Encrypted Array Freeze Dump, Scan Dump and Register Dump – Enabling Debug for External Customers

Major Project Report

Submitted in fulfillment of the requirements for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Juhi Pabuwal
(13MECE07)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
May 2015

Encrypted Array Freeze Dump, Scan Dump and Register Dump – Enabling Debug for External Customers

Major Project Report

Submitted in fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Juhi Pabuwal
(13MECE07)

Under the guidance of

External Project Guide:

Mr. Pankaj Moharikar
System Validation Engineer,
Intel India Technology Pvt. Ltd.,
Banglore.

Internal Project Guide:

Prof. Akash Mecwan
Assistant Professor,
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Juhi Pabuwal

13MECE07

Disclaimer

Certificate

This is to certify that the Major Project entitled **"Encrypted Array Freeze Dump, Scan Dump and Register Dump – Enabling Debug for External Customers "** submitted by **Juhi Pabuwal (13MECE07)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof. Akash Mecwan
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head, EC

Dr. P.N.Tekwani
Head of EE Dept.

Dr. K. Kotecha
Director, IT

Certificate

This is to certify that the Major Project entitled **"Encrypted Array Freeze Dump, Scan Dump And Register Dump - Enabling Debug for External Customers "** submitted by **Juhi Pabuwal (13MECE07)**, towards the fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Date: Place: Bangalore

Mr. Pankaj Moharikar	**Mr. Vinothkumar Ethiraj**
System Validation Engineer	Engineering Manager
Intel India Technology Pvt. Ltd.	Intel India Technology Pvt. Ltd.
Bangalore	Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to Dr. P.N.Tekwani, Head of Electrical Engineering Department, and Dr. N. P. Gajjar, Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I am deeply indebted to my thesis supervisors, Mr. Pankaj Moharikar, System Validation Engineer at Intel India Technology Pvt. Ltd. and Prof. Akash Mecwan, Assistant Professor, Institute of Technology, Nirma University for their constant guidance and motivation. I also wish to thank Mr. Vinothkumar Ethiraj (Manager, Intel India Technology Pvt. Ltd.), Mr. C. Krishnaraj (System Validation Engineer at Intel India Technology Pvt. Ltd) and all other team members at Intel for their constant help and support. Without their experience and insights, it would have been very difficult to do quality work.

I wish to thank my friends for their delightful company which kept me in good humor throughout the year.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Juhi Pabuwal

13MECE07

Abstract

Design for Debugging (DFD) and Design for Testing (DFT) are two designing techniques which add certain testability features to microelectronic design. The goal of debugging is not simply to determine that the chip is not working, but to find out why it is not working. DFD is basically derived from DFT and is used for debugging now days.

Till now Intel customers dont have much debug visibility in microprocessor. Customers need to call Intel Engineers for debugging at the time of error and Intel Engineers will go to customer site and solve the error issue. But now a design is made for extracting data by customers which would facilitate quicker triage of fast escalating customer issues and decreases the turn-around time for debugging. This new component is called as Validation Control Unit (VCU) which is added to provide limited access to data extraction by customers. Customer can collect some information and then pass it on to Intel for re-solution.

Thesis contain a method which describes how to extract data/values using VCU. At the base level, a controller, named as VCU, is integrated that abstracts low level details of the hardware for debugging. In addition, the abstraction layer through an interface, such as APIs, provides services, routines, and data structures to higher-level software/presentation layers, which are able to collect test data for validation and debug of a platform under test. Moreover, VCU provides secure access to the test architecture.

Thesis contain description of hardware used for debugging. These are connected between host and platform for data extraction. The data which is dumped for debugging contains register values, array values and scan chain values. The project is intended towards validating the data extracted using VCU via JTAG. The encrypted array dump tool allows the customer to extract internal array and debug scan chains along with the exposed registers without having an Intel engineer involved. However, to protect IP, the data they are able to extract will be encrypted and requires an Intel engineer to decrypt and evaluate it

Contents

List of Figures

Abbreviation Notation and Nomenclature

PSV	Post Silicon Validation
DFT	Design For Testing
RTL	Register Transfer Logic
DFD	Design For Debugging
CUD	Circuit Under Debug
EDS	External Design Specification
IP	Intellectual Property
VCU	Validation Control Unit
API	Application Peripheral Interface
AFD	Array Freeze Dump
ITP	In Target Probe
PECI	Platform Environment Control Interface
JTAG	Joint Test Access Group
LDAT	Local Data Access Test
DMI	Direct Media Interface
PCU	Power Control Unit

Chapter 1

Introduction

1.1 Postsilicon Validation

Validation techniques may be classified as pre-silicon and post-silicon. Throughout the pre-silicon method, styles are tested by engineers in a very virtual surroundings with difficult simulation, emulation and formal verification tools. However, in post-silicon validation tests occur on actual devices that runs at real speed once deployed in industrial, real-world system boards using logic analyser and assertion-based tools. Presilicon validation is finished on a RTL machine or emulator and has been the first technique which is successfully utilized by variety of business makers. Once the primary element is prepared, post-silicon validation is finished in a system surroundings to get rid of bugs that can not be detected in presilicon validation due to slow simulation speed that prevents running an out -sized number of tests, tests not run in a particular mode, innovations in circuit technology, and lots of things. Typical postsilicon errors are from interactions between totally different elements. The goal of postsilicon validation is to form certain that no bugs are delievered to consumers and to confirm that the chips meet its specification. Postsilicon validation (PSV) is required to demonstrate that the enforced circuit meets its supposed behavior.

1.2 Design For Debug

Due to lack of in system controllability and observability, DFD hardware is deployed to support post-silicon validation which helps in accessing internal signals in system that is one of the main challenges to PSV. Variety of solutions are planned to implement the design-for-debug hardware. DFD hardware is employed to rectify the system. DFD hardware is inserted into the planning to assemble the maximum amount knowledge from the CUD so as to know the character of the error[2]. At the time of error in the system, register dump, array freeze dump (AFD) and scan dump are the important information which need to be accessed and look into for debugging.

1.3 Motivation

Post silicon validation is very important part in validation of processors/servers. Responsibility of post silicon validation team does not end up after selling the servers/chip to customers. At the time of error scenarios, Intel Engineers need to go to customer site to collect data and then come back to Intel site for debugging. This is a long procedure. So, a microcontroller unit is added which would facilitate quicker triage of fast escalating customer issues and decreases the turn-around time for debugging. With the help of this validation unit, customer can dump data in different error scenarios and pass it to the Intel engineers. This unit will provide encrypted data which is important for confidentiality of the company. This thesis provides debugging facility which will work on future technologies as well. The project will help in reducing the turn around time for debugging.

1.4 Objective

Till now Intel customers dont have much debug visibility in microprocessor. Customers have to every time call Intel for debugging and Intel Engineers will go and solve the issue. Now, a new component called Validation Control Unit (VCU) is added for providing limited debug access to customers. Customer can collect encrypted information and then pass it on to Intel for re-solution. VCU is included to provide control of and access to design for test features. This is microcontroller based unit. API is run to talk to VCU for accessing information according to a security level. Security level is defined in a way that the customers will not be able to access all the information but only the information which Intel wants customer to see. In short, a limited access to information is given using VCU.

1.5 Broadwell Server

The platform used for thesis is Broadwell server with multiple cores and sockets. Broadwell is the codename for the Intel processor family that is the successor to the Haswell processor family and will be built on Intels 14nm manufacturing process. Most Broadwell products are targeted at the 2 in 1 tablets and notebooks but there will also be versions for desktop and server platforms. Broadwell delivers significant performance advancements over the prior generation, including power efficiency and security. Broadwell DE, EP and EX are different variants of Broadwell server.

Broadwell-DE processors are designed for low power servers. The latest Intel Xeon processors take performance, energy-efficiency and compute density to new heights, while delivering best-in-class support for virtualization, cloud computing, in-memory computing, big-data analytics and technical computing. Servers based on these pro-

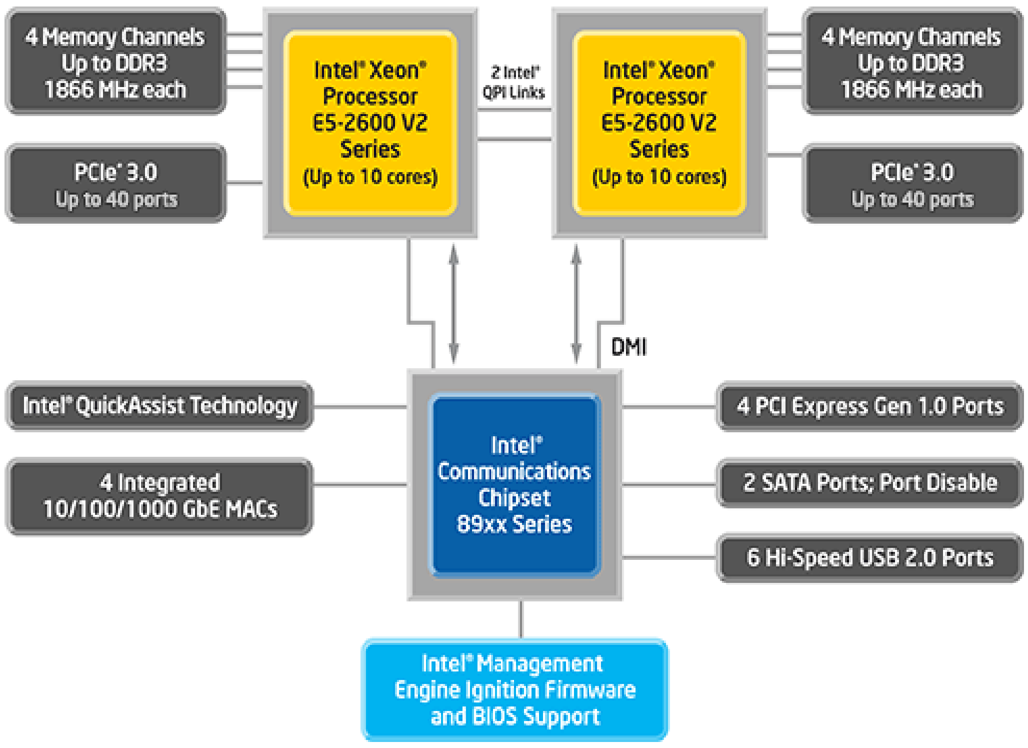

4 Memory Channels Up to DDR3 1866 MHz each
PCIe® 3.0 Up to 40 ports
Intel® Xeon® Processor E5-2600 V2 Series (Up to 10 cores)
2 Intel® QPI Links
Intel® Xeon® Processor E5-2600 V2 Series (Up to 10 cores)
4 Memory Channels Up to DDR3 1866 MHz each
PCIe® 3.0 Up to 40 ports
DMI
Intel® QuickAssist Technology
4 Integrated 10/100/1000 GbE MACs
Intel® Communications Chipset 89xx Series
4 PCI Express Gen 1.0 Ports
2 SATA Ports; Port Disable
6 Hi-Speed USB 2.0 Ports
Intel® Management Engine Ignition Firmware and BIOS Support

Figure 1.1: Block Diagram of Server
[7]

cessors The latest Intel Xeon processors take performance, energy-efficiency and compute density to new heights and deliver best-in-class support for virtualization, cloud computing, in-memory computing, big-data analytics and technical computing. Servers supported by these processors provide the foundation for a standardized, efficient and agile data center that can help to drive down space, power, cooling and management costs, while providing exceptional performance and reliability across the full range of workloads.[6]. New server Xeon microprocessors can feature up to 8 cores with 12 MB L3 cache. The cores support Intels Hyper-Threading, Turbo Boost, virtualization and Trusted Execution technologies. A new feature known as

Processor Trace will be included in Xeon D which will allow capturing details of code execution, supervisor mode access prevention and extensions to ADC instruction.

1.6 Thesis Organization

The rest of the thesis organized as follows.
Chapter 2 explains the different stages of the tool with its background.
Chapter 3 deals with brief description of Validation Control Unit and how API's work.
Chapter 4 gives description ITP and PECI. It also explains how ITP console works.
Chapter 5 describes what is array freeze dump, scan dump and register dump. It also explains the steps used to access these dumps.
Chapter 6 describes the work done with the result.
Chapter 7 concludes what is the output of whole project.

1.7 Summary

This chapter gives the background details of the thesis with the motivation and objective in detail. This chapter also gives details of the server on which the project is tested. The thesis organization gives the brief of every chapter.

Chapter 2

Encrypted AFD and Scan Dump Tool Description

2.1 Background

External debug by the customer requires extracting architectural and hardware state under error conditions. Historically this state has been limited to access to registers exposed in the platform EDS documentation and in a few cases exposure of internal outstanding request buffers. Internal debug teams have access to all of the registers which include those not exposed to customers, all internal arrays, debug scan flops, internal trace data, and all internal flop state. If a customer is unable to root cause their issue with the state extraction mechanisms at their disposal, they contact Intel teams to use the internal debug tools to extract the information they need. Running internal tools to customer systems at remote customer sites slows down the customer debug throughput and puts a burden on the Intel teams needed to arrange the tool attachment and running the tool itself. So, the tool is to needed to do all things faster and efficient way. The encrypted array dump tool allows the customer to extract internal array and debug scan chains along with the exposed registers without having an Intel engineer involved. However, to protect IP, the data they are able to extract will be encrypted and requires an Intel engineer to decrypt and evaluate it. The capability to extract the array and scanout data on their own represents an important improvement in debug throughput. The tool will extract data into a format that will be compatible with the internal debug toolset so all internal tools will be able to operate on the captured information as if it were captured using internal tools vs the customer scripts. As shown in figure 2.1, the unlocked ITP flow is the one which was done previously but after the eAFD tool developed the flow changes to eAFD tool flow.

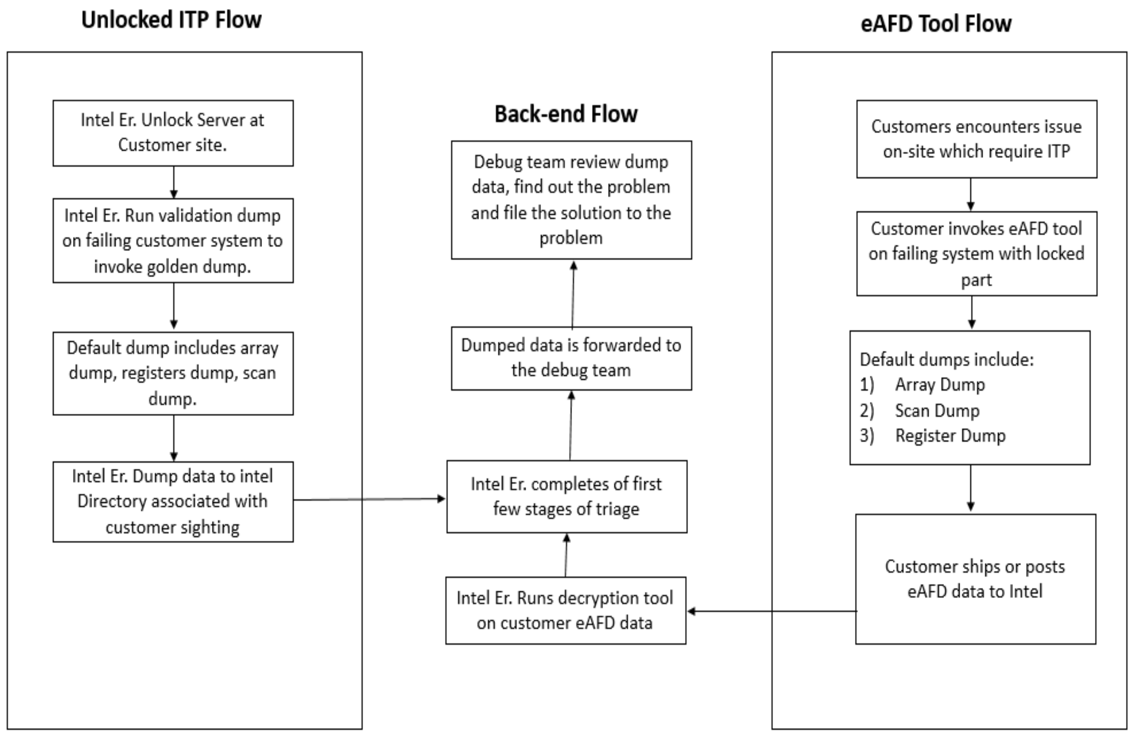

Unlocked ITP Flow
Intel Er. Unlock Server at Customer site.
Intel Er. Run validation dump on failing customer system to invoke golden dump.
Default dump includes array dump, registers dump, scan dump.
Intel Er. Dump data to intel Directory associated with customer sighting
Back-end Flow
Debug team review dump data, find out the problem and file the solution to the problem
Dumped data is forwarded to the debug team
Intel Er. completes of first few stages of triage
Intel Er. Runs decryption tool on customer eAFD data
eAFD Tool Flow
Customers encounters issue on-site which require ITP
Customer invokes eAFD tool on failing system with locked part
Default dumps include:
1) Array Dump
2) Scan Dump
3) Register Dump
Customer ships or posts eAFD data to Intel

Figure 2.1: Comparison of Unlocked Flow (Without VCU) and Locked flow (With VCU)

2.2 Tool Description

The encrypted static data extraction tool is Customer Script (CScript) command flow initiated on a Python command layer driving an ITP connected via JTAG to a platform. Many of the CScript commands in this flow will themselves call VCU API functions to perform trigger setup, data extraction, and data encryption. This internal toolset is composed of python scripts and processor state definition files needed to tell the script how to extract and decorate the data for each array and scanout chain.

2.3 Tool Flow

There are four stages in tool flow (Figure:2.2):

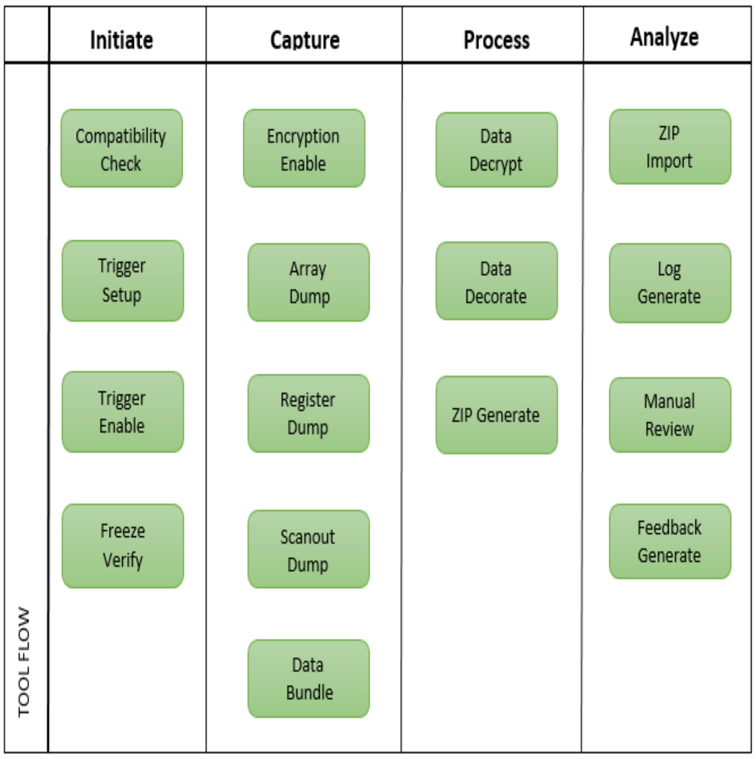


Figure 2.2: Tool Flow

a. **Initiate Stage :** Make sure that the tool has all the resources it needs to run successfully, then setup and trigger a freeze. Steps included in the initiation steps are:

   - Compatibility Check: It is done by running CScript commands to verify the tool can run successfully and includes checking for software versions, firmware versions, hardware compatibility, and tool connectivity.
   - Trigger Setup: It is done by running CScript commands to set up all hardware resources properly to ensure a successful trigger and freeze before actually enabling the trigger.
   - Trigger Enable Setup: It is done by running CScript commands to enable the trigger and freeze actions.
   - Freeze Verify: It is to verify that the trigger has occurred and the requested arrays and scan agents have frozen.

b. **Capture Stage :** Run the CScript commands to gather information needed for proper processing of the data, then capture and bundle the data to be sent to Intel for processing. Steps included in the capture steps are:

   - Encryption Enable: Enable encryption is necessary before dumping scan or arrays to secure confidential data.
   - Array Dump: Dump requested array data or all arrays.
   - Register Dump: Run CScript commands to dump all available registers.
   - Scanout Dump: Dump all scan chain data.
   - Data Bundle: Bundle all of the above encrypted data into a format required for proper processing at Intel.

c. **Process Stage :** Decrypt the bundled data and processes into a format that can be imported into the internal toolset. Steps included in the process steps are:

- Data Decrypt: Uses a utility to decrypt the scan and array data.
- Data Decorate: Uses a utility that uses the definition files to generate internal tool importable data from the decrypted scan and array data.
- Zip Generate: Uses a utility to generate zip files from the register dumps and decorated scan and array data into a format that can be imported directly into internal tool for analysing.

d. **Analyze Stage :** Import the debug data into the internal toolset, then analyze the data using internal log generators such as watch windows, debug scripts, and manual debug methodologies. Based on the debug results, generate feedback guidance to be submitted back to the customer. Steps included in the analyze steps are:

- Zip Import: Zip file processed in the process stage is imported into internal tool.
- Log Generate: Use internal toolsets to generate logs which can be used for debugging.
- Manual Review:Use internal toolsets to manually inspect the stub model.
- Feedback Generate: Generate and deliver debug feedback and recommended next steps based on the analysis.

2.4 Summary

The chapter gives the details of the tool which is used for extraction of array and scan chains. Before and after tool development flow for debugging is also discussed. Description of steps of the eAFD tool is also given.

Chapter 3

Validation Control Unit

The Validation Control Unit (VCU) is intended to facilitate fast and unobtrusive processor validation under various use conditions including power-on and post-silicon debug, survivability, electrical validation, manufacturing, and RAS manageability. It consists of a dedicated on-die microcontroller with associated firmware, ROM, RAM, and an IO space that serves as the interface to the rest of the Server processor device. Software is able to request the VCU to coordinate different validation tasks, such as program various breakpoints, micro breakpoint trigger events, extract stored data, deliver validation information and provide different levels of access [3]. Request is sent to VCU through ITP (In Target Probe) via JTAG or through PECI (Platform Environment Control Interface) using APIs and console.

3.1 Flow of VCU API working

In figure 3.1,a flow diagram for a method of servicing a request from software which utilizes VCU is illustrated. First of all, a request is generated using an Application Programming Interface (API) in response to executing a software program, such as a test and debug program from a third party vendor. The software program is written to fulfil the demand specified by the API, such as calling conventions recognized by the API. The API is working as an interface between software and VCU hardware. As an example, code stored in memory of a VCU microcontroller is executed in response to the request. In this scenario, the code stored in the VCU may include libraries, routines, data structures, object classes, protocols, etc. In flow it is mentioned that the API request is executed based on a security level. A VCU may include a number of levels of secure access, each of which allows different levels of access to hardware. For example, each level may be associated with an encrypted passcode. So when the software program begins executing, it may provide a secure passcode to unlock its security level, which designates what features the software program is able to access. Therefore, according to the levels of access predefined by the VCU, the information get dumped into the file.

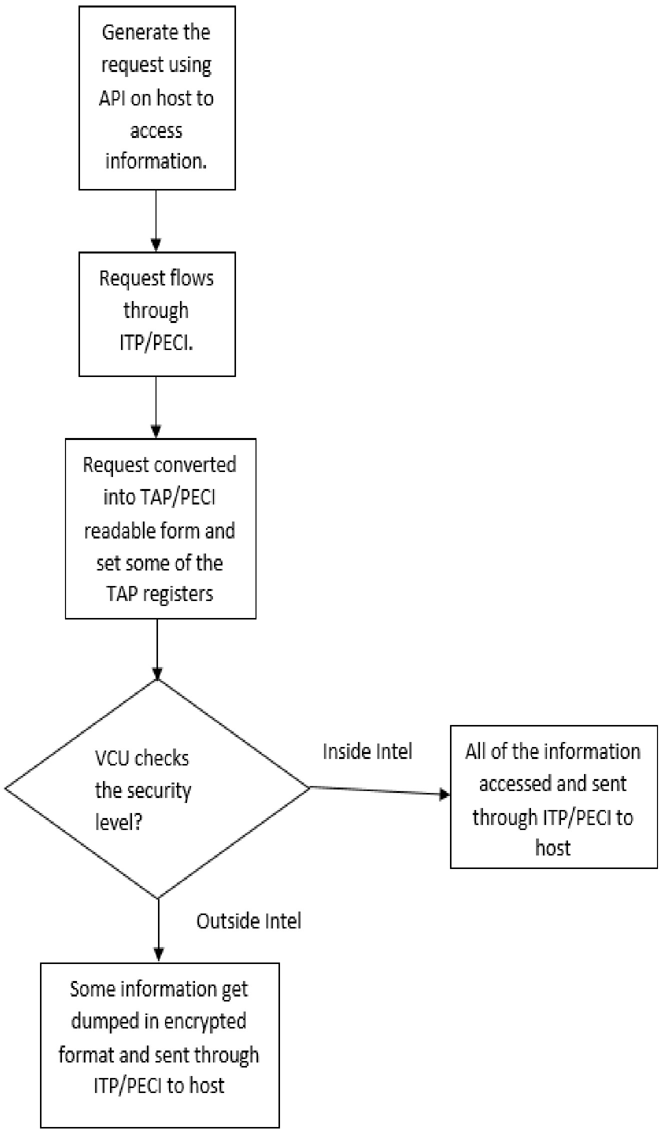

Generate the request using API on host to access information.
Request flows through ITP/PECI.
Request converted into TAP/PECI readable form and set some of the TAP registers
VCU checks the security level?
Inside Intel
All of the information accessed and sent through ITP/PECI to host
Outside Intel
Some information get dumped in encrypted format and sent through ITP/PECI to host

Figure 3.1: Flow of VCU API Working

3.2 VCU API Sequence

Sequence used by VCU API for accessing data are (Figure 3.2):

a. Opening of sequence with a predefined value to tell VCU that process of reading of array/writing into registers is going to start.

b. Parameters are set which will tell which array to read, how many bits of array to read etc.

c. Reading of data is done using the function predefined in the library

d. Closing of sequence which will tell reading is finished.

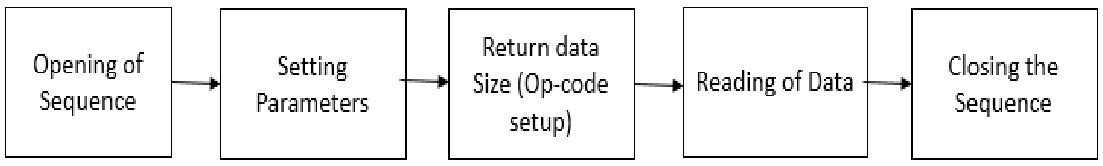


Figure 3.2: VCU API Sequence

3.3 Summary

This chapter describes the main unit (VCU) used for extracting data in encrypted form to protect Intel IP. The flow of how VCU works is also described. VCU API Sequence which every function of VCU script follow is also given.

Chapter 4

In Target Probe and Platform Environment Control Interface

4.1 In Target Probe (ITP)

Standard software debuggers are attached to a running process and can stop execution at any point. After halting the process, instructions, their memory locations, and data currently stored in variables can be looked at. An In-Target Probe (ITP)(4.1) is a hardware device that stops the processor and provides visibility to things like registers and variable values. It's a costly piece of hardware but permits to see at several things on the processor and motherboard. ITPs are generally used to debug hardware (processors and motherboards).ITP is a device attached to computer hardware and microprocessor design to regulate a target chip or similar ASIC at the register level. It typically permits full control of the target device and permits the engineer to access to individual processor registers, program counter and instructions among the device. It permits the processor to set breakpoints.ITP is completely different from In Circuit emulator (ICE) as ITP uses the target device to execute instead of substituting for the target device.

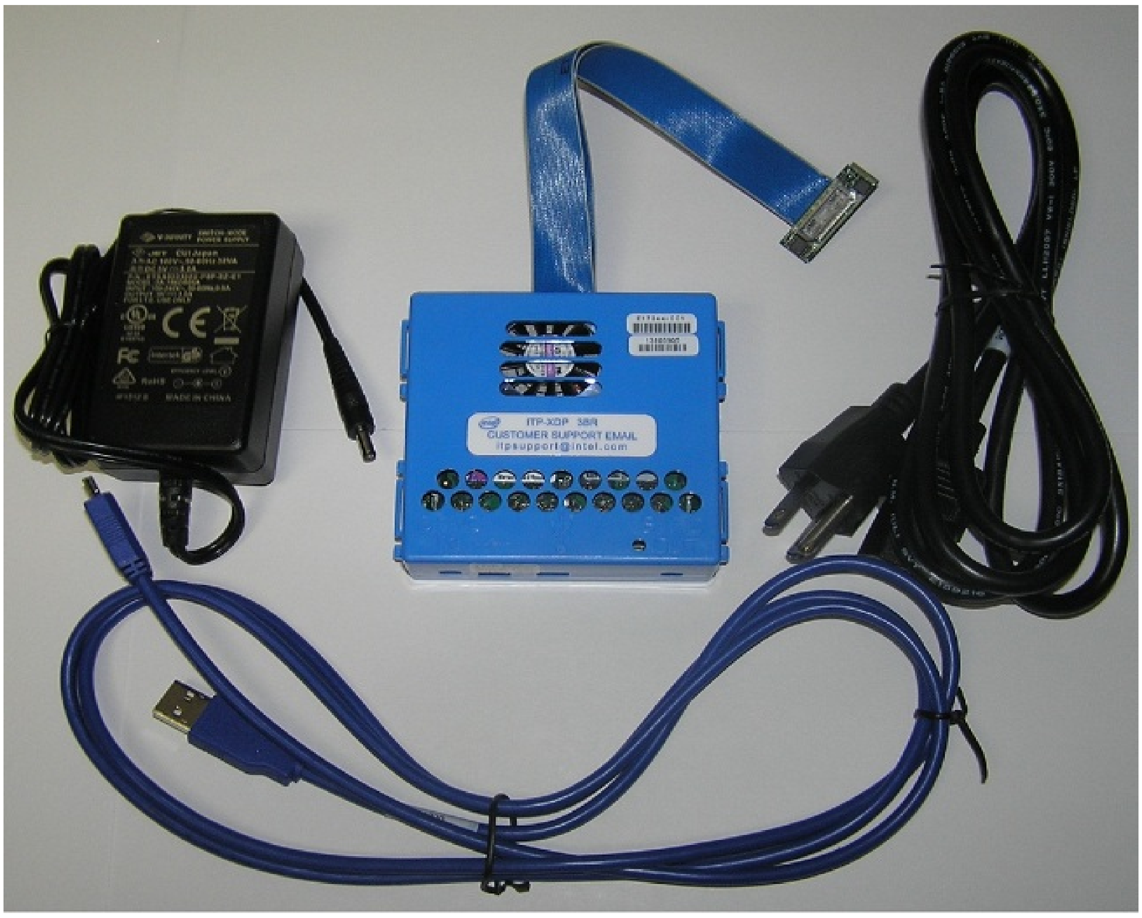


Figure 4.1: ITP with Cables

The ITP is a specialised JTAG master that interfaces into chipsets through a rigorously routed non-public scan chain on the target system. The first operations of the ITP is to provide system interface, execution interface and JTAG interface to the target system. The system interface tells the tool for debug if power, clock and access state are available in the target system. The execution interface is employed to coordinate debug activities with the present execution state of the agents hooked up to the debug port. The JTAG interface permits for question and editing of registers and state inside the agents hooked up to the scan chain.[4].

In the apparatus related to this thesis, the ITP is connected between host and platform (the target machine) and converts API command coming from host to data understood by JTAG. JTAG pins are attached to VCU, which on receiving commands from JTAG sets registers and do the function of accessing data for debugging (Figure 4.2). There are many components integrated on the platform like PCU, PCIe slots, etc. as shown in figure 4.2.

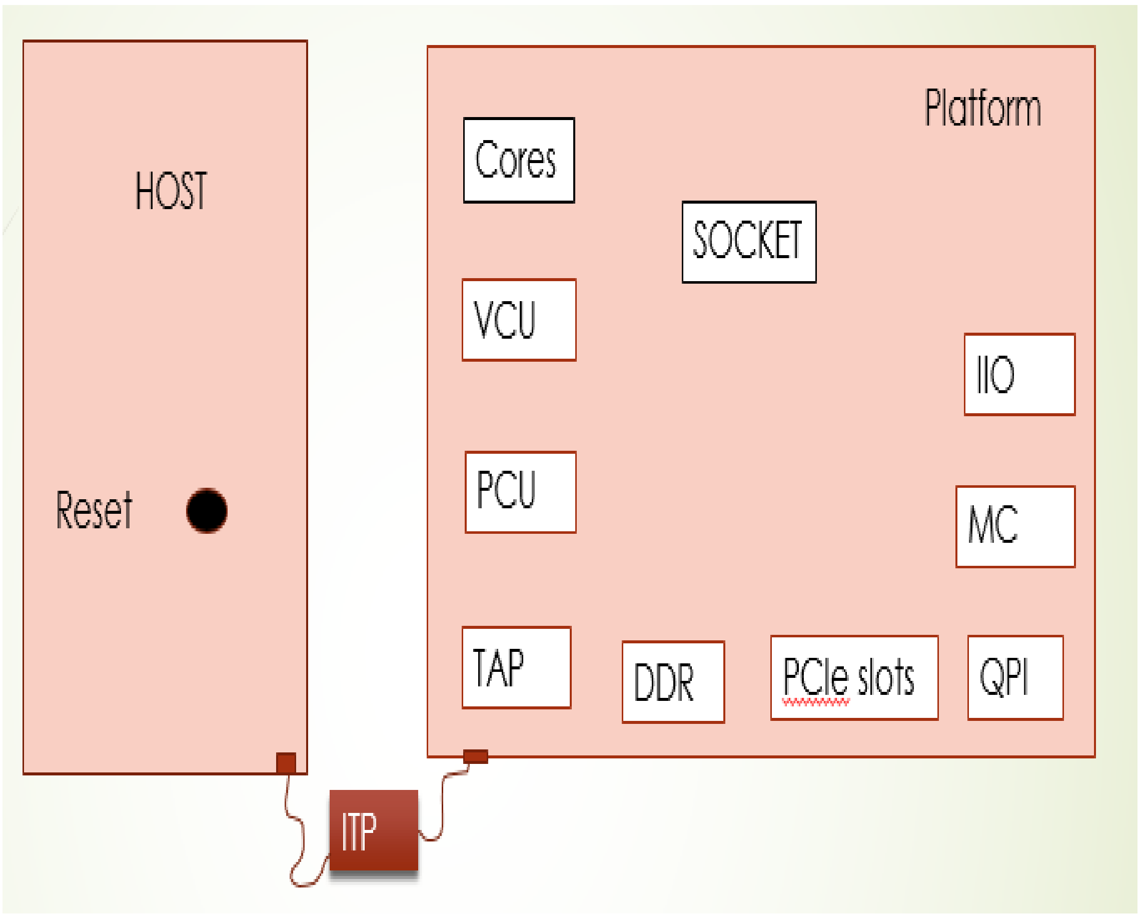
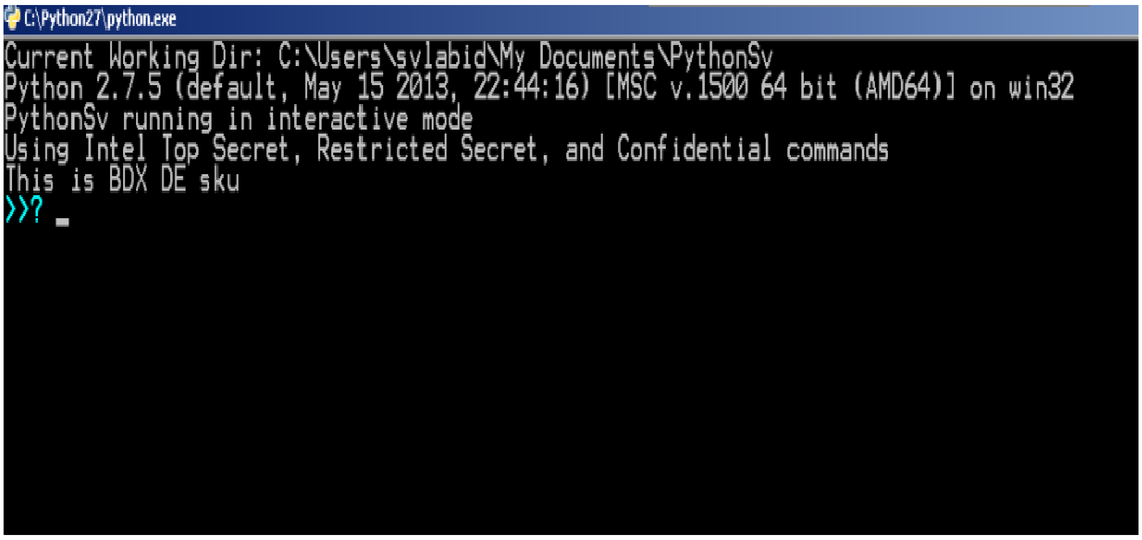


Figure 4.2: Connection between host and platform using ITP

4.2 ITP with Python Console

The console used to send API commands is ITP console which has python features incorporated in it. Figure 4.3 shows the console window used for the thesis to dump the data under different error scenario using JTAG. The API is imported and different parameters are passed to get required data in encrypted form. The property of the console and commands used in the thesis are:

- ITP console uses the interactive mode of python.
- It is easy to import the script and run the function which is written inside the script.
- Script is imported using import function on the interactive prompt of python. **Eg:** import VCU_Script as vcu
- For calling any function within script use: script_name.function_name(arguments) **Eg:** in above eg. Script is imported as vcu : vcu.afd(arguments)



```
C:\Python27\python.exe
Current Working Dir: C:\Users\svlabid\My Documents\PythonSv
Python 2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)] on win32
PythonSv running in interactive mode
Using Intel Top Secret, Restricted Secret, and Confidential commands
This is BDX DE sku
>>? _
```

Figure 4.3: ITP Plus Python Console

4.3 Platform Environment Control Interface (PECI)

The PECI is a one-wire interface that provides a communication channel between a PECI client (the processor) and a PECI master (the PCH).

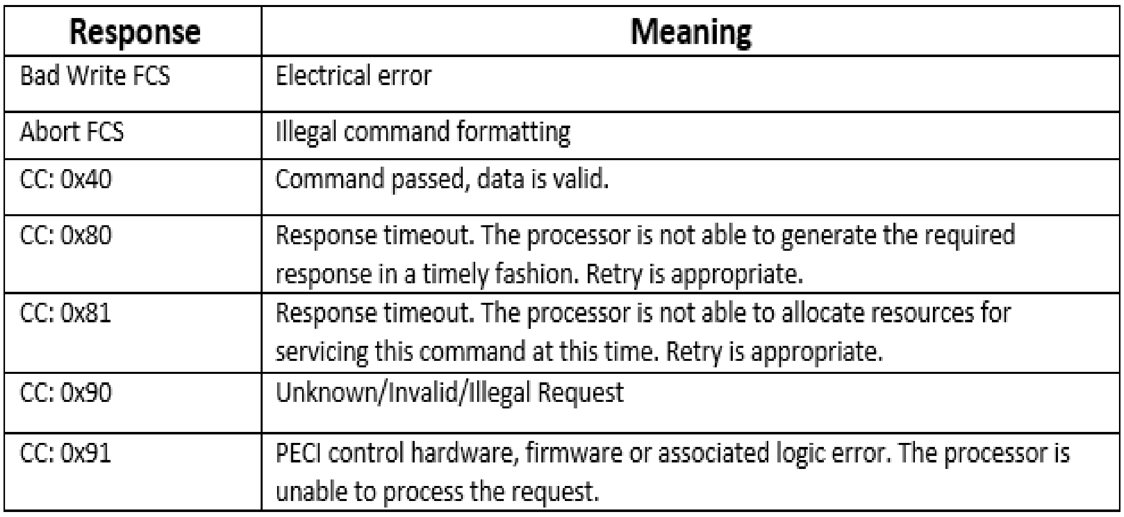
- Supports operation from 2 Kbps to 2 Mbps data transfers.
- Services provided by PECI includes CPU thermal and power information, control functions for power limiting and access to Machine Check registers.

Host talks to PECI through StartDebugger hardware. Commands are send in the same sequence as they are send through ITP to JTAG but here the difference is command are send through StartDebugger to PECI.

PECI is used to access machine check registers with the same VCU sequence as ITP via JTAG is using. It is also validated to access registers after different error scenarios like crash core and Fatal error. The commands used to access registers through PECI are[8]:

- RdPkgConfig() : This command provides read access to the various registers within the processor and returns the data which is specified by index and parameter fields. The index field contains the encoded data for the requested service and is used with the parameter field to specify the exact data being requested. This command returns dword(32bit) data everytime. The response (Responses are data which tells whether the command executed correctly or not) returned with data are different for different scenarios are illustrated in figure 4.4
- WrPkgConfig() : This command provides write access to the within the processor. This command writes data to the processor which is specified by index

and parameter fields.The response returned with data are different for different scenarios are illustrated in figure 4.4



Response	Meaning
Bad Write FCS	Electrical error
Abort FCS	Illegal command formatting
CC: 0x40	Command passed, data is valid.
CC: 0x80	Response timeout. The processor is not able to generate the required response in a timely fashion. Retry is appropriate.
CC: 0x81	Response timeout. The processor is not able to allocate resources for servicing this command at this time. Retry is appropriate.
CC: 0x90	Unknown/Invalid/Illegal Request
CC: 0x91	PECI control hardware, firmware or associated logic error. The processor is unable to process the request.

Figure 4.4: Response after RdPkgConfig() and WrPkgConfig() commands execution

4.4 Summary

This chapter gives the details of the hardware (ITP and PECI) connected between host and platform to access data at the time of error scenario. It also describes in brief how interactive python console helps in debugging and running the scripts with the knowledge of responses if script fails to extract data.

Chapter 5

Data Extraction Steps

Data accesses is done at the time of failures which helps in debugging. There are two ways for accessing data: Internal and External. Internal access is done through directly talking to JTAG/PECI to set value to registers to access/read data without encryption. External access is the access of data in the encrypted form using a controller unit via JTAG/PECI. Dump means accessing a value or dumping the values in a file.

- Array Freeze Dump
- Scan Dump
- Register Dump

5.1 Array Freeze and Dump

Arrays are structures in the RTL. The sizes of array are different for different arrays and mostly all arrays contain information of state. In order to accurately capture array data, the array needs to first be frozen to prevent the data from being modified during the data extraction process. This is known as freezing of array. The controller registers, named as Local Data Access Test (LDAT) controllers, are set to read data of the arrays. LDAT is directly connected to the arrays. Different set of arrays have different LDAT controllers. To set LDAT controller registers, JTAG (internally) or VCU (externally) can be used. As shown in figure5.1, every LDAT controller is attached to two or more arrays which shows if we want to dump any data in the array we need to set the particular LDAT controller registers.

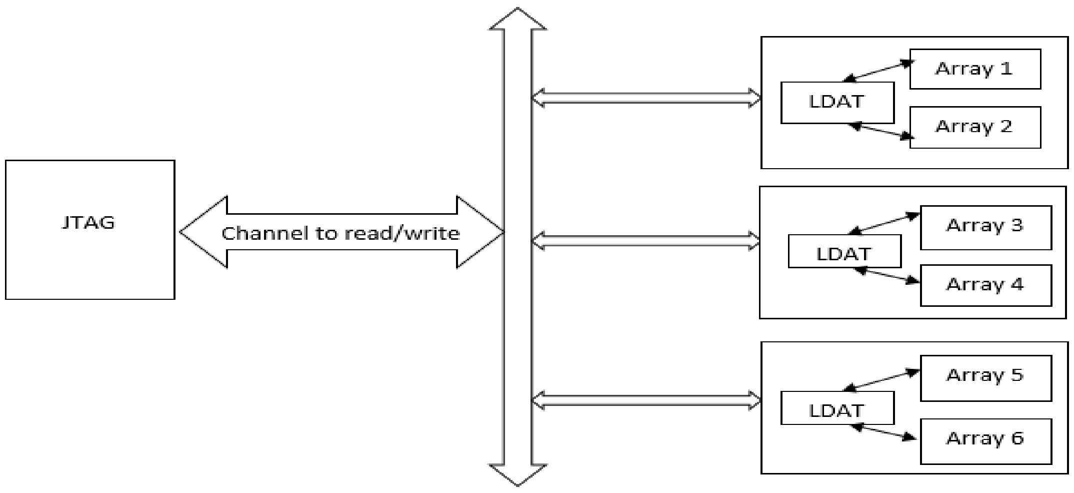


Figure 5.1: Access of array using JTAG

Every array used for debugging responds to a freeze command which is broad-casted in the processor. After receiving this command,write logic of array is disabled immediately which freezes its contents even when the processor clock remains running. There are two ways of freezing the array:

1.) Debug Mode: This mode stops a running processor and places it in an interactive testing mode. After testing operations have been completed, the processor can be released into normal execution mode again[5].

2.) Internal breakpoint mechanism: An internal breakpoint mechanism sets a breakpoint at particular instant which freezes the content of array. This mechanism can be set up to respond to any of a variety of internal micro-architectural events.

5.2 Scan Dump

These are internal flops instantiated in the processor logic to capture state specifically for debug purposes. Scan-out technique provides facility to see data of internal nodes in the design. Scan-out can work in snapshot mode. It takes a single sample of the key processor state, saves it in shadow flip flops and shifts it out through the JTAG port while the processor is running. Shadow flip-flops or latches are used to provide a non-destructive scan out ability that preserves the existing system state. This mode is used while the processor is running at full clock frequency, without disturbing its operation. As shown in figure5.2,each flip-flop has its shadow flip-flop. When the scan dump command is executed, chain is formed and data get stored in shadow flip flop. The data stored in shadow flip flop is taken out by inserting dummy data in.

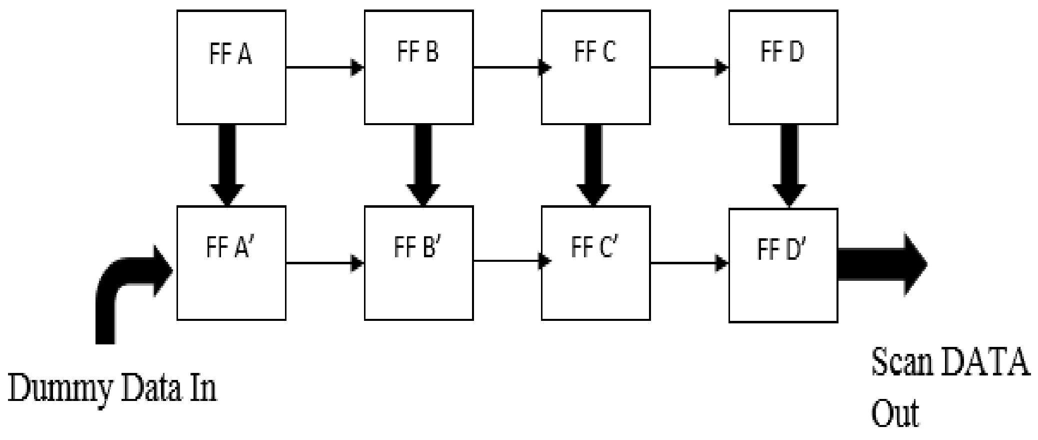


Figure 5.2: Scan Dump

5.3 Register Dump

Some of the registers are exposed to customer as part of the EDS documentation. Register values are important to determine the architectural configuration being applied and to determine what errors are being indicated at the point of failure. The CScripts already have mechanisms to read out these registers as part of their existing functions. Some of them are control registers, memory input output registers and status registers.

5.4 Steps for dumping data

a. **Array Freeze Dump:**

Encrypted array dump is done using VCU via JTAG through ITP. Scripts are used to set LDAT controller and VCU registers to start array dump after freezing the data in it. CScipt is called to start array freeze dump at the time of error. The AFD script is divided into two parts.Following are things done by front-end script (Figure 5.3)(given to customers) :

(1) One by one array will be taken from the list defined in the script.

(2) For individual array the repeated steps will be:

- Freezing using VCU API (breakpoint set)
- Encryption enabling using VCU API
- Dump using VCU API

(3) Wrap all the encrypted dumped data in a zip folder.

Following are things done by back-end script (Figure 5.4)(internal at Intel) :

(1) Load the zip file into the internal tool for decoration , so that it can be compared easily.

(2) Decryption of data loaded into the tool.

(3) Put all the log generated in the file for manual comparison.

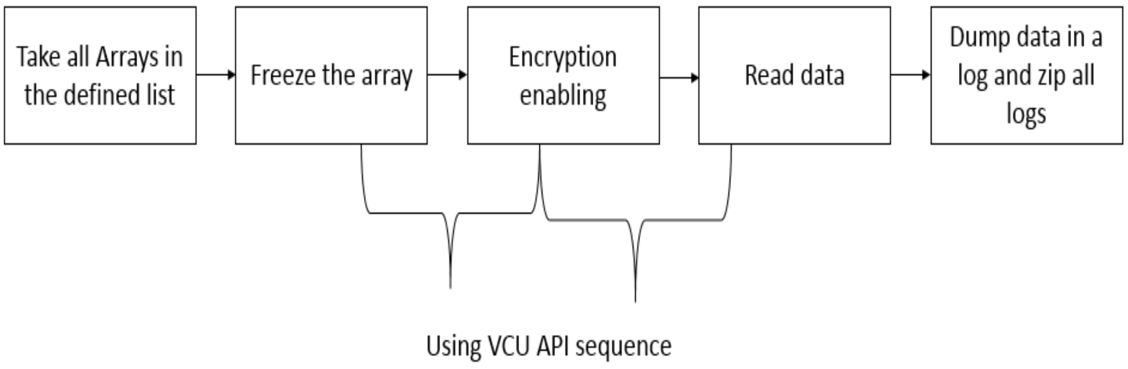

Take all Arrays in the defined list
Freeze the array
Encryption enabling
Read data
Dump data in a log and zip all logs
Using VCU API sequence

Figure 5.3: Front-end Flow

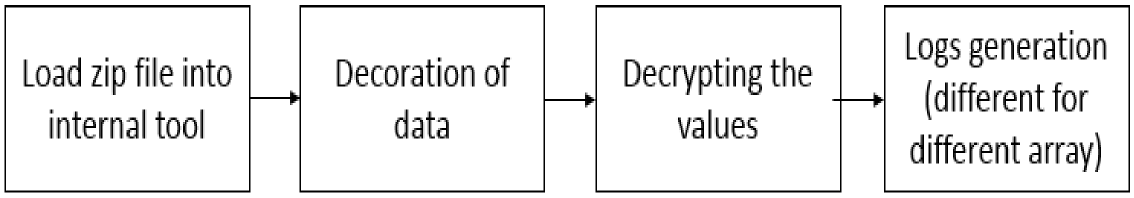

Load zip file into internal tool
Decoration of data
Decrypting the values
Logs generation (different for different array)

Figure 5.4: Back-end Flow

b. **Scan Dump:**

Steps to implement scan dump through VCU are similar to VCU API sequence of array dump. The registers which are set for scan dump are different from the one which are set for array dump. Linear scan chains are formed for scan dump as shown in figure5.2. In scan dump first we need to select chains and units which we want to dump and then freeze all chains as shown in fiure 5.5.

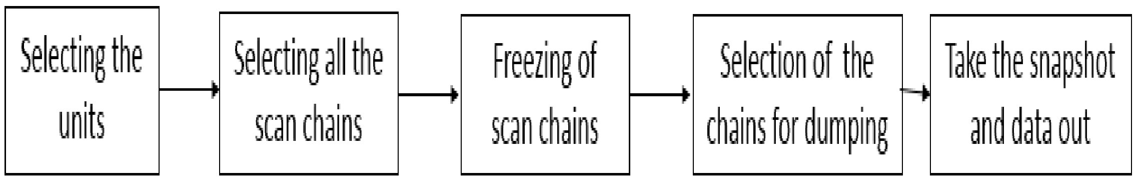

Selecting the units
Selecting all the scan chains
Freezing of scan chains
Selection of the chains for dumping
Take the snapshot and data out

Figure 5.5: Scan Dump

c. **Register Dump:**

Register dump can be done in several ways i.e. can be directly read setting VCU as access or using VCU API.

5.5 Error Scenarios

Several error scenarios are generated to dump data to debug further. Some of the errors are:

a. **Crash Cores:**When the cores does not respond to any command given by user then crash core error occurs. It happens due to invalid data written into registers or several instruction running simultaneously leading to crash core and hung the system. After core crashes a crash core dump (using VCU API) is done which consists of the registers in which records the working memory of a computer program when the program has terminated abnormally (crashed). The processor registers which are dumped in the crash core scenario include the program counter and stack pointer, memory management information and other processor and operating system flags and information. Core dumps are used to assist in analyzing and debugging errors in computer programs. Array dump, scan dump and register dump also checked in this scenario for debugging.

b. **DMI Error Injection:**If there is a break in communication between different components, it is also considered to be an error. In this scenario debugging is done through array, scan and register dump.

c. **Thermal Error:**When processors get heated up, it halt its operation. Due to some operation which goes to infinite loop the processor gets heated up and error occurs in the system.

5.6 Summary

This chapter includes the work done as the part of the thesis/project. Details of different kinds of data extracted using the VCU in encryption form and validating the data after decryption is given. The steps did for extracting data with block diagrams is also described. The error scenarios in which the array , register and scan data are validated is also described in this chapter.

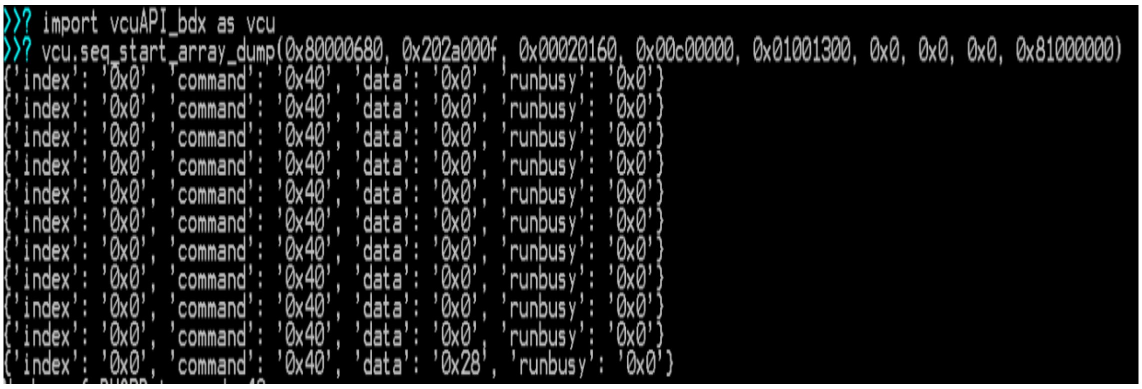
Chapter 6

Result

The VCU microcontroller assembly coding was already written for reading the data. This thesis is based on writing the script for dumping data through VCU, checking the correctness of encryption and decryption of data and validating the data in different error scenarios. For validation of data knowledge of some basic JTAG commands which can directly access the data without any VCU intervention was important part. This is called to be an internal accessing of data. This data is always correct until unless ITP is not screwed up. External data which is accessed through VCU is compared to this data for validation. If the data matches then VCU API is working perfectly. Parameters are passed in the function defined in the API to tell VCU which data to dump. This data ,as discussed in previous chapters, is dumped in encrypted form in different error scenarios and so can be used for debugging after decryption.

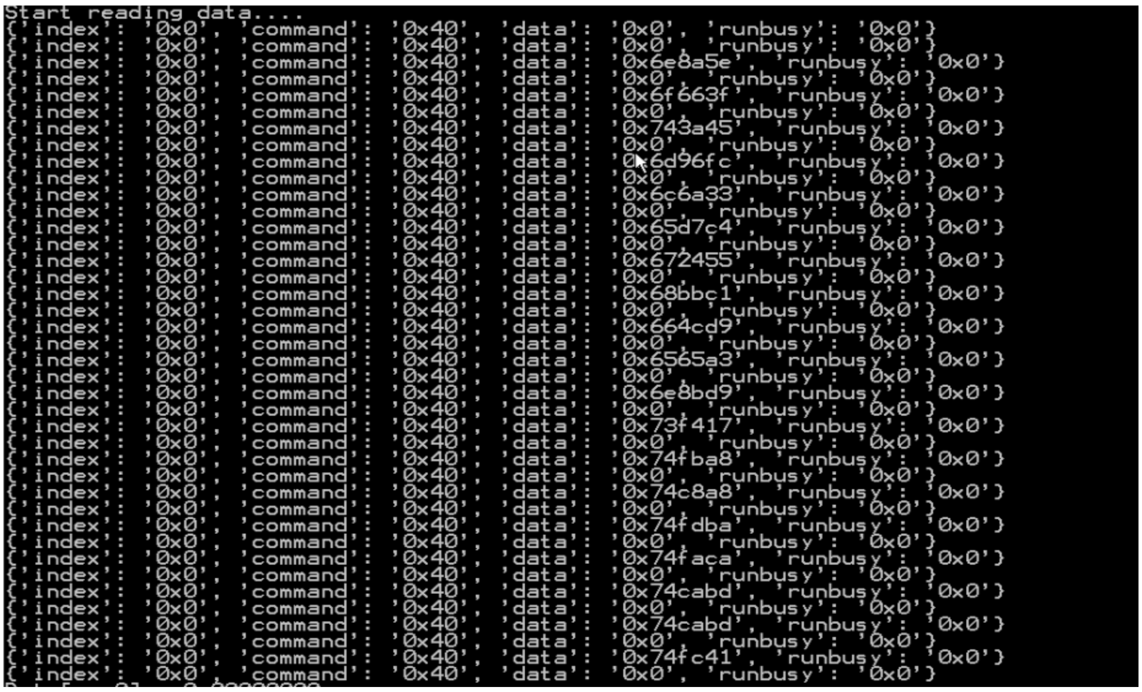
Scripts are written to access array and scan dump data in encrypted form. These scripts are checked under normal conditions and after spoofing different error scenarios. The encrypted data is then decorated and decrypted with internal tool. This decrypted data is then compared with the golden values and then analyse for debugging. The validation of this data is necessary before going further.

Figure 6.1 shows how the VCU API script is imported on Python Console and array dump is done. Figure 6.2 represents the output of array dump after settingg the parameters. Figure 6.3 shows two logs. One of them is taken using the script written for tool and other one is taken using the internal script which uses JTAG directly to access data. Figure 6.4 show the comparison of scan results using VCU with JTAG values.



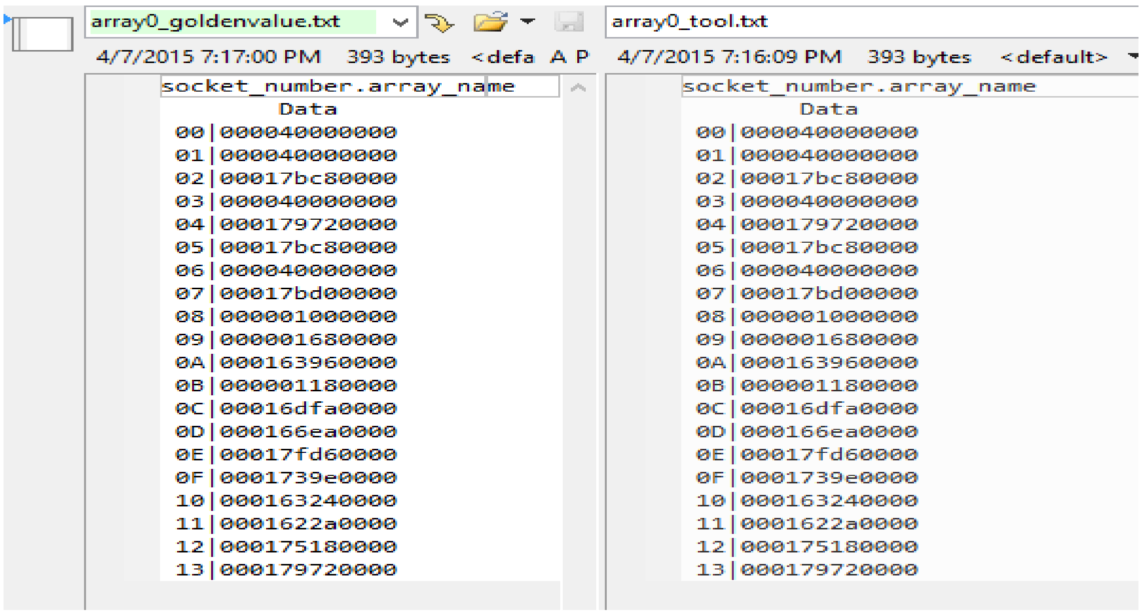
```
>>? import vcuAPI_bdx as vcu
>>? vcu.seq_start_array_dump(0x80000680, 0x202a000f, 0x00020160, 0x00c00000, 0x01001300, 0x0, 0x0, 0x0, 0x81000000)
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x28', 'runbusy': '0x0'}
```

Figure 6.1: Importing script and running a function



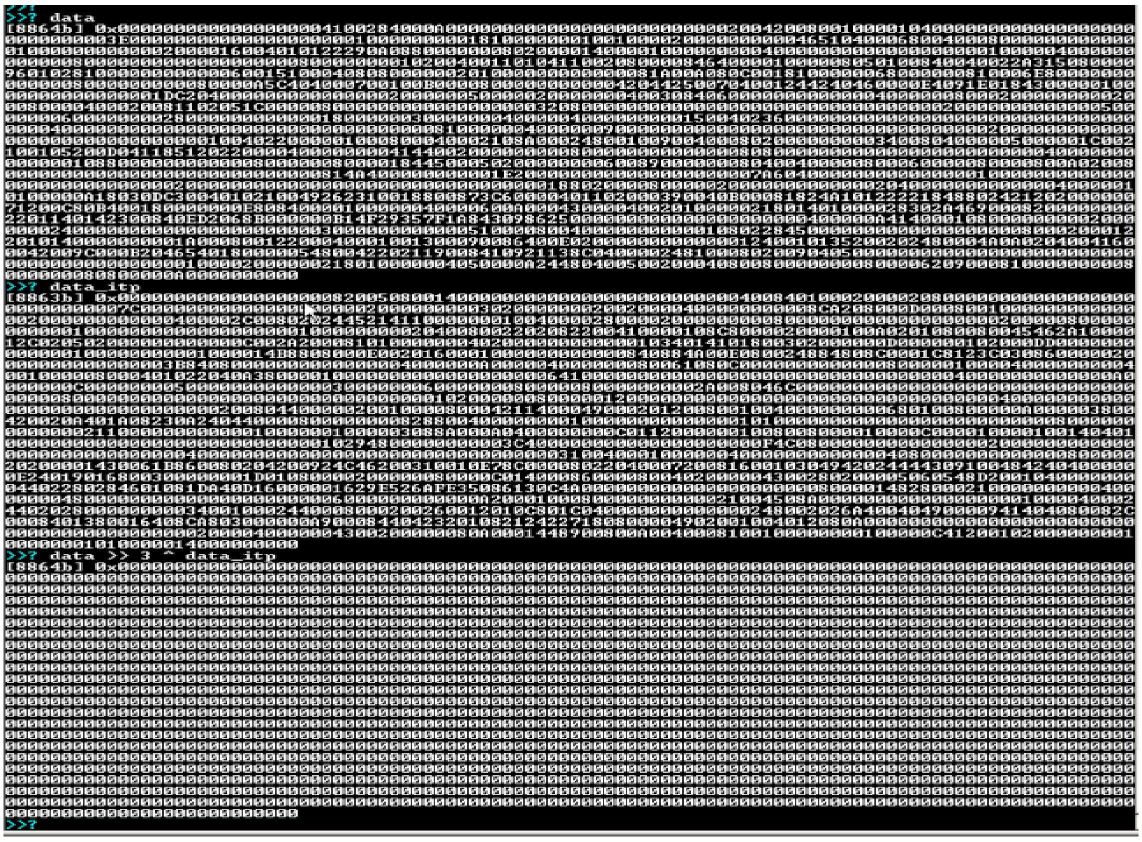
```
Start reading data....
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6e8a5e', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6f663f', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x743a45', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6d96fc', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6c6a33', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x65d7c4', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x672455', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x68bbc1', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x664cd9', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6565a3', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x6e8bd9', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x73f417', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74fba8', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74c8a8', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74fdba', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74faca', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74cabd', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74cabd', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x74fc41', 'runbusy': '0x0'}
{'index': '0x0', 'command': '0x40', 'data': '0x0', 'runbusy': '0x0'}
```

Figure 6.2: Output data of an array dump after running script



```
array0_goldenvalue.txt                      array0_tool.txt
4/7/2015 7:17:00 PM  393 bytes  <defa  A P  4/7/2015 7:16:09 PM  393 bytes  <default>
socket_number.array_name                    socket_number.array_name
         Data                                        Data
 00|000040000000                             00|000040000000
 01|000040000000                             01|000040000000
 02|00017bc80000                             02|00017bc80000
 03|000040000000                             03|000040000000
 04|000179720000                             04|000179720000
 05|00017bc80000                             05|00017bc80000
 06|000040000000                             06|000040000000
 07|00017bd00000                             07|00017bd00000
 08|000001000000                             08|000001000000
 09|000001680000                             09|000001680000
 0A|000163960000                             0A|000163960000
 0B|000001180000                             0B|000001180000
 0C|00016dfa0000                             0C|00016dfa0000
 0D|000166ea0000                             0D|000166ea0000
 0E|00017fd60000                             0E|00017fd60000
 0F|0001739e0000                             0F|0001739e0000
 10|000163240000                             10|000163240000
 11|0001622a0000                             11|0001622a0000
 12|000175180000                             12|000175180000
 13|000179720000                             13|000179720000
```

Figure 6.3: Array Data Comparison - VCU API output with JTAG output



```
>>?
>>? data
[8864b] 0x00000000000000000000041002840000A000000000000000000000000000002004200800100001040000000000000000000000
0000000003E0000000000000000000000100000000018100000000100100002000000000000465104000068004000800000000000000000
0100000000000000200001600040101222290A088000000000802000014000010000000000400000000000000000000010000004000000
0000000800000000000000000000000000800000000010200400110104110020800000846400000100000080501008400040022A315080000
96010281000000000000000060015100004080800000000201000000000000000081A00A080C001810000000680000000810006E80000000
00000008000000000000080000A5C40400007000100B0000080000000000000042044250007040001244240460000E4091E01843000000100
000000000000000001DC20400000000000000000000200000000500000020000000000400300840060000000000000004000000080000200000020
008000004000200081102051C000000080000000000000000000000320800000000000000000000000000000200000000000000000000500
000006000000000000280000000000000000000180000000300000000004000000400000000000015000402360000000000000000000000000000
000040000000000000000000000000000000000000000000000810000000400000009000000000000000000000000000200000000000000000
00000000000000000000000100402200000010000800040000210 8A00002480010009004000080200000000000340008040000050000001C002
1000105200D004118512022000004000000000000041440020000000000000800000000000000000000000000000000000000040000000000
0000000100880000000000000000008000000000800000184450000502000000000000600089000000000000804000000804000400000000800060000000800000000800A02008
0000000000000000000000000000000000814A400000000000001E20000000000000000000000000007A6040000000000000000000001000000000000000000
0000000000000000000020000000000000000000000000000000000000000000018802000000800000000020400000000000000000040000000001
0100000A18030DC30004010210004926231000188000873C600000401102000039000400B000081824A1012222184880024212020000000
71200C800B400018000000000E800840000001000000040000060000A000043000000400020100000000218001401000000283020A4690000820000000000
22011400142300840ED2068B000000000B14F29357F1A84309862500000000000000000000000000000000004000000A41400001080000000000000002000
0000240000000000000000000000000003000000000000051000000800040000000000000001080022845000000000000000000000080000200012
201014000000000001A000080012200004000100130000900086400E02000000000000000012400010135200020249000004A0A0204000041600
0042009C000B204654018000000548000422021190008410921138C040000024810000802009040050000000000000000000000000000000000
000000000000000000001000020000000021801000000000405000000A24480400050020000400800000800000062090000810000000000000
00000008000000000A0000000000000
>>? data_itp
[8863b] 0x0000000000000000000000082005080001400000000000000000000000000000000040084010002000002080000000000000000000
000000000007C000000000000000000000000000000000000020000000000030200000000200200004000000000008CA20800D00080010000000000000000
002000000000000000400002C00802024452141100000000100400002800002000000000000800000000000000000000200000800000
000000100000000000000000000000000100000000200400800220208220041000010 8C800002000001 00A020100800045462A10000
12C02050200000000000000000000C002A2000081010000000000400000000000000000001034014101800300000000000000000D000001020000DD000000
0000001000000000000010000014B808000E0020160001000000000000000000000840884A00E080024884808C0001C8123C030860000020
0000000000000003B840800000000000000000400000000A0000040000000008061080C0000000000000000000008000001000040000000004
00100000800040102204A38000010000000000000000000000006410000000000000000000000000000000000000000400000000000000000A0
000000C000000000050000000000000000000030000000600000000800000000800000000000000002A008046C000000000000000000000000000000
000000800000000000000000000000000000000000000001020000008000000120000000000000000000000000000000000000000000000000
00000000000000000002008044000000200100008000421140004900020120080010040000000000006801008000000A000003800
420020A401A08230A24044000080000000000000288004000000000100000000000000000101000000000000000000000000000000000000
000000002110000000000000100000001000003088A000A04000000000000C0112000000100800800001000C000010000100140401
000000000000000000000000000000000010294800000000003C40000000000000000000000F4C08000000000000000002000000000000
000000000000000000000000000000000000000000031004000100000400000000000000000000000000000000000000000000000000000
202000001430061B860080204200724C46200310010E78C0000802040007200816001030474202444307100484240400000
0E24019016800300000001D01080000200000008000C014000860000800402000004300280200005060548D20010400000000
0440228028460108 1DA40D16000001629E526AFE350861 30C4A00000000000000000000001482800021000000000400
00000480000000000000000000000600000000000000A200010008000000000000000210045 08A000000000000000000001000040002
44020280000000000003400100024400080002002600120100C801C0400000000000000248000202 6A4000 40470000941404080082C
00084013800016408C A8030000000970008440423201 08212422271808000497020010040120800000000000000000000000000000
000000000000000000002000040000000430020000000800A000144870080 0A004000810010000000001000000C41200102000000000001
000000010100000140000000000
>>? data >> 3 ^ data_itp
[8864b] 0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000
>>?
```

Figure 6.4: Scan Dump Data Comparison - VCU output with Golden Values

Chapter 7

Conclusion

Postsilicon Validation is considered to be an important step in validation of any chip. So, DFD is included as a part of postsilicon validation which helps in debugging the chip even after it has been shipped to customers. This thesis is intended towards giving the visibility to customers for dumping data, at time of error scenarios, which can be used for debugging. As we know providing confidential data to customers may lead to exploitation of the companys IP, so a microcontroller unit is embedded into the design, named as VCU, which will dump the data and take care of the encrypting the data before dumping.

In this project, the script for array freeze, scan dump and register dump is written which helps in extraction data in different error scenarios. These scripts are written and validated with encryption enabled and then decryption of encrypted values.There are two modes: unlocked mode for Intel Engineers and locked mode for customers. These scripts are checked for unlocked mode where data is dumped without encryption. Then checked for locked mode where data is dumped with encryption. The extracted data is matched with the golden values. These scripts are checked for different error scenarios as well. In short correctness of extracted data using VCU is checked with the validation of proper functioning of encryption and decryption of data.

Now, in the near future the customers can extract internal array and debug scan chains along with the exposed registers without having an Intel engineer involved. However, to protect IP, the data they are able to extract will be encrypted and requires an Intel engineer to decrypt and evaluate it. The capability to extract the array and scanout data on their own represents an important improvement in debug throughput. The tool will extract data into a format that will be compatible with the internal debug toolset. So all internal tools will be able to operate on the captured information.

References

[1] Hemant Rotithor, Intel Corporation, *“Postsilicon Validation Methodology for Microprocessors”*, IEEE, October–December 2000

[2] Nicola Nicolici and Ho Fai Ko, *“Design-for-debug for post-silicon validation: Can high-level descriptions help?”*, IEEE,2009

[3] *“Test, Validation and Debug Architecture”*,International Application Published Under Patent Corporation Treaty, Publication no. WO 2012/087330 A2

[4] INTEL Corporation, *“ITP700 Debug Port Design Guide”*, February 2004

[5] Adrian Carbine and Derek Feltham,*“Pentium Pro Processor Design for Test and Debug”*, IEEE,1998

[6] Intel Corporation ,*“Intel Processor-Based Server Selection Guide”*, 2014

[7] `https://embedded.communities.intel.com`

[8] INTEL Corporation,*“Intel Xeon Processor E5-1600/ E5-2600/E5-4600 Product Families,Datasheet-Volume One”*, May 2012