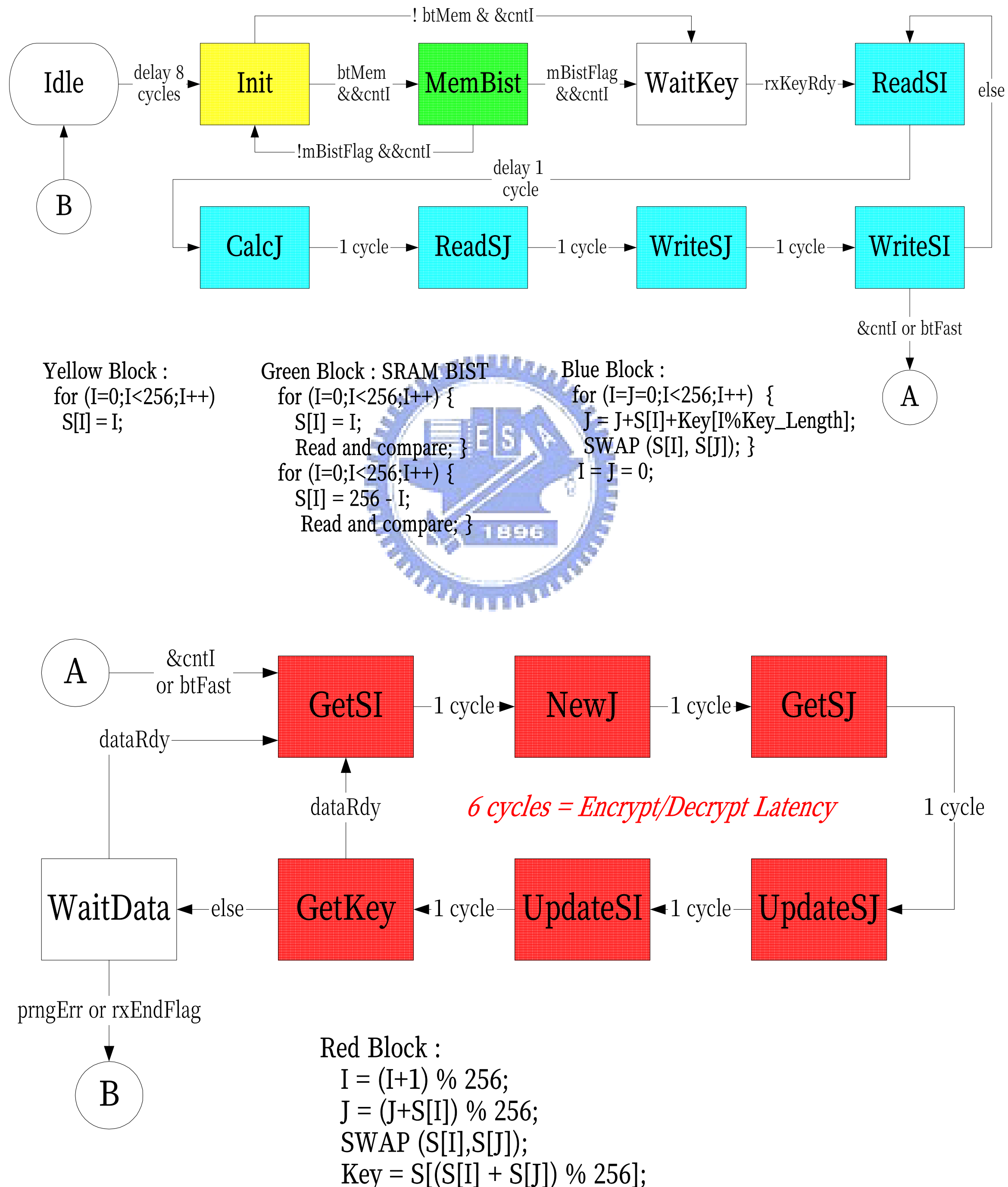


Figure 2.12 The FSM of WEP_PRNG

The comment text on Figure 2.12 is the equation of the RC4 PRNG algorithm.

In the general function, “WEP_PRNG” will find out the secret key from the S-Box and generate the encrypted or decrypted data with the input data. For this reason, the other duty of “WEP_PRNG” is to control the S-Box (a Sync. SRAM 256x8).

The third sub-module in “WEP_CORE” is named “WEP_ICV”. In the decryption mode (Mode_Sel = 0), the “WEP_ICV” checks the 32 bits ICV input data decrypted from the “WEP_PRNG” module and set the “WEP_ICVo” signal when the ICV check is successful.

In the encryption mode (Mode_Sel = 1), the “WEP_ICV” transfers the CRC-32 output data to “WEP_RX” module. Then, “WEP_RX” transfers this CRC-32 data to “WEP_PRNG” module to generate the encrypted ICV output data.

2.2.4 Simulation Results

Referring to Figure 2.13, the flowchart in the left side is the procedure of WEP encryption and the comments in the right side are conditions of this procedure.

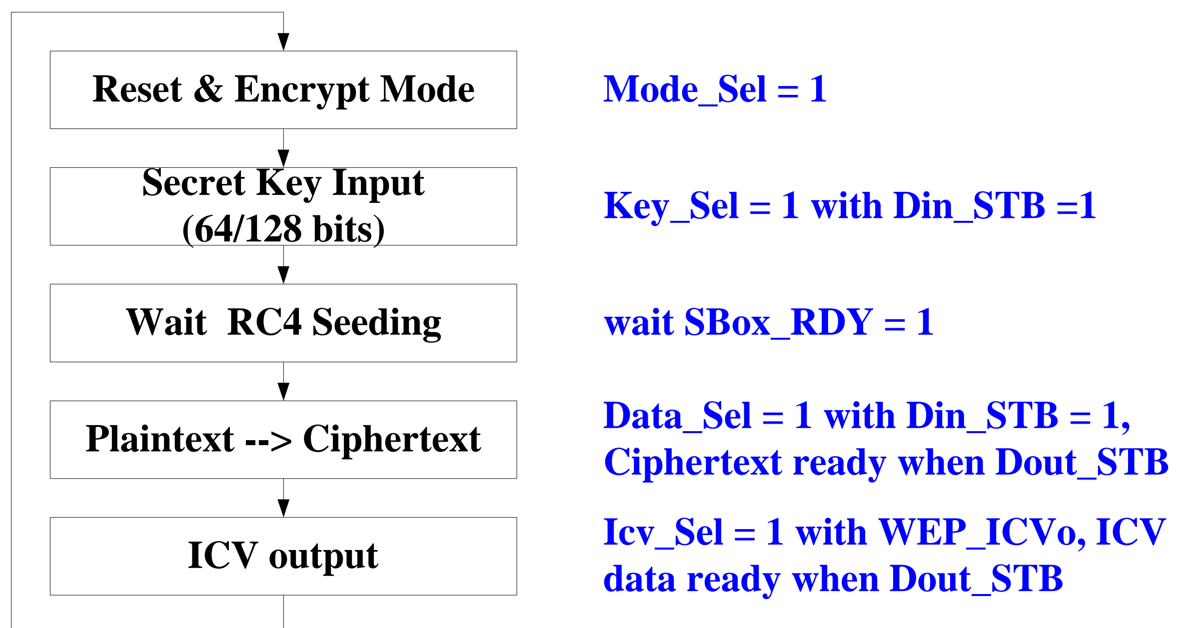


Figure 2.13 The WEP encrypt procedure

As soon as “WEP_RX” receives the complete secret key from the external application circuit, “WEP_PRNG” processes the “RC4 Seeding” until the “Sbox_RDY” signal is set. This process takes 1536 cycle time. Setting “Sbox_RDY” means that the process of “RC4 Seeding” is finished and the “S-Box” is ready for encipherment or decipherment process. For the red blocks in Figure 2.12 “The FSM of WEP_PRNG”, there are six steps to transfer an incoming plaintext to a ciphertext. For this reason, the latency delay to encrypt or decrypt a byte is six cycles time. We can find out the encryption procedure and the timing relationship on Figure 2.14 “The encrypt chart” and Figure 2.15 “The simulation waveform of encipherment”.

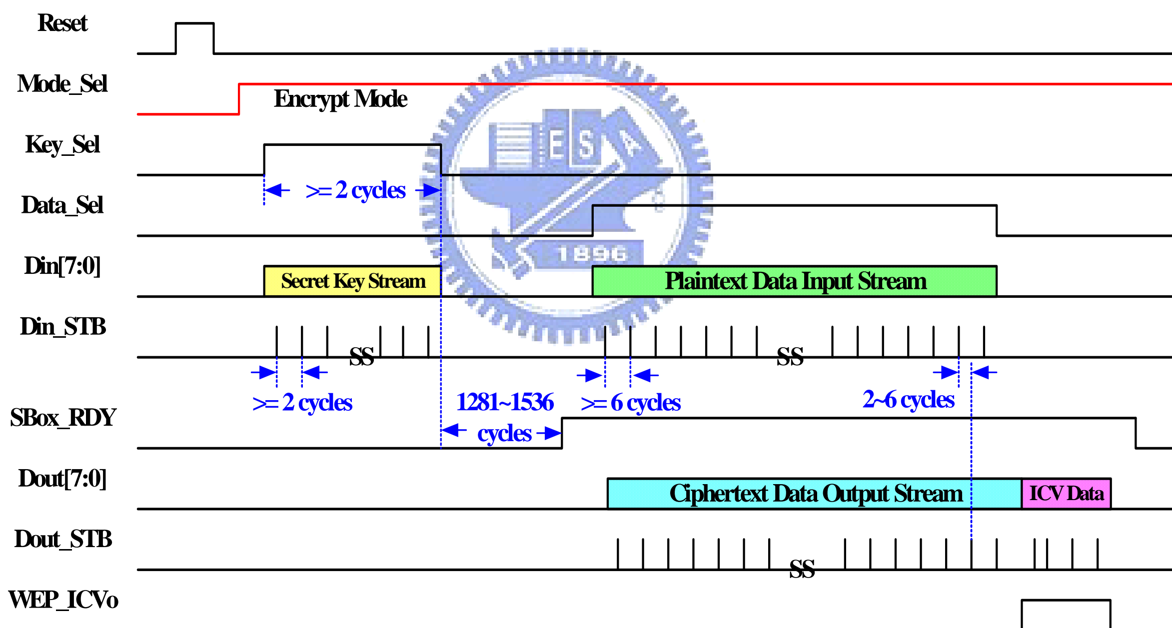


Figure 2.14 The encryption chart

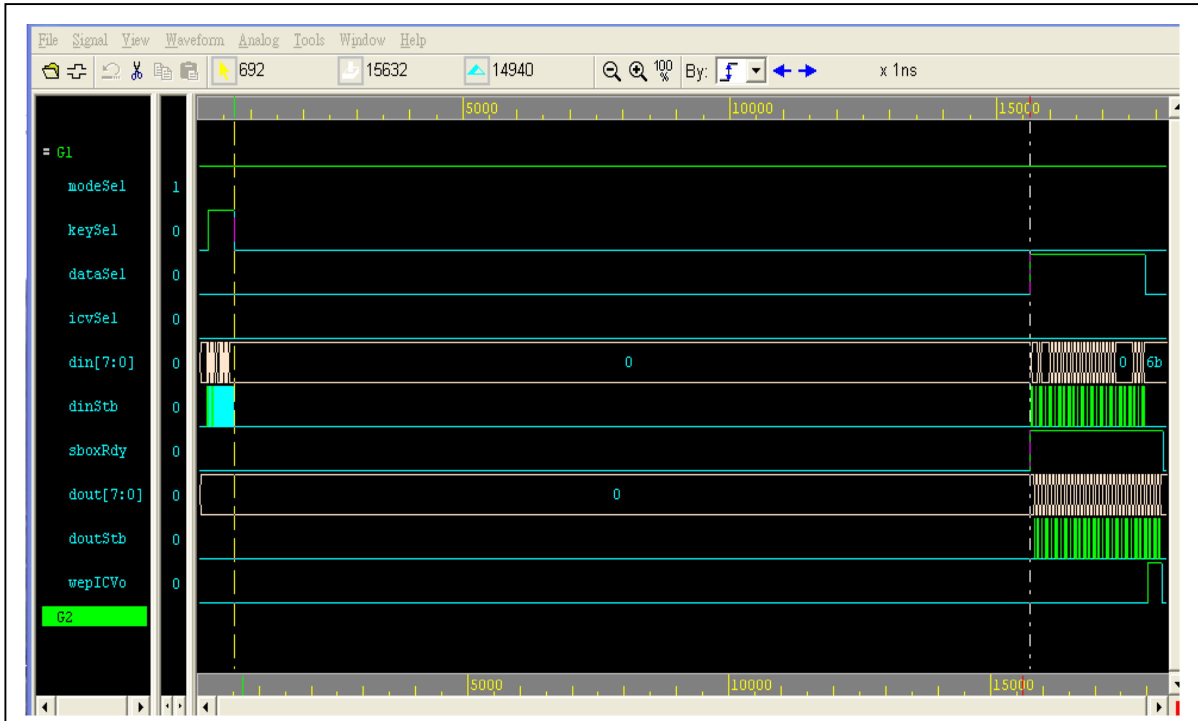


Figure 2.15 The simulation waveform of encipherment

Like the encipherment procedure, the decipherment procedure uses the secret key to processes the “RC4 Seeding” until the “S-Box” is ready. Then WEP decrypts the incoming ciphertext with the keystream. Please refer to Figure 2.16 “The WEP decrypt procedure”.

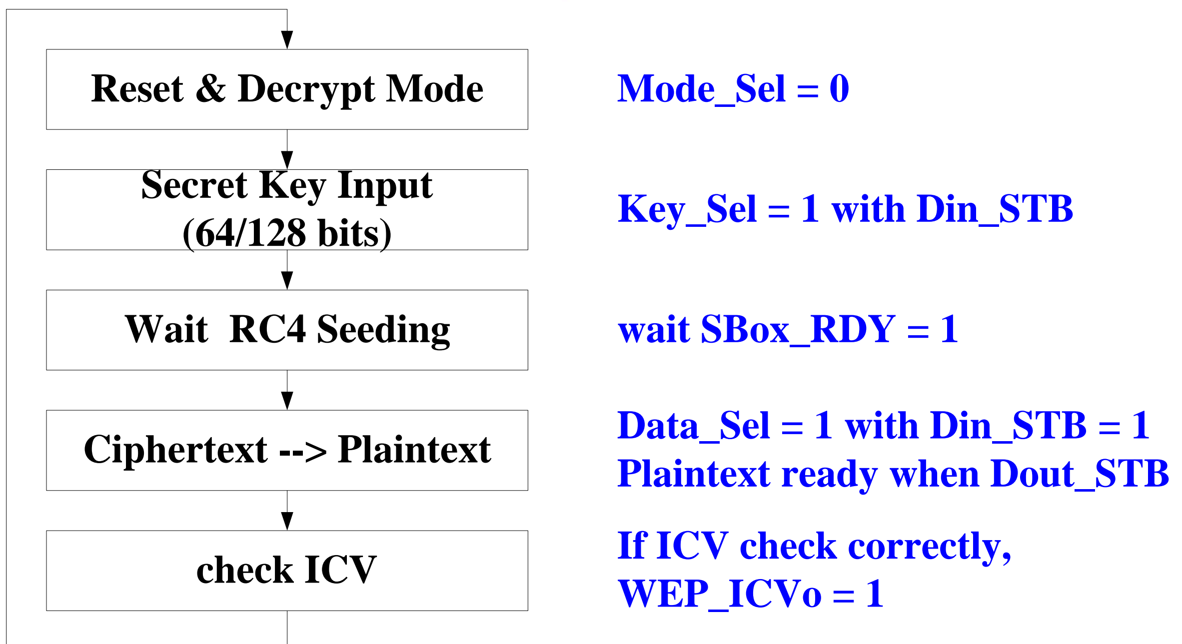


Figure 2.16 The WEP decrypt procedure

Figure 2.17 shows the decryption chart in an ideal case. Figure 2.18 shows the simulation waveforms of decipherment

.

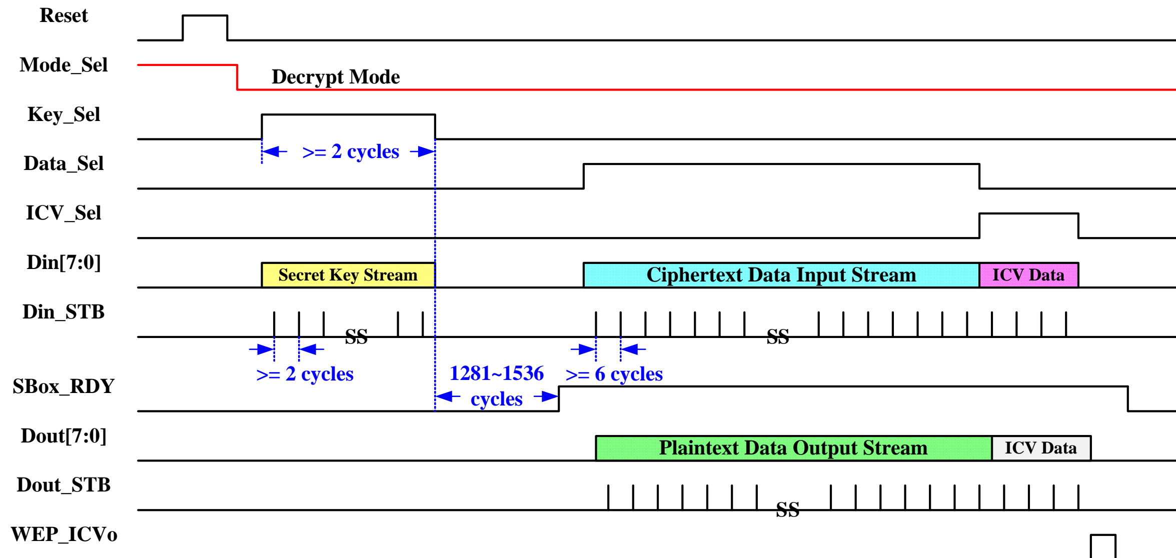


Figure 2.17 The decryption chart

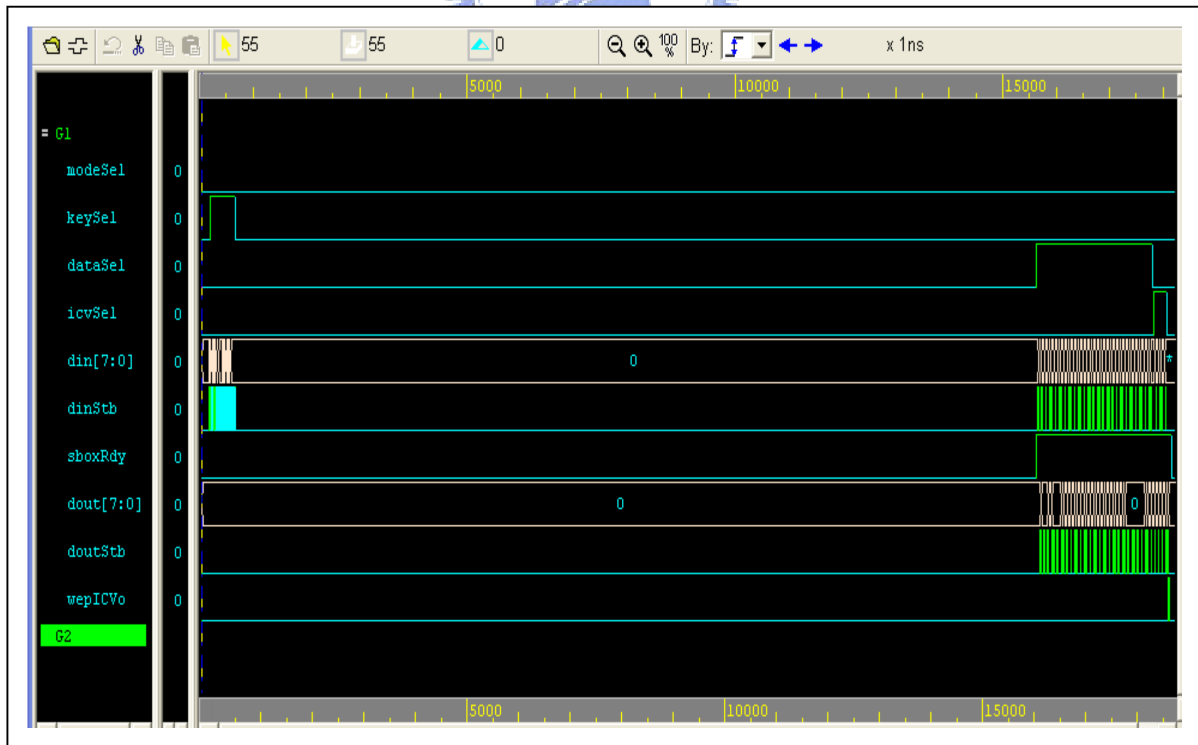


Figure 2.18 The simulation waveform of decipherment

2.2.5 Synthesis Results

After passed all simulations in RTL design, we synthesis this WEP IP from Verilog RTL code to the tsmc 0.35um process standard cell netlist with the Synopsys EDA tool “Design Compiler”. The synthesis results are described in Table 2.2 below.

Process: tsmc 0.35 standard cell library (cb35os142) Area: Logic = 4139 gate count (1 gate = 1 2-input NAND cell) Marco = The synchronous SRAM 256x8 The maximal operating frequency = 125MHz in the worst case The critical path is the path from SRAM data output to “WEP_PRNG” sub-module.

Table 2.2 The synthesis results of the WEP IP

2.3 Summary

In order to implement our BIST method in a general digital design, we follow the reuse methodology of VLSI IP to finish an optimal WEP IP based on the IEEE 802.11 standard. In this chapter, we have described the detail specification, pin assignment, architecture and the FSM of each sub-module in our WEP IP design. We also go through simulations and the synthesis to create the WEP IP as a platform for BIST design in our later thesis.

Chapter 3 Traditional Testing Methods for WEP IP

For design or product engineers, how to verify the function and guarantee the quality of chips is an important topic. There are many theories and methods which such as functional testing, and scan chain testing can help engineers to verify or test their chips. In this chapter, we will describe how to use these two traditional testing methods to test the WEP IP.

3.1 Functional Testing

The first traditional testing method for the WEP IP is "functional test". A functional test is used to verify that the model or logic behavior as it was intended in specifications. A functional test is application of functional, operational, or behavioral vectors to the circuit under test. If the vectors are not evaluated for their structural coverage, then the vectors are generally referred to as "design verification" vectors. Section 3.1.1 describes the generating flow of the WEP IP's functional patterns.

Many designers still rely on functional vectors to accomplish the manufacturing, or product test. A functional test could be a good choice if the device is small or if the majority of the functional vectors exist from a previous but similar design. However, there are several problems with the use of functional vectors for any kind of manufacturing test on the modern VLSI design.

First, functional vectors are very good for determining behavior, but are not especially good or efficient for structural verification.

Second, functional vectors must be evaluated to verify the structural fault coverage. This is an extra and difficult task. In most cases, the fault coverage of functional vectors is very difficult to achieve the required quality level.

Third, functional vector are designed to verify circuit behavior, and they are not as efficient as “deterministic” structural vectors. In modern SoC and VLSI designs, the test data volume is becoming a critical issue of the test cost. More test time and on-board memories are requested to achieve the expected quality [9].

Finally, the tester’s working frequency limits the maximal operating frequency of the functional vectors. In some cases, an engineer needs more expensive tester to verify his chips at-speed.

3.1.1 Flow of Function Pattern Generation

As in the description in Chapter 2, the WEP IP is an independent reusable privacy module and its logic behavior is intended in specifications. Both encryption and decryption processes were verified in the design stage. We can transfer the behavioral function patterns into function vectors. Figure 3.1 shows the total function vectors of the WEP IP.

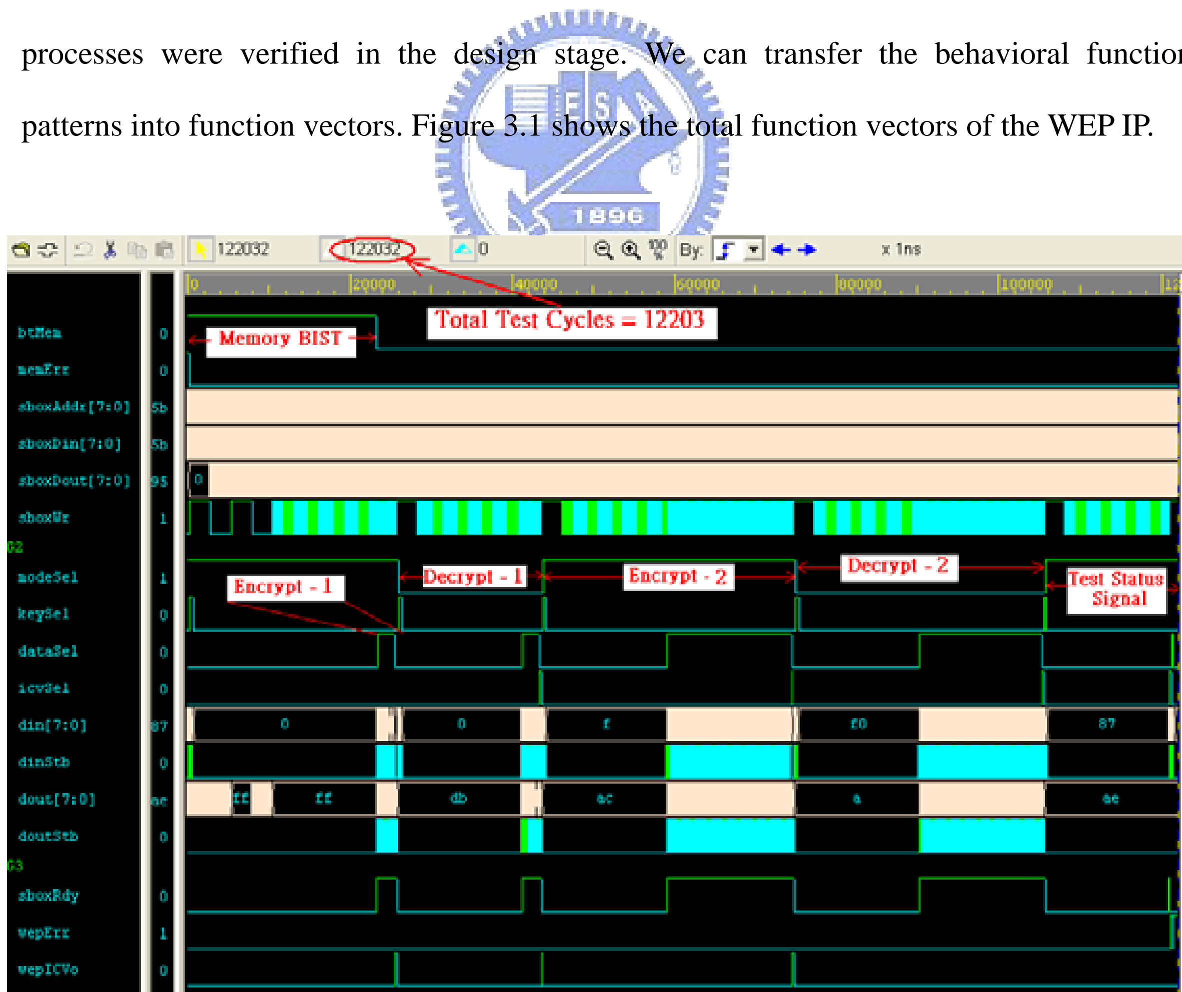


Figure 3.1 The WEP IP simulation waveform of functional vectors

The first stage of the functional test vector is “Memory BIST” to verify the embedded 256-bytes SRAM. We use some design skills to reduce the extra gate count for the memory BIST logic. Please refer to Section 3.1.2 for more detailed description.

The second and the third stages are “Encrypt 1” and “Decrypt 1” to verify the encryption and decryption function with a 64-bits secret key.

In order to achieve the expected quality, the extra “Encrypt 2” and “Decrypt 2” stages are added into the procedure. Both these two processes are verified with a 128-bits secret key.

At the last stage “Test Status Signal”, we try to trigger all status output signals with some expected conditions and function patterns to verify these ports.

Finally, we must collect and transfer these function patterns into the test vectors that base on the types of test mechanisms. In order to reduce the length of test vectors, the data compression such as RZI and NRZI is also necessary. Figure 3.2 shows an example of test vectors without compression.

```
crbdddddddddmkdidddddddddwwsm|1011000000001100HLHLLLHLLLLLL
letiiiiiiiiioeacoooooooooeebe|1011000000011100HLHLLLHLLLLLL
ksMnnnnnnnnndytvuuuuuuuuuppom|1011000000001100HHHLLHHLLLLLL
 ee01234567SeSaStttttttttEIxE|1010000000001100HLLHLLLHLLLLL
 tm        tSeSe01234567SrCRr|1010000000011100HLLHLLLHLLLLL
           belel        trVdr|1010000000001100LLLLLLLLLLLLL
            l l         b oy |1010000000001000LLLLHHLLLLLLL
                             |1010000000001000HLLLHHLLLLLLL
1110000000001000LLLLLLLLLLLLH|1010000000001000LHLLHHLLLLLLL
1011001000111100LLLLLLLLLLLLH|1010000000001000HHLLHHLLLLLLL
1011001000101100LLLLLLLLLLLLH|1010000000001000LLHLHHLLLLLLL
1011110011001100LLLLLLLLLLLLH|1010000000001000HLHLHHLLLLLLL
1011110011011100HLLHLLLHLLLLL|1010000000001000LHHLHHLLLLLLL
1011110011001100LLLLLLLLLLLLL|1010000000001000HHHLHHLLLLLLL
1010000000001100HLLLLLLLLLLLL|1010000000001000LLLHHHLLLLLLL
1010000000011100HLLLLLLLLLLLL|1010000000001000HLLHHHLLLLLLL
1010000000001100HHLLLHLLLLLLL|1010000000001000LHLHHHLLLLLLL
1011001000101100HLHLLLHLLLLLL|1010000000001000HHLHHHLLLLLLL
1011001000111100HLHLLLHLLLLLL|1010000000001000LHHHHHLLLLLLL
1011001000101100HHHLLHHLLLLLL|1010000000001000HHHHHHLLLLLLL
1011110011001100HLLHLLLHLLLLL|1010000000001000LLLLLLLHLLLLL
1011110011011100HLLHLLLHLLLLL|1010000000001000HLLLLLLHLLLLL
1011110011001100LLLLLLLLLLLLL|1010000000001000LHLLLLLHLLLLL
```

Figure 3.2 A test vector example without compression.

In Figure 3.2, “1”, “0”, or “Z” are the symbols for the inputs when they are “high”, “low” or “high-impedance” respectively and “H”, “L” or “X” are the symbols for the outputs when they are “high”, “low” or “unknown” respectively.

3.1.2 Testing for Embedded Memory

In the WEP IP, there is an embedded memory. This section describes how to use the traditional testing method to verify the embedded 256-bytes SRAM.

In the modern SoC design, the topic of memory testing is not limited to just testing the embedded memory by using a tester. Three common test methods can be applied to verify the embedded memory. These test method options are: Embedded Microprocessor Access; Direct Memory Access by Tester; and *Memory Built-in Self Test* (MBIST). Each of these methods has trade-offs that involve: cost-of-test, chip area, chip pin requirements, chip-level timing and chip power requirements [9].

In our design, we implement an MBIST into the WEP IP to verify the embedded SRAM. Considering the extra area overhead, we prefer the “MATS” [10] algorithm and try to combine the MBIST controller with the original logic.

As everyone knows, the “MATS” is the simplest algorithm to verify a memory. The “MATS” algorithm is described as follows:

$$\{ \Updownarrow (W\ 0);\ \Updownarrow (R\ 0, W\ 1);\ \Updownarrow (R\ 1) \} \qquad \textbf{(5)}$$

Every cell of the memory is written both value “0” and “1” and is read and verified. So the “MATS” can verify all *stuck-at faults* (SAF) in memory cells and some *address decode faults* (AF). The complexity of “MATS” is 4n. In our design, we need extra 1024 cycles to test the embedded 256-bytes SRAM with the “MATS” algorithm.

In order to reduce the extra area and test time in MBIST, we try to combine this “MATS” algorithm with the original logic. First, we review the formula for RC4 described in Section 2.1.4. The first step of “RC4 Seeding” is defined as follows:

$$\text{For } (I = 0 ; I < 256 ; I = I + 1) \quad S[I] = I; \qquad (6)$$

We can transfer equation (6) above into equation (7) below.

$$\{ \uparrow (W(\text{Address})) ; \} \qquad (7)$$

If we hope to combine the "MATS" and "RC4 Seeding" algorithm to reduce the extra gate-count and test time, we must add equation (8) below into our MBIST procedure.

$$\{ \uparrow W (\overline{\text{Address}})) ; \} \qquad (8)$$

Because the address range of the WEP IP's embedded 256-bytes SRAM is from 00h to FFh, we can use the same 8-bit address counter to generate both address and data for the test algorithm. Then, we can re-define the "MATS" MBIST algorithm as the following equation (9).

$$\{ \uparrow (W(\overline{\text{Address}})); \uparrow (R(\overline{\text{Address}})); \underbrace{\uparrow (W(\text{Address}));}_{\text{Eq. (7)}} \uparrow (R(\text{Address})); \} \qquad (9)$$

In equation (9) above, we can find equation (7) that is described in the red text. So, we can merge the "MATS" and the FSM of RC4 PRNG to save test time of equation (7). By the way, we can reuse the same address counter that originally existed in "WEP_PRNG" module to generate the test address and data for the "MATS". As shown in Figure 2.12, the "Memory BIST" state is inserted between the "Initial" and "Wait Secret Key" state. In order to avoid the redundant MBIST process, we also create a status signal "mBistFlag" to record the MBIST procedure. If this status signal is set, the FSM will bypass the "Memory BIST" state to guarantee that only one MBIST cycle will be done.

In summary, this MBIST algorithm is good to save extra gate-count and test time. There are only 284 gates extra to build up this "MATS"-comparable MBIST in the WEP IP.

However the low fault coverage of the “MATS” algorithm is its drawback. We will re-design a “14N MARCH C+” MBIST into our WEP BIST that will be described in Section 4.2.4.

3.1.3 Fault Coverage Results

As described in Figure 2.9 “Design Flow of WEP IP with BIST”, we used the EDA tool “Syntest TurboFault” [8] to fault simulate the WEP IP with the patterns described in Section 3.1.1 to evaluate the fault coverage. The fault coverage report for those functional test vectors are shown in Table 3.1.

Item	Number	Percentage
Optimal Fault Coverage	**--**	**92.69 %**
Total Faults	**13042**	**100.00 %**
Hard Detected Faults	**11106**	**85.16 %**
Probably Detected Faults	**983**	**7.54 %**
Undetected Faults	**950**	**7.28 %**

Table 3.1 WEP IP fault coverage report with functional test vectors

1896

The total testing time is 12203 clock cycles. In other words, if the tester operating frequency is 10MHz, it will take about 1.22 ms to finish the procedure. The total gate count is 4454 gates. To compare the gate count with the WEP IP gate count described in Table 2.2, the area overhead is 315 gates, almost 7.07 %. The extra gate count was caused of the MBIST controller described in the last section.

In order to improve the fault coverage, additional functional patterns such as “Encrypt 2” and “Decrypt 2” must be included.

3.2 Test Method by Scan-Chain

Because “higher frequency”, “higher pin-count”, “higher levels of integration”, and “higher complexity” drive up the cost of the test platforms required to test modern chips, the scan-based testing is famous in modern SoC design. Today, there are many EDA tools, which can help designers to insert scan-chains and generate effective vectors for their chips. So we use the Syntest’s EDA tool “TurboScan” [11] to generate the scan-chain in our WEP IP and discuss the advantages and disadvantages in different design styles.

For an embedded module or IP in a SoC design, there are many testing problems, which must be solved when a scan-chain is to be inserted into this IP.

1. If the IP is a black-box and hard-macro, end-users can not directly control and observe the internal nets of the embedded module. It is impossible to verify this IP with scan-chains to get its fault coverage report.
2. If the IP is a firm-macro, i.e. a pure gate-level netlist, end-users cannot re-synthesize and insert the scan-chains because of the timing issues.

In next sections, we will describe and analyze the three scan-chain methods implemented in the WEP IP.

3.2.1 ATPG with a WEP Hard-Macro

What is a hard-macro? A hard-macro is a black-box in VLSI design that end-users can not control and observe its internal nets and can not insert the scan-chain into these unknown logic. The IP provider can provide a hard-macro IP with local scan-chain. But there are many uncontrollable *parallel inputs* (PI) and unobservable *parallel outputs* (PO) in the hard-macro IP, as described in Figure 3.3 below. Unfortunately, these uncertain PI and PO will cause low fault coverage as shown in Table 3.2.

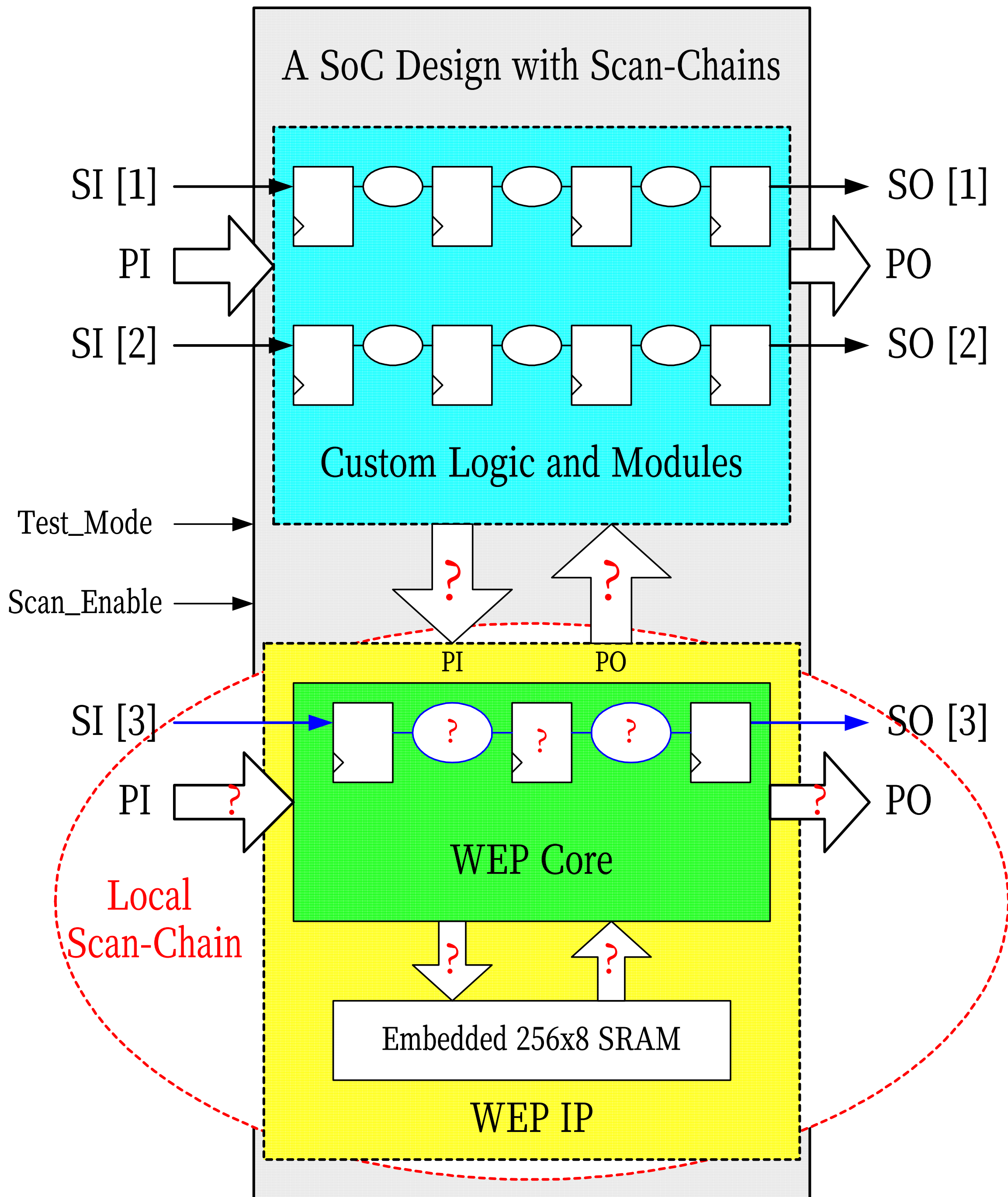


Figure 3.3 Local Scan-Chain in a Hard-Macro IP

Fault Coverage	79.49 %
Total Gate-Count	4836
Area Overhead	14.41 %

Test Patterns	**66**
Chain Length	**305**
Estimated Cycles	**20632**

Table 3.2 Fault coverage report of the Hard-Macro WEP IP

3.2.2 ATPG on WEP IP without Memory Wrapper

If we provide the WEP IP RTL code or the gate-level netlist to end-users, the end-user can implement his scan-chains into all logic including “WEP_CORE”. Because the WEP IP includes an embedded memory array, the memory looks like a large “blocked area” within a scan logic test area. Everything in the downstream of the memory may experience a loss of controllability because the output from the memory array is unknown.

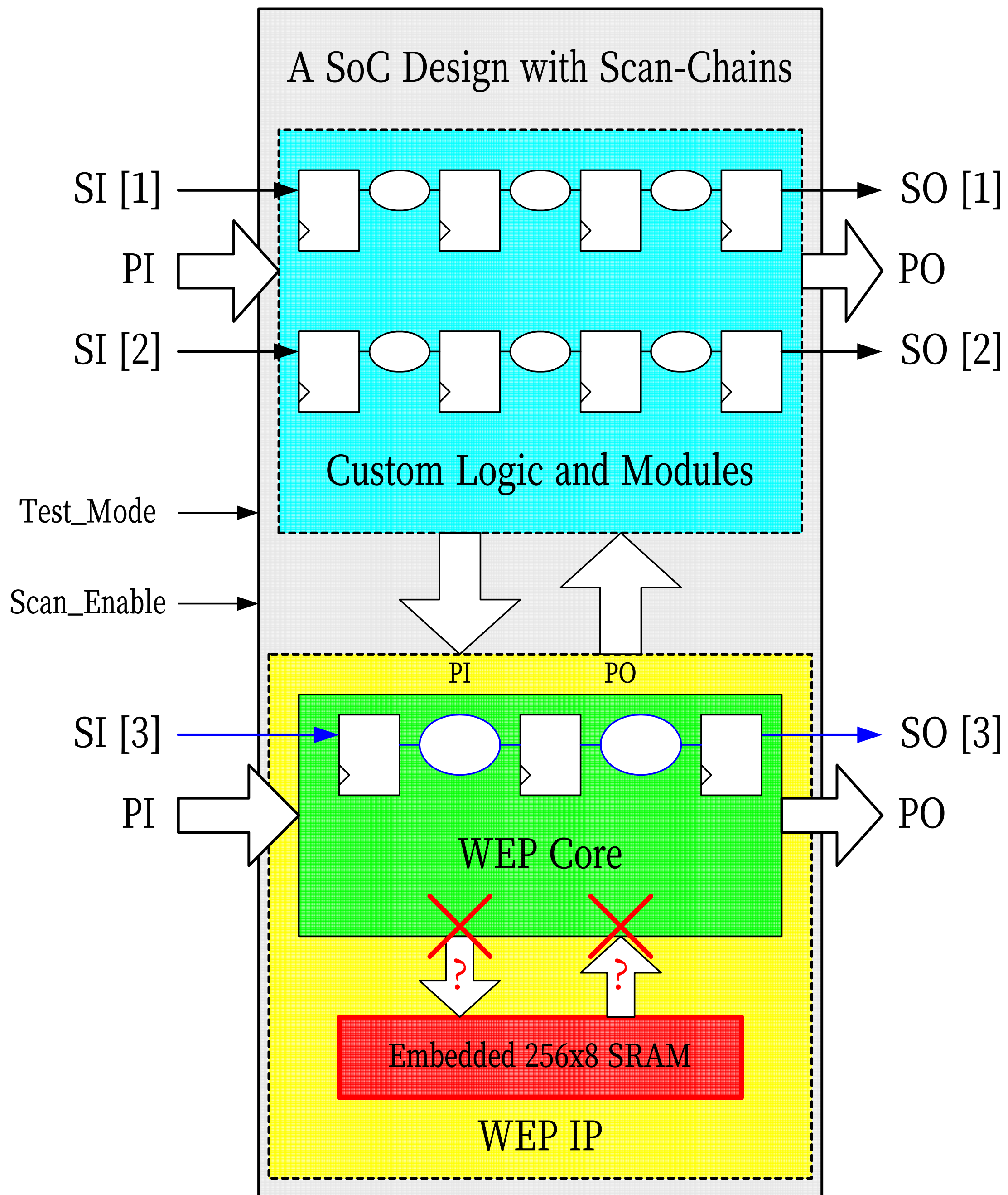


Figure 3.4 ATPG on WEP IP without Memory Wrapper

Please refer to Figure 3.4 above, we can find that all signals connected with the embedded SRAM are uncontrollable and unobservable. If we leave this large "blocked area" in the WEP IP, the low quality fault detection is expected as Table 3.3.

Fault Coverage	90.22 %
Total Gate-Count	4836
Area Overhead	14.41 %

Test Patterns	**87**
Chain Length	**305**
Estimated Cycles	**27100**

Table 3.3 Fault coverage report of the WEP IP without Memory Wrapper

3.2.3 ATPG on WEP IP with Memory Wrapper

In order to get as much coverage of the area between the scan architecture and the memory test architecture, several methods are used for the interaction of a memory with a general chip scan logic. The most general method is to insert a memory wrapper to isolate the "blocked area" from scan-chains, as illustrated in Figure 3.5 [9].

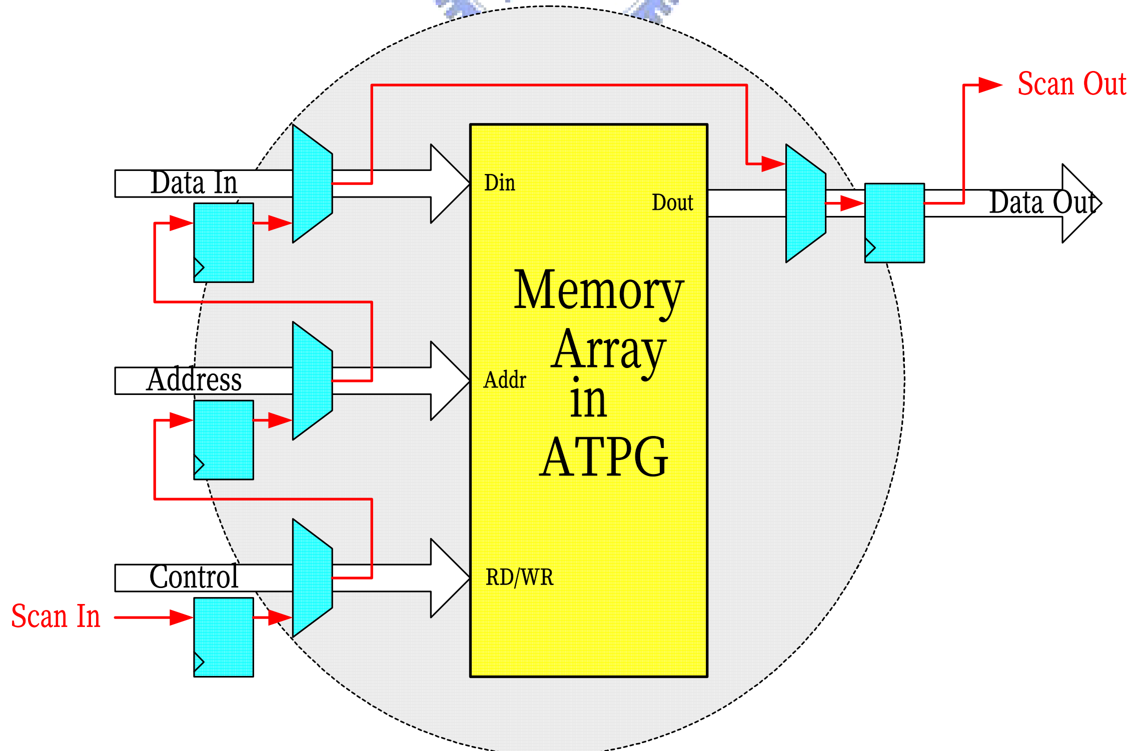


Figure 3.5 Memory Wrapper in ATPG

In modern digital design methods, most ATPG EDA tools have the ability to handle memory interaction automatically. In our WEP IP, we try to insert the memory wrapper by designing some extra circuit to string the inputs and outputs of the embedded SRAM as described in Figure 3.5. There are totally 25 flip-flops and 25 multiplexers for the memory wrapper. In order to isolate the memory macro from the scan-chain, these unavoidable wrapper cells are the extra area overhead. The fault coverage report of the ATPG with the memory wrapper is described in Table 3.4.

Fault Coverage	98.91 %	**Test Patterns**	**101**
Total Gate-Count	5124	**Chain Length**	**330**
Area Overhead	19.22 %	**Estimated Cycles**	**33962**

Table 3.4 Fault coverage report of the WEP IP with Memory Wrapper

We get an exhilarating fault coverage report with this ATPG method. The area overhead is 19.22% because the total gate-counts of the WEP without MBIST are only 4139. If we only consider the area overhead of the scan-chain and memory wrapper in this design, the original gate-count of the "WEP with MATS MBIST" is 4423. So, the pure area overhead of the scan-chain with memory wrapper is almost 14%. In summary, for a small design, the ATPG area overhead is appreciable.

3.3 Summary

Usually, designers do not pay much attention to take care the testing issues during the design phase. They always try to finish their design on specification and on schedule then leave these testing problems to functional vectors or ATPG EDA tools. Undoubtedly, most ATPG EDA tools can handle the full-chip scan smartly and effectively. Here scan vectors

are more efficient than functional vectors, and a higher fault coverage report can be obtained.

However, the full scan still has some disadvantages and limitations as follows.

1. The scan method cannot reach an embedded “blocked area”, such as embedded memories and a hard-macro IP.
2. A designer must handle the clock skew issue on the scan clocks when the scan clocks string many different clock domains. An extra effort and extra area overhead should be paid to fix these timing violations caused by these different clock delays.
3. In order to reach the quality level fault coverage, the number of ATPG patterns and the testing cycles are appreciable. For an IP design, it is impossible to prepare a larger of memory to store these test vectors in the built-in self test circuit. This is why we try to design a BIST module in the WEP IP without any memory storage.

Chapter 4 A Built-In Self Test Design for WEP IP

Because there are many disadvantages and limitations when implementing an effective testing method into the independent IP, we would like to design a BIST circuit to qualify the WEP IP.

4.1 Features

Refer to the disadvantages and the advantages of the traditional testing methods described in Chapter 3, we work out the features of the BIST module as follows.

1. This BIST module can provide the high fault coverage for our WEP IP. The target fault coverage value is 99%.
2. It provides an at-speed testing method to detect delay faults in circuit. The target operating frequency is 125MHz in 0.35um process.
3. Only a few of extra pin must be provided for the BIST module. In other words, a friendly interface is necessary.
4. The BIST module will generate the testing patterns and check the correctness automatically. There is no extra memory to store the vectors.
5. Use the least pattern to meet these features above.
6. Use the least extra gate-count to create this BIST module.

4.2 Description of the BIST for WEP IP

In this section, we describe the general idea and the method of the BIST design.

4.2.1 Architecture

Referring to Figure 4.1 below, the general idea of the testing method is to design an extra external BIST module to generate and multiplex the input signals that are transferred

to the WEP IP. The BIST module not only generates and feed the testing vectors into the WEP IP, it also collects and calculates the output signals from the WEP IP to verify the checksum value. If the calculated value matches with the golden pattern that was stored in the BIST module, the "BIST_Good" signal will be set to show the result.

In Figure 4.1, the top module is named "weptop". The "weptop" includes three sub-modules, the "WEP_IP" (wepip), the "WEP_BIST" (wepbist), and the "WEP BIST Clock Controller" (wepbtclk). The WEP IP was introduced in Chapter 2. In the next section, we will give a detailed description about the "WEP BIST" design. The major function of the "WEP BIST Clock Controller" is to disable the clock source to save the power in the normal mode. We will describe this module in Section 4.3.1.

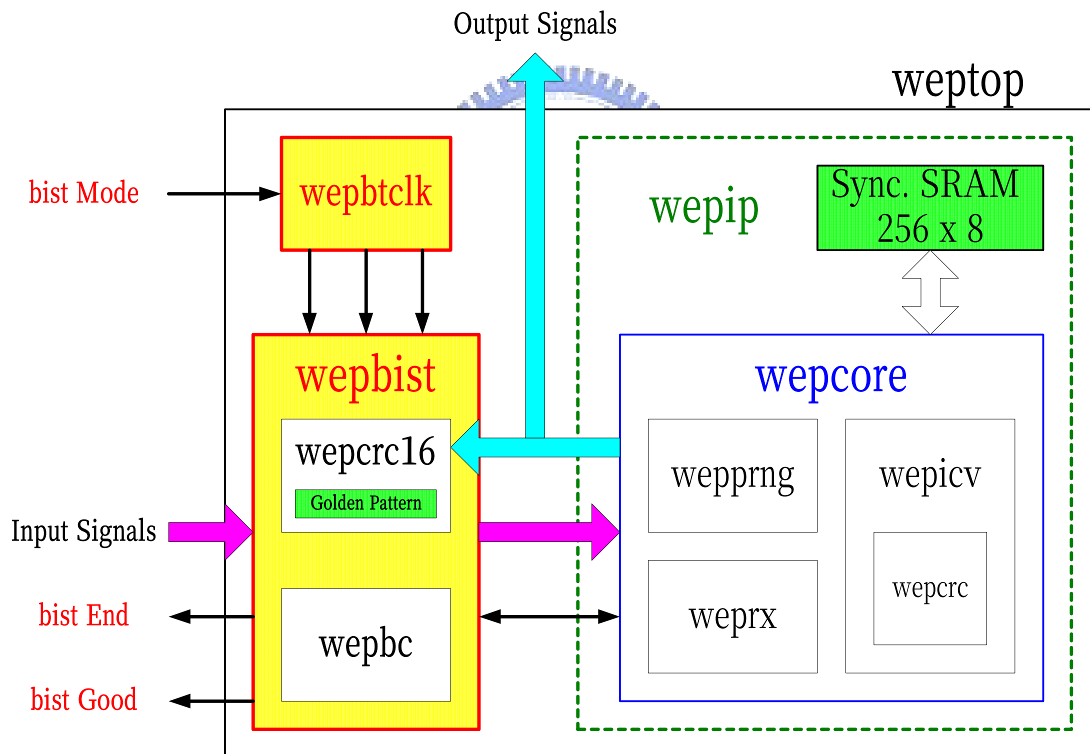


Figure 4.1 The block diagram of the WEP IP with the BIST module

Figure 4.2 describes the data flow of the "WEP_IP" in the normal operation mode (bist_mode = 0). "WEP_BIST" bypasses the input data and does not cause any influence on the WEP_IP's function during the normal operation mode.

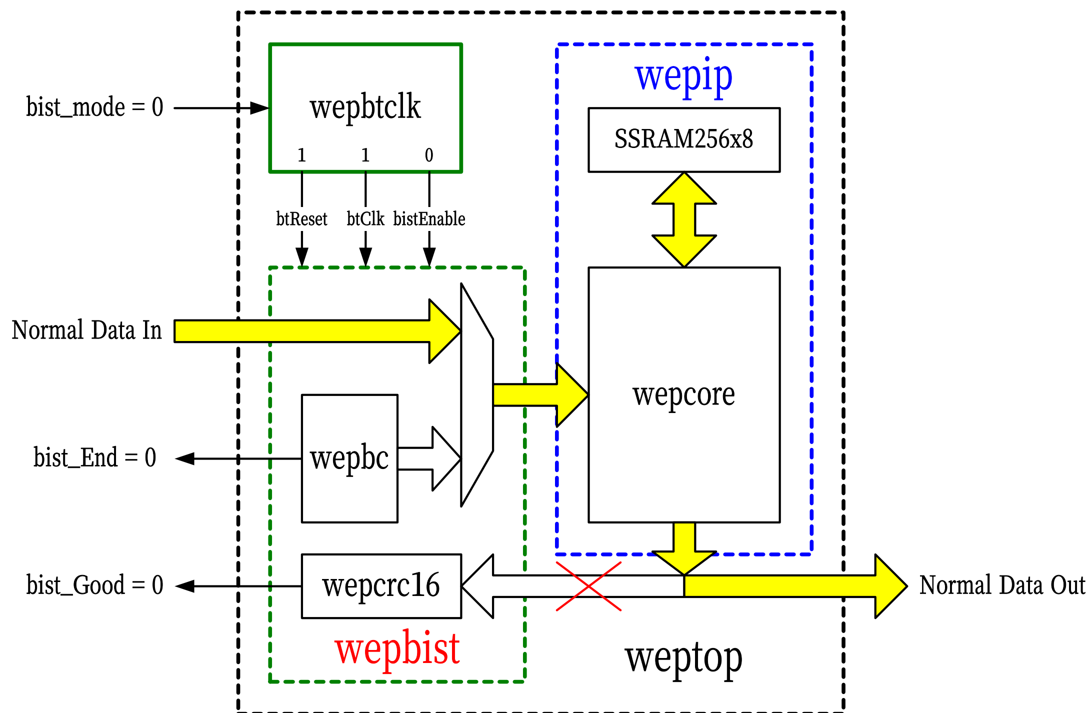


Figure 4.2 The WEP IP data flow in the normal operation mode

Figure 4.3 describes the data flow of "WEP_IP" in BIST mode (bist_mode = 1). "WEP_BIST" generates all inputs of "WEP_IP" and compares the outputs from "WEP_IP" with a golden checksum which is stored in "WEP_BIST".

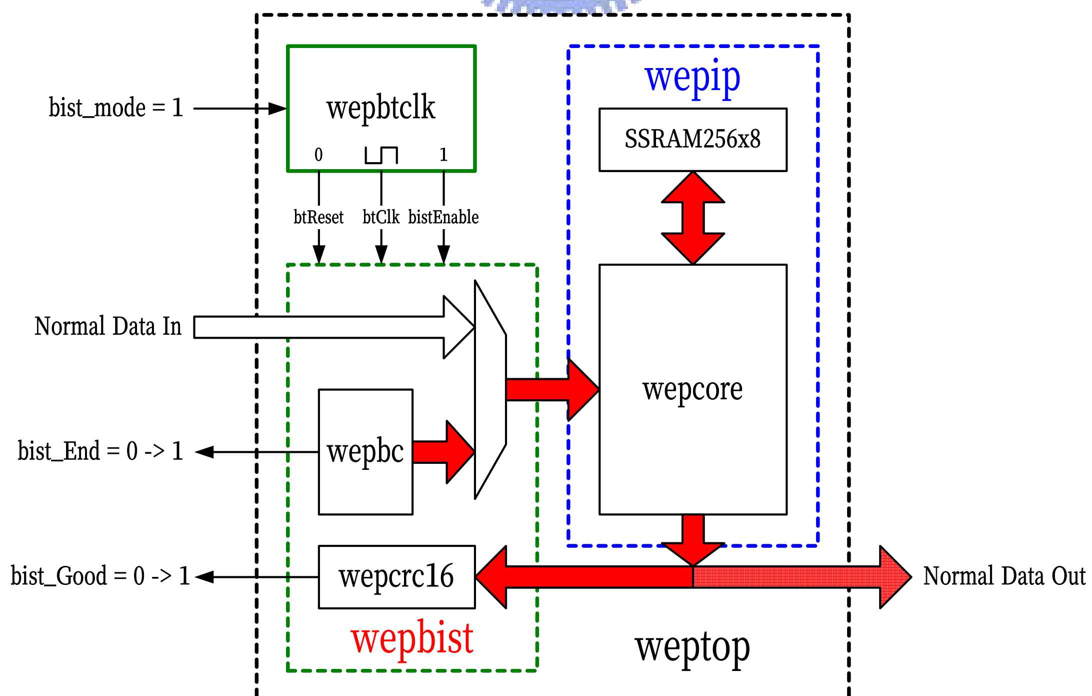


Figure 4.3 The WEP IP data flow in BIST operation mode

4.2.2 Testing Method

In this section, we focus on the function description of the WEP BIST method. The general idea of the BIST module is to design an automatic functional patterns generator to feed the WEP IP with some expected vectors. At the same time, this BIST module checks the output signals from the WEP IP then exports the testing result. In order to generate the testing vectors and ensure the normal functions, we design many multiplexers to select the WEP IP's input data between the normal signals and the BIST vectors. Please refer to Figure 4.4 below. While the "BIST_Mode" signal, described in Table 2.1, is set, the "WEP BIST Clock Controller (wepbtclk)" sets the "bist_Enable" signal to active the select signal of the multiplexers in "WEP_BIST" module. The WEP_IP's input signals will be switched from the normal input signals to the functional vectors that are generated by "WEP_BIST" module.

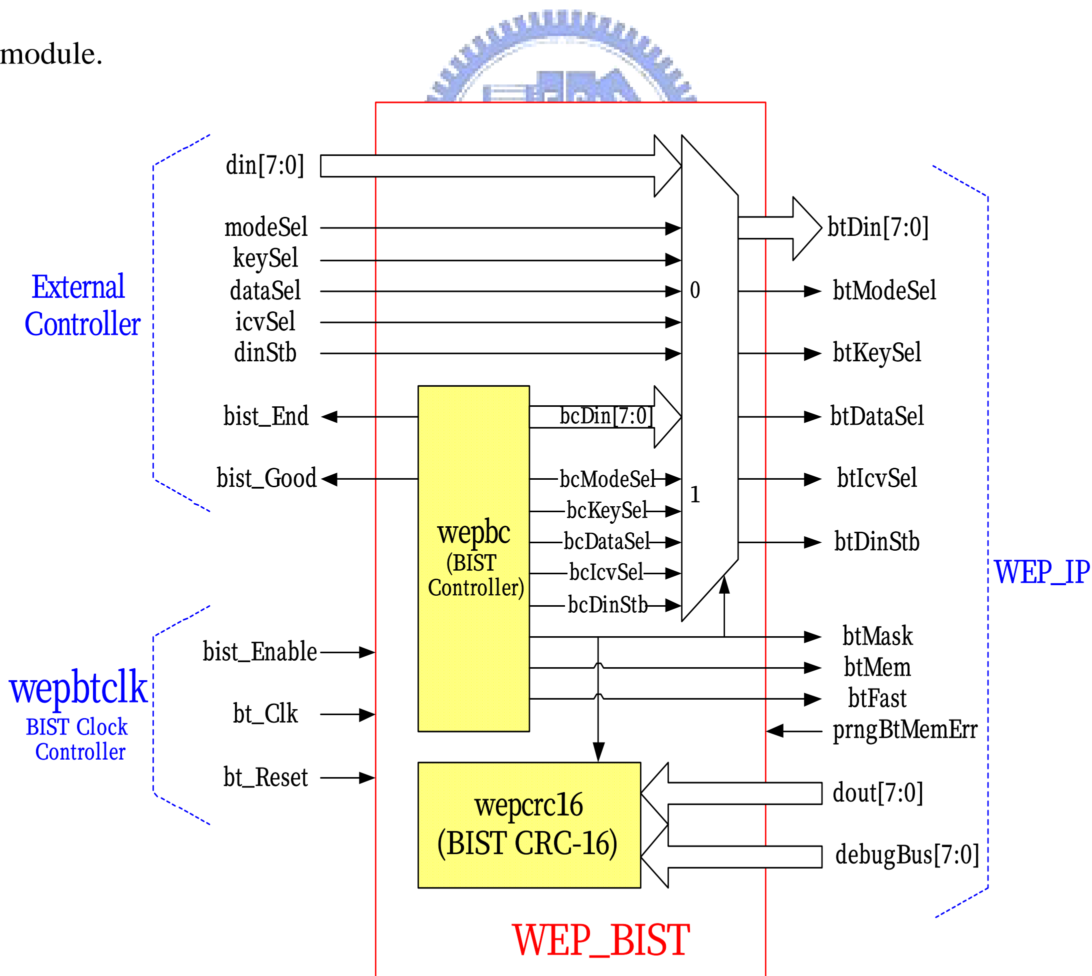


Figure 4.4 The multiplexers of the WEP BIST

In Figure 4.4, there are two sub-modules. The first one is named “wepcrc16”. This sub-module collects and compares the signals from “WEP_IP”. In order to reduce the length of the golden patterns, a CRC-16 algorithm is implemented in this design. The initial value of this CRC-16 is “4ABAh”. Equation (10) is the polynomial of this CRC-16 algorithm.

$$\mathbf{G(x) = X^{16} + X^{12} + X^{5} + 1} \qquad \mathbf{(10)}$$

For more detail descriptions and aliasing analysis of the CRC-16, please refer to Section 4.2.5 below.

The other sub-module in “WEP_BIST” is “wepbc”. The “wepbc” is abbreviated from “WEP BIST Controller”. This sub-module controls and generates the functional vectors to feed “WEP_IP”. How does the “WEP BIST Controller” generate the expected function vectors to achieve the quality fault coverage? Because there are many *finial state machines* (FSM) in the module under test “WEP_IP”, we must generate some valid vectors in the correct sequence to trigger and go through these FSMs to obtain the comfortable fault coverage. For this reason, we design a complicated FSM in the “WEP BIST Controller” sub-module to generate the expected control signals for “WEP_IP”. There are total 34 states in this FSM and the functional testing covers the encipherment, decipherment and the error signal testing. We will describe the FSM in Section 4.2.3.

About the input data for the “Din[7:0]” bus of “WEP IP”, we reuse the 8-bit counter that is to count the bytes number of patterns to generate the “Din[7:0]” input vectors. In order to verify both the stuck-at-one faults and the stuck-at-zero faults in the CUT, we use the complementary patterns in the encipherment and decipherment testing modes. In other words, the “Din[7:0]” is from “00h” to “FFh” in the encipherment mode and from “FFh” to “00h” in the decipherment mode. With this method, we can reduce the extra area of “WEP_BIST” and obtain the acceptable fault coverage.

4.2.3 Testing Flow

In this section, we will describe the testing flow of "WEP_BIST" and the main FSM of the "WEP BIST Controller". First, please refer to Figure 4.5 below. This is the simulation waveform of the full chip during the BIST mode.

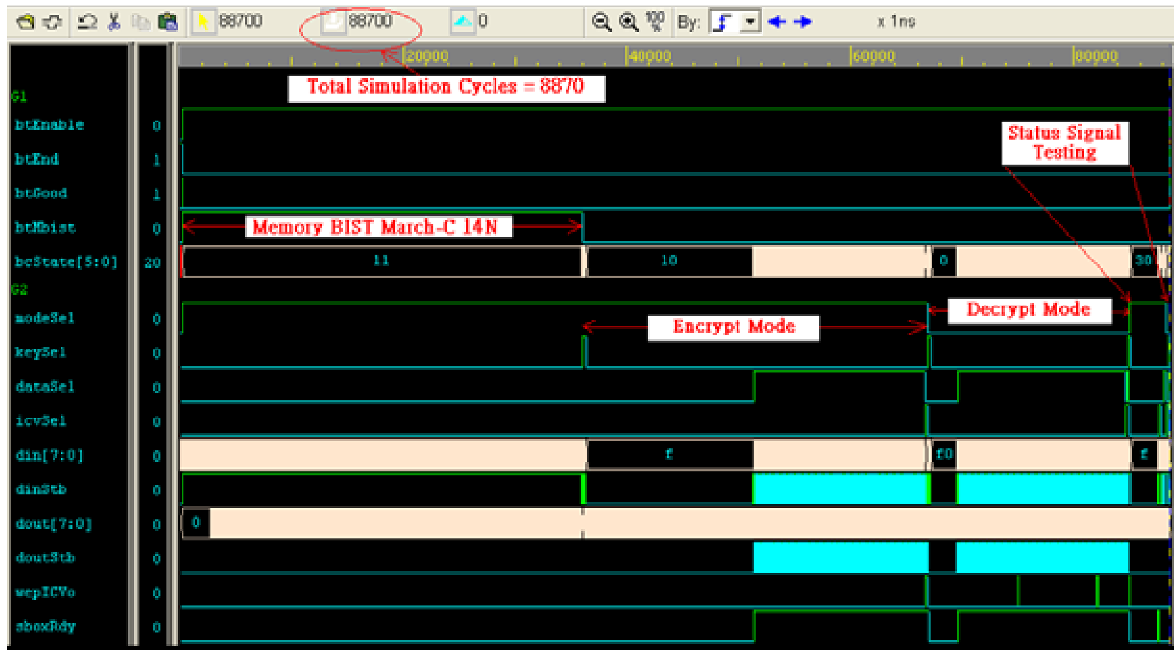


Figure 4.5 The WEP IP simulation waveform during BIST mode

Although there are 34 states in the FSM of the "WEP BIST Controller", we can classify them into four major procedures, "Memory BIST Mode", "Encrypt Mode", "Decrypt Mode", and "Status Signals Testing Mode" as illustrated in the Figure 4.5. In order to create the valid waveform to trigger the FSMs of "WEP IP" sequentially, we design a complex FSM in the BIST controller. Figure 4.6 shows this FSM on the next page.

The first procedure is the "Memory BIST" state. In this procedure, "WEP_IP" enters the MBIST mode and verifies the embedded 256-bytes SRAM with "14N MARCH C+ MBIST" algorithm. About this MBIST module, it will be described in next section.

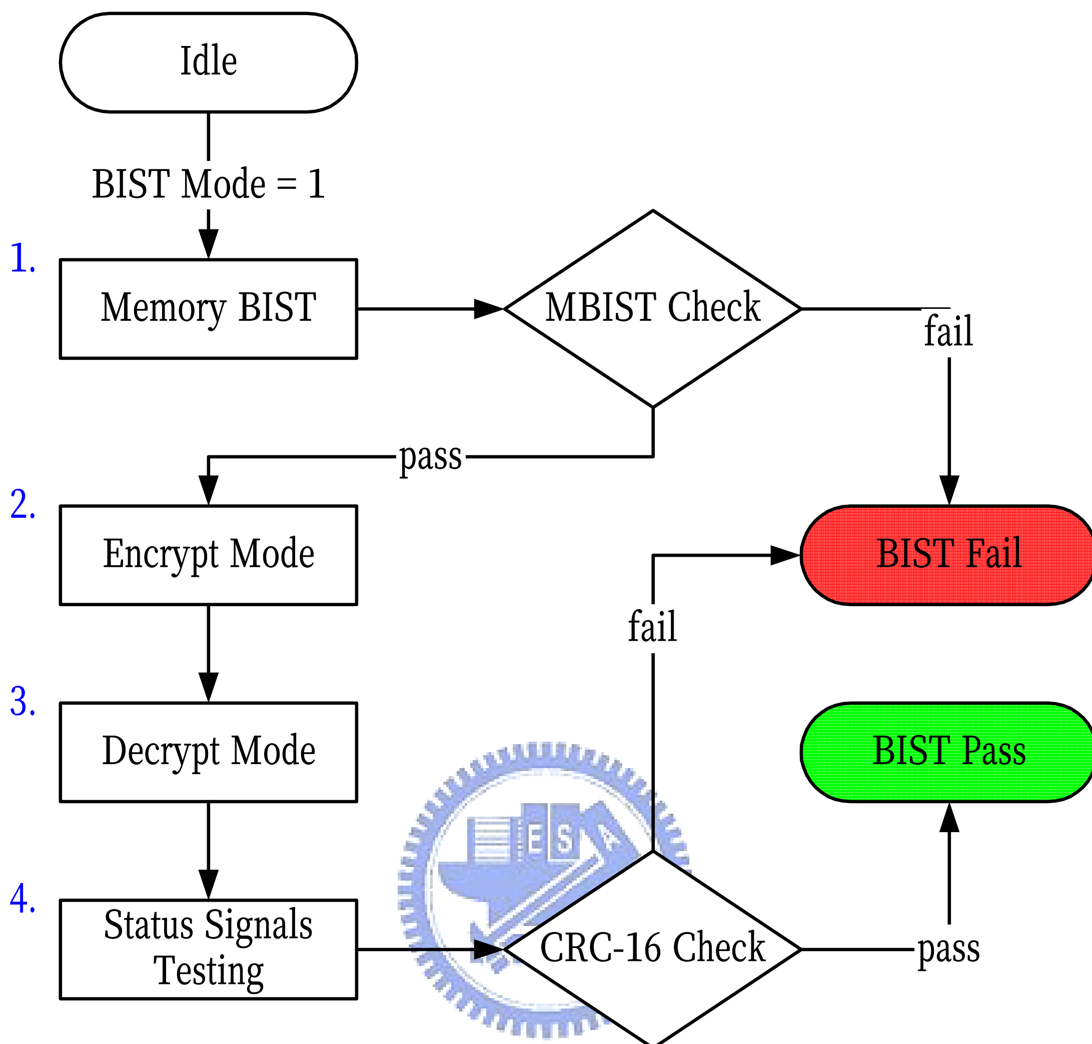


Figure 4.6　The Major FSM of the WEP BIST Controller

The second procedure is the "Encrypt Mode". The "WEP BIST Controller" generates the valid control signals and data by the sub-FSM that illustrated in Figure 4.7(a.) below. These signals test all circuits and FSMs that are related to encryption process in "WEP_IP".

The third procedure is the "Decrypt Mode". Like "Encrypt Mode", the "WEP BIST Controller" generates the valid signals and data to trigger all related circuits in "WEP_IP". The sub-FSM of "Decrypt Mode" is illustrated in Figure 4.7(b.). There are some different points, which must be highlighted in the "Decrypt Mode". First, the input patterns on the

“Din[7:0]” bus between the “Encrypt Mode” and “Decrypt Mode” are complementary. Second, we insert a control signal to speed up the procedure in “Decrypt Mode”. This “speed up mode” will be described in Section 4.3.2.

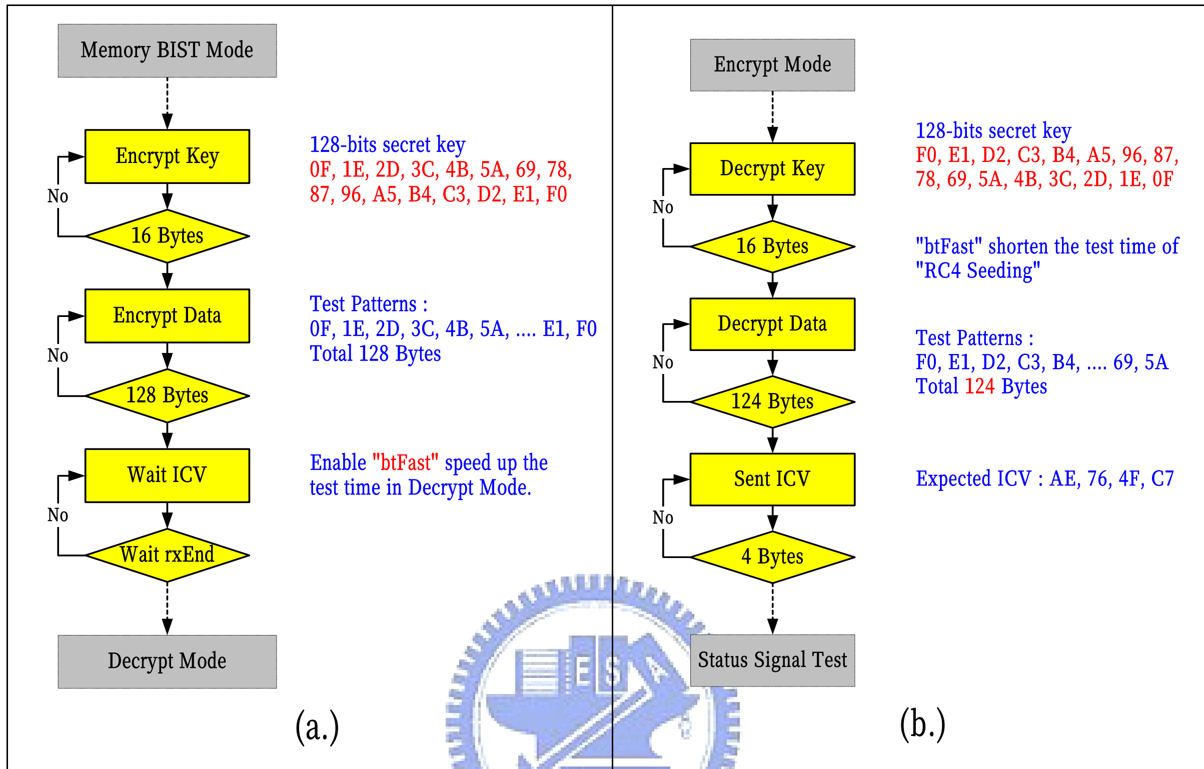


Figure 4.7 The sub-FSM of Encrypt Mode (a.) and Decrypt Mode (b.)

The last procedure is the “Status Signals Testing Mode”. There are many status signals to inform the end-users in “WEP_IP”, such as “WEP_ERR” output. The “WEP BIST Controller” generates some specific conditions to active these status signals. The purpose of this procedure is to enhance the fault coverage by verifying some corner cases in “WEP_IP”.

The following Figure 4.8 shows the relationship between the FSM of WEP BIST Controller and WEP RX described in Section 2.2.3. In Figure 4.8, the first line (blue line) is the data path of the “Encrypt Mode”, the second one (red line) is the “Decrypt Mode”, and the last line (green line) is the “Status Signals Testing Mode” in the FSM of “WEP_RX”. We can find that all states of the WEP_RX’s FSM are verified by the three

procedures.

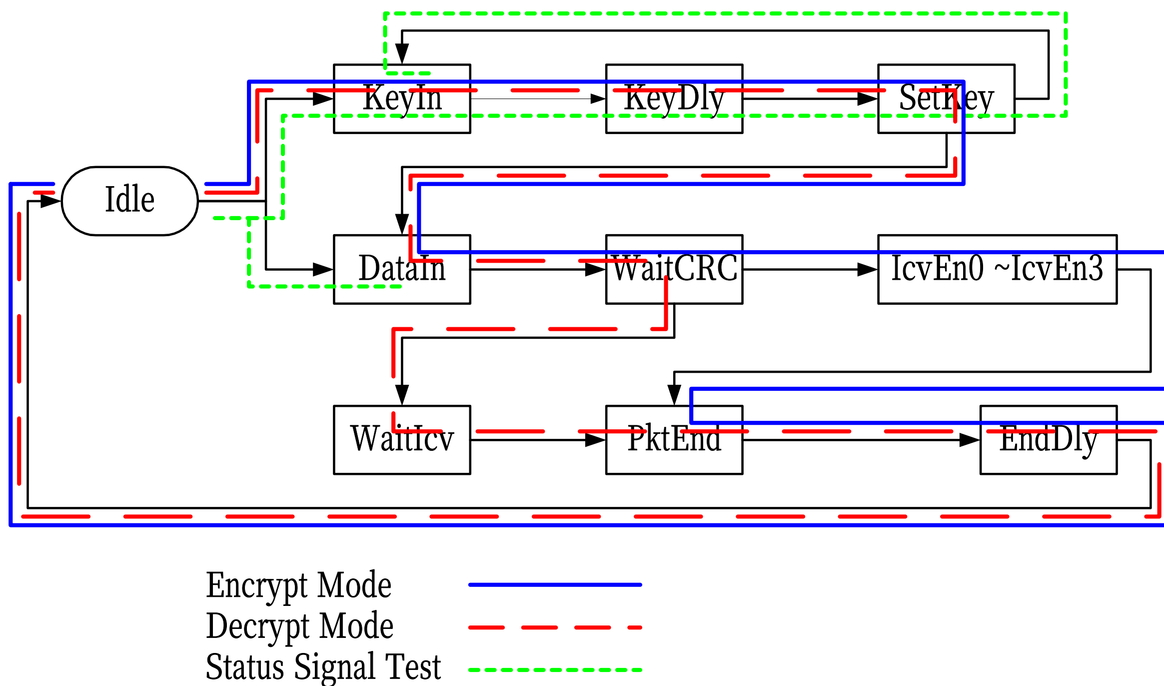


Figure 4.8 WEP BIST Controller's FSM vs. WEP RX's FSM

4.2.4 Embedded 14N March C+ for Embedded Memory

In "WEP_IP", there is an embedded memory macro, a synchronous 256-bytes SRAM. The die area of this 256-bytes SRAM is almost equal to the die area dedicated to the logic. Therefore, the memory array is more sensitive to defects. There is a higher probability that the random defect content will land on and will have a deleterious effect on the memory array rather than the general logic.

In Section 3.1.2, we described the special "MATS" MBIST algorithm that used in the traditional memory test. This special "MATS" algorithm can be combined with the original FSM of "WEP_PRNG" to reduce the extra gate-count and testing time, but its low fault coverage, only all stuck-at faults (SAF) in memory cells and some address decode fault (AF), is its deadly disadvantage. In order to get a quality level MBIST in our WEP IP, we replace the "MATS" with the "14N MARCH C+" MBIST algorithm.

The "14N MARCH C+" MBIST algorithm is defined as follows [9]:

$$\{ \uparrow (w0); \uparrow (r0, w1, r1); \uparrow (r1, w0, r0);$$

$$\downarrow (r0, w1, r1); \downarrow (r1, w0, r0); \downarrow (r0); \} \quad (11)$$

This "14N MARCH C+" algorithm can detect stuck-at faults, bridging faults, address decode faults, data decode faults, transition faults, and access time faults. Its test complexity is 14n. In our WEP IP, it takes extra 3584 cycles to test the embedded 256-bytes SRAM. It occupies almost 40% of all BIST cycles (8870).

What is the area overhead of this "14N MARCH C+" in the WEP IP? In order to reduce the extra gate-count to implement this MBIST, we do our best to combine the MBIST logic with the original circuit. Because the WEP function is halted during the MBIST mode, we can reuse the 8-bits counter in the "WEP BIST Controller" module to generate the address and data to test the memory. The extra area overhead are one FSM with 16 states to manage the procedure of the "14N MARCH C+" and some multiplexers to arbitrate the data and address bus between the MBIST and the normal mode. The extra gate-count is about 180 gates.

Compared with the "MATS" algorithm described in Section 3.1.2, the "14N MARCH C+" has a higher performance to test this embedded SRAM. For this reason, we modify the architecture and replace the "MATS" with "14N MARCH C+" in our WEP BIST design.

4.2.5 The Golden Checksum by CRC-16

Reviewing Figure 1.2 illustrated in Section 1.2, we find that the BIST architecture includes the output response analyzer (ORA) to collect and check the output responses of the circuit under test (CUT) into some type of Pass/Fail flag. It is impossible to compare the outputs of CUT bit-by-bit with the golden patterns that are stored in a huge memory. In our design, we try to compact all outputs from CUT with a well-known algorithm,

“CRC-16”. The CRC-16 generator polynomial is described as follows:

$$G(x) = X^{16} + X^{12} + X^{5} + 1 \qquad (10)$$

The initial seed of this CRC-16 algorithm is “4ABAh”. We can reduce the length of the golden pattern to 16 bits with this method.

How the 16-bits golden pattern is generated? First, the “WEP BIST Controller” collects all output signals and some internal signals of “WEP_IP” and compacts them into 16 bits with XOR algorithm. Second, the CRC-16 module in “WEP_BIST” calculates with every 16-bits pattern and updates the checksum in every clock period. After going through the all test flow described in Section 4.2.3, we can get the golden value at the last simulation cycle with Verilog_XL. Then, we program this golden value into our circuit. In the general BIST mode, “WEP_BIST” compares the checksum that is generated by the CRC-16 module with the golden pattern that is stored in the circuit to indicate a Pass/Fail output.

Is there any risk in the CRC-16 compression? The answer is “Yes”. Unfortunately, it is possible that a signature of a bad machine may match the good machine signature, which is called aliasing. In such cases, a failing circuit will pass the testing process. Aliasing is always a problem with compaction because information is lost. For now, we try to analyze the aliasing probability with two experiments and simulations.

1. Aliasing Analysis with C Language:

We create a CRC-16 behavioral model in C language and generate a lot of testing vector sets randomly. The length of each testing vector set is equal to the pattern length of the WEP BIST design. Then, the CRC-16 behavioral model will generate the checksums with these testing vector sets. We can compare these checksums with the golden pattern to calculate the aliasing probability. The aliasing probability of the experiment is 34/2,500,000 (**1.36 X 10^{-5}**).

2. Aliasing Analysis with Verilog Simulation

Like the method above, we create a test bench to analyze the aliasing probability of the CRC-16 in Verilog language. This test bench generates many testing vector sets randomly into the CRC-16 module to calculate and check the checksum. The aliasing probability of this case is 17/1,310,720 (**1.30×10^{-5}**).

In these experiments, we find that the aliasing probability is unconcerned with the length of test vectors. The aliasing probability is less than 2^{-16} (1.52×10^{-5}), the aliasing of LFSR-16.

4.3 Other Special Skills

To consider how to design a high performance BIST module with high quality level, low power consumption, low area overhead, less testing time, and the self-test structure, there are many special skills that are implemented in our WEP BIST design. The following sections will describe these skills that are used in our design.

4.3.1 Power Down Mode

In general, the BIST module is only operated during manufacturing, not in the normal operation. But, the power consumption of these extra cells for BIST is usually unavoidable even in the normal operation mode. In order to reduce the dynamic power consumption in the normal operation mode, we design a clock controller to mute the clock source and activate the reset signal of those BIST cells.

In order to avoid the clock glitch when the operation mode switched, we also take care of the gating clock circuit. Figure 4.9 shows the diagram of the gating clock circuit and the waveform in the “WEP BIST Clock Controller (wepbtclk)”.

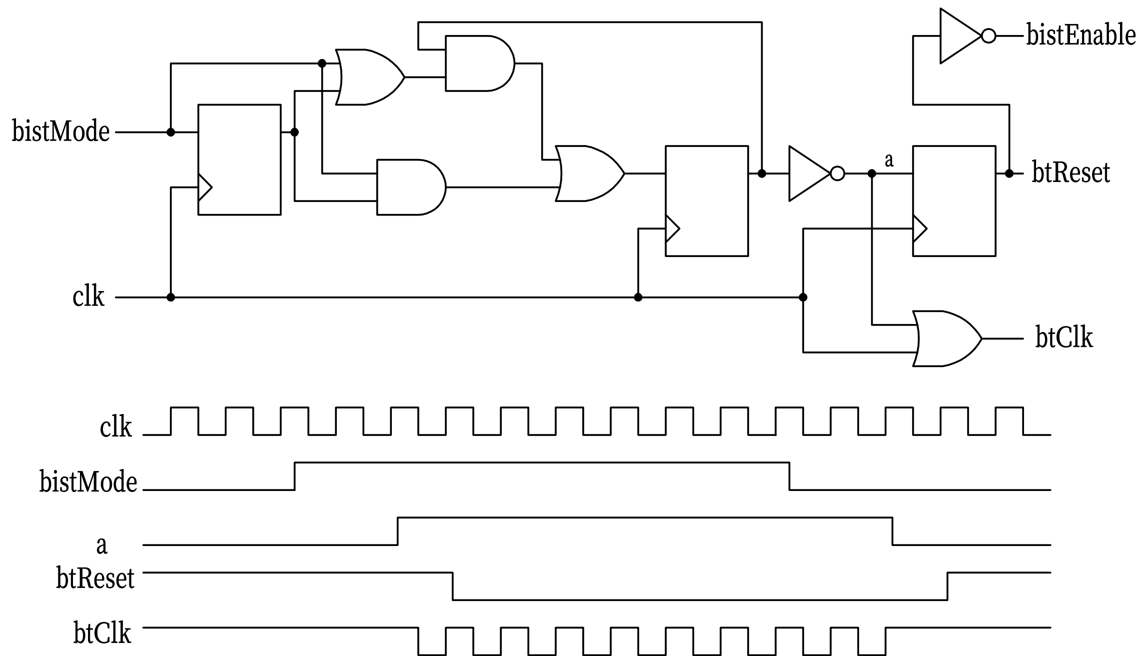


Figure 4.9 WEP BIST Clock Controller circuit and waveform

Finally, the WEP Clock Controller will generate a reset pulse to initiate all FSM and all flip-flops in "WEP_IP" while the BIST procedure is done.

4.3.2 Speed Up Mode

In the general functional testing, to design a speed up function to reduce the testing time is very popular. Review the WEP's RC4 PRNG algorithm described in Section 2.1.4, the "RC4 Seeding" process always takes 1536 cycles to stir the order of number in S-Box whether it is in encryption or decryption. Because this "RC4 Seeding" is a routine procedure to access the embedded 256-bytes SRAM, we try to reduce this process with a speed up mode.

We design a speed up flag in the "WEP BIST Controller" to achieve this purpose. In the FSM of the "WEP BIST Controller", this speed up flag will be set while the encryption process is done to declare the "Speed Up Mode". In the next procedure, decryption process, its test cycle of "RC4 Seeding" will be shortened into 273 cycles. Reviewing the "WEP simulation waveform in BIST Mode" illustrated in Figure 4.5 above, we can find the

influence of the speed up mode.

Does the speed up mode influence the fault coverage? The answer is “Yes”. Only a few nodes cannot be detected while the speed mode is implemented. The influence on fault coverage is less than 0.2% in our project.

4.3.3 Internal Multiplexers

The test patterns that are generated from the FSM of the “WEP BIST Controller” are similar to the functional testing vectors, not the structural vectors. It is difficult to improve the fault coverage into 100% by modifying the FSM. After check the fault coverage analysis report, we find that most undetectable faults are in the deep internal nodes. For example, in Figure 4.10 below, there are many signals that are through the “flip-flop 1” or the “flip-flop 2”, and these internal nodes are usually difficult to be observed at output pins. The fault coverage is hardly to be improved because the testing vectors are difficult to propagate these internal signals to the limited output pins.

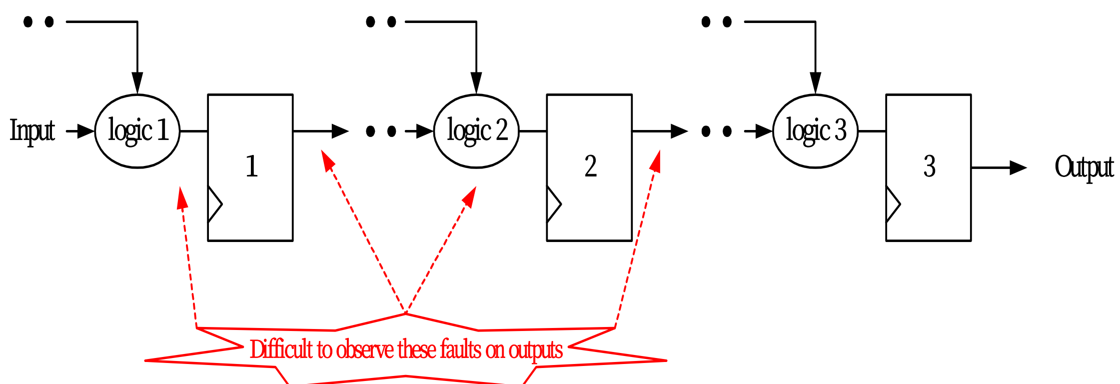


Figure 4.10 The internal faults on a normal design

For the reason, we design some internal multiplexers into the WEP IP to extract and propagate these hard detectable signals to the output pins, as illustrated in Figure 4.11 below.

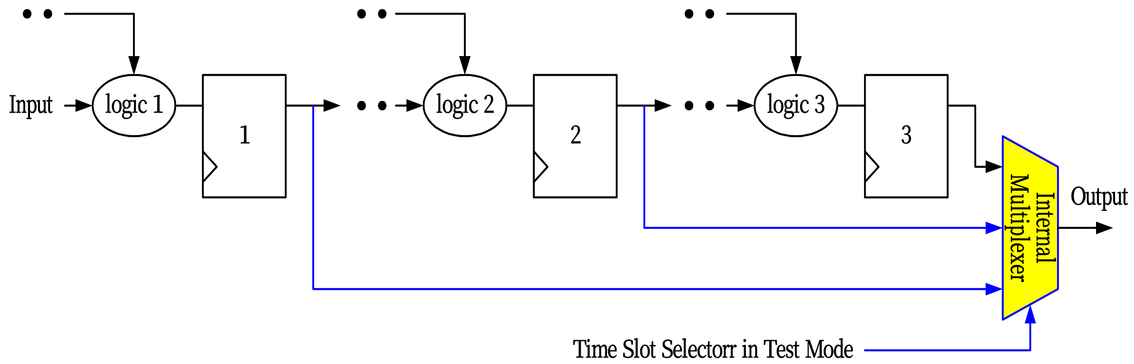


Figure 4.11 The circuit with the internal multiplexer

In order to observe more internal signals during the BIST period, we separate the BIST period into several time slots. In the method, we can trace the best node point at the best time point.

These internal multiplexers improve the fault coverage, unfortunately, but increase some extra gate-count. This is a trade-off depended on the specification and requirement.

4.3.4 Self-Test of BIST for WEP IP

We usually are questioned, "how to verify your BIST circuit?" The BIST of a BIST is always a problem to perplex the testing engineers and circuit designers. We are troubled with the same problem. In this section, we try to claim some solutions about the self-test of the WEP BIST module.

1. Unavoidably, any defect in the WEP BIST module will cause the BIST process to fail except the stuck-at one fault at the "BIST_Good" output signal.
2. In order to verify the stuck-at one fault at the "BIST_Good" output signal, the WEP BIST Controller will drive zero at this output when both the "Reset" and "BIST_Mode" inputs are active. Under this condition, all FSMs in our design will be disabled and end-users can check the "BIST_Good" stuck-at one fault on their application circuits.

3. Another problem is on the “BIST_End” output. If there were a stuck-at zero fault at this output, end-users would never know when the BIST procedure is done? To solve this problem, we suggest end-users to implement an extra watchdog timer on their system or the ATEs to monitor the BIST procedure. If the “BIST_End” was not active for an overcounted period, the BIST would be aborted because of a stuck-at zero fault at “BIST_End” output.
4. The last issue is how to verify the input ports in the BIST mode? In the BIST mode, normal input signals will be blocked by the BIST pattern as illustrated in Figure 4.12. We are unable to detect the stuck-at faults on these inputs.

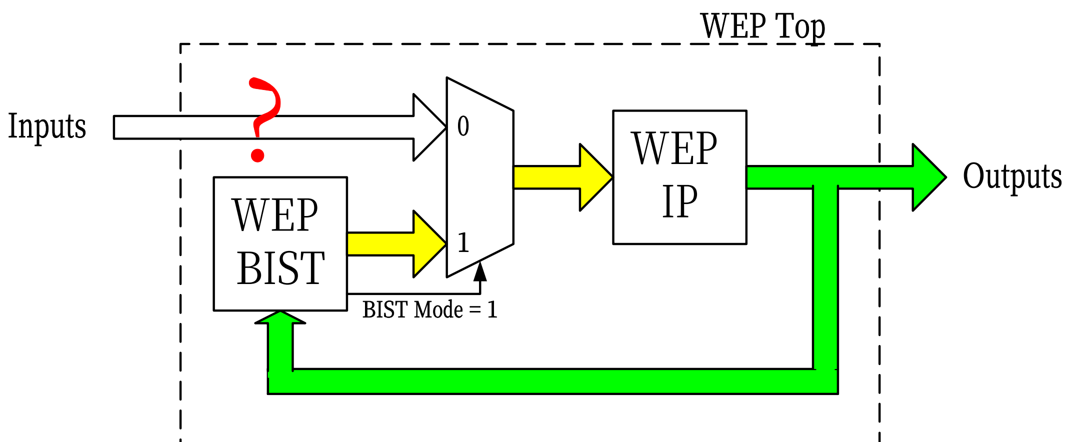


Figure 4.12 The undetectable faults on normal input ports

In order to verify these input ports, we try to create a bypass mode to propagate the status of all inputs to the output ports. Please refer to Figure 4.13 below. First, we create a bypass signal from “Reset” and “BIST_Mode”. During this bypass mode, (both “Reset” and “BIST_Mode” are active), end-users can propagate the status from all inputs to all outputs. Table 4.1 lists the mapping table of the all inputs versus all outputs under the bypass mode.

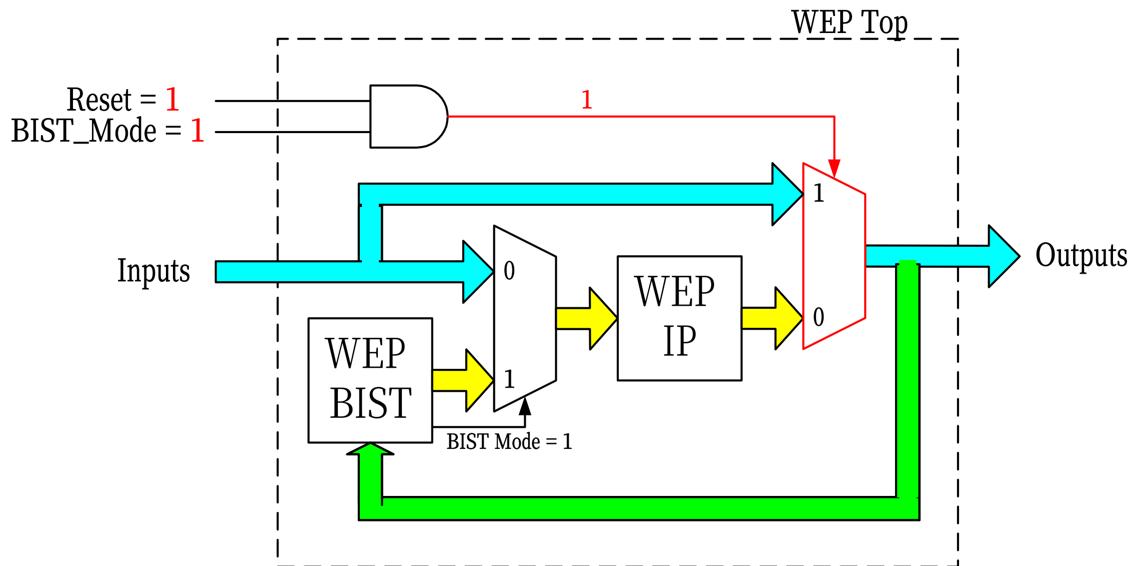


Figure 4.13 The data flow in the bypass mode

Inputs	Outputs
Din [7:0]	Dout [7:0]
Din_STB	Dout_STB
Data_Sel	WEP_ERR
ICV_Sel	WEP_ICVo
Key_Sel	Sbox_RDY
Mode_Sel	BIST_Good
BIST_Mode	BIST_End

Table 4.1 The inputs vs. outputs mapping table in the bypass mode.

Note: Please refer to Table 2.1 for the detail pin descriptions.

4.4 Experiment Reports

We had presented the design flow and the experiment environment in Section 2.2.1. This section describes the simulation results of the "WEP IP with BIST".

4.4.1 Synthesis Results

We synthesize the "WEP IP with BIST" from Verilog RTL code to the tsmc 0.35um process standard cell netlist with the Synopsys EDA tool "Design Compiler". The synthesis result is described in Table 4.2.

Process: tsmc 0.35 standard cell library (cb35os142)

Area: Total Logic = 5293 gate count (1 gate = 1 2-input NAND cell)

Where: WEP_Core = 4651 gate count

WEP_BIST = 642 gate count

WEP_MBIST = 346 gate count

Marco = One synchronous SRAM 256x8

The max. operating frequency = 125MHz in the worst case.

Table 4.2 The synthesis result of the WEP IP with BIST

If we calculate the area overhead directly, the value is equal to 12.13%.

$$642 / (4651 + 642) \times 100 \% = 12.13 \% \quad (12)$$

Comparing the logic gate-count of the WEP IP with the value in Table 2.2, we find that the increased area of the WEP IP is due to the internal multiplexers and the "14N MARCH C+" MBIST circuit. So we recount the area overhead in the worst case. The area overhead is equal to 21.8 % where the value "4139" is the original logic gate-count of the WEP IP that was presented in Table 2.2.

$$(5293 - 4139) / 5293 \times 100 \% = 21.8 \% \quad (13)$$

4.4.2 Fault Coverage Results

As described in Figure 4.1, “WEP_BIST” generates some test vectors to verify “WEP_IP”. Because the embedded synchronous SRAM is a macro and can be verified by the “14N MARCH C+” MBIST, we only calculate the fault coverage of “WEP_CORE” with Syntest’s EDA tool “TurboFault”.

The fault coverage report of “WEP_CORE” is described as follows.

Item	Number	Percentage
Optimal Fault Coverage	**--**	**95.97 %**
Total Faults	**14336**	**100.00 %**
Hard Detected Faults	**12715**	**88.69 %**
Probably Detected Faults	**1043**	**7.28 %**
Undetected Faults	**575**	**4.01 %**

Table 4.3 The fault coverage report of WEP IP with BIST

The total testing time is 8870 clock cycles. It is less than the testing time of the functional vectors (12203) and the ATPG patterns (33962). If the operating frequency of this “WEP_IP” is 125MHz, it will take only 70.96us to finish the BIST and the “14N MARCH C+” MBIST procedures. Meanwhile, the at-speed testing can verify all delay faults in the WEP IP design.

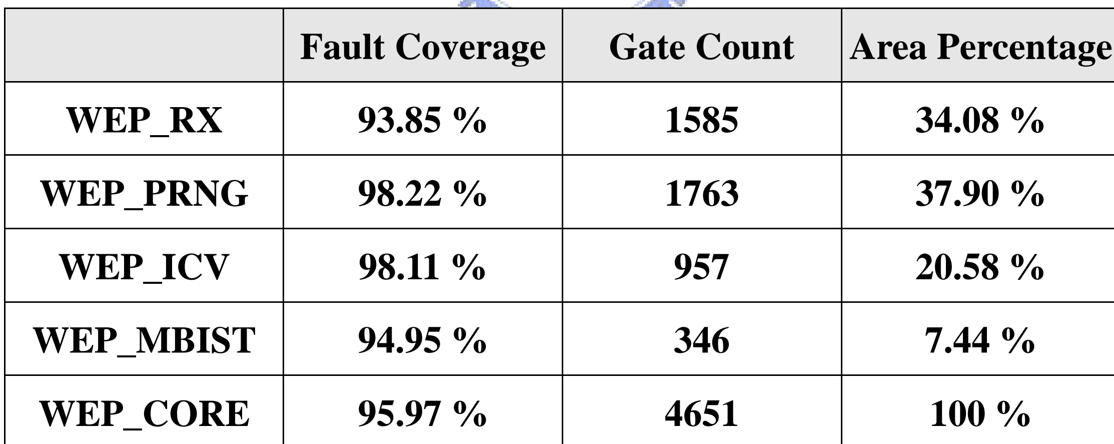
4.5 Analysis and Discussion of BIST for WEP IP

In this section, we will analyze the fault coverage reports of the all sub-modules in the WEP IP and discuss the improvement history of the fault coverage reports in order to find out the limitation of our WEP BIST design.

4.5.1 The fault coverage of sub-modules

Unfortunately, the finial fault coverage is only about 96% that fails to meet our target fault coverage, 99%. We try to find out some solutions by analyzing the fault coverage of each sub-module. With this method, we can improve the worst sub-module by upgrading the test patterns in the WEP BIST design and then propagate those signals to the output ports of the WEP IP with some internal multiplexers.

The following Table 4.4 lists the fault coverage and gate count reports of all sub-modules in the “WEP_CORE”.



	Fault Coverage	Gate Count	Area Percentage
WEP_RX	93.85 %	1585	34.08 %
WEP_PRNG	98.22 %	1763	37.90 %
WEP_ICV	98.11 %	957	20.58 %
WEP_MBIST	94.95 %	346	7.44 %
WEP_CORE	95.97 %	4651	100 %

Table 4.4 Fault coverage and area report of sub-modules

From Table 4.4, we can find that the bottleneck of the total fault coverage lies at “WEP_RX”. We find that there are some problems to improve the fault coverage after tracing the undetectable faults in this sub-module.

First, “WEP_RX” is the entrance module of “WEP_CORE”. The signals in “WEP_RX” are more difficult to be observed in the output ports than other sub-modules.

Second, there is a 128-bits shift register to store the WEP secret key. There are many internal nodes and faults in these flip-flops. The faults on these shift registers are difficult to be verified with the FSM of “WEP_BIST”.

In fact, we had upgraded and improved the FSM of “WEP_BIST” several times to enhance the fault coverage. And we also tried many digital design skills to improve the results and save the testing time. These skills ware described in Section 4.3. We will describe the history of the fault coverage improvements in following section.

4.5.2 Improvements of the Fault Coverage

Reviewing the history, we mark five milestones in the procedure of the WEP BIST improving. The descriptions of these milestones are as follows.

1. Create a WEP BIST module to generate the test vectors for the WEP IP directly.
2. Insert a “MATS” MBIST module into the WEP BIST design to verify the embedded SRAM. We also add the “status signal testing” procedure to verify the internal status registers.
3. Add the “Speed-Up” mode to reduce the testing time.
4. Insert some internal multiplexers into the WEP IP to improve the fault coverage.
5. Replace the “MATS” MBIST module with the “14N MARCH C+” algorithm to guarantee the quality level of test for the embedded memory.

Undoubtedly, every change makes many differences in fault coverage, gate count, and testing time, etc. We summarize these reports in Table 4.5 and the diagrams below. The total fault coverage had improved from 82.70% to 95.97% in our experiment. The fault coverage “96%” is the saturated value in our BIST design because there are some hard controllable and hard observable internal signals in the WEP IP. If we tried to control or

observe them by inserting more internal multiplexers into the WEP IP, we would create more unpredictable faults in the circuit. With this method, we only increase the extra gate count and testing time, while not improving the total fault coverage.

Milestone	1	2	3	4	5
Fault Coverage	82.70%	87.40%	90.37%	95.76%	95.97%
Total Gate Count	4600	4735	5148	5262	5293
Total Fault Number	11090	12935	13302	14005	14336
Area Overhead	10.02%	12.59%	19.60%	20.32%	21.80%
Testing Time (Cycle)	3150	4926	4032	4530	8870
MBIST	No	MATS 4N	MATS 4N	MATS 4N	March C+

Table 4.5 Improvements of the fault coverage

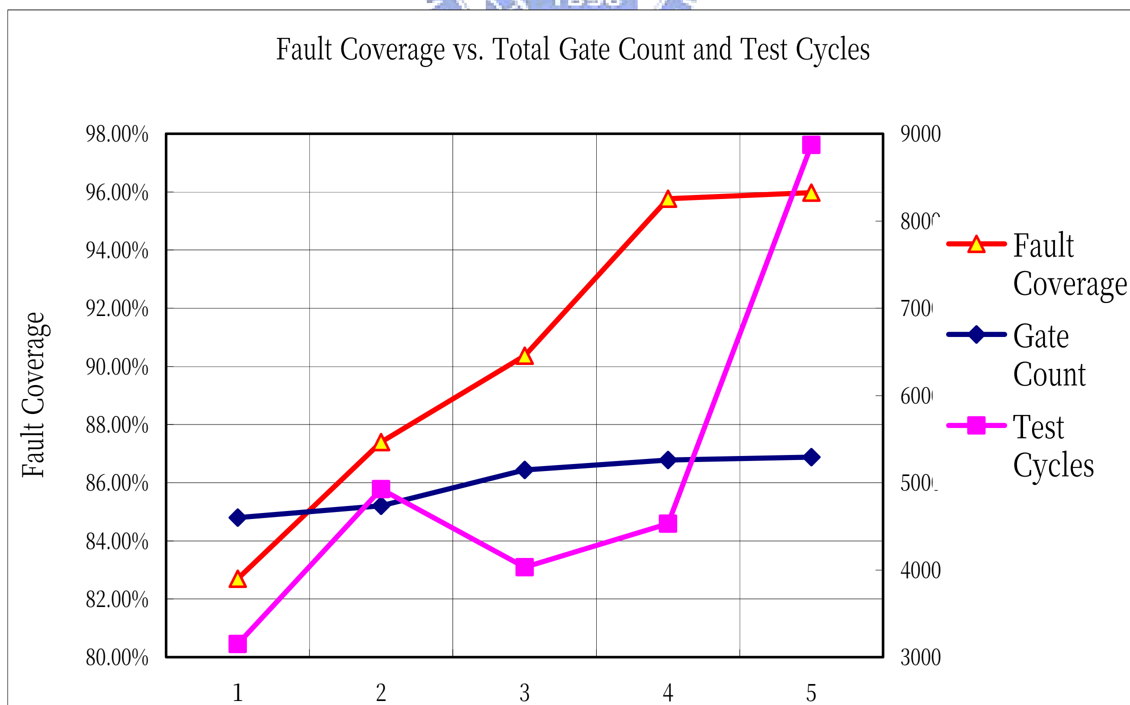


Figure 4.14 Fault Coverage vs. Gate Count and Testing Time

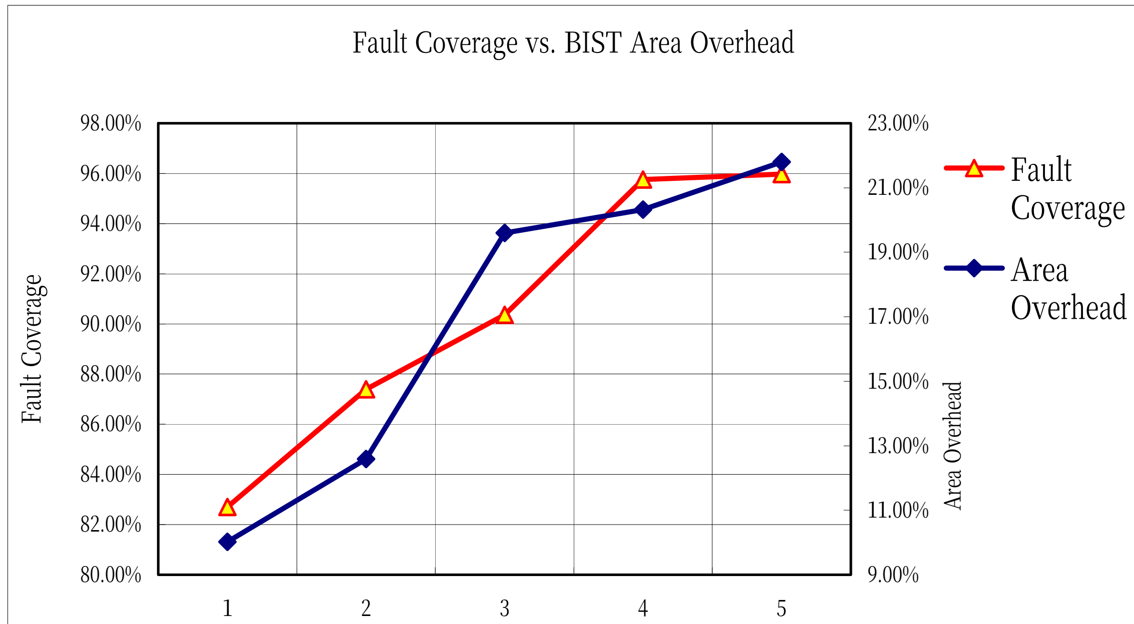


Figure 4.15 Fault Coverage vs. BIST Area Overhead

4.6 BIST vs. Traditional Testing Methods for WEP IP

Summarizing the experiment reports of the WEP BIST design above and the traditional testing methods that presented in Chapter 3, we obtain Table 4.6 and Figures 4.16 and 4.17.

	WEP BIST		ATPG + Wrappers	Functional Testing
Fault Coverage	95.97 %	95.76 %	98.91 %	92.69 %
Area Overhead	21.80 %	20.32 %	19.22 %	7.07 %
Total Gate Count	5293	5262	5124	4454
Total Testing Cycles	8870	4530	33962	12203
Ext. Memory Storage	**Small**	**Small**	**Large**	**Large**
MBIST Algorithm	**MARCH C+**	**MATS**	**MATS**	**MATS**
ATE Needed	**No**	**No**	**Yes**	**Yes**

Table 4.6 WEP BIST vs. ATPG vs. Functional Testing

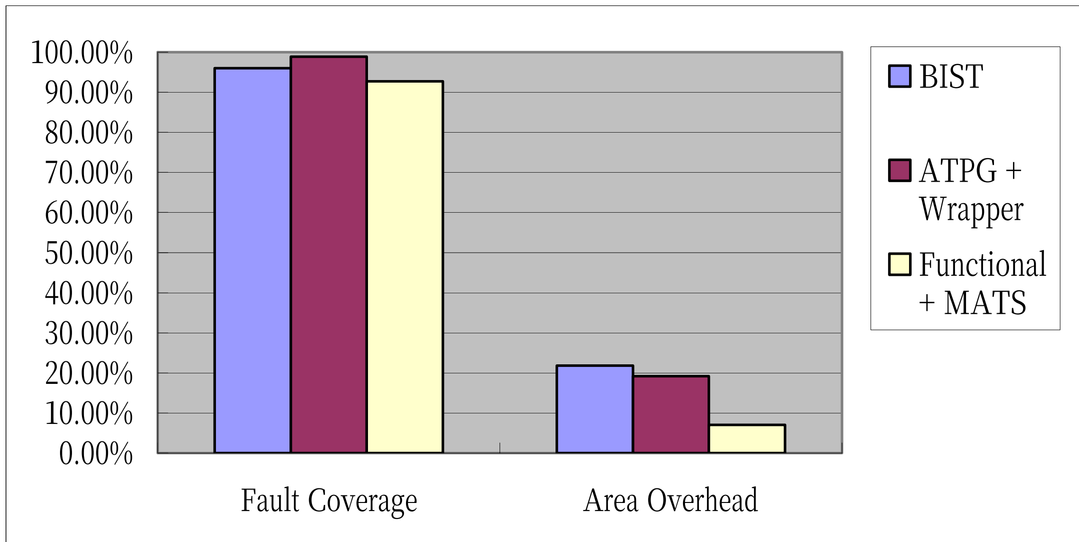


Figure 4.16 Comparing diagrams of the fault coverage and the area overhead

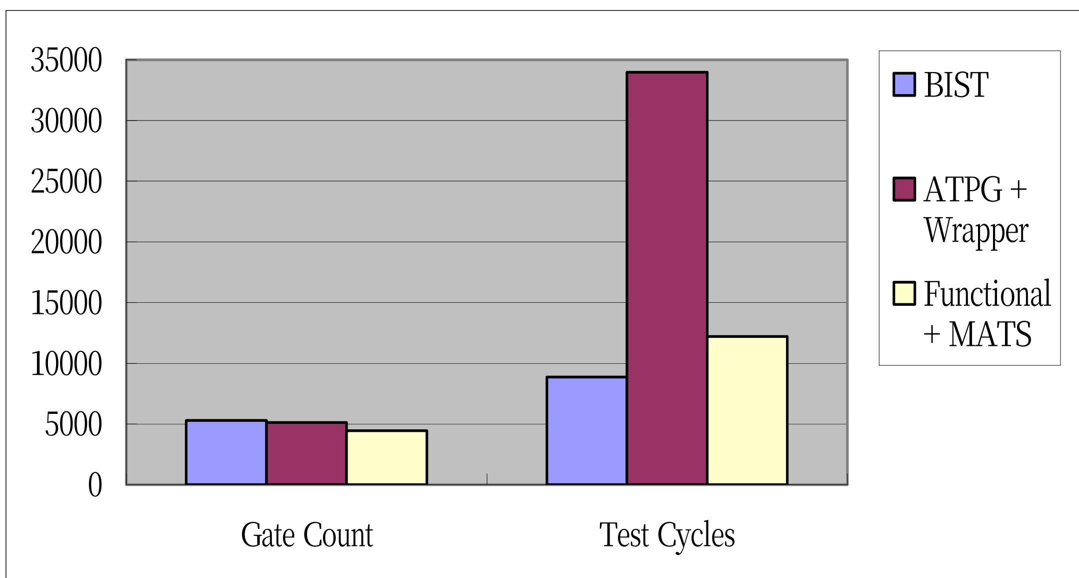


Figure 4.17 Comparing diagrams of the total gate count and the test cycles

From the table and figures, we can see that the functional testing has the least area overhead but it is very difficult to get a comfortable fault coverage result. Although the fault coverage of the scan-chain in the WEP IP is very exhilarating, we also pay a large of memory storage and a long testing time to achieve the object.

In our design, the WEP BIST module only generates a few test vectors to verify the

WEP IP efficiently. And all these patterns are executed at-speed to detect the delay faults in the CUT. The least memory storage is another advantage. In summary, the overall performance of the WEP BIST module is heartening.

4.7 Summary

This chapter has covered all design methods and all experiment reports in our WEP BIST design. We summarize the methods and skills of the WEP BIST design as the following lists:

1. We use an extra small module (WEP_BIST) to generate some expected and valid patterns to test the CUT (WEP_IP).
2. In order to trigger and go through the complex FSMs in the CUT, we insert another complex FSM into the BIST module to generate all necessary waveforms to meet the protocol.
3. In order to reduce the extra area overhead, we use some present circuits, such as a 8-bits counter, to generate the test vectors for "Din[7:0]" inputs.
4. We design a standard CRC-16 module to be the output response analyzer (ORA) and the compressor to reduce the length of the golden pattern. Its aliasing probability is less than 2^{-16} ($1.52 \text{ X } 10^{-5}$).
5. There is a "14N MARCH C+" MBIST circuit to verify the embedded SRAM to meet the quality-level of test.
6. Insert a power saving mode to disable the clock source of BIST modules.
7. Insert a speed up mode to reduce the testing time.
8. In order to improve the fault coverage, we insert some internal multiplexers to propagate the internal signals.
9. We also create some self-test methods to test the BIST module, such as the bypass mode.

With these methods above, we can create a high performance WEP BIST module to test our WEP IP at-speed without the large vector storage. Because of the at-speed tester, the testing time is less than other testing methods and almost all delay faults in the CUT are detected. However, the high area overhead (21.8%) and limited fault coverage (96%) is its disadvantage. For a designer, the extra design schedule and design effort for this BIST module is unavoidable.

Chapter 5 Conclusion

In this thesis, we have designed a reusable WEP IP based on the IEEE 802.11 standard. In order to guarantee the validity of function and the quality of manufacturing, we have qualified the WEP IP with several testing methods.

First, we have used the traditional functional patterns to test the WEP IP. For a designer, the functional testing pattern is very easy to be implemented, and these patterns can verify the behavior of his circuit with less extra design effort. However, the low fault coverage and the limited operating frequency are its disadvantages.

Second, we have used the Syntest's EDA tool "TurboScan" to generate the scan-chain and ATPG in our WEP IP. It is easy to gain a high quality pattern with these EDA tools, if you knew how to operate these tools. But the extra area of these scan-cells, the memory wrapper, and the external memory storage for the ATPG vectors are unavoidable. Like functional testing, the operating frequency is determined by the ATE.

Finally, we have proposed and demonstrated designing a WEP BIST module for this WEP IP. Although the WEP BIST has taken more area overhead and its fault coverage values is less than the scan-chain, the BIST still has many exhilarating advantages. First, this BIST module generates the at-speed patterns to test the WEP IP automatically. And these at-speed vectors can verify the delay faults in CUT, and is independent of the operation frequency of the ATE. Second, the simple interface is very friendly to an end-user. Finally, the large extra memory storage for test vectors is not necessary.

In modern SoC design, it is impossible to finish a multimillion-gate design from only a few designers. Being a sub-module designer or an IP provider, how to qualify his design is a major task. In our thesis, we propose that a designer should pay more attention to insert some BIST circuit for some basic verification in an IP. These simple BIST circuits can save the testing time, the memory storage and reduce the complexity of the full-chip testing and

detect some delay faults with the at-speed testing.

At the end of this thesis, we bring up two future works. First, we hope that the "WEP IP with BIST" can be implemented in some SoC designs to verify the function of the WEP and to qualify the BIST circuit. Second, we wish that this thesis is an example and a reference for more future and BIST research and practice.

References

[1] Michael Keating and Pierre Bricaud, "Reuse Methodology Manual for System-on-a-Chip Design", Kluwer Academic Publishing, 1998, pp. 1-100

[2] Charles E. Stroud, "A Designer's Guide to Built-In Self-Test", Kluwer Academic Publishing, 2002, pp. 1-14

[3] L. M. S. C. of the IEEE Computer Society. "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications." IEEE Standard 802.11, First Edition, 1999, pp. 9-67

[4] R. L. Rivest, "The RC4 Encryption Algorithm", RSA Data Security, Inc., Mar 1992.

[5] "Verilog-XL User Guide", Cadence Design Systems, Inc, 1998.

[6] "Debussy User Guide and Tutorial", NOVAS Software, Inc., July 2003

[7] "Synopsys Online Documentation", Synopsys, Inc. October 1999.

[8] "TurboFault User's Guide", SynTest Technologies, Inc., December 1999.

[9] Alfred L. Crouch, "Design-for-Test for Digital IC's and Embedded Core Systems", Prentice-Hall, Inc., 1999, pp.1-240

[10] Michael L. Bushnell and Vishwani D. Agrawal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits", Kluwer Academic Publishing, 2000, pp. 253-308

[11] "TurboScan User's Guide" and "Full-Scan Tutorial", SynTest Technologies, Inc., August 2002.

簡　歷

姓　　名：　林隆裕

性　　別：　男

生　　日：　民國六十年二月二十二日

籍　　貫：　臺灣省台中縣

通訊地址：　新竹縣竹北市國盛街240巷15號1樓

學　　歷：　民國九十年七月至民國九十三年八月

　　國立交通大學電機資訊學院 電子與光電學程碩士班

民國七十九年七月至民國八十三年六月

　　私立淡江大學電機工程學系

經　　歷：　民國九十二年七月迄今

　　世紀民生科技股份有限公司

民國九十年七月至民國九十二年二月

　　金麗半導體股份有限公司

民國八十八年四月至民國八十九年十月

　　世紀科技股份有限公司

論文題目：　可測試性設計與內建自我測試技術於有線等效保密智財之應用與研究

A Case Study on DFT and BIST Design for a Wired Equivalent Privacy IP