IMPLEMENTATION OF SECURITY WITHIN GEN2 PROTOCOL

by

SHESH KUMAR JAGANNATHA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2010

To my parents, sister and

all my friends !!!!

ACKNOWLEDGEMENTS

I am grateful to my research advisor, Dr. Daniel Engels who has been my guide, inspiration and support throughout my course work. His words of motivation totally changed my approach towards life which helped me achieve my goal. I am also thankful to Dr. Stephen Gibbs, for all his encouragement and support throughout my course work. I am grateful to Dr. William E. Dillon for supporting me and agreeing to be a part of my thesis committee. Also I would like to thank Stanley Howard and Bob Combs for supporting me and allowing me to work in their lab.

I am indebted to my family, my Mother Vedavathi, my Father Jagannatha and my Sister Lakshmi for believing in me and encouraging me throughout my life. I would also like to thank Sai for helping me climb the initial steps of my thesis and guiding me to achieve my goal. Finally, I would like to thank Gaurov, Amit, Tanvi, Kirti, Pranav and Jason for all the memorable fun moments and serious discussions we have had which will be etched in my memories for the rest of my life.

April 19, 2010

ABSTRACT

IMPLEMENTATION OF SECURITY WITHIN GEN2 PROTOCOL

SHESH KUMAR JAGANNATHA, M.S.

The University of Texas at Arlington, 2010

Supervising Professor: Daniel W. Engels

In this thesis, I propose and evaluate several methods to integrate security within the identification process and memory access processes of the ISO 18000-6C (or GEN2) passive UHF radio frequency identification (RFID) protocol. The RFID Tags are promiscuous and they emit their unique identifiers in clear upon interrogation from the Reader. They do not have a mutual authentication process to validate the Readers and Tags, this allows eavesdropping, illicit reading from a malicious Reader and counterfeit Tags. GEN2 Tags are being integrated into driver's licenses and being used for border crossing identities; thus, there is a need to secure both the identification process and memory access. To mitigate these threats, I integrate the strong cryptographic ciphers AES, PRESENT and XTEA into the GEN2 protocol and investigate their properties in secure identification and mutual authentication process. The mutual authentication protocol also enables a secure communication channel for the Readers and the Tags to communicate.

An FPGA implementation of the GEN2 protocol integrated with the ciphers are done using VHDL and a comparison is made in terms of area, speed and power consumption. The simulation results show that integration of PRESENT and XTEA

within GEN2 allows faster operation as compared to AES. PRESENT occupies the least hardware resources and has lower power consumption compared to AES and XTEA. The GEN2 protocol can be secured with strong cryptographic ciphers with minimal impact on identification rate and memory access.

TABLE OF CONTENTS

Appendix

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Standing on the edge of new communication era, we have instant access to the information from a wide variety of sources through the Internet. The Internet has become an integral and intimate part of everyday life for millions of people which has solved the problem of sharing information. There are many solutions for capturing information about an object like data entry, barcode scanning. But these are manual solutions which are costly, time consuming and sometimes inaccurate. These issues limit the performance of the optical barcode system. Radio Frequency Identification(RFID) Systems are potential solutions which lacks the flaws of optical barcodes.

A typical RFID system consists three main components: Tags, Readers and back-end database system. The Tags are placed on the items that need to be identified and the Readers are placed to interrogate the Tags for the required information. RFID systems allows wireless powering of the tags and have potential to be very low cost and does not need line of sight communication. A Passive UHF RFID Tag harvests the energy from the signal transmitted by the Reader, demodulates and decodes the command and then backscatters the response back to the Reader. A typical UHF RFID system operates in the ISM band (902 - 928 MHz in the United States) and most of the passive UHF RFID Tags use the GEN2 protocol to communicate with the Reader. The GEN2 protocol was originally developed under the Auto-ID Center and completed under EPCglobal [23]. The different applications of RFID

systems include automatic highway tolling, access control, airline baggage handling, inventory management and asset tracking, smart cards, automatic automobile locking and security system.

1.2 Security and Privacy issues

With the increased adoption of the RFID technology in industry as well as day to day life, because of its low cost and the convenience in identifying objects without LOS, there are lot of security and privacy issues. For example, unauthorized access to tags memory content, tag forgery, consumer tracking, theft protection and snooping [26] are some of the major threats to security. In the RFID enabled credit cards which has no security implemented, it will be very easy for unauthorized readers to access the information in the Tag without the knowledge of the owner. This can prove to be major threat to the security of personal information of the owner. Even if Tag's contents are protected, individuals may be tracked through predictable tag responses essentially a traffic analysis attack violating location privacy. Introducing a counterfeit Tag into the supply chain which has a same ID as a legitimate Tag, results in a confusion in the inventory process. To mitigate these security and privacy issues, a secure identification and mutual authentication protocol is required for a Tag and Reader to have a secure communication.

Researchers have done a good deal of work to minimize security and privacy threats by implementing electronic security using protocols. For example Feldhofer et al.[1] presented a strong authentication scheme using a challenge response method. This technique uses Advanced Encryption Standard (AES) as cryptographic primitive to achieve security. S A Weis et al. [59] proposed a Hash Lock approach where a authentication key could be used to send tag in locked state and the same key is used to bring it back to unlocked state.

But to the best of our knowledge there is no attempt made to secure the identity (EPC/Object-ID) of the Tag, which enables tracking of the tagged objects. Also a secure communication channel is not established between the Tag and the Reader to exchange the data. This can prove to be a serious threat to privacy in some applications.

1.3 Overview of GEN2 Protocol

The passive UHF RFID technology is widely used because of its low cost and ease of availability which makes it affordable to the common man to use in daily life. The GEN2 protocol which is commonly used in most of the passive Tags do not have mutual authentication between RFID Tags and the Readers to have secure communication. Also the unique identifiers transmitted by the Tags are not secure which may be a threat to consumer privacy.

An RFID Tag has four major components viz. Antenna, RF front end (receiver and transponder), Physical layer (encoder and decoder) and Tag Identification layer. To make the RFID GEN2 communication secure, security should be integrated within Tag identification layer. In my Thesis, I concentrate on the implementation of the Tag ID layer in hardware using VHDL. The high level architecture of the design is made of seven major modules as given below.

1. Command Detection module
2. CRC Engine
3. Finite State Machine (FSM)
4. Random Number Generator
5. Slot Counter
6. Response Module
7. Memory Module

To secure the data communicated by the Reader and the Tag, cryptographic algorithms needs to be implemented within GEN2 protocol. Hence a significant research on lightweight implementations of ciphers are done which are suitable for resource constrained applications like RFID. In my Thesis I have considered three encryption algorithms PRESENT, XTEA and AES which is implemented in hardware using VHDL and integrated with GEN2 protocol to achieve a Secured GEN2 Protocol.

1.4 Secured GEN2 Protocol

In this thesis, I developed protocols to implement security in passive RFID Tags which secures the identity of the Tag and performs a mutual authentication before revealing the memory contents of the Tag. A slight modification in the GEN2 protocol for the state transition of a Tag is proposed which enables a secure communication channel for the Reader and the Tag to exchange data in the Secured state of the Tag. The protocol uses cryptographic techniques to achieve security and provides flexibility in choosing the cipher.

1.4.1 Secure Identification Protocol

In the normal tag identification process of the GEN2 protocol the EPC bits backscattered by the Tag is insecure and it can be very easily captured by malicious readers which will give access to the details of the tagged object. To make the communication secure, information being transmitted by the Tag to the Reader in the inventory process can be randomized. This novel approach will confuse the malicious readers by sending different information every time by the same Tag.

There are three different designs that I have implemented to achieve secure identification.

- Design A: In this design a 48-bit random number along with its CRC-16 is encrypted and transmitted to the Reader as a response to the ACK command. The Reader will decrypt this information using all the keys in its database and selects the key that yields correct CRC value. The key in turn points to the object in a database.
- Design B: In this design a 64-bit random number (secureID) is transmitted along with its encrypted value as a response to the ACK command. The Reader will decrypt a part of this information using all the keys in its database and validate the value to get the right key. The key in turn points to the object in a database.
- Design C: In this design the encryption is done on the power-up of the Tag. Here the cipher engine will run in the CBC mode which gives more security as compared to the block mode.

1.4.2 Mutual Authentication Protocol

In this process a challenge-response technique is used to achieve mutual authentication of the Tag and the Reader. By doing this a forged tag cannot convince the reader of its authenticity.

There are two different designs that I have implemented to achieve mutual authentication.

- Design 1: After the secure identification process of the Tag, it enters the open state with a *Req_RN* command. The Reader then issues a challenge which is a double encrypted value of secureID. The Tag validates the Reader and encrypts the secureID again and backscatters to the Reader. The Reader then validates the Tag. After the mutual authentication process the Tag enters the secured state to enable secure communication channel.

- Design 2: In this design, immediately after the secure identification process of the Tag, the Reader requests a new RN64. On receiving this, the reader decrypts this along with a new RN64 and sends it as a challenge. After receiving the challenge from the Reader, the Tag authenticates the Reader successfully and responds by encrypting the a part of the challenge along with a new RN64. The Reader in turn authenticates the Tags response. After the mutual authentication process the Tag enters the secured state to enable secure communication channel.

A detailed timing analysis of the simulation results shows that AES, PRESENT and XTEA when integrated with GEN2 protocol will satisfy the "T1" requirement of the GEN2 protocol. The synthesis results show that PRESENT is a good fit into the GEN2 protocol since it occupies less number of slices and other hardware resources and also consumes less power as compared to XTEA and AES.

1.5 Thesis Overview

The increase in use of passive UHF RFID systems in various applications has made security as an important aspect to be considered with the GEN2 protocol. Secure Identification and mutual identification protocol when integrated with GEN2 protocol, is capable of addressing the security and privacy issues, by securing the Tag's ID and also providing a mutual authentication between the Tag and the Reader. The security is provided by using strong lightweight cryptographic algorithms which secures the data being communicated and also provides a mutual authentication. When the security is integrated with GEN2 protocol, a timing analysis is done to verify the validity of T1 constraint specified in the GEN2 protocol.

The implementation of various designs in VHDL is discussed and a detailed analysis of constraints in terms of speed, area, throughput and power consumption is presented.

Chapter 2 presents the contributions from various researchers to improve the security in RFID system and their advantages and disadvantages which highlights the open areas of research.

Chapter 3 presents an overview of the GEN2 protocol and the idea behind choosing Tag Identification layer to integrate security. It briefly discusses about the different states the Tag shall implement and associated state transitions for different commands from the Interrogator.

Chapter 4 deals with the design and implementation of Tag Identification layer in VHDL. It also gives a detailed explanation of various modules in TID layer, functionalities and the pin descriptions of different modules for Implementation in VHDL.

Chapter 5 deals with the cryptographic algorithms used to integrate security with the GEN2 protocol. It gives a detailed explanation of each algorithm and discusses the design and implementation of each algorithm in VHDL. A comparison and analysis of the results are also presented.

Chapter 6 deals with the integration of secure identification and mutual authentication protocol with GEN2. The various designs of secure ID and mutual authentication protocol is presented and detailed security analysis of the designs is discussed.

Chapter 7 analyzes the performance of various implementations in terms of speed, area and power consumption. The implementation shows that Present consumes least hardware resources and consumes low power as compared to XTEA and AES.

Chapter 8 summarizes the contributions and presents the future work which can be potential open areas of research. Thus, the GEN2 protocol can be secured with strong cryptographic ciphers with minimal impact on identification rate and memory access.

CHAPTER 2

RELATED WORK

2.1 Introduction

A variety of techniques can be used to make the RFID Tags secure. A suitable solution was found where the RFID tags can become smarter by adding the security features to the Tag itself. Majority of the recent work by the research community to add cryptographic security addresses the issues of reader authentication and securing the Tag's identity. S.A.Weis proposed the Hash lock approach proposed in his paper [59] to achieve authentication. Sangshin Lee et al. [34] proposed a mutual authentication scheme based on synchronized secret information. Juels and Pappu [28] proposed a technique to have privacy protection in Banknotes by securing the ID of the Tag.

2.2 Previous work on Authentication protocols for Tags

2.2.1 Hash-Lock Approach

Weis along with Sarma, Rivest and Engels suggested Hash-lock mechanism as access control and a randomized version of it to deal with consumer tracking [59]. In this design some portion of memory in the tag is reserved for a meta-ID. The tag can be in either locked or unlocked state. To lock the tag, the owner will compute the hash value of a key which will be saved in the memory of the tag reserved for meta-ID.When the tag is in locked state, it responds with the meta-ID value for all queries from the reader. To unlock the tag owner can send the original key value. Problems with the approach:

- Major flaw being a discovery of meta-ID and label-ID pair will result in spoofing [45].
- No implementation results.
- Do not have secure Identification.
- No secure communication channel.

2.2.2 Lightweight Authentication Protocols for Low cost Tags

In this paper they propose an extremely light weight challenge-response authentication protocols for authenticating the tags. Since the resources like computing power, storage and communication capabilities are limited in a low cost tag, instead of using the standard cryptographic primitives and hash function other protocols like XOR, subset, squaring, RSA and knapsack are proposed to achieve the privacy [57].

Problems with the approach:

- No strong security
- No secure Identification
- No secure comm. channel

2.2.3 Strong Authentication using AES

A strong symmetric authentication scheme which is suitable for low power and low die-size requirements is proposed and implemented in this paper [12]. Here Advanced Encryption Standard (AES) is the cryptographic primitive used and a novel approach of implementation of AES hardware is presented which encrypts 128-bit block data in 1016 clock cycles and has a power consumption of less than 9μA on a 0.35μm CMOS. Here by authentication is achieved by using the challenge-response method and by doing this consumer tracking can be avoided if the tags reply only

to authorized readers, and only authorized readers can read from or write to tag's memory.

Problems with the approach:

- Did not provide Secure Identification
- No mutual authentication
- No secure communication channel

2.2.4 Challenge-Response based Authentication

Keunwoo Rhee et al. [46] proposed a Challenge-Response based authentication protocol using hash function and random number which is secure against common attacks like replay attack and spoofing attack. Here the back-end database stores the ID and other information of the Tags. When the reader queries the tag with a a random number the tag responds using this random number and a random number generated by itself. The reader authenticates the tag by hash function computations.

Problems with the approach:

- The ID of the Tag is not updated
- No secure identification
- No secure channel

2.2.5 LMAP - Lightweight Mutual Authentication Protocol

Pedro Peris-Lopez et al. [41] proposed a real lightweight mutual authentication protocol for low-cost RFID tags using LMAP. The protocol first provides a secure tag identification by sending the index pseudonym in response to the query from reader. It then provides a mutual authentication based on exchange of two messages between the tag and the reader. After the mutual authentication the index-pseudonym and the key updating stage must be carried out. He has also proposed a light weight

minimalist mutual authentication protocol [39]. A security analysis and comparison of LMAP implementation with M^2AP implementation is done by Tieyan Li et al [35].

Problems with the approach:

- Less secure as compared to cipher implementations
- No secure comm. channel

2.2.6 ECC Processor with Low die size

This technique presents a design of a special purpose processor with Elliptic Curve Digital Signature Algorithm(ECDSA) functionality. This is designed to work as digital processor in RFID tag and provide secret key storage and PRNG. The asymmetric cryptosystem allows authentication of the tag to untrusted third parties without revealing the secret key. The GF(p192) version will need 23K gate equivalents or 1.3mm2 for a 0.35μ m process. 502K clock cycles are used for signature generation [16].

The asymmetric scheme of encryption which involves RSA, elliptic curve implementations required tens of thousands of gates which makes it entirely infeasible to implement in a tag. Even storing of the RSA keys will be an issue with limited memory available in tags [48].

2.3 Previous work on securing Tag ID

2.3.1 Re-encryption approach

Juels and Pappu [28] proposed a technique to have privacy protection in Banknotes. This used a simple cryptographic design which does the public key encryption of the serial numbers in RFID tags with a corresponding private key stored appropriately by a law enforcement agency. These cipher texts are subject to re-encryption

by computational devices in shops, thereby rendering multiple appearances of a given RFID tag unlinkable. Problems with the approach:

- No Mutual authentication.
- No secure communication channel.

2.3.2 Universal Encryption approach

Golle, Jakobsson, Juels, and Syverson [18] on the same idea used the universal encryption technique where re-encryption is possible without knowing about the public key. Thus the public key in the tag need not be made readable, protecting the tag from being tracked. Hence it provides an extra security and privacy by permitting agents to perform re-encryption without public key.

Problems with the approach:

- No Mutual authentication.
- No secure communication channel.

2.3.3 Hash-based Enhancement of Location Privacy

Using the same idea used in "Hash lock" a scheme that provides location privacy along with the data privacy is proposed [22]. The idea here is to change the ID of the tag on every read attempt in a secure manner. In this scheme the one-way hash lock functions and use of the random values generated at the back-end so that the tracking numbers themselves cannot be used for tracking purposes.

Problems with the approach:

- Security level is less compared to cryptographic implementations.
- No Mutual authentication.
- No secure communication channel.

2.3.4 Privacy-Friendly Tags

Miyako Ohkubo et al. [33], [37] proposed a privacy protection scheme to achieve forward security. Here they use a hash chain technique to renew the secret information contained in Tag. In the i-th transaction with the reader, the RFID tag sends a hash value of secret information and renews it to hash value of previous secret. Thus it achieves indistinguishability and forward security of the information.

Problems with the approach:

- No Mutual authentication.
- No secure communication channel.

2.3.5 Blocker Tag

Juels proposes Blocker Tag as a cheap passive RFID device for privacy protection [29]. The blocker tag uses the structure of tree-walking algorithm and simulate many RFID tags simultaneously to successively accomplish blocking. The blocker tag can simulate the complete set of 2^k combinations of RFID tag serial numbers and forcing the collision causing the reader to explore entire tree. The blocker tag can do selective blocking by simulating only selected subset of IDs. Problems with the approach:

- Data being transmitted is still insecure.
- No secure Identification.
- No secure communication channel.

2.4 Summary

In all the above techniques discussed, a secure communication channel is not established between the Reader and the Tag to exchange data. Also the protocols developed to make the identification secure was implemented on the top of GEN2

protocol. Also authentication is achieved only from the Reader's side and there is no mutual authentication between the two entities. In my thesis, I propose to make the RFID system more secure by implementing the security and privacy features within the protocol by securing the data that is communicated in the identification process and also have a mutual authentication to enable a secure communication channel.

CHAPTER 3

ISO 180006-C GENERATION-2 PROTOCOL

3.1 Introduction

Class-1 Gen-2 protocol was originally developed under the Auto-ID Center and completed under EPCglobal [23]. This protocol is implemented in RFID systems which is widely used in case and pallet level retail supply chain applications. It is also used in many other applications like airline baggage handling, building access tags, asset tagging etc. This is an Interrogator-talks-first protocol where an interrogator transmits information to the tag by modulating an RF signal in the 860 MHz to 960 MHz frequency range [54]. The tag receives both the information and operating energy from the RF signal. The tags being passive receives energy from the RF signal and reply to the reader at the same frequency. The radiated power from the reader is according to FCC regulations 15.247 [38], the intentional radiators that employ frequency hopping or direct sequence spread spectrum techniques that they're allowed 1W of transmitter power combined with a maximum allowed antenna gain of 6dB, makes 4W EIRP. Also the FCC regulations say that the reader will have to hop the frequency band every 200msec.

The GEN2 protocol has two layers of operation viz the physical layer and tag-identification layer.

3.1.1 Physical Layer

A Reader sends information to one or more Tags by modulating the RF carrier using double-sideband amplitude shift keying (DSB-ASK), single-sideband amplitude

shift keying (SSB-ASK) or phase-reversal amplitude shift keying (PR-ASK) using a pulse interval encoding (PIE) format [23]. Tags being passive receive their operating energy from this same modulated RF carrier. Tags communicated by backscatter modulating the amplitude and/or phase of the RF carrier. The encoding format, selected in response to interrogator commands, is either FM0 or miller-modulated subcarrier. The communications link between the reader and tags is half-duplex, which means that at a time only a reader or tag can talk.

3.1.2 Tag Identification Layer

A reader manages the tag population using the following three basic operations.

1. **Select**: This does the operation of choosing a Tag population for inventory and access. A select command can be used to select a particular tag population based on user specified criteria. This operation is analogous to selecting records from a database.
2. **Inventory**: This is the operation of identifying Tags. An Interrogator begins an inventory round by transmitting a Query command in one of the four sessions. One or more tags may reply. The Interrogator a single Tag reply and requests the PC, EPC and CRC from the Tag. The Inventory process includes more than one commands and it operates in one and only one session.
3. **Access**: The operation of communicating with reading from or writing on to a tag. Access also consists of multiple commands [23].

To make the RFID GEN2 protocol secure, security should be integrated within Tag identification layer of the Tag. Hence in my Thesis, I concentrate on the implementation of the Tag ID layer in hardware using VHDL. This chapter briefly discusses about the different states the Tag shall implement and associated state transitions for different commands from the Interrogator. The logical memory structure of the Tag,

different sessions of a Tag and CRC engine implemented by the Tag are also discussed which are the key modules to be implemented on hardware to integrate security.

3.2 Tag Memory

A Tag memory is logically separated into four distinct banks. The logical memory map is shown in the Figure 3.1. The memory banks are

3.2.1 Reserved Memory

The logical address of reserved memory is (00_2). It shall contain kill and/or access passwords, if they are implemented on Tag. The kill password will be stored at the memory addresses 00_h to $1F_h$. and the access password at the memory address 20_h to $3F_h$.

3.2.2 EPC Memory

The logical address of EPC memory is (01_2). It shall contain a StoredCRC at memory addresses 00_h to $0F_h$, a StoredPC at addresses 10_h to $1F_h$ and EPC that identifies the object to which the Tag is attached beginning at address 20_h.

3.2.3 TID Memory

The logical address of TID memory is (10_2). It shall contain an 8-bit class identifier from 00_h to 07_h, 12-bit task mask-designer and higher functionality tag may contain tag and vendor specific data.

3.2.4 User Memory

The logical address of User memory is (11_2). This space allows specific data storage.

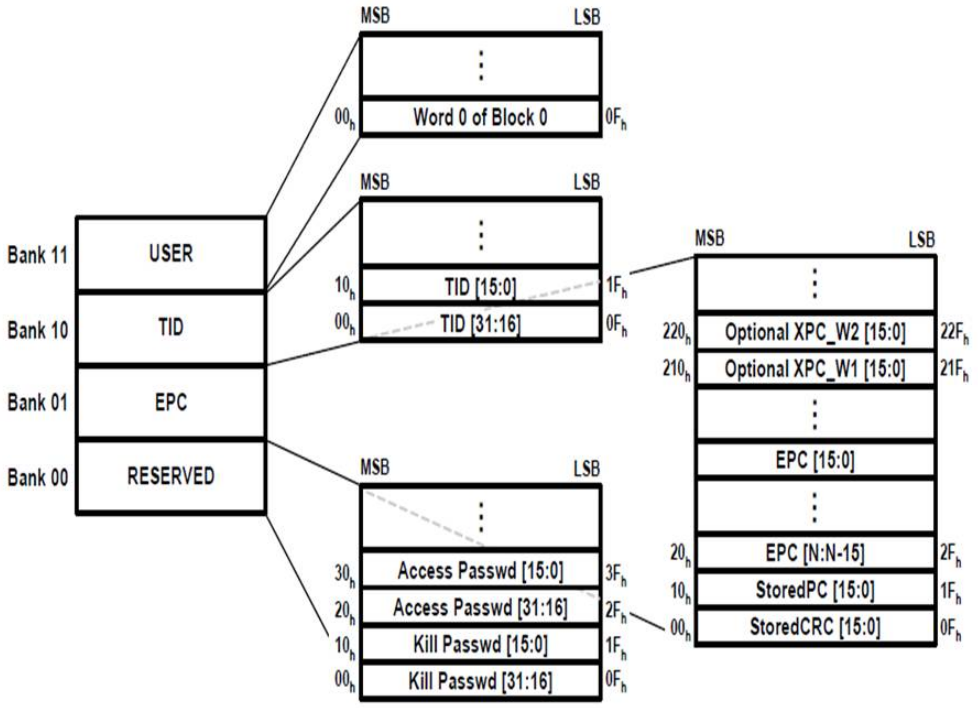

Figure 3.1. Logical Memory Map [23].

3.2.5 CRC-16 (StoredCRC and PacketCRC)

All Tags shall implement StoredCRC and the Tags that support XPC functionality shall also implement a PacketCRC. At power-up a Tag shall calculate a CRC-16 over StoredPC and EPC specified by EPC length field in the StoredPC and shall map the calculated CRC-16 into EPC memory 00_h to $0F_h$, MSB first. The Tag calculates the PacketCRC dynamically over the backscattered PC word, optional XPC word, and EPC. Tags that do not support XPC functionality need not implement PacketCRC.

3.2.6 Protocol-control(PC) word (StoredPC and PacketPC))

All Tags shall implement a StoredPC whose fields are EPC length, UMI, XI and NSI. Tags that support XPC functionality shall also implement a PacketPC that

differs from StoredPC in its EPC length field. The StoredPC will be located in EPC memory at addresses 10_h to $1F_h$.

3.3 Sessions, Inventoried flags and Selected flag

Interrogators shall support and Tags shall provide four sessions (S0, S1, S2 and S3). Tags shall participate in one and only one sessions during an inventory round. Tags shall have inventoried flags for each session called A and B. At the beginning of every inventory round an Interrogator chooses A or B Tags in one of the four sessions. Tags in one session shall neither use or modify the inventoried flag of different session. All Tag resources are shared among sessions except the inventoried flag.

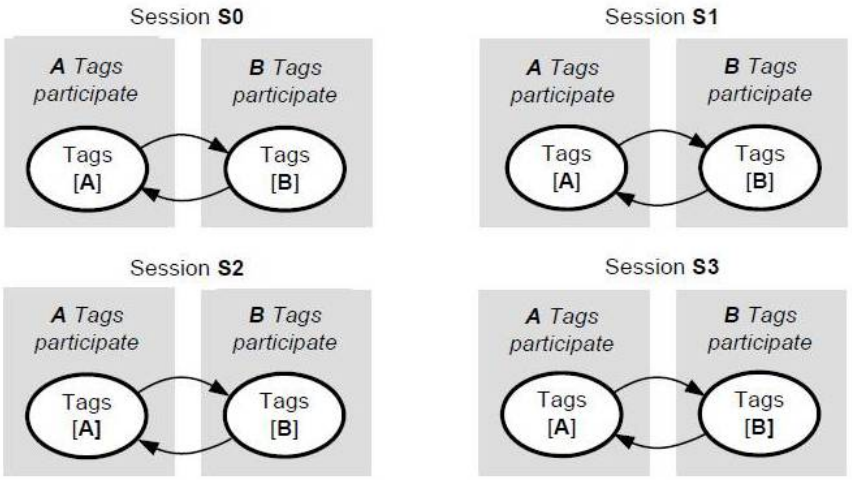

Figure 3.2. Session diagram [23].

Tags shall implement a selected flag SL, which an Interrogator may assert or deassert using a select command. The **Sel** parameter in the Query command allows Interrogator to inventory Tags that have SL wither asserted or deasserted or to ignore the flag and inventory Tags regardless of the SL value.

3.4 Cyclic-redundancy check (CRC)

The Tag uses a Cyclic-redundancy check to ensure validity of some Reader to Tag commands and the Interrogator uses it to ensure validity of certain backscattered replies. This protocol uses two types of CRC i.e. CRC-5 and CRC-16. The CRC-5 shall use a polynomial $x^5 + x^3 + 1$ with a 01001_2 preset value. The CRC-16 uses a $x^{16} + x^{12} + x^5 + 1$ polynomial with $FFFF_2$ preset value.

At power-up the tag calculates and saves into memory a 16-bit StoredCRC and during inventory a Tag can backscatter this storedCRC or a 16-bit PacketCRC that it calculates dynamically. The commands and the backscatter replies for which the CRC needs to be appended are defined in the protocol.

3.5 T1 requirement specification

The Figure 3.3 shows the R ⇒ T and T ⇒ R link timings. The Tags and Interrogators shall meet the requirements shown in Table 3.1. The transmission for all R ⇒ T and T ⇒ R communications within each message and each word shall be most-significant bit (MSB) first.

Inventory round of a single tag reply shown in Figure 3.3 shows the reader transmitting CW for a duration of T_4 which is minimum of 2RTcal. The Reader shall use a constant R⇒T link rate, if a change of R⇒T link rate is required to be changed then a CW for a minimum of 8RTcal is transmitted to indicate there is a change in data rate. When a Query is communicated with the tag the Tag sends RN16 after a time T_1 as shown. Reader waits for time T_2 and sends Ack command. Now the tag reply with a command consisting of PC, OID and CRC16. An QueryRep or QueryAdjust is sent by reader to acknowledge that OID is valid, else a NAK command is sent [23].

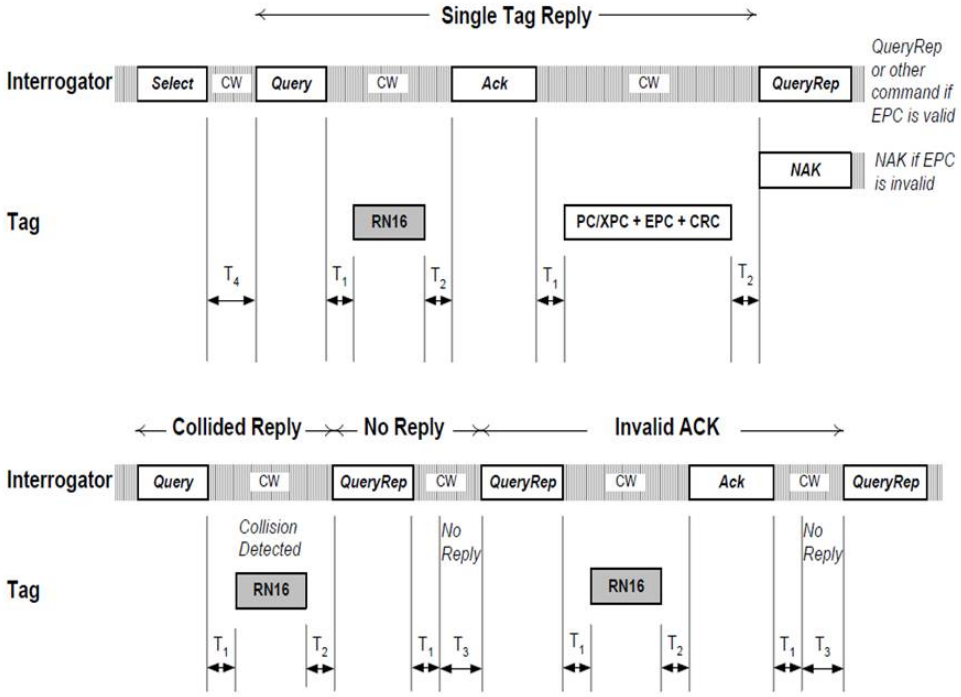

Figure 3.3. Link Timing [23].

Table 3.1. Link timing parameters [23]

Parameter	Minimum	Typical	Maximum	Description
T_1	$MAX(RTcal, 10T_{pri})$ $\times (1 - FT) - 2\mu$ s	$MAX(RTcal, 10T_{pri})$	$MAX(RTcal, 10T_{pri})$ $\times (1 - FT) + 2\mu$ s	Time from interrogrator transmission to tag response, measured at antenna terminal.
T_2	$3.0\mathrm{T}_{pri}$		$20.0\mathrm{T}_{pri}$	Time from last falling edge of the last bit of the Tag response to first falling edge of a Reader signal.
T_3	$0.0\mathrm{T}_{pri}$			Time an Reader waits, after T_1,before it issues another command
T_4	$2.0\mathrm{T}_{pri}$			Minimum time between Reader commands

3.6 Tag states and slot counter

3.6.1 Ready State

Tags upon entering an energizing RF field shall enter ready state if they are not killed. The Tag shall remain in ready until it receives a Query command whose

inventoried parameter (for the session specified in Query) and Sel parameter match its current flag values. Matching Tags shall determine the slot counter value and transition to arbitrate state if the number is nonzero, else to the reply state if the number is zero. If a Tag in any state except killed loses power it shall return to ready upon regaining power.

3.6.2 Arbitrate State

A Tag in arbitrate state shall decrement slot counter every time it receives QueryRep command whose session parameter matches the session for the inventory round. Tags shall transition to reply state and backscatter RN16 when slot counter reaches 0000_h and remain in same state if slot value is non zero.

3.6.3 Reply State

The Tag upon entering reply state shall backscatter an RN16. If the Tag receives an valid ACK command then it transitions to acknowledged state backscattering the $PC||EPC||CRC$ as reply. If it receives an Invalid ACK then the Tag returns to arbitrate state.

3.6.4 Acknowledged State

A Tag in acknowledged state can transition to any state except killed depending on the received command. If it receives a valid ACK then Tag shall re-backscatter $PC||EPC||CRC$ again.

3.6.5 Open State

A Tag in the acknowledged state whose access password is nonzero shall transition to open state upon receiving Req_RN command backscattering a new RN16

called handle. A Tag in open state can transition to any state except acknowledged state depending on the received command. If it receives a valid ACK then Tag shall re-backscatter $PC||EPC||CRC$.

3.6.6 Secured State

A Tag in the acknowledged state whose access password is zero shall transition to secured state upon receiving Req_RN command backscattering a new RN16 called handle. A Tag in open state whose access password is nonzero shall transition to secured state upon receiving valid Access command sequence. A Tag in secured state can transition to any state except open or acknowledged state depending on the received command. If it receives a valid ACK then Tag shall re-backscatter $PC||EPC||CRC$.

3.6.7 Killed State

A Tag in open or secured state shall enter killed stated upon receiving kill command sequence with a valid nonzero kill password. Upon entering killed state a Tag shall notify the Interrogator that the kill operation was successful, and shall not respond to Interrogator thereafter.

3.6.8 Slot counter

A Tag shall have a 15-bit slot counter which will be preloaded with a value between 0 and $2^Q - 1$ upon receiving a Query or QueryAdjust. Q is an integer in the range (0,15). A Query specifies Q and a QueryAdjust may modify Q. Tags in the arbitrate state decrement the slot counter every time it receives a QueryRep. Tags which are not acknowledged shall return to arbitrate with a slot value 0000_h.

The State transitions in the Tag on receiving different commands are shown in the Figure 3.4.

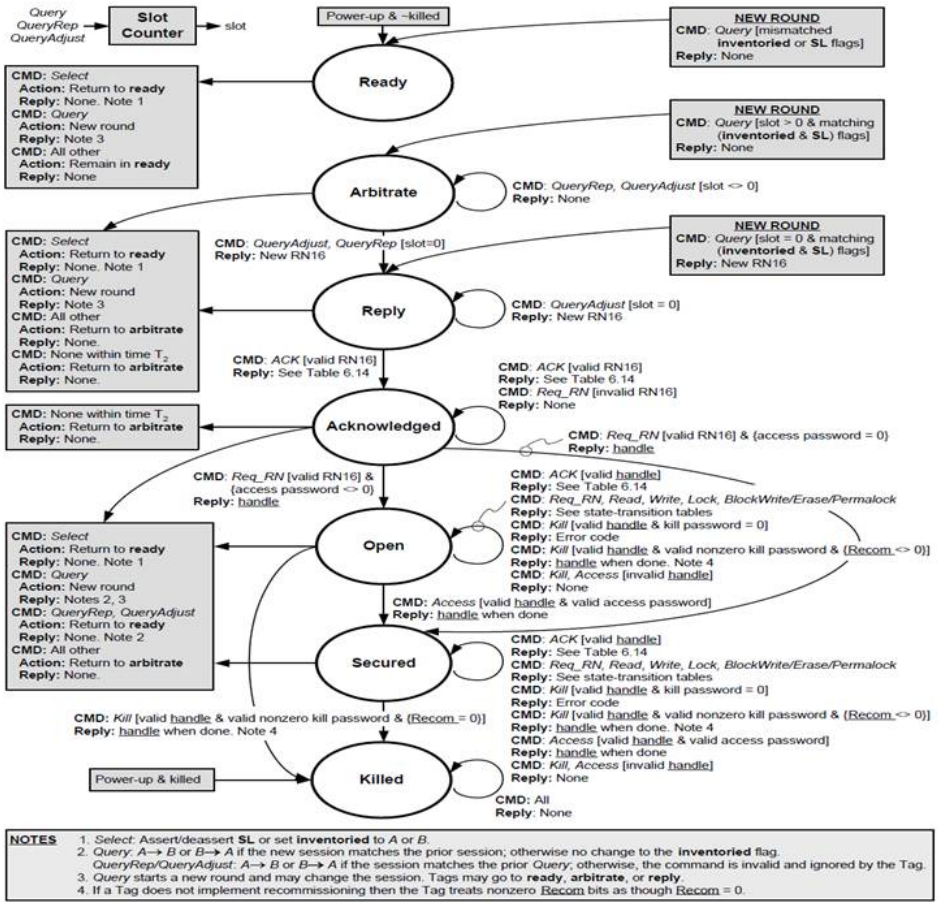

Figure 3.4. Tag state diagram [23].

3.7 Gen2 Commands

Interrogators manage Tag populations using the three basic operations and each of these operations comprises of one or more commands [23]. The three operations are

- **Select**: The process using which the Interrogator selects a Tag population for inventory and access.
- **Inventory**: The process using which the Interrogator identifies Tags. An inventory operates on one and only one session.
- **Access**: The process by which the Interrogator reads from or writes to individual Tags.

The different commands used in the above three operations are described briefly below.

3.7.1 Select (Mandatory)

The Select command implemented by the Reader/Tag selects a particular Tag population. The Select command comprises of the following bits as shown in Table 3.2 Based on the Target, Acton, MemBank, Pointer, Length and Mask fields the Tag matching and non-matching criteria is specified. The response to the Action parameters is as shown in Table 3.3

Table 3.2. Select Command [23]

	Command	**Target**	**Action**	**MemBank**	**Pointer**	**Length**	**Mask**	**Truncate**	**CRC-16**
# of bits	4	3	3	2	EBV	8	Variable	1	16
description	1010	000: **Inventoried (S0)** 001: **Inventoried (S1)** 010: **Inventoried (S2)** 011: **Inventoried (S3)** 100: **SL** 101: RFU 110: RFU 111: RFU	See Table Table 3.3	00: RFU 01: EPC 10: TID 11: User	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncation 1: Enable truncation	

Table 3.3. Tag response to action parameter [23]

Action	Matching	Non-Matching
00	assert SL or inventoried→ A	deassert SL or inventoried→ B
01	assert SL or inventoried→ A	do nothing
10	do nothing	deassert SL or inventoried→ B
00	negate SL or(A→B,B→A)	negate SL or(A→B,B→A)

3.7.2 Query (Mandatory)

The Query command initiates and specifies an inventory round. The Query command includes the fields as shown the Table 3.4. The Tag upon receiving a Query command with the matching *Sel* and *Target* parameters, will pick a random value in the range from 0 to 2^Q-1, and shall load this value to the slot counter. If the slot counter value is zero then the Tag responds with an RN16 as shown in Table 3.5 and transitions to reply state else the Tag shall remain silent by transitioning to arbitrate state. The DR (divide ratio) sets the T ⇒ R link frequency. *M* (cycles per symbol) sets the T ⇒ R data rate and modulation format. *Sel* chooses the Tags to respond for the Query. *Session* chooses a session for inventory round and Q sets the number of slots in the round.

Table 3.4. Query Command [23]

	Command	DR	M	TRext	Sel	Session	Target	Q	CRC-5
# of bits	4	1	2	1	2	2	1	4	5
description	1000	0: DR=8 1: DR=64/3	00: M=1 01: M=2 10: M=4 11: M=8	0: No Pilot Tone 1: Use Pilot Tone	00: All 01: All 10: SL 11: SL	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	

Table 3.5. Tag reply for Query Command [23]

	Response
# of bits	16
description	RN16

3.7.3 QueryAdjust (Mandatory)

A QueryAdjust command is sent by the Interrogator to adjust the Q parameters of the inventory round. It consist of the following bits as shown in the Table 3.6. A QueryAdjust is always prepended by a frame-sync. In response to this command the Tag shall send RN16 command as shown in Table 3.7. QueryAdjust command consist of command, session and UpDn. The UpDn determines if the Tag shall increment/decrement the Q or have the Q value to be same. Upon receiving a QueryAdjust Tags first update Q, then pick a random value to update the slot counter.

Table 3.6. QueryAdjust Command [23]

	Command	**Session**	**UpDn**
# of bits	4	2	3
description	1001	00: S0 01: S1 10: S2 11: S3	110: Q = Q+1 000: No change to Q 011: Q = Q-1

Table 3.7. Tag reply for QueryAdjust Command [23]

	Response
# of bits	16
description	RN16

3.7.4 QueryRep (Mandatory)

QueryRep instructs Tags to decrement their slot counters and, if slot value is zero after decrementing, to backscatter an RN16 to the Interrogator. The command bits of the QueryRep is shown in Table 3.8. If a Tag receives a QueryRep whose session number is different from the session number in the Query that initiated the round it shall ignore the command.

Table 3.8. QueryRep Command [23]

	Command	**Session**
# of bits	2	2
description	00	00: S0 01: S1 10: S2 11: S3

Table 3.9. Tag reply for QueryRep Command [23]

	Response
# of bits	16
description	RN16

3.7.5 ACK (Mandatory)

An Interrogator sends an ACK to acknowledge a single Tag. ACK echoes the Tags backscattered RN16. The Tag reply to a successful ACK (valid RN16)is as described in the Table 3.10. If the RN16 is invalid then the Tag shall return to arbitrate state without responding.

Table 3.10. ACK Command [23]

	Command	**RN**
# of bits	2	16
description	01	Echoed RN16 or handle

Table 3.11. Tag reply for successful ACK Command [23]

	Response
# of bits	21 to 528
description	EPC bits

3.7.6 NAK (Mandatory)

A NAK command is implemented by Reader and Tag to command all Tags to return to arbitrate state. Table 3.12 shows the command bits.

Table 3.12. NAK Command [23]

	Command
# of bits	8
description	11000000

3.7.7 Req_RN (Mandatory)

Req_RN instructs a Tag to backscatter a new RN16. The command bits are as described in the Table 3.13. Here both the Interrogators command, and the Tags response, depend on the Tags state.

3.7.8 Read (Mandatory)

This allows reader to read part or all of the Tags reserved, OID, TIO or user memory. The command bits are as described in the Table 3.15. The three fields

Table 3.13. REQ_RN Command [23]

	Command	**RN**	**CRC-16**
# of bits	8	16	16
description	11000001	Prior RN16 or handle	

Table 3.14. Tag reply for REQ_RN Command [23]

	RN	**CRC-16**
# of bits	16	16
description	handle or new RN16	

viz. MemBank specifies what bank is to be read, WordPtr specifies the starting word address for memory read, WordCount specifies the number of 16-bit to be read. Read command also includes the Tags handle and CRC-16. If a Tag receives a read with valid CRC-16 but an invalid handle it will ignore read command. If a one or more of the memory words specified in the Read command either do not exist or are read-locked, the Tag shall backscatter an error code. For a successful read the Tag backscatters the bits as shown in Table 3.16.

Table 3.15. Read Command [23]

	Command	**MemBank**	**WordPtr**	**WordCount**	**RN**	**CRC-16**
# of bits	8	2	EVB	16	16	16
description	1100010	00: Reserved 01: EPC 10: TID 11: User	Starting address pointer	Number of words to read	handle	

Table 3.16. Tag reply for successful Read Command [23]

	Header	**Memory Words**	**RN**	**CRC-16**
# of bits	1	Variable	16	16
description	0	Data	handle	

3.7.9 Write (Mandatory)

This command allows reader to write a word in the Tags reserved, OID, TID a or user memory. It has following fields, MemBank which specifies where write occurs as shown in Table 3.17. WordPtr which specifies word address for memory write and the data that needs to be written into the memory. If the memory is write-locked or if the write fails then the Tag shall backscatter an error code.

Table 3.17. Write Command [23]

	Command	**MemBank**	**WordPtr**	**Data**	**RN**	**CRC-16**
# of bits	8	2	EVB	16	16	16
description	1100011	00: Reserved 01: EPC 10: TID 11: User	Address pointer	$RN16 \otimes Word$ to be written	handle	

Table 3.18. Tag reply for successful Write Command [23]

	Header	**RN**	**CRC-16**
# of bits	1	16	16
description	0	handle	

3.7.10 Kill (Mandatory)

Kill allows an Interrogator to permanently disable a Tag. the Interrogator issues two Kill commands, the first containing the 16 MSBs of the Tags kill password EXORed with an RN16, and the second containing the 16 LSBs of the Tags kill password EXORed with a different RN16. Just prior to issuing each Kill command the Interrogator first issues a Req_RN to obtain a new RN16.

Table 3.19. First Kill Command [23]

	Command	**Password**	**RFU/Recom**	**RN**	**CRC-16**
# of bits	8	16	3	16	16
description	11000100	(1/2 kill password)⊗ RN16	000_2	handle	

Table 3.20. Second Kill Command [23]

	Command	**Password**	**RFU/Recom**	**RN**	**CRC-16**
# of bits	8	16	3	16	16
description	11000100	(1/2 kill password)⊗ RN16	Recommissioning bits	handle	

Table 3.21. Tag reply for first Kill Command [23]

	RN	**CRC-16**
# of bits	16	16
description	handle	

Table 3.22. Tag reply for second Kill Command [23]

	Header	**RN**	**CRC-16**
# of bits	1	16	16
description	0	handle	

3.8 Summary

ISO 180006-C GENERATION-2 PROTOCOL is designed for a specific purpose, to have a standardized communication layer for all the Tags. In my Thesis, I have implemented the Tag ID layer of GEN2 protocol with all the states, commands and their functionality using VHDL. We observe that GEN2 protocol lacks the security features to make the EPC-ID secure or to have a mutual authentication between the Tag and the Reader. Hence there is a need to integrate security protocol with the GEN2 to have a secure communication.

CHAPTER 4

IMPLEMENTATION OF GEN2 PROTOCOL

4.1 Introduction

A GEN2 compliant RFID tag has four major components viz. Antenna, RF front end (receiver and transponder), Physical layer (encoder and decoder) and Tag Identification layer. A high level hardware architecture of RFID Tag is shown in Figure 4.1.

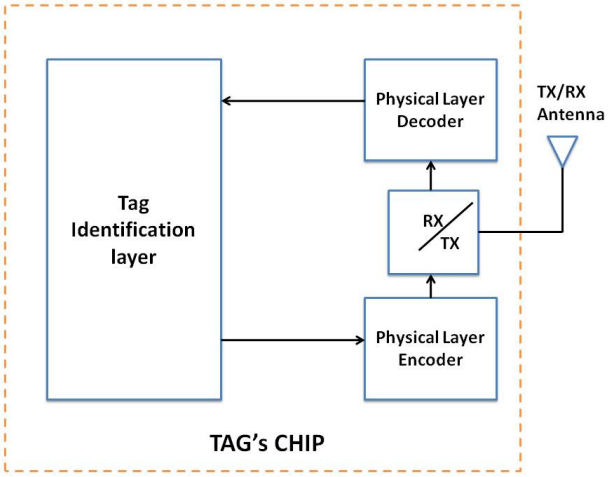

Figure 4.1. Tag hardware.

It is assumed that the RF block will receive the signal from the Reader, and pass the signal to physical layer demodulator and decoder which extracts the data rate and all timing information included in the delimiter and preamble of the received signal. The delimiter marks the beginning of the communication. The two key parameters in the preamble are RT_{cal} and TR_{cal} which is used by the clock data recovery module to

check if they are in the valid range. Then the Tag calculates the threshold value called pivot, which is used to decode the data. Each symbol smaller than the threshold value is decoded as data-0 and symbols greater than the threshold are decoded as data-1. The clock signal generated by the oscillator should also be calibrated to count the number of clock cycles precisely [36]. After the decoding is done the digitized and stable command bits are given as input to the Tag identification layer.

A Tag ID layer is mainly centered around the operational state machine functionality (FSM), although other entities like memory banks, Slot counter, Random Number generator and CRC are also a part of it. To make the RFID GEN2 communication secure, security should be integrated within Tag identification layer. Moreover, the Tag ID layer needs a significant amount of hardware resources over physical layer as we find that Tag ID layer has more functions to perform. We thus implement Tag ID layer in hardware using VHDL.

4.2 Basic Architecture

Figure 4.2 shows a high level architecture of the GEN2 Tag ID layer design. As shown in the figure the architecture has 7 major modules categorized into three main sections.

1. Command Detection module
    - Input Buffer Module
    - CRC Engine Module
2. Control Unit
    - Finite State Machine
    - Slot Counter
3. Response Module
    - RNG Module

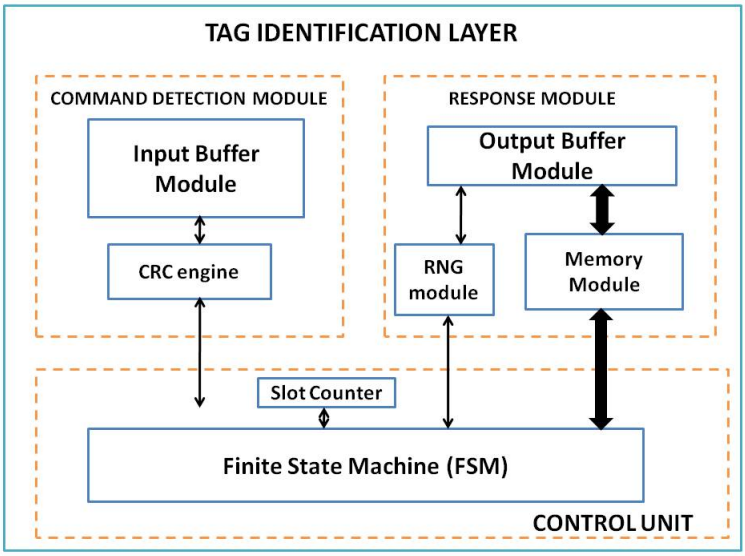

Figure 4.2. Basic block diagram.

- Memory Module
- Output Buffer Module

The Top level entity of the implementation has data bits, clock, preset as inputs to the module and response data bits as output from the module. A detailed description of the input and output pins in the digital block is given Table 4.1.

Table 4.1. Top level Pin description

Pin	**Input/Output**	**Length(bits)**	**Description**
data_in	Input	1	Stable input command bits from the Reader
clk	Input	1	Clock enable which is common to all modules
preset	Input	1	Preset to initialize the state of the Tag
data_out	Output	1	The response bits from the Tag

4.3 Command Detection module

The command detection section is responsible for receiving, decoding and validating the command that is sent from the Reader. The following modules incorporate the necessary logic:

4.3.1 Input Buffer Module

This module is responsible for receiving and collecting the incoming bits from the RF block. Upon receiving these bits the module will detect the command that is sent from the Reader by looking at the first 2,4 or 8 bits from the MSB (which represents the code) as shown in the Table 4.2. Also as described in GEN2 specification the length of the commands can be decided by decoding the type of the command. Depending on the command detected, the bits are passed through CRC engine and the validity of the command is verified. The important input and output pins of the module is described in the Table 4.3.

Table 4.2. GEN2 Commands

Command	**Code**	**Length (bits)**	**CRC Protection**
QueryRep	00	4	No CRC
ACK	01	18	No CRC
Query	1000	22	CRC-5
QueryAdjust	1001	9	No CRC
Select	1010	>44	CRC-16
NAK	11000000	8	No CRC
REQ_RN	11000001	40	CRC-16
Read	11000010	>57	CRC-16
Write	11000011	>58	CRC-16
Kill	11000100	59	CRC-16
Lock	11000101	60	CRC-16

Table 4.3. Command detection module - Pin description

Pin	Input/Output	Length(bits)	Description
clk	Input	1	Clock enable which is common to all modules
preset	Input	1	Preset to initialize the state of the Tag
data	Input	1	Input command bits from Reader
done	Input	1	Done flag to indicate complete of response from Tag
stop_in	Input	1	Stop bit to indicate end of command
stop	Output	1	Stop bit to indicate end of command
bit_len_out	Output	9	Length of the command bits
what_cmd	Output	5	The command sent by the Reader
out_serial	Output	1	Command bits to check CRC
Q	Output	352	Collected data bits from the Reader

4.3.2 CRC Engine Module

The CRC module is made up of a CRC-5 and a CRC-16 engine. This module is used to ensure validity of the Reader to Tag commands.

- CRC-5: The CRC-5 engine uses the polynomial and preset defined in the Table 4.4. The schematic diagram of the CRC-5 engine is shown in the Figure 4.3. To calculate CRC-5, the CRC registers are first loaded with a value 01001_2, then the data is clocked in bit by bit. At the end of the data, Q[4:0] will hold the CRC-5 value. To check CRC-5, preload the CRC registers with a value 01001_2, then the {data,CRC-5} is clocked in bit by bit. After all bits are passed, Q[4:0] should have 00000_2.
- CRC-16: The CRC-16 engine uses the polynomial and preset defined in the Table 4.5. The schematic diagram of the CRC-16 engine is shown in the Figure

Table 4.4. CRC-5 definition

Polynomial	Preset	Residue
$x^5 + x^3 + 1$	01001_2	00000_2

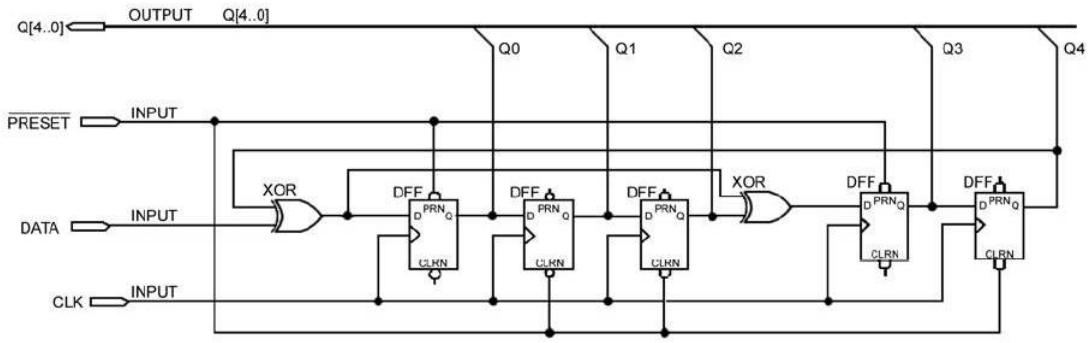

Figure 4.3. CRC-5 circuit [23].

4.4. To calculate CRC-16, the CRC registers are first loaded with a value $FFFF_h$, then the data is clocked in bit by bit. At the end of the data, Q[15:0] will hold the CRC-16 value. To check CRC-16, preload the CRC registers with a value $FFFF_h$, then the {data,CRC-16} is clocked in bit by bit. After all bits are passed, Q[15:0] should have $1D0F_h$.

Table 4.5. CRC-16 definition

Polynomial	Preset	Residue
$x^{16} + x^{12} + x^5 + 1$	$FFFF_h$	$1D0F_h$

The CRC is calculated on the incoming command bits and a CRC_VALID signal is sent to the state machine module. The further operation continues only if the CRC test is passed, else the command is ignored.

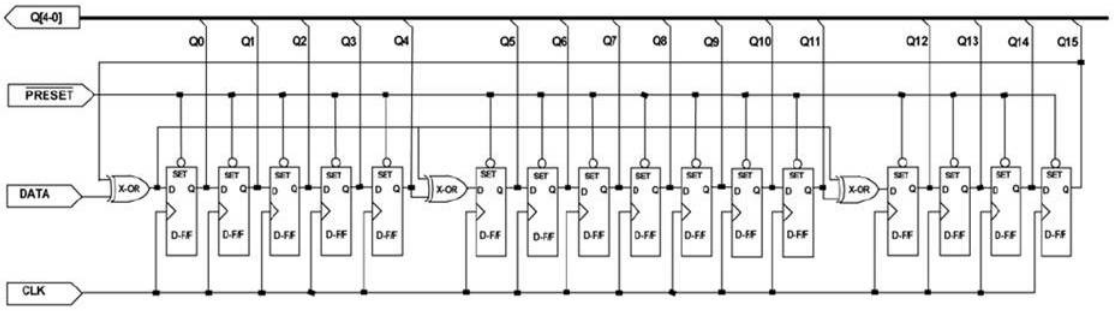

Figure 4.4. CRC-16 circuit [23].

4.4 Control Unit

The control unit is the heart of the design. This section is responsible for performing necessary functions as specified by the operational state machines in the GEN2.

4.4.1 Finite State Machine

The FSM module is responsible for controlling the state transitions of the Tag shown in the Figure 3.4 and deciding the response to be sent from the Tag. The Tag will be in the Ready state on power-up. Depending on the current state of the Tag and valid command from the Reader the state machine is going to make the necessary validations given in the specification and backscatter the response from the Tag. The pin description of the FSM module is shown in Table 4.6.

4.4.2 Slot Counter

The slot counter module is responsible for calculating the slot number at which the Tag is going to reply. The slot number is calculated using the Q-value transmitted by the Reader in the Query command and RN16 generated by the RNG module. The slot number is calculated using the equation 4.1.

Table 4.6. FSM module - Pin description

Pin	Input/Output	Length(bits)	Description
clk	Input	1	Clock enable which is common to all modules
preset	Input	1	Preset to initialize the state of the Tag
crc_valid	Input	1	Describes CRC validity of the Command
data_in	Input	352	Received command bits from the Reader
bit_len_in	Input	9	Length of the command bits
wht_cmd	Input	5	The command sent by the Reader
RN_16_in	Input	16	RN16 generated by the RNG module
*RN*16*_done*	Input	1	Flag to indicate RN16 generation is completed
*RN*64*_done*	Input	1	Flag to indicate RN64 generation is completed
slot_count	Input	1	Slot number for the Tag to reply
slot_done	Input	1	Flag to indicate the success of Slot counter module
$start\_buffer$	Output	1	Flag to start the response module
$MemBnk\_out$	Output	2	The memory bank to be accessed to retrieve data
start_address_out	Output	24	The start address of the data in memory bank
end_address_out	Output	24	The end address of the data in memory bank
RN_16	Output	1	Flag to extract 16-bit number from RNG module
RN_64	Output	1	Flag to extract 64-bit number from RNG module
Q_value_out	Output	4	The Q value to calculate slot number
error_code	Output	1	Flag to respond Error code from response module

$$slotnumber = (RN16) \bmod (2^{Q}) \tag{4.1}$$

The input and output pins of the module is described in the Table 4.7.

Table 4.7. Slot counter - Pin description

Pin	Input/Output	Length(bits)	Description
preset	Input	1	Preset to initialize the state of the Tag
Q_value_in	Input	4	Q-value transmitted by Reader
RN_16_in	Input	16	RN16 data generated by RNG module
slot_value_out	Output	16	slot number for the Tag to reply
slot_done	Output	1	Flag to indicate the completion of slot gen

4.5 Response Module

The response module, depending on the command sent by the Reader and validation made in the FSM module backscatters the response from the Tag.

4.5.0.1 RNG Module

Random number generator (RNG) module is responsible to generate either 16-bit or 64-bit random numbers. The random number is generated by using a Linear feedback shift register(LFSR) with clock enable and parallel load. The input and output pins of the module is described in the Table 4.8.

In the current RFID technologies, an oscillator based Truly Random Number Generator based application scheme is used to generate a random number. In this technique, the thermal noise of two resistors are used to modulate the edge of sampling clock. In oscillator-based TRNG, a jittered low-frequency clock is used to sample a high frequency clock. For lower power considerations, a combination of TRNG and

Table 4.8. RNG module - Pin description

Pin	Input/Output	Length(bits)	Description
clk	Input	1	Clock enable which is common to all modules
$preset$	Input	1	Preset to initialize the state of the Tag
$RN16\_flag$	Input	1	Flag to request a 16-bit random number
$RN64\_flag$	Input	1	Flag to request a 64-bit random number
$lfsr\_out$	Output	16	RN16 data generated
RN_64_data	Output	64	RN64 data generated
$slot\_15$	Output	16	RN16 data to generate slot number
$RN16\_done$	Output	1	Flag to indicate RN16 generation is completed
$RN64\_done$	Output	1	Flag to indicate RN64 generation is completed

PRNG is used where a 1-bit true random number generated by a TRNG is used in cycle ring of 16-bit LFSR as a seed the output sequence of the LFSR will also be unpredictable [8].

An experiment was carried out by Rushikesh for his Masters' Thesis using some standard RFID tags to find the distribution function of their random number generators. Pearson's chi-square test was carried out to verify the uniform distribution of the Tags. The results are summarized in Table 4.9 [32].

Table 4.9. RNG performance of common tags [32]

Tag	Mean Square Error
Bow-Tie	41.43
Squiggle	22.53
TI	20.77
Reference data set	26.22

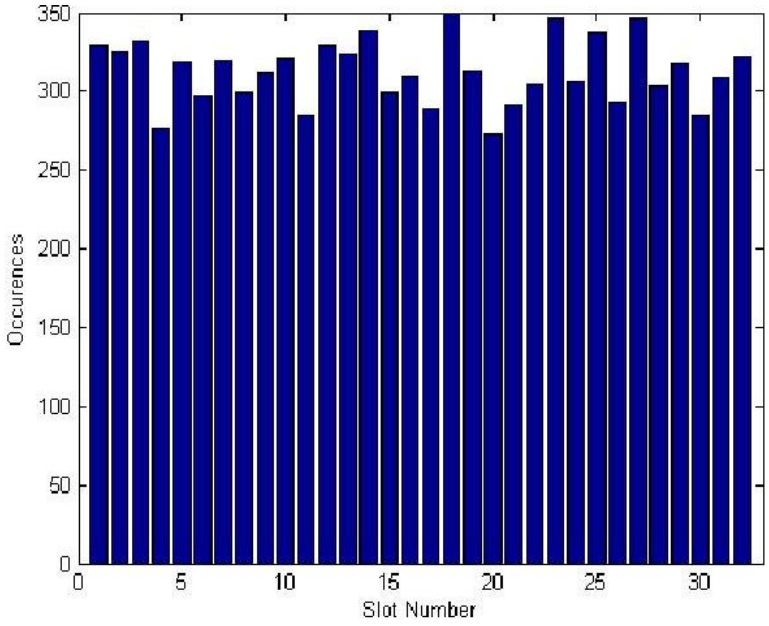

Figure 4.5. Bow-Tie tag [32].

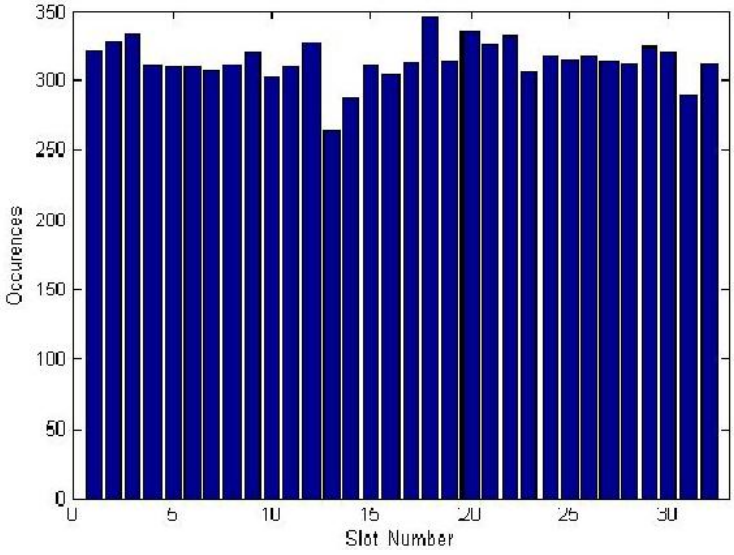

Figure 4.6. Squiggle tag [32].

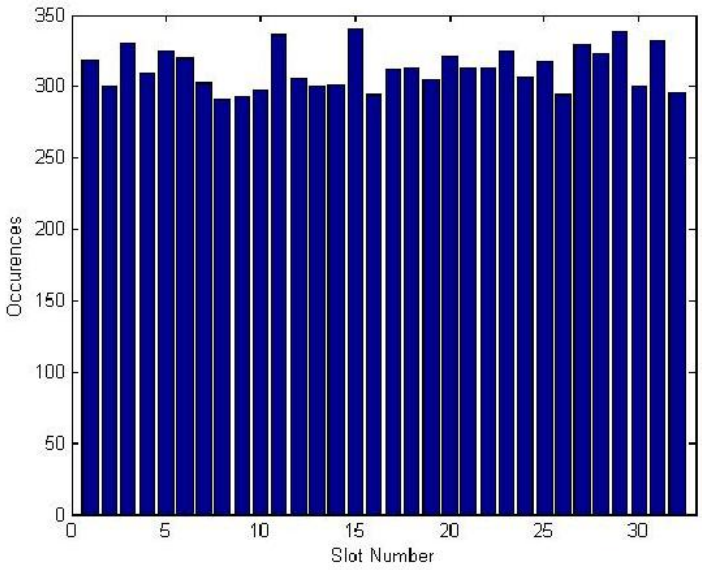

Figure 4.7. TI tag [32].

4.5.1 Memory Module

This module implements the four memory banks viz. EPC memory, Reserved Memory, Tag ID memory and User memory as specified in the GEN2 spec. The option to access the data from the memory 1-bit or 16-bits at a time is provided to have more flexibility. The EPC memory ranges from 00_h to $20F_h$ which stores the EPC-ID of the Tag which is the object identification number. The Reserved memory ranges from 00_h to $3F_h$ and it stores the 32-bit access password and 32-bit kill password of the Tag. The TID memory ranges from 00_h to $1F_h$ and it stores the an 8-bit class identifier and other information to uniquely identify custom commands.

The User memory is not implemented as it is an optional feature, instead a key memory is implemented to store the unique key to encrypt the data for security. More than one module cannot access a memory block at the same time. To manage this there are **rd_en** and **wr_en** flags which is used to read data from memory and write

Table 4.10. Memory Module - Pin description

Pin	Input/Output	Length(bits)	Description
clk	Input	1	Clock enable which is common to all modules
preset	Input	1	Preset to initialize the state of the Tag
send	Input	1	Flag to initiate memory access
start_address_out	Input	24	The start address of the data in memory bank
end_address_out	Input	24	The end address of the data in memory bank
mem_bank	Input	2	Memory bank from which the data needs to be accessed
lock_value	Input	2	The lock command bits transmitted by the Reader
data_in_write	Input	16	Data to be written into the memory by the Reader
data_16_bit	Input	1	Flag to indicate Memory access as 1-bit/16-bit accessible
open_state_valid	Input	1	Flag to indicate state of the Tag as open or secured
rd_en	Input	1	Flag to enable read access from Tag
wr_en	Input	1	Flag to enable write access to Tag
data_out_bit	Output	1	1-Bit data read from memory
data_out_16	Output	16	16-Bit data read from memory
rd_wr_lock_done	Output	1	Flag to indicate the successful memory access
error	Output	1	Flag to indicate the fail in memory access
stop	Output	1	stop memory access

data into memory. The **start_address** and **end_address** which is a 24-bit pointer is used to access data from the memory.

The Reader can change the read/write access to the memory by using a Lock command. If the Reader tries to read the information from memory when it is locked, then a error code is backscattered.

The input and output pins of the module is described in the Table 4.10.

4.5.2 Output Buffer Module

This module is responsible for arranging the outgoing bits in order and sends bit by bit at each rising edge of the clock to the physical layer encoder.

The input and output pins of the module is described in the Table 4.11.

Table 4.11. Output Buffer Module - Pin description

Pin	Input/Output	Length(bits)	Description
clk	Input	1	Clock enable which is common to all modules
preset	Input	1	Preset to initialize the state of the Tag
wht_cmd	Input	5	The command sent by the Reader
start_address	Input	24	The start address of the data in memory bank
end_address	Input	24	The end address of the data in memory bank
*rn*16*_data*	Input	16	The RN16 value generated by RNG
lock_value	Input	2	The lock command bits transmitted by the Reader
data_key	Input	128	The key to encrypt plain text
cipher_text	Input	64	Encrypted plain text
*data*_16*_in*	Input	16	16-bit data read from the memory
data_plain_text	Output	64/128	Plain text to be encrypted
mem_bank_out	Output	2	Memory bank to be accessed for data
lock_value_out	Output	2	The lock command bits transmitted by the Reader

4.6 Summary

The RFID GEN2 communication can be made secure, by integration of security within Tag identification layer. Thus the implementation of TID layer of the GEN2 protocol in VHDL provides a strong foundation to integrate security and analyze the various performance parameters like speed, area and power consumption. The high level design of the TID layer consists of modules like command detection module, FSM module, CRC engine, RNG module, slot counter, memory module and response module. These modules are integrated together to perform the functionality of a GEN2 Tag. Security algorithms can now be integrated with this design to achieve security in GEN2 protocol.

CHAPTER 5

CRYPTOGRAPHIC ALGORITHMS

5.1 Introduction

With a diverge scope of applications in RFID world the security of RFID systems have been of extreme concern. The major RFID security threat is the illegal tracking of RFID tags. Sniffing the data that is being communicated between the Reader and the Tag is very easy as the ID is not protected. To address such issues, and to secure the information exchanged between Readers and Tags, data encryption can be incorporated. The security protocols may be based on symmetric or public key cryptography. However, current commercial data encryption modules may prove be too expensive in terms of area (cost) and power (range or lifetime) to be feasible for a passive RFID system. Therefore, research is carried out on how to find low-cost encryption techniques suitable for RFID systems. Thus the RFID systems look at the "lightweight" cryptographic techniques which consume fewer resources and still provide suitable security. Thus implementation of these security algorithms must be chosen depending on the requirements of the application. The cryptographic techniques are commonly either Symmetric-key algorithms or Public-key (Asymmetric-key) cryptography. Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key. In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. The public key is typically used for encryption, while the private or secret key is used for decryption.

5.2 Public-key cryptography

There are two main branches in public key cryptography viz. Public key encryption and Digital Signatures. A message that is encrypted using the public key of the recipient cannot be decrypted by anyone other than one having the matching private key. Presumably the owner/recipient will have the associated private key to decrypt the cipher text. To verify the authenticity of the sender's information digital signatures are used. Here a message signed with sender's private key can be verified by anyone who has access to sender's public key thereby proving the authenticity of the sender and the message is not tampered. The various asymmetric key techniques include RSA encryption algorithm, Elliptic curve cryptography techniques, ElGamal etc. It is proven that these implementations need tens of thousands of gates to be implemented in hardware [16] and hence not suitable for resource constraint applications like RFID.

5.3 Symmetric-key algorithms

The symmetric-key algorithms can be classified as block ciphers and stream ciphers algorithms. The block ciphers take a block of plain text and a key as input, and gives a block of cipher-text of the same size as output. Since the information in real world are more than just one block of text, there is way of interconnecting these blocks and also providing with extra level of security. There are several modes of operations which allow block ciphers to be chained. The most common modes of operation are EBC (Electronic codebook), CBC (Cipher-block chaining), CFB (Cipher feedback) and CTR (Counter) [11]. Examples of Block cipher symmetric-key encryption algorithms are DES (Data Encryption Standard), AES (Advanced Encryption Standard) have been designated as the cryptographic standards by the US

government. Some of the other well known block ciphers are AES, DES, IDEA, TEA, XTEA, XXTEA, Triple DES, Blow fish etc. The size of block/plain text and the key text is unique for each algorithm. For example, PRESENT has two implementations with a block size of 64-bit, but has 80-bit key and 128-bit key algorithms, AES works on block size of 128-bits and key size of 128-bits. The key size is one of the parameters that decide the complexity or the security level of the algorithm. In my Thesis I have chosen AES, Present and XTEA algorithms to implement and integrate them with the GEN2 protocol and validate the performance in terms of area, power and throughput (speed). The table below the block size, key size and security level for each algorithm.

5.3.1 AES - Advanced Encryption Standard

AES or Advanced Encryption standard is a symmetric-key block cipher. AES is perhaps the most widely used encryption standard, especially in Internet technology. AES is also called Rijndael cipher, named after its designers Vincent Rijmen and Joan Daemen. AES was adopted as a standard by United States Government.

AES is designed based on the principle of Substitution permutation structure. The inputs to a SP-network are plain text and key. A substitution permutation network or a SP-network is chain of mathematical calculations performed on the plain text by substituting a subset of the text with another text and then permutation is performed after every subset is substituted and XOR'd with key to create a cipher text. The chain of calculations can be viewed as iterations of mathematical formula applied several times called rounds. In my thesis, I consider AES with 128-bit block size and 128-bit key size.

There are 4 steps involved in the encryption process:

1. AddRoundKey
2. S-box or Substitution boxes

3. Shift Rows
4. Mix Columns

A combination of these steps is executed iteratively for N number of rounds, where N depends on the key size. N is 11 for 128-bit key, 13 for 192-bit key and 15 for 256-bit key size. The encryption process for 11 rounds is shown in the figure below. Here as we see Round0 has just a AddRoundKey step, Round1 - Round9 involves all the four steps mentioned above and Round10 has AddRoundKey, S-box and Shift Rows.

5.3.1.1 S-Box

An S-box or a substitution box, (also called Rijndael S-box) is a non-linear substitution of a block of subset bits by another block bits. The key property of an S-box is that this is non-linear.A 4-bit S-box substitutes non-linearly a block of 4-bits by another block of 4-bits. Thus a plain text of 128-bits is used as 32 4-bit inputs for the S-boxes to get a new substituted 128-bit. In AES the 128-bit plain text is arranged as a 4x4 matrix grid of 8-bits (or 1 byte, thus also called SubBytes box) each. Each block of the grid is passed through the S-box to get new substituted 128-bits. This S-box is constructed by composing two transformations

- Take the multiplicative inverse over $GF(2^8)$.
- Apply the affine transformation over GF(2):

$$b_i^{'} = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i \tag{5.1}$$

In matrix form, the affine transformation element of the S-box can be expressed as:

$$\begin{bmatrix} b_0{}' \\ b_1{}' \\ b_2{}' \\ b_3{}' \\ b_4{}' \\ b_5{}' \\ b_6{}' \\ b_7{}' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

5.3.1.2 ShiftRows

In ShiftRows step, the operation is performed on the last three rows of the state such that each row is shifted circularly by a certain offset. In AES, the offset depends on the row number (or row state). After the S-box or the SubBytes step, substituted bytes are again arranged in a 4x4 grid like fashion, and the ShiftRows operate on the rows of this grid. The mathematical representation of this is given in equation 5.2.

$$S_{r,c}{}' = S_{r,(c+shift(r,N_b))modN_b} \quad for \quad 0 < r < 4 \quad and \quad 0 \leq c < N_b \tag{5.2}$$

where the shift value depends on row number.

The ShiftRows operation is performed in AES as shown in the Figure 5.1 where the the first row is left unchanged without any circular shifts, the second row is left shifted circularly by an offset of 1, third with an offset of 2 and fourth row with an offset of 3.

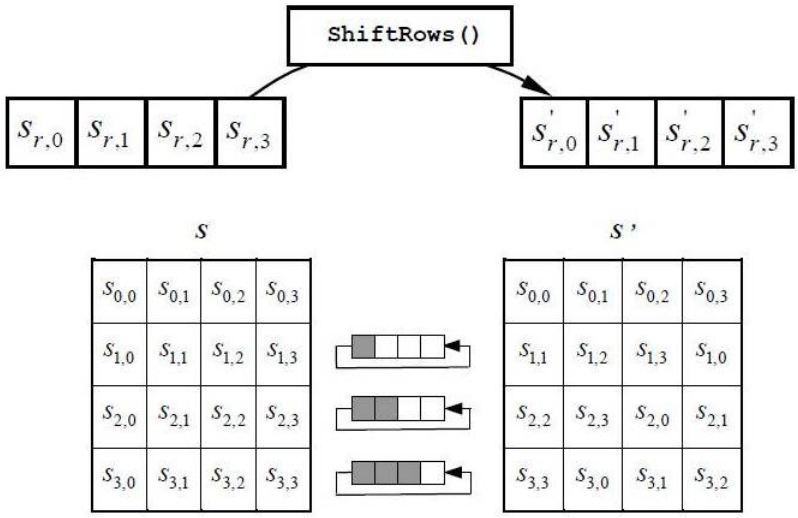

Figure 5.1. ShiftRows operation [44].

5.3.1.3 MixColumns

After the ShiftRows operation, the MixColumns() transformation operates on the State column-by-column. The four bytes in a column are then replaced by the values calculated over the following equations.

$$s_{0,c}{}' = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \tag{5.3}$$

$$s_{1,c}{}' = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \tag{5.4}$$

$$s_{2,c}{}' = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \tag{5.5}$$

$$s_{3,c}{}' = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \tag{5.6}$$

The Figure 5.2 explains the mixcolumn operation.

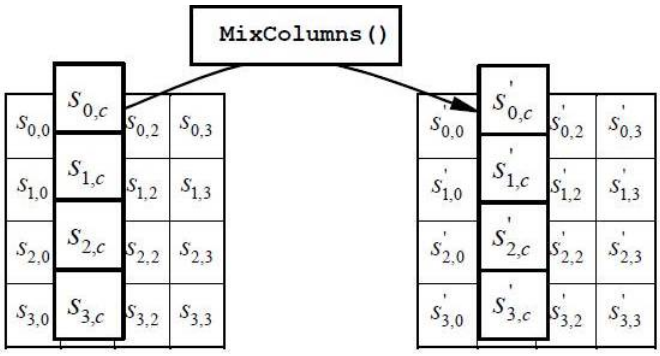

Figure 5.2. MixColumns operation [44].

5.3.1.4 AddRoundKey

AddRoundkey is a simple operation where the Round Key is added to the state by a bitwise XOR operation. The key for each round is provided by a key expansion process. The N_b words in the Round Key is added to the columns of the state as given in the equation 5.7.

$$[s_{0,c}{}', s_{1,c}{}', s_{2,c}{}', s_{3,c}{}'] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*N_b+c}] \tag{5.7}$$

5.3.1.5 Key Expansion

The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule.The 128-bit key expansion algorithm takes a 4-word(16-byte) key as input and produces a linear array of 44 words. This is used to provide a 4-word round key for all the 11 rounds of the cipher. The different steps for the key expansion is described below:

1. The original key forms the beginning of the expanded key. This is divided into four words indexed from 0-3.
2. A RotWord operation is performed which rotates the first byte of the 32-bit word from beginning to its end.

3. The SubWord operation performs a S-box substitution on four separate bytes of 32-bit word.
4. The results of the above two steps is XORed with the first value of RCON (Round constant).
5. The result of the step 4 is XORed with the remaining words as shown in the figure below until entire expanded key is calculated.

   The Table 5.1 shows the values of the RCON array.

Table 5.1. Rcon Array [20]

Round #	**Round value(Hex)**
1	01
2	02
3	04
4	08
5	10
6	20
7	40
8	80
9	1B
10	36

5.3.2 PRESENT

PRESENT is a block cipher which works on the principle of substitution-permutation (SP) network. The block length is 64-bits and two key lengths of 80 and 128-bits are supported. In this thesis I have considered the 128-bit key implementation as it gives an added security as compared to 80-bit key implementation. The steps required to calculate the cipher for both the cases are similar, except that the key scheduling process for each round has to handle 128bits, thus increasing the computational complexity.

The steps to perform encryption for the key lengths are:

1. AddRoundKey: The result of the key scheduling operation for every round is taken as a round key and it is added to the current state. This operation is performed as shown below.

$$b_j \rightarrow b_j \oplus {k_j}^i \quad where \quad 1 \leq i \leq 32 \quad and \quad 0 \leq j \leq 63 \tag{5.8}$$

2. S-Box Layer: The S-box used here is a 4-bit to 4-bit S-Box. The output of the S-box for all possibilities of 4-bit inputs are given in the table below.
3. P-Layer: The permutation used in this algorithm is described in the following table. Here Bit i is moved to bit position P(i).

The steps involved to achieve the encryption are as described below:

1. The 64-bit plain text is split into 16, 4bit blocks.
2. Each 4-bit data is sent through an S-box. All the 16 S-boxes together is called an S-layer.
3. The result of the S-Layer is given to the P-Layer and the permutated data is then XORed with the round key. An S-layer, P-layer and XOR steps together are called one round.
4. Steps 1 through 5 are repeated 32 times to obtain the final 64-bit cipher text.

The Figure 5.3 demonstrates the steps mentioned above.

5.3.2.1 Key Scheduling - 80 bit

The key scheduling process operates on the user-supplied 80-bit key and outputs a 64-bit key each round. The output of the key scheduling process depends on 80-bit key and 5-bit round counter value. The following steps describes the key scheduling process.

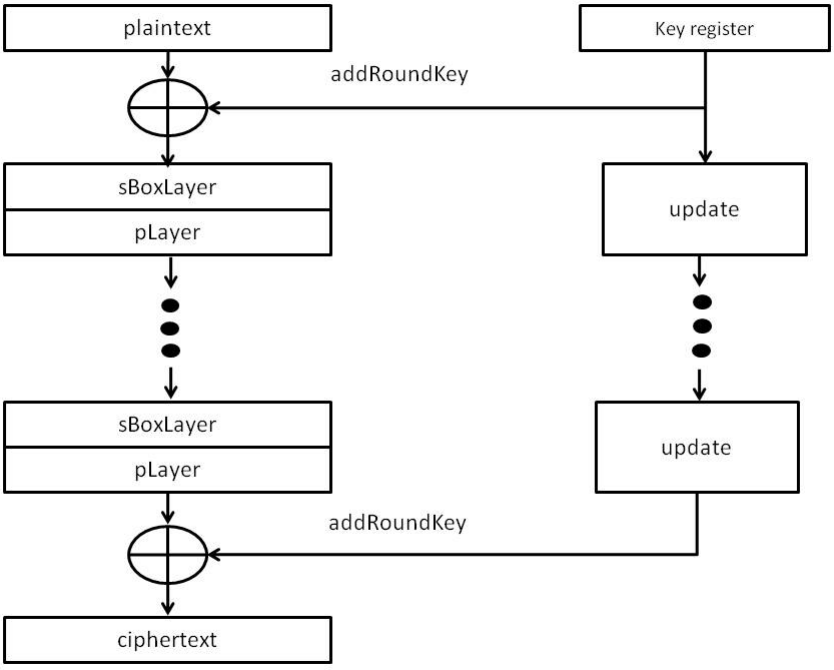

Figure 5.3. Encryption Block Diagram of PRESENT [3].

1. At each round, the round key consists of the 64 leftmost bits of the current value of the Key register. After extracting the round key, the key register is updated.
2. To update the key register the 80-bit key is circularly left shifted by 61-bits, and new 80-bit key is obtained.
3. Bits from 79 - 76 is replaced by its value passed through S-Box.
4. Bits from 19 - 15 is replaced by the bits XORed with the round counter value.
5. Steps 1 through 4 are repeated for 31 rounds to obtain the key for each round.

5.3.2.2 Key Scheduling - 128 bit

The key scheduling process operates on the user-supplied 128-bit key and outputs a 64-bit key each round. The output of the key scheduling process depends on

128-bit key and 5-bit round counter value. The steps are similar to that described for 80-bit key with a few differences listed below.

1. At each round, the round key consists of the 64 leftmost bits of the current value of the Key register. After extracting the round key, the key register is updated.
2. To update the key register the 80-bit key is circularly left shifted by 61-bits, and new 80-bit key is obtained.
3. Bits from 127 - 124 is replaced by its value passed through S-Box.
4. Bits from 123 - 120 is replaced by its value passed through S-Box.
5. Bits from 66 - 62 is replaced by the bits XORed with the round counter value.
6. Steps 1 through 5 are repeated for 31 rounds to obtain the key for each round.

5.3.3 XTEA - Extended Tiny Encryption Algorithm

Extended Tiny Encryption Algorithm or XTEA is one of the efficient cryptographic techniques and is believed to be one of the fastest algorithms. XTEA is designed to overcome the attacks possible on TEA. XTEA is based on a Feistel Network which uses "XOR", "ADD" and "SHIFT" operations. XTEA works on 64-bit blocks with 128-bit key operating on them to result in 64-bit cipher text. It performs 32 round iterations to obtain the cipher results. The basic encryption process of XTEA can be diagrammatically represented as shown in the Figure 5.4. As shown it involves a series of addition, XOR, left shift and right shift operations which makes the hardware implementation simple by avoiding the use of bigger blocks such as S-Box.

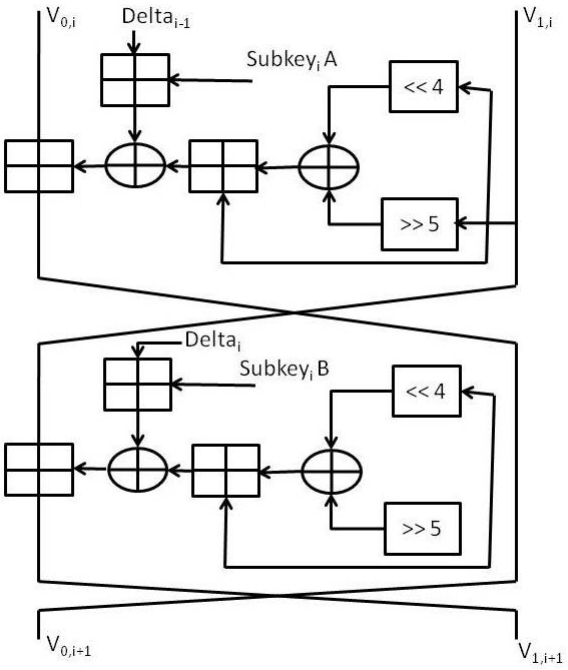

Figure 5.4. XTEA Encryption Block Diagram [10].

5.4 Results and comparison

The results for the FPGA implementation of AES, PRESENT and XTEA using VHDL is shown in Table 5.2. A comparison of the implementations with other low power implementation of Present reported by Sbeiti et al. and XTEA implementation by Kaps and AES-128 implementations by Chodowiec et al., Good et al. and Rouvroy et al. is shown in terms of Area and max operating clock frequency.

Table 5.2. FPGA Implementation Results and comparison

	# of clock cycles	CLB Slices	Slice FF	4 input LUT's	Max Frequency[MHz]	Device
Present-128 (ours)	32	210	149	319	236.061	xc3s4000-5
Present-128 [53]	32	202	-	-	254	xcS400-5
XTEA (ours)	32	974	211	1769	71.524	xc3s4000-5
XTEA [31]	-	266	-	-	-	xc3s50-5
AES-128 (ours)	72	1785	948	3420	136.061	xc3s4000-5
AES-128 [7]	44	522	-	-	60	xc2s30-6
AES-128 [19]	-	264	140	-	67	xc2S15-6
AES-128 [19]	-	17425	-	-	196.1	xc3S2000-5
AES-128 [50]	44	1231	-	-	123	xc3S50-4

The published results of various researchers on the ASIC implementations of the AES, PRESENT and XTEA is shown in the Table 5.3.

Table 5.3. ASIC Implementation Results and comparison

	# of clock cycles	Area(GE)	Power(μW)	Throughput(Kbps)	Process
Present-80 [3]	32	1570	5	200	0.18μm
Present-128 [3]	32	1886	-	200	0.18μm
Tiny XTEA-1 [31]	240	3500	18.8	26.7	-
Tiny XTEA-3 [31]	112	3490	19.5	57.1	-
AES-128 [21]	160	3100	62	121	0.13μm
AES-128 [12]	1016	3600	-	-	0.35μm
AES-128 [52]	54	5400	-	311	0.11μm
AES-128 [43]	92	8500	-	70	0.60μm

5.5 Summary

Symmetric-key algorithms like AES, PRESENT and XTEA are the strong ciphers that can be used to integrate security in GEN2 protocol. These algorithms are implemented in hardware using VHDL and can be integrated with the GEN2 implementation to achieve security within protocol. It is observed that PRESENT is fast and also uses less resources as compared to AES and XTEA, hence a suitable choice for resource constraint applications.

CHAPTER 6

SECURE-ID AND MUTUAL AUTHENTICATION PROTOCOL

6.1 Introduction

The security and privacy in RFID systems have always been a concern and heavily discussed in public. A lot of research is going to on make the communication between the Reader and Tag secure. To the best of our knowledge there is no attempt made to secure the identity (EPC/Object-ID) of the Tag. This information (EPC) being transmitted in the clear, enables tracking of the tagged objects. This can prove to be a serious threat to privacy in some applications. In this chapter a protocol has been developed to add security in the identification process(Inventory process) of the Tag and also have an authentication which serves as proof of concept for authenticating an RFID tag to a Reader device. The cryptographic components are assumed to be too costly because of the limited resources available in a passive Tag. In contrast to this, in my Thesis I have implemented security using AES, PRESENT and XTEA encryption algorithms and have provided a detailed analysis on the effect on the performance of the Tag in terms of power, speed, timing and area.

6.2 Security design

In this thesis, I propose a protocol which provides secure identification and mutual authentication and enables a secure communication channel when integrated within the GEN2 protocol.

6.2.1 Securing the Tags Identity

The Figure 6.1 shows the normal inventory process in the GEN2 protocol. As seen in the figure the $\{PC||EPC\}$ that is being backscattered from the Tag is insecure and it can be very easily captured by malicious readers which will give access to the details of the tagged object.

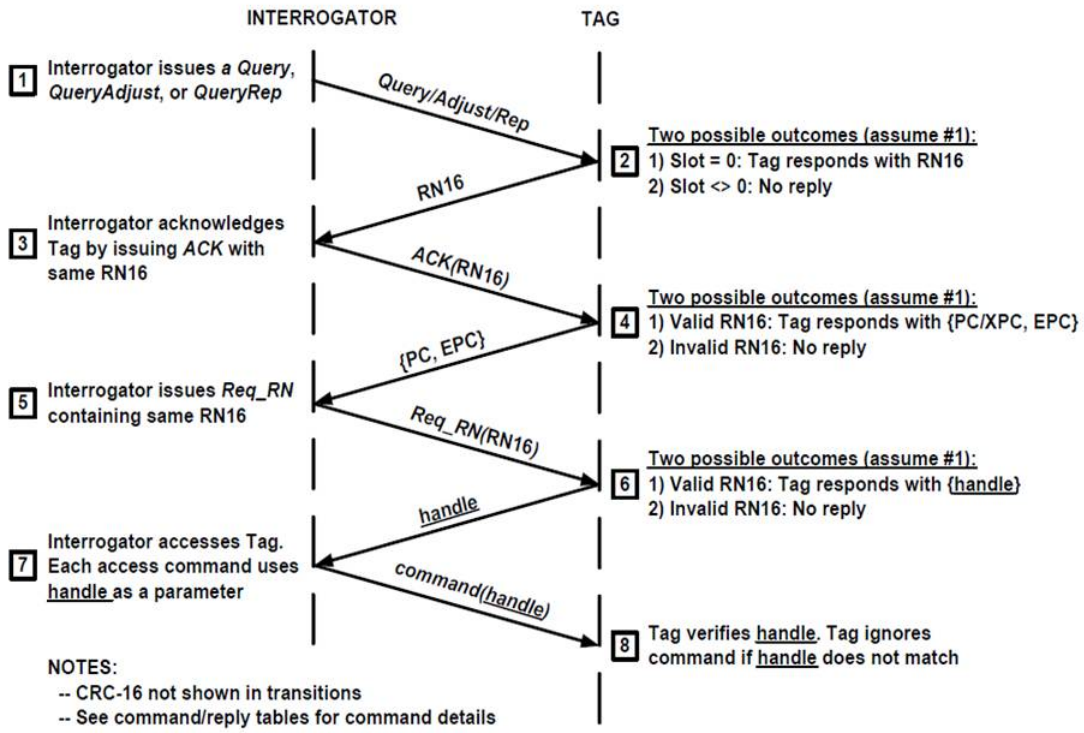

Figure 6.1. Tag Inventory and access [23].

A method to make the communication secure is by randomizing the information being transmitted by the Tag to the Reader in the identification process. This novel

approach will confuse the malicious readers by sending different information every time by the same Tag.

To achieve this, encryption algorithms are integrated within the GEN2 protocol which will encrypt the information being transmitted. The important data that needs to be secured is the EPC-ID. But encrypting the EPC-ID with a unique key stored in the Tag will not enable privacy as it is the same encrypted data backscattered every time.

Another approach of doing this is by transmitting an encrypted random number using the a unique key stored in the Tag. On the other end, the Reader will decrypt this information using the same key which will in turn point to the information of the tagged object in the database. It is assumed that every Tag has its own unique key for encryption and decryption.

In this thesis, three types of design has been proposed and a detailed discussion of these designs has been given in the following sections.

6.2.1.1 Design A

In this design, the Tag instead of sending the EPC or encrypted EPC will follow the following steps to secure the data. This design can be Implemented using AES, PRESENT and XTEA since they satisfy the T1 requirements of the protocol.

1. The Tag will generate a 48-bit random number and will pass it through the CRC-16 engine.
2. It will then concatenate the Random number with the CRC-16 generated, and will pass the 64-bit number together to the cipher engine.
3. The Tag will backscatter this cipher text instead of the EPC-ID as a response to ACK command.

4. The Interrogator on receiving this will decrypt the information using all the keys stored in its database.
5. Then the decrypted data is fed to the CRC-16 engine to get a residue of $1D0F_h$, if its the correct key.

The design of this approach in the inventory process is shown in the Figure 6.2

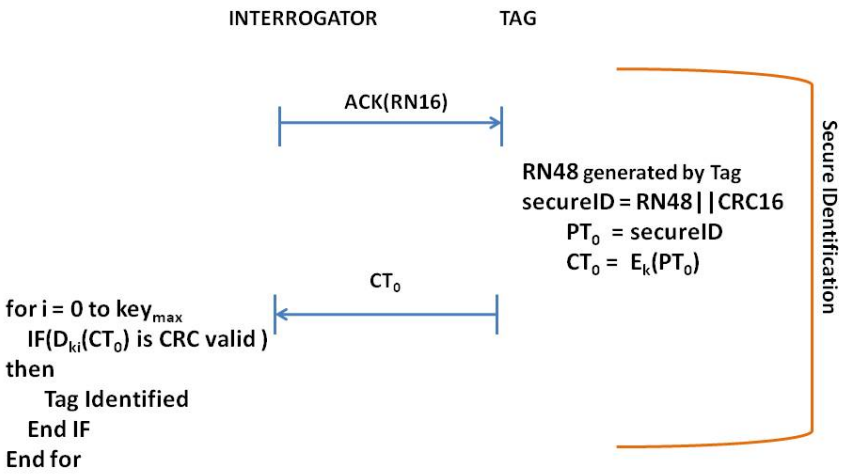

Figure 6.2. Design-A for Secure Identification protocol.

There are some drawbacks in this approach which makes the protocol less secure. Since there 2^{48} combinations of random number and only 2^{16} combinations of CRC available, there will more than one number having same CRC. Due of this on the Reader, a wrong key used to decrypt the cipher might give a $1D0F_h$ when passed through CRC engine. The probability of this error is as shown on the equation 6.1.

$$p(\epsilon) = 2^{48} \times [2^{64}/2^{128}] = 1/2^{16} \tag{6.1}$$

When the cipher engine has a 128-bit key and 64-bit block size, the probability of a key decrypting the cipher to a plain text which gives $1D0F_h$ when passed through CRC engine is $1/2^{16}$. This error is intolerable since this might lead to a wrong key hence a wrong object identification. Hence one more design is proposed to overcome this problem.

6.2.1.2 Design B

In this design the Tag instead of sending the EPC or encrypted EPC will follow the following steps to secure the data. This design is Implemented using AES, PRESENT and XTEA.

1. The Tag will generate a 64-bit random number and will encrypt it using the unique key in its memory.
2. It will then concatenate the plain text with the corresponding cipher text generated to form a 128-bit number.
3. The Tag will backscatter this data instead of the EPC-ID as a response to ACK command.
4. The Interrogator on receiving the data will decrypt the LSB 64-bits or encrypt the MSB 64-bits using all the keys stored in its database.
5. It will then find the MSB 64-bits equal to the LSB 64-bits if it is the correct key.

In the Figure 6.3, secureID is the 64-bit random number generated by the Tag. A 64-bit random number is good if the cipher has a block size of 64 as in the case of PRESENT or XTEA. Any malicious reader which tries to sniff the data has to use a brute-force attack to get the key, decrypt and understand the information.

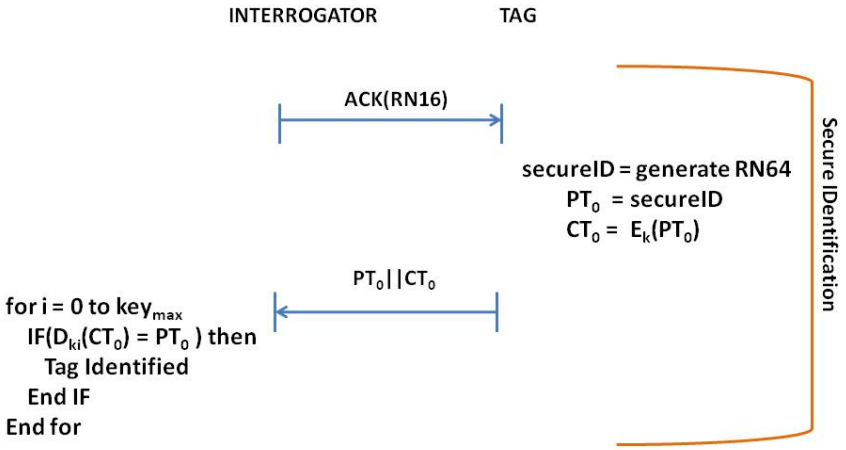

Figure 6.3. Design-B for Secure Identification protocol.

6.2.1.3 Design C

The passive RFID GEN2 Tag has a 1ms power-up time for performing initializations like

- Setting the sessions of Tag.
- Initialize Tag to Ready state.
- Copying the memory contents to registers.

This power-up time can be used to calculate the objectID that has to be backscattered in response to the ACK command. This design can be Implemented using AES, PRESENT and XTEA in CBC mode since it is more secure than a EBC mode.

1. On power-up the Tag will generate a 64-bit random number (PT_0) and will encrypt it using the unique key in its memory to obtain CT_0.

2. The Tag then operates on the PT_0 and CT_0 to get CT_1. Again the Tag operates on the same PT_0 and CT_1 to get CT_2.
3. It then calculates objectID which is $PT_0||CT_2||EPC \otimes (CT_0||CT_1)$.
4. In the inventory process the Tag on receiving an ACK command, will backscatter the calculated objectID.
5. The Interrogator on receiving the data will calculate $CT_0^{'}$, $CT_1^{'}$ and $CT_2^{'}$ and verify $CT_2^{'} = CT_2$ to obtain the correct key.
6. The interrogator will then retrieve EPC by $(EPC \otimes (CT_0||CT_1)) \otimes (CT_0^{'}||CT_1^{'})$.

At a on-chip clock rate of 2MHz, we have 2000 clock cycles to perform the initialization operations and to calculate objectID. It is observed that pipelining the initialization and encryption process makes the process more fast and efficient. It takes 124 clocks, 124 clocks and 215 clocks in case of PRESENT, XTEA and AES respectively to perform the operations and calculate the objectID.

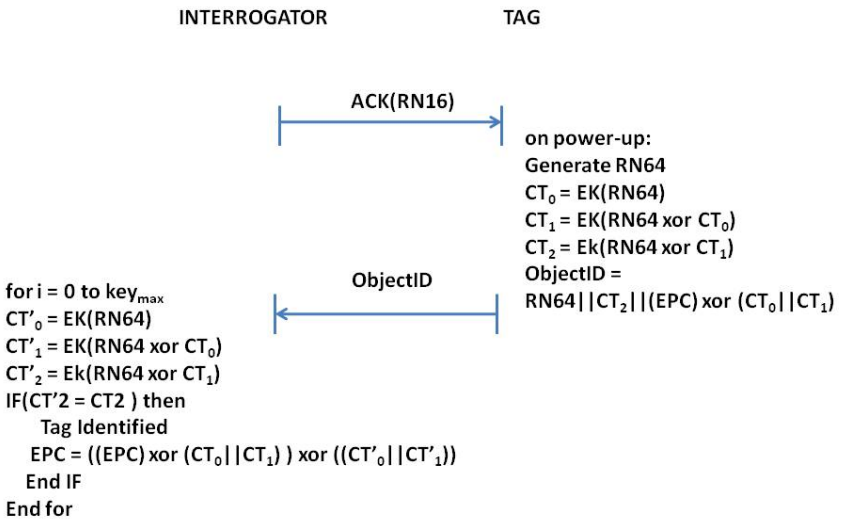

Figure 6.4. Design-C for Secure Identification protocol.

In the Figure 6.4, the length of the objectID is minimum of 224-bits which is going to affect the performance of Tag identification.

6.2.1.4 Possible Attacks on Identification process

- In design B, for a particular plain text the corresponding cipher text is is always the same. Hence the $PT_0||CT0$ pair is unique even if the PT_0 is random. This can be easy to crack the key and identify the Tag every time it generates the same unique pair.
- If the seed for the RNG is not randomized, then all the Tags will generate same random numbers at an instance of time.

The simulation and timing analysis of the implementations are discussed in the next chapter.

6.2.2 Mutual authentication

Authentication means that an object proves its identity to its communication partner. The authentication of the Reader to the Tags will solve the problems of unauthorized readers reading from and writing into the tags memory. Authentication of Tags means that the Tag proves its identity to the reader. Also a forged tag cannot convince the reader of its authenticity [12].

The challenge-response is a broadly used technique to achieve mutual authentication. In addition to mandatory commands which the Tags must implement, a **challenge** must be added to the protocol which is used for authentication. The mutual authentication process along with secure communication channel is described in the Figure 6.5.

A modification to the normal GEN2 state machine is proposed to achieve Mutual authentication and secure communication. As shown in the figure, the Tag will

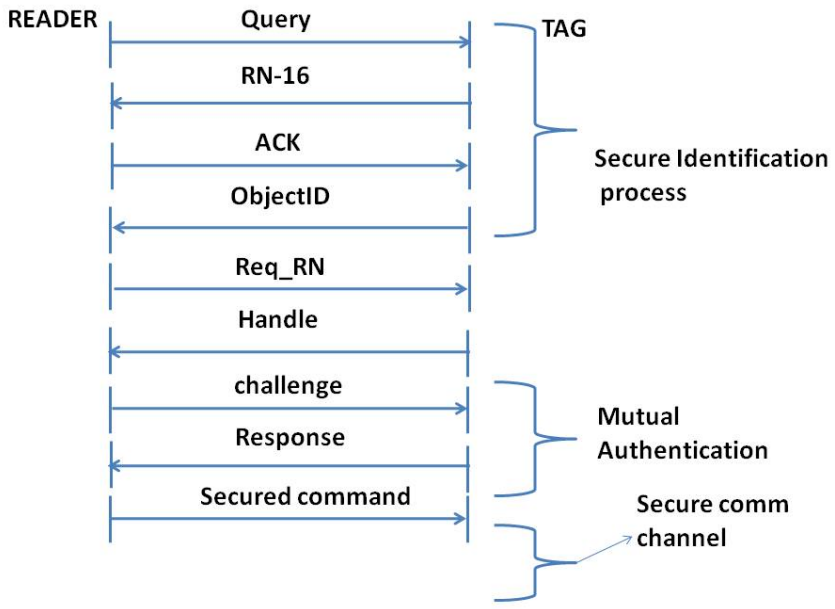

Figure 6.5. Secure Identification with Mutual Authentication.

move to secured state only after the mutual authentication process. If either the Tag or the Reader fails this process, a secure communication channel is not established. A normal unsecured Tag cannot participate in the secured communication, but the Readers can still read/write from the Tags memory.

In this thesis, I propose two designs to achieve mutual authentication. A detailed explanation of both the designs are given below.

6.2.2.1 Design 1

This step is performed immediately after the inventory round. Then a Reader issues a Req_RN which requests a new handle. On receiving handle the Reader sends a challenge(CT_1) which is $E_k(E_k(secureID))$ where it is the same secureID transmitted during inventory process. The Tag will verify the challenge by encrypting the CT_0 and in response will backscatter CT_2 $E_k(E_k(E_k(secureID)))$. The Reader in turn will

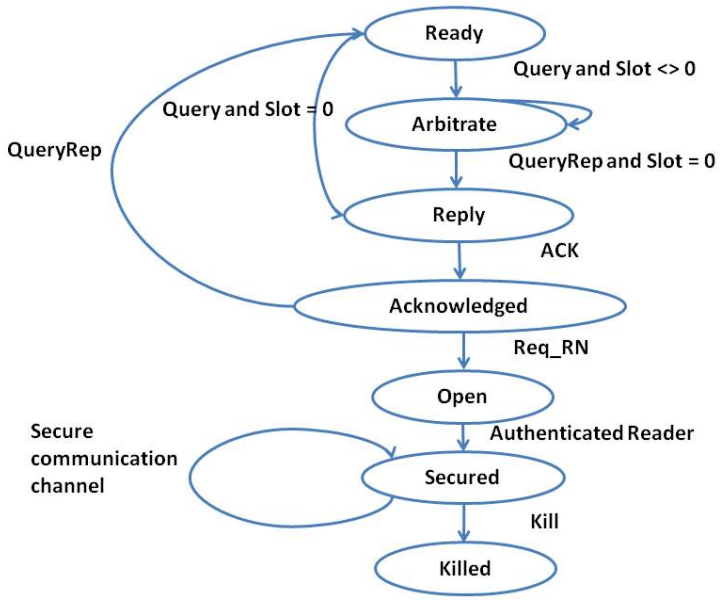

Figure 6.6. Modified State diagram for Mutual Authentication.

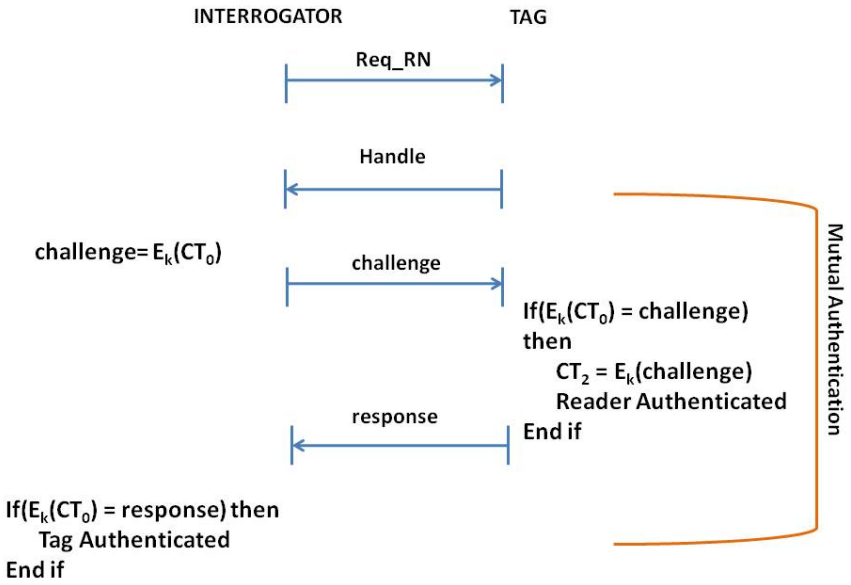

Figure 6.7. Design-1 for Mutual Authentication protocol.

verify the response by encrypting the cipher again. Hence a secure communication channel is established between the Tag and the Reader by a mutual authentication procedure using challenge-response technique.

A Detailed analysis of this design of the protocol shows that a Replay attack is possible on the Tag. A Replay attack is a form of network attack where a valid communication is recorded maliciously and replayed at a later stage to get unauthorized access to data. For example,Let us consider a sniffer which records a valid communication between a Tag and Reader. The sniffer can then act as malicious Tag and replay the whole communication again to get access to a Reader and keep it occupied. This can prove to be a threat as the Reader is occupied by a malicious Tag and is blocked from doing other valid communication. To overcome this problem a design B is proposed as explained below.

6.2.2.2 Design 2

This step is performed immediately after the inventory round. Then a Reader issues a *Req_RN* which requests a new RN64 (PT_1) and the Tag enters the open state. On receiving PT_1, the reader decrypts this PT_1 along with a new RN64 (PT_2). After receiving the challenge from the Reader, the Tag authenticates the Reader successfully and responds by encrypting the PT_2 along with a new RN64 (PT_3) and transitions to the Secured state. The Reader in turn authenticates the Tags response. The above described process is shown in the Figure 6.8.

The above explained technique holds good for AES in EBC mode since the block size is 128-bits. For the case of PRESENT and XTEA, a CBC mode of encryption/decryption is used to enable more security. The protocol when used in CBC mode in case of PRESENT and XTEA is described in Figure 6.9.

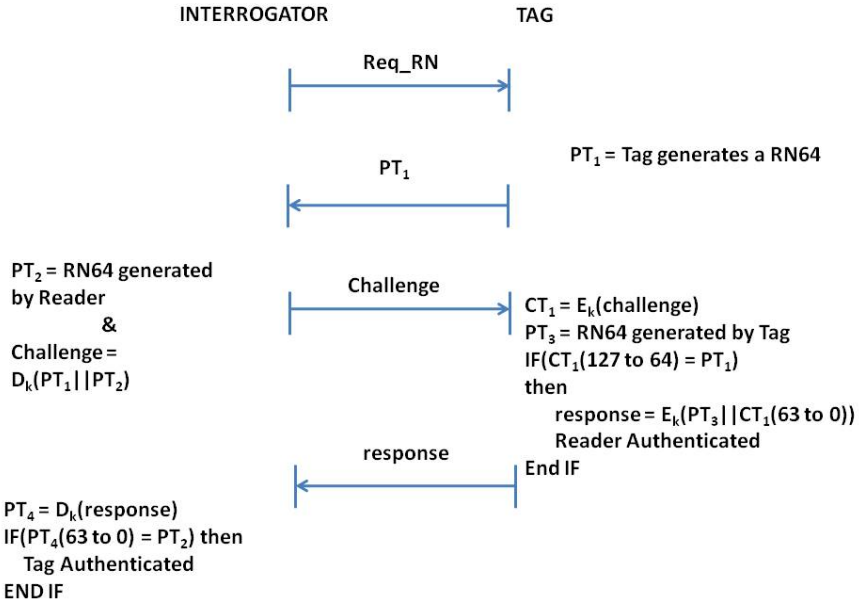

Figure 6.8. Design-2 for Mutual Authentication protocol.

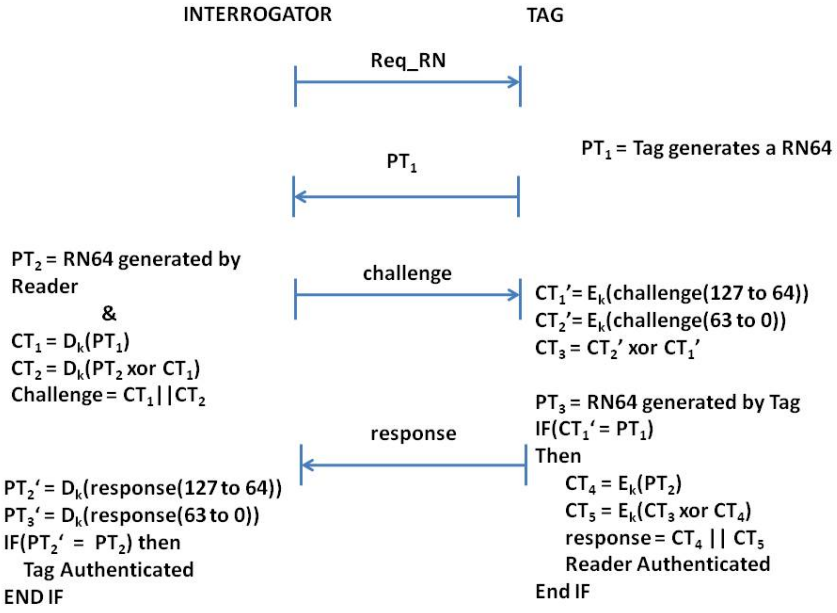

Figure 6.9. Design-2 for Mutual Authentication protocol in CBC mode.

The high level architecture of the protocol discussed is shown in the Figure 6.10. As seen from the diagram, the cipher engine is integrated in the Tag ID layer of the GEN2 protocol. The cipher engine here uses AES, PRESENT or XTEA algorithms for encryption.

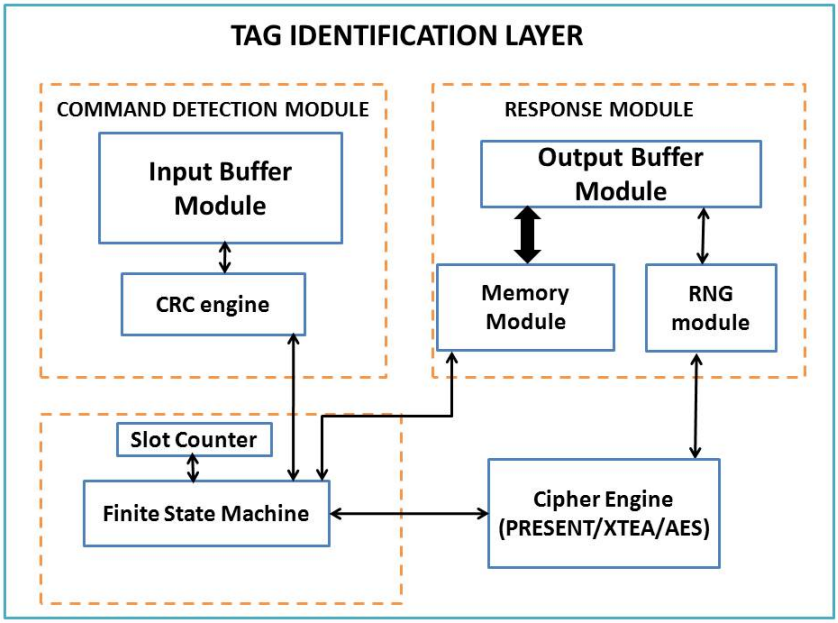

Figure 6.10. High Level Architecture of GEN2 with Security.

6.3 Summary

The secure-ID and mutual authentication protocol developed can be integrated with the GEN2 protocol to secure the identity of the Tag and also have a mutual authentication to validate the authenticity of the Tag and Reader. We can also have a secure communication channel for the Tag and the Reader to communicate. The security analysis of the protocol shows that the protocol is resistant common attacks including replay attack.

CHAPTER 7

RESULTS AND ANALYSIS

7.1 Introduction

In this chapter, the synthesis and simulation results of the implementation of GEN2, encryption algorithms and integration with security using VHDL are discussed. An analysis of the effect on performance of the RFID system due to integration of security for authentication and mutual identification is discussed. For synthesizing the VHDL code, Xilinx ISE 11.1 XST Tool is used which records the desired output values (particularly area, and max frequency). For the simulation of the synthesized code, Mentor graphics Modelsim 6.4a is used.

7.2 Simulation results and Timing analysis

The GEN2 protocol specifies a time T1 from the Interrogator transmission to the Tag response. All tag responses in the Inventory should be $< T1$. In a query command if DR and M are set to 8 and 1 respectively and Tari is 12.5μsec, we calculate the link rate to be 128Kbps and T1 to be 78.125μsec according to the GEN2 specification. At a typical 2 MHz on-chip clock rate, the T1 is 157 clocks and hence all the Tag responses should be less than this. Table **??** shows the simulation results for T1 constraint and we see that the requirement of 157 clocks is matched for Design A, Design B and Design C for AES, PRESENT and XTEA.

The mutual authentication is done after the secured tag identification process of the Tag and hence there is no T1 constraint requirement here. The Table 7.2 shows the timing analysis for all designs of mutual authentication process. Timing values in

Table 7.1. T1 Constraint for Tag Identification

	GEN2	**Design A (on fly)**		**Design B (on fly)**			**Design C (Power-up)**		
ObjectID	$PC\|\|EPC\|\|CRC$	$E_k(RN48\|\|CRC16)$		$E_k(secureID)$			$CT_2\|\|(EPC)XOR(CT_0\|\|CT_1)$		
		PRESENT	**XTEA**	**PRESENT**	**XTEA**	**AES**	**PRESENT**	**XTEA**	**AES**
# of clocks in time slot T1 before Transmitting RN16	18	18	18	18	18	18	18	18	18
# of clocks in time slot T1 before Transmitting ObjectID	6	98	98	43	43	88	6	6	6

the table signifies the minimum clocks the Reader has to wait to get response from the Tag. 152 clocks in this case.

Table 7.2. T1 Analysis for Mutual Authentication

	Design A		**Design B in open state**		
Challenge	$E_k(E_k(secureID))$		$E_k(PT_0\|\|PT_1))$		
Response	$E_k(E_k(E_k(secureID)))$		$E_k(PT_1\|\|PT_3))$		
	PRESENT	**XTEA**	**AES(EBC)**	**PRESENT(CBC)**	**XTEA(CBC)**
# of clocks to Authenticate Reader and send Response	72	72	152	140	140

7.3 Synthesis results and discussions

Table 7.3 shows the synthesis results of the ciphers i.e PRESENT, XTEA and AES take 32, 32 and 72 clock cycles respectively for encryption. As discussed earlier, AES and XTEA have more complexity compared to PRESENT and hence the synthesis results show that PRESENT is more hardware efficient compared to XTEA and AES.

Table 7.3. Simulation and Synthesis results

	GEN2	PRESENT(ours)	XTEA(ours)	AES(ours)
# of clock cycles	Refer Table 7.1	32	32	72
CLB Slices	22047	210	974	1785
Slice Flipflops	1963	149	211	948
4 input LUT's	42428	319	1769	3420
Max Frequency[MHz]	62.335	236.061	71.524	136.061
Power consumption[W]	-	0.289876602	0.290170474	0.291347327
Device	xc3s4000-5	xc3s4000-5	xc3s4000-5	xc3s4000-5

Table 7.4 and Table 7.5 shows the synthesis results of GEN2 with security. When the encryption operations are done on the fly the Table 7.4 shows that PRESENT requires less hardware compared to XTEA and AES. When the cipher engine does the encryption process on power-up the Table 7.5 shows that PRESENT requires less hardware compared to XTEA and AES. We observe that the power consumed by PRESNET is less than XTEA and AES and thus more suitable for low-power, resource constraint applications.

AES being most secure takes more hardware resources. Hence for resource constrained applications like RFID, PRESENT or XTEA implementations are preferred with some compromise with the security level.

Table 7.4. Synthesis results of GEN2 with security

	GEN2 with PRESENT	GEN2 with XTEA	GEN2 with AES
CLB Slices	21497	21792	24067
Slice Flipflops	2108	2132	3134
4 input LUT's	41151	41699	46234
Max Frequency[MHz]	43.717	55.115	54.376
Power consumption[W]	0.284547866	0.286606762	0.287885702
Device	xc3s4000-5	xc3s4000-5	xc3s4000-5

Table 7.5. Synthesis results of GEN2 with security : Power-up Implementation

	GEN2 with PRESENT	**GEN2 with XTEA**	**GEN2 with AES**
CLB Slices	21531	22085	25367
Slice Flipflops	2583	2584	3131
4 input LUT's	40930	41771	44488
Max Frequency[MHz]	45.289	46.258	51.546
Power consumption[W]	0.285706345	0.287588506	0.288279497
Device	xc3s4000-5	xc3s4000-5	xc3s4000-5

7.4 Power Analysis

The power analysis is done using a Xilinx recommended XPower Estimator(XPE) 11.1 tool. Implementations show that a single D-FF on Spartan xc3s4000-5 [60] consumes 0.271592377 Watts of power. Hence from the power values observed in Table 7.5 and 7.4 show that the implementation of GEN2 with security consumes around 14mW of power. But this value is very high compared to the power harvested by a passive UHF RFID Tag. Power-up implementation uses a CBC mode of encryption and hence more hardware resources are utilized as compared to on-fly implementation which adds to the power consumption.

7.5 System Performance Analysis

The system performance analysis of a passive UHF RFID Tag includes the time taken for a single Tag reply and the time taken for Tag identification process. A number of parameters that need to be considered to do the analysis are divide ratio(DR), M, TR_{cal}, RT_{cal}, Tari and Baud link frequency (BLF). The T1 timing is calculated using these parameters as given in the Equation 7.3.

$$T1 = max(RT_{cal}, 10 * T_{pri}) \tag{7.1}$$

$$T_{pri} = 1/BLF \tag{7.2}$$

$$BLF = DR/TR_{cal} \tag{7.3}$$

From the Table 7.1 we see that Design A, B and C need a maximum of 98, 43 and 18 clocks respectively to do the operations in the time slot T1. The change in T1 values for different BLFs at 2MHz chip clock rate is shown in the Table 7.6. Hence by comparing the theoretical and practical results we can make the following observations:

- When the encryption is done on the fly, Design A can satisfy the T1 requirements upto 300Kbps and Design B can work upto 450Kbps.
- When the encryption is done on power-up, Design C can satisfy the T1 requirements for all the link frequencies upto 600Kbps.

Table 7.6. BLF vs T1

BLF(kbps)	**T1(μsec)**	**T1(clks)**
128	78.125	157
227.5	52.734	106
256	39.0625	79
387.87	27.5	55
451.5	26.57	54
541.8	22.148	45
620.60	19.33	39

Table 7.7 shows the communication timing analysis for a single Tag reply at a communication rate of 128kbps. As seen from the table the following observations can be made:

- In the practical implementation of GEN2 the operations performed in the time slot T1 is very fast. This decrease in the time slot T1 yields a better performance of the Tag as compared to the theoretical analysis.
- In the Design A, the number of bits being backscattered in response to the ACK is very less. This reduces the time taken for Tag identification. But as analyzed in the previous chapter this design has some flaws and hence less secure.
- In the Design B, the integration of security has not made much impact on the performance of Tag identification.
- In the Design C, the length of the objectID backscattered in response to the ACK is more which increases the time taken to complete Tag identification.

Table 7.7. Performance Analysis: Tag Identification @ 128Kbps

Tari = 12.5μs, DR=8, M=1, BLF = 128Kbps, Chip Clock rate = 2MHz					
	GEN2 Theoretical	**Design B**		**Design C**	
Length of ObjectID	128bits	Present and XTEA - 128bits	AES - 192bits	Present and XTEA - 224bits	AES - 288bits
Query(μsec)	462.5	462.5	462.5	462.5	462.5
T1(μsec)	78.125	78.125	78.125	78.125	78.125
RN16(μsec)	203.125	203.125	203.125	203.125	203.125
T2(μsec)	93.75	93.75	93.75	93.75	93.75
ACK(μsec)	337.5	337.5	337.5	337.5	337.5
T1(μsec)	78.125	78.125	78.125	78.125	78.125
SecureID(μsec)	1078.125	1078.125	1578.125	1828.125	2328.125
T2(μsec)	156.25	156.25	156.25	156.25	156.25
Total Time for Tag Identification(μsec)	2487.5	2487.5	2987.5	3237.5	3737.5

Table 7.8 shows the communication timing analysis for the Tag identification process at a communication rate of 128kbps. The system performance depends on the length of the challenge and response commands used for mutual authentication in the protocol.

Table 7.8. Performance Analysis: Tag Identification with mutual auth.

Tari = 12.5μs, DR=8, M=1, BLF = 128Kbps, Chip Clock rate = 2MHz					
	GEN2 Theoretical	**Design B**		**Design C**	
Length of ObjectID	128bits	Present and XTEA - 128bits	AES - 192bits	Present and XTEA - 224bits	AES - 288bits
Query(μsec)	462.5	462.5	462.5	462.5	462.5
T1(μsec)	78.125	78.125	78.125	78.125	78.125
RN16(μsec)	203.125	203.125	203.125	203.125	203.125
T2(μsec)	93.75	93.75	93.75	93.75	93.75
ACK(μsec)	337.5	337.5	337.5	337.5	337.5
T1(μsec)	78.125	78.125	78.125	78.125	78.125
SecureID(μsec)	1078.125	1078.125	1578.125	1828.125	2328.125
T2(μsec)	156.25	156.25	156.25	156.25	156.25
Req_RN(μsec)	675	675	675	675	675
T1(μsec)	78.125	78.125	78.125	78.125	78.125
Handle(μsec)	328.125	328.125	328.125	578.125	578.125
T2(μsec)	93.75	93.75	93.75	93.75	93.75
challenge(μsec)	-	2181.25	2181.25	2181.25	2181.25
T1(μsec)	-	78.125	78.125	78.125	78.125
Response(μsec)	-	1078.125	1078.125	1078.125	1078.125
Total Time for Tag Identification with Authentication(μsec)	3662.5	7008.5	7750	8000	8500

7.6 Summary

The synthesis and simulation of all the implementations are done on a Spartan 3 target device using Xilinx ISE and modelsim for simulation. The results show that all the three implementations of Present, AES and XTEA with GEN2 satisfy the "T1" constraint specified by the GEN2 protocol. The synthesis results show that PRESENT when integrated with GEN2 protocol is very efficient as it utilizes less resources and has low power consumption as compared to XTEA and AES. The performance of the Tag identification depends on the length of the object ID transmitted in response to the interrogation from the Reader.

CHAPTER 8

CONCLUSIONS

The SecureID protocols proposed in this paper enable a low cost solution for many applications where privacy and security has been a major threat. These protocols enable not just the encryption of the tag contents, but also a way to secure the identification along with mutual authentication process. The implementation of this protocol has small area and low power consumption which enables the feasibility to implement within GEN2. It also meets the T1 timing boundary specified by the EPC Class 1 GEN 2 protocol for the Tag.

Using strong ciphers such has AES, PRESENT and XTEA has proved that this implementation is feasible for passive UHF RFID technology. The VHDL implementation of the secureID protocol presented in this paper may be targeted for inclusion in an ASIC, FPGA, CPLD, or other programmable logic device and is technology independent i.e., no technology-specific or device-specific code constructs have been used, nor should be required.

From the experimental results, PRESENT and XTEA encrypts the plaintext in 32 clock cycles and AES in 72 clock cycles. Thus the integration of AES, PRESENT and XTEA for the identification process will meet the timing (T1) requirements of the GEN2 specification where the encryption is done on the fly or power-up. However, PRESENT consumes fewer hardware resources when compared to XTEA or AES, and is thus more suitable for passive UHF RFID technology. Also it is observed that the power consumption is less in PRESENT as compared to AES and XTEA and this more suitable for resource constraint applications.

The smallest tag response parameter given in the GEN2 protocol is 19.33μs, and the encryption of plaintext in AES, PRESENT and XTEA takes 36μs, 16μs and 16μs at 2MHz on-chip clock rate respectively. Thus, we can integrate ciphers into GEN2 identification and authentication process for a communication rate upto 450Kbps and 600Kbps for Design B and C respectively without slowing down the process.

My future research is focused on optimizing the VHDL modules to make it more efficient in terms of area, power and speed. Intensive security analysis of the protocol needs to be done to make it more secure. Other lightweight ciphers like Grain, ICEBERG, HIGHT, PUFFIN etc can be considered for implementation within the GEN2 protocol.

APPENDIX A

VHDL SOURCE CODE

Figure A.1 shows the detailed architecture of the GEN2 implementation in VHDL with security. The architecture gives a clear picture of integration of different modules. The VHDL source code of the power-up implementation (Design C) for securing the Tag's identity and Design 1 for mutual authentication is presented below.

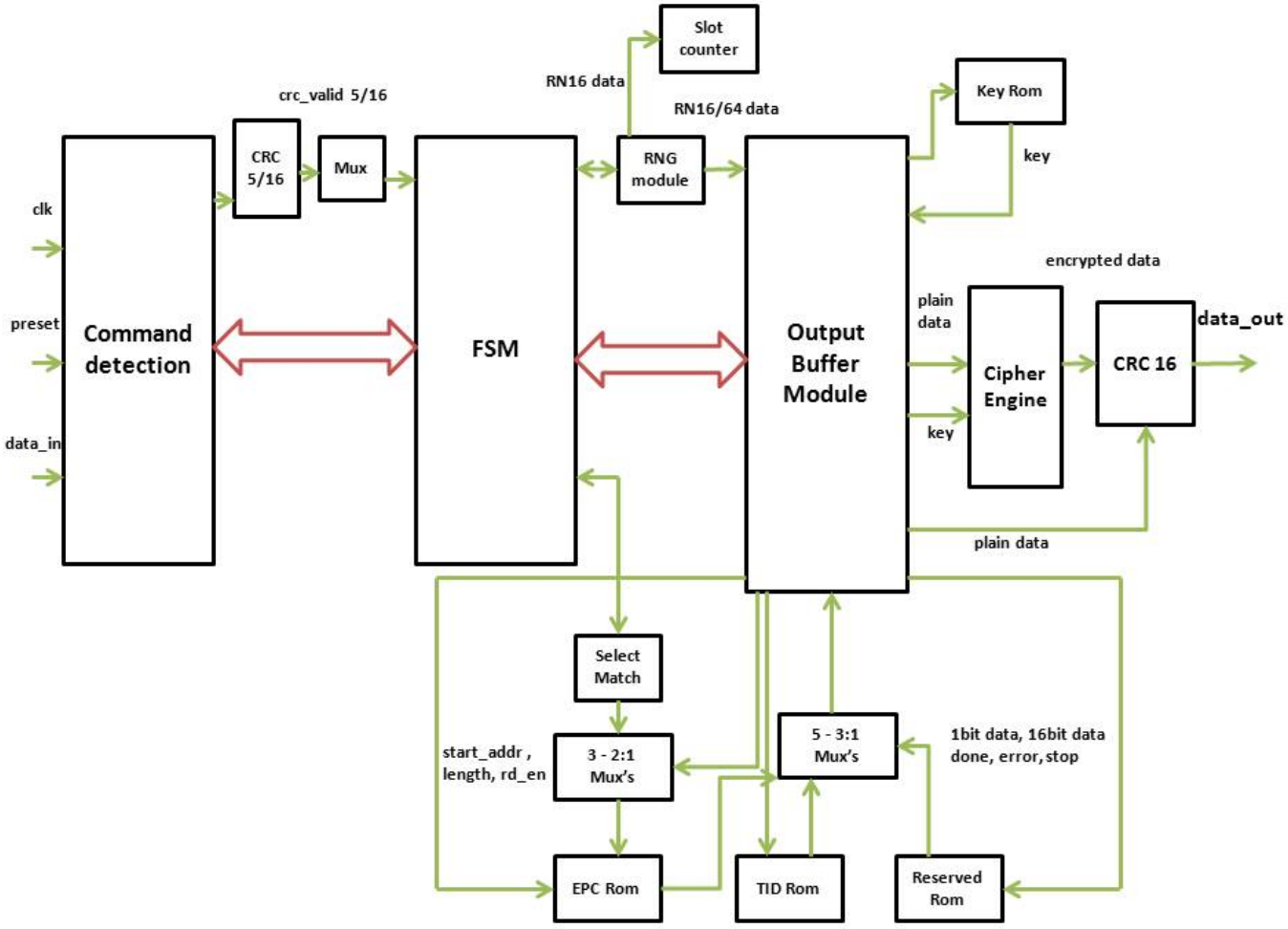

Figure A.1. Detailed Architecture of GEN2 with security.

Listing A.1. GEN2 with PRESENT - Top Level Entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity GEN2_Top is
port(
clk                     : in std_logic;
data_in          : in std_logic;
data_out         : out std_logic
);
end entity GEN2_Top;

architecture GEN2_Top_behavioral of GEN2_Top is

signal bit_len_sgl : STD_LOGIC_VECTOR(8 downto
    0);
signal stop_sgl, Out_Serial_sgl, crc_valid_sgl,
    CRC_valid_5_sgl, CRC_valid_16_sgl,
    mask_done_sgl, Reserved_done_sgl,
    match_sel_sgl, crc_flag_buff_sgl,
    stop_mem_sgl, send_mem_sgl: STD_LOGIC;
signal rd_wr_lock_done_sig, len_16_bit_mem_sgl,
    data_mem_bit_sgl, mem_error_sgl, rd_en_sgl,
    wr_en_sgl, data_sgl, start_crc_sgl,
    RN16_done_sgl, slot_done_sgl : STD_LOGIC;

signal mem_error_sgl_EPC, mem_error_sgl_RES,
    mem_error_sgl_TID, mem_error_sgl_USER,
    data_crc_sgl, start_bit_sgl, preset,
    powerup_sgl : STD_LOGIC;
signal stop_mem_sgl_EPC, stop_mem_sgl_RES,
    stop_mem_sgl_TID, stop_mem_sgl_USER :
    STD_LOGIC;
signal rd_wr_lock_done_sig_EPC,
    rd_wr_lock_done_sig_RES,
    rd_wr_lock_done_sig_TID,
    rd_wr_lock_done_sig_USER : STD_LOGIC;
signal data_mem_bit_sgl_EPC, data_mem_bit_sgl_RES
    , data_mem_bit_sgl_TID,
    data_mem_bit_sgl_USER : STD_LOGIC;
signal data_mem_sgl_EPC, data_mem_sgl_RES,
    data_mem_sgl_TID, data_mem_sgl_USER :
    STD_LOGIC_VECTOR(15 downto 0);

signal len_16_bit_buff_sgl, len_16_bit_SM_sgl,
    len_16_bit_buff_SM_sgl, key_crc_done_sgl,
    EPC_done_sgl : STD_LOGIC;
signal rd_en_buff_sgl, rd_en_SM_sgl,
    rd_en_buff_SM_sgl : STD_LOGIC;
signal start_address_buff_sgl,
    start_address_match_sgl,
    start_address_select_buff_sgl :
    STD_LOGIC_VECTOR(23 downto 0);
signal EPC_sgl : STD_LOGIC_VECTOR(95 downto 0);

signal key_access_sgl, key_done_sgl, key_crc_sgl,
    RN_64_flag_sgl, RN64_done_sgl, crypt_en_sgl,
    preset_engine_sgl, data_64_bit_sgl,
    cipher_done_sgl : std_logic;
signal data_key_sgl: std_logic_vector(127 downto
    0);
signal crc_plain_text_sgl: std_logic_vector(15
    downto 0);
signal RN_64_sgl, challenge_data_sgl:
    std_logic_vector(63 downto 0);
signal cipher_text_sgl, data_plain_text_sgl:
    std_logic_vector(63 downto 0);

signal use_epc_sgl, done_sgl, no_reply_sgl :
    STD_LOGIC;
signal RN_16_flag_sgl, truncate_sgl,
    start_match_sgl, start_buffer_sgl,
    kill_count_sgl, Reserved_access_sgl,
    OpenSecureMemRef_sgl, error_code_sgl:
    STD_LOGIC;
signal what_cmd_sgl : STD_LOGIC_VECTOR(4 downto
    0);
signal Q_value_sgl : STD_LOGIC_VECTOR(3 downto
    0);
signal target_sel_sgl, action_sel_sgl:
    STD_LOGIC_VECTOR(2 downto 0);
signal Lock_Kill_sgl, Lock_Access_sgl,
    Lock_EPC_sgl, Lock_TID_sgl, Lock_User_sgl :
    STD_LOGIC_VECTOR(1 downto 0);
signal Lock_Kill_buff_sgl, Lock_Access_buff_sgl,
    Lock_EPC_buff_sgl, Lock_TID_buff_sgl,
    Lock_User_buff_sgl : STD_LOGIC_VECTOR(1
    downto 0);
signal MemBnk_sgl, MemBnk_buff_sgl, membnk_sel_sgl
    , crc_flag_sgl, EBV_write_sgl, EBV_read_sgl,
    EBV_sel_sgl: STD_LOGIC_VECTOR(1 downto 0);
signal pointer_sel_sgl, start_address_sgl,
    end_address_sgl, end_address_buff_sgl:
    STD_LOGIC_VECTOR(23 downto 0);
signal Mask_len_sgl: STD_LOGIC_VECTOR(7 downto
    0);
signal Mask_value_sgl: STD_LOGIC_VECTOR(255
    downto 0);
signal Q_sgl : STD_LOGIC_VECTOR(351 downto 0);
signal RN_16_sgl, slot_count_sgl, data_in_16_sgl,
    WrData_out_sgl, storedCRC_sgl, data_mem_sgl,
    write_mem_data_sgl, slot_15_sgl:
    std_logic_vector(15 downto 0);

component present is
port(
clk                     : in std_logic;
n_reset          : in std_logic;
crypt_en         : in std_logic;
preset_engine: in std_logic;
key                     : in std_logic_vector
    (127 downto 0);
data_out         : out std_logic_vector(63 downto
    0);
data_in          : in std_logic_vector(63 downto
    0);
done             : out std_logic
);
end component present;


component CMD_Detect_23 is
port(
data, done, stop_in : IN STD_LOGIC;
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
bit_len_out: OUT STD_LOGIC_VECTOR(8 downto 0);
stop: OUT STD_LOGIC;
Out_Serial : OUT STD_LOGIC;
what_cmd: OUT STD_LOGIC_VECTOR(4 downto 0);
target_sel, action_sel: OUT STD_LOGIC_VECTOR(2
    downto 0);
membnk_sel, crc_flag, EBV_write, EBV_read,
    EBV_sel: OUT STD_LOGIC_VECTOR(1 downto 0);
pointer_sel: OUT STD_LOGIC_VECTOR(23 downto 0);
Mask_len: OUT STD_LOGIC_VECTOR(7 downto 0);
Mask_value: OUT STD_LOGIC_VECTOR(255 downto 0);
Q : OUT STD_LOGIC_VECTOR(351 downto 0)
);
end component CMD_Detect_23;

component FSM_1 is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset, powerup: in std_logic;
done, crc_valid, mask_done, Reserved_done,
    match_sel, stop, RN16_done, slot_done,
    RN64_done: in std_logic;
ebv, ebv_read, ebv_write, membnk_sel: in
    std_logic_vector(1 downto 0);
target_sel, action_sel: in std_logic_vector(2
    downto 0);
wht_cmd: in std_logic_vector(4 downto 0);
mask_length: in std_logic_vector(7 downto 0);
bit_len_in: in std_logic_vector(8 downto 0);
RN_16_in, slot_count, data_in_16: in
    std_logic_vector(15 downto 0);
pointer_sel: in std_logic_vector(23 downto 0);
data_in: in std_logic_vector(351 downto 0);
-- OUT PORTS --
RN_16, RN_64, truncate, start_match, start_buffer,
    kill_count_out, Reserved_access, no_reply:
    out std_logic;
EPC_done: out std_logic:='0';
EPC: out std_logic_vector(95 downto 0);
MemBnk_out: out std_logic_vector(1 downto 0);
OpenSecureMemRef, error_code: out std_logic;
```

```
WrData_out, storedCRC: out std_logic_vector(15
    downto 0);
Q_value_out: out std_logic_vector(3 downto 0);
start_address_out, end_address_out: out
    std_logic_vector(23 downto 0);
challenge_data : out std_logic_vector(63 downto
    0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: out std_logic_vector(1 downto 0)
);
end component FSM_1;

component Test_CRC is
port(
data, stop,done : IN STD_LOGIC;
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
crc_valid : OUT STD_LOGIC;
Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
end component Test_CRC;

component CRC16 is
port(
Preset : IN STD_LOGIC;
clk , powerup: IN STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
data, stop,done,key_crc : IN STD_LOGIC;
crc_valid,key_crc_done : OUT STD_LOGIC;
crc_plain_text: out std_logic_vector(15 downto
    0);
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
end component CRC16;

component CRC16_final is
port(
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
data : IN STD_LOGIC;
start_crc, crc_flag,start_bit : IN STD_LOGIC;
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC;
data_bit_out, done : OUT STD_LOGIC
);
end component CRC16_final;

component slot_counter is
port(
--clk: in std_logic;--clock signal.
Preset: in std_logic;
Q_value_in: in std_logic_vector(3 downto 0);
RN_16_in: in std_logic_vector(15 downto 0);
slot_value_out: out std_logic_vector(15 downto
    0);
slot_done : out std_logic
);
end component slot_counter;

component select_match is
port(
clk: in std_logic;--clock signal.
Preset,mask_match,rd_done, powerup: in std_logic
    ;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address :in std_logic_vector(23 downto 0);
Mask_len: in STD_LOGIC_VECTOR(7 downto 0);
Mask_value_in: in STD_LOGIC_VECTOR(255 downto 0)
    ;
data: in STD_LOGIC_VECTOR(15 downto 0);
rd_en,match_valid,data_16_bit,select_done,
    use_epc : out std_logic;
start_address_out:out STD_LOGIC_VECTOR(23 downto
     0)
);
end component select_match;

component RN_16 is
port(
preset, clk, RN16_flag, RN64_flag: in std_logic;
lfsr_out,slot_15 : out std_logic_vector (15
    downto 0);
RN_64_data : out std_logic_vector(63 downto 0);
RN16_done,RN64_done : out std_logic
);
end component RN_16;

component buffer_module is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset,key_done, powerup: in std_logic;
start_buffer,truncate,error_code,rd_wr_lock_done
    ,kill_count,Reserved_access,rn_16,
    data_bit_in,mem_error,stop,use_epc_SM,
    no_reply,cipher_done: in std_logic;
wht_cmd: in std_logic_vector(4 downto 0);
start_address,end_address: in std_logic_vector
    (23 downto 0);
mem_bank_in: in std_logic_vector(1 downto 0);
data_16_in, rn16_data, stored_crc_in: in
    std_logic_vector(15 downto 0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: in std_logic_vector(1 downto 0);
EPC: in std_logic_vector(95 downto 0);
EPC_done: in std_logic;
key_crc_done,rn_64: in std_logic;
data_key: in std_logic_vector(127 downto 0);
rn64_data,challenge_data: in std_logic_vector(63
     downto 0);
crc_plain_text: in std_logic_vector(15 downto 0)
    ;
cipher_text : in std_logic_vector(63 downto 0);


-- OUT PORT
data_bit_out,rd_en,wr_en,reserved_done,
    data_16_bit,crc_flag,send_mem, start_crc,
    data_64_bit,key_crc,crypt_en,preset_engine,
    key_access,start_bit: out std_logic;
data_out: out std_logic_vector(15 downto 0);
mem_bank_out: out std_logic_vector(1 downto 0);
Lock_Kill_out, Lock_Access_out, Lock_EPC_out,
    Lock_TID_out, Lock_User_out: out
    std_logic_vector(1 downto 0);
data_plain_text : out std_logic_vector(63 downto
     0);
start_address_out,end_address_out: out
    std_logic_vector(23 downto 0)
);
end component buffer_module;

component EPC_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_value,mem_bank : in std_logic_vector(1
    downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop : out
     std_logic
);
end component EPC_ROM;

component Reserved_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_kill_value, lock_access_value,mem_bank : in
     std_logic_vector(1 downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
    std_logic
);
end component Reserved_ROM;

component TID_ROM is
```

```
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_value,mem_bank : in std_logic_vector(1
    downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
    std_logic
);
end component TID_ROM;

component mux_data16 is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic_vector(15 downto 0);
D_RES    : in std_logic_vector(15 downto 0);
D_TID    : in std_logic_vector(15 downto 0);
D_USER   : in std_logic_vector(15 downto 0);
mux_out: out std_logic_vector(15 downto 0)
);
end component mux_data16;

component mux_1bit is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit;

component mux_1bit_done is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_done;

component mux_1bit_error is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_error;

component mux_1bit_stop is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_stop;

component mux_1bit_crc is
port(
clk   : in std_logic;
crc_sel: in std_logic_vector(1 downto 0);
CRC_valid_5      : in std_logic;
CRC_valid_16: in std_logic;
mux_out : out std_logic
);
end component mux_1bit_crc;

component mux_start_addr is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
start_address_Buffer: in std_logic_vector(23
    downto 0);
start_address_SM: in std_logic_vector(23 downto
    0);
mux_out : out std_logic_vector(23 downto 0)
);
end component mux_start_addr;

component mux_rd_en is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
rd_en_buffer: in std_logic;
rd_en_SM: in std_logic;
mux_out : out std_logic
);
end component mux_rd_en;

component mux_length_16_bit is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
data_16_bit_buffer: in std_logic;
data_16_bit_SM: in std_logic;
mux_out : out std_logic
);
end component mux_length_16_bit;

component key_rom is
port (  clk: in std_logic;--clock signal.
Preset: in std_logic;
key_access : in std_logic;
key_done : out std_logic;
data_key : out std_logic_vector(127 downto 0)
);
end component key_rom;

component powerup_count is
port(
clk      : in std_logic;
--n_reset,crypt_en: in std_logic;
powerup,preset   : out std_logic
);

end component powerup_count;

begin

present80 : present
port map(
clk      => clk,
n_reset => preset,
crypt_en => crypt_en_sgl,
preset_engine => preset_engine_sgl,
key      => data_key_sgl,
data_out => cipher_text_sgl,
data_in => data_plain_text_sgl,
done => cipher_done_sgl
);

cmd_dec : CMD_Detect_23
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_in,
stop_in => stop_sgl,
bit_len_out => bit_len_sgl,
stop => stop_sgl,
done => done_sgl,
Out_Serial => Out_Serial_sgl,
what_cmd => what_cmd_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
membnk_sel => membnk_sel_sgl,
crc_flag => crc_flag_sgl,
EBV_write => EBV_write_sgl,
EBV_read => EBV_read_sgl,
EBV_sel => EBV_sel_sgl,
pointer_sel => pointer_sel_sgl,
Mask_len => Mask_len_sgl,
Mask_value => Mask_value_sgl,
```

```
Q => Q_sgl
);

FSM : FSM_1
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
done => done_sgl,
RN16_done => RN16_done_sgl,
RN64_done => RN64_done_sgl,
slot_done => slot_done_sgl,
crc_valid => crc_valid_sgl,
stop => stop_sgl,
mask_done => mask_done_sgl,
Reserved_done => Reserved_done_sgl,
match_sel => match_sel_sgl,
ebv => EBV_sel_sgl,
ebv_read => EBV_read_sgl,
ebv_write => EBV_write_sgl,
membnk_sel => membnk_sel_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
wht_cmd => what_cmd_sgl,
Mask_length => Mask_len_sgl,     -- to be given
    to Select match also
bit_len_in => bit_len_sgl,
RN_16_in => RN_16_sgl,           --data
slot_count => slot_count_sgl,
data_in_16 => data_in_16_sgl,
pointer_sel => pointer_sel_sgl,
data_in => Q_sgl,

-- OUT PORTS
RN_16 => RN_16_flag_sgl,                 -- Flag
    to intiate RN16
RN_64 => RN_64_flag_sgl,
truncate => truncate_sgl,
start_match => start_match_sgl,
start_buffer => start_buffer_sgl,
kill_count_out => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
MemBnk_out => MemBnk_sgl,
OpenSecureMemRef => OpenSecureMemRef_sgl,
error_code => error_code_sgl,
WrData_out => WrData_out_sgl,
storedCRC => storedCRC_sgl,
Q_value_out => Q_value_sgl,
start_address_out => start_address_sgl,
end_address_out => end_address_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,
Lock_User => Lock_User_sgl
);

CRC5 : Test_CRC
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
stop => stop_sgl,
done => done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_5_sgl
);

CRC_16 : CRC16
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
key_crc => key_crc_sgl,
crc_plain_text => crc_plain_text_sgl,
stop => stop_sgl,
done => done_sgl,
key_crc_done => key_crc_done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_16_sgl
);

CRC16_end : CRC16_final
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_sgl,
start_bit => start_bit_sgl,
start_crc => start_crc_sgl,
crc_flag => crc_flag_buff_sgl,
done => done_sgl,
data_bit_out => data_out
);

slot : slot_counter
port map(
--clk => clk,
preset => preset,
Q_value_in => Q_value_sgl,
RN_16_in => slot_15_sgl,                    --data (
    same as FSM)
slot_value_out => slot_count_sgl,
slot_done => slot_done_sgl
);

match_select : select_match
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
mask_match => start_match_sgl,
rd_done => rd_wr_lock_done_sig,
start_address => start_address_sgl,
end_address => end_address_sgl,
Mask_len => Mask_len_sgl,
Mask_value_in => Mask_value_sgl,
data => data_mem_sgl_EPC,
rd_en => rd_en_SM_sgl,
use_epc => use_epc_sgl,
match_valid => match_sel_sgl,
data_16_bit => len_16_bit_SM_sgl,
select_done => mask_done_sgl,
start_address_out => start_address_match_sgl
);

RN16 : RN_16
port map(
clk => clk,
preset => preset,
RN16_flag => RN_16_flag_sgl,
RN64_flag => RN_64_flag_sgl,
lfsr_out => RN_16_sgl,
RN_64_data => RN_64_sgl,
slot_15 => slot_15_sgl,
RN64_done => RN64_done_sgl,
RN16_done => RN16_done_sgl
);

response : buffer_module
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
start_buffer => start_buffer_sgl,
use_epc_SM => use_epc_sgl,
truncate => truncate_sgl,
error_code => error_code_sgl,
rd_wr_lock_done => rd_wr_lock_done_sig,
kill_count => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
rn_16 => RN16_done_sgl,
data_bit_in => data_mem_bit_sgl,
mem_error => mem_error_sgl,
stop => stop_mem_sgl,
wht_cmd => what_cmd_sgl,
start_address => start_address_sgl,
end_address => end_address_sgl,
mem_bank_in => MemBnk_sgl,
data_16_in => data_mem_sgl,
rn16_data => RN_16_sgl,
stored_crc_in => storedCRC_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
```

```
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
Lock_User => Lock_User_sgl,
key_crc_done => key_crc_done_sgl,
RN_64 => RN64_done_sgl,
data_key => data_key_sgl,
rn64_data => RN_64_sgl,
crc_plain_text => crc_plain_text_sgl,
cipher_done => cipher_done_sgl,
cipher_text => cipher_text_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,

-- OUT PORTS

data_bit_out => data_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
reserved_done => Reserved_done_sgl,
data_16_bit => len_16_bit_buff_sgl,
crc_flag => crc_flag_buff_sgl,
send_mem => send_mem_sgl,
start_crc => start_crc_sgl,
data_out => data_in_16_sgl,
mem_bank_out => MemBnk_buff_sgl,
Lock_Kill_out => Lock_Kill_buff_sgl,
Lock_Access_out => Lock_Access_buff_sgl,
Lock_EPC_out => Lock_EPC_buff_sgl,
Lock_TID_out => Lock_TID_buff_sgl,
Lock_User_out => Lock_User_buff_sgl,
start_address_out => start_address_buff_sgl,
key_done => key_done_sgl,
data_64_bit => data_64_bit_sgl,
key_crc => key_crc_sgl,
crypt_en => crypt_en_sgl,
start_bit => start_bit_sgl,
key_access => key_access_sgl,
preset_engine => preset_engine_sgl,
data_plain_text  => data_plain_text_sgl,
end_address_out => end_address_buff_sgl
);

EPC : EPC_Rom
port map(
-- IN PORTS
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_select_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_EPC_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_SM_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_SM_sgl,

-- OUT PORTS
data_out_bit => data_mem_bit_sgl_EPC, -- Bit o/p
     from mem
data_out_16 => data_mem_sgl_EPC, -- Word o/p
     from mem
rd_wr_lock_done => rd_wr_lock_done_sig_EPC,
error => mem_error_sgl_EPC,
stop => stop_mem_sgl_EPC
);

Reserved : Reserved_ROM
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_kill_value => Lock_Kill_buff_sgl,
lock_access_value => Lock_Access_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_RES, -- Bit o/p
     from mem
data_out_16 => data_mem_sgl_RES, -- Word o/p
     from mem
rd_wr_lock_done => rd_wr_lock_done_sig_RES,
error => mem_error_sgl_RES,
stop => stop_mem_sgl_RES
);

TID : TID_Rom
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_TID_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_TID, -- Bit o/p
     from mem
data_out_16 => data_mem_sgl_TID, -- Word o/p
     from mem
rd_wr_lock_done => rd_wr_lock_done_sig_TID,
error => mem_error_sgl_TID,
stop => stop_mem_sgl_TID
);

Key : key_rom
port map(
clk => clk,
Preset => preset,
key_access => key_access_sgl,
key_done => key_done_sgl,
data_key => data_key_sgl
);

mux16_mod : mux_data16
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => data_mem_sgl_EPC,
D_RES   => data_mem_sgl_RES,
D_TID   => data_mem_sgl_TID,
D_USER  => data_mem_sgl_USER,
mux_out => data_mem_sgl
);


mux1bit_mod : mux_1bit
port map(
clk => clk,
membnk_sel=> MemBnk_buff_sgl,
D_EPC   => data_mem_bit_sgl_EPC,
D_RES   => data_mem_bit_sgl_RES,
D_TID   => data_mem_bit_sgl_TID,
D_USER  => data_mem_bit_sgl_USER,
mux_out => data_mem_bit_sgl
);

mux1bit_done_mod : mux_1bit_done
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => rd_wr_lock_done_sig_EPC,
D_RES   => rd_wr_lock_done_sig_RES,
D_TID   => rd_wr_lock_done_sig_TID,
D_USER  => rd_wr_lock_done_sig_USER,
mux_out => rd_wr_lock_done_sig
);

mux1bit_error_mod : mux_1bit_error
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => mem_error_sgl_EPC,
D_RES   => mem_error_sgl_RES,
D_TID   => mem_error_sgl_TID,
D_USER  => mem_error_sgl_USER,
mux_out => mem_error_sgl
);

mux1bit_stop_mod : mux_1bit_stop
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => stop_mem_sgl_EPC,
```

```
D_RES    => stop_mem_sgl_RES,
D_TID    => stop_mem_sgl_TID,
D_USER   => stop_mem_sgl_USER,
mux_out => stop_mem_sgl
);

mux1bit_crc_mod : mux_1bit_crc
port map(
clk      => clk,
crc_sel => crc_flag_sgl,
CRC_valid_5      => CRC_valid_5_sgl,
CRC_valid_16 => CRC_valid_16_sgl,
mux_out => crc_valid_sgl
);

mux_start_addr_mod : mux_start_addr
port map(
clk => clk,
select_match_sel => use_epc_sgl,
start_address_Buffer => start_address_buff_sgl,
start_address_SM => start_address_match_sgl,
mux_out => start_address_select_buff_sgl
);

mux_rd_en_mod : mux_rd_en
port map(
clk => clk,
select_match_sel => use_epc_sgl,
rd_en_buffer => rd_en_buff_sgl,
rd_en_SM => rd_en_SM_sgl,
mux_out => rd_en_buff_SM_sgl
);

mux_length_16_bit_mod : mux_length_16_bit
port map(
clk => clk,
select_match_sel => use_epc_sgl,
data_16_bit_buffer => len_16_bit_buff_sgl,
data_16_bit_SM => len_16_bit_SM_sgl,
mux_out => len_16_bit_buff_SM_sgl
);
powerup : powerup_count
port map(
clk      => clk,
preset => preset,
powerup => powerup_sgl
);

end architecture GEN2_Top_behavioral;
```

Listing A.2. GEN2 with XTEA - Top Level Entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity GEN2_Top is
port(
clk      : in std_logic;
data_in          : in std_logic;
data_out         : out std_logic
);
end entity GEN2_Top;

architecture GEN2_Top_behavioral of GEN2_Top is

signal bit_len_sgl : STD_LOGIC_VECTOR(8 downto
    0);
signal stop_sgl, Out_Serial_sgl, crc_valid_sgl,
    CRC_valid_5_sgl, CRC_valid_16_sgl,
    mask_done_sgl, Reserved_done_sgl,
    match_sel_sgl, crc_flag_buff_sgl,
    stop_mem_sgl, send_mem_sgl: STD_LOGIC;
signal rd_wr_lock_done_sig, len_16_bit_mem_sgl,
    data_mem_bit_sgl, mem_error_sgl, rd_en_sgl,
    wr_en_sgl, data_sgl, start_crc_sgl,
    RN16_done_sgl, slot_done_sgl : STD_LOGIC;

signal mem_error_sgl_EPC, mem_error_sgl_RES,
    mem_error_sgl_TID, mem_error_sgl_USER,
    data_crc_sgl, start_bit_sgl, preset,
    powerup_sgl : STD_LOGIC;
signal stop_mem_sgl_EPC, stop_mem_sgl_RES,
    stop_mem_sgl_TID, stop_mem_sgl_USER :
    STD_LOGIC;
signal rd_wr_lock_done_sig_EPC,
    rd_wr_lock_done_sig_RES,
    rd_wr_lock_done_sig_TID,
    rd_wr_lock_done_sig_USER : STD_LOGIC;
signal data_mem_bit_sgl_EPC, data_mem_bit_sgl_RES
    , data_mem_bit_sgl_TID,
    data_mem_bit_sgl_USER : STD_LOGIC;
signal data_mem_sgl_EPC, data_mem_sgl_RES,
    data_mem_sgl_TID, data_mem_sgl_USER :
    STD_LOGIC_VECTOR(15 downto 0);

signal len_16_bit_buff_sgl, len_16_bit_SM_sgl,
    len_16_bit_buff_SM_sgl, key_crc_done_sgl,
    EPC_done_sgl : STD_LOGIC;
signal rd_en_buff_sgl, rd_en_SM_sgl,
    rd_en_buff_SM_sgl : STD_LOGIC;
signal start_address_buff_sgl,
    start_address_match_sgl,
    start_address_select_buff_sgl :
    STD_LOGIC_VECTOR(23 downto 0);
signal EPC_sgl : STD_LOGIC_VECTOR(95 downto 0);

signal key_access_sgl, key_done_sgl, key_crc_sgl,
    RN_64_flag_sgl, RN64_done_sgl, crypt_en_sgl,
    preset_engine_sgl, data_64_bit_sgl,
    cipher_done_sgl : std_logic;
signal data_key_sgl: std_logic_vector(127 downto
    0);
signal crc_plain_text_sgl: std_logic_vector(15
    downto 0);
signal RN_64_sgl, challenge_data_sgl:
    std_logic_vector(63 downto 0);
signal cipher_text_sgl, data_plain_text_sgl:
    std_logic_vector(63 downto 0);

signal use_epc_sgl, done_sgl, no_reply_sgl :
    STD_LOGIC;
signal RN_16_flag_sgl, truncate_sgl,
    start_match_sgl, start_buffer_sgl,
    kill_count_sgl, Reserved_access_sgl,
    OpenSecureMemRef_sgl, error_code_sgl:
    STD_LOGIC;
signal what_cmd_sgl : STD_LOGIC_VECTOR(4 downto
    0);
signal Q_value_sgl : STD_LOGIC_VECTOR(3 downto
    0);
signal target_sel_sgl, action_sel_sgl:
    STD_LOGIC_VECTOR(2 downto 0);
signal Lock_Kill_sgl, Lock_Access_sgl,
    Lock_EPC_sgl, Lock_TID_sgl, Lock_User_sgl :
    STD_LOGIC_VECTOR(1 downto 0);
signal Lock_Kill_buff_sgl, Lock_Access_buff_sgl,
    Lock_EPC_buff_sgl, Lock_TID_buff_sgl,
    Lock_User_buff_sgl : STD_LOGIC_VECTOR(1
    downto 0);
signal MemBnk_sgl, MemBnk_buff_sgl, membnk_sel_sgl
    , crc_flag_sgl, EBV_write_sgl, EBV_read_sgl,
    EBV_sel_sgl: STD_LOGIC_VECTOR(1 downto 0);
signal pointer_sel_sgl, start_address_sgl,
    end_address_sgl, end_address_buff_sgl:
    STD_LOGIC_VECTOR(23 downto 0);
signal Mask_len_sgl: STD_LOGIC_VECTOR(7 downto
    0);
signal Mask_value_sgl: STD_LOGIC_VECTOR(255
    downto 0);
signal Q_sgl : STD_LOGIC_VECTOR(351 downto 0);
signal RN_16_sgl, slot_count_sgl, data_in_16_sgl,
    WrData_out_sgl, storedCRC_sgl, data_mem_sgl,
    write_mem_data_sgl, slot_15_sgl:
    std_logic_vector(15 downto 0);

component tea_top is
port(
clk      : in std_logic;
n_reset: in std_logic;
crypt_en         : in std_logic;
preset_engine: in std_logic;
text_in: in std_logic_vector(63 downto 0);
key_in: in std_logic_vector(127 downto 0);
cipher_out: out std_logic_vector(63 downto 0);
done: out std_logic
);
```

```
end component tea_top;

component CMD_Detect_23 is
port(
data, done,stop_in : IN STD_LOGIC;
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
bit_len_out: OUT STD_LOGIC_VECTOR(8 downto 0);
stop: OUT STD_LOGIC;
Out_Serial : OUT STD_LOGIC;
what_cmd: OUT STD_LOGIC_VECTOR(4 downto 0);
target_sel, action_sel: OUT STD_LOGIC_VECTOR(2
    downto 0);
membnk_sel,crc_flag, EBV_write, EBV_read,
    EBV_sel: OUT STD_LOGIC_VECTOR(1 downto 0);
pointer_sel: OUT STD_LOGIC_VECTOR(23 downto 0);
Mask_len: OUT STD_LOGIC_VECTOR(7 downto 0);
Mask_value: OUT STD_LOGIC_VECTOR(255 downto 0);
Q : OUT STD_LOGIC_VECTOR(351 downto 0)
);
end component CMD_Detect_23;

component FSM_1 is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset, powerup: in std_logic;
done,crc_valid, mask_done,Reserved_done,
    match_sel,stop,RN16_done,slot_done,
    RN64_done: in std_logic;
ebv, ebv_read, ebv_write,membnk_sel: in
    std_logic_vector(1 downto 0);
target_sel, action_sel: in std_logic_vector(2
    downto 0);
wht_cmd: in std_logic_vector(4 downto 0);
mask_length: in std_logic_vector(7 downto 0);
bit_len_in: in std_logic_vector(8 downto 0);
RN_16_in,slot_count,data_in_16: in
    std_logic_vector(15 downto 0);
pointer_sel: in std_logic_vector(23 downto 0);
data_in: in std_logic_vector(351 downto 0);
-- OUT PORTS --
RN_16,RN_64, truncate, start_match,start_buffer,
    kill_count_out, Reserved_access,no_reply:
    out std_logic;
EPC_done: out std_logic:='0';
EPC: out std_logic_vector(95 downto 0);
MemBnk_out: out std_logic_vector(1 downto 0);
OpenSecureMemRef, error_code: out std_logic;
WrData_out, storedCRC: out std_logic_vector(15
    downto 0);
Q_value_out: out std_logic_vector(3 downto 0);
start_address_out, end_address_out: out
    std_logic_vector(23 downto 0);
challenge_data : out std_logic_vector(63 downto
    0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: out std_logic_vector(1 downto 0)
);
end component FSM_1;

component Test_CRC is
port(
data, stop,done : IN STD_LOGIC;
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
crc_valid : OUT STD_LOGIC;
Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
end component Test_CRC;

component CRC16 is
port(
Preset : IN STD_LOGIC;
clk , powerup: IN STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
data, stop,done,key_crc : IN STD_LOGIC;
crc_valid,key_crc_done : OUT STD_LOGIC;
crc_plain_text: out std_logic_vector(15 downto
    0);
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
end component CRC16;

component CRC16_final is
port(
Preset, powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
data : IN STD_LOGIC;
start_crc, crc_flag,start_bit : IN STD_LOGIC;
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC;
data_bit_out, done : OUT STD_LOGIC
);
end component CRC16_final;

component slot_counter is
port(
--clk: in std_logic;--clock signal.
Preset: in std_logic;
Q_value_in: in std_logic_vector(3 downto 0);
RN_16_in: in std_logic_vector(15 downto 0);
slot_value_out: out std_logic_vector(15 downto
    0);
slot_done : out std_logic
);
end component slot_counter;

component select_match is
port(
clk: in std_logic;--clock signal.
Preset,mask_match,rd_done, powerup: in std_logic
    ;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address :in std_logic_vector(23 downto 0);
Mask_len: in STD_LOGIC_VECTOR(7 downto 0);
Mask_value_in: in STD_LOGIC_VECTOR(255 downto 0)
    ;
data: in STD_LOGIC_VECTOR(15 downto 0);
rd_en,match_valid,data_16_bit,select_done,
    use_epc : out std_logic;
start_address_out:out STD_LOGIC_VECTOR(23 downto
     0)
);
end component select_match;

component RN_16 is
port(
preset, clk, RN16_flag, RN64_flag: in std_logic;
lfsr_out,slot_15 : out std_logic_vector (15
    downto 0);
RN_64_data : out std_logic_vector(63 downto 0);
RN16_done,RN64_done : out std_logic
);
end component RN_16;

component buffer_module is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset,key_done, powerup: in std_logic;
start_buffer,truncate,error_code,rd_wr_lock_done
    ,kill_count,Reserved_access,rn_16,
    data_bit_in,mem_error,stop,use_epc_SM,
    no_reply,cipher_done: in std_logic;
wht_cmd: in std_logic_vector(4 downto 0);
start_address,end_address: in std_logic_vector
    (23 downto 0);
mem_bank_in: in std_logic_vector(1 downto 0);
data_16_in, rn16_data, stored_crc_in: in
    std_logic_vector(15 downto 0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: in std_logic_vector(1 downto 0);
EPC: in std_logic_vector(95 downto 0);
EPC_done: in std_logic;
key_crc_done,rn_64: in std_logic;
data_key: in std_logic_vector(127 downto 0);
rn64_data,challenge_data: in std_logic_vector(63
     downto 0);
crc_plain_text: in std_logic_vector(15 downto 0)
    ;
cipher_text : in std_logic_vector(63 downto 0);


-- OUT PORT
data_bit_out,rd_en,wr_en,reserved_done,
    data_16_bit,crc_flag,send_mem, start_crc,
    data_64_bit,key_crc,crypt_en,preset_engine,
    key_access,start_bit: out std_logic;
data_out: out std_logic_vector(15 downto 0);
mem_bank_out: out std_logic_vector(1 downto 0);
```

```
Lock_Kill_out, Lock_Access_out, Lock_EPC_out,
     Lock_TID_out, Lock_User_out: out
     std_logic_vector(1 downto 0);
data_plain_text : out std_logic_vector(63 downto
     0);
start_address_out,end_address_out: out
     std_logic_vector(23 downto 0)
);
end component buffer_module;

component EPC_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
     ;
end_address : in std_logic_vector(23 downto 0);
--pointer_sel:in STD_LOGIC_VECTOR(23 downto 0);
--mask_length: in std_logic_vector(7 downto 0);
lock_value,mem_bank : in std_logic_vector(1
     downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
     ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop : out
     std_logic
);
end component EPC_ROM;

component Reserved_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
     ;
end_address : in std_logic_vector(23 downto 0);
lock_kill_value, lock_access_value,mem_bank : in
     std_logic_vector(1 downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
     ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
     std_logic
);
end component Reserved_ROM;

component TID_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
     ;
end_address : in std_logic_vector(23 downto 0);
lock_value,mem_bank : in std_logic_vector(1
     downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
     ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
     std_logic
);
end component TID_ROM;

component mux_data16 is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic_vector(15 downto 0);
D_RES    : in std_logic_vector(15 downto 0);
D_TID    : in std_logic_vector(15 downto 0);
D_USER   : in std_logic_vector(15 downto 0);
mux_out: out std_logic_vector(15 downto 0)
);
end component mux_data16;

component mux_1bit is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit;

component mux_1bit_done is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_done;

component mux_1bit_error is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_error;

component mux_1bit_stop is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_stop;

component mux_1bit_crc is
port(
clk   : in std_logic;
crc_sel: in std_logic_vector(1 downto 0);
CRC_valid_5      : in std_logic;
CRC_valid_16: in std_logic;
mux_out : out std_logic
);
end component mux_1bit_crc;

component mux_start_addr is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
start_address_Buffer: in std_logic_vector(23
     downto 0);
start_address_SM: in std_logic_vector(23 downto
     0);
mux_out : out std_logic_vector(23 downto 0)
);
end component mux_start_addr;

component mux_rd_en is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
rd_en_buffer: in std_logic;
rd_en_SM: in std_logic;
mux_out : out std_logic
);
end component mux_rd_en;

component mux_length_16_bit is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
data_16_bit_buffer: in std_logic;
data_16_bit_SM: in std_logic;
mux_out : out std_logic
);
end component mux_length_16_bit;

component key_rom is
port (  clk: in std_logic;--clock signal.
Preset: in std_logic;
```

```
key_access : in std_logic;
key_done : out std_logic;
data_key : out std_logic_vector(127 downto 0)
);
end component key_rom;

component powerup_count is
port(
clk      : in std_logic;
--n_reset, crypt_en: in std_logic;
powerup, preset   : out std_logic
);

end component powerup_count;

begin

Xtea_top : tea_top
port map(
clk      => clk,
n_reset => preset,
crypt_en => crypt_en_sgl,
preset_engine => preset_engine_sgl,
text_in => data_plain_text_sgl,
key_in  => data_key_sgl,
cipher_out => cipher_text_sgl,
done => cipher_done_sgl
);

cmd_dec : CMD_Detect_23
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_in,
stop_in => stop_sgl,
bit_len_out => bit_len_sgl,
stop => stop_sgl,
done => done_sgl,
Out_Serial => Out_Serial_sgl,
what_cmd => what_cmd_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
membnk_sel => membnk_sel_sgl,
crc_flag => crc_flag_sgl,
EBV_write => EBV_write_sgl,
EBV_read => EBV_read_sgl,
EBV_sel => EBV_sel_sgl,
pointer_sel => pointer_sel_sgl,
Mask_len => Mask_len_sgl,
Mask_value => Mask_value_sgl,
Q => Q_sgl
);

FSM : FSM_1
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
done => done_sgl,
RN16_done => RN16_done_sgl,
RN64_done => RN64_done_sgl,
slot_done => slot_done_sgl,
crc_valid => crc_valid_sgl,
stop => stop_sgl,
mask_done => mask_done_sgl,
Reserved_done => Reserved_done_sgl,
match_sel => match_sel_sgl,
ebv => EBV_sel_sgl,
ebv_read => EBV_read_sgl,
ebv_write => EBV_write_sgl,
membnk_sel => membnk_sel_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
wht_cmd => what_cmd_sgl,
Mask_length => Mask_len_sgl,     -- to be given
    to Select match also
bit_len_in => bit_len_sgl,
RN_16_in => RN_16_sgl,           --data
slot_count => slot_count_sgl,
data_in_16 => data_in_16_sgl,
pointer_sel => pointer_sel_sgl,
data_in => Q_sgl,

-- OUT PORTS
RN_16 => RN_16_flag_sgl,                 -- Flag
    to intiate RN16
RN_64 => RN_64_flag_sgl,
truncate => truncate_sgl,
start_match => start_match_sgl,
start_buffer => start_buffer_sgl,
kill_count_out => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
MemBnk_out => MemBnk_sgl,
OpenSecureMemRef => OpenSecureMemRef_sgl,
error_code => error_code_sgl,
WrData_out => WrData_out_sgl,
storedCRC => storedCRC_sgl,
Q_value_out => Q_value_sgl,
start_address_out => start_address_sgl,
end_address_out => end_address_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,
Lock_User => Lock_User_sgl
);

CRC5 : Test_CRC
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
stop => stop_sgl,
done => done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_5_sgl
);

CRC_16 : CRC16
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
key_crc => key_crc_sgl,
crc_plain_text => crc_plain_text_sgl,
stop => stop_sgl,
done => done_sgl,
key_crc_done => key_crc_done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_16_sgl
);

CRC16_end : CRC16_final
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_sgl,
start_bit => start_bit_sgl,
start_crc => start_crc_sgl,
crc_flag => crc_flag_buff_sgl,
done => done_sgl,
data_bit_out => data_out
);

slot : slot_counter
port map(
--clk => clk,
preset => preset,
Q_value_in => Q_value_sgl,
RN_16_in => slot_15_sgl,                 --data (
    same as FSM)
slot_value_out => slot_count_sgl,
slot_done => slot_done_sgl
);

match_select : select_match
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
mask_match => start_match_sgl,
rd_done => rd_wr_lock_done_sig,
start_address => start_address_sgl,
end_address => end_address_sgl,
```

```
Mask_len => Mask_len_sgl,
Mask_value_in => Mask_value_sgl,
data => data_mem_sgl_EPC,
rd_en => rd_en_SM_sgl,
use_epc => use_epc_sgl,
match_valid => match_sel_sgl,
data_16_bit => len_16_bit_SM_sgl,
select_done => mask_done_sgl,
start_address_out => start_address_match_sgl
);

RN16 : RN_16
port map(
clk => clk,
preset => preset,
RN16_flag => RN_16_flag_sgl,
RN64_flag => RN_64_flag_sgl,
lfsr_out => RN_16_sgl,
RN_64_data => RN_64_sgl,
slot_15 => slot_15_sgl,
RN64_done => RN64_done_sgl,
RN16_done => RN16_done_sgl
);

response : buffer_module
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
start_buffer => start_buffer_sgl,
use_epc_SM => use_epc_sgl,
truncate => truncate_sgl,
error_code => error_code_sgl,
rd_wr_lock_done => rd_wr_lock_done_sig,
kill_count => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
rn_16 => RN16_done_sgl,
data_bit_in => data_mem_bit_sgl,
mem_error => mem_error_sgl,
stop => stop_mem_sgl,
wht_cmd => what_cmd_sgl,
start_address => start_address_sgl,
end_address => end_address_sgl,
mem_bank_in => MemBnk_sgl,
data_16_in => data_mem_sgl,
rn16_data => RN_16_sgl,
stored_crc_in => storedCRC_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
Lock_User => Lock_User_sgl,
key_crc_done => key_crc_done_sgl,
RN_64 => RN64_done_sgl,
data_key => data_key_sgl,
rn64_data => RN_64_sgl,
crc_plain_text => crc_plain_text_sgl,
cipher_done => cipher_done_sgl,
cipher_text => cipher_text_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,

-- OUT PORTS

data_bit_out => data_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
reserved_done => Reserved_done_sgl,
data_16_bit => len_16_bit_buff_sgl,
crc_flag => crc_flag_buff_sgl,
send_mem => send_mem_sgl,
start_crc => start_crc_sgl,
data_out => data_in_16_sgl,
mem_bank_out => MemBnk_buff_sgl,
Lock_Kill_out => Lock_Kill_buff_sgl,
Lock_Access_out => Lock_Access_buff_sgl,
Lock_EPC_out => Lock_EPC_buff_sgl,
Lock_TID_out => Lock_TID_buff_sgl,
Lock_User_out => Lock_User_buff_sgl,
start_address_out => start_address_buff_sgl,
key_done => key_done_sgl,
data_64_bit => data_64_bit_sgl,
key_crc => key_crc_sgl,
crypt_en => crypt_en_sgl,
start_bit => start_bit_sgl,
key_access => key_access_sgl,
preset_engine => preset_engine_sgl,
data_plain_text => data_plain_text_sgl,
end_address_out => end_address_buff_sgl
);

EPC : EPC_Rom
port map(
-- IN PORTS
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_select_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_EPC_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_SM_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_SM_sgl,

-- OUT PORTS
data_out_bit => data_mem_bit_sgl_EPC, -- Bit o/p
    from mem
data_out_16 => data_mem_sgl_EPC, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_EPC,
error => mem_error_sgl_EPC,
stop => stop_mem_sgl_EPC
);

Reserved : Reserved_ROM
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_kill_value => Lock_Kill_buff_sgl,
lock_access_value => Lock_Access_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_RES, -- Bit o/p
    from mem
data_out_16 => data_mem_sgl_RES, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_RES,
error => mem_error_sgl_RES,
stop => stop_mem_sgl_RES
);

TID : TID_Rom
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_TID_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_TID, -- Bit o/p
    from mem
data_out_16 => data_mem_sgl_TID, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_TID,
error => mem_error_sgl_TID,
stop => stop_mem_sgl_TID
);

Key : key_rom
port map(
clk => clk,
Preset => preset,
key_access => key_access_sgl,
key_done => key_done_sgl,
data_key => data_key_sgl
```

```
);

mux16_mod : mux_data16
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC    => data_mem_sgl_EPC,
D_RES    => data_mem_sgl_RES,
D_TID    => data_mem_sgl_TID,
D_USER   => data_mem_sgl_USER,
mux_out => data_mem_sgl
);


mux1bit_mod : mux_1bit
port map(
clk => clk,
membnk_sel=> MemBnk_buff_sgl,
D_EPC    => data_mem_bit_sgl_EPC,
D_RES    => data_mem_bit_sgl_RES,
D_TID    => data_mem_bit_sgl_TID,
D_USER   => data_mem_bit_sgl_USER,
mux_out => data_mem_bit_sgl
);

mux1bit_done_mod : mux_1bit_done
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC    => rd_wr_lock_done_sig_EPC,
D_RES    => rd_wr_lock_done_sig_RES,
D_TID    => rd_wr_lock_done_sig_TID,
D_USER   => rd_wr_lock_done_sig_USER,
mux_out => rd_wr_lock_done_sig
);

mux1bit_error_mod : mux_1bit_error
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC    => mem_error_sgl_EPC,
D_RES    => mem_error_sgl_RES,
D_TID    => mem_error_sgl_TID,
D_USER   => mem_error_sgl_USER,
mux_out => mem_error_sgl
);

mux1bit_stop_mod : mux_1bit_stop
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC    => stop_mem_sgl_EPC,
D_RES    => stop_mem_sgl_RES,
D_TID    => stop_mem_sgl_TID,
D_USER   => stop_mem_sgl_USER,
mux_out => stop_mem_sgl
);

mux1bit_crc_mod : mux_1bit_crc
port map(
clk      => clk,
crc_sel => crc_flag_sgl,
CRC_valid_5      => CRC_valid_5_sgl,
CRC_valid_16 => CRC_valid_16_sgl,
mux_out => crc_valid_sgl
);

mux_start_addr_mod : mux_start_addr
port map(
clk => clk,
select_match_sel => use_epc_sgl,
start_address_Buffer => start_address_buff_sgl,
start_address_SM => start_address_match_sgl,
mux_out => start_address_select_buff_sgl
);

mux_rd_en_mod : mux_rd_en
port map(
clk => clk,
select_match_sel => use_epc_sgl,
rd_en_buffer => rd_en_buff_sgl,
rd_en_SM => rd_en_SM_sgl,
mux_out => rd_en_buff_SM_sgl
);

mux_length_16_bit_mod : mux_length_16_bit
port map(
clk => clk,
select_match_sel => use_epc_sgl,
data_16_bit_buffer => len_16_bit_buff_sgl,
data_16_bit_SM => len_16_bit_SM_sgl,
mux_out => len_16_bit_buff_SM_sgl
);
powerup : powerup_count
port map(
clk      => clk,
preset => preset,
powerup => powerup_sgl
);

end architecture GEN2_Top_behavioral;
```

Listing A.3. GEN2 with AES - Top Level Entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity GEN2_Top is
port(
clk                      : in std_logic;
data_in          : in std_logic;
data_out         : out std_logic
);
end entity GEN2_Top;

architecture GEN2_Top_behavioral of GEN2_Top is

signal bit_len_sgl : STD_LOGIC_VECTOR(8 downto
    0);
signal stop_sgl, Out_Serial_sgl, crc_valid_sgl,
    CRC_valid_5_sgl, CRC_valid_16_sgl,
    mask_done_sgl, Reserved_done_sgl,
    match_sel_sgl, crc_flag_buff_sgl,
    stop_mem_sgl, send_mem_sgl: STD_LOGIC;
signal rd_wr_lock_done_sig, len_16_bit_mem_sgl,
    data_mem_bit_sgl, mem_error_sgl, rd_en_sgl,
    wr_en_sgl, data_sgl, start_crc_sgl,
    RN16_done_sgl, slot_done_sgl : STD_LOGIC;

signal mem_error_sgl_EPC, mem_error_sgl_RES,
    mem_error_sgl_TID, mem_error_sgl_USER,
    data_crc_sgl, start_bit_sgl, preset,
    powerup_sgl : STD_LOGIC;
signal stop_mem_sgl_EPC, stop_mem_sgl_RES,
    stop_mem_sgl_TID, stop_mem_sgl_USER :
    STD_LOGIC;
signal rd_wr_lock_done_sig_EPC,
    rd_wr_lock_done_sig_RES,
    rd_wr_lock_done_sig_TID,
    rd_wr_lock_done_sig_USER : STD_LOGIC;
signal data_mem_bit_sgl_EPC, data_mem_bit_sgl_RES
    , data_mem_bit_sgl_TID,
    data_mem_bit_sgl_USER : STD_LOGIC;
signal data_mem_sgl_EPC, data_mem_sgl_RES,
    data_mem_sgl_TID, data_mem_sgl_USER :
    STD_LOGIC_VECTOR(15 downto 0);

signal len_16_bit_buff_sgl, len_16_bit_SM_sgl,
    len_16_bit_buff_SM_sgl, key_crc_done_sgl,
    EPC_done_sgl : STD_LOGIC;
signal rd_en_buff_sgl, rd_en_SM_sgl,
    rd_en_buff_SM_sgl : STD_LOGIC;
signal start_address_buff_sgl,
    start_address_match_sgl,
    start_address_select_buff_sgl :
    STD_LOGIC_VECTOR(23 downto 0);
signal EPC_sgl : STD_LOGIC_VECTOR(95 downto 0);

signal key_access_sgl, key_done_sgl, key_crc_sgl,
    RN_64_flag_sgl, RN64_done_sgl, crypt_en_sgl,
    preset_engine_sgl, data_64_bit_sgl,
    cipher_done_sgl : std_logic;
signal data_key_sgl: std_logic_vector(127 downto
     0);
signal crc_plain_text_sgl: std_logic_vector(15
    downto 0);
signal RN_64_sgl: std_logic_vector(63 downto 0);
```

```
signal challenge_data_sgl: std_logic_vector(127
    downto 0);
signal cipher_text_sgl,data_plain_text_sgl:
    std_logic_vector(127 downto 0);

signal use_epc_sgl,done_sgl,no_reply_sgl :
    STD_LOGIC;
signal RN_16_flag_sgl, truncate_sgl,
    start_match_sgl,start_buffer_sgl,
    kill_count_sgl, Reserved_access_sgl,
    OpenSecureMemRef_sgl, error_code_sgl:
    STD_LOGIC;
signal what_cmd_sgl : STD_LOGIC_VECTOR(4 downto
    0);
signal Q_value_sgl : STD_LOGIC_VECTOR(3 downto
    0);
signal target_sel_sgl, action_sel_sgl:
    STD_LOGIC_VECTOR(2 downto 0);
signal Lock_Kill_sgl, Lock_Access_sgl,
    Lock_EPC_sgl, Lock_TID_sgl, Lock_User_sgl :
     STD_LOGIC_VECTOR(1 downto 0);
signal Lock_Kill_buff_sgl, Lock_Access_buff_sgl,
     Lock_EPC_buff_sgl, Lock_TID_buff_sgl,
    Lock_User_buff_sgl : STD_LOGIC_VECTOR(1
    downto 0);
signal MemBnk_sgl,MemBnk_buff_sgl,membnk_sel_sgl
    ,crc_flag_sgl, EBV_write_sgl, EBV_read_sgl,
     EBV_sel_sgl: STD_LOGIC_VECTOR(1 downto 0);
signal pointer_sel_sgl,start_address_sgl,
    end_address_sgl,end_address_buff_sgl:
    STD_LOGIC_VECTOR(23 downto 0);
signal Mask_len_sgl: STD_LOGIC_VECTOR(7 downto
    0);
signal Mask_value_sgl: STD_LOGIC_VECTOR(255
    downto 0);
signal Q_sgl : STD_LOGIC_VECTOR(351 downto 0);
signal RN_16_sgl,slot_count_sgl,data_in_16_sgl,
    WrData_out_sgl, storedCRC_sgl,data_mem_sgl,
    write_mem_data_sgl,slot_15_sgl:
    std_logic_vector(15 downto 0);

component aes_top is
port(
clk     : in std_logic;
reset: in std_logic;
crypt_en        : in std_logic;
preset_engine: in std_logic;
text_in: in std_logic_vector(0 to 127);
key_in: in std_logic_vector(0 to 127);
cipher_out1: out std_logic_vector(0 to 127);
done: out std_logic
--flag2: out std_logic
);

end component aes_top;

component CMD_Detect_23 is
port(
data, done,stop_in :  IN  STD_LOGIC;
Preset, powerup :  IN  STD_LOGIC;
clk :  IN  STD_LOGIC;
bit_len_out: OUT STD_LOGIC_VECTOR(8 downto 0);
stop: OUT STD_LOGIC;
--crc_U_DU: OUT STD_LOGIC;
Out_Serial :  OUT  STD_LOGIC;
what_cmd: OUT STD_LOGIC_VECTOR(4 downto 0);
target_sel, action_sel: OUT STD_LOGIC_VECTOR(2
    downto 0);
membnk_sel,crc_flag, EBV_write, EBV_read,
    EBV_sel: OUT STD_LOGIC_VECTOR(1 downto 0);
pointer_sel: OUT STD_LOGIC_VECTOR(23 downto 0);
Mask_len: OUT STD_LOGIC_VECTOR(7 downto 0);
Mask_value: OUT STD_LOGIC_VECTOR(255 downto 0);
Q :  OUT  STD_LOGIC_VECTOR(351 downto 0)
);
end component CMD_Detect_23;

component FSM_1 is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset, powerup: in std_logic;
done,crc_valid, mask_done,Reserved_done,
    match_sel,stop,RN16_done,slot_done,
    RN64_done: in std_logic;
ebv, ebv_read, ebv_write,membnk_sel: in
    std_logic_vector(1 downto 0);
target_sel, action_sel: in std_logic_vector(2
    downto 0);
wht_cmd: in std_logic_vector(4 downto 0);
mask_length: in std_logic_vector(7 downto 0);
bit_len_in: in std_logic_vector(8 downto 0);
RN_16_in,slot_count,data_in_16: in
    std_logic_vector(15 downto 0);
pointer_sel: in std_logic_vector(23 downto 0);
data_in: in std_logic_vector(351 downto 0);
-- OUT PORTS --
RN_16,RN_64, truncate, start_match,start_buffer,
    kill_count_out, Reserved_access,no_reply:
    out std_logic;
EPC_done: out std_logic:='0';
EPC: out std_logic_vector(95 downto 0);
MemBnk_out: out std_logic_vector(1 downto 0);
OpenSecureMemRef, error_code: out std_logic;
WrData_out, storedCRC: out std_logic_vector(15
    downto 0);
Q_value_out: out std_logic_vector(3 downto 0);
start_address_out, end_address_out: out
    std_logic_vector(23 downto 0);
challenge_data : out std_logic_vector(127 downto
     0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: out std_logic_vector(1 downto 0)
);
end component FSM_1;

component Test_CRC is
port(
data, stop,done :  IN  STD_LOGIC;
Preset, powerup :  IN  STD_LOGIC;
clk :  IN  STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
crc_valid : OUT STD_LOGIC;
Q4, Q3, Q2, Q1, Q0 :  OUT  STD_LOGIC
);
end component Test_CRC;

component CRC16 is
port(
Preset :  IN  STD_LOGIC;
clk, powerup:  IN  STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
data, stop,done,key_crc :  IN  STD_LOGIC;
crc_valid,key_crc_done :  OUT  STD_LOGIC;
crc_plain_text: out std_logic_vector(15 downto
    0);
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 :  OUT  STD_LOGIC
);
end component CRC16;

component CRC16_final is
port(
Preset, powerup :  IN  STD_LOGIC;
clk :  IN  STD_LOGIC;
data :  IN  STD_LOGIC;
start_crc, crc_flag,start_bit :  IN  STD_LOGIC;
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 :  OUT  STD_LOGIC;
data_bit_out, done : OUT STD_LOGIC
);
end component CRC16_final;

component slot_counter is
port(
--clk: in std_logic;--clock signal.
Preset: in std_logic;
Q_value_in: in std_logic_vector(3 downto 0);
RN_16_in: in std_logic_vector(15 downto 0);
slot_value_out: out std_logic_vector(15 downto
    0);
slot_done : out std_logic
);
end component slot_counter;

component select_match is
port(
clk: in std_logic;--clock signal.
Preset,mask_match,rd_done, powerup: in std_logic
    ;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address :in std_logic_vector(23 downto 0);
Mask_len: in STD_LOGIC_VECTOR(7 downto 0);
```

```
Mask_value_in: in STD_LOGIC_VECTOR(255 downto 0)
    ;
data: in STD_LOGIC_VECTOR(15 downto 0);
rd_en, match_valid, data_16_bit, select_done,
    use_epc : out std_logic;
start_address_out:out STD_LOGIC_VECTOR(23 downto
    0)
);
end component select_match;

component RN_16 is
port(
preset, clk, RN16_flag, RN64_flag: in std_logic;
lfsr_out, slot_15 : out std_logic_vector (15
    downto 0);
RN_64_data : out std_logic_vector(63 downto 0);
RN16_done, RN64_done : out std_logic
);
end component RN_16;

component buffer_module is
port(
-- IN PORT
clk: in std_logic;--clock signal.
Preset, key_done, powerup: in std_logic;
start_buffer, truncate, error_code, rd_wr_lock_done
    , kill_count, Reserved_access, rn_16,
    data_bit_in, mem_error, stop, use_epc_SM,
    no_reply, cipher_done: in std_logic;
wht_cmd: in std_logic_vector(4 downto 0);
start_address, end_address: in std_logic_vector
    (23 downto 0);
mem_bank_in: in std_logic_vector(1 downto 0);
data_16_in, rn16_data, stored_crc_in: in
    std_logic_vector(15 downto 0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: in std_logic_vector(1 downto 0);
EPC: in std_logic_vector(95 downto 0);
EPC_done: in std_logic;
key_crc_done, rn_64: in std_logic;
data_key: in std_logic_vector(127 downto 0);
rn64_data: in std_logic_vector(63 downto 0);
challenge_data: in std_logic_vector(127 downto
    0);
crc_plain_text: in std_logic_vector(15 downto 0)
    ;
cipher_text : in std_logic_vector(127 downto 0);


-- OUT PORT
data_bit_out, rd_en, wr_en, reserved_done,
    data_16_bit, crc_flag, send_mem, start_crc,
    data_64_bit, key_crc, crypt_en, preset_engine,
    key_access, start_bit: out std_logic;
data_out: out std_logic_vector(15 downto 0);
mem_bank_out: out std_logic_vector(1 downto 0);
Lock_Kill_out, Lock_Access_out, Lock_EPC_out,
    Lock_TID_out, Lock_User_out: out
    std_logic_vector(1 downto 0);
data_plain_text : out std_logic_vector(127
    downto 0);
start_address_out, end_address_out: out
    std_logic_vector(23 downto 0)
);
end component buffer_module;

component EPC_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_value, mem_bank : in std_logic_vector(1
    downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
    std_logic;
data_out_bit, rd_wr_lock_done, error, stop : out
    std_logic
);
end component EPC_ROM;

component Reserved_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_kill_value, lock_access_value, mem_bank : in
    std_logic_vector(1 downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
    std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
    std_logic
);
end component Reserved_ROM;

component TID_ROM is
port(
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address : in std_logic_vector(23 downto 0);
lock_value, mem_bank : in std_logic_vector(1
    downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
    std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
    std_logic
);
end component TID_ROM;

component mux_data16 is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC   : in std_logic_vector(15 downto 0);
D_RES   : in std_logic_vector(15 downto 0);
D_TID   : in std_logic_vector(15 downto 0);
D_USER  : in std_logic_vector(15 downto 0);
mux_out: out std_logic_vector(15 downto 0)
);
end component mux_data16;

component mux_1bit is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC   : in std_logic;
D_RES   : in std_logic;
D_TID   : in std_logic;
D_USER  : in std_logic;
mux_out : out std_logic
);
end component mux_1bit;

component mux_1bit_done is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC   : in std_logic;
D_RES   : in std_logic;
D_TID   : in std_logic;
D_USER  : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_done;

component mux_1bit_error is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC   : in std_logic;
D_RES   : in std_logic;
D_TID   : in std_logic;
D_USER  : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_error;

component mux_1bit_stop is
port(
clk   : in std_logic;
```

```
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end component mux_1bit_stop;

component mux_1bit_crc is
port(
clk   : in std_logic;
crc_sel: in std_logic_vector(1 downto 0);
CRC_valid_5      : in std_logic;
CRC_valid_16: in std_logic;
mux_out : out std_logic
);
end component mux_1bit_crc;

component mux_start_addr is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
start_address_Buffer: in std_logic_vector(23
     downto 0);
start_address_SM: in std_logic_vector(23 downto
     0);
mux_out : out std_logic_vector(23 downto 0)
);
end component mux_start_addr;

component mux_rd_en is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
rd_en_buffer: in std_logic;
rd_en_SM: in std_logic;
mux_out : out std_logic
);
end component mux_rd_en;

component mux_length_16_bit is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
data_16_bit_buffer: in std_logic;
data_16_bit_SM: in std_logic;
mux_out : out std_logic
);
end component mux_length_16_bit;

component key_rom is
port (  clk: in std_logic;--clock signal.
Preset: in std_logic;
key_access : in std_logic;
key_done : out std_logic;
data_key : out std_logic_vector(127 downto 0)
);
end component key_rom;

component powerup_count is
port(
clk      : in std_logic;
--n_reset,crypt_en: in std_logic;
powerup,preset   : out std_logic
);

end component powerup_count;

begin
top_aes: aes_top
port map(
clk      => clk,
reset => preset,
crypt_en => crypt_en_sgl,
preset_engine => preset_engine_sgl,
text_in => data_plain_text_sgl,
key_in  => data_key_sgl,
cipher_out1 => cipher_text_sgl,
done => cipher_done_sgl
--flag2: out std_logic
);

cmd_dec : CMD_Detect_23
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_in,
stop_in => stop_sgl,
bit_len_out => bit_len_sgl,
stop => stop_sgl,
done => done_sgl,
Out_Serial => Out_Serial_sgl,
what_cmd => what_cmd_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
membnk_sel => membnk_sel_sgl,
crc_flag => crc_flag_sgl,
EBV_write => EBV_write_sgl,
EBV_read => EBV_read_sgl,
EBV_sel => EBV_sel_sgl,
pointer_sel => pointer_sel_sgl,
Mask_len => Mask_len_sgl,
Mask_value => Mask_value_sgl,
Q => Q_sgl
);

FSM : FSM_1
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
done => done_sgl,
RN16_done => RN16_done_sgl,
RN64_done => RN64_done_sgl,
slot_done => slot_done_sgl,
crc_valid => crc_valid_sgl,
stop => stop_sgl,
mask_done => mask_done_sgl,
Reserved_done => Reserved_done_sgl,
match_sel => match_sel_sgl,
ebv => EBV_sel_sgl,
ebv_read => EBV_read_sgl,
ebv_write => EBV_write_sgl,
membnk_sel => membnk_sel_sgl,
target_sel => target_sel_sgl,
action_sel => action_sel_sgl,
wht_cmd => what_cmd_sgl,
Mask_length => Mask_len_sgl,     -- to be given
     to Select match also
bit_len_in => bit_len_sgl,
RN_16_in => RN_16_sgl,           --data
slot_count => slot_count_sgl,
data_in_16 => data_in_16_sgl,
pointer_sel => pointer_sel_sgl,
data_in => Q_sgl,

-- OUT PORTS
RN_16 => RN_16_flag_sgl,                    -- Flag
     to intiate RN16
RN_64 => RN_64_flag_sgl,
truncate => truncate_sgl,
start_match => start_match_sgl,
start_buffer => start_buffer_sgl,
kill_count_out => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
MemBnk_out => MemBnk_sgl,
OpenSecureMemRef => OpenSecureMemRef_sgl,
error_code => error_code_sgl,
WrData_out => WrData_out_sgl,
storedCRC => storedCRC_sgl,
Q_value_out => Q_value_sgl,
start_address_out => start_address_sgl,
end_address_out => end_address_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,
Lock_User => Lock_User_sgl
);

CRC5 : Test_CRC
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
stop => stop_sgl,
```

```
done => done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_5_sgl
);

CRC_16 : CRC16
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => Out_Serial_sgl,
key_crc => key_crc_sgl,
crc_plain_text => crc_plain_text_sgl,
stop => stop_sgl,
done => done_sgl,
key_crc_done => key_crc_done_sgl,
crc_flag => crc_flag_sgl,
crc_valid => CRC_valid_16_sgl
);

CRC16_end : CRC16_final
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
data => data_sgl,
start_bit => start_bit_sgl,
start_crc => start_crc_sgl,
crc_flag => crc_flag_buff_sgl,
done => done_sgl,
data_bit_out => data_out
);

slot : slot_counter
port map(
--clk => clk,
preset => preset,
Q_value_in => Q_value_sgl,
RN_16_in => slot_15_sgl,                    --data (
    same as FSM)
slot_value_out => slot_count_sgl,
slot_done => slot_done_sgl
);

match_select : select_match
port map(
clk => clk,
preset => preset,
powerup => powerup_sgl,
mask_match => start_match_sgl,
rd_done => rd_wr_lock_done_sig,
start_address => start_address_sgl,
end_address => end_address_sgl,
Mask_len => Mask_len_sgl,
Mask_value_in => Mask_value_sgl,
data => data_mem_sgl_EPC,
rd_en => rd_en_SM_sgl,
use_epc => use_epc_sgl,
match_valid => match_sel_sgl,
data_16_bit => len_16_bit_SM_sgl,
select_done => mask_done_sgl,
start_address_out => start_address_match_sgl
);

RN16 : RN_16
port map(
clk => clk,
preset => preset,
RN16_flag => RN_16_flag_sgl,
RN64_flag => RN_64_flag_sgl,
lfsr_out => RN_16_sgl,
RN_64_data => RN_64_sgl,
slot_15 => slot_15_sgl,
RN64_done => RN64_done_sgl,
RN16_done => RN16_done_sgl
);

response : buffer_module
port map(
-- IN PORTS
clk => clk,
preset => preset,
powerup => powerup_sgl,
start_buffer => start_buffer_sgl,
use_epc_SM => use_epc_sgl,
truncate => truncate_sgl,
error_code => error_code_sgl,
rd_wr_lock_done => rd_wr_lock_done_sig,
kill_count => kill_count_sgl,
Reserved_access => Reserved_access_sgl,
rn_16 => RN16_done_sgl,
data_bit_in => data_mem_bit_sgl,
mem_error => mem_error_sgl,
stop => stop_mem_sgl,
wht_cmd => what_cmd_sgl,
start_address => start_address_sgl,
end_address => end_address_sgl,
mem_bank_in => MemBnk_sgl,
data_16_in => data_mem_sgl,
rn16_data => RN_16_sgl,
stored_crc_in => storedCRC_sgl,
challenge_data => challenge_data_sgl,
Lock_kill => Lock_Kill_sgl,
Lock_Access => Lock_Access_sgl,
Lock_EPC => Lock_EPC_sgl,
Lock_TID => Lock_TID_sgl,
no_reply => no_reply_sgl,
Lock_User => Lock_User_sgl,
key_crc_done => key_crc_done_sgl,
RN_64 => RN64_done_sgl,
data_key => data_key_sgl,
rn64_data => RN_64_sgl,
crc_plain_text => crc_plain_text_sgl,
cipher_done => cipher_done_sgl,
cipher_text => cipher_text_sgl,
EPC_done => EPC_done_sgl,
EPC => EPC_sgl,

-- OUT PORTS

data_bit_out => data_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
reserved_done => Reserved_done_sgl,
data_16_bit => len_16_bit_buff_sgl,
crc_flag => crc_flag_buff_sgl,
send_mem => send_mem_sgl,
start_crc => start_crc_sgl,
data_out => data_in_16_sgl,
mem_bank_out => MemBnk_buff_sgl,
Lock_Kill_out => Lock_Kill_buff_sgl,
Lock_Access_out => Lock_Access_buff_sgl,
Lock_EPC_out => Lock_EPC_buff_sgl,
Lock_TID_out => Lock_TID_buff_sgl,
Lock_User_out => Lock_User_buff_sgl,
start_address_out => start_address_buff_sgl,
key_done => key_done_sgl,
data_64_bit => data_64_bit_sgl,
key_crc => key_crc_sgl,
crypt_en => crypt_en_sgl,
start_bit => start_bit_sgl,
key_access => key_access_sgl,
preset_engine => preset_engine_sgl,
data_plain_text  => data_plain_text_sgl,
end_address_out => end_address_buff_sgl
);

EPC : EPC_Rom
port map(
-- IN PORTS
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_select_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_EPC_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_SM_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_SM_sgl,

-- OUT PORTS
data_out_bit => data_mem_bit_sgl_EPC, -- Bit o/p
     from mem
data_out_16 => data_mem_sgl_EPC, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_EPC,
error => mem_error_sgl_EPC,
stop => stop_mem_sgl_EPC
);

Reserved : Reserved_ROM
```

```
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_kill_value => Lock_Kill_buff_sgl,
lock_access_value => Lock_Access_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_RES, -- Bit o/p
    from mem
data_out_16 => data_mem_sgl_RES, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_RES,
error => mem_error_sgl_RES,
stop => stop_mem_sgl_RES
);

TID : TID_Rom
port map(
clk => clk,
preset => preset,
send => send_mem_sgl,
start_address => start_address_buff_sgl,
end_address => end_address_buff_sgl,
lock_value => Lock_TID_buff_sgl,
mem_bank => MemBnk_buff_sgl,
data_in_write => WrData_out_sgl,
rd_en => rd_en_buff_sgl,
wr_en => wr_en_sgl,
open_state_valid => OpenSecureMemRef_sgl,
data_16_bit => len_16_bit_buff_sgl,
data_out_bit => data_mem_bit_sgl_TID, -- Bit o/p
    from mem
data_out_16 => data_mem_sgl_TID, -- Word o/p
    from mem
rd_wr_lock_done => rd_wr_lock_done_sig_TID,
error => mem_error_sgl_TID,
stop => stop_mem_sgl_TID
);

Key : key_rom
port map(
clk => clk,
Preset => preset,
key_access => key_access_sgl,
key_done => key_done_sgl,
data_key => data_key_sgl
);

mux16_mod : mux_data16
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => data_mem_sgl_EPC,
D_RES   => data_mem_sgl_RES,
D_TID   => data_mem_sgl_TID,
D_USER  => data_mem_sgl_USER,
mux_out => data_mem_sgl
);


mux1bit_mod : mux_1bit
port map(
clk => clk,
membnk_sel=> MemBnk_buff_sgl,
D_EPC   => data_mem_bit_sgl_EPC,
D_RES   => data_mem_bit_sgl_RES,
D_TID   => data_mem_bit_sgl_TID,
D_USER  => data_mem_bit_sgl_USER,
mux_out => data_mem_bit_sgl
);

mux1bit_done_mod : mux_1bit_done
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => rd_wr_lock_done_sig_EPC,
D_RES   => rd_wr_lock_done_sig_RES,
D_TID   => rd_wr_lock_done_sig_TID,
D_USER  => rd_wr_lock_done_sig_USER,
mux_out => rd_wr_lock_done_sig
);

mux1bit_error_mod : mux_1bit_error
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => mem_error_sgl_EPC,
D_RES   => mem_error_sgl_RES,
D_TID   => mem_error_sgl_TID,
D_USER  => mem_error_sgl_USER,
mux_out => mem_error_sgl
);

mux1bit_stop_mod : mux_1bit_stop
port map(
clk => clk,
membnk_sel => MemBnk_buff_sgl,
D_EPC   => stop_mem_sgl_EPC,
D_RES   => stop_mem_sgl_RES,
D_TID   => stop_mem_sgl_TID,
D_USER  => stop_mem_sgl_USER,
mux_out => stop_mem_sgl
);

mux1bit_crc_mod : mux_1bit_crc
port map(
clk     => clk,
crc_sel => crc_flag_sgl,
CRC_valid_5     => CRC_valid_5_sgl,
CRC_valid_16 => CRC_valid_16_sgl,
mux_out => crc_valid_sgl
);

mux_start_addr_mod : mux_start_addr
port map(
clk => clk,
select_match_sel => use_epc_sgl,
start_address_Buffer => start_address_buff_sgl,
start_address_SM => start_address_match_sgl,
mux_out => start_address_select_buff_sgl
);

mux_rd_en_mod : mux_rd_en
port map(
clk => clk,
select_match_sel => use_epc_sgl,
rd_en_buffer => rd_en_buff_sgl,
rd_en_SM => rd_en_SM_sgl,
mux_out => rd_en_buff_SM_sgl
);

mux_length_16_bit_mod : mux_length_16_bit
port map(
clk => clk,
select_match_sel => use_epc_sgl,
data_16_bit_buffer => len_16_bit_buff_sgl,
data_16_bit_SM => len_16_bit_SM_sgl,
mux_out => len_16_bit_buff_SM_sgl
);
powerup : powerup_count
port map(
clk     => clk,
preset => preset,
powerup => powerup_sgl
);
end architecture GEN2_Top_behavioral;
```

Listing A.4. Power-up counter module

```
--Including IEEE libraries.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

-- In an entity named "gen2", all required
     commands are initialised to the required
     bit lengths as specified in the protocol.
entity powerup_count is
port(
clk     : in std_logic;
powerup,preset  : out std_logic
);
end entity powerup_count;
```

```
architecture dfl of powerup_count  is
signal count_reg : unsigned(11 downto 0):="
    000000000000";
begin
process(clk)
begin
if(clk'event and clk = '1') then
if(count_reg = "000000000000") then
count_reg <= count_reg + 1;
powerup <= '0';
preset <= '0';
elsif(count_reg = "000000000001") then
count_reg <= count_reg + 1;
powerup <= '0';
preset <= '0';
elsif( (count_reg > "000000000001") and (
    count_reg < "011111010001") ) then
count_reg <= count_reg + 1;
powerup <= '0';
preset <= '1';
else
powerup <= '1';
end if;
end if;
end process;
end architecture dfl;
```

Listing A.5. Command Detection Module

```
-- PROGRAM                  "COMMAND DETECTION
    MODULE HDL"
-- VERSION                  "Version 1.8"
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

ENTITY CMD_Detect_23 IS
PORT
(
--input: IN     STD_LOGIC_VECTOR(24 downto 0);
data :  IN  STD_LOGIC;
Preset,powerup :  IN  STD_LOGIC;
clk :  IN  STD_LOGIC;
done, stop_in: IN   STD_LOGIC;
bit_len_out: OUT STD_LOGIC_VECTOR(8 downto 0);
stop: OUT STD_LOGIC;
Out_Serial :  OUT  STD_LOGIC;
what_cmd: OUT STD_LOGIC_VECTOR(4 downto 0);
target_sel, action_sel: OUT STD_LOGIC_VECTOR(2
    downto 0);
membnk_sel,crc_flag, EBV_write, EBV_read,
    EBV_sel: OUT STD_LOGIC_VECTOR(1 downto 0);
pointer_sel: OUT STD_LOGIC_VECTOR(23 downto 0);
Mask_len: OUT STD_LOGIC_VECTOR(7 downto 0);
Mask_value: OUT STD_LOGIC_VECTOR(255 downto 0);
Q :  OUT  STD_LOGIC_VECTOR(351 downto 0)
);
END CMD_Detect_23;

ARCHITECTURE RTL_CMD OF CMD_Detect_23 IS

SIGNAL  DFF_inst :  STD_LOGIC_VECTOR(351 downto
    0);
SIGNAL  mask_val :  STD_LOGIC_VECTOR(255 downto
    0);
signal count_reg, bit_len, bit_len1: unsigned(8
    downto 0);
signal mask_length : unsigned(7 downto 0);
signal EBV : unsigned(1 downto 0);
signal valid, fixed, stop_sgl,stop_data,
    buffer_ff: std_logic;
signal wht_cmd: STD_LOGIC_VECTOR(4 downto 0);
BEGIN
Q(0) <= DFF_inst(0);
Q(1) <= DFF_inst(1);                Q(2) <= DFF_inst
    (2);            Q(3) <= DFF_inst(3);
Q(4) <= DFF_inst(4);                Q(5) <= DFF_inst
    (5);            Q(6) <= DFF_inst(6);
Q(7) <= DFF_inst(7);                Q(8) <= DFF_inst
    (8);            Q(9) <= DFF_inst(9);
Q(10) <= DFF_inst(10);              Q(11) <=
    DFF_inst(11);           Q(12) <= DFF_inst
    (12);
Q(13) <= DFF_inst(13);              Q(14) <=
    DFF_inst(14);           Q(15) <= DFF_inst
    (15);
Q(16) <= DFF_inst(16);              Q(17) <=
    DFF_inst(17);           Q(18) <= DFF_inst
    (18);
Q(19) <= DFF_inst(19);              Q(20) <=
    DFF_inst(20);           Q(21) <= DFF_inst
    (21);
Q(22) <= DFF_inst(22);              Q(23) <=
    DFF_inst(23);           Q(24) <= DFF_inst
    (24);
Q(25) <= DFF_inst(25);              Q(26) <=
    DFF_inst(26);           Q(27) <= DFF_inst
    (27);
Q(28) <= DFF_inst(28);              Q(29) <=
    DFF_inst(29);           Q(30) <= DFF_inst
    (30);
Q(31) <= DFF_inst(31);              Q(32) <=
    DFF_inst(32);           Q(33) <= DFF_inst
    (33);
Q(34) <= DFF_inst(34);              Q(35) <=
    DFF_inst(35);           Q(36) <= DFF_inst
    (36);
Q(37) <= DFF_inst(37);              Q(38) <=
    DFF_inst(38);           Q(39) <= DFF_inst
    (39);
Q(40) <= DFF_inst(40);              Q(41) <=
    DFF_inst(41);           Q(42) <= DFF_inst
    (42);
Q(43) <= DFF_inst(43);              Q(44) <=
    DFF_inst(44);           Q(45) <= DFF_inst
    (45);
Q(46) <= DFF_inst(46);              Q(47) <=
    DFF_inst(47);           Q(48) <= DFF_inst
    (48);
Q(49) <= DFF_inst(49);              Q(50) <=
    DFF_inst(50);           Q(51) <= DFF_inst
    (51);
Q(52) <= DFF_inst(52);              Q(53) <=
    DFF_inst(53);           Q(54) <= DFF_inst
    (54);
Q(55) <= DFF_inst(55);              Q(56) <=
    DFF_inst(56);           Q(57) <= DFF_inst
    (57);
Q(58) <= DFF_inst(58);              Q(59) <=
    DFF_inst(59);           Q(60) <= DFF_inst
    (60);
Q(61) <= DFF_inst(61);              Q(62) <=
    DFF_inst(62);           Q(63) <= DFF_inst
    (63);
Q(64) <= DFF_inst(64);              Q(65) <=
    DFF_inst(65);           Q(66) <= DFF_inst
    (66);
Q(67) <= DFF_inst(67);              Q(68) <=
    DFF_inst(68);           Q(69) <= DFF_inst
    (69);
Q(70) <= DFF_inst(70);              Q(71) <=
    DFF_inst(71);           Q(72) <= DFF_inst
    (72);
Q(73) <= DFF_inst(73);              Q(74) <=
    DFF_inst(74);           Q(75) <= DFF_inst
    (75);
Q(76) <= DFF_inst(76);              Q(77) <=
    DFF_inst(77);           Q(78) <= DFF_inst
    (78);
Q(79) <= DFF_inst(79);              Q(80) <=
    DFF_inst(80);           Q(81) <= DFF_inst
    (81);
Q(82) <= DFF_inst(82);              Q(83) <=
    DFF_inst(83);           Q(84) <= DFF_inst
    (84);
Q(85) <= DFF_inst(85);              Q(86) <=
    DFF_inst(86);           Q(87) <= DFF_inst
    (87);
Q(88) <= DFF_inst(88);              Q(89) <=
    DFF_inst(89);           Q(90) <= DFF_inst
    (90);
Q(91) <= DFF_inst(91);              Q(92) <=
    DFF_inst(92);           Q(93) <= DFF_inst
    (93);
Q(94) <= DFF_inst(94);              Q(95) <=
    DFF_inst(95);           Q(96) <= DFF_inst
    (96);
Q(97) <= DFF_inst(97);              Q(98) <=
    DFF_inst(98);           Q(99) <= DFF_inst
    (99);
```

```
Q(100) <= DFF_inst(100);        Q(101) <=
    DFF_inst(101);          Q(102) <= DFF_inst
    (102);
Q(103) <= DFF_inst(103);        Q(104) <=
    DFF_inst(104);          Q(105) <= DFF_inst
    (105);
Q(106) <= DFF_inst(106);        Q(107) <=
    DFF_inst(107);          Q(108) <= DFF_inst
    (108);
Q(109) <= DFF_inst(109);        Q(110) <=
    DFF_inst(110);          Q(111) <= DFF_inst
    (111);
Q(112) <= DFF_inst(112);        Q(113) <=
    DFF_inst(113);          Q(114) <= DFF_inst
    (114);
Q(115) <= DFF_inst(115);        Q(116) <=
    DFF_inst(116);          Q(117) <= DFF_inst
    (117);
Q(118) <= DFF_inst(118);        Q(119) <=
    DFF_inst(119);          Q(120) <= DFF_inst
    (120);
Q(121) <= DFF_inst(121);        Q(122) <=
    DFF_inst(122);          Q(123) <= DFF_inst
    (123);
Q(124) <= DFF_inst(124);        Q(125) <=
    DFF_inst(125);          Q(126) <= DFF_inst
    (126);
Q(127) <= DFF_inst(127);        Q(128) <=
    DFF_inst(128);          Q(129) <= DFF_inst
    (129);
Q(130) <= DFF_inst(130);        Q(131) <=
    DFF_inst(131);          Q(132) <= DFF_inst
    (132);
Q(133) <= DFF_inst(133);        Q(134) <=
    DFF_inst(134);          Q(135) <= DFF_inst
    (135);
Q(136) <= DFF_inst(136);        Q(137) <=
    DFF_inst(137);          Q(138) <= DFF_inst
    (138);
Q(139) <= DFF_inst(139);        Q(140) <=
    DFF_inst(140);          Q(141) <= DFF_inst
    (141);
Q(142) <= DFF_inst(142);        Q(143) <=
    DFF_inst(143);          Q(144) <= DFF_inst
    (144);
Q(145) <= DFF_inst(145);        Q(146) <=
    DFF_inst(146);          Q(147) <= DFF_inst
    (147);
Q(148) <= DFF_inst(148);        Q(149) <=
    DFF_inst(149);          Q(150) <= DFF_inst
    (150);
Q(151) <= DFF_inst(151);        Q(152) <=
    DFF_inst(152);          Q(153) <= DFF_inst
    (153);
Q(154) <= DFF_inst(154);        Q(155) <=
    DFF_inst(155);          Q(156) <= DFF_inst
    (156);
Q(157) <= DFF_inst(157);        Q(158) <=
    DFF_inst(158);          Q(159) <= DFF_inst
    (159);
Q(160) <= DFF_inst(160);        Q(161) <=
    DFF_inst(161);          Q(162) <= DFF_inst
    (162);
Q(163) <= DFF_inst(163);        Q(164) <=
    DFF_inst(164);          Q(165) <= DFF_inst
    (165);
Q(166) <= DFF_inst(166);        Q(167) <=
    DFF_inst(167);          Q(168) <= DFF_inst
    (168);
Q(169) <= DFF_inst(169);        Q(170) <=
    DFF_inst(170);          Q(171) <= DFF_inst
    (171);
Q(172) <= DFF_inst(172);        Q(173) <=
    DFF_inst(173);          Q(174) <= DFF_inst
    (174);
Q(175) <= DFF_inst(175);        Q(176) <=
    DFF_inst(176);          Q(177) <= DFF_inst
    (177);
Q(178) <= DFF_inst(178);        Q(179) <=
    DFF_inst(179);          Q(180) <= DFF_inst
    (180);
Q(181) <= DFF_inst(181);        Q(182) <=
    DFF_inst(182);          Q(183) <= DFF_inst
    (183);
Q(184) <= DFF_inst(184);        Q(185) <=
    DFF_inst(185);          Q(186) <= DFF_inst
    (186);
Q(187) <= DFF_inst(187);        Q(188) <=
    DFF_inst(188);          Q(189) <= DFF_inst
    (189);
Q(190) <= DFF_inst(190);        Q(191) <=
    DFF_inst(191);          Q(192) <= DFF_inst
    (192);
Q(193) <= DFF_inst(193);        Q(194) <=
    DFF_inst(194);          Q(195) <= DFF_inst
    (195);
Q(196) <= DFF_inst(196);        Q(197) <=
    DFF_inst(197);          Q(198) <= DFF_inst
    (198);
Q(199) <= DFF_inst(199);        Q(200) <=
    DFF_inst(200);          Q(201) <= DFF_inst
    (201);
Q(202) <= DFF_inst(202);        Q(203) <=
    DFF_inst(203);          Q(204) <= DFF_inst
    (204);
Q(205) <= DFF_inst(205);        Q(206) <=
    DFF_inst(206);          Q(207) <= DFF_inst
    (207);
Q(208) <= DFF_inst(208);        Q(209) <=
    DFF_inst(209);          Q(210) <= DFF_inst
    (210);
Q(211) <= DFF_inst(211);        Q(212) <=
    DFF_inst(212);          Q(213) <= DFF_inst
    (213);
Q(214) <= DFF_inst(214);        Q(215) <=
    DFF_inst(215);          Q(216) <= DFF_inst
    (216);
Q(217) <= DFF_inst(217);        Q(218) <=
    DFF_inst(218);          Q(219) <= DFF_inst
    (219);
Q(220) <= DFF_inst(220);        Q(221) <=
    DFF_inst(221);          Q(222) <= DFF_inst
    (222);
Q(223) <= DFF_inst(223);        Q(224) <=
    DFF_inst(224);          Q(225) <= DFF_inst
    (225);
Q(226) <= DFF_inst(226);        Q(227) <=
    DFF_inst(227);          Q(228) <= DFF_inst
    (228);
Q(229) <= DFF_inst(229);        Q(230) <=
    DFF_inst(230);          Q(231) <= DFF_inst
    (231);
Q(232) <= DFF_inst(232);        Q(233) <=
    DFF_inst(233);          Q(234) <= DFF_inst
    (234);
Q(235) <= DFF_inst(235);        Q(236) <=
    DFF_inst(236);          Q(237) <= DFF_inst
    (237);
Q(238) <= DFF_inst(238);        Q(239) <=
    DFF_inst(239);          Q(240) <= DFF_inst
    (240);
Q(241) <= DFF_inst(241);        Q(242) <=
    DFF_inst(242);          Q(243) <= DFF_inst
    (243);
Q(244) <= DFF_inst(244);        Q(245) <=
    DFF_inst(245);          Q(246) <= DFF_inst
    (246);
Q(247) <= DFF_inst(247);        Q(248) <=
    DFF_inst(248);          Q(249) <= DFF_inst
    (249);
Q(250) <= DFF_inst(250);        Q(251) <=
    DFF_inst(251);          Q(252) <= DFF_inst
    (252);
Q(253) <= DFF_inst(253);        Q(254) <=
    DFF_inst(254);          Q(255) <= DFF_inst
    (255);
Q(256) <= DFF_inst(256);        Q(257) <=
    DFF_inst(257);          Q(258) <= DFF_inst
    (258);
Q(259) <= DFF_inst(259);        Q(260) <=
    DFF_inst(260);          Q(261) <= DFF_inst
    (261);
Q(262) <= DFF_inst(262);        Q(263) <=
    DFF_inst(263);          Q(264) <= DFF_inst
    (264);
Q(265) <= DFF_inst(265);        Q(266) <=
    DFF_inst(266);          Q(267) <= DFF_inst
    (267);
Q(268) <= DFF_inst(268);        Q(269) <=
    DFF_inst(269);          Q(270) <= DFF_inst
    (270);
Q(271) <= DFF_inst(271);        Q(272) <=
    DFF_inst(272);          Q(273) <= DFF_inst
    (273);
```

```
Q(274) <= DFF_inst(274);          Q(275) <=
    DFF_inst(275);            Q(276) <= DFF_inst
    (276);
Q(277) <= DFF_inst(277);          Q(278) <=
    DFF_inst(278);            Q(279) <= DFF_inst
    (279);
Q(280) <= DFF_inst(280);          Q(281) <=
    DFF_inst(281);            Q(282) <= DFF_inst
    (282);
Q(283) <= DFF_inst(283);          Q(284) <=
    DFF_inst(284);            Q(285) <= DFF_inst
    (285);
Q(286) <= DFF_inst(286);          Q(287) <=
    DFF_inst(287);            Q(288) <= DFF_inst
    (288);
Q(289) <= DFF_inst(289);          Q(290) <=
    DFF_inst(290);            Q(291) <= DFF_inst
    (291);
Q(292) <= DFF_inst(292);          Q(293) <=
    DFF_inst(293);            Q(294) <= DFF_inst
    (294);
Q(295) <= DFF_inst(295);          Q(296) <=
    DFF_inst(296);            Q(297) <= DFF_inst
    (297);
Q(298) <= DFF_inst(298);          Q(299) <=
    DFF_inst(299);            Q(300) <= DFF_inst
    (300);
Q(301) <= DFF_inst(301);          Q(302) <=
    DFF_inst(302);            Q(303) <= DFF_inst
    (303);
Q(304) <= DFF_inst(304);          Q(305) <=
    DFF_inst(305);            Q(306) <= DFF_inst
    (306);
Q(307) <= DFF_inst(307);          Q(308) <=
    DFF_inst(308);            Q(309) <= DFF_inst
    (309);
Q(310) <= DFF_inst(310);          Q(311) <=
    DFF_inst(311);            Q(312) <= DFF_inst
    (312);
Q(313) <= DFF_inst(313);          Q(314) <=
    DFF_inst(314);            Q(315) <= DFF_inst
    (315);
Q(316) <= DFF_inst(316);          Q(317) <=
    DFF_inst(317);            Q(318) <= DFF_inst
    (318);
Q(319) <= DFF_inst(319);          Q(320) <=
    DFF_inst(320);            Q(321) <= DFF_inst
    (321);
Q(322) <= DFF_inst(322);          Q(323) <=
    DFF_inst(323);            Q(324) <= DFF_inst
    (324);
Q(325) <= DFF_inst(325);          Q(326) <=
    DFF_inst(326);            Q(327) <= DFF_inst
    (327);
Q(328) <= DFF_inst(328);          Q(329) <=
    DFF_inst(329);            Q(330) <= DFF_inst
    (330);
Q(331) <= DFF_inst(331);          Q(332) <=
    DFF_inst(332);            Q(333) <= DFF_inst
    (333);
Q(334) <= DFF_inst(334);          Q(335) <=
    DFF_inst(335);            Q(336) <= DFF_inst
    (336);
Q(337) <= DFF_inst(337);          Q(338) <=
    DFF_inst(338);            Q(339) <= DFF_inst
    (339);
Q(340) <= DFF_inst(340);          Q(341) <=
    DFF_inst(341);            Q(342) <= DFF_inst
    (342);
Q(343) <= DFF_inst(343);          Q(344) <=
    DFF_inst(344);            Q(345) <= DFF_inst
    (345);
Q(346) <= DFF_inst(346);          Q(347) <=
    DFF_inst(347);            Q(348) <= DFF_inst
    (348);
Q(349) <= DFF_inst(349);          Q(350) <=
    DFF_inst(350);            Q(351) <= DFF_inst
    (351);



PROCESS(clk,stop_in,done)

BEGIN
if(stop_in'event) then
stop_data <= '1';
end if;
if(done'event) then
stop_data <= '0';
count_reg <= "000000000";
wht_cmd <= "UUUUU";
end if;
if(powerup = '0') then
IF (Preset = '0') THEN
DFF_inst(0) <= 'Z';
DFF_inst(1) <= 'Z';       DFF_inst(1) <= 'Z';
    DFF_inst(2) <= 'Z';
DFF_inst(4) <= 'Z';       DFF_inst(4) <= 'Z';
    DFF_inst(5) <= 'Z';
DFF_inst(7) <= 'Z';       DFF_inst(7) <= 'Z';
    DFF_inst(8) <= 'Z';
DFF_inst(10) <= 'Z';      DFF_inst(10) <= 'Z';
    DFF_inst(11) <= 'Z';
DFF_inst(13) <= 'Z';      DFF_inst(13) <= 'Z';
    DFF_inst(14) <= 'Z';
DFF_inst(16) <= 'Z';      DFF_inst(16) <= 'Z';
    DFF_inst(17) <= 'Z';
DFF_inst(19) <= 'Z';      DFF_inst(19) <= 'Z';
    DFF_inst(20) <= 'Z';
DFF_inst(22) <= 'Z';      DFF_inst(22) <= 'Z';
    DFF_inst(23) <= 'Z';
DFF_inst(25) <= 'Z';      DFF_inst(25) <= 'Z';
    DFF_inst(26) <= 'Z';
DFF_inst(28) <= 'Z';      DFF_inst(28) <= 'Z';
    DFF_inst(29) <= 'Z';
DFF_inst(31) <= 'Z';      DFF_inst(31) <= 'Z';
    DFF_inst(32) <= 'Z';
DFF_inst(34) <= 'Z';      DFF_inst(34) <= 'Z';
    DFF_inst(35) <= 'Z';
DFF_inst(37) <= 'Z';      DFF_inst(37) <= 'Z';
    DFF_inst(38) <= 'Z';
DFF_inst(40) <= 'Z';      DFF_inst(40) <= 'Z';
    DFF_inst(41) <= 'Z';
DFF_inst(43) <= 'Z';      DFF_inst(43) <= 'Z';
    DFF_inst(44) <= 'Z';
DFF_inst(46) <= 'Z';      DFF_inst(46) <= 'Z';
    DFF_inst(47) <= 'Z';
DFF_inst(49) <= 'Z';      DFF_inst(49) <= 'Z';
    DFF_inst(50) <= 'Z';
DFF_inst(52) <= 'Z';      DFF_inst(52) <= 'Z';
    DFF_inst(53) <= 'Z';
DFF_inst(55) <= 'Z';      DFF_inst(55) <= 'Z';
    DFF_inst(56) <= 'Z';
DFF_inst(58) <= 'Z';      DFF_inst(58) <= 'Z';
    DFF_inst(59) <= 'Z';
DFF_inst(61) <= 'Z';      DFF_inst(61) <= 'Z';
    DFF_inst(62) <= 'Z';
DFF_inst(64) <= 'Z';      DFF_inst(64) <= 'Z';
    DFF_inst(65) <= 'Z';
DFF_inst(67) <= 'Z';      DFF_inst(67) <= 'Z';
    DFF_inst(68) <= 'Z';
DFF_inst(70) <= 'Z';      DFF_inst(70) <= 'Z';
    DFF_inst(71) <= 'Z';
DFF_inst(73) <= 'Z';      DFF_inst(73) <= 'Z';
    DFF_inst(74) <= 'Z';
DFF_inst(76) <= 'Z';      DFF_inst(76) <= 'Z';
    DFF_inst(77) <= 'Z';
DFF_inst(79) <= 'Z';      DFF_inst(79) <= 'Z';
    DFF_inst(80) <= 'Z';
DFF_inst(82) <= 'Z';      DFF_inst(82) <= 'Z';
    DFF_inst(83) <= 'Z';
DFF_inst(85) <= 'Z';      DFF_inst(85) <= 'Z';
    DFF_inst(86) <= 'Z';
DFF_inst(88) <= 'Z';      DFF_inst(88) <= 'Z';
    DFF_inst(89) <= 'Z';
DFF_inst(91) <= 'Z';      DFF_inst(91) <= 'Z';
    DFF_inst(92) <= 'Z';
DFF_inst(94) <= 'Z';      DFF_inst(94) <= 'Z';
    DFF_inst(95) <= 'Z';
DFF_inst(97) <= 'Z';      DFF_inst(97) <= 'Z';
    DFF_inst(98) <= 'Z';
DFF_inst(100) <= 'Z';     DFF_inst(100) <= 'Z';
    DFF_inst(101) <= 'Z';
DFF_inst(103) <= 'Z';     DFF_inst(103) <= 'Z';
    DFF_inst(104) <= 'Z';
DFF_inst(106) <= 'Z';     DFF_inst(106) <= 'Z';
    DFF_inst(107) <= 'Z';
DFF_inst(109) <= 'Z';     DFF_inst(109) <= 'Z';
    DFF_inst(110) <= 'Z';
DFF_inst(112) <= 'Z';     DFF_inst(112) <= 'Z';
    DFF_inst(113) <= 'Z';
DFF_inst(115) <= 'Z';     DFF_inst(115) <= 'Z';
    DFF_inst(116) <= 'Z';
```

```
DFF_inst(118) <= 'Z';   DFF_inst(118) <= 'Z';
    DFF_inst(119) <= 'Z';
DFF_inst(121) <= 'Z';   DFF_inst(121) <= 'Z';
    DFF_inst(122) <= 'Z';
DFF_inst(124) <= 'Z';   DFF_inst(124) <= 'Z';
    DFF_inst(125) <= 'Z';
DFF_inst(127) <= 'Z';   DFF_inst(127) <= 'Z';
    DFF_inst(128) <= 'Z';
DFF_inst(130) <= 'Z';   DFF_inst(130) <= 'Z';
    DFF_inst(131) <= 'Z';
DFF_inst(133) <= 'Z';   DFF_inst(133) <= 'Z';
    DFF_inst(134) <= 'Z';
DFF_inst(136) <= 'Z';   DFF_inst(136) <= 'Z';
    DFF_inst(137) <= 'Z';
DFF_inst(139) <= 'Z';   DFF_inst(139) <= 'Z';
    DFF_inst(140) <= 'Z';
DFF_inst(142) <= 'Z';   DFF_inst(142) <= 'Z';
    DFF_inst(143) <= 'Z';
DFF_inst(145) <= 'Z';   DFF_inst(145) <= 'Z';
    DFF_inst(146) <= 'Z';
DFF_inst(148) <= 'Z';   DFF_inst(148) <= 'Z';
    DFF_inst(149) <= 'Z';
DFF_inst(151) <= 'Z';   DFF_inst(151) <= 'Z';
    DFF_inst(152) <= 'Z';
DFF_inst(154) <= 'Z';   DFF_inst(154) <= 'Z';
    DFF_inst(155) <= 'Z';
DFF_inst(157) <= 'Z';   DFF_inst(157) <= 'Z';
    DFF_inst(158) <= 'Z';
DFF_inst(160) <= 'Z';   DFF_inst(160) <= 'Z';
    DFF_inst(161) <= 'Z';
DFF_inst(163) <= 'Z';   DFF_inst(163) <= 'Z';
    DFF_inst(164) <= 'Z';
DFF_inst(166) <= 'Z';   DFF_inst(166) <= 'Z';
    DFF_inst(167) <= 'Z';
DFF_inst(169) <= 'Z';   DFF_inst(169) <= 'Z';
    DFF_inst(170) <= 'Z';
DFF_inst(172) <= 'Z';   DFF_inst(172) <= 'Z';
    DFF_inst(173) <= 'Z';
DFF_inst(175) <= 'Z';   DFF_inst(175) <= 'Z';
    DFF_inst(176) <= 'Z';
DFF_inst(178) <= 'Z';   DFF_inst(178) <= 'Z';
    DFF_inst(179) <= 'Z';
DFF_inst(181) <= 'Z';   DFF_inst(181) <= 'Z';
    DFF_inst(182) <= 'Z';
DFF_inst(184) <= 'Z';   DFF_inst(184) <= 'Z';
    DFF_inst(185) <= 'Z';
DFF_inst(187) <= 'Z';   DFF_inst(187) <= 'Z';
    DFF_inst(188) <= 'Z';
DFF_inst(190) <= 'Z';   DFF_inst(190) <= 'Z';
    DFF_inst(191) <= 'Z';
DFF_inst(193) <= 'Z';   DFF_inst(193) <= 'Z';
    DFF_inst(194) <= 'Z';
DFF_inst(196) <= 'Z';   DFF_inst(196) <= 'Z';
    DFF_inst(197) <= 'Z';
DFF_inst(199) <= 'Z';   DFF_inst(199) <= 'Z';
    DFF_inst(200) <= 'Z';
DFF_inst(202) <= 'Z';   DFF_inst(202) <= 'Z';
    DFF_inst(203) <= 'Z';
DFF_inst(205) <= 'Z';   DFF_inst(205) <= 'Z';
    DFF_inst(206) <= 'Z';
DFF_inst(208) <= 'Z';   DFF_inst(208) <= 'Z';
    DFF_inst(209) <= 'Z';
DFF_inst(211) <= 'Z';   DFF_inst(211) <= 'Z';
    DFF_inst(212) <= 'Z';
DFF_inst(214) <= 'Z';   DFF_inst(214) <= 'Z';
    DFF_inst(215) <= 'Z';
DFF_inst(217) <= 'Z';   DFF_inst(217) <= 'Z';
    DFF_inst(218) <= 'Z';
DFF_inst(220) <= 'Z';   DFF_inst(220) <= 'Z';
    DFF_inst(221) <= 'Z';
DFF_inst(223) <= 'Z';   DFF_inst(223) <= 'Z';
    DFF_inst(224) <= 'Z';
DFF_inst(226) <= 'Z';   DFF_inst(226) <= 'Z';
    DFF_inst(227) <= 'Z';
DFF_inst(229) <= 'Z';   DFF_inst(229) <= 'Z';
    DFF_inst(230) <= 'Z';
DFF_inst(232) <= 'Z';   DFF_inst(232) <= 'Z';
    DFF_inst(233) <= 'Z';
DFF_inst(235) <= 'Z';   DFF_inst(235) <= 'Z';
    DFF_inst(236) <= 'Z';
DFF_inst(238) <= 'Z';   DFF_inst(238) <= 'Z';
    DFF_inst(239) <= 'Z';
DFF_inst(241) <= 'Z';   DFF_inst(241) <= 'Z';
    DFF_inst(242) <= 'Z';
DFF_inst(244) <= 'Z';   DFF_inst(244) <= 'Z';
    DFF_inst(245) <= 'Z';
DFF_inst(247) <= 'Z';   DFF_inst(247) <= 'Z';
    DFF_inst(248) <= 'Z';
DFF_inst(250) <= 'Z';   DFF_inst(250) <= 'Z';
    DFF_inst(251) <= 'Z';
DFF_inst(253) <= 'Z';   DFF_inst(253) <= 'Z';
    DFF_inst(254) <= 'Z';
DFF_inst(256) <= 'Z';   DFF_inst(256) <= 'Z';
    DFF_inst(257) <= 'Z';
DFF_inst(259) <= 'Z';   DFF_inst(259) <= 'Z';
    DFF_inst(260) <= 'Z';
DFF_inst(262) <= 'Z';   DFF_inst(262) <= 'Z';
    DFF_inst(263) <= 'Z';
DFF_inst(265) <= 'Z';   DFF_inst(265) <= 'Z';
    DFF_inst(266) <= 'Z';
DFF_inst(268) <= 'Z';   DFF_inst(268) <= 'Z';
    DFF_inst(269) <= 'Z';
DFF_inst(271) <= 'Z';   DFF_inst(271) <= 'Z';
    DFF_inst(272) <= 'Z';
DFF_inst(274) <= 'Z';   DFF_inst(274) <= 'Z';
    DFF_inst(275) <= 'Z';
DFF_inst(277) <= 'Z';   DFF_inst(277) <= 'Z';
    DFF_inst(278) <= 'Z';
DFF_inst(280) <= 'Z';   DFF_inst(280) <= 'Z';
    DFF_inst(281) <= 'Z';
DFF_inst(283) <= 'Z';   DFF_inst(283) <= 'Z';
    DFF_inst(284) <= 'Z';
DFF_inst(286) <= 'Z';   DFF_inst(286) <= 'Z';
    DFF_inst(287) <= 'Z';
DFF_inst(289) <= 'Z';   DFF_inst(289) <= 'Z';
    DFF_inst(290) <= 'Z';
DFF_inst(292) <= 'Z';   DFF_inst(292) <= 'Z';
    DFF_inst(293) <= 'Z';
DFF_inst(295) <= 'Z';   DFF_inst(295) <= 'Z';
    DFF_inst(296) <= 'Z';
DFF_inst(298) <= 'Z';   DFF_inst(298) <= 'Z';
    DFF_inst(299) <= 'Z';
DFF_inst(301) <= 'Z';   DFF_inst(301) <= 'Z';
    DFF_inst(302) <= 'Z';
DFF_inst(304) <= 'Z';   DFF_inst(304) <= 'Z';
    DFF_inst(305) <= 'Z';
DFF_inst(307) <= 'Z';   DFF_inst(307) <= 'Z';
    DFF_inst(308) <= 'Z';
DFF_inst(310) <= 'Z';   DFF_inst(310) <= 'Z';
    DFF_inst(311) <= 'Z';
DFF_inst(313) <= 'Z';   DFF_inst(313) <= 'Z';
    DFF_inst(314) <= 'Z';
DFF_inst(316) <= 'Z';   DFF_inst(316) <= 'Z';
    DFF_inst(317) <= 'Z';
DFF_inst(319) <= 'Z';   DFF_inst(319) <= 'Z';
    DFF_inst(320) <= 'Z';
DFF_inst(322) <= 'Z';   DFF_inst(322) <= 'Z';
    DFF_inst(323) <= 'Z';
DFF_inst(325) <= 'Z';   DFF_inst(325) <= 'Z';
    DFF_inst(326) <= 'Z';
DFF_inst(328) <= 'Z';   DFF_inst(328) <= 'Z';
    DFF_inst(329) <= 'Z';
DFF_inst(331) <= 'Z';   DFF_inst(331) <= 'Z';
    DFF_inst(332) <= 'Z';
DFF_inst(334) <= 'Z';   DFF_inst(334) <= 'Z';
    DFF_inst(335) <= 'Z';
DFF_inst(337) <= 'Z';   DFF_inst(337) <= 'Z';
    DFF_inst(338) <= 'Z';
DFF_inst(340) <= 'Z';   DFF_inst(340) <= 'Z';
    DFF_inst(341) <= 'Z';
DFF_inst(343) <= 'Z';   DFF_inst(343) <= 'Z';
    DFF_inst(344) <= 'Z';
DFF_inst(346) <= 'Z';   DFF_inst(346) <= 'Z';
    DFF_inst(347) <= 'Z';
DFF_inst(349) <= 'Z';   DFF_inst(349) <= 'Z';
    DFF_inst(350) <= 'Z';

mask_value <= '0';
---------
count_reg <= "000000000"; -- initialisation
bit_len <= "ZZZZZZZZZ";
stop_sgl <= '0';
bit_len1 <= "ZZZZZZZZZ";
wht_cmd <= "ZZZZZ";
stop_data <= '0';
end if;
ELSIF (RISING_EDGE(clk)) THEN
count_reg <= count_reg + 1;
Mask_len <= std_logic_vector(mask_length);
what_cmd <= wht_cmd;
--      stop <= stop_sgl;

EBV_sel <= std_logic_vector(EBV);
```

```
if( stop_data = '0')  then

DFF_inst(0)  <= data;
DFF_inst(1) <= DFF_inst(0);      DFF_inst(2) <=
    DFF_inst(1);      DFF_inst(3) <= DFF_inst(2)
    ;
DFF_inst(4) <= DFF_inst(3);      DFF_inst(5) <=
    DFF_inst(4);      DFF_inst(6) <= DFF_inst(5)
    ;
DFF_inst(7) <= DFF_inst(6);      DFF_inst(8) <=
    DFF_inst(7);      DFF_inst(9) <= DFF_inst(8)
    ;
DFF_inst(10) <= DFF_inst(9);     DFF_inst(11) <=
    DFF_inst(10);    DFF_inst(12) <= DFF_inst
    (11);
DFF_inst(13) <= DFF_inst(12);    DFF_inst(14) <=
    DFF_inst(13);    DFF_inst(15) <= DFF_inst
    (14);
DFF_inst(16) <= DFF_inst(15);    DFF_inst(17) <=
    DFF_inst(16);    DFF_inst(18) <= DFF_inst
    (17);
DFF_inst(19) <= DFF_inst(18);    DFF_inst(20) <=
    DFF_inst(19);    DFF_inst(21) <= DFF_inst
    (20);
DFF_inst(22) <= DFF_inst(21);    DFF_inst(23) <=
    DFF_inst(22);    DFF_inst(24) <= DFF_inst
    (23);
DFF_inst(25) <= DFF_inst(24);    DFF_inst(26) <=
    DFF_inst(25);    DFF_inst(27) <= DFF_inst
    (26);
DFF_inst(28) <= DFF_inst(27);    DFF_inst(29) <=
    DFF_inst(28);    DFF_inst(30) <= DFF_inst
    (29);
DFF_inst(31) <= DFF_inst(30);    DFF_inst(32) <=
    DFF_inst(31);    DFF_inst(33) <= DFF_inst
    (32);
DFF_inst(34) <= DFF_inst(33);    DFF_inst(35) <=
    DFF_inst(34);    DFF_inst(36) <= DFF_inst
    (35);
DFF_inst(37) <= DFF_inst(36);    DFF_inst(38) <=
    DFF_inst(37);    DFF_inst(39) <= DFF_inst
    (38);
DFF_inst(40) <= DFF_inst(39);    DFF_inst(41) <=
    DFF_inst(40);    DFF_inst(42) <= DFF_inst
    (41);
DFF_inst(43) <= DFF_inst(42);    DFF_inst(44) <=
    DFF_inst(43);    DFF_inst(45) <= DFF_inst
    (44);
DFF_inst(46) <= DFF_inst(45);    DFF_inst(47) <=
    DFF_inst(46);    DFF_inst(48) <= DFF_inst
    (47);
DFF_inst(49) <= DFF_inst(48);    DFF_inst(50) <=
    DFF_inst(49);    DFF_inst(51) <= DFF_inst
    (50);
DFF_inst(52) <= DFF_inst(51);    DFF_inst(53) <=
    DFF_inst(52);    DFF_inst(54) <= DFF_inst
    (53);
DFF_inst(55) <= DFF_inst(54);    DFF_inst(56) <=
    DFF_inst(55);    DFF_inst(57) <= DFF_inst
    (56);
DFF_inst(58) <= DFF_inst(57);    DFF_inst(59) <=
    DFF_inst(58);    DFF_inst(60) <= DFF_inst
    (59);
DFF_inst(61) <= DFF_inst(60);    DFF_inst(62) <=
    DFF_inst(61);    DFF_inst(63) <= DFF_inst
    (62);
DFF_inst(64) <= DFF_inst(63);    DFF_inst(65) <=
    DFF_inst(64);    DFF_inst(66) <= DFF_inst
    (65);
DFF_inst(67) <= DFF_inst(66);    DFF_inst(68) <=
    DFF_inst(67);    DFF_inst(69) <= DFF_inst
    (68);
DFF_inst(70) <= DFF_inst(69);    DFF_inst(71) <=
    DFF_inst(70);    DFF_inst(72) <= DFF_inst
    (71);
DFF_inst(73) <= DFF_inst(72);    DFF_inst(74) <=
    DFF_inst(73);    DFF_inst(75) <= DFF_inst
    (74);
DFF_inst(76) <= DFF_inst(75);    DFF_inst(77) <=
    DFF_inst(76);    DFF_inst(78) <= DFF_inst
    (77);
DFF_inst(79) <= DFF_inst(78);    DFF_inst(80) <=
    DFF_inst(79);    DFF_inst(81) <= DFF_inst
    (80);
DFF_inst(82) <= DFF_inst(81);    DFF_inst(83) <=
    DFF_inst(82);    DFF_inst(84) <= DFF_inst
    (83);
DFF_inst(85) <= DFF_inst(84);    DFF_inst(86) <=
    DFF_inst(85);    DFF_inst(87) <= DFF_inst
    (86);
DFF_inst(88) <= DFF_inst(87);    DFF_inst(89) <=
    DFF_inst(88);    DFF_inst(90) <= DFF_inst
    (89);
DFF_inst(91) <= DFF_inst(90);    DFF_inst(92) <=
    DFF_inst(91);    DFF_inst(93) <= DFF_inst
    (92);
DFF_inst(94) <= DFF_inst(93);    DFF_inst(95) <=
    DFF_inst(94);    DFF_inst(96) <= DFF_inst
    (95);
DFF_inst(97) <= DFF_inst(96);    DFF_inst(98) <=
    DFF_inst(97);    DFF_inst(99) <= DFF_inst
    (98);
DFF_inst(100) <= DFF_inst(99);   DFF_inst(101) <=
     DFF_inst(100); DFF_inst(102) <= DFF_inst
    (101);
DFF_inst(103) <= DFF_inst(102); DFF_inst(104) <=
     DFF_inst(103); DFF_inst(105) <= DFF_inst
    (104);
DFF_inst(106) <= DFF_inst(105); DFF_inst(107) <=
     DFF_inst(106); DFF_inst(108) <= DFF_inst
    (107);
DFF_inst(109) <= DFF_inst(108); DFF_inst(110) <=
     DFF_inst(109); DFF_inst(111) <= DFF_inst
    (110);
DFF_inst(112) <= DFF_inst(111); DFF_inst(113) <=
     DFF_inst(112); DFF_inst(114) <= DFF_inst
    (113);
DFF_inst(115) <= DFF_inst(114); DFF_inst(116) <=
     DFF_inst(115); DFF_inst(117) <= DFF_inst
    (116);
DFF_inst(118) <= DFF_inst(117); DFF_inst(119) <=
     DFF_inst(118); DFF_inst(120) <= DFF_inst
    (119);
DFF_inst(121) <= DFF_inst(120); DFF_inst(122) <=
     DFF_inst(121); DFF_inst(123) <= DFF_inst
    (122);
DFF_inst(124) <= DFF_inst(123); DFF_inst(125) <=
     DFF_inst(124); DFF_inst(126) <= DFF_inst
    (125);
DFF_inst(127) <= DFF_inst(126); DFF_inst(128) <=
     DFF_inst(127); DFF_inst(129) <= DFF_inst
    (128);
DFF_inst(130) <= DFF_inst(129); DFF_inst(131) <=
     DFF_inst(130); DFF_inst(132) <= DFF_inst
    (131);
DFF_inst(133) <= DFF_inst(132); DFF_inst(134) <=
     DFF_inst(133); DFF_inst(135) <= DFF_inst
    (134);
DFF_inst(136) <= DFF_inst(135); DFF_inst(137) <=
     DFF_inst(136); DFF_inst(138) <= DFF_inst
    (137);
DFF_inst(139) <= DFF_inst(138); DFF_inst(140) <=
     DFF_inst(139); DFF_inst(141) <= DFF_inst
    (140);
DFF_inst(142) <= DFF_inst(141); DFF_inst(143) <=
     DFF_inst(142); DFF_inst(144) <= DFF_inst
    (143);
DFF_inst(145) <= DFF_inst(144); DFF_inst(146) <=
     DFF_inst(145); DFF_inst(147) <= DFF_inst
    (146);
DFF_inst(148) <= DFF_inst(147); DFF_inst(149) <=
     DFF_inst(148); DFF_inst(150) <= DFF_inst
    (149);
DFF_inst(151) <= DFF_inst(150); DFF_inst(152) <=
     DFF_inst(151); DFF_inst(153) <= DFF_inst
    (152);
DFF_inst(154) <= DFF_inst(153); DFF_inst(155) <=
     DFF_inst(154); DFF_inst(156) <= DFF_inst
    (155);
DFF_inst(157) <= DFF_inst(156); DFF_inst(158) <=
     DFF_inst(157); DFF_inst(159) <= DFF_inst
    (158);
DFF_inst(160) <= DFF_inst(159); DFF_inst(161) <=
     DFF_inst(160); DFF_inst(162) <= DFF_inst
    (161);
DFF_inst(163) <= DFF_inst(162); DFF_inst(164) <=
     DFF_inst(163); DFF_inst(165) <= DFF_inst
    (164);
DFF_inst(166) <= DFF_inst(165); DFF_inst(167) <=
     DFF_inst(166); DFF_inst(168) <= DFF_inst
    (167);
```

```
DFF_inst(169) <= DFF_inst(168); DFF_inst(170) <=
      DFF_inst(169); DFF_inst(171) <= DFF_inst
     (170);
DFF_inst(172) <= DFF_inst(171); DFF_inst(173) <=
      DFF_inst(172); DFF_inst(174) <= DFF_inst
     (173);
DFF_inst(175) <= DFF_inst(174); DFF_inst(176) <=
      DFF_inst(175); DFF_inst(177) <= DFF_inst
     (176);
DFF_inst(178) <= DFF_inst(177); DFF_inst(179) <=
      DFF_inst(178); DFF_inst(180) <= DFF_inst
     (179);
DFF_inst(181) <= DFF_inst(180); DFF_inst(182) <=
      DFF_inst(181); DFF_inst(183) <= DFF_inst
     (182);
DFF_inst(184) <= DFF_inst(183); DFF_inst(185) <=
      DFF_inst(184); DFF_inst(186) <= DFF_inst
     (185);
DFF_inst(187) <= DFF_inst(186); DFF_inst(188) <=
      DFF_inst(187); DFF_inst(189) <= DFF_inst
     (188);
DFF_inst(190) <= DFF_inst(189); DFF_inst(191) <=
      DFF_inst(190); DFF_inst(192) <= DFF_inst
     (191);
DFF_inst(193) <= DFF_inst(192); DFF_inst(194) <=
      DFF_inst(193); DFF_inst(195) <= DFF_inst
     (194);
DFF_inst(196) <= DFF_inst(195); DFF_inst(197) <=
      DFF_inst(196); DFF_inst(198) <= DFF_inst
     (197);
DFF_inst(199) <= DFF_inst(198); DFF_inst(200) <=
      DFF_inst(199); DFF_inst(201) <= DFF_inst
     (200);
DFF_inst(202) <= DFF_inst(201); DFF_inst(203) <=
      DFF_inst(202); DFF_inst(204) <= DFF_inst
     (203);
DFF_inst(205) <= DFF_inst(204); DFF_inst(206) <=
      DFF_inst(205); DFF_inst(207) <= DFF_inst
     (206);
DFF_inst(208) <= DFF_inst(207); DFF_inst(209) <=
      DFF_inst(208); DFF_inst(210) <= DFF_inst
     (209);
DFF_inst(211) <= DFF_inst(210); DFF_inst(212) <=
      DFF_inst(211); DFF_inst(213) <= DFF_inst
     (212);
DFF_inst(214) <= DFF_inst(213); DFF_inst(215) <=
      DFF_inst(214); DFF_inst(216) <= DFF_inst
     (215);
DFF_inst(217) <= DFF_inst(216); DFF_inst(218) <=
      DFF_inst(217); DFF_inst(219) <= DFF_inst
     (218);
DFF_inst(220) <= DFF_inst(219); DFF_inst(221) <=
      DFF_inst(220); DFF_inst(222) <= DFF_inst
     (221);
DFF_inst(223) <= DFF_inst(222); DFF_inst(224) <=
      DFF_inst(223); DFF_inst(225) <= DFF_inst
     (224);
DFF_inst(226) <= DFF_inst(225); DFF_inst(227) <=
      DFF_inst(226); DFF_inst(228) <= DFF_inst
     (227);
DFF_inst(229) <= DFF_inst(228); DFF_inst(230) <=
      DFF_inst(229); DFF_inst(231) <= DFF_inst
     (230);
DFF_inst(232) <= DFF_inst(231); DFF_inst(233) <=
      DFF_inst(232); DFF_inst(234) <= DFF_inst
     (233);
DFF_inst(235) <= DFF_inst(234); DFF_inst(236) <=
      DFF_inst(235); DFF_inst(237) <= DFF_inst
     (236);
DFF_inst(238) <= DFF_inst(237); DFF_inst(239) <=
      DFF_inst(238); DFF_inst(240) <= DFF_inst
     (239);
DFF_inst(241) <= DFF_inst(240); DFF_inst(242) <=
      DFF_inst(241); DFF_inst(243) <= DFF_inst
     (242);
DFF_inst(244) <= DFF_inst(243); DFF_inst(245) <=
      DFF_inst(244); DFF_inst(246) <= DFF_inst
     (245);
DFF_inst(247) <= DFF_inst(246); DFF_inst(248) <=
      DFF_inst(247); DFF_inst(249) <= DFF_inst
     (248);
DFF_inst(250) <= DFF_inst(249); DFF_inst(251) <=
      DFF_inst(250); DFF_inst(252) <= DFF_inst
     (251);
DFF_inst(253) <= DFF_inst(252); DFF_inst(254) <=
      DFF_inst(253); DFF_inst(255) <= DFF_inst
     (254);
DFF_inst(256) <= DFF_inst(255); DFF_inst(257) <=
      DFF_inst(256); DFF_inst(258) <= DFF_inst
     (257);
DFF_inst(259) <= DFF_inst(258); DFF_inst(260) <=
      DFF_inst(259); DFF_inst(261) <= DFF_inst
     (260);
DFF_inst(262) <= DFF_inst(261); DFF_inst(263) <=
      DFF_inst(262); DFF_inst(264) <= DFF_inst
     (263);
DFF_inst(265) <= DFF_inst(264); DFF_inst(266) <=
      DFF_inst(265); DFF_inst(267) <= DFF_inst
     (266);
DFF_inst(268) <= DFF_inst(267); DFF_inst(269) <=
      DFF_inst(268); DFF_inst(270) <= DFF_inst
     (269);
DFF_inst(271) <= DFF_inst(270); DFF_inst(272) <=
      DFF_inst(271); DFF_inst(273) <= DFF_inst
     (272);
DFF_inst(274) <= DFF_inst(273); DFF_inst(275) <=
      DFF_inst(274); DFF_inst(276) <= DFF_inst
     (275);
DFF_inst(277) <= DFF_inst(276); DFF_inst(278) <=
      DFF_inst(277); DFF_inst(279) <= DFF_inst
     (278);
DFF_inst(280) <= DFF_inst(279); DFF_inst(281) <=
      DFF_inst(280); DFF_inst(282) <= DFF_inst
     (281);
DFF_inst(283) <= DFF_inst(282); DFF_inst(284) <=
      DFF_inst(283); DFF_inst(285) <= DFF_inst
     (284);
DFF_inst(286) <= DFF_inst(285); DFF_inst(287) <=
      DFF_inst(286); DFF_inst(288) <= DFF_inst
     (287);
DFF_inst(289) <= DFF_inst(288); DFF_inst(290) <=
      DFF_inst(289); DFF_inst(291) <= DFF_inst
     (290);
DFF_inst(292) <= DFF_inst(291); DFF_inst(293) <=
      DFF_inst(292); DFF_inst(294) <= DFF_inst
     (293);
DFF_inst(295) <= DFF_inst(294); DFF_inst(296) <=
      DFF_inst(295); DFF_inst(297) <= DFF_inst
     (296);
DFF_inst(298) <= DFF_inst(297); DFF_inst(299) <=
      DFF_inst(298); DFF_inst(300) <= DFF_inst
     (299);
DFF_inst(301) <= DFF_inst(300); DFF_inst(302) <=
      DFF_inst(301); DFF_inst(303) <= DFF_inst
     (302);
DFF_inst(304) <= DFF_inst(303); DFF_inst(305) <=
      DFF_inst(304); DFF_inst(306) <= DFF_inst
     (305);
DFF_inst(307) <= DFF_inst(306); DFF_inst(308) <=
      DFF_inst(307); DFF_inst(309) <= DFF_inst
     (308);
DFF_inst(310) <= DFF_inst(309); DFF_inst(311) <=
      DFF_inst(310); DFF_inst(312) <= DFF_inst
     (311);
DFF_inst(313) <= DFF_inst(312); DFF_inst(314) <=
      DFF_inst(313); DFF_inst(315) <= DFF_inst
     (314);
DFF_inst(316) <= DFF_inst(315); DFF_inst(317) <=
      DFF_inst(316); DFF_inst(318) <= DFF_inst
     (317);
DFF_inst(319) <= DFF_inst(318); DFF_inst(320) <=
      DFF_inst(319); DFF_inst(321) <= DFF_inst
     (320);
DFF_inst(322) <= DFF_inst(321); DFF_inst(323) <=
      DFF_inst(322); DFF_inst(324) <= DFF_inst
     (323);
DFF_inst(325) <= DFF_inst(324); DFF_inst(326) <=
      DFF_inst(325); DFF_inst(327) <= DFF_inst
     (326);
DFF_inst(328) <= DFF_inst(327); DFF_inst(329) <=
      DFF_inst(328); DFF_inst(330) <= DFF_inst
     (329);
DFF_inst(331) <= DFF_inst(330); DFF_inst(332) <=
      DFF_inst(331); DFF_inst(333) <= DFF_inst
     (332);
DFF_inst(334) <= DFF_inst(333); DFF_inst(335) <=
      DFF_inst(334); DFF_inst(336) <= DFF_inst
     (335);
DFF_inst(337) <= DFF_inst(336); DFF_inst(338) <=
      DFF_inst(337); DFF_inst(339) <= DFF_inst
     (338);
DFF_inst(340) <= DFF_inst(339); DFF_inst(341) <=
      DFF_inst(340); DFF_inst(342) <= DFF_inst
     (341);
```

```
DFF_inst(343) <= DFF_inst(342); DFF_inst(344) <=
     DFF_inst(343); DFF_inst(345) <= DFF_inst
    (344);
DFF_inst(346) <= DFF_inst(345); DFF_inst(347) <=
     DFF_inst(346); DFF_inst(348) <= DFF_inst
    (347);
DFF_inst(349) <= DFF_inst(348); DFF_inst(350) <=
     DFF_inst(349); DFF_inst(351) <= DFF_inst
    (350);

mask_val(1) <= mask_val(0);      mask_val(2) <=
    mask_val(1);      mask_val(3) <= mask_val(2)
    ;
mask_val(4) <= mask_val(3);      mask_val(5) <=
    mask_val(4);      mask_val(6) <= mask_val(5)
    ;
mask_val(7) <= mask_val(6);      mask_val(8) <=
    mask_val(7);      mask_val(9) <= mask_val(8)
    ;
mask_val(10) <= mask_val(9);     mask_val(11) <=
    mask_val(10);    mask_val(12) <= mask_val
    (11);
mask_val(13) <= mask_val(12);    mask_val(14) <=
    mask_val(13);    mask_val(15) <= mask_val
    (14);
mask_val(16) <= mask_val(15);    mask_val(17) <=
    mask_val(16);    mask_val(18) <= mask_val
    (17);
mask_val(19) <= mask_val(18);    mask_val(20) <=
    mask_val(19);    mask_val(21) <= mask_val
    (20);
mask_val(22) <= mask_val(21);    mask_val(23) <=
    mask_val(22);    mask_val(24) <= mask_val
    (23);
mask_val(25) <= mask_val(24);    mask_val(26) <=
    mask_val(25);    mask_val(27) <= mask_val
    (26);
mask_val(28) <= mask_val(27);    mask_val(29) <=
    mask_val(28);    mask_val(30) <= mask_val
    (29);
mask_val(31) <= mask_val(30);    mask_val(32) <=
    mask_val(31);    mask_val(33) <= mask_val
    (32);
mask_val(34) <= mask_val(33);    mask_val(35) <=
    mask_val(34);    mask_val(36) <= mask_val
    (35);
mask_val(37) <= mask_val(36);    mask_val(38) <=
    mask_val(37);    mask_val(39) <= mask_val
    (38);
mask_val(40) <= mask_val(39);    mask_val(41) <=
    mask_val(40);    mask_val(42) <= mask_val
    (41);
mask_val(43) <= mask_val(42);    mask_val(44) <=
    mask_val(43);    mask_val(45) <= mask_val
    (44);
mask_val(46) <= mask_val(45);    mask_val(47) <=
    mask_val(46);    mask_val(48) <= mask_val
    (47);
mask_val(49) <= mask_val(48);    mask_val(50) <=
    mask_val(49);    mask_val(51) <= mask_val
    (50);
mask_val(52) <= mask_val(51);    mask_val(53) <=
    mask_val(52);    mask_val(54) <= mask_val
    (53);
mask_val(55) <= mask_val(54);    mask_val(56) <=
    mask_val(55);    mask_val(57) <= mask_val
    (56);
mask_val(58) <= mask_val(57);    mask_val(59) <=
    mask_val(58);    mask_val(60) <= mask_val
    (59);
mask_val(61) <= mask_val(60);    mask_val(62) <=
    mask_val(61);    mask_val(63) <= mask_val
    (62);
mask_val(64) <= mask_val(63);    mask_val(65) <=
    mask_val(64);    mask_val(66) <= mask_val
    (65);
mask_val(67) <= mask_val(66);    mask_val(68) <=
    mask_val(67);    mask_val(69) <= mask_val
    (68);
mask_val(70) <= mask_val(69);    mask_val(71) <=
    mask_val(70);    mask_val(72) <= mask_val
    (71);
mask_val(73) <= mask_val(72);    mask_val(74) <=
    mask_val(73);    mask_val(75) <= mask_val
    (74);
mask_val(76) <= mask_val(75);    mask_val(77) <=
    mask_val(76);    mask_val(78) <= mask_val
    (77);
mask_val(79) <= mask_val(78);    mask_val(80) <=
    mask_val(79);    mask_val(81) <= mask_val
    (80);
mask_val(82) <= mask_val(81);    mask_val(83) <=
    mask_val(82);    mask_val(84) <= mask_val
    (83);
mask_val(85) <= mask_val(84);    mask_val(86) <=
    mask_val(85);    mask_val(87) <= mask_val
    (86);
mask_val(88) <= mask_val(87);    mask_val(89) <=
    mask_val(88);    mask_val(90) <= mask_val
    (89);
mask_val(91) <= mask_val(90);    mask_val(92) <=
    mask_val(91);    mask_val(93) <= mask_val
    (92);
mask_val(94) <= mask_val(93);    mask_val(95) <=
    mask_val(94);    mask_val(96) <= mask_val
    (95);
mask_val(97) <= mask_val(96);    mask_val(98) <=
    mask_val(97);    mask_val(99) <= mask_val
    (98);
mask_val(100) <= mask_val(99);   mask_val(101) <=
     mask_val(100); mask_val(102) <= mask_val
    (101);
mask_val(103) <= mask_val(102); mask_val(104) <=
     mask_val(103); mask_val(105) <= mask_val
    (104);
mask_val(106) <= mask_val(105); mask_val(107) <=
     mask_val(106); mask_val(108) <= mask_val
    (107);
mask_val(109) <= mask_val(108); mask_val(110) <=
     mask_val(109); mask_val(111) <= mask_val
    (110);
mask_val(112) <= mask_val(111); mask_val(113) <=
     mask_val(112); mask_val(114) <= mask_val
    (113);
mask_val(115) <= mask_val(114); mask_val(116) <=
     mask_val(115); mask_val(117) <= mask_val
    (116);
mask_val(118) <= mask_val(117); mask_val(119) <=
     mask_val(118); mask_val(120) <= mask_val
    (119);
mask_val(121) <= mask_val(120); mask_val(122) <=
     mask_val(121); mask_val(123) <= mask_val
    (122);
mask_val(124) <= mask_val(123); mask_val(125) <=
     mask_val(124); mask_val(126) <= mask_val
    (125);
mask_val(127) <= mask_val(126); mask_val(128) <=
     mask_val(127); mask_val(129) <= mask_val
    (128);
mask_val(130) <= mask_val(129); mask_val(131) <=
     mask_val(130); mask_val(132) <= mask_val
    (131);
mask_val(133) <= mask_val(132); mask_val(134) <=
     mask_val(133); mask_val(135) <= mask_val
    (134);
mask_val(136) <= mask_val(135); mask_val(137) <=
     mask_val(136); mask_val(138) <= mask_val
    (137);
mask_val(139) <= mask_val(138); mask_val(140) <=
     mask_val(139); mask_val(141) <= mask_val
    (140);
mask_val(142) <= mask_val(141); mask_val(143) <=
     mask_val(142); mask_val(144) <= mask_val
    (143);
mask_val(145) <= mask_val(144); mask_val(146) <=
     mask_val(145); mask_val(147) <= mask_val
    (146);
mask_val(148) <= mask_val(147); mask_val(149) <=
     mask_val(148); mask_val(150) <= mask_val
    (149);
mask_val(151) <= mask_val(150); mask_val(152) <=
     mask_val(151); mask_val(153) <= mask_val
    (152);
mask_val(154) <= mask_val(153); mask_val(155) <=
     mask_val(154); mask_val(156) <= mask_val
    (155);
mask_val(157) <= mask_val(156); mask_val(158) <=
     mask_val(157); mask_val(159) <= mask_val
    (158);
mask_val(160) <= mask_val(159); mask_val(161) <=
     mask_val(160); mask_val(162) <= mask_val
    (161);
```

```
mask_val(163) <= mask_val(162); mask_val(164) <=
      mask_val(163); mask_val(165) <= mask_val
     (164);
mask_val(166) <= mask_val(165); mask_val(167) <=
      mask_val(166); mask_val(168) <= mask_val
     (167);
mask_val(169) <= mask_val(168); mask_val(170) <=
      mask_val(169); mask_val(171) <= mask_val
     (170);
mask_val(172) <= mask_val(171); mask_val(173) <=
      mask_val(172); mask_val(174) <= mask_val
     (173);
mask_val(175) <= mask_val(174); mask_val(176) <=
      mask_val(175); mask_val(177) <= mask_val
     (176);
mask_val(178) <= mask_val(177); mask_val(179) <=
      mask_val(178); mask_val(180) <= mask_val
     (179);
mask_val(181) <= mask_val(180); mask_val(182) <=
      mask_val(181); mask_val(183) <= mask_val
     (182);
mask_val(184) <= mask_val(183); mask_val(185) <=
      mask_val(184); mask_val(186) <= mask_val
     (185);
mask_val(187) <= mask_val(186); mask_val(188) <=
      mask_val(187); mask_val(189) <= mask_val
     (188);
mask_val(190) <= mask_val(189); mask_val(191) <=
      mask_val(190); mask_val(192) <= mask_val
     (191);
mask_val(193) <= mask_val(192); mask_val(194) <=
      mask_val(193); mask_val(195) <= mask_val
     (194);
mask_val(196) <= mask_val(195); mask_val(197) <=
      mask_val(196); mask_val(198) <= mask_val
     (197);
mask_val(199) <= mask_val(198); mask_val(200) <=
      mask_val(199); mask_val(201) <= mask_val
     (200);
mask_val(202) <= mask_val(201); mask_val(203) <=
      mask_val(202); mask_val(204) <= mask_val
     (203);
mask_val(205) <= mask_val(204); mask_val(206) <=
      mask_val(205); mask_val(207) <= mask_val
     (206);
mask_val(208) <= mask_val(207); mask_val(209) <=
      mask_val(208); mask_val(210) <= mask_val
     (209);
mask_val(211) <= mask_val(210); mask_val(212) <=
      mask_val(211); mask_val(213) <= mask_val
     (212);
mask_val(214) <= mask_val(213); mask_val(215) <=
      mask_val(214); mask_val(216) <= mask_val
     (215);
mask_val(217) <= mask_val(216); mask_val(218) <=
      mask_val(217); mask_val(219) <= mask_val
     (218);
mask_val(220) <= mask_val(219); mask_val(221) <=
      mask_val(220); mask_val(222) <= mask_val
     (221);
mask_val(223) <= mask_val(222); mask_val(224) <=
      mask_val(223); mask_val(225) <= mask_val
     (224);
mask_val(226) <= mask_val(225); mask_val(227) <=
      mask_val(226); mask_val(228) <= mask_val
     (227);
mask_val(229) <= mask_val(228); mask_val(230) <=
      mask_val(229); mask_val(231) <= mask_val
     (230);
mask_val(232) <= mask_val(231); mask_val(233) <=
      mask_val(232); mask_val(234) <= mask_val
     (233);
mask_val(235) <= mask_val(234); mask_val(236) <=
      mask_val(235); mask_val(237) <= mask_val
     (236);
mask_val(238) <= mask_val(237); mask_val(239) <=
      mask_val(238); mask_val(240) <= mask_val
     (239);
mask_val(241) <= mask_val(240); mask_val(242) <=
      mask_val(241); mask_val(243) <= mask_val
     (242);
mask_val(244) <= mask_val(243); mask_val(245) <=
      mask_val(244); mask_val(246) <= mask_val
     (245);
mask_val(247) <= mask_val(246); mask_val(248) <=
      mask_val(247); mask_val(249) <= mask_val
     (248);
mask_val(250) <= mask_val(249); mask_val(251) <=
      mask_val(250); mask_val(252) <= mask_val
     (251);
mask_val(253) <= mask_val(252); mask_val(254) <=
      mask_val(253); mask_val(255) <= mask_val
     (254);


Out_Serial <= data;


-- CONTROL UNIT

--CASE: INITIAL 2 or 4 or 8 bit command check

case (count_reg ) is

when "000000100" =>      -- Check the first 2 or
     4 bits to detect the command, at the 4th
     clock cycle
-- If a QueryRep command
if (DFF_inst(3) = '0' and DFF_inst(2) = '0')
     then
wht_cmd <= "00000";
bit_len <= "000000100";  fixed <= '1'; valid <=
     '1';    crc_flag <= "00";
end if;
-- Ack
if (DFF_inst(3) = '0' and DFF_inst(2) = '1')
     then
wht_cmd <= "00001";
bit_len <= "000010010"; fixed <='1';  valid <=
     '1'; crc_flag <= "00";
end if;
-- Query
if (DFF_inst(3) = '1' and DFF_inst(2) = '0' and
     DFF_inst(1) = '0' and DFF_inst(0) = '0')
     then
wht_cmd <= "00010";
bit_len <= "000010110"; fixed <='1';  valid <=
     '1'; crc_flag <= "01";
end if;
-- QueryAdjust
if (DFF_inst(3) = '1' and DFF_inst(2) = '0' and
     DFF_inst(1) = '0' and DFF_inst(0) = '1')
     then
wht_cmd <= "00011";
bit_len <= "000001001"; fixed <='1';  valid <=
     '1'; crc_flag <= "00";
end if;
-- Select
if (DFF_inst(3) = '1' and DFF_inst(2) = '0' and
     DFF_inst(1) = '1' and DFF_inst(0) = '0')
     then
wht_cmd <= "00100";
bit_len <= "000100101"; fixed <='0';  valid <=
     '0'; crc_flag <= "10";
end if;

when "000001000" =>   -- Check the first 8 bits
     to detect the command, at the 8th clock
     cycle
-- Challenge
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '1' and DFF_inst(2)= '0' and
     DFF_inst(1)= '1' and DFF_inst(0)= '0') then
wht_cmd <= "10000";
bit_len <= "001001000"; fixed <='1'; valid <=
     '1'; crc_flag <= "00";
end if;

-- NAK
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '0' and
     DFF_inst(1)= '0' and DFF_inst(0)= '0') then
wht_cmd <= "00110";
bit_len <= "000001000"; fixed <='1'; valid <=
     '1'; crc_flag <= "00";
end if;
-- Req_RN
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '0' and
     DFF_inst(1)= '0' and DFF_inst(0)= '1') then
wht_cmd <= "00111";
```

```
bit_len <= "000101000"; fixed<='1'; valid <=
     '1'; crc_flag <= "10";
end if;
-- Read
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '0' and
     DFF_inst(1)= '1' and DFF_inst(0)= '0') then
wht_cmd <= "01000";
bit_len <= "000110010"; fixed<='0'; valid <=
     '0'; crc_flag <= "10";
end if;
-- Write
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '0' and
     DFF_inst(1)= '1' and DFF_inst(0)= '1') then
wht_cmd <= "01001";
bit_len <= "000110011"; fixed<='0'; valid <=
     '0'; crc_flag <= "10";
end if;
-- Kill
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '1' and
     DFF_inst(1)= '0' and DFF_inst(0)= '0') then
wht_cmd <= "01010";
bit_len <= "000111011"; fixed<='1'; valid <=
     '1'; crc_flag <= "10";
end if;
-- Lock
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '1' and
     DFF_inst(1)= '0' and DFF_inst(0)= '1') then
wht_cmd <= "01011";
bit_len <= "000111100"; fixed<='1'; valid <=
     '1'; crc_flag <= "10";
end if;
--Access
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '1' and
     DFF_inst(1)= '1' and DFF_inst(0)= '0') then
wht_cmd <= "01100";
bit_len <= "000111000"; fixed<='1'; valid <=
     '1';
end if;
-- BlockWrite
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '0' and DFF_inst(2)= '1' and
     DFF_inst(1)= '1' and DFF_inst(0)= '1') then
wht_cmd <= "01101";
bit_len <= "000110010"; fixed<='0'; valid <=
     '0';
end if;
-- BlockErase
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '1' and DFF_inst(2)= '0' and
     DFF_inst(1)= '0' and DFF_inst(0)= '0') then
wht_cmd <= "01110";
bit_len <= "000110010"; fixed<='0'; valid <=
     '0';
end if;
-- BlockPermalock
if (DFF_inst(7)= '1' and DFF_inst(6)= '1' and
     DFF_inst(5)= '0' and DFF_inst(4)= '0' and
     DFF_inst(3)= '1' and DFF_inst(2)= '0' and
     DFF_inst(1)= '0' and DFF_inst(0)= '1') then
wht_cmd <= "01111";
bit_len <= "000111011"; fixed<='0'; valid <=
     '0';
end if;

when others => bit_len <= bit_len;

end case;
----CASE: INITIAL 2 or 4 or 8 bit command check


if(valid='0') then

case (bit_len) is
when "000100101" =>          -- Select command
if(count_reg="000001101") then
if(DFF_inst(0)='0') then
bit_len1 <= bit_len + 8; --"000001000";
EBV <= "00";
else
bit_len1 <= bit_len + "000010000";
EBV <= "01";
end if;
end if;
if(count_reg="000010101") then
if(DFF_inst(0)='1' and EBV="01") then
bit_len1 <= bit_len + "000011000";
EBV <= "10";
end if;
end if;

when "000110010" =>      -- Read command
if(count_reg="000001011") then
if(DFF_inst(0)='0') then
bit_len1 <= bit_len+"000001000";
EBV_Read <= "00";
valid <= '1';
elsif(DFF_inst(0)='1') then
bit_len1 <= bit_len+"000010000";
EBV_Read <= "01";
valid <= '1';
end if;
elsif(count_reg="000010011") then
if(DFF_inst(0)='1') then
bit_len1 <= bit_len+"000011000";
EBV_Read <= "10";
valid <= '1';
end if;
end if;

when "000110011" =>      -- Write Command
if(count_reg="000001011") then
if(DFF_inst(0)='0') then
bit_len1 <= bit_len+"000001000";
EBV_write <= "00";
valid <= '1';
elsif(DFF_inst(0)='1') then
bit_len1 <= bit_len+"000010000";
EBV_write <= "01";
valid <= '1';
end if;
elsif(count_reg="000010011") then
if(DFF_inst(0)='1') then
bit_len1 <= bit_len+"000011000";
EBV_write <= "10";
valid <= '1';
end if;
end if;

when others => bit_len1 <= bit_len1;
end case;


if(wht_cmd = "00100") then                --
     WHT_CMD?  Ans: Select CMD

-- Dealing with the Mask length of SELECT
     command
if((EBV="00" and count_reg="000011100") or (EBV=
     "01" and count_reg="000100100") or (EBV="10
     " and count_reg="000101100")) then
mask_length <= DFF_inst(7) & DFF_inst(6) &
     DFF_inst(5) & DFF_inst(4) & DFF_inst(3) &
     DFF_inst(2) & DFF_inst(1) & DFF_inst(0);
bit_len1 <= bit_len1 + mask_length;
valid <= '1';
end if;   -- Dealing with the Mask length of
     SELECT command

if(EBV="00") then
if(count_reg="000010100") then
          -- Collecting "Target, Action,
     Memory Bank and Pointer" values.
target_sel <= DFF_inst(15 downto 13);
action_sel <= DFF_inst(12 downto 10);
membnk_sel <= DFF_inst(9 downto 8);
pointer_sel(7 downto 0) <= DFF_inst(7 downto 0);
end if;
if((count_reg > 28) and (count_reg < (bit_len1
     -17))) then       -- Collecting "Mask Value
      - 256 bits"
mask_val(0) <= DFF_inst(0);
end if;
```

```
end if;

if(EBV="01") then
    -- Collecting "Target, Action, Memory Bank
    and Pointer" values.
if(count_reg="000011100") then
target_sel <= DFF_inst(23 downto 21);
action_sel <= DFF_inst(20 downto 18);
membnk_sel <= DFF_inst(17 downto 16);
pointer_sel(15 downto 0) <= DFF_inst(15 downto
    0);
end if;
if((count_reg > 36) and (count_reg < (bit_len1
    -17))) then        -- Collecting "Mask Value
     - 256 bits"
mask_val(0) <= DFF_inst(0);
end if;
end if;

if(EBV="10") then
if(count_reg="00100100") then
            -- Collecting "Target, Action,
    Memory Bank and Pointer" values.
target_sel <= DFF_inst(31 downto 29);
action_sel <= DFF_inst(28 downto 26);
membnk_sel <= DFF_inst(25 downto 24);
pointer_sel <= DFF_inst(23 downto 0);
end if;
if((count_reg > 44) and (count_reg < (bit_len1
    -17))) then        -- Collecting "Mask Value
     - 256 bits"
mask_val(0) <= DFF_inst(0);
end if;
end if;
end if;
end if;          -- END Valid = 0
if (fixed = '0' and valid = '1') then
if ((count_reg+1) = (bit_len1) ) then
stop <= not(stop_sgl);          bit_len_out <=
    std_logic_vector(bit_len1); stop_sgl <= not
    (stop_sgl);
end if;
end if;
if (fixed = '1' and valid = '1') then
if ((count_reg+1) = bit_len) then
stop <= not(stop_sgl);          bit_len_out <=
    std_logic_vector(bit_len); stop_sgl <= not(
    stop_sgl);
end if;
end if;

end if;
End if;
END PROCESS;
END RTL_CMD;
```

Listing A.6. CRC-5 Engine

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

LIBRARY work;

ENTITY Test_CRC IS
PORT
(
data : IN STD_LOGIC;
Preset,stop,done,powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
crc_valid : OUT STD_LOGIC;
Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
END Test_CRC;

ARCHITECTURE bdf_type OF Test_CRC IS

SIGNAL SYNTHESIZED_WIRE_3, stop_crc :
    STD_LOGIC;
SIGNAL DFF_inst : STD_LOGIC;
SIGNAL DFF_inst1 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_1 : STD_LOGIC;
SIGNAL DFF_inst3 : STD_LOGIC;
SIGNAL DFF_inst4 : STD_LOGIC;
SIGNAL DFF_inst2 : STD_LOGIC;
signal crc_value :      STD_LOGIC_VECTOR(4
    downto 0);
signal count : unsigned(1 downto 0);

BEGIN
Q4 <= DFF_inst4;
Q3 <= DFF_inst3;
Q2 <= DFF_inst2;
Q1 <= DFF_inst1;
Q0 <= DFF_inst;


PROCESS(clk,stop,done)
BEGIN
if(powerup = '0') then
IF (Preset = '0') THEN
DFF_inst <= '1'; DFF_inst1 <= '0'; DFF_inst2 <=
    '0'; DFF_inst3 <= '1'; DFF_inst4 <= '0';
    stop_crc <= '0'; count <= "00";
end if;
ELSIF (RISING_EDGE(clk)) THEN
if(stop_crc = '1') then
if(crc_flag = "00") then
crc_valid <= '1';
else
if(count = "01") then
DFF_inst <= SYNTHESIZED_WIRE_3;
DFF_inst1 <= DFF_inst;
DFF_inst2 <= DFF_inst1;
DFF_inst3 <= SYNTHESIZED_WIRE_1;
DFF_inst4 <= DFF_inst3;
count <= count+1;
else
stop_crc <= 'U';
if(DFF_inst4 = '0' and DFF_inst3 ='0' and
    DFF_inst2 = '0' and DFF_inst1 = '0' and
    DFF_inst = '0') then
crc_valid <= '1';
else
crc_valid <= '0';
end if;
DFF_inst <= '1'; DFF_inst1 <= '0'; DFF_inst2 <=
    '0'; DFF_inst3 <= '1'; DFF_inst4 <= '0';
end if;
end if;

elsif(stop_crc = '0') then
if(count = "00")then
DFF_inst <= '1'; DFF_inst1 <= '0'; DFF_inst2 <=
    '0'; DFF_inst3 <= '1'; DFF_inst4 <= '0';
count <= count+1;
else
DFF_inst <= SYNTHESIZED_WIRE_3;
DFF_inst1 <= DFF_inst;
DFF_inst2 <= DFF_inst1;
DFF_inst3 <= SYNTHESIZED_WIRE_1;
DFF_inst4 <= DFF_inst3;
end if;
end if;
End if;
if(stop'event) then
stop_crc <= '1';
end if;
if(done'event) then
stop_crc <= '0';
count <= "00";
crc_valid <= 'U';
end if;
END PROCESS;
SYNTHESIZED_WIRE_3 <= DFF_inst4 XOR data;
SYNTHESIZED_WIRE_1 <= SYNTHESIZED_WIRE_3 XOR
    DFF_inst2;

END bdf_type;
```

Listing A.7. CRC-16 Engine

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
```

```
LIBRARY work;

ENTITY CRC16 IS
PORT
(
Preset,powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
data, stop,done,key_crc : IN STD_LOGIC;
crc_valid,key_crc_done : OUT STD_LOGIC;
crc_flag : IN STD_LOGIC_VECTOR(1 downto 0);
crc_plain_text: out std_logic_vector(15 downto
    0);
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
    , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC
);
END CRC16;

ARCHITECTURE bdf_type OF CRC16 IS

SIGNAL SYNTHESIZED_WIRE_5, DFF_inst11,
    DFF_inst1, DFF_inst2, DFF_inst3,
    SYNTHESIZED_WIRE_4, DFF_inst5, stop_crc :
    STD_LOGIC;
SIGNAL DFF_inst, DFF_inst9, DFF_inst10,
    SYNTHESIZED_WIRE_1, DFF_inst14, DFF_inst15,
     DFF_inst12, DFF_inst13, DFF_inst4 :
    STD_LOGIC;
SIGNAL DFF_inst6, DFF_inst7, DFF_inst8 :
    STD_LOGIC;
signal crc_value : STD_LOGIC_VECTOR(15
    downto 0);
signal count_key : unsigned(5 downto 0);
signal count : unsigned(1 downto 0);

BEGIN
Q15 <= DFF_inst13; Q14 <= DFF_inst14; Q13 <=
    DFF_inst15; Q12 <= DFF_inst12; Q11 <=
    DFF_inst11; Q10 <= DFF_inst10; Q9 <=
    DFF_inst9; Q8 <= DFF_inst8;
Q7 <= DFF_inst7; Q6 <= DFF_inst6; Q5 <=
    DFF_inst5; Q4 <= DFF_inst4; Q3 <= DFF_inst3
    ; Q2 <= DFF_inst2; Q1 <= DFF_inst1; Q0 <=
    DFF_inst;

PROCESS(clk,stop,done)
BEGIN
if(powerup = '0') then
IF (Preset = '0') THEN
DFF_inst <= '1'; DFF_inst1 <= '1'; DFF_inst2 <=
    '1'; DFF_inst3 <= '1'; DFF_inst4 <= '1';
    DFF_inst5 <= '1'; DFF_inst6 <= '1';
    DFF_inst7 <= '1';
DFF_inst8 <= '1'; DFF_inst9 <= '1'; DFF_inst10
    <= '1'; DFF_inst11 <= '1'; DFF_inst12 <=
    '1'; DFF_inst13 <= '1'; DFF_inst14 <= '1';
DFF_inst15 <= '1'; stop_crc <= '0'; count <= "00
    "; count_key <= "000000"; key_crc_done <=
    '0';
end if;
ELSIF (RISING_EDGE(clk)) THEN
if(stop_crc = '1') then
if(crc_flag = "00") then
crc_valid <= '1';
else
if(count = "01") then
count <= count+1;
DFF_inst <= SYNTHESIZED_WIRE_5; DFF_inst1 <=
    DFF_inst; DFF_inst2 <= DFF_inst1;
    DFF_inst3 <= DFF_inst2; DFF_inst4 <=
    DFF_inst3;
DFF_inst5 <= SYNTHESIZED_WIRE_4; DFF_inst6 <=
    DFF_inst5; DFF_inst7 <= DFF_inst6;
    DFF_inst8 <= DFF_inst7;
DFF_inst9 <= DFF_inst8; DFF_inst10 <= DFF_inst9;
     DFF_inst11 <= DFF_inst10; DFF_inst12 <=
    SYNTHESIZED_WIRE_1; DFF_inst13 <=
    DFF_inst14;
DFF_inst14 <= DFF_inst15; DFF_inst15 <=
    DFF_inst12;
else
stop_crc <= 'U';
if(DFF_inst15 = '0' and DFF_inst14 ='0' and
    DFF_inst13 = '0' and DFF_inst12 = '1' and
    DFF_inst11 = '1' and DFF_inst10 = '1' and
    DFF_inst9 ='0' and DFF_inst8 = '1' and
    DFF_inst7 = '0' and DFF_inst6 = '0' and
    DFF_inst5 = '0' and DFF_inst4 = '0' and
    DFF_inst3 ='1' and DFF_inst2 = '1' and
    DFF_inst1 = '1' and DFF_inst = '1' ) then
crc_valid <= '1';
else
crc_valid <= '0';
end if;
DFF_inst <= '1'; DFF_inst1 <= '1'; DFF_inst2 <=
    '1'; DFF_inst3 <= '1'; DFF_inst4 <= '1';
    DFF_inst5 <= '1'; DFF_inst6 <= '1';
DFF_inst7 <= '1'; DFF_inst8 <= '1'; DFF_inst9 <=
     '1'; DFF_inst10 <= '1'; DFF_inst11 <= '1';
     DFF_inst12 <= '1'; DFF_inst13 <= '1';
DFF_inst14 <= '1'; DFF_inst15 <= '1';
end if;
end if;
elsif(stop_crc = '0') then
if(count = "00") then
DFF_inst <= '1'; DFF_inst1 <= '1'; DFF_inst2 <=
    '1'; DFF_inst3 <= '1'; DFF_inst4 <= '1';
    DFF_inst5 <= '1'; DFF_inst6 <= '1';
    DFF_inst7 <= '1';
DFF_inst8 <= '1'; DFF_inst9 <= '1'; DFF_inst10
    <= '1'; DFF_inst11 <= '1'; DFF_inst12 <=
    '1'; DFF_inst13 <= '1'; DFF_inst14 <= '1';
DFF_inst15 <= '1';

count <= count+1;
else

DFF_inst <= SYNTHESIZED_WIRE_5; DFF_inst1 <=
    DFF_inst; DFF_inst2 <= DFF_inst1;
    DFF_inst3 <= DFF_inst2; DFF_inst4 <=
    DFF_inst3;
DFF_inst5 <= SYNTHESIZED_WIRE_4; DFF_inst6 <=
    DFF_inst5; DFF_inst7 <= DFF_inst6;
    DFF_inst8 <= DFF_inst7;
DFF_inst9 <= DFF_inst8; DFF_inst10 <= DFF_inst9;
     DFF_inst11 <= DFF_inst10; DFF_inst12 <=
    SYNTHESIZED_WIRE_1; DFF_inst13 <=
    DFF_inst14;
DFF_inst14 <= DFF_inst15; DFF_inst15 <=
    DFF_inst12;
end if;
end if;
END IF;

if(stop'event) then
stop_crc <= '1';
end if;
if(done'event) then
stop_crc <= '0';
crc_valid <= 'U';
count <= "00";
end if;
END PROCESS;

SYNTHESIZED_WIRE_5 <= DFF_inst13 XOR data;
SYNTHESIZED_WIRE_4 <= SYNTHESIZED_WIRE_5 XOR
    DFF_inst4;
SYNTHESIZED_WIRE_1 <= SYNTHESIZED_WIRE_5 XOR
    DFF_inst11;
END bdf_type;
```

Listing A.8. Finite State Machine Module

```
--VHDL IMPLEMENTATION OF GEN2 PROTOCOL.



--Including IEEE libraries.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

-- In an entity named "gen2", all required
     commands are initialised to the required
     bit lengths as specified in the protocol.
entity FSM_1 is
port(

-- IN PORT
clk: in std_logic;--clock signal.
```

```
Preset,powerup: in std_logic;
done,crc_valid, mask_done,Reserved_done,
    match_sel,stop,RN16_done,RN64_done,
    slot_done: in std_logic;
ebv, ebv_read, ebv_write,membnk_sel: in
    std_logic_vector(1 downto 0);
target_sel, action_sel: in std_logic_vector(2
    downto 0);
wht_cmd: in std_logic_vector(4 downto 0);
mask_length: in std_logic_vector(7 downto 0);
bit_len_in: in std_logic_vector(8 downto 0);
RN_16_in,slot_count,data_in_16: in
    std_logic_vector(15 downto 0);
pointer_sel: in std_logic_vector(23 downto 0);
data_in: in std_logic_vector(351 downto 0);

-- OUT PORTS --
RN_16,RN_64, truncate, start_match,start_buffer,
    kill_count_out, Reserved_access,no_reply:
    out std_logic;
EPC_done: out std_logic:='0';
EPC: out std_logic_vector(95 downto 0);
MemBnk_out: out std_logic_vector(1 downto 0);
OpenSecureMemRef, error_code: out std_logic;
WrData_out, storedCRC: out std_logic_vector(15
    downto 0);
Q_value_out: out std_logic_vector(3 downto 0);
start_address_out, end_address_out: out
    std_logic_vector(23 downto 0);
challenge_data : out std_logic_vector(63 downto
    0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: out std_logic_vector(1 downto 0)
);
end FSM_1;
--End of Entity GEN2.

architecture RTL_FSM of FSM_1 is --Architecture

type state_type is (ready, arbitrate, reply,
    acknowledged, secured, openstate, killed );
signal next_state, current_state: state_type;
signal RN_16_sig,RN_64_sig,Inventory_match,
    trunc_select,Match_select,PrevQuery_kill,
    Query_complete, Req_RN_flag,
    start_buffer_sgl,error_code_sgl,
    mask_done_sgl: std_logic;
signal RdMemAcc_valid, WrMemAcc_valid,
    access_password, select_valid,access_count,
    kill_count,crc_valid_sgl,stop_crc,
    Reserved_done_sgl,count_test,RN16_done_sgl,
    RN64_done_sgl,slot_done_sgl : std_logic;
signal SL_query, Session, mem_bnk, SL_select,
    WrMemBnk,RdMemBnk: std_logic_vector(1
    downto 0);
signal Q_value: unsigned(3 downto 0);
signal slot: unsigned(15 downto 0);
signal mask_value : std_logic_vector(255 downto
    0);
signal slot_stdlogic, session_inventory_flag:
    std_logic_vector(3 downto 0);
signal pointer_select, WrPntr, RdPntr:
    std_logic_vector(23 downto 0);
signal bit_len: unsigned(8 downto 0);
signal get_pc_crc_kill,epc_count: unsigned(2
    downto 0);
signal storedpc, AccPass_MSB, AccPass_LSB,
    KillPass_MSB, WrData: std_logic_vector(15
    downto 0);
signal KillPass_LSB, KillPassReader_MSB,
    KillPassReader_LSB: std_logic_vector(15
    downto 0);
signal RdWrdCnt: std_logic_vector(7 downto 0);
BEGIN
process(clk,Reserved_done,mask_done,RN16_done,
    done,slot_done,RN64_done)
begin
if (RISING_EDGE(clk)) THEN
if(powerup = '0') then
IF (Preset = '0') THEN

Req_RN_flag <= '0';
no_reply <= '0';
stop_crc <= '0';
Reserved_done_sgl <= '0';
mask_done_sgl <= '0';
get_pc_crc_kill <= "000";
epc_count <= "000";
Reserved_access <= '0';
current_state <= ready;
OpenSecureMemRef <= 'Z';
SL_select <= "00";
PrevQuery_kill <= '0';
session_inventory_flag <= "0000";
trunc_select <= '0';
start_buffer_sgl <= '0';
current_state <= ready;
next_state <= ready;
Query_complete <= 'Z';
select_valid <= '1';
kill_count <= '0';
access_count <= '0';
RdMemBnk <= "ZZ";
RdWrdCnt <= "ZZZZZZZZ";
RdMemAcc_valid <= '0';
WrMemBnk <= "ZZ";
RdPntr <= "000000000000000000000000";
WrPntr <= "000000000000000000000000";
pointer_select<= "000000000000000000000000";
WrMemAcc_valid <= '0';
count_test <= '0';
RN_16_sig <= '0';
RN_64_sig <= '0';
RN16_done_sgl <= '0';
RN64_done_sgl <= '0';
slot_done_sgl <= '0';
Lock_Kill<= "00"; Lock_Access <= "00"; Lock_EPC
    <= "00"; Lock_TID <= "00"; Lock_User <= "00
    ";
elsif (Preset = '1') THEN
if(get_pc_crc_kill = "000") then
if(epc_count = "000") then
start_address_out <= "000000000000000000100000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(95 downto 80 ) <= data_in_16;
end if;
elsif(epc_count = "001") then
start_address_out <= "000000000000000000110000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(79 downto 64 ) <= data_in_16;
end if;
elsif(epc_count = "010") then
start_address_out <= "000000000000000001000000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(63 downto 48 ) <= data_in_16;
end if;
elsif(epc_count = "011") then
start_address_out <= "000000000000000001010000";
```

```
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
RN_64 <= NOT(RN_64_sig);
RN_64_sig <= NOT(RN_64_sig);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(47 downto 32 ) <= data_in_16;
end if;
elsif(epc_count = "100") then
start_address_out <= "00000000000000000001100000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(31 downto 16 ) <= data_in_16;
end if;
elsif(epc_count = "101") then
start_address_out <= "00000000000000000001110000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
epc_count <= epc_count + 1;
get_pc_crc_kill <= get_pc_crc_kill + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
EPC(15 downto 0 ) <= data_in_16;
EPC_done <= '1';
end if;
end if;
end if;
if(get_pc_crc_kill = "001") then
start_address_out <= "00000000000000000000010000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
get_pc_crc_kill <= get_pc_crc_kill + 1;
Reserved_done_sgl <= '0';
start_buffer <= '0';
Reserved_access <= '0';
count_test <= '0';
storedPC(4 downto 0) <= data_in_16(15 downto 11)
    ;
end if;
end if;
if(get_pc_crc_kill = "010") then
start_address_out <= "00000000000000000000000000";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
MemBnk_out <= "01";
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
get_pc_crc_kill <= get_pc_crc_kill + 1;
Reserved_access <= '0';
Reserved_done_sgl <= '0';
count_test <= '0';
storedCRC <= data_in_16;
end if;
end if;
if(get_pc_crc_kill = "011") then
start_address_out <= "00000000000000000000000000";
MemBnk_out <= "00";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
Reserved_access <= '0';
Reserved_done_sgl <= '0';
count_test <= '0';
KillPass_MSB <= data_in_16;
get_pc_crc_kill <= get_pc_crc_kill + 1;
end if;
end if;
if(get_pc_crc_kill = "100") then
start_address_out <= "00000000000000000000010000";
MemBnk_out <= "00";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
Reserved_access <= '0';
Reserved_done_sgl <= '0';
KillPass_LSB <= data_in_16;
count_test <= '0';
get_pc_crc_kill <= get_pc_crc_kill + 1;
end if;
end if;
if(get_pc_crc_kill = "101") then
start_address_out <= "00000000000000000000100000";
MemBnk_out <= "00";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
Reserved_access <= '0';
Reserved_done_sgl <= '0';
count_test <= '0';
AccPass_MSB <= data_in_16;
get_pc_crc_kill <= get_pc_crc_kill + 1;
end if;
end if;
if(get_pc_crc_kill = "110") then
start_address_out <= "00000000000000000000110000";
MemBnk_out <= "00";
if(count_test = '0') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
count_test <= '1';
end if;
Reserved_access <= '1';
if(Reserved_done_sgl = '1') then
Reserved_access <= '0';
Reserved_done_sgl <= '0';
count_test <= '0';
AccPass_LSB <= data_in_16;
get_pc_crc_kill <= get_pc_crc_kill + 1;
end if;
end if;
end if;
elsif(powerup = '1') then

crc_valid_sgl <= crc_valid;
Q_value_out <= std_logic_vector(Q_value);
bit_len <= unsigned(bit_len_in);
pointer_select <= pointer_sel;
kill_count_out <= kill_count;

if ( crc_valid = '1' ) then
case current_state is
when ready =>
case wht_cmd is
when "00010" =>          -- when Query CMD
Session <= data_in(11 downto 10);
```

```
case Session is
when "00" =>
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "01" =>
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "10" =>
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "11" =>
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when others =>
-- next_state <= current_state;
-- start_buffer <= not(start_buffer_sgl);
-- start_buffer_sgl <= not(start_buffer_sgl);
-- no_reply <= '1';
end case;
if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
    (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;


when "00100" =>          -- when Select CMD
MemBnk_out <= membnk_sel;
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';                    -- GIVE
    SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1'; --select_valid <= '1';
else
Match_select <= '0'; --select_valid <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
    mask_length)) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) then
Match_select <= '1'; -- select_valid <= '1';
else
Match_select <= '0'; -- select_valid <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                          --EPC Memory
if( (unsigned(pointer_select) + unsigned(
    mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                          --TID Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                          --User Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
```

```
no_reply <= '1';
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>           -- Session S0
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
else
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
else
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>           -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
else
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
else
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
else
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
else
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
```

```
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
else
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));              --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
else
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));              --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>            -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);            --
    change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
SL_select <= "11";              -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";              -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);            --
    change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
elsif(select_valid = '0') then
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when arbitrate =>
case wht_cmd is
when "00010" =>          -- when Query CMD
Session <= data_in(11 downto 10);
case Session is
when "00" =>
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "01" =>
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "10" =>
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "11" =>
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
    (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
end if;
```

```
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
when "00000"      =>              -- when QueryRep
     CMD
if(Query_complete = '1') then
if (Session = data_in(1 downto 0)) then
slot <= slot - 1;
if(slot ="0000000000000000") then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;

when "00011" =>                   -- when
     QueryAdjust
if(Query_complete = '1') then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
if (Session = data_in(4 downto 3)) then
case data_in(2 downto 0) is
when "000" => Q_value <= Q_value;
when "011" =>
if(Q_value = "0000") then
Q_value <= Q_value;
else
Q_value <= Q_value - 1;
end if;
when "110" =>
if(Q_value = "1111") then
Q_value <= Q_value;
else
Q_value <= Q_value + 1;
end if;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
end if;
end if;

when "00100" =>          -- when Select CMD
MemBnk_out <= membnk_sel;
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';                    -- GIVE
     SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
     storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1'; --select_valid <= '1';
else
Match_select <= '0'; --select_valid <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
     mask_length)) < ((unsigned(storedpc(4
     downto 0)) * 16)+ 32)) then
Match_select <= '1'; -- select_valid <= '1';
else
Match_select <= '0'; -- select_valid <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                          --EPC Memory
if( (unsigned(pointer_select) + unsigned(
     mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                          --TID Memory
if((unsigned(pointer_select) + unsigned(
     mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                          --User Memory
if((unsigned(pointer_select) + unsigned(
     mask_length)) <= "00001111") then
start_address_out <= pointer_select;
```

```
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>           -- Session S0
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
else
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
else
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>           -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
else
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
else
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
else
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
else
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
```

```
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
else
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
else
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>           -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);         --
    change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
SL_select <= "11";              -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";              -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);         --
    change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
elsif(select_valid = '0') then
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when reply =>
case wht_cmd is
when "00010" =>          -- when Query CMD
Session <= data_in(11 downto 10);
case Session is
when "00" =>
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "01" =>
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "10" =>
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when "11" =>
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
when others =>
next_state<= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
    (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
```

```
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;

when "00000"      =>                -- when QueryRep
     CMD
if(Query_complete = '1' and Session = data_in(1
    downto 0)) then
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
when "00011" =>                     -- when
     QueryAdjust
if(Query_complete = '1') then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
if (Session = data_in(4 downto 3)) then
case data_in(2 downto 0) is
when "000" => Q_value <= Q_value;
when "011" =>
if(Q_value = "0000") then
Q_value <= Q_value;
else
Q_value <= Q_value - 1;
end if;
when "110" =>
if(Q_value = "1111") then
Q_value <= Q_value;
else
Q_value <= Q_value + 1;
end if;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
end if;
end if;
```

```
when "00001" =>           -- when ACK CMD
     ; SHOULD GO TO THE RESPONSE MODULE
MemBnk_out <= "01";
-- if (data_in(15 downto 0) = RN_16_in(15 downto
     0) ) then
-- next_state <= acknowledged;
-- truncate <= trunc_select;
-- if(trunc_select = '0')then
-- start_address_out <=
     "00000000000000000000010000";
-- end_address_out <= std_logic_vector
     ("00000000000000000000000000" + ((unsigned(
     storedpc(4 downto 0)) * 16)+ 32));
-- elsif(trunc_select = '1')then
-- start_address_out <= std_logic_vector(
     unsigned(pointer_select) + unsigned(
     mask_length) + 1);
-- end_address_out <= std_logic_vector
     ("00000000000000000000000000" + ((unsigned(
     storedpc(4 downto 0)) * 16)+ 32));
-- end if;
-- start_buffer <= not(start_buffer_sgl);
-- start_buffer_sgl <= not(start_buffer_sgl);
-- else
-- next_state <= arbitrate;
-- start_buffer <= not(start_buffer_sgl);
-- start_buffer_sgl <= not(start_buffer_sgl);
-- no_reply <= '1';
-- end if;
if (data_in(15 downto 0) = RN_16_in(15 downto 0)
     ) then
next_state <= acknowledged;
if(count_test = '0') then
RN_64 <= NOT(RN_64_sig);
RN_64_sig <= NOT(RN_64_sig);
count_test <= '1';
end if;
if(RN64_done_sgl = '1') then
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
else
next_state <= arbitrate;
slot <= "0000000000000000";
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00100" =>           -- when Select CMD
MemBnk_out <= membnk_sel;
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';                       -- GIVE
     SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
     storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1'; --select_valid <= '1';
else
Match_select <= '0'; --select_valid <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
     mask_length)) < ((unsigned(storedpc(4
     downto 0)) * 16)+ 32)) then
Match_select <= '1'; -- select_valid <= '1';
else
Match_select <= '0'; -- select_valid <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
```

```
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                              --EPC Memory
if( (unsigned(pointer_select) + unsigned(
     mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                              --TID Memory
if((unsigned(pointer_select) + unsigned(
     mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                              --User Memory
if((unsigned(pointer_select) + unsigned(
     mask_length)) <= "00001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
     pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>             -- Session S0
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
     -- set target = A
else
session_inventory_flag(0) <= '1';
     -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
     -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
                -- Non-matching
session_inventory_flag(0) <= '1';
     -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
     session_inventory_flag(0));          --
     change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
     -- set target = B
else
session_inventory_flag(0) <= '0';
     -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
     -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
     -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
     session_inventory_flag(0));          --
     change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>             -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
     -- set target = A
else
session_inventory_flag(1) <= '1';
     -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
     -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
                -- Non-matching
session_inventory_flag(1) <= '1';
     -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
     session_inventory_flag(1));          --
     change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
     -- set target = B
else
session_inventory_flag(1) <= '0';
     -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
     -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
     -- set target = A
```

```
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
     session_inventory_flag(1));          --
     change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
     -- set target = A
else
session_inventory_flag(2) <= '1';
     -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
     -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
          -- Non-matching
session_inventory_flag(2) <= '1';
     -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
     session_inventory_flag(2));          --
     change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
     -- set target = B
else
session_inventory_flag(2) <= '0';
     -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
     -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
     -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
     session_inventory_flag(2));          --
     change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
     -- set target = A
else
session_inventory_flag(3) <= '1';
     -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
     -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
          -- Non-matching
session_inventory_flag(3) <= '1';
     -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
     session_inventory_flag(3));          --
     change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
     -- set target = B
else
session_inventory_flag(3) <= '0';
     -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
     -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
     -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
     session_inventory_flag(3));          --
     change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>           -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
          -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);          --
     change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
SL_select <= "11";              -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";              -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);          --
     change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
elsif(select_valid = '0') then
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
```

```
when others =>                                  -- All
     other COMMANDS go to Arbitrate state
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when acknowledged=>
case wht_cmd is
when "00010" =>          -- when Query CMD
case data_in(11 downto 10) is
when "00" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(0) <= not(
     session_inventory_flag(0));
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "01" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(1) <= not(
     session_inventory_flag(1));
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "10" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(2) <= not(
     session_inventory_flag(2));
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "11" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(3) <= not(
     session_inventory_flag(3));
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when others =>
session_inventory_flag <= session_inventory_flag
     ;
end case;
if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
     (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;

when "00000"      =>              -- when QueryRep
     CMD
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
     then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
     session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
     session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
     session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
     session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00011" =>                  -- when
     QueryAdjust
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
     then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
     session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
     session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
     session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
     session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
```

```
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
when "00001" =>          -- when ACK CMD
MemBnk_out <= "01";
if (data_in(15 downto 0) = RN_16_in(15 downto 0)
    ) then
next_state <= acknowledged;
truncate <= trunc_select;
if(trunc_select = '0')then
start_address_out <= "00000000000000000000010000";
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
elsif(trunc_select = '1')then
start_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length) +
    1);
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
end if;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
when "00111" =>                    -- when REQ_RN
    Command
if(count_test = '0') then
count_test <= '1';
if (data_in(31 downto 16) = RN_16_in(15 downto
    0) ) then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
if(AccPass_MSB = "0000000000000000" and
    AccPass_LSB = "0000000000000000") then
next_state <= openstate;
else
next_state <= secured;
end if;
else
next_state <= acknowledged;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;

when "00100" =>          -- when Select CMD
MemBnk_out <= membnk_sel;
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';                -- GIVE
    SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1'; --select_valid <= '1';
else
Match_select <= '0'; --select_valid <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
    mask_length)) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) then
Match_select <= '1'; -- select_valid <= '1';
else
Match_select <= '0'; -- select_valid <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                          --EPC Memory
if( (unsigned(pointer_select) + unsigned(
    mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                          --TID Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                          --User Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>             -- Session S0
case action_sel is
when "000" =>
```

```
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
else
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
else
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>           -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
else
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
else
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
else
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
else
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
else
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
```

```
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
else
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>           -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);          --
    change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
SL_select <= "11";              -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";              -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
elsif(select_valid = '0') then
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when others =>                    -- when other
    Commands
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when openstate=>
case wht_cmd is
when "00010" =>          -- when Query CMD
Req_RN_flag <= '0';
if(kill_count = '1') then
PrevQuery_kill <= '1';
end if;

case data_in(11 downto 10) is
when "00" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "01" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "10" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "11" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
```

```
Inventory_match <= '0';
end if;
end if;
when others =>
session_inventory_flag <= session_inventory_flag
    ;
end case;
if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
    (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;

when "00000"      =>              -- when QueryRep
    CMD
Req_RN_flag <= '0';
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
    then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00011" =>                   -- when
    QueryAdjust
Req_RN_flag <= '0';
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
    then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00001" =>          -- when ACK CMD
MemBnk_out <= "01";
Req_RN_flag <= '0';
if (data_in(15 downto 0) = RN_16_in(15 downto 0)
    ) then
next_state <= openstate;
truncate <= trunc_select;
if(trunc_select = '0')then
start_address_out <= "00000000000000000000010000";
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
elsif(trunc_select = '1')then
start_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length) +
    1);
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
end if;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00110" =>                   -- when NAK CMD
Req_RN_flag <= '0';
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';

when "00111" =>                   -- when REQ_RN
    Command
next_state <= openstate;
Req_RN_flag <= '1';
if(count_test = '0') then
count_test <= '1';
if (data_in(31 downto 16) = RN_16_in(15 downto
    0) ) then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
else
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;

when "00100" =>          -- when Select CMD
Req_RN_flag <= '0';
MemBnk_out <= membnk_sel;
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
```

```
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';                    -- GIVE
    SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1'; --select_valid <= '1';
else
Match_select <= '0'; --select_valid <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
    mask_length)) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) then
Match_select <= '1'; -- select_valid <= '1';
else
Match_select <= '0'; -- select_valid <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                          --EPC Memory
if( (unsigned(pointer_select) + unsigned(
    mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                          --TID Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                          --User Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>              -- Session S0
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
else
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));                --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
else
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));                --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>              -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
else
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
```

```
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
else
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
else
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
else
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
else
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
else
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>           -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);          --
    change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
```

```
SL_select <= "11";                -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";                -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";                -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);              --
    change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
elsif(select_valid = '0') then
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "01000" =>           -- when Read CMD
Req_RN_flag <= '0';
next_state <= openstate;
MemBnk_out <=  RdMemBnk;
OpenSecureMemRef <= '0';
    -- Input to the Response(or)Memory Module.
if(ebv_read = "00") then
RdMemBnk <= data_in(49 downto 48);
RdPntr(7 downto 0) <= data_in(47 downto 40);
elsif(ebv_read = "01") then
RdMemBnk <= data_in(57 downto 56);
RdPntr(15 downto 0) <= data_in(55 downto 40);
elsif(ebv_read = "10") then
RdMemBnk <= data_in(65 downto 64);
RdPntr <= data_in(63 downto 40);
end if;
RdWrdCnt <= data_in(39 downto 32);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
case RdMemBnk is
when "00" =>                          --Reserved
    Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00111111") then           -- max 3Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "10" =>                          --TID Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00011111") then           -- max 1Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "11" =>                          --User Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00001111") then           -- max 0Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "01" =>                          --EPC Memory
if(RdWrdCnt = "00000000") then
if((unsigned(RdPntr) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32))) then
start_address_out <= RdPntr;
end_address_out <= std_logic_vector("
    000000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
end if;
if( (unsigned(RdPntr) >= ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) and (unsigned(RdPntr
    ) <= "001000001111") ) then
start_address_out <= RdPntr;
end_address_out <= "000000000000001000001111";
end if;
if(unsigned(RdPntr) > "001000001111") then
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
else
if( (unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    001000001111") then       -- max 20Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
end if;
when others =>
next_state <= current_state;
end case;
end if;
when "01001" =>           -- when Write CMD
if(Req_RN_flag = '1') then
Req_RN_flag <= '0';
next_state <= openstate;
MemBnk_out <=  WrMemBnk;
WrData_out <= WrData;
OpenSecureMemRef <= '0';
    -- Input to the Response(or)Memory Module.

if(ebv_write = "00") then
WrMemBnk <= data_in(57 downto 56);
WrPntr(7 downto 0) <= data_in(55 downto 48);
elsif(ebv_write = "01") then
WrMemBnk <= data_in(65 downto 64);
WrPntr(15 downto 0) <= data_in(63 downto 48);
elsif(ebv_write = "10") then
WrMemBnk <= data_in(73 downto 72);
WrPntr <= data_in(71 downto 48);
end if;
WrData <= RN_16_in(15 downto 0) xor data_in(47
    downto 32);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then

if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
case WrMemBnk is
when "00" =>                          --Reserved
    Memory
if((unsigned(WrPntr)) <= "00111111") then
            -- max 3Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "10" =>                          --TID Memory
if((unsigned(WrPntr)) <= "00011111") then
            -- max 1Fh
```

```
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "11" =>                          --User Memory
if((unsigned(WrPntr)) <= "00001111") then
              -- max 0Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "01" =>                          --EPC Memory
if( (unsigned(WrPntr)) <= "001000001111") then
    -- max 20Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when others =>
next_state <= current_state;
end case;
end if;
else
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "01010" =>                       -- when Kill CMD
if(Req_RN_flag = '1') then
Req_RN_flag <= '0';
if(KillPass_MSB = "0000000000000000" and
    KillPass_LSB = "0000000000000000") then
                    -- Tag has zero kill
    password
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
next_state <= openstate;
else

                    -- Tag has nonzero kill
    password
if(kill_count = '1') then
    -- That is the kill has occured twice.
KillPassReader_LSB <=  data_in(50 downto 35) XOR
     data_in(31 downto 16);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then          -- valid handle
if( (KillPass_MSB = KillPassReader_MSB) and (
    KillPass_LSB = KillPassReader_LSB)) then
next_state <= killed;
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
else

                                -- invalid handle
next_state <= openstate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
else                                  -- kill
    has occured only once.
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then          -- valid handle
KillPassReader_MSB <=  data_in(50 downto 35) XOR
     data_in(31 downto 16);

if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
kill_count <= '1';
else
next_state <= openstate;
                                                --
    invalid handle
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(PrevQuery_kill = '1') then
next_state <= openstate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "10000" =>                       --
    Challenge
next_state <= current_state;
challenge_data <= data_in(63 downto 0);
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);

when others=>
Req_RN_flag <= '0';
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when secured=>
case wht_cmd is
when "00010" =>          -- when Query CMD
Req_RN_flag <= '0';
if(kill_count = '1') then
PrevQuery_kill <= '1';
end if;

case data_in(11 downto 10) is
when "00" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(0) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "01" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
if (session_inventory_flag(1) = data_in(9)) then
```

```
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(1) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "10" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(2) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when "11" =>
if(Session = data_in(11 downto 10)) then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
else
if (session_inventory_flag(3) = data_in(9)) then
Inventory_match <= '1';
else
Inventory_match <= '0';
end if;
end if;
when others =>
session_inventory_flag <= session_inventory_flag
    ;
end case;
if (Inventory_match = '1') then
if ((SL_select = "00") or (SL_select = "01") or
    (SL_select = data_in(13 downto 12) ) ) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
Q_value <= unsigned(data_in(8 downto 5));
Query_complete <= '1';
if(slot_done_sgl = '1') then
slot <= unsigned(slot_count);
if(slot_count = "0000000000000000") then
slot_done_sgl <= '0';
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
next_state <= reply;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(Inventory_match = '0') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;


when "00000"      =>              -- when QueryRep
    CMD
Req_RN_flag <= '0';
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
    then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00011" =>                    -- when
    QueryAdjust
Req_RN_flag <= '0';
if(Session = data_in(1 downto 0)) then
if ((kill_count = '1') or (access_count = '1'))
    then
kill_count <= '0';
access_count <= '0';
else
case data_in(1 downto 0) is
when "00" =>
session_inventory_flag(0) <= not(
    session_inventory_flag(0));
when "01" =>
session_inventory_flag(1) <= not(
    session_inventory_flag(1));
when "10" =>
session_inventory_flag(2) <= not(
    session_inventory_flag(2));
when "11" =>
session_inventory_flag(3) <= not(
    session_inventory_flag(3));
when others =>
next_state<= current_state;
end case;
end if;
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "00001" =>          -- when ACK CMD
MemBnk_out <= "01";
Req_RN_flag <= '0';
if (data_in(15 downto 0) = RN_16_in(15 downto 0)
    ) then
next_state <= secured;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
truncate <= trunc_select;
if(trunc_select = '0')then
start_address_out <= "00000000000000000000010000";
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
elsif(trunc_select = '1')then
start_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length) +
    1);
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
end if;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
```

```
end if;

when "00110" =>                    -- when NAK CMD
Req_RN_flag <= '0';
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';

when "00111" =>                    -- when REQ_RN
    Command
next_state <= secured;
Req_RN_flag <= '1';
if(count_test = '0') then
count_test <= '1';
if (data_in(31 downto 16) = RN_16_in(15 downto
    0) ) then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
else
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;

when "00100" =>          -- when Select CMD
MemBnk_out <= membnk_sel;
Req_RN_flag <= '0';
trunc_select <= data_in(16);
if (trunc_select='1') then
if(membnk_sel = "01") then
if(target_sel = "100") then
select_valid <= '1';
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';                -- GIVE
    SELECT MATCH MODULE THE CONTROL
if(mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
if( mask_length = "00000000") then
if((unsigned(pointer_select) < ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32))) then
Match_select <= '1';
else
Match_select <= '0';
end if;
else
if((unsigned(pointer_select) + unsigned(
    mask_length)) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) then
Match_select <= '1';
else
Match_select <= '0';
end if;
end if;
else
start_match <= '0';      Match_select <= '0';
end if;
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '0';
end if;
else
select_valid <= '1';
case membnk_sel is
when "01" =>                       --EPC Memory
if( (unsigned(pointer_select) + unsigned(
    mask_length)) <= "001000001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "10" =>                       --TID Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00011111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when "11" =>                       --User Memory
if((unsigned(pointer_select) + unsigned(
    mask_length)) <= "00001111") then
start_address_out <= pointer_select;
end_address_out <= std_logic_vector(unsigned(
    pointer_select) + unsigned(mask_length));
start_match <= '1';
if( mask_done_sgl = '1') then
mask_done_sgl <= '0';
if(match_sel = '1') then
Match_select <= '1';
start_match <= '0';
else
start_match <= '0'; Match_select <= '0';
end if;
end if;
else
Match_select <= '0';
end if;
when others =>
next_state <= current_state;
end case;
end if;

if(select_valid = '1') then
next_state <= ready;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
case target_sel is
when "000" =>              -- Session S0
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
else
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
          -- Non-matching
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));            --
    change
end if;
when "100" =>
```

```
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
else
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(0) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(0) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(0) <= not(
    session_inventory_flag(0));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "001" =>           -- Session S1
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
else
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
else
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(1) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(1) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(1) <= not(
    session_inventory_flag(1));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "010" =>           -- Session S2
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
else
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
else
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(2) <= '1';
    -- set target = B
end if;
when "110" =>
if (Match_select = '0') then
session_inventory_flag(2) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(2) <= not(
    session_inventory_flag(2));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "011" =>           -- Session S3
case action_sel is
when "000" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
else
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "001" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
when "011" =>
if (Match_select = '1') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when "100" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
else
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "101" =>
if (Match_select = '1') then
session_inventory_flag(3) <= '1';
    -- set target = B
end if;
```

```
when "110" =>
if (Match_select = '0') then
session_inventory_flag(3) <= '0';
    -- set target = A
end if;
when "111" =>
if (Match_select = '0') then
session_inventory_flag(3) <= not(
    session_inventory_flag(3));          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when "100" =>           -- SL
case action_sel is
when "000" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
else
SL_select <= "10";              -- deassert
end if;
when "001" =>
if (Match_select = '1') then
SL_select <= "11";              -- assert
end if;
when "010" =>
if (Match_select = '0') then
            -- Non-matching
SL_select <= "10";              -- deassert
end if;
when "011" =>
if (Match_select = '1') then
SL_select <= not(SL_select);          --
    change
end if;
when "100" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
else
SL_select <= "11";              -- assert
end if;
when "101" =>
if (Match_select = '1') then
SL_select <= "10";              -- deassert
end if;
when "110" =>
if (Match_select = '0') then
SL_select <= "11";              -- assert
end if;
when "111" =>
if (Match_select = '0') then
SL_select <= not(SL_select);          --
    change
end if;
when others =>
next_state <= current_state;
end case;
when others =>
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;
else
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "01000" =>         -- when Read CMD
Req_RN_flag <= '0';
next_state <= secured;
MemBnk_out <=  RdMemBnk;
OpenSecureMemRef <= '1';
    -- Input to the Response(or)Memory Module.
if(ebv_read = "00") then
RdMemBnk <= data_in(49 downto 48);
RdPntr(7 downto 0) <= data_in(47 downto 40);
elsif(ebv_read = "01") then
RdMemBnk <= data_in(57 downto 56);
RdPntr(15 downto 0) <= data_in(55 downto 40);
elsif(ebv_read = "10") then
RdMemBnk <= data_in(65 downto 64);
RdPntr <= data_in(63 downto 40);
end if;
RdWrdCnt <= data_in(39 downto 32);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
case RdMemBnk is
when "00" =>                        --Reserved
    Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00111111") then          -- max 3Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "10" =>                        --TID Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00011111") then          -- max 1Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "11" =>                        --User Memory
if((unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    00001111") then          -- max 0Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "01" =>                        --EPC Memory
if(RdWrdCnt = "00000000") then

if((unsigned(RdPntr) < ((unsigned(storedpc(4
    downto 0)) * 16)+ 32))) then
start_address_out <= RdPntr;
end_address_out <= std_logic_vector("
    00000000000000000000000" + ((unsigned(
    storedpc(4 downto 0)) * 16)+ 32));
end if;
if( (unsigned(RdPntr) >= ((unsigned(storedpc(4
    downto 0)) * 16)+ 32)) and (unsigned(RdPntr
    ) <= "001000001111") ) then
start_address_out <= RdPntr;
end_address_out <= "000000000000001000001111";
end if;
if(unsigned(RdPntr) > "001000001111") then
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
else
if( (unsigned(RdPntr) + unsigned(RdWrdCnt)) <= "
    001000001111") then      -- max 20Fh
start_address_out <= RdPntr;
end_address_out <= std_logic_vector(unsigned(
    RdPntr) + unsigned(RdWrdCnt));
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
end if;
when others =>
next_state <= current_state;
end case;
end if;
when "01001" =>          -- when Write CMD
if(Req_RN_flag = '1') then
Req_RN_flag <= '0';
next_state <= secured;
MemBnk_out <=  WrMemBnk;
WrData_out <= WrData;
```

```
OpenSecureMemRef <= '1';
    -- Input to the Response(or)Memory Module.
if(ebv_write = "00") then
WrMemBnk <= data_in(57 downto 56);
WrPntr(7 downto 0) <= data_in(55 downto 48);
elsif(ebv_write = "01") then
WrMemBnk <= data_in(65 downto 64);
WrPntr(15 downto 0) <= data_in(63 downto 48);
elsif(ebv_write = "10") then
WrMemBnk <= data_in(73 downto 72);
WrPntr <= data_in(71 downto 48);
end if;
WrData <= RN_16_in(15 downto 0) xor data_in(47
    downto 32);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then

if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
case WrMemBnk is
when "00" =>                          --Reserved
    Memory
if((unsigned(WrPntr)) <= "00111111") then
            -- max 3Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "10" =>                          --TID Memory
if((unsigned(WrPntr)) <= "00011111") then
            -- max 1Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "11" =>                          --User Memory
if((unsigned(WrPntr)) <= "00001111") then
            -- max 0Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when "01" =>                          --EPC Memory
if( (unsigned(WrPntr)) <= "001000001111") then
    -- max 20Fh
start_address_out <= WrPntr;
else
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
end if;
when others =>
next_state <= current_state;
end case;
end if;
else
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "01010" =>                       -- when Kill CMD
if(Req_RN_flag = '1') then
Req_RN_flag <= '0';
if(KillPass_MSB = "0000000000000000" and
    KillPass_LSB = "0000000000000000") then
                    -- Tag has zero kill
    password
error_code <= not(error_code_sgl);
error_code_sgl <= not(error_code_sgl);
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
next_state <= secured;
else

                    -- Tag has nonzero kill
    password
if(kill_count = '1') then
    -- That is the kill has occured twice.
KillPassReader_LSB <=  data_in(50 downto 35) XOR
     data_in(31 downto 16);
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then          -- valid handle
if( (KillPass_MSB = KillPassReader_MSB) and (
    KillPass_LSB = KillPassReader_LSB)) then
next_state <= killed;
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
else

                                -- invalid handle
next_state <= secured;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
else                                  -- kill
    has occured only once.
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then          -- valid handle
KillPassReader_MSB <=  data_in(50 downto 35) XOR
     data_in(31 downto 16);
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
kill_count <= '1';
else
next_state <= secured;
                                      -- invalid
    handle
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;
end if;
end if;
elsif(PrevQuery_kill = '1') then
next_state <= secured;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
else
next_state <= arbitrate;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "01011" =>                   -- when Lock
    command
Req_RN_flag <= '0';
if (data_in(31 downto 16) = RN_16_in(15 downto
    0)) then          -- valid handle
next_state <= secured;
```

```
if(count_test = '0') then
RN_16 <= NOT(RN_16_sig);
RN_16_sig <= NOT(RN_16_sig);
count_test <= '1';
end if;
if(RN16_done_sgl = '1') then
RN16_done_sgl <= '0';
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
end if;
-- data_in(51 downto 32) - 20 bits Payload
if(data_in(51) = '1') then
Lock_Kill(1) <= data_in(41);
end if;
if(data_in(50) = '1') then
Lock_Kill(0) <= data_in(40);
end if;
if(data_in(49) = '1') then
Lock_Access(1) <= data_in(39);
end if;
if(data_in(48) = '1') then
Lock_Access(0) <= data_in(38);
end if;
if(data_in(47) = '1') then
Lock_EPC(1) <= data_in(37);
end if;
if(data_in(46) = '1') then
Lock_EPC(0) <= data_in(36);
end if;
if(data_in(45) = '1') then
Lock_TID(1) <= data_in(35);
end if;
if(data_in(44) = '1') then
Lock_TID(0) <= data_in(34);
end if;
if(data_in(43) = '1') then
Lock_User(1) <= data_in(33);
end if;
if(data_in(42) = '1') then
Lock_User(0) <= data_in(32);
end if;
else
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end if;

when "10000" =>                          --
     Challenge
next_state <= current_state;
challenge_data <= data_in(63 downto 0);
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);

when others=>
Req_RN_flag <= '0';
next_state <= current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
end case;

when killed =>
next_state<=killed;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';
when others =>
next_state<=current_state;
start_buffer <= not(start_buffer_sgl);
start_buffer_sgl <= not(start_buffer_sgl);
no_reply <= '1';

end case;
End if;
END IF;
end if;
if(Reserved_done'event) then
Reserved_done_sgl <= '1';
end if;
if(mask_done'event) then
mask_done_sgl <= '1';
end if;
if(RN16_done'event) then
RN16_done_sgl <= '1';
end if;
if(RN64_done'event) then
RN64_done_sgl <= '1';
end if;
if(slot_done'event) then
slot_done_sgl <= '1';
end if;
if(done'event) then
count_test <= '0';
Inventory_match <= 'U';
select_valid <= 'U';
current_state <= next_state;
no_reply <= '0';
end if;
end process;
end RTL_FSM;
```

Listing A.9. Random Number Generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity RN_16 is

port (  CLK      : in STD_LOGIC;
preset   : in STD_LOGIC;
RN16_flag    : in STD_LOGIC;
RN64_flag    : in STD_LOGIC;
lfsr_out : out STD_LOGIC_VECTOR(15 downto 0);
slot_15 : out STD_LOGIC_VECTOR(15 downto 0);
RN16_done,RN64_done : out std_logic;
RN_64_data       : out STD_LOGIC_VECTOR(63 downto
     0)
);

end RN_16;
architecture RTL_48 of RN_16 is

signal  d    : STD_LOGIC_VECTOR(63 downto 0);
signal  q    : STD_LOGIC_VECTOR(63 downto 0);
signal  I        : STD_LOGIC_VECTOR(15 downto 0)
    := "0111010101001101";
signal  RN64_flag_sgl,RN16_flag_sgl,VI,
     RN16_done_sgl,RN64_done_sgl       :
     STD_LOGIC;

begin

process(CLK,RN16_flag,RN64_flag)
begin
if (preset = '0') then
RN64_flag_sgl <= '0';
RN16_flag_sgl <= '0';
RN16_done_sgl <= '0';
RN64_done_sgl <= '0';
VI <= '1';
elsif (CLK'event and CLK = '1') then
VI <= '0';
if (RN64_flag_sgl = '1') then
slot_15 <= q(15 downto 0);
RN64_flag_sgl <= '0';
RN64_done <= not(RN64_done_sgl);
RN64_done_sgl <= not(RN64_done_sgl);
RN_64_data <= q;
q <= d;
elsif(RN16_flag_sgl = '1') then
RN16_flag_sgl <= '0';
RN16_done <= not(RN16_done_sgl);
RN16_done_sgl <= not(RN16_done_sgl);
lfsr_out <= q(15 downto 0);
slot_15 <= q(15 downto 0);
q <= d;
else
q <= d;
end if;
end if;
if(RN16_flag'event) then
RN16_flag_sgl <= '1';
end if;
if(RN64_flag'event) then
RN64_flag_sgl <= '1';
end if;
end process reg;
```

```
_____________
d(0)  <= I(0)  when VI = '1' else q(1);
d(1)  <= I(1)  when VI = '1' else q(2);
d(2)  <= I(2)  when VI = '1' else q(0) xor q(3);
d(3)  <= I(3)  when VI = '1' else q(4);
d(4)  <= I(4)  when VI = '1' else q(5);
d(5)  <= I(5)  when VI = '1' else q(6);
d(6)  <= I(6)  when VI = '1' else q(0) xor q(7);
d(7)  <= I(7)  when VI = '1' else q(8);
d(8)  <= I(8)  when VI = '1' else q(9);
d(9)  <= I(9)  when VI = '1' else q(0) xor q(10)
    ;
d(10) <= I(10) when VI = '1' else q(11);
d(11) <= I(11) when VI = '1' else q(0) xor q(12)
    ;
d(12) <=  '1'  when VI = '1' else q(13);
d(13) <= I(13) when VI = '1' else q(14);
d(14) <= I(14) when VI = '1' else q(0) xor q(15)
    ;
d(15) <= I(15) when VI = '1' else q(0);

d(16)  <= I(0) xor I(15)        when VI = '1'
    else q(15);
d(17)  <= I(1)                          when VI
    = '1' else q(16);
d(18)  <= I(2) xor I(11)        when VI = '1'
    else q(0) xor q(17);
d(19)  <= I(3)                          when VI
    = '1' else q(18);
d(20)  <= I(4) xor I(9)         when VI = '1'
    else q(19);
d(21)  <= I(5)                          when VI
    = '1' else q(20);
d(22)  <= I(6) xor I(13)        when VI = '1'
    else q(0) xor q(21);
d(23)  <= I(7)                          when VI
    = '1' else q(22);
d(24)  <= I(8) xor I(7)         when VI = '1'
    else q(23);
d(25)  <= I(9)                          when VI
    = '1' else q(0) xor q(24);
d(26) <= I(10)  xor  I(5)       when VI = '1'
    else q(25);
d(27) <= I(11)                          when VI
    = '1' else q(0) xor q(26);
d(28) <=  '1'   xor      I(3)   when VI = '1'
    else q(27);
d(29) <= I(13)                          when VI
    = '1' else q(28);
d(30) <= I(14) xor I(1)         when VI = '1'
    else q(0) xor q(29);
d(31) <= I(15)                          when VI
    = '1' else q(30);

d(32)  <= I(0)                          when VI
    = '1' else q(31);
d(33)  <= I(1) xor I(14)        when VI = '1'
    else q(32);
d(34)  <= I(2)                          when VI
    = '1' else q(0) xor q(33);
d(35)  <= I(3) xor I(12)        when VI = '1'
    else q(34);
d(36)  <= I(4)                          when VI
    = '1' else q(35);
d(37)  <= I(5) xor I(10)        when VI = '1'
    else q(36);
d(38)  <= I(6)                          when VI
    = '1' else q(0) xor q(37);
d(39)  <= I(7) xor I(9)         when VI = '1'
    else q(38);
d(40)  <= I(8)                          when VI
    = '1' else q(39);
d(41)  <= I(9) xor I(7)         when VI = '1'
    else q(0) xor q(40);
d(42) <= I(10)                          when VI
    = '1' else q(41);
d(43) <= I(11) xor I(5)         when VI = '1'
    else q(0) xor q(42);
d(44) <=  '1'                           when VI
    = '1' else q(43);
d(45) <= I(13) xor I(3)         when VI = '1'
    else q(44);
d(46) <= I(14)                          when VI
    = '1' else q(0) xor q(45);
d(47) <= I(15) xor I(2)         when VI = '1'
    else q(46);
d(48)  <= I(0)                          when VI
    = '1' else q(47);
d(49)  <= I(1) xor I(14)        when VI = '1'
    else q(48);
d(50)  <= I(2)                          when VI
    = '1' else q(0) xor q(49);
d(51)  <= I(3) xor I(12)        when VI = '1'
    else q(50);
d(52)  <= I(4)                          when VI
    = '1' else q(51);
d(53)  <= I(5) xor I(10)        when VI = '1'
    else q(52);
d(54)  <= I(6) xor I(8)         when VI = '1'
    else q(0) xor q(53);
d(55)  <= I(7) xor I(9)         when VI = '1'
    else q(54);
d(56)  <= I(8)                          when VI
    = '1' else q(55);
d(57)  <= I(9) xor I(7)         when VI = '1'
    else q(0) xor q(56);
d(58) <= I(10)                          when VI
    = '1' else q(57);
d(59) <= I(11) xor I(5)         when VI = '1'
    else q(0) xor q(58);
d(60) <=  '1'  xor I(4)         when VI = '1'
    else q(59);
d(61) <= I(13) xor I(3)         when VI = '1'
    else q(60);
d(62) <= I(14) xor I(1)         when VI = '1'
    else q(0) xor q(61);
d(63) <= I(15)                          when VI
    = '1' else q(62);


end RTL_48;
```

Listing A.10. Memory Module - EPC Rom

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

entity EPC_ROM is
port (
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(11 downto 0)
    ;
end_address : in std_logic_vector(11 downto 0);
mem_bank : in std_logic_vector(1 downto 0);
lock_value : in std_logic_vector(1 downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
    std_logic
);
end entity EPC_ROM;

architecture EPC_behavioral of EPC_ROM is
type mem is array (0 to 527) of std_logic;
signal mem_lock_status : std_logic_vector(1
    downto 0);
signal rd_wr_lock_done_sig, wr_done_sig,
    stop_sgl,error_sgl,rd_en_flag: std_logic;
signal count_bit, mask_len_sig: unsigned(7
    downto 0);
signal my_Rom : mem := (
0  => '0',1  => '1',2 => '1', 3 => '1',4 => '0',
    5 => '0', 6 => '1', 8  => '0',
7  => '0',9  => '0',10 => '1', 11  => '0', 12 =>
     '0',13 => '1',14 => '1',15 => '1',
16 => '0',17 => '0',18 => '0', 19  => '0', 20 =>
     '1',21 => '1',22 => '1',23 => '0',
24 => '0',25 => '1',26 => '1', 27  => '0', 28 =>
     '0',29 => '1',30 => '1',31 => '0',
32 => '1',33 => '0',34 => '0', 35  => '1', 36 =>
     '0',37 => '0',38 => '1',39 => '0',
```

```
40 => '0',41 => '0',42 => '0', 43 => '0', 44 =>
      '0',45 => '0',46 => '0',47 => '1',
48 => '0',49 => '0',50 => '0', 51 => '0', 52 =>
      '0',53 => '0',54 => '0',55 => '0',
56 => '0',57 => '0',58 => '0', 59 => '0', 60 =>
      '0',61 => '0',62 => '0',63 => '0',
64 => '0',65 => '0',66 => '0', 67 => '0', 68 =>
      '0',69 => '0',70 => '0',71 => '0',
72 => '0',73 => '0',74 => '0', 75 => '0', 76 =>
      '0',77 => '0',78 => '0',79 => '0',
80 => '0',81 => '0',82 => '0',83 => '0',84 =>
     '0',85 => '0',86 => '0',87 => '0',
88 => '0',89 => '0',90 => '0',91 => '0',92 =>
     '0',93 => '0',94 => '0',95 => '0',
96 => '0',97 => '0',98 => '0',99 => '0',100 =>
     '0',101 => '0',102 => '0',103 => '0',
104 => '0',105 => '0',106 => '0',107 => '0',108
     => '0',109 => '0',110 => '0',111 => '0',
112 => '0',113 => '0',114 => '0',115 => '0',116
     => '0',117 => '0',118 => '0',119 => '0',
120 => '0',121 => '0',122 => '0',123 => '0',124
     => '0',125 => '0',126 => '0',127 => '0',
128 => '0',129 => '0',130 => '0',131 => '0',132
     => '0',133 => '0',134 => '0',135 => '0',
136 => '0',137 => '0',138 => '0',139 => '0',140
     => '0',141 => '0',142 => '0',143 => '0',
144 => '0',145 => '0',146 => '0',147 => '0',148
     => '0',149 => '0',150 => '0',151 => '0',
152 => '0',153 => '0',154 => '0',155 => '0',156
     => '0',157 => '0',158 => '0',159 => '0',
160 => '0',161 => '0',162 => '0',163 => '0',164
     => '0',165 => '0',166 => '0',167 => '0',
168 => '0',169 => '0',170 => '0',171 => '0',172
     => '0',173 => '0',174 => '0',175 => '0',
176 => '0',177 => '0',178 => '0',179 => '0',180
     => '0',181 => '0',182 => '0',183 => '0',
184 => '0',185 => '0',186 => '0',187 => '0',188
     => '0',189 => '0',190 => '0',191 => '0',
192 => '0',193 => '0',194 => '0',195 => '0',196
     => '0',197 => '0',198 => '0',199 => '0',
200 => '0',201 => '0',202 => '0',203 => '0',204
     => '0',205 => '0',206 => '0',207 => '0',
208 => '0',209 => '0',210 => '0',211 => '0',212
     => '0',213 => '0',214 => '0',215 => '0',
216 => '0',217 => '0',218 => '0',219 => '0',220
     => '0',221 => '0',222 => '0',223 => '0',
224 => '0',225 => '0',226 => '0',227 => '0',228
     => '0',229 => '0',230 => '0',231 => '0',
232 => '0',233 => '0',234 => '0',235 => '0',236
     => '0',237 => '0',238 => '0',239 => '0',
240 => '0',241 => '0',242 => '0',243 => '0',244
     => '0',245 => '0',246 => '0',247 => '0',
248 => '0',249 => '0',250 => '0',251 => '0',252
     => '0',253 => '0',254 => '0',255 => '0',
256 => '0',257 => '0',258 => '0',259 => '0',260
     => '0',261 => '0',262 => '0',263 => '0',
264 => '0',265 => '0',266 => '0',267 => '0',268
     => '0',269 => '0',270 => '0',271 => '0',
272 => '0',273 => '0',274 => '0',275 => '0',276
     => '0',277 => '0',278 => '0',279 => '0',
280 => '0',281 => '0',282 => '0',283 => '0',284
     => '0',285 => '0',286 => '0',287 => '0',
288 => '0',289 => '0',290 => '0',291 => '0',292
     => '0',293 => '0',294 => '0',295 => '0',
296 => '0',297 => '0',298 => '0',299 => '0',300
     => '0',301 => '0',302 => '0',303 => '0',
304 => '0',305 => '0',306 => '0',307 => '0',308
     => '0',309 => '0',310 => '0',311 => '0',
312 => '0',313 => '0',314 => '0',315 => '0',316
     => '0',317 => '0',318 => '0',319 => '0',
320 => '0',321 => '0',322 => '0',323 => '0',324
     => '0',325 => '0',326 => '0',327 => '0',
328 => '0',329 => '0',330 => '0',331 => '0',332
     => '0',333 => '0',334 => '0',335 => '0',
336 => '0',337 => '0',338 => '0',339 => '0',340
     => '0',341 => '0',342 => '0',343 => '0',
344 => '0',345 => '0',346 => '0',347 => '0',348
     => '0',349 => '0',350 => '0',351 => '0',
352 => '0',353 => '0',354 => '0',355 => '0',356
     => '0',357 => '0',358 => '0',359 => '0',
360 => '0',361 => '0',362 => '0',363 => '0',364
     => '0',365 => '0',366 => '0',367 => '0',
368 => '0',369 => '0',370 => '0',371 => '0',372
     => '0',373 => '0',374 => '0',375 => '0',
376 => '0',377 => '0',378 => '0',379 => '0',380
     => '0',381 => '0',382 => '0',383 => '0',
384 => '0',385 => '0',386 => '0',387 => '0',388
     => '0',389 => '0',390 => '0',391 => '0',
392 => '0',393 => '0',394 => '0',395 => '0',396
     => '0',397 => '0',398 => '0',399 => '0',
400 => '0',401 => '0',402 => '0',403 => '0',404
     => '0',405 => '0',406 => '0',407 => '0',
408 => '0',409 => '0',410 => '0',411 => '0',412
     => '0',413 => '0',414 => '0',415 => '0',
416 => '0',417 => '0',418 => '0',419 => '0',420
     => '0',421 => '0',422 => '0',423 => '0',
424 => '0',425 => '0',426 => '0',427 => '0',428
     => '0',429 => '0',430 => '0',431 => '0',
432 => '0',433 => '0',434 => '0',435 => '0',436
     => '0',437 => '0',438 => '0',439 => '0',
440 => '0',441 => '0',442 => '0',443 => '0',444
     => '0',445 => '0',446 => '0',447 => '0',
448 => '0',449 => '0',450 => '0',451 => '0',452
     => '0',453 => '0',454 => '0',455 => '0',
456 => '0',457 => '0',458 => '0',459 => '0',460
     => '0',461 => '0',462 => '0',463 => '0',
464 => '0',465 => '0',466 => '0',467 => '0',468
     => '0',469 => '0',470 => '0',471 => '0',
472 => '0',473 => '0',474 => '0',475 => '0',476
     => '0',477 => '0',478 => '0',479 => '0',
480 => '0',481 => '0',482 => '0',483 => '0',484
     => '0',485 => '0',486 => '0',487 => '0',
488 => '0',489 => '0',490 => '0',491 => '0',492
     => '0',493 => '0',494 => '0',495 => '0',
496 => '0',497 => '0',498 => '0',499 => '0',500
     => '0',501 => '0',502 => '0',503 => '0',
504 => '0',505 => '0',506 => '0',507 => '0',508
     => '0',509 => '0',510 => '0',511 => '0',
512 => '0',513 => '0',514 => '0',515 => '0',516
     => '0',517 => '0',518 => '0',519 => '0',
520 => '0',521 => '0',522 => '0',523 => '0',524
     => '0',525 => '0',526 => '0',527 => '0',

);
begin
process (clk,rd_en)
begin
IF (Preset = '0') THEN
mem_lock_status <= "00";
count_bit <= "00000000";
rd_wr_lock_done_sig <= '0';
stop_sgl <= '0';
error_sgl <= '0';
rd_en_flag <= '0';
ELSIF (RISING_EDGE(clk)) THEN
-- ****************** EXCECUTE LOCK COMMAND
     ***************************--
if( (mem_lock_status = "00") or (mem_lock_status
     = "10") ) then
if( (lock_value = "00") or (lock_value = "01")
     or (lock_value = "10") or (lock_value = "11
     ") ) then
mem_lock_status <= lock_value;
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
end if;
else
error <= not(error_sgl);
error_sgl <= not(error_sgl);
end if;
-- ****************** EXCECUTE LOCK COMMAND
     ***************************--
if(mem_bank = "01") then
if( (data_16_bit = '1') and (rd_en_flag = '1') )
     then
data_out_16 <= my_rom(conv_integer(unsigned(
     start_address))) & my_rom(conv_integer(
     unsigned(start_address)+1)) & my_rom(
     conv_integer(unsigned(start_address)+2)) &
     my_rom(conv_integer(unsigned(start_address)
     +3)) & my_rom(conv_integer(unsigned(
     start_address)+4)) & my_rom(conv_integer(
     unsigned(start_address)+5)) & my_rom(
     conv_integer(unsigned(start_address)+6)) &
     my_rom(conv_integer(unsigned(start_address)
     +7)) & my_rom(conv_integer(unsigned(
     start_address)+8)) & my_rom(conv_integer(
     unsigned(start_address)+9)) & my_rom(
     conv_integer(unsigned(start_address)+10)) &
      my_rom(conv_integer(unsigned(start_address
     )+11)) & my_rom(conv_integer(unsigned(
```

```
      start_address)+12)) & my_rom(conv_integer(
      unsigned(start_address)+13)) & my_rom(
      conv_integer(unsigned(start_address)+14)) &
       my_rom(conv_integer(unsigned(start_address
      )+15));
rd_en_flag <= '0';
rd_wr_lock_done <=       not(rd_wr_lock_done_sig)
      ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
      ;
end if;
if( (data_16_bit = '0') and (rd_en_flag = '1')
      and (send = '1') ) then
if( (unsigned(start_address) + count_bit) <
      unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
      start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=       not(rd_wr_lock_done_sig)
      ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
      ;
elsif( (unsigned(start_address) + count_bit) =
      unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
      start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=       not(rd_wr_lock_done_sig)
      ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
      ;
stop <= not(stop_sgl);
stop_sgl <= not(stop_sgl);
else
rd_en_flag <= '0';
count_bit <= "00000000";
end if;
end if;
if(wr_en = '1') then
-- ****************** READ/WRITE
      ************************************--
if (mem_lock_status = "11") then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
elsif (mem_lock_status = "10" and
      open_state_valid = '0') then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
else
my_rom(conv_integer(unsigned(start_address))) <=
       data_in_write(0);       my_rom(conv_integer
      (unsigned(start_address)+8)) <=
      data_in_write(8);
my_rom(conv_integer(unsigned(start_address)+1))
      <= data_in_write(1);     my_rom(conv_integer
      (unsigned(start_address)+9)) <=
      data_in_write(9);
my_rom(conv_integer(unsigned(start_address)+2))
      <= data_in_write(2);     my_rom(conv_integer
      (unsigned(start_address)+10)) <=
      data_in_write(10);
my_rom(conv_integer(unsigned(start_address)+3))
      <= data_in_write(3);     my_rom(conv_integer
      (unsigned(start_address)+11)) <=
      data_in_write(11);
my_rom(conv_integer(unsigned(start_address)+4))
      <= data_in_write(4);     my_rom(conv_integer
      (unsigned(start_address)+12)) <=
      data_in_write(12);
my_rom(conv_integer(unsigned(start_address)+5))
      <= data_in_write(5);     my_rom(conv_integer
      (unsigned(start_address)+13)) <=
      data_in_write(13);
my_rom(conv_integer(unsigned(start_address)+6))
      <= data_in_write(6);     my_rom(conv_integer
      (unsigned(start_address)+14)) <=
      data_in_write(14);
my_rom(conv_integer(unsigned(start_address)+7))
      <= data_in_write(7);     my_rom(conv_integer
      (unsigned(start_address)+15)) <=
      data_in_write(15);

rd_wr_lock_done <=       not(rd_wr_lock_done_sig)
      ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
      ;
end if;
end if;
end if;
end if;
-- ****************** READ/WRITE
      ************************************--
if((rd_en'event) and (mem_bank = "01") and (
      rd_en = '1')) then
rd_en_flag <= '1';
end if;
end process;
end architecture EPC_behavioral;
```

Listing A.11. Memory Module - Reserved Rom

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

entity Reserved_ROM is
port (
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(11 downto 0)
      ;
end_address : in std_logic_vector(11 downto 0);
lock_kill_value, lock_access_value : in
      std_logic_vector(1 downto 0);
mem_bank : in std_logic_vector(1 downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
      ;
rd_en, wr_en, open_state_valid, data_16_bit : in
       std_logic;
data_out_bit, rd_wr_lock_done, error, stop: out
      std_logic
);
end entity Reserved_ROM;

architecture Reserved_behavioral of Reserved_ROM
      is
type mem is array (0 to 63) of std_logic;
signal mem_kill_lock_status,
      mem_access_lock_status : std_logic_vector(1
       downto 0);
signal rd_wr_lock_done_sig,error_sgl,stop_sgl,
      rd_en_flag: std_logic;
signal count_bit, mask_len_sig: unsigned(7
      downto 0);
signal my_Rom : mem := (
0   => '0', 1   => '1', 2   => '1', 3   => '1',
      4   => '1', 5   => '0', 6   => '0', 8   =>
      '1', 7   => '0',
9   => '1', 10  => '0', 11  => '1', 12  => '0',
      13  => '0', 14  => '0', 15  => '1', 16  =>
      '1', 17  => '1',
18  => '0', 19  => '1', 20  => '0', 21  => '0',
      22  => '1', 23  => '0', 24  => '1', 25  =>
      '0', 26  => '1',
27  => '1', 28  => '0', 29  => '1', 30  => '0',
      31  => '0', 32  => '0', 33  => '1', 34  =>
      '0', 35  => '0',
36  => '0', 37  => '0', 38  => '1', 39  => '0',
      40  => '1', 41  => '0', 42  => '0', 43  =>
      '1', 44  => '0',
45  => '1', 46  => '0', 47  => '1', 48  => '0',
      49  => '0', 50  => '1', 51  => '0', 52  =>
      '1', 53  => '0',
54  => '0', 55  => '0', 56  => '1', 57  => '0',
      58  => '0', 59  => '1', 60  => '1', 61  =>
      '0', 62  => '1',
63  => '0'
);
begin
process (clk,rd_en)
begin
IF (Preset = '0') THEN
mem_kill_lock_status <= "00";
mem_access_lock_status <= "00";
count_bit <= "00000000";
rd_wr_lock_done_sig <= '0';
stop_sgl <= '0';
```

```
error_sgl <= '0';
rd_en_flag <= '0';
ELSIF (RISING_EDGE(clk)) THEN
-- ****************** EXCECUTE LOCK COMMAND
     ***************************--
if( (mem_kill_lock_status = "00") or (
     mem_kill_lock_status = "10") ) then
if( (lock_kill_value = "00") or (lock_kill_value
     = "01") or (lock_kill_value = "10") or (
     lock_kill_value = "11") ) then
mem_kill_lock_status <= lock_kill_value;
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
end if;
else
error <= not(error_sgl);
error_sgl <= not(error_sgl);
end if;

if( (mem_access_lock_status = "00") or (
     mem_access_lock_status = "10") ) then
if( (lock_access_value = "00") or (
     lock_access_value = "01") or (
     lock_access_value = "10") or (
     lock_access_value = "11") ) then
mem_access_lock_status <= lock_access_value;
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
end if;
else
error <= not(error_sgl);
error_sgl <= not(error_sgl);
end if;

-- ****************** EXCECUTE LOCK COMMAND
     ***************************--

-- ****************** READ/WRITE
     ************************************--
if(mem_bank = "00") then
if( (data_16_bit = '1') and (rd_en_flag = '1') )
     then
data_out_16 <= my_rom(conv_integer(unsigned(
     start_address))) & my_rom(conv_integer(
     unsigned(start_address)+1)) & my_rom(
     conv_integer(unsigned(start_address)+2)) &
     my_rom(conv_integer(unsigned(start_address)
     +3)) & my_rom(conv_integer(unsigned(
     start_address)+4)) & my_rom(conv_integer(
     unsigned(start_address)+5)) & my_rom(
     conv_integer(unsigned(start_address)+6)) &
     my_rom(conv_integer(unsigned(start_address)
     +7)) & my_rom(conv_integer(unsigned(
     start_address)+8)) & my_rom(conv_integer(
     unsigned(start_address)+9)) & my_rom(
     conv_integer(unsigned(start_address)+10)) &
      my_rom(conv_integer(unsigned(start_address
     )+11)) & my_rom(conv_integer(unsigned(
     start_address)+12)) & my_rom(conv_integer(
     unsigned(start_address)+13)) & my_rom(
     conv_integer(unsigned(start_address)+14)) &
      my_rom(conv_integer(unsigned(start_address
     )+15));
rd_en_flag <= '0';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
end if;
if( (data_16_bit = '0') and (rd_en_flag = '1')
     and (send = '1') ) then
if( (unsigned(start_address) + count_bit) <
     unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
     start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
elsif( (unsigned(start_address) + count_bit) =
     unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
     start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
stop <= not(stop_sgl);
stop_sgl <= not(stop_sgl);
else
rd_en_flag <= '0';
count_bit <= "00000000";
end if;
end if;
if(wr_en = '1') then
-- ****************** READ/WRITE
     ************************************--
if ( (mem_kill_lock_status = "11") or(
     mem_access_lock_status = "11") ) then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
elsif ( (mem_kill_lock_status = "10" and
     open_state_valid = '0') or (
     mem_access_lock_status = "10" and
     open_state_valid = '0') ) then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
else
my_rom(conv_integer(unsigned(start_address))) <=
      data_in_write(0);     my_rom(conv_integer
     (unsigned(start_address)+8)) <=
     data_in_write(8);
my_rom(conv_integer(unsigned(start_address)+1))
     <= data_in_write(1);     my_rom(conv_integer
     (unsigned(start_address)+9)) <=
     data_in_write(9);
my_rom(conv_integer(unsigned(start_address)+2))
     <= data_in_write(2);     my_rom(conv_integer
     (unsigned(start_address)+10)) <=
     data_in_write(10);
my_rom(conv_integer(unsigned(start_address)+3))
     <= data_in_write(3);     my_rom(conv_integer
     (unsigned(start_address)+11)) <=
     data_in_write(11);
my_rom(conv_integer(unsigned(start_address)+4))
     <= data_in_write(4);     my_rom(conv_integer
     (unsigned(start_address)+12)) <=
     data_in_write(12);
my_rom(conv_integer(unsigned(start_address)+5))
     <= data_in_write(5);     my_rom(conv_integer
     (unsigned(start_address)+13)) <=
     data_in_write(13);
my_rom(conv_integer(unsigned(start_address)+6))
     <= data_in_write(6);     my_rom(conv_integer
     (unsigned(start_address)+14)) <=
     data_in_write(14);
my_rom(conv_integer(unsigned(start_address)+7))
     <= data_in_write(7);     my_rom(conv_integer
     (unsigned(start_address)+15)) <=
     data_in_write(15);
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
     ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
     ;
end if;
end if;
end if;
end if;
-- ****************** READ/WRITE
     ************************************--
if((rd_en'event) and (mem_bank = "00")and (rd_en
     = '1')) then
rd_en_flag <= '1';
end if;
end process;
end architecture Reserved_behavioral;
```

Listing A.12. Memory Module - TID Rom

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

```
entity TID_ROM is
port (
clk, send: in std_logic;--clock signal.
Preset: in std_logic;
start_address : in std_logic_vector(11 downto 0)
    ;
end_address : in std_logic_vector(11 downto 0);
lock_value,mem_bank : in std_logic_vector(1
    downto 0);
data_out_16 : out std_logic_vector(15 downto 0);
data_in_write : in std_logic_vector(15 downto 0)
    ;
rd_en, wr_en, open_state_valid, data_16_bit : in
     std_logic;
data_out_bit, rd_wr_lock_done, wr_done, error,
    stop : out std_logic
);
end entity TID_ROM;

architecture TID_behavioral of TID_ROM is
type mem is array (0 to 31) of std_logic;
signal mem_lock_status : std_logic_vector(1
    downto 0);
signal rd_wr_lock_done_sig, wr_done_sig,
    error_sgl,stop_sgl,rd_en_flag: std_logic;
signal count_bit, mask_len_sig: unsigned(7
    downto 0);
signal my_Rom : mem := (
0   => '0', 1   => '0', 2   => '0', 3   => '0',
    4   => '0', 5   => '0', 6   => '0', 8   =>
    '0', 7   => '0',
9   => '0', 10  => '0', 11  => '0', 12  => '0',
    13  => '0', 14  => '0', 15  => '0', 16  =>
    '0', 17  => '0',
18  => '0', 19  => '0', 20  => '0', 21  => '0',
    22  => '0', 23  => '0', 24  => '0', 25  =>
    '0', 26  => '0',
27  => '0', 28  => '0', 29  => '0', 30  => '0',
    31  => '0');
begin
process (clk,rd_en)
begin
IF (Preset = '0') THEN
mem_lock_status <= "00";
count_bit <= "00000000";
rd_wr_lock_done_sig <= '0';
error_sgl <= '0';
stop_sgl <= '0';
rd_en_flag <= '0';
ELSIF (RISING_EDGE(clk)) THEN
-- ***************** EXCECUTE LOCK COMMAND
    ***************************--
if( (mem_lock_status = "00") or (mem_lock_status
    = "10") ) then
if( (lock_value = "00") or (lock_value = "01")
    or (lock_value = "10") or (lock_value = "11
    ") ) then
mem_lock_status <= lock_value;
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
    ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
    ;
end if;
else
error <= not(error_sgl);
error_sgl <= not(error_sgl);
end if;
-- ***************** EXCECUTE LOCK COMMAND
    ***************************--

-- ***************** READ/WRITE
    ************************************--
if(mem_bank = "10") then
if( (data_16_bit = '1') and (rd_en_flag = '1') )
    then
data_out_16 <= my_rom(conv_integer(unsigned(
    start_address))) & my_rom(conv_integer(
    unsigned(start_address)+1)) & my_rom(
    conv_integer(unsigned(start_address)+2)) &
    my_rom(conv_integer(unsigned(start_address)
    +3)) & my_rom(conv_integer(unsigned(
    start_address)+4)) & my_rom(conv_integer(
    unsigned(start_address)+5)) & my_rom(
    conv_integer(unsigned(start_address)+6)) &
    my_rom(conv_integer(unsigned(start_address)
    +7)) & my_rom(conv_integer(unsigned(
    start_address)+8)) & my_rom(conv_integer(
    unsigned(start_address)+9)) & my_rom(
    conv_integer(unsigned(start_address)+10)) &
     my_rom(conv_integer(unsigned(start_address
    )+11)) & my_rom(conv_integer(unsigned(
    start_address)+12)) & my_rom(conv_integer(
    unsigned(start_address)+13)) & my_rom(
    conv_integer(unsigned(start_address)+14)) &
     my_rom(conv_integer(unsigned(start_address
    )+15));
rd_en_flag <= '0';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
    ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
    ;
end if;
if( (data_16_bit = '0') and (rd_en_flag = '1')
    and (send = '1') ) then
if( (unsigned(start_address) + count_bit) <
    unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
    start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
    ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
    ;
elsif( (unsigned(start_address) + count_bit) =
    unsigned(end_address) -1 ) then
data_out_bit <= my_rom(conv_integer(unsigned(
    start_address) + count_bit));
count_bit <= count_bit + '1';
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
    ;
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
    ;
stop <= not(stop_sgl);
stop_sgl <= not(stop_sgl);
else
rd_en_flag <= '0';
count_bit <= "00000000";
end if;
end if;
if(wr_en = '1') then
-- ***************** READ/WRITE
    ************************************--
if (mem_lock_status = "11") then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
elsif (mem_lock_status = "10" and
    open_state_valid = '0') then
error <= not(error_sgl);
error_sgl <= not(error_sgl);
else
my_rom(conv_integer(unsigned(start_address))) <=
     data_in_write(0);      my_rom(conv_integer
    (unsigned(start_address)+8)) <=
    data_in_write(8);
my_rom(conv_integer(unsigned(start_address)+1))
    <= data_in_write(1);    my_rom(conv_integer
    (unsigned(start_address)+9)) <=
    data_in_write(9);
my_rom(conv_integer(unsigned(start_address)+2))
    <= data_in_write(2);    my_rom(conv_integer
    (unsigned(start_address)+10)) <=
    data_in_write(10);
my_rom(conv_integer(unsigned(start_address)+3))
    <= data_in_write(3);    my_rom(conv_integer
    (unsigned(start_address)+11)) <=
    data_in_write(11);
my_rom(conv_integer(unsigned(start_address)+4))
    <= data_in_write(4);    my_rom(conv_integer
    (unsigned(start_address)+12)) <=
    data_in_write(12);
my_rom(conv_integer(unsigned(start_address)+5))
    <= data_in_write(5);    my_rom(conv_integer
    (unsigned(start_address)+13)) <=
    data_in_write(13);
my_rom(conv_integer(unsigned(start_address)+6))
    <= data_in_write(6);    my_rom(conv_integer
    (unsigned(start_address)+14)) <=
    data_in_write(14);
my_rom(conv_integer(unsigned(start_address)+7))
    <= data_in_write(7);    my_rom(conv_integer
    (unsigned(start_address)+15)) <=
    data_in_write(15);
rd_wr_lock_done <=      not(rd_wr_lock_done_sig)
    ;
```

```
rd_wr_lock_done_sig <=  not(rd_wr_lock_done_sig)
    ;
end if;
end if;
end if;
end if;
if((rd_en'event) and (mem_bank = "10") and(rd_en
    = '1')) then
rd_en_flag <= '1';
end if;
-- ***************** READ/WRITE
    ************************************--
end process;
end architecture TID_behavioral;
```

Listing A.13. Select command module

```
library ieee;
use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity select_match is
port (
clk: in std_logic;--clock signal.
Preset,mask_match,rd_done,powerup: in std_logic;
start_address : in std_logic_vector(23 downto 0)
    ;
end_address :in std_logic_vector(23 downto 0);
Mask_len: in STD_LOGIC_VECTOR(7 downto 0);
Mask_value_in: in STD_LOGIC_VECTOR(255 downto 0)
    ;
data: in STD_LOGIC_VECTOR(15 downto 0);
rd_en,match_valid,data_16_bit,select_done,
    use_epc : out std_logic;
start_address_out:out STD_LOGIC_VECTOR(23 downto
     0)
);
end entity select_match;

architecture select_match_behavioral of
    select_match is
signal count: unsigned(3 downto 0);
signal select_done_sgl,rd_done_sgl: std_logic;
signal mask_len_sig, count_bit: unsigned(7
    downto 0);
signal xor_value: unsigned(15 downto 0);
signal mask_value: unsigned(255 downto 0);
signal data_in: unsigned(15 downto 0);
signal start_address_sig,end_address_sig:
    unsigned(23 downto 0);

begin
process (clk,rd_done)
begin
if(powerup = '0') then
IF (Preset = '0') THEN
count <= "0000";
match_valid <= '0';
rd_done_sgl <= '0';
select_done_sgl <= '0';
use_epc <= '0';
start_address_sig <= "000000000000000000000000";
end_address_sig <= "000000000000000000000000";
count_bit <= "00000000";
end if;
ELSIF (RISING_EDGE(clk)) THEN
mask_len_sig <= unsigned(Mask_len);
mask_value <= unsigned(mask_value_in);
data_in <= unsigned(data);
end_address_sig(23 downto 4) <= unsigned(
    start_address(23 downto 4));
start_address_out <= std_logic_vector(
    start_address_sig);
if(mask_match = '1') then
start_address_sig(23 downto 4) <= unsigned(
    end_address(23 downto 4)) - count_bit;
rd_en <= '1';
data_16_bit <= '1';
use_epc <= '1';
if(end_address_sig(23 downto 4) <= (
    start_address_sig(23 downto 4) - count_bit)
     and (rd_done_sgl = '1')) then
rd_done_sgl <= '0';
count <= count + 1;
count_bit <= count_bit + 16;
if(count = "0001") then
if(end_address(3 downto 0) = "1111") then
if(data_in(15 downto 0) = mask_value(15 downto
    0)) then
mask_value <= mask_value srl 16 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1110") then
if(data_in(15 downto 1) = mask_value(14 downto
    0)) then
mask_value <= mask_value srl 15 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1101") then
if(data_in(15 downto 2) = mask_value(13 downto
    0)) then
mask_value <= mask_value srl 14 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1100") then
if(data_in(15 downto 3) = mask_value(12 downto
    0)) then
mask_value <= mask_value srl 13 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1011") then
if(data_in(15 downto 4) = mask_value(11 downto
    0)) then
mask_value <= mask_value srl 12 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1010") then
if(data_in(15 downto 5) = mask_value(10 downto
    0)) then
mask_value <= mask_value srl 11 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1001") then
if(data_in(15 downto 6) = mask_value(9 downto 0)
    ) then
mask_value <= mask_value srl 10 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "1000") then
if(data_in(15 downto 7) = mask_value(8 downto 0)
    ) then
mask_value <= mask_value srl 9 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0111") then
if(data_in(15 downto 8) = mask_value(7 downto 0)
    ) then
mask_value <= mask_value srl 8 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0110") then
if(data_in(15 downto 9) = mask_value(6 downto 0)
    ) then
mask_value <= mask_value srl 7 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0101") then
if(data_in(15 downto 10) = mask_value(5 downto
    0)) then
mask_value <= mask_value srl 6 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0100") then
if(data_in(15 downto 11) = mask_value(4 downto
    0)) then
mask_value <= mask_value srl 5 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0011") then
if(data_in(15 downto 12) = mask_value(3 downto
    0)) then
mask_value <= mask_value srl 4 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0010") then
if(data_in(15 downto 13) = mask_value(2 downto
    0)) then
mask_value <= mask_value srl 3 ;
match_valid <= '1';
end if;
```

```
elsif(end_address(3 downto 0) = "0001") then
if(data_in(15 downto 14) = mask_value(1 downto
    0)) then
mask_value <= mask_value srl 2 ;
match_valid <= '1';
end if;
elsif(end_address(3 downto 0) = "0000") then
if(data_in(15) = mask_value(0)) then
mask_value <= mask_value srl 1 ;
match_valid <= '1';
end if;
end if;
elsif((unsigned(start_address(11 downto 4) )-
    unsigned(end_address(11 downto 4))) = (
    count - 1)) then
if(start_address(3 downto 0) = "0000") then
if(data_in(15 downto 0) = mask_value(15 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0001") then
if(data_in(14 downto 0) = mask_value(14 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0010") then
if(data_in(13 downto 0) = mask_value(13 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0011") then
if(data_in(12 downto 0) = mask_value(12 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0100") then
if(data_in(11 downto 0) = mask_value(11 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0101") then
if(data_in(10 downto 0) = mask_value(10 downto
    0)) then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0110") then
if(data_in(9 downto 0) = mask_value(9 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "0111") then
if(data_in(8 downto 0) = mask_value(8 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1000") then
if(data_in(7 downto 0) = mask_value(7 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1001") then
if(data_in(6 downto 0) = mask_value(6 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1010") then
if(data_in(5 downto 0) = mask_value(5 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1011") then
if(data_in(4 downto 0) = mask_value(4 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1100") then
if(data_in(3 downto 0) = mask_value(3 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1101") then
if(data_in(2 downto 0) = mask_value(2 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1110") then
if(data_in(1 downto 0) = mask_value(1 downto 0))
    then
match_valid <= '1';
end if;
elsif(start_address(3 downto 0) = "1111") then
if(data_in(0) = mask_value(0)) then
match_valid <= '1';
end if;
end if;
else
if(data_in(15 downto 0) = mask_value(15 downto
    0)) then
mask_value <= mask_value srl 16 ;
match_valid <= '1';
end if;
end if;
else
rd_en <= '0';
use_epc <= '0';
select_done <= not(select_done_sgl);
select_done_sgl <= not(select_done_sgl);
end if;
end if;
end if;
if(rd_done'event) then
rd_done_sgl <= '1';
end if;
end process;
end architecture select_match_behavioral;
```

Listing A.14. Slot Counter module

```
library ieee;
use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
entity slot_counter is
port (
Preset: in std_logic;
Q_value_in: in std_logic_vector(3 downto 0);
RN_16_in: in std_logic_vector(15 downto 0);
slot_value_out: out std_logic_vector(15 downto
    0);
slot_done : out std_logic
);
end entity slot_counter;
architecture slot_counter_behavioral of
    slot_counter is
signal RN_16_sig: unsigned(15 downto 0);
signal slot_value_sig: unsigned(15 downto 0);
signal slot_done_sig : std_logic;
begin
RN_16_sig <= unsigned(RN_16_in);
process (Q_value_in)
begin
slot_done_sig <= '0';
if(Q_value_in = "0000") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    1);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0001") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    2);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0010") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    4);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0011") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    8);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0100") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    16);
slot_done <= not(slot_done_sig);
```

```
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0101") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    32);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0110") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    64);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "0111") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    128);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1000") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    256);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1001") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    512);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1010") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    1024);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1011") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    2048);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1100") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    4096);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1101") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    8192);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1110") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    16384);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
if(Q_value_in = "1111") then
slot_value_out <= std_logic_vector(RN_16_sig mod
    32768);
slot_done <= not(slot_done_sig);
slot_done_sig <= not(slot_done_sig);
end if;
end process;
end architecture slot_counter_behavioral;
```

Listing A.15. Output Buffer Module (Design 1:mutual auth Design C: secure ID)

```
--Including IEEE libraries.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

-- In an entity named "gen2", all required
     commands are initialised to the required
     bit lengths as specified in the protocol.
entity buffer_module is
port(

-- IN PORT
clk: in std_logic;--clock signal.
preset: in std_logic;
powerup: in std_logic;
start_buffer,truncate,error_code,rd_wr_lock_done
    ,kill_count,Reserved_access,rn_16,
    data_bit_in,mem_error,stop,use_epc_SM,
    no_reply,cipher_done,key_done: in std_logic
    ;
wht_cmd: in std_logic_vector(4 downto 0);
start_address,end_address: in std_logic_vector
    (23 downto 0);
mem_bank_in: in std_logic_vector(1 downto 0);
data_16_in, rn16_data, stored_crc_in: in
    std_logic_vector(15 downto 0);
Lock_Kill, Lock_Access, Lock_EPC, Lock_TID,
    Lock_User: in std_logic_vector(1 downto 0);
EPC: in std_logic_vector(95 downto 0);
EPC_done: in std_logic;
key_crc_done,rn_64: in std_logic;
data_key: in std_logic_vector(127 downto 0);
rn64_data, challenge_data: in std_logic_vector
    (63 downto 0);
crc_plain_text: in std_logic_vector(15 downto 0)
    ;
cipher_text : in std_logic_vector(63 downto 0);

-- OUT PORT
data_bit_out,rd_en,wr_en,reserved_done,
    data_16_bit,crc_flag,send_mem,start_crc,
    data_64_bit,key_crc,crypt_en,preset_engine,
    key_access,start_bit: out std_logic;
data_out: out std_logic_vector(15 downto 0);
mem_bank_out: out std_logic_vector(1 downto 0);
Lock_Kill_out, Lock_Access_out, Lock_EPC_out,
    Lock_TID_out, Lock_User_out: out
    std_logic_vector(1 downto 0);
data_plain_text : out std_logic_vector(63 downto
     0);
start_address_out,end_address_out: out
    std_logic_vector(23 downto 0)
);
end entity buffer_module;

architecture buffer_module_behavioral of
    buffer_module is
signal reserved_done_sgl,error_flag,rn16_flag,
    send_crc,start_buffer_flag,error_code_flag,
    start_crc_sgl,rd_wr_lock_done_sgl,stop_sgl,
    stored,mem_error_flag,rd_en_sig,
    start_bit_sgl,count_test: std_logic;
signal buffer_16,stored_crc: std_logic_vector(15
     downto 0);
signal count_16: unsigned(7 downto 0);
signal buffer_cipher_text: std_logic_vector(127
    downto 0);
signal buffer_cipher_text0,buffer_cipher_text1,
    buffer_cipher_text2: std_logic_vector(63
    downto 0);
signal delay_bit :unsigned(1 downto 0);
signal cipher_done_sgl:unsigned(2 downto 0);
signal rn_64_flag,count_preset_engine: std_logic
    ;
--signal key_80: std_logic_vector(79 downto 0);
signal buffer_64: std_logic_vector(63 downto 0);
signal secureid: std_logic_vector(159 downto 0);
signal error_bit: std_logic_vector(8 downto 0)
    := "100001111";

begin

process (clk,start_buffer,rd_wr_lock_done,
    error_code,rn_16,stop,mem_error,rn_64)
begin
if (RISING_EDGE(clk)) THEN
if(powerup = '0') then
if(preset = '0') then
count_16 <= "00000000";
crc_flag <= '0';
rn16_flag <= '0';
rn_64_flag <= '0';
error_flag <= '0';
send_crc <= '0';
stored <= '0';
```

```
start_buffer_flag <= '0';
rd_wr_lock_done_sgl <= '0';
error_code_flag <= '0';
start_crc_sgl <= '0';
reserved_done_sgl <= '0';
stop_sgl <= '0';
data_plain_text <= 'Z';
preset_engine <= 'Z';
count_preset_engine <= '0';
mem_error_flag <= '0';
rd_en_sig <= '0';
key_access <= '1';
cipher_done_sgl <= "000";
delay_bit <= "00";
start_bit_sgl <= '0';
count_test <= '0';

elsif(preset = '1') then
mem_bank_out <= mem_bank_in;
if(start_buffer_flag = '1') then
if(Reserved_access = '1') then
start_address_out <= start_address;
end_address_out <= end_address;
rd_en <= '1';
data_16_bit <= '1';
if(rd_wr_lock_done_sgl = '1') then
data_out <= data_16_in;
rd_wr_lock_done_sgl <= '0';
data_16_bit <= '0';
rd_en <= '0';
start_buffer_flag <= '0';
reserved_done <= not(reserved_done_sgl);
reserved_done_sgl <= not(reserved_done_sgl);
end if;
end if;
end if;
if(key_done = '1') then
key_access <= '0';
end if;
if(rn_64_flag = '1') then
if(count_preset_engine = '0') then
preset_engine <= '0';
crypt_en <= '1';
count_preset_engine <= '1';
elsif(count_preset_engine = '1') then
preset_engine <= '1';
crypt_en <= '1';
end if;
if(cipher_done = '1') then
cipher_done_sgl <= cipher_done_sgl + 1;
count_preset_engine <= '0';
end if;
if(cipher_done_sgl = "000") then
data_plain_text <= rn64_data;
elsif(cipher_done_sgl = "001") then
if(delay_bit = "00") then
data_plain_text <= rn64_data xor cipher_text;
buffer_cipher_text0 <= cipher_text;
delay_bit <= "01";
end if;
elsif(cipher_done_sgl = "010") then
if(delay_bit = "01") then
data_plain_text <= rn64_data xor cipher_text;
buffer_cipher_text1 <= cipher_text;
delay_bit <= "10";
end if;
elsif(cipher_done_sgl = "011") then
if(delay_bit = "10") then
data_plain_text <= rn64_data xor cipher_text;
buffer_cipher_text2 <= cipher_text;
delay_bit <= "11";
end if;
crypt_en <= '0';
if(EPC_done = '1') then
buffer_cipher_text <= buffer_cipher_text0 &
    buffer_cipher_text1;
if (delay_bit = "11") then
secureid <= buffer_cipher_text2 & (EPC xor
    buffer_cipher_text(95 downto 0));
delay_bit <= "00";
rn_64_flag <= '0';
count_preset_engine <= '0';
cipher_done_sgl <= "000";
end if;
end if;
end if;
end if;
end if;
elsif(powerup = '1') then
mem_bank_out <= mem_bank_in;
stored_crc <= stored_crc_in;
if( (use_epc_SM = '0' and mem_bank_in="01") or (
    mem_bank_in = "00") or (mem_bank_in ="10")
    or (mem_bank_in = "11") )then
if(start_buffer_flag = '1') then
if(no_reply = '1') then
crc_flag <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);

count_test <= '1';
end if;
end if;
if(error_code_flag = '1') then
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00001000") then
data_bit_out <= error_bit(8); error_bit(8) <=
    error_bit(7); error_bit(7) <= error_bit(6);
error_bit(6) <= error_bit(5); error_bit(5) <=
    error_bit(4); error_bit(4) <= error_bit(3);
error_bit(3) <= error_bit(2); error_bit(2) <=
    error_bit(1); error_bit(1) <= error_bit(0);

count_16 <= count_16 + 1;
end if;

if(rn16_flag = '1') then
if( (count_16 > "00001000") and (count_16 <= "
    00011000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
error_code_flag <= '0';
start_buffer_flag <= '0';
crc_flag <= '0';
count_test <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
end if;

case wht_cmd is
when "00010" =>          -- when Query CMD
crc_flag <= '0';

if(rn16_flag = '1') then
if(delay_bit = "00") then
buffer_16 <= rn16_data;
delay_bit <= "01";
else
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "00001111") then
```

```
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
elsif(count_16 = "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
end if;
end if;
end if;
end if;

when "00000"      =>              -- when QueryRep
     CMD
crc_flag <= '0';

if(rn16_flag = '1') then
if(delay_bit = "00") then
buffer_16 <= rn16_data;
delay_bit <= "01";
else
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
elsif(count_16 = "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
end if;
end if;
end if;
end if;

when "00011" =>                       -- when
     QueryAdjust
crc_flag <= '0';

if(rn16_flag = '1') then
if(delay_bit = "00") then
buffer_16 <= rn16_data;
delay_bit <= "01";
else
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
end if;
end if;
end if;
end if;

when "00001" =>           -- when ACK CMD
crc_flag <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "10011111") then
```

```
data_bit_out <= secureid(159);

secureid(159) <= secureid(158); secureid(158) <=
     secureid(157); secureid(157) <= secureid
    (156);
secureid(156) <= secureid(155); secureid(155) <=
     secureid(154); secureid(154) <= secureid
    (153);
secureid(153) <= secureid(152); secureid(152) <=
     secureid(151); secureid(151) <= secureid
    (150);
secureid(150) <= secureid(149); secureid(149) <=
     secureid(148); secureid(148) <= secureid
    (147);
secureid(147) <= secureid(146); secureid(146) <=
     secureid(145); secureid(145) <= secureid
    (144);
secureid(144) <= secureid(143); secureid(143) <=
     secureid(142); secureid(142) <= secureid
    (141);
secureid(141) <= secureid(140); secureid(140) <=
     secureid(139); secureid(139) <= secureid
    (138);
secureid(138) <= secureid(137); secureid(137) <=
     secureid(136); secureid(136) <= secureid
    (135);
secureid(135) <= secureid(134); secureid(134) <=
     secureid(133); secureid(133) <= secureid
    (132);
secureid(132) <= secureid(131); secureid(131) <=
     secureid(130); secureid(130) <= secureid
    (129);
secureid(129) <= secureid(128); secureid(128) <=
     secureid(127); secureid(127) <= secureid
    (126);
secureid(126) <= secureid(125); secureid(125) <=
     secureid(124); secureid(124) <= secureid
    (123);
secureid(123) <= secureid(122); secureid(122) <=
     secureid(121); secureid(121) <= secureid
    (120);
secureid(120) <= secureid(119); secureid(119) <=
     secureid(118); secureid(118) <= secureid
    (117);
secureid(117) <= secureid(116); secureid(116) <=
     secureid(115); secureid(115) <= secureid
    (114);
secureid(114) <= secureid(113); secureid(113) <=
     secureid(112); secureid(112) <= secureid
    (111);
secureid(111) <= secureid(110); secureid(110) <=
     secureid(109); secureid(109) <= secureid
    (108);
secureid(108) <= secureid(107); secureid(107) <=
     secureid(106); secureid(106) <= secureid
    (105);
secureid(105) <= secureid(104); secureid(104) <=
     secureid(103); secureid(103) <= secureid
    (102);
secureid(102) <= secureid(101); secureid(101) <=
     secureid(100); secureid(100) <= secureid
    (99);
secureid(99) <= secureid(98); secureid(98) <=
    secureid(97); secureid(97) <= secureid(96);
secureid(96) <= secureid(95); secureid(95) <=
    secureid(94); secureid(94) <= secureid(93);
secureid(93) <= secureid(92); secureid(92) <=
    secureid(91); secureid(91) <= secureid(90);
secureid(90) <= secureid(89); secureid(89) <=
    secureid(88); secureid(88) <= secureid(87);
secureid(87) <= secureid(86); secureid(86) <=
    secureid(85); secureid(85) <= secureid(84);
secureid(84) <= secureid(83); secureid(83) <=
    secureid(82); secureid(82) <= secureid(81);
secureid(81) <= secureid(80); secureid(80) <=
    secureid(79); secureid(79) <= secureid(78);
secureid(78) <= secureid(77); secureid(77) <=
    secureid(76); secureid(76) <= secureid(75);
secureid(75) <= secureid(74); secureid(74) <=
    secureid(73); secureid(73) <= secureid(72);
secureid(72) <= secureid(71); secureid(71) <=
    secureid(70); secureid(70) <= secureid(69);
secureid(69) <= secureid(68); secureid(68) <=
    secureid(67); secureid(67) <= secureid(66);
secureid(66) <= secureid(65); secureid(65) <=
    secureid(64); secureid(64) <= secureid(63);
secureid(63) <= secureid(62); secureid(62) <=
    secureid(61); secureid(61) <= secureid(60);
secureid(60) <= secureid(59); secureid(59) <=
    secureid(58); secureid(58) <= secureid(57);
secureid(57) <= secureid(56); secureid(56) <=
    secureid(55); secureid(55) <= secureid(54);
secureid(54) <= secureid(53); secureid(53) <=
    secureid(52); secureid(52) <= secureid(51);
secureid(51) <= secureid(50); secureid(50) <=
    secureid(49); secureid(49) <= secureid(48);
secureid(48) <= secureid(47); secureid(47) <=
    secureid(46); secureid(46) <= secureid(45);
secureid(45) <= secureid(44); secureid(44) <=
    secureid(43); secureid(43) <= secureid(42);
secureid(42) <= secureid(41); secureid(41) <=
    secureid(40); secureid(40) <= secureid(39);
secureid(39) <= secureid(38); secureid(38) <=
    secureid(37); secureid(37) <= secureid(36);
secureid(36) <= secureid(35); secureid(35) <=
    secureid(34); secureid(34) <= secureid(33);
secureid(33) <= secureid(32); secureid(32) <=
    secureid(31); secureid(31) <= secureid(30);
secureid(30) <= secureid(29); secureid(29) <=
    secureid(28); secureid(28) <= secureid(27);
secureid(27) <= secureid(26); secureid(26) <=
    secureid(25); secureid(25) <= secureid(24);
secureid(24) <= secureid(23); secureid(23) <=
    secureid(22); secureid(22) <= secureid(21);
secureid(21) <= secureid(20); secureid(20) <=
    secureid(19); secureid(19) <= secureid(18);
secureid(18) <= secureid(17); secureid(17) <=
    secureid(16); secureid(16) <= secureid(15);
secureid(15) <= secureid(14); secureid(14) <=
    secureid(13); secureid(13) <= secureid(12);
secureid(12) <= secureid(11); secureid(11) <=
    secureid(10); secureid(10) <= secureid(9);
secureid(9) <= secureid(8); secureid(8) <=
    secureid(7); secureid(7) <= secureid(6);
secureid(6) <= secureid(5); secureid(5) <=
    secureid(4); secureid(4) <= secureid(3);
secureid(3) <= secureid(2); secureid(2) <=
    secureid(1); secureid(1) <= secureid(0);


count_16 <= count_16 + 1;
elsif( count_16 = "10011111") then
data_bit_out <= secureid(159);

secureid(159) <= secureid(158); secureid(158) <=
     secureid(157); secureid(157) <= secureid
    (156);
secureid(156) <= secureid(155); secureid(155) <=
     secureid(154); secureid(154) <= secureid
    (153);
secureid(153) <= secureid(152); secureid(152) <=
     secureid(151); secureid(151) <= secureid
    (150);
secureid(150) <= secureid(149); secureid(149) <=
     secureid(148); secureid(148) <= secureid
    (147);
secureid(147) <= secureid(146); secureid(146) <=
     secureid(145); secureid(145) <= secureid
    (144);
secureid(144) <= secureid(143); secureid(143) <=
     secureid(142); secureid(142) <= secureid
    (141);
secureid(141) <= secureid(140); secureid(140) <=
     secureid(139); secureid(139) <= secureid
    (138);
secureid(138) <= secureid(137); secureid(137) <=
     secureid(136); secureid(136) <= secureid
    (135);
secureid(135) <= secureid(134); secureid(134) <=
     secureid(133); secureid(133) <= secureid
    (132);
secureid(132) <= secureid(131); secureid(131) <=
     secureid(130); secureid(130) <= secureid
    (129);
secureid(129) <= secureid(128); secureid(128) <=
     secureid(127); secureid(127) <= secureid
    (126);
secureid(126) <= secureid(125); secureid(125) <=
     secureid(124); secureid(124) <= secureid
    (123);
secureid(123) <= secureid(122); secureid(122) <=
     secureid(121); secureid(121) <= secureid
    (120);
```

```
secureid(120) <= secureid(119); secureid(119) <=
     secureid(118); secureid(118) <= secureid
    (117);
secureid(117) <= secureid(116); secureid(116) <=
     secureid(115); secureid(115) <= secureid
    (114);
secureid(114) <= secureid(113); secureid(113) <=
     secureid(112); secureid(112) <= secureid
    (111);
secureid(111) <= secureid(110); secureid(110) <=
     secureid(109); secureid(109) <= secureid
    (108);
secureid(108) <= secureid(107); secureid(107) <=
     secureid(106); secureid(106) <= secureid
    (105);
secureid(105) <= secureid(104); secureid(104) <=
     secureid(103); secureid(103) <= secureid
    (102);
secureid(102) <= secureid(101); secureid(101) <=
     secureid(100); secureid(100) <= secureid
    (99);
secureid(99) <= secureid(98); secureid(98) <=
    secureid(97); secureid(97) <= secureid(96);
secureid(96) <= secureid(95); secureid(95) <=
    secureid(94); secureid(94) <= secureid(93);
secureid(93) <= secureid(92); secureid(92) <=
    secureid(91); secureid(91) <= secureid(90);
secureid(90) <= secureid(89); secureid(89) <=
    secureid(88); secureid(88) <= secureid(87);
secureid(87) <= secureid(86); secureid(86) <=
    secureid(85); secureid(85) <= secureid(84);
secureid(84) <= secureid(83); secureid(83) <=
    secureid(82); secureid(82) <= secureid(81);
secureid(81) <= secureid(80); secureid(80) <=
    secureid(79); secureid(79) <= secureid(78);
secureid(78) <= secureid(77); secureid(77) <=
    secureid(76); secureid(76) <= secureid(75);
secureid(75) <= secureid(74); secureid(74) <=
    secureid(73); secureid(73) <= secureid(72);
secureid(72) <= secureid(71); secureid(71) <=
    secureid(70); secureid(70) <= secureid(69);
secureid(69) <= secureid(68); secureid(68) <=
    secureid(67); secureid(67) <= secureid(66);
secureid(66) <= secureid(65); secureid(65) <=
    secureid(64); secureid(64) <= secureid(63);
secureid(63) <= secureid(62); secureid(62) <=
    secureid(61); secureid(61) <= secureid(60);
secureid(60) <= secureid(59); secureid(59) <=
    secureid(58); secureid(58) <= secureid(57);
secureid(57) <= secureid(56); secureid(56) <=
    secureid(55); secureid(55) <= secureid(54);
secureid(54) <= secureid(53); secureid(53) <=
    secureid(52); secureid(52) <= secureid(51);
secureid(51) <= secureid(50); secureid(50) <=
    secureid(49); secureid(49) <= secureid(48);
secureid(48) <= secureid(47); secureid(47) <=
    secureid(46); secureid(46) <= secureid(45);
secureid(45) <= secureid(44); secureid(44) <=
    secureid(43); secureid(43) <= secureid(42);
secureid(42) <= secureid(41); secureid(41) <=
    secureid(40); secureid(40) <= secureid(39);
secureid(39) <= secureid(38); secureid(38) <=
    secureid(37); secureid(37) <= secureid(36);
secureid(36) <= secureid(35); secureid(35) <=
    secureid(34); secureid(34) <= secureid(33);
secureid(33) <= secureid(32); secureid(32) <=
    secureid(31); secureid(31) <= secureid(30);
secureid(30) <= secureid(29); secureid(29) <=
    secureid(28); secureid(28) <= secureid(27);
secureid(27) <= secureid(26); secureid(26) <=
    secureid(25); secureid(25) <= secureid(24);
secureid(24) <= secureid(23); secureid(23) <=
    secureid(22); secureid(22) <= secureid(21);
secureid(21) <= secureid(20); secureid(20) <=
    secureid(19); secureid(19) <= secureid(18);
secureid(18) <= secureid(17); secureid(17) <=
    secureid(16); secureid(16) <= secureid(15);
secureid(15) <= secureid(14); secureid(14) <=
    secureid(13); secureid(13) <= secureid(12);
secureid(12) <= secureid(11); secureid(11) <=
    secureid(10); secureid(10) <= secureid(9);
secureid(9) <= secureid(8); secureid(8) <=
    secureid(7); secureid(7) <= secureid(6);
secureid(6) <= secureid(5); secureid(5) <=
    secureid(4); secureid(4) <= secureid(3);
secureid(3) <= secureid(2); secureid(2) <=
    secureid(1); secureid(1) <= secureid(0);
```

```
count_16 <= count_16 + 1;

start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn_64_flag <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
count_test <= '0';
count_preset_engine <= '0';
cipher_done_sgl <= "000";

end if;
end if;

when "00110" =>                    -- when NAK CMD

when "00111" =>                    -- when REQ_RN
    Command
crc_flag <= '1';

if(rn16_flag = '1') then
if(delay_bit = "00") then
buffer_16 <= rn16_data;
delay_bit <= "01";
else
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
elsif(count_16 = "00001111") then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
end if;
end if;
end if;
end if;

when "00100" =>          -- when Select CMD

when "01000" =>          -- when Read CMD
crc_flag <= '1';
if (error_flag = '0') then
start_address_out <= start_address;
end_address_out <= end_address;
send_mem <= '1';
```

```
rd_en <= '1';
data_16_bit <= '0';
if(rd_wr_lock_done_sgl = '1') then
rd_wr_lock_done_sgl <= '0';
rd_en <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00000000") then
data_bit_out <= '0';
count_16 <= count_16 + 1;
else
data_bit_out <= data_bit_in;
end if;
end if;
end if;
if(stop_sgl = '1') then
rd_en <= '0';
if(rn16_flag = '1') then
if((count_16 > "00000000") and (count_16 <= "
    00010000")) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <= "00000000";
rn16_flag <= '0';
stop_sgl <= '0';
count_test <= '0';
start_buffer_flag <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
if(mem_error_flag = '1') then
error_flag <= '1';
send_mem <= '0';
mem_error_flag <= '0';
count_16 <="00000000";
rd_en <= '0';
end if;
end if;
if(error_flag = '1') then
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00001000") then
data_bit_out <= error_bit(8); error_bit(8) <=
    error_bit(7); error_bit(7) <= error_bit(6);
error_bit(6) <= error_bit(5); error_bit(5) <=
    error_bit(4); error_bit(4) <= error_bit(3);
error_bit(3) <= error_bit(2); error_bit(2) <=
    error_bit(1); error_bit(1) <= error_bit(0);

count_16 <= count_16 + 1;
end if;

if(rn16_flag = '1') then
if( (count_16 > "00001000") and (count_16 <= "
    00011000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
error_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
end if;

when "01001" =>          -- when Write CMD
crc_flag <= '1';
if (error_flag = '0') then
start_address_out <= start_address;
end_address_out <= end_address;
send_mem <= '1';
wr_en <= '1';
if(rd_wr_lock_done_sgl = '1') then
wr_en <= '0';
rd_wr_lock_done_sgl <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00000000") then
data_bit_out <= '0';
count_16 <= count_16 + 1;
end if;
end if;
end if;
if(mem_error_flag = '1') then
error_flag <= '1';
send_mem <= '0';
mem_error_flag <= '0';
wr_en <= '0';
count_16 <="00000000";
end if;


if(rn16_flag = '1') then
if( (count_16 > "00000000") and (count_16 <= "
    00010000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
    buffer_16(14); buffer_16(14) <= buffer_16
    (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
     buffer_16(11); buffer_16(11) <= buffer_16
    (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
    buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
    buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
    buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
start_buffer_flag <= '0';
count_test <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;

if(error_flag = '1') then
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
```

```
if(count_16 <= "00001000") then
data_bit_out <= error_bit(8); error_bit(8) <=
     error_bit(7); error_bit(7) <= error_bit(6);
error_bit(6) <= error_bit(5); error_bit(5) <=
     error_bit(4); error_bit(4) <= error_bit(3);
error_bit(3) <= error_bit(2); error_bit(2) <=
     error_bit(1); error_bit(1) <= error_bit(0);

count_16 <= count_16 + 1;
end if;
if(rn16_flag = '1') then
if( (count_16 > "00001000") and (count_16 <= "
     00011000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
error_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
end if;

when "01010" =>                    -- when Kill CMD
crc_flag <= '1';
if(kill_count = '0') then

if(rn16_flag = '1') then
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 <= "00001111" ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
elsif(kill_count = '1') then
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00000000") then
data_bit_out <= '0';
count_16 <= count_16 + 1;
end if;

if(rn16_flag = '1') then
if( (count_16 > "00000000") and (count_16 <= "
     00010000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);

count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
start_buffer_flag <= '0';
count_test <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
end if;
end if;
when "01011" =>                    -- when Lock
     command
crc_flag <= '1';
Lock_Kill_out <= Lock_Kill;
Lock_Access_out <= Lock_Access;
Lock_EPC_out <= Lock_EPC;
Lock_TID_out <= Lock_TID;
Lock_User_out <= Lock_User;

if(rd_wr_lock_done_sgl = '1') then
rd_wr_lock_done_sgl <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00000000") then
data_bit_out <= '0';
count_16 <= count_16 + 1;
end if;
end if;
end if;
if(mem_error_flag = '1') then
mem_error_flag <= '0';
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if(count_16 <= "00000000") then
data_bit_out <= '1';
count_16 <= count_16 + 1;
end if;
end if;
end if;

if(rn16_flag = '1') then
if( (count_16 > "00000000") and (count_16 <= "
     00011000") ) then
data_bit_out <= buffer_16(15); buffer_16(15) <=
     buffer_16(14); buffer_16(14) <= buffer_16
     (13);
buffer_16(13) <= buffer_16(12); buffer_16(12) <=
      buffer_16(11); buffer_16(11) <= buffer_16
     (10);
buffer_16(10) <= buffer_16(9); buffer_16(9) <=
     buffer_16(8); buffer_16(8) <= buffer_16(7);
buffer_16(7) <= buffer_16(6); buffer_16(6) <=
     buffer_16(5); buffer_16(5) <= buffer_16(4);
buffer_16(4) <= buffer_16(3); buffer_16(3) <=
     buffer_16(2); buffer_16(2) <= buffer_16(1);
buffer_16(1) <= buffer_16(0);
```

```
count_16 <= count_16 + 1;
else
count_16 <="00000000";
rn16_flag <= '0';
count_test <= '0';
start_buffer_flag <= '0';
start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
end if;
end if;
when "10000" =>                                 --
     Challenge
crc_flag <= '0';
if(count_preset_engine = '0') then
preset_engine <= '0';
crypt_en <= '1';
count_preset_engine <= '1';
elsif(count_preset_engine = '1') then
preset_engine <= '1';
crypt_en <= '1';
end if;

if(cipher_done = '1') then
cipher_done_sgl <= cipher_done_sgl + 1;
count_preset_engine <= '0';
end if;
if(cipher_done_sgl = "000") then
data_plain_text <= cipher_text;
elsif(cipher_done_sgl = "001") then
if(cipher_text = challenge_data) then
data_plain_text <= cipher_text;
else
crypt_en <= '0';
end if;
elsif(cipher_done_sgl = "010") then
crypt_en <= '0';
if(delay_bit = "00") then
buffer_cipher_text(63 downto 0) <= cipher_text;
delay_bit <= "01";
else
if(count_test = '0') then
start_bit <= not(start_bit_sgl);
start_bit_sgl <= not(start_bit_sgl);

count_test <= '1';
else
if( count_16 < "00111111") then
data_bit_out <= buffer_cipher_text(63);

buffer_cipher_text(63) <= buffer_cipher_text(62)
     ; buffer_cipher_text(62) <=
     buffer_cipher_text(61);
buffer_cipher_text(61) <= buffer_cipher_text(60)
     ; buffer_cipher_text(60) <=
     buffer_cipher_text(59); buffer_cipher_text
     (59) <= buffer_cipher_text(58);
buffer_cipher_text(58) <= buffer_cipher_text(57)
     ; buffer_cipher_text(57) <=
     buffer_cipher_text(56); buffer_cipher_text
     (56) <= buffer_cipher_text(55);
buffer_cipher_text(55) <= buffer_cipher_text(54)
     ; buffer_cipher_text(54) <=
     buffer_cipher_text(53); buffer_cipher_text
     (53) <= buffer_cipher_text(52);
buffer_cipher_text(52) <= buffer_cipher_text(51)
     ; buffer_cipher_text(51) <=
     buffer_cipher_text(50); buffer_cipher_text
     (50) <= buffer_cipher_text(49);
buffer_cipher_text(49) <= buffer_cipher_text(48)
     ; buffer_cipher_text(48) <=
     buffer_cipher_text(47); buffer_cipher_text
     (47) <= buffer_cipher_text(46);
buffer_cipher_text(46) <= buffer_cipher_text(45)
     ; buffer_cipher_text(45) <=
     buffer_cipher_text(44); buffer_cipher_text
     (44) <= buffer_cipher_text(43);
buffer_cipher_text(43) <= buffer_cipher_text(42)
     ; buffer_cipher_text(42) <=
     buffer_cipher_text(41); buffer_cipher_text
     (41) <= buffer_cipher_text(40);
buffer_cipher_text(40) <= buffer_cipher_text(39)
     ; buffer_cipher_text(39) <=
     buffer_cipher_text(38); buffer_cipher_text
     (38) <= buffer_cipher_text(37);
buffer_cipher_text(37) <= buffer_cipher_text(36)
     ; buffer_cipher_text(36) <=
     buffer_cipher_text(35); buffer_cipher_text
     (35) <= buffer_cipher_text(34);
buffer_cipher_text(34) <= buffer_cipher_text(33)
     ; buffer_cipher_text(33) <=
     buffer_cipher_text(32); buffer_cipher_text
     (32) <= buffer_cipher_text(31);
buffer_cipher_text(31) <= buffer_cipher_text(30)
     ; buffer_cipher_text(30) <=
     buffer_cipher_text(29); buffer_cipher_text
     (29) <= buffer_cipher_text(28);
buffer_cipher_text(28) <= buffer_cipher_text(27)
     ; buffer_cipher_text(27) <=
     buffer_cipher_text(26); buffer_cipher_text
     (26) <= buffer_cipher_text(25);
buffer_cipher_text(25) <= buffer_cipher_text(24)
     ; buffer_cipher_text(24) <=
     buffer_cipher_text(23); buffer_cipher_text
     (23) <= buffer_cipher_text(22);
buffer_cipher_text(22) <= buffer_cipher_text(21)
     ; buffer_cipher_text(21) <=
     buffer_cipher_text(20); buffer_cipher_text
     (20) <= buffer_cipher_text(19);
buffer_cipher_text(19) <= buffer_cipher_text(18)
     ; buffer_cipher_text(18) <=
     buffer_cipher_text(17); buffer_cipher_text
     (17) <= buffer_cipher_text(16);
buffer_cipher_text(16) <= buffer_cipher_text(15)
     ; buffer_cipher_text(15) <=
     buffer_cipher_text(14); buffer_cipher_text
     (14) <= buffer_cipher_text(13);
buffer_cipher_text(13) <= buffer_cipher_text(12)
     ; buffer_cipher_text(12) <=
     buffer_cipher_text(11); buffer_cipher_text
     (11) <= buffer_cipher_text(10);
buffer_cipher_text(10) <= buffer_cipher_text(9);
      buffer_cipher_text(9) <=
     buffer_cipher_text(8); buffer_cipher_text
     (8) <= buffer_cipher_text(7);
buffer_cipher_text(7) <= buffer_cipher_text(6);
     buffer_cipher_text(6) <= buffer_cipher_text
     (5); buffer_cipher_text(5) <=
     buffer_cipher_text(4);
buffer_cipher_text(4) <= buffer_cipher_text(3);
     buffer_cipher_text(3) <= buffer_cipher_text
     (2); buffer_cipher_text(2) <=
     buffer_cipher_text(1);
buffer_cipher_text(1) <= buffer_cipher_text(0);

count_16 <= count_16 + 1;
elsif( count_16 = "00111111") then
data_bit_out <= buffer_cipher_text(63);

buffer_cipher_text(63) <= buffer_cipher_text(62)
     ; buffer_cipher_text(62) <=
     buffer_cipher_text(61);
buffer_cipher_text(61) <= buffer_cipher_text(60)
     ; buffer_cipher_text(60) <=
     buffer_cipher_text(59); buffer_cipher_text
     (59) <= buffer_cipher_text(58);
buffer_cipher_text(58) <= buffer_cipher_text(57)
     ; buffer_cipher_text(57) <=
     buffer_cipher_text(56); buffer_cipher_text
     (56) <= buffer_cipher_text(55);
buffer_cipher_text(55) <= buffer_cipher_text(54)
     ; buffer_cipher_text(54) <=
     buffer_cipher_text(53); buffer_cipher_text
     (53) <= buffer_cipher_text(52);
buffer_cipher_text(52) <= buffer_cipher_text(51)
     ; buffer_cipher_text(51) <=
     buffer_cipher_text(50); buffer_cipher_text
     (50) <= buffer_cipher_text(49);
buffer_cipher_text(49) <= buffer_cipher_text(48)
     ; buffer_cipher_text(48) <=
     buffer_cipher_text(47); buffer_cipher_text
     (47) <= buffer_cipher_text(46);
buffer_cipher_text(46) <= buffer_cipher_text(45)
     ; buffer_cipher_text(45) <=
     buffer_cipher_text(44); buffer_cipher_text
     (44) <= buffer_cipher_text(43);
buffer_cipher_text(43) <= buffer_cipher_text(42)
     ; buffer_cipher_text(42) <=
     buffer_cipher_text(41); buffer_cipher_text
     (41) <= buffer_cipher_text(40);
buffer_cipher_text(40) <= buffer_cipher_text(39)
     ; buffer_cipher_text(39) <=
     buffer_cipher_text(38); buffer_cipher_text
     (38) <= buffer_cipher_text(37);
```

```
buffer_cipher_text(37) <= buffer_cipher_text(36)
     ; buffer_cipher_text(36) <=
     buffer_cipher_text(35); buffer_cipher_text
     (35) <= buffer_cipher_text(34);
buffer_cipher_text(34) <= buffer_cipher_text(33)
     ; buffer_cipher_text(33) <=
     buffer_cipher_text(32); buffer_cipher_text
     (32) <= buffer_cipher_text(31);
buffer_cipher_text(31) <= buffer_cipher_text(30)
     ; buffer_cipher_text(30) <=
     buffer_cipher_text(29); buffer_cipher_text
     (29) <= buffer_cipher_text(28);
buffer_cipher_text(28) <= buffer_cipher_text(27)
     ; buffer_cipher_text(27) <=
     buffer_cipher_text(26); buffer_cipher_text
     (26) <= buffer_cipher_text(25);
buffer_cipher_text(25) <= buffer_cipher_text(24)
     ; buffer_cipher_text(24) <=
     buffer_cipher_text(23); buffer_cipher_text
     (23) <= buffer_cipher_text(22);
buffer_cipher_text(22) <= buffer_cipher_text(21)
     ; buffer_cipher_text(21) <=
     buffer_cipher_text(20); buffer_cipher_text
     (20) <= buffer_cipher_text(19);
buffer_cipher_text(19) <= buffer_cipher_text(18)
     ; buffer_cipher_text(18) <=
     buffer_cipher_text(17); buffer_cipher_text
     (17) <= buffer_cipher_text(16);
buffer_cipher_text(16) <= buffer_cipher_text(15)
     ; buffer_cipher_text(15) <=
     buffer_cipher_text(14); buffer_cipher_text
     (14) <= buffer_cipher_text(13);
buffer_cipher_text(13) <= buffer_cipher_text(12)
     ; buffer_cipher_text(12) <=
     buffer_cipher_text(11); buffer_cipher_text
     (11) <= buffer_cipher_text(10);
buffer_cipher_text(10) <= buffer_cipher_text(9);
      buffer_cipher_text(9) <=
     buffer_cipher_text(8); buffer_cipher_text
     (8) <= buffer_cipher_text(7);
buffer_cipher_text(7) <= buffer_cipher_text(6);
     buffer_cipher_text(6) <= buffer_cipher_text
     (5); buffer_cipher_text(5) <=
     buffer_cipher_text(4);
buffer_cipher_text(4) <= buffer_cipher_text(3);
     buffer_cipher_text(3) <= buffer_cipher_text
     (2); buffer_cipher_text(2) <=
     buffer_cipher_text(1);
buffer_cipher_text(1) <= buffer_cipher_text(0);

count_16 <= count_16 + 1;

start_crc <= not(start_crc_sgl);
start_crc_sgl <= not(start_crc_sgl);
else
count_16 <="00000000";
rn_64_flag <= '0';
start_buffer_flag <= '0';
delay_bit <= "00";
count_test <= '1';
count_preset_engine <= '0';
cipher_done_sgl <= "000";
end if;
end if;
end if;
end if;
when others =>
end case;
end if;
end if;
end if;
end if;
if(start_buffer'event) then
start_buffer_flag <= '1';
end if;
if(rd_wr_lock_done'event and (rd_wr_lock_done =
     '1' or rd_wr_lock_done = '0')) then
rd_wr_lock_done_sgl <= '1';
end if;
if(error_code'event) then
crc_flag <= '1';
error_code_flag <= '1';
end if;
if(rn_16'event) then
rn16_flag <= '1';
end if;
if(rn_64'event) then
rn_64_flag <= '1';
end if;
if(stop'event) then
stop_sgl <= '1';
end if;
if(mem_error'event) then
mem_error_flag <= '1';
end if;
end process;
end architecture buffer_module_behavioral;
```

Listing A.16. Output CRC-16 Engine - TID Rom

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
LIBRARY work;

ENTITY CRC16_final IS
PORT
(
Preset,powerup : IN STD_LOGIC;
clk : IN STD_LOGIC;
data : IN STD_LOGIC;
start_crc, crc_flag,start_bit : IN STD_LOGIC;
Q15, Q14, Q13, Q12, Q11, Q10, Q9, Q8, Q7, Q6, Q5
     , Q4, Q3, Q2, Q1, Q0 : OUT STD_LOGIC;
data_bit_out, done : OUT STD_LOGIC
);
END CRC16_final;

ARCHITECTURE bdf_type OF CRC16_final IS

SIGNAL SYNTHESIZED_WIRE_5, stop_crc,
     start_crc_flag,copy_crc_flag,done_sgl :
     STD_LOGIC;
SIGNAL DFF_inst : STD_LOGIC;
SIGNAL DFF_inst9 : STD_LOGIC;
SIGNAL DFF_inst10 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_1 : STD_LOGIC;
SIGNAL DFF_inst14 : STD_LOGIC;
SIGNAL DFF_inst15 : STD_LOGIC;
SIGNAL DFF_inst12 : STD_LOGIC;
SIGNAL DFF_inst13 : STD_LOGIC;
SIGNAL DFF_inst4 : STD_LOGIC;
SIGNAL DFF_inst11 : STD_LOGIC;
SIGNAL DFF_inst1 : STD_LOGIC;
SIGNAL DFF_inst2 : STD_LOGIC;
SIGNAL DFF_inst3 : STD_LOGIC;
SIGNAL SYNTHESIZED_WIRE_4 : STD_LOGIC;
SIGNAL DFF_inst5 : STD_LOGIC;
SIGNAL DFF_inst6 : STD_LOGIC;
SIGNAL DFF_inst7 : STD_LOGIC;
SIGNAL start_bit_flag : STD_LOGIC;
SIGNAL delay_bit : STD_LOGIC_VECTOR(1 downto
     0);
SIGNAL DFF_inst8 : STD_LOGIC;
signal crc_value : STD_LOGIC_VECTOR(15 downto
     0);
signal count_16 : unsigned(4 downto 0);

BEGIN
Q15 <= DFF_inst13;       Q14 <= DFF_inst14; Q13
     <= DFF_inst15; Q12 <= DFF_inst12; Q11 <=
     DFF_inst11; Q10 <= DFF_inst10; Q9 <=
     DFF_inst9;
Q8 <= DFF_inst8; Q7 <= DFF_inst7; Q6 <=
     DFF_inst6; Q5 <= DFF_inst5; Q4 <= DFF_inst4
     ; Q3 <= DFF_inst3; Q2 <= DFF_inst2; Q1 <=
     DFF_inst1;
Q0 <= DFF_inst;

PROCESS(clk,Preset,start_crc,start_bit)
BEGIN
if(powerup = '0') then
IF (Preset = '0') THEN
DFF_inst <= '1'; DFF_inst1 <= '1';
     DFF_inst2 <= '1'; DFF_inst3 <= '1';
     DFF_inst4 <= '1'; DFF_inst5 <= '1';
     DFF_inst6 <= '1';
```

```
DFF_inst7 <= '1'; DFF_inst8 <= '1'; DFF_inst9 <=
      '1'; DFF_inst10 <= '1'; DFF_inst11 <= '1';
       DFF_inst12 <= '1'; DFF_inst13 <= '1';
DFF_inst14 <= '1'; DFF_inst15 <= '1'; stop_crc
     <= '0'; count_16 <= "00000"; start_crc_flag
      <= '0'; copy_crc_flag <= '0'; done_sgl <=
      '0';
delay_bit <= "00"; start_bit_flag <= '0';
end if;
ELSIF (RISING_EDGE(clk)) THEN
--done <= done_sgl;
if(start_bit_flag = '1') then
if(copy_crc_flag = '0') then
if(delay_bit = "00") then
DFF_inst <= '1'; DFF_inst1 <= '1';
     DFF_inst2 <= '1'; DFF_inst3 <= '1';
     DFF_inst4 <= '1'; DFF_inst5 <= '1';
     DFF_inst6 <= '1';
DFF_inst7 <= '1'; DFF_inst8 <= '1'; DFF_inst9 <=
      '1'; DFF_inst10 <= '1'; DFF_inst11 <= '1';
       DFF_inst12 <= '1'; DFF_inst13 <= '1';
DFF_inst14 <= '1'; DFF_inst15 <= '1';

delay_bit <= "01";
else
data_bit_out <= data;

DFF_inst <= SYNTHESIZED_WIRE_5; DFF_inst1 <=
     DFF_inst; DFF_inst2 <= DFF_inst1; DFF_inst3
      <= DFF_inst2; DFF_inst4 <= DFF_inst3;
DFF_inst5 <= SYNTHESIZED_WIRE_4; DFF_inst6 <=
     DFF_inst5; DFF_inst7 <= DFF_inst6;
     DFF_inst8 <= DFF_inst7; DFF_inst9 <=
     DFF_inst8;
DFF_inst10 <= DFF_inst9; DFF_inst11 <=
     DFF_inst10; DFF_inst12 <=
     SYNTHESIZED_WIRE_1; DFF_inst13 <=
     DFF_inst14;
DFF_inst14 <= DFF_inst15; DFF_inst15 <=
     DFF_inst12;
end if;
end if;

if(copy_crc_flag = '1') then
if(delay_bit = "01") then
data_bit_out <= data;

DFF_inst <= SYNTHESIZED_WIRE_5; DFF_inst1 <=
     DFF_inst; DFF_inst2 <= DFF_inst1; DFF_inst3
      <= DFF_inst2; DFF_inst4 <= DFF_inst3;
DFF_inst5 <= SYNTHESIZED_WIRE_4; DFF_inst6 <=
     DFF_inst5; DFF_inst7 <= DFF_inst6;
     DFF_inst8 <= DFF_inst7; DFF_inst9 <=
     DFF_inst8;
DFF_inst10 <= DFF_inst9; DFF_inst11 <=
     DFF_inst10; DFF_inst12 <=
     SYNTHESIZED_WIRE_1; DFF_inst13 <=
     DFF_inst14;
DFF_inst14 <= DFF_inst15; DFF_inst15 <=
     DFF_inst12;

delay_bit <= "10";
else
if(crc_flag = '1') then
if( count_16 < "10000") then
data_bit_out <= not(DFF_inst13); DFF_inst13 <=
     DFF_inst14; DFF_inst14 <= DFF_inst15;
DFF_inst15 <= DFF_inst12; DFF_inst12 <=
     DFF_inst11; DFF_inst11 <= DFF_inst10;
DFF_inst10 <= DFF_inst9; DFF_inst9 <= DFF_inst8;
      DFF_inst8 <= DFF_inst7;
DFF_inst7 <= DFF_inst6; DFF_inst6 <= DFF_inst5;
     DFF_inst5 <= DFF_inst4;
DFF_inst4 <= DFF_inst3; DFF_inst3 <= DFF_inst2;
     DFF_inst2 <= DFF_inst1;
DFF_inst1 <= DFF_inst;

count_16 <= count_16 + 1;
else
count_16 <="00000";
delay_bit <= "00";
start_crc_flag <= '0';
copy_crc_flag <= '0';
start_bit_flag <= '0';
data_bit_out <= 'Z';
done <= not(done_sgl);
done_sgl <= not(done_sgl);
DFF_inst <= '1'; DFF_inst1 <= '1';
     DFF_inst2 <= '1'; DFF_inst3 <= '1';
     DFF_inst4 <= '1'; DFF_inst5 <= '1';
     DFF_inst6 <= '1';
DFF_inst7 <= '1'; DFF_inst8 <= '1'; DFF_inst9 <=
      '1'; DFF_inst10 <= '1'; DFF_inst11 <= '1';
       DFF_inst12 <= '1'; DFF_inst13 <= '1';
DFF_inst14 <= '1'; DFF_inst15 <= '1';
end if;
else
done <= not(done_sgl);
done_sgl <= not(done_sgl);
delay_bit <= "00";
start_bit_flag <= '0';
data_bit_out <= 'Z';
copy_crc_flag <= '0';
DFF_inst <= '1'; DFF_inst1 <= '1';
     DFF_inst2 <= '1'; DFF_inst3 <= '1';
     DFF_inst4 <= '1'; DFF_inst5 <= '1';
     DFF_inst6 <= '1';
DFF_inst7 <= '1'; DFF_inst8 <= '1'; DFF_inst9 <=
      '1'; DFF_inst10 <= '1'; DFF_inst11 <= '1';
       DFF_inst12 <= '1'; DFF_inst13 <= '1';
DFF_inst14 <= '1'; DFF_inst15 <= '1';
end if;
end if;
end if;
end if;
end if;
if(start_crc'event) then
copy_crc_flag <= '1';
end if;
if(start_bit'event) then
start_bit_flag <= '1';
end if;
END PROCESS;

SYNTHESIZED_WIRE_5 <= DFF_inst13 XOR data;
SYNTHESIZED_WIRE_4 <= SYNTHESIZED_WIRE_5 XOR
     DFF_inst4;
SYNTHESIZED_WIRE_1 <= SYNTHESIZED_WIRE_5 XOR
     DFF_inst11;
END bdf_type;
```

Listing A.17. 4:1 Mux - 16bit data Selector

```
-- 4:1 Multiplexer for 16 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity mux_data16 is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic_vector(15 downto 0);
D_RES    : in std_logic_vector(15 downto 0);
D_TID    : in std_logic_vector(15 downto 0);
D_USER   : in std_logic_vector(15 downto 0);
mux_out: out std_logic_vector(15 downto 0)
);
end entity mux_data16;
architecture rtl_mux_data16 of mux_data16 is
begin
process(membnk_sel,clk)
begin
case membnk_sel is
when "01" =>
mux_out <= D_EPC;
when "00" =>
mux_out <= D_RES;
when "10" =>
mux_out <= D_TID;
when "11" =>
mux_out <= D_USER;
when others =>
end case;

end process;

end architecture;
```

Listing A.18. 2:1 Mux - EPC Address Selector

```
-- 2:1 Multiplexer for Start Address EPC
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_start_addr is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
start_address_Buffer: in std_logic_vector(23
    downto 0);
start_address_SM: in std_logic_vector(23 downto
    0);
mux_out : out std_logic_vector(23 downto 0)
);

end entity mux_start_addr;

architecture rtl_mux_start_addr of
    mux_start_addr is

begin
process(select_match_sel,clk)
begin
if(select_match_sel = '0') then
mux_out <= start_address_Buffer;
else
mux_out <= start_address_SM;
end if;
end process;

end architecture;
```

Listing A.19. 2:1 Mux - CRC Valid Selector

```
-- 2:1 Multiplexer for CRC
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_1bit_crc is
port(
clk     :  in std_logic;
crc_sel: in std_logic_vector(1 downto 0);
CRC_valid_5     : in std_logic;
CRC_valid_16: in std_logic;
mux_out : out std_logic
);
end entity mux_1bit_crc;
architecture rtl_mux_1bit_crc of mux_1bit_crc is
begin
process(clk,crc_sel)
begin
if(crc_sel = "10") then
mux_out <= CRC_valid_16;
elsif( (crc_sel = "00") or (crc_sel = "01") )
    then
mux_out <= CRC_valid_5;
end if;
end process;

end architecture;
```

Listing A.20. 2:1 Mux - Length Selector

```
-- 2:1 Multiplexer for data_16_bit flag
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity mux_length_16_bit is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
data_16_bit_buffer: in std_logic;
data_16_bit_SM: in std_logic;
mux_out : out std_logic
);
end entity mux_length_16_bit;
architecture rtl_mux_length_16_bit of
    mux_length_16_bit is
begin
process(select_match_sel,clk)
begin
if(select_match_sel = '0') then
mux_out <= data_16_bit_buffer;
else
mux_out <= data_16_bit_SM;
end if;
end process;
end architecture;
```

Listing A.21. 2:1 Mux - EPC Read Enable Selector

```
-- 2:1 Multiplexer for read enable flag
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_rd_en is
port(
clk   : in std_logic;
select_match_sel: in std_logic;
rd_en_buffer: in std_logic;
rd_en_SM: in std_logic;
mux_out : out std_logic
);

end entity mux_rd_en;

architecture rtl_mux_rd_en of mux_rd_en is

begin
process(select_match_sel,clk)
begin
if(select_match_sel = '0') then
mux_out <= rd_en_buffer;
else
mux_out <= rd_en_SM;
end if;
end process;

end architecture;
```

Listing A.22. 4:1 Mux - 1bit data Selector

```
-- 4:1 Multiplexer for 1 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_1bit is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);

end entity mux_1bit;

architecture rtl_mux_1bit of mux_1bit is

begin
```

```
process(membnk_sel,clk)
begin
case membnk_sel is
when "01" =>
mux_out <= D_EPC;
when "00" =>
mux_out <= D_RES;
when "10" =>
mux_out <= D_TID;
when "11" =>
mux_out <= D_USER;
when others =>
end case;

end process;

end architecture;
```

Listing A.23. 4:1 Mux - Done bit Selector

```
-- 4:1 Multiplexer for 1 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_1bit_done is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);
end entity mux_1bit_done;
architecture rtl_mux_1bit_done of mux_1bit_done
    is
begin
process(membnk_sel,clk)
begin
case membnk_sel is
when "01" =>
mux_out <= D_EPC;
when "00" =>
mux_out <= D_RES;
when "10" =>
mux_out <= D_TID;
when "11" =>
mux_out <= D_USER;
when others =>
end case;
end process;

end architecture;
```

Listing A.24. 4:1 Mux - Error bit Selector

```
-- 4:1 Multiplexer for 1 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_1bit_error is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);

end entity mux_1bit_error;

architecture rtl_mux_1bit_error of
    mux_1bit_error is

begin
process(membnk_sel,clk)
begin
case membnk_sel is
when "01" =>
mux_out <= D_EPC;
when "00" =>
mux_out <= D_RES;
when "10" =>
mux_out <= D_TID;
when "11" =>
mux_out <= D_USER;
when others =>
end case;

end process;

end architecture;
```

Listing A.25. 4:1 Mux - Stop bit Selector

```
-- 4:1 Multiplexer for 1 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_1bit_stop is
port(
clk   : in std_logic;
membnk_sel: in std_logic_vector(1 downto 0);
D_EPC    : in std_logic;
D_RES    : in std_logic;
D_TID    : in std_logic;
D_USER   : in std_logic;
mux_out : out std_logic
);

end entity mux_1bit_stop;

architecture rtl_mux_1bit_stop of mux_1bit_stop
    is

begin
process(membnk_sel,clk)
begin
case membnk_sel is
when "01" =>
mux_out <= D_EPC;
when "00" =>
mux_out <= D_RES;
when "10" =>
mux_out <= D_TID;
when "11" =>
mux_out <= D_USER;
when others =>
end case;

end process;

end architecture;
```

Listing A.26. Cipher Key Memory

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

entity key_rom is
port ( clk: in std_logic;--clock signal.
Preset: in std_logic;
key_access : in std_logic;
key_done : out std_logic;
data_key : out std_logic_vector(127 downto 0)
);
end entity key_rom;

architecture key_behavioral of key_rom is
```

```
type mem is array (0 to 0 ) of std_logic_vector
     (127 downto 0);
signal my_Rom : mem := (
0   => "127_bit_data"
);
begin
process (clk)
begin
IF (Preset = '0') THEN
key_done <= '0';
ELSIF (RISING_EDGE(clk)) THEN
if(key_access = '1') then
data_key <= my_rom(0) ;
key_done <= '1';
end if;
end if;
end process;
end architecture key_behavioral;
```

Listing A.27. AES - Key scheduling

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

entity key is
Port (
round_inc: in std_logic;
round_num_in : in std_logic_vector(0 to 3);
aes_key_in: in std_logic_vector(0 to 127);
clock: in std_logic;
reset: in std_logic;
crypt_en: in std_logic;
round_complete_out : out std_logic;
final_key_out: out std_logic_vector(0 to 127)
); --output block of data from this round
end key;

architecture rtl of key is
signal aes_key_out: std_logic_vector (0 to 127)
     :="127_0's";
signal round_num: unsigned (0 to 3):="0000";
signal round_complete: std_logic:='0';
signal round_complete_dup: std_logic:='1';
signal round_inc_dup: std_logic:='0';
--signal output_valid_I: std_logic;

type state is array (0 to 3) of std_logic_vector
     (0 to 7);
signal instate: state;
signal substate0,substate1,substate2,substate3:
     std_logic_vector(0 to 7);
signal NWord0,NWord1,NWord2,NWord3,Word1,Word2,
     Word3: std_logic_vector(0 to 31);
signal NWord3_dup: std_logic_vector(0 to 31);

--following is output of MixColumns() (in state
     format (dbyteROWCOLUMN)

subtype S_BOX_FIELD is integer range 0 to 255;
subtype SBOX_INDEX_TYPE is integer range 0 to
     15;
type SBOX_TYPE is array (0 to 255) of
     S_BOX_FIELD;

constant SBOXs : SBOX_TYPE := (
99, 124, 119, 123, 242, 107, 111, 197, 48, 1,
     103, 43, 254, 215, 171, 118,
202, 130, 201, 125,250, 89,
     71,240,173,212,162,175,156,164,114,192,
183, 253, 147, 38, 54, 63,247,204, 52, 165,
     229,241, 113,216, 49, 21,
4, 199, 35, 195, 24, 150, 5, 154, 7, 18,
     128,226,235, 39, 178, 117,
9, 131, 44, 26, 27, 110, 90, 160, 82, 59,214,
     179, 41, 227, 47, 132,
83, 209, 0,237, 32, 252, 177, 91, 106,203, 190,
     57, 74, 76, 88, 207,
208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2,
     127, 80, 60, 159, 168,
81, 163, 64, 143, 146, 157, 56, 245, 188, 182,
     218, 33, 16,255, 243, 210,
205, 12, 19,236, 95,151, 68, 23,196,167,126,
     61,100, 93, 25,115,
96, 129, 79,220, 34, 42,144,136, 70,238,184,
     20,222, 94, 11,219,
224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172,
     98, 145, 149,228, 121,
231, 200, 55, 109, 141,213, 78, 169, 108,
     86,244, 234, 101, 122, 174, 8,
186, 120, 37, 46, 28, 166, 180, 198, 232, 221,
     116, 31, 75, 189, 139, 138,
112, 62, 181, 102, 72, 3,246, 14, 97, 53, 87,
     185, 134, 193, 29, 158,
225, 248, 152, 17, 105, 217, 142, 148, 155, 30,
     135, 233, 206, 85, 40, 223,
140,161,137, 13,191,230, 66,104, 65,153, 45,
     15,176,84,187,22
);

function ESubBytes (inbyte: std_logic_vector(0
     to 7)) return std_logic_vector is
variable return_val: std_logic_vector(0 to 7);
begin
return conv_std_logic_vector(SBOXs(conv_integer(
     inbyte)),8);
end function;

begin

round_complete_out <= round_complete;
final_key_out <= aes_key_out;
process (clock)
begin
if(reset = '0') then
round_num <= "0000";
aes_key_out <= "127_0's";
round_complete <= '0';
round_complete_dup <= '1';
round_inc_dup <= '0';

elsif((crypt_en = '1') and (clock'event and
     clock = '1')) then

case round_num is

when "0000" =>

-- aes_key_in_R <= aes_key_in;

-- first four instate(0 to 3) is Rotword of
     WORD0
instate(0) <= aes_key_in(8 to 15);
instate(1) <= aes_key_in(16 to 23);
instate(2) <= aes_key_in(24 to 31);
instate(3) <= aes_key_in(0 to 7);

Word1 <= aes_key_in(32 to 39) & aes_key_in(40 to
      47) & aes_key_in(48 to 55) & aes_key_in(56
      to 63);

Word2 <= aes_key_in(64 to 71) & aes_key_in(72 to
      79) & aes_key_in(80 to 87) & aes_key_in(88
      to 95);

Word3 <= aes_key_in(96 to 103) & aes_key_in(104
     to 111) & aes_key_in(112 to 119) &
     aes_key_in(120 to 127);

--perform the SubBytes function on all bytes of
     the state.
substate0 <= ESubBytes(instate(0));
substate1 <= ESubBytes(instate(1));
substate2 <= ESubBytes(instate(2));
substate3 <= ESubBytes(instate(3));

aes_key_out <= aes_key_in;


when "0001" =>
NWord0 <= "00000001000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;
```

```
when "0010" =>
NWord0 <= "00000010000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "0011" =>
NWord0 <= "00000100000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "0100" =>
NWord0 <= "00001000000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "0101" =>
NWord0 <= "00010000000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "0110" =>
NWord0 <= "00100000000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "0111" =>
NWord0 <= "01000000000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "1000" =>
NWord0 <= "10000000000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "1001" =>
NWord0 <= "00011011000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when "1010" =>
NWord0 <= "00110110000000000000000000000000" xor
      (substate0 & substate1 & substate2 &
     substate3);
NWord1 <= NWord0 xor Word1;
NWord2 <= NWord1 xor Word2;
NWord3 <= NWord2 xor Word3;
aes_key_out <= NWord0 & NWord1 & NWord2 & NWord3
     ;
NWord3_dup <= NWord3;

when others =>

end case;

if ((aes_key_out = aes_key_in)  and (round_num =
      0)) then --(round_inc_dup = not(round_inc)
     )
round_complete <= not(round_complete);
round_num <= unsigned(round_num_in);
round_num <= round_num + 1;
end if;

if((NWord3_dup /= NWord3) and (round_num < 11))
     then
round_complete <= not(round_complete);
end if;

if (((round_inc_dup = not(round_inc)) and (
     round_complete_dup = not(round_complete))
     and (round_num /= 0))) then
round_inc_dup <= round_inc;
round_complete_dup <= round_complete;
round_num <= unsigned(round_num_in);
end if;

end if;
end process;
end rtl;
```

Listing A.28. AES - Mux

```
-- 4:1 Multiplexer for 16 bit data
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity aes_mux is
port(
round_num: in std_logic_vector(0 to 3);
aes_data_out_round0      : in std_logic_vector(0
     to 127);
aes_data_out_mid         : in std_logic_vector(0
     to 127);
plain_text       : in std_logic_vector(0 to 127);
aes_data: out std_logic_vector(0 to 127)
);

end entity aes_mux;

architecture rtl_aes_text of aes_mux is


begin
process(round_num,aes_data_out_round0,
     aes_data_out_mid,plain_text)
begin
if(round_num = "0000") then
aes_data <= plain_text;
elsif(round_num = "0001") then
aes_data <= aes_data_out_round0;
elsif( (round_num > "0001") and (round_num < "
     1011") ) then
aes_data <= aes_data_out_mid;
end if;
end process;

end architecture;
```

Listing A.29. AES - Cipher engine

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

entity aes_cipher is
Port (
aes_data_in : in std_logic_vector(0 to 127); --
    data block to encrpyt
aes_key_in: in std_logic_vector(0 to 127); --the
     key to use for this round
round_complete_in : in std_logic;
clock: in std_logic;
reset: in std_logic;
crypt_en : in std_logic;
round_num_key: out std_logic_vector(0 to 3);
round_num_cipher: out std_logic_vector(0 to 3);
round_inc: out std_logic;
cipher_done : out std_logic;
aes_data_out_mid,aes_data_out_round0 : out
    std_logic_vector(0 to 127);
aes_data_out_final : out std_logic_vector(0 to
    127)
); --output block of data from this round
end aes_cipher;

architecture rtl of aes_cipher is
signal round_inc_dup: std_logic:='0';
signal round_complete_in_dup: std_logic:='0';
signal aes_key_in_R: std_logic_vector (0 to 127)
    ;
signal round_num_R: unsigned (0 to 3);
signal aes_data,mix_data_out_dup:
    std_logic_vector (0 to 127);
signal round_num_key_dup: unsigned(0 to 3):="
    0000";
--signal output_valid_I: std_logic;

type state is array (0 to 15) of
    std_logic_vector(0 to 7);
signal instate: state;
signal substate: state;
signal shiftstate: state;
signal shiftstate_2: state;
signal outmixState: state;

signal shift_data_out: std_logic_vector(0 to
    127);
signal mix_data_out: std_logic_vector(0 to 127);
signal last_round_out: std_logic_vector(0 to
    127);

--following is output of MixColumns() (in state
    format (dbyteROWCOLUMN)
signal OutMixByte00: std_logic_vector(0 to 7);
signal OutMixByte01: std_logic_vector(0 to 7);
signal OutMixByte02: std_logic_vector(0 to 7);
signal OutMixByte03: std_logic_vector(0 to 7);

signal OutMixByte10: std_logic_vector(0 to 7);
signal OutMixByte11: std_logic_vector(0 to 7);
signal OutMixByte12: std_logic_vector(0 to 7);
signal OutMixByte13: std_logic_vector(0 to 7);

signal OutMixByte20: std_logic_vector(0 to 7);
signal OutMixByte21: std_logic_vector(0 to 7);
signal OutMixByte22: std_logic_vector(0 to 7);
signal OutMixByte23: std_logic_vector(0 to 7);

signal OutMixByte30: std_logic_vector(0 to 7);
signal OutMixByte31: std_logic_vector(0 to 7);
signal OutMixByte32: std_logic_vector(0 to 7);
signal OutMixByte33: std_logic_vector(0 to 7);

subtype S_BOX_FIELD is integer range 0 to 255;
subtype SBOX_INDEX_TYPE is integer range 0 to
    15;
type SBOX_TYPE is array (0 to 255) of
    S_BOX_FIELD;

constant SBOXs : SBOX_TYPE := (
99, 124, 119, 123, 242, 107, 111, 197, 48, 1,
    103, 43, 254, 215, 171, 118,
202, 130, 201, 125,250, 89,
    71,240,173,212,162,175,156,164,114,192,
183, 253, 147, 38, 54, 63,247,204, 52, 165,
    229,241, 113,216, 49, 21,
4, 199, 35, 195, 24, 150, 5, 154, 7, 18,
    128,226,235, 39, 178, 117,
9, 131, 44, 26, 27, 110, 90, 160, 82, 59,214,
    179, 41, 227, 47, 132,
83, 209, 0,237, 32, 252, 177, 91, 106,203, 190,
    57, 74, 76, 88, 207,
208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2,
     127, 80, 60, 159, 168,
81, 163, 64, 143, 146, 157, 56, 245, 188, 182,
    218, 33, 16,255, 243, 210,
205, 12, 19,236, 95,151, 68, 23,196,167,126,
    61,100, 93, 25,115,
96, 129, 79,220, 34, 42,144,136, 70,238,184,
    20,222, 94, 11,219,
224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172,
    98, 145, 149,228, 121,
231, 200, 55, 109, 141,213, 78, 169, 108,
    86,244, 234, 101, 122, 174, 8,
186, 120, 37, 46, 28, 166, 180, 198, 232, 221,
    116, 31, 75, 189, 139, 138,
112, 62, 181, 102, 72, 3,246, 14, 97, 53, 87,
    185, 134, 193, 29, 158,
225, 248, 152, 17, 105, 217, 142, 148, 155, 30,
    135, 233, 206, 85, 40, 223,
140,161,137, 13,191,230, 66,104, 65,153, 45,
    15,176,84,187,22
);

function ESubBytes (inbyte: std_logic_vector(0
    to 7)) return std_logic_vector is
variable return_val: std_logic_vector(0 to 7);
begin
return conv_std_logic_vector(SBOXs(conv_integer(
    inbyte)),8);
end function;

function DubBytes (inbyte: std_logic_vector(0 to
     7)) return std_logic_vector is
begin
case inbyte(0) is
when '0' =>
return inbyte(1 to 7) & '0';
when '1' =>
return ((inbyte(1 to 7) & '0') xor "00011011");
when others =>
return "00000000";
end case;
end function;

begin

aes_data <= aes_data_in;
--aes_key_in_R <= aes_key_in;
--round_num_R <= round_num;

--place the input data in the state, as defined
    in the AES spec.
instate(0) <= aes_data(0 to 7);
instate(1) <= aes_data(8 to 15);
instate(2) <= aes_data(16 to 23);
instate(3) <= aes_data(24 to 31);

instate(4) <= aes_data(32 to 39);
instate(5) <= aes_data(40 to 47);
instate(6) <= aes_data(48 to 55);
instate(7) <= aes_data(56 to 63);

instate(8) <= aes_data(64 to 71);
instate(9) <= aes_data(72 to 79);
instate(10) <= aes_data(80 to 87);
instate(11) <= aes_data(88 to 95);

instate(12) <= aes_data(96 to 103);
instate(13) <= aes_data(104 to 111);
instate(14) <= aes_data(112 to 119);
instate(15) <= aes_data(120 to 127);

--perform the SubBytes function on all bytes of
    the state.
substate(0) <= ESubBytes(instate(0));
substate(1) <= ESubBytes(instate(1));
substate(2) <= ESubBytes(instate(2));
substate(3) <= ESubBytes(instate(3));
```

```
substate(4) <= ESubBytes(instate(4));
substate(5) <= ESubBytes(instate(5));
substate(6) <= ESubBytes(instate(6));
substate(7) <= ESubBytes(instate(7));

substate(8) <= ESubBytes(instate(8));
substate(9) <= ESubBytes(instate(9));
substate(10) <= ESubBytes(instate(10));
substate(11) <= ESubBytes(instate(11));

substate(12) <= ESubBytes(instate(12));
substate(13) <= ESubBytes(instate(13));
substate(14) <= ESubBytes(instate(14));
substate(15) <= ESubBytes(instate(15));

--perform the ShiftRows function on all rows of
    the state.
shiftstate(0) <= substate(0);
shiftstate(4) <= substate(4);
shiftstate(8) <= substate(8);
shiftstate(12) <= substate(12);

shiftstate(1) <= substate(5);
shiftstate(5) <= substate(9);
shiftstate(9) <= substate(13);
shiftstate(13) <= substate(1);

shiftstate(2) <= substate(10);
shiftstate(6) <= substate(14);
shiftstate(10) <= substate(2);
shiftstate(14) <= substate(6);

shiftstate(3) <= substate(15);
shiftstate(7) <= substate(3);
shiftstate(11) <= substate(7);
shiftstate(15) <= substate(11);

--for the last round, the output is the xor of
    the state after shiftrows, and the key,
--so create the last round output word
shift_data_out(0 to 7) <= shiftstate(0);
shift_data_out(8 to 15) <= shiftstate(1);
shift_data_out(16 to 23) <= shiftstate(2);
shift_data_out(24 to 31) <= shiftstate(3);

shift_data_out(32 to 39) <= shiftstate(4);
shift_data_out(40 to 47) <= shiftstate(5);
shift_data_out(48 to 55) <= shiftstate(6);
shift_data_out(56 to 63) <= shiftstate(7);

shift_data_out(64 to 71) <= shiftstate(8);
shift_data_out(72 to 79) <= shiftstate(9);
shift_data_out(80 to 87) <= shiftstate(10);
shift_data_out(88 to 95) <= shiftstate(11);

shift_data_out(96 to 103) <= shiftstate(12);
shift_data_out(104 to 111) <= shiftstate(13);
shift_data_out(112 to 119) <= shiftstate(14);
shift_data_out(120 to 127) <= shiftstate(15);

--for mixcolumns, we need to take 1, 2, and 3
    times various bytes in the columns
--the following creates the x2. x3 is the xor of
     x2 and the original (xl) value.
shiftstate_2(0) <= DubBytes(substate(0));
shiftstate_2(4) <= DubBytes(substate(4));
shiftstate_2(8) <= DubBytes(substate(8));
shiftstate_2(12) <= DubBytes(substate(12));

shiftstate_2(1) <= DubBytes(substate(5));
shiftstate_2(5) <= DubBytes(substate(9));
shiftstate_2(9) <= DubBytes(substate(13));
shiftstate_2(13) <= DubBytes(substate(1));

shiftstate_2(2) <= DubBytes(substate(10));
shiftstate_2(6) <= DubBytes(substate(14));
shiftstate_2(10) <= DubBytes(substate(2));
shiftstate_2(14) <= DubBytes(substate(6));

shiftstate_2(3) <= DubBytes(substate(15));
shiftstate_2(7) <= DubBytes(substate(3));
shiftstate_2(11) <= DubBytes(substate(7));
shiftstate_2(15) <= DubBytes(substate(11));

--Following groups perform the MixColumns
    operation, as defined in the AES standard
--OutMixByte00, OutMixByte1e0, OutMixByte20,
    OutMixByte30
OutMixByte00 <= (shiftstate(2) xor shiftstate(3)
     xor shiftstate_2(0) xor (shiftstate_2(1)
    xor shiftstate(1)));
OutMixByte10 <= (shiftstate(0) xor shiftstate(3)
     xor shiftstate_2(1) xor (shiftstate_2(2)
    xor shiftstate(2)));
OutMixByte20 <= (shiftstate(0) xor shiftstate(1)
     xor shiftstate_2(2) xor (shiftstate_2(3)
    xor shiftstate(3)));
OutMixByte30 <= (shiftstate(1) xor shiftstate(2)
     xor shiftstate_2(3) xor (shiftstate_2(0)
    xor shiftstate(0)));

--OutMixByte0l, OutMixByte1e1 OutMixByte2l,
    OutMixByte3l
OutMixByte01 <= (shiftstate(6) xor shiftstate(7)
     xor shiftstate_2(4) xor (shiftstate_2(5)
    xor shiftstate(5)));
OutMixByte11 <= (shiftstate(4) xor shiftstate(7)
     xor shiftstate_2(5) xor (shiftstate_2(6)
    xor shiftstate(6)));
OutMixByte21 <= (shiftstate(4) xor shiftstate(5)
     xor shiftstate_2(6) xor (shiftstate_2(7)
    xor shiftstate(7)));
OutMixByte31 <= (shiftstate(5) xor shiftstate(6)
     xor shiftstate_2(7) xor (shiftstate_2(4)
    xor shiftstate(4)));

OutMixByte02 <= (shiftstate(10) xor shiftstate
    (11) xor shiftstate_2(8) xor (shiftstate_2
    (9) xor shiftstate(9)));
OutMixByte12 <= (shiftstate(8) xor shiftstate
    (11) xor shiftstate_2(9) xor (shiftstate_2
    (10) xor shiftstate(10)));
OutMixByte22 <= (shiftstate(8) xor shiftstate(9)
     xor shiftstate_2(10) xor (shiftstate_2(11)
     xor shiftstate(11)));
OutMixByte32 <= (shiftstate(9) xor shiftstate
    (10) xor shiftstate_2(11) xor (shiftstate_2
    (8) xor shiftstate(8)));
--OutMixByte03, OutMixByte1e3, OutMixByte23,
    OutMixByte33
OutMixByte03 <= (shiftstate(14) xor shiftstate
    (15) xor shiftstate_2(12) xor (shiftstate_2
    (13) xor shiftstate(13)));
OutMixByte13 <= (shiftstate(12) xor shiftstate
    (15) xor shiftstate_2(13) xor (shiftstate_2
    (14) xor shiftstate(14)));
OutMixByte23 <= (shiftstate(12) xor shiftstate
    (13) xor shiftstate_2(14) xor (shiftstate_2
    (15) xor shiftstate(15)));
OutMixByte33 <= (shiftstate(13) xor shiftstate
    (14) xor shiftstate_2(15) xor (shiftstate_2
    (12) xor shiftstate(12)));

--following is the output for rounds 1-9 of the
    cipher
mix_data_out(0 to 7) <= OutMixByte00;
mix_data_out(8 to 15) <= OutMixByte10;
mix_data_out(16 to 23) <= OutMixByte20;
mix_data_out(24 to 31) <= OutMixByte30;

mix_data_out(32 to 39) <= OutMixByte01;
mix_data_out(40 to 47) <= OutMixByte11;
mix_data_out(48 to 55) <= OutMixByte21;
mix_data_out(56 to 63) <= OutMixByte31;

mix_data_out(64 to 71) <= OutMixByte02;
mix_data_out(72 to 79) <= OutMixByte12;
mix_data_out(80 to 87) <= OutMixByte22;
mix_data_out(88 to 95) <= OutMixByte32;

mix_data_out(96 to 103) <= OutMixByte03;
mix_data_out(104 to 111) <= OutMixByte13;
mix_data_out(112 to 119) <= OutMixByte23;
mix_data_out(120 to 127) <= OutMixByte33;
mix_data_out_dup <= mix_data_out;
round_num_cipher <= std_logic_vector(round_num_R
    );

process (clock)
begin
if(reset = '0') then
round_num_R <= "0000";
cipher_done <= '0';
```

```
round_inc_dup <= '0';
round_complete_in_dup <= '0';
round_num_key_dup <= "0000";

elsif ((crypt_en = '1') and (clock'event and
    clock = '1')) then
if (round_num_R = 0) then
-- round_inc <= not(round_inc_dup);
-- round_inc_dup <= not(round_inc_dup);
if(round_complete_in_dup = not(round_complete_in
    )) then
round_complete_in_dup <= round_complete_in;
aes_data_out_round0 <= aes_data xor aes_key_in;
round_num_key <= round_num_key_dup + 1;
round_num_key_dup <= round_num_key_dup + 1;
round_num_R <= round_num_R + 1;
end if;
elsif((round_num_R > 0) and (round_num_R < 10))
    then
if(round_complete_in_dup = not(round_complete_in
    )) then
round_complete_in_dup <= round_complete_in;
aes_data_out_mid <= mix_data_out xor aes_key_in;
round_inc <= not(round_inc_dup);
round_inc_dup <= not(round_inc_dup);
round_num_key <= round_num_key_dup + 1;
round_num_key_dup <= round_num_key_dup + 1;
round_num_R <= round_num_R + 1;
end if;
elsif(round_num_R = 10) then
aes_data_out_final <= shift_data_out xor
    aes_key_in;
round_num_R <= round_num_R + 1;
cipher_done <= '1';
else
cipher_done <= '0';
end if;
end if;
end process;



-- aes_data_out_round0 <= aes_data xor
    aes_key_in_R;
-- aes_data_out_mid <= mix_data_out xor
    aes_key_in_R;
-- aes_data_out_final <= shift_data_out xor
    aes_key_in_R;
end rtl;
```

Listing A.30. AES - Top Level Entity

```
-- TOP MODULE
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity aes_top is
port(
clk     : in std_logic;
reset: in std_logic;
crypt_en        : in std_logic;
preset_engine: in std_logic;
text_in: in std_logic_vector(0 to 127);
key_in: in std_logic_vector(0 to 127);
cipher_out1: out std_logic_vector(0 to 127);
done: out std_logic
);

end entity aes_top;

architecture rtl_aes of aes_top is

--Signal Declarations here:
signal round_num_sgl,round_num_key_sgl:
    std_logic_vector(0 to 3);
signal aes_data_out_round0_sgl,
    aes_data_out_mid_sgl,aes_data_sgl,
    aes_key_in_sgl: std_logic_vector(0 to 127);
signal round_complete_in_sgl,round_inc_sgl,
    n_reset: std_logic;

--Component Declarations here:
component aes_mux is
port(
round_num: in std_logic_vector(0 to 3);
aes_data_out_round0     : in std_logic_vector(0
    to 127);
aes_data_out_mid        : in std_logic_vector(0
    to 127);
plain_text      : in std_logic_vector(0 to 127);
aes_data: out std_logic_vector(0 to 127)
);

end component aes_mux;

component aes_cipher is
Port (
aes_data_in : in std_logic_vector(0 to 127); --
    data block to encrpyt
aes_key_in: in std_logic_vector(0 to 127); --the
     key to use for this round
round_complete_in : in std_logic;
clock: in std_logic;
crypt_en        : in std_logic;
reset: in std_logic;
round_num_key: out std_logic_vector(0 to 3);
round_num_cipher: out std_logic_vector(0 to 3);
round_inc: out std_logic;
cipher_done : out std_logic;
aes_data_out_mid,aes_data_out_round0 : out
    std_logic_vector(0 to 127);
aes_data_out_final : out std_logic_vector(0 to
    127)
); --output block of data from this round
end component aes_cipher;


component key is
Port (
round_inc: in std_logic;
round_num_in : in std_logic_vector(0 to 3);
aes_key_in: in std_logic_vector(0 to 127);
clock: in std_logic;
reset: in std_logic;
crypt_en        : in std_logic;
round_complete_out : out std_logic;
final_key_out: out std_logic_vector(0 to 127)
); --output block of data from this round
end component key;

begin

muxer_1: aes_mux
port map(
round_num => round_num_sgl,
aes_data_out_round0     =>
    aes_data_out_round0_sgl,
aes_data_out_mid => aes_data_out_mid_sgl,
plain_text => text_in,
aes_data =>     aes_data_sgl
);

cipher: aes_cipher
port map(
aes_data_in => aes_data_sgl,
aes_key_in => aes_key_in_sgl,
round_complete_in => round_complete_in_sgl,
clock => clk,
reset => n_reset,
crypt_en => crypt_en,
round_num_key => round_num_key_sgl,
round_num_cipher => round_num_sgl,
round_inc => round_inc_sgl,
aes_data_out_round0     =>
    aes_data_out_round0_sgl,
aes_data_out_mid => aes_data_out_mid_sgl,
cipher_done => done,
aes_data_out_final => cipher_out1
);
aes_key: key
port map(
round_inc => round_inc_sgl,
round_num_in => round_num_key_sgl,
aes_key_in => key_in,
clock => clk,
reset => n_reset,
crypt_en => crypt_en,
```

```
round_complete_out => round_complete_in_sgl,
final_key_out => aes_key_in_sgl
);

n_reset <=      crypt_en and preset_engine;

end architecture;
```

Listing A.31. XTEA - 1:2 Demux

```
-- Right Shift by 5
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity demux2 is
port(
sel: in std_logic;
D1      : in std_logic_vector(31 downto 0);
Out_1: out std_logic_vector(31 downto 0);
Out_2: out std_logic_vector(31 downto 0)
);
end entity demux2;
architecture rtl_demux of demux2 is
begin
process(sel,D1)
begin
if (sel='1') then
Out_1<= D1;
else
Out_2<=D1;
end if;
end process;

end architecture;
```

Listing A.32. XTEA - Counter for rounds

```
--Counter Module.
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity counter is
port(
clk     : in std_logic;
n_reset : in std_logic;
q       : out std_logic_vector(4 downto 0)
);
end entity counter;
architecture dfl of counter  is
signal count_reg : unsigned(4 downto 0);
begin
process(clk)
begin
if (n_reset = '0') then
count_reg <= (others => '0'); -- initialisation
else
if(clk'event and clk = '1') then
count_reg <= count_reg + 1;
end if;
end if;
end process;
q <= std_logic_vector(count_reg);
end architecture dfl;
```

Listing A.33. XTEA - Sum calculation

```
-- Sum = Sum + Delta   II
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity lut_sum is
port(
clk: in std_logic;
n_reset: in std_logic;
address_in      : in std_logic_vector(4 downto
    0);
k0,k1,k2,k3: in std_logic_vector(31 downto 0);
data_out0: out std_logic_vector(31 downto 0);
data_out1: out std_logic_vector(31 downto 0)
);
end entity lut_sum;
architecture rtl_lut of lut_sum is
begin
lut_process: process(clk, n_reset)
begin
if n_reset='0' then
data_out0<= "00000000000000000000000000000000";
data_out1<= "00000000000000000000000000000000";
elsif (clk'event and clk='1') then
address_cases: case address_in is
when "00000" => data_out1<= "
    10011110001101110111100110111001" + k3;
    data_out0<= "
    00000000000000000000000000000000" + k0;
when "00001" => data_out1<= "
    00111100011011101111001101110010" + k2;
    data_out0<= "
    10011110001101110111100110111001" + k1;
when "00010" => data_out1<= "
    11011010101001100110110100101011"+ k1;
    data_out0<= "
    00111100011011101111001101110010" + k2;
when "00011" => data_out1<= "
    01111000110111011110011011100100"+ k0;
    data_out0<= "
    11011010101001100110110100101011" + k3;
when "00100" => data_out1<=     "
    00010111000101010110000010011101"+ k0;
    data_out0<= "
    01111000110111011110011011100100" + k0;
when "00101" => data_out1<= "
    10110101010011001101101001010110"+ k1;
    data_out0<= "
    00010111000101010110000010011101" + k1;
when "00110" => data_out1<= "
    01010011100001000101010000001111"+ k2;
    data_out0<= "
    10110101010011001101101001010110" + k2;
when "00111" => data_out1<= "
    11110001101110111100110111001000"+ k1;
    data_out0<= "
    01010011100001000101010000001111" + k3;
when "01000" => data_out1<= "
    10001111111100110100011110000001"+ k0;
    data_out0<= "
    11110001101110111100110111001000" + k0;
when "01001" => data_out1<= "
    00101110001010101100000100111010"+ k0;
    data_out0<= "
    10001111111100110100011110000001" + k1;
when "01010" => data_out1<= "
    11001100011000100011101011110011"+ k3;
    data_out0<= "
    00101110001010101100000100111010" + k2;
when "01011" => data_out1<= "
    01101010100110011011010010101100"+ k2;
    data_out0<= "
    11001100011000100011101011110011" + k3;
when "01100" => data_out1<= "
    00001000110100010010111001100101"+ k1;
    data_out0<= "
    01101010100110011011010010101100" + k0;
when "01101" => data_out1<= "
    10100111000010001010100000011110"+ k1;
    data_out0<= "
    00001000110100010010111001100101" + k1;
when "01110" => data_out1<= "
    01000101010000000010000111010111"+ k0;
    data_out0<= "
    10100111000010001010100000011110" + k2;
when "01111" => data_out1<= "
    11100011011101111001101110010000"+ k3;
    data_out0<= "
    01000101010000000010000111010111" + k3;
when "10000" => data_out1<= "
    10000001101011110001010101001001"+ k2;
```

```
      data_out0<= "
      11100011011011100110110010000" + k0;
when "10001" => data_out1<= "
      00011111111001101000111100000010"+ k1;
      data_out0<= "
      10000001101011110001010101001001" + k1;
when "10010" => data_out1<= "
      10111110000111100000100010111011"+ k1;
      data_out0<= "
      00011111111001101000111100000010" + k2;
when "10011" => data_out1<= "
      01011100010101011000001001110100"+ k0;
      data_out0<= "
      10111110000111100000100010111011" + k3;
-----
when "10100" => data_out1<= "
      11111010100011001111110000101101"+ k3;
      data_out0<= "
      01011100010101011000001001110100" + k0;
when "10101" => data_out1<= "
      10011000110001000111010111100110"+ k2;
      data_out0<= "
      11111010100011001111110000101101" + k1;
when "10110" => data_out1<= "
      00110110111110111110111110011111"+ k1;
      data_out0<= "
      10011000110001000111010111100110" + k2;
when "10111" => data_out1<= "
      11010101001100110110100101011000"+ k1;
      data_out0<= "
      00110110111110111110111110011111" + k3;
when "11000" => data_out1<= "
      01110011011010101110001100010001"+ k0;
      data_out0<= "
      11010101001100110110100101011000" + k0;
when "11001" => data_out1<= "
      00010001101000100101110011001010"+ k3;
      data_out0<= "
      01110011011010101110001100010001" + k1;
when "11010" => data_out1<= "
      10101111110110011101011010000011"+ k2;
      data_out0<= "
      00010001101000100101110011001010" + k2;
when "11011" => data_out1<= "
      01001110000100010101000000111100"+ k2;
      data_out0<= "
      10101111110110011101011010000011" + k3;
when "11100" => data_out1<= "
      11101100010010001100100111110101"+ k1;
      data_out0<= "
      01001110000100010101000000111100" + k0;
when "11101" => data_out1<= "
      10001010100000000100001110101110"+ k0;
      data_out0<= "
      11101100010010001100100111110101" + k1;
when "11110" => data_out1<= "
      00101000101101111011101011001111"+ k3;
      data_out0<= "
      10001010100000000100001110101110" + k2;
when "11111" => data_out1<= "
      11000110111011110011011100100000"+ k2;
      data_out0<= "
      00101000101101111011101011001111" + k3;
when others => data_out1<= "
      00000000000000000000000000000000";
      data_out0<= "
      00000000000000000000000000000000";
end case;
end if;
end process;
end architecture;
```

Listing A.34. XTEA - Logic Unit

```
-- XTEA Logic unit: a0= a0 + (((a14 ^ a15) +a1)
     xor lut_sum)
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity logic_unit is
port(
clk      : in std_logic;
n_reset: in std_logic;
a0       : in std_logic_vector(31 downto 0);
a1       : in std_logic_vector(31 downto 0);
lut_sum : in std_logic_vector(31 downto 0);
count: in std_logic_vector(4 downto 0);
a0_out: out std_logic_vector(31 downto 0);
a_initial_flag: out std_logic;
done: out std_logic
);
end entity logic_unit;
architecture rtl_logic of logic_unit is
signal a14,a15: std_logic_vector(31 downto 0);
begin
a15<= a1(4 downto 0)&a1(31 downto 5); --rt shift
      by 5
a14<= a1(27 downto 0)&a1(31 downto 28); --lf
     shift by 4
process(clk)
begin
if n_reset='0' then
a0_out<="00000000000000000000000000000000";
done<='0';
a_initial_flag<= '0';
else
if (clk'event and clk='1') then
a0_out<= a0 + (((a14 xor a15) + a1) xor lut_sum)
     ;
if count="11111" then
done<='1';
else
done<= '0';
end if;
if count="00001" then
a_initial_flag <='1';
else
a_initial_flag<= '0';
end if;
end if;
end if;
end process;
end architecture;
```

Listing A.35. XTEA - 2:1 Mux

```
-- 2:1 Multiplexer
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux2 is
port(
sel: in std_logic;
D1       : in std_logic_vector(31 downto 0);
D2       : in std_logic_vector(31 downto 0);
mux_out: out std_logic_vector(31 downto 0)
);
end entity mux2;
architecture rtl_mux of mux2 is
begin
process(sel,D1,D2)
begin
if (sel='1') then
mux_out<= D2;
else
mux_out<=D1;
end if;
end process;
end architecture;
```

Listing A.36. XTEA - Top Level Entity

```
-- XTEA TOP MODULE- ENCRYPTION
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tea_top is
port(
clk      : in std_logic;
n_reset: in std_logic;
crypt_en         : in std_logic;
preset_engine: in std_logic;
text_in: in std_logic_vector(63 downto 0);
key_in: in std_logic_vector(127 downto 0);
cipher_out: out std_logic_vector(63 downto 0);
done: out std_logic
);
end entity tea_top;
architecture rtl_tea of tea_top is
--Signal Declarations here:
signal count: std_logic_vector(4 downto 0);
signal flag_sig1,flag_sig2,sel_done,a_ini_flag1,
    a_ini_flag2,preset: std_logic;
signal a0,a1,lut_sum_0,lut_sum_1,a0_dmux:
    std_logic_vector(31 downto 0);
signal k0,k1,k2,k3: std_logic_vector(31 downto
    0);
signal a1_dmux: std_logic_vector(31 downto 0);
signal a0_initial,a1_initial,cipher_sig1,
    cipher_sig2,a1_new,a0_new:std_logic_vector
    (31 downto 0);
--Component Declarations here:
component mux2 is
port(
sel: in std_logic;
D1       : in std_logic_vector(31 downto 0);
D2       : in std_logic_vector(31 downto 0);
mux_out: out std_logic_vector(31 downto 0)
);
end component mux2;
component counter is
port(
clk      : in std_logic;
n_reset  : in std_logic;
q        : out std_logic_vector(4 downto 0)
);
end component counter;
component lut_sum is
port(
clk: in std_logic;
n_reset: in std_logic;
address_in        : in std_logic_vector(4 downto
    0);
k0,k1,k2,k3: in std_logic_vector(31 downto 0);
data_out0: out std_logic_vector(31 downto 0);
data_out1: out std_logic_vector(31 downto 0)
);
end component lut_sum;
component logic_unit is
port(
clk      : in std_logic;
n_reset: in std_logic;
a0         : in std_logic_vector(31 downto 0);
a1         : in std_logic_vector(31 downto 0);
lut_sum  : in std_logic_vector(31 downto 0);
count: in std_logic_vector(4 downto 0);
a0_out: out std_logic_vector(31 downto 0);
done: out std_logic;
a_initial_flag: out std_logic
);
end component logic_unit;
component demux2 is
port(
sel: in std_logic;
D1       : in std_logic_vector(31 downto 0);
Out_1: out std_logic_vector(31 downto 0);
Out_2: out std_logic_vector(31 downto 0)
);
end component demux2;
begin
a0_initial<= text_in(63 downto 32);
a1_initial<= text_in(31 downto 0);
k0<= key_in(127 downto 96);
k1<= key_in(95 downto 64);
k2<= key_in(63 downto 32);
k3<= key_in(31 downto 0);
muxer_1: mux2
port map(
sel=>sel_done,
D1       => a0_initial,
D2       => a0_new,
mux_out=>a0
);
muxer_2: mux2
port map(
sel=>sel_done,
D1       => a1_initial,
D2       => a1_new,
mux_out=>a1
);
Counter_module: counter
port map(
clk      => clk,
n_reset => preset,
q        => count
);
LUT_SUM_MODULE: lut_sum
port map(
clk=>clk,
n_reset=>preset,
address_in => count,
k0=>k0,
k1=>k1,
k2=>k2,
k3=>k3,
data_out0=> lut_sum_0,
data_out1=> lut_sum_1
);
logic_unit_module_1: logic_unit
port map(
clk      =>clk,
n_reset=>preset,
a0        =>a0,
a1        =>a1,
lut_sum =>lut_sum_0,
count=>count,
a0_out=>a0_dmux,
done=>flag_sig1,
a_initial_flag=> a_ini_flag1
);
logic_unit_module_2: logic_unit
port map(
clk      =>clk,
n_reset=> preset,
a0        =>a1,
a1        =>a0_dmux,
lut_sum =>lut_sum_1,
count=>count,
a0_out=>a1_dmux,
done=>flag_sig2,
a_initial_flag=> a_ini_flag2
);
demultiplexer_1: demux2
port map(
sel=>flag_sig1,
D1       => a0_dmux,
Out_2=> a0_new,
Out_1=> cipher_sig1
);
demultiplexer_2: demux2
port map(
sel=>flag_sig2,
D1       => a1_dmux,
Out_2=> a1_new,
Out_1=> cipher_sig2
);
cipher_out <= cipher_sig1 & cipher_sig2;
sel_done<= NOT(flag_sig2 or a_ini_flag1 or
    a_ini_flag2);
done<= flag_sig2;
preset <= crypt_en and preset_engine;

end architecture;
```

Listing A.37. Present - Counter for rounds

```
--asynchron reset
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
port(
```

```
clk      : in std_logic;
n_reset,crypt_en: in std_logic;
q        : out std_logic_vector(4 downto 0)
);

end entity counter;

architecture dfl of counter  is

signal count_reg : unsigned(4 downto 0);
begin

process(clk)
begin
if (n_reset = '0') then
count_reg <= (others => '0'); -- initialisation
else
if((clk'event and clk = '1') and (crypt_en =
    '1'))then
count_reg <= count_reg + 1;
end if;
end if;
end process;

q <= std_logic_vector(count_reg+1);

end architecture dfl;
```

Listing A.38. Present - 64bit DFF component

```
-- D-Flip-Flop
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dflipflop_2 is
port(
clk      : in std_logic;
crypt_en : in std_logic;
D        : in std_logic_vector(63 downto 0);
Q        : out std_logic_vector(63 downto 0)
);

end entity dflipflop_2;

architecture dfl of dflipflop_2 is

signal s_current_state, s_next_state     :
    std_logic_vector(63 downto 0);

begin
s_next_state    <= d;
q <= s_current_state;

FLIP_FLOP:      Process(clk, s_next_state)
begin
if((clk'event and clk = '1') and (crypt_en =
    '1'))then
s_current_state <= s_next_state;
end if;
end process;

end architecture;
```

Listing A.39. Present - 128bit DFF component

```
-- D-Flip-Flop 128
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dflipflop128 is
port(
clk      : in std_logic;
D        : in std_logic_vector(127 downto 0);
crypt_en : in std_logic;
Q        : out std_logic_vector(127 downto 0)
);

end entity dflipflop128;

architecture dfl of dflipflop128 is

signal s_current_state, s_next_state     :
    std_logic_vector(127 downto 0);

begin
s_next_state    <= d;
q <= s_current_state;

FLIP_FLOP:      Process(clk, s_next_state)
begin
if((clk'event and clk = '1') and (crypt_en =
    '1'))then
s_current_state <= s_next_state;
end if;
end process;

end architecture;
```

Listing A.40. Present - 64bit DFF

```
--Template for FlipFlops
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity flipflop is
port(
clk              : in std_logic;
sel      : in std_logic;
crypt_en : in std_logic;
D0       : in std_logic_vector(63 downto 0);
D1       : in std_logic_vector(63 downto 0);
Q        : out std_logic_vector(63 downto 0)
);

end entity flipflop;

architecture dfl of flipflop is

component dflipflop_2 is
port(
clk      : in std_logic;
crypt_en : in std_logic;
D        : in std_logic_vector(63 downto 0);
Q        : out std_logic_vector(63 downto 0)
);

end component dflipflop_2 ;

signal ff_in:   std_logic_vector(63 downto 0);

begin

DFF: dflipflop_2
port map(
clk => clk,
crypt_en => crypt_en,
D => ff_in,
Q => Q
);

ff_in <= D0 when sel = '0'
else D1;
end architecture dfl;
```

Listing A.41. Present - 128bit DFF

```
--Template for FlipFlops 128
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity flipflop128 is
port(
clk             : in std_logic;
sel     : in std_logic;
crypt_en : in std_logic;
D0      : in std_logic_vector(127 downto 0);
D1      : in std_logic_vector(127 downto 0);
Q       : out std_logic_vector(127 downto 0)
);

end entity flipflop128;

architecture dfl of flipflop128 is

component dflipflop128 is
port(
clk     : in std_logic;
crypt_en : in std_logic;
D       : in std_logic_vector(127 downto 0);
Q       : out std_logic_vector(127 downto 0)
);

end component dflipflop128 ;

signal ff_in:   std_logic_vector(127 downto 0);

begin

DFF: dflipflop128
port map(
clk => clk,
D => ff_in,
crypt_en => crypt_en,
Q => Q
);

ff_in <= D0 when sel = '0'
else D1;
end architecture dfl;
```

Listing A.42. Present - Key scheduling

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity keyscheduling_128 is
port(
        clk     : in std_logic;
        n_reset : in std_logic;
        key     : in std_logic_vector(127 downto
              0);
        crypt_en : in std_logic;
        counter : in std_logic_vector(4 downto
             0);
        roundkey: out std_logic_vector(63 downto
              0)
        );

end entity keyscheduling_128;

architecture dfl of keyscheduling_128 is

--component declaration

component flipflop128 is
        port(
                clk     : in std_logic;
                sel     : in std_logic;
                crypt_en : in std_logic;
                D0      : in std_logic_vector
                     (127 downto 0);
                D1      : in std_logic_vector
                     (127 downto 0);
                Q       : out std_logic_vector
                     (127 downto 0)
                );

end component flipflop128;

component sbox is

        port(
                sbox_in : in std_logic_vector(3
                     downto 0);
                sbox_out: out std_logic_vector(3
                     downto 0)
                );

end component sbox;


--signal declaration
signal pk_in            : std_logic_vector(127
     downto 0);
signal pk_out           : std_logic_vector(127
     downto 0);
signal sb_out           : std_logic_vector(127
     downto 0);
signal new_keystate     : std_logic_vector(127
     downto 0);


signal sbox1_out   : std_logic_vector(3 downto
     0);
signal sbox2_out   : std_logic_vector(3 downto
     0);
signal counter_out   : std_logic_vector(4 downto
      0);

begin

--component instantiation
keystate: flipflop128
port map(
        clk => clk,
        sel => n_reset,
        D0 => key,
        crypt_en => crypt_en,
        D1 => new_keystate,
        Q => pk_in
        );

sbox1: sbox
port map(
    sbox_in => pk_out(127 downto 124),
    sbox_out => sbox1_out
    );

sbox2: sbox
port map(
    sbox_in => pk_out(123 downto 120),
    sbox_out => sbox2_out
    );



--Permutation pk
-- leftshift by 61 bits in each round

pk_out <= pk_in(66 downto 0)&pk_in(127 downto
     67);


--S-boxes

sb_out <= sbox1_out&sbox2_out&pk_out(119 downto
     0);


--Counter addition

counter_out <= sb_out(66 downto 62) xor counter;
new_keystate <= sb_out(127 downto 67)&
     counter_out&sb_out(61 downto 0);

--output before permutation
roundkey <= pk_in(127 downto 64);

end architecture dfl;
```

Listing A.43. Present - Permutation module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pdata is
port(
        data_in : in std_logic_vector(63 downto
            0);
        data_out:       out std_logic_vector(63
            downto 0)
        );
end entity pdata;
architecture dfl of pdata is
begin
data_out <=data_in(63)&data_in(47)&data_in(31)&
     data_in(15)&data_in(62)&data_in(46)&data_in
     (30)&data_in(14)&data_in(61)&data_in(45)&
     data_in(29)&data_in(13)&data_in(60)&data_in
     (44)&data_in(28)&data_in(12)&data_in(59)&
     data_in(43)&data_in(27)&data_in(11)&data_in
     (58)&data_in(42)&data_in(26)&data_in(10)&
     data_in(57)&data_in(41)&data_in(25)&data_in
     (9)&data_in(56)&data_in(40)&data_in(24)&
     data_in(8)&data_in(55)&data_in(39)&data_in
     (23)&data_in(7)&data_in(54)&data_in(38)&
     data_in(22)&data_in(6)&data_in(53)&data_in
     (37)&data_in(21)&data_in(5)&data_in(52)&
     data_in(36)&data_in(20)&data_in(4)&data_in
     (51)&data_in(35)&data_in(19)&data_in(3)&
     data_in(50)&data_in(34)&data_in(18)&data_in
     (2)&data_in(49)&data_in(33)&data_in(17)&
     data_in(1)&data_in(48)&data_in(32)&data_in
     (16)&data_in(0);
end architecture dfl;
```

Listing A.44. Present - Substitution Box: component

```
-- This entity is the SBOX with boolean Minterms

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity sbox is
port(
sbox_in : in std_logic_vector(3 downto 0);
sbox_out: out std_logic_vector(3 downto 0)
);
end entity sbox;
architecture dfl of sbox is
signal v0, v1, v2, v3                    :
     std_logic;
signal out_0, out_1, out_2, out_3        :
     std_logic;
begin
v3 <= sbox_in(0);
v2 <= sbox_in(1);
v1 <= sbox_in(2);
v0 <= sbox_in(3);
sbox_out <= out_0&out_1&out_2&out_3;

out_0 <= (not v0 and v1 and v2 and not v3) or (
     not v0 and not v1 and not v2 and not v3) or
      (not v0 and v1 and not v2 and not v3) or (
     v0 and not v1
and v2) or (not v0 and v1 and v2 and v3) or (v0
     and not v1 and not v2 and v3) or (not v0
     and not v1 and v2 and v3);

out_1 <= (not v0 and not v1 and not v2 and not
     v3) or (not v0 and not v1 and not v2 and v3
     ) or (v0 and v1 and not v2) or (not v0 and
     v1 and v2 and v3) or (
not v1 and v2 and not v3) or (v0 and not v1 and
     not v2 and v3);

out_2 <= (not v0 and v1 and v2 and not v3) or (
     v0 and v1 and v3) or (not v1 and v2 and not
      v3) or (v0 and not v1 and not v2 and v3)
     or (
not v0 and not v1 and v2 and v3) or (v0 and not
     v1 and not v3);

out_3 <= (v0 and v1 and not v2 and v3) or (not
     v0 and v1 and not v2 and not v3) or (not v0
      and not v1 and not v2 and v3) or (v0 and
     v2 and not v3) or (
not v0 and v1 and v2 and v3) or (not v0 and not
     v1 and v2 and v3) or (v0 and not v1 and not
      v3);
end architecture dfl;
```

Listing A.45. Present - Substitution Box: Instantiation

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sboxes is
port(
data_in : in std_logic_vector(63 downto 0);
data_out:       out std_logic_vector(63 downto
     0)
);

end entity sboxes;

architecture dfl of sboxes is

component sbox is       port(
sbox_in : in std_logic_vector(3 downto 0);
sbox_out: out std_logic_vector(3 downto 0)
);

end component sbox;
begin
sbox0: sbox
port map(
sbox_in => data_in(63 downto 60),
sbox_out => data_out(63 downto 60)
);

sbox1: sbox
port map(
sbox_in => data_in(59 downto 56),
sbox_out => data_out(59 downto 56)
);

sbox2: sbox
port map(
sbox_in => data_in(55 downto 52),
sbox_out => data_out(55 downto 52)
);

sbox3: sbox
port map(
sbox_in => data_in(51 downto 48),
sbox_out => data_out(51 downto 48)
);

sbox4: sbox
port map(
sbox_in => data_in(47 downto 44),
sbox_out => data_out(47 downto 44)
);

sbox5: sbox
port map(
sbox_in => data_in(43 downto 40),
sbox_out => data_out(43 downto 40)
);

sbox6: sbox
port map(
sbox_in => data_in(39 downto 36),
sbox_out => data_out(39 downto 36)
);
```

```
sbox7: sbox
port map(
sbox_in => data_in(35 downto 32),
sbox_out => data_out(35 downto 32)
);

sbox8: sbox
port map(
sbox_in => data_in(31 downto 28),
sbox_out => data_out(31 downto 28)
);

sbox9: sbox
port map(
sbox_in => data_in(27 downto 24),
sbox_out => data_out(27 downto 24)
);

sbox10: sbox
port map(
sbox_in => data_in(23 downto 20),
sbox_out => data_out(23 downto 20)
);

sbox11: sbox
port map(
sbox_in => data_in(19 downto 16),
sbox_out => data_out(19 downto 16)
);

sbox12: sbox
port map(
sbox_in => data_in(15 downto 12),
sbox_out => data_out(15 downto 12)
);

sbox13: sbox
port map(
sbox_in => data_in(11 downto 8),
sbox_out => data_out(11 downto 8)
);

sbox14: sbox
port map(
sbox_in => data_in(7 downto 4),
sbox_out => data_out(7 downto 4)
);

sbox15: sbox
port map(
sbox_in => data_in(3 downto 0),
sbox_out => data_out(3 downto 0)
);


end architecture dfl;
```

Listing A.46. Present - Top Level Entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity present is
port(
clk                  : in std_logic;
n_reset          : in std_logic;
crypt_en         : in std_logic;
preset_engine: in std_logic;
key                  : in std_logic_vector
    (127 downto 0);
data_out         : out std_logic_vector(63 downto
     0);
data_in          : in std_logic_vector(63 downto
    0);
done             : out std_logic
);
end entity present;

architecture dfl of present is

--signal declaration
signal roundkey                :
    std_logic_vector(63 downto 0);
signal sbox_in          : std_logic_vector(63
    downto 0);
signal select_data : std_logic;
signal sbox_out                :
    std_logic_vector(63 downto 0);
signal pd_out           : std_logic_vector(63
    downto 0);
signal state_out               :
    std_logic_vector(63 downto 0);
signal round            : std_logic_vector(4
    downto 0);


--component declaration
component sboxes is
port(
data_in : in std_logic_vector(63 downto 0);
data_out:       out std_logic_vector(63 downto
    0)
);

end component sboxes;

component pdata is
port(
data_in : in std_logic_vector(63 downto 0);
data_out:       out std_logic_vector(63 downto
    0)
);

end component pdata;

component flipflop is
port(
clk     : in std_logic;
sel,crypt_en: in std_logic;
D0      : in std_logic_vector(63 downto 0);
D1      : in std_logic_vector(63 downto 0);
Q       : out std_logic_vector(63 downto 0)
);

end component flipflop;

component counter is
port(
clk     : in std_logic;
n_reset, crypt_en       : in std_logic;
q       : out std_logic_vector(4 downto 0)
);

end component counter;

component keyscheduling_128 is
port(
clk     : in std_logic;
n_reset,crypt_en: in std_logic;
counter : in std_logic_vector(4 downto 0);
key     : in std_logic_vector(127 downto 0);
roundkey:       out std_logic_vector(63 downto
    0)
);

end component keyscheduling_128;

begin

--component instantiation

keyschedule: keyscheduling_128
port map(
clk => clk,
n_reset => select_data,
crypt_en => crypt_en,
key => key,
counter => round,
roundkey => roundkey
);

slayer: sboxes
port map(
data_in => sbox_in,
data_out => sbox_out
);

player: pdata
```

```
port map(
data_in => sbox_out,
data_out => pd_out
);

datastate: flipflop
port map(
clk => clk,
sel => select_data,
crypt_en => crypt_en,
D0 => data_in,
D1 => pd_out,
Q => state_out
);

roundcounter: counter
port map(
clk => clk,
n_reset => select_data,
crypt_en => crypt_en,
q => round
);
--signal wiring

sbox_in <= roundkey xor state_out;

data_out <= sbox_in when round="00000";

done <= '1' when round="11111"
else '0';

select_data <=  crypt_en and preset_engine;


end architecture dfl;
```

REFERENCES

[1] M. Aigner and M. Feldhofer. Secure symmetric authentication for RFID tags. In *Telecommunication and Mobile Computing TCMC2005 Workshop, Graz, Austria*. Citeseer, 2005.

[2] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-tags. In *PerSec Conference*. Citeseer, 2007.

[3] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. *Lecture Notes in Computer Science*, 4727:450, 2007.

[4] S. Bono, M. Green, A. Stubblefield, A. Juels, A. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In *14th USENIX Security Symposium*, volume 1, page 16. USENIX, 2005.

[5] P. Bulens, F.X. Standaert, J.J. Quisquater, P. Pellegrin, and G. Rouvroy. Implementation of the AES-128 on Virtex-5 FPGAs. *Progress in Cryptology–AFRICACRYPT 2008*, pages 16–26.

[6] B. Calmels, S. Canard, M. Girault, and H. Sibert. Low-cost cryptography for privacy in RFID systems. *Lecture Notes in Computer Science*, 3928:237, 2006.

[7] P. Chodowiec and K. Gaj. Very compact FPGA implementation of the AES algorithm. *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 319–333, 2004.

[8] P.H. Cole and D.C. Ranasinghe. *Networked RFID systems and lightweight cryptography*. Springer.

[9] J. Daemen and V. Rijmen. *The design of Rijndael: AES–the Advanced Encryption Standard*. Springer Verlag, 2002.

[10] I. Damaj, S. Hamade, and H. Diab. Efficient Tiny Hardware Cipher under Verilog.

[11] M. Dworkin, NATIONAL INST OF STANDARDS, and TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV. Recommendation for Block Cipher Modes of Operation. Methods and Techniques. 2001.

[12] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. *Lecture notes in computer science*, pages 357–370, 2004.

[13] M. Feldhofer, K. Lemke, E. Oswald, F.X. Standaert, T. Wollinger, and J. Wolkerstorfer. State of the art in hardware architectures. *Information Society Technologies*, 2005.

[14] M. Feldhofer and J. Wolkerstorfer. Strong Crypto for RFID Tags-A Comparison of Low-Power Hardware Implementations. In *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, pages 1839–1842, 2007.

[15] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings-Information Security*, 152:13, 2005.

[16] F. Fürbass and J. Wolkerstorfer. ECC processor with low die size for RFID applications. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS07)*, pages 1835–1838.

[17] M. Girault, L. Juniot, and MJB Robshaw. The feasibility of on-the-tag public key cryptography. In *Proceedings of the International Conference on RFID Security*, volume 2007. Citeseer, 2007.

[18] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. *Lecture Notes in Computer Science*, pages 163–178, 2004.

[19] T. Good and M. Benaissa. AES on FPGA from the Fastest to the Smallest. *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 427–440.

[20] J.S. Grabowski. An FPGA implementation of the advanced encryption standard with support for counter and feedback modes. In *Masters Abstracts International*, volume 46, 2007.

[21] P. Hämäläinen, T. Alho, M. Hännikäinen, and T.D. Hämäläinen. Design and implementation of low-area and low-power AES encryption hardware core. In *Proc. 9th Euromicro Conference on Digital System Design (DSD 2006)*.

[22] D. Henrici and P. Müller. Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 149. Citeseer, 2004.

[23] E.P.C. Inc. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz-960 MHz. *url http://www. epcglobalinc. org/standards-technology/EPCglobalClass-1Generation-2UHFRFIDProtocolV109. pdf*.

[24] P. Israsena. Design and implementation of low power hardware encryption for low cost secure RFID using TEA. In *Information, Communications and Signal Processing, 2005 Fifth International Conference on*, pages 1402–1406, 2005.

[25] P. Israsena. Securing ubiquitous and low-cost RFID using tiny encryption algorithm. In *Wireless Pervasive Computing, 2006 1st International Symposium on*, page 4, 2006.

[26] A. Juels. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.

[27] A. Juels. The Vision of Secure RFID. *Proceedings of the IEEE*, 95(8):1507–1508, 2007.

[28] A. Juels and R. Pappu. Squealing Euros: Privacy protection in RFID-enabled banknotes. *Lecture notes in computer science*, pages 103–121, 2003.

[29] A. Juels, R.L. Rivest, and M. Szydlo. The blocker tag: Selective blocking of RFID tags for consumer privacy. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 103–111. ACM New York, NY, USA, 2003.

[30] A. Juels and S.A. Weis. Defining Strong Privacy for RFID. 2006.

[31] J.P. Kaps. Chai-tea, cryptographic hardware implementations of xTEA. *Progress in Cryptology-INDOCRYPT 2008*, pages 363–375.

[32] R.S. Khasgiwale. Utilizing physical layer information to improve RFID tag identification. Master's thesis, THE UNIVERSITY OF TEXAS AT ARLINGTON, 2009.

[33] S. Kinosita, F. Hoshino, T. Komuro, A. Fujimura, and M. Ohkubo. Nonidentifiable anonymous-ID scheme for RFID privacy protection. *Joho Shori Gakkai Shinpojiumu Ronbunshu*, 15:497–502, 2003.

[34] S. Lee, T. Asano, and K. Kim. RFID mutual authentication scheme based on synchronized secret information. In *Symposium on Cryptography and Information Security*. Citeseer, 2006.

[35] T. Li and G. Wang. Security Analysis of Two Ultra-Lightweight RFID Authentication Protocols.

[36] V. Najafi, S. Mohammadi, V. Roostaie, and A. Fotowat-Ahmady. A dual mode UHF EPC Gen 2 RFID tag in 0.18μ m CMOS. *Microelectronics Journal*, 2010.

[37] M. Ohkubo, K. Suzuki, S. Kinoshita, et al. Cryptographic approach to privacy-friendly tags. In *RFID Privacy Workshop*, volume 82. Citeseer, 2003.

[38] FCC Part. 15.247,". *Radio Frequency Device: operation with the bands*, pages 2400–2483.

[39] P. Peris-Lopez, J. Hernandez-Castro, J. Estevez-Tapiador, and A. Ribagorda. M 2 AP: A minimalist mutual-authentication protocol for low-cost RFID tags. *Ubiquitous Intelligence and Computing*, pages 912–923.

[40] P. Peris-Lopez, J. Hernandez-Castro, J. Estevez-Tapiador, and A. Ribagorda. RFID systems: A survey on security threats and proposed solutions. In *Personal Wireless Communications*, pages 159–170. Springer.

[41] P. Peris-Lopez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, and A. Ribagorda. LMAP: A real lightweight mutual authentication protocol for low-cost RFID tags. In *Proc. of 2nd Workshop on RFID Security*, page 06. Citeseer, 2006.

[42] A. Poschmann, G. Leander, K. Schramm, and C. Paar. New light-weight crypto algorithms for RFID. In *Proceedings of The IEEE International Symposium on Circuits and Systems*, pages 1843–1846, 2007.

[43] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer. Efficient AES implementations on ASICs and FPGAs. *Advanced Encryption Standard–AES*, pages 98–112.

[44] N.F. Pub. 197: Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/NIST, November 26, 2001. Available from the NIST website.

[45] D.C. Ranasinghe, D.W. Engels, and P.H. Cole. Low-Cost RFID Systems: Confronting Security and Privacy.

[46] K. Rhee, J. Kwak, S. Kim, and D. Won. Challenge-response based RFID authentication protocol for distributed database environment. *Security in Pervasive Computing*, pages 70–84.

[47] MRM Rizk and M. Morsy. Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA. In *International Design and Test Workshop, 2007 2nd*, pages 207–217, 2007.

[48] M.J.B. Robshaw. An overview of RFID tags and new cryptographic developments. *Information Security Technical Report*, 11(2):82–88, 2006.

[49] C. Rolfes, A. Poschmann, G. Leander, and C. Paar. Ultra-lightweight implementations for smart devices-security for 1000 gate equivalents. In *Proceedings of the 8th Smart Card Research and Advanced Application IFIP Conference–CARDIS 2008*, pages 89–103. Springer.

[50] G. Rouvroy, F.X. Standaert, J.J. Quisquater, and J.D. Legat. Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, volume 2, 2004.

[51] S.E. Sarma, S.A. Weis, and D.W. Engels. RFID systems and security and privacy implications. *Lecture notes in computer science*, pages 454–469, 2003.

[52] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization.

[53] M. Sbeiti, M. Silbermann, A. Poschmann, and C. Paar. DESIGN SPACE EXPLORATION OF PRESENT IMPLEMENTATIONS FOR FPGAS.

[54] T.A. Scharfeld. An Analysis of the Fundamental Constraints on Low Cost Passive RFID System Design. *Master's thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering*, 2001.

[55] F.X. Standaert, G. Rouvroy, J.J. Quisquater, and J.D. Legat. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs.

[56] B. Toiruul and K.O. Lee. An Advanced Mutual-Authentication Algorithm Using AES for RFID Systems. *IJCSNS*, 6(9B):156, 2006.

[57] I. Vajda, L. Buttyán, et al. Lightweight authentication protocols for low-cost RFID tags. In *Second Workshop on Security in Ubiquitous Computing–Ubicomp 2003*. Citeseer, 2003.

[58] S. Wakelin. *VHDL implementation of a security co-processor*. PhD thesis, SIMON FRASER UNIVERSITY, 2005.

[59] S.A. Weis, S.E. Sarma, R.L. Rivest, D.W. Engels, et al. Security and privacy aspects of low-cost radio frequency identification systems. *Lecture notes in computer science*, pages 201–212, 2004.

[60] I. Xilinx. Spartan-3 FPGA family: Complete data sheet. *Product Docummentation,(November 2005)*.

[61] J. Yang, J. Park, H. Lee, K. Ren, and K. Kim. Mutual authentication protocol for low-cost RFID. In *Ecrypt Workshop on RFID and Lightweight Crypto*, pages 17–24, 2005.

BIOGRAPHICAL STATEMENT

Shesh Kumar Jagannatha was born in Bangalore, India in 1985. He completed his Bachelor's in Telecommunication Engineering from the Visvesvaraya Technological University, India in 2006. Shesh then worked as a Software Engineer at Infosys Technologies ltd, India for 2 years. He then joined for his Masters in Electrical Engineering from University of Texas at Arlington, Texas. Shesh was working as a graduate teaching assistant for the digital design lab. He published 3 papers in peer reviewed conferences at the time of his graduation. His research interests lies in different aspects of RFID including security protocols, antenna design, reader design and implementations. He is also interested in RTL design, implementation and verification and has a good knowledge in Embedded design. He is a member of IEEE and Eta Kappa Nu - Electrical Engineering Honors Society.