ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

IMPLEMENTATION OF NIST TESTS ON A CHAOTIC RANDOM NUMBER GENERATOR INSIDE A SYSTEM ON CHIP

SENIOR DESIGN PROJECT

İlayda YAMAN
Merve FİRİK

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

MAY 2018

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL-ELECTRONICS FACULTY

IMPLEMENTATION OF NIST TESTS ON A CHAOTIC RANDOM NUMBER GENERATOR INSIDE A SYSTEM ON CHIP

SENIOR DESIGN PROJECT

İlayda YAMAN 040140057
Merve FİRİK 040130109

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

Project Advisor: Prof. Dr. Müştak Erhan YALÇIN

MAY 2018

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

NIST TESTLERİNİN KIRMIK ÜSTÜ GERÇEKLENMESİ VE RASTGELE SAYI ÜRETECİ ÜZERİNDE TESTLERİ

LİSANS BİTİRME TASARIM PROJESİ

İlayda YAMAN 040140057
Merve FİRİK 040130109

Proje Danışmanı: Prof. Dr. Müştak Erhan YALÇIN

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

MAYIS, 2018

We are submitting the Senior Design Project Interim Report entitled as “Implementation of NIST Tests on a Chaotic Random Number Generator inside a System on Chip”. The Senior Design Project Interim Report has been prepared as to fulfill the relevant regulations of the Electronics and Communication Engineering Department of Istanbul Technical University. We hereby confirm that we have realized all stages of the Senior Design Project Interim Report by ourselves, and we have abided by the ethical rules with respect to academic and professional integrity.

İlayda YAMAN .....
040140057

Merve FİRİK .....
040130109

Project Advisor: **Prof. Dr. Müştak Erhan YALÇIN** .....

FOREWORD

In the memory of my grandad, Halil İlbasmış, who passed away this year...

Starting with the greatest teacher Mustafa K. Atatürk and my supervisor Prof. Dr. Müştak E. Yalçın, I would like to thank everyone who inspired me, taught me and challenged me so that I can develop both personally and professionally.

Research assistant Latif Akçay, with his extensive knowledge and kindness, and my team mate Merve Firik, with her endless patience, has been a great support for me; their contribute on this thesis is tremendous and irreplaceable. I would also like to give my gratitude to two other people for never turning me down when I needed help, Ass. Prof. Dr. Metin Hüner and my former employee İnan Erdem.

Moreover, I cannot be more thankful to all of my friends, especially Caner Özer, for their mental and professional support which includes spending several hours to solve technical problems, inspiring me on new ideas or never letting me give up even in the most hopeless times.

On the other hand, it is really hard for me to explain my gratefulness to my family for being there whenever I needed them and for their endless belief in me. I have endless appreciation and gratitude for them. Istanbul Technical University has become a second family to me and I am willing to show my gratitude toward my university after I graduate.

I would like to end my words with one of my favorite quotes which was given by another professor in one of my childhood books:

Happiness can be found, even in the darkest of times, if one only remembers to turn on the light.

May 2018 İlayda YAMAN

FOREWORD

I would like to thank my supervisor Prof. Dr. Müştak Erhan Yalçın for giving me the opportunity of working with him and his excellent guidance during my project. I would like to emphasize my gratitude to research assistants Latif Akçay, for his great assistance.

Moreover, I would like to thank my team İlayda Yaman for her kindness, understanding and endless patience specially when I ask thousands of questions about the project has been a great support for me.

Furthermore, I would like to thank all my friends for their support which kept me going through hard times and especially to my dear friend Melike Bekar.

In truth, I could not have achieved my current level of success without my parents.

I would like to express my endless appreciation and gratitude to my parents, who had always supported me and believed in me.

May 2018 Merve FİRİK

TABLE OF CONTENTS

Pages

FOREWORD ..... **v**
TABLE OF CONTENTS ..... **vii**
ABBREVIATIONS ..... **viii**
LIST OF TABLES ..... **ix**
LIST OF FIGURES ..... **x**
SUMMARY ..... **xii**
ÖZET ..... **xiv**
1. INTRODUCTION **1**
1.1 Purpose of Project **2**
1.2 Literature Review **2**
2. BACKGROUND INFORMATIONS ABOUT THE PROJECT **4**
3. HARDWARE DESIGN: GENERATE RANDOM NUMBERS **8**
3.1 Design of the Modules **8**
3.1.1 Control Unit **8**
3.1.2 Memory **8**
3.1.3 Random Number Generator **10**
3.2 Create and Package Intellectual Property **15**
3.3 Create Block Design **20**
3.4 Timing and Area Constraints **23**
4. IMPLEMENTATION OF NIST TESTS **30**
4.1 NIST Tests **30**
4.2 Embedded Linux Using C Language **35**
4.2.1 Why Linux? **35**
4.2.2 SD Card **36**
4.2.3 Petalinux and Create Simple Application **38**
5. FINAL REMARKS **38**
5.1 Possible Applications of this Project and Future Work Recommendations **48**
5.2 Realistic Constraints **48**
5.2.1 Cost analysis **48**
5.2.2 Standards **48**
5.2.3 Social, environmental and economic impact **49**
6.CONCLUSION **50**
REFERENCES **51**
CURRICULUM VITAE **53**

ABBREVIATIONS

ASIC	**:** Application Specific Integrated Circuit
CMOS	**:** Complementary Metal Oxide Semiconductor
DFF	**:** D Flip-Flop
DLUT	**:** Delay Block of Look-Up Tables
FIPS	**:** Federal Information Processing Standarts
FPGA	**:** Field Programmable Gate Array
HDL	**:** Hardware Description Language
INTCOMP	**:** Integrate and Compare
IP	**:** Intellectual Property
I/O	**:** Input/Output
LUT	**:** Look-Up Table
PLB	**:** Processor Local Bus
PUF	**:** Physically Unclonable Function
RAM	**:** Random Access Memory
RFID	**:** Radio Frequency Identification
RNG	**:** Random Number Generator
RSÜ	**:** Rastgele Sayı Üretici
SDK	**:** Software Development Kit
SOC	**:** System on Chip
XPS	**:** Xilinx Platform Studio
UART	**:** Universal Asynchronous Receiver/Transmitter
XDC	**:** Xilinx Design Constraints
VLSI	**:** Very Large Scale Integration
NIST	**:** National Institute of Standards and Technology
IOT	**:** Internet of Things
ARM	**:** Advanced RISC Machine
RISC	**:** Reduced Instruction Set Computer
BSP	**:** Board Support Package

LIST OF TABLES

Table 4.2 : Statistical Test Results

LIST OF FIGURE

Figure 1.1: Design Hierarchy for SoC

Figure 1.2: ZedBoard Zynq®-7000 ARM/FPGA SoC Development Board

Figure 3.1: Chaotic attractor in x(t) – x(t - T) plane for T = 9

Figure 3.2: Schematic Diagram of Look-Up Table based Delay Line (DLUT).

Figure 3.3: Schematic diagram of integrate and compare block(INTCOMP)

Figure 3.4 : The RNG module block design

Figure 3.5: Vivado Window to choose ZedBoard

Figure 3.6: Create and Package New IP Window on Vivado

Figure 3.7: AXI4 Interfaces

Figure 3.8: Add or Create Design Sources Window

Figure 3.9: Assigning Output Data of RNG to Slave Registers

Figure 3.10: Module instantiation of our RNG to AXI peripheral

Figure 3.11: Hierarchy of our module and AXI Peripheral

Figure 3.12: Simple Dual Port[14]

Figure 3.13: Block Memory Generator

Figure 3.14: Summary of Creating IP

Figure 3.15: Block Design

Figure 3.16: ZYNQ Block Design

Figure 3.17: Block Design

Figure 3.18: The Hierarchy

Figure 3.19: Layout without area constraints

Figure 3.20: Specify constraints file

Figure 3.21: Layout showing the area constraints are met

Figure 3.22: Including the bitstream When Exporting Hardware

Figure 3.23: View of the SDK

Figure 3.24: Our Application Project with SD Card

Figure 3.25: Adding “xilffs” library on the BSP

Figure 3.26: BSP Settings

Figure 3.27: View of the Exported Hardware Platform from Vivado including IP driver
Figure 3.28: Adding a Linker to Math Library
Figure 3.29: Application Project of NIST Tests in SDK
Figure 4.1 : User prescribed input file
Figure 4.2 : Statistical Tests
Figure 4.3 : Number of Bitstream
Figure 4.4 : NIST Test Parameter Adjustments
Figure 4.5 : Contensts of SD card
Figure 4.6 : General view of SD Card
Figure 4.7 : Zed-boot
Figure 4.8 : General Design Flow for Hardware and Software Design with Vivado and Petalinux
Figure 4.9 : Some PetaLinux Libraries
Figure 4.10 : Minimum Hardware on Vivado to run Linux
Figure 4.11 : Building the project and opening putty
Figure 4.12 : Putty Configuration
Figure 4.13: Booting Linux Window
Figure 4.14 : General view on Linux
Figure 4.15 : PetaLinux makefile
Figure 5.1: Power Consumption

IMPLEMENTATION OF NIST TESTS ON A CHAOTIC RANDOM NUMBER GENERATOR INSIDE A SYSTEM ON CHIP

SUMMARY

In today's world, as Internet of Things become a phenomenon, security of the communication between smart devices has become a vital issue. Security of these devices highly dependent upon encryption of the data exchanged between devices which is done by random numbers. We aim to build a system that can create true random numbers, record them and test their true randomness without external influence.

This project consists of both hardware and software design of a System on Chip that can be used for many purposes. Digilent ZedBoard Zynq®-7000 ARM/FPGA SoC Development Board is selected to develop the design and Vivado Design Suite, Xilinx Software Development Kit (SDK) and PetaLinux tools have been used. As hardware design, a chaotic random number generator is implemented on the FPGA board by Vivado. The randomness of these values are effected by process and environmental variations which is unique for every board and different environments. Timing and area constraints are met and power consumption analysis is given in the report. Also all the steps have been explained in detail.

In order to record the output values of this random number generator, we used Software Development Kit to reach to the prebuilt Secure Digital Card (SD-Card) in the board. These output values needed to be tested by National Institute of Standards and Technology (NIST) Tests, which is a statistical test suite for the validation of random number generators which consists our second part, software development. Because of the tests' configuration and limitations on the SoC, it become essential to build a Linux image on the SD-Card by PetaLinux Tool and cross-compile the NIST tests. First we tried to use only Xilinx SDK or try other ways to perform the tests but none of them was successful because of the complex and huge amount of file operations. We could have replaced all file system functions with special library functions that ARM can operate but that would disturb the universally accepted structure of the NIST tests codes.

Building a Linux image for ZedBoard that can communicate with the hardware we created, requires many pre-requisites such as a partitioned SD-Card in order to both boot Linux as an Operating System and read text data at the same time.

Another example is a driver to reach memory address to connect with the Intellectual Property we created, which is the random number generator. The driver is not complete since we do not have the computer science background which is necessary to build drivers. However, our application that applies NIST tests to an existing text file inside the board is complete, which means we can make the NIST tests in real time which was the main accomplishment.

Moreover, the NIST tests can be applied to any random data sequence (that satisfies the requirements indicated by NIST to apply the test) in a text file inside the SD-Card. If the results of these tests are successful, this means the random number generator creates true random numbers and these numbers can be used in applications where security is crucial. Since everything will run on the board, the tests of randomness will be able to done in different environments with high speed and low cost.

In the end, we created a system that can generate random numbers by process and environmental variations, store them in a Secure Digital Card and test them with universally accepted randomness tests of NIST. As a result, considering there is no source indicating that this is done before, our project can be very valuable especially in Internet of Things industry.

GERÇEK RASTGELE SAYI ÜRETECİNİN FPGA ÜZERİNDE GERÇEKLENMESİ VE EŞ ZAMANLI OLARAK NIST TESTLERİNİN KIRMIKÜSTÜ SİSTEMDE UYGULANMASI

ÖZET

Bilgisayar bilimlerinde çeşitli amaçlarla rastgele sayılara ihtiyaç duyulur. Örneğin şifreleme algoritmalarında önemli bir role sahip olan rastgele sayılar şifreleme işleminin gizliliği ve güvenilirlik açısından önemlidir.Bu projede, var olan bir
rastgele sayı üretici sistemi FPGA üzerinde ARM işlemci ile birlikte gerçeklenmiştir ve gerçeklenen bu sistem üzerinde NIST testleri yapılıp sonuçlar incelenmiştir.
Verilog'da kodlanmış rastgele sayı üretici sistemi işlemciye çevresel olarak eklenmiştir. ZedBoard Zynq 7000 FPGA kullanılmıştır. FPGA üzerinde yapılan bütün tasarımlar Vivado Design Studio programı yardımı ile Verilog donanım tasarlama dili kullanılarak kodlanmıştır. Testlerin kodlanması Xilinx Software Development Kit programında C dili ile yapılmıştır Rastgele sayı üretici sisteminin işlemci ile birlikte gerçeklenmesi sistem üzerinde NIST testlerinin oldukça hızlı bir şekilde yapılmasına olanak sağlar ve bu testlerin sonuçlarına göre sistem daha iyi sonuçlar elde etmek için optimize edilebilir. NIST testleri bu sistemlerin sağlaması gereken uluslararası standartları içermektedir. Bu standartlardan herhangi birini sağlamayan sistem rastgele olarak kabul edilmez.
Tezin ilk bölümünde proje ile ilgili gerekli alt yapı bilgileri verilmiştir. Kullanılan donanım tasarımı programı detaylıca açıklanmıştır.Ardından ikinci bölümde sistemin donanım tasarımı gerekli aşamalar ele alınarak açıklanmıştır.Donanım tasarımında gerekli modüllerin açıklanıp,sisteme nasıl eklendiği hakkında bilgi verilmiştir.Sistemdeki modüller,bu modüllerin bağlantılarının nasıl yapıldığı kullanılan dizayn programı ile birlikte incelenmiştir.Bir IP nin nasıl oluşturulduğu ve pakelndiği adımlar halinde verilmiştr.Oluşturulan bu IP,sisteme eklenip ilgili düzenlemeler yapılmıştır.Bağlantı blok otomasyonu ile IP içindeki modüller,işlemci olan ARM a bağlanmıştır. Donanım tasarımındaki kısıtlamalar baz alınmıştır. Alan kısıtlamasına uygun olarak IP, ARM işlemcide sınrlanmıştır.

Takip eden bölümde NIST tesleri hakkında ön bilgi verilerek,basit bir NIST kodunun nasıl derlendiği anlatılmıştır. NIST testlerinin derlenmesinde SDKda bazı sorunlarla karşılaşıldı.Gerekli araştırmalar yapıldığında karşılaşılan sorunun sisteme Linux işletim sistemi yüklenerek çözülebileceğine karar verildi.Linux işletim sistemi yüklendikten sonr PetaLinux yardımı ile NIST testleri derlendi ve sisteme uygulandı.Son olarak proje sonlandırılmıştır.

1.INTRODUCTION

In today's world, electronic devices which need to communicate with each other are everywhere and this sector is expanding very quickly. This leads to an increase in need for building secure systems, which can send and receive data without the interference of third unknown parties, without increasing the cost. Safe communication becomes more important with the developments in technology industry. Random number generators are used as sources of keywords in encryption and decryption of information; hence, they are an important part of cryptology. In our project, we are focused on improving the security of the new technological devices. A new random number generator is designed which can met the need of secure devices that can be used in daily life without a high additional cost. The randomness of such generators is an important issue and must be tested according to some standards such as National Institute of Standards and Technology (NIST) tests. If random number generator (RNG) cannot satisfy these international standards, it is not considered to be random.

In this project, the system on chip implementation of an existing random number generator on ZedBoard Zynq™-7000 Development Board which includes an ARM processor is realized. The SOC implementation of the random number generator makes it possible to evaluate and test the generated numbers in a rather quick way. The system is tested by NIST (National Institute of Standards and Technology) tests. NIST tests are implemented in C language and they require huge amount of file operations. Unfortunately, also the file system that ARM provides is also limited to custom libraries which does not allow the NIST tests to be done without changing its original source code. As a result, after a long research we developed a solution of building a Linux image in a SD-card where the tests are precompiled compatible with ARM compiler. We used PetaLinux Tool to create the Boot image and other necessary files.

1.1 Purpose of Project

The main purpose of the project is to apply NIST Tests to ZedBoard Zynq™-7000 Development Board. The board includes an ARM processor which is a key to implement NIST tests to the system. The random number generator is based on a chaotic time delay which is affected by process and environmental variations. The system on chip must be good enough to pass most of the NIST tests. On the other hand, a chaotic time delay random number generator is used to generate true random numbers for making the system more secure. This will suit the needs of business people who are willing to use as many IoT devices as they can but need the protection of their data extensively. The system can be used in many other projects which involve ARM processors to provide a secure system as a whole and it will be safe, won't be costly and complicated.

1.2 Literature Review

In pervious works including RNGs the system was designed on soft processors such as MicroBlaze and OpenRisc. However, with these processors doing the NIST tests required a huge amount of effort and time as MicroBlaze and other processors didn't have file system support. Our project is unique on its own way because in the previous works similar to this subject, other processors didn't have file system support while ARM processors' compiler allows us to do NIST tests without extreme workload. The main advantages of ARM processors are, the ability to write and read from both the SD-Card and Flash and high efficiency in tasks require high level of RAM. With ZedBoard, we are able to implement the RNG or other cryptographic modules into online NIST tests.

Even though there are many resources on Xilinx Zynq-7000 All Programmable SoC and NIST tests, there are no implementations where the test suit is implemented on NIST algorithms. Our project is a first in its category. The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nation's measurement and standards infrastructure. ITL develops tests, test

methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems.

This Special Publication 800-series reports on ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

For cryptographic purposes, the output of RNGs needs to be unpredictable. However, some physical sources (e.g., date/time vectors) are quite predictable. These problems may be mitigated by combining outputs from different types of sources to use as the inputs for an RNG. However, the resulting outputs from the RNG may still be deficient when evaluated by statistical tests. In addition, the production of high-quality random numbers may be too time consuming, making such production undesirable when a large quantity of random numbers is needed.

2. BACKGROUND INFORMATIONS ABOUT THE PROJECT

In this project, a system on chip implementation of an existing RNG on FPGA is realized. The RNG is embedded with the processor ARM. The system o chip implementation of the RNG makes it possible to evaluate and test the generated numbers in a rather quick way. The NIST tests are then implemented in C language and run on the RNG implementation and the results of the tests are evaluated. With technological advancement in very large scale integration (VLSI) circuits it is possible to integrate all component into a single chip which is called as system on chip. It may contain analog, digital, mixed signal and other radio frequency functions all lying on a single chip substrate. Technological advances mean that complete systems can be implemented on a single chip brings many benefits as lower cost per gate, faster circuit operation, more reliable implementation, smaller physical size, greater design security, significant in terms of speed, lower power consumption.

However, it has higher fabrication cost, increased complexity, time to market demand and more verification. Today, SoCs are very common in electronics industry due to its low power consumption. Also, embedded system applications make great use of SoCs. Most common use of SoCs has been found in the mobile devices industry. The use of SoCs have enabled manufacturers of such devices to come up with devices good of very small form factor that offer ample performance. It also enables them to focus on features they project to the target customers than relying on capabilities of chips provided by some other company. SoCs also brought about a revolution in embedded systems by paving way for very small and portable single- board computers.SoCs consists of control unit, memory blocks, timing units, analog interfaces, external interfaces, voltage regulators and power management units. In general, the design flow of SoCs consists of hardware and software modules.

Hardware blocks of SoCs are developed from pre-qualified hardware elements and software modules integrated using software development environment. The hardware description languages like Verilog, VHDL and SystemC are being used for the development of the modules. In this project Verilog language is used. Verilog is hardware description language, which is used to describe a digital circuit in a language which the tool understands. Applied to electronic design, Verilog is intended to be used for verification through simulation, for timing analysis, for test analysis and for logic.

The SoCs are verified for the logic correctness before it is being given to the foundry. For the verification and debug of hardware and software of SoC designs, engineers have employed FPGA, simulation acceleration, emulation and other technologies. After the debugging of the SoC, the next step is to place and route the entire design to the integrated circuit before it is being given to the fabrication. In the fabrication process, full custom, standard cell and FPGA technologies are commonly used. The generic design hierarchy used is summarized in Figure. 1.1

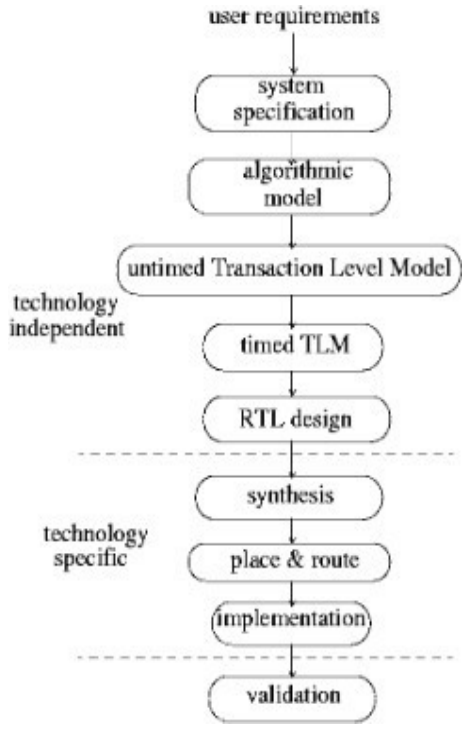

Figure 1.1: Design Hierarchy for SoC. [13]

In our project, the user requirements are translated into a formal system specification written in C language.

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. ASIC and FPGAs have different value propositions, and they must be carefully evaluated before choosing any one over the other. Information abounds that compares the two technologies. While FPGAs used to be selected for lower speed,complexity, volume designs in the past, today's FPGAs easily push the high performance. With unprecedented logic density increases and a host of other features, FPGAs are a compelling proposition for almost any type of design. The great advantage of the FPGA is that the chip is completely programmable and can be re-programmed. In this way it becomes a large logic circuit that can be configured according to a design, but if changes are required it can be re-programmed with an update. Thus if circuit card or board is manufactured and contains an FPGA as part of the circuit, this is programmed during the manufacturing process, but can later be re-programmed to reflect any changes. Thus it is field programmable, giving rise to its name. Although FPGAs offer many advantages, there are naturally some disadvantages. They are slower than equivalent ASICs (Application Specific Integrated Circuit) or other equivalent ICs, and additionally they are more expensive. FPGAs are programmed using Hardware Description languages (HDL). This description can be in the form of structural, behavioral or gate level. In this project, ZedBoard Zynq™-7000 Development Board shown in the Figure 1.2 is used.

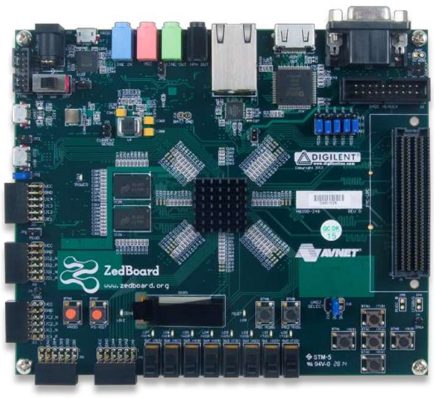

Figure 1.2: ZedBoard Zynq®-7000 ARM/FPGA SoC Development Board [2]

In Zedboard there is embedded ARM processor. An ARM processor is one of a family of CPU's based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines(ARM).

3.HARDWARE DESIGN:GENERATE RANDOM NUMBERS

3.1 Design of the Modules

3.1.1 Control Unit (CU)

A CU takes its input from the instruction and status registers. Its rules of operation, or micro program, are encoded in a programmable logic array (PLA), random logic or read-only memory (ROM). CU functions are as follows controls sequential instruction execution, interprets instructions, guides data flow through different computer areas, regulates and controls processor timing sends and receives control signals from other computer devices, handles multiple tasks, such as fetching, decoding, execution handling and storing results. CUs are designed in two ways.

Firstly, hardware control design that is based on a fixed architecture. The CU is made up of flip-flops, logic gates, digital circuits and encoder and decoder circuits that are wired in a specific and fixed way. When instruction set changes are required, wiring and circuit changes must be made. This is preferred in a reduced instruction set computing (RISC) architecture, which only has a small number of instructions. Secondly, micro program control that is stored in a special control memory and are based on flowcharts. They are replaceable and ideal because oftheir simplicity .In this project, CU module is used for generating a user interface and adjusting system parameters.

3.1.2 Memory (MEM)

Memory blocks are required for implementations of RNG in order to record data. Because of this necessity MEM block is generated by using Block Memory Generator. The Block Memory Generator core uses embedded Block Memory primitives in Xilinx FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations.

The Block Memory Generator core uses embedded block RAM to generate five types of memories with their applications are shown below.

1. Single-port RAM : Processor scratch RAM, look-up tables
2. Simple Dual-port RAM : Content addressable memories, FIFOs
3. True Dual-port RAM : Multi-processor storage
4. Single-port ROM : Program code storage, initialization ROM
5. Dual-port ROM : Single ROM shared between two processors/systems

For dual-port memories, each port operates independently. Operating mode, clock frequency, optional output registers, and optional pins are selectable per port. The Block Memory Generator core is used to build custom memory modules from block RAM primitives in Xilinx FPGAs. The core implements an optimal memory by arranging block RAM primitives based on user selections, automating the process of primitive instantiation and concatenation. A Simple Dual-port RAM with symmetric ports can use the special Simple Dual-port RAM primitive in 7 series devices, which can save as much as fifty percent of the block RAM resources for memories 512 words deep or fewer.[5]
The Single-port ROM allows Read access to the memory space through a single port. The Simple Dual-port RAM provides two ports, A and B, as illustrated in Figure 3-4. Write access to the memory is allowed through port A, and Read access is allowed through port B.

In this project, memory module includes necessary control signals and the memory block generated by block memory generator. The data width is 32 bits and the address range is set to 10 bits.

3.1.3 Random Number Generator

3.1.3.1 Categories of Random Number Generator

Random number generators can divide by two categories as:

1.Pseudo Random Number Generators use computational algorithms that can produce long sequences of apparently random results. Sequence can be reproduced if the seed value is known.

2.True random number generators measures some physical phenomenon that is expected to be random. One of the most common examples is jitter noise of digital clock signals in embedded TRGNs where chaotic circuits offer a promising alternative way.

In this project, we are dealing with true number generator for our system.

An RNG uses a non-deterministic source, along with some processing function to produce randomness. The use of a distillation process is needed to overcome any weakness in the entropy source that results in the production of non-random numbers. The entropy source typically consists of some physical quantity, such as the noise in an electrical circuit, the timing of user processes, or the quantum effects in a semiconductor. Various combinations of these inputs may be used.For cryptographic purposes, the output of RNGs needs to be unpredictable. Outputs from different types of sources to use as the inputs for an RNG.

3.1.3.2 Main Parts of True Number Generator

A binary feedback function is offered to reduce the number of physical components and idealized elements. The mathematical model of the system where f(x) is nonlinear binary feedback function is as follows [5]:

$$\dot{x}(t) = -x(t) + \alpha f(x(t_k - \tau)),\ t_k \leq t < t_k + T_s$$

The resultant chaotic behavior of the system is given in the Figure 3.1.

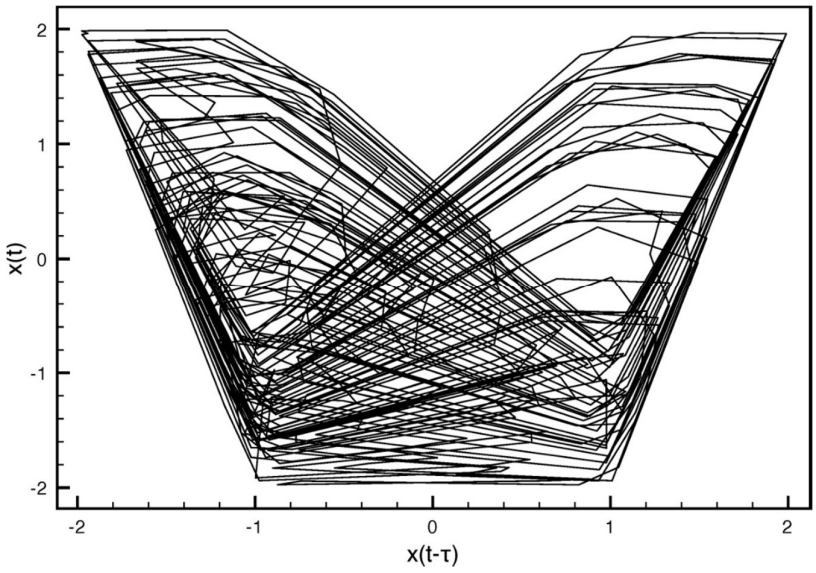

Figure 3.1: Chaotic attractor in x(t) – x(t - T) plane for T = 9

The system is realized by usage of a flip-flop chain with the purpose of delaying the binary output of the nonlinear feedback part of the introduced system results in with a new system that is a sampled-data feedback system. Delay line composed of Look-up-tables (LUT) provides both the delayed signal and a delay uncertainty that breaks the periodic trajectory. It includes 1024 LUTs and one D-type Flip Flop (DFF) at the end of line.

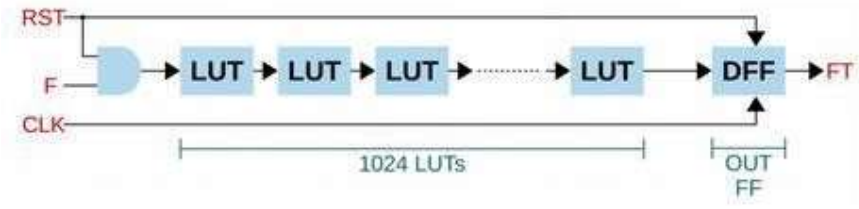

Figure 3.2: Schematic Diagram of Look-Up Table based Delay Line (DLUT).

Source of the propagation delay is low-pass characteristics of digital gates. At the output of each gate, voltage is increased or decreased exponentially. Thus it takes some amount of time to cross the threshold voltage level of the next gate. Combination of these effects at each gate generates the delay line on LUT chain. Process and environmental (PVT) variations effects the delay line.

PVT variations induce randomly varying delay. DLUT line has a significant jitter which is caused by the noise causes the random process of this line. Chaotic basis of INTCOMP has a sensitivity to initial conditions. Basically, INTCOMP means integrate and compare. INTCOMP dramatically changes its trajectory when the delayed signal has a slight change with respect to the expected one.

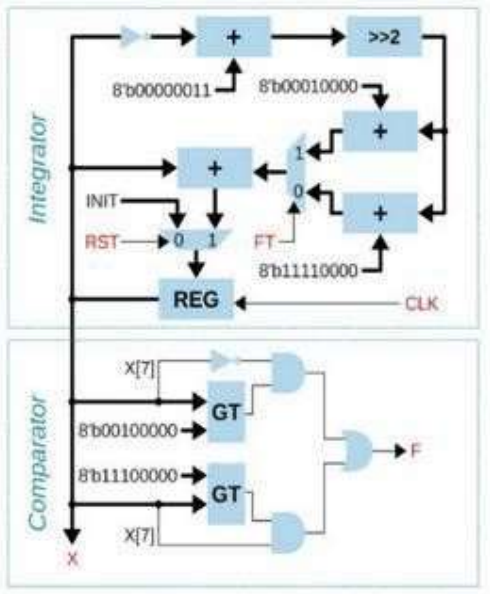

Figure 3.3: Schematic diagram of integrate and compare block(INTCOMP)

In this project, ARM processor is used as a control unit of the entire design. In addition to the RNG, blocks are implemented as hardware in Verilog HDL. All peripherals are connected to the processor with AXI4 interface by using Xilinx Vivado Design Suite. The RNG and cryptographic IPs are embedded with the processor ARM. The SOC implementation of the RNG makes it possible to evaluate and test the generated numbers in a rather quick way.

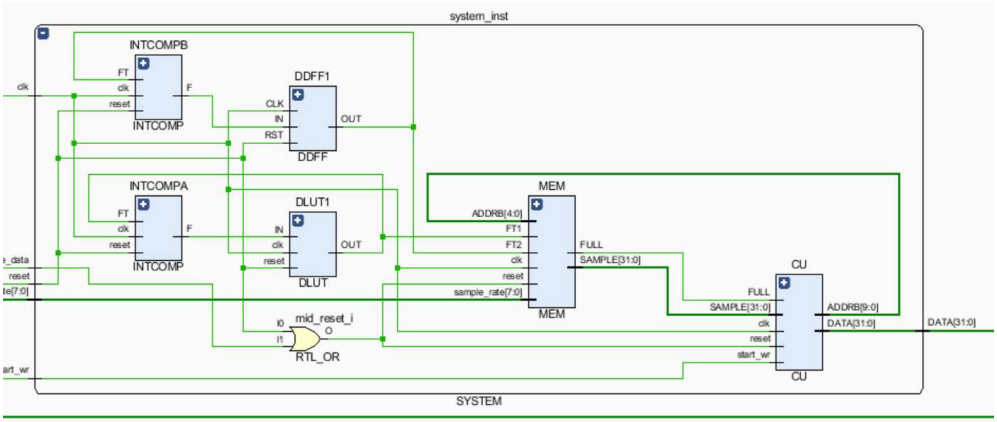

Figure 3.4 : The RNG module block design

3.2 Create and Package IP

Intellectual Property (IP) is any product of the human intellect that is unique, novel, and unobvious and has some value in the marketplace such as an idea, invention, expression, literary creation, business method, industrial process, chemical formula, computer program, algorithm. Xilinx have a rich library of Intellectual Property (IP), which goes through a vigorous test and validation effort to have success the first time.

We create IP to be able to connect our design to ZYNQ Processing System. This feature allows the user to do is easily and take off the high work load from the user. By choosing AXI4 as the protocol, we establish an easy and user friendly hierarchy between custom or prebuilt IPs. Also it is very convenient to use pre-built IPs such as DSP or Block Memory Generator and connect them directly to our design.

Here is the design flow to create and package IP.

Firstly, a new Register Transfer Level (RTL) project which is used for add sources, create block design in IP integrator, generate IP, run analysis, synthesis, implementation is generated on Xilinx Vivado Design Suite. An RTL description is usually converted to a gate level description of the circuit by a logic synthesis tool such as Xilinx Vivado Design Suite. The synthesis results are then used by placement and routing tools to create a physical layout.

In this RTL project Verilog is selected as target language. ZedBoard Zynq 7000 is chosen as board that is shown in Figure 3.5.

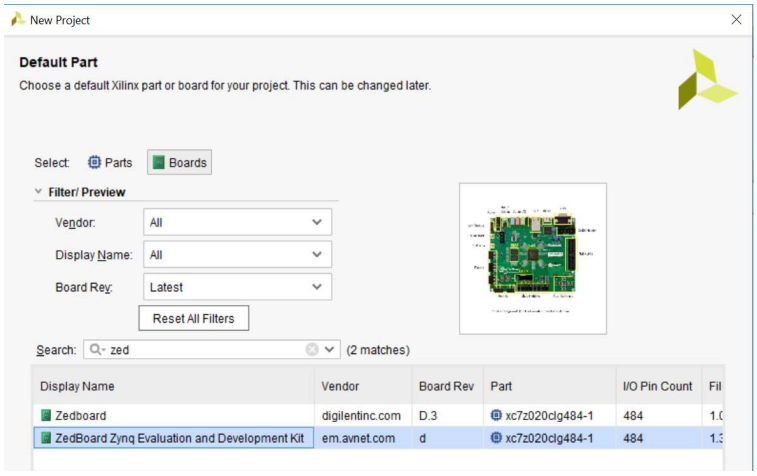

Figure 3.5: Vivado Window to choose ZedBoard

Then, by following Tools>Create and Package new IP, a new IP is added into board in order to provide communication with ARM processor which is inside ZedBoard.

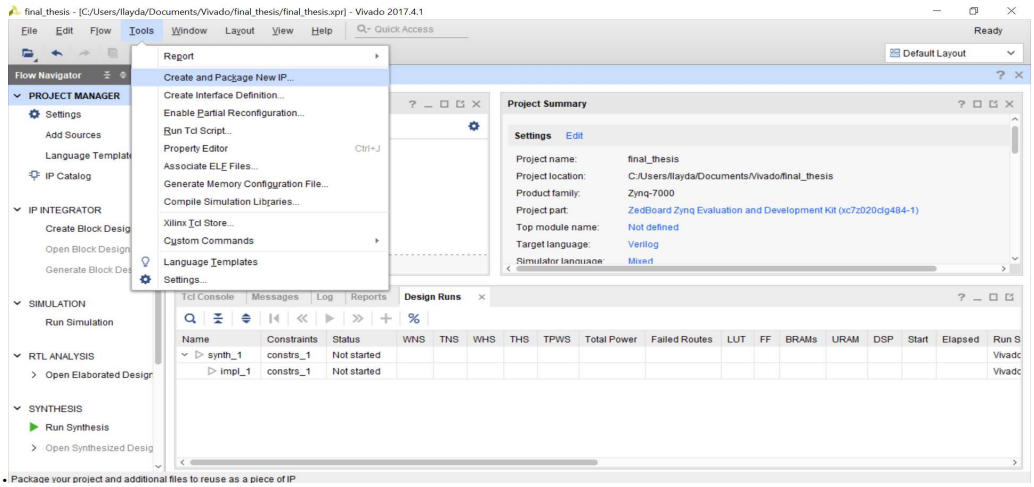

Figure 3.6: Create and Package New IP Window on Vivado

Xilinx has adopted the Advanced Extensible Interface (AXI) protocol for Intellectual Property (IP) cores. The AXI specifications describe an interface between a single AXI master and a single AXI slave, representing IP cores that exchange information with each other. We have created our IP interface which is a AXI4 peripheral in Figure 3.8. We have defined 5 registers to our IP since we have 4 input and one output registers in our system. Clock of the IP doesn't count as a register in this stage and defined separately later.

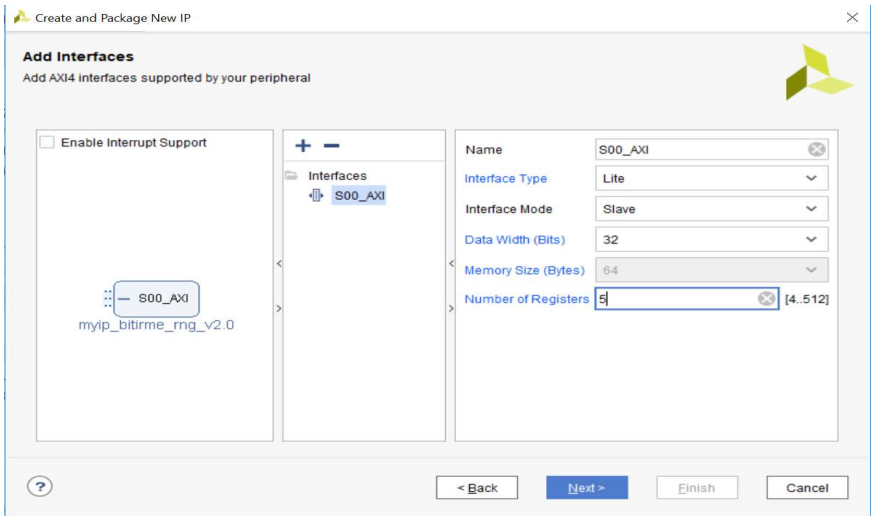

Figure 3.7: Add AXI4 Interface

Then, edit IP is selected to be able edit our IP. Sources are added into peripheral and they are copied into IP directory.

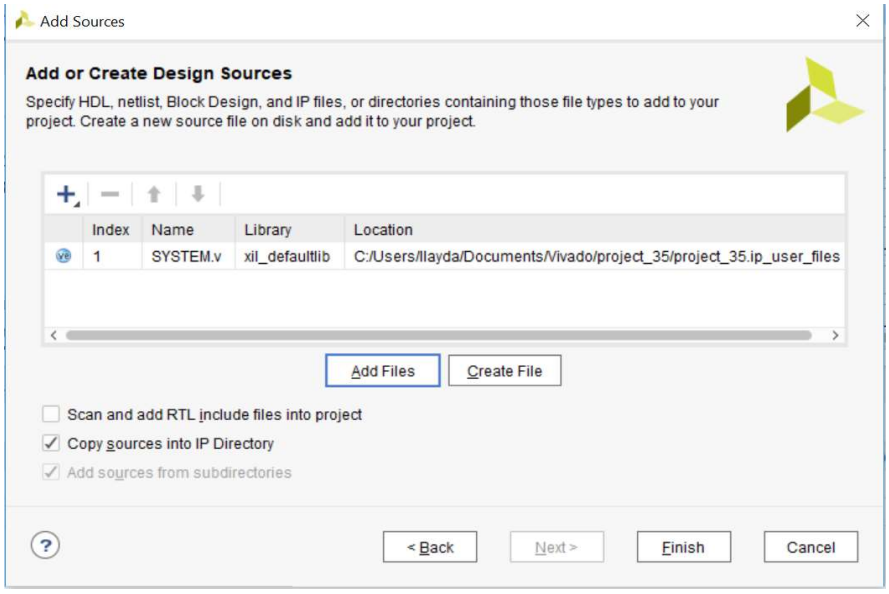

Figure 3.8: Add or Create Design Sources Window

We defined 32 bits 'data_out' signal above I/O assignment as a wire type in the file "myip_bitirme_rng_v2_0_S00_AXI". We also put 32 bits of 'data_out' signal to slv_reg3 and defined it in Verilog code.

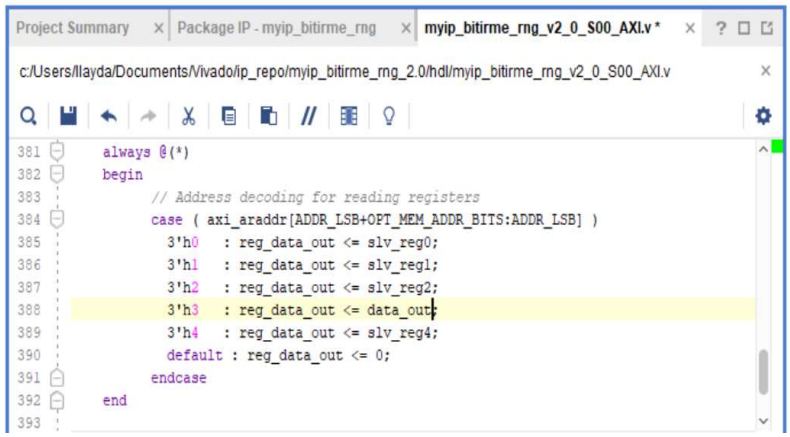

Figure 3.9: Assigning Output Data of RNG to Slave Registers

Then we instantiate our RNG to AXI peripheral

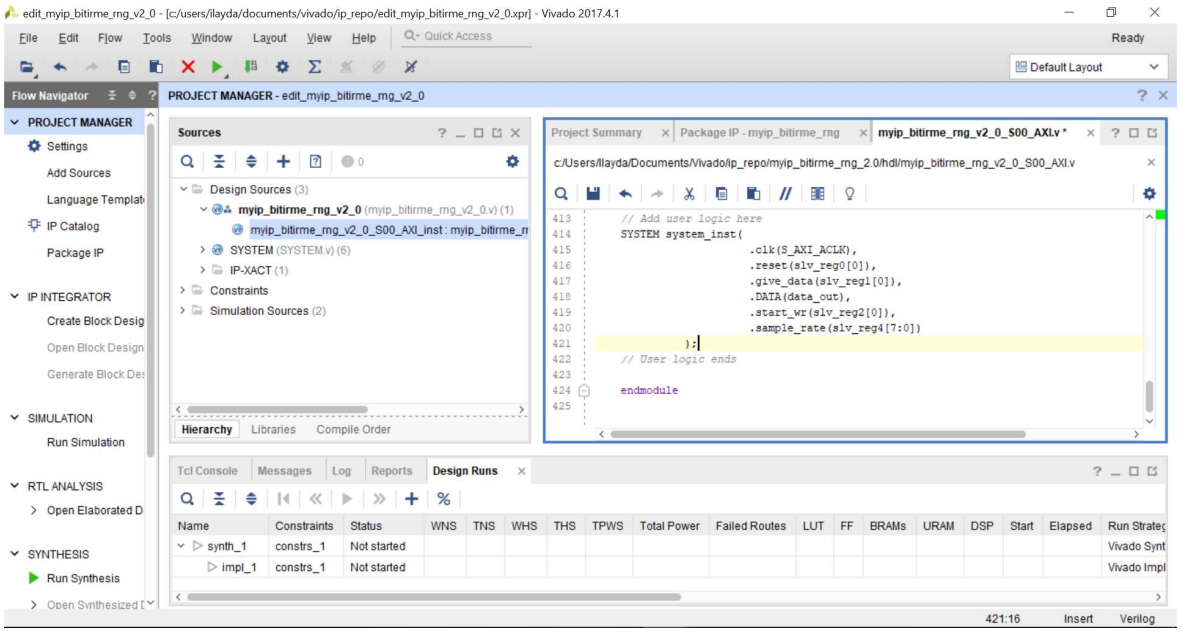

Figure 3.10: Module instantiation of our RNG to AXI peripheral

After we save our file, we will see that our design has been added to hierarchy.

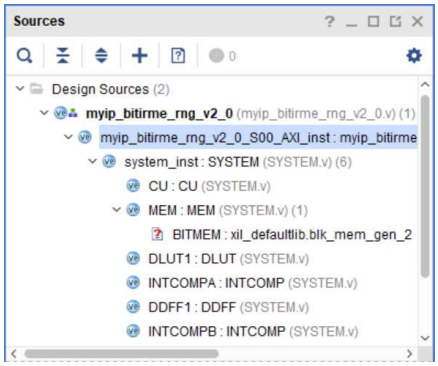

Figure 3.11: Hierarchy of our module and AXI Peripheral

Secondly, we generated a block memory by using Block Memory Generator (8.4) on Vivado Design Suite. The Block Memory Generator core uses embedded Block Memory primitives in FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations. We designed our block memory generator special to our FPGA. We selected Simple Dual Port RAM as a memory type.

The Simple Dual-port RAM provides two ports, A and B, as illustrated in Figure 3.12.

Write access to the memory is allowed through port A, and Read access is allowed through port B.

”

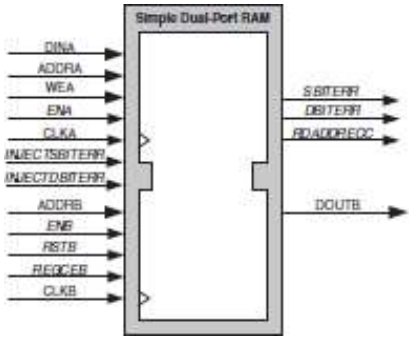

Figure 3.12 : Simple Dual Port[14]

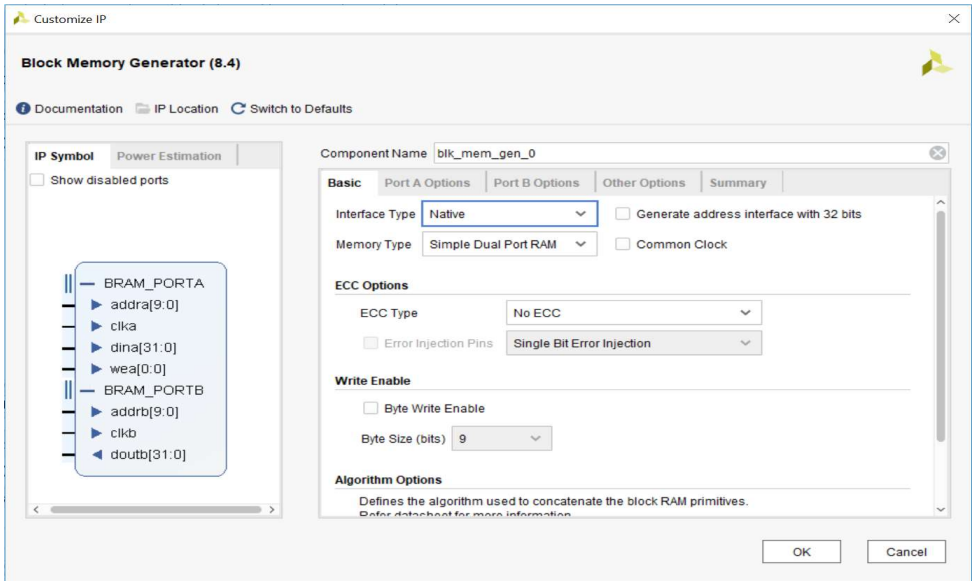

Figure 3.13: Block Memory Generator

When using BRAM controller mode in BMG IP, the width/depth of Port-A/B depends on the address range assigned for the AXI BRAM controller and the value of the data width selected in the AXI BRAM controller.

You need to change the data width and address range of the BRAM controller to change the width and depth of the Block memory generator IP.

For example, if you have assigned a 4k (i.e., 4x1024x8 bits = 32768 bits) address range to the AXI BRAM controller, then if you set the data width as 32, the depth will be auto selected as 32768/32=1024. [17] We arranged the width of port A is 32 and the depth of port B is 1024.

We clicked to “ok” and then to “Generate Output Products” and “Copy All Files into Project”.

Then the following step is Project Manager > Package IP > File Groups tab is clicked to merge changes from File Groups Wizard. After that, IP is created and packeged in a right way.

We closed create IP project and go back to the main project after seeing Figure 3.14.

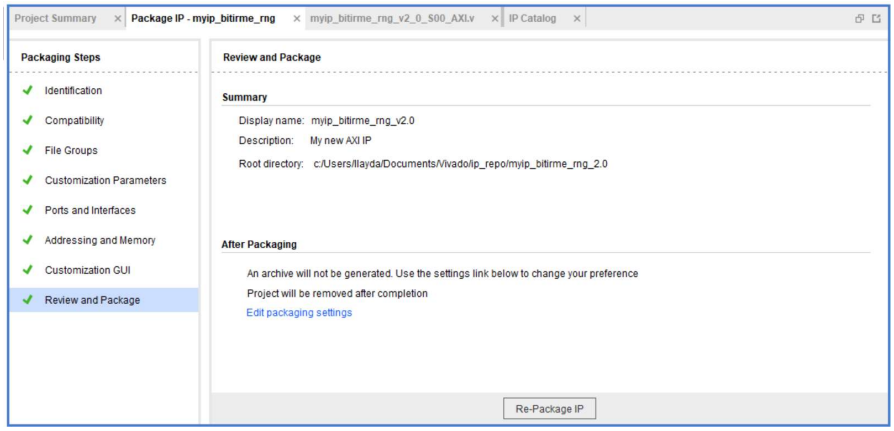

Figure 3.14: Summary of Creating IP

3.3 Creating a Block Design

Under the IP Integrator IP option, “Create Block Design” is clicked and created IP and ZYNQ7 Processing System are added into block design with making interface externals. Then we see the Figure 3.15.

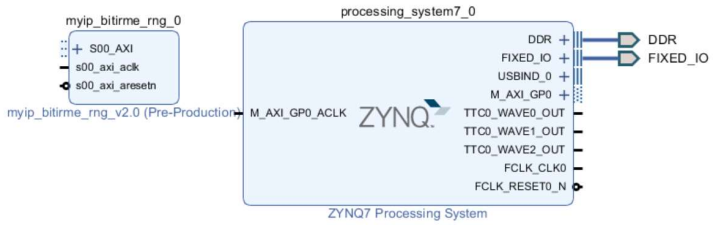

Figure 3.15: Block Design

Also we need to make sure when we double click to our ZYNQ7 Processing System SD-Card Power is turned on in MIO Configuration/IO Peripherals by double clicking the IP.

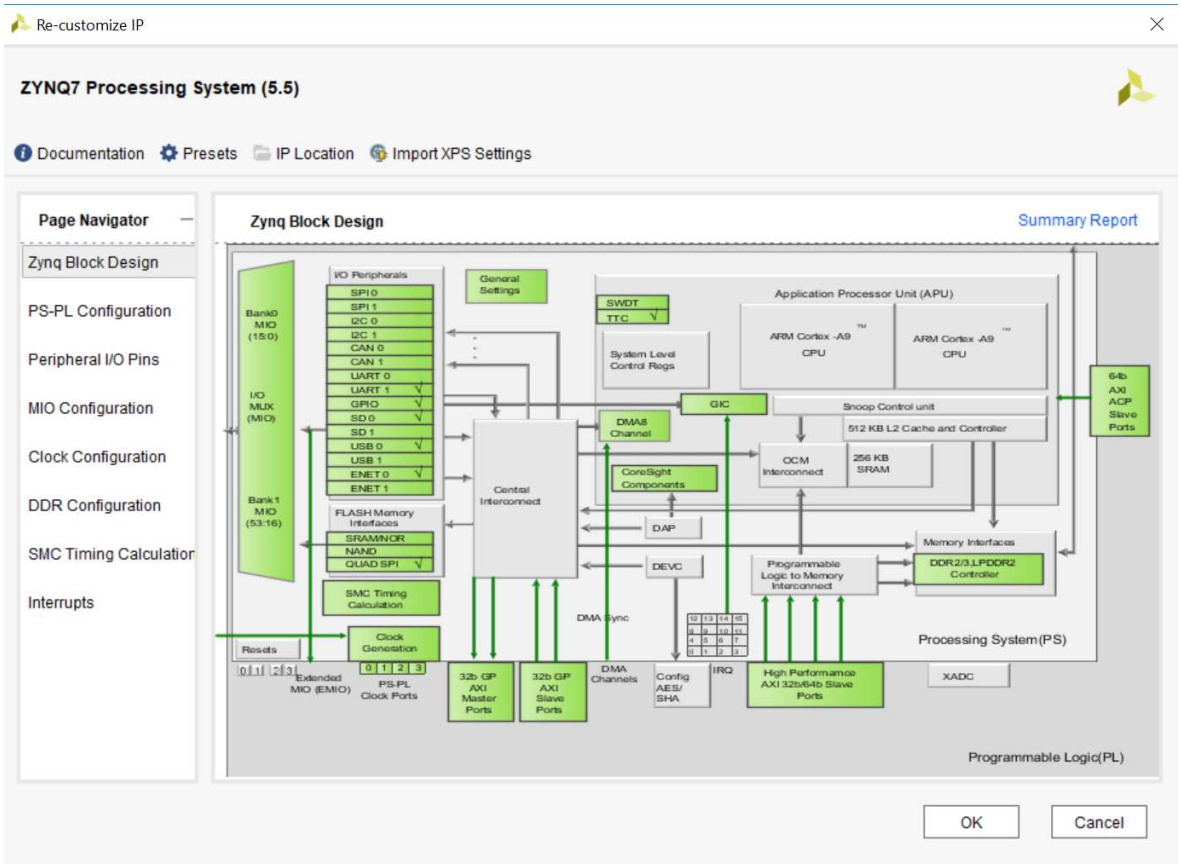

Figure 3.16: ZYNQ Block Design

We used Run Connection Automation to make connections in our design by checking the interfaces to connect automatically. To sum up, ZYNQ Processing System IP has been selected and added to the search box. Later, with "Run Block Automation", interfaces were created for Zynq and the block automation was activated.

Then we met with validating whole design with all connection which is shown in Figure 3.17.

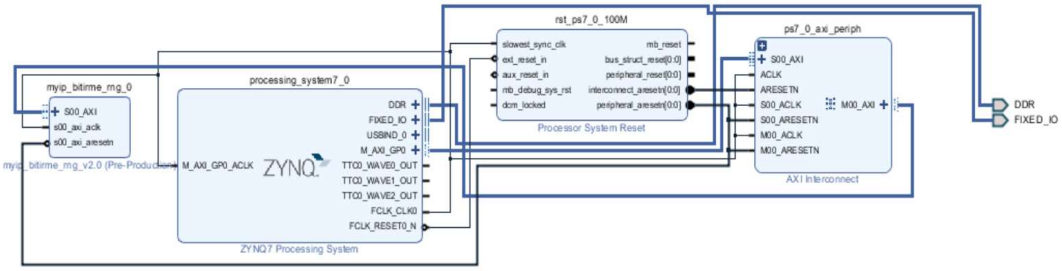

Figure 3.17: Block Design

Then we Generate Output Products by selecting synthesis option as out of context per IP. After that the hierarchy should look like Figure 3.18.

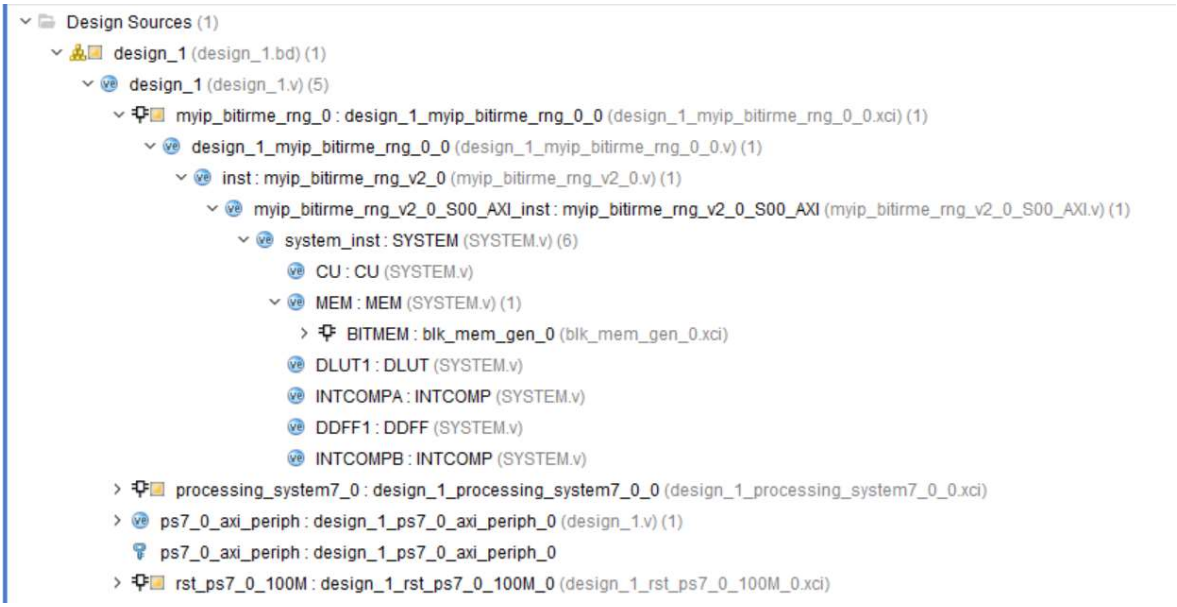

Figure 3.18: The Hierarchy

Clicked to “Create HDL Wrapper” to connect the output/input port of our design to the physical pin described in the constraint file since we created a RTL project with our design hardware that needs to be connected to our target board. For example, in our case, since we have a ZYNQ processor in our design, it needs to be connected to the DDR, clock, IO_MIO pins and etc.

3.4 Timing and Area Constraints

Design constraints define the requirements that must be met by the compilation flow in order for the design to be functional on the board. Not all constraints are used by all steps in the compilation flow. For example, physical constraints are used only during the implementation steps that is, by the placer and the router. Because the Vivado™ Integrated Design Environment (IDE) synthesis and implementation algorithms are timing-driven, you must create proper timing constraints. Over- constraining or under-constraining design makes timing closure difficult. Thus we used reasonable constraints that correspond to our project requirements.

We can easily edit our IP by right clicking on the IP in the Block Design and clicking “Edit IP in the IP-Packager”. Afterwards “Upgrade Selected” IP. Then we run implementation to see reports and the layout.

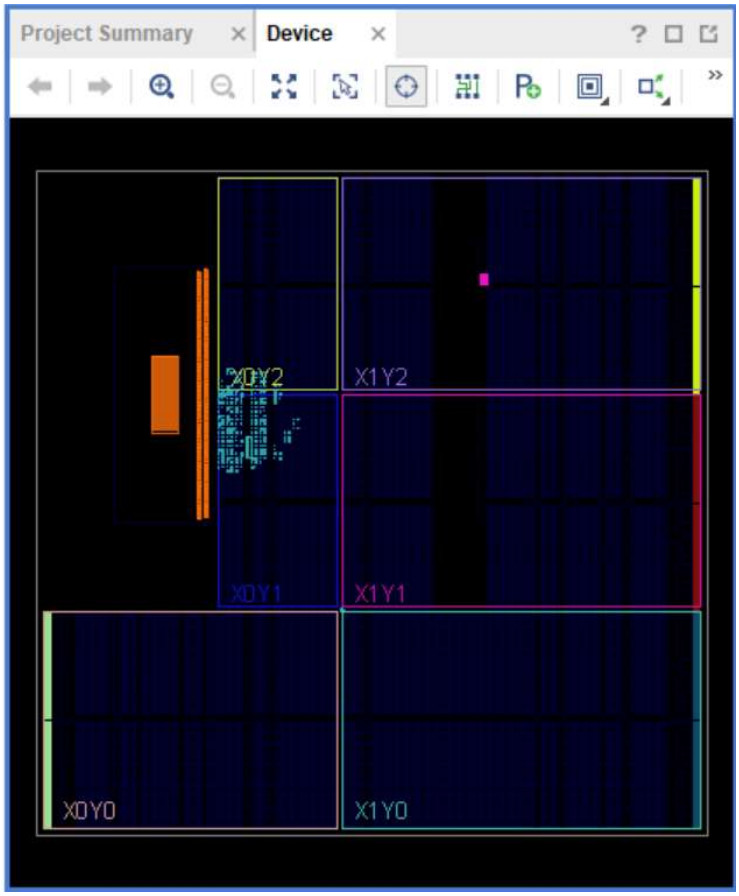

Figure 3.19: Layout without area constraints

To add constraints in our system we used “Constraints Wizard”.

In our case we need to add a constraint file to specify where our RNG is located. We changed our constraints file as following in Figure 3.20.

```
create_pblock pblock_myip_bitirme_rng_0
add_cells_to_pblock [get_pblocks pblock_myip_bitirme_rng_0] [get_cells -quiet [list design_1_i/myip_bitirme_rng_0]]
resize_pblock [get_pblocks pblock_myip_bitirme_rng_0] -add {SLICE_X26Y113:SLICE_X35Y123}
resize_pblock [get_pblocks pblock_myip_bitirme_rng_0] -add {RAMB18_X2Y46:RAMB18_X2Y47}
resize_pblock [get_pblocks pblock_myip_bitirme_rng_0] -add {RAMB36_X2Y23:RAMB36_X2Y23}
```

Figure 3.20: Specify constraints file

Our Layout looks like in Figure 3.21 and our RNG is inside the box.

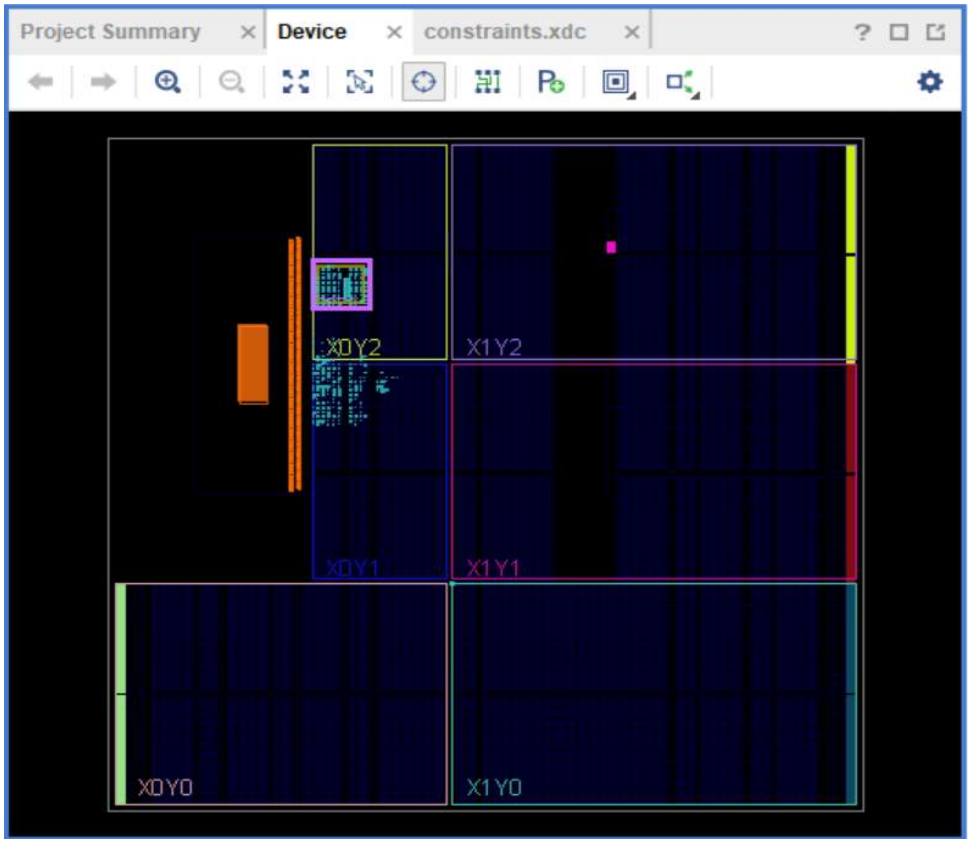

Figure 3.21: Layout showing the area constraints are met

.

The Vivado IDE allows to use one or many constraint files. While using a single constraint

file for the entire compilation flow might seem more convenient, it can be a challenge to maintain all the constraints as the design becomes more complex. This is usually the case for designs that use several IPs or large blocks developed by different teams.

In this project, area constraints are in XDC file which is the precedence rules for Xilinx® Design Constraints. XDC constraints are commands interpreted sequentially. For equivalent constraints, the last constraint takes precedence.

After the design is built, in Vivado file menu export hardware platform for software development tools is selected by including bitstream. In this step, we transferred the hardware description to the SDK. We have to notice the IP integrator block diagram and the implemented design should be clear.

Now we can "Generate Bitsream", "Export Hardware" and "Launch SDK" to start software development.

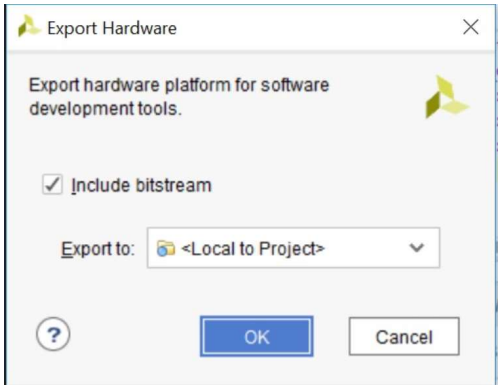

Figure 3.22: Including the Bitstream when Exporting Hardware

By using launch software development tool, we exported hardware to SDK. We created one of the available templates to generate a fully functioning application project by selecting 'Hello Word' with C language. Then the program started to compile until we saw the 'Build Finished' comment which means the software sent to card with FPGA program.

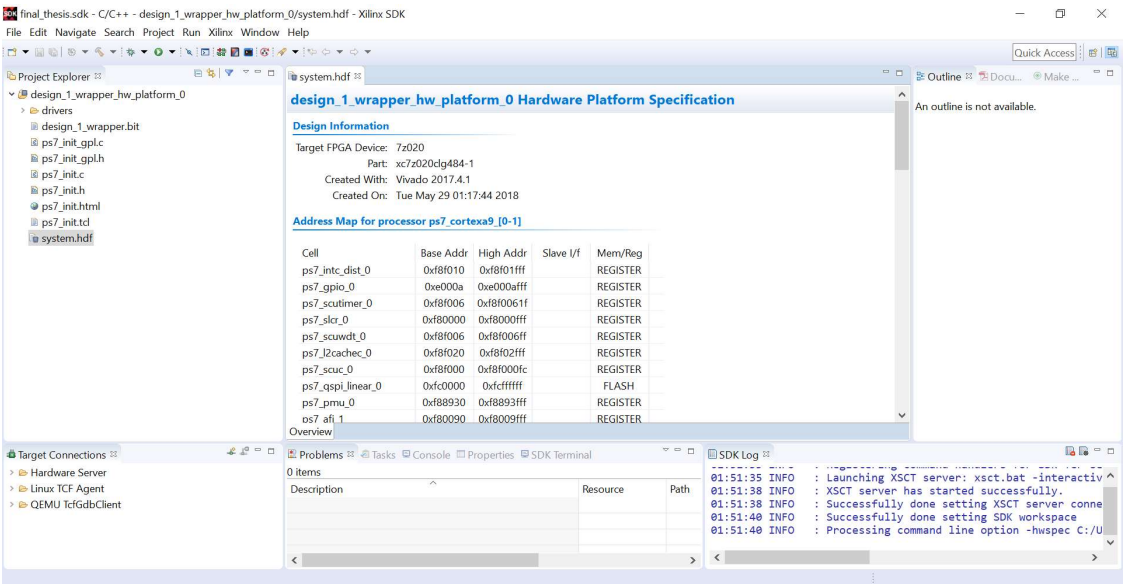

Figure 3.23: View of the SDK

Now, we click on the "New Application Project" and choose "Hello World" as a template. First we need to write our output data into SD_Card. We use f_mount, f_open functions to accomplish that.

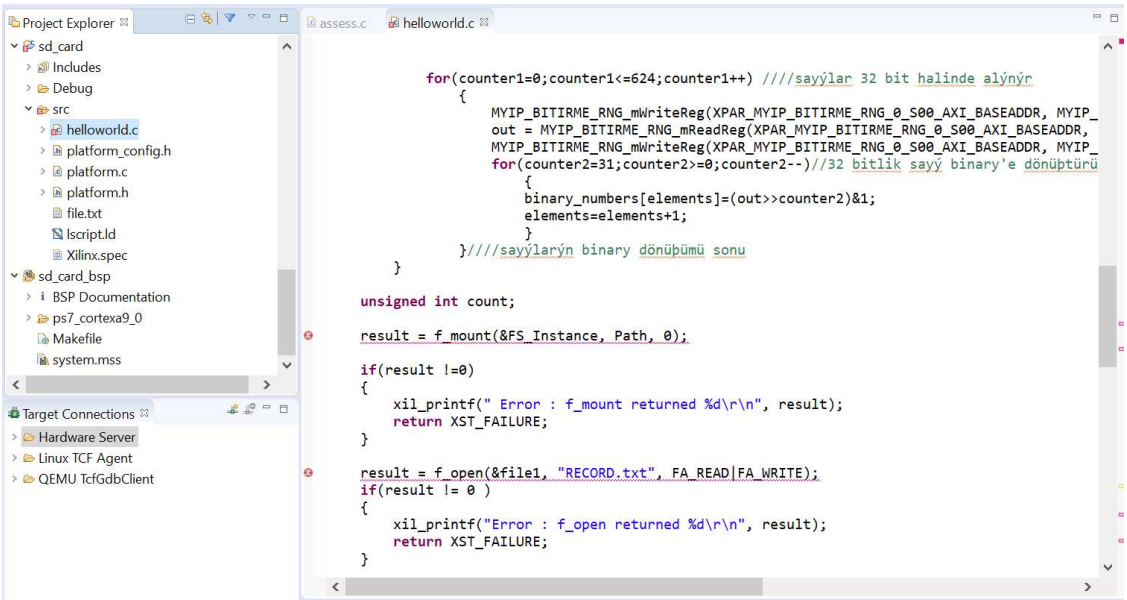

Figure 3.24: Our Application Project with SD Card

We need to define these functions by adding "xilffs" library by right clicking on the Board Support Package (BSP).

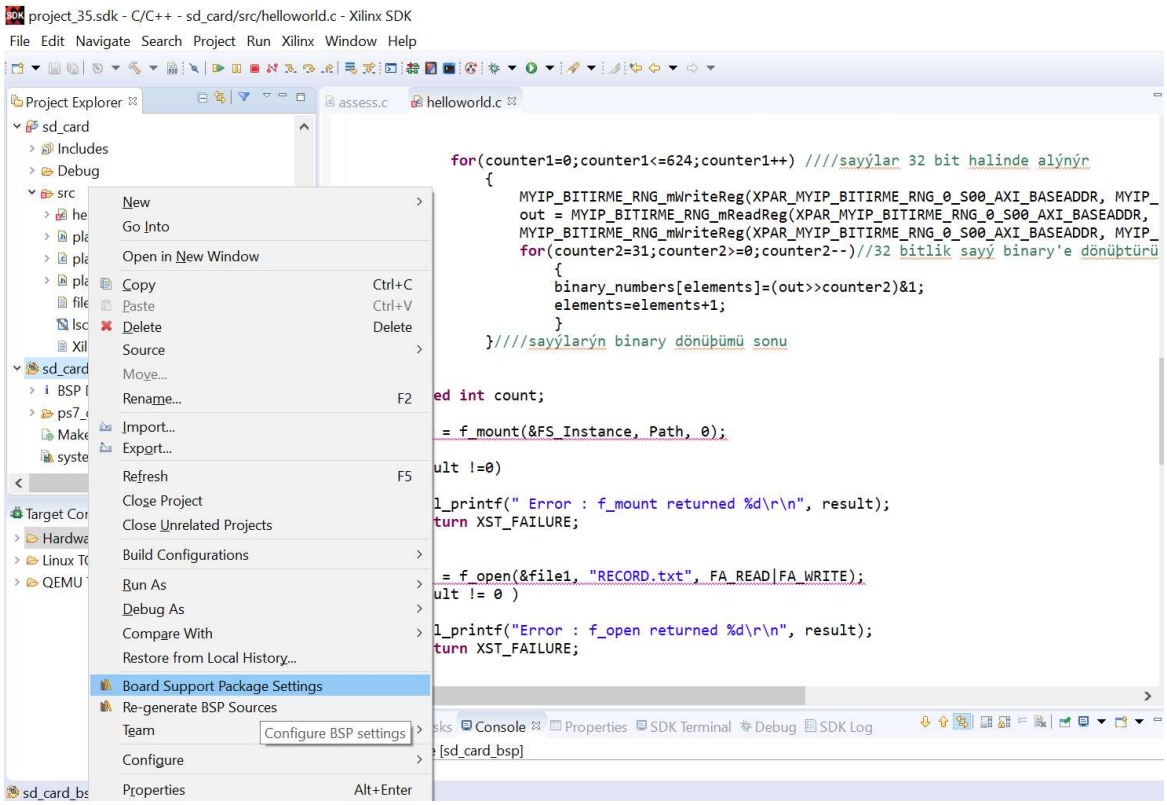

Figure 3.25: Adding “xilffs” library on the BSP

Xilffs Library is a Generic Fat File System Library that allows user to open or create a file in the SD_Card and read and write from it.

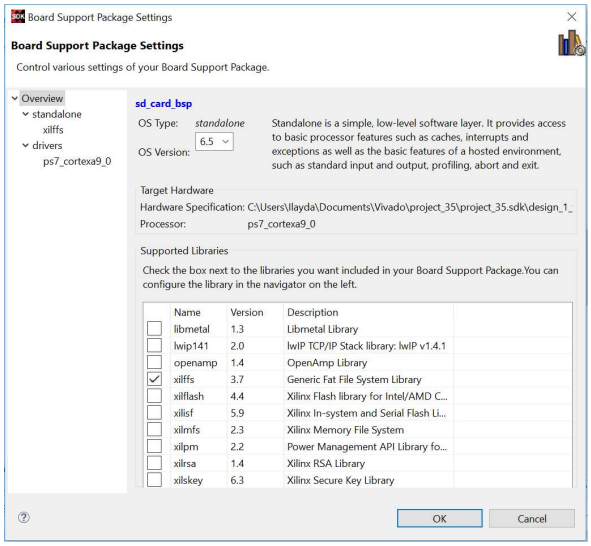

Figure 3.26: BSP Settings

By using “Launch Software Development Tool” we exported hardware to SDK.

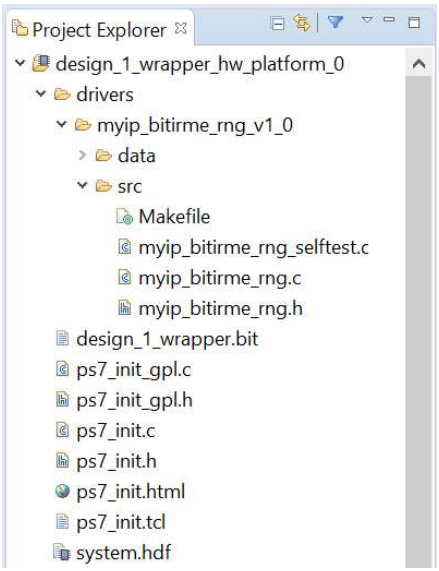

Figure 3.27: View of the Exported Hardware Platform from Vivado including IP driver

Another important remark is that, we need to add a linker to math library in order to be able to use “pow”, “sqrt” etc. functions that exist in the source code of NIST tests.

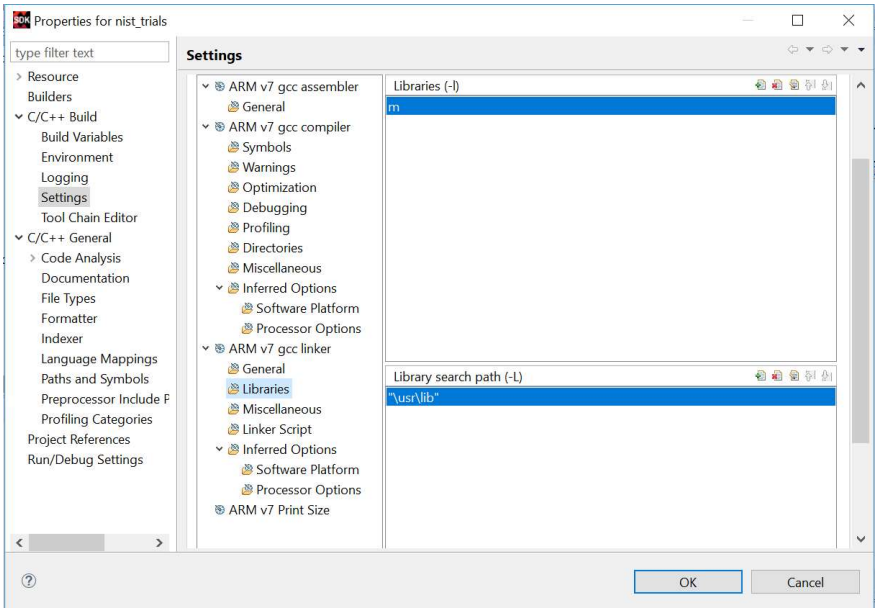

Figure 3.28: Adding a Linker to Math Library

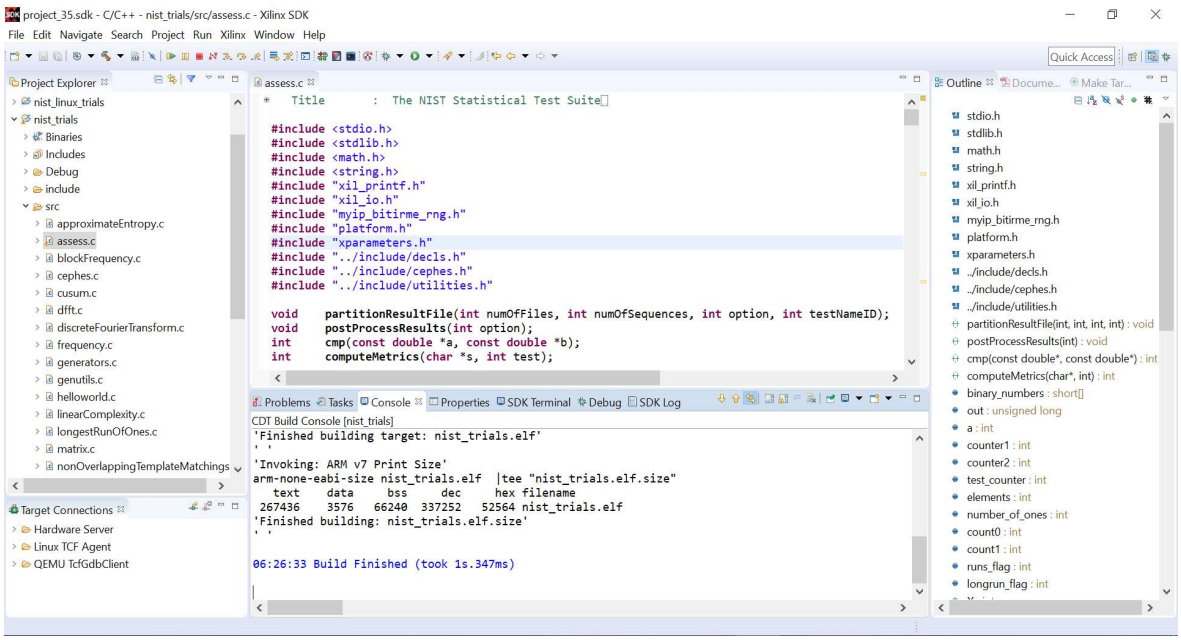

Figure 3.29: Application Project of NIST Tests in SDK

Even though the build is finished, we chose the 'Debug FPGA on System Debugger' option and we added the UART connection to check outcomes by using 'SDK Terminal' we have seen that we cannot find either the input prescribed file where our output data is written or other results files that should exist in the memory in order to complete the tests.

By saying "which doesn't support file system", we mean that the default "fopen", "fclose", "fwrite" and "fread" file system functions are not functioning correctly since ARM doesn't have neither memory allocation nor necessary cases such as threads.

We could replace them with specific functions that either "xillffs" library or "xilmfs" library. These functions actually support file system but changing default C functions such as "fopen" with these specific library functions such as "f_open" would have a huge impact in the code and would damage the originality of the code.

In order to test the numbers randomness with NIST Tests, we have to write the outputs in a text file and afterwards read from SD card and give them to tests as a prescribed input file.

4. SOFTWARE DEVELOPMENT: IMPLEMENTATION OF NIST TESTS

4.1 NIST Tests

The outputs of such generators may be used in many cryptographic applications, such as the generation of key material. Generators suitable for use in cryptographic applications may need to meet stronger requirements than for other applications. In particular, their outputs must be unpredictable in the absence of knowledge of the inputs. These tests may be useful as a first step in determining whether or not a generator is suitable for a particular cryptographic application. However, no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application. The National Institute of Standards and Technology (NIST) believes that these procedures are useful in detecting deviations of a binary sequence from randomness. Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. Randomness is a probabilistic property; that is, the properties of a random sequence can be characterized and described in terms of probability. There are an infinite number of possible statistical tests, each assessing the presence or absence of a pattern would indicate that the sequence is nonrandom because there are so many tests for judging whether a sequence is random or not. A statistical test is formulated to test a specific null hypothesis (H0). Associated with this null hypothesis is the alternative hypothesis (Ha), which is that the sequence is not random.

TRUE SITUATION	CONCLUSION	
	Accept H_0	Accept H_a (reject H_0)
Data is random (H_0 is true)	No error	Type I error
Data is not random (H_a is true)	Type II error	No error

Table 4.1 : Statistical tests results.[12]

If the data is, in truth, random, then a conclusion to reject the null hypothesis will occur a small percentage of the time. This conclusion is called a Type I error.

If the data is, in truth, non-random, then a conclusion to accept the null hypothesis called a Type II error.

The probability of a Type I error is often called the level of significance of the test. This probability can be set prior to a test and is denoted as α. For the test, α is the probability that the test will indicate that the sequence is not random when it really is random. That is, a sequence appears to have non-random properties even when a "good" generator produced the sequence. Common values of α in cryptography are about 0.01.

The probability of a Type II error is denoted as β. For the test, β is the probability that the test will indicate that the sequence is random when it is not; that is, a "bad" generator produced a sequence that appears to have random properties. Unlike α, β is not a fixed value. β can take on many different values because there are an infinite number of ways that a data stream can be non-random, and each different way yields a different β. The calculation of the Type II error β is more difficult than the calculation of α because of the many possible types of non-randomness.

One of the primary goals of the following tests is to minimize the probability of a Type II error. The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nations measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Special Publication 800-series reports on ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations. For cryptographic purposes, the output of RNGs needs to be unpredictable. However, some physical sources (e.g., date/time vectors) are quite predictable.[12]

These problems may be mitigated by combining outputs from different types of sources to use as the inputs for an RNG.

However, the resulting outputs from the RNG may still be deficient when evaluated by statistical tests. In addition, the production of high-quality random numbers may be too time consuming, making such production undesirable when a large quantity of random numbers is needed.

NIST Test Suite provides to user statistical tests that were developed to test the randomness of binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. There are 15 tests in suite.

1. Frequency Tests
2. Block Frequency
3. Cumulative Sums
4. Runs
5. Longest Run of Ones
6. Rank
7. Discrete Fourier Transform
8. Nonperiodic Template Matching
9. Overlapping Template Matching
10. Universal Statistical
11. Approximate Entropy
12. Random Excursions
13. Random Excursions Variant
14. Serial
15. Linear Complexity

Intellectual Property (IP) is developed on Zedboard which communicates with ARM processor to generate random numbers on a Secure Communication System and the system is verified with a Statistical Test Suite (NIST tests). Basicly, hardware design is exported to SDK and compiled by using NIST tests. Then NIST tests are applied to project.

This means the system works as a cryptographic system which can send and receive data correctly and the key used to encrypt/ decrypt this data has true randomness, which is verified by NIST tests.

Putting the system on NIST tests to measure the randomness of the key used in the process. After the NIST test are obtained, the values can be changed to optimize the system to be more random and make the NIST tests again.

Here is the some steps for compile NIST tests.

Figure 4.1 : User prescribed input file

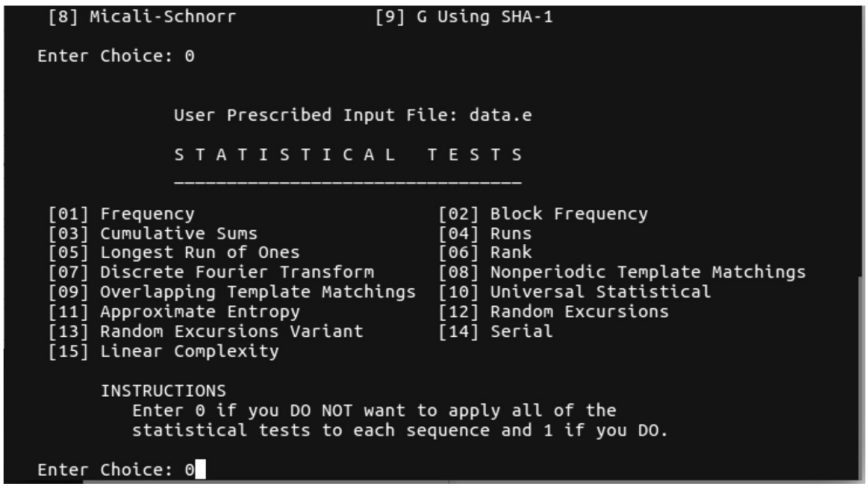

Figure 4.2 : Statistical Tests

In assess program the length of bit streams are arranged as a parameter. After entering a number of bit streams in the launch, some parameters are adjusted that are shown below.

```
Enter Choice: 0

      INSTRUCTIONS
         Enter a 0 or 1 to indicate whether or not the numbered statistical
         test should be applied to each sequence.

   123456789111111
            012345
   111111111111111


     P a r a m e t e r   A d j u s t m e n t s
     ----------------------------------------
  [1] Block Frequency Test - block length(M):         128
  [2] NonOverlapping Template Test - block length(m): 9
  [3] Overlapping Template Test - block length(m):    9
  [4] Approximate Entropy Test - block length(m):     10
  [5] Serial Test - block length(m):                  16
  [6] Linear Complexity Test - block length(M):       500

  Select Test (0 to continue): 0

  How many bitstreams? 10
```

Figure 4.3 : Number of Bitstream

```
     P a r a m e t e r   A d j u s t m e n t s
     ----------------------------------------
 [1] Block Frequency Test - block length(M):         128
 [2] NonOverlapping Template Test - block length(m): 9
 [3] Overlapping Template Test - block length(m):    9
 [4] Approximate Entropy Test - block length(m):     10
 [5] Serial Test - block length(m):                  16
 [6] Linear Complexity Test - block length(M):       500

Select Test (0 to continue): 0

How many bitstreams? 10

Input File Format:
 [0] ASCII - A sequence of ASCII 0's and 1's
 [1] Binary - Each byte in data file contains 8 bits of data

Select input mode:  0
```

Figure 4.4 : NIST Test Parameter Adjustments

4.2 Embedded Linux using C Language

NIST tests use file system operation functions such as fclose, fopen. NIST tests can not apply on some boards as Spartan because there is no file system in them. In ARM processor, that functions are defined and with Xilinx Vivado SDK we can compile directly without extra support system. Thus, we did not think face to any problem in process of compile but we realized that although we managed to compile, we did not reach the files. After some research about this problem, we found that there are too many libraries that support file system as MFS library. Multi Function Shield (MFS) library eliminates the need for using code snippets and pin definitions. Thus, it is enough to call the library functions. In this case, we had to change NIST Tests codes from the start to finish. This situation would cause the code to lose its originality and international validity. For this reason, we decided to install operating system on SD Card. The most efficient and applicable operating system was the Linux. We realized that we could solve the problem with petalinux application. Then we started to work on make the code compiled and executable. It was difficult process because Linux which is generated by petalinux does not have GCC library to compile code so we had negative result when we tried to build the project. The GNU Compiler Collection (GCC) includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages. In our project,the main aim is basicly put the Linux to boot on the ZedBoard.

4.2.1 Why Linux?

Linux has lots of device drivers. Device drivers is a kind of specialized software that is going to maka a particular hardware avaliable for the system. For an instance,Linex has drivers for USB cameras,GPRS modems etc.

Linux has network and file system support.So user can easily reach the SD card.Thank to memory management,user can share and use the same resource in memory. The program is be able to ask memory to the system to do something and give this memory back to the resources or to the other programs.

In Linux with interprocess communication,one or more crosses and excange data with each other in a safe manner. Also it is portable.Linux has huge community where user can get information,learn features eaily thanks to free open source advantage.Also if the user face o any kind of problem,finding the solution is easier than other operating systems.By means of all these reasons,we used Linux operating system to compile NIST Tests and execute our project. Although the all advanteges,Linux has disadvantages too as compiling,porting the Kernel and drivers can be a challenge for the user. Also it is complex which means it takes time to boot compared to baremetal stuff.

4.2.2 SD Card

We used SD card to boot our system. Majority of the space used by SD card is root filesystem. Bootloader and Kernel have less space that root filesystem in SD Card.
We need 4 items to be placed on the ZedBoard SD Card as shown below.

1.Linux filesystem : linaro or busyboz
2.Linux kernel image : zImage
3. BOOT.bin (U-boot+ FSBL+ Bitstream)
4.Compiled device tree

SD Card Partitions:
The user need at least 2 partitions, the first with 1Gb with FAT format that will hold u-boot, Linux and the device tree. The other partition with 3Gb with ext4 format for the file system that is used by linaro. Also the user can use Linux gparted utility to create partitions, or the normal disk utility in Ubuntu.

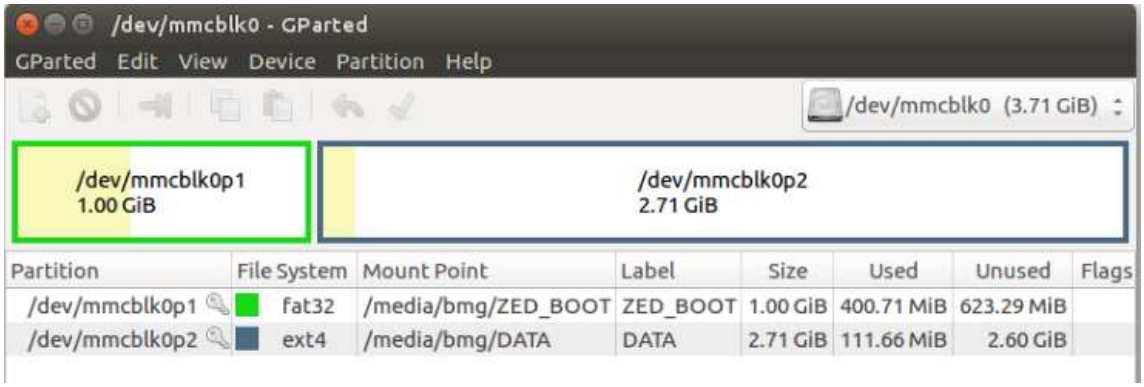

Figure 4.5 : Contensts of SD card

1.Boot.bin it has mix together our first stage boot loader or our bitstream. BOOT.bin this the one that is going to have boot the first stage bootloader in our bitstream.

2.System.dtb device tree

3.zImage that is linux image is prepared to be loaded from the boot.

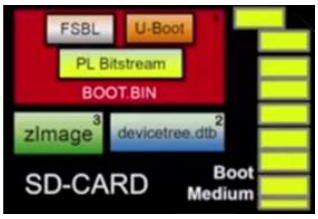

Figure 4.6 : General view of SD Card

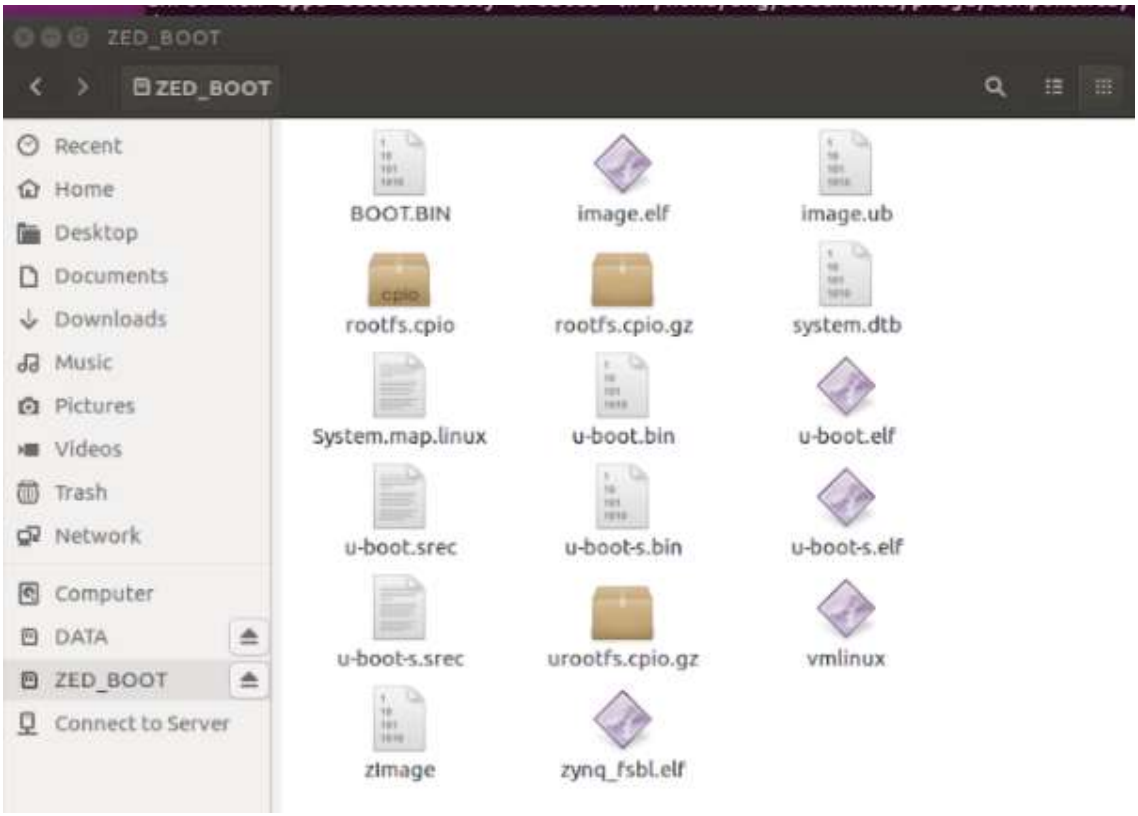

Figure 4.7 : Zed-boot

4.2.3 Petalinux and Create Simple Application

Petalinux is a free of charge but commercial if the user want some support. Petalinux is responsible to create the following packages as u-boot, FSBL, Linux Kernel, device tree, rootfs.

Linux Kernel has implemented mainly in C a little bit of assembly components as memory management, device drivers, driver frameworks, scheduler task management, low level architecture specific code, file system layer and drivers, and network stack. Also it has device tree on some architectures that is written in a Device tree specific language. New feature called device tree which is set of text file that is compiled in boot with the Linux. It specifies which hardware user have on their board and which device driver is responsible to control that hardware.

The Linux Kernel job is on a system a lot of programs asking for resources at the same time, is the job of kernel to manage all off these demands as process management, memory management, file systems, device control, network.

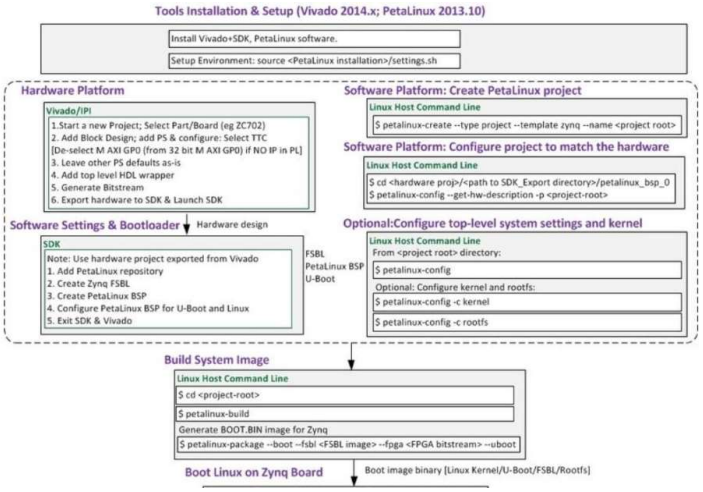

Figure 4.8: General Design Flow for Hardware and Software Design with Vivado and Petalinux.

Rootfs:

On Linux systems, several filesystems are mounted and create a global hierarchy of files and directories, as a particular filesystem,the root filesystem,is mounted as this root filesystem contains all the libraries, applications and data of the system.

Therefore, building the root filesystem is one of the main tasks of integrating embedded Linux components into a device.

System Call:

The Kernel offers an API library of functions to user space programs,those libraries are avaiable through the GNU C library.For an instance printf,fopen,fclose,exit, etc...

Kernel and the User Space Memory:

On Linux the memory is divided in two main parts the kernel and user space and they communicate through system calls.The job of the kernel is allow multiple user space programs protection between them and a bridge to the system through device drivers.

The user space have access to a memory called virtual memory which is managed by the Kernel.

Device Drivers:

On linux systems the hardware is made available to the user space programs through files by the way everything on linux is a file. So the job of the device driver is to make the user space think that the hardware is just a file that is placed on a special directory.Also when we created drivers on baremetal systems. We have access to the physical memory, and on Linux the memory is virtual and always translated by a piece of hardware called MMU (Memory Mangement Unit).

We followed some steps to boot Linux on ZedBoard Zynq 700 by SD-Card. One of the challenge is that, Petalinux does not run on Windows Operating Systems. Also since there are many versions of Petalinux, it is hard to follow a guide.

PetaLinux Tools Installation Requirements:

- 4 GB RAM (recommended minimum for Xilinx tools)
- Pentium 4 2GHz CPU clock or equivalent
- 5 GB free HDD space
- Supported OS:

  RHEL 5.9 (32-bit or 64-bit)

  RHEL 6 (32-bit or 64-bit)

  SUSE Enterprise 11 (32-bit or 64-bit)

CentOS 6 (64-bit)

Ubuntu 14.04 (64 bit)

• You need to have root access to perform some operations

PetaLinux requires a number of standard development tools and libraries to be installed on Linux host workstation. The PetaLinux installation process checks for these packages, and reports an error if any are missing. However, it does not attempt to install them - you must do this manually. For a 64-bit Linux host, you must install the appropriate 32-bit compatible libraries. The table below describes the required packages, and how to install them on different Linux workstation environments. After installing is done reconfiguration of dash is recommended for building problems.

Tool/Library	YUM/RPM Package for RHEL/CentOS/Fedora	APT Package for Debian/Ubuntu	RPM Package for SuSE
dos2unix	dos2unix	tofrodos	dos2unix
ip	iproute	iproute	iproute2
gawk	gawk	gawk	gawk
gcc	gcc	gcc	gcc
git	git	git-core	git-core
make	gnutls-devel	make	make
netstat	net-tools	net-tools	net-tools
ncurses	ncurses-devel	ncurses-dev libncurses5-dev	ncurses-devel
tftp server	tftp-server	tftpd	tftp-server
zlib	zlib-devel	zlib1g-dev	zlib-devel
flex	flex	flex	flex
bison	bison	bison	bison
32bit libs	libstdc++.i686, glibc.i686, libgcc.i686, libgomp.i386, ncursese-libs.i686, zlib.i686	setup 32bit libs support in ubuntu, lib32z1, lib32ncurses5, lib32bz2-1.0, ia32gcc1, lib32stdc++6, libselinux1	32-bit runtime environment

Figure 4.9 : Some PetaLinux Libraries

After requirements are met, we prepared the Ubuntu 16.04. The steps are basically to compile Petalinux such as downloading Petalinux version 2014.4, downloading the board BSP (Board Support Package), creating directory (/opt/Petalinux) and running the installer to pass the directory. BSP package we downloaded from the website was Avnet-Digilent-ZedBoard-v2014.4-final.bsp

The following command is executed in Linux for installing the Petalinux into the system

$ ./petalinux-v2014.4-final-installer.run /opt/pkg

For creating Linux project based on a BSP board (in our case it is ZedBoard) that command is

$ petalinux-create –t project –n <NameoftheProject> –s bsps/Avnet-Digilent-ZedBoard-v2014.4-final.bsp

After pressing enter button, basically the project has been created. Then we compile the sources by using petalinux-build and we can copy the necessary files to SD-Card. Shortly, the project is created on Petalinux and it is built. The parameters as Kernel and rootfs are changed. Then simple app is added on the system.

For building a simple system which will run in Linux controlling a minimum hardware system; on Vivado, following steps can be followed. Firstly, a normal project will be created with the ZYNQ IP core. The key point is that when making connection automation, the clock should be connected to the M_AXI_GP0_ACLK so that the system can be connected to Linux. Basic steps such as generating the bitstream after synthesis and implementation, exporting the project to Xilinx SDK and lastly creating new application in language.

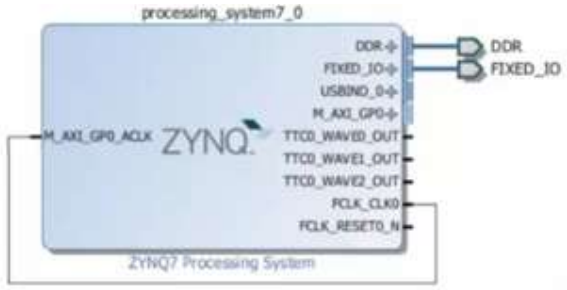

Figure 4.10 : Minimum Hardware on Vivado to run Linux

Here is the some directories as /tftboot it has the linux images,the rootfs and the device tree.PRJ/pre-built/linux/images: it has the FSBL and BOOT.bin.After building process,we copied items to SD card and brought them on ZedBoard and it is turned on.BOOT.bin this the one that is going to have boot the first stage bootloader in our bitstream. The file BOOT.bin is precompiled on the prebuild directory, but if you would like to regenerate it,(for instance if you would like to use another hardware on the FPGA, you can recreate with the petalinux-package command, the file will be generated on your current directory. Basically just reconfigured the rootfs to add the dropbear package, that is included on File System/Console Network to access ZedBoard through the network.

Petalinux-create –t apps –template c++ --name <ProjectName> command used to compile C program. A simple project is created on components/apps/, then you need to reconfigure your rootfs to include your program, then build (petalinux-build).

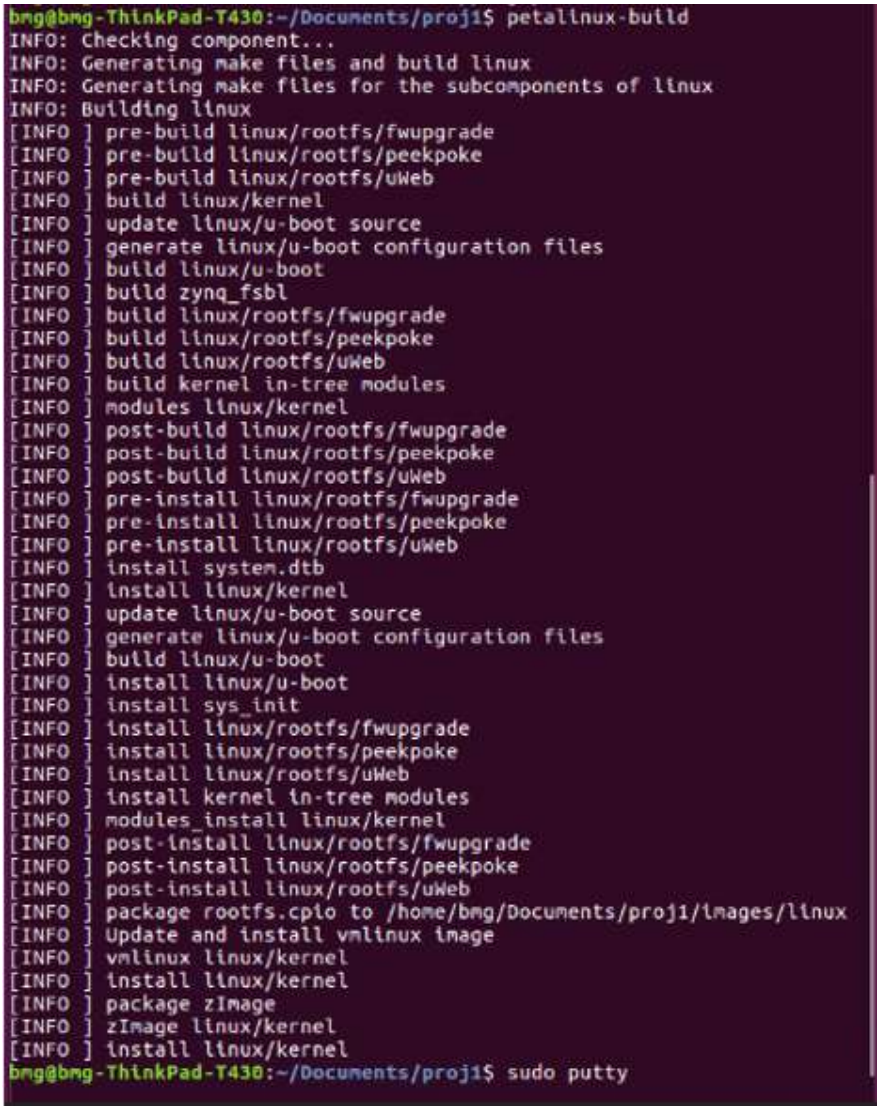

Figure 4.11 : Building the project and opening putty

Putty is a client program used for things like telnet and SSH. It's very handy for operating a remote or headless system that enables any users to remotely access computers over the internet. Log into a server via putty means users are authenticated/authorized for the server and users can run your command on their server which is mostly in encrypted form. In other word you can say secured log in via putty. We used putty which is a terminal program for the configuration of the project. We selected ttyACM0 for serial port. We checked jumpers are configured to SD card, placed it, then booted, used putty to view the results from the serial port. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.

A cross compiler is necessary to compile for multiple platforms from one machine. A platform could be infeasible for a compiler to run on, such as for the microcontroller of an embedded system because those systems contain no operating system. In virtualization one machine runs many operating systems, and a cross compiler could generate an executable for each of them from one main source.

The command run bootcmd or just boot is used to run u-boot. The Kernel is trying to check the network interface.

Some configuration commands are explained like petalinux –c kernel be used to change kernel parameters as enable some debug messages, petalinux –c rootfs command be used to change rootfs parameters. After those changes we need to rebuild the system. By using Syste.map.linux command is used to boot. After boot is done all the files are copied that has been generated. Then putty console terminal is run. Also, we can see time information that is enabled on Linux Kernel. If every steps are done correctly, application should be in the rootfs filesystem. When go to the bin directory, the user should see the application. We learned that by using petalinux we can create linux distributions that are compiled with the ZedBoard. All that we need is board support package in our case that is Zynq.

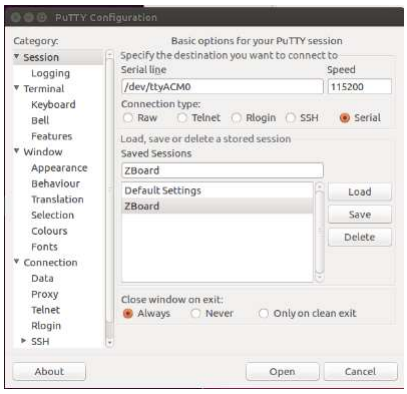

Figure 4.12 : Putty Configuration

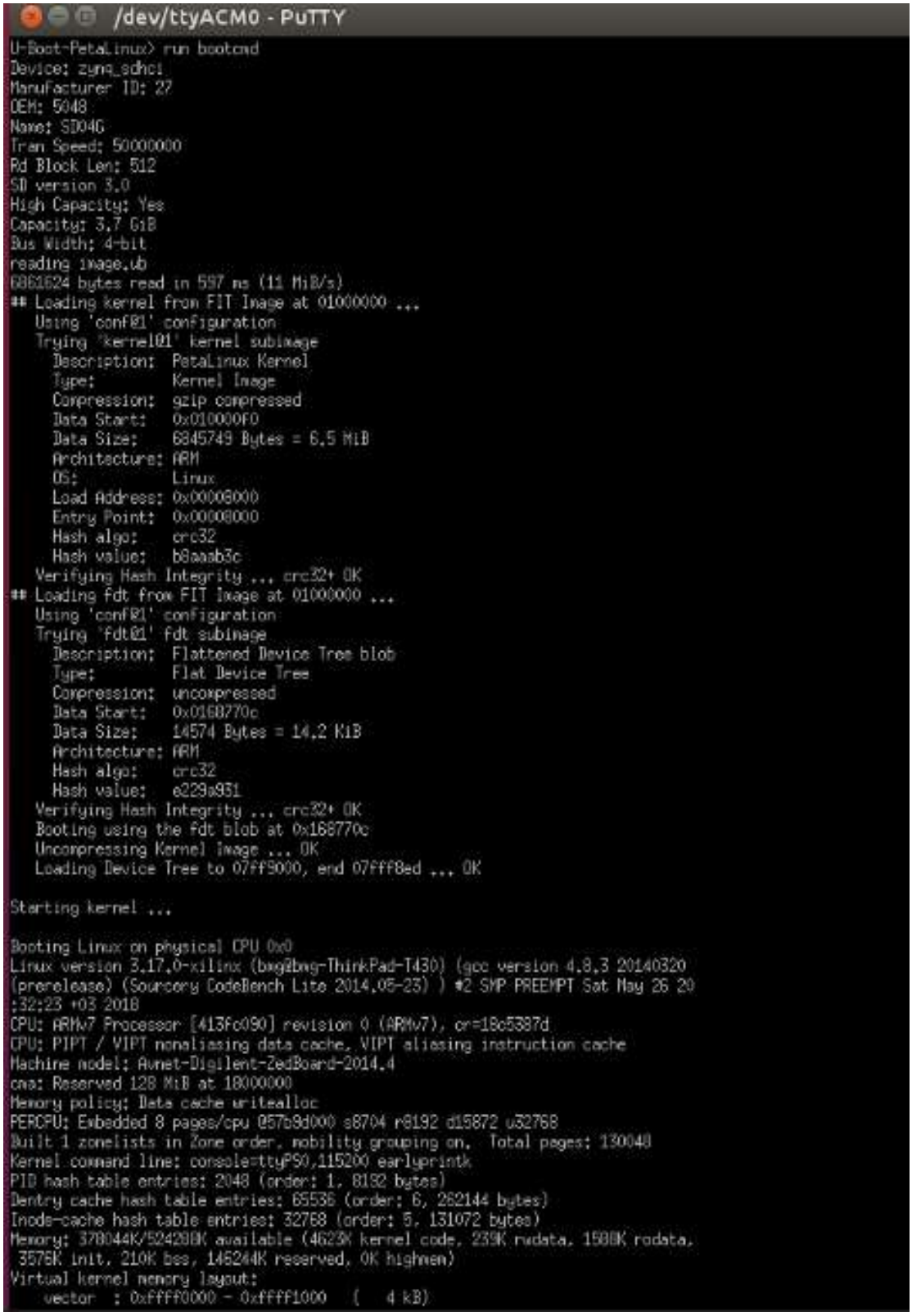

Figure 4.13: Booting Linux Window

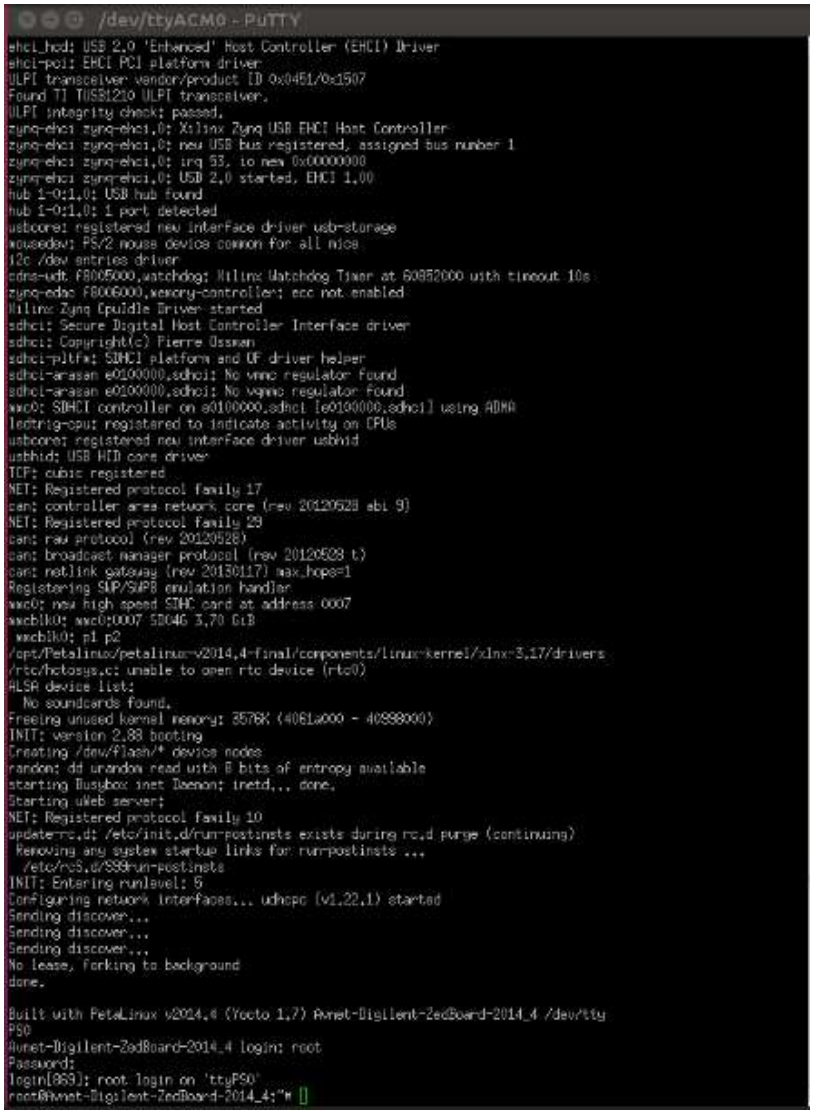

Figure 4.14 : General view on Linux

When we compile NIST tests, we connected the existing files each other which was twelve C files and eight Header files. NIST has chosen to use the C language special functions math library in the test software.

The ARM compiler in SDK is failing to compile applications containing “pow”, “sqrt” etc. functions as mentioned. In order to fix this problem, the math "m" option needs to be specified in the Libraries in the C/C++ Build Settings. We use makefile to compile NIST Tests as ARM can recognize. We obtain makefile file by creating application with PetaLinux. To solve linker problem, we added –lm line to our code as shown in below. On the other hand,to compile tests we changed the asses.c file completely.

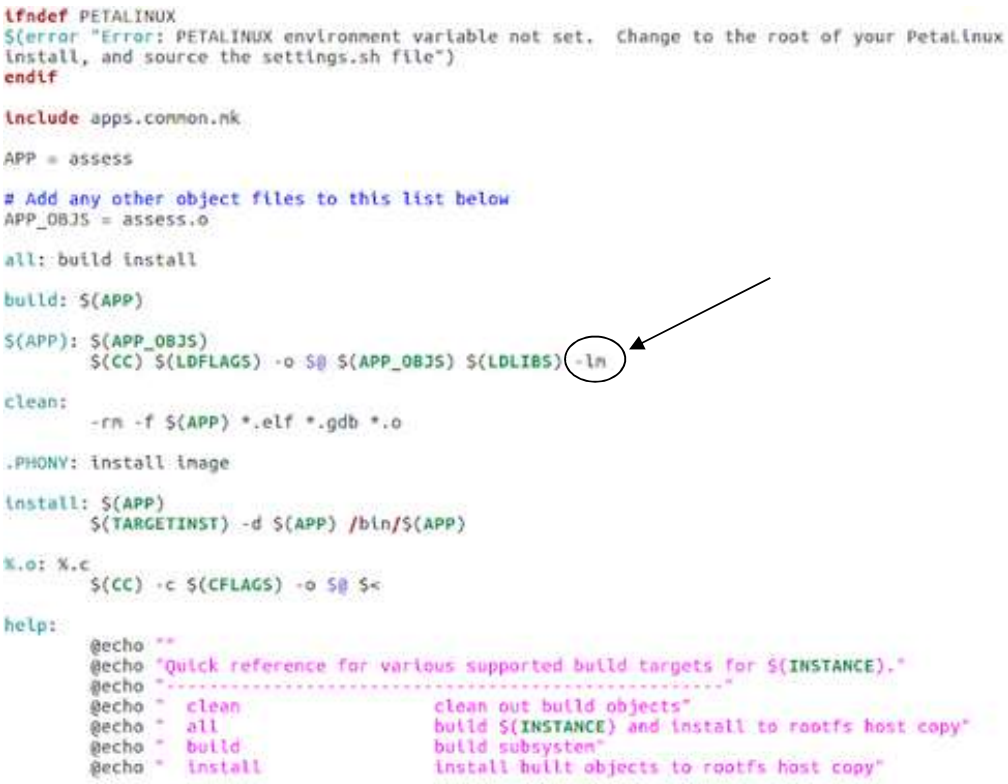

Figure 4.15: PetaLinux makefile

5.FINAL REMARKS

5.1 Possible Applications and Future Work Recommendations

In combined design of software and hardware, a device driver is necessary to control or operate the hardware design with the software. In our system a device driver should be designed to control our random number generator by the operating system we generated, which is Linux.
This way, the tests will be in real time which will allow experiments of different applications such as grasping the effect of temperature. The design can also be tried with different clock frequencies to see if the randomness increases and a possible Physically Unclonable Function (PUF) is created. Last but not least, there is a possibility that SDK can create a Linux image without the help of Petalinux Tool. This should be investigated and applied to see if it works.

5.2 Realistic Constraints

5.2.1. Cost

To determine the cost of the system, three different criteria were chosen: cost of additional circuitry, unit price and price of the labor. Some devices may require additional circuitry to generate random numbers but in our system, there is no additional circuitry cost in our project. The unit price of the FPGA used is a very important factor that contributes to the overall cost of the design. The specific FPGA that will be used in our system is the Zedboard Zynq-7000 and unit price of this device in Turkey is TRY2,356.90. Our salary based on a junior engineer's salary will be calculated assuming working one day per week (full time) for 28 weeks. Which means 28 days * salary* two people should be estimated. We can assume this number rounds up to TRY8,500 for the whole project.

5.2.2. Standards

IEEE and NIST standards will be used for documentation of the research project.

5.2.3. Social, environmental and economic impact

Safe communication became more important with the developments in technology industry. A new random number generator will be designed which meet the need of can secure devices that can be used in daily life without a high additional cost. This will improve the secure

communication which will affect the society and economy positively. Also no additional circuitry is good for environmental factors as it will decrease the pollution by decreasing the power consumption.

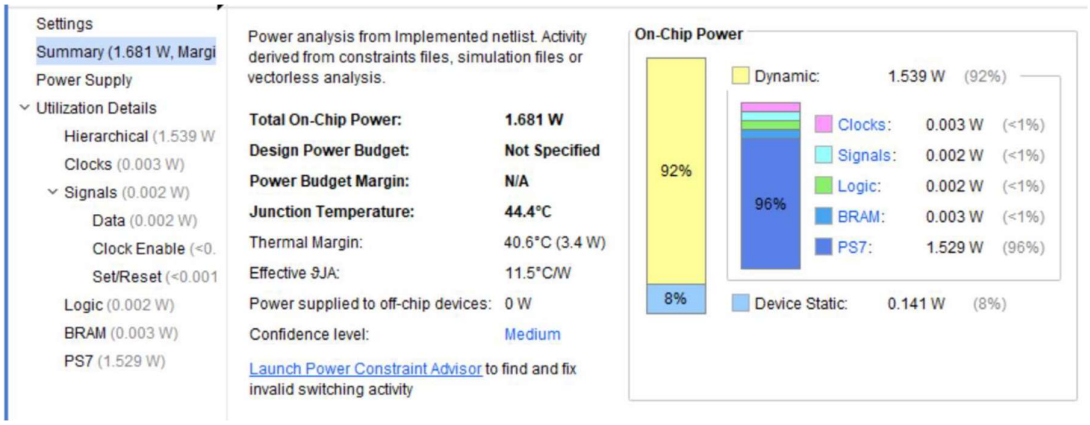

Figure 5.1: Power Consumption

6.CONCLUSION

To sum up our project, first we generated random numbers and wrote them into a SD card and then we tested them by using Linux operating system. The system is tested on both computer and FPGA board in order to compare and contrast the data we obtained.

Our hardware design generates true random numbers and these numbers are tested inside the specified board which is ZedBoard Zynq-7000 ARM/FPGA SoC Development Board by our software design. Hardware section includes generating random numbers by a custom hardware and software section that has compiled NIST Tests to test random numbers created. The random number generator was implemented with the help of Hardware Description Language, Verilog, and Vivado Design Suite. In software part, using Xilinx SDK to compile NIST tests wasn't enough so a solution is developed by extensive research; which is running Linux as an operating system. Creating the image of Linux is done by PetaLinux Tool. As a result, NIST Tests are compiled with the Linux operating system and booted from the SD Card. The system was not tested in real time because there was not any device driver developed for this specific purpose.

As a result, for our random number generator can be embedded into Internet of Think devices to make the system more secure and create useful applications. Our system can be used to build secure systems since we used a chaotic time delay random number generator and tested our results with universally accepted NIST tests. The NIST tests are quantitative and measureable and also the input and the output of the system must be the same, which can be evaluated easily.

REFERENCES

[1] Menezes, Alfred J., Paul C. Van Oorschot, and Scott A.Vanstone.1996. Handbook of applied cryptography. CRC press.

[2] Galajda, M. D. P. (2006). True Random Number Generator Embedded in a Reconfigurable Hardware. Journal of Electrical Engineering, 57(4), 218-225.

[3] Pareschi, Fabio, Gianluca Setti, and Riccardo Rovatti. ”Implementation and testing of high-speed CMOS true random number generators based on chaotic systems.” IEEE transactions on circuits and systems I: regular papers 57.12 (2010): 3124-3137.

[4] Callegari, Sergio, Riccardo Rovatti, and Gianluca Setti. ”Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos.” IEEE Transactions on Signal Processing 53.2 (2005): 793-805.

[5] R. Yeniceri and M. E. Yalcin, “True random bit generation with sampled-data feedback system”, Electronics Letters, vol. 49, no. 8, pp. 543-545, April 2013

[6] Yalcin, Mustak E., Ramazan Yeniceri, and Serdar Ozoguz

”A chaotic time-delay sampled-data system and its implementation.” International Journal of Bifurcation and Chaos 24.03 (2014): 1450039.

[7] Yeniceri, R. and Vardar, A. and Yalcin, M. E., ”Full Digital Implementation of A Chaotic Time-delay Sampled-data System”, Circuits 2017 IEEE International Symposium on and Systems (ISCAS), May 2017, accepted.

[8] Yeniceri, R., Vardar, A., Cil, E., Akcay, L., Goncu, E., and Yalcin, M. E., ”A Chaotic Time-delay System Based Digital RNG and Integrated Autonomous Test Suite”, The 23European Conference on Circuit Theory and Design (ECCTD), September 2017, accepted.

[9] Çil,Erdem ‘Soft RNG Tester:System on Chip Implementation of an RNG’,Bacholar Thesis.

[10] Wheeler, David J., and Roger M. Needham. ”TEA, a tiny encryption algorithm.” International Workshop on Fast Software Encryption. Springer Berlin Heidelberg, 1994.

[11] American National Standards Institute: Financial Institution Key Management (Wholesale), American Bankers Association, ANSI X9.17 - 1985 (Reaffirmed 1991).

[12]Rukhin A.,Soto, J.,Nechvatal,J.,Smid,M.,Barker,E. And Leigh,S. 'A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications'

[13] Linda E.M. Brackenbury, Luis A. Plana, *Senior Member, IEEE*
and Jeffrey Pepper. 'System on Chip Design and Implementation'

[14] Block Memory Generator v8.3 *LogiCORE IP Product Guide,April 5,2017*

[15] Vivado Design Suite User Guide *Using Constraints* ,UG903 (v2012.2) September 4,2012

[16] PetaLinux Tools Documentation Reference Guide,UG1144 (v2014.4) November 25, 2014

CURRICULUM VITAE

Name Surname **:** İlayda Yaman

Personal Information

Work Address: Istanbul Technical University,
Electrical and Electronics Engineering Department
Telephone: 05304901096,
E-mail : ilaydayaman@gmail.com

Professional Profile

A committed and proactive professional who can think in many ways and give lots of perspectives to projects. Also, an extremely quick learner, an analytical and creative thinker who can adopt very well, communicate efficiently and get along with different types of people easily. Quick to get into the team, understand its potential and innovate vision. Can push limits to achieve goals, encourage others to hard work also and looking forward to playing a key role in computer based projects as well as new, high technology projects. A world citizen and sportsman scientist who enjoys being part of successful and productive team, and thrives in highly challenging working environments.

Objective

Currently searching for a Master's Degree challenging, best fitting to existing skills and giving self-improvement in personal and professional skills.

Education and Qualifications

- Istanbul Technical University- Electronics and Communication Engineering (2014-2018)

  University of Waterloo, Electrical Engineering, Exchange student (2016-2017)
- Ted Ankara College (2002-2014)
- International Baccalaureate Diploma from Ted Ankara College (2014)

Conference Papers

ELECO 2017 10th International Conference on Electrical and Electronics Engineering published in IEEE Xplore Library

- *Latif Akçay, Erdem Çil, Alptekin Vardar, İlayda Yaman, Ramazan Yeniçeri, Müştak E. Yalçın* "Implementation of a Chaotic Time-Delay RNG Based Secure Communication System on FPGA"

Experience

- Final Engineering BSc. thesis on Xilinx ZedBoard, ARM FPGA development board Developing an IP on board for communication with ARM processor to generate random numbers on a Secure Communication System and verify it with a Statistical Test Suite (NIST tests). Supervisors: Latif Akçay, Müştak E. Yalçın
- Internship at AnkaSys, Anka Microelectronics Systems in 2017 summer on Verilog, SystemVerilog, UVM and FPGAs.

  - Worked in ModelSim and Questa Simulators with Verilog and SystemVerilog

  - Teaching assistant on Verilog and SystemVerilog Trainings for Roketsan Company
- Summer intern at RF Electronics Laboratory about Microcontrollers and Analog circuit design in 2016 summer.
- 3 months of lab experience on Embedded Microprocessor Systems in University of Waterloo, worked on FPGA development and embedded systems with C language
- Lead Earth Charter project (as a part of Global Citizen program of AIESEC) in South America for 7 weeks in 2015 summer

Further Skills

- Comprehensive knowledge of Xilinx ISE and Vivado Toolchain understanding of how to persuade this tool to work with FPGAs.
- Accomplished embedded programmer (C languages) understands working with hardware/debugging of the register level
- Extended knowledge in HDL languages with a deep understanding on Behavioral Level, RTL, Gate/Structural Level leading to FPGA/ASIC design

- Studied 80 hours of course in Beginning to Programming, 160 hours of course in C language and currently, studying “Advanced C and System Programming” in “C ve Sistem Programcıları Derneği”
- Worked on Altera Toolchain and persuade the tool to communicate with FPGAs
- IAR Embedded Workbench

Languages

Fluent English, intermediate German and beginner Portuguese

Interests and Activities

- Member of Toastmasters Club in İstanbul
- Attended a summer program on personal development in Pine Manor College in Boston for one month in 2012
- Attended Summer Discovery program in Cambridge, England for three weeks in 2011
- IEEE Communications Society’s highly active member

CURRICULUM VITAE

Name Surname : Merve Firik

Place and Date of Birth : 18.04.1995

E-Mail : firik@itu.edu.tr

Personal Information

Work Address: Istanbul Technical University,
Electrical and Electronics Engineering Department
Telephone: 05438119009

Education and Qualifications

- Istanbul Technical University- Electronics and Communication Engineering (2013-2018)
- University of Technology Vienna, Electrical Engineering, Erasmus student (2016-2017)

Experience

- Final Engineering BSc. thesis on Xilinx ZedBoard, ARM FPGA development board Developing an IP on board for communication with ARM processor to generate random numbers on a Secure Communication System and verify it with a Statistical Test Suite (NIST tests). Supervisors: Latif Akçay, Müştak E. Yalçın
- Internship at Roketsan A.Ş
- Internship at Mercedes Benz
- Internship at Tubitak SAGE AŞ