IMPLEMENTING FIPS 140-2 (TAMPER RESISTANCE) STANDARD USING FPGA

By

NC RomailaBaqar

PC Sara Farrukh

NC Mamoona Shah

NC Rao Ahmad Rahil

Submitted to the Faculty of Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Islamabad

in partial fulfillment for the requirements of a B.E Degree in Telecom Engineering

JUNE 2015

ABSTRACT

It is an important duty of every IT organization to provide sufficient security in computer and telecommunication systems thus protecting IP (intellectual property). Application data is vulnerable during transmission and storage so solutions for its protection have been developed, however less consideration has been paid to protect the FPGA configuration data. This research based project concerns itself mainly with the protection of the device against all kinds of physical tampering in conformity with the level 3 of Federal Information Processing Standards (FIPS) 140-2. FIPS publications are issued by the National Institute of Standards and Technology (NIST).In order to cover a broad spectrum of possible applications and environments, this standard proposes four growing qualitative levels of security. This project implements Level 3 of the FIPS 140-2 standard. It provides certain physical security techniques to avert any unauthenticated user from accessing the Critical Security Parameters (CSPs) held within the cryptographic unit. When the detachable covers or doors of the cryptographic unit are interfered with the intent of an intrusion, this tamper response technique,zeroize all CSPs. Some of the implemented techniques use different FPGA features while others are developed using user logic. The simulations match with the results observed through ChipScope Pro. All of the modules are implemented on the Xilinx ML605 Virtex 6 (XC6VLX240T) FPGA kit.

CERTIFICATE

It is certified that the work contained in this thesis entitled **"Implementing FIPS 140-2 (Tamper Resistance) standard using FPGA"** carried out by Sara Farrukh, Romaila Baqar, Mamoona Shah and Rao Ahmad Rahil under the supervision of Asst. Prof. Waseem Iqbal for the partial fulfillment of degree of Bachelors of Telecom (Electrical) Engineering is correct and approved.

Asst. Prof. Muhammad Waseem Iqbal
Project Supervisor

Dated:

DECLARATION

No portion of work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To our parents and teachers, without whose support and cooperation, this project would not have been possible.

ACKNOWLEDGEMENT

There is no success without the will of ALLAH. We are grateful to ALLAH for giving the strength which enabled us to complete this project. We are also grateful to our parents, family, teachers and everyone who supported us through this project. Without their guidance and help we would not have been able to achieve our goals.

Table of Contents

CHAPTER 4

CHAPTER 5

CHAPTER 6

List of Figures:

List of Tables:

Abbreviations: -

AES: Advanced Encryption Standard

AT: Anti-Tamper

BBRAM: Battery Backed RAM

CSPs: Critical Security Parameters

CT: Critical Technology

DDR3: Double Data Rate type three

DSP: Digital Signal Processing

EMP: Electromagnetic Pulse

FPGA: Field Programmable Gate Array

GSR: Global Set Reset

GTS: Global Tri-state

ICAP: Internal Configuration Access Port

IP: Intellectual Property

ISE: Integrated Synthesis Environment

JTAG: Joint Test Action Group

NIST: National Institute of Standards and Technology

PR: Partial Reconfiguration

RM: Reconfigurable Module

RP: Reconfigurable Partitions

WBSTAR: Warm Boot Start Address Register

XST: Xilinx Synthesis Technology

CHAPTER 1

INTRODUCTION

1. Introduction:

1.1. Background/Motivation:

The design security is considered in terms of protecting any intellectual property (IP) from breach, since the impending losses are not just financial but also pose a threat to any classified information. The programmable logic has expanded beyond commercial markets and extended to avionic and military applications, with such confidential data, design security is obligated to take additional measures for the protection of any critical information. Protecting application data during transmission and storage is brought to pass, but lesser attention has been paid to FPGA design security—that is, protecting the configuration data of an FPGA.

The aim of this project is to secure this FPGA configuration data using different tamper resistant techniques, in compliance with FIPS 140-2 standard.

1.2. Project Description and Salient Features:

1.2.1. Project Description:

This project provides physical security to the FPGA by preventing any adversary from tampering with the equipment through some attack on the device in which the FPGA kit is installed. When any unauthorized user tampers with the protective casing of FPGA kit, FPGA will be configured (using Verilog) in such a way that the CSPs would be erased and the interfaces would be secured.

1.2.2. Prospective Application Area:

It is a research based company project sponsored by Government Organization. Implemented solution will be useful for any organization for providing physical security to any device. All of the modules in this project are implemented on the FPGA kit (as shown in ***Figure 1***) which will be installed in one of Government Organization's device i.e. Security Gateway. After its installation it will provide physical security to the device.

1.2.3. Salient Features:

Active and passive AT features are available in Virtex-6 FPGA. The features that are built into the FPGA are known as passive AT features. During the normal life cycle of the FPGA, these features are executed at different times:

- Pre-configuration (e.g., JTAG Disable)
- During-configuration (e.g., BitStream Encryption)
- Post-configuration (e.g., JTAG disable)

Active security features necessitate the user to develop an algorithm using FPGA design. These features only become effective once the FPGA is configured by means of user bitstream and as a result user design becomes active. For example, asserting KEYCLEARB to zerioze the battery-backed AES key, GSR, GTS etc.

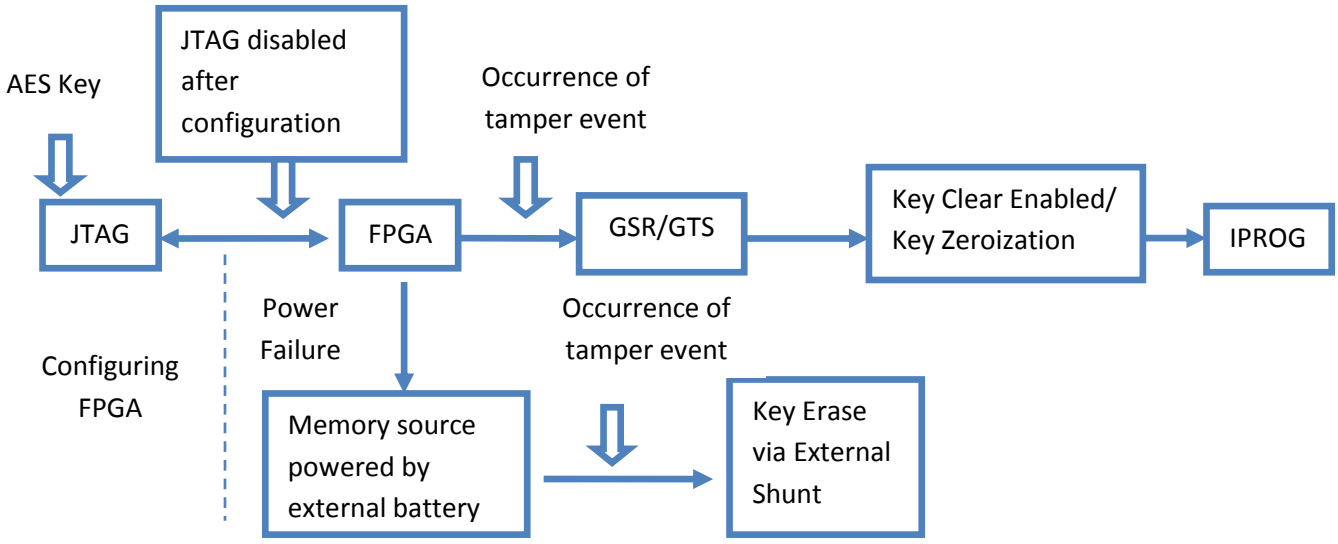


Figure 1-1: System Model Diagram of Tamper Response Techniques for both on/off conditions of FPGA

The above flow chart shows how tamper response techniques are executed in a sequence in case of a tamper event whether FPGA is in on or off condition.

1.3. Scope, Objectives, Specifications and Deliverables of the project :

1.3.1. Scope/Objectives:

The basic design would include a physical protective casing which will cover the cryptographic unit as shown in ***Figure 2***. If any intruder tries to tamper with this casing, the anti-tamper security measures will come into play. They will lock all the I/O ports so that no further data flows in or out of the system. They will disable the JTAG port and the encryption key along with all the memory stored in the FPGA device will be wiped off completely. Several different anti-tamper techniques will be implemented to protect the device.

1.3.2. Technical Specifications:

All work shall be done on the Xilinx ML605 Virtex 6 FPGA board. The choice of programming language is Verilog. Model SIM and Xilinx ISE Design Suite are the two simulation environments that will be used. In Xilinx ISE Design Suite, Impact tool, ChipScope pro and Isim were used.

1.3.3. Deliverables:

This project would cover the tamper resistance/response techniques as envisaged by FIPS 140-2 standard. The deliverable would be a tamper resistant FPGA device/module, which would resist any physical attack from an intruder. The FPGA would be enclosed in a casing which, if removed, would result in an instant resistive response by the FPGA. Any intrusion would result in erasure of AES key and device memory and locking of all I/O ports which would render the device useless to the intruder as shown in ***Figure 2***. This ensures the foolproof security of the cryptographic module.

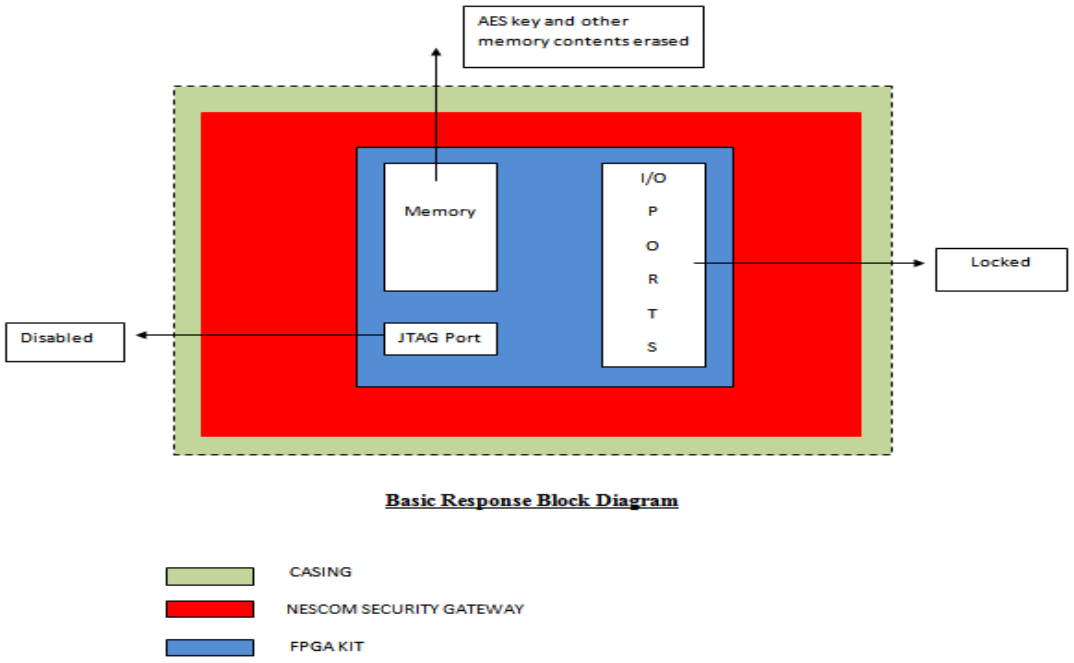


Figure 1-2: Basic response of the device in case of physical tampering

The above figure shows the basic response of the device in case of physical tampering. It shows that if any unauthorized activity is detected then JTAG port is disabled, I/O ports are locked so that no more critical data flows out and secure parameters like AES key are zeroized.

CHAPTER 2

LITERATURE REVIEW

2. Literature Review/Background Study:

Literature review for this project was done from various application notes and white papers of Xilinx. Basic source of information for this project was obtained from twenty-two documents, written by *"NIST" [1],"Ed Peterson" [2]," "Juwayriyah Hussain"[3], "Juju Joyce"[4] ,"Steven McNeil [5], "Microsemi Application note"[6] , "Austin Lesea"[7], "Wang Lie"[8], "P. Sedcole"[9], "Maurice Aarts"[10], "Steve H. Weingart"[11], "Steven J. Murdoch"[12], "Ross Anderson"[13], "Christopher T. Rathgeb"[14], "James M. Lewrs"[15], "Virtex-6 Configuration User Guide"[16], "ML605 Hardware User Guide"[17], "ML605 Reference Design"[18], "Virtex-II Platform FPGA User Guide"[19], "Virtex-6 Libraries Guide for HDL Designs User Guide"[20], "ML605 Block Diagram Schematic"[21], and "Virtex-6 FPGA Memory Interface Solutions User Guide"[22]* respectively.

The first document describes security specifications for cryptographic units. It provides four growing, qualitative levels of security proposed to cover a broad spectrum of potential applications and environments. In the second document author tells about different methods which can be implemented to secure FPGA. He presented anti-tamper (AT) guidance in order to protect the intellectual property (IP) and sensitive data from any kind of physical intrusion. Sensitive data can include the different critical security parameters (CSP`s). The third and fourth documents describe the four components to creating an anti-tamper solution namely: Tamper resistance, Tamper evidence, Tamper detection and Tamper response. They also describe the Anti-Tamper capabilities present in FPGA Designs. In the fifth document author reviews the available AT security features in the Virtex family, explains why these features exist, and gives implementation details for each feature. The sixth document is an application note explaining a quick FPGA zeriozing algorithm that will zeroize Microsemi flash based FPGAs in 5 seconds by means of JTAG. The seventh document is a white paper which overviews the four levels of security as described by NIST in FIPS 140-2. A high-level explanation regarding security features of the Xilinx Virtex-4 and Virtex-5 FPGA families is given. The eighth document gives overview of a simple reconfigurable system and dynamic partial reconfiguration design flow. The ninth document describes

how modular systems put into practice on FPGAs can take advantage from being able to load and unload multiple modules at run-time. The tenth document gives a briefly introduces technologies used to secure embedded devices like FPGAs against hardware based side channel attacks. In the eleventh document discusses tamper response methods which are means of erasing data from RAM containing CSPs. The twelfth document inspects the security limitations in tamper-proof systems. The thirteenth document describes a number of attacks that can be mounted by opponents with much shallower pockets. The fourteenth document describes the dynamic partial FPGA reconfiguration techniques and tackles the security implications for these techniques. The fifteenth document describes the self-modifying FPGA system that includes an FPGA and a configuration memory coupled to the FPGA for providing the FPGA with configuration data including SAFE configuration data and dormant configuration data. The sixteenth document describes Virtex-6 FPGA configuration overview, interfaces and modes. The seventeenth document provides details of ML605 hardware and software tools. It gives the detailed description of the hardware of Virtex-6 XC6VLX240T-1FFG1156 FPGA. The eighteenth document introduces numerous designs that exhibit Virtex-6 FPGA features using the ML605 evaluation board. The nineteenth document explains the function and operation of Virtex-II devices. The twentieth document contains a list of design elements in FPGA architecture, organized into categories based on their functionality and individual explanation of each every primitive. The twenty-first document describes the schematic of the FPGA. The twenty-second document describes the Memory Interface Generator (MIG) that generates DDRII SRAM, DDR SDRAM, DDR2 SDRAM interfaces for Virtex-4 FPGAs and generates DDR SDRAM, DDR2 SDRAM, SRAM interfaces for Virtex-5 FPGAs. It also generates DDR and DDR2 SDRAM interfaces for Spartan-3, Spartan-3A, Spartan-3E, Virtex-6 and Spartan-3A DSP FPGAs.

These application notes ensure best AT practices are adopted to provide the highest level of protection to the FPGA design. Our basic design requirements and specifications are taken from these documents.

CHAPTER 3

DESIGN

AND

DEVELOPMENT

3. Design and Development:

3.1. Design Description:

A high level of confidentiality is achieved when an encrypted bitstream is stored in an external flash and decrypted while FPGA is being configured. This makes sure that the bitstream, containing information, is only available to those who share the same secret key. Bitstream encryption offers secrecy even when the system is off and during configuration. It protects the FPGA design content which includes block RAMS and registers initialization data. The configuration bitstream is first encrypted with a key provided by the user to carry out encryption. By using the iMPACT[1] software this key is loaded into the FPGA via JTAG port. The key is saved in the volatile BBRAM of the FPGA. JTAG readback is automatically disabled when we use the encrypted configuration. Now that the FPGA is configured and the key is loaded into it as well, it will enter into the normal operation mode (operational only when Vaux is being supplied). In this mode any authorized user can gain access to the CSPs. If power failure occurs, a backup battery will supply power to BBRAM only in order to retain the AES key (as shown in ***Figure 1***). Once the tamper signal is asserted (intrusion by unauthorized user) the techniques mentioned in the table below will come into play. We have categorized all of these techniques into two major categories (shown in ***Table 1***):

[1] **iMPACT:** configures one or more devices

Main Categories	Sub Categories	Techniques	Type of AT Feature
When the tamper signal is asserted	Before FPGA configuration	Key Erasure	Active
		Physical Destruction	Nil
		Readback/JTAG Disable	Active
	During FPGA configuration	BitStream Encryption	Passive
		Global 3-State	Active
		Physical Destruction	Nil
	During Normal Operation	Readback/JTAG Disable	Active
		Global Reset/Force Reset	Active
		Load CT Only When Needed	Can be used with Active Features
		RAM Power Drop	Nil
		RAM Overwrite	Active
		Physical Destruction	Nil
What type of data is protected	FPGA Configuration Data	Readback/JTAG Disable	Active
		BitStream Encryption	Passive
		Key Erasure	Active
		Physical Destruction	Nil
	FPGA Settings/Registers Data	IPROG	Active
		Global Reset/Force Reset	Active
		Physical Destruction	Nil
	Other Data from Memories	IPROG	Active
		RAM Power Drop	Nil
		RAM Overwrite	Active
		Physical Destruction	Nil

Table 3-1: Tamper Response Techniques: Division of techniques into sub-categories

3.2. Explanation of Implemented Techniques:

[1] Bitstream Encryption:

Bitstream Encryption is one of the techniques that add an elevated level of protection to the logic design. An unauthenticated user cannot analyze externally intercepted bitstream to comprehend or replicate the design, without the knowledge of the encryption key. This technique provides discretion/secrecy even as the system is off and for the duration of configuration. The design contents of FPGA,

including block RAMs and registers initialization data, is protected by this method.

First the Xilinx ISE software does the encryption process by generating an encrypted bitstream and a corresponding encryption key. The key is stored within FPGA's devoted RAM that is powered by an external battery. Through the JTAG port, this encryption key can be programmed onto the device. The Virtex6 device carries out the reverse function, which is the decryption of the encrypted bitstream, during the configuration. The on-chip AES decryption logic is dedicated for bitstream decryption and cannot be used for any other purpose. The following shows the steps in generating encrypted bitstream:

1. On the Hierarchy panel, select the top module of the design.
2. On the process panel, right click on **"Generate Programming File"** and select **"Process Properties"** as shown in ***Figure 3.***

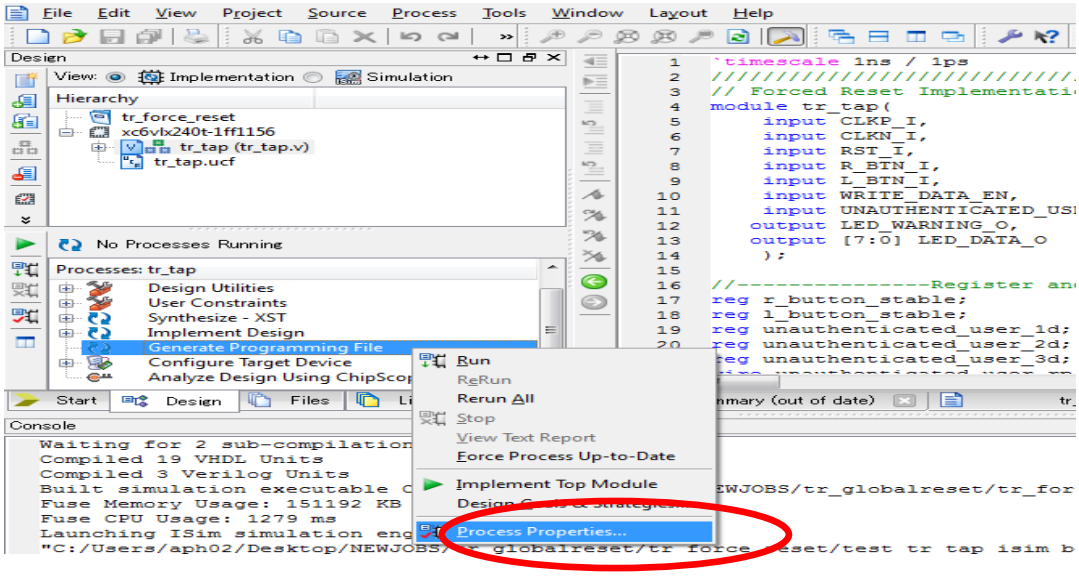


Figure 3-1: Bitstream encryption: Steps performed in Xilinx ISE to achieve this option

3. A **"Process Properties"** window will pop up. On the Category list, click **"Encryption Options"** and check the option for **"Encrypt Bitstream".**

4. On the Encrypt Key Select, select BBRAM or eFUSE, for now we select BBRAM as it is the requirement in our case.
5. Enter 256 bit Encryption key AES key, as shown in ***Figure 4.***

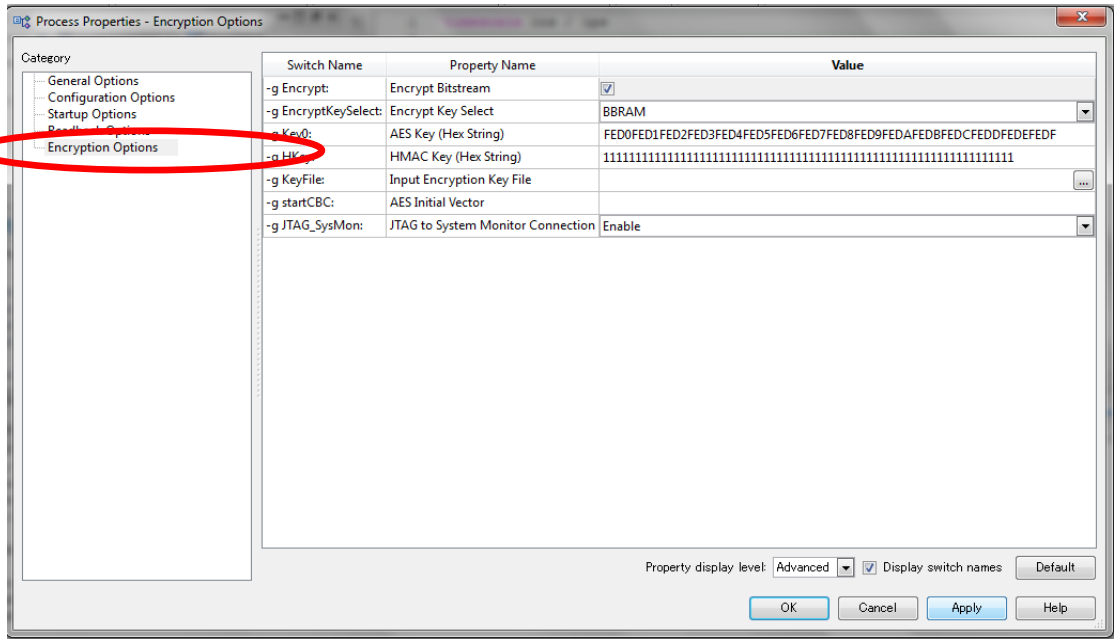


Figure 3-2: Bitstream encryption: Steps performed in Xilinx ISE to achieve this option

6. Go to step 1, but this time choose "**Rerun All**", as shown in ***Figure 5.*** After the process the tool will generate both encrypted bit file and corresponding key file ***".nky".***

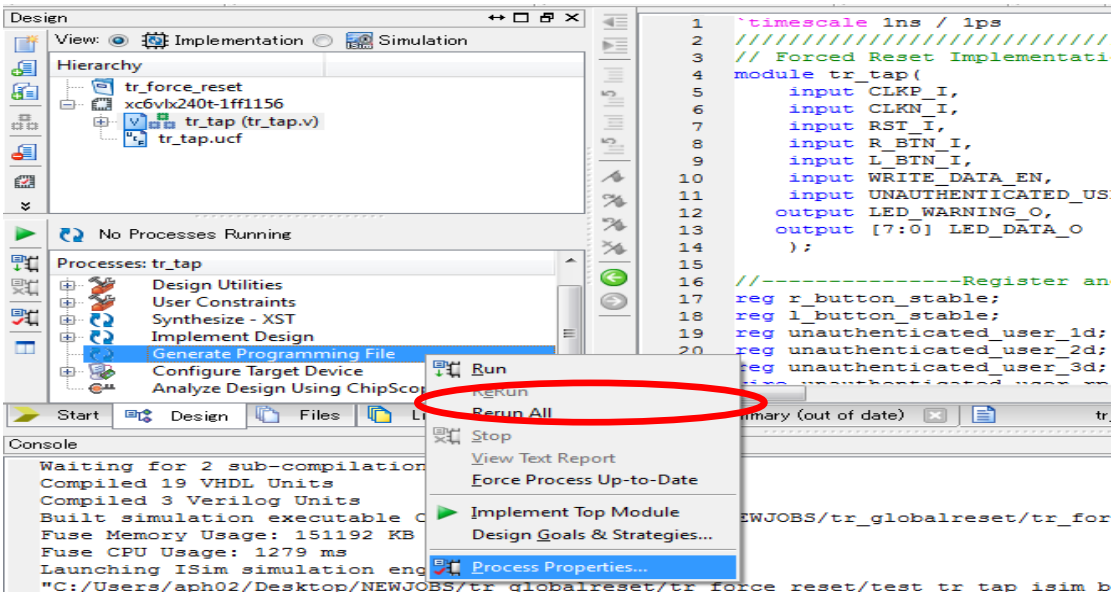


Figure 3-3: Bitstream encryption: Steps performed in Xilinx ISE to achieve this option

The Virtex-6 FPGA AES encryption system decrypts segments of 128 bits of data at a time, using a 256-bit encryption key that was used to encrypt previously. There are 1.1 x 10^77 achievable key combinations for a 256-bit key as stipulated by NIST. The same key is used for encryption and decryption by AES. The security of the data is reliant on the confidentiality of the key. An encrypted bitstream file (BIT) and an encryption key file (NKY) is created through the above process and programmed through iMPACT tool.

[2] J-tag Disable:

JTAG disable prevents an attacker from breaking into the system. Usually an intruder always starts by accessing the JTAG, to read possible information from the FPGA. But once JTAG is disabled, the JTAG chain initialized by the intruders is broken. There are two ways to do this. One is to instantiate the Boundaryscan module using HDL. The other way is by changing a setting on the BitGen process, so that the produced bit file already has this functionality. We

implemented the latter one. Below are the steps needed to do this :

1. On the Hierarchy panel, select the top module of the design.
2. On the process panel, right click on **"Generate Programming File"** and select **"Process Properties"** as shown in ***Figure 6.***

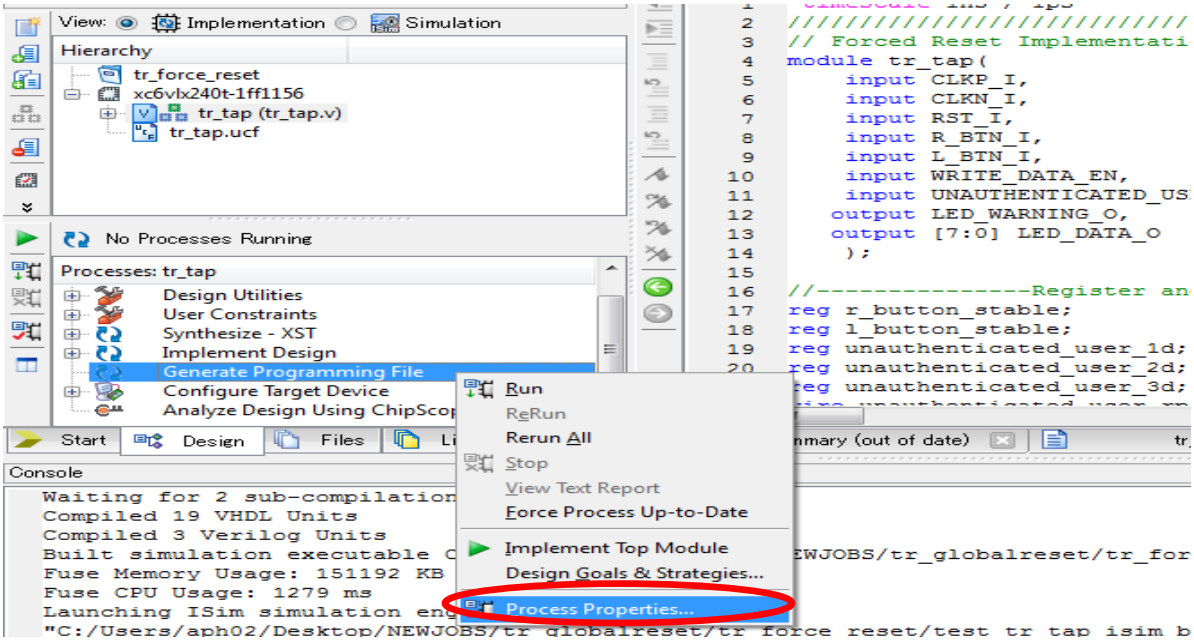


Figure 3-4: JTAG Disable: Steps performed in Xilinx ISE to achieve this option

3. A Process Properties window will pop up. On the Category list, click ***"Configuration Options"*** and check the option for ***"Disable JTAG Connection."***Then click Apply and close the window, as shown in ***Figure 7.***

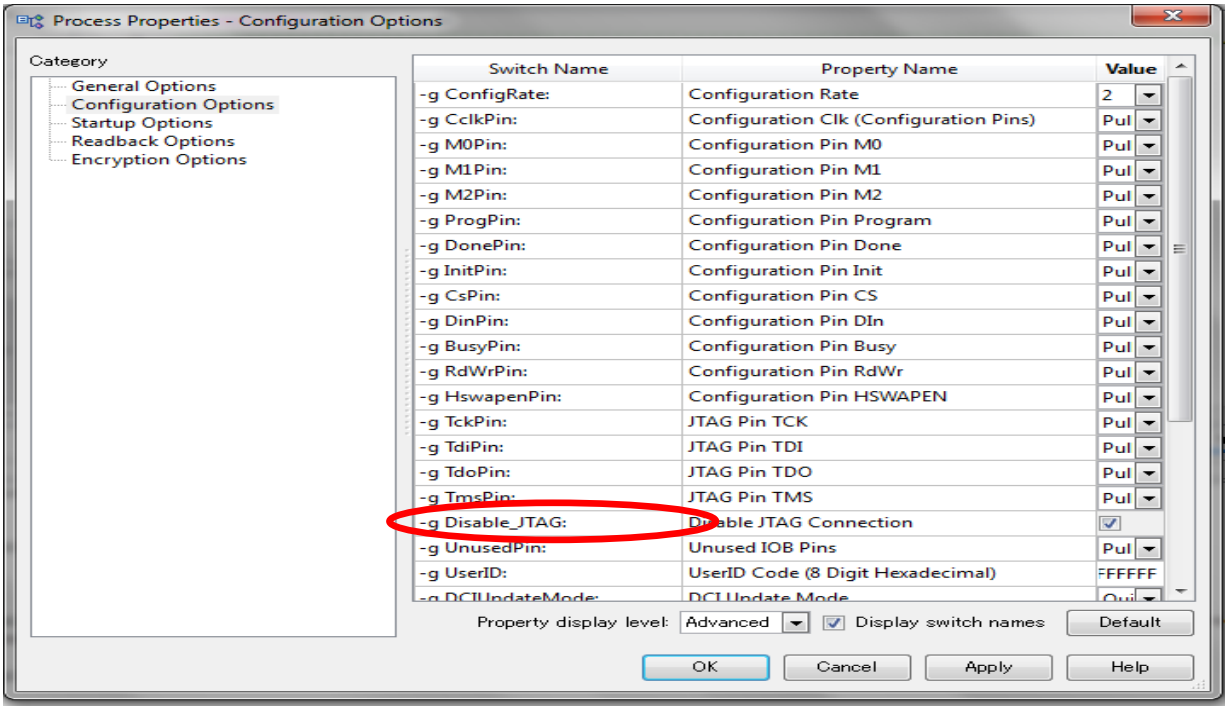


Figure 3-5: JTAG Disable: Steps performed in Xilinx ISE to achieve this option

4. Now, we can generate a bit programming file with JTAG disable function. Go to step 1, but this time choose "**Rerun All**"as shown in ***Figure 8.***

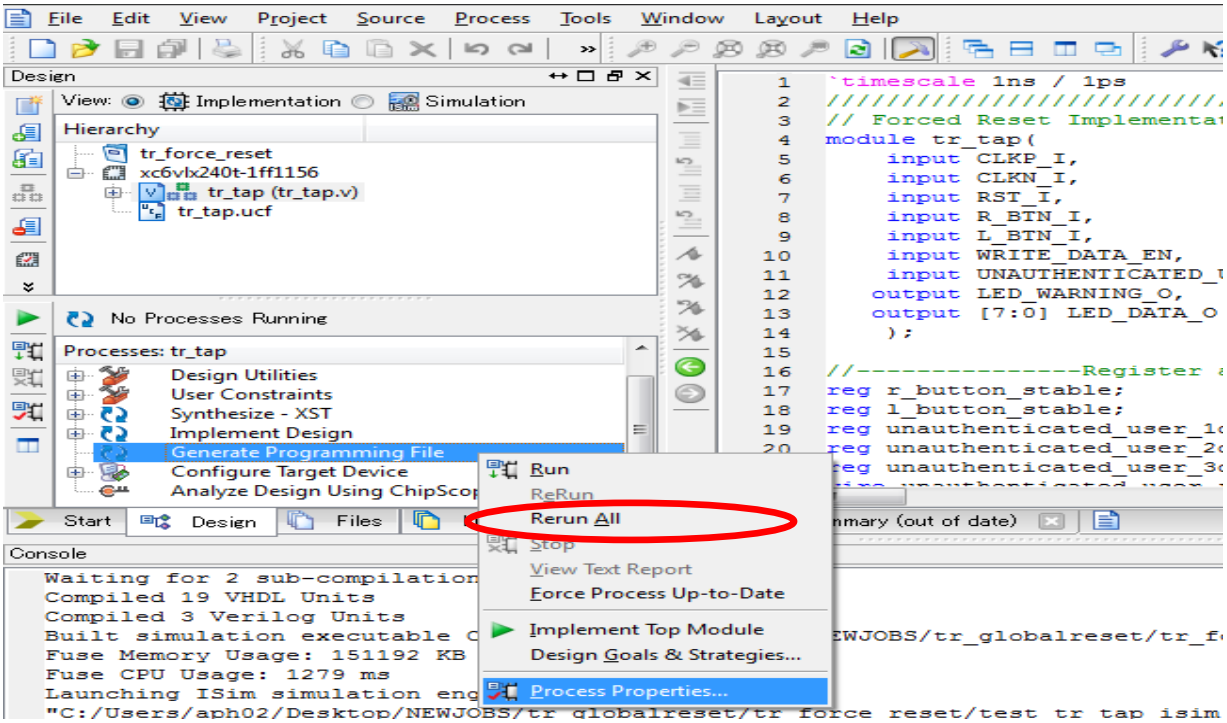


Figure 3-6: JTAG Disable: Steps performed in Xilinx ISE to achieve this option

When trying to break into a system, an invader always initiates with the JTAG port. The incorporated JTAG primitive (BSCAN) is instantiated to break the JTAG chain and to supervise any unauthorized commotion. The user can take up two different techniques to rupture the JTAG chain for the FPGA. The first method instantiates a "BSCAN primitive" in the user logic design and attaches the **DISABLE_JTAG => TRUE** characteristic to the user logic algorithm. The second method often employed to fracture the JTAG chain is to affix the **DISABLE_JTAG** option when running BitGen. Both methods are equally feasible. The first method makes sure that the chain is ruptured by the user logic algorithm. The second method puts off this decision, to break the chain, to an afterward step on the tool flow. The system designer decides where to make this decision.

[3] Global Tri-State (GTS):

Global Tri-Sate (GTS) pin is an active-High pin that is an input from the FPGA fabric. Significant information, which is highly confidential, can flow out of the external pins of FPGA. All FPGA outputs, without delay enter a high-Z state and avert data from flowing outside the FPGA when global 3-state (GTS) input on the STARTUP block is asserted. This action takes place in response to a tamper event. GTS immobilizes all the I/O drivers apart from the configuration pins. This is an instant step prior to IPROG and KEYCLEAR to ensure that data flow is brought to a standstill imminently. This pin is kept low for most applications. Summary of this feature is shown in ***Table 2.***

This method is done by instantiating the STARTUP_VIRTEX6 primitive and controlling the GTS pin. The global signals are controlled by STARTUP_VIRTEX6 primitive. When unauthenticated user

is detected, asserting the GTS will set all output pins to High Impedance.

The simulations observed were correct according to the algorithm. Simulation results are presented in the next part. Block diagram, port descriptions and Verilog Instantiation Template for STARTUP primitive is given in ***Appendix B.***

Feature	Type	Use case	User How To
GTS	Input	Outputs enter a high-Z state	STARTUP primitive is instantiated and user logic is made to develop proper conditions for GTS assertion.

Table 3-2-GTS: Features of Global 3 state are explained in detail.

[4] Global Set Reset (GSR) :

This method can be implemented using two different approaches i.e. instantiating the GSR in STARTUP primitive or by setting the registers to their default values during configuration. Both of these approaches are explained below.

- **Using STARTUP Primitive:**

  GSR is an asynchronous active high global signal from the FPGA fabric and it is activated at the end of device configuration. FPGA logic registers store CSPs such as user key. All FPGA registers (i.e., flip-flops) are restored to their default states on assertion of GSR (on STARTUP block) in response to a tamper event. In order to secure all sensitive data in the FPGA, GSR should occur prior to IPROG OR KEYCLEAR. This signal has no effect on internal memory contents of FPGA like shift registers, look-up table or block RAM; these must be cleared by developing appropriate user logic or by an IPROG command. This pin should be tied Low for most application. GSR has no input pin at the functional level as it is a global signal from FPGA fabric. A write operation is not always successful when GSR is in its active high state.

This method is done by instantiating the STARTUP_VIRTEX6 primitive and controlling the GSR pin. A fabric interface is provided by this primitive thus allowing the users to control some of global signals present in FPGA fabric. STARTUP primitive is instantiated and user logic is made to develop proper conditions for GSR assertion. When unauthenticated user is detected, asserting the GRS will accomplish the Global Reset. Summary of this feature is shown in ***Table 3.***

Feature	Type	Use case	User How To
GSR	Input	Reset flip flops to their default state when a tamper event occurs.	STARTUP primitive is instantiated and user logic is made to develop proper conditions for GSR assertion.

Table 3-3-GSR: Features of Global Set Reset are explained in detail.

- **Forced Reset:**

  We introduce this technique during normal operation. This function sets back the default value of all flip-flop and registers of the FPGA when a tamper event is detected. At the moment that an UNAUTHENTICATED_USER is detected, all the registers reset back to its default values.

  The forced reset requires every register used in the design to be defined in the code and it would be a tedious task to do so as there can be many registers present. Therefore, the technique using the startup primitive is more preferred.

  The simulations observed in both approaches were correct according to the algorithm. Simulation results are presented in the next part. Block diagram, port descriptions and Verilog Instantiation Template for STARTUP primitive is given in ***Appendix B.***

[5] Key Erasure:

- **Keyclear: -**

  It is an internal signal, when triggered as result of a tamper event clears the AES decryption key. The 256-bit key is stored in battery-backed RAM which constitutes of on-chip volatile memory cells. This ability comes into play when a tamper event is detected. The KEYCLEAR declaration must take place prior to an IPROG command. The FPGA device is rendered ineffective until reprogrammed, after KEYCLEAR is asserted. This internal signal is described in ***Table 4***.

  User How-To: This method can be used together with AES encryption method. Develop FPGA logic to verify the proper conditions for KEYCLEAR affirmation before instantiating STARTUP primitive. When unauthenticated user is detected, asserting the KEYCLEAR clears the decryption key stored in the BBRAM. (Note: effuse registers cannot be cleared.)

  The simulations were correct according to the algorithm. Simulation results are presented in the next part. Block diagram, port descriptions and Verilog Instantiation Template for STARTUP primitive is given in ***Appendix B.***

Pin_Name	Type	Category	Description
KEYCLEARB	Input	Response	Clear Key from BBRAM. This is an input pin. When seized Low for 250 ns, the contents of the BBRAM are erased.

Table 3-4-Key Clear: Features of the KEYCLEAR signal are explained in detail.

- **Key Erase via External Shunt: -**

  The memory cells of BBRAM, which holds the encryption key, are volatile and should get uninterrupted power to keep hold of their contents. During standard process, BRAM is powered by the V_{CCAUX} input. When

V_{CCAUX} is detached a separate V_{BATT} power input is supplied to preserve the key. V_{BATT} requires modest current; and so a small watch battery is appropriate for this supply. While V_{CCAUX} is being supplied, V_{BATT} does not require any current and can be detached. Schematic for V_{BATT} line is shown in ***Appendix A.***

Another method to erase the BBRAM key when Vccaux isn`t applied is by means of an exterior shunt to ground on the V_{BATT} line. If a tamper event is detected in this condition, the external battery power line to the V_{BATT} pin can be untied and the pin is driven to ground with a transistor shunt. BBRAM supply voltage is in V_{BATT} pin which is in the N8 pin of VIRTEX-6 FPGA.

[6] IPROG:

It is an internal command equivalent to the assertion of the external PROGRAM_B pin but it don`t reset the dedicated reconfiguration logic. This command is sent through the ICAP interface and it effectively zeroizes the FPGA Configuration memory (configuration data, block RAMs, and flip-flop state), all flip-flop contents, and key expansion memory, but not the key itself. It can be combined with the KEYCLEAR signal in response to an occurrence of a tamper event. IPROG command should be sent to the ICAP once KEY CLEAR is asserted. It should not take place before the assertion of KEYCLEAR.

After the assertion of both these penalties, the FPGA becomes useless as it can`t decrypt the existing bitstream. Once the device can`t be configured with the encrypted bitstream, it is indicated that some kind of tamper event has taken place.

The ICAP primitive must be instantiated in the user design to send an IPROG command as this primitive offers user design access to the Virtex-6 FPGA configuration interface.

An appropriate sequence of commands must be written in it. Readback can be performed by the user through the ICAP interface even if bitstream is encrypted. FPGA AES encryption scheme can`t be defeated by the attackers through this interface.

After configuration, the start address of the next bitstream is determined by the user logic, and WBSTAR register is set, and then IPROG command is issued using ICAP. When a tamper event is detected the system sends the following sequence of command by the ICAP interface:

1. Sync word is sent.
2. WBSTAR register is programmed for the start address of next bitstream.
3. IPROG command is sent.

Everything except the dedicated reconfiguration logic is reset by FPGA, once configuration logic receives the IPROG command. After all configuration memory is cleared, the value stored in WBSTAR is used for the starting address of bitstream. An example bitstream showed in ***Table 5*** is used to send the commands through a state machine to the ICAP.

Configuration Data (Hex)	Explanation
FFFFFFFF	Dummy Word
AA995566	Sync Word
20000000	Type 1 NO OP
30020001	Type 1 Write 1 Words to WBSTAR
00000000	Warm Boot Start Address (Load the Desired Address)
30008001	Type 1 Write 1 Words to CMD
0000000F	IPROG Command
20000000	Type 1 NO OP

Table 3-5-IPROG Command Sequence: An appropriate example bitstream to send the IPROG command [16].

The simulations were correct according to the algorithm. Simulation results are presented in the next part. Block diagram, port descriptions and Verilog Instantiation Template for ICAP primitive is given in ***Appendix B.***

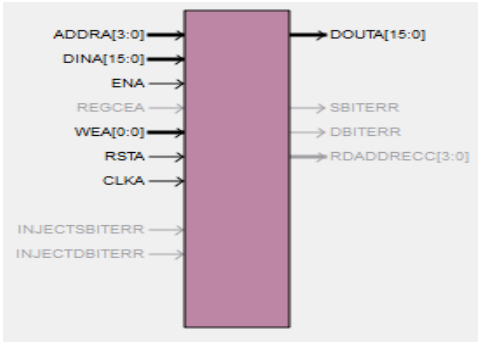
[7] Overwrite:

This method is widely accepted in administration due to its straight forwardness; however in calamitous conditions it is difficult to guarantee that enough power will be accessible to activate the over-write circuit. RAM overwrite is a technique for tamper resistance that overwrites all the addresses of a Memory with "0000000…." or "1111111111111…." when a tamper event is detected. The widespread technique is to over-write the critical with a couple of times with all 0's, then all 1's. However it is more suitable to use random or pseudo-random data since it would be more efficient. This can be time consuming since it takes more time to completely overwrite the data since the data would have to be generated.

This method had been done and tested, once the unauthenticated signal has been detected, the system enables writing to the RAM via the memory interface, and writes data and then increment the write address until all the data addresses are written with all '1's.

In this method block RAM present on the FPGA is overwritten. Block RAM in our design is generated using the Xilinx CORE Generator as shown in ***Figure 9 (a), (b)***.The simulations were correct according to the algorithm. Simulation results are presented in the next part.

```
//----------------Memory Instantiation------------//
memory u_memory (
   .clka (CLK_I),
   .rsta (RST_I),
   .ena (1'b1),
   .wea (WriteEnable),
   .addra (mem_address),
   .dina (mem_in),
   .douta (mem_out)
 );
```



a) Memory Instantiation in code **b)** Block RAM generated through core generator

Figure 3-7: Block RAM generation - Steps performed to generate Block RAM

[8] RAM Power Drop:

This is a clear-cut method to erase the data. In this techniques the power supply to the RAM modules is detached which efficiently clears the contents since RAM modules are volatile. Power supplies required for configuration of FPGA are mentioned in ***Table 6*** below. A crow bar circuit can be connected with the V_{CORE} (powers the internal logic functions such as CLBs, block RAM and DSP blocks.), and also a supervisory circuit. On the detection of a tamper event, a voltage level greater than that of the threshold of the crow bar circuit would cause the fuse/circuit breaker to trip and thus breaking the circuit and causing a power drop. This power drop will clear the block RAM.

To implement this method the RAM voltage supply is dropped. Note that we can't apply this to the ML605 since the RAM has fixed connections to Board supply voltage. We can directly cut off the main power supply to the board because due to fix connections of RAM on ML605 this method can`t be implemented only to clear the block RAM through V_{Core}.

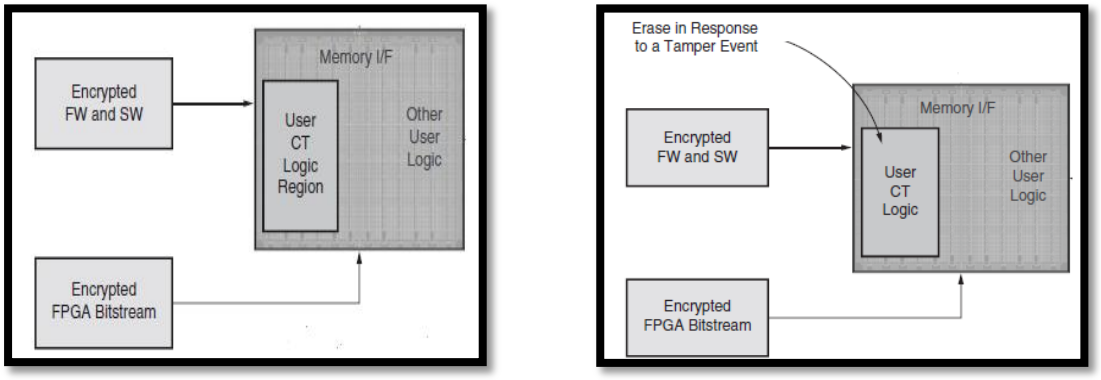
Pin Name	Description
V_{CCINT}	Internal Core Voltage
V_{BATT}	Encryption Key battery Supply
V_{CCO_0}	Configuration bank supply voltage
V_{CCAUX}	Auxiliary power input for configuration logic and other FPGA functions
V_{CCO_24} V_{CCO_34}	Dual mode configuration pin output supply voltage. Standard I/O voltage levels supported for configuration are 1.8V and 2.5V.

Table 3-6-Power Supplies Required for Configuration [16]

[9] Load CT when needed:

Partial reconfiguration features of the FPGA allow the user design to be portioned i.e. non-critical and critical technology. The CT portion of the design is loaded only when required while non-critical data remains present at all times. The CT can be erased when it has performed its

desired function. Both the PR region and the key for the CT is erased in response to a tamper event (as shown in ***Figure 10***).



a) Protecting System CT **b)** Protecting System CT—Tamper Response

Figure 3-8: Load CT when needed [2]

The license for partial re-configuration option in PlanAhead has not been provided so the PR project can`t be created.

A project must not be used for flat ISE implementation, once it is set as a PR project because all interface and options are going to work for the PR features. The PlanAhead interface is modified specifically for a PR design after the selection of PR option.

A design can have multiple RPs by running the *"Set Partition"* command for each RP in a design. Each RP will have its own RM by using the *"Add Reconfigurable Module"* command. New netlists or constraints must be brought into PlanAhead if any changes are made to the source files. After defining all the RM variants of all RPs in the PlanAhead software, physical layout of the design is defined by using Floorplanning mode. After the creation of PR design, to ensure that the design don`t violate any basic premises of PR, design checks should be run from time to time. Once all modules have been defined, different configurations can be defined and implemented. To create a configuration, any grouping of RMs and black boxes can be

used and after their creation they can then be implemented. After implementation these configurations can be controlled and verified. After verification, full configuration and partial BIT files for each RM in a chosen configuration can be generated.

[10] Physical Destruction:

This method of erasing the data is entirely reliable. On the other hand, this technique is held in reserve for the most sensitive situations. Implementing this technique practically isn`t feasible. Some such techniques are mentioned below:

- **Electromagnetic Pulse (EMP):**

  An EMP is an instantaneous burst of high power energy caused by man-made sources, and occurs as a radiated field that couples on to electrical/telecommunication lines etc. In an EMP event, every piece of unprotected equipment that relies on integrated circuits for operation could be immediately disabled or destroyed. EMP protection has become an essential element for all electronic equipment.

  ICs breakdown; undergo loss of data, thermal runaway, gate-insulator breakdown, avalanche breakdown, tunnel breakdown, and metallization burnout when exposed to EMP. The energy required is supplied by the power supply or storage capacitors or other power resources. ICs when exposed to a few hundred watts of power for few nano seconds destroy most ICs.

  Handheld EMP Device is a device which releases an Electromagnetic pulse, which breaks an electrical device within a certain range. Energy provided by the EMP devices downpour the circuits in electronic devices. The pulse melts the metal pathways in the

circuits. The energy burst can even destroy unplugged electronics.

- **Short Circuit:**

  First illustrated by Erwin Otto Marx in 1924, a Marx generator is an electrical circuit. Its causes a high-voltage pulse by means of a low-voltage DC supplies. The circuit charges numerous capacitors in parallel, and then connects them in series thus producing a high-voltage pulse. The capacitors are electrically charged in parallel to a voltage V by using a DC power supply through the resistors. The spark gaps behave as open circuits while the capacitors are being charged, as they have voltage V across them, however the breakdown voltage across them is greater than V. The last gap, if absent, would avert the capacitors form being charged however it is present and separates the load from the generator's output. In order to create the output pulse, the first gap is broken down which as a result shorts the gap thus causing the first two capacitors to be in series. As a result, the second gap is broken down thus adding the third capacitor; this process carries on breaking down all of the gaps in succession. The stack of these capacitors is connected to the load with the last gap. This outputs a short pulse as the capacitors are discharged through the load.

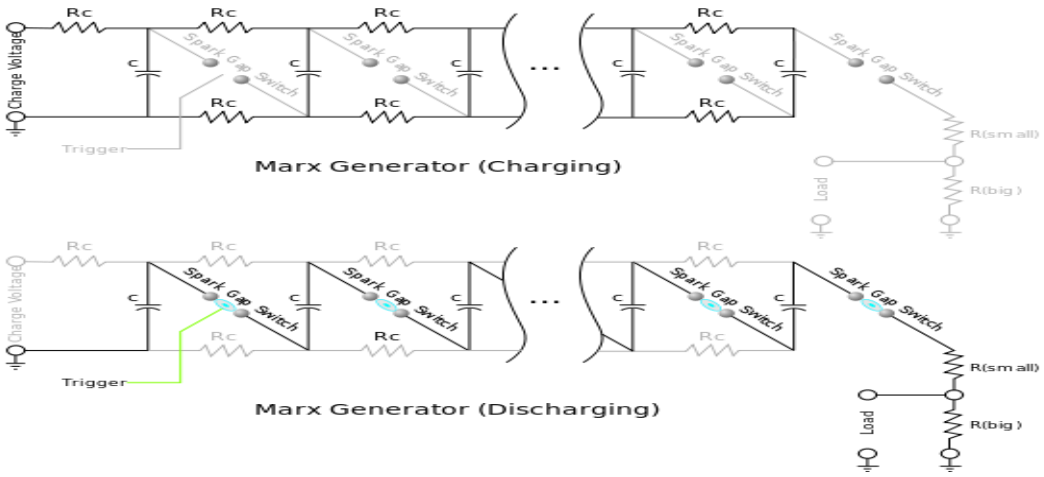


Figure 3-9: Marx Generator

Above figure shows a Marx generator charging and discharging to produce a high voltage pulse.

- **<u>Degaussing:</u>**

  Degaussing requires machine that generates strong electromagnetic field to obliterate all data that is magnetically recorded. It is done in general by a reducing alternating field. The field is produced by a standard copper coil. The amplitude is reduced by pulling the workpiece away from the coil. As the alternating field reduces the magnetic moments present inside the workpiece are randomly oriented. Virtually all magnetic field can be removed with this method.

  Magnetically hard materials require a strong magnetic field to rise above the coercive force. The penetration depth of the alternating magnetic field is reasonably reduced by the screening effects of the ferromagnetic materials. So high field strength boosts the absorption depth of the alternating field into the ferromagnetic material. To increase the absorption of the magnetic field is further enhanced by lower frequencies.

  Degaussing machines are costly and care must be taken to avoid nearby equipment from being destroyed

  .

CHAPTER 4

PROJECT ANALYSIS AND EVALUATION

4. Project Analysis and Evaluation:

Analysis and evaluation of results (analyzed using iSim and ChipScope pro) of all the implemented techniques is given below.

4.1. Results generated through iSim:

[1] GSR:

- **Forced Reset:**

  In forced reset 18 registers are defined in the program. Each registers has been assigned a default value according to its respective number. After sometime these registers are overwritten by critical data that is assumed to be "AA" in the code. This critical data will be retained by the registers as long as UNAUTHENTICATED_USER_I isn't asserted. The instant UNAUTHENTICATED_USER_I is asserted all registers are reset back to their default value that is their respective number.

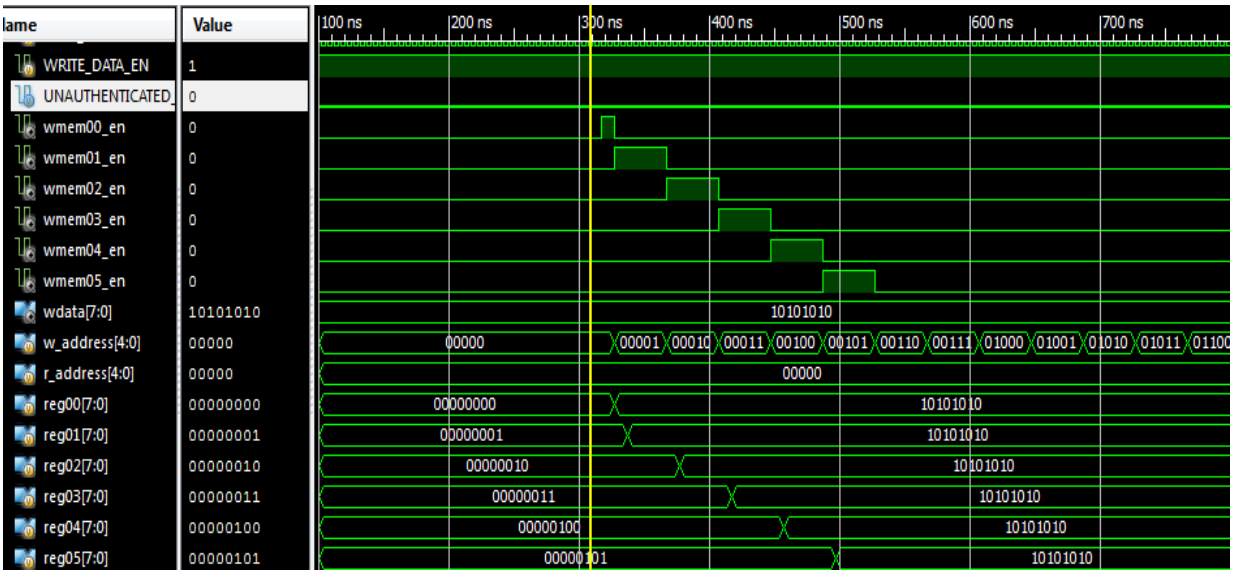


Figure 4-1: Writing critical data in registers

Critical data is written in each register once its respective write enable i.e. **wmemxx_en** is high.

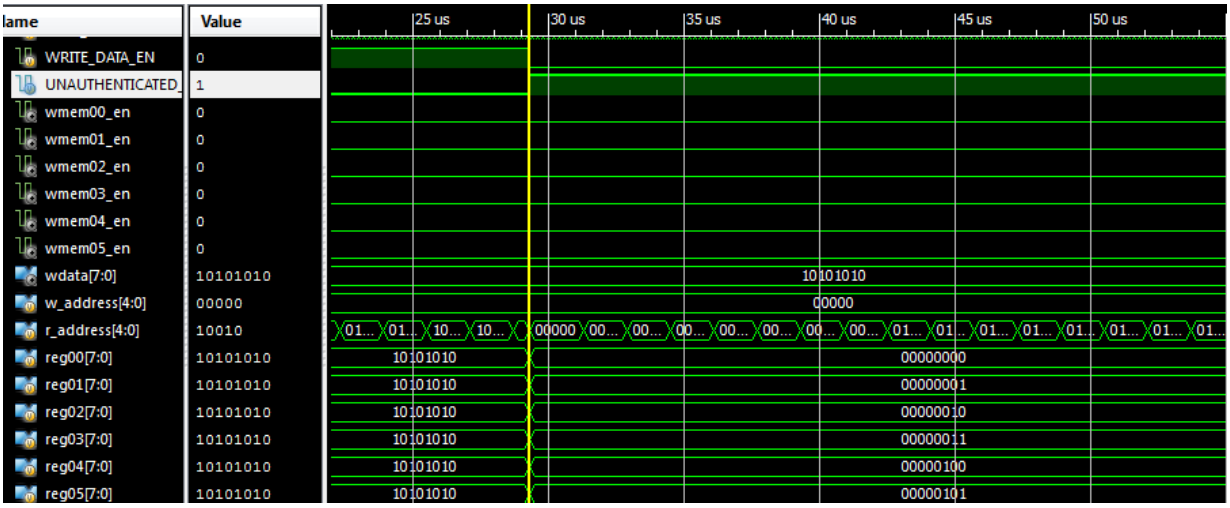


Figure 4-2: Response once UNAUTHENTICATED_USER_I is asserted

All registers are reset back to their default value once **UNAUTHENTICATED_USER_I** is asserted.

- **<u>Using STARTUP Primitive:</u>**

  In this code GSR is instantiated in the STARTUP PRIMITIVE Block. GSR is instantiated with the UNAUTHENTICATED_USER_I signal. In the code a 16X16 block RAM is generated. It set to a default value of "16'h5555". After this memory is written with critical data assumed to be "16'hFFFF". Once any tamper event occurs (UNAUTHENTICATED_USER_I) GSR goes high, thereby restoring all flipflops to their default values.

```
STARTUP_VIRTEX6 #(
.PROG_USR("FALSE") // Activate program event security feature
)

//INstantiation of startup primitive
u_STARTUP_VIRTEX6 (
  //generic map (
  //  PROG_USR => TRUE ) -- turn on PROGRAM_B intercept security feature
  //port map (
  //   .CFGCLK     ( open),
     .CFGMCLK    ( cfgmclk_signal),
    // .DINSPI     ( open),
     //.EOS        ( open),
     //.PREQ       ( preq_signal), //-- PROGRAM request to FPGA logic output
     //.TCKSPI     ( open),
     .CLK        ( 1'b0),
     .GSR        ( UNAUTHENTICATED_USER_I),// use for global reset
     .GTS        ( gts_signal),//When unauthenticated user is detected al port become tristate
     .KEYCLEARB  ( keyclearb), //When Unauthenticated user is detected, it automatically  assert to keyclearb
     .PACK       ( pack_signal),// -- PROGRAM acknowledge input (rising edge)
     .USRCCLKO   ( 1'b0),
     .USRCCLKTS  ( 1'b0),
     .USRDONEO   ( 1'b0),
     .USRDONETS  ( 1'b0)
);
```

Figure 4-3: Startup Primitive Block in Code

In the above figure instantiation of GSR in STARTUP primitive is shown.

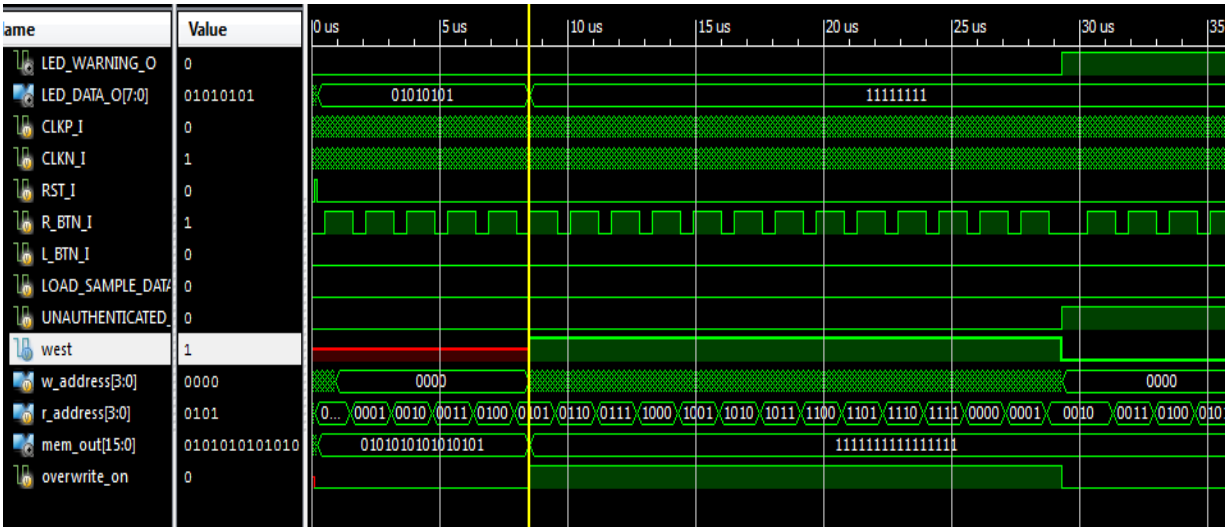


Figure 4-4: Writing of critical data and assertion of UNAUTHENTICATED_USER_I signal

In the above figure it is shown that block RAM is written with critical data. Once an unauthenticated user is detected, all the memory locations are reset back to their default values as a result of GSR pin assertion.GSR is an input pin on the FPGA fabric, so it is instantiated in the STARTUP PRIMITIVE block. As GSR isn`t an internal signal defined in the code so its assertion can`t be observed in the simulation.

[2] GTS:

In this method GTS pin is instantiated in the STARTUP PRIMITIVE Block with the UNAUTHENTICATED_USER_I signal. So once any tamper event occurs GTS pin goes high, thereby shutting off all output ports.

GTS_CFG_B bit in the Status Register, which indicates the value of numerous global signals, can be observed at run time using ChipScope pro.

```
STARTUP_VIRTEX6 #(
.PROG_USR("FALSE") // Activate program event security feature
)

//INstantiation of startup primitive
u_STARTUP_VIRTEX6 (
  //generic map (
  //  PROG_USR => TRUE ) -- turn on PROGRAM_B intercept security feature
  //port map (
  //  .CFGCLK     ( open),
    .CFGMCLK    ( cfgmclk_signal),
   // .DINSPI     ( open),
    //.EOS        ( open),
    //.PREQ       ( preq_signal), //-- PROGRAM request to FPGA logic output
    //.TCKSPI     ( open),
    .CLK        ( 1'b0),
    .GSR        ( gsr_signal) // use for global reset
    .GTS        ( UNAUTHENTICATED_USER_I),//When unauthenticated user is detected al port become tristate
    .KEYCLEARB  ( keyclearb), //When Unauthenticated user is detected, it automatically assert to keyclearb
    .PACK       ( pack_signal),// -- PROGRAM acknowledge input (rising edge)
    .USRCCLKO   ( 1'b0),
    .USRCCLKTS  ( 1'b0),
    .USRDONEO   ( 1'b0),
    .USRDONETS  ( 1'b0)
);
```

Figure 4-5: Startup Primitive Block in Code

In the above figure instantiation of GTS in STARTUP primitive is shown.

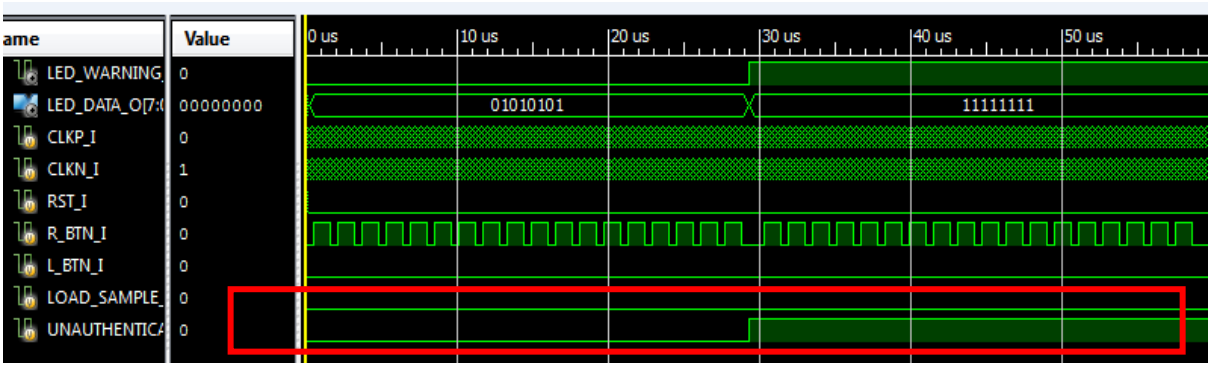


Figure 4-6: Assertion of UNAUTHENTICATED_USER_I signal

In the above waveform UNAUTHENTICATED_USER_I is detected. At the moment that an unauthenticated user is detected, all the output ports are locked as a result of GTS pin assertion.GTS is an input pin on the FPGA fabric, so it is instantiated in the STARTUP PRIMITIVE block. As GTS isn`t an internal signal defined in the code so its assertion can`t be observed in the simulation.

[3] Key Erasure:

- **Key Clear:**

  In this method KEY CLEAR pin is instantiated in the STARTUP PRIMITIVE Block with the UNAUTHENTICATED_USER_I signal. So once any tamper event occurs KEY CLEAR pin is asserted, thereby erasing the AES key.

```
STARTUP_VIRTEX6 #(
.PROG_USR("FALSE") // Activate program event security feature
)

//INstantiation of startup primitive
u_STARTUP_VIRTEX6 (
  //generic map (
  //  PROG_USR => TRUE ) -- turn on PROGRAM_B intercept security feature
  //port map (
  //  .CFGCLK     ( open),
    .CFGMCLK    ( cfgmclk_signal),
   // .DINSPI     ( open),
    //.EOS        ( open),
    //.PREQ       ( preq_signal), //-- PROGRAM request to FPGA logic output
    //.TCKSPI     ( open),
    .CLK        ( 1'b0),
    .GSR        ( gsr_signal),// use for global reset
    .GTS        ( gts_signal),//use for global tristate pin
    .KEYCLEARB  ( UNAUTHENTICATED_USER_I), //When Unauthenticated user is detected, it automatically assert to keyclearb
    .PACK       ( pack_signal),// -- PROGRAM acknowledge input (rising edge)
    .USRCCLKO   ( 1'b0),
    .USRCCLKTS  ( 1'b0),
    .USRDONEO   ( 1'b0),
    .USRDONETS  ( 1'b0)
);
```

Figure 4-7: Startup Primitive Block in Code

In the above figure instantiation of KEYCLEAR in STARTUP primitive is shown.

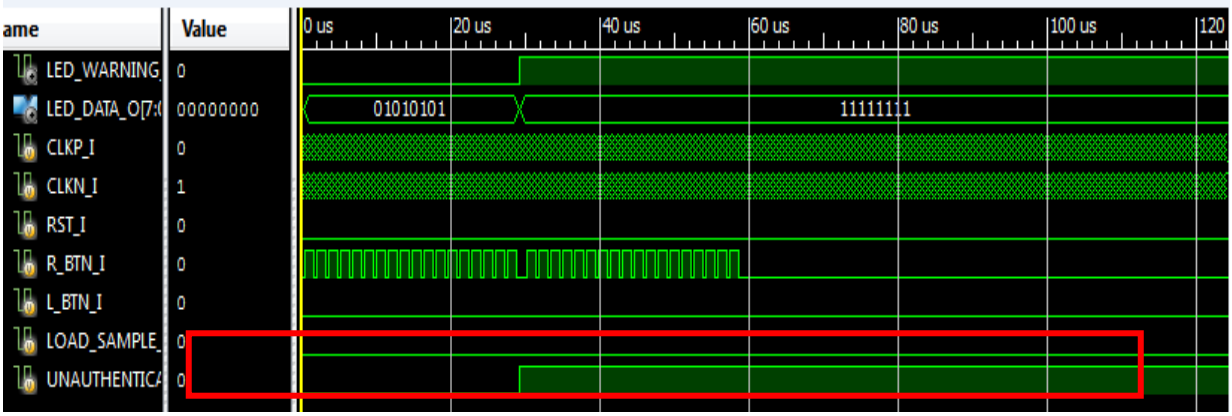


Figure 4-8:Assertion of UNAUTHENTICATED_USER_I signal

In the above waveform UNAUTHENTICATED_USER_I is detected. At the moment that an unauthenticated user is detected, AES decryption key stored in BBRAM is erased. KEY CLEAR is an input pin on the FPGA fabric, so it is instantiated in the STARTUP PRIMITIVE block. As KEY CLEAR isn`t an internal signal defined in the code so its assertion can`t be observed in the simulation.

- **Key Erase Via External Shunt:**

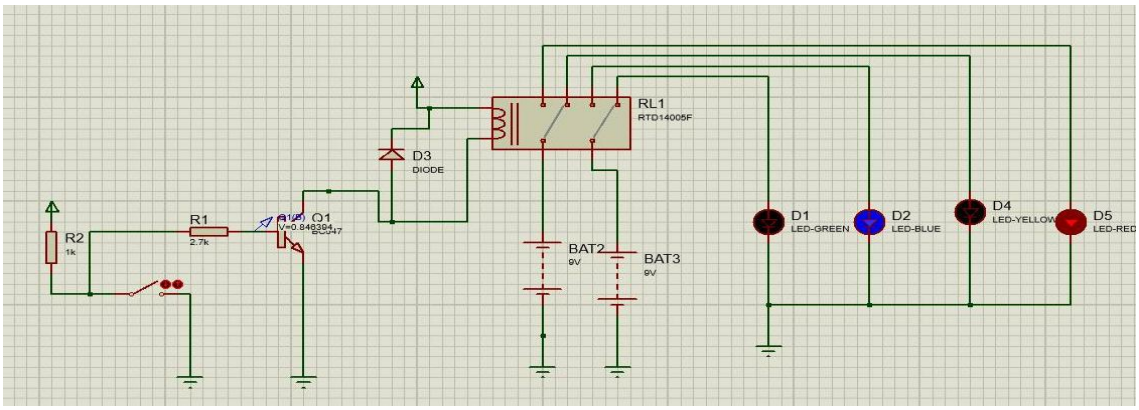


Figure 4-9:External Shunt Circuits: When switch is open

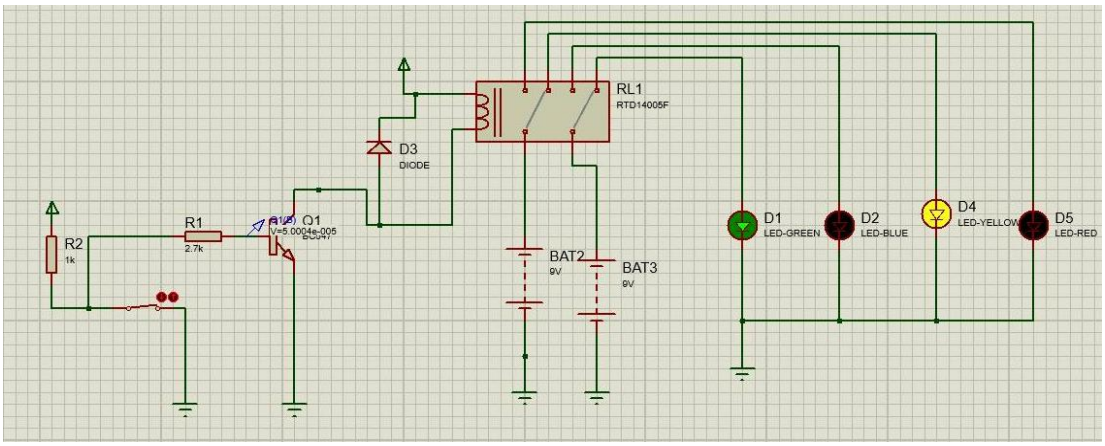


Figure 4-10: External Shunt Circuit: When switch is closed

In the above figures external shunt circuit to erase key from BBRAM when FPGA power is off is given. In the given circuit memory is retained until an UNAUTHENTICATED_USER_I is asserted (i.e. closing of switch). In case of tamper signal relay switches its connections and memory is no more supplied with power to retain the key (i.e. key is erased).

[4] IPROG:

In this method state machine for sending Iprog command through ICAP was developed. In the state machine proper logic was developed to send “Iprogsequence_data” corresponding to its respective “iprog_state”.

```
u_ICAP_VIRTEX6 (
    .BUSY(BUSY),    // 1-bit output: Busy/Ready output
    .O(O),          // 32-bit output: Configuration data output bus
    .CLK(CLK_I),      // 1-bit input: Clock Input
    .CSB(Iprog_chipselect),      // 1-bit input: Active-Low ICAP input Enable
    .I(Iprogsequence_data),          // 32-bit input: Configuration data input bus
    .RDWRB(RDWRB)   // 1-bit input: Read/Write Select input
);
```

Figure 4-11: ICAP Primitive Block in Code

In the above figure instantiation of IPROG in ICAP primitive is shown.

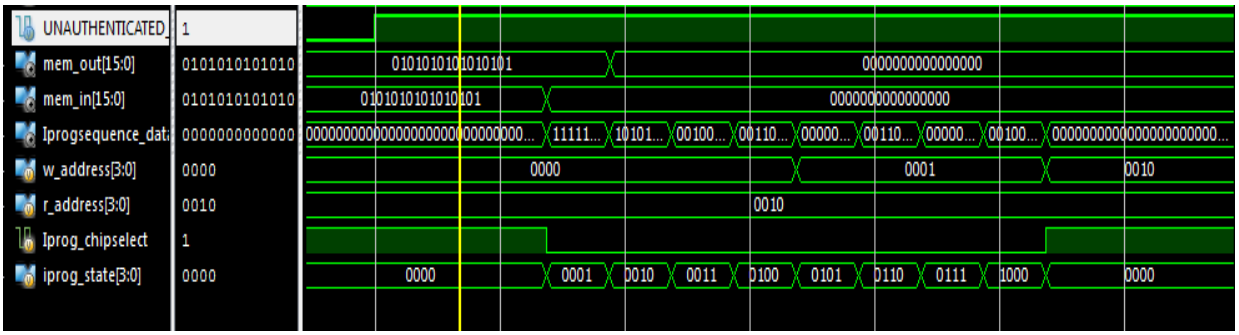


Figure 4-12: Sending the IPROG command via ICAP

In the above waveform IPROG command is sent through ICAP primitive as a result of assertion of UNAUTHENTICATED_USER_I.

[5] Overwrite:

In this method a block RAM is generated. This block RAM contains critical data i.e. 16'h5555 on every memory location. Once the tamper signal is asserted these memory locations will be overwritten by 16'hFFFF.

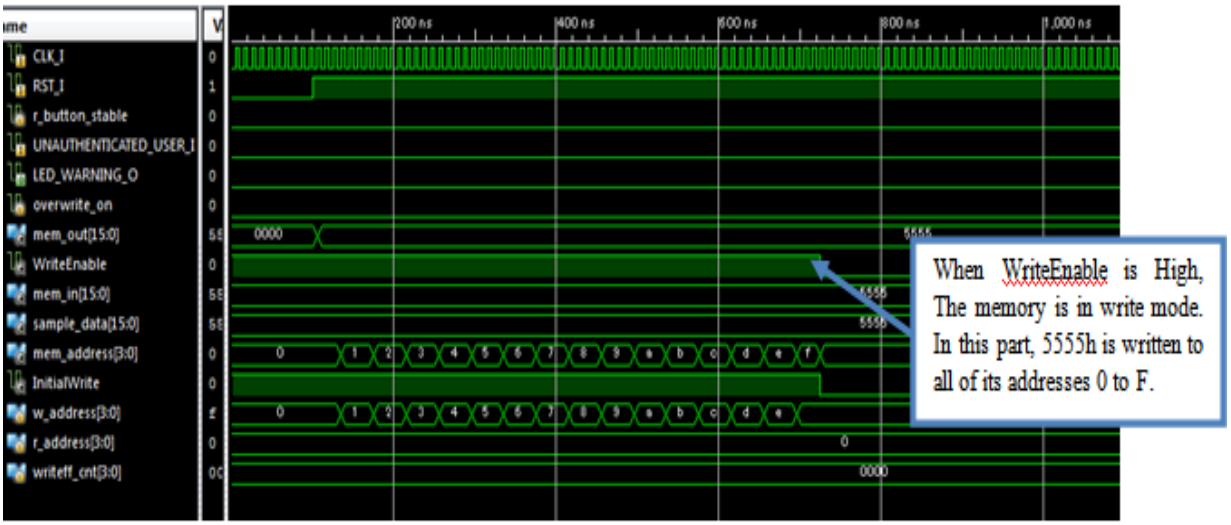


Figure 4-13: Initial Writing of Memory data

In the above waveform sample data used is fixed to 5555h and it is written to all addresses.

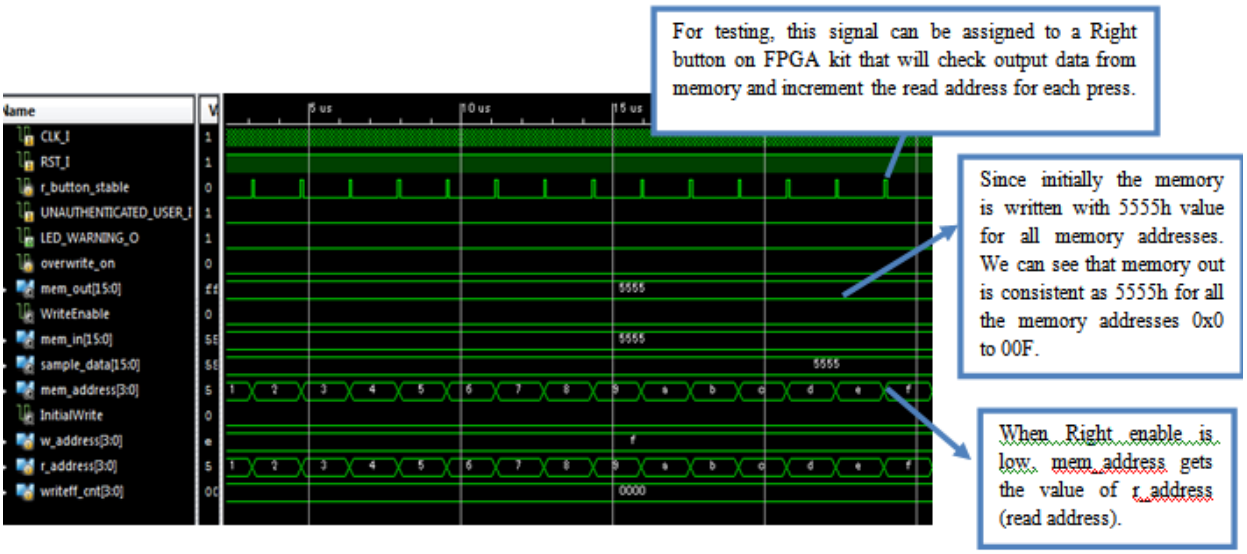


Figure 4-14: Read process after initialization.

In the above waveform after initializing the memory with 5555h it is read from the respective locations.

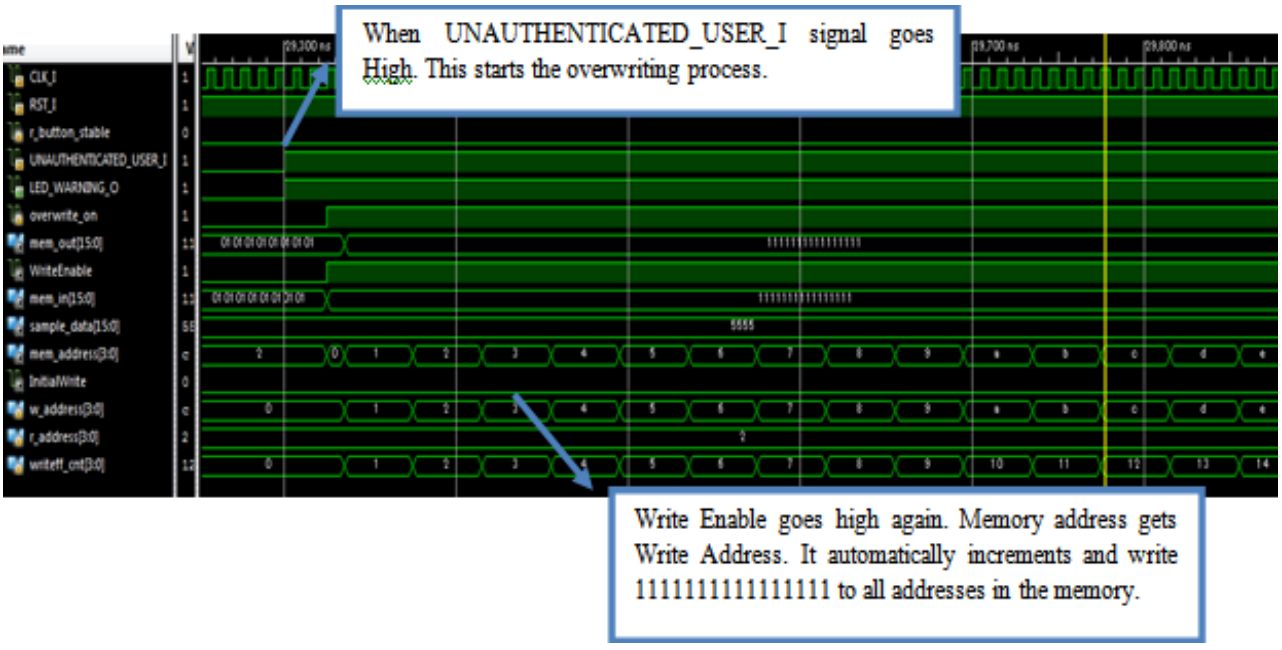


Figure 4-15: Overwrite Process

In the above waveform it is shown UNAUTHENTICATED_USER_I input is asserted. Start of overwrite process is shown.

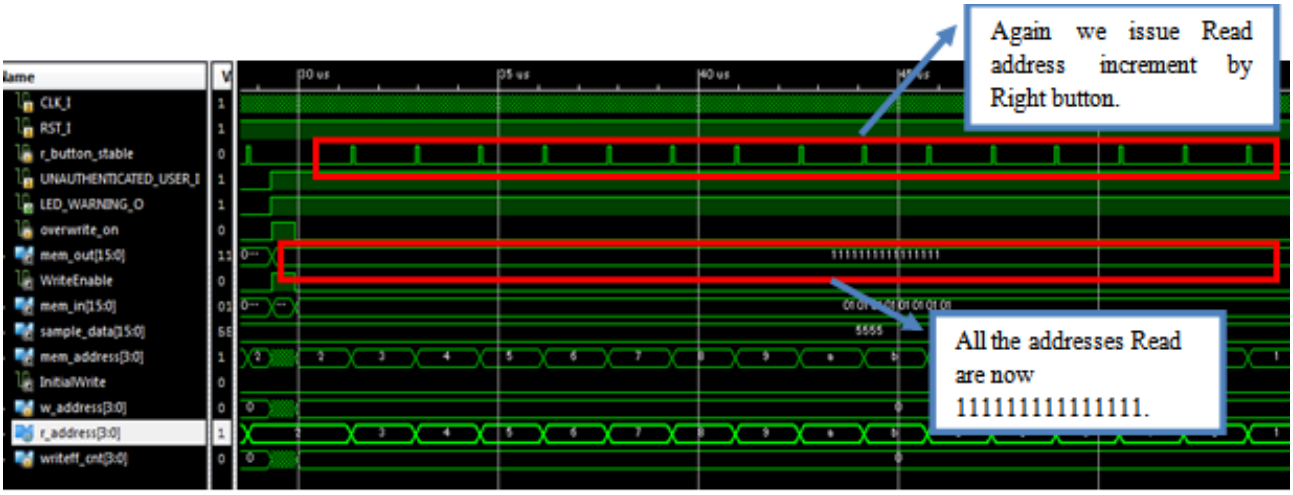


Figure 4-16: Read Process after Overwrite

In the above waveform it is shown that when UNAUTHENTICATED_USER_I input signal is asserted, 11111111 is written to all addresses.

4.2. Results generated through ChipScope pro:

[1] GTS:

```
wire [35:0] CONTROL0;
wire [31 : 0] DATA1;
wire [7 : 0] TRIG;

chipscope_icon(
    .CONTROL0(CONTROL0)
    );
chipscope_ila(
    .CONTROL(CONTROL0),
    .CLK(CLK_I),
    .DATA(DATA1),
    .TRIG0(TRIG)
    );

assign TRIG= UNAUTHENTICATED_USER_I;
assign DATA1 [7:0] = LED_DATA_O;
assign DATA1 [11:8] = w_address;
assign DATA1 [15:12] = r_address;
assign DATA1 [16] = overwrite_on;
```

Figure 4-17: icon and ila instantiation for chipscope

The code above shows that UNAUTHENTICATED_USER_I signal is used as a trigger. The values of LED_DATA_O, w_address, r_address and overwrite_on are observed on DATA1 port. The icon and ila cores are generated using CORE generator and instantiated as shown above.

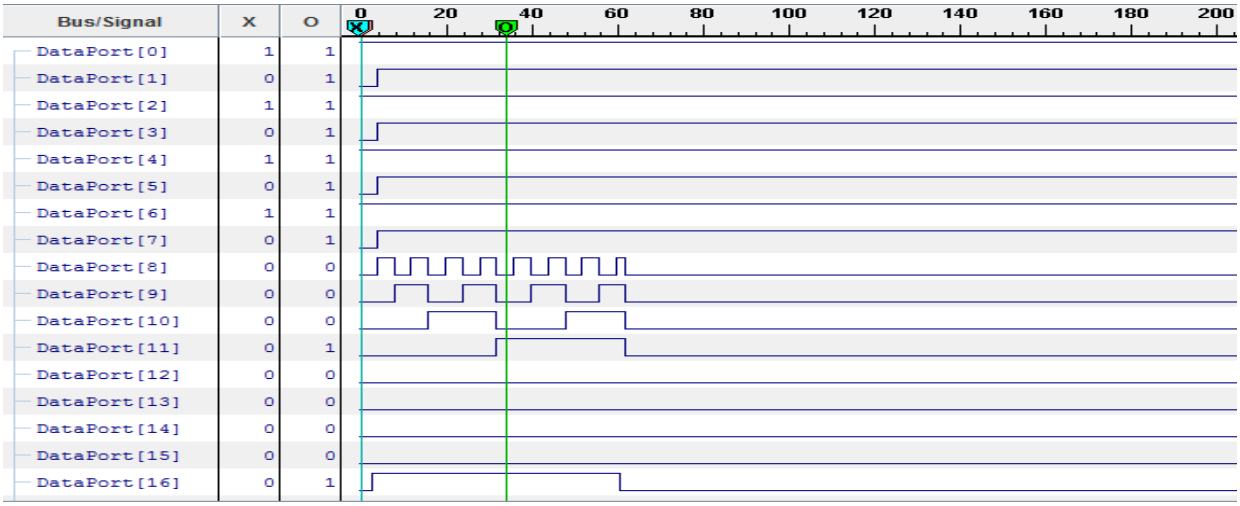


Figure 4-18: ChipScope Waveform

The X-pointer show the values of DATA1 port before trigger is asserted while O-pointer shows the values after trigger is asserted. The values observed in real time using ChipScope Pro are in accordance with the simulation observed and evaluated in the previous portion.

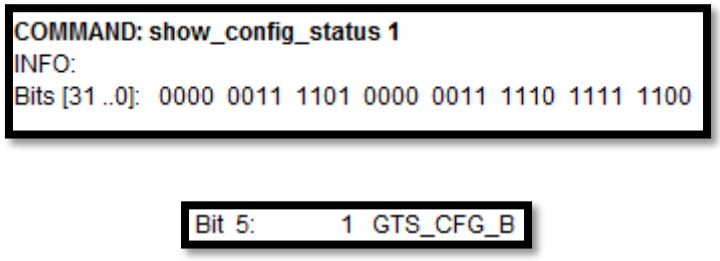


Figure 4-19: GTS_CFG_B in Status Register

The above figure shows the value of status register. The fifth bit shows the Status of GTS_CFG_B. Its value is 0 when all I/Os behave as configured while 1 when all I/Os are placed in High-Z state.

[2] GSR:

- **Using STARTUP Primitive:**

```
wire [35:0] CONTROL0;
wire [31 : 0] DATA1;
wire [7 : 0] TRIG;

chipscope_icon(
    .CONTROL0(CONTROL0)
    );
chipscope_ila(
    .CONTROL(CONTROL0),
    .CLK(CLK_I),
    .DATA(DATA1),
    .TRIG0(TRIG)
    );

assign TRIG= west ;
assign DATA1 [7:0] = LED_DATA_O;
assign DATA1 [11:8] = w_address;
assign DATA1 [15:12] = r_address;
assign DATA1 [16] = overwrite_on;
assign DATA1 [17] = UNAUTHENTICATED USER I;
```

Figure 4-20: icon and ila instantiation for chipscope

The code above shows that west signal is used as a trigger. The values of LED_DATA_O, w_address, r_address, overwrite_on and UNAUTHENTICATED_USER_I are observed on DATA1 port. The icon and ila cores are generated using CORE generator and instantiated as shown above.

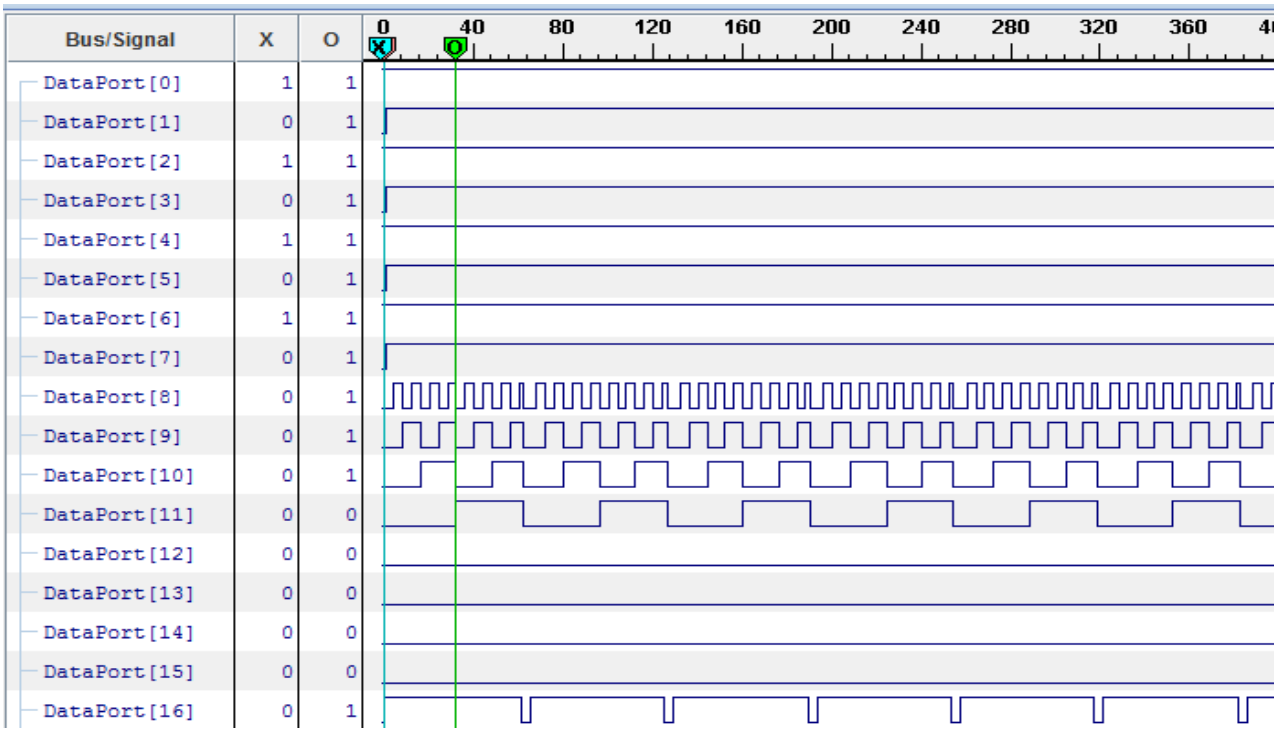


Figure 4-21: ChipScope Waveform

The X-pointer show the values of DATA1 port before trigger is asserted while O-pointer shows the values after trigger is asserted. The values observed in real time using ChipScope Pro are in accordance with the simulation observed and evaluated in the previous portion.

- **Forced Reset:**

```
wire [35:0] CONTROL0;
wire [31:0] DATA1;
wire [7:0] TRIG;
assign TRIG=UNAUTHENTICATED_USER_I;
assign DATA1[7:0]=LED_DATA_O;
assign DATA1[11:8]=reg00;
assign DATA1[15:12]=reg01;

chipscope_icon(
    .CONTROL0(CONTROL0)) /* synthesis syn_black_box syn_noprune=1 */;
chipscope_ila(
    .CONTROL(CONTROL0),
    .CLK(CLK_I),
    .DATA(DATA1),
    .TRIG0(TRIG)) /* synthesis syn_black_box syn_noprune=1 */;
```

Figure 4-22: icon and ila instantiation for chipscope

The code above shows that UNAUTHENTICATED_USER_I signal is used as a trigger. The values of LED_DATA_O, reg00 and reg01 are observed on DATA1 port.

The icon and ila cores are generated using CORE generator and instantiated as shown above.

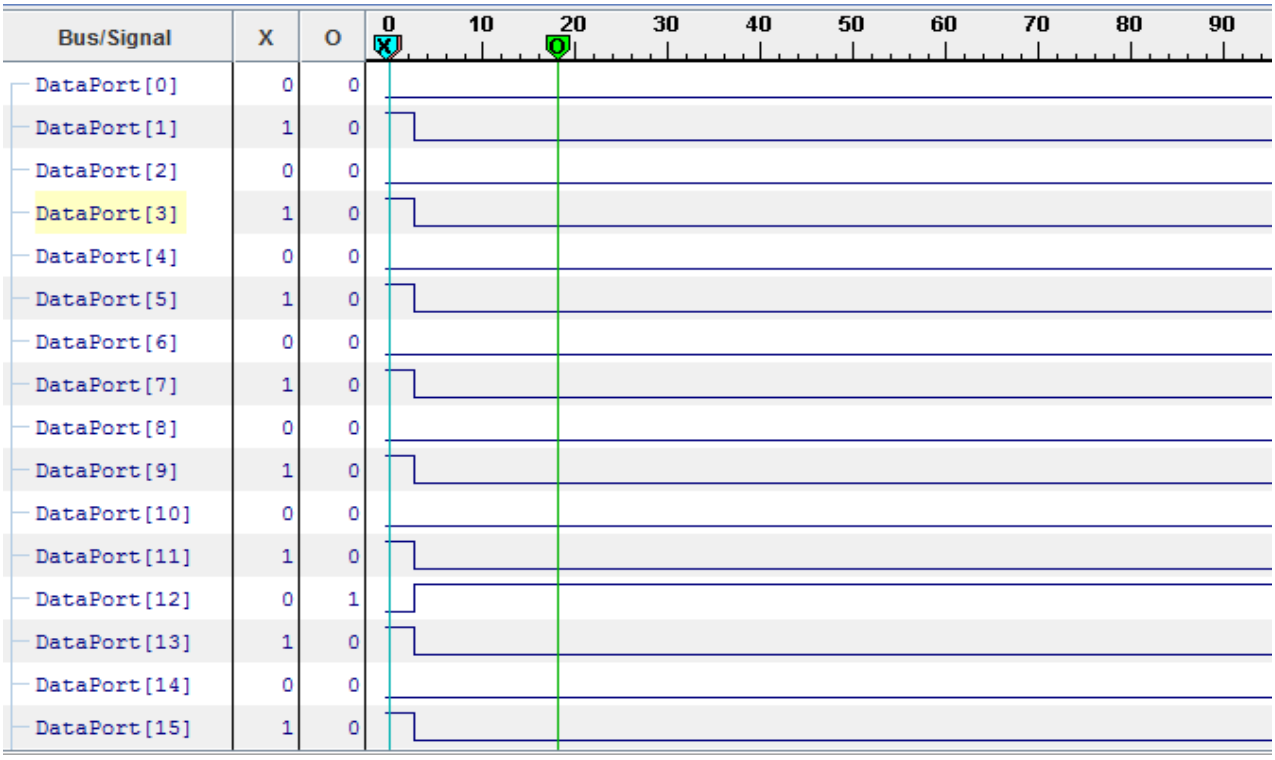


Figure 4-23: ChipScope Waveform

The X-pointer show the values of DATA1 port before trigger is asserted while O-pointer shows the values after which trigger is asserted. The values observed in real time using ChipScope Pro are in accordance with the simulation observed and evaluated in the previous portion.

[3] Key Erasure:

Key can be programmed into FPGA using iMPACT tool only and so this method can't be analyzed in real time using ChipScope pro. Hence, the real time operation will be shown on the kit.

[4] IPROG:

```
wire [35:0] CONTROL0;
wire [31:0] DATA1;
wire [7:0] TRIG;
assign TRIG=UNAUTHENTICATED_USER_I;
assign DATA1[7:0]=LED_DATA_O;
chipscope_icon(
    .CONTROL0(CONTROL0)) /* synthesis syn_black_box syn_noprune=1 */;
chipscope_ila(
    .CONTROL(CONTROL0),
    .CLK(CLK_I),
    .DATA(DATA1),
    .TRIG0(TRIG)) /* synthesis syn_black_box syn_noprune=1 */;
```

Figure 4-24: icon and ila instantiation for chipscope

The code above shows that UNAUTHENTICATED_USER_I signal is used as a trigger. The values of LED_DATA_O, reg00 and reg01 are observed on DATA1 port. The icon and ila cores are generated using CORE generator and instantiated as shown above.

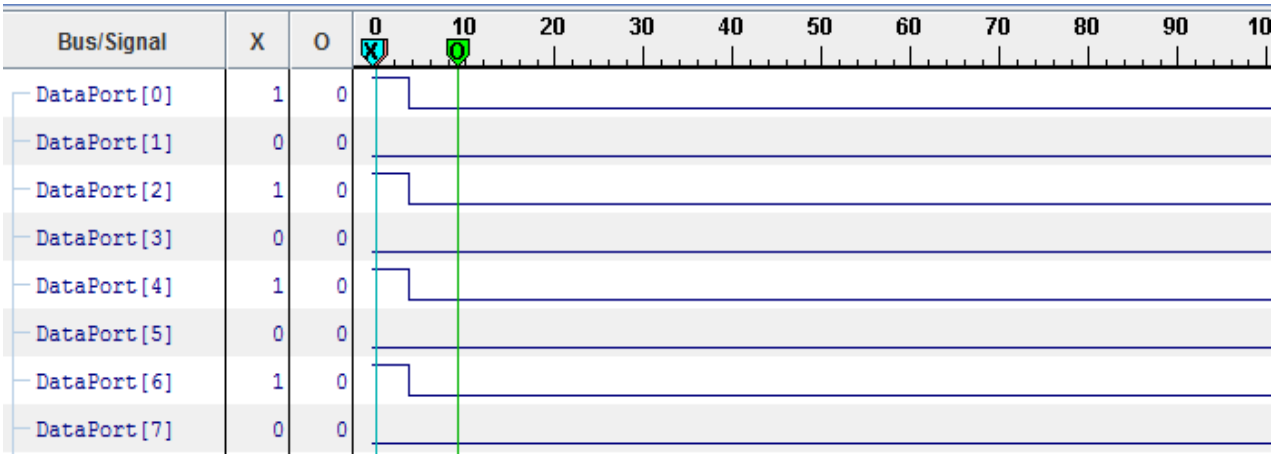


Figure 4-25: ChipScope Waveform

The X-pointer show the values of DATA1 port before trigger is asserted while O-pointer shows the values after trigger is asserted. The values observed in real time using ChipScope Pro are in accordance with the simulation observed and evaluated in the previous portion.

[5] Overwrite:

```
wire [35:0] CONTROL0;
wire [31 : 0] DATA1;
wire [7 : 0] TRIG;

chipscope_icon(
    .CONTROL0(CONTROL0)
    );
chipscope_ila(
    .CONTROL(CONTROL0),
    .CLK(CLK_I),
    .DATA(DATA1),
    .TRIG0(TRIG)
    );

assign TRIG= UNAUTHENTICATED_USER_I;
assign DATA1 [7:0] = LED_DATA_O;
assign DATA1 [11:8] = w_address;
assign DATA1 [15:12] = r_address;
assign DATA1 [16] = overwrite_on;
```

Figure 4-26: icon and ila instantiation for chipscope

The code above shows that UNAUTHENTICATED_USER_I signal is used as a trigger. The values of LED_DATA_O, w_address, r_address and overwrite_on are observed on DATA1 port. The icon and ila cores are generated using CORE generator and instantiated as shown above.

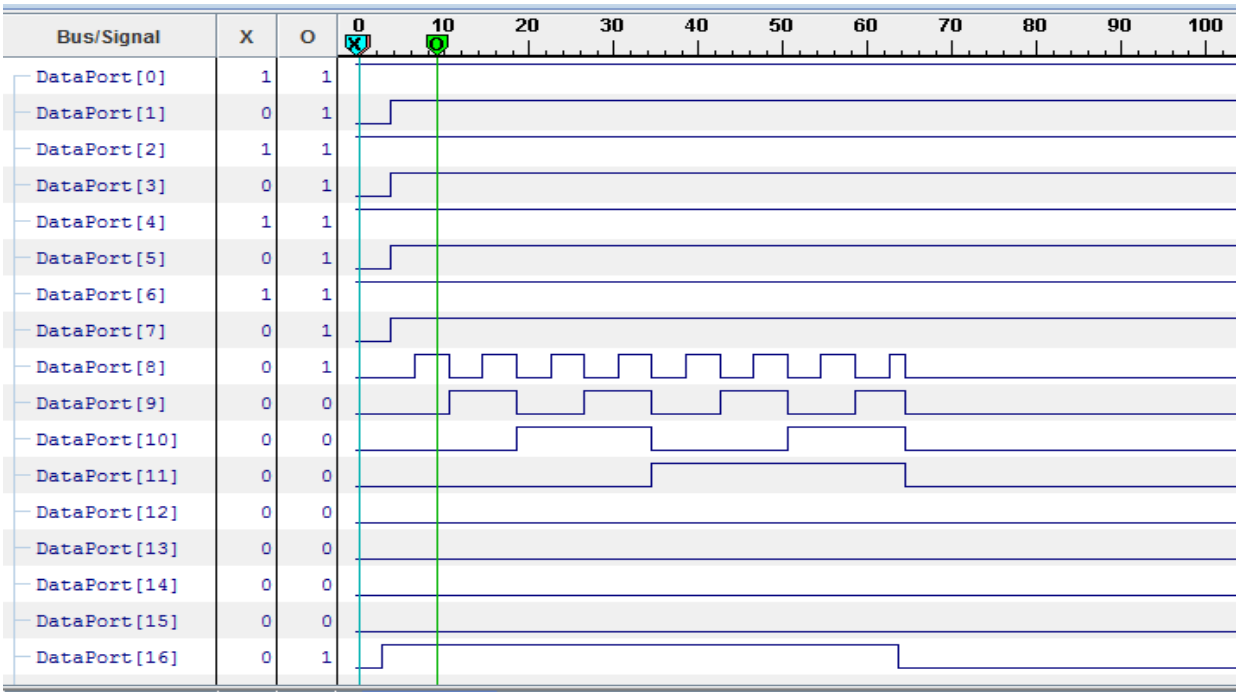


Figure 4-27: ChipScope Waveform

The X-pointer show the values of DATA1 port before trigger is asserted while O-pointer shows the values after trigger is asserted. The values observed in real time using ChipScope Pro are in accordance with the simulation observed and evaluated in the previous portion.

CHAPTER 5

FUTURE WORK

5. Recommendations for Future Work:

FIPS 140-2 comprises of 4 increasing levels of security. The top most level of security defined by FIPS 140-2 standard is Security Level 4. At Security Level 4 level cryptographic units are useful to operate in physically vulnerable environments. Protection against security compromises due to environmental conditions or fluctuations outside of the unit's normal operating ranges for voltage and temperature is also given at this level. Cryptographic unit's protection can be thwarted by a attacker by intentionally operating it beyond its normal operating ranges.

To add another dimension to this project, tamper status information can be sent back to the main system (or written to a non-volatile memory) by the user design, upon occurrence of a tamper event (in addition to asserting an appropriate response mechanism). The system could then store this data away for future auditing purposes.

These recommendations can be implemented in the future to further secure the cryptographic unit.

CHAPTER 6

CONCLUSION

6. Conclusion:

6.1. Overview:

This project focuses on providing physical security to devices in conformity with FIPS 140-2 standard in order to achieve level 3(tamper resistance) which prevents any adversary from tampering with the device and responds to any kind of physical attack on the device in which the FPGA kit is installed. Federal Information Processing Standard (FIPS) 140-2 Level 3 is the highest level that most of the devices have reached globally.

6.2. Achievements:

During the course of this project, a thorough study of various anti-tamper techniques, in compliance with FIPS 140-2, was carried out. The following were achieved:

- Basic Understanding of ML605 Virtex 6 board
- Learning of Verilog
- Familiarization with Xilinx ISE : Impact Tool, ChipScope pro, Isim, CORE Generator
- Successful implementation of tamper resistant techniques using AT features available in the Virtex-6 family.

6.3. Contributions:

This project ensures secure environment for cryptographic modules employed within computer and telecommunication systems. This project will contribute to field related to the secure design within a cryptographic unit. The area targets the cryptographic module specification, cryptographic module ports and interfaces, roles, services, and authentication, physical security, operational environment, cryptographic key management, electromagnetic interference/electromagnetic compatibility (EMI/EMC) and mitigation of other attacks.

6.4. Limitations:

Once the FPGA (inside the Security Gateway) is configured, it will be deployed into the field. In case of a tamper event, all configuration

memory and AES key will be zeroized. In order to properly work again, the device needs to be configured all over again. FIPS 140-2 provides security against physical attacks only. It doesn`t provide any protection against software based intrusions.

6.5. Applications:

FPGAs form the core of many systems due to their expanding usage in applications and sub-systems. The excess use of FPGAs has given rise to the immense need of protecting the IP held within an FPGA and countering the adversaries who attempt to do so. The techniques implemented in this project are used to prevent any outside agent to gain unauthorized access to any Intellectual property (IP).

.

7. References:

[1] (2001).Security Requirements for Cryptographic Modules.*FIPS PUB 140-2, (Supersedes FIPS PUB 140-1)*

[2] Peterson, E. (2013). Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs. *Application Note-Xilinx, XAPP1084 (v1.3).*

[3] Hussain, J., Quintana, P. (2009). Protecting the FPGA Design from Common Threats. *White paper- ALTERA, ver. 1.0*,1-2

[4] Joyce, J., Kenny, J.R.(2008). Anti-Tamper Capabilities in FPGA Designs. *White paper- ALTERA, ver. 1.0*,1-3

[5] McNeil, S.(2012). Solving Today's Design Security Concerns. *White Paper-Xilinx,WP365 (v1.2)*

[6] Zeroization, *Microsemi-Application Note AC382*

[7] Lesea, A.(2007). IP Security in FPGAs. *Application Note-Xilinx, WP261(v1.0)*

[8] Lie,W. ,Feng-yan, W. (2009). Dynamic partial reconfiguration in FPGAs, *Third International Symposium on Intelligent Information Technology Application,* 445-446

[9] Sedcole, P., Blodget, B., Becker, T., Anderson. J., Lysaght, P.(2006).Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proc.-Comput. Digit. Tech., Vol. 153, No. 3,* 157-159

[10] Aarts, M. Hardware Attacks, Tamper Resistance, Tamper Response and Tamper Evidence, 8-10

[11] Weingart, S.H. (2008). Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses. *Research Paper*,3-5,10-14

[12] Drimer, S., Murdoch, S.J., Anderson , R. Thinking inside the box: system-level failures of tamper proofing

[13] Anderson1, R., Kuhn, M., (1997). Low Cost Attacks on Tamper Resistant Devices, *Protocols, 5th International Workshop, Paris, France,*

[14] Rathgeb, C.T. , Peterson, G.D. Secure Processing Using Dynamic Partial Reconfiguration

[15] Lewis, J.M., Haddock, J.R., Walther, D.R. (2012). SELF-MODIFYING FPGA FOR ANTI TAMPERAPPLICATIONS.*US Patent, Patent No.: US 8,159,259 B1*

[16] (2013).Virtex 6 FPGA Configuration. *User Guide Xilinx, UG360 (v3.7)*

[17] (2012). ML605 Hardware UG. *User Guide Xilinx, UG534 (v1.8)*

[18] (2009). ML605 Reference Design UG. *User Guide Xilinx, UG535 (v1.0)*

[19] (2005). Virtex-II Platform FPGA UG. *User Guide Xilinx, UG002 (v2.0),485-8*

[20] (2009).Virtex-6 Libraries Guide for HDL Designs UG. *User Guide Xilinx,UG623 (v 11.4), 162-3, 308-311*

[21] (2012).ML605 Block Diagram Schematic, *Xilinx, (2,34)*

[22] (2011). Virtex-6 FPGA Memory Interface Solutions UG, *User Guide Xilinx,UG406, 7*

APPENDICES

APPENDIX A

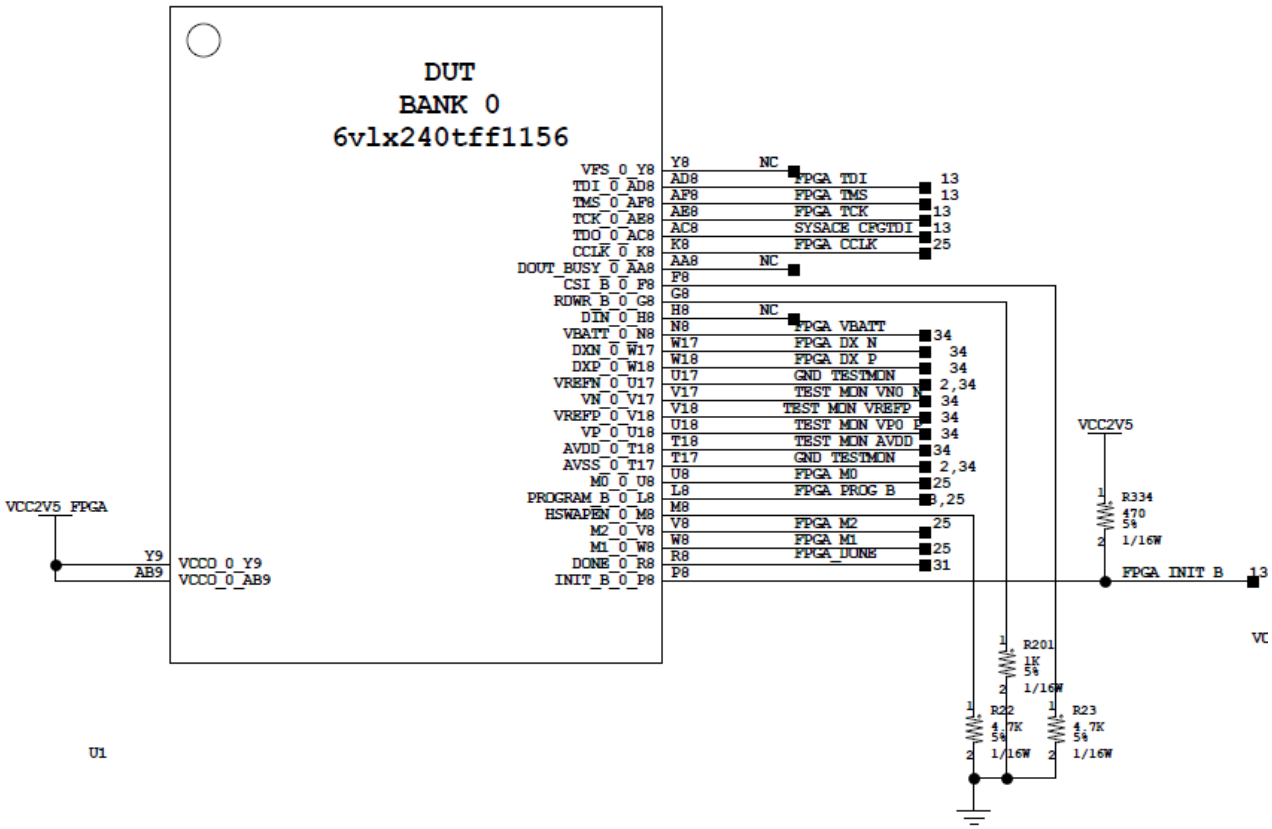


Figure A.1 -Pin Connections for V_{BATT} [21]

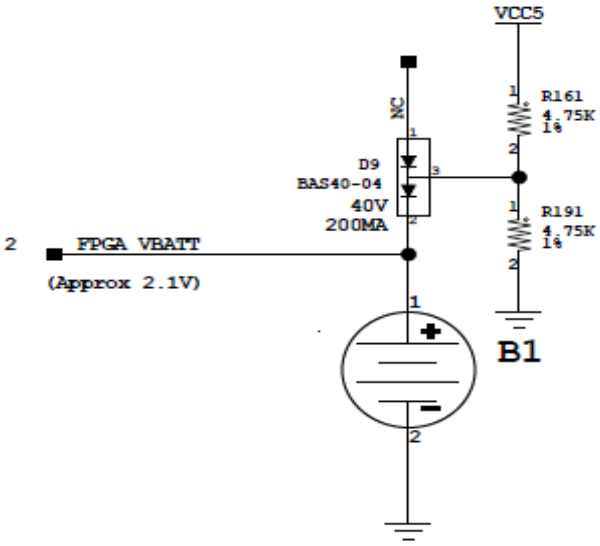


Figure A.2 - Circuit connected on V_{BATT} [21]

APPENDIX B

ICAP_VIRTEX6:

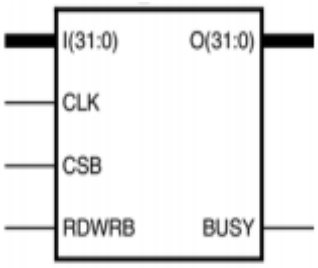


Figure B.1 -ICAP Primitive [20]

Port	Type	Width	Function
BUSY	Output	1	Busy/Ready output.
CLK	Input	1	Clock Input.
CSB	Input	1	Active-Low ICAP Enable.
I[31:0]	Input	32	Configuration data input bus.
O[31:0]	Output	32	Configuration data output bus.
RDWRB	Input	1	Read/Write Select.

Table B.1- Port description of ICAP Primitive [20]

```
ICAP_VIRTEX6 #(
   .ICAP_AUTO_SWITCH("DISABLE"),
   .ICAP_WIDTH("X8")              // Specifies the input and output data width to be used with the
                                  // ICAP_VIRTEX6.
)
ICAP_VIRTEX6_inst (
   .BUSY(BUSY),    // 1-bit Busy/Ready output
   .O(O),          // 32-bit Configuration data output bus
   .CLK(CLK),      // 1-bit Clock Input
   .CSB(CSB),      // 1-bit Active-Low ICAP Enable
   .I(I),          // 32-bit Configuration data input bus
   .RDWRB(RDWRB)   // 1-bit Read/Write Select
);

// End of ICAP_VIRTEX6_inst instantiation
```

Table B.2- Verilog Instantiation of ICAP Primitive [20]

STARTUP_VIRTEX6:

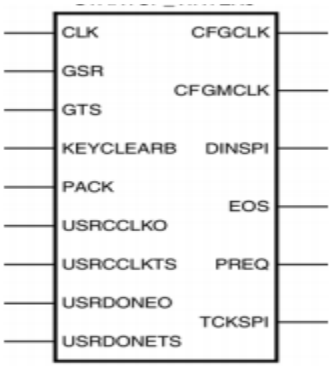


Figure B.3-STARTUP Primitive [20]

Port	Type	Width	Function
CFGCLK	Output	1	Configuration main clock output.
CFGMCLK	Output	1	Configuration internal oscillator clock output.
CLK	Input	1	User startup clock.
DINSPI	Output	1	Internal access to the DIN configuration pin when using SPI PROM configuration
EOS	Output	1	Active High signal indicates the End Of Configuration.
GSR	Input	1	Active High Global Set/Reset (GSR) signal.
GTS	Input	1	Active High Global Tristate (GTS) signal.
KEYCLEARB	Input	1	Clear AES Decrypter Key from Battery-Backed RAM (BBRAM). To erase the battery-backed-RAM contents, pull KEYCLEARB Low for more than ~200ns.
PACK	Input	1	PROGRAM acknowledge.
PREQ	Output	1	PROGRAM request to fabric
TCKSPI	Output	1	Internal access to the TCK configuration pin when using SPI PROM configuration.
USRCCLKO	Input	1	Internal user CCLK.
USRCCLKTS	Input	1	Internal user CCLK tristate enable.
USRDONEO	Input	1	Internal user DONE pin output control
USRDONETS	Input	1	User DONE tristate enable

Table B.2-Port description of STARTUP Primitive [20]

```
STARTUP_VIRTEX6 #(
   .PROG_USR("FALSE")  // Activate program event security feature
)
STARTUP_VIRTEX6_inst (
   .CFGCLK(CFGCLK),       // 1-bit Configuration main clock output
   .CFGMCLK(CFGMCLK),     // 1-bit Configuration internal oscillator clock output
   .DINSPI(DINSPI),       // 1-bit Internal access to the DIN configuration pin when using SPI PROM
                          // configuration

   .EOS(EOS),             // 1-bit Active high signal indicates the End Of Configuration.
   .PREQ(PREQ),           // 1-bit PROGRAM request to fabric
   .TCKSPI(TCKSPI),       // 1-bit Internal access to the TCK configuration pin when using SPI PROM
                          // configuration

   .CLK(CLK),             // 1-bit User start-up clock
   .GSR(GSR),             // 1-bit Active high Global Set/Reset signal
   .GTS(GTS),             // 1-bit Active high Global 3-State signal
   .KEYCLEARB(KEYCLEARB), // 1-bit Clear AES Decrypter Key from Battery-Backed RAM (BBRAM). To erase the
                          // battery-backed-RAM contents, pull KEYCLEARB low for more than ~200ns.

   .PACK(PACK),           // 1-bit PROGRAM acknowledge
   .USRCCLKO(USRCCLKO),   // 1-bit Internal user CCLK
   .USRCCLKTS(USRCCLKTS), // 1-bit Internal user CCLK 3-state enable
   .USRDONEO(USRDONEO),   // 1-bit Internal user DONE pin output control
   .USRDONETS(USRDONETS)  // 1-bit User DONE 3-state enable
);

// End of STARTUP_VIRTEX6 inst instantiation
```

Figure B.4-Verilog Instantiation of ICAP Primitive [20]

APPENDIX C

Cost Estimate

This is a company sponsored project hence all the cost was borne by Government Organization.