MAJOR PROJECT REPORT

ON

Implementation of SHA Algorithm using VHDL for Hardware Security

A Major Dissertation Submitted in Partial Fulfilment of
the requirement for the award of degree of

MASTER OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Meenakshi

Roll No. 2811

Under the Guidance

of

Mrs. S. Indu
Senior Lecturer
Dept. of Electronics and Communication Engineering
Delhi University, Delhi

Department of Electronics and Communication Engineering

Delhi College of Engineering

June-2007

Acknowledgement

I wish to express my deep sense of gratitude and indebtness to **Mrs. S. Indu (Senior Lecturer, Dept. of Electronics And Communication Engineering)** for her guidance and assistance without which completion of this project report would not have been possible.

I also express my sincere gratitude to the **Prof. Asok Bhattacharyya (Head of Department, Dept. of Electronics And Communication Engineering)** and faculty of Dept. of Electronics And Communication Engineering Department.

I also express my sincere gratitude to **Mrs. Mini Cherion** (Scientist 'F', Divisional Head, Integrated Hardware and Software Group, Defense avionics Research Establishment, DRDO)

Last but not the least, I am thankful to my parents, my Husband and friends for their encouragement and guidance.

Meenakshi

Univ. roll no. 2811

Certificate

This is to certify that **Ms. Meenakshi** student of Delhi College of Engineering, Delhi worked under my guidance on major project entitled **"Implementation of SHA Algorithm using VHDL for Hardware security"** being submitted in partial fulfilment of the requirement for the award of the degree of "**MASTER OF ENGINEERING"** to the Department of Electronics and Communication Engineering, Delhi College of Engineering**,** Bawana Road, Delhi-42.

Prof. Asok Bhattacharyya
Head Of Division
Dept. Electronics & Comm. Engg.
Delhi University, Delhi

Mrs. S.Indu
Senior Lecturer,
Dept. Of Electronics & Comm. Engg.
Delhi University, Delhi

Department of Electronics and Communication Engineering
Delhi College of Engineering
Delhi-42

Abstract

With introduction of new technologies inventors are more concerned about the security of the new inventions they are bringing out. Data security is becoming ever more important in embedded and portable electronic devices. External interfaces to memory in digital devices and communication interfaces to other digital devices are more vulnerable to probing. The principal goal guiding the design of any encryption algorithm must be security against unauthorized attacks. These technique were sufficient to prevent the unauthorized access of the device but if the attack is at the device interface level not to access the system but to get the knowledge of complete architecture of device, algorithm used etc. This is very serious security threat from IP (intellectual property) point of view. So nowadays apart from the user authentication check device architecture information and algorithm are encrypted so as to avoid any kind of reverse engineering. The analysis techniques used by attackers are equally advanced. So the Defensive measures for protecting a device must be more sophisticated and robust.

This thesis presents an architecture that provides the security to the embedded application on a Field Programmable Gate Array (FPGA). The Identification Friend or Foe method is presented as the framework for creating a secure authentication system for the embedded system applications. It is shown that the Identification Friend or Foe method behaves like a secure wrapper around the user design and protects it from the leakage of the algorithm details of the device. The IFF concept is challenge and response based authentication scheme that protects the Intellectual Property from the threat of the cloning of the embedded application design. This scheme is capable of securing a variety of embedded applications. The cryptographic method used in the system is Secure Hash Algorithm. SHA is a standard specified in Federal Information Publication 180-1 and 180-2 (FIPS 180-1, FIPS 180-2). The SHA series hashes are currently the only FIPS-approved method. SHA-1 is also specified in ISO/IEC 10118-3.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

With the rapidly growing concerns of security in nearly every aspect of electronics system design, manufacturers and circuit designers are facing the challenges that never before existed. Companies are coming up with new technologies and ideas and countries are coming up with new devices and Weapons for Defense. With introduction of new technologies inventors are also concerned about the security of the idea they are bringing out. IP (Intellectual Property) security is becoming more and more important in today's competitive world. In the past security within electronic equipment was only faced by software related technologies, military and access control markets. This will face a change as designers will get set of new standard to meet in order to provide security.

[10]In early days security was provided only at the user authentication level. This technique was sufficient to prevent the unauthorized access of the device but if the attack is at the device interface level no to access the system but to get the knowledge of complete architecture of device, algorithm used etc. This is very serious security threat from IP (intellectual property) point of view. So nowadays apart from the user authentication check device architecture information and algorithm are encrypted so as to avoid any kind of reverse engineering.

[2] The principal goal guiding the design of any encryption algorithm must be security against unauthorized attacks. However, for all practical applications, performance and the cost of implementation are also important concerns. A data encryption algorithm would not be of much use if it is secure enough but slow in performance because it is a common practice to embed encryption algorithms in other applications such as e-commerce,

banking, and online transaction processing applications. Embedding of encryption algorithms in other applications also precludes a hardware implementation, and is thus a major cause of degraded overall performance of the system.

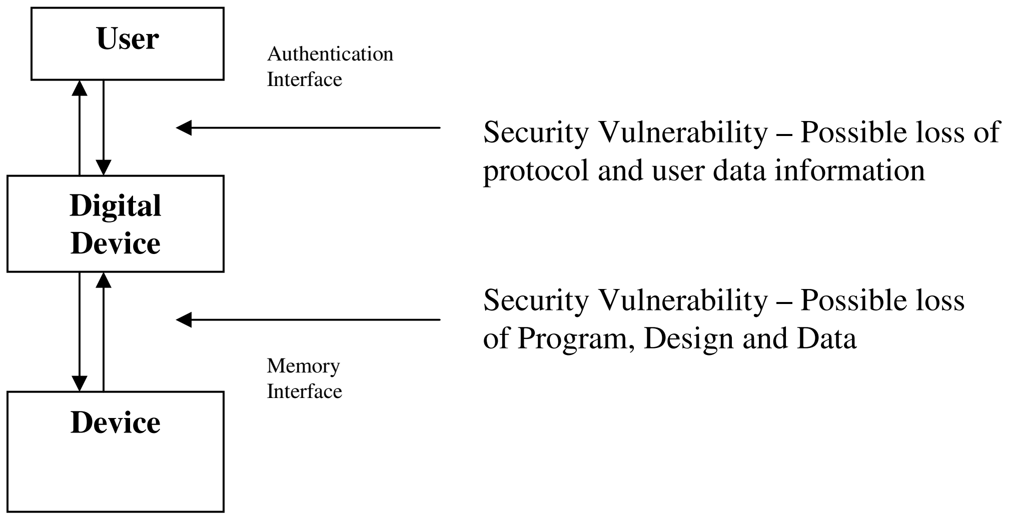

User
Authentication Interface
Security Vulnerability – Possible loss of protocol and user data information
Digital Device
Security Vulnerability – Possible loss of Program, Design and Data
Memory Interface
Device

Fig 1.1 Security Vulnerability at two difference interfaces

[1]In information security, message authentication is an essential technique to verify that received messages come from the alleged source and has not been altered. A key element of authentication schemes is the use of a message authentication code (MAC). One technique to produce a MAC is based on using a hash function and is referred' to as an HMAC. Secure Hash Algorithm 1 (SHA-I) is one of the algorithms, which has been specified for use in Internet Protocol Security (IPSEC), as the basis for an HMAC. As we shall show in the paper: it is reasonable to construct cryptographic accelerators using hardware implementations based on SHA-I hash algorithm

1.2 Overview

To secure the digital applications, the IFF (Identification Friend or Foe) concept is implemented in FPGA. This architecture creates a wrapper around the FPGA based application and provides authentication interface for user design. As the wrapper doesn't allow revealing the information to the unauthenticated user, thus the FPGA is secured from unauthorized usage of the user design.

The IFF concept has been implemented using Virtex-2Pro Development Platform. This design utilizes the key management systems to secure user design on the FPGA, thus protecting its contents from discovery. The Maxim Dallas DS2432 1Kbit Protected 1-wire EEPROM with SHA-1 Engine is used to authenticate a user and establishes a secure channel to the FPGA. The communication between the DS2432 and the FPGA is through 1-wire protocol. The IFF test is designed to authenticate the system by using the knowledge of Secret key, serial number, page data, and challenge present in the FPGA. If EEPROM is valid the user design is activated, otherwise only limited functions are made operational.

The IFF test has been synthesized using Xilinx 8.1i Foundation Series and ChipScope Pro 8.1i has been used for the real time debugging and on-chip verification of FPGA at operating system speed.

1.3 Thesis Organization

Chapter 1 gives the introduction of the intention of this thesis work. Chapter 2 is about the discussion about the background of Encryption technique and brief introduction of different encryption techniques and requirement of encryption techniques. Chapter 3 is the description of Secure Hash Algorithm anf IFFTest in detail. Chapter 4 explains the functional blocks for IFF concept. Functional description consists of explanation of 1

Wire Protocol Loadtest, Ifftest, Crc Generation, 3 To 8 Decoder. Chapter 5 describes the hardware platform used for implementation of SHA algorithm which deals with the overview Of DS2432. Chapter 6 gives the selected blocks of the SHA algorithm. Chapter 7 and 8 is about the discussion and results conclusion.

Chapter 2

Background

2.1 Need of Encryption

[4] Encryption is the process of obscuring information to make it unreadable without special knowledge. In cryptography, a *cipher* (or *cypher*) is an algorithm for performing encryption and decryption — a series of well-defined steps that can be followed as a procedure. Cipher is also called as *encipherment*. The concept of encryption is based on some Key value which changes the reception of data depending upon the correctness of the key.

The original in information/ message is known as plaintext and after encryption it is converted to ciphertext. As *ciphertext* is the encrypted form of the original message so it contains all the information as the original message but it can't be interpreted by a human or computer without some extra information which is called *key value*. The key value and *ciphertext* together are necessary and sufficient criteria to decrypt the plaintext.

The operation of a cipher usually depends on a piece of auxiliary information, called a key a *cryptovariable*. The encrypting procedure is varied depending on the key, which changes the detailed operation of the algorithm. A key must be selected before using a cipher to encrypt a message. Without knowledge of the key, it should be difficult, if not impossible, to decrypt the resulting *ciphertext* into readable plaintext.

2.2 Evolution of Encryption Techniques

Encryption techniques can be broadly classified into two classes, tradition and modern encryption techniques. Traditional encryption techniques were developed when there were no computers, so these are pen and paper based. With the invention of computer the computer era started and encryption techniques underwent a major change and the idea of working on bits instead on alphabets was conceived.

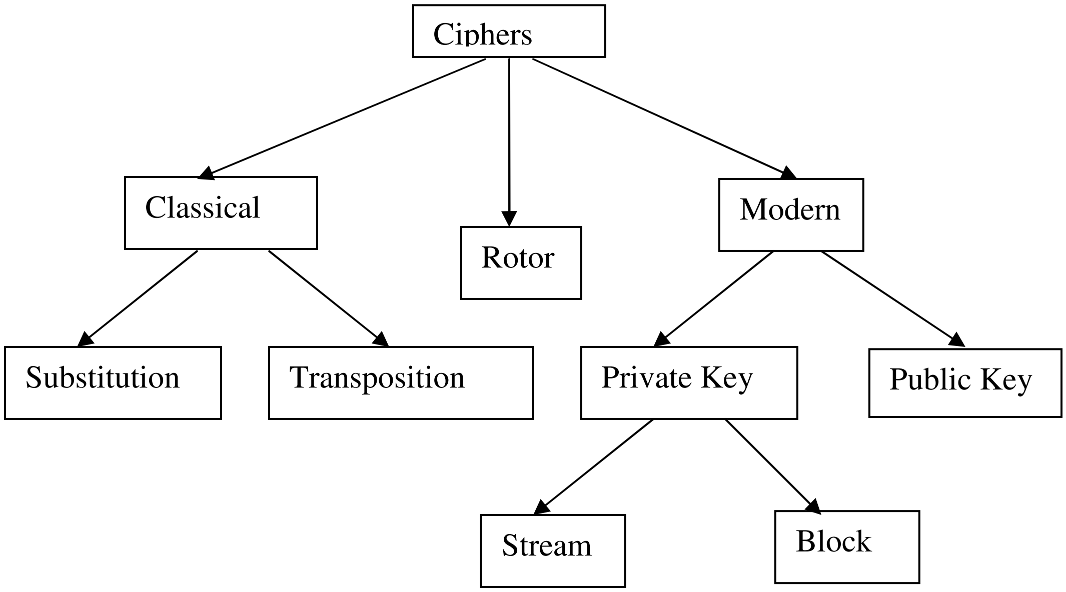


Fig 2.1 Evolution of Encryption Techniques

[3]Chinese were the first to use the encryption concept and written language itself was used as encryption technique. Caesar Cipher is the most popular traditional encryption method which was developed by Julius Caesar between 50 and 60 BC. Principal of substitution was used, where a letter is substituted by another letter. In 1553 the idea of a password was first given by Giovan Belaso. Gilbert Vernam developed the Vernam Cipher in 1917 which is the oldest encryption technique still in use. It uses substitution where no pattern can arise.

[3]Encryption played a major role in the Second World War in 1942. Many encryption techniques came as a result of necessity of perfect encryption techniques for the information security. American military used spoken and written language as an encryption device known as Navajo windtalkers. US military used wheel Ciphers in the First World War. German government created TYPEX machine from Enigma machine and used in the Second World War.

Historical pen and paper ciphers used in the past are sometimes known as classical ciphers. They include simple substitution ciphers and transposition ciphers. For example MEENAKSHI may be encrypted as ODFMZLRIH where all the odd position alphabet is replaced by next alphabet and even position character is replaced with previous alphabet These simple ciphers are easy to crack, even without plaintext-ciphertext pairs.

With the invention of computer it became very easy to break the algorithms, which were once very difficult to solve, in a short time. Now, it was not possible to rely on pen paper techniques anymore and need for encryption techniques specific to computers re-invented the encryption field. At this time the use of bits was focused instead of written alphabets.

2.3 Security Threats

[6] Attack is any malicious intent to subvert a system to defraud it. In other words attack is the intent to get access to a secure system. Few types of security are discussed here in short:

Copy Attack

Copy attack is done by copying the valid service data from a device which is part of the service to a device which is not part of the service. Purpose of this attack is to create an unauthenticated but technically valid copy of the service device to get any kind of advantage out of it.

Eavesdrop Attack

This technique is used where 1 wire communication is monitored to reveal the secret of a repeating pattern that could be replicated.

Emulation Attack

In this type of attack a microprocessor is used to emulate the behavior of a 1- Wire token. The emulator must be fast enough respond to a 1-Wre master as if it is a real device. This attack is not useful if authentication secret is not known. Risk can be further minimized by including a ROM ID as a component in the calculation.

Secret Brute Force Attack

The Secret Brute Force Attack tries all the possible combination until it gets the correct Service Identification code called MAC.A Token which is part of the service supplies the correct MAC. After getting the secret it can be used to try all possible options in few seconds.

Secret Microprobe Physical Attack

The physical attack is an attempt to probe the internal silicon chip to read the Unique Identification Secret.

2.2.1 Modern Encryption Techniques

2.2.1.1 Symmetric key algorithms (Private-key cryptography) - In a symmetric key algorithm (e.g., DES and AES), the sender and receiver must have a shared key set up in advance and kept secret from all other parties; the sender uses this key for encryption, and the receiver uses the same key for decryption.

Symmetric key ciphers can be distinguished into two types

- Block ciphers- work on blocks of symbols of fixed size
- Stream ciphers – Work on a continuous stream of symbols.

2.2.1.2 Asymmetric key algorithms (Public-key cryptography)- In an asymmetric key algorithm (e.g., RSA), there are two separate keys: a public key is published and enables any sender to perform encryption, while a private key is kept secret by the receiver and enables only him to perform decryption.

2.3 Requirement of Encription Techniques

Encryption Technique mush be able to provide/Ensure Integrity, Authenticity and Confidentiality of the information.

Integrity – It means that the receiver of data can detect any modification of the data on the way from the sender to the receiver.
Authenticity - Property that allows verifying that the data really originates from the alleged sender.
Confidentiality - Feature that enables the sender to transform the original data in a way that only the designated recipient can reconstruct it. No eavesdropper between sender and receiver can recover the original data.

Chapter 3

Algorithm Description

3.1 Secure Hash Algorithm

3.1.1 Introduction of Secure Hash Algorithm

[5] SHA is a standard specified in Federal Information Publication 180-1 and 180-2 (FIPS 180-1, FIPS 180-2). The SHA series hashes are currently the only FIPS-approved method. SHA-1 is also specified in ISO/IEC 10118-3.

The Secure Hash Algorithm (SHA-1) is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for federal applications. For a message of length < 2^64 bits, the SHA-1 produces a 160-bit condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

3.1.2 Properties of a Secure Hash Algorithm

1. **Irreversibility** - making it computationally infeasible to determine the input corresponding to a HASH result.
2. **Collision-resistance** – One Hash result should not produce more than one input message.
3. **High avalanche effect** – Large change in Output for slight change in input.

The input bit string of length < 2^{64} is called as the *message* and the condensed output of 160 bits is called as *message digest*.

3.1.3 Terminology

Following terminologies related to the bit string will be used in this algorithm:

1. Message: The input string which can be of any length l < 2^64 .
2. Message Digest: Condesed output of length 160 bits.
3. Word: A word is represented by a 32 bit string.
4. Block: A block is 512 bit string which may be represented by 16 words.

The following operations will be applied to the words:

1. X AND Y : Bitwise logical AND of X and Y.
2. X OR Y : Bitwise logical "inclusive OR" of X and Y.
3. X XOR Y : Bitwise logical "exclusive OR" of X and Y.
4. NOT X : Bitwise logical " complement " of X.
5. $S^n(X)$: Circular left shift of X by n positions to the left.

3.1.4 Message padding

SHA-1 computes the message digest for the input message bit string provided. The message is processed as 512 bits at a time. So message padding is done to make the length multiple of 512 bits. The SHA-1 processes the 512 bit blocks sequentially to compute the message digest of 160 bits. In this process 1 is appended at the end of the message and then enough zeros followed by 64-bit representation of the length of the original message so that the padded message is multiple of 512 bits.

The padded message can be represented as M_1, M2, M3,…,M_n where M_i is 16 words block and M_1 contains the first character of the input message.

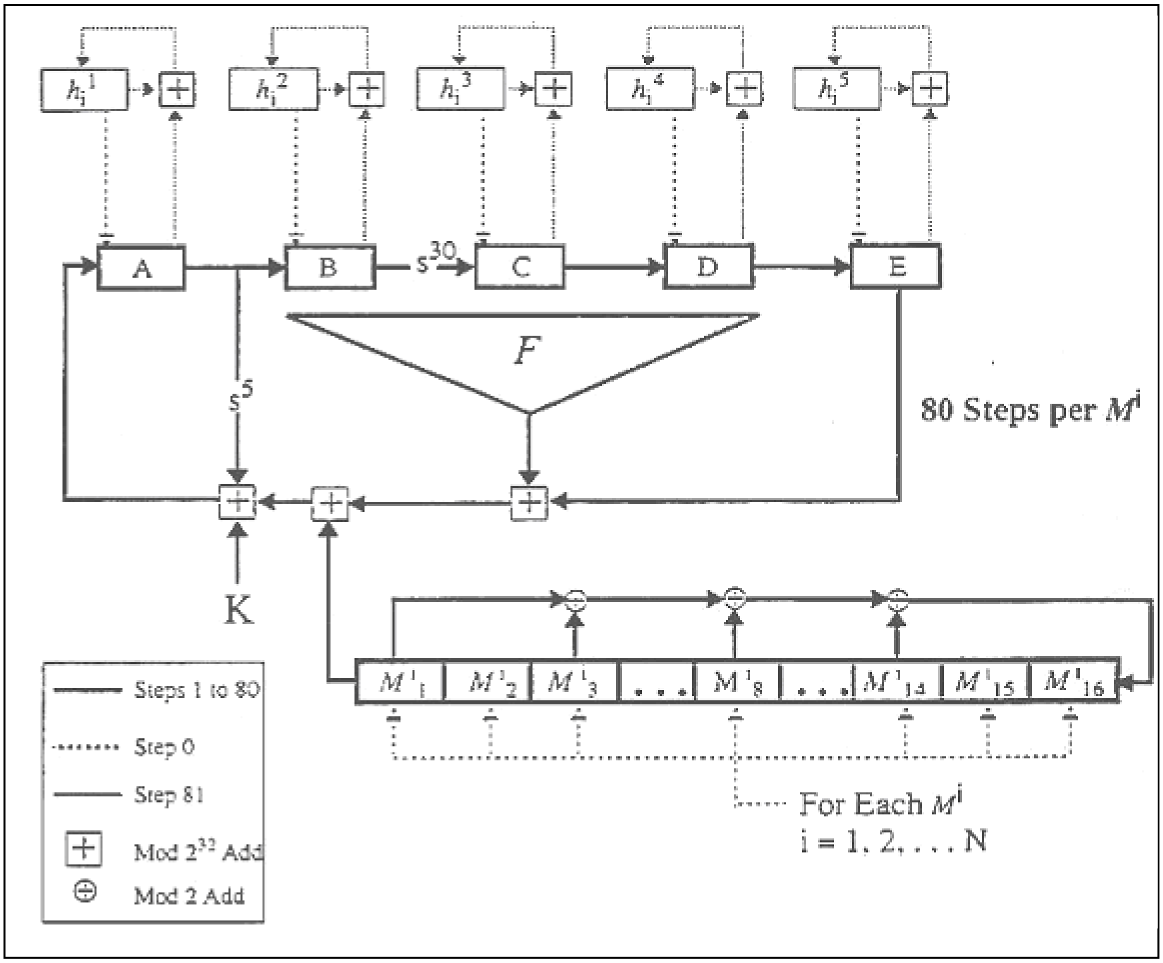


Fig 3.1 Secure Hash Algorithm

3.1.4 Algorithm

SHA-1 is based on a nonlinear function f_t which operates on 32-bit words B,C,D and produces a 32-bit word. f_t(B,C,D) is defined as follows:

$$f_t(B,C,D) = \begin{cases} \text{(B AND C) OR ((NOT B) AND D)} & (0 <= t <= 19) \\ \text{B XOR C XOR D} & (20 <= t <= 39) \\ \text{(B AND C) OR (B AND D) OR (C AND D)} & (40 <= t <= 59) \\ \text{B XOR C XOR D} & (60 <= t <= 79) \end{cases}$$

SHA-1 uses a sequence of constant words defined as:

$$K_t = \begin{cases} \text{0x5A827999} & (0 <= t <= 19) \\ \text{0x6ED9EBA1} & (20 <= t <= 39) \\ \text{0x8F1BBCDC} & (40 <= t <= 59) \\ \text{0xCA62C1D6} & (60 <= t <= 79). \end{cases}$$

The 512-bit message block is broken into sixteen 32-bit words m_0,m_1,m_2,…,m15 which are used for the calculating the 80 words W_0,W_1,W_2,…,W_{79} using formula :

$$W_t = \begin{cases} m_t & (0 <= t <= 15) \\ \text{St(Wt-3 XOR Wt-8XOR Wt-14 XOR Wt-16)} & (16 <= t <= 19) \end{cases}$$

In the beginning of the process five constant words are initialized as follows:

H_0 = 0x67452301

H_1 = 0xEFCDAB89

H_2 = 0x98BADCFE

H_3 = 0x10325476

H_4 = 0xC3D2E1F0

Before calculation starts five word variables A, B, C, D, E are initialized as:

$A = H_{0,}\ B = H_{1,}\ C = H_{2,}\ D = H_{3,}\ E = H_4$

SHA-1 algorithm is as follows:

```
For t = 0 to 79
        TEMP = $S^5$ (A) + $f_t$(B,C,D) + E + $W_t$ + $K_t$;
        E = D;
```

$D = C;$

$C = S^{30}(B);$

$B = A;$

$A = TEMP$

After all these calculations the values of $H_{0,}$ H_1, H_2, H_3 and H_4 are updated as follows

$H_0 = H_0 + A;$

$H_1 = H_1 + B;$

$H_2 = H_2 + C;$

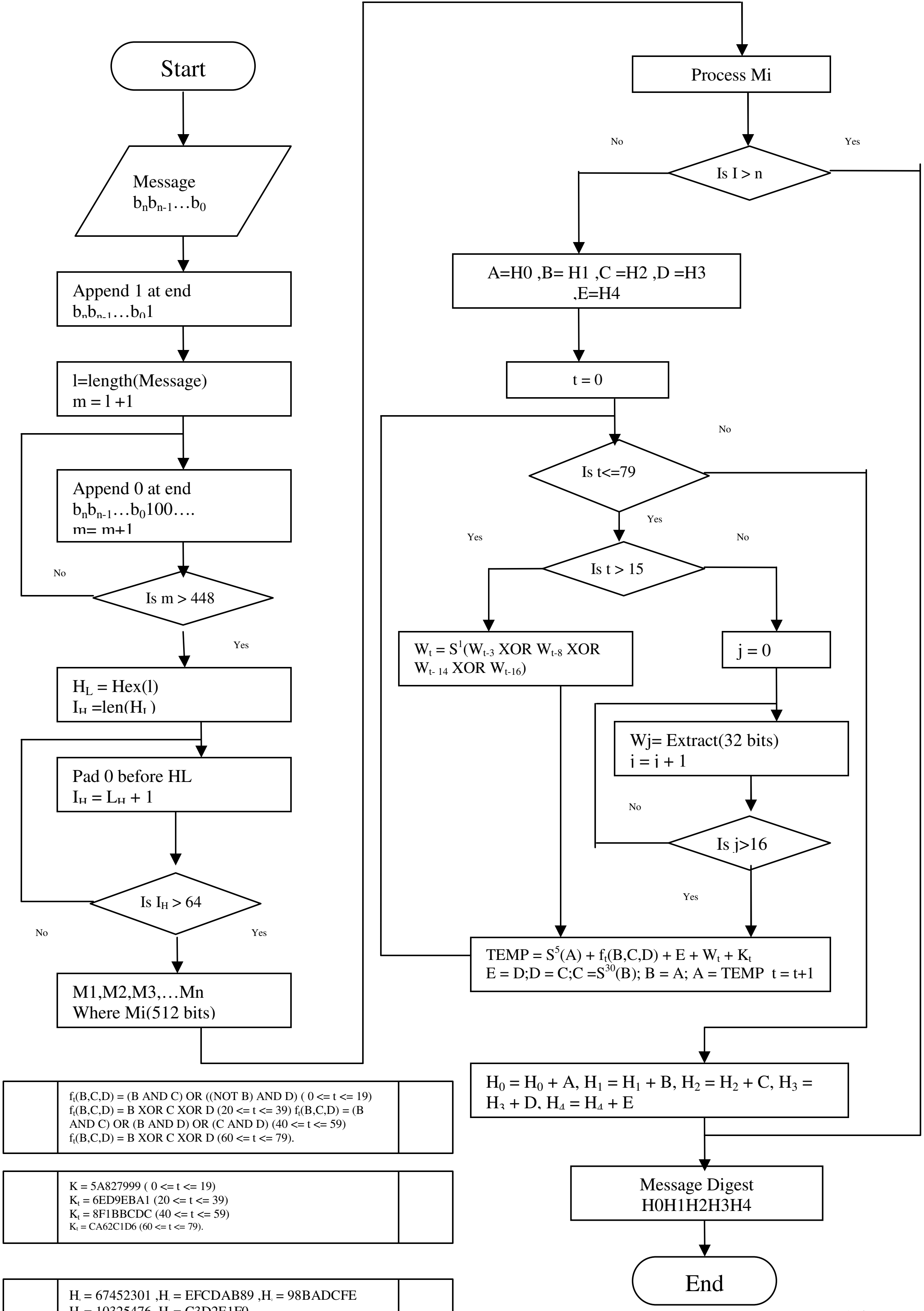
$H_3 = H_3 + D;$

$H_4 = H_4 + E.$

If there are more 512-bit blocks then these are processed using the updated values of $H_{0,}$ H_1, H_2, H_3 and H_4. After the processing of the last block the condensed 160-bit output, message digest, is the concatenation of the final values of $H_{0,}$ H_1, H_2, H_3 and H_4 :

Message digest = $H_0H_1H_2H_3H_4$

3.1.5 Flow chart



$f_t(B,C,D)$ = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)
$f_t(B,C,D)$ = B XOR C XOR D (20 <= t <= 39) $f_t(B,C,D)$ = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)
$f_t(B,C,D)$ = B XOR C XOR D (60 <= t <= 79).

K = 5A827999 (0 <= t <= 19)
K_t = 6ED9EBA1 (20 <= t <= 39)
K_t = 8F1BBCDC (40 <= t <= 59)
K_t = CA62C1D6 (60 <= t <= 79).

H_0 = 67452301 ,H_1 = EFCDAB89 ,H_2 = 98BADCFE
H_3 = 10325476 ,H_4 = C3D2E1F0.

3.2 IFFTEST Concept

[7] Figure 3.2 shows the IFF concept. The following steps describe the concept of the Identification as Friend and Foe test for the authentication of the Secure EEPROM DS2432:

1. FPGA sends a Challenge ‘Q’ to DS2432.

2. The DS2432 uses a secret key programmed by the designer which is known only to the designer and SHA engine generates a 512-bit MAC, the actual response ‘A’ for the Challenge ‘Q’.

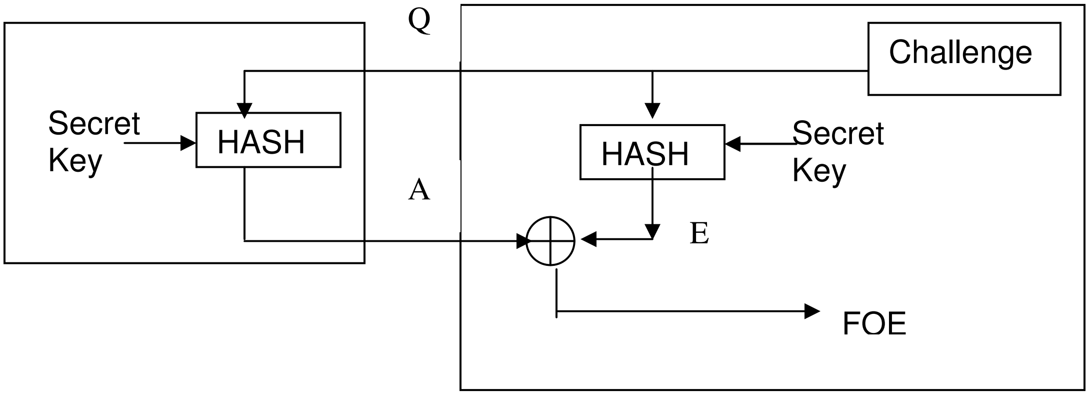


Fig. 3.2 Identification Friend and Foe

3. The FPGA calculates the MAC, the expected response ‘E’ using the same key and compares it with the actual response A, produced by the DS2432.

4. Now expected and actual response is matched to determine whether the design is Friend or Foe. If it matches design is Friend, otherwise the design is Foe. It may occur because of the tampering.

5. FPGA application is designed in such a way that if design is detected as Foe, then either application does not operate or it operates with limited functionality. If the design is detected as Friend then it operates correctly and all features are operational.

Chapter 4

Architecture

4.1 Functional Blocks

The design includes six functional blocks:

1. **1-wire Protocol**: For the interfacing of EEPROM with the FPGA.

2. **IFFTEST**: To authenticate the user design in FPGA using EEPROM and the secret key written in the FPGA.

3. **LOADTEST**: To load the secret key in the EEPROM. It is not included in the user design.

4. **SHA (Secure Hash Algorithm)**: To generate unique 160-bit message digest using 512-bits as input.

5. **3 to 8 decoder**: It converts the decimal number from 0 to 7 into the binary equivalent. It is designed only for demonstration purpose.

6. **CRC (Cyclic Redundancy Check)**: To determine whether the ROM data has been read without error or not.

4.1.1 Wire Protocol

[8] The DS2432 uses 1-wire protocol to communicate with host on single line. It consists of following type of signaling:

- Reset Sequence with Reset Pulse and Presence Pulse
- Write 0
- Write 1
- Read Data.

All these signals are initiated by the bus master except the presence pulse. The Fig. 4.1 shows the initialization process. DS2432 receives data when a reset pulse is followed by the Presence Pulse. If the device is in Overdrive mode then t_{RSTL} is less than 80µs and if

t_{RSTL} is longer than 480µs it operates on standard speed. Speed is undetermined if t_{RSTL} is between 80µs and 480µs.

Once the bus master has released the line it goes into the receive mode. The pullup resistor pulls the 1-wire bus to V_{PUP}. The DS2432 waits for the t_{PDH} after the threshold V_{TH} is crossed and then transmits a Presence Pulse for t_{PDL} by pulling the line low.

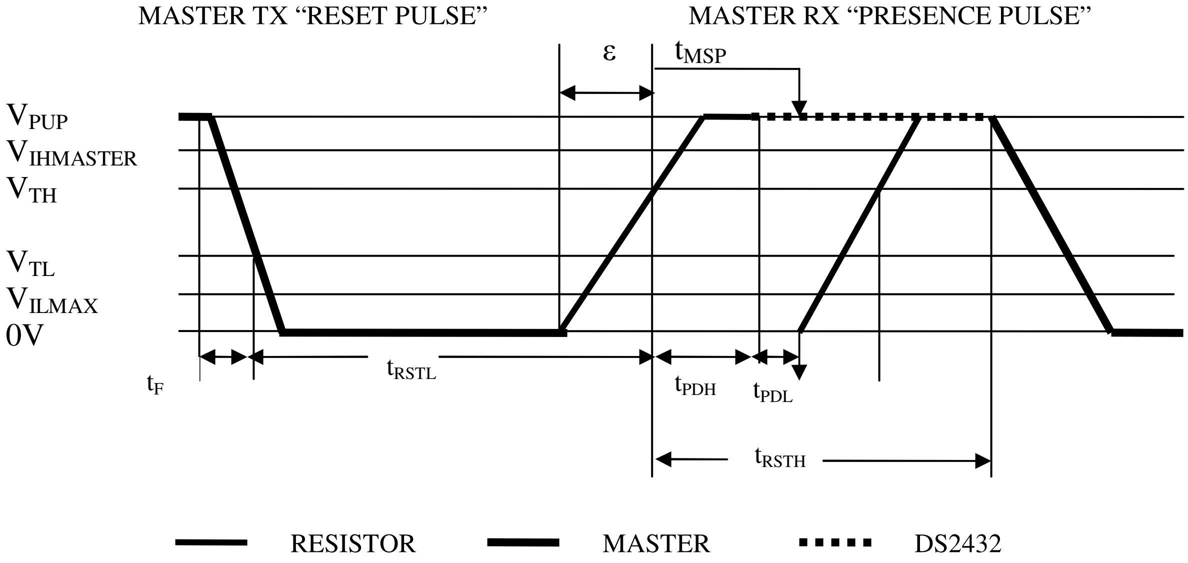


Fig. 4.1 Initialization Procedure: Reset and Presence Pulse

During data communication with DS2432 each time slot carries a single bit. During the Write time slots data is transferred from bus master to slave and during Read time slots data transfer takes place from slave to master. Master pulls the data line low to start communication. DS2432 starts its internal timing generator, when voltage on 1-Wire line falls below threshold V_{TH}, to determine when to sample the data line during a write time slot and how data is valid during Read time slot.

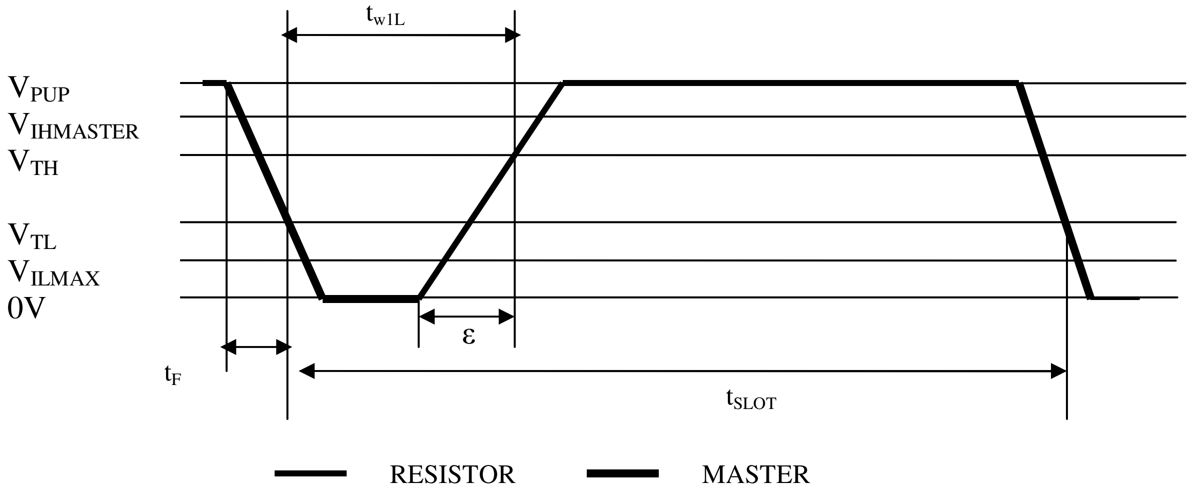


Fig. 4.2 Write One Time Slot

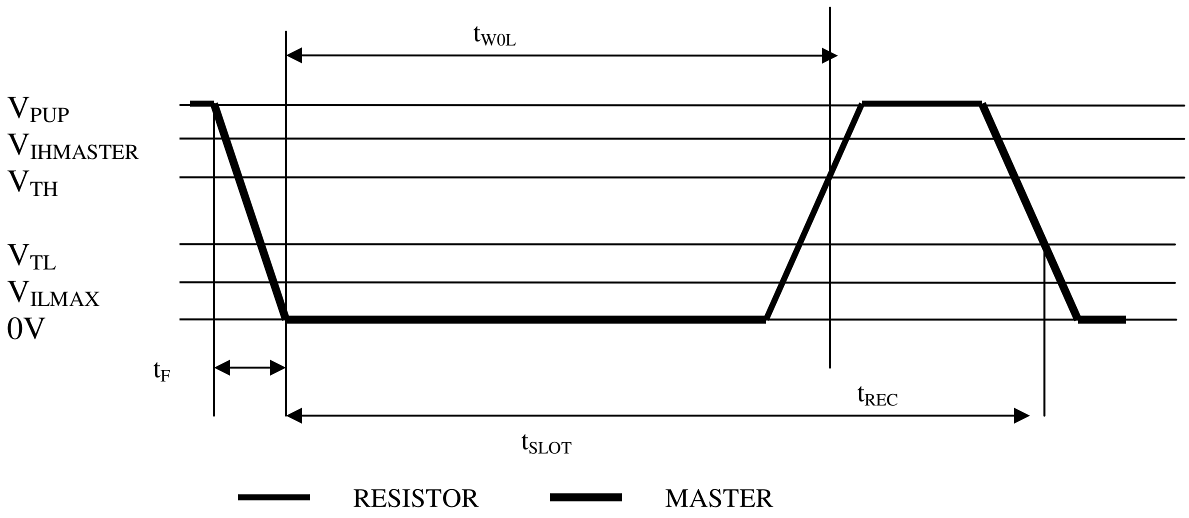


Fig. 4.3 Write-Zero Time Slot

In the Fig 4.2 write-One time slot the voltage on data line should cross the V_{TH} before expiration of write one low time t_{W1LMAX}. In the Fig. 4.3 write-zero time slot the voltage on data line should stay below V_{TH} before expiration of write one low time t_{W0LMIN}.

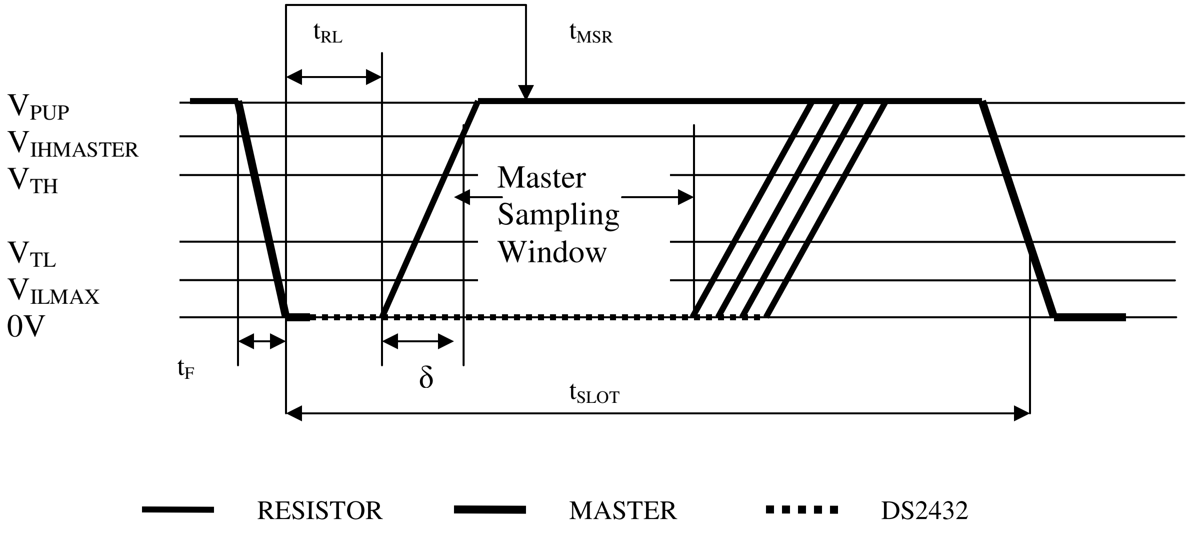


Fig. 4.4 Read-Data Time Slot

During the Read time slot (Fig 4.4) the data line voltage should remain low till expiration of read time low time t_{RL}. While responding to 0, the DS2432 pulls the data line low and its internal timing generator will determine when this pull down ends and the voltage starts rising again. . While responding to 1, the voltage will rise again when t_{RL} is over. The master should wait for t_{SLOT} to expire after reading from the data line. It ensures the recovery for DS2432 to start next time slot.

4.1.2 LOADTEST

Loading of the key into the DS2432 device is done using the LOADTEST design. The 64-bit Secret key is loaded in a very secure environment and it is protected from the leakage of information. LOADTEST is not included in the final user design. The figure 4.5 shows the instantiation block diagram of the LOADTEST design. The inputs to the LODATEST design are:

1. C100MHz: The available clock of 100MHz is used as input and it is divided by 100 to get the 1MHz frequency. It is done to met the timing specifications of the DS2432.
2. RSTIN: An active-Low RSTIN signal is provided for the initialization of the design process.

3. START: Start signal initiates the loading process of the Secret Key. It’s an active-High signal.
4. BI: BI is single wire interface between the DS2432 and the FPGA. All communication between the two devices is possible through this signal only.
5. LDDONE: This signal shows the status of the LOADTEST design. This signal will be high only if the key is loaded successfully, otherwise it remains low. It’s an active high signal.

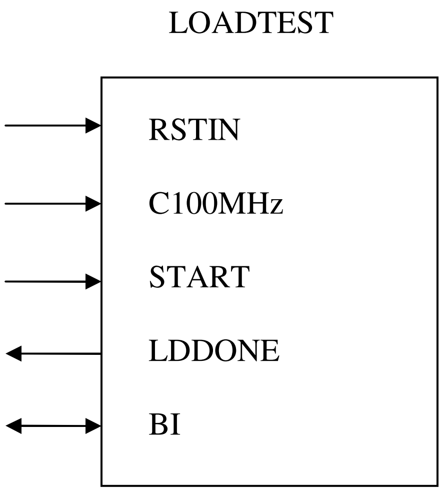
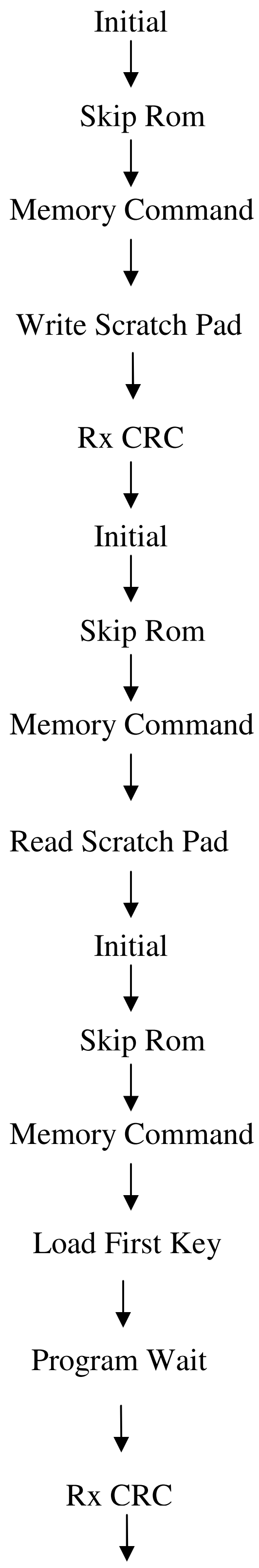
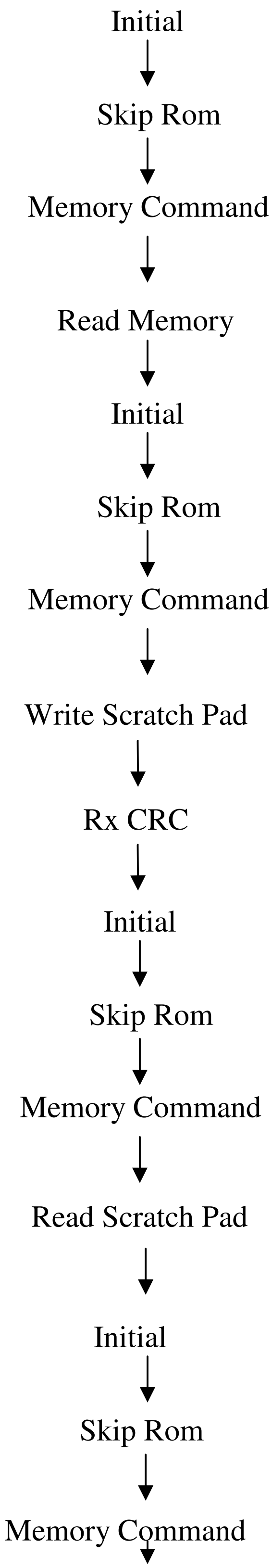
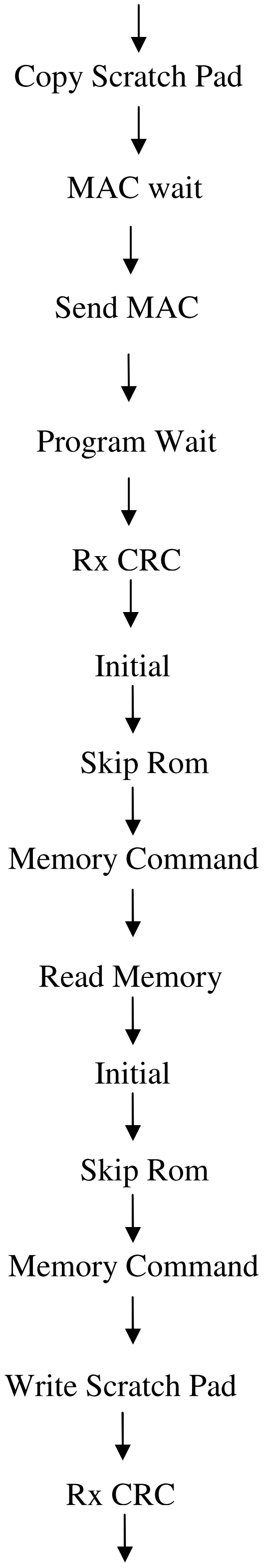


Fig. 4.5 LOADTEST Instantiation Block Diagram

The Fig. 4.6 shows the data flow of the LOADTEST design.


Initial
Skip Rom
Memory Command
Write Scratch Pad
Rx CRC
Initial
Skip Rom
Memory Command
Read Scratch Pad
Initial
Skip Rom
Memory Command
Load First Key
Program Wait
Rx CRC


Initial
Skip Rom
Memory Command
Read Memory
Initial
Skip Rom
Memory Command
Write Scratch Pad
Rx CRC
Initial
Skip Rom
Memory Command
Read Scratch Pad
Initial
Skip Rom
Memory Command


Copy Scratch Pad
MAC wait
Send MAC
Program Wait
Rx CRC
Initial
Skip Rom
Memory Command
Read Memory
Initial
Skip Rom
Memory Command
Write Scratch Pad
Rx CRC

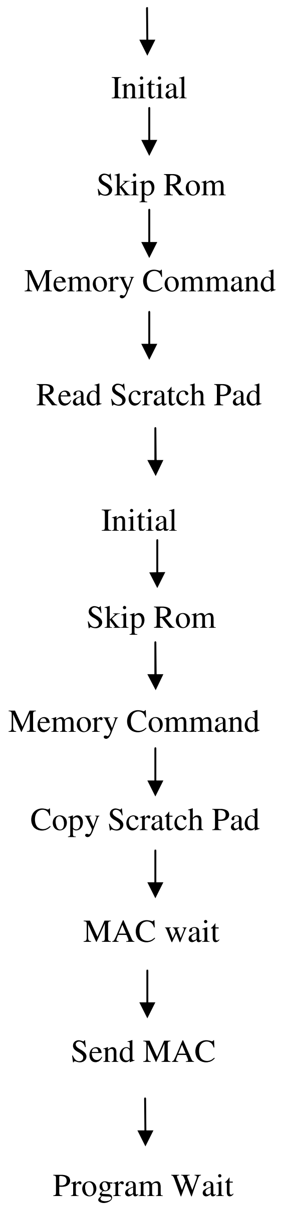


Fig. 4.6 Load Test Flow diagram

4.1.3 IFFTEST

IFFTEST is designed as a part of user design to authenticate the system. The DS2432 is used as the platform for the IFFTEST. The MAXIM-DALLAS DS2432 consists of a 64-bit secret key which can be rewritten by the user but can’t be read back from the external interface. The DS2432 also consists a 64-bit serial number which is unique to all device, no two devices can have same serial number. IFFTEST is designed in such a way that the user design is deactivated on power on. Upon power on the IFFTEST authenticate the system by using the knowledge of Secret key, serial number, page data, and challenge present in the FPGA. If EEPROM is valid the user design is activated, otherwise only limited functions are made operational. The Fig. 4.7 shows the instantiation block diagram of the IFFTEST design. The inputs to the IFFTEST design are:

1. C100MHz: The available clock of 100MHz is used as input and it is divided by 100 to get the 1MHz frequency. It is done to met the timing specifications of the DS2432.
2. RSTIN: An active-Low RSTIN signal is provided for the initialization of the design process.
3. START: START signal initiates the authentication process of the DS2432, the Secret EEPROM device. It’s an active-High signal.
4. FRND_OUT: This signal shows the status of the IFFTEST design. This signal will be high only if the secret EEPROM device is Friend i.e. authenticated and its low if the system integrity is altered.
5. BI: BI is single wire interface between the DS2432 and the FPGA. All communication between the two devices is possible through this signal only.

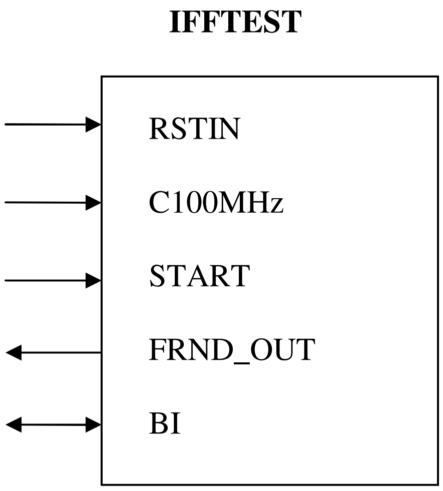


Fig. 4.7 IFFTEST instantiation Block Diagram

The Fig. 4.8 describes the process of EEPROM authentication.

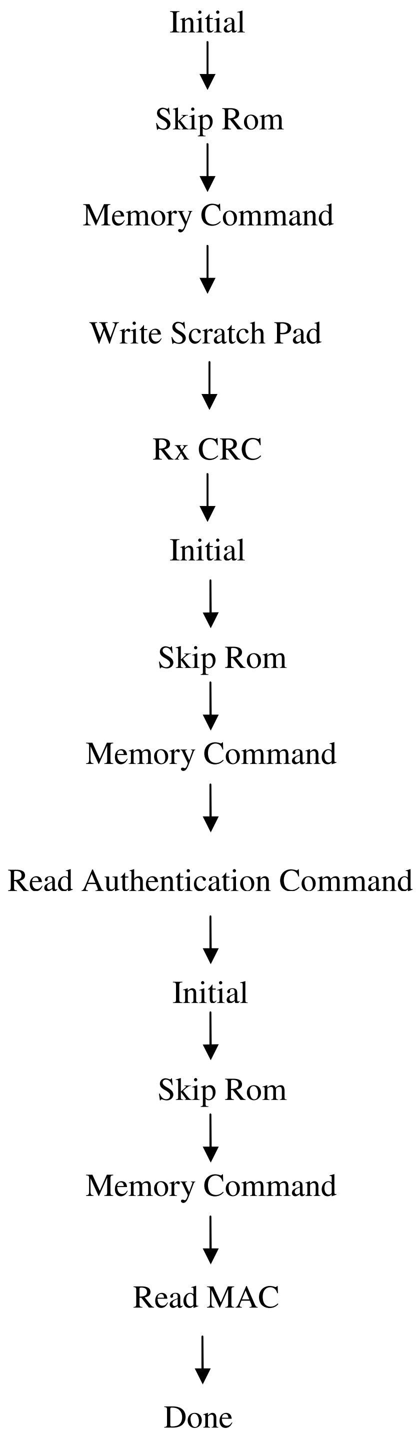


Fig. 4.8 IFF Test Flow diagram

4.1.4 CRC Generation

With the DS2432 there are two different types of CRCs (Cyclic Redundancy Checks). One CRC is an 8-bit type. It is computed at the factory and lasered into the most significant byte of the 64-bit ROM. The equivalent polynomial function of this CRC is $X^8 + X^5 + X^4 + 1$. To determine whether the ROM data has been read without error the bus master can compute the CRC value from the first 56 bits of the 64-bit ROM and compare it to the value read from the DS2432. This 8-bit CRC is received in the true form (non-inverted) when reading the ROM.

The other CRC is a 16-bit type, generated according to the standardized CRC16-polynomial function $X^{16} + X^{15} + X^2 + 1$. This CRC is used for error detection with the Read Authenticated Page command, when reading the scratchpad and for fast verification of a data transfer when writing to the scratchpad. In contrast to the 8-bit CRC, the 16-bit CRC is always communicated in the inverted form. A CRC-generator inside the DS2432 chip calculates a new 16-bit CRC as shown in the command flow chart of Fig. 4.9 . The bus master may compare the CRC value read from the device to the one it calculates from the data and decide whether to continue with an operation or to re-read the portion of the data with the CRC error.

With the Write Scratchpad command the CRC is generated by first clearing the CRC generator and then shifting in the command code, the Target Addresses TA1 (with T2 to T0 set to 0) and TA2, and all data bytes as sent by the master. The DS2432 transmits this CRC only if the master has sent exactly eight bytes.

With the Read Scratchpad command the CRC is generated by first clearing the CRC generator and then shifting in the command code, the Target Addresses TA1 and TA2, the E/S byte, and the scratchpad data, which may have been modified by the DS2432.

The DS2432 will transmit this CRC only if the reading continues through the end of the scratchpad.

With the Read Authenticated Page command the 16-bit CRC value is the result of shifting the command byte into the cleared CRC generator, followed by the two address bytes, the data bytes, and the FFh byte. The CRC that follows the Message Authentication Code (MAC) results from clearing the CRC generator and then shifting in the 160-bit MAC in the same bit sequence as the master receives it.

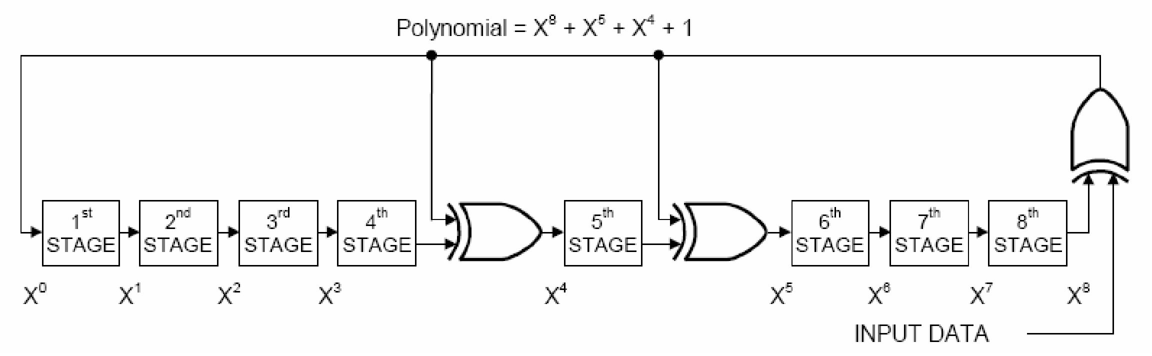


Fig. 4.9 CRC-8 Hardware Description and Polynomial

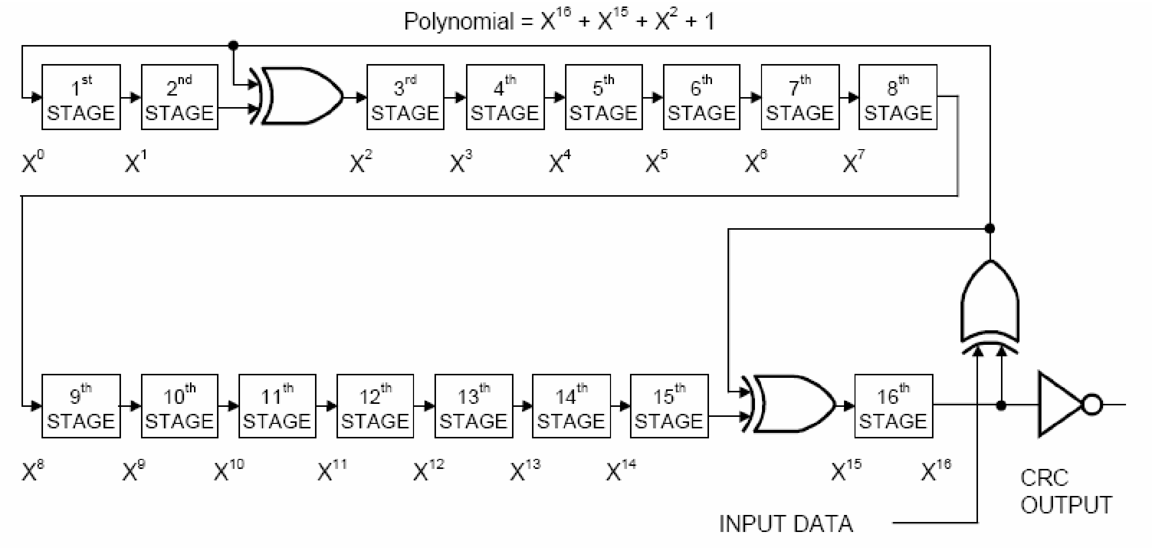


Fig. 4.10 CRC-16 Hardware Description and Polynomial

4.1.5 3 to 8 Decoder

3 to 8 decoder has been implemented for the verification of the design. This functional block is the main application which is to be protected from the security attacks. This application is executed only when it is authenticated against the secure EEPROM DS2432. 3 to 8 decoder converts the decimal numbers from 0 to 7 into their equivalent binary value.

Chapter 5

Platform

5.1 Implementation

We have used DALLAS Semiconductor/ MAXIM DS2432 1KBit protected 1-Wire secure EEPROM with SHA Engine and VIRTEX-2PRO P4 development kit to implement the IFF concept.

The features of the DS2432 secure EEPROM include:

1. 64-bit read-only unique serial number (no two devices share the same ID).
2. 64-bit write-only secret key that can be rewritten at any time, but there is no way of reading it back.
3. Four 256-bit pages that can be write protected.
4. Five general purpose read/write registers.
5. On-chip 512-bit SHA-1(ISO/IEC 10118-3) engine to compute 160-bit Message Authentication Codes (MAC)
6. Serial 1 wire interface for low pin count.
7. Reads and writes over a wide voltage range of 2.8V to 5.25V from –40^{0}C to +85^{0}C.

The 1-wire PROTOCOL is implemented for interfacing FPGA and the secure EEPROM. LOADTEST loads the secret key into the EEPROM and its not included in the final user design. IFFTEST authenticates the EEPROM .It enables the design only when FPGA contains the valid key otherwise, the design is disabled. A 3 to 8 decoder is implemented for the verification purpose.

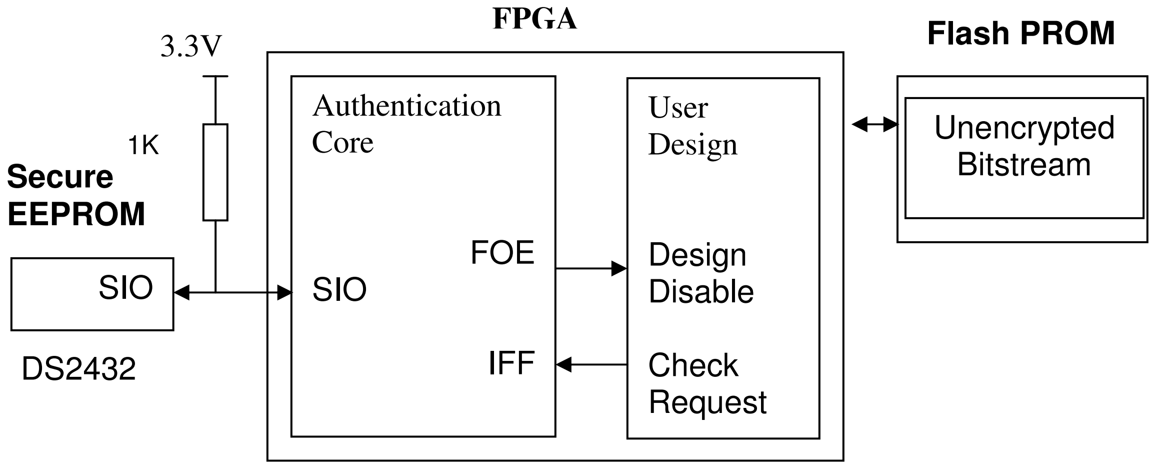


Fig. 5.1 DS2432 Secure EEPROM Connectivity to XILINX FPGA

The Fig 5.1 shows the connectivity diagram between the DS2432 secure EEPROM and FPGA to implement the copy protection scheme. First, the FPGA configures itself from a flash PROM. When the FPGA is configured, the user design is automatically disabled until it authenticates with the secure EEPROM using a secret key that is stored in the FPGA against the stored encrypted key in the secure EEPROM.

5.2 Overview of DS2432

[8] The DS2432 has five main data components:

1) 64-bit lasered ROM,
2) 64-bit scratchpad,
3) four 32-byte pages of EEPROM,
4) 64-bit register page,
5) 64-bit Secrets Memory, and
6) a 512-bit SHA-1 Engine

The bus master must first provide one of the seven ROM Function Commands:

1) Read ROM,
2) Match ROM,
3) Search ROM,
4) Skip ROM,
5) Resume Communication,
6) Overdrive-Skip ROM
7) Overdrive-Match ROM.

The protocol required for these ROM function commands is described in *Figure* . After a ROM function command is successfully executed, the memory and SHA-1 functions become accessible and the master can provide any one of the seven available function commands. The protocol for these memory function commands is described in *Figure* . All data is read and written least significant bit first.

5.2.1 64-BIT LASERED ROM

Each DS2432 contains a unique ROM code that is 64 bits long. The first eight bits are a 1-Wire family code. The next 48 bits are a unique serial number. The last eight bits are a CRC of the first 56 bits. The 1-Wire CRC is generated using a polynomial generator consisting of a shift register and XOR gates as shown in *Figure*. The polynomial is X_8 + X_5 + X_4 + 1. The shift register bits are initialized to zero. Then starting with the least significant bit of the family code, one bit at a time is shifted in. After the 8th bit of the family code has been entered, then the serial number is entered. After the 48th bit of the serial number has been entered, the shift register contains the CRC value. Shifting in the eight bits of CRC should return the shift register to all zeros.

5.2.2 Memory

The DS2432 has four memory areas: data memory, secrets memory, register page with special function registers and user-bytes, and a scratchpad. The data memory is organized in pages of 32 bytes. Secret, register page and scratchpad are 8 bytes each. The scratchpad acts as a buffer when writing to the data memory, loading the initial secret or when writing to the register page.

The data memory and the register page have unrestricted read access. Writing to the data memory and the register page requires the knowledge of the secret .The secret can be installed either by copying data from the scratchpad to the secrets memory The secret cannot be read directly; only the SHA engine has access to it for computing message authentication codes.

5.2.3 Address Registers and Transfer Status

The DS2432 employs three address registers: TA1, TA2 and E/S DS2432. Registers TA1 and TA2 must be loaded with the target address to which the data will be written or from which data will be read. Register E/S is a read-only transfer-status register, used to verify data integrity with write commands. Since the scratchpad of the DS2432 is designed to accept data in blocks of eight bytes only, the lower three bits of TA1 will be forced to 0 and the lower three bits of the E/S register (Ending Offset) will always read 1. This indicates that all the data in the scratchpad will be used for a subsequent copying into main memory or secret. is not an integer multiple of 8 or if the data in the scratchpad is not valid due to a loss of power. A valid write to the scratchpad will clear the PF bit. Bits 3, 4 and 6 have no function; they always read 1. The Partial Flag supports the master checking the data integrity after a Write command.

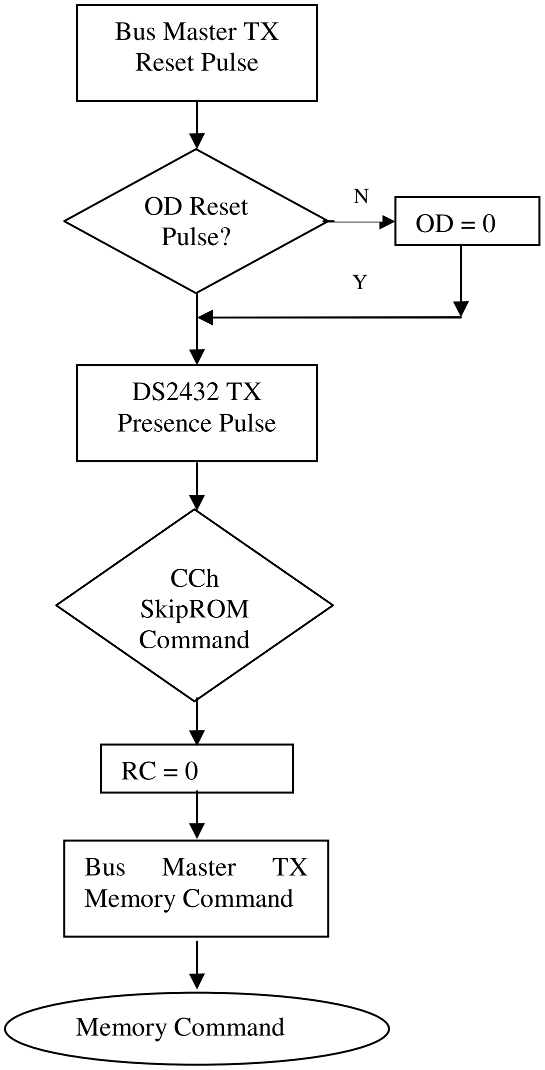
5.2.4 Writing With Verification

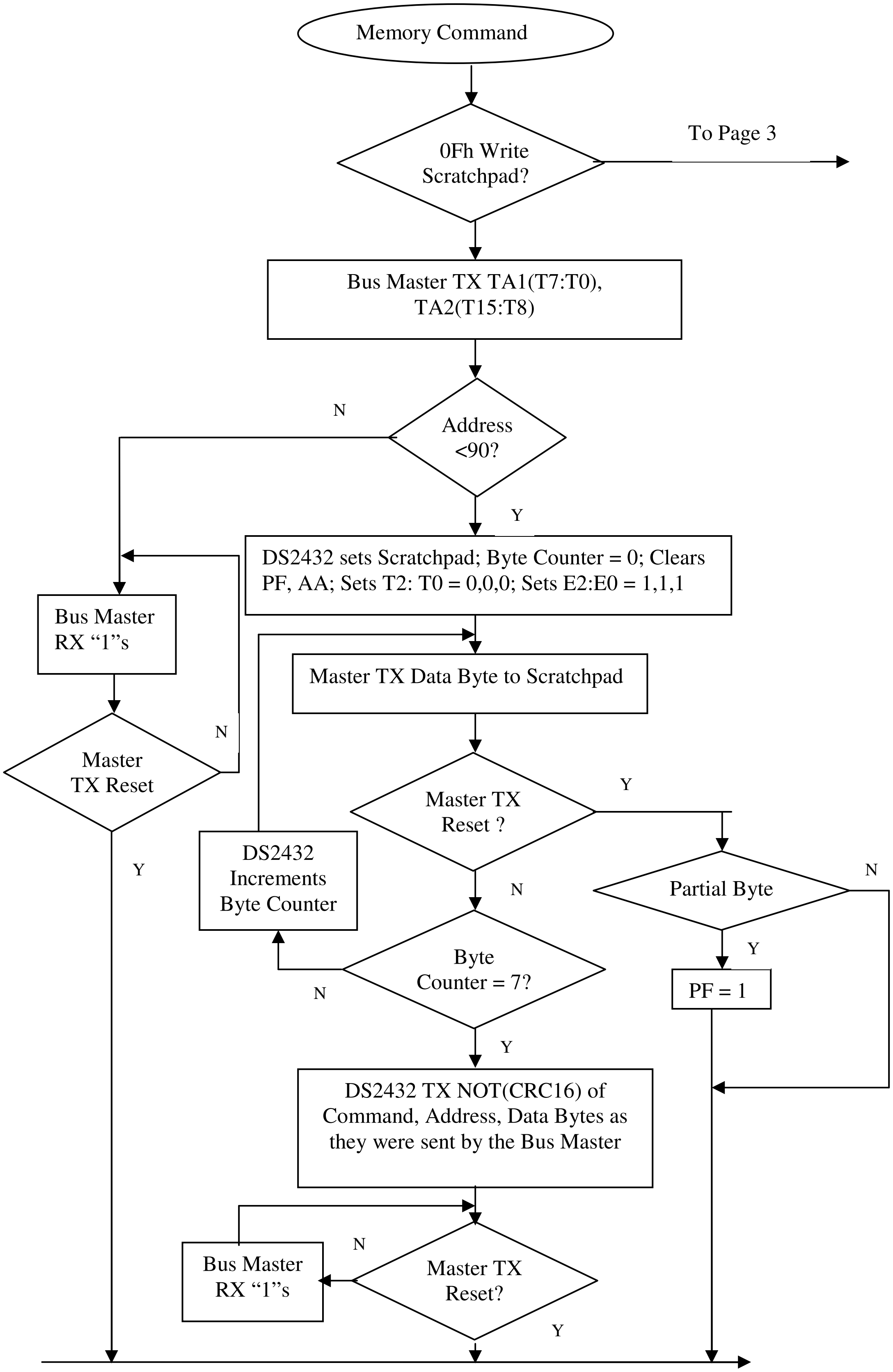
To write data to the DS2432, the scratchpad has to be used as intermediate storage. First the master issues the Write Scratchpad command to specify the desired target address, followed by memory when commanded; therefore eight bytes of data should be written into the scratchpad to ensure that the data to be copied is known. Under certain conditions the master will receive an inverted CRC16 of the command, address and data at the end of the write scratchpad command sequence. Knowing this CRC value, the master can compare it to the value it has calculated itself to decide if the communication was successful and proceed to the Copy Scratchpad command. If the master could not receive the CRC16, it should send the Read Scratchpad command to verify data integrity. As preamble to the scratchpad data, the DS2432 repeats the target address TA1 and TA2 and sends the contents of the E/S register. If the PF flag is set, data did not arrive correctly in the scratchpad or there was a loss of power since data was last written to the scratchpad. The master does not need to continue reading; it can start a new trial to write data to the scratchpad. Similarly, a set AA flag together with a cleared PF flag indicates that the device did not recognize the Write command. If everything went correctly, both flags are cleared. Now the master can continue reading and verifying every data byte. After the master has verified the data, it can send the Copy Scratchpad command.

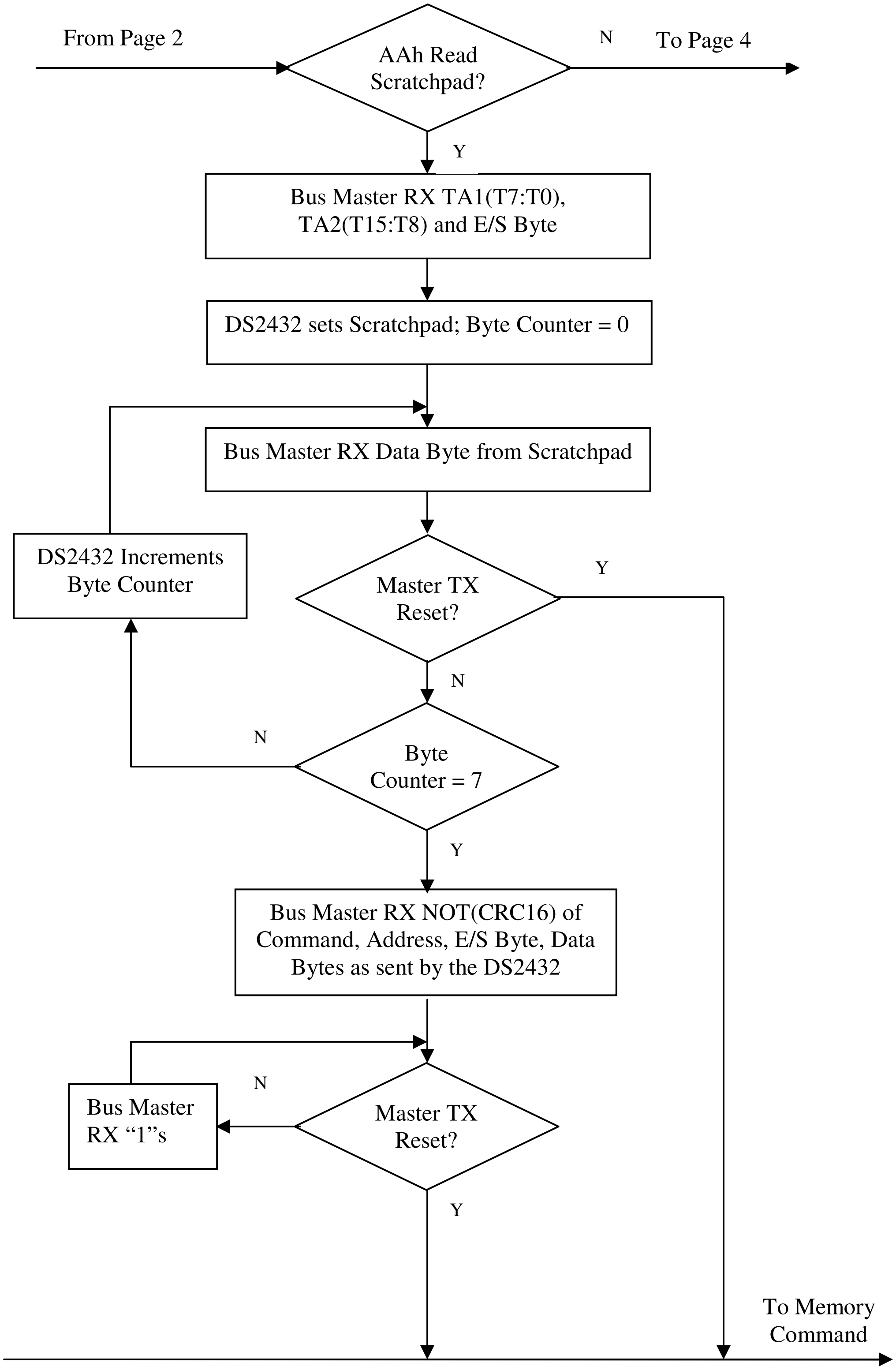
5.2.5 Memory and SHA Function Commands

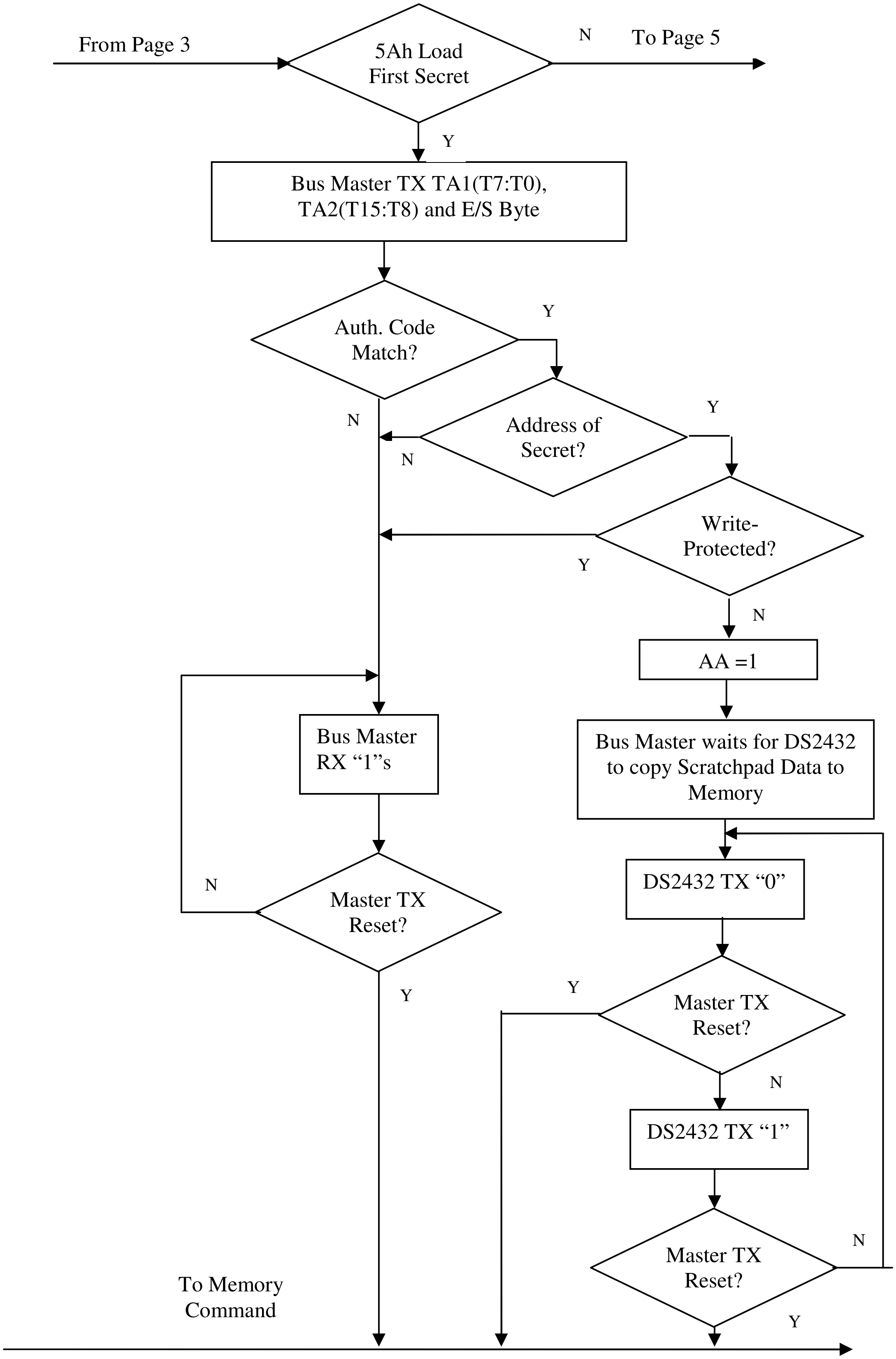
The “Memory and SHA Function Flow Chart” (Fig. 5.2) describes the protocols necessary for accessing the memory and operating the SHA engine.

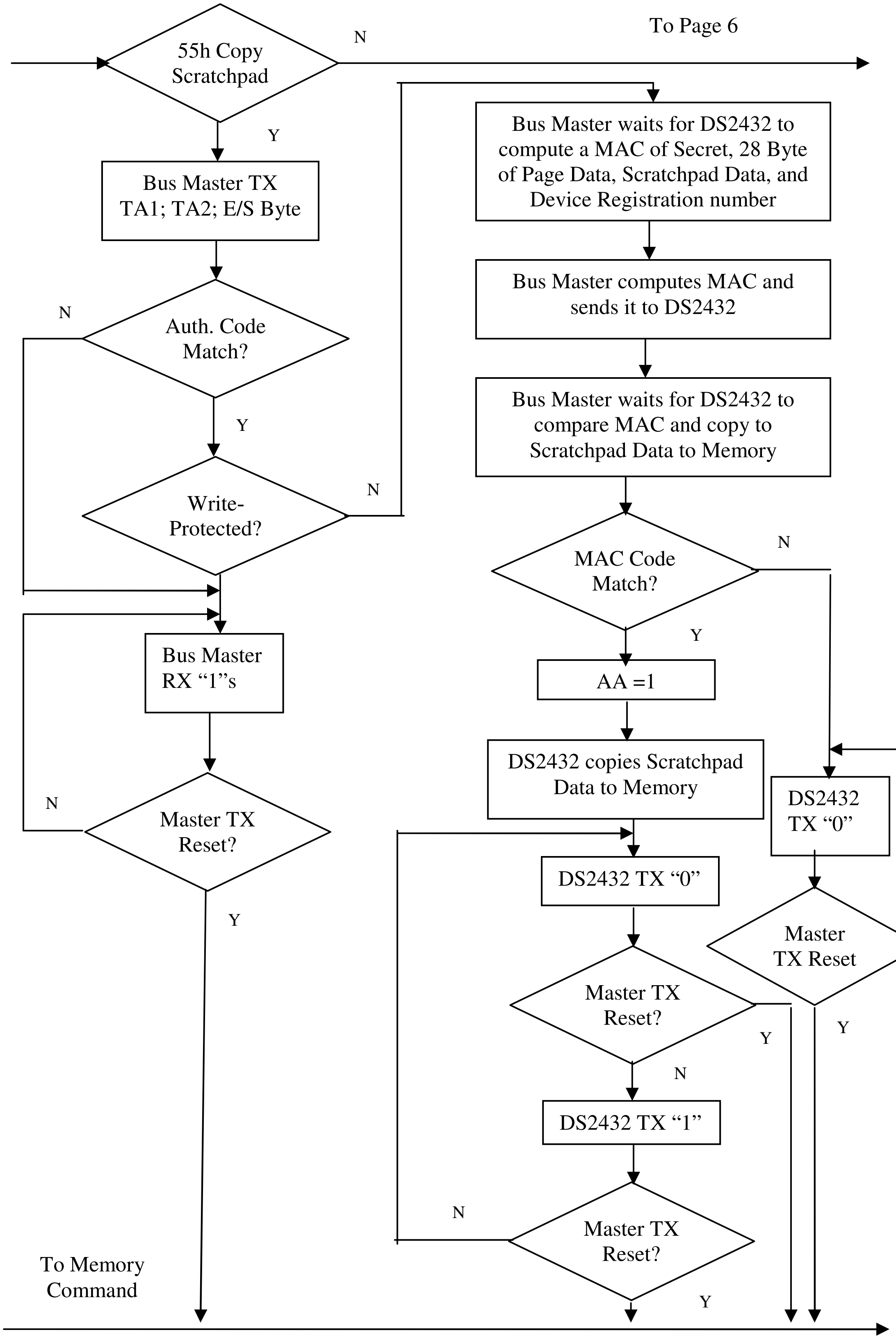
Fig 5.2 Memory and SHA Function Flow Chart

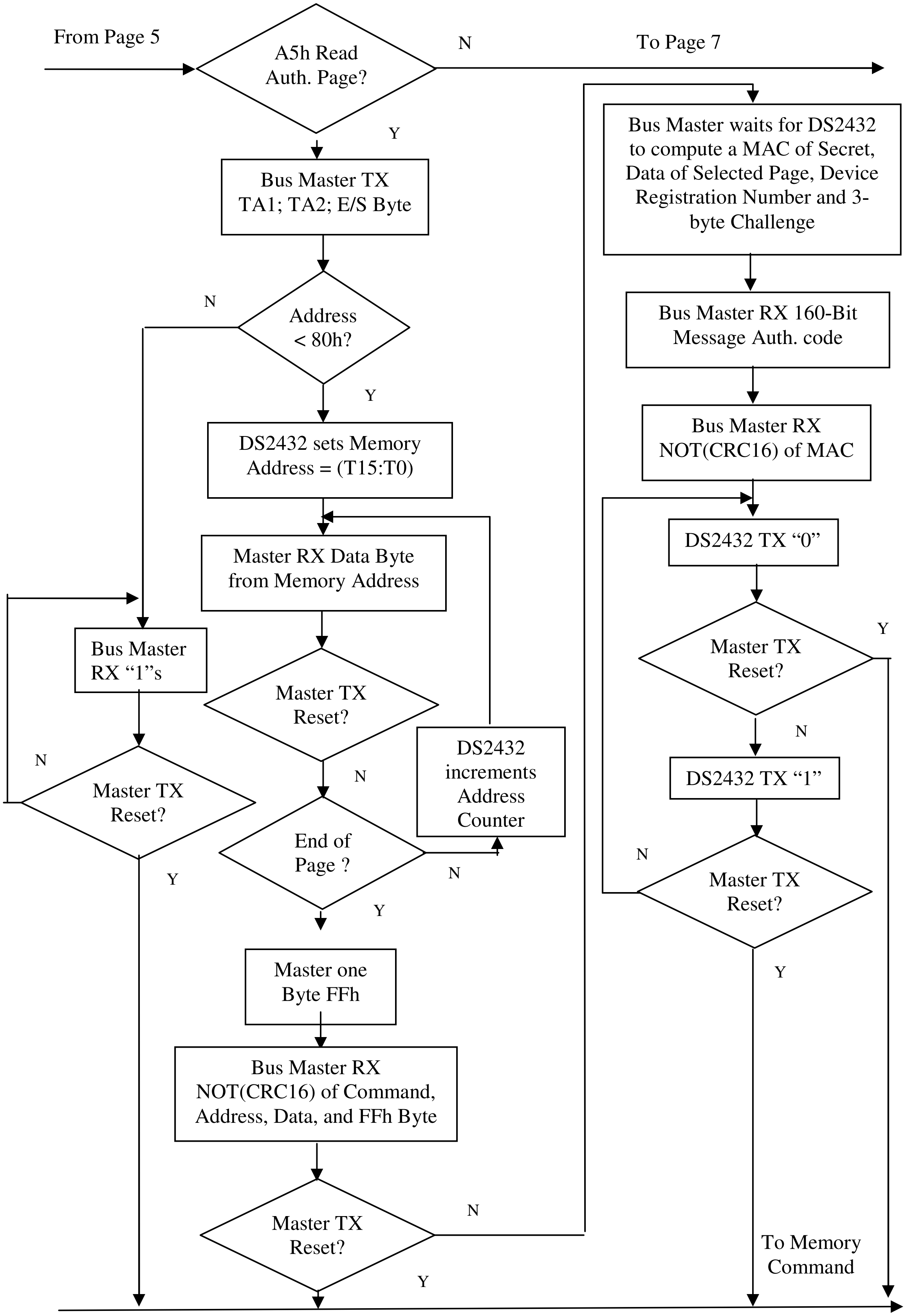


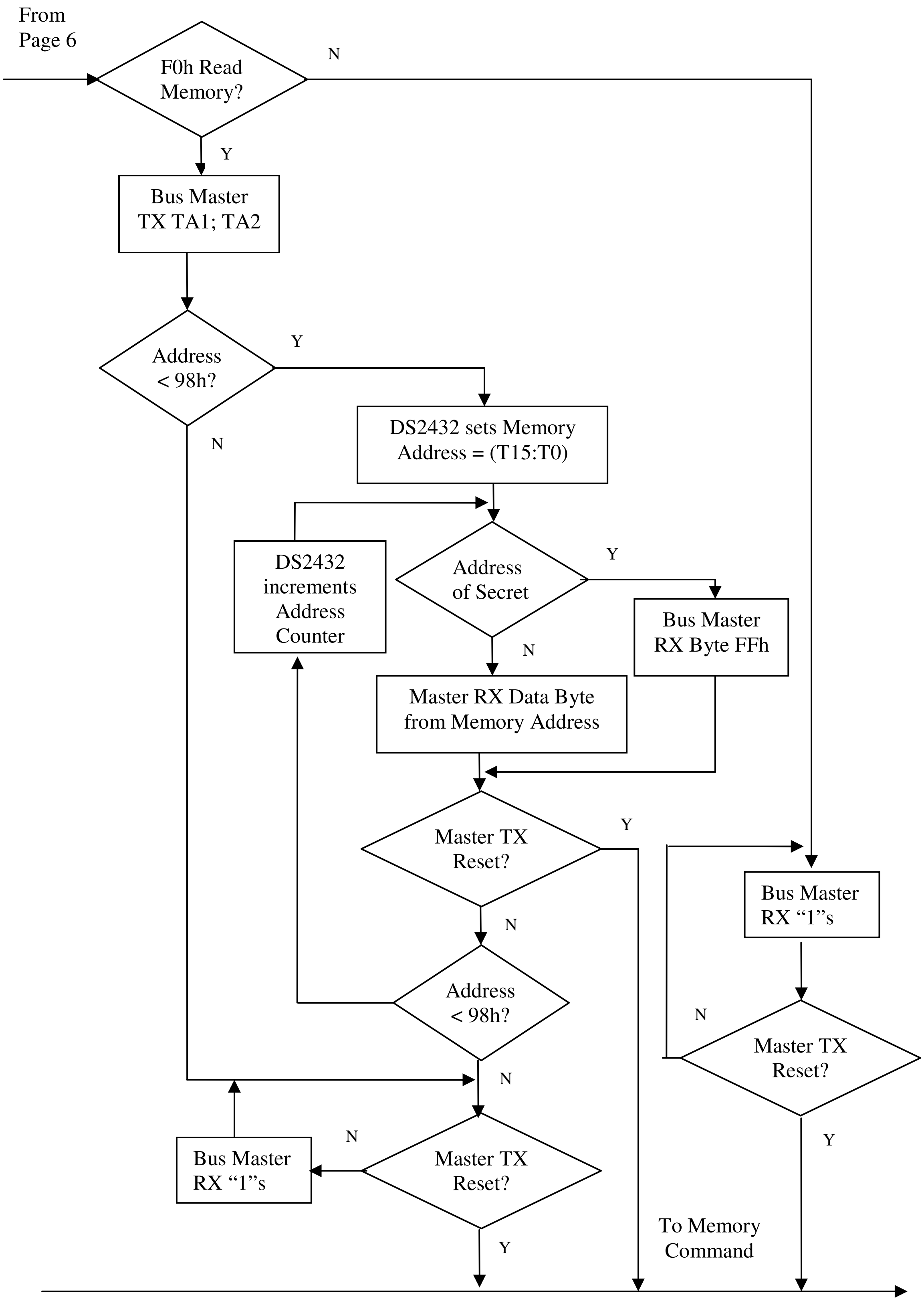

Memory Command
0Fh Write Scratchpad?
To Page 3
Bus Master TX TA1(T7:T0), TA2(T15:T8)
N
Address <90?
Y
DS2432 sets Scratchpad; Byte Counter = 0; Clears PF, AA; Sets T2: T0 = 0,0,0; Sets E2:E0 = 1,1,1
Bus Master RX “1”s
Master TX Data Byte to Scratchpad
N
Master TX Reset
Y
Master TX Reset ?
Y
DS2432 Increments Byte Counter
N
Partial Byte
N
Byte Counter = 7?
N
Y
PF = 1
Y
DS2432 TX NOT(CRC16) of Command, Address, Data Bytes as they were sent by the Bus Master
N
Bus Master RX “1”s
Master TX Reset?
Y


From Page 2
AAh Read Scratchpad?
N
To Page 4
Y
Bus Master RX TA1(T7:T0), TA2(T15:T8) and E/S Byte
DS2432 sets Scratchpad; Byte Counter = 0
Bus Master RX Data Byte from Scratchpad
DS2432 Increments Byte Counter
Master TX Reset?
Y
N
N
Byte Counter = 7
Y
Bus Master RX NOT(CRC16) of Command, Address, E/S Byte, Data Bytes as sent by the DS2432
N
Bus Master RX "1"s
Master TX Reset?
Y
To Memory Command


From Page 3
5Ah Load First Secret
N
To Page 5
Y
Bus Master TX TA1(T7:T0), TA2(T15:T8) and E/S Byte
Auth. Code Match?
Y
N
Address of Secret?
Y
N
Write-Protected?
Y
N
AA =1
Bus Master waits for DS2432 to copy Scratchpad Data to Memory
Bus Master RX "1"s
DS2432 TX "0"
N
Master TX Reset?
Y
Master TX Reset?
Y
N
DS2432 TX "1"
Master TX Reset?
N
Y
To Memory Command


55h Copy Scratchpad
N
To Page 6
Y
Bus Master TX TA1; TA2; E/S Byte
N
Auth. Code Match?
Y
Write-Protected?
N
Bus Master waits for DS2432 to compute a MAC of Secret, 28 Byte of Page Data, Scratchpad Data, and Device Registration number
Bus Master computes MAC and sends it to DS2432
Bus Master waits for DS2432 to compare MAC and copy to Scratchpad Data to Memory
MAC Code Match?
N
Y
AA =1
DS2432 copies Scratchpad Data to Memory
DS2432 TX “0”
Master TX Reset?
Y
N
DS2432 TX “1”
N
Master TX Reset?
Y
DS2432 TX “0”
N
Master TX Reset
Y
Bus Master RX “1”s
N
Master TX Reset?
Y
To Memory Command


From Page 5
A5h Read Auth. Page?
N
To Page 7
Y
Bus Master TX TA1; TA2; E/S Byte
N
Address < 80h?
Y
DS2432 sets Memory Address = (T15:T0)
Master RX Data Byte from Memory Address
Bus Master RX "1"s
Master TX Reset?
N
Y
Master TX Reset?
N
DS2432 increments Address Counter
End of Page ?
N
Y
Master one Byte FFh
Bus Master RX NOT(CRC16) of Command, Address, Data, and FFh Byte
Master TX Reset?
N
Y
Bus Master waits for DS2432 to compute a MAC of Secret, Data of Selected Page, Device Registration Number and 3-byte Challenge
Bus Master RX 160-Bit Message Auth. code
Bus Master RX NOT(CRC16) of MAC
DS2432 TX "0"
Master TX Reset?
Y
N
DS2432 TX "1"
N
Master TX Reset?
Y
To Memory Command


From
Page 6
F0h Read
Memory?
N
Y
Bus Master
TX TA1; TA2
Address
< 98h?
Y
N
DS2432 sets Memory
Address = (T15:T0)
DS2432
increments
Address
Counter
Address
of Secret
Y
N
Bus Master
RX Byte FFh
Master RX Data Byte
from Memory Address
Master TX
Reset?
Y
N
Address
< 98h?
N
Bus Master
RX "1"s
N
Master TX
Reset?
Y
Bus Master
RX "1"s
N
Master TX
Reset?
Y
To Memory
Command

5.2.6 Write Scratchpad [0Fh]

The Write Scratchpad command applies to the data memory, the secret and the writable addresses in the register page. If the bus master sends a target address higher than 90h, the command will not be executed. After issuing the write scratchpad command, the master must first provide the 2-byte target address, followed by the data to be written to the scratchpad. The data will be written to the scratchpad starting at the beginning of the scratchpad. the master should always send 8 bytes, especially if the data is to be loaded as a secret. If the master sends less than eight data bytes and does not read back the scratchpad for verification, parts of the new secret may be random data that is unknown to the master. Only full data bytes are accepted. If the last data byte is incomplete its content will be ignored and the partial byte flag PF will be set.

When executing the Write Scratchpad command the CRC generator inside the DS2432 (Figure) calculates a CRC of the entire data stream, starting at the command code and ending at the last data byte as sent by the master. This CRC is generated using the CRC16 polynomial by first clearing the CRC generator and then shifting in the command code (0FH) of the Write Scratchpad command, the Target Addresses (TA1 and TA2), and all the data bytes. The master may end the Write Scratchpad command at any time. However, if the scratchpad is filled to its capacity, the master may send 16 read time slots and will receive the CRC generated by the DS2432.

5.2.7 Read Scratchpad [AAh]

The Read Scratchpad command allows verifying the target address and the integrity of the scratchpad data. After issuing the command code, the master begins reading. The first two bytes will be the target address with T2 to T0 = 0. The next byte will be the ending offset/data status byte (E/S) followed by the scratchpad data, which may be different from what the master has originally sent. This is of particular importance if the target address is the secret, the register page or page 1 in EPROM mode. The master should read through the end of the scratchpad after which it will receive the inverted CRC. This is based on data as it was sent by the DS2432. If the master continues reading after the CRC all data will be logic 1's.

5.2.8 Load First Secret [5Ah]

The Load First Secret command is used to replace the device's current secret with the contents of the scratchpad, provided that the secret is not write-protected. This command does not require the knowledge of the device's current secret. Before the Load First Secret command can be used the master must have written the new secret to the scratchpad using the starting address of the secret (0080h). After issuing the Load First Secret command, the master must provide a 3-byte authorization pattern, which should have been obtained by an immediately preceding Read Scratchpad command. This 3-byte pattern must exactly match the data contained in the three address registers (TA1, TA2, E/S, in that order). If the pattern matches and the secret is not write-protected, the AA (Authorization Accepted) flag will be set and the copy will begin. All eight bytes of scratchpad contents will be copied to the secret's memory location. Reading AAh indicates that the copy was successful, while reading FFh indicates that the copy was not successful.

5.2.9 Copy Scratchpad [55h]

The data memory of the DS2432 can be read without any restrictions. Executing the Copy Scratchpad command to write new data to the memory or register page requires the knowledge of the device's secret and the ability to perform a SHA-1 computation to generate the 160-bit Message Authentication Code (MAC) to start the data transfer from the scratchpad to the memory. Table 5.1 Show how the various data components are entered into the SHA engine.

After issuing the Copy Scratchpad command, the master must provide a 3-byte authorization pattern, which should have been obtained by an immediately preceding Read Scratchpad command. If the authorization code matches and the target memory is not write-protected, the DS2432 will start its SHA engine to compute a 160-bit MAC that is based on the current secret, all of the scratchpad data, the first 28 bytes of the addressed memory page, and the DS2432's registration number (without the CRC). Simultaneously the master computes a MAC from the same data and sends it to the DS2432 as evidence that it is authorized to write to the EEPROM. If the MAC generated by the DS2432 matches the MAC that the master computed, the DS2432 will set its AA (Authorization Accepted) flag, and copy the entire scratchpad contents to the data EEPROM. As indication for a successful copy the master will be able to read a pattern of alternating 1's and 0's until it issues a Reset Pulse. A pattern of all zeros tells the master that the copy did not take place.

M0[31:24] = (SS + 0)	M0[23:16] = (SS + 1)	M0[15:8] =(SS + 2)	M0[7:0] = (SS + 3)
M1[31:24] = (PP + 0)	M1[23:16] = (PP + 1)	M1[15:8] = (PP + 2)	M1[7:0] = (PP + 3)
M2[31:24] = (PP + 4)	M2[23:16] = (PP + 5)	M2[15:8] = (PP + 6)	M2[7:0] = (PP + 7)
M3[31:24] = (PP + 8)	M3[23:16] = (PP + 9)	M3[15:8] = (PP + 10)	M3[7:0] = (PP + 11)
M4[31:24] = (PP + 12)	M4[23:16] = (PP + 13)	M4[15:8] = (PP + 14)	M4[7:0] = (PP + 15)
M5[31:24] = (PP + 16)	M5[23:16] = (PP + 17)	M5[15:8] = (PP + 18)	M5[7:0] = (PP + 19)
M6[31:24] = (PP + 20)	M6[23:16] = (PP + 21)	M6[15:8] = (PP + 22)	M6[7:0] = (PP + 23)
M7[31:24] = (PP + 24)	M7[23:16] = (PP + 25)	M7[15:8] = (PP + 26)	M7[7:0] = (PP + 27)
M8[31:24] = (SP + 0)	M8[23:16] = (SP + 1)	M8[15:8] = (SP + 2)	M8[7:0] = (SP + 3)
M9[31:24] = (SP + 4)	M9[23:16] = (SP + 5)	M9[15:8] = (SP + 6)	M9[7:0] = (SP + 7)
M10[31:24] = MP	M10[23:16] = (RN + 0)	M10[15:8] = (RN + 1)	M10[7:0] = (RN + 2)
M11[31:24] = (RN + 3)	M11[23:16] = (RN + 4)	M11[15:8] = (RN + 5)	M11[7:0] = (RN + 6)
M12[31:24] = (SS + 4)	M12[23:16] = (SS + 5)	M12[15:8] = (SS + 6)	M12[7:0] = (SS + 7)
M13[31:24] = FFh	M13[23:16] = FFh	M13[15:8] = FFh	M13[7:0] = 80h
M14[31:24] = 00h	M14[23:16] = 00h	M14[15:8] = 00h	M14[7:0] = 00h
M15[31:24] = 00h	M15[23:16] = 00h	M15[15:8] = 01h	M15[7:0] = B8h

Table 5.1 SHA-1 Input for Copy Scratchpad to a Data Memory Page

Mt	Input Buffer of SHA Engine; 0 <= t <= 15; 32-Bit Words
(SS + N)	Byte N of Secret; Secret Begins at Address 80h
(PP + N)	Byte N of Memory Page; Memory Pages Begin at 00h, 20h, 40h and 60h
(SP + N)	Byte N of Scratchpad
MP	MP[7:4] = 0000b, MP[3:0] = T8:T5
(RN + N)	Byte N of Registration Number
(RP + N)	Byte N of Register Page; Page Begins at 88h RP + 8 to RP + 15 is the location of the Registration Number.

Table 5.2 : Legends

Special attention is required when copying data to the register page. In order to prevent unintentional locking of a special function register or user byte it is recommended to first read the register page and then write it all with the intended modification to the scratchpad.

M0[31:24] = (SS + 0)	M0[23:16] = (SS + 1)	M0[15:8] =(SS + 2)	M0[7:0] = (SS + 3)
M1[31:24] = (SS + 0)	M1[23:16] = (SS + 1)	M1[15:8] = (SS + 2)	M1[31:24] = (SS + 0)
M2[31:24] = (SP + 4)	M2[23:16] = (SP + 5)	M2[15:8] = (SP + 6)	M2[7:0] = (SP + 7)
M3[31:24] = (RP + 0)	M3[23:16] = (RP + 1)	M3[15:8] = (RP + 2)	M3[7:0] = (RP + 3)
M4[31:24] = (RP + 4)	M4[23:16] = (RP + 5)	M4[15:8] = (RP + 6)	M4[7:0] = (RP + 7)
M5[31:24] = (RP + 8)	M5[23:16] = (RP + 9)	M5[15:8] = (RP + 10)	M5[7:0] = (RP + 11)
M6[31:24] = (RP + 12)	M6[23:16] = (RP + 13)	M6[15:8] = (RP + 14)	M6[7:0] = (RP + 15)
M7[31:24] = FFh	M7[23:16] = FFh	M7[15:8] = FFh	M7[7:0] = FFh
M8[31:24] = (SP + 0)	M8[23:16] = (SP + 1)	M8[15:8] = (SP + 2)	M8[7:0] = (SP + 3)
M9[31:24] = (SP + 4)	M9[23:16] = (SP + 5)	M9[15:8] = (SP + 6)	M9[7:0] = (SP + 7)
M10[31:24] = MP	M10[23:16] = (RN + 0)	M10[15:8] = (RN + 1)	M10[7:0] = (RN + 2)
M11[31:24] = (RN + 3)	M11[23:16] = (RN + 4)	M11[15:8] = (RN + 5)	M11[7:0] = (RN + 6)
M12[31:24] = (SS + 4)	M12[23:16] = (SS + 5)	M12[15:8] = (SS + 6)	M12[7:0] = (SS + 7)
M13[31:24] = FFh	M13[23:16] = FFh	M13[15:8] = FFh	M13[7:0] = 80h
M14[31:24] = 00h	M14[23:16] = 00h	M14[15:8] = 00h	M14[7:0] = 00h
M15[31:24] = 00h	M15[23:16] = 00h	M15[15:8] = 01h	M15[7:0] = B8h

Table 5.3 SHA-1 Input for Copy Scratchpad to the Register Page

5.2.10 Read Authenticated Page [A5h]

The Read Authenticated Page command provides the master with the data of a full or partial memory page plus a message authentication code (MAC). The MAC allows the master to determine whether the secret stored in the DS2432 is valid within the application. The DS2432 computes the MAC from its secret, all the data of the selected memory page, its registration number and a 3-byte challenge, which the master should write to the scratchpad prior to issuing the Read Authenticated Page command. The data input to the SHA engine as it applies to the Read Authenticated Page command is shown in Table 5.4. After the master has issued the command code and specified a valid target address it will receive the page data beginning at the target address through the end of the data page, one byte FFh and the inverted CRC of the command code, target address, transmitted page data and FFh byte. During this time the SHA engine of the DS2432 computes the message authentication code over the secret, all 32 data bytes of the selected page, the device's registration number (without the CRC) and the 3-byte challenge. Now the master reads the 160-bit MAC, which is followed by an inverted CRC.

M0[31:24] = (SS + 0)	M0[23:16] = (SS + 1)	M0[15:8] =(SS + 2)	M0[7:0] = (SS + 3)
M1[31:24] = (PP + 0)	M1[23:16] = (PP + 1)	M1[15:8] = (PP + 2)	M1[7:0] = (PP + 3)
M2[31:24] = (PP + 4)	M2[23:16] = (PP + 5)	M2[15:8] = (PP + 6)	M2[7:0] = (PP + 7)
M3[31:24] = (PP + 8)	M3[23:16] = (PP + 9)	M3[15:8] = (PP + 10)	M3[7:0] = (PP + 11)
M4[31:24] = (PP + 12)	M4[23:16] = (PP + 13)	M4[15:8] = (PP + 14)	M4[7:0] = (PP + 15)
M5[31:24] = (PP + 16)	M5[23:16] = (PP + 17)	M5[15:8] = (PP + 18)	M5[7:0] = (PP + 19)
M6[31:24] = (PP + 20)	M6[23:16] = (PP + 21)	M6[15:8] = (PP + 22)	M6[7:0] = (PP + 23)
M7[31:24] = (PP + 24)	M7[23:16] = (PP + 25)	M7[15:8] = (PP + 26)	M7[7:0] = (PP + 27)
M8[31:24] = (PP + 28)	M8[23:16] = (PP + 29)	M8[15:8] = (PP + 30)	M8[7:0] = (PP + 31)
M9[31:24] = FFh	M9[23:16] = FFh	M9[15:8] = FFh	M9[7:0] = FFh
M10[31:24] = MP	M10[23:16] = (RN + 0)	M10[15:8] = (RN + 1)	M10[7:0] = (RN + 2)
M11[31:24] = (RN + 3)	M11[23:16] = (RN + 4)	M11[15:8] = (RN + 5)	M11[7:0] = (RN + 6)
M12[31:24] = (SS + 4)	M12[23:16] = (SS + 5)	M12[15:8] = (SS + 6)	M12[7:0] = (SS + 7)
M13[31:24] = (SP + 4)	M13[23:16] = (SP + 5)	M13[15:8] = (SP + 6)	M13[7:0] = 80h
M14[31:24] = 00h	M14[23:16] = 00h	M14[15:8] = 00h	M14[7:0] = 00h
M15[31:24] = 00h	M15[23:16] = 00h	M15[15:8] = 01h	M15[7:0] = B8h

Table 5.4 SHA-1 Input for Read Authenticated Page

5.2.11 Read Memory [F0h]

The read memory command may be used to read all memory except for the secret. Attempting to read the secret will not reveal any data. After issuing the command, the master must provide the 2-byte target address. After these two bytes, the master reads data beginning from the target address and may continue until address 0097h. If the master continues reading the result will be logic 1's.

Chapter 6

Source Code

6.1 3 to 8 decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity demo_14_11 is
 port (     C100MHz : in std_logic;
                                RSTIN   : in std_logic;
                                IFF_OUT : out std_logic;
                                dec_in_N  : in std_logic_vector(7 downto 0);
                                dig_out_N : out std_logic_vector(2 downto 0);
                                BI      : inout  std_logic
                        );
end demo_14_11;

architecture Behavioral of demo_14_11 is

component IFF_TEST is
        Port (
                        C100MHz : in  STD_LOGIC;
       RSTIN : in  STD_LOGIC;
                        FRND_OUT : out std_logic;
                        BI : inout STD_LOGIC);
end component;

signal IFF : std_logic;
signal dig_out : std_logic_vector(2 downto 0);
signal dec_in : std_logic_vector(7 downto 0);

begin

IFF_OUT <= IFF;
X1: IFF_TEST port map(C100MHz,RSTIN,IFF,BI);
dig_out_N <= not dig_out;
dec_in <= not dec_in_N;

process(dec_in, C100MHz,IFF)
```

```
begin

                if( IFF = '0') then
                                case dec_in is
                                when "00000001" => dig_out <= "000";
                                when "00000010" => dig_out <= "001";
                                when "00000100" => dig_out <= "010";
                                when "00001000" => dig_out <= "011";
                                when "00010000" => dig_out <= "100";
                                when "00100000" => dig_out <= "101";
                                when "01000000" => dig_out <= "110";
                                when "10000000" => dig_out <= "111";
                                when others=>NULL;
                                end case;
                else
                                dig_out <= "111";
                end if;
        end process;

end Behavioral;
```

6.2 IFFTEST

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity IFF_TEST is
        Port (
                        C100MHz : in  STD_LOGIC;
       RSTIN : in  STD_LOGIC;
                        FRND_OUT : out std_logic;
                        BI : inout STD_LOGIC);
end IFF_TEST;

architecture Behavioral of IFF_TEST is

type q is array (0 to 11) of std_logic_vector(7 downto 0);
type rr is array (0 to 3) of std_logic_vector(7 downto 0);
--type s is array (0 to 11) of p;

-- (1byte command + 2byte address + 8byte secret + 1extra byte )
constant secret  : q := (
X"0F",X"80",X"00",X"01",X"02",X"03",X"04",X"05",X"06",X"07",X"08",X"FF");
```

```
-- (1byte command +  2byte address + 8byte secret)

constant page_01 : q := (
X"0F",X"00",X"00",X"11",X"12",X"13",X"14",X"15",X"16",X"17",X"18",X"FF");
constant page_02 : q := (
X"0F",X"08",X"00",X"21",X"22",X"23",X"24",X"25",X"26",X"27",X"28",X"FF");
constant page_03 : q := (
X"0F",X"10",X"00",X"31",X"32",X"33",X"34",X"35",X"36",X"37",X"38",X"FF");
constant page_04 : q := (
X"0F",X"18",X"00",X"41",X"42",X"43",X"44",X"45",X"46",X"47",X"48",X"FF");

constant RDAUTH : rr := (X"A5",X"00",X"00",X"FF");

type stat_typ is
(initial,skiprom,RXCRC,RDAUTH_cmd,RDMAC,memcmd,wrscrpad,done);--
,rdscrpad,ldfkey,sendmac,progwait,macwait,rdmem,cpyscrpad);
signal pr_stat, nx_stat : stat_typ ;
signal A,B,C,D,E : std_logic_vector(31 downto 0);
signal sflag : integer range 0 to 6 ;--std_logic_vector(2 downto 0);
signal c1, cnt,strtcnt, rdcnt,RDDATACNT : integer range 0 to 31;
signal waitcnt : integer;
signal j,mscnt : integer range 0 to 30;
signal CRCOK,cnten, RDDATACNT_EN,mscnten : std_logic;
signal CRCdata : std_logic_vector(87 downto 0);

signal CRCDATAld : std_logic;
signal CCRC : std_logic_vector(15 downto 0);
--***********
type state_type is (IDEAL, RSTPLS, PRS, WRITE_0, WRITE_1, READ);
        signal pr_state, nx_state : state_type;
        signal timer : std_logic_vector(9 downto 0);--integer range 0 to 1000;
        signal clkcnt,rxcnt : integer range 0 to 62;
        signal C1MHz,prsnt, rs,X, rxd, loaddone,r : std_logic;
        signal txsiftr : std_logic_vector(7 downto 0);
        signal rxdata : std_logic_vector(31 downto 0);
        signal flag, wr, rd : std_logic;

--**************
        signal rden, wren, start : std_logic;
        signal dbcnt : integer range 0 to 500005;

        signal txdata : std_logic_vector(7 downto 0);
        signal start1,FRND : std_logic;
        signal SRSTIN : std_logic_vector(10 downto 0);
        signal rxdata_out :  std_logic_vector(31 downto 0);
        signal crcok_out,start_out,rstout,digest_vld_out,prsnt_out:  std_logic;


component CRC is
   Port ( clk, rst : in  STD_LOGIC;
                        CRCDATAld : in std_logic;
                        datain : in std_logic_vector(87 downto 0);
                        CRC_out : out std_logic_vector(15 downto 0));
end component;

component MAC is
        port(
                clk : in std_logic;
                reset : in std_logic;
                m0i,m1i,m2i,m3i,m4i,m5i,m6i,m7i,m8i,m13i,m10i,m11i,m12i : in
std_logic_vector(31 downto 0);
                message_vld : in std_logic;
                digest : out std_logic_vector(159 downto 0);
                digest_vld : out std_logic        );
end component;

 signal m0,m1,m2,m3,m4,m5,m6,m7,m8,m13,m10,m11,m12 : std_logic_vector(31
downto 0);
 signal message_vld, digest_vld, MACRST : std_logic;
 signal digest,RXMAC : std_logic_vector(159 downto 0);
 signal ldtsten : std_logic;
begin

rstout <= rst ;
start_out <= start;
prsnt_out <= prsnt;

crcok_out <= not CRCOK;
digest_vld_out <= not digest_vld;
FRND_OUT <= not FRND;

inst_mac: MAC port map

        (C1MHz,MACRST,m0,m1,m2,m3,m4,m5,m6,m7,m8,m13,m10,m11,m12,messag
e_vld,digest,digest_vld);

inst_CRC : CRC port map
                                (clk => C1MHz,
                                 rst => rst,
                                        CRCDATAld => CRCDATAld,
                                        datain => CRCdata,
                                        CRC_out => CCRC);



process(C100MHz)
begin
if(C100MHz'event and C100MHz = '1')then
                SRSTIN <= RSTIN & SRSTIN(10 downto 1);
        if(SRSTIN(10) = '1' and SRSTIN(0) = '0') then
                        RST <= '0';
         else
                        RST <= '1';
         end if;
end if;
end process;


--*************************

-- generate 1MHz clock from 50MHz system clock
process(RST,C100MHz)
        begin
                if(RST = '0') then
                        clkcnt <= 0;
                        C1MHz <= '0';
                elsif(C100MHz'event and C100MHz = '1') then
                        if(clkcnt = 49) then
                                C1MHz <= not C1MHz;
                                clkcnt <= 0;
                        else
                                clkcnt <= clkcnt + 1;
                        end if;
                end if;
end process;

--*******************************************


process(rst,C1MHz,rden)
begin
        if(rst = '0' ) then
                rxcnt <= 0;
                r <= '0';
        elsif(rden = '0') then
```

```
                r <= '1';
                rxcnt <= 0;
                rd <= '0';
        elsif(C1MHz'event and C1MHz = '1') then
                if(rd = '0' and timer = 25) then
                        rxcnt <= rxcnt + 1;
                end if;
                if (rxcnt > 23 and rd = '0' and RDDATACNT = 8) then
                        rd <= '1';
                elsif (rxcnt <= 31 and rd = '0') then
                        rd <= '0';
                else
                        rd <= '1';
                end if;
        end if;
end process;

 rxdata_out <=  rxdata and A and B and C and D and E;-- and digest(159 downto 128)
and digest(127 downto 96) and digest(95 downto 64) and digest(63 downto 32) and
digest(31 downto 0);--

--
        wr <= NOT(flag or txsiftr(7) or txsiftr(6) or txsiftr(5) or txsiftr(4) or txsiftr(3)
                                or txsiftr(2) or txsiftr(1));

process(RST,C1MHz)
begin
        if(RST = '0') then
                rxdata <= (others => '0');
                flag <= '0';
                txsiftr <= (others => '0');
        elsif(C1MHz'event and C1MHz = '1') then
                if(rd = '0' and timer = 25) then
                        rxdata <= RXd & rxdata(31 downto 1);
                end if;

                if(wren = '0') then
                        flag <= '1';
                        txsiftr <= txdata;
                elsif(wr = '0' and timer = 10) then
                        txsiftr <= flag & txsiftr(7 downto 1) ;
                        flag <= '0';
                end if;

        end if;
end process;
```

```
--***********************************************
-- process
process(rst,C1MHz)
begin
    if(rst = '0') then
        pr_state <= IDEAL;
    elsif(C1MHz'event and C1MHz = '1') then
        pr_state <= nx_state;
    end if;
end process;


process(pr_state,timer,start,rs,rd,txsiftr,wr)
begin
    case pr_state is
        when IDEAL =>
            if start = '0' then
                nx_state <= RSTPLS;
            elsif rs = '1' then
                nx_state <= PRS;
            elsif rd = '0' then
                nx_state <= READ;
            elsif(txsiftr(0) = '1' and wr = '0')then
                    nx_state <= WRITE_1;
            elsif(txsiftr(0) = '0' and wr = '0')then
                    nx_state <= WRITE_0;
            else
                 nx_state <= IDEAL;
            end if;


        when RSTPLS =>
            if timer < 540 then     -- rst pulse is  480 < Trst > 640
                nx_state <= RSTPLS;
            else
                nx_state <= IDEAL;
            end if;

        when PRS =>
            if(timer < 100)then     -- should be > [15us(Tpdh) + 60us (Tpdl) +
5us(Trec)]
                nx_state <= PRS;
            else
                nx_state <= IDEAL;
            end if;
        when WRITE_0 =>                         --   should be  > Tslot(65us)
            if(timer <= 75) then
                nx_state <= WRITE_0 ;
            else
                nx_state <= IDEAL;
            end if;

        when WRITE_1 =>                              --   should be  > Tslot(65us)
            if(timer <= 75) then
                nx_state <= WRITE_1;
            else
                nx_state <= IDEAL;
            end if;

        when READ =>                                          --  should be  >
Tslot(65us)
            if(timer <= 75 ) then
                nx_state <= READ ;
            else
                nx_state <= IDEAL;
            end if;

    end case;
end process;


process(C1MHz,RST)
begin
    if(RST = '0') then
        timer <= (others => '0');
        BI <= 'Z';
        rs <= '0';
        rxd <= '0';
        prsnt <= '1';
    elsif(C1MHz'event and C1MHz = '1') then

        case pr_state is

            when IDEAL =>
                BI <= 'Z';
                timer <= (others => '0');
                prsnt <= '1';

            when RSTPLS =>
                BI <= '0';
                rs <= '1';      -- to indicate that next state is
                timer <= timer + 1;

            when PRS =>
                if (timer >= 65 and timer < 70) then
                    prsnt <= BI;
                end if;
                rs <= '0';
                timer <= timer + 1;

            when WRITE_0 =>
                if (timer < 65) then   --   Twol should be  ( >60us and <
120us)
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                timer <= timer + 1;

            when WRITE_1 =>
            -- Tw1l should be  ( > 1us and < 15us ) -- should not exceed
Trl(5us)
                if timer < 2 then
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                timer <= timer + 1;

            when READ =>
                if (timer <= 5) then -- Trl (> 5us and < 15us) *should use
min
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                if(timer > 8 and timer < 12)then --Tmsr ( > Trl and < 15us)
                    rxd <= BI;
                end if;
                timer <= timer + 1;
        end case;
    end if;
end process;

--*************************************************************
```

```
process(C1MHz,rst)
 begin
        if rst = '0' then
                pr_stat <= initial;
        elsif(C1MHz'event and C1MHz = '1') then
                pr_stat <= nx_stat;
        end if;
end process;


process(pr_stat,prsnt,cnt,wr,c1,j,rd,rdcnt,waitcnt,RDDATACNT,pr_state,CRCOK)
begin
        case pr_stat is

                when initial =>
                        if prsnt = '0' then
                                nx_stat <= skiprom;
                        else
                                nx_stat <= initial;
                        end if;

                when skiprom =>
                        if(cnt > 5 and wr = '1') then
                                nx_stat <= memcmd;
                        else
                                nx_stat <= skiprom;
                        end if;

                when memcmd =>
                        if(c1 = 1)then -- or C1=4 or c1=8 or c1=12 or c1=16)then
                                nx_stat <= wrscrpad; -- write
                        elsif(c1 = 2)then --or c1=5 or c1=9 or c1=13 or c1=17) then
                                nx_stat <= RDAUTH_cmd;  --rdscrpad;        -- read
                        elsif(c1 = 3) then
                                nx_stat <= RDMAC; --ldfkey;

                        else
                                nx_stat <= memcmd;
                        end if;

                when wrscrpad =>
                        if (j > 11 and wr = '1') then   -- if CRC\ = calculated CRC
                                nx_stat <= RXCRC;
                        else
                                nx_stat <= wrscrpad;
                        end if;
                -- if rxdata = X"AAAAAAAA" at ldcnt = 3 then keydn '1'
                when RXCRC =>
                        if(CRCOK = '1' and pr_state = ideal)then
                                nx_stat <= initial;
                        else
                                nx_stat <= RXCRC;
                        end if;

                when RDAUTH_cmd =>
                        if(RDDATACNT > 8 and pr_state = ideal)then
                                nx_stat <= memcmd; --MACWAIT;
                        else
                                nx_stat <= RDAUTH_cmd;
                        end if;


                when RDMAC =>
                        if(RDDATACNT > 6) then
                                nx_stat <= done;
                        else
                                nx_stat <= RDMAC;
                        end if;
                when done =>
                  nx_stat <= done;


        end case;
end process;

process(wr,rst)
begin
        if rst = '0' then
                j <= 1;
        elsif(wr'event and wr = '1') then
                if(cnten = '1') then
                        j <= j + 1;
                else
                        j <= 1;
                end if;
end process;

process(rst,RDDATACNT_EN,rd)
begin
        if(rst = '0' or RDDATACNT_EN = '0') then
                RDDATACNT <= 0;
```

```
            elsif(rd'event and rd = '1') then
                        RDDATACNT <= RDDATACNT + 1;
            end if;
end process;

process(C1MHz,RST)
begin
 if(RST = '0') then
                m0 <= secret(3) & secret(4) & secret(5) & secret(6);
                m12 <= secret(7) & secret(8) & secret(9) & secret(10);
                m1 <= (others => '0');
                m2 <= (others => '0');
                m3 <= (others => '0');
                m4 <= (others => '0');
                m5 <= (others => '0');
                m6 <= (others => '0');
                m7 <= (others => '0');
                m8 <= (others => '0');
                m13 <= (others => '0');
                m10 <= (others => '0');
                m11 <= (others => '0');
                A <= (others => '0') ;
                b <= (others => '0') ;
                c <= (others => '0') ;
                d <= (others => '0') ;
                e <= (others => '0') ;
                message_vld <= '0';
                MACRST <= '1';
 elsif(C1MHz'event and C1MHz = '1') then
                MACRST <= '0';
                        m1 <=  page_01(3) & page_01(4) & page_01(5) & page_01(6);

                        m2 <=  page_01(7) & page_01(8) & page_01(9) & page_01(10);

                        m3 <=  page_02(3) & page_02(4) & page_02(5) & page_02(6);
                        m4 <=  page_02(7) & page_02(8) & page_02(9) & page_02(10);
                        m5 <=  page_03(3) & page_03(4) & page_03(5) & page_03(6);
                        m6 <=  page_03(7) & page_03(8) & page_03(9) & page_03(10);
                        m7 <= page_04(3) & page_04(4) & page_04(5) & page_04(6);
                        m8 <= page_04(7) & page_04(8) & page_04(9) & page_04(10);
                        m10 <= X"40" & X"33" & X"6D" & X"1F";
                        m11 <= X"25" & X"01" & X"00" & X"00";
                        m13 <= page_01(7) & page_01(8) & page_01(9) & X"80";

                if(digest_vld <= '1') then
                        A <= digest(159 downto 128);
                        B <= digest(127 downto 96);
                        C <= digest(95 downto 64);
                        D <= digest(63 downto 32);
                        E <= digest(31 downto 0);
                end if;

                if(pr_stat = wrscrpad) then
                        message_vld <= '1';
                end if;

 end if;

 end process;

end Behavioral;
```

6.3 LOADTEST

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity single_mod is
        Port (
                         C100MHz : in  STD_LOGIC;
       RSTIN : in  STD_LOGIC;
                         push3 : in std_logic;
                         rxdata_out : out std_logic_vector(31 downto 0);
                         lddone,crcok_out,start_out,rstout,keydn_out,
                         digest_vld_out,prsnt_out : out std_logic;
                         BI : inout  STD_LOGIC);

end single_mod;

architecture Behavioral of single_mod is

--************
type q is array (0 to 11) of std_logic_vector(7 downto 0);
type rr is array (0 to 3) of std_logic_vector(7 downto 0);
--type s is array (0 to 11) of p;

-- (1byte command + 2byte address + 8byte secret + 1extra byte )
```

```
constant secret  : q := (
X"0F",X"80",X"00",X"01",X"02",X"03",X"04",X"05",X"06",X"07",X"08",X"FF");

-- (1byte command +  2byte address + 8byte secret)

constant page_01 : q := (
X"0F",X"00",X"00",X"11",X"12",X"13",X"14",X"15",X"16",X"17",X"18",X"FF");
constant page_02 : q := (
X"0F",X"08",X"00",X"21",X"22",X"23",X"24",X"25",X"26",X"27",X"28",X"FF");
constant page_03 : q := (
X"0F",X"10",X"00",X"31",X"32",X"33",X"34",X"35",X"36",X"37",X"38",X"FF");
constant page_04 : q := (
X"0F",X"18",X"00",X"41",X"42",X"43",X"44",X"45",X"46",X"47",X"48",X"FF");

constant rdmemadd : rr := (X"F0",X"00",X"00",X"FF");

type stat_typ is
(initial,skiprom,RXCRC,memcmd,wrscrpad,rdscrpad,ldfkey,sendmac,progwait,macwait,r
dmem,cpyscrpad);
signal pr_stat, nx_stat : stat_typ ;
signal A,B,C,D,E : std_logic_vector(31 downto 0);
signal sflag : integer range 0 to 6 ;--std_logic_vector(2 downto 0);
signal c1, cnt,strtcnt, rdcnt,rdmemcnt,i : integer range 0 to 62;
signal waitcnt : integer;
signal j,mscnt : integer range 0 to 30;
signal CRCOK,cnten, rdmemcnten,mscnten : std_logic;
signal CRCdata : std_logic_vector(87 downto 0);

signal CRCDATAld : std_logic;
signal CCRC : std_logic_vector(15 downto 0);
--***********
type state_type is (IDEAL, RSTPLS, PRS, WRITE_0, WRITE_1, READ);
        signal pr_state, nx_state : state_type;
        signal timer : std_logic_vector(9 downto 0);--integer range 0 to 1000;
        signal clkcnt,rxcnt : integer range 0 to 100;
        signal C1MHz,prsnt, rs,X, rxd, loaddone,r : std_logic;
        signal txsiftr : std_logic_vector(7 downto 0);
        signal rxdata : std_logic_vector(31 downto 0);
        signal flag, wr, rd : std_logic;

--**************
        signal rden, wren, start : std_logic;
        signal Lpush2, Dpush2,pusx : std_logic;
  signal Lpush3, Dpush3,RST  : std_logic;
        signal dbcnt : integer range 0 to 500005;
        signal txdata : std_logic_vector(7 downto 0);
        signal keydn,start1 : std_logic;
        signal SRSTIN : std_logic_vector(10 downto 0);

--***************
component CRC is
   Port ( clk, rst : in  STD_LOGIC;
                              CRCDATAld : in std_logic;
                              datain : in std_logic_vector(87 downto 0);
                              CRC_out : out std_logic_vector(15 downto 0));
end component;

component sha_256 is
        port(
                clk : in std_logic;
                reset : in std_logic;
                m0i,m1i,m2i,m3i,m4i,m5i,m6i,m7i,m8i,m9i,m10i,m11i,m12i : in
std_logic_vector(31 downto 0);
                message_vld : in std_logic;
                digest : out std_logic_vector(159 downto 0);
                digest_vld : out std_logic        );
end component;

 signal m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12 : std_logic_vector(31
downto 0);
 signal message_vld, digest_vld, MACRST : std_logic;
 signal digest : std_logic_vector(159 downto 0);
 signal ldtsten : std_logic;
begin

rstout <= rst;
start_out <= start;
prsnt_out <= prsnt;
lddone <= not loaddone;
crcok_out <= not CRCOK;
digest_vld_out <= not digest_vld;
keydn_out <= not keydn;

inst_mac: sha_256 port map

        (C1MHz,MACRST,m0,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,message
_vld,digest,digest_vld);

inst_CRC : CRC port map
                                        (clk => C1MHz,
                                         rst => rst,
                                         CRCDATAld => CRCDATAld,
                                        datain => CRCdata,
                                        CRC_out => CCRC);



process(C100MHz)
begin
if(C100MHz'event and C100MHz = '1')then
                SRSTIN <= RSTIN & SRSTIN(10 downto 1);
        if(SRSTIN(10) = '1' and SRSTIN(0) = '0') then
                        RST <= '0';
         else
                        RST <= '1';
         end if;
end if;
end process;


--*************************
process(C1MHZ, RST)
begin
        if RST = '0' then
                start <= '1';
                ldtsten <= '1';
        elsif(C1MHZ'event and C1MHZ = '1') then
                Lpush3 <= push3;
                Dpush3 <= Lpush3;


         if((Lpush3 = '1' and Dpush3 = '0')  and (dbcnt < 2 or dbcnt > 500000)) then
                ldtsten <= '0';
         else
                        ldtsten <= '1';
         end if;

         if(ldtsten = '0' or start1 = '0') then
                        start <= '0';
         else
                        start <= '1';
         end if;

        end if;
        end process;

--Counter satrts counting after the chipselect switch is pressed and released
--Switch press is '0',release is '1'
```

```
process(C1MHZ,ldtsten,RST)
begin

        if(RST = '0') then
                dbcnt <= 0;
                X <= '0';
        elsif(ldtsten = '0') then
                dbcnt <= 0;
                X <= '1';
        elsif(C1MHZ'event and C1MHZ = '1') then
                if( ldtsten = '1' and X = '1')then
                if(dbcnt <= 500002) then
                        dbcnt <= dbcnt + 1;
                end if;
                end if;
        end if;
end process;
--*********************************************

-- generate 1MHz clock from 50MHz system clock
process(RST,C100MHz)
        begin
                if(RST = '0') then
                        clkcnt <= 0;
                        C1MHz <= '0';
                elsif(C100MHz'event and C100MHz = '1') then
                        if(clkcnt = 49) then
                                C1MHz <= not C1MHz;
                                clkcnt <= 0;
                        else
                                clkcnt <= clkcnt + 1;
                        end if;
                end if;
end process;

--*******************************************


process(rst,C1MHz,rden)
begin
        if(rst = '0' ) then
                rxcnt <= 0;
                r <= '0';
        elsif(rden = '0') then
                r <= '1';
                rxcnt <= 0;
                rd <= '0';
        elsif(C1MHz'event and C1MHz = '1') then
                if(rd = '0' and timer = 25) then
                        rxcnt <= rxcnt + 1;
                end if;
                if (rxcnt <= 31 and rd = '0') then
                        rd <= '0';
                else
                        rd <= '1';
                end if;
        end if;
end process;

 rxdata_out <= rxdata and A and B and C and D and E;

--
        wr <= NOT(flag or txsiftr(7) or txsiftr(6) or txsiftr(5) or txsiftr(4) or txsiftr(3)
                                or txsiftr(2) or txsiftr(1));

process(RST,C1MHz)
begin
        if(RST = '0') then
                rxdata <= (others => '0');
                flag <= '0';
                txsiftr <= (others => '0');
        elsif(C1MHz'event and C1MHz = '1') then
                if(rd = '0' and timer = 25) then
                        rxdata <= RXd & rxdata(31 downto 1);
                end if;

                if(wren = '0') then
                        flag <= '1';
                        txsiftr <= txdata;
                elsif(wr = '0' and timer = 10) then
                        txsiftr <= flag & txsiftr(7 downto 1) ;
                        flag <= '0';
                end if;

        end if;
end process;

--**************************************************
-- process
process(rst,C1MHz)
begin
```

```
    if(rst = '0') then
        pr_state <= IDEAL;
    elsif(C1MHz'event and C1MHz = '1') then
        pr_state <= nx_state;
    end if;
end process;


process(pr_state,timer,start,rs,rd,txsiftr,wr)
begin
    case pr_state is
        when IDEAL =>
            if start = '0' then
                nx_state <= RSTPLS;
            elsif rs = '1' then
                nx_state <= PRS;
            elsif rd = '0' then
                nx_state <= READ;
            elsif(txsiftr(0) = '1' and wr = '0')then
                    nx_state <= WRITE_1;
            elsif(txsiftr(0) = '0' and wr = '0')then
                    nx_state <= WRITE_0;
            else
                 nx_state <= IDEAL;
            end if;


        when RSTPLS =>
            if timer < 540 then      -- rst pulse is  480 < Trst > 640
                nx_state <= RSTPLS;
            else
                nx_state <= IDEAL;
            end if;

        when PRS =>
            if(timer < 100)then      -- should be > [15us(Tpdh) + 60us (Tpdl) +
5us(Trec)]
                nx_state <= PRS;
            else
                nx_state <= IDEAL;
            end if;

        when WRITE_0 =>                    --   should be  > Tslot(65us)
            if(timer <= 75) then
                nx_state <= WRITE_0 ;
            else
```

```
                nx_state <= IDEAL;
            end if;

        when WRITE_1 =>                         --   should be  > Tslot(65us)
            if(timer <= 75) then
                nx_state <= WRITE_1;
            else
                nx_state <= IDEAL;
            end if;

        when READ =>                                      --  should be  >
Tslot(65us)
            if(timer <= 75 ) then
                nx_state <= READ ;
            else
                nx_state <= IDEAL;
            end if;

    end case;
end process;


process(C1MHz,RST)
begin
    if(RST = '0') then
        timer <= (others => '0');
        BI <= 'Z';
        rs <= '0';
        rxd <= '0';
        prsnt <= '1';
    elsif(C1MHz'event and C1MHz = '1') then

        case pr_state is

            when IDEAL =>
                BI <= 'Z';
                timer <= (others => '0');
                prsnt <= '1';

            when RSTPLS =>
                BI <= '0';
                rs <= '1';     -- to indicate that next state is
                timer <= timer + 1;

            when PRS =>
                if (timer >= 65 and timer < 70) then
```

```
                    prsnt <= BI;
                end if;
                rs <= '0';
                timer <= timer + 1;

            when WRITE_0 =>
                if (timer < 65) then   --   Twol should be  ( >60us and <
120us)
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                timer <= timer + 1;

            when WRITE_1 =>
            -- Tw1l should be  ( > 1us and < 15us ) -- should not exceed
Trl(5us)
                if timer < 2 then
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                timer <= timer + 1;

            when READ =>
                if (timer <= 5) then -- Trl (> 5us and < 15us) *should use
min
                    BI <= '0';
                else
                    BI <= 'Z';
                end if;
                if(timer > 8 and timer < 12)then --Tmsr ( > Trl and < 15us)
                    rxd <= BI;
                end if;
                timer <= timer + 1;

--          others =>
--              BI <= 'Z';
--              timer <= 0;
        end case;
    end if;
end process;

--**************************************************************

process(C1MHz,rst)
```

```
 begin
        if rst = '0' then
                pr_stat <= initial;
        elsif(C1MHz'event and C1MHz = '1') then
                pr_stat <= nx_stat;
        end if;
end process;


process(pr_stat,prsnt,cnt,wr,c1,j,keydn,digest_vld,rd,rdcnt,waitcnt,rdmemcnt,pr_state,CR
COK,mscnt,pusx )
begin
        case pr_stat is

                when initial =>
                        if prsnt = '0' then
                                nx_stat <= skiprom;
                        else
                                nx_stat <= initial;
                        end if;

                when skiprom =>
                        if(cnt > 5 and wr = '1') then
                                nx_stat <= memcmd;
                        else
                                nx_stat <= skiprom;
                        end if;

                when memcmd =>
--                      if(i = 10) then
                        if(c1=0 or C1=4 or c1=8 or c1=12 or c1=16)then
                                nx_stat <= wrscrpad; -- write secret
                        elsif(c1=1 or c1=5 or c1=9 or c1=13 or c1=17) then
                                nx_stat <= rdscrpad;   -- read
                        elsif(c1 = 2) then
                                nx_stat <= ldfkey;
                        elsif(c1 = 3 or c1=7 or c1=11 or c1=15) then
                                nx_stat <= rdmem;
                        elsif(c1=6 or c1=10 or c1=14 or c1=18) then
                                nx_stat <= cpyscrpad;
                        else
                                nx_stat <= memcmd;
                        end if;
                when wrscrpad =>
                        if (j = 12 and wr = '1') then   -- if CRC\ = calculated CRC
                                nx_stat <= RXCRC;
                        else
                                nx_stat <= wrscrpad;
                        end if;

                -- if rxdata = X"AAAAAAAA" at ldcnt = 3 then keydn '1'
                when RXCRC =>
                        if((CRCOK = '1' or keydn = '1') and pr_state = ideal)then

                                nx_stat <= initial;
                        else
                                nx_stat <= RXCRC;
                        end if;

                when rdscrpad =>
                        if(rdcnt > 27 and rd = '1' and pr_state = ideal) then
                                nx_stat <= initial;
                        else
                                nx_stat <= rdscrpad;
                        end if;

                when ldfkey =>
                        if(j = 5) then
                                nx_stat <= progwait;
                        else
                                nx_stat <= ldfkey;
                        end if;

                when progwait =>
                        if(waitcnt = 10003) then
                                nx_stat <= RXCRC;
                        else
                                nx_stat <= progwait;
                        end if;

                when rdmem =>
                        if(rdmemcnt = 8 and pr_state = ideal)then
                                nx_stat <= initial;
                        else
                                nx_stat <= rdmem;
                        end if;


                when cpyscrpad =>
                        if(j > 4) then
                                nx_stat <= macwait;
                        else
```

```
                nx_stat <= cpyscrpad;
            end if;

        when macwait =>
            if(waitcnt > 2125)then
                nx_stat <= sendmac;
            else
                nx_stat <= macwait;
            end if;
        when sendmac =>
            if(mscnt = 20) then
                nx_stat <= progwait;
            else
                nx_stat <= sendmac;
            end if;


    end case;
end process;


process(wr,rst)
begin
    if rst = '0' then
        j <= 1;
        mscnt <= 0;
    elsif(wr'event and wr = '1') then
        if(cnten = '1') then
            j <= j + 1;
        else
            j <= 1;
        end if;
        if(mscnten = '1')then
            mscnt <= mscnt + 1;
        else
            mscnt <= 0;
        end if;
    end if;
end process;

process(rst,rd)
begin
    if rst = '0' then
        rdmemcnt <= 0;
    elsif(rd'event and rd = '1') then
        if(rdmemcnten = '1') then
```

```
                        rdmemcnt <= rdmemcnt + 1;
                else
                        rdmemcnt <= 0;
                end if;
        end if;
end process;

process(C1MHz,RST)
begin
 if(RST = '0') then
                m0 <= secret(3) & secret(4) & secret(5) & secret(6);
                m12 <= secret(7) & secret(8) & secret(9) & secret(10);
                m1 <= (others => '0');
                m2 <= (others => '0');
                m3 <= (others => '0');
                m4 <= (others => '0');
                m5 <= (others => '0');
                m6 <= (others => '0');
                m7 <= (others => '0');
                m8 <= (others => '0');
                m9 <= (others => '0');
                m10 <= (others => '0');
                m11 <= (others => '0');
                A <= (others => '0') ;
                b <= (others => '0') ;
                c <= (others => '0') ;
                d <= (others => '0') ;
                e <= (others => '0') ;
                message_vld <= '0';
                MACRST <= '1';
 elsif(C1MHz'event and C1MHz = '1') then
                if(rd = '1' and pr_stat = rdmem) then
                  if(rdmemcnt = 1) then
                        m1 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23
downto 16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 2) then
                        m2 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23
downto 16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 3) then
                        m3 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23
downto 16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 4) then
                        m4 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23
downto 16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 5) then
```

```
                    m5 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23
downto 16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 6) then
                   m6 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23 downto
16) & rxdata(31 downto 24);
                  elsif(rdmemcnt = 7) then
                   m7 <=  rxdata(7 downto 0) & rxdata(15 downto 8) & rxdata(23 downto
16) & rxdata(31 downto 24);
                  end if;
                end if;

                if(c1 = 4 and pr_stat = initial) then
                    m8 <= page_01(3) & page_01(4) & page_01(5) & page_01(6);
                    m9 <= page_01(7) & page_01(8) & page_01(9) & page_01(10);
                    m10 <= X"00" & X"33" & X"6D" & X"1F";
                    m11 <= X"25" & X"01" & X"00" & X"00";
                elsif(c1 = 8 and pr_stat = initial) then
                    m8 <= page_02(3) & page_02(4) & page_02(5) & page_02(6);
                    m9 <= page_02(7) & page_02(8) & page_02(9) & page_02(10);
                    m10 <= X"02" & X"33" & X"6D" & X"1F";
                    m11 <= X"25" & X"01" & X"00" & X"00";
                elsif(c1 = 12 and pr_stat = initial) then
                    m8 <= page_03(3) & page_03(4) & page_03(5) & page_03(6);
                    m9 <= page_03(7) & page_03(8) & page_03(9) & page_03(10);
                    m10 <= X"04" & X"33" & X"6D" & X"1F";
                    m11 <= X"25" & X"01" & X"00" & X"00";
                elsif(c1 = 16 and pr_stat = initial) then
                    m8 <= page_04(3) & page_04(4) & page_04(5) & page_04(6);
                    m9 <= page_04(7) & page_04(8) & page_04(9) & page_04(10);
                    m10 <= X"06" & X"33" & X"6D" & X"1F";
                    m11 <= X"25" & X"01" & X"00" & X"00";
                end if;

                if(digest_vld <= '1') then
                    A <= digest(159 downto 128);
                    B <= digest(127 downto 96);
                    C <= digest(95 downto 64);
                    D <= digest(63 downto 32);
                    E <= digest(31 downto 0);
                end if;

                if(pr_stat = rdscrpad) then
                    message_vld <= '1';
                end if;

                if(pr_stat = progwait) then
```

```
                        MACRST <= '1';
                        message_vld <= '0';
                else
                        MACRST <= '0';
                end if;

 end if;

 end process;

end Behavioral;
```

6.4 Secure Hash Algorithm

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;



entity MAC is
port(
                clk : in std_logic;
                reset : in std_logic;
                m0i,m1i,m2i,m3i,m4i,m5i,m6i,m7i,m8i,m13i,m10i,m11i,m12i : in
std_logic_vector(31 downto 0);
                message_vld : in std_logic;
                digest : out std_logic_vector(159 downto 0);
                digest_vld : out std_logic
                );
end MAC;

architecture behavioral of MAC is
signal run_hash : std_logic;
signal w0,w1,w2,w3,w4,w5,w6, w7, w8, w9, w10,w11,
                 w12,w13,w14,w15,w, wi : std_logic_vector (31 downto 0);
```

```
signal a, b, c, d, e, Kt, temp : std_logic_vector (31 downto 0);
signal count : std_logic_vector (4 downto 0);
signal stage : std_logic_vector (1 downto 0);
type sha_state_type is (setup, calc, done);
signal sha_state : sha_state_type;
signal m0,m1,m2,m3,m4,m5,m6,m7,m8,m13,m10,m11,m12 : std_logic_vector(31
downto 0);

-- appended a 1, then zeroes result 80
constant m9 : std_logic_vector(31 downto 0) := X"FFFFFFFF";
constant m14 : std_logic_vector(31 downto 0)         := X"00000000";
constant m15 : std_logic_vector(31 downto 0)         := X"000001B8";

-- 64-bit representation of the length of the string
constant Ka : std_logic_vector(31 downto 0)
        := "01100111010001010010001100000001"; --X"67452301";
constant Kb : std_logic_vector(31 downto 0)
        := "11101111110011011010101110001001"; --X"efcdab89";
constant Kc : std_logic_vector(31 downto 0)
        := "10011000101110101101110011111110"; --X"98badcfe";
constant Kd : std_logic_vector(31 downto 0)
        := "00010000001100100101010001110110"; --X"10325476";
constant Ke : std_logic_vector(31 downto 0)
        := "11000011110100101110000111110000"; --X"c3d2e1f0";
constant Kt0 : std_logic_vector(31 downto 0)
        := "01011010100000100111100110011001"; --X"5a827999";
constant Kt20 : std_logic_vector(31 downto 0)
        := "01101110110110011110101110100001"; --X"6ed9eba1";
constant Kt40 : std_logic_vector(31 downto 0)
        := "10001111000110111011110011011100"; --X"8f1bbcdc";
constant Kt60 : std_logic_vector(31 downto 0)
```

```
        := "11001010011000101100000111010110"; --X"ca62c1d6";

begin
start_signal : process (clk, reset)
begin
if (reset = '1') then
        m0 <= (others => '0');
        m1 <= (others => '0');
        m2 <= (others => '0');
        m3 <= (others => '0');
        m4 <= (others => '0');
        m5 <= (others => '0');
        m6 <= (others => '0');
        m7 <= (others => '0');
        m8 <= (others => '0');
        m13 <= (others => '0');
        m10 <= (others => '0');
        m11 <= (others => '0');
        m12 <= (others => '0');
        run_hash <= '0';
 elsif (clk'event and clk = '1') then
        if (message_vld = '1') then

                m0 <= m0i;
                m1 <= m1i;
                m2 <= m2i;
                m3 <= m3i;
                m4 <= m4i;
                m5 <= m5i;
                m6 <= m6i;
                m7 <= m7i;
        m8 <= m8i;
        m13 <= m13i;
        m10 <= m10i;
        m11 <= m11i;
        m12 <= m12i;

        run_hash <= '1';

    end if;
 end if;
end process;


wi <= w13 XOR w8 XOR w2 XOR w0;
---******
HASH : process (clk, reset)
begin
if (reset = '1') then
    a <= Ka; b <= Kb; c <= Kc; d <= Kd; e <= Ke;
    w0 <= (others => '0');
    w1 <= (others => '0');
    w2 <= (others => '0');
    w3 <= (others => '0');
    w4 <= (others => '0');
    w5 <= (others => '0');
    w6 <= (others => '0');
    w7 <= (others => '0');
    w8 <= (others => '0');
    w13 <= (others => '0');
    w10 <= (others => '0');
    w11 <= (others => '0');
    w12 <= (others => '0');

    w9 <= m9;
    w14 <= m14; w15 <= m15;
    w <= (others => '0');

    Kt <= Kt0;
    temp <= (others => '0');
    count <= "00000";
    stage <= "00";
    sha_state <= setup;
    digest <= (others => '0');
    digest_vld <= '0';
elsif(clk'event and clk = '1') then

if (run_hash = '1') then


case sha_state is
 when setup =>
  if (count < "10000" and stage = "00") then
    case count is
        when "00000" => w <= m0; w0 <= m0; w1 <= m1; w2 <=  m2; w3 <=
m3; w4 <= m4; w5 <= m5; w6 <= m6; w7 <= m7;
                                w8 <= m8; w13 <= m13;
w10 <= m10; w11 <= m11; w12 <= m12;
        when "00001" => w <= w1;
        when "00010" => w <= w2;
        when "00011" => w <= w3;
        when "00100" => w <= w4;
        when "00101" => w <= w5;
```

```
                when "00110" => w <= w6;
                when "00111" => w <= w7;
                when "01000" => w <= w8;
                when "01001" => w <= w9;
                when "01010" => w <= w10;
                when "01011" => w <= w11;
                when "01100" => w <= w12;
                when "01101" => w <= w13;
                when "01110" => w <= w14;
                when "01111" => w <= w15;
                when others => w <= (others => '1') ;
        end case;
  else
        sha_state <= calc;

  when calc =>
                e <= d;
                d <= c;
                c <= b(1 downto 0) & b(31 downto 2);
                b <= a;

        case stage is
                when "00" =>
                        a <= ( a(26 downto 0) & a(31 downto 27) ) +
                                        ( (b and c) or ((not b) and d) ) +
                                        ( e + w(31 downto 0) + Kt );                if
(count = "10011") then
                                stage <= "01";
                        end if;
                sha_state <= setup;
                when "01" =>
        a <= ( a(26 downto 0) & a(31 downto 27) ) +
                ( b xor c xor d ) +
                ( e + w(31 downto 0) + Kt );
        if (count = "10011") then
            stage <= "10";
        end if;
    sha_state <= setup;
    when "10" =>
        a <= ( a(26 downto 0) & a(31 downto 27) ) +
                ( (b and c) or (b and d ) or (c and d)) +
                ( e + w(31 downto 0) + Kt );
        if (count = "10011") then
            stage <= "11";
        end if;
        sha_state <= setup;
    when "11" =>
        a <= ( a(26 downto 0) & a(31 downto 27) ) +
                ( b xor c xor d ) +
                ( e + w(31 downto 0) + Kt );
        if (count = "10011") then
            sha_state <= done;
        else
        sha_state <= setup;
        end if;
    when others =>
        a <= (others => '1');
 end case;
if (count > "01111" or stage > "00") then
    w0 <= w1; w1 <= w2; w2 <= w3; w3 <= w4; w4 <= w5;
    w5 <= w6; w6 <= w7; w7 <= w8; w8 <= w9; w9 <= w10;
    w10 <= w11; w11 <= w12; w12 <= w13;
```

```
                w13 <= w14; w14 <= w15;
                w15 <= wi(30 downto 0) & wi(31);
        end if;
  when done =>
                digest <= a & b & c & d & e;
                digest_vld <= '1';
end case;
end if;
end if;
end process;


end behavioral;
```

6.5 CRC Generation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CRC is
   Port ( clk,rst : in  STD_LOGIC;
                              CRCDATAld : in std_logic;
                              datain : in std_logic_vector(87 downto 0);
                              CRC_out : out std_logic_vector(15 downto 0));
end CRC;

architecture Behavioral of CRC is
        signal CRC : std_logic_vector(15 downto 0);
        signal data : std_logic_vector(87 downto 0);
        signal cnt : integer;
        signal P : std_logic;
begin

 P <= CRC(0) xor data(0);
process(clk,CRCDATAld,rst)
 begin
        if(CRCDATAld = '1' or rst = '0') then
                        data <= datain;
                        cnt <= 0;
                        CRC <= (others => '0');
```

```
          elsif(clk'event and clk = '1') then
                    if cnt < 88 then
                              CRC(15) <= P;
                              CRC(14) <= CRC(15);
                              CRC(13) <= CRC(14) xor p;
                              CRC(12 downto 1) <= CRC(13 downto 2);
                              CRC(0) <= CRC(1) xor P;

                              data <= '0' & data (87 downto 1);
                              cnt <= cnt + 1;
                    else
                              CRC_out <= not CRC;
                    end if;
          end if;
 end process;

end Behavioral;
```

Chapter 7

Results

On chip Verification and real time debugging of the Virtex II Pro is done using ChipScope Pro 8.1i Embedded Design tool. Fgure 7.2 shows the obtained results for the LOADTEST design. Table 7.1 shows the values for the various 1-wire signaling.

Pr_state[1:3]	**Value**
Ideal	000
Rstpls	001
Prs	011
Write-0	111
Write-1	110
Read	010

Table7.1 1-wire signaling

Table 7.2 shows the values for the various states during the execution of the LOADTEST.

Pr_stat[1:4]	**Values**
Initial	0000
Skiprom	0001
Rxcrc	1100
Memcmd	0011
Wrscrpad	0010
Rdscrpad	0110

Ldfkey	0111
Sendmac	1110
Progwait	1101
Macwait	1111
Rdmem	0101
Cpyscrpad	0100

Table 7.2 Execution states of LOADTEST

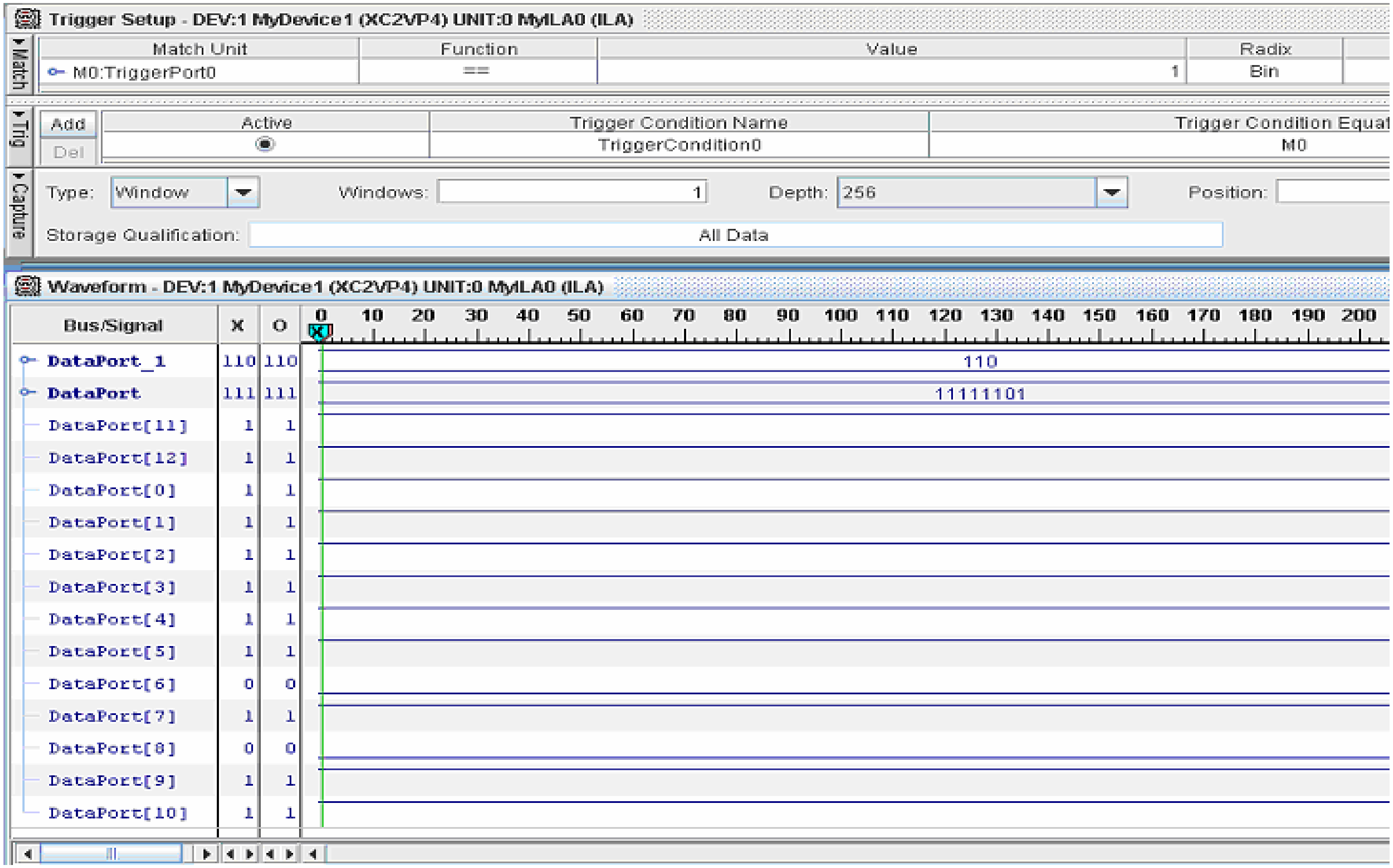
Figure 7.2 Shows the results obtained for the LOADTEST design. The ‘loaddone’ signal indicates that the secret key has been written correctly into the DS2432. While reading the page data from the DS2432 ‘CRCOK’ is activated if data is read correctly from the DS2432. Signal ‘rxdata’ contains the data received from the DS2432. During the verification of the LOADTEST is found that all the signal are generated properly as expected. And results confirms the design. The count ‘c1’ corresponds to the different processes in the LOADTEST design.

Table shows the various states of the execution process of the IFFTEST.

Pr_stat[1:3]	**Values**
Initial	000
Skiprom	001
Rxcrc	101
Rdauth_cmd	110
Rdmac	111
Memcmd	011
Wrscrpad	010
Done	100

Table 7.3 Execution states of IFFTEST

Figure 7.3 shows the various waveforms for the IFFTEST design. ‘FRND’ is activated only when the secret key stored in the DS2432 matches with that stored in the FPGA. Figure 7.1 shows the results associated with the 3 to 8 decoder. It is shown that the 3 to 8 decoder is activated only when ‘FRND’ is activated. Tests have been performed by changing the key stored in the FPGA. It is seen that the user application is remains deactivated. It is activated only when the keys of FPGA and DS2432 matches. So, the IFFTEST is verified and it is performing according to the design. The signal ‘c1’ corresponds to the different processes of the IFFTEST design.



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix
M0:TriggerPort0	==	1	Bin

Active	Trigger Condition Name	Trigger Condition Equat
●	TriggerCondition0	M0

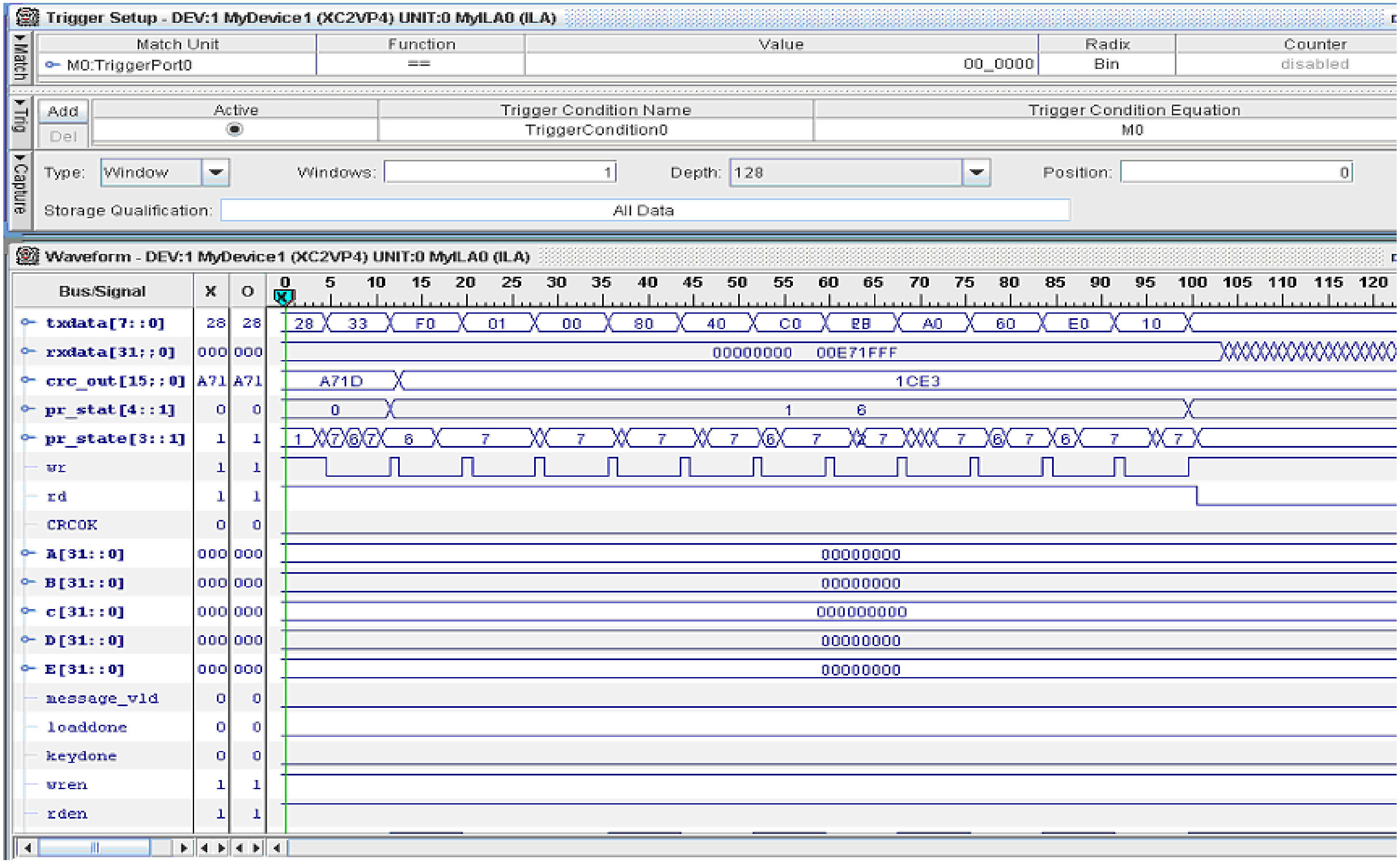
Type: Window Windows: 1 Depth: 256 Position:

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	
DataPort_1	110	110	110
DataPort	111	111	11111101
DataPort[11]	1	1	
DataPort[12]	1	1	
DataPort[0]	1	1	
DataPort[1]	1	1	
DataPort[2]	1	1	
DataPort[3]	1	1	
DataPort[4]	1	1	
DataPort[5]	1	1	
DataPort[6]	0	0	
DataPort[7]	1	1	
DataPort[8]	0	0	
DataPort[9]	1	1	
DataPort[10]	1	1	

Figure 7.1: 3 to 8 decoder



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0000	Bin	disabled

Add / Del	Active	Trigger Condition Name	Trigger Condition Equation
	●	TriggerCondition0	M0

Type: Window

Windows: 1

Depth: 128

Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	28	28
rxdata[31;;0]	000	000
crc_out[15::0]	A71	A71
pr_stat[4::1]	0	0
pr_state[3::1]	1	1
wr	1	1
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	0	0
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

Figure 7.2 (a): Execution of LOADTEST

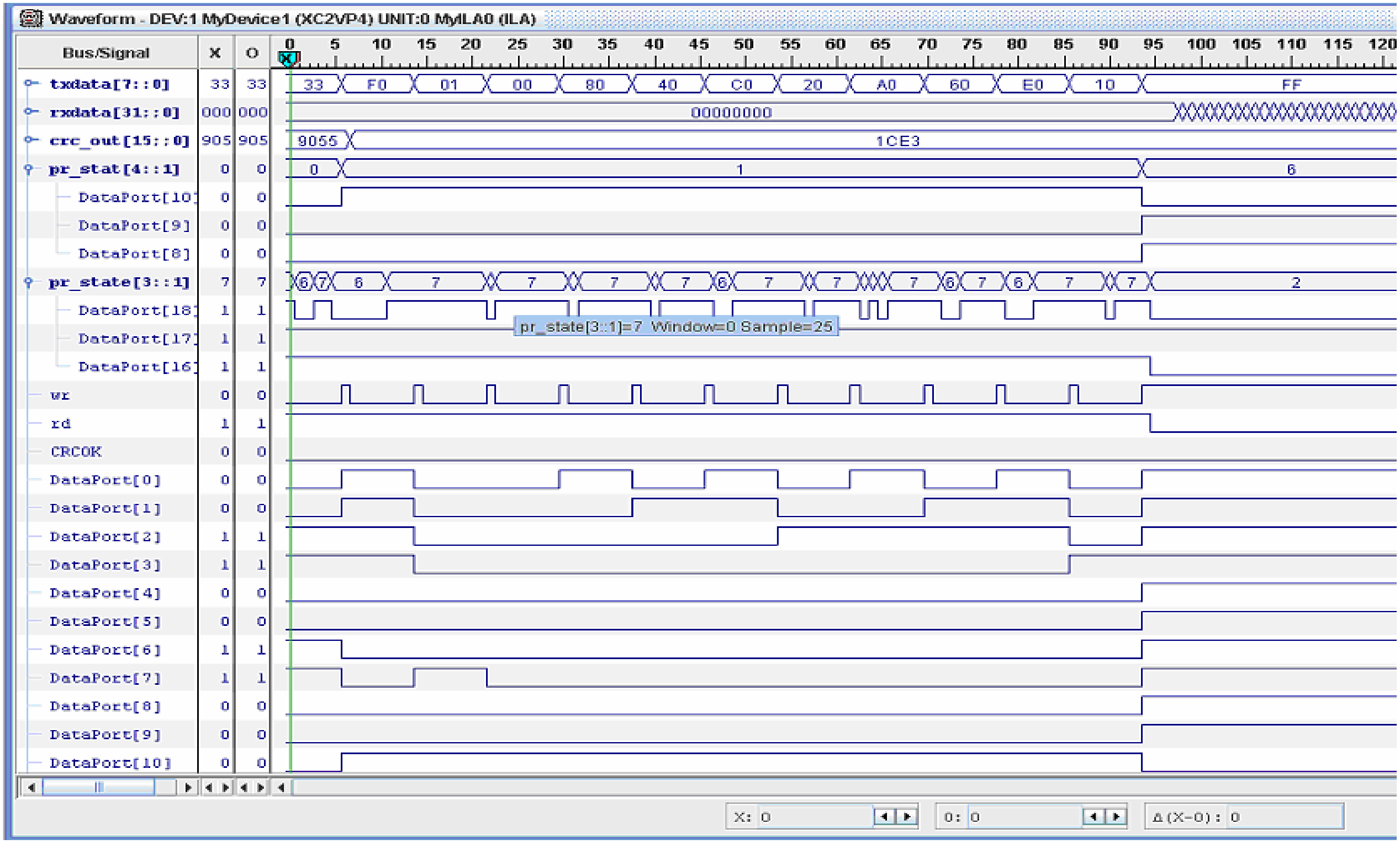
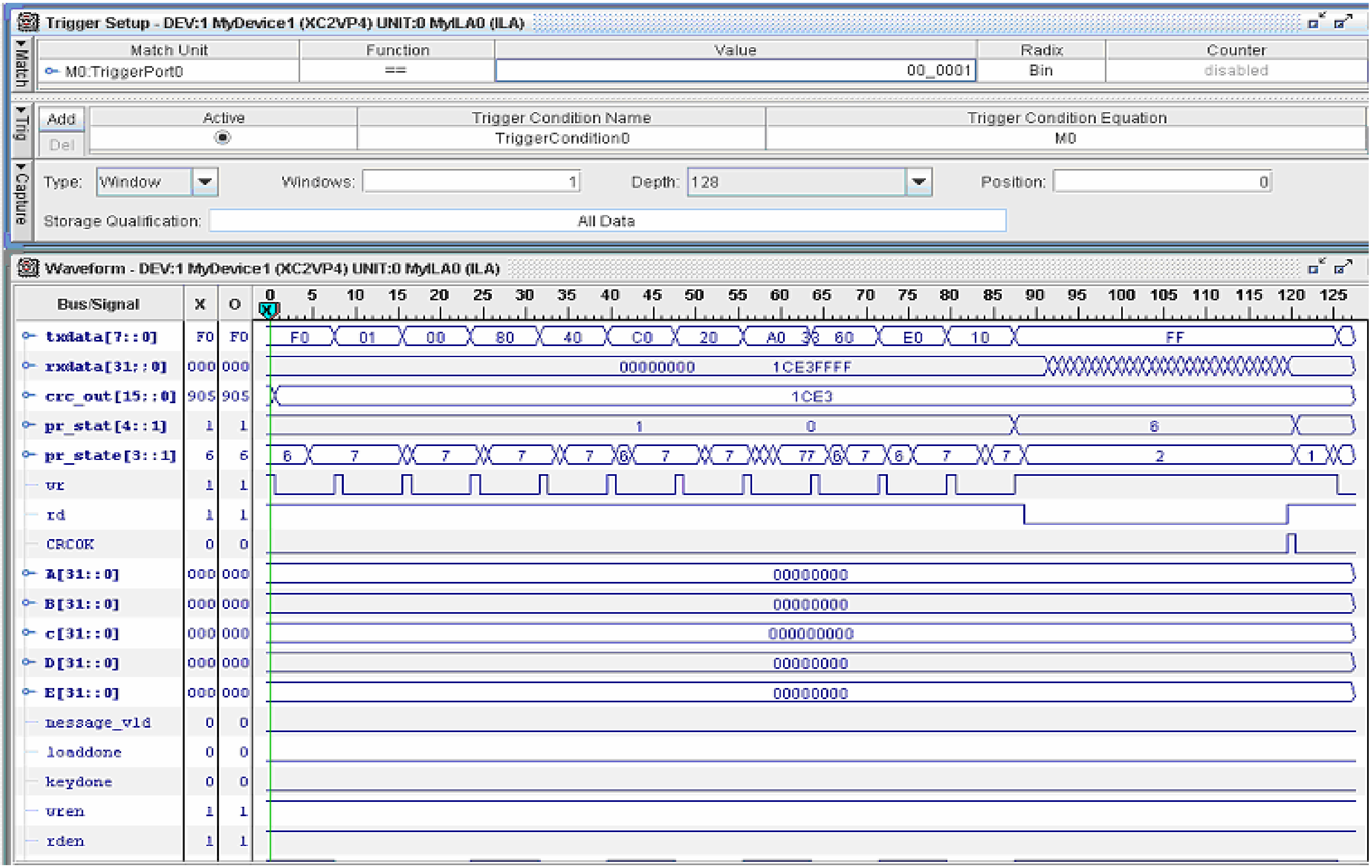


Figure 7.2 (b): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0001	Bin	disabled

Add / Del	Active	Trigger Condition Name	Trigger Condition Equation
	●	TriggerCondition0	M0

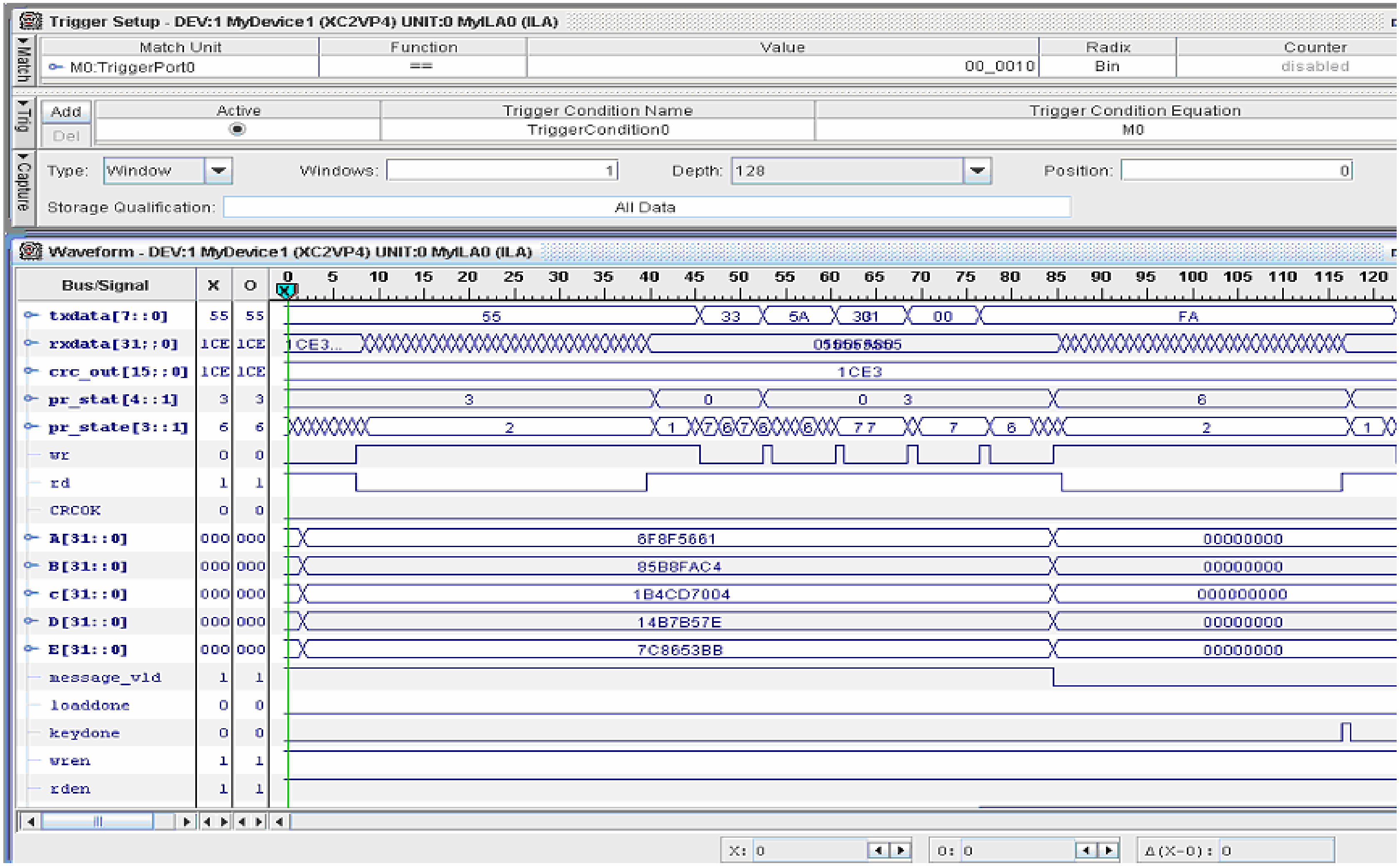
Type: Window Windows: 1 Depth: 128 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	F0	F0
rxdata[31::0]	000	000
crc_out[15::0]	905	905
pr_stat[4::1]	1	1
pr_state[3::1]	6	6
wr	1	1
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	0	0
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

Figure 7.2 (c): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0010	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 128 Position: 0

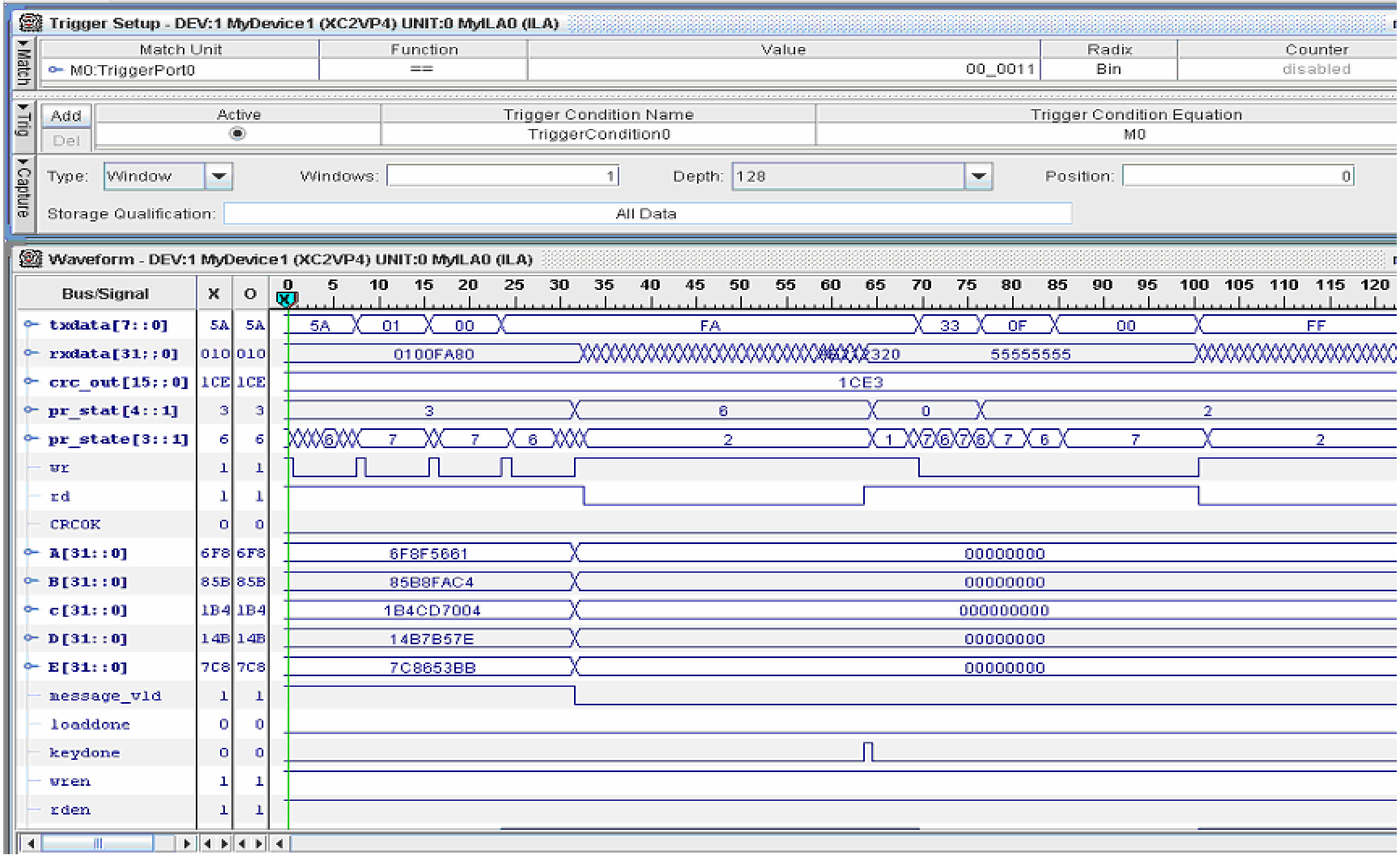
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	55	55
rxdata[31::0]	1CE	1CE
crc_out[15::0]	1CE	1CE
pr_stat[4::1]	3	3
pr_state[3::1]	6	6
wr	0	0
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	1	1
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

X: 0 O: 0 Δ(X-O): 0

Figure 7.2 (d): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0011	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

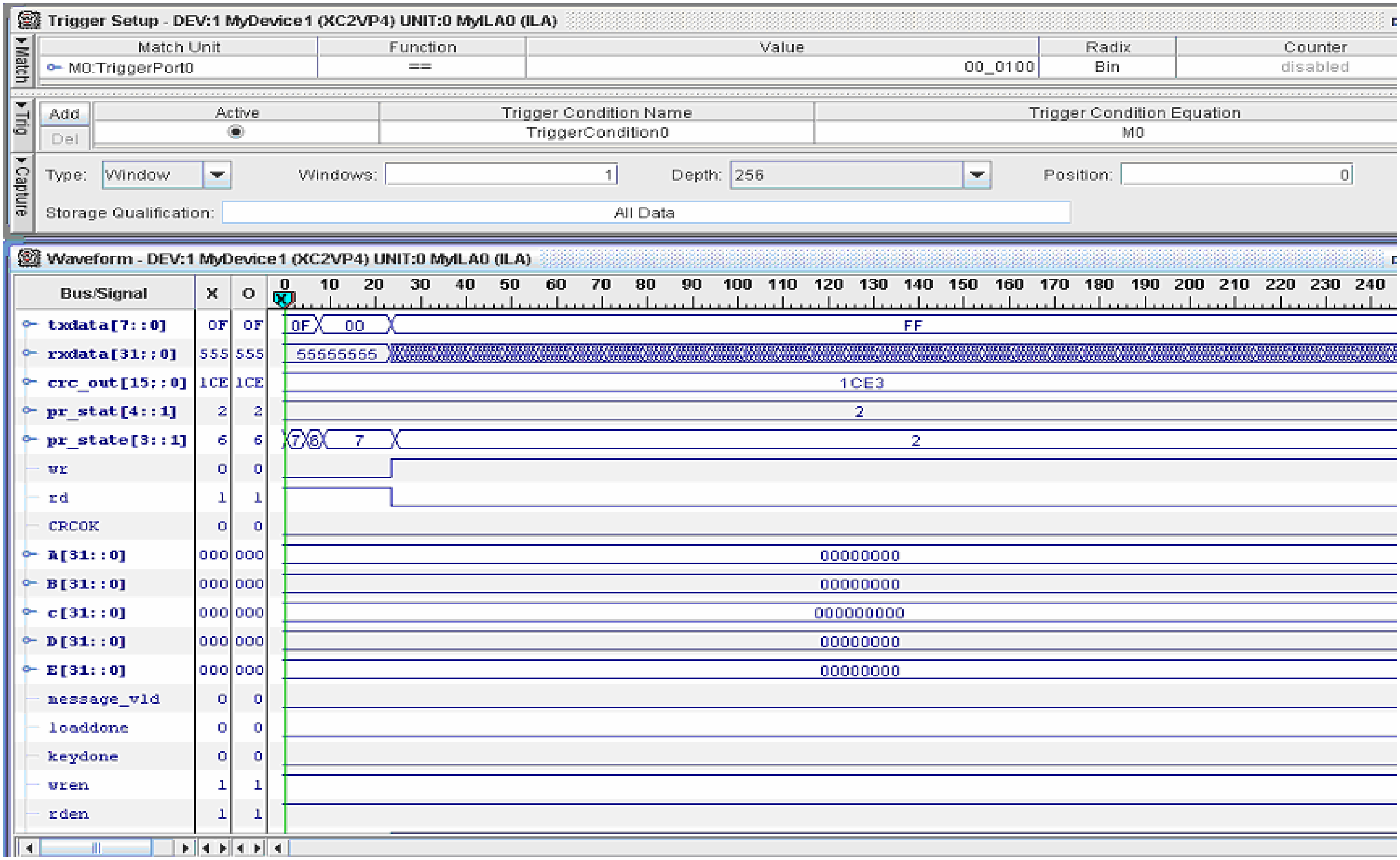
Type: Window Windows: 1 Depth: 128 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	5A	5A
rxdata[31::0]	010	010
crc_out[15::0]	1CE	1CE
pr_stat[4::1]	3	3
pr_state[3::1]	6	6
wr	1	1
rd	1	1
CRCOK	0	0
A[31::0]	6F8	6F8
B[31::0]	85B	85B
c[31::0]	1B4	1B4
D[31::0]	14B	14B
E[31::0]	7C8	7C8
message_vld	1	1
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

Figure 7.2 (e): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0100	Bin	disabled

Trig

	Active	Trigger Condition Name	Trigger Condition Equation
Add Del	●	TriggerCondition0	M0

Capture

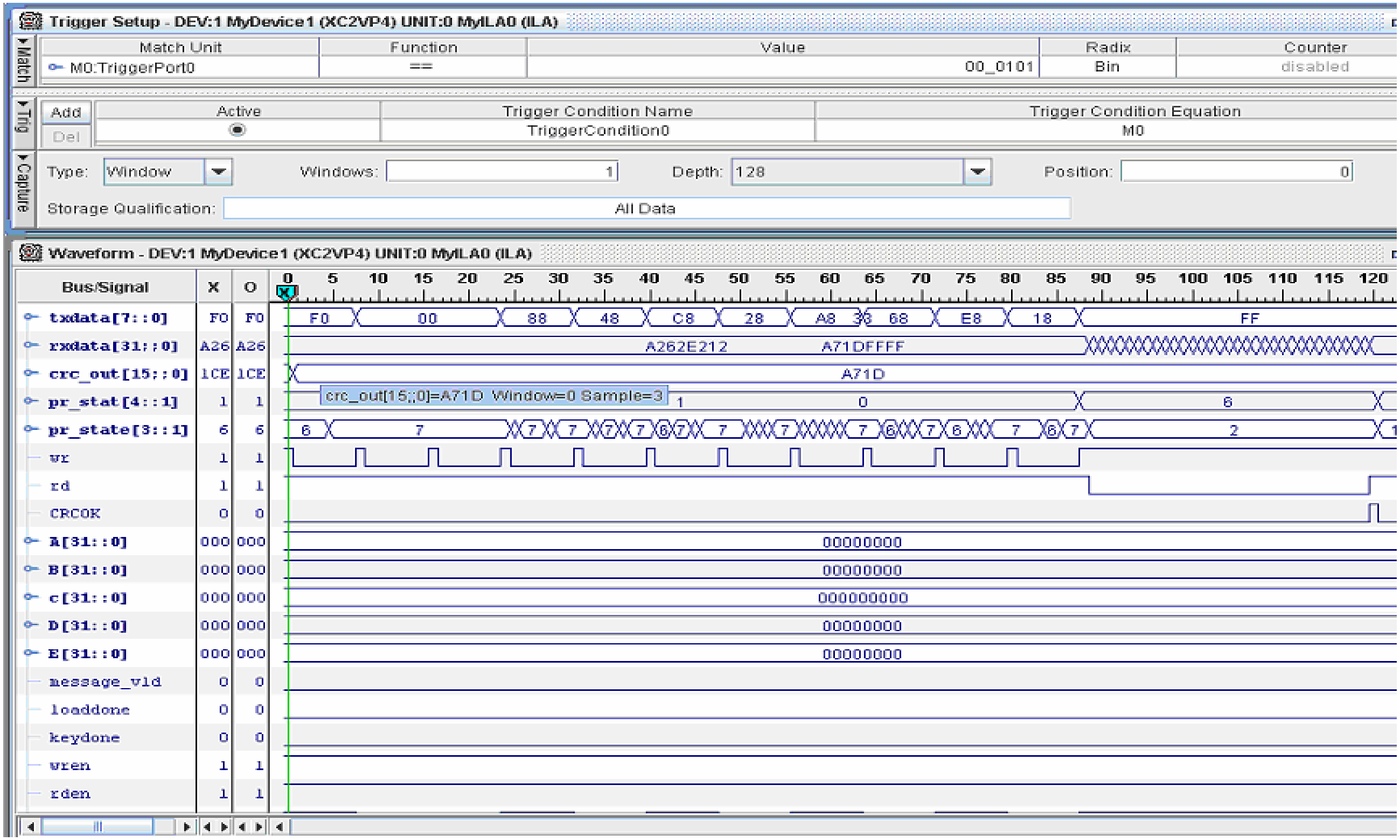
Type: Window Windows: 1 Depth: 256 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform
txdata[7::0]	0F	0F	0F, 00, FF
rxdata[31::0]	555	555	55555555
crc_out[15::0]	1CE	1CE	1CE3
pr_stat[4::1]	2	2	2
pr_state[3::1]	6	6	7, 6, 7, 2
wr	0	0	
rd	1	1	
CRCOK	0	0	
A[31::0]	000	000	00000000
B[31::0]	000	000	00000000
c[31::0]	000	000	000000000
D[31::0]	000	000	00000000
E[31::0]	000	000	00000000
message_vld	0	0	
loaddone	0	0	
keydone	0	0	
wren	1	1	
rden	1	1	

Figure 7.2 (f): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0101	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

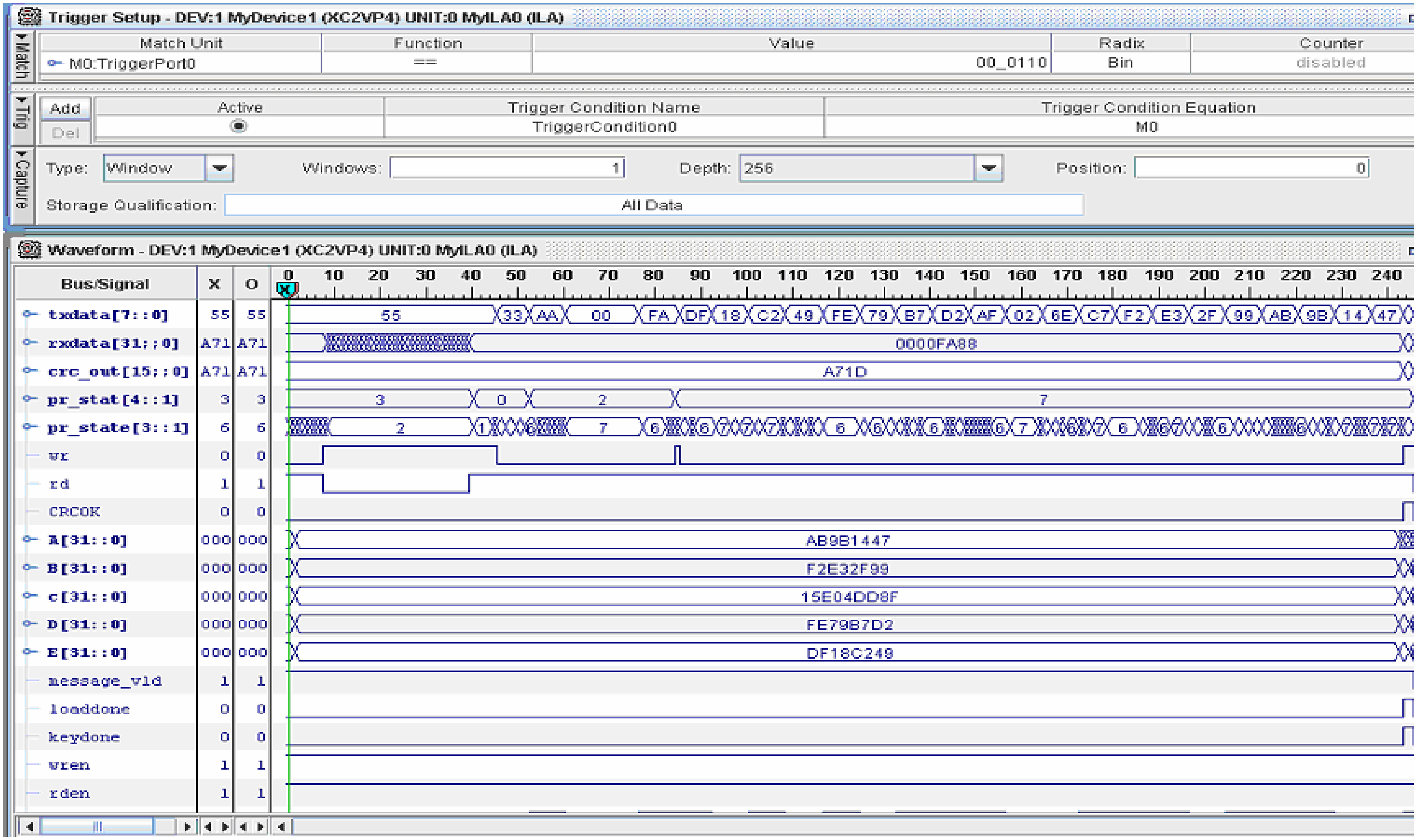
Type: Window
Windows: 1
Depth: 128
Position: 0
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	F0	F0
rxdata[31::0]	A26	A26
crc_out[15::0]	1CE	1CE
pr_stat[4::1]	1	1
pr_state[3::1]	6	6
wr	1	1
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	0	0
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

crc_out[15::0]=A71D Window=0 Sample=3

Figure 7.2 (g): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0110	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

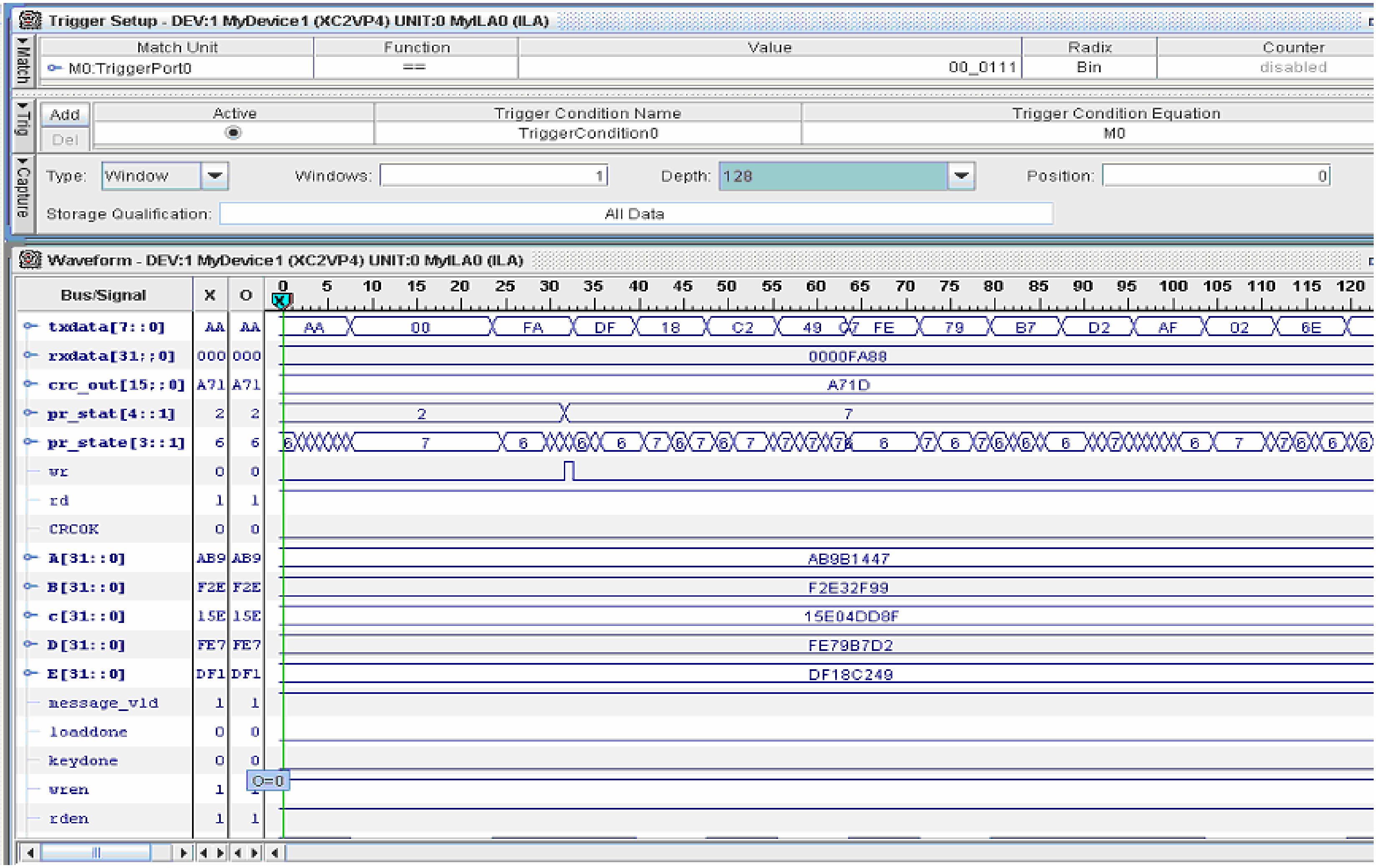
Type: Window Windows: 1 Depth: 256 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	55	55
rxdata[31::0]	A71	A71
crc_out[15::0]	A71	A71
pr_stat[4::1]	3	3
pr_state[3::1]	6	6
wr	0	0
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	1	1
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

Figure 7.2 (h): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_0111	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

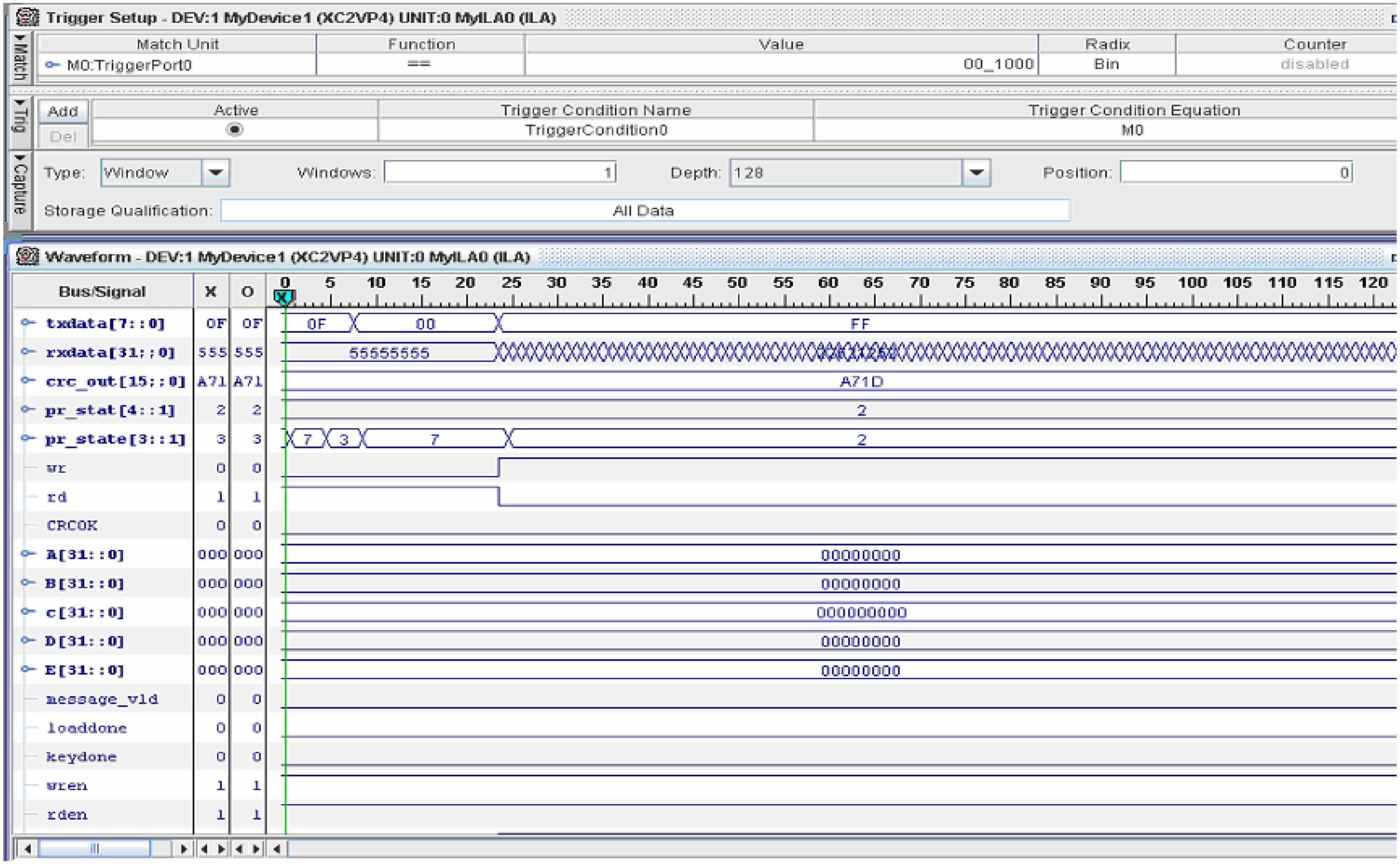
Type: Window Windows: 1 Depth: 128 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform
txdata[7::0]	AA	AA	AA, 00, FA, DF, 18, C2, 49, C7, FE, 79, B7, D2, AF, 02, 6E
rxdata[31::0]	000	000	0000FA88
crc_out[15::0]	A71	A71	A71D
pr_stat[4::1]	2	2	2, 7
pr_state[3::1]	6	6	6, 7, 6, 6, 6, 7, 6, 7, 6, 7, 7, 7, 7, 6, 7, 6, 7, 6, 6, 6, 7, 6, 7, 7, 6, 6, 6
wr	0	0	
rd	1	1	
CRCOK	0	0	
A[31::0]	AB9	AB9	AB9B1447
B[31::0]	F2E	F2E	F2E32F99
c[31::0]	15E	15E	15E04DD8F
D[31::0]	FE7	FE7	FE79B7D2
E[31::0]	DF1	DF1	DF18C249
message_vld	1	1	
loaddone	0	0	
keydone	0	0	
wren	1	1	
rden	1	1	

Figure 7.2 (i): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	00_1000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

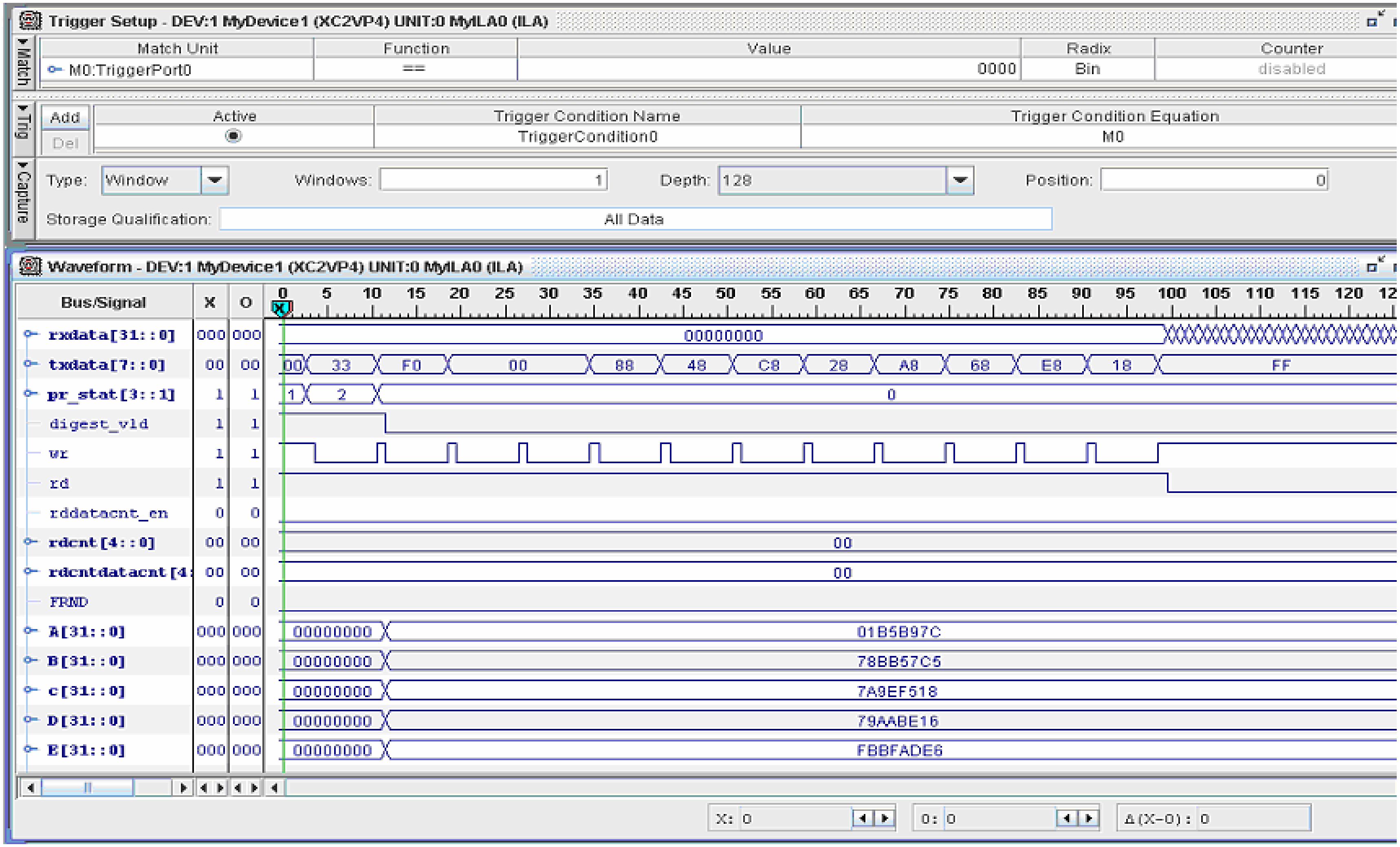
Type: Window Windows: 1 Depth: 128 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
txdata[7::0]	0F	0F
rxdata[31::0]	555	555
crc_out[15::0]	A71	A71
pr_stat[4::1]	2	2
pr_state[3::1]	3	3
wr	0	0
rd	1	1
CRCOK	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000
message_vld	0	0
loaddone	0	0
keydone	0	0
wren	1	1
rden	1	1

Figure 7.2 (j): Execution of LOADTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 128 Position: 0

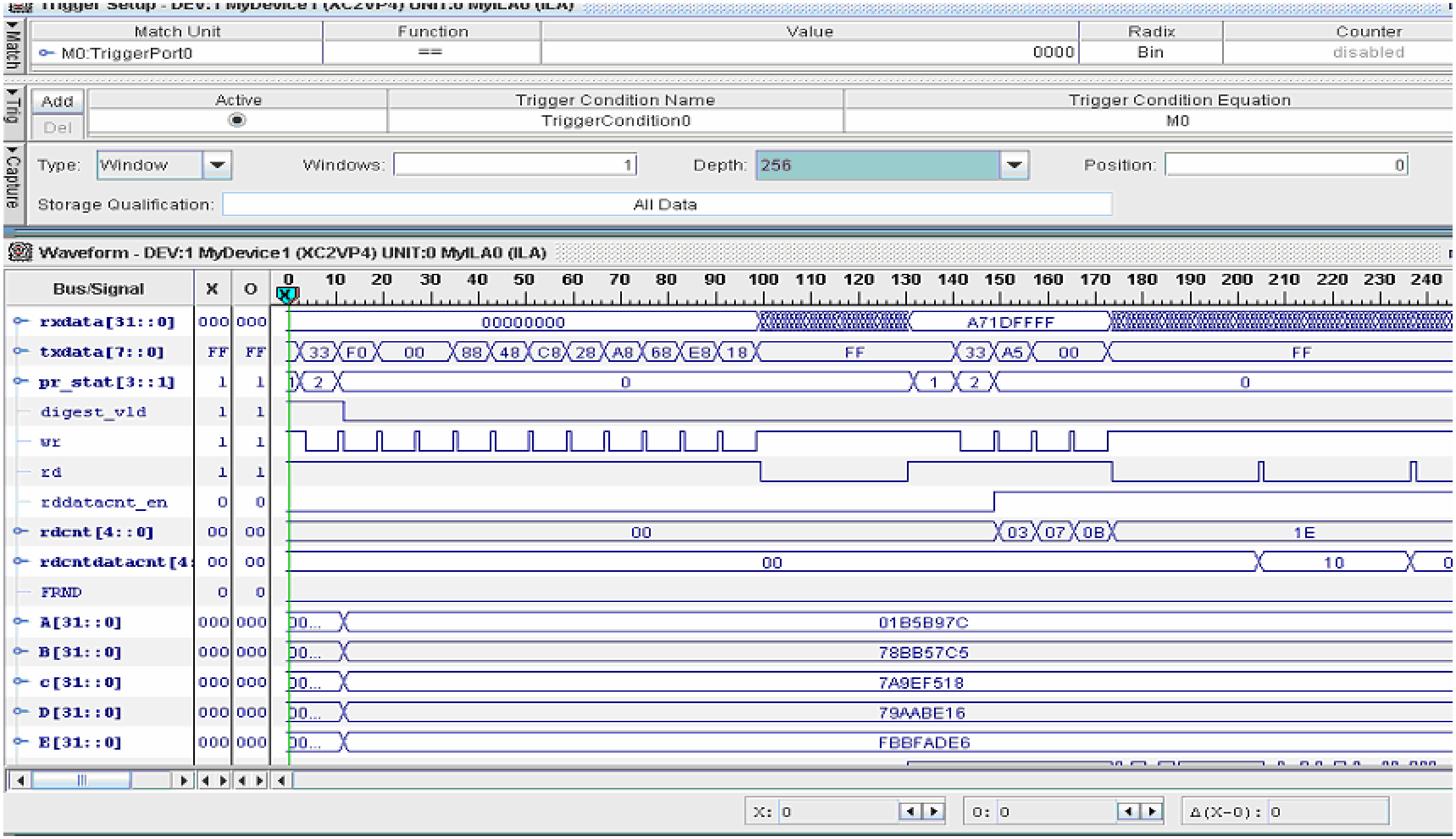
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform
rxdata[31::0]	000	000	00000000
txdata[7::0]	00	00	00, 33, F0, 00, 88, 48, C8, 28, A8, 68, E8, 18, FF
pr_stat[3::1]	1	1	1, 2, 0
digest_vld	1	1	
wr	1	1	
rd	1	1	
rddatacnt_en	0	0	
rdcnt[4::0]	00	00	00
rdcntdatacnt[4:	00	00	00
FRND	0	0	
A[31::0]	000	000	00000000, 01B5B97C
B[31::0]	000	000	00000000, 78BB57C5
C[31::0]	000	000	00000000, 7A9EF518
D[31::0]	000	000	00000000, 79AABE16
E[31::0]	000	000	00000000, FBBFADE6

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (a): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 256 Position: 0

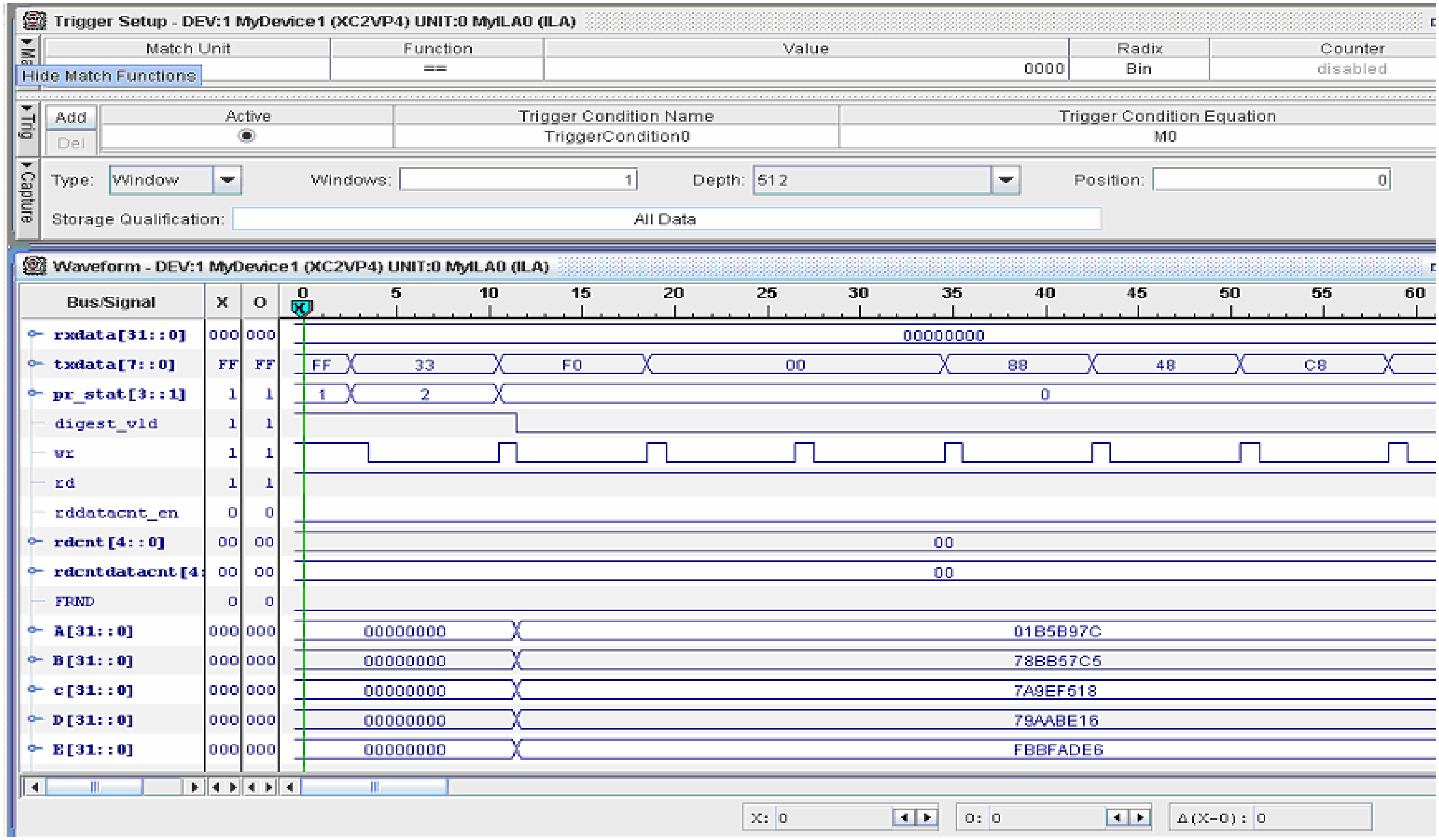
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
rxdata[31::0]	000	000
txdata[7::0]	FF	FF
pr_stat[3::1]	1	1
digest_vld	1	1
wr	1	1
rd	1	1
rddatacnt_en	0	0
rdcnt[4::0]	00	00
rdcntdatacnt[4:	00	00
FRND	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (b): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
Hide Match Functions	==	0000	Bin	disabled

	Active	Trigger Condition Name	Trigger Condition Equation
Add Del	●	TriggerCondition0	M0

Type: Window   Windows: 1   Depth: 512   Position: 0

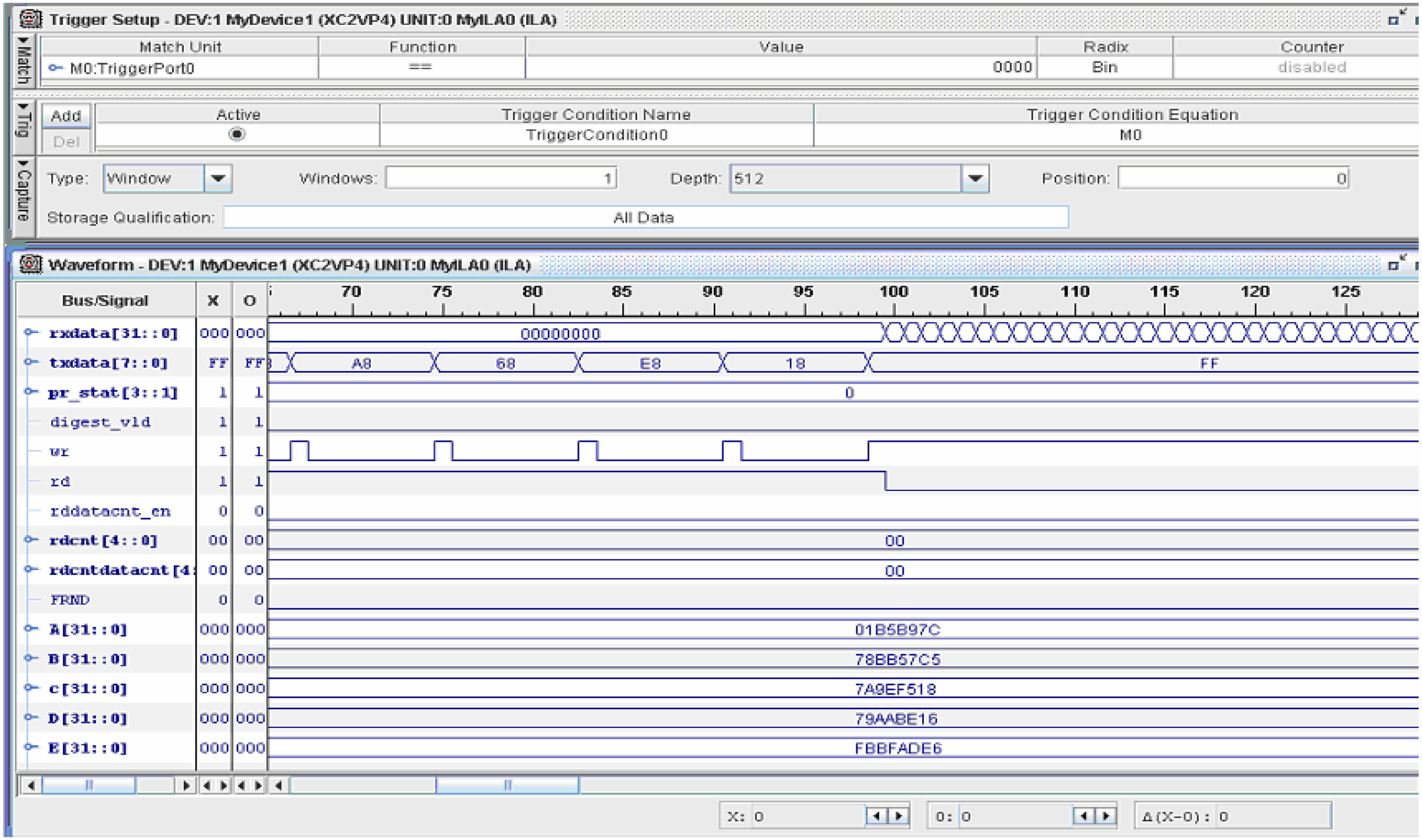
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform values
rxdata[31::0]	000	000	00000000
txdata[7::0]	FF	FF	FF, 33, F0, 00, 88, 48, C8
pr_stat[3::1]	1	1	1, 2, 0
digest_vld	1	1	
wr	1	1	
rd	1	1	
rddatacnt_en	0	0	
rdcnt[4::0]	00	00	00
rdcntdatacnt[4:	00	00	00
FRND	0	0	
A[31::0]	000	000	00000000, 01B5B97C
B[31::0]	000	000	00000000, 78BB57C5
C[31::0]	000	000	00000000, 7A9EF518
D[31::0]	000	000	00000000, 79AABE16
E[31::0]	000	000	00000000, FBBFADE6

X: 0   O: 0   Δ(X-O): 0

Figure 7.3 (c): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

	Active	Trigger Condition Name	Trigger Condition Equation
Add Del	●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position: 0

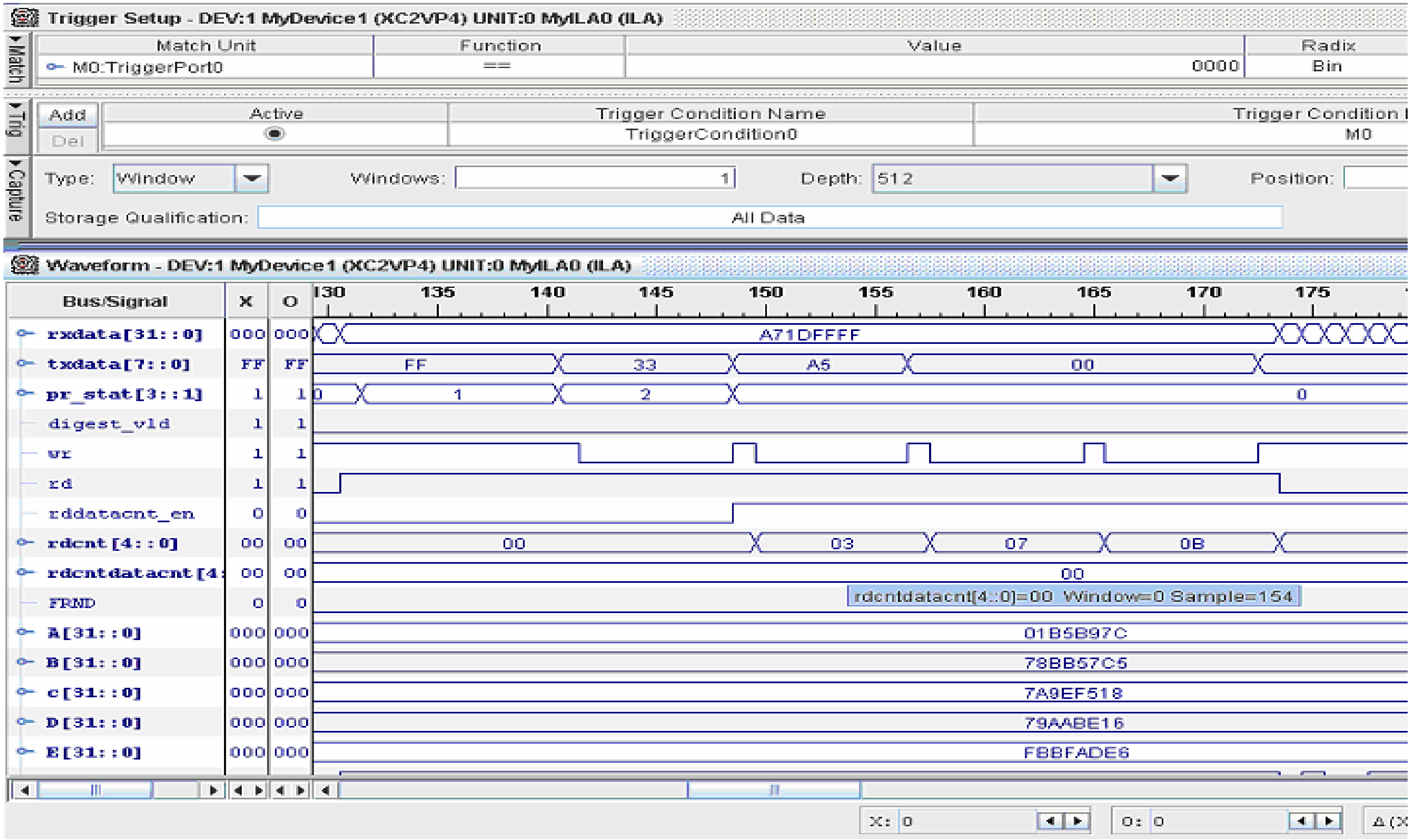
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
rxdata[31::0]	000	000
txdata[7::0]	FF	FF
pr_stat[3::1]	1	1
digest_vld	1	1
wr	1	1
rd	1	1
rddatacnt_en	0	0
rdcnt[4::0]	00	00
rdcntdatacnt[4:	00	00
FRND	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (d): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix
M0:TriggerPort0	==	0000	Bin

Active	Trigger Condition Name	Trigger Condition I
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position:

Storage Qualification: All Data

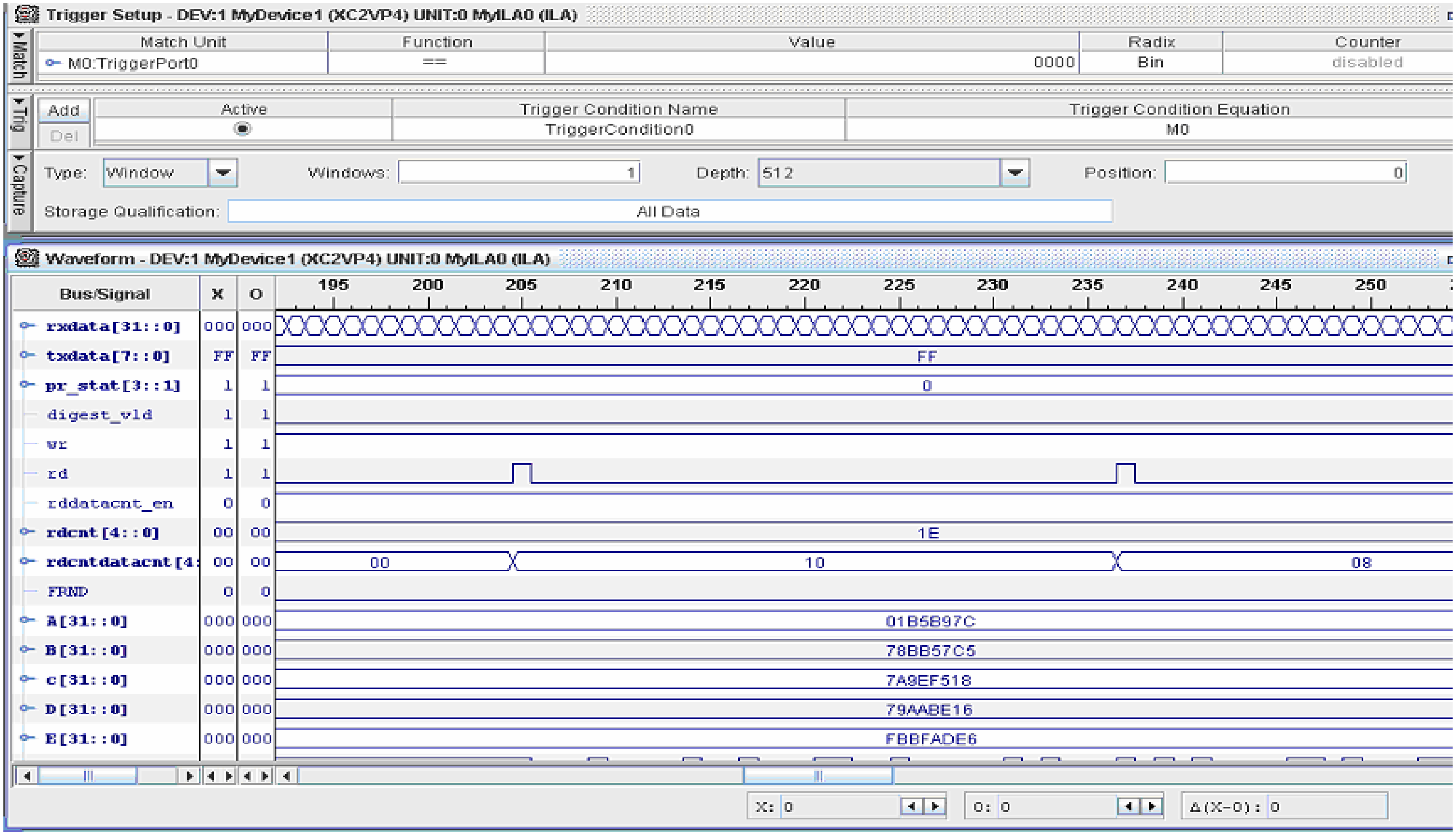
Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
rxdata[31::0]	000	000
txdata[7::0]	FF	FF
pr_stat[3::1]	1	1
digest_vld	1	1
wr	1	1
rd	1	1
rddatacnt_en	0	0
rdcnt[4::0]	00	00
rdcntdatacnt[4:	00	00
FRND	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000

rdcntdatacnt[4::0]=00 Window=0 Sample=154

X: 0 O: 0 Δ(X

Figure 7.3 (e): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position: 0

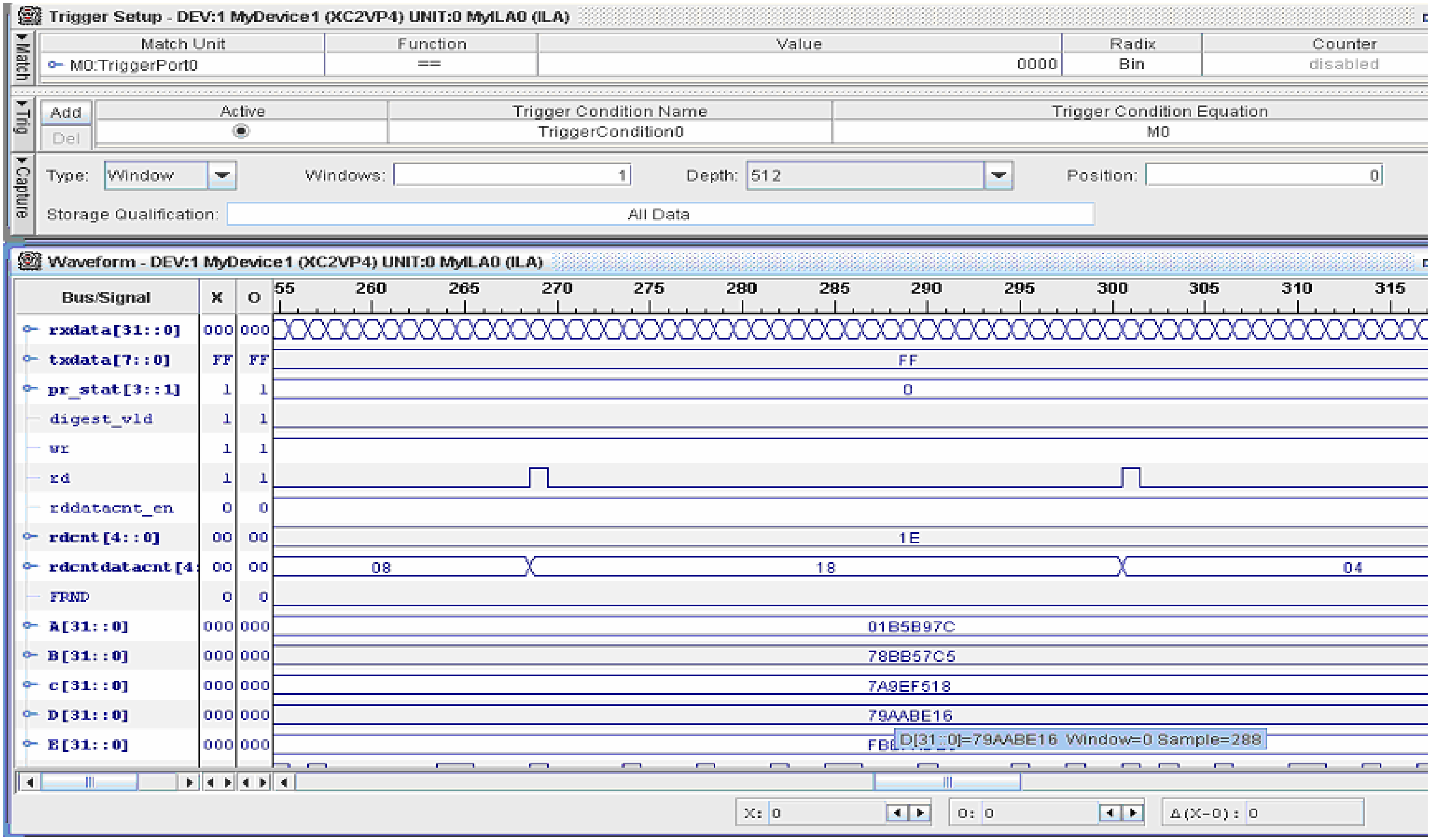
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	
rxdata[31::0]	000	000	
txdata[7::0]	FF	FF	FF
pr_stat[3::1]	1	1	0
digest_vld	1	1	
wr	1	1	
rd	1	1	
rddatacnt_en	0	0	
rdcnt[4::0]	00	00	1E
rdcntdatacnt[4:	00	00	00 10 08
FRND	0	0	
A[31::0]	000	000	01B5B97C
B[31::0]	000	000	78BB57C5
c[31::0]	000	000	7A9EF518
D[31::0]	000	000	79AABE16
E[31::0]	000	000	FBBFADE6

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (f): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

	Active	Trigger Condition Name	Trigger Condition Equation
Add Del	●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position: 0

Storage Qualification: All Data

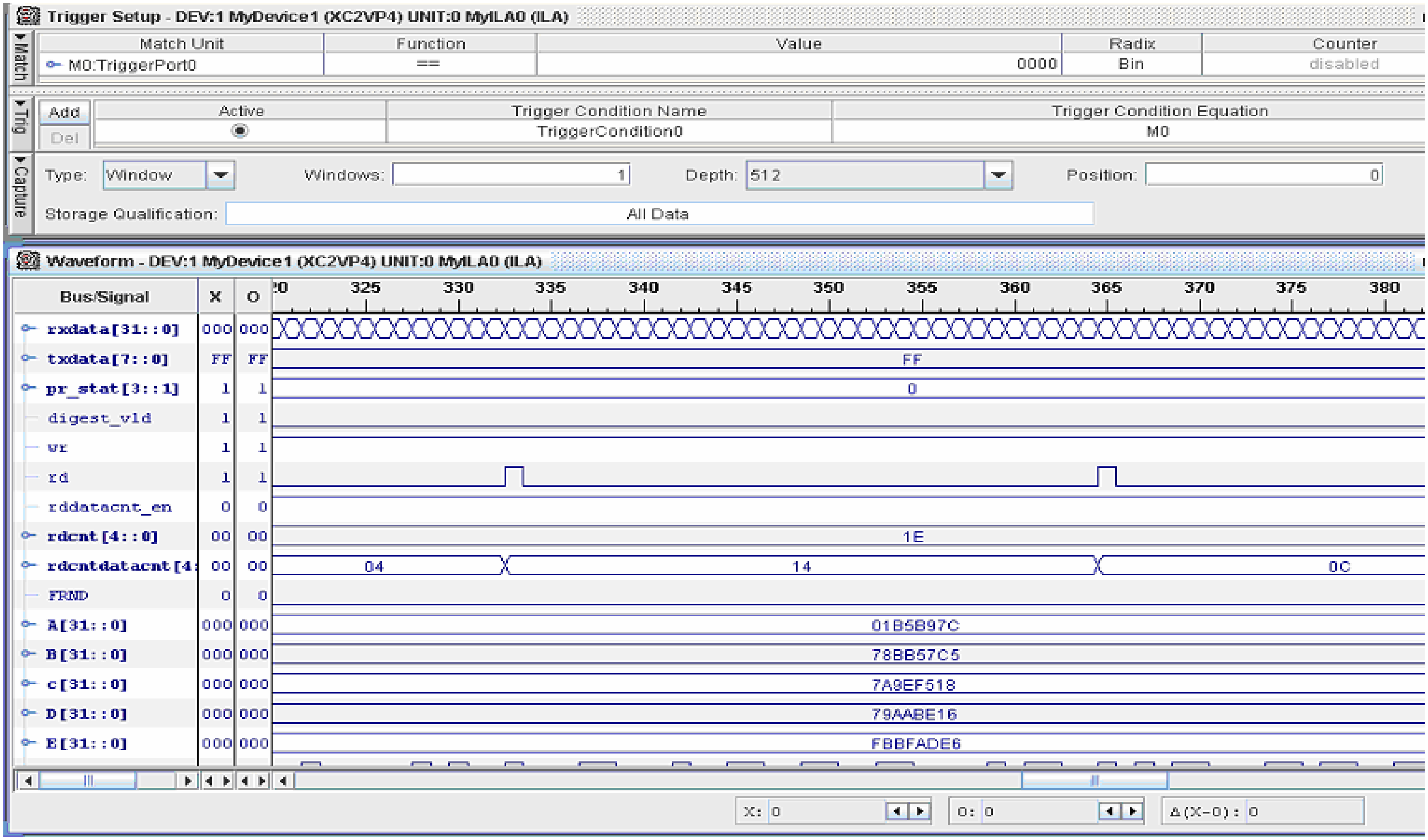
Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
rxdata[31::0]	000	000
txdata[7::0]	FF	FF
pr_stat[3::1]	1	1
digest_vld	1	1
wr	1	1
rd	1	1
rddatacnt_en	0	0
rdcnt[4::0]	00	00
rdcntdatacnt[4:	00	00
FRND	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000

D[31::0]=79AABE16 Window=0 Sample=288

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (g): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

	Active	Trigger Condition Name	Trigger Condition Equation
Add	●	TriggerCondition0	M0
Del			

Type: Window Windows: 1 Depth: 512 Position: 0

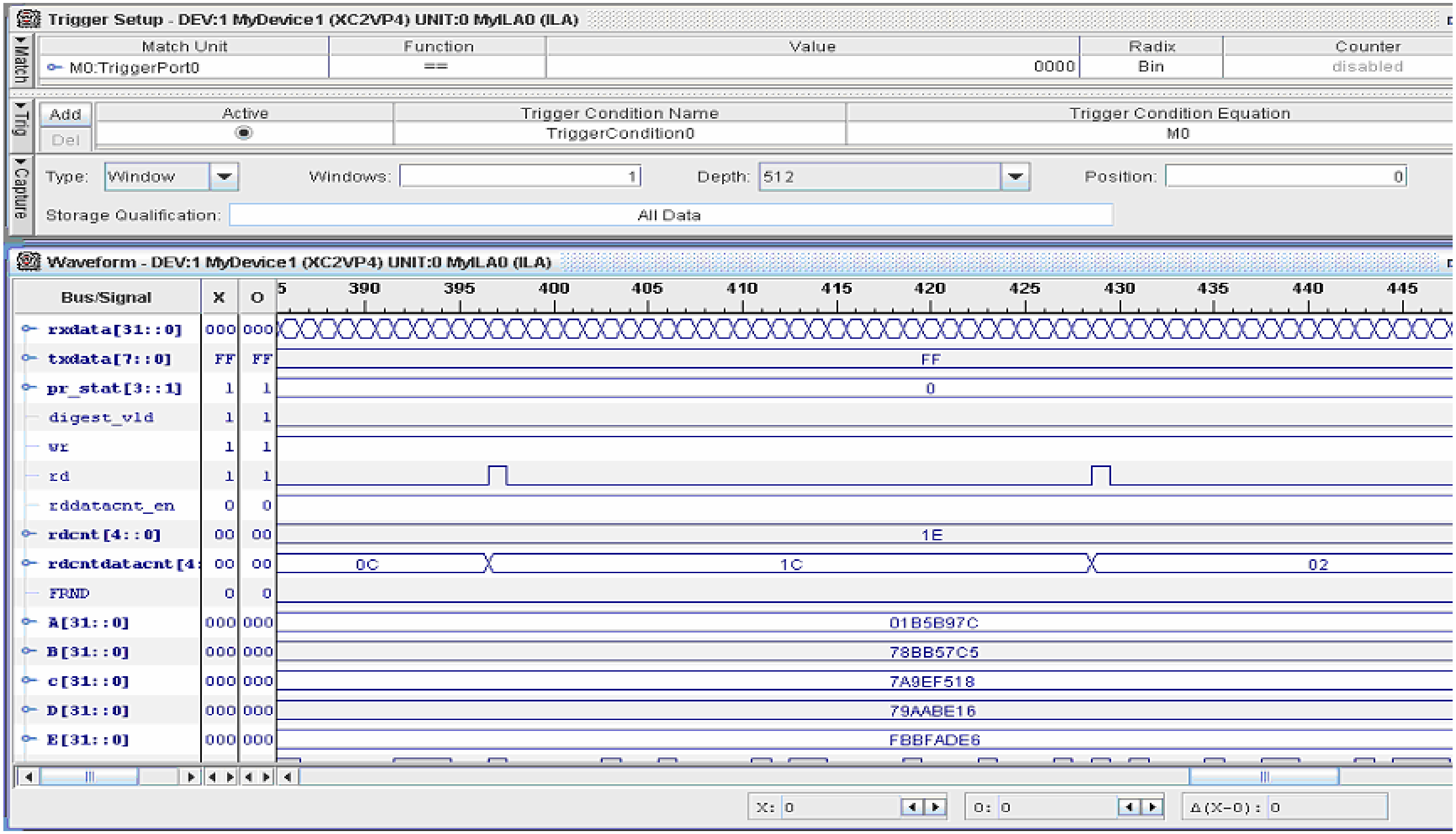
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O
rxdata[31::0]	000	000
txdata[7::0]	FF	FF
pr_stat[3::1]	1	1
digest_vld	1	1
wr	1	1
rd	1	1
rddatacnt_en	0	0
rdcnt[4::0]	00	00
rdcntdatacnt[4:	00	00
FRND	0	0
A[31::0]	000	000
B[31::0]	000	000
c[31::0]	000	000
D[31::0]	000	000
E[31::0]	000	000

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (h): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position: 0

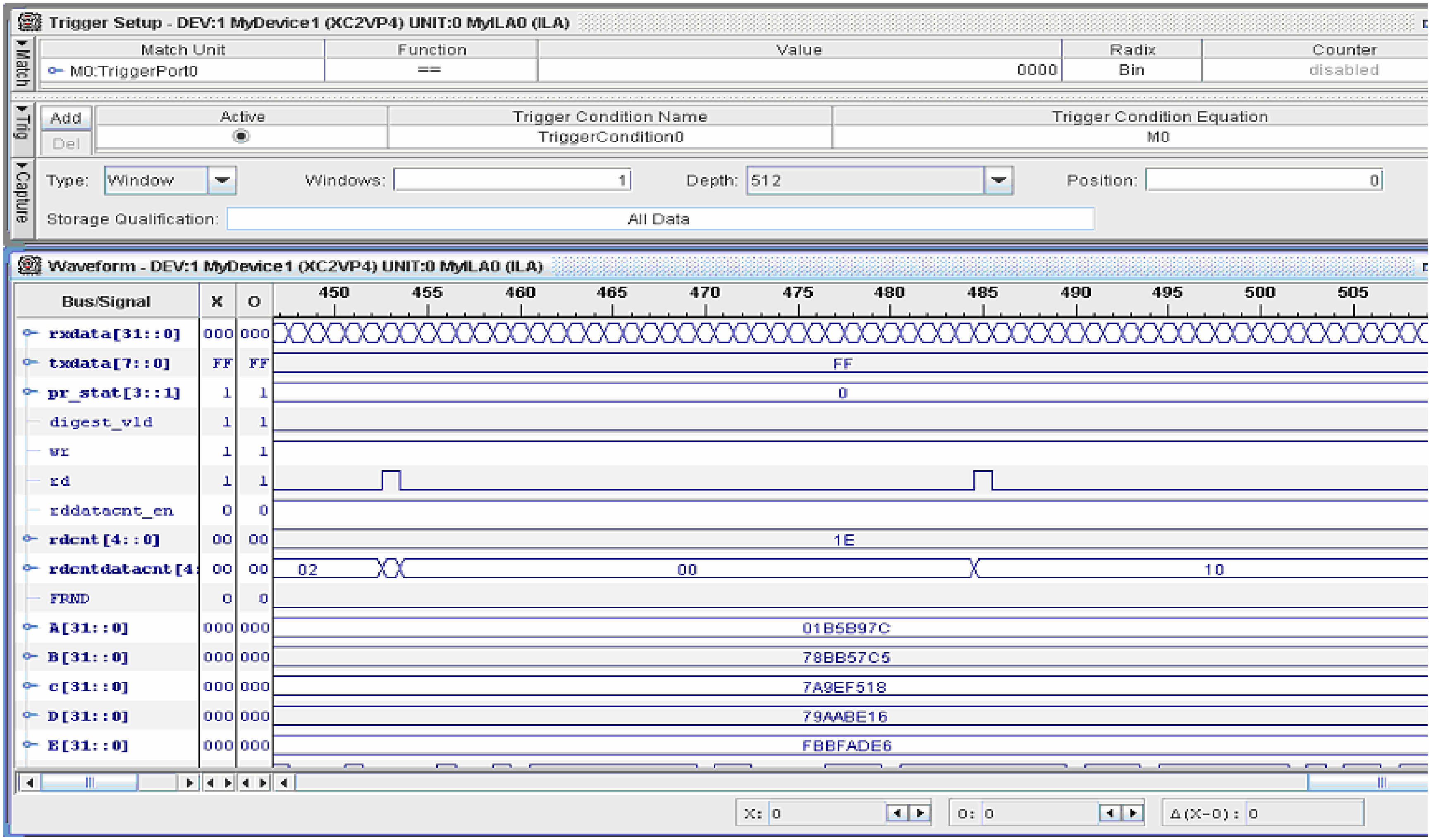
Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform
rxdata[31::0]	000	000	
txdata[7::0]	FF	FF	FF
pr_stat[3::1]	1	1	0
digest_vld	1	1	
wr	1	1	
rd	1	1	
rddatacnt_en	0	0	
rdcnt[4::0]	00	00	1E
rdcntdatacnt[4:	00	00	0C 1C 02
FRND	0	0	
A[31::0]	000	000	01B5B97C
B[31::0]	000	000	78BB57C5
C[31::0]	000	000	7A9EF518
D[31::0]	000	000	79AABE16
E[31::0]	000	000	FBBFADE6

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (i): Execution of IFFTEST



Trigger Setup - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Match Unit	Function	Value	Radix	Counter
M0:TriggerPort0	==	0000	Bin	disabled

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 512 Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP4) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	O	Waveform
rxdata[31::0]	000	000	
txdata[7::0]	FF	FF	FF
pr_stat[3::1]	1	1	0
digest_vld	1	1	
wr	1	1	
rd	1	1	
rddatacnt_en	0	0	
rdcnt[4::0]	00	00	1E
rdcntdatacnt[4:	00	00	02 00 10
FRND	0	0	
A[31::0]	000	000	01B5B97C
B[31::0]	000	000	78BB57C5
C[31::0]	000	000	7A9EF518
D[31::0]	000	000	79AABE16
E[31::0]	000	000	FBBFADE6

X: 0 O: 0 Δ(X-O): 0

Figure 7.3 (j): Execution of IFFTEST

Chapter 8

Conclusion

This thesis has presented the Identification Friend or Foe method as the framework for creating a secure authentication system for the embedded system applications. It is shown that the Identification Friend or Foe method behaves like a secure wrapper around the user design and protects it from the leakage of the algorithm details of the device. The IFF concept is challenge and response based authentication scheme that protects the Intellectual Property from cloning threat. This scheme is capable of securing a variety of embedded applications.

As the IFF test takes up a little space in the FPGA so application can be designed complex. The DS2432 is a Low Pin Count Device so it can be accommodated on the board very conveniently. The prediction of the algorithms and data interfaces will be very difficult when the embedded application will be paired with the IFF scheme. The embedded applications, which are under the scrutiny of competitive and hostile entities, will be well protected when paired with the IFF Scheme. This Scheme can used with other advanced block cipher algorithms and is capable to keep up with new security threats and for use in more advanced FPGA architecture.

This work has been implemented using Virtex-II Pro and DS2432 for the use with the embedded applications. Further research will continue to replace SHA-1 algorithm to find better method of securing the embedded applications. The strength of IFF Scheme and its ability to adapt other new encryption algorithms gives it a bright future and a wide variety of applications.

Bibliography

[1] Dai Zibin Zhou Ning, "FPGA Implementation of SHA-1 Algorithm**"**, *Institute of Electronic Technology, Information Engineering University Zhengzhou*, P.R China

[2] Aamer Nadeem, Dr M. Younus Javed, "Encryption Algorithms*", National University of Sciences and Technology Rawalpindi, Pakistan*.

[3] T Morkel 1, JHP Eloff 2*,* **"** Encryption Techniques: A Timeline Approach**",** *Information and Computer Security Architecture (ICSA) Research Group*

[4] "Different types of Encryption", https://www.wikipedia.org/wiki

[5] "Secure Hash Standard", National Institute of Standards and Technology, *Federal Information Processing Standards Publication 180-1, Secure Hash Standard*, 1995

[6] Maxim/Dallas Semiconductor Corporation, Dallas, Texas, "Why are 1-Wire SHA-1 Devices Secure", Application Note 1098:, http://www.maxim-ic.com/an1098, Jun 07, 2002

[7] Catalin Baetoniu and Shalin Sheth , "FPGA IFF Copy Protection Using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs", XAPP780 (v1.0) August 17, 2005

[8] Maxim/Dallas Semiconductor Corporation, Dallas, Texas, "DS2432 1K-Bit Protected 1-Wire EEPROM with SHA-1 Engine", http://www.maxim-ic.com (Datasheet)

[9] Jonathan Peter Graf, "A Key Management Architecture for Securing Off-Chip Data Transfers on an FPGA", M.S. Thesis, *Virginia Polytechnic Institute and State University*, June 18, 2004

[10] " Embedded System going Forward ", Maxim/Dallas Semiconductor Corporation, Dallas, Texas, Enginering Journal, Vol. 59, pp 10-13.

List of Figures

List of Tables