**ISTANBUL TECHNICAL UNIVERSITY**
**ELECTRICAL-ELECTRONICS FACULTY**

# FPGA Implementation for OnSite Target Detection with a Low Cost and Portable Ground Penetrating Radar System

**SENIOR DESIGN PROJECT**

**Muhammed Furkan ERTURAL**

**Çisem KURT**

**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT**

**JANUARY, 2023**

**ISTANBUL TECHNICAL UNIVERSITY**
**ELECTRICAL-ELECTRONICS FACULTY**

# FPGA Implementation for OnSite Target Detection with a Low Cost and Portable Ground Penetrating Radar System

**SENIOR DESIGN PROJECT**

**Muhammed Furkan ERTURAL**

**(040180122)**

**Çisem KURT**

**(040200790)**

**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT**

**Project Advisor: Prof. Dr. Berna Örs Yalçın**

**Project Co-Advisor: Prof. Dr. Işın Erer**

**JANUARY, 2023**

**İSTANBUL TEKNİK ÜNİVERSİTESİ**
**ELEKTRİK-ELEKTRONİK FAKÜLTESİ**

# Düşük Maliyetli ve Taşınabilir Yer Nüfuz Eden Radar Sistemi ile Yerinde Hedef Tespiti için FPGA Uygulaması

**LİSANS BİTİRME TASARIM PROJESİ**

**Muhammed Furkan ERTURAL**
**(040180122)**

**Çisem KURT**
**(040200790)**

**Proje Danışmanı: Prof. Dr. Berna ÖRS YALÇIN**
**Proje Yardımcı Danışmanı: Prof. Dr. Işın ERER**

**ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ**

**OCAK, 2023**

We are submitting the Senior Design Project Interim Report entitled as “FPGA Implementation for OnSite Target Detection with a Low Cost and Portable Ground Penetrating Radar System”. The Senior Design Project 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.

[illegible]

**Muhammed Furkan ERTURAL**
(040180122)

[illegible]

**Çisem KURT**
(040200790)

# FOREWORD

First of all, we would like to thank our project advisor Prof. Dr. Sıddıka Berna ÖRS YALÇIN, who helped us with her knowledge and experience in this project and is understanding in every respect, Prof. Dr. Işın ERER who is our assistant advisor and Prof. Dr. Selçuk PAKER who is supported us for their support and assistance during the project. We would also like to thank RA. Orhan APAYDIN and Canberk TATLI who worked with us on this project. Finally, we would like to express our endless gratitude to our families who have supported us and made all kinds of sacrifices throughout our university life.

January 2023

Çisem KURT
Muhammed Furkan ERTURAL

## TABLE OF CONTENTS

## ABBREVIATIONS

| | |
|---|---|
| **AC** | : Alternative Current |
| **ASIC** | : Application Specific Integrated Circuit |
| **AXI** | : Advanced eXtensible Interface |
| **BRAM** | : Block Random Access Memory |
| **CANBUS** | : Controller Area Network Bus |
| **CPU** | : Central Process Unit |
| **DC** | : Direct Current |
| **DDR** | : Double Data Rate Synchronous Dynamic Random-Access Memory |
| **DMA** | : Direct Memory Access |
| **DSP** | : Digital Signal Processing |
| **EMI** | : Electromagnetic Interference |
| **FPGA** | : Field Programmable Gate Array |
| **GMAC** | : Galois Message Authentication Code |
| **GPR** | : Ground Penetrating Radar |
| **GPIO** | : General Purpose Input and Output |
| **HDL** | : Hardware Description Language |
| **HLS** | : High Level Synthesis |
| **IDE** | : Integrated Development Environment |
| **I2C** | : Inter-Intergrated Circuit |
| **IEEE** | : Institute of Electrical and Electronics Engineers |
| **IP** | : Intellectual Property |
| **LED** | : Light Emission Diode |
| **LUT** | : Look-Up Table |
| **MCU** | : Micro Controller Unit |

| | |
|---|---|
| **MIO** | **:** Multi Input-Output |
| **MMC** | **:** Multi Media Card |
| **NMF** | **:** Non-negative Matrix Factorization |
| **OTG** | **:** On The Go |
| **OLED** | **:** Organic Light Emitting Diode |
| **PS** | **:** Programmable Software |
| **PL** | **:** Programmable Logic |
| **RNMF** | **:** Robust Nonnegative Matrix Factorization |
| **RPN** | **:** Region Proposal Network |
| **RXD** | **:** Receive Data |
| **SD** | **:** Secure Digital Memory Card |
| **SDIO** | **:** Secure Digital Input Output |
| **SDSoC** | **:** Software Defined System On Chip |
| **SDK** | **:** Software Development Kit |
| **SPI** | **:** Serial Peripheral Interface |
| **SoC** | **:** System on Chip |
| **TXD** | **:** Trasmit Data |
| **UART** | **:** Universal Asynchronous Receiver Transmitter |
| **USART** | **:** Universal Synchronous Receiver Transmitter |
| **USB** | **:** Universal Serial Bus |
| **JTAG** | **:** Joint Test Action Group |
| **VHDL** | **:** Very High Speed Integrated Circuit Hardware Description Language |

## LIST OF FIGURES

# FPGA IMPLEMENTATION FOR ONSITE TARGET DETECTION WITH A LOW COST AND PORTABLE GROUND PENETRATING RADAR SYSTEM

## SUMMARY

In recent years, the processing speed and processing capacity of images have become increasingly important in ground-penetrating radar technologies. Mobile radars developed for the detection of buried objects are of great importance for use in military and civilian life. For this reason, studies are being carried out for the ability to perform fast operations on radars.

In order to accelerate the image processing algorithms made in ground-penetrating mobile radar systems, applications such as optimizing the software are made, but they are insufficient. In order to make up for this shortcoming, in this project, one of the large matrix multiplications written for image processing has been designed to transfer to hardware. Thus, when the matrix multiplication part, which will provide communication between the FPGA and the SoC on it, is passed, a joint design of hardware and software has been made so that the multiplication process is performed much faster than the processor through the FPGA.

Since the software and hardware will work together for the mobile ground penetrating radar system, the Verilog hardware design language (HDL) was preferred on the Zedboard and FPGA, which is a Field Programmable Gate Arrays (FPGA) with System On-Chip (SoC) in the previous works of the project. The target image and the clutter are separated from each other by running the RNMF algorithm, which is a program written in C language using the Zynq-7000 processor. It has been observed that this decomposition is faster than using only the processor since it is done using FPGA.

**DÜŞÜK MALİYETLİ,TAŞINABİLİR YERE NÜFUZ EDEN RADAR SİSTEMİ İLE YERİNDE ETKİN KARGAŞA GİDERME VE HEDEF TESPİTİ**

## ÖZET

Son yıllarda gelişen yere nüfuz eden radar teknolojilerinde görüntülerin işlenme hızı ve işlenme kapasitesi giderek önem kazanmıştır. Gömülü cisimlerin tespiti için geliştirilen mobil radarlar askeri ve sivil yaşamda kullanım için büyük bir önem taşımaktadır. Bu sebeple radarlar üzerinde hızlı işlem yapabilme kabiliyeti için çalışmalar yapılmaktadir.

Yere nüfuz eden mobil radar sistemlerinde yapılan görüntü işleme algoritmalarını hızlandırabilmek için yazılımların optimize edilmesi gibi uygulamalar yapılmakta ancak yetersiz kalmaktadır. Bu eksikliği giderebilmek için bu projede görüntü işleme için yazılmış olan büyük matris çarpımlarından birini donanıma aktarma tasarımı yapılmıştır. Böylece donanım ve üzerinde bulunan işlemci arasında haberleşme sağlanacak matris çarpım kısmına geçildiğinde çarpma işlemi Alan Programlanabilir Kapı Dizileri (FPGA) aracılığıyla işlemciden çok daha hızlı bir şekilde gerçekleştirilmesi için donanım ve yazılım ortak bir tasarım yapılmıştır.

Mobil yere nüfuz eden radar sistemi için yazılım ve donanım ortak çalışacağından dolayı projenin daha önceki çalışmalarında üzerinde System On-Chip (SoC) bulunan bir  Alan Programlanabilir Kapı Dizileri (FPGA) olan Zedboard ve FPGA üzerinde Verilog donanım tasarlama dili (HDL) tercih edilmiştir. Zynq-7000 işlemcisi kullanılarak C dilinde yazılmış bir program olan RNMF algoritması çalıştırılarak hedef görüntü ve clutter birbirinden ayrıştırılmıştır. Bu ayrıştırmanın FPGA kullanılarak yapıldığından sadece işlemci kullanılarak yapılana göre daha hızlı olduğu gözlemlenmiştir.

# 1. INTRODUCTION

## 1.1 About the Project

This report is the report of the senior design project named “FPGA Implementation for On-Site Target Detection with a Low Cost and Portable Ground Penetrating Radar System” and describes the work done within the scope of the senior design project. The project, which is within the scope of the TUBUTAK 1001 Supporting Scientific and Technological Research Projets, covers the on-site detection of buried objects with the ground penetrating radar system. However, this graduation project includes the part of the TUBITAK project, where the radar data read from buried objects is simultaneously read, transferred to the ZedBoard development board and processed on the processor to obtain clutter-free data.

The first part of this report includes the Vivado Program used throughout the project, the Vitis IDE where C codes are written and compiled, the ZedBoard which is the development board used, the Zynq-7000 SOC, and the tutorial on creating projects, developing and testing C code.

The second part of this report deals generally with reading data from ground penetrating radar. In this context, UART serial communication protocol, accessing and configuring the UART protocol on ZedBoard, testing the simple Hello World application that tests the card is working, tried and unsuccessful applications to read data from the radar with the UART serial communication protocol, and finally, successful with the UART serial communication protocol from the radar. The application that enables data to be read will be explained.

The third part of this report covers the removal of clutter from the data by processing the data read from the ground penetrating radar with the RNMF algorithm. In this section, the RNMF algorithm, GPR image, literature review on previous studies in these areas and the execution of the C code of the RNMF algorithm written in the Vitis interface on the block design designed in the Vivado program will be explained.

The fourth part of this report is generally about the hardware implementation of the software implemented RNMF algorithm. In this section, Vivado CUSTOM IP design, block diagram used in the project and Vitis code of the project will be explained.
The final section of this report deals with realistic constraints, conclusions and recommendations. In this section, the application areas of the study, realistic design constraints, the cost of the project, the standards for which the project is suitable, the social, environmental and economic impacts of the project, health and safety risks, the implications of the project and suggestions for the future will be explained.

## 1.2 Purpose of Project

Ground penetrating radar system is a widely used method for finding buried non-metallic objects. Existing and modeled methods for detecting and classifying buried objects require data from multiple targets. In order to detect the buried target object, it is necessary to separate the clutter in the radar image from the target object.

This project covers the first phase of the TUBITAK project, which is being studied for the detection of buried objects, and the application of clearing up the confusion. With this graduation project aimed to:

- simultaneous reading of the data from the radar in the correct format,
- Implementation of the RNMF (Robust Nonnegative Matrix Factorization) algorithm previously performed on Matlab in the C language and Vitis interface of the read data
- Accelerating the project by hardware implementation of the implemented RNMF algorithm

## 1.3 Project Steps

In this section, the operations carried out during the project will be explained.

First, a literature search was conducted on the RNMF algorithm and GPR image to be used in the project. Then, a test code was written to prove that the ZedBoard development board works correctly. With the test code, the sentences "Hello World" and "Successfully ran Hello World application" were observed from the terminal, it was ensured that the card was working correctly. In the first stage of data acquisition, it was requested to take the data online. However, because the ZedBoard development board works with the Linux operating system, but the computer we use has the Windows operating system, positive results could not be obtained. Afterwards, it was decided by our advisor to transfer the data offline. For this, the working logic of the UART serial communication protocol was investigated. Block design was created in Vivado for data transfer. Various block designs and C codes were tried until the data was read correctly. These applications are described in detail in Chapter 3. The Zynq-7000 processor was added to the created design. At this stage, based on the architectural design of the Zynq-7000 chip, it has been learned that the card can communicate with the UART protocol directly over the Multi Input-Output (MIO) ports. Then it is configured so that the UART protocol can work in the Vitis interface. The data in the text file has been read successfully with the UART protocol. However, since the data from the radar is in exponential form, it is not in a format suitable for processing. Since these data are read from the text file in string form, operations related to clutter removal could not be performed. Accordingly, the data format needs to be regulated in order to process the data. Since the data is in double format in the RNMF code, the data received as strings are converted to double type. However, since the function used at this stage limits the number of data received, the data is converted to float data type. This section has been completed by taking the data transfer in the correct format. After the data is received in the correct format, the RNMF algorithm reads the data from the text file via the terminal, not as a constant, in the Vitis interface. Since the most repeated operations of this algorithm are matrix multiplication and subtraction, this part is implemented on hardware. Thus, the operation of the code is shortened in time. In the hardware part, IP Floating Point Integrator has been added, which performs multiplication and subtraction of floating-points. These IPs were instantiated and the main code was written. After the testbench code was written to test the system, it was observed that the design worked correctly in the simulation. Then the custom IP is packed. A block design consisting of Zynq-7000 and custom IP was created. After the design was synthesized, implemented and the bitstream file was produced, exported hardware. Next, a new platform is created by importing the hardware file into the Vitis interface. The main code is implemented by calling the hardware-customized registers within the appropriate functions in the header file.

# 2. BASIC INFORMATION AND CONCEPTS

Field-Programmable Gate Arrays (FPGAs) are pre-fabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system [1]. A field-programmable gate array (FPGA) is an integrated circuit that can be programmed or reprogrammed to the required functionality or application after manufacturing. They are formed by a two-dimensional array of programmable logic cells and managed switches. Logic cells can be configured to implement a function and with connections between programmable keys and logic cells can be established to make new configurations. Digital hardware is implemented by programming logic cells and switches in this way. Important characteristics of field-programmable gate arrays include lower complexity, higher speed, volume designs and programmable functions. After the circuit is designed and synthesized using hardware description languages such as Verilog, Very High-Speed Integrated Circuit Hardware Description Language (VHDL), the data string containing the desired logic cell and switch configuration is embedded in the FPGA with the help of a cable. They provide a number of compelling advantages over fixed-function Application Specific Integrated Circuit (ASIC) technologies such as standard cells [2]. It is cheaper to produce than ASIC structures and takes less time to prepare circuits. In addition, the feature of being programmable again and again gives the opportunity to make new additions for the developers, and also provides an opportunity to fix the errors. FPGAs consist of an array of programmable logic blocks, including general logic, memory and multiplier blocks, digital signal processing blocks, of potentially different forms, surrounded by a programmable routing fabric that enables programmable interconnection of blocks. In FPGA, the "programmable" concept means an ability to program a feature into the chip after completion of silicon manufacturing. This customization is made possible by the programming technology, which is a method that can cause a change in the behavior of the pre-fabricated chip after fabrication, in the “field,” where system users create designs. Thus, chips produced in a single type can be programmed and used for many different purposes.

## 2.1 Xilinx Vivado Environment General Information

Xilinx Vivado Environment is an interface software used to program FPGAs developed by Xilinx company [3]. This package, which includes many programs that enable to make block-level design, facilitate the packaging of the designed hardware, design hardware over Matlab, turn the code written according to a certain rule with 11 the C language into hardware, this package has managed to become a great solution in FPGA design by eliminating the errors and accelerating as the updates arrive [4]. Thus, Xilinx has provided great convenience to designers by collecting solutions from many areas with Vivado in one package. Vivado's 2020.2 version, was used in this project with the licensed by Istanbul Technical University (ITU). Since the design will be run with a processor in this project to run the RNMF algorithm, the Vitis tool has been installed with Vivado.

The Verilog language is a language used for designing a digital system like a microprocessor or a flip-flop. It supports a design at many levels of abstraction like: behavioral level,register-transfer level, gate level. It provides the digital system designer with the ability to define a digital system at a wide range of abstraction levels, while at the same time providing access to computer-aided design software to help in the design process at these levels. Structurally similar to the C language will be preferred in digital system design [5].

## 2.2 Vitis IDE General Information

This research work employs a Xilinx ZedBoard Xynq-7000 SoC series FPGA, which can enable to combine embedded software programmability of an existing ARM processor with programmable FPGA. In this FPGA has a dual core ARM Cortex A9 processor which is the main material to write a software program to the board. FPGAs programmable logic parts consists of several Block Random Access Memories (BRAMs), Digital Signal Processing (DSP) blocks, programmable I/O pins, configurable logic blocks, transceivers and Analog to Digital converters, AXI interconnects that connecting FPGA and processor [9]. Xilinx Vitis environment is an software interface used to program SoCs on the FPGA with C and/or C++ language. Vitis IDE is a complete set of graphical and command-line developer tools that include the Vitis compilers, analyzers, and debuggers to build applications, analyze performance bottlenecks, and debug accelerated algorithms, developed in C, C++, or OpenCL™ APIs. Vitis is for writing software to run in an FPGA and is the combination of a couple of different previous Xilinx tools, including what was Xilinx SDK, Vivado High-Level Synthesis (HLS), and SDSoC. The functionality of each of these is now merged together under Vitis. Writing a C/C++ code to run on a processor in a design which is created in Vivado. This code ends up being partially used to configure and control elements of the hardware design – it's easier to rebuild, tweak, and debug on the Vitis IDE than the hardware portion is. Vitis is used to write C/C++ codes on IPs on a previously created block design in Vivado and perform operations on them. It is a platform that takes some I/O pins built-in and accelerates certain data processing functions through software by placing them in hardware with software languages such as C/C++.

## 2.3 ZedBoard Development Kit General Information

The ZedBaord Development Kit is a low cost, complete, ready to use digital circuit development platform based on the Xilinx Zynq-7000 all programmable SoCs XC7Z020-CLG484 tightly coupled dual core ARM Cortex A9 processors. [6] Target

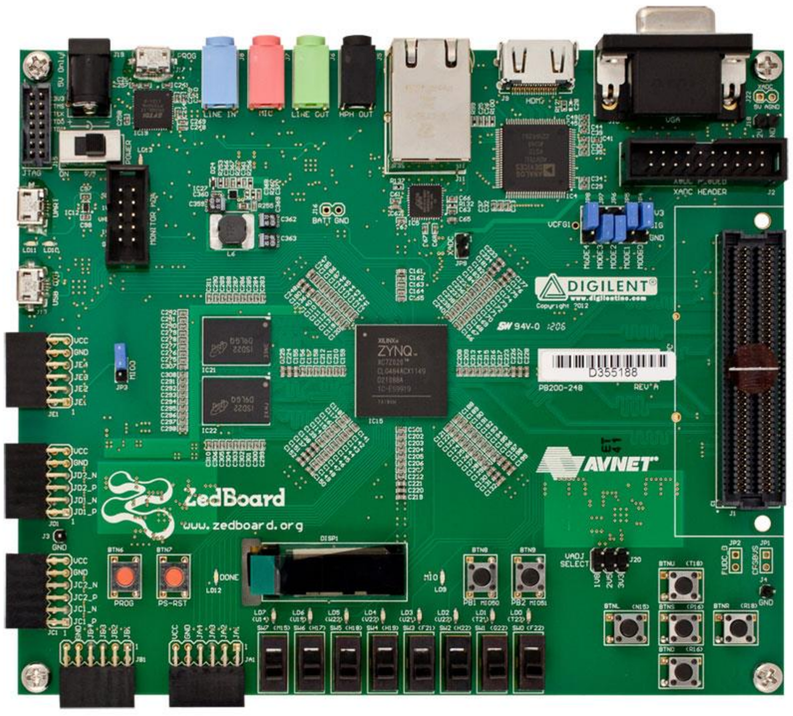


**Figure 2.1:** ZedBaord Development Kit

applications include video processing, software acceleration and general Zynq-7000 prototyping. Zedboard Development Kit is optimized for high performance logic and offers more capacity, higher performance and more DSP blocks than earlier designs. The apperance of the Zedboard Zynq-7000 ARM/FPGA SoC Development board can be seen in Figure 2.1 [6]

The features of the Zedboard device can be listed as follows:

- Xilinx Zynq-7000 all programmable SoC XC7Z020-CLG484
- Dual-core ARM Cortex™-A9 processor
- DDR3 512 MB
- Quad-SPI Flash 256 MB
- On-board USB-JTAG Programming
- Ethernet 10/100/1G
- USB OTG 2.0 and USB-UART
- Oscillator 33.333 MHz (PS), 100 MHz (PL)
- 128x32 OLED Display
- 12V 5A AC/DC regulator
- Logic Level 3.3V
- 85k logic cells
- Around 1.3 million ASIC gates
- 53.200 look-up tables (LUT)
- 106.400 flip-flops
- 560 kB of BRAM organized to 140 units, each containing 2048 by 18-bit storage
- 220 DSP slices (Multiplier-Accumulator) organized to 18 x 25
- 276 GMACs
- USB-UART Bridge
- 8 user LEDs
- 8 user Switches
- SD Card connector
- Digilent USB-JTAG port for FPGA programming and communication

## 2.4 Zynq-7000 All Programmable System on Chip

System on Chip (SoC) is an integrated electronics circuits which is made up with one base layer contains all peripherals, inputs/outputs, pins, analog/digital converters etc. at the substrate level. SoC is a hardware platform for different modules so they can with each other effectively and efficiently [7]. SoC includes almost all electronics and computer architecture system in a one base layer. Depending on the design, functions such as signal processing, wireless communication, artificial intelligence can be implemented in a system reduced to a chip size.

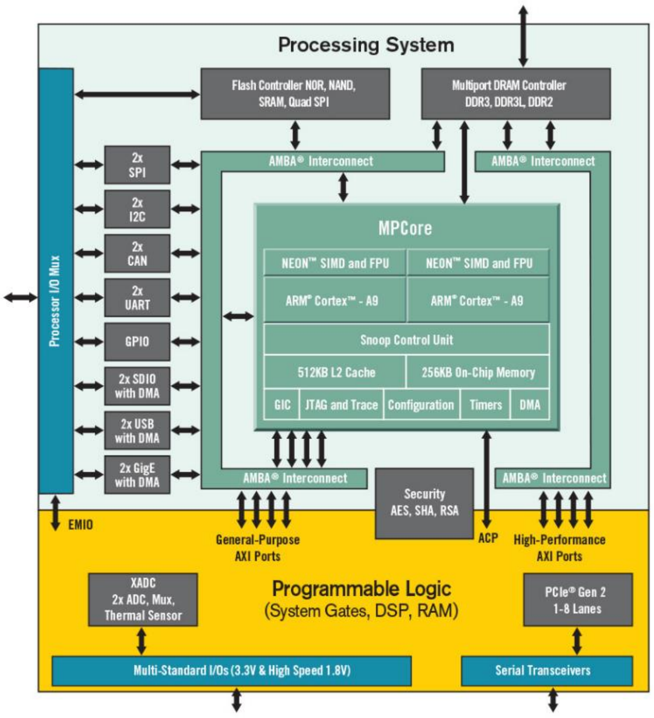


**Figure 2.2:** Zynq-7000 All Programmable SOC

The features of the Zynq-7000 SoC device can be listed as follows:

- Zynq-7000 devices are equipped with dual-core ARM Cortex-A9 processors integrated with 28nm Artix-7 or Kintex®-7 based programmable logic for excellent performance-per-watt and maximum design flexibility.
- Up to 6.6M logic cells
- Offered with transceivers ranging from 6.25Gb/s to 12.5Gb/s

- Zynq-7000 devices enable highly differentiated designs for a wide range of embedded applications including multi-camera drivers assistance systems and 4K2K Ultra-HDTV.
- MCU, FPGA architecture
- Dual ARM® Cortex®-A9 MPCore™ with CoreSight™ Core Processor
- 256 KB ram size
- DMA peripherals
- CANbus, EBI/EMI, Ethernet, I²C, MMC/SD/SDIO, SPI, UART/USART, USB OTG connections
- 766 MHz speed
- Artix™-7 FPGA, 85K Logic Cells
- 130 I/O Pins [13]

The apperance of the Zynq-7000 ARM/FPGA SoC can be seen in Figure 2.2 [8]

## 2.5 Vivado Tutorial

In this section, the stages of creating a project in Vivado, transitioning to Vitis Integrated Development Environment (IDE) environment, creating a project in Vitis Integrated Development Environment will be explained.

Firstly, the Vivado program is opened and the "Create a New Project" option is clicked to create a new project.

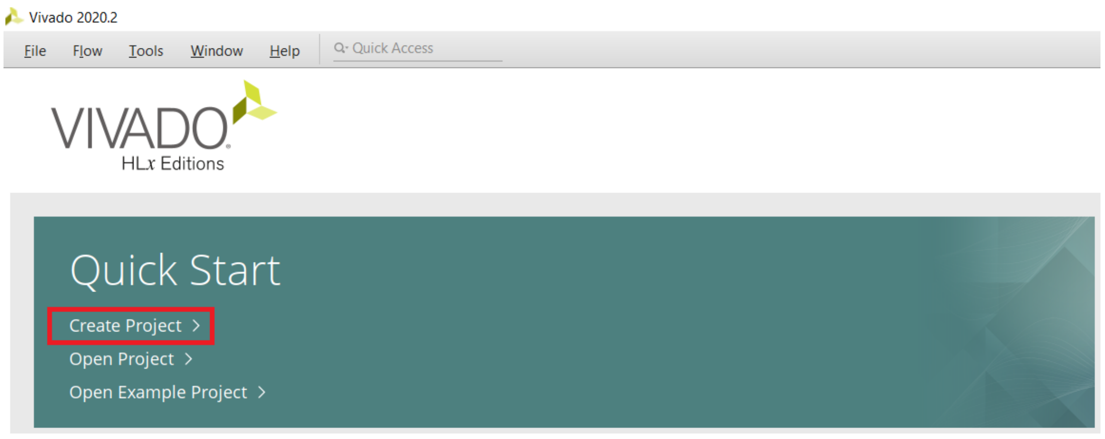
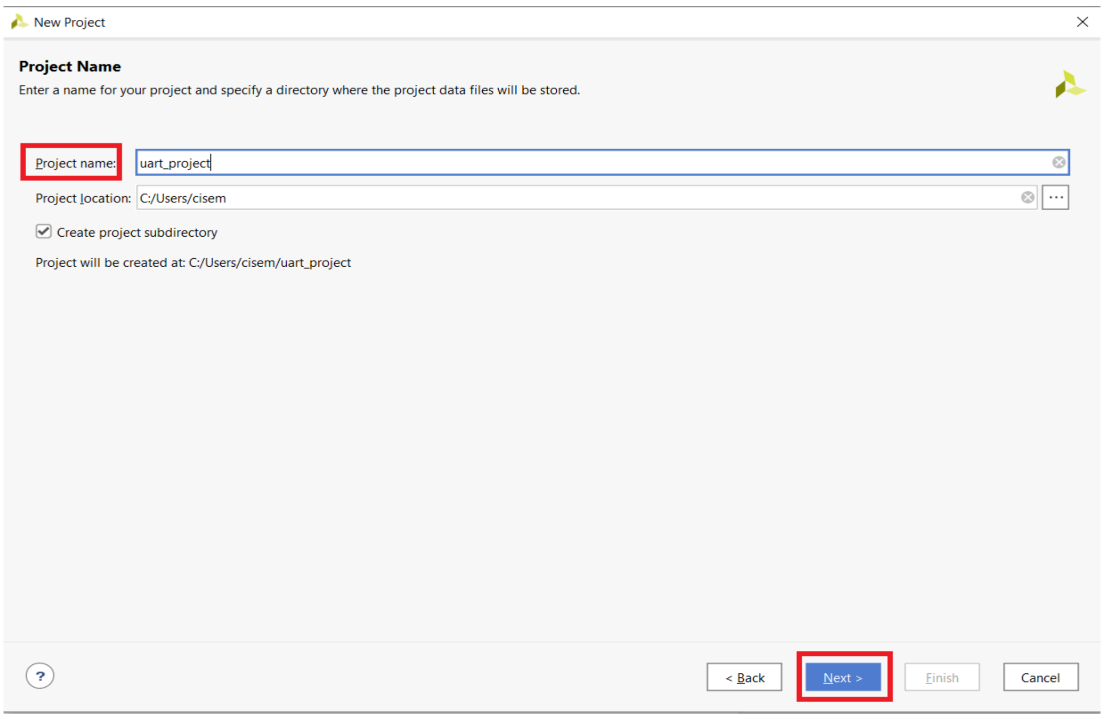


**Figure 2.3:** Creating a New Project

The new project is given a name and saved under the C folder on the computer and in a folder with a user name that does not contain Turkish characters.



**Project Name**

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: uart_project

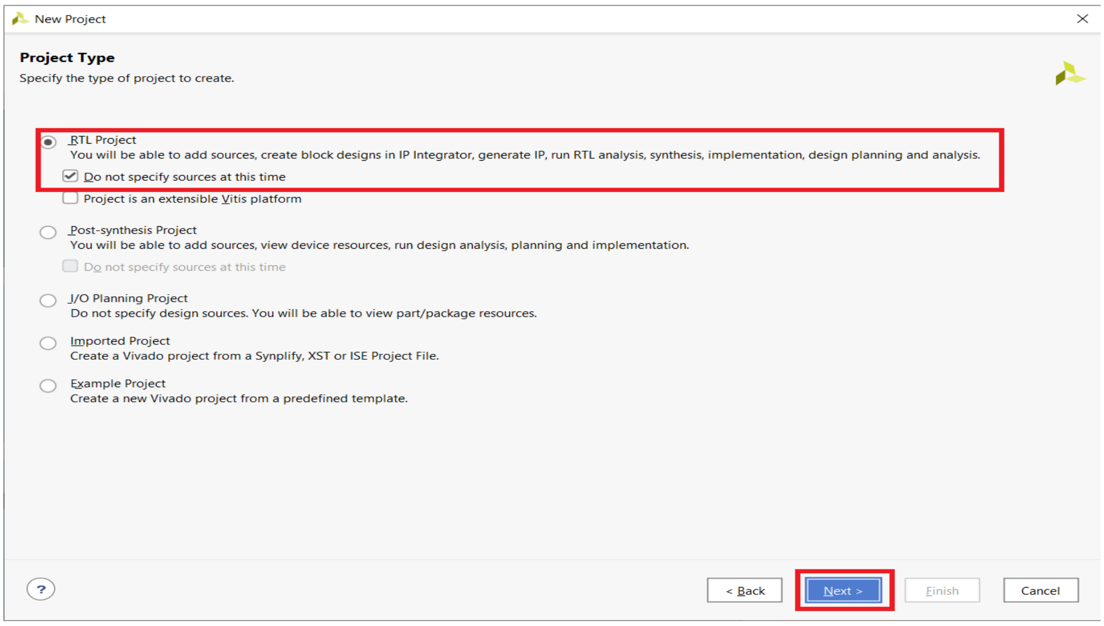
Project location: C:/Users/cisem

☑ Create project subdirectory

Project will be created at: C:/Users/cisem/uart_project

**Figure 2.4:** Naming the Project

The project type is selected as it does not contain the Register Transfer Level (RTL) project and resources.


New Project

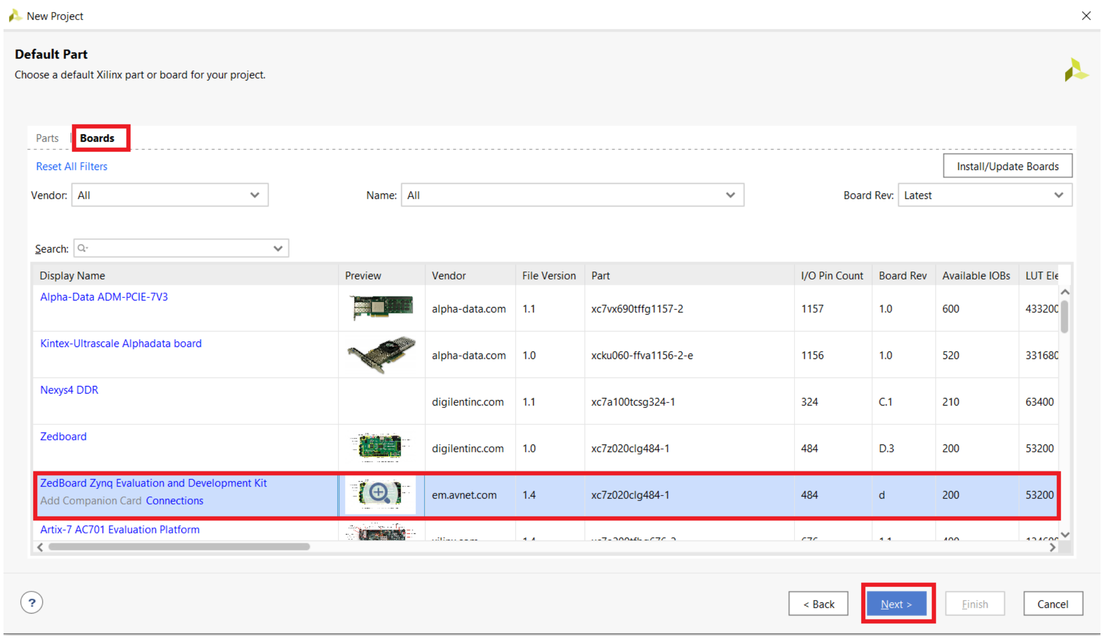
**Project Type**

Specify the type of project to create.

- [✓ RTL Project]
  You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
  - [✓ Do not specify sources at this time]
  - Project is an extensible Vitis platform
- Post-synthesis Project
  You will be able to add sources, view device resources, run design analysis, planning and implementation.
  - Do not specify sources at this time
- I/O Planning Project
  Do not specify design sources. You will be able to view part/package resources.
- Imported Project
  Create a Vivado project from a Synplify, XST or ISE Project File.
- Example Project
  Create a new Vivado project from a predefined template.

**Figure 2.5:** Determining the Project Type

Since the project will be carried out on the Zedboard Development card, the card to be worked on has been selected as the ZedBoard Zynq Evaluation and Development Kit from the Boards section at this stage. Then, the project opening process was completed by clicking the Finish button on the page that opened [9].


New Project

**Default Part**

Choose a default Xilinx part or board for your project.

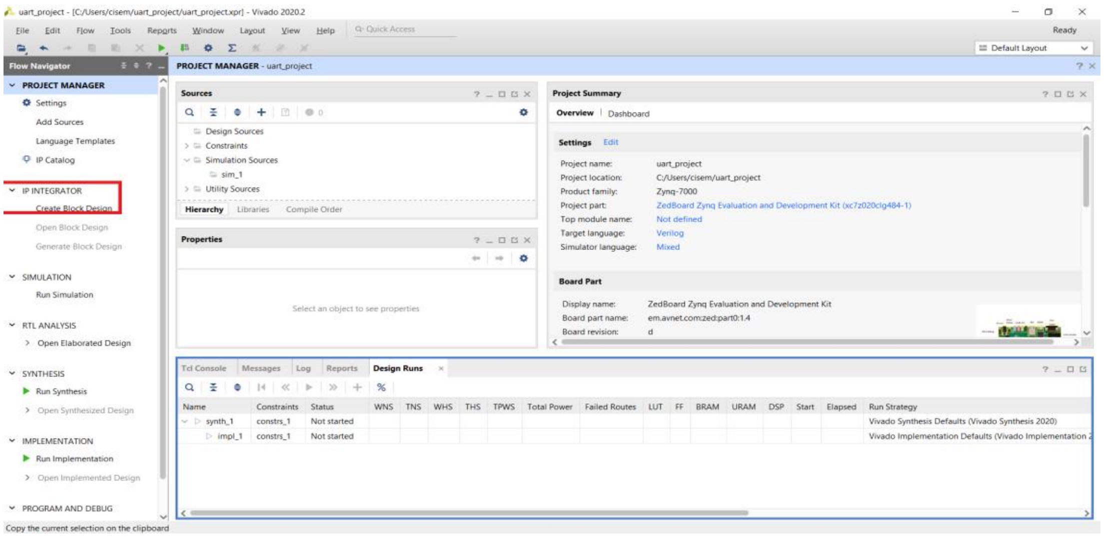
Vendor: All    Name: All    Board Rev: Latest

Search:

| Display Name | Preview | Vendor | File Version | Part | I/O Pin Count | Board Rev | Available IOBs | LUT Ele |
|---|---|---|---|---|---|---|---|---|
| Alpha-Data ADM-PCIE-7V3 | | alpha-data.com | 1.1 | xc7vx690tffg1157-2 | 1157 | 1.0 | 600 | 433200 |
| Kintex-Ultrascale Alphadata board | | alpha-data.com | 1.0 | xcku060-ffva1156-2-e | 1156 | 1.0 | 520 | 331680 |
| Nexys4 DDR | | digilentinc.com | 1.1 | xc7a100tcsg324-1 | 324 | C.1 | 210 | 63400 |
| Zedboard | | digilentinc.com | 1.0 | xc7z020clg484-1 | 484 | D.3 | 200 | 53200 |
| ZedBoard Zynq Evaluation and Development Kit<br>Add Companion Card Connections | | em.avnet.com | 1.4 | xc7z020clg484-1 | 484 | d | 200 | 53200 |
| Artix-7 AC701 Evaluation Platform | | | | | | | | |

**Figure 2.6:** Selecting the Card to Use

Since the block design is being worked on in the created project, a new block design is opened from the Intellectual Property (IP) Integrator section.


uart_project - [C:/Users/cisem/uart_project/uart_project.xpr] - Vivado 2020.2

PROJECT MANAGER - uart_project

Project Summary

Settings

| | |
|---|---|
| Project name: | uart_project |
| Project location: | C:/Users/cisem/uart_project |
| Product family: | Zynq-7000 |
| Project part: | ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1) |
| Top module name: | Not defined |
| Target language: | Verilog |
| Simulator language: | Mixed |

Board Part

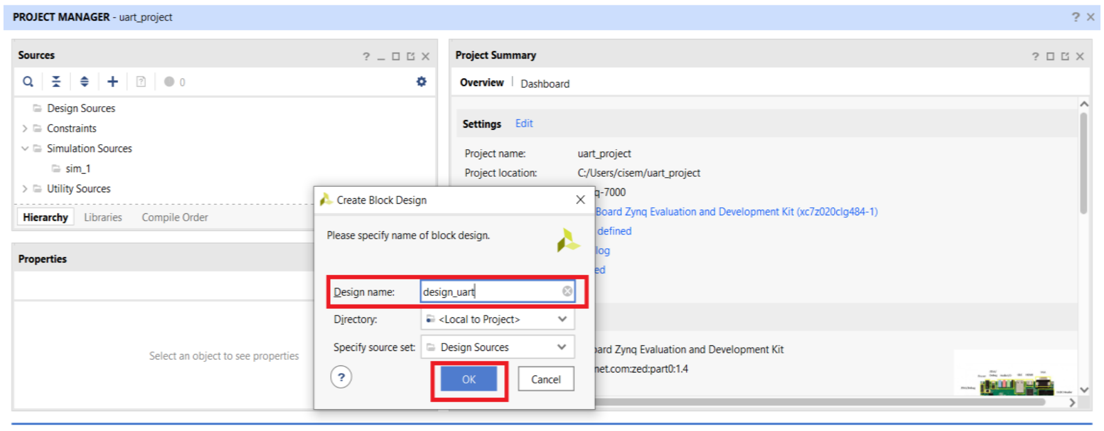
| | |
|---|---|
| Display name: | ZedBoard Zynq Evaluation and Development Kit |
| Board part name: | em.avnet.com:zed:part0:1.4 |
| Board revision: | d |

Design Runs

| Name | Constraints | Status | Run Strategy |
|---|---|---|---|
| synth_1 | constrs_1 | Not started | Vivado Synthesis Defaults (Vivado Synthesis 2020) |
| impl_1 | constrs_1 | Not started | Vivado Implementation Defaults (Vivado Implementation 2 |

**Figure 2.7:** Creating the Block Design

For the block design, the name is given in accordance with the project.

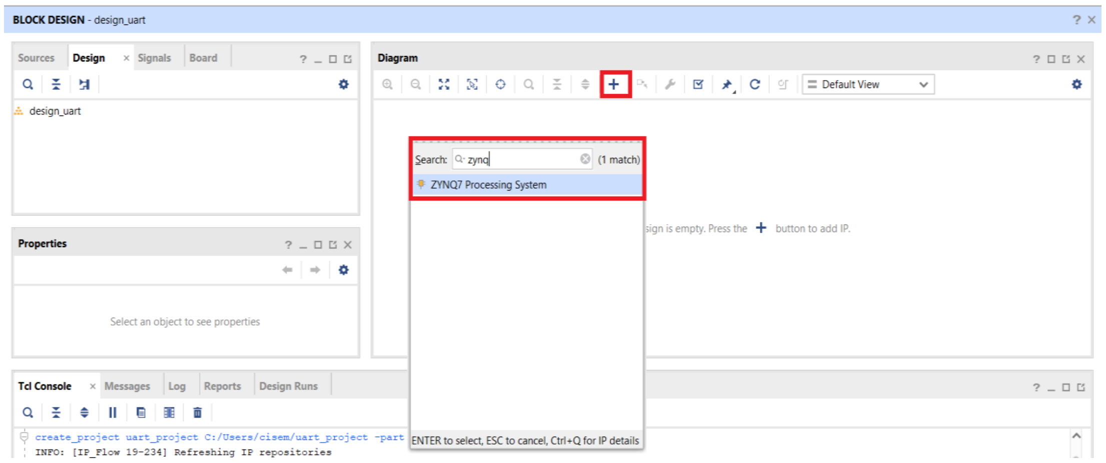

Create Block Design

Please specify name of block design.

| | |
|---|---|
| Design name: | design_uart |
| Directory: | <Local to Project> |
| Specify source set: | Design Sources |

**Figure 2.8:** Naming the Block Design

First, the processor to be worked on is added to the block design. Since the processor for this project is ZYNQ, the name of the processor is added to the design by typing the ADD IP button shown in the figure.


BLOCK DESIGN - design_uart
Sources
Design
Signals
Board
design_uart
Properties
Select an object to see properties
Diagram
Default View
Search: zynq
(1 match)
ZYNQ7 Processing System
sign is empty. Press the + button to add IP.
ENTER to select, ESC to cancel, Ctrl+Q for IP details
Tcl Console
Messages
Log
Reports
Design Runs

```
create_project uart_project C:/Users/cisem/uart_project -part
INFO: [IP_Flow 19-234] Refreshing IP repositories
```

**Figure 2.9:** Adding the Processor

When Run Block Automation is clicked, the screen shown in Figure 2.10 opens. After the settings are selected as in the figure, the connections of the processor are completed automatically.

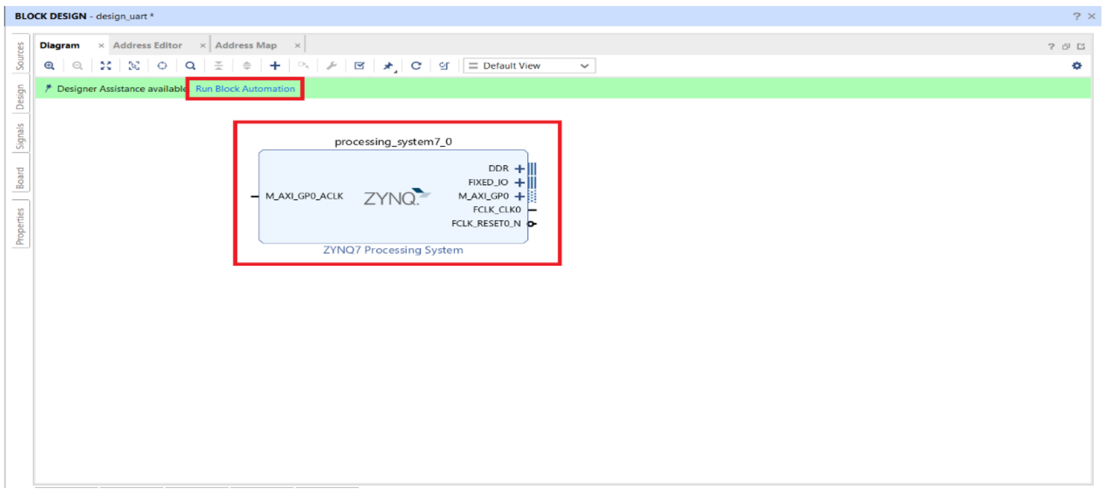
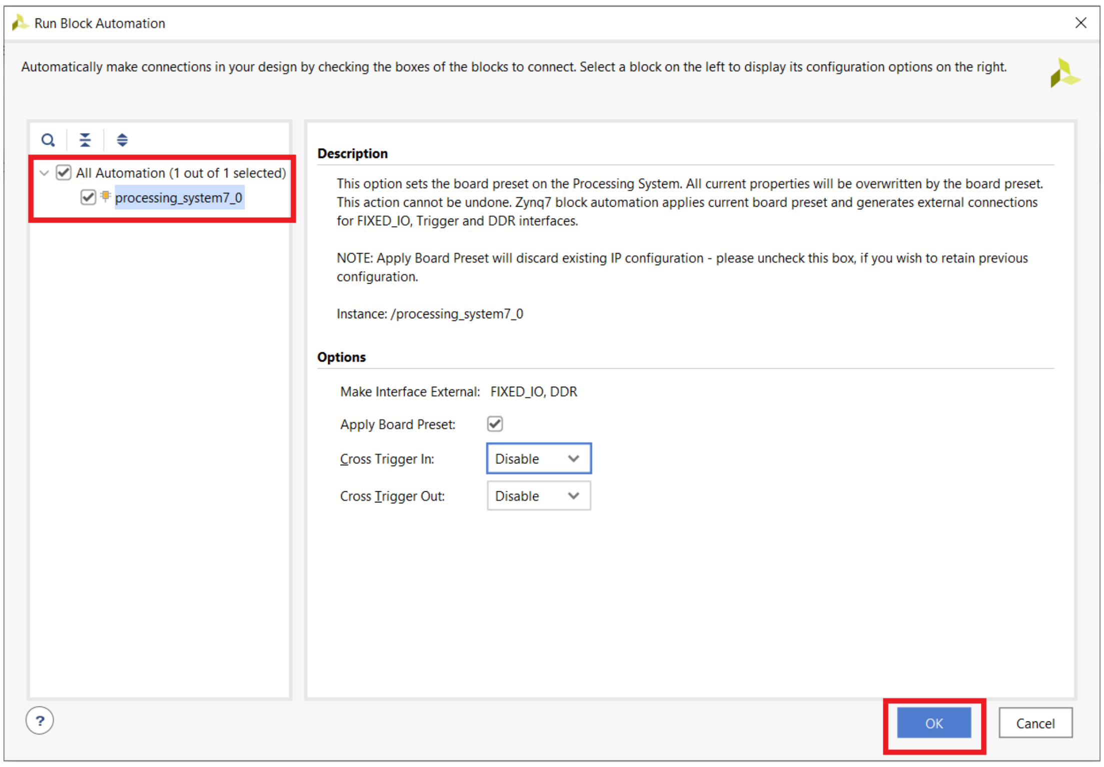


**Figure 2.10:** Making Automatic Connections of the Processor


Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 out of 1 selected)
processing_system7_0

**Description**

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

**Options**

Make Interface External: FIXED_IO, DDR
Apply Board Preset: ✓
Cross Trigger In: Disable
Cross Trigger Out: Disable

**Figure 2.11:** Customizing the Processor

After the block design is completed, the HDL Wrapper file is generated from the block diagram created by right-clicking on the "Sources" section. Thus, the project part in the Vivado interface is completed and the transition to the Vitis IDE is made.

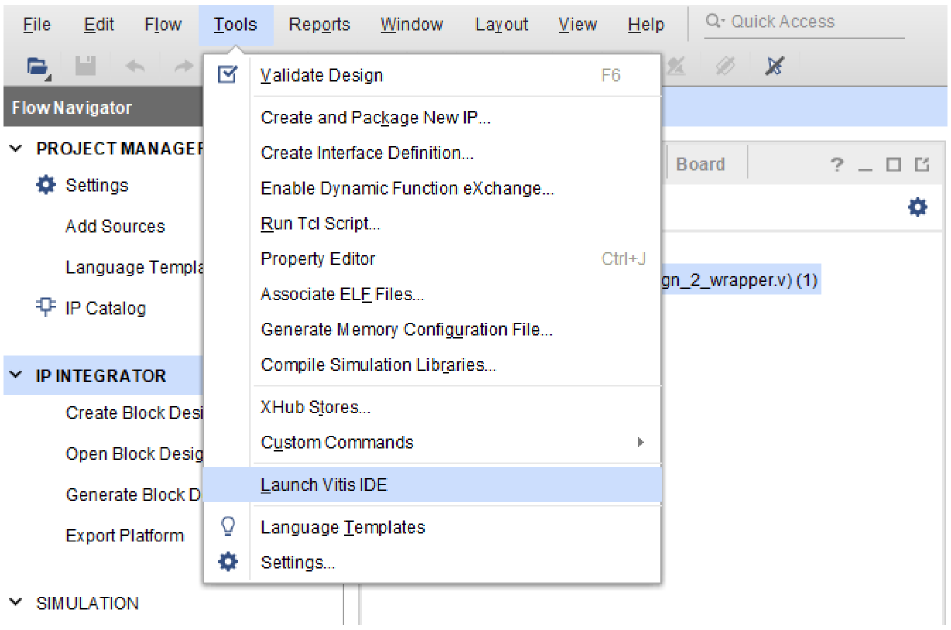


**Figure 2.12:** Transitioning to the Vitis Platform

When transitioning from Vivado to Vitis Platform, a new platform project is created to write code under the created workspace.

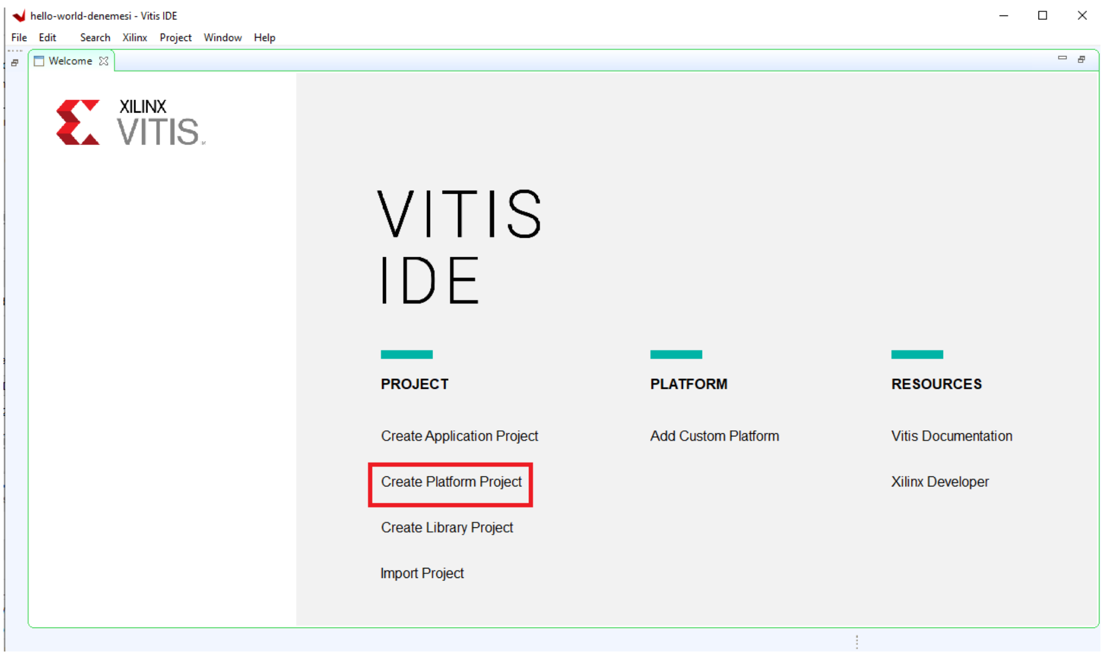
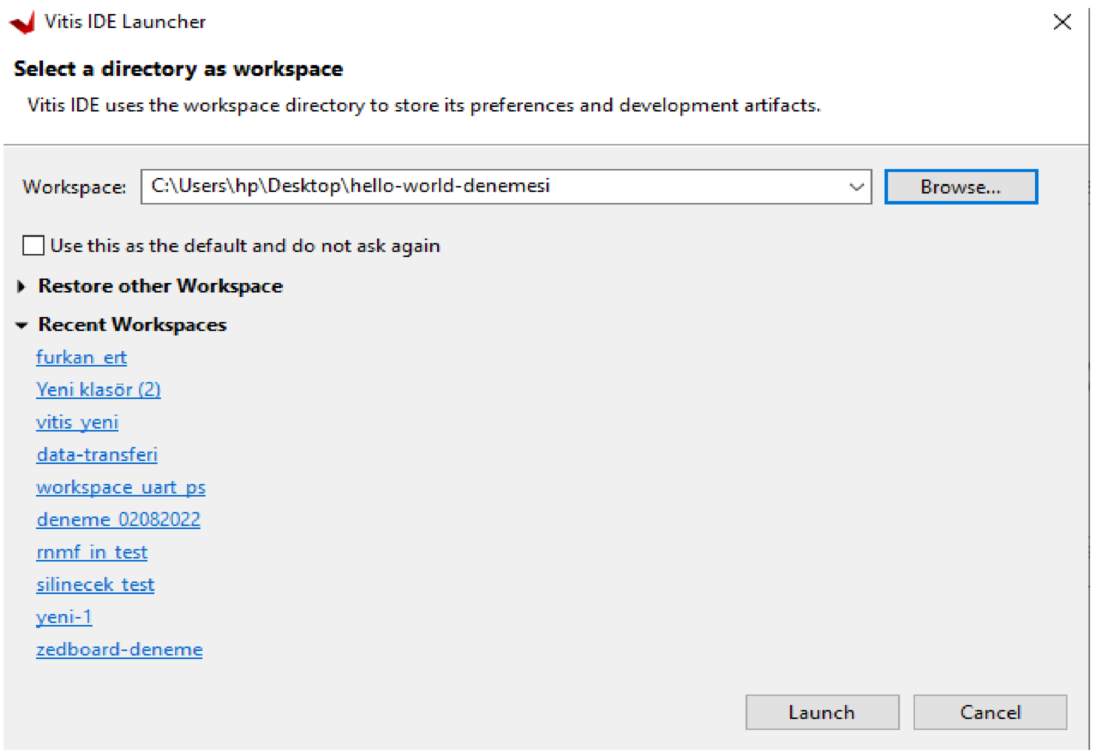


**Figure 2.13:** Creating a New Application Project

A new workspace is created and named with an appropriate name under the C folder.



Vitis IDE Launcher

**Select a directory as workspace**

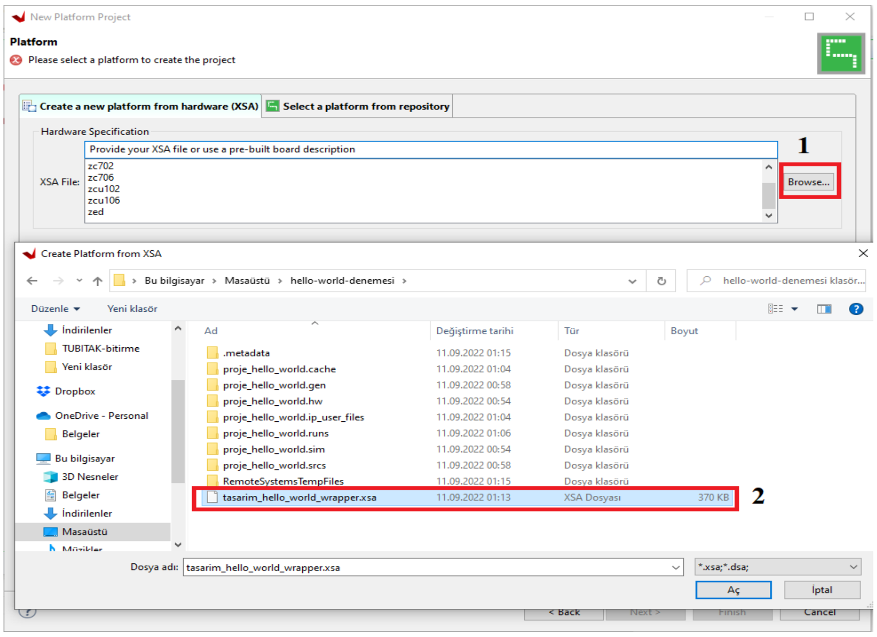
Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\hp\Desktop\hello-world-denemesi

[ ] Use this as the default and do not ask again

**Figure 2.14:** Creating a New Workspace on the Vitis Platform

The HDL Wrapper file created during the block design phase is added to the platform for the code to work properly in the block design. This file is under the folder where the block design was created before.



New Platform Project

**Platform**

Please select a platform to create the project

Create a new platform from hardware (XSA) | Select a platform from repository

Hardware Specification

XSA File: Provide your XSA file or use a pre-built board description

zc702
zc706
zcu102
zcu106
zed

1 Browse...

Create Platform from XSA

Bu bilgisayar › Masaüstü › hello-world-denemesi ›

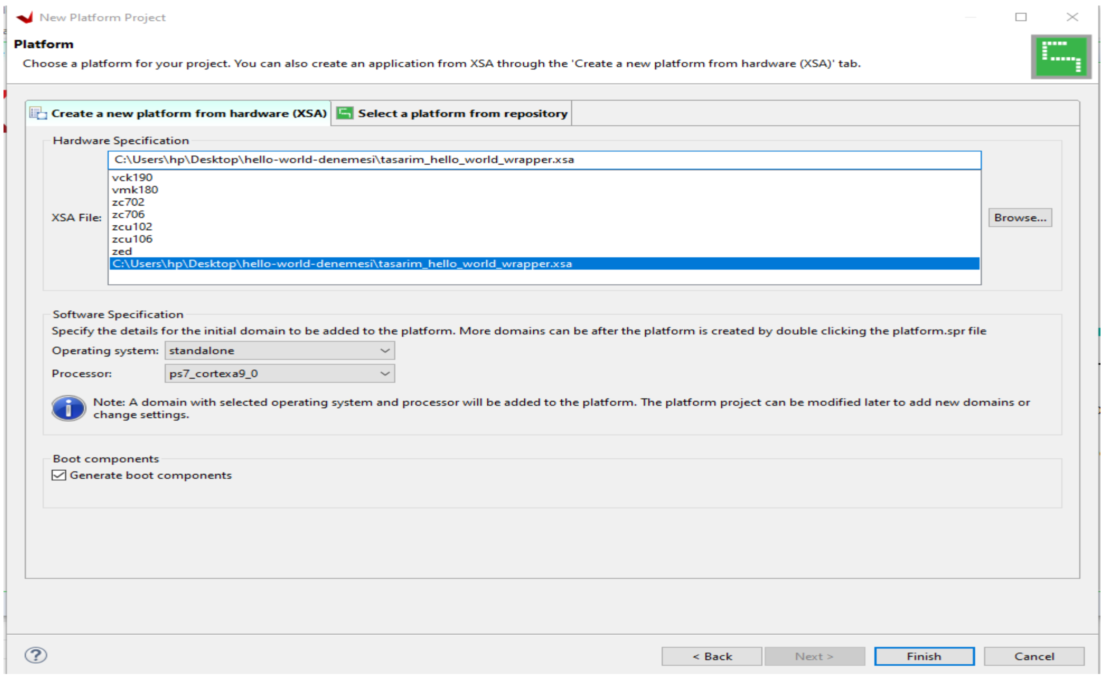
| Ad | Değiştirme tarihi | Tür | Boyut |
|---|---|---|---|
| .metadata | 11.09.2022 01:15 | Dosya klasörü | |
| proje_hello_world.cache | 11.09.2022 01:04 | Dosya klasörü | |
| proje_hello_world.gen | 11.09.2022 00:58 | Dosya klasörü | |
| proje_hello_world.hw | 11.09.2022 00:54 | Dosya klasörü | |
| proje_hello_world.ip_user_files | 11.09.2022 01:04 | Dosya klasörü | |
| proje_hello_world.runs | 11.09.2022 01:06 | Dosya klasörü | |
| proje_hello_world.sim | 11.09.2022 00:54 | Dosya klasörü | |
| proje_hello_world.srcs | 11.09.2022 00:58 | Dosya klasörü | |
| RemoteSystemsTempFiles | 11.09.2022 01:15 | Dosya klasörü | |
| tasarim_hello_world_wrapper.xsa | 11.09.2022 01:13 | XSA Dosyası | 370 KB |

2

Dosya adı: tasarim_hello_world_wrapper.xsa   *.xsa;*.dsa;

**Figure 2.15:** Embedding the Block Design in the Project

Since the data will be received and processed in an offline environment, the operating system setting is selected as standalone. In the processor part, the cortexa9_0 part continues to work without making any changes.



New Platform Project

**Platform**

Choose a platform for your project. You can also create an application from XSA through the 'Create a new platform from hardware (XSA)' tab.

Create a new platform from hardware (XSA) | Select a platform from repository

Hardware Specification

XSA File: C:\Users\hp\Desktop\hello-world-denemesi\tasarim_hello_world_wrapper.xsa

vck190
vmk180
zc702
zc706
zcu102
zcu106
zed
C:\Users\hp\Desktop\hello-world-denemesi\tasarim_hello_world_wrapper.xsa

Software Specification

Specify the details for the initial domain to be added to the platform. More domains can be after the platform is created by double clicking the platform.spr file

Operating system: standalone

Processor: ps7_cortexa9_0

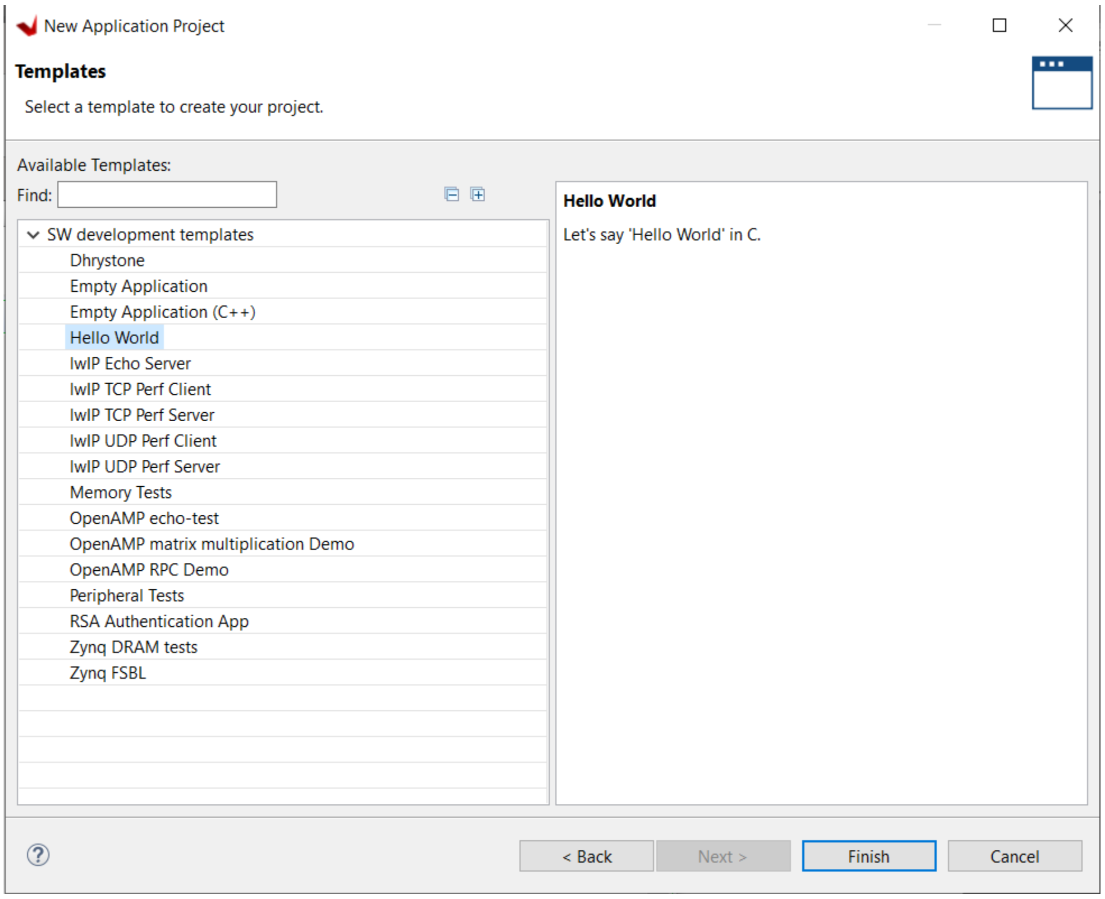
Note: A domain with selected operating system and processor will be added to the platform. The platform project can be modified later to add new domains or change settings.

Boot components

☑ Generate boot components

**Figure 2.16:** Adjusting the Block Design on Vitis

After adding the platform to work on, a draft suitable for the project is selected. This draft can be selected as an empty C draft, as well as a Hello World draft. In this project, the Hello World draft was chosen and Vitis code was written. In this draft, it is easier to implement because the platform header file is compact.



New Application Project

**Templates**

Select a template to create your project.

Available Templates:

Find:

- SW development templates
  - Dhrystone
  - Empty Application
  - Empty Application (C++)
  - Hello World
  - lwIP Echo Server
  - lwIP TCP Perf Client
  - lwIP TCP Perf Server
  - lwIP UDP Perf Client
  - lwIP UDP Perf Server
  - Memory Tests
  - OpenAMP echo-test
  - OpenAMP matrix multiplication Demo
  - OpenAMP RPC Demo
  - Peripheral Tests
  - RSA Authentication App
  - Zynq DRAM tests
  - Zynq FSBL

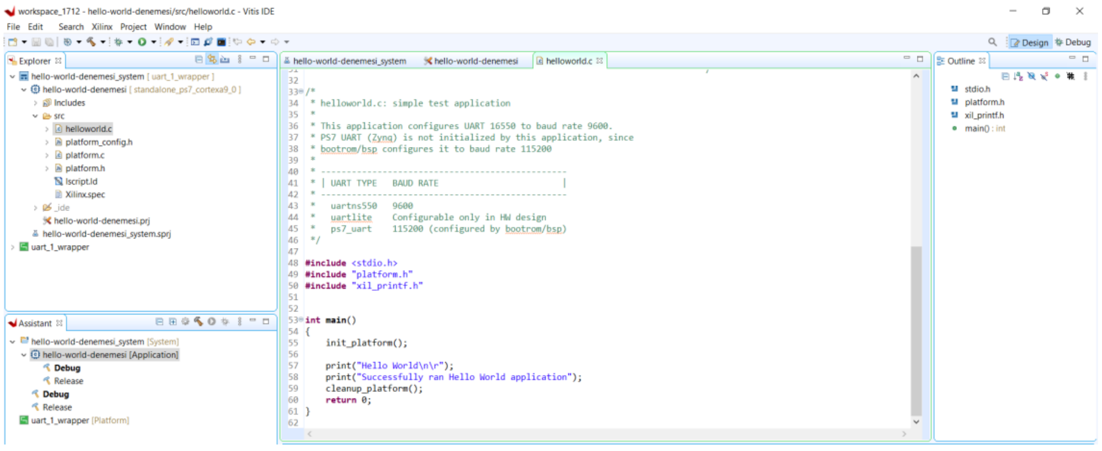
**Hello World**

Let's say 'Hello World' in C.

< Back | Next > | Finish | Cancel

**Figure 2.17:** Adding New Platform Template

By pressing the Finish button, the new project (Figure 2.18) is opened successfully. After that, a new C project is opened under the src folder and the projects are carried out.



workspace_1712 - hello-world-denemesi/src/helloworld.c - Vitis IDE

Explorer
- hello-world-denemesi_system [ uart_1_wrapper ]
  - hello-world-denemesi [ standalone_ps7_cortexa9_0 ]
    - Includes
    - src
      - helloworld.c
      - platform_config.h
      - platform.c
      - platform.h
      - lscript.ld
      - Xilinx.spec
    - _ide
    - hello-world-denemesi.prj
  - hello-world-denemesi_system.sprj
- uart_1_wrapper

Assistant
- hello-world-denemesi_system [System]
  - hello-world-denemesi [Application]
    - Debug
    - Release
  - Debug
  - Release
- uart_1_wrapper [Platform]

hello-world-denemesi_system | hello-world-denemesi | helloworld.c

```c
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

Outline
- stdio.h
- platform.h
- xil_printf.h
- main() : int

**Figure 2.18:** Adding New Platform Template

# 3. IMPLEMENTATION of APPLICATIONS on FPGA

## 3.1 UART Serial Communication Protocol

UART is a type of serial communication protocol in which data is transmitted and received one bit at a time over a single communication line or channel. It stands for Universal Asynchronous Receiver Transmitter, and it allows devices to transmit and receive data asynchronously (without a fixed clock rate). This means that the sender and receiver do not need to be precisely synchronized in order to communicate. UART is commonly used in embedded systems and in communication between computers and devices, such as printers, keyboards, and mice.

UART uses a combination of a start bit, which signals the start of a transmission, and a stop bit, which signals the end of a transmission, to enable the receiver to synchronize with the sender and accurately receive the data. UART also allows for the use of parity bits, which can be used for error detection.

High dependability and a long transmission distance are benefits of asynchronous serial communication. A UART is frequently used in data communications and control systems because it enables full-duplex communication in serial communications. Consequently, it is frequently utilized in data interchange between peripherals and processor. By adding a few more control bits and utilizing a shift register, the UART transfers data from parallel to serial on the transmitter side and back again on the reception side. The UART appears as an 8-bit write-read parallel interface on the other end [20].

To perform full-duplex data transfer with a basic UART, just two signal lines—one for receive and one for transmit—are required. To regulate the UART receive and transmit, a local clock signal that is significantly faster than the baud rate is generated using a baud rate generator. The serial signals are received at RXD by the UART receiver block, which transforms them to parallel data. Bytes are converted into serial bits using the basic frame format by the UART transmitter block, which then sends those bits across the TXD line. The data line's high logic state is present while the transmitter is not in use. A "Start Bit" is inserted to a beginning of every word that is to be communicated if the UART is activated for transmission. The start bit is used to compel the clock in the peripheral receiver to synchronize with the clock in the transmitter and to notify the peripheral receiver that such a word containing data is going to be delivered. Following the transmission of the start bit, the specific data bits of a word are. The receiver sampling at the wire roughly halfway through to the period given to each bit to identify whether it is a "1" or a "0". Each bit is broadcast for the exact same amount of time as all the previous bits. The transmitters add a parity bit that was created in the transmitter module once the complete data word has been sent. If the transmitter delivers another frame, the new word's Start bit can be transmitted as soon as the preceding word's stop bit is sent [10].

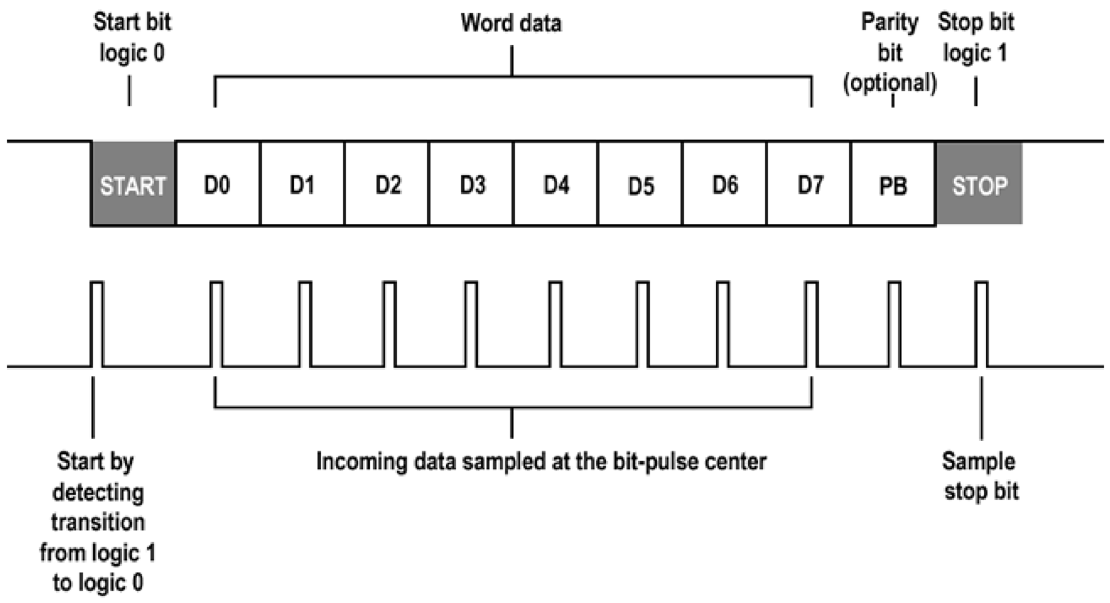


**Figure 3.1:** UART Data Frame Format [Y*]

## 3.2 Architectural Design of Zynq-7000

In this project, the data received from the ground penetrating radar must be transferred to the FPGA for processing. UART serial communication protocol is used for data transfer. Therefore, ZedBaord's UART connections must be made. In line with the researches, it has been seen that it can be done directly from the Multi Input-Output inputs of the zynq-7000 without the need for special IPs such as UART_LITE or UART_PS to communicate with the UART. The architectural design of the Zynq-7000 chip is given in Figure 3.2.

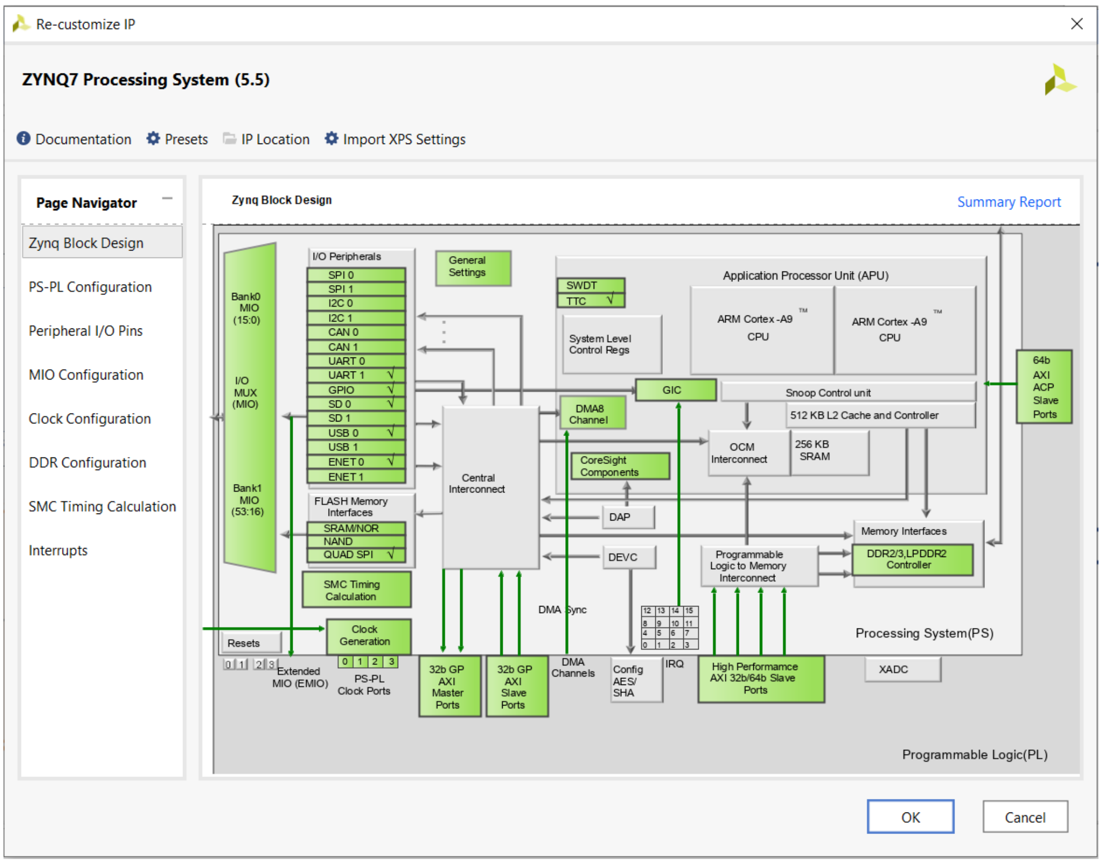
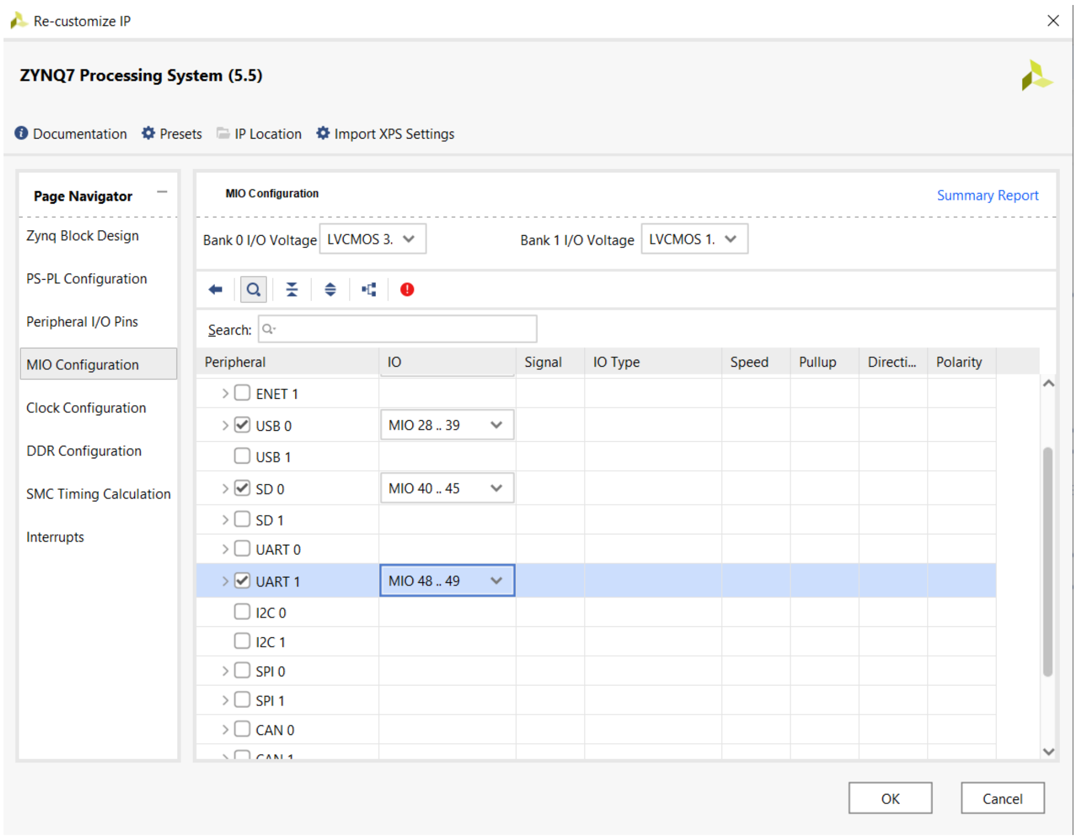


**Figure 3.2:** Architectural Design of the Zynq-7000 Chip [11]

Therefore, Multi Input-Outputs are used to communicate with UART. Zedboard's UART/USB port is accessed via MI48 and MIO_49 pins. Thus, MIO48 and MIO49, which are related to the UART, are activated.



ZYNQ7 Processing System (5.5)

MIO Configuration

Bank 0 I/O Voltage: LVCMOS 3.

Bank 1 I/O Voltage: LVCMOS 1.

| Peripheral | IO | Signal | IO Type | Speed | Pullup | Directi... | Polarity |
|---|---|---|---|---|---|---|---|
| ☐ ENET 1 | | | | | | | |
| ✓ USB 0 | MIO 28 .. 39 | | | | | | |
| ☐ USB 1 | | | | | | | |
| ✓ SD 0 | MIO 40 .. 45 | | | | | | |
| ☐ SD 1 | | | | | | | |
| ☐ UART 0 | | | | | | | |
| ✓ UART 1 | MIO 48 .. 49 | | | | | | |
| ☐ I2C 0 | | | | | | | |
| ☐ I2C 1 | | | | | | | |
| ☐ SPI 0 | | | | | | | |
| ☐ SPI 1 | | | | | | | |
| ☐ CAN 0 | | | | | | | |

**Figure 3.3:** UART Access of Zynq-7000 Chip [11]

## 3.3 Hello World Application

First, we checked whether the ZedBoard development board we used in the project is working. This basic project is one of the testing program that shows us we can access FPGA and write a C/C++ code in the ARM processor core. For this, we created a simple block design and wrote a test code. We did the creating the project in the Vitis interface and block design according to the steps given in section 2.4. Only the Zynq-7000 processor has been added to the block design. As shown in Section 3.2, the UART port of the Zynq-7000 processor was accessed from the Multi Input-Output port. Therefore, the block design given in figure 3.4 was sufficient for the test

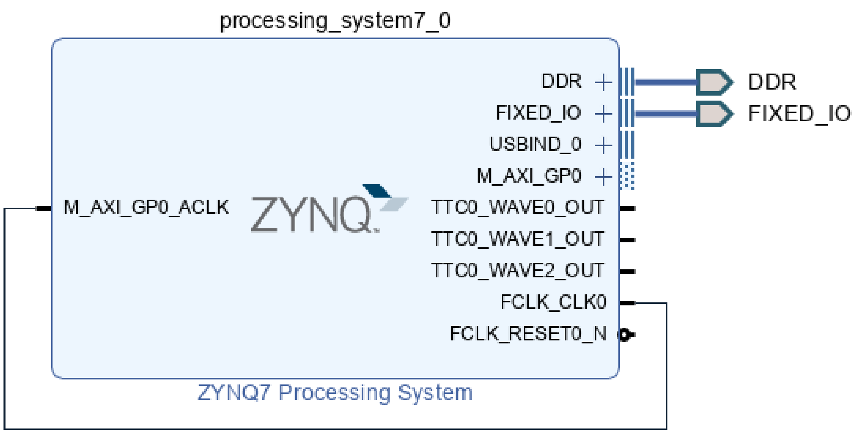
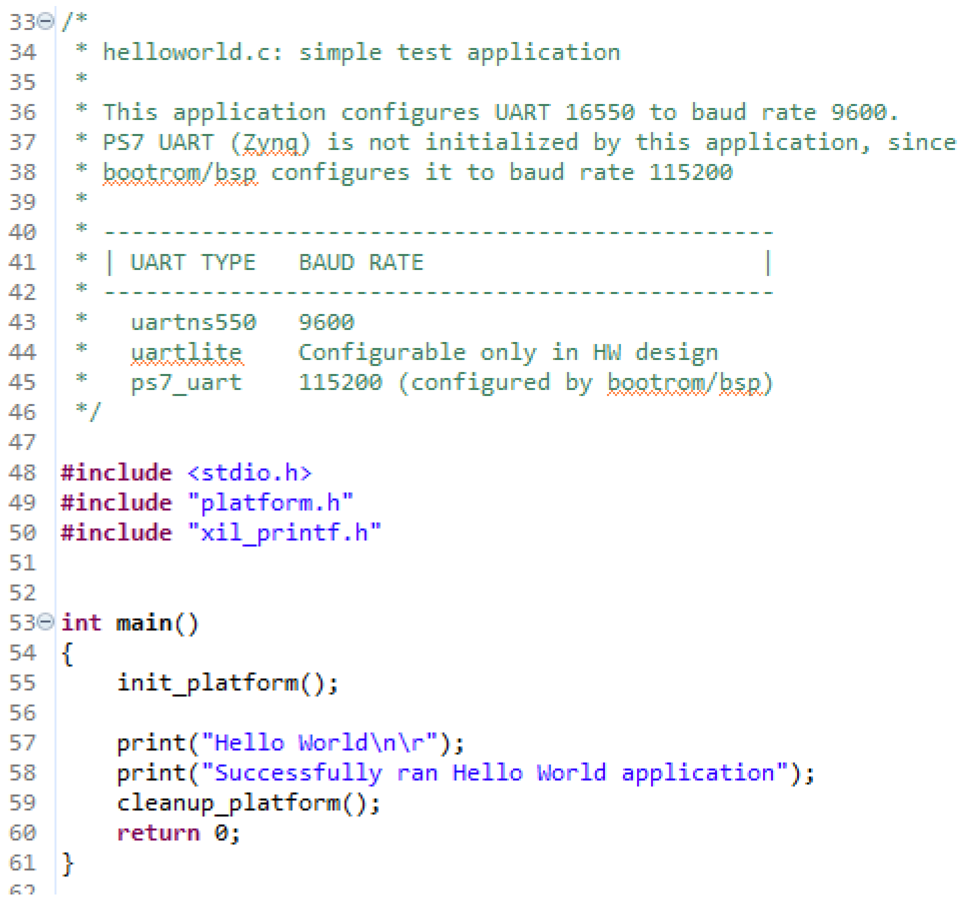


**Figure 3.4:** Hello World Test Application Block Design

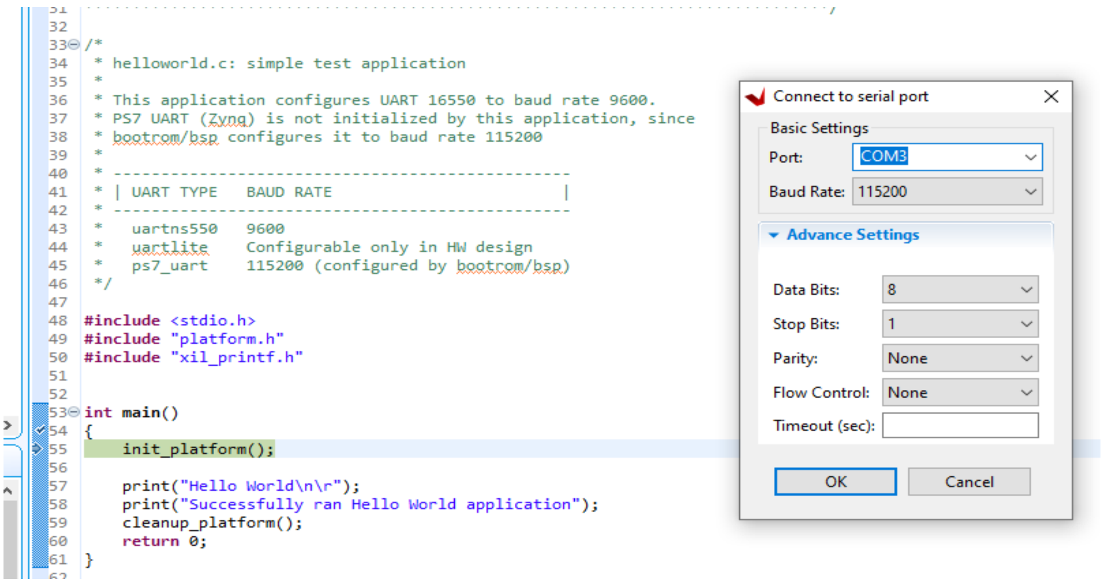
application. The designed block design has been validated by automatically. Block design was designed in Vivado interface and translated into hardware language. After the HDL Wrapper file is produced from the block diagram, synthesis, implementation and bitstream file production processes are performed respectively. Finally, the hardware platform created is exported and a file is produced in xsa format and the hardware design is completed in the Vivado program. Later, the prepared hardware platform was added to the Vitis interface, which is Vivado's software tool, in XSA format. A new workspace is created on the Vitis platform and a Hello World template project is created. The code in Figure 3.5 is the C code written for testing.



```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

**Figure 3.5:** Hello World Test Application C Code

With the #include<stdio.h> line, the general C library is included so that the general functions in the project can work. The #include“platform.h” line has been added to enable Hello World test application to work on block design. The #include“xil_printf.h” line is used in the project to write data to the terminal. In the main code part, the platform on which the application will run is initialized with the init_platform() command. Then, the expressions to be observed in the terminal are given in the print() function. The platform used is released with the cleanup_platform() command and the code is terminated.



```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

Connect to serial port

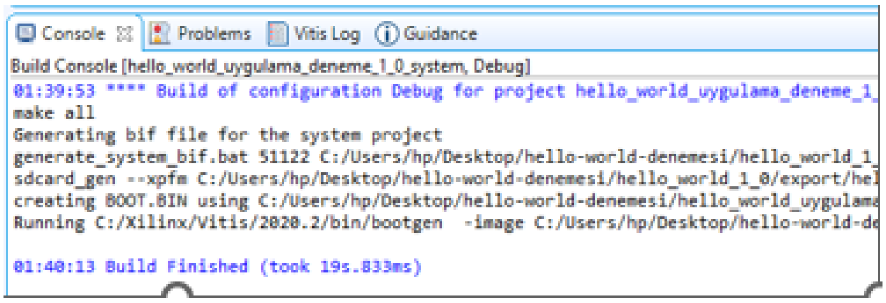
Basic Settings
Port: COM3
Baud Rate: 115200

Advance Settings
Data Bits: 8
Stop Bits: 1
Parity: None
Flow Control: None
Timeout (sec):

**Figure 3.6:** Making Hello World Project UART Communication Settings

During the communication of UART with Zedboard, the port to which it is connected on the computer is selected. Data settings are selected by default. The data to be sent

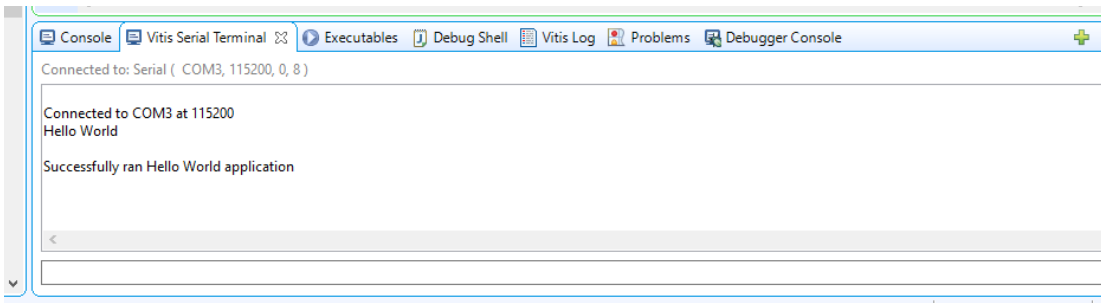
is set to 8 bits long, 1 bit set to stop bit, and no parity bit (Digilent,2017). Baud Rate is updated to 115200 in accordance with the Block diagram. The project is built and run with Run as -> Hardware.



```
Build Console [hello_world_uygulama_deneme_1_0_system, Debug]
01:39:53 **** Build of configuration Debug for project hello_world_uygulama_deneme_1_
make all
Generating bif file for the system project
generate_system_bif.bat 51122 C:/Users/hp/Desktop/hello-world-denemesi/hello_world_1_
sdcard_gen --xpfm C:/Users/hp/Desktop/hello-world-denemesi/hello_world_1_0/export/hel
creating BOOT.BIN using C:/Users/hp/Desktop/hello-world-denemesi/hello_world_uygulama
Running C:/Xilinx/Vitis/2020.2/bin/bootgen  -image C:/Users/hp/Desktop/hello-world-de

01:40:13 Build Finished (took 19s.833ms)
```

**Figure 3.7:** Build the Project

As a result, the terms "Hello World" and "Successfully ran Hello World application" were observed from the terminal as shown in Figure 3.8.



```
Connected to: Serial ( COM3, 115200, 0, 8 )

Connected to COM3 at 115200
Hello World

Successfully ran Hello World application
```

**Figure 3.8:** Obtaining the Output on the Terminal with the Test Code Applied on the Vitis Platform

Thus, the Zedboard development board used has been proven to work.

## 3.4 Failed Data Transfer Applications

To collect the data in the right format, experiments with various block designs and C scripts have been conducted. This section will discuss applications for data transmission from radar to FPGA that have been tried but failed.

### 3.4.1 AXI GPIO block diagram and C code application

First, the Zynq-7000 SoC ARM processor was added to the block diagram. Then AXI GPIO IP was added to print the data. BRAM and block memory generator are used to hold data in memory. Processor system reset, AXI Connector and AXI BRAM Controller came automatically when other IPs are added. The designed block diagram has been validated by automatically.

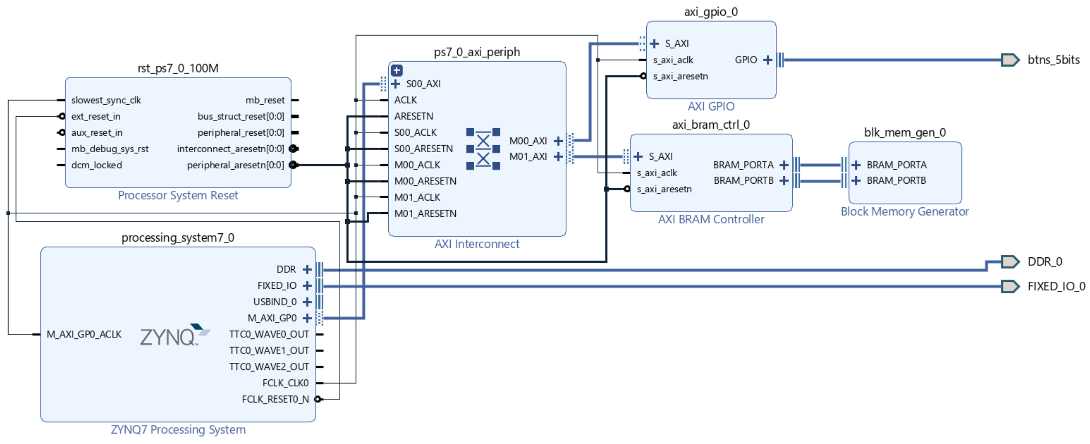


**Figure 3.9:** AXI GPIO Block Diagram for Radar Data Transfer via UART

Block diagram was designed in Vivado IP Integrator Section and translated into hardware language. Afterwards, the necessary Verilog code was automatically generated, hardware platform was added to the Vitis interface, which is Vivado's software tool, in .xsa format. After opening a new project in the Vitis interface and selecting the Hello World platform, the C code in figure 3.10 was written.

```c
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"

int main()
{
	init_platform();
	int N=46848;
	int Status;
	int i=0;
	u8 *radarData;
	radarData = malloc(sizeof(u8)*(N));

	Status = XGpio_Initialize(&Gpio, GPIO_EXAMPLE_DEVICE_ID);
	if(status!= XST_SUCCESS){
		xil_printf("Gpio initialization failed...\n\r");
		return XST_FAILURE;
	}
	while(1){
		scanf("%s",radar);
		XGpio_DiscreteWrite(&Gpio,radar[i]);
		i++;
	}
	cleanup_platform();
	return 0;
}
```

**Figure 3.10:** AXI GPIO C Code for Radar Data Transfer via UART

Memory is dynamically made available for radar data. Then the GPIO is initialized. It is intended to read the radar data with the scanf function in the while loop. Afterwards, the read data is wanted to be written to the terminal with GPIO. However, this code and block diagram were not suitable for both the transfer and format of the data. The desired result was not achieved.

### 3.4.2 AXI UART_LITE block diagram and C code application

It has been determined that data transfer with AXI GPIO is neither possible nor appropriate. As a result, the UART serial communication protocol was researched, and it was agreed that UART would be used for data transfer. As a result, the ZYNQ-7000 CPU and AXI Uartlite IP were immediately added to the block diagram and connected. It appeared automatically to link the Processor reset system to the AXI Interconnect Uartlite and Zynq-7000.

Block diagram was designed in Vivado IP Integrator Section and translated into hardware language. Afterwards, the necessary Verilog code was automatically generated, hardware platform was added to the Vitis interface, which is Vivado's software tool, in .xsa format. After opening a new project in the Vitis interface and selecting the Hello World platform, the C code in figure 3.12 was written.

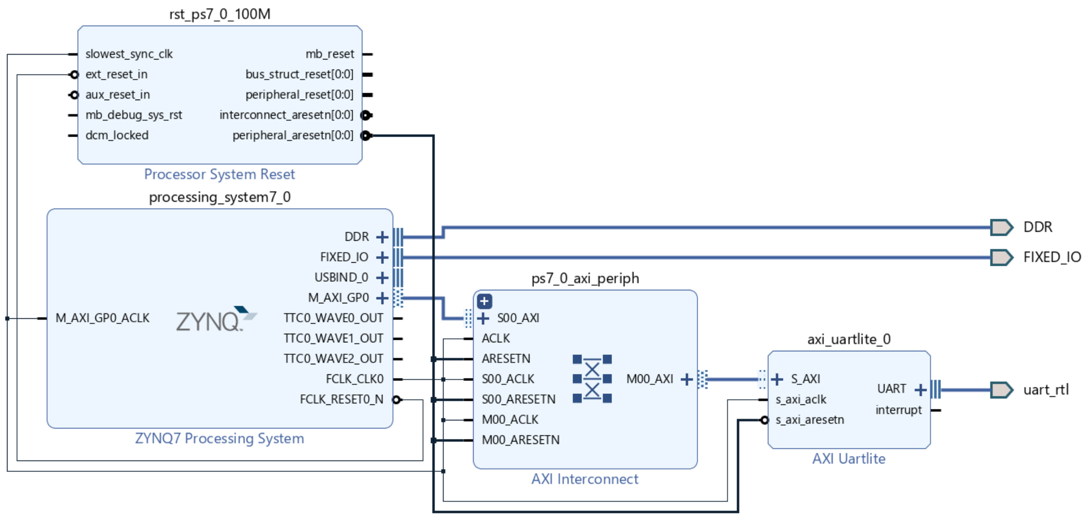


**Figure 3.11:** AXI UART_PS LITE Block Diagram for Radar Data Transfer via UART

```c
#include <stdio.h>
#include <stdlib.h>
#include "xil_types.h"
#include "xuartps.h"
#include "xparameters.h"

#define dataSize    5095*3
#define headSize    10
#define fileSize dataSize + headSize

int main(){
        u8 *radarData;
        u32 receivedBytes=0;
        u32 totalReceivedBytes=0;
        u32 status;
        u32 transmittedBytes=0;
        u32 totalTransmittedBytes=0;
        XUartPs_Config *myUartConfig;
        XUartPs myUart;

        radarData = malloc(sizeof(u8)*(fileSize));

        myUartConfig = XUartPs_LookupConfig(XPAR_PS7_UART_1_DEVICE_ID);
        status = XUartPs_CfgInitialize(&myUart, myUartConfig,
                                      myUartConfig->BaseAddress);
        if(status != XST_SUCCESS)
                print("UART initialization failed...\n\r");

        status = XUartPs_SetBaudRate(&myUart, 115200);
        if(status != XST_SUCCESS)
                print("Baud rate initialization failed...\n\r");

        while(totalReceivedBytes < fileSize){

        receivedBytes=XUartPs_Recv(&myUart,
                                (u8*)&radarData[totalReceivedBytes],
                                fileSize);
        totalReceivedBytes += receivedBytes;
        }

        for (int i=0;i<fileSize; i++)
                xil_printf("%0x",radarData[i]);
        // to see what we sent via UART
        //read data from ddr
        for(int i=headSize;i<fileSize;i++)
                radarData[i];
        // data can be processing in this part

        //send data back to the computer
        while(totalTransmittedBytes< fileSize){

        transmittedBytes = XUartPs_Send(&myUart,
                        (u8*)&radarData [totalTransmittedBytes],1);
        totalTransmittedBytes += transmittedBytes;

        }
```

**Figure 3.12:** AXI UART_PS LITE C Code for Radar Data Transfer via UART

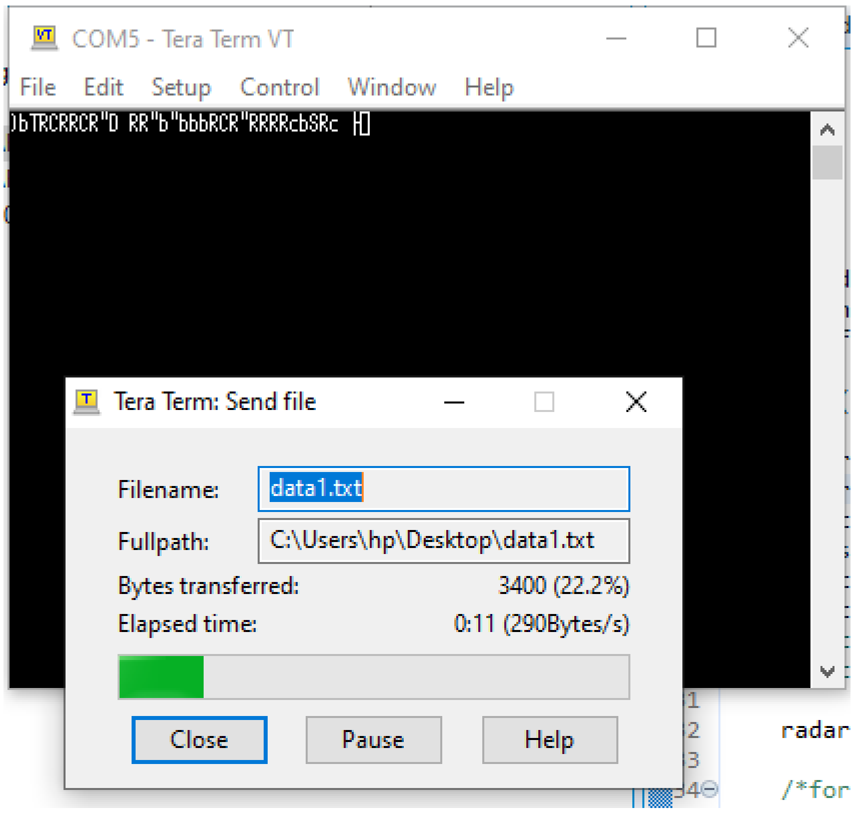
In the code, first of all, the necessary libraries are added. The library "stdio.h" is for general C functions, "stdlib.h" is for the malloc function, "xuartps.h" is for functions that contain the configuration of the UART protocol, "xil_types.h" is for the xil_printf function, and "xparameters.h" is the library used to initialize drivers.

Then, in order to test different sizes of the data and increase the understandability of the code, data sizes were determined with define. Variable radarData to receive data from radar, variable receivedBytes to assign radar data transferred with UART protocol, variable totalReceivedBytes to keep the total number of radar data transferred, variable status to control UART protocol status, variable transmittedBytes to send received radar data to the terminal, the variable totalTransmittedBytes to control the total number of radar data sent to the terminal and myUartConfig variable to make the configuration settings of the UART protocol is defined.

In the main body, first of all, memory has been made in the dimensions determined by the malloc function for the radar data to be received. Then, the UART protocol is configured by sending the necessary parameters into the XUartPs_LookupConfig function, which is taken from the header file, and the result is assigned to the status variable.

With the if statement, it is checked whether the configuration step has been carried out successfully. If the configuration is unsuccessful, the "UART initialization failed" sentence is observed from the terminal. The baud rate of the UART protocol is set with the XUartPs_SetBaudRate function and assigned to the status variable. With the if statement, it is checked whether the baud rate setting is successful or not. If the baud rate is wrong, the "Baud rate initialization failed" sentence is observed from the terminal.

In the first while loop, the total received data is updated, which is used to collect all the data received via the UART. The for loop is used to see what data is being sent via the UART. Secondly, the for loop has been added for data processing stages on the received data. If the data acquisition is completed successfully, the RNMF algorithm will be implemented in this part. The last while loop is used to send the processed data back from the UART to the computer and update the transmitted byte pointer. In this way, it will be checked whether the data read from the terminal and the data sent are the same. By integrating the C code written into the designed block diagram, the data in Figure 3.13 is observed from the terminal. However, these data are different from the sent data both in format and value. Therefore, it has been observed that this block diagram is also not suitable for UART serial communication.



```
)bTRCRRCR"D RR"b"bbbRCR"RRRRcbSRc H
```

Filename: data1.txt
Fullpath: C:\Users\hp\Desktop\data1.txt
Bytes transferred: 3400 (22.2%)
Elapsed time: 0:11 (290Bytes/s)

**Figure 3.13:** Observed Read Data in TeraTerm Terminal

### 3.4.3 Only Zynq-7000 processor block diagram and C code

When the block diagrams and C codes explained in sections 3.4.1 and 3.4.2 did not give successful results for data transfer, it was learned that the UART port of the Zedboard was accessed with MIOs, as explained in section 3.2. Therefore, it is thought that there is no need to use AXI GPIO and AXI UART_LITE in the block diagram.

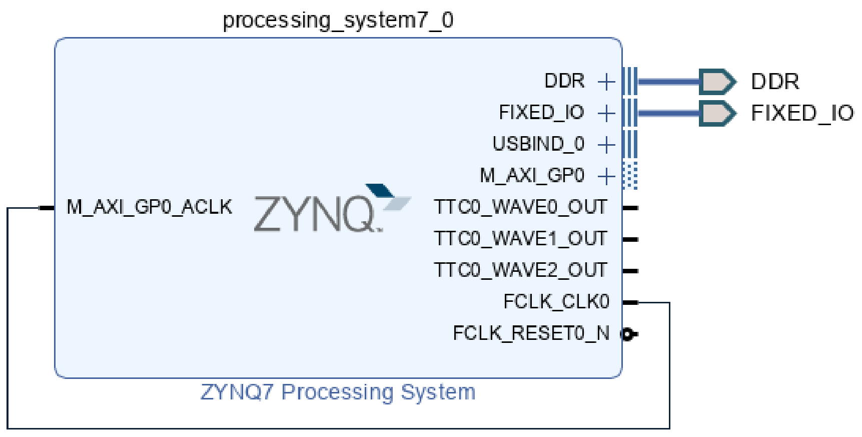


**Figure 3.14:** Zynq-7000 Processor Block Diagram for Radar Data Transfer via UART

For this reason, the block diagram was created by adding only the Zynq-7000 processor, without adding any extra IP. DDR and Fixed_IO outputs are taken from the processor, the clock setting is made by default. Then the generated block diagram is validated. The hardware part is completed by generating HDL Wrapper file from the created diagram. By exporting the hardware, the coding part was made in the Vitis interface.

```
#include <stdio.h>
#include "platform.h"
#include "xuartps.h"

XUartPs_Config *Config_0;
XUartPs Uart_PS_0;

int main()
{
	init_platform();
	int status;
	int N=46848;

	Config_0 = XUartPs_LookupConfig(XPAR_XUARTPS_0_DEVICE_ID);

	if(NULL==Config_0){
		return XST_FAILURE;
	}
	Status = XUartPs_CfgInitialize(&Uart_PS_0,Config_0,Config_0-
	>BaseAddress);
	if(status!= XST_SUCCESS){
		return XST_FAILURE;
	}
	char dizi[N];
	while(1){
		scanf("%s",dizi);
		printf("%s",dizi);
	}
	cleanup_platform();
	return 0;
}
```

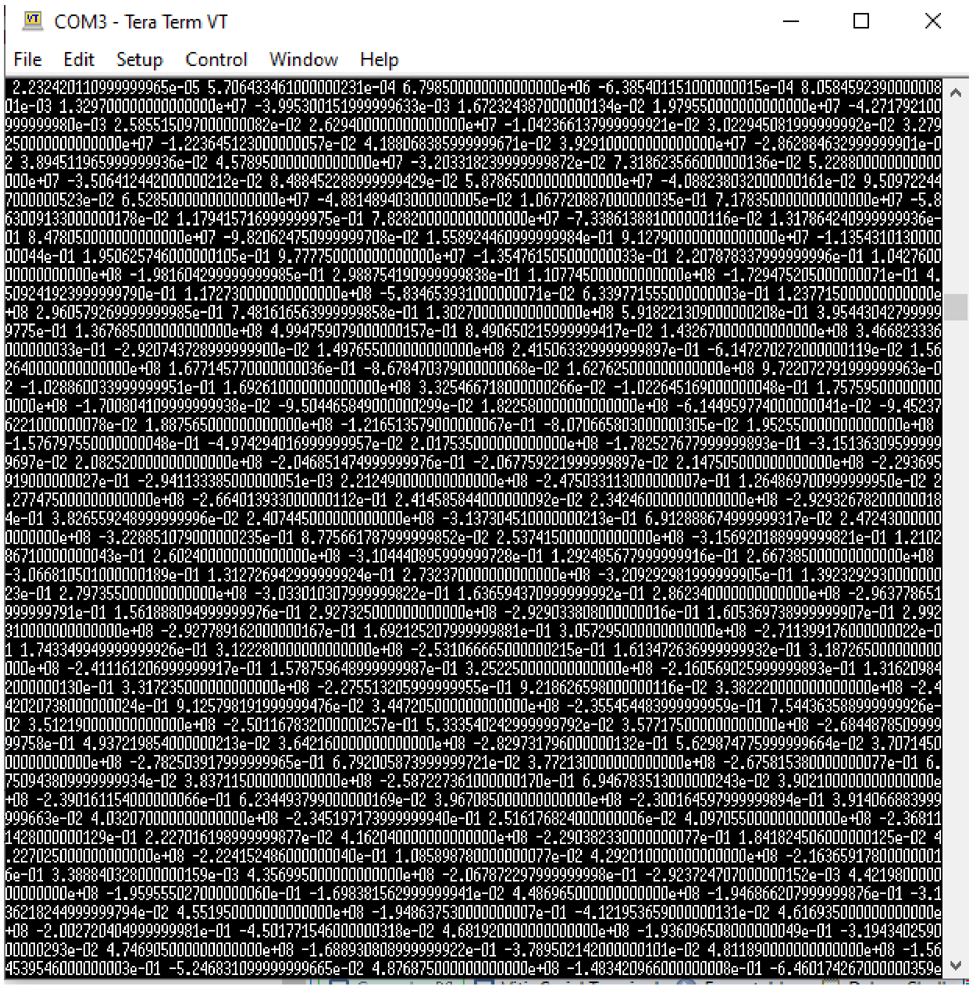
**Figure 3.15:** Zynq-7000 C Code for Radar Data Transfer via UART

First of all, necessary libraries are added in the code. The library "stdio.h" is for general C functions, "platform.h" is for init_platform and cleanup_platform functions, "xuartps.h" is for functions that contain the configuration of the UART protocol. Variable *Config_0 of type XUartPs_Config is defined for Configuration data structure, variable UART_PS_0 is of type XUartPs which contains information about UART to make UART configurations.

In the main body, first, the platform is initialized with the init_platform() function. Status is defined to control the status of the UART, and N is defined for the total number of radar data sent.

XUartPs_LookupConfig looks up the device configuration based on the unique device ID. With the if statement, it is checked whether this value is equal to NULL, if it is, it means that the device could not be configured and the XST_FAILURE value is returned without further processing. With the XUartPs_CfgInitialize function, a specific XUartPs instance is initialized to be ready for use. The device's data format is set to 8 data bits, 1 stop bit and no parity by default. With the if statement, it is checked whether the status value is different from XST_SUCCESS, if it is, it means that the UART configuration has not been done successfully and the code ends by returning the XST_FAILURE value without further action.

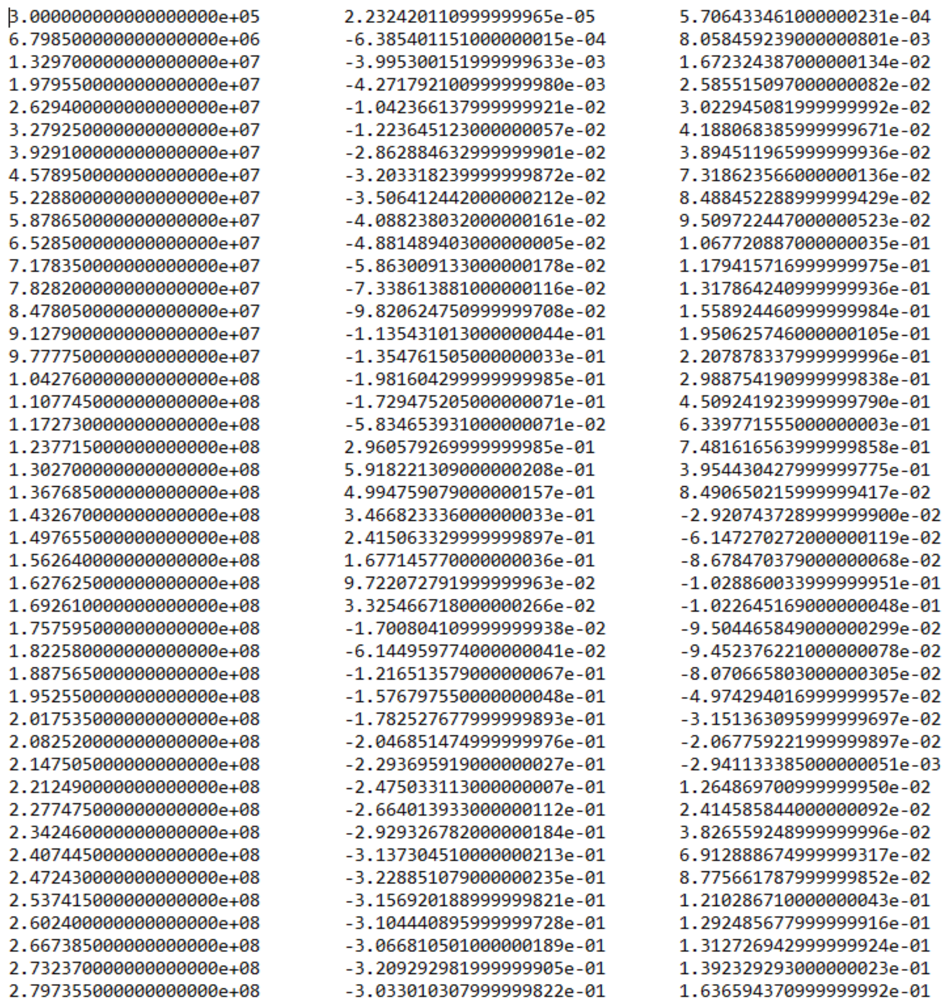
An array of type char and size N is defined for receiving radar data. The reason why this variable is char type is because radar data will be read from the text file. Then, within the while loop, the radar data is read one by one from the text file with the scanf function and written to the buffer with the printf function. It is checked whether the values written from the terminal are the same as the values sent.



```
 2.23242011099999965e-05 5.70643346100000231e-04 6.79850000000000000e+06 -6.38540115100000015e-04 8.058459239000008
01e-03 1.32970000000000000e+07 -3.99530015199999633e-03 1.67232438700000134e-02 1.97955000000000000e+07 -4.271792100
999999980e-03 2.58551509700000082e-02 2.62940000000000000e+07 -1.04236613799999921e-02 3.02294508199999992e-02 3.279
25000000000000e+07 -1.22364512300000057e-02 4.18806838599999671e-02 3.92910000000000000e+07 -2.86288463299999901e-0
2 3.89451196599999936e-02 4.57895000000000000e+07 -3.20331823999999872e-02 7.31862356600000136e-02 5.22880000000000
000e+07 -3.50641244200000212e-02 8.48845228899999429e-02 5.87865000000000000e+07 -4.08823803200000161e-02 9.50972244
7000000523e-02 6.52850000000000000e+07 -4.88148940300000005e-02 1.06772088700000035e-01 7.17835000000000000e+07 -5.8
6300913300000178e-02 1.17941571699999975e-01 7.82820000000000000e+07 -7.33861388100000116e-02 1.31786424099999936e-
01 8.47805000000000000e+07 -9.82062475099999708e-02 1.55892446099999984e-01 9.12790000000000000e+07 -1.135431013000
00044e-01 1.95062574600000105e-01 9.77775000000000000e+07 -1.35476150500000033e-01 2.20787833799999996e-01 1.0427600
0000000000e+08 -1.98160429999999985e-01 2.98875419099999838e-01 1.10774500000000000e+08 -1.72947520500000071e-01 4.
50924192399999790e-01 1.17273000000000000e+08 -5.83465393100000071e-02 6.33977155500000003e-01 1.23771500000000000e
+08 2.96057926999999985e-01 7.48161656399999858e-01 1.30270000000000000e+08 5.91822130900000208e-01 3.95443042799999
9775e-01 1.36768500000000000e+08 4.99475907900000157e-01 8.49065021599999417e-02 1.43267000000000000e+08 3.466823336
00000033e-01 -2.92074372899999900e-02 1.49765500000000000e+08 2.41506332999999897e-01 -6.14727027200000119e-02 1.56
26400000000000e+08 1.67714577000000036e-01 -8.67847037900000068e-02 1.62762500000000000e+08 9.72207279199999963e-0
2 -1.02886003399999951e-01 1.69261000000000000e+08 3.32546671800000266e-02 -1.02264516900000048e-01 1.7575950000000
0000e+08 -1.70080410999999938e-02 -9.50446584900000299e-02 1.82258000000000000e+08 -6.14495977400000041e-02 -9.45237
6221000000078e-02 1.88756500000000000e+08 -1.21651357900000067e-01 -8.07066580300000305e-02 1.95255000000000000e+08 
-1.57679755000000048e-01 -4.97429401699999957e-02 2.01753500000000000e+08 -1.78252767799999893e-01 -3.15136309599999
9697e-02 2.08252000000000000e+08 -2.04685147499999976e-01 -2.06775922199999897e-02 2.14750500000000000e+08 -2.293695
91900000027e-01 -2.94113338500000051e-03 2.21249000000000000e+08 -2.47503311300000007e-01 1.26486970099999950e-02 2
.27747500000000000e+08 -2.66401393300000112e-01 2.41458584400000092e-02 2.34246000000000000e+08 -2.929326782000000018
4e-01 3.82655924899999996e-02 2.40744500000000000e+08 -3.13730451000000213e-01 6.91288867499999317e-02 2.4724300000
0000000e+08 -3.22885107900000235e-01 8.77566178799999852e-02 2.53741500000000000e+08 -3.15692018899999821e-01 1.2102
8671000000043e-01 2.60240000000000000e+08 -3.10444089599999728e-01 1.29248567799999916e-01 2.66738500000000000e+08 
-3.06681050100000189e-01 1.31272694299999924e-01 2.73237000000000000e+08 -3.20929298199999905e-01 1.392329293000000
23e-01 2.79735500000000000e+08 -3.03301030799999822e-01 1.63659437099999992e-01 2.86234000000000000e+08 -2.963778651
999999791e-01 1.56188809499999976e-01 2.92732500000000000e+08 -2.92903380800000016e-01 1.60536973899999907e-01 2.992
31000000000000e+08 -2.92778916200000167e-01 1.69212520799999881e-01 3.05729500000000000e+08 -2.71139917600000022e-0
1 1.74334994999999926e-01 3.12228000000000000e+08 -2.53106666500000215e-01 1.61347263699999932e-01 3.1872650000000
000e+08 -2.41116120699999917e-01 1.57875964899999987e-01 3.25225000000000000e+08 -2.16056902599999893e-01 1.31620984
2000000130e-01 3.31723500000000000e+08 -2.27551320599999955e-01 9.21862659800000116e-02 3.38222000000000000e+08 -2.4
4202073800000024e-01 9.12579819199999476e-02 3.44720500000000000e+08 -2.35545448399999959e-01 7.54436358899999926e-
02 3.51219000000000000e+08 -2.50116783200000257e-01 5.33354024299999792e-02 3.57717500000000000e+08 -2.6844878509999
99758e-01 4.93721985400000213e-02 3.64216000000000000e+08 -2.82973179600000132e-01 5.62987477599999664e-02 3.7071450
0000000000e+08 -2.78250391799999965e-01 6.79200587399999721e-02 3.77213000000000000e+08 -2.67581538000000077e-01 6.
75094380999999934e-02 3.83711500000000000e+08 -2.58722736100000170e-01 6.94678351300000243e-02 3.90210000000000000e
+08 -2.39016115400000066e-01 6.23449379900000169e-02 3.96708500000000000e+08 -2.30016459799999894e-01 3.914066883999
999663e-02 4.03207000000000000e+08 -2.34519717399999940e-01 2.51617682400000006e-02 4.09705500000000000e+08 -2.36811
1428000000129e-01 2.22701619899999877e-02 4.16204000000000000e+08 -2.29038233000000077e-01 1.84182450600000125e-02 4
.22702500000000000e+08 -2.22415248600000040e-01 1.08589878000000077e-02 4.29201000000000000e+08 -2.163659178000001
6e-01 3.38884032800000159e-03 4.35699500000000000e+08 -2.06787229799999998e-01 -2.92372470700000152e-03 4.4219800000
0000000e+08 -1.95955502700000060e-01 -1.69838156299999941e-02 4.48696500000000000e+08 -1.94686620799999876e-01 -3.1
3621824499999794e-02 4.55195000000000000e+08 -1.94863753000000007e-01 -4.12195365900000131e-02 4.61693500000000000e
+08 -2.00272040499999981e-01 -4.50177154600000318e-02 4.68192000000000000e+08 -1.93609650800000049e-01 -3.1943402590
00000293e-02 4.74690500000000000e+08 -1.68893080899999922e-01 -3.78950214200000101e-02 4.81189000000000000e+08 -1.56
4539546000000003e-01 -5.24683109999999665e-02 4.87687500000000000e+08 -1.48342096600000008e-01 -6.46017426700000359e
```

**Figure 3.16:** Observed Read Data in TeraTerm Terminal

The values read from the terminal in Figure 3.16 and the actual radar data in Figure 3.17 are exactly the same. Thus, a radar data obtained from the measurement was taken in order from left to right in the terminal and displayed on the terminal, and the acquisition of the radar data was successfully performed.



```
3.000000000000000000e+05     2.232420110999999965e-05     5.706433461000000231e-04
6.798500000000000000e+06     -6.385401151000000015e-04    8.058459239000000801e-03
1.329700000000000000e+07     -3.995300151999999633e-03    1.672324387000000134e-02
1.979550000000000000e+07     -4.271792100999999980e-03    2.585515097000000082e-02
2.629400000000000000e+07     -1.042366137999999921e-02    3.022945081999999992e-02
3.279250000000000000e+07     -1.223645123000000057e-02    4.188068385999999671e-02
3.929100000000000000e+07     -2.862884632999999901e-02    3.894511965999999936e-02
4.578950000000000000e+07     -3.203318239999999872e-02    7.318623566000000136e-02
5.228800000000000000e+07     -3.506412442000000212e-02    8.488452288999999429e-02
5.878650000000000000e+07     -4.088238032000000161e-02    9.509722447000000523e-02
6.528500000000000000e+07     -4.881489403000000005e-02    1.067720887000000035e-01
7.178350000000000000e+07     -5.863009133000000178e-02    1.179415716999999975e-01
7.828200000000000000e+07     -7.338613881000000116e-02    1.317864240999999936e-01
8.478050000000000000e+07     -9.820624750999999708e-02    1.558924460999999984e-01
9.127900000000000000e+07     -1.135431013000000044e-01    1.950625746000000105e-01
9.777750000000000000e+07     -1.354761505000000033e-01    2.207878337999999996e-01
1.042760000000000000e+08     -1.981604299999999985e-01    2.988754190999999838e-01
1.107745000000000000e+08     -1.729475205000000071e-01    4.509241923999999790e-01
1.172730000000000000e+08     -5.834653931000000071e-02    6.339771555000000003e-01
1.237715000000000000e+08     2.960579269999999985e-01     7.481616563999999858e-01
1.302700000000000000e+08     5.918221309000000208e-01     3.954430427999999775e-01
1.367685000000000000e+08     4.994759079000000157e-01     8.490650215999999417e-02
1.432670000000000000e+08     3.466823336000000033e-01     -2.920743728999999900e-02
1.497655000000000000e+08     2.415063329999999897e-01     -6.147270272000000119e-02
1.562640000000000000e+08     1.677145770000000036e-01     -8.678470379000000068e-02
1.627625000000000000e+08     9.722072791999999963e-02     -1.028860033999999951e-01
1.692610000000000000e+08     3.325466718000000266e-02     -1.022645169000000048e-01
1.757595000000000000e+08     -1.700804109999999938e-02    -9.504465849000000299e-02
1.822580000000000000e+08     -6.144959774000000041e-02    -9.452376221000000078e-02
1.887565000000000000e+08     -1.216513579000000067e-01    -8.070665803000000305e-02
1.952550000000000000e+08     -1.576797550000000048e-01    -4.974294016999999957e-02
2.017535000000000000e+08     -1.782527677999999893e-01    -3.151363095999999697e-02
2.082520000000000000e+08     -2.046851474999999976e-01    -2.067759221999999897e-02
2.147505000000000000e+08     -2.293695919000000027e-01    -2.941133385000000051e-03
2.212490000000000000e+08     -2.475033113000000007e-01    1.264869700999999950e-02
2.277475000000000000e+08     -2.664013933000000112e-01    2.414585844000000092e-02
2.342460000000000000e+08     -2.929326782000000184e-01    3.826559248999999996e-02
2.407445000000000000e+08     -3.137304510000000213e-01    6.912888674999999317e-02
2.472430000000000000e+08     -3.228851079000000235e-01    8.775661787999999852e-02
2.537415000000000000e+08     -3.156920188999999821e-01    1.210286710000000043e-01
2.602400000000000000e+08     -3.104440895999999728e-01    1.292485677999999916e-01
2.667385000000000000e+08     -3.066810501000000189e-01    1.312726942999999924e-01
2.732370000000000000e+08     -3.209292981999999905e-01    1.392329293000000023e-01
2.797355000000000000e+08     -3.033010307999999822e-01    1.636594370999999992e-01
```

**Figure 3.17:** Real Radar Measurement Data

With this application, radar data was taken from the text file and correct values could be observed from the terminal. But the received data format is not suitable for processing. In order for radar data to be processed in the RNMF algorithm, it must be of double or float data type. However, the data we observe is of char type and is not suitable for processing. In the next step, the data received is converted to double or float data type and proceeded.

## 3.5 Succesful Data Transfer Implementation

In Section 3.4.3, we managed to receive the data via UART. However, the received data could not be processed because it was not in the desired format. Therefore, we need to convert the data we receive in string form to a double or float type suitable for processing. For this reason, the block diagram in Figure 3.18 was designed firstly. This block diagram was created using only the Zynq-7000 processor, on the basis that ZedBoard's UART connection can be provided from MIOs.

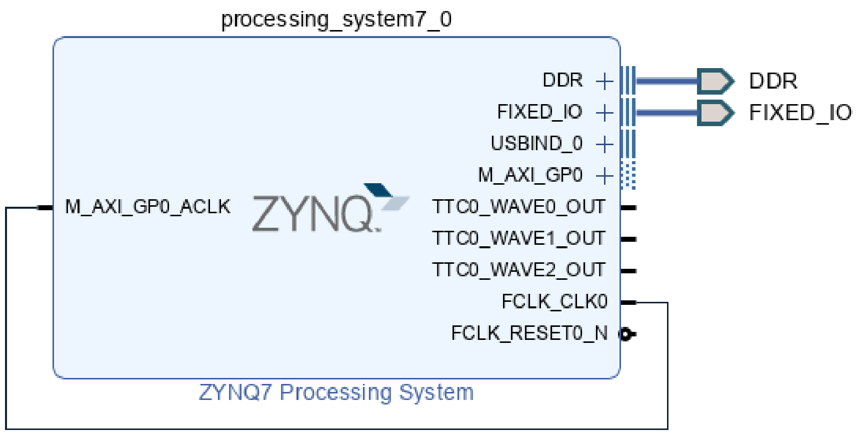


**Figure 3.18:** Block Design of Successful Data Transfer Implementation

Then the generated block diagram is validated. The hardware part is completed by generating HDL Wrapper file from the created diagram. By exporting the hardware, the coding part was made in the Vitis interface.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <stdlib.h>
#include "xuartps.h"

XUartPs_Config *Config_0;
XUartPs Uart_PS_0;

int main()
{
   init_platform();
       int status;
       Config_0 = XUartPs_LookupConfig(XPAR_XUARTPS_0_DEVICE_ID);
       if(NULL == Config_0){
              return XST_FAILURE;
       }
       Status = XUartPs_CfgInitialize(&Uart_PS_0,Config_0,Config_0-
       >BaseAddress);
       if(status!= XST_SUCCESS){
              return XST_FAILURE;
       }
       int N = 46848;
       int i = 0;
       int count = 0;
       float *X;
       X=malloc(N);
       char dizi[N];
       for(i=0;i<N;i++){
              scanf("%s",dizi);
              X[i] = strtof(dizi,NULL);
              printf("%.7f ",X[i]);
              printf("%d ",count);
              count++;
              if(i == N-1){
                  cleanup_platform();
                  return X;
              }
       }
}
```

**Figure 3.19:** Vitis C Code of Successful Data Transfer Implementation

First of all, necessary libraries are added in the code. The library "stdio.h" is for general C functions, "platform.h" is for init_platform and cleanup_platform functions, "xuartps.h" is for functions that contain the configuration of the UART protocol. Variable *Config_0 of type XUartPs_Config is defined for Configuration data structure, variable UART_PS_0 is of type XUartPs which contains information about UART to make UART configurations.

In the main body, first, the platform is initialized with the init_platform() function. Status is defined to control the status of the UART.

XUartPs_LookupConfig looks up the device configuration based on the unique device ID. With the if statement, it is checked whether this value is equal to NULL, if it is, it means that the device could not be configured and the XST_FAILURE value is returned without further processing. With the XUartPs_CfgInitialize function, a specific XUartPs instance is initialized to be ready for use. The device's data format is set to 8 data bits, 1 stop bit and no parity by default. With the if statement, it is checked whether the status value is different from XST_SUCCESS, if it is, it means that the UART configuration has not been done successfully and the code ends by returning the XST_FAILURE value without further action.

N is defined for the total number of radar data sent and i is defined as index to use in for loop. The count variable is defined in order to control how many data are read. X pointer of type float is defined to be used in the processing of data after conversion from string type to float type. Later, this variable X is dynamically made available in memory with the malloc function. A char type array is defined to read radar data with UART.

In the for loop, firstly, data is read from the radar with the scanf function. These data, read in string format, are converted to float type with the strtof function and transferred to the variable X. By repeating this process N steps, X array is filled one by one. In order to control the sent and received data, each data is printed to the terminal with printf. In addition, to see how many data have been read, the count variable is also printed after each data read. In order for this part to be integrated as a function in the whole code in later operations, the X array is returned in the if block.

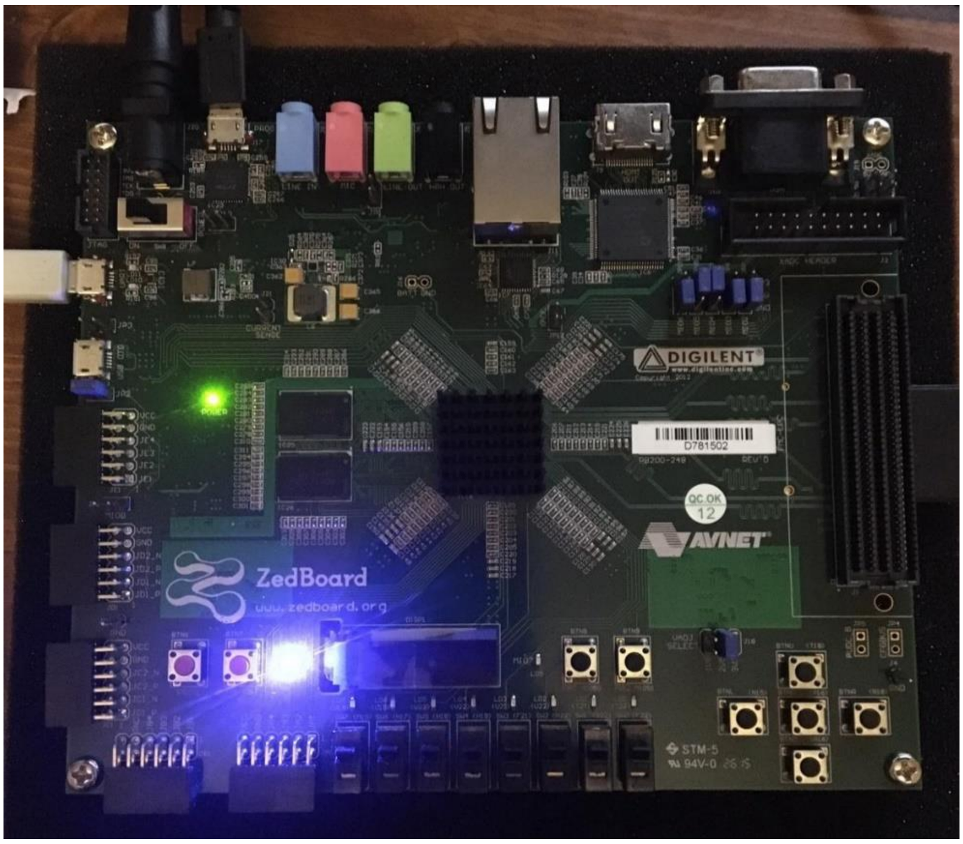
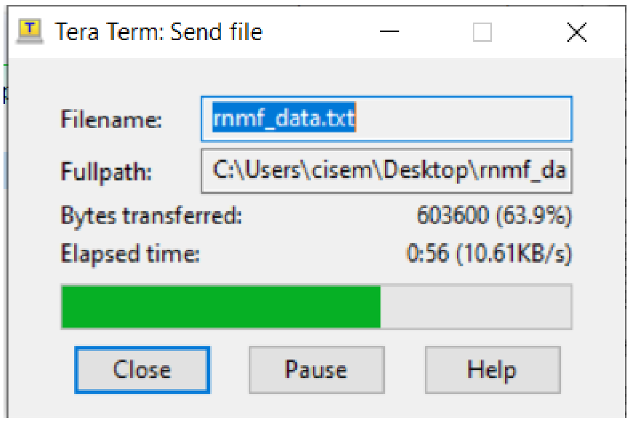


**Figure 3.20:** FPGA Programmed Correctly, FPGA Display

The blue LED, which indicates that the FPGA board has been programmed correctly and without errors, is on as shown in Figure 3.19. With TeraTerm, the data in the text file is sent to the FPGA side.


Tera Term: Send file

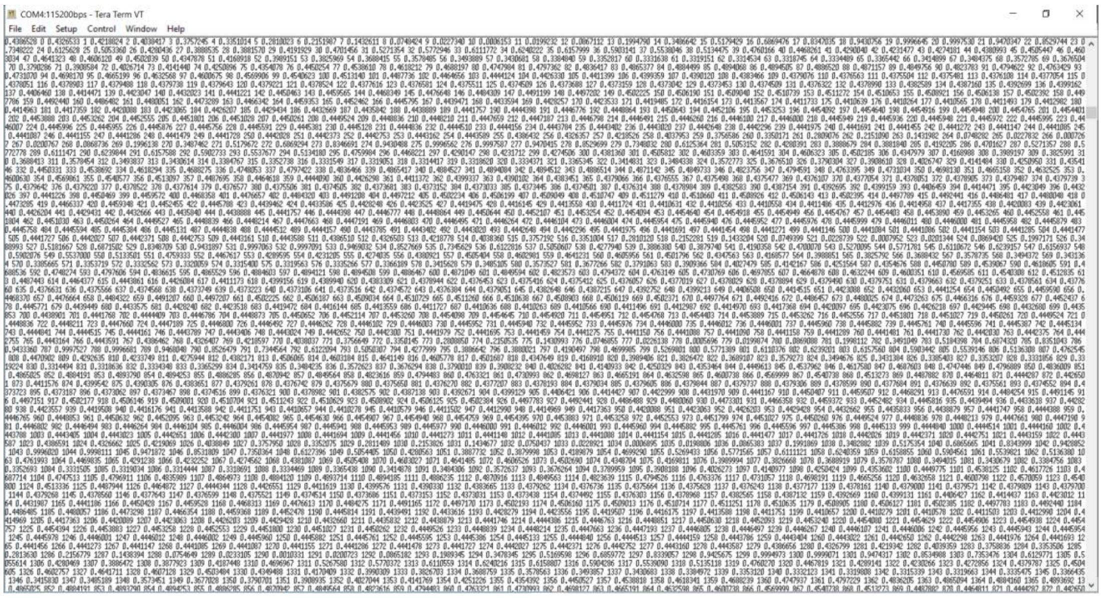
Filename: rnmf_data.txt

Fullpath: C:\Users\cisem\Desktop\rnmf_da

Bytes transferred: 603600 (63.9%)

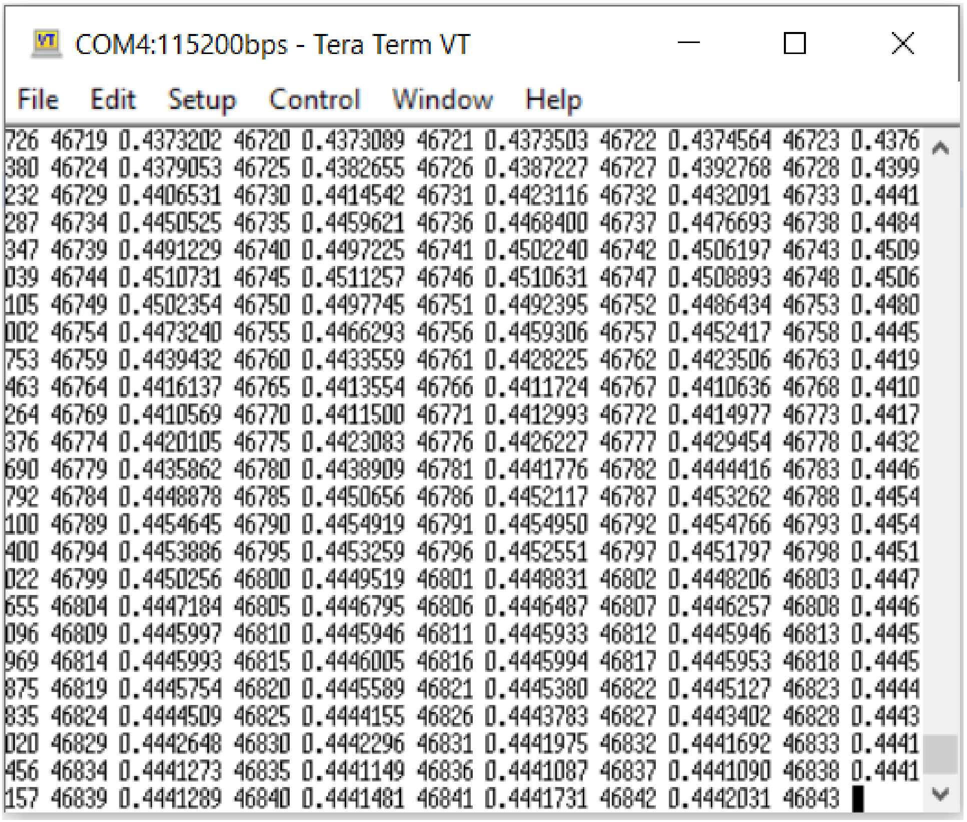
Elapsed time: 0:56 (10.61KB/s)

**Figure 3.21:** Sending Data From Terminal to FPGA


COM4:115200bps - Tera Term VT
File Edit Setup Control Window Help

```
[illegible]
```

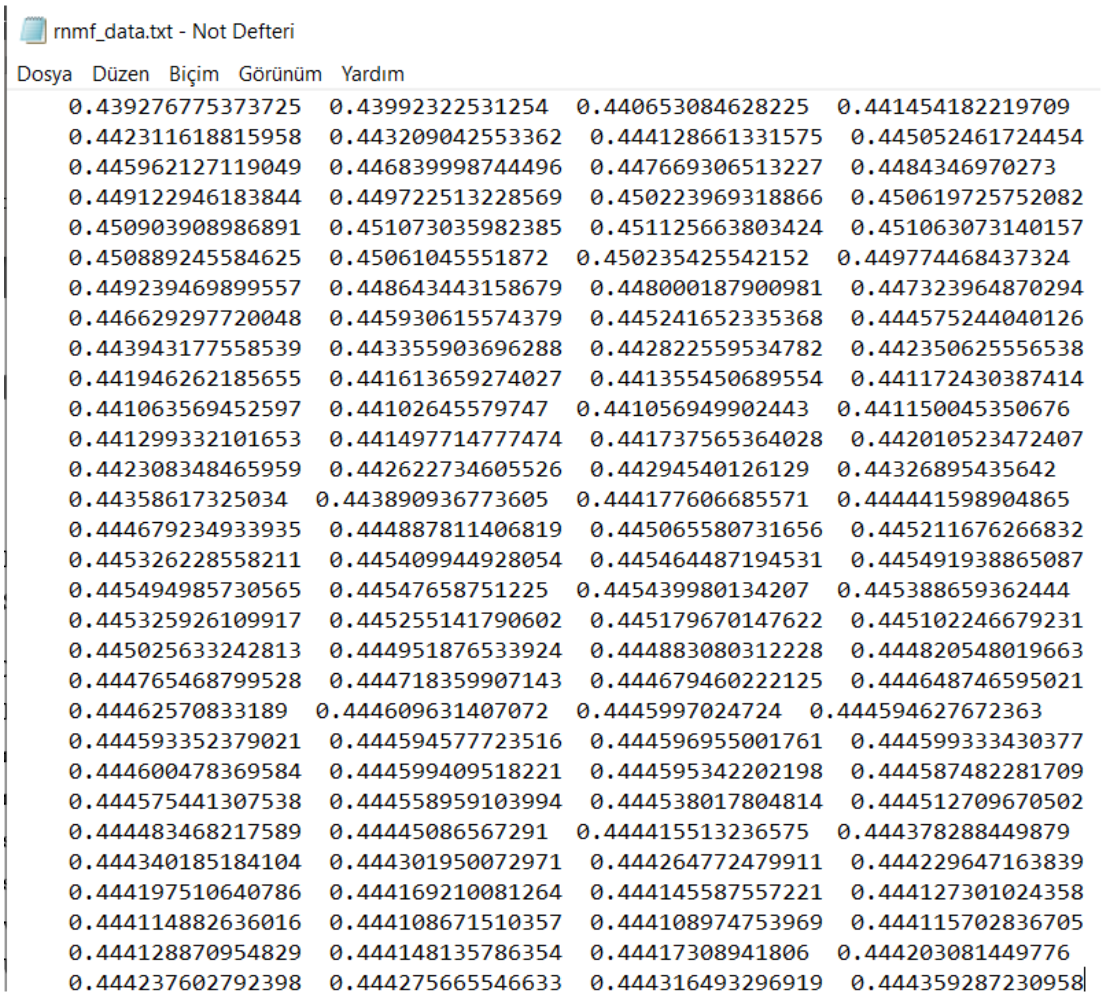
**Figure 3.22:** Data Read From Terminal


COM4:115200bps - Tera Term VT
File Edit Setup Control Window Help

```
726 46719 0.4373202 46720 0.4373089 46721 0.4373503 46722 0.4374564 46723 0.4376
380 46724 0.4379053 46725 0.4382655 46726 0.4387227 46727 0.4392768 46728 0.4399
232 46729 0.4406531 46730 0.4414542 46731 0.4423116 46732 0.4432091 46733 0.4441
287 46734 0.4450525 46735 0.4459621 46736 0.4468400 46737 0.4476693 46738 0.4484
347 46739 0.4491229 46740 0.4497225 46741 0.4502240 46742 0.4506197 46743 0.4509
039 46744 0.4510731 46745 0.4511257 46746 0.4510631 46747 0.4508893 46748 0.4506
105 46749 0.4502354 46750 0.4497745 46751 0.4492395 46752 0.4486434 46753 0.4480
002 46754 0.4473240 46755 0.4466293 46756 0.4459306 46757 0.4452417 46758 0.4445
753 46759 0.4439432 46760 0.4433559 46761 0.4428225 46762 0.4423506 46763 0.4419
463 46764 0.4416137 46765 0.4413554 46766 0.4411724 46767 0.4410636 46768 0.4410
264 46769 0.4410569 46770 0.4411500 46771 0.4412993 46772 0.4414977 46773 0.4417
376 46774 0.4420105 46775 0.4423083 46776 0.4426227 46777 0.4429454 46778 0.4432
690 46779 0.4435862 46780 0.4438909 46781 0.4441776 46782 0.4444416 46783 0.4446
792 46784 0.4448878 46785 0.4450656 46786 0.4452117 46787 0.4453262 46788 0.4454
100 46789 0.4454645 46790 0.4454919 46791 0.4454950 46792 0.4454766 46793 0.4454
400 46794 0.4453886 46795 0.4453259 46796 0.4452551 46797 0.4451797 46798 0.4451
022 46799 0.4450256 46800 0.4449519 46801 0.4448831 46802 0.4448206 46803 0.4447
655 46804 0.4447184 46805 0.4446795 46806 0.4446487 46807 0.4446257 46808 0.4446
096 46809 0.4445997 46810 0.4445946 46811 0.4445933 46812 0.4445946 46813 0.4445
969 46814 0.4445993 46815 0.4446005 46816 0.4445994 46817 0.4445953 46818 0.4445
875 46819 0.4445754 46820 0.4445589 46821 0.4445380 46822 0.4445127 46823 0.4444
835 46824 0.4444509 46825 0.4444155 46826 0.4443783 46827 0.4443402 46828 0.4443
020 46829 0.4442648 46830 0.4442296 46831 0.4441975 46832 0.4441692 46833 0.4441
456 46834 0.4441273 46835 0.4441149 46836 0.4441087 46837 0.4441090 46838 0.4441
157 46839 0.4441289 46840 0.4441481 46841 0.4441731 46842 0.4442031 46843 █
```

**Figure 3.23:** Last Data Read From Terminal

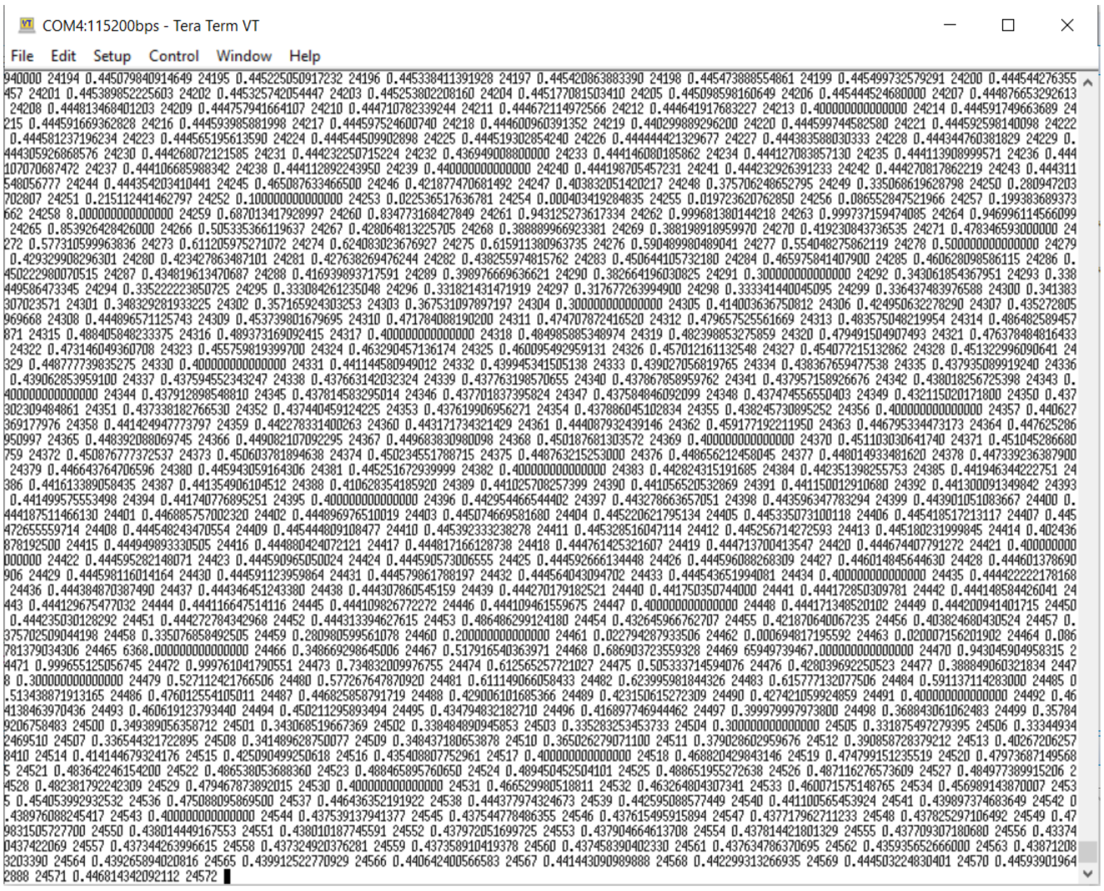
Figure 3.21 shows the first data read from the terminal and Figure 3.22 shows the last data read from the terminal. It has been proven that these data agree with the data in Figure 3.23.



```
    0.439276775373725  0.43992322531254  0.440653084628225  0.441454182219709
    0.442311618815958  0.443209042553362  0.444128661331575  0.445052461724454
    0.445962127119049  0.446839998744496  0.447669306513227  0.4484346970273
    0.449122946183844  0.449722513228569  0.450223969318866  0.450619725752082
    0.450903908986891  0.451073035982385  0.451125663803424  0.451063073140157
    0.450889245584625  0.45061045551872  0.450235425542152  0.449774468437324
    0.449239469899557  0.448643443158679  0.448000187900981  0.447323964870294
    0.446629297720048  0.445930615574379  0.445241652335368  0.444575244040126
    0.443943177558539  0.443355903696288  0.442822559534782  0.442350625556538
    0.441946262185655  0.441613659274027  0.441355450689554  0.441172430387414
    0.441063569452597  0.44102645579747  0.441056949902443  0.441150045350676
    0.441299332101653  0.441497714777474  0.441737565364028  0.442010523472407
    0.442308348465959  0.442622734605526  0.44294540126129  0.44326895435642
    0.44358617325034  0.443890936773605  0.444177606685571  0.444441598904865
    0.444679234933935  0.444887811406819  0.445065580731656  0.445211676266832
    0.445326228558211  0.445409944928054  0.445464487194531  0.445491938865087
    0.445494985730565  0.44547658751225  0.445439980134207  0.445388659362444
    0.445325926109917  0.445255141790602  0.445179670147622  0.445102246679231
    0.445025633242813  0.444951876533924  0.444883080312228  0.444820548019663
    0.444765468799528  0.444718359907143  0.444679460222125  0.444648746595021
    0.44462570833189  0.444609631407072  0.4445997024724  0.444594627672363
    0.444593352379021  0.444594577723516  0.444596955001761  0.444599333430377
    0.444600478369584  0.444599409518221  0.444595342202198  0.444587482281709
    0.444575441307538  0.444558959103994  0.444538017804814  0.444512709670502
    0.444483468217589  0.44445086567291  0.444415513236575  0.444378288449879
    0.444340185184104  0.444301950072971  0.444264772479911  0.444229647163839
    0.444197510640786  0.444169210081264  0.444145587557221  0.444127301024358
    0.444114882636016  0.444108671510357  0.444108974753969  0.444115702836705
    0.444128870954829  0.444148135786354  0.44417308941806  0.444203081449776
    0.444237602792398  0.444275665546633  0.444316493296919  0.444359287230958
```

**Figure 3.24:** Data Sent to FPGA

With the code in Figure 3.18, the data is taken in float type. Radar data is of double type. For this, double type data was obtained by making a few changes on the code. For this, float *X as double *X, X[i] = strtof(array,NULL) as X[i] = strtod(array,NULL) and printf("%.7f ", X[i]) line is changed to printf("%.15f ", X[i]) because it has 15 digits in double type. The data read with this code is given in Figure 3.22.



```
940000 24194 0.445079840914649 24195 0.445225050917232 24196 0.445338411391928 24197 0.445420863883390 24198 0.445473888554861 24199 0.445499732579291 24200 0.444544276355
457 24201 0.445389852225603 24202 0.445325742054447 24203 0.445253802208160 24204 0.445177081503410 24205 0.445098598160649 24206 0.445444524680000 24207 0.444876653292613
 24208 0.444813468401203 24209 0.444757941664107 24210 0.444710782339244 24211 0.444672114972566 24212 0.444641917683227 24213 0.400000000000000 24214 0.444591749663689 24
215 0.444591669362828 24216 0.444593985881998 24217 0.444597524600740 24218 0.444600960391352 24219 0.440299889296200 24220 0.444599744582580 24221 0.444592598140098 24222
 0.444581237196234 24223 0.444565195613590 24224 0.444544509902898 24225 0.444519302854240 24226 0.444444421329677 24227 0.444383588030333 24228 0.444344760381829 24229 0.
444305926868576 24230 0.444268072121585 24231 0.444232250715224 24232 0.436949008800000 24233 0.444146080185862 24234 0.444127083857130 24235 0.444113908999571 24236 0.444
107070687472 24237 0.444106685988342 24238 0.444112892243950 24239 0.440000000000000 24240 0.444198705457231 24241 0.444232926391233 24242 0.444270817862219 24243 0.444311
548056777 24244 0.444354203410441 24245 0.465087633466500 24246 0.421877470681492 24247 0.403832051420217 24248 0.375706248652795 24249 0.335068619628798 24250 0.280947203
702807 24251 0.215112441462797 24252 0.100000000000000 24253 0.022536517636781 24254 0.000403419284835 24255 0.019723620762850 24256 0.086552847521966 24257 0.199383689373
662 24258 8.000000000000000 24259 0.687013417928997 24260 0.834773168427849 24261 0.943125273617334 24262 0.999681380144218 24263 0.999737159474085 24264 0.946996114566099
 24265 0.853926428426000 24266 0.505335366119637 24267 0.428064813225705 24268 0.388889966923381 24269 0.388198918959970 24270 0.419230843736535 24271 0.478346593000000 24
272 0.577310599963836 24273 0.611205975271072 24274 0.624083023676927 24275 0.615911380963735 24276 0.590489980489041 24277 0.554048275862119 24278 0.500000000000000 24279
 0.429329908296301 24280 0.423427863487101 24281 0.427638269476244 24282 0.438255974815762 24283 0.450644105732180 24284 0.465975841407900 24285 0.460628098586115 24286 0.
450222980070515 24287 0.434819613470687 24288 0.416939893717591 24289 0.398976669636621 24290 0.382664196030825 24291 0.300000000000000 24292 0.343061854367951 24293 0.338
449586473345 24294 0.335222223850725 24295 0.333084261235048 24296 0.331821431471919 24297 0.317677263994900 24298 0.333341440045095 24299 0.336437483976588 24300 0.341383
307023571 24301 0.348329281933225 24302 0.357165924303253 24303 0.367531097897197 24304 0.300000000000000 24305 0.414003636750812 24306 0.424950632278290 24307 0.435272805
969668 24308 0.444896571125743 24309 0.453739801679695 24310 0.471784088190200 24311 0.474707872416520 24312 0.479657525561669 24313 0.483575048219954 24314 0.486482589457
871 24315 0.488405848233375 24316 0.489373169092415 24317 0.400000000000000 24318 0.484985885348974 24319 0.482398853275859 24320 0.479491504907493 24321 0.476378484816433
 24322 0.473146049360708 24323 0.455759819399700 24324 0.463290457136174 24325 0.460095492959131 24326 0.457012161132548 24327 0.454077215132862 24328 0.451322996090641 24
329 0.448777739835275 24330 0.400000000000000 24331 0.441144580949012 24332 0.439945341505138 24333 0.439027056819765 24334 0.438367659477538 24335 0.437935089919240 24336
 0.439062853959100 24337 0.437594552343247 24338 0.437663142032324 24339 0.437763198570655 24340 0.437867858959762 24341 0.437957158926676 24342 0.438018256725398 24343 0.
400000000000000 24344 0.437912898548810 24345 0.437814583295014 24346 0.437701837395824 24347 0.437584846092099 24348 0.437474556550403 24349 0.432115020171800 24350 0.437
302309484861 24351 0.437338182766530 24352 0.437440459124225 24353 0.437619906956271 24354 0.437886045102834 24355 0.438245730895252 24356 0.400000000000000 24357 0.440627
369177976 24358 0.441424947773797 24359 0.442278331400263 24360 0.443171734321429 24361 0.444087932439146 24362 0.459177192211950 24363 0.446795334473173 24364 0.447625286
950997 24365 0.448392088069745 24366 0.449082107092295 24367 0.449683830980098 24368 0.450187681303572 24369 0.400000000000000 24370 0.451103030641740 24371 0.451045286680
759 24372 0.450876777372537 24373 0.450603781894638 24374 0.450234551788715 24375 0.448763215253000 24376 0.448656212458045 24377 0.448014933481620 24378 0.447339236387900
 24379 0.446643764706596 24380 0.445943059164306 24381 0.445251672939999 24382 0.400000000000000 24383 0.442824315191685 24384 0.442351398255753 24385 0.441946344222751 24
386 0.441613389058435 24387 0.441354906104512 24388 0.410628354185920 24389 0.441025708257399 24390 0.441056520532869 24391 0.441150012910680 24392 0.441300091349842 24393
 0.441499575553498 24394 0.441740776895251 24395 0.400000000000000 24396 0.442954466544402 24397 0.443278663657051 24398 0.443596347783294 24399 0.443901051083667 24400 0.
444187511466130 24401 0.446885757002320 24402 0.444896976510019 24403 0.445074669581680 24404 0.445220621795134 24405 0.445335073100118 24406 0.445418517213117 24407 0.445
472655559714 24408 0.444548243470554 24409 0.445444809108477 24410 0.445392333238278 24411 0.445328516047114 24412 0.445256714272593 24413 0.445180231999845 24414 0.402436
878192500 24415 0.444949893330505 24416 0.444880424072121 24417 0.444817166128738 24418 0.444761425321607 24419 0.444713700413547 24420 0.444674407791272 24421 0.400000000
000000 24422 0.444595282148071 24423 0.444590965050024 24424 0.444590573006555 24425 0.444592666134448 24426 0.444596088268309 24427 0.446014845644630 24428 0.444601378690
906 24429 0.444598116014164 24430 0.444591123959864 24431 0.444579861788197 24432 0.444564043094702 24433 0.444543651994081 24434 0.400000000000000 24435 0.444422222178168
 24436 0.444384870387490 24437 0.444346451243380 24438 0.444307860545159 24439 0.444270179182521 24440 0.441750350744000 24441 0.444172850309781 24442 0.444148584426041 24
443 0.444129675477032 24444 0.444116647514116 24445 0.444109826772272 24446 0.444109461559675 24447 0.400000000000000 24448 0.444171348520102 24449 0.444200941401715 24450
 0.444235030128292 24451 0.444272784342968 24452 0.444313394627615 24453 0.486486299124180 24454 0.432645966762707 24455 0.421870640067235 24456 0.403824680430524 24457 0.
375702509044198 24458 0.335076858492505 24459 0.280980599561078 24460 0.200000000000000 24461 0.022794287933506 24462 0.000694817195592 24463 0.020007156201902 24464 0.086
781379034306 24465 6368.000000000000000 24466 0.348669298645006 24467 0.517916540363971 24468 0.686903723559328 24469 65949739467.000000000000000 24470 0.943045904958315 2
4471 0.999655125056745 24472 0.999761041790551 24473 0.734832009976755 24474 0.612565257721027 24475 0.505333714594076 24476 0.428039692250523 24477 0.388849060321834 2447
8 0.300000000000000 24479 0.527112421766506 24480 0.577267647870920 24481 0.611149066058433 24482 0.623995981844326 24483 0.615777132077506 24484 0.591137114283000 24485 0
.513438871913165 24486 0.476012554105011 24487 0.446825858791719 24488 0.429006101685366 24489 0.423150615272309 24490 0.427421059924859 24491 0.400000000000000 24492 0.46
4138463970436 24493 0.460619123793440 24494 0.450211295893494 24495 0.434794832182710 24496 0.416897746944462 24497 0.399979997973800 24498 0.368843061062483 24499 0.35784
9206758483 24500 0.349389056358712 24501 0.343068519667369 24502 0.338484890945853 24503 0.335283253453733 24504 0.300000000000000 24505 0.331875497279395 24506 0.33344934
2469510 24507 0.336544321722895 24508 0.341489628750077 24509 0.348437180653878 24510 0.365026279071100 24511 0.379028602959676 24512 0.390858728379212 24513 0.40267206257
8410 24514 0.414144679324176 24515 0.425090499250618 24516 0.435408807752961 24517 0.400000000000000 24518 0.468820429843146 24519 0.474799151235519 24520 0.47973687149568
5 24521 0.483642246154200 24522 0.486538053688360 24523 0.488465895760650 24524 0.489450452504101 24525 0.488651955272638 24526 0.487116276573609 24527 0.484977389915206 2
4528 0.482381792242309 24529 0.479467873892015 24530 0.400000000000000 24531 0.466529980518811 24532 0.463264804307341 24533 0.460071575148765 24534 0.456989143870007 2453
5 0.454053992932532 24536 0.475088095869500 24537 0.446436352191922 24538 0.444377974324673 24539 0.442595088577449 24540 0.441100565453924 24541 0.439897374683649 24542 0
.438976088245417 24543 0.400000000000000 24544 0.437539137941377 24545 0.437544778486355 24546 0.437615495915894 24547 0.437717962711233 24548 0.437825297106492 24549 0.47
9831505727700 24550 0.438014449167553 24551 0.438010187745591 24552 0.437972051699725 24553 0.437904664613708 24554 0.437814421801329 24555 0.437709307180680 24556 0.43374
0437422069 24557 0.437344263996615 24558 0.437324920376281 24559 0.437358910419378 24560 0.437458390402330 24561 0.437634786370695 24562 0.435935652666000 24563 0.43871208
3203390 24564 0.439265894020816 24565 0.439912522770929 24566 0.440642400566583 24567 0.441443090989888 24568 0.442299313266935 24569 0.444503224830401 24570 0.44593901964
2888 24571 0.446814342092112 24572 ▌
```

**Figure 3.25:** Data Read from Terminal

At this stage, looking at the last read data and the number of read data, the data in both float and double data types are ready to be processed and read correctly. However, while 48844 data can be read in float data type, 24573 data can be read in double data type. This is due to the size of the buffer.

# 4. RNMF APPLICATION ON VITIS

In this section, what the RNMF algorithm is, for what purpose it is used, what the GPR picture is, the literature review on these issues, and finally the C code and outputs of the RNMF algorithm implemented in the Vitis interface will be explained.

## 4.1 RNMF Algorithm

NMF algorithm has become very popular in recent years for those who produce results by processing data with the importance of data science. The NMF algorithm aims to automatically extract hidden layers from data consisting of high-dimensional matrices and predicts the solution of data-driven problems such as matrices reduction, unsupervised learning, and classification problems.

The most critical problem encountered in ground penetrating radar (GPR) studies is the clutter that reflects back to the radar from the ground and obscures the targeted image. Clutter prevents the images of objects under the ground and makes it difficult to detect. In this project, RNMF, which is an improved version of the NMF algorithm, was used to detect the real image by separating the data collected from the field by the ground penetrating radar system from the clutter. The GPR image is represented by a rectangular matrix X with dimensions $M \times N$, where M is the depth index and N is the number of antenna positions. The X rectangular matrix consists of two parts, the target and the clutter. Using the RNMF optimization algorithm, the X matrix is assumed to be sparsely degraded and decomposes it from a sparse error matrix S as non-negative W and H matrices defined as the data matrix [12]. Thus, the optimization formula proposed by the NMF algorithm will be updated as RNMF. By iteratively solving the optimization problem in this new formula, the S matrix will be 0, the W and H values will be found by normalizing.

## 4.2 Information About GPR Image

In this study, ground penetrating radar (GPR) image of 256x183 size was obtained by measuring with vivaldi antennas. The GPR image, which was defined as X above, consists of the target and the clutter. In the RNMF algorithm, the X image is considered to be sparsely distorted and defined as X = W*H+S.

## 4.3 Literatur Review

In matrix factorization model, three essential questions need answering: 1) existence, whether the nontrivial NMF solutions exist; 2) uniqueness, under what assumptions NMF is, at least in some sense, unique; 3) effectiveness, under what assumptions NMF is able to recover the "right answer." [14,17]. A method called robust NMF (RNMF) was proposed, which has a better clutter removal effect than other low-rank and sparse decomposition methods, but the solution process is still not fast enough due to its iterative characteristics [15].As a more advanced version of NMF, the RNMF algorithm is used in remote sensing and image processing studies to detect objects and distinguish undesired effects from the target. RNMF is more successful than other decomposition methods in reducing the error rate at the output to 0 by putting a small amount of error on the input data and in distinguishing the object to be detected from the clutter [16,18].

The radar image (raw image) obtained in the simulations using real data in the MATLAB environment in the previous studies of the project is given in Figure 4.1.

Target image is observed in Figure 4.2 and clutter image is observed in Figure 4.3. The target and clutter were separated by running the RNMF algorithm on the Zynq-7000 processor, which was written in the Matlab environment and then translated into C language.

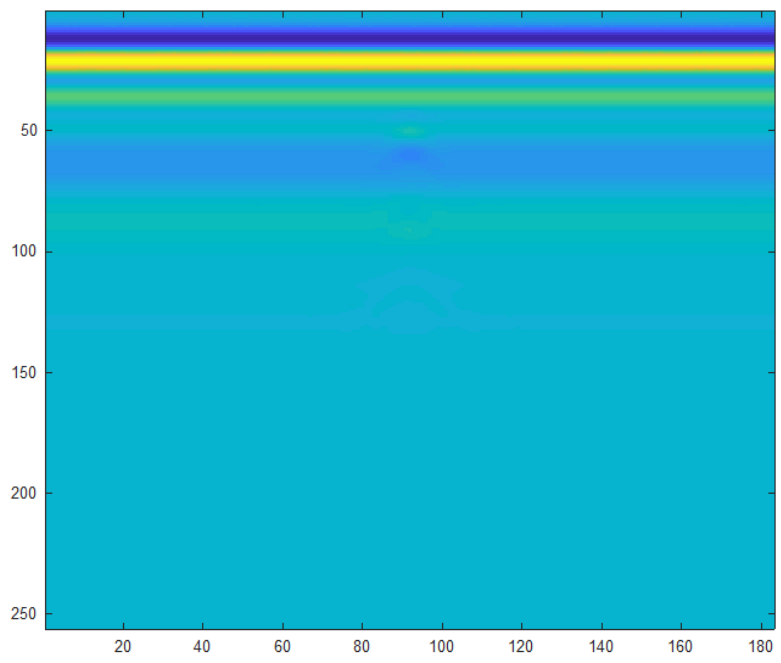


**Figure 4.1:** Raw GPR Image Collected from Target Object

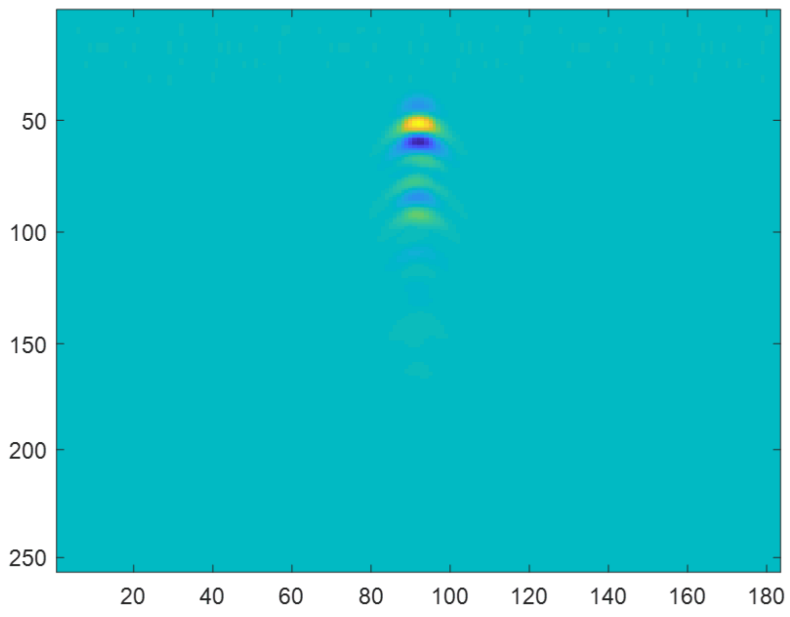


**Figure 4.2:** Target Object Removed Clutter Image

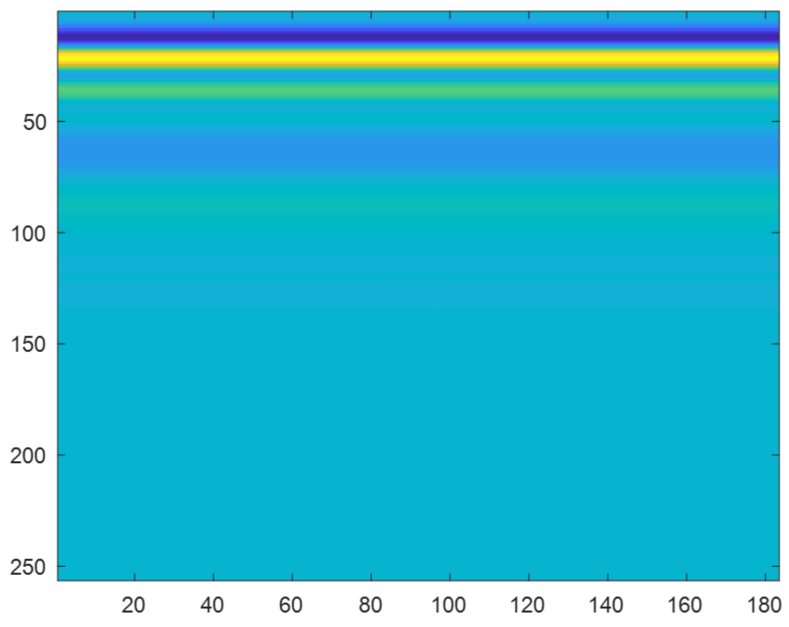
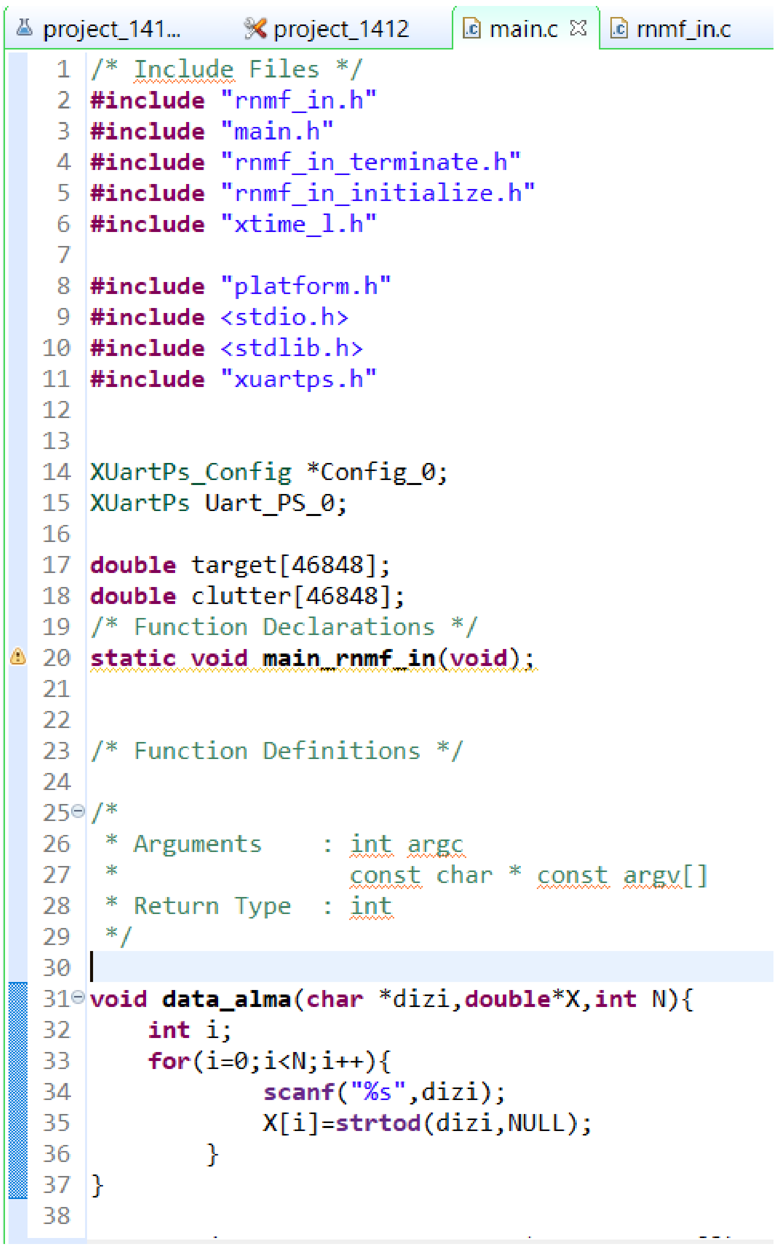


**Figure 4.3:** Clutter (Object Removed) Image

## 4.4 RNMF Application VITIS C Code

In this project, the RNMF algorithm is used to distinguish the data received from the radar as clutter and target, and consequently to remove the complexity. Simpler and shorter C codes to implement the RNMF algorithm are available in the literature. However, since the code written in this project is aimed to work on FPGA, the code is arranged to be applicable in FPGA. Since there are many matrix multiplications in this code and it will take a long time to perform these operations in software, the hardware designed and designed system of matrix multiplications is integrated into the software.



```
/* Include Files */
#include "rnmf_in.h"
#include "main.h"
#include "rnmf_in_terminate.h"
#include "rnmf_in_initialize.h"
#include "xtime_l.h"

#include "platform.h"
#include <stdio.h>
#include <stdlib.h>
#include "xuartps.h"


XUartPs_Config *Config_0;
XUartPs Uart_PS_0;

double target[46848];
double clutter[46848];
/* Function Declarations */
static void main_rnmf_in(void);


/* Function Definitions */

/*
 * Arguments    : int argc
 *                const char * const argv[]
 * Return Type  : int
 */

void data_alma(char *dizi,double*X,int N){
    int i;
    for(i=0;i<N;i++){
            scanf("%s",dizi);
            X[i]=strtod(dizi,NULL);
        }
}
```

**Figure 4.4:** RNMF Vitis Main C Code (1-38)

In the code written before, the target variable was defined as a fixed array and the values were initialized. Since this variable is the data to be received from the radar, in this project, it was taken from the radar with the code written in the third section.

In Figure 4.4, rnmf_in, rnmf_in_terminate and rnmf_in_initialize header files are included first, as the rnmf algorithm consists of many interconnected .c files in main fumction. Other header files have been added for time analysis and for general functions to work. UART configuration is done to receive data from UART. The target and clutter variables are defined globally. To read radar data from UART, the code written in section 3 is performed under the data_receive function.

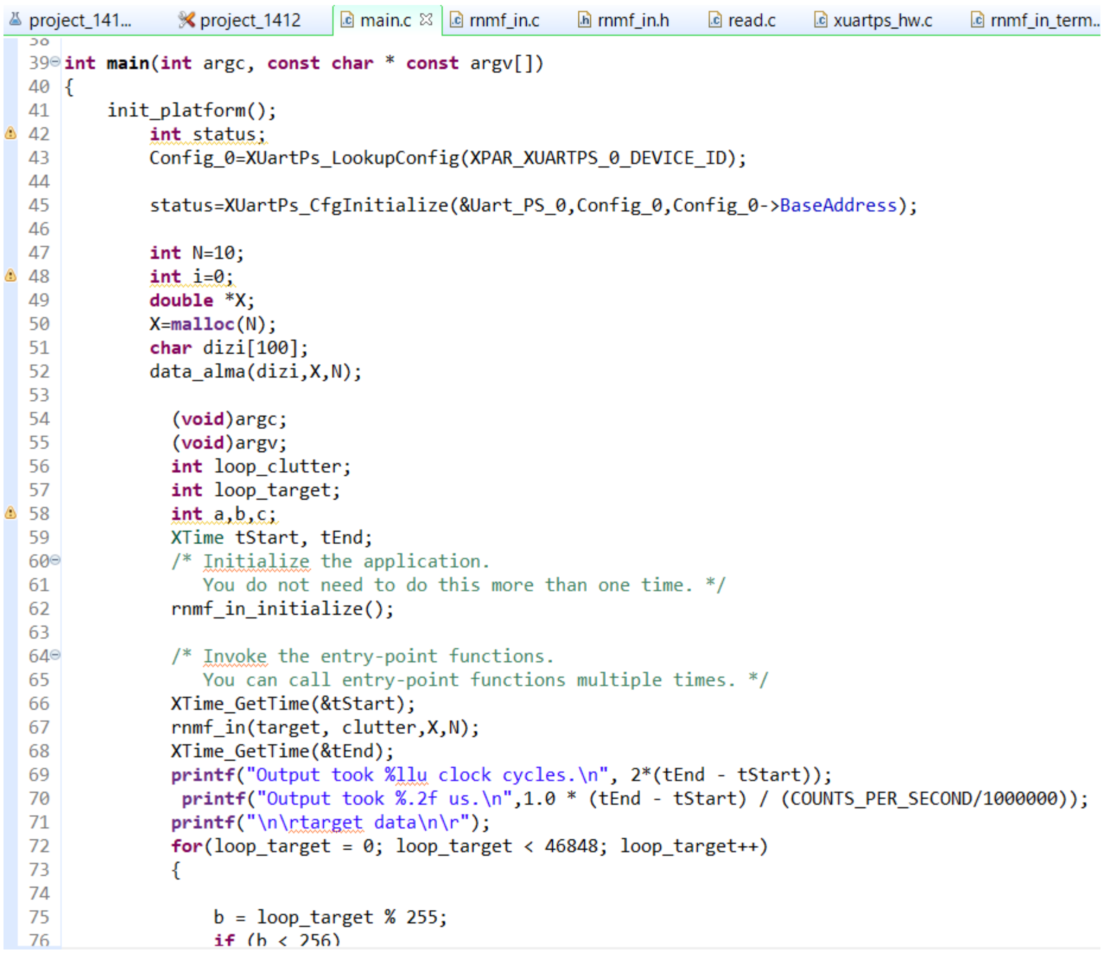
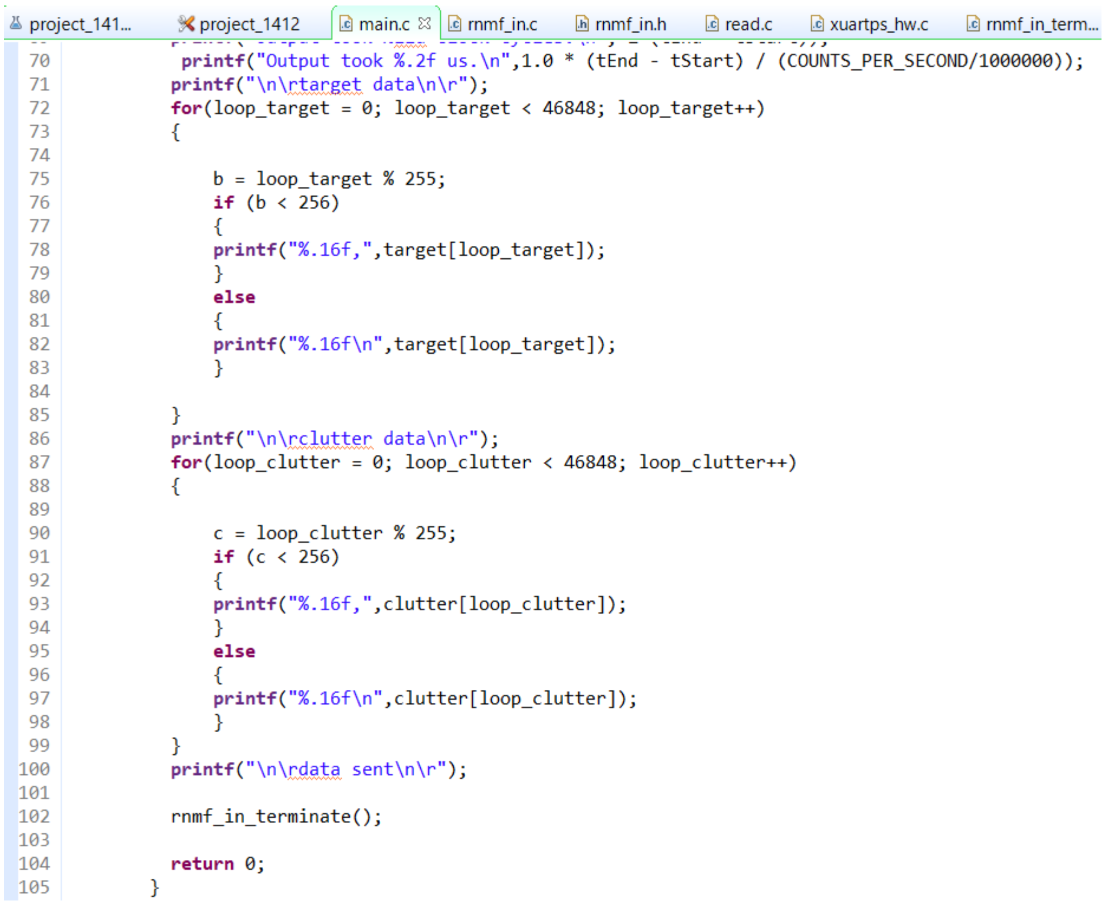

```
project_141...   project_1412   main.c   rnmf_in.c   rnmf_in.h   read.c   xuartps_hw.c   rnmf_in_term..

int main(int argc, const char * const argv[])
{
    init_platform();
        int status;
        Config_0=XUartPs_LookupConfig(XPAR_XUARTPS_0_DEVICE_ID);

        status=XUartPs_CfgInitialize(&Uart_PS_0,Config_0,Config_0->BaseAddress);

        int N=10;
        int i=0;
        double *X;
        X=malloc(N);
        char dizi[100];
        data_alma(dizi,X,N);

          (void)argc;
          (void)argv;
          int loop_clutter;
          int loop_target;
          int a,b,c;
          XTime tStart, tEnd;
          /* Initialize the application.
             You do not need to do this more than one time. */
          rnmf_in_initialize();

          /* Invoke the entry-point functions.
             You can call entry-point functions multiple times. */
          XTime_GetTime(&tStart);
          rnmf_in(target, clutter,X,N);
          XTime_GetTime(&tEnd);
          printf("Output took %llu clock cycles.\n", 2*(tEnd - tStart));
           printf("Output took %.2f us.\n",1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000));
          printf("\n\rtarget data\n\r");
          for(loop_target = 0; loop_target < 46848; loop_target++)
          {

              b = loop_target % 255;
              if (b < 256)
```

**Figure 4.5:** RNMF Vitis Main C Code (39-76)

In Figure 4.5, the values to be used are initialized. With the data_receive function, data is read from the radar. With the rnmf_in function, the code in which the RNMF algorithm is written is processed. With the XTime_GetTime function, it is calculated how long the code is processed and in the clock cycle.

The target variable obtained from the processing of the RNMF algorithm with the for loop and the clutter variable are printed to the terminal.



```
         printf("Output took %.2f us.\n",1.0 * (tEnd - tStart) / (COUNTS_PER_SECOND/1000000));
        printf("\n\rtarget data\n\r");
        for(loop_target = 0; loop_target < 46848; loop_target++)
        {

            b = loop_target % 255;
            if (b < 256)
            {
            printf("%.16f,",target[loop_target]);
            }
            else
            {
            printf("%.16f\n",target[loop_target]);
            }

        }
        printf("\n\rclutter data\n\r");
        for(loop_clutter = 0; loop_clutter < 46848; loop_clutter++)
        {

            c = loop_clutter % 255;
            if (c < 256)
            {
            printf("%.16f,",clutter[loop_clutter]);
            }
            else
            {
            printf("%.16f\n",clutter[loop_clutter]);
            }
        }
        printf("\n\rdata sent\n\r");

        rnmf_in_terminate();

        return 0;
    }
```

**Figure 4.6:** RNMF Vitis Main C Code (76-105)

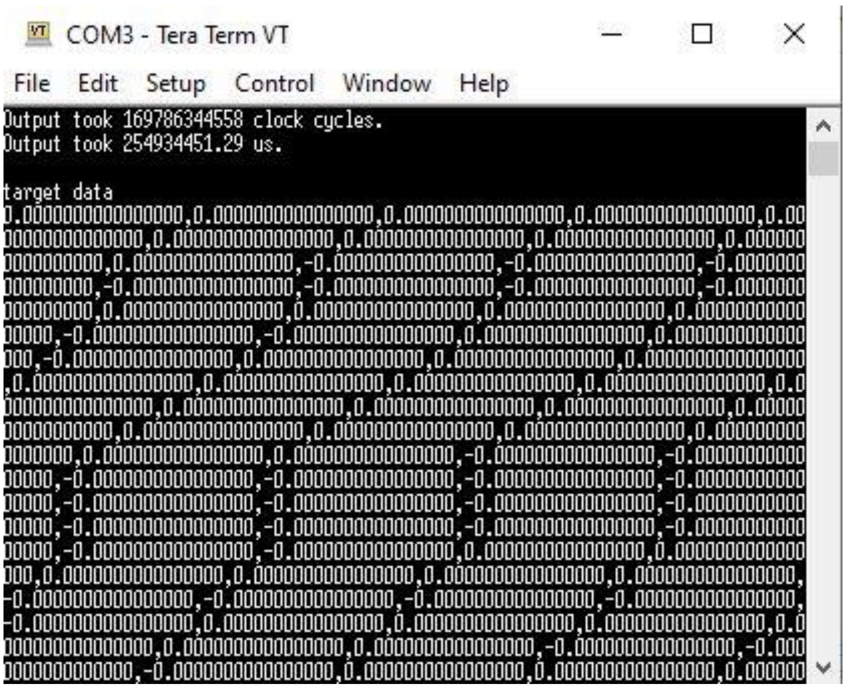
```
#include <math.h>
#include <string.h>
#include "rnmf_in.h"
#include "sqrt.h"
#include "sum.h"
#include "sign.h"
#include "abs.h"

void rnmf_in(float target[46848], float clutter[46848],float X[46848],int
N)
{
  int i;
  float W[256];

  static const float dv0[256] = { 0.80747046945450007, 0.26463438417187…}
  float H[183];
  static const float dv1[183] = { 0.197970832005718, 0.624752290113142…}
  int iter;
  int i0;
  int i1;
  static float varargin_2[46848];
```

```c
  int target_tmp;
  float norms;

  static float z1[46848];
  float b;
  float MSXHt[256];
  float MWtSX[183];

  memcpy(&W[0], &dv0[0], sizeof(float) << 8);
  memcpy(&H[0], &dv1[0], 183U * sizeof(float));
  for (iter = 0; iter < 10000; iter++) {
    for (i0 = 0; i0 < 256; i0++) {
      for (i1 = 0; i1 < 183; i1++) {
        target_tmp = i0 + (i1 << 8);
        target[target_tmp] = X[target_tmp] - W[i0] * H[i1];
      }
    }

    b_abs(target, varargin_2);
    for (target_tmp = 0; target_tmp < 46848; target_tmp++) {
      norms = varargin_2[target_tmp] - 0.00015;
      varargin_2[target_tmp] -= 0.00015;
      z1[target_tmp] = fmax(0.0, norms);
    }

    b_sign(target);
    for (i0 = 0; i0 < 46848; i0++) {
      target[i0] *= z1[i0];
    }

    for (target_tmp = 0; target_tmp < 256; target_tmp++) {
      norms = 0.0;
      b = 0.0;
      for (i0 = 0; i0 < 183; i0++) {
        i1 = target_tmp + (i0 << 8);
        norms += target[i1] * H[i0];
        b += X[i1] * H[i0];
      }

      norms -= b;
      MSXHt[target_tmp] = -norms;
      if (norms > 0.0) {
        MSXHt[target_tmp] = 0.0;
      }
    }
    norms = 0.0;
    for (i0 = 0; i0 < 183; i0++) {
      norms += H[i0] * H[i0];
    }

    for (target_tmp = 0; target_tmp < 256; target_tmp++) {
      W[target_tmp] = MSXHt[target_tmp] * W[target_tmp] /
                      fmax(W[target_tmp] * norms, 1.0E-20);
    }
```

```
for (target_tmp = 0; target_tmp < 183; target_tmp++) {
      MWtSX[target_tmp] = 0.0;
      norms = 0.0;
      b = 0.0;
      for (i0 = 0; i0 < 256; i0++) {
        i1 = i0 + (target_tmp << 8);
        norms += W[i0] * target[i1];
        b += W[i0] * X[i1];
      }

      norms -= b;
      MWtSX[target_tmp] = -norms;
      if (norms > 0.0) {
        MWtSX[target_tmp] = 0.0;
      }
    }

    norms = 0.0;
    for (i0 = 0; i0 < 256; i0++) {
      norms += W[i0] * W[i0];
    }

    for (target_tmp = 0; target_tmp < 183; target_tmp++) {
      H[target_tmp] = MWtSX[target_tmp] * H[target_tmp] / fmax(norms *
        H[target_tmp], 1.0E-20);
    }

    for (i0 = 0; i0 < 256; i0++) {
      MSXHt[i0] = W[i0] * W[i0];
    }

    norms = sum(MSXHt);
    b_sqrt(&norms);
    b = 1.0 / norms;

    for (i0 = 0; i0 < 256; i0++) {
      W[i0] *= b;
    }

    for (i0 = 0; i0 < 183; i0++) {
      H[i0] *= norms;
    }
  }

  for (i0 = 0; i0 < 256; i0++) {
    for (i1 = 0; i1 < 183; i1++) {
      clutter[i0 + (i1 << 8)] = W[i0] * H[i1];
    }
  }

}
```

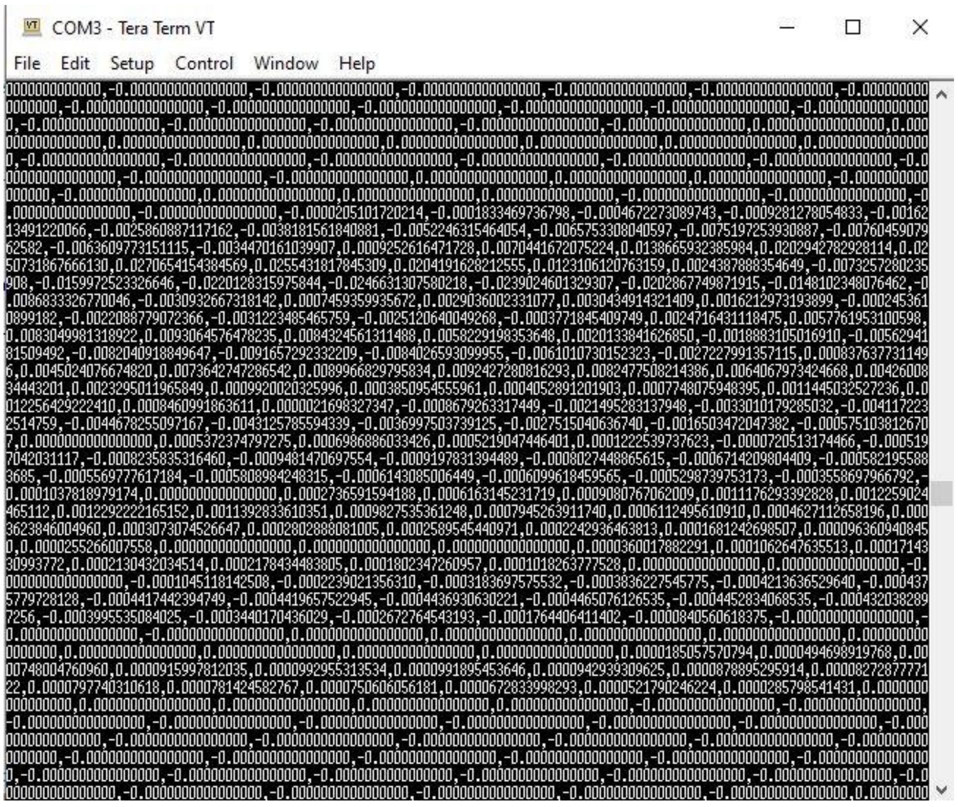
**Figure 4.7:** RNMF Algorithm Vitis C Code (rnmf_in.c)

The code written in accordance with the RNMF algorithm explained in section 4.1 is in figure 4.7.



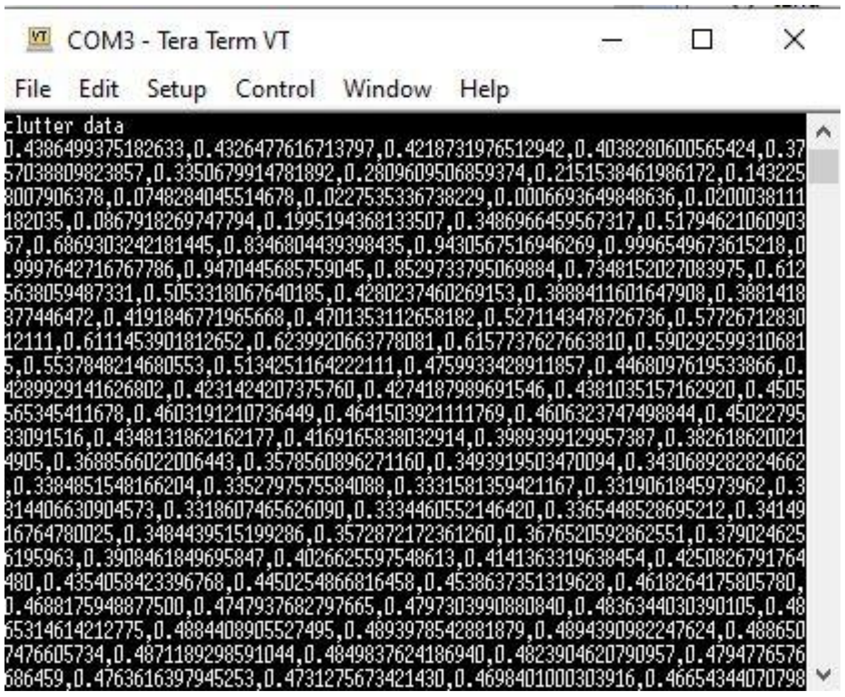
```
Output took 169786344558 clock cycles.
Output took 254934451.29 us.

target data
0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.00
00000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.000000
0000000000,0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000
000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000
000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.00000000000
00000,-0.0000000000000000,-0.0000000000000000,0.0000000000000000,0.0000000000000
000,-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000
,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0
000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.00000
00000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.000000000
0000000,0.0000000000000000,0.0000000000000000,-0.0000000000000000,-0.00000000000
00000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.00000000000
00000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.00000000000
00000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.00000000000
00000,-0.0000000000000000,-0.0000000000000000,0.0000000000000000,0.0000000000000
000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,
-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,
-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0
000000000000000,0.0000000000000000,0.0000000000000000,-0.0000000000000000,-0.000
0000000000000,-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.000000
```

Figure 4.8: Time Analysis Terminal Result



```
0000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.000000000
0000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000
0,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,0.0000000000000000,0.000
0000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000
0,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0
000000000000000,-0.0000000000000000,-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,-0.000000000
000000,-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0
.0000000000000000,-0.0000000000000000,-0.0000205101720214,-0.0001833469736798,-0.0004672273089743,-0.0009281278054833,-0.00162
13491220066,-0.0025860887117162,-0.0038181561840881,-0.0052246315464054,-0.0065753308040597,-0.0075197253930887,-0.00760459079
62582,-0.0063609773151115,-0.0034470161039907,0.0009252616471728,0.0070441672075224,0.0138665932385984,0.0202942782928114,0.02
5073186766130,0.0270654154384569,0.0255431817845309,0.0204191628212555,0.0123106120763159,0.0024387888354649,-0.0073257280235
908,-0.0159972523326646,-0.0220128315975844,-0.0246631307580218,-0.0239024601329307,-0.0202867749871915,-0.0148102348076462,-0
.0086833326770046,-0.0030932667318142,0.0007459359935672,0.0029036002331077,0.0030434914321409,0.0016212973193899,-0.000245361
0899182,-0.0022088779072366,-0.0031223485465759,-0.0025120640049268,-0.0003771845409749,0.0024716431118475,0.0057761953100598,
0.0083049981318922,0.0093064576478235,0.0084324561311488,0.0058229198353648,0.0020133841626850,-0.0018883105016910,-0.00562941
81509492,-0.0082040918849647,-0.0091657292332209,-0.0084026593099955,-0.0061010730152323,-0.0027227991357115,0.00083763773149
6,0.0045024076674820,0.0073642747286542,0.0089966829795834,0.0092427280816293,0.0082477508214386,0.0064067973424668,0.00426008
34443201,0.0023295011965849,0.0009920020325996,0.0003850954555961,0.0004052891201903,0.0007748075948395,0.0011445032527236,0.0
012256429222410,0.0008460991863611,0.0000021698327347,-0.0008679263317449,-0.0021495283137948,-0.0033010179285032,-0.004117223
2514759,-0.0044678255097167,-0.0043125785594339,-0.0036997503739125,-0.0027515040636740,-0.0016503472047382,-0.00057510381267
7,0.0000000000000000,0.0005372374797275,0.0006986886033426,0.0005219047446401,0.0001222539737623,-0.0000720513174466,-0.000519
7042031117,-0.0008235835316460,-0.0009481470697554,-0.0009197831394489,-0.0008027448865615,-0.0006714209804409,-0.000582195588
3685,-0.0005569777617184,-0.0005808984248315,-0.0006143085006449,-0.0006099618459565,-0.0005298739753173,-0.0003558697966792,-
0.0001037818979174,0.0000000000000000,0.0002736591594188,0.0006163145231719,0.0009080767062009,0.0011176293392828,0.0012259024
465112,0.0012292222165152,0.0011392833610351,0.0009827535361248,0.0007945263911740,0.0006112495610910,0.0004627112658196,0.000
3623846004960,0.0003073074526647,0.0002802888081005,0.0002589545440971,0.0002242936463813,0.0001681242698507,0.00009636094084
50,0.0000255266007558,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000360017882291,0.0001062647635513,0.00017143
30993772,0.0002130432034514,0.0002178434483805,0.0001802347260957,0.0001018263777528,0.0000000000000000,0.0000000000000000,-0.
0000000000000000,-0.0001045118142508,-0.0002239021356310,-0.0003183697575532,-0.0003836227545775,-0.0004213636529640,-0.000437
5779728128,-0.0004417442394749,-0.0004419657522945,-0.0004436930630221,-0.0004465076126535,-0.0004452834068535,-0.000432038289
7256,-0.0003995535084025,-0.0003440170436029,-0.0002672764543193,-0.0001764406411402,-0.0000840560618375,-0.0000000000000000,-
0.0000000000000000,-0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.00000000
0000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000185057570794,0.0000494698919768,0.00
00748004760960,0.0000915997812035,0.0000992955313534,0.0000991895453646,0.0000942939309625,0.0000878895295914,0.00008272877771
22,0.0000797740310618,0.0000781424582767,0.0000750606056181,0.0000672833998293,0.0000521790246224,0.0000285798541431,0.0000000
000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,0.0000000000000000,-0.0000000000000000,-0.0000000000000000,
-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.000
0000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.000000000
0000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000
0,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0
000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,-0.0000000000000000,0.00000000
```

**Figure 4.9:** Target Data Removed Form Clutter by RNMF Algorithm



```
clutter data
0.4386499375182633,0.4326477616713797,0.4218731976512942,0.4038280600565424,0.37
57038809823857,0.3350679914781892,0.2809609506859374,0.2151538461986172,0.143225
8007906378,0.0748284045514678,0.0227535336738229,0.0006693649848636,0.0200038111
182035,0.0867918269747794,0.1995194368133507,0.3486966459567317,0.51794621060903
67,0.6869303242181445,0.8346804439398435,0.9430567516946269,0.9996549673615218,0
.9997642716767786,0.9470445685759045,0.8529733795069884,0.7348152027083975,0.612
5638059487331,0.5053318067640185,0.4280237460269153,0.3888411601647908,0.3881418
377446472,0.4191846771965668,0.4701353112658182,0.5271143478726736,0.5772671283O
12111,0.6111453901812652,0.6239920663778081,0.6157737627663810,0.590292599310681
5,0.5537848214680553,0.5134251164222111,0.4759933428911857,0.4468097619533866,0.
4289929141626802,0.4231424207375760,0.4274187989691546,0.4381035157162920,0.4505
565345411678,0.4603191210736449,0.4641503921111769,0.4606323747498844,0.45022795
33091516,0.4348131862162177,0.4169165838032914,0.3989399129957387,0.382618620021
4905,0.3688566022006443,0.3578560896271160,0.3493919503470094,0.3430689282824662
,0.3384851548166204,0.3352797575584088,0.3331581359421167,0.3319061845973962,0.3
314406630904573,0.3318607465626090,0.3334460552146420,0.3365448528695212,0.34149
16764780025,0.3484439515199286,0.3572872172361260,0.3676520592862551,0.379024625
6195963,0.3908461849695847,0.4026625597548613,0.4141363319638454,0.4250826791764
480,0.4354058423396768,0.4450254866816458,0.4538637351319628,0.4618264175805780,
0.4688175948877500,0.4747937682797665,0.4797303990880840,0.4836344030390105,0.48
65314614212775,0.4884408905527495,0.4893978542881879,0.4894390982247624,0.488650
7476605734,0.4871189298591044,0.4849837624186940,0.4823904620790957,0.4794776576
686459,0.4763616397945253,0.4731275673421430,0.4698401000303916,0.46654344070798
```

**Figure 4.10:** Clutter Data

The target data removed from the clutter by the RNMF algorithm, in Figure 4.9, and the clutter-generating clutter data, in Figure 4.10, are observed from the terminal.

# 5. HARDWARE IMPLEMENTATION OF RNMF ALGORITHM

In this section, the hardware implementation of the line of code, which includes matrix multiplication and subtraction, which was previously done in software in the RNMF code, will be explained. Therefore, a custom IP is designed for the following line of code.

$$\text{target[target\_tmp]} = \text{X[target\_tmp]} - \text{W}[i_0] * \text{H}[i_1]$$

Block design is created with the designed IP and Zynq-7000 processor. After the created block design is synthesized, implemented, and the bitstream file is produced, the hardware is exported and the data is sent to the appropriate registers in the line where the operation is performed in the Vitis part.

## 5.1 Implementation of Floating-Point Numbers on Vivado

Since Verilog or VHDL digital hardware languages operate on bits, it is not easy to do arithmetic with floating-point numbers. Actually, in Vivado, arithmetic operations can be done by defining floating point numbers in real data type. But the real data type cannot be synthesized. This method cannot be used for this project, as synthesis and implementation processes are required after the design is created. Therefore, a special code is required for the arithmetic operations of floating-point numbers. Vivado defines floating point numbers in IEEE-754 format. In this case, there are two methods for dealing with floating points. The first is to write the Verilog code that will multiply and subtract in accordance with the IEEE-754 format by including the float_pkg.all and float_generic_pkg.all libraries, and the other is to create a design using the Float IP Generator (7.1) custom IP already available in the Vivado IP Catalog [24].

### 5.1.1 IEEE 754 Format floating point numbers arithmetic calculations

Numerous applications, including signal processing, scientific computations, etc., make extensive use of floating-point math and calculations using it. Because there is no requirement for large dynamic number ranges or to scale the values, floating point arithmetic techniques are simpler than others. However, due to the restricted number of circuits, implementing floating points on hardware is rather difficult. Researchers are instructed on how to implement the IEEE-754 floating point standard since it is essential to processor performance [21].

Calculations using binary integers are done by supposing a certain location for the comma. The accuracy of the number is really altered by moving the comma. As a result, binary number format is offered for values with various sensitivities. These sensitivities are guaranteed to be within a specific standard by the IEEE-754 standard [26,29].

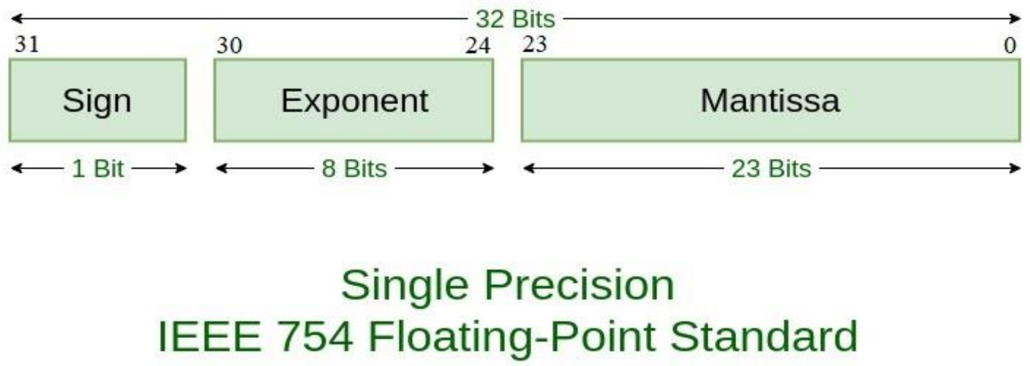


**Figure 5.1:** Single Precision IEEE-754 Floating-Point Standard

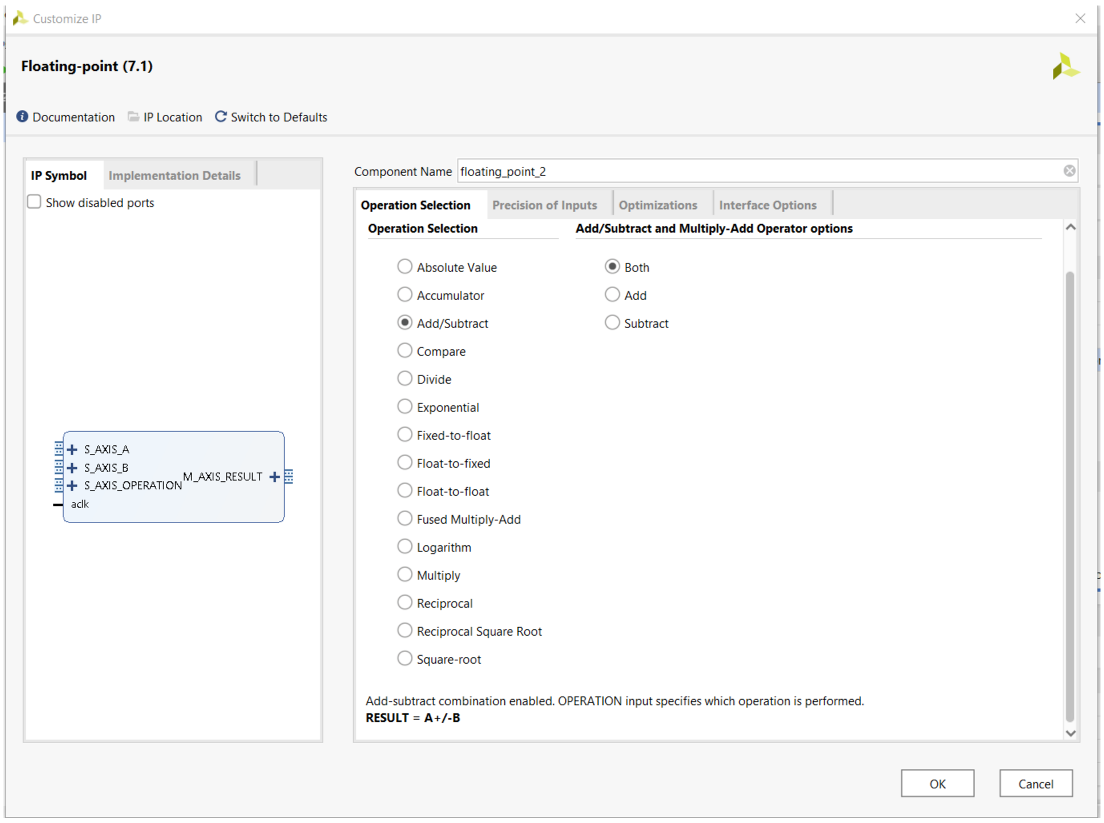
According to this standard, floating point numbers consist of three parts. The most important bit, the Sign part, indicates whether the number is negative or positive, the exponent part indicates the biased part of the decimal part of the number, and the mantissa part indicates the decimal part of the number. Single precision is equal to the float type in the programming language, has a maximum precision of 6 digits, and holds a 32-bit number. Double precision is equal to the double type in the software language, it has a maximum precision of 15 digits and holds a 64-bit number. Sign bit is one bit in both double precision and single precision. The exponent part is 8 bits in single precision, 11 bits in double precision, and the mantissa part is 23 bits in single precision and 52 bits in double precision [23].

$$Floating - point\ number\ = (-1)^{s} * m * 2^{e}$$

With this formula, the multiplication of two numbers in floating-point numbers is performed with the following steps. Sign bits XOR, mantissa parts are multiplied and the exponent exponents are added to get the result. In floating-point numbers, the sum of two numbers is performed by shifting the exponent of the smaller number until it equals the exponent of the larger number, and then adding the two values [27,28]. VHDL or Verilog code can be written according to the algorithm described to perform these operations in Vivado. Another method suitable for this format is to include the float_pkg.all and float_generic_pkg.all libraries and perform operations by generating var float x. But since the libraries written in this method are compatible with Vivado 2008 version, it is necessary to update the libraries [24,25].

### 5.1.2 Floating-Point IP Generator (7.1) custom IP design

Floating Point IP, available in Vivado IP Catalog → Math Functions, is used for floating point arithmetic operations. The floating point AXI IP has data ports named


Customize IP

Floating-point (7.1)

Documentation   IP Location   Switch to Defaults

IP Symbol   Implementation Details

Show disabled ports

Component Name floating_point_2

Operation Selection   Precision of Inputs   Optimizations   Interface Options

Operation Selection

Absolute Value
Accumulator
[✓ Add/Subtract]
Compare
Divide
Exponential
Fixed-to-float
Float-to-fixed
Float-to-float
Fused Multiply-Add
Logarithm
Multiply
Reciprocal
Reciprocal Square Root
Square-root

Add/Subtract and Multiply-Add Operator options

[✓ Both]
Add
Subtract

Add-subtract combination enabled. OPERATION input specifies which operation is performed.
**RESULT = A+/-B**

OK   Cancel

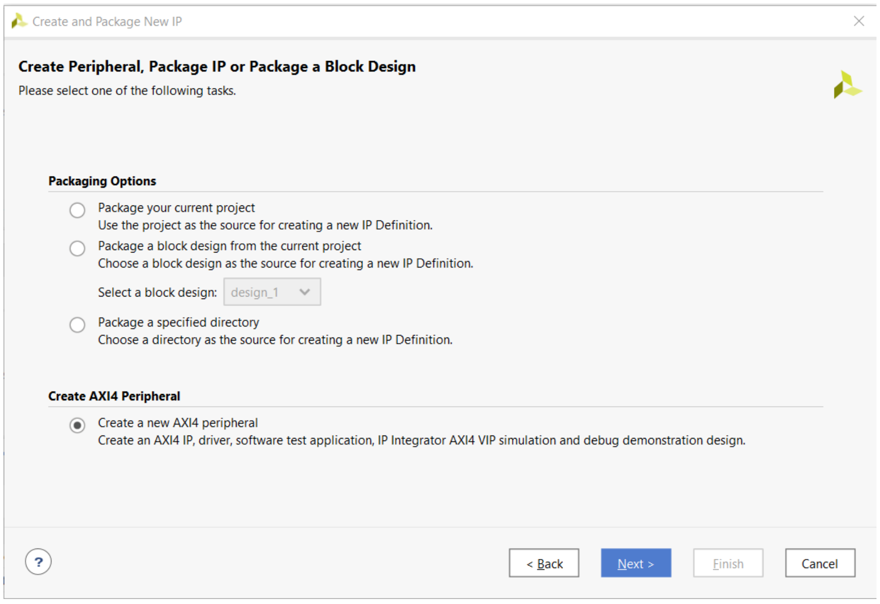
**Figure 5.2:** Floating Point (7.1) IP

S_AXIS_A and S_AXIS_B to be used in arithmetic operations, a port named S_AXIS_OPERATION where the operation to be performed is selected, an operation result port named M_AXIS_RESULT, asynchronous clock signal called aclk and asynchronous reset signal called aresetn. This IP performs arithmetic operations using DSP blocks. Input type can be changed from Precision of Inputs tab to half precision, single precision, double precision or custom precision. In addition, how much delay the process will have, the reset pin and the ready pin can also be configured. Because the floating point IPsi uses DSP blocks, it performs transactions quickly. Therefore, in this project, floating point transactions were made using this IP [22].

## 5.2 Custom IP Design of Floating-Point Arithmetic in Vivado

In this section, the custom ip designed to perform the arithmetic operation performed in the software will be explained. The line that performs the T = X - (W*H) arithmetic operation by taking a float value from the X, W and H arrays in each loop within the for loopin the software will be made in the hardware part.

First of all, the project is opened in Vivado. Click on Tools → Create and Package New IP button.


Create and Package New IP

**Create Peripheral, Package IP or Package a Block Design**

Please select one of the following tasks.

**Packaging Options**

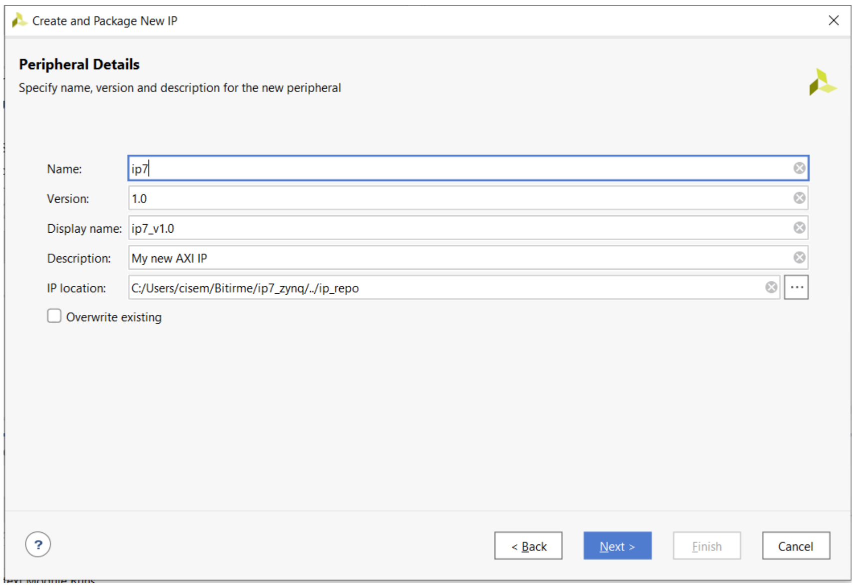
- ○ Package your current project
  Use the project as the source for creating a new IP Definition.
- ○ Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
  Select a block design: design_1
- ○ Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

**Create AXI4 Peripheral**

- ● Create a new AXI4 peripheral
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

**Figure 5.3:** IP Package Type Selection

Since the Floating-Point has an IP AXI4-Stream interface, IP Package type is selected as AXI4 Peripheral.


Create and Package New IP

**Peripheral Details**

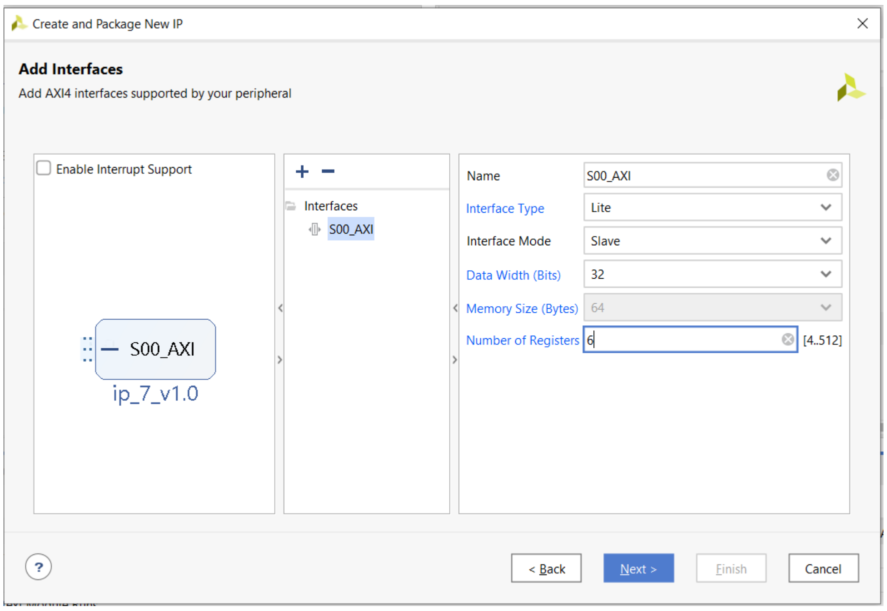
Specify name, version and description for the new peripheral

| | |
|---|---|
| Name: | ip7 |
| Version: | 1.0 |
| Display name: | ip7_v1.0 |
| Description: | My new AXI IP |
| IP location: | C:/Users/cisem/Bitirme/ip7_zynq/../ip_repo |

☐ Overwrite existing

**Figure 5.4:** Naming the IP Packet

The IP packet is named ip7 and the place where it will be saved is chosen as the folder where the final block design will be created.


Create and Package New IP

**Add Interfaces**

Add AXI4 interfaces supported by your peripheral

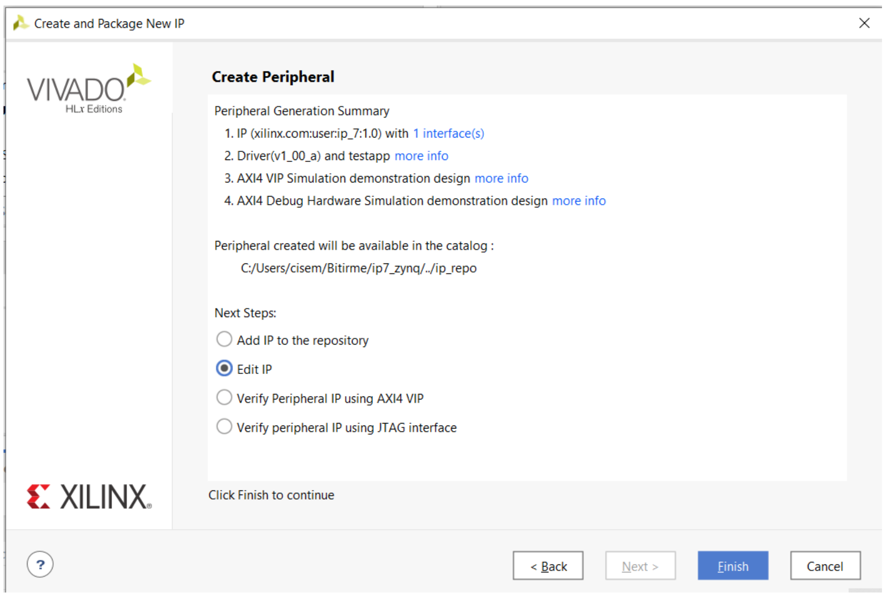
☐ Enable Interrupt Support

Interfaces
- S00_AXI

| | |
|---|---|
| Name | S00_AXI |
| Interface Type | Lite |
| Interface Mode | Slave |
| Data Width (Bits) | 32 |
| Memory Size (Bytes) | 64 |
| Number of Registers | 6 [4..512] |

**Figure 5.5:** Configuring Interface Settings

Since the data will come from the Zynq processor, the mode has been selected as the slave for this interface and the number of registers has been updated as 6 due to the variables X, W, H, T, valid and ready.


Create and Package New IP

**Create Peripheral**

Peripheral Generation Summary
1. IP (xilinx.com:user:ip_7:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

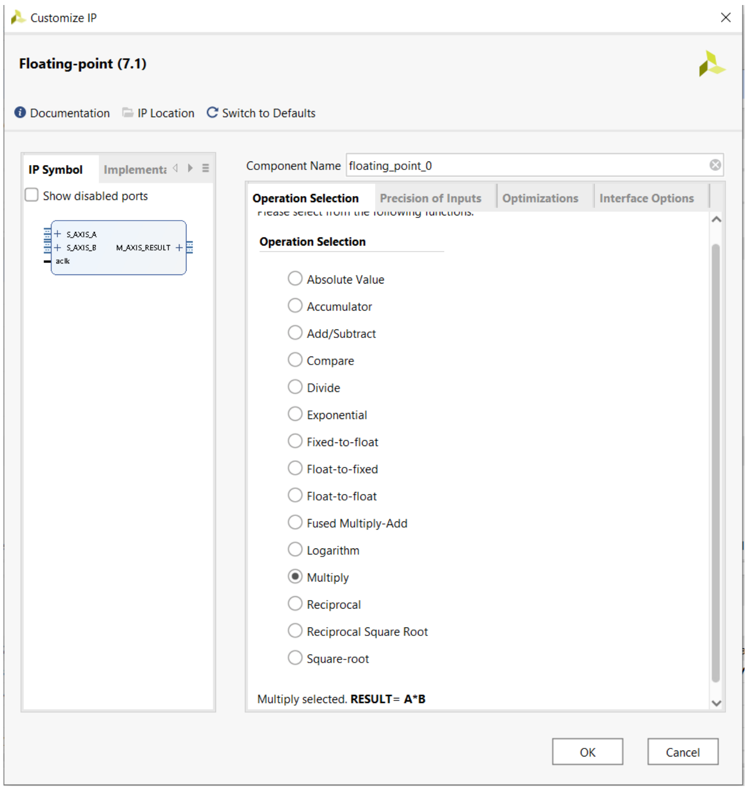
Peripheral created will be available in the catalog :
C:/Users/cisem/Bitirme/ip7_zynq/../ip_repo

Next Steps:
- ○ Add IP to the repository
- ● Edit IP
- ○ Verify Peripheral IP using AXI4 VIP
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

**Figure 5.6:** Edit IP

Since the registers will be edited and other operations will be coded in the created IP, the Edit IP option is selected. In the project that opens, we first add the floating-point IPs.


Customize IP

**Floating-point (7.1)**

Documentation | IP Location | Switch to Defaults

IP Symbol | Implement

☐ Show disabled ports

Component Name: floating_point_0

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Please select from the following functions.

**Operation Selection**

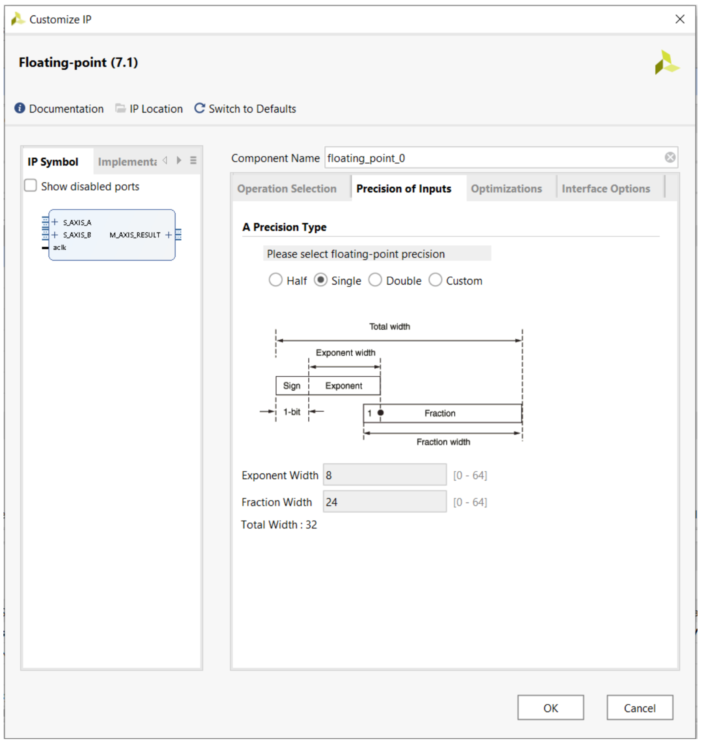
- ○ Absolute Value
- ○ Accumulator
- ○ Add/Subtract
- ○ Compare
- ○ Divide
- ○ Exponential
- ○ Fixed-to-float
- ○ Float-to-fixed
- ○ Float-to-float
- ○ Fused Multiply-Add
- ○ Logarithm
- [✓ Multiply]
- ○ Reciprocal
- ○ Reciprocal Square Root
- ○ Square-root

Multiply selected. **RESULT= A*B**

OK | Cancel

**Figure 5.7:** Operation Selection

First we add the IP that do the multiplication for the (W * H) operation.


Customize IP

**Floating-point (7.1)**

Documentation | IP Location | Switch to Defaults

IP Symbol | Implement

☐ Show disabled ports

Component Name: floating_point_0

Operation Selection | Precision of Inputs | Optimizations | Interface Options

**A Precision Type**

Please select floating-point precision

Half [✓ Single] Double Custom

Exponent Width: 8 [0 - 64]

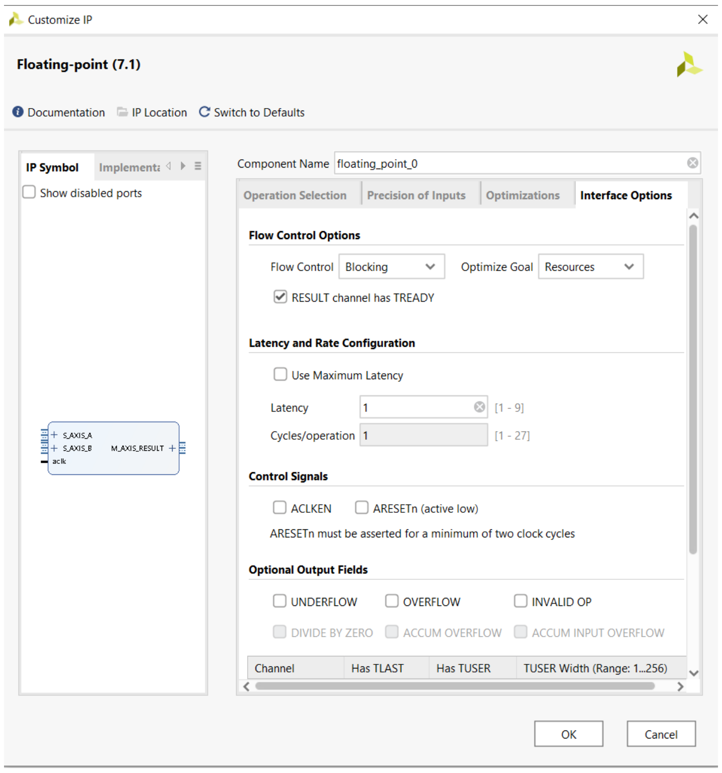
Fraction Width: 24 [0 - 64]

Total Width : 32

OK | Cancel

**Figure 5.8:** Selection of Precision of Input

Precision type is selected as single because float values will be multiplied.


Customize IP

Floating-point (7.1)

Documentation | IP Location | Switch to Defaults

IP Symbol | Implementa

☐ Show disabled ports

Component Name: floating_point_0

Operation Selection | Precision of Inputs | Optimizations | **Interface Options**

**Flow Control Options**

Flow Control: Blocking — Optimize Goal: Resources

☑ RESULT channel has TREADY

**Latency and Rate Configuration**

☐ Use Maximum Latency

Latency: 1 [1 - 9]

Cycles/operation: 1 [1 - 27]

**Control Signals**

☐ ACLKEN ☐ ARESETn (active low)

ARESETn must be asserted for a minimum of two clock cycles

**Optional Output Fields**

☐ UNDERFLOW ☐ OVERFLOW ☐ INVALID OP

☐ DIVIDE BY ZERO ☐ ACCUM OVERFLOW ☐ ACCUM INPUT OVERFLOW

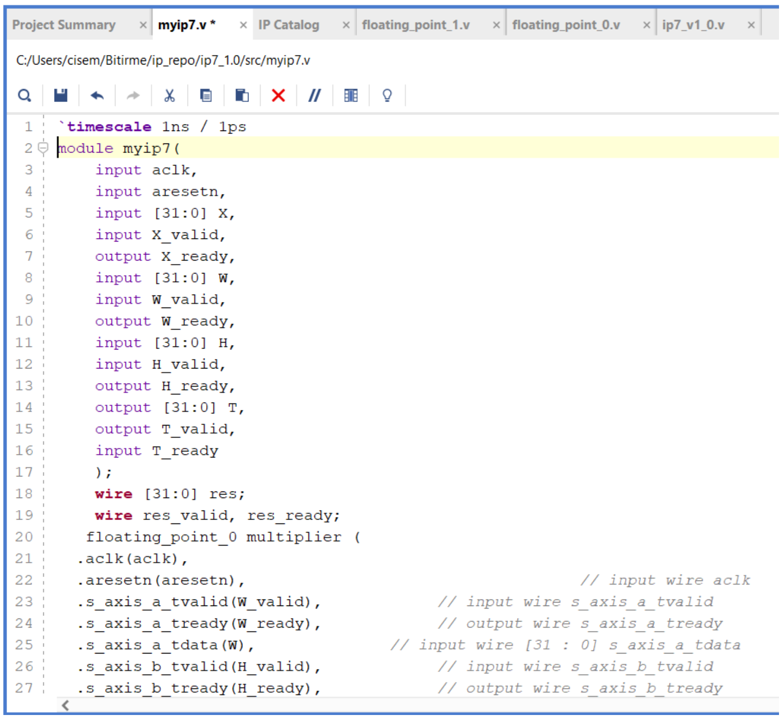
| Channel | Has TLAST | Has TUSER | TUSER Width (Range: 1..256) |
|---|---|---|---|

OK | Cancel

**Figure 5.9:** Interface Options Settings

A ready input is added to be able to observe the signs in the simulation. In order to minimize the delay, Latency is updated as 1 and reset pin is added. The same steps are repeated to add the subtract block by selecting subtract in the operator selection section.

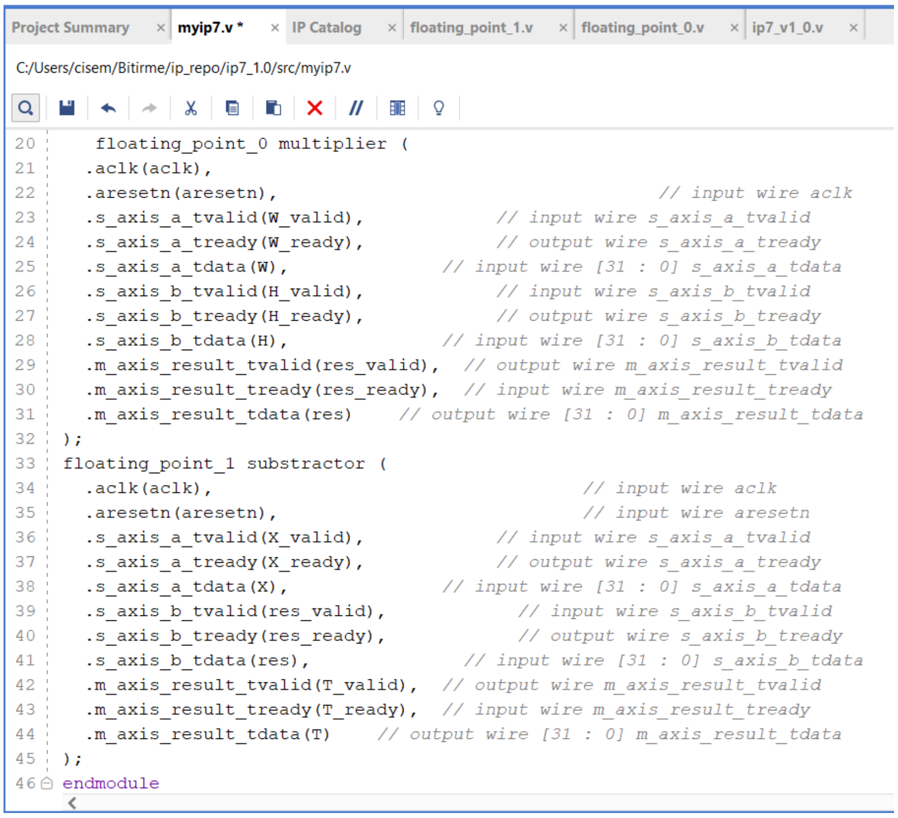
Then the added IP needs to be implemented into the code. For this, the Verilog code is copied from the floating_point_0.veo file from IP Sources → floating_point_0 → Instantiation Template.



```verilog
`timescale 1ns / 1ps
module myip7(
    input aclk,
    input aresetn,
    input [31:0] X,
    input X_valid,
    output X_ready,
    input [31:0] W,
    input W_valid,
    output W_ready,
    input [31:0] H,
    input H_valid,
    output H_ready,
    output [31:0] T,
    output T_valid,
    input T_ready
    );
    wire [31:0] res;
    wire res_valid, res_ready;
   floating_point_0 multiplier (
  .aclk(aclk),
  .aresetn(aresetn),                                // input wire aclk
  .s_axis_a_tvalid(W_valid),             // input wire s_axis_a_tvalid
  .s_axis_a_tready(W_ready),             // output wire s_axis_a_tready
  .s_axis_a_tdata(W),               // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(H_valid),             // input wire s_axis_b_tvalid
  .s_axis_b_tready(H_ready),             // output wire s_axis_b_tready
```

**Figure 5.10:** IP7 Main Verilog Code (1-27)

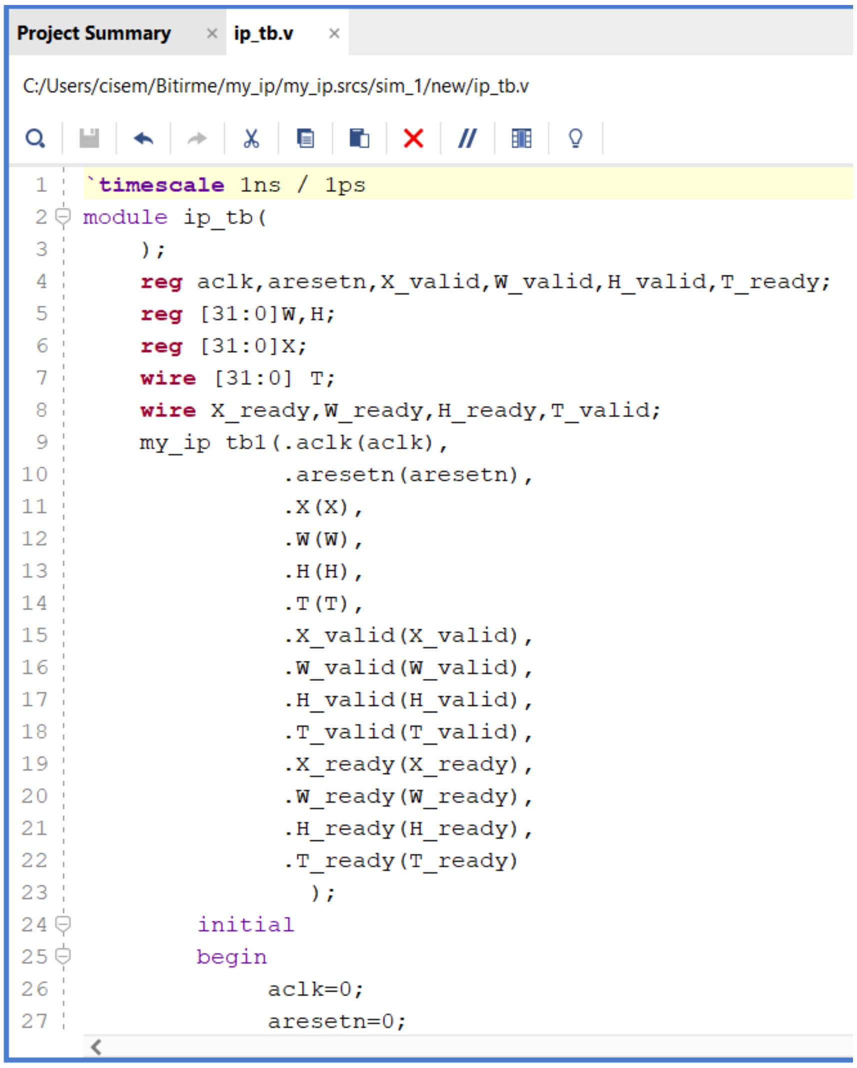
X, W, H are defined as input, T as output, and valid and ready variables are defined for these variables.



```verilog
   floating_point_0 multiplier (
  .aclk(aclk),
  .aresetn(aresetn),                          // input wire aclk
  .s_axis_a_tvalid(W_valid),             // input wire s_axis_a_tvalid
  .s_axis_a_tready(W_ready),             // output wire s_axis_a_tready
  .s_axis_a_tdata(W),               // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(H_valid),             // input wire s_axis_b_tvalid
  .s_axis_b_tready(H_ready),             // output wire s_axis_b_tready
  .s_axis_b_tdata(H),               // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(res_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(res_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(res)     // output wire [31 : 0] m_axis_result_tdata
);
floating_point_1 substractor (
  .aclk(aclk),                                // input wire aclk
  .aresetn(aresetn),                          // input wire aresetn
  .s_axis_a_tvalid(X_valid),             // input wire s_axis_a_tvalid
  .s_axis_a_tready(X_ready),             // output wire s_axis_a_tready
  .s_axis_a_tdata(X),               // input wire [31 : 0] s_axis_a_tdata
  .s_axis_b_tvalid(res_valid),             // input wire s_axis_b_tvalid
  .s_axis_b_tready(res_ready),             // output wire s_axis_b_tready
  .s_axis_b_tdata(res),             // input wire [31 : 0] s_axis_b_tdata
  .m_axis_result_tvalid(T_valid),  // output wire m_axis_result_tvalid
  .m_axis_result_tready(T_ready),  // input wire m_axis_result_tready
  .m_axis_result_tdata(T)     // output wire [31 : 0] m_axis_result_tdata
);
endmodule
```

**Figure 5.11:** IP7 Main Verilog Code (27-46)

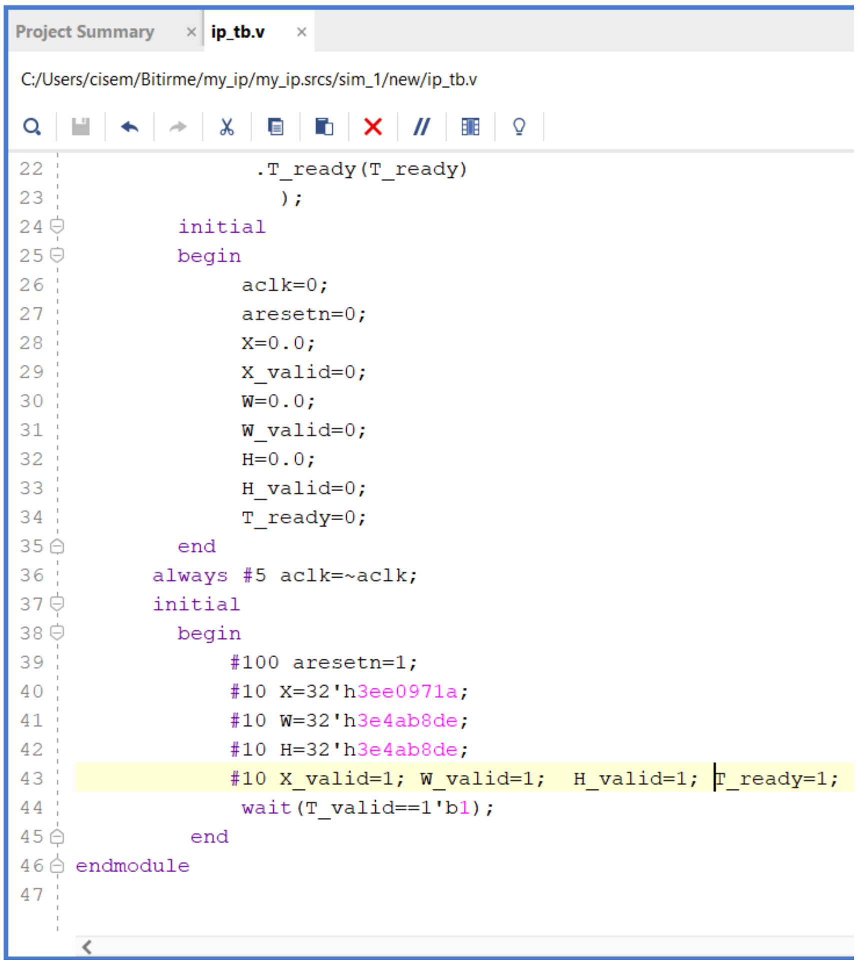
Since floating_point_0 makes multiplication block, W variable is directed to A data, H variable to B data and res variable to result data. Then, the variable X is directed to the A data of the floating_point_1 function, the res variable to the B data, and the T variable to the result data for subtraction. In order to test that the written code works, the testbench code is written and observed in the simulation.



```
`timescale 1ns / 1ps
module ip_tb(
    );
    reg aclk,aresetn,X_valid,W_valid,H_valid,T_ready;
    reg [31:0]W,H;
    reg [31:0]X;
    wire [31:0] T;
    wire X_ready,W_ready,H_ready,T_valid;
    my_ip tb1(.aclk(aclk),
              .aresetn(aresetn),
              .X(X),
              .W(W),
              .H(H),
              .T(T),
              .X_valid(X_valid),
              .W_valid(W_valid),
              .H_valid(H_valid),
              .T_valid(T_valid),
              .X_ready(X_ready),
              .W_ready(W_ready),
              .H_ready(H_ready),
              .T_ready(T_ready)
                );
        initial
        begin
            aclk=0;
            aresetn=0;
```

**Figure 5.12:** IP7 TestBench Verilog Code (1-27)

For the module defined in the main Verilog code, the inputs are reg and the outputs are defined as wire. The module in the main code is initialized.

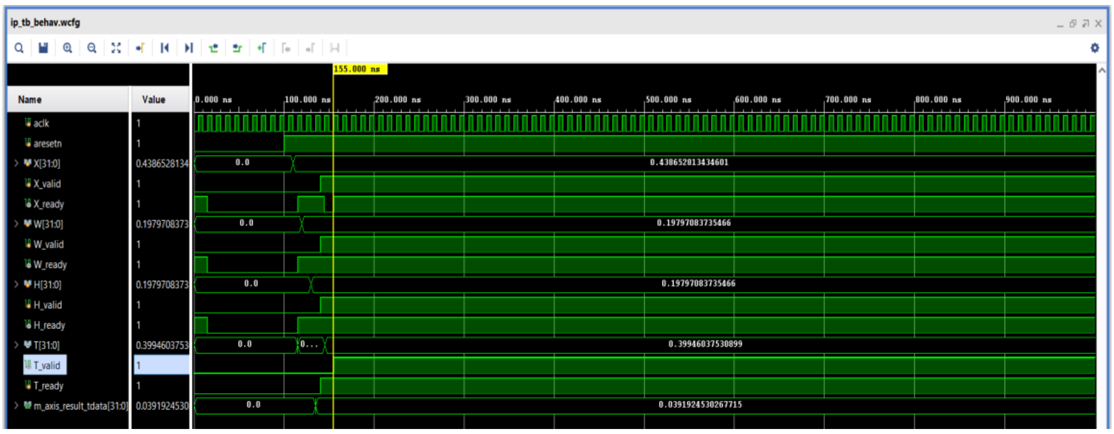


C:/Users/cisem/Bitirme/my_ip/my_ip.srcs/sim_1/new/ip_tb.v

```verilog
                  .T_ready(T_ready)
                    );
          initial
          begin
               aclk=0;
               aresetn=0;
               X=0.0;
               X_valid=0;
               W=0.0;
               W_valid=0;
               H=0.0;
               H_valid=0;
               T_ready=0;
          end
        always #5 aclk=~aclk;
        initial
          begin
              #100 aresetn=1;
              #10 X=32'h3ee0971a;
              #10 W=32'h3e4ab8de;
              #10 H=32'h3e4ab8de;
              #10 X_valid=1; W_valid=1;  H_valid=1; T_ready=1;
               wait(T_valid==1'b1);
           end
endmodule
```

**Figure 5.13:** IP7 Main Verilog Code (27-46)

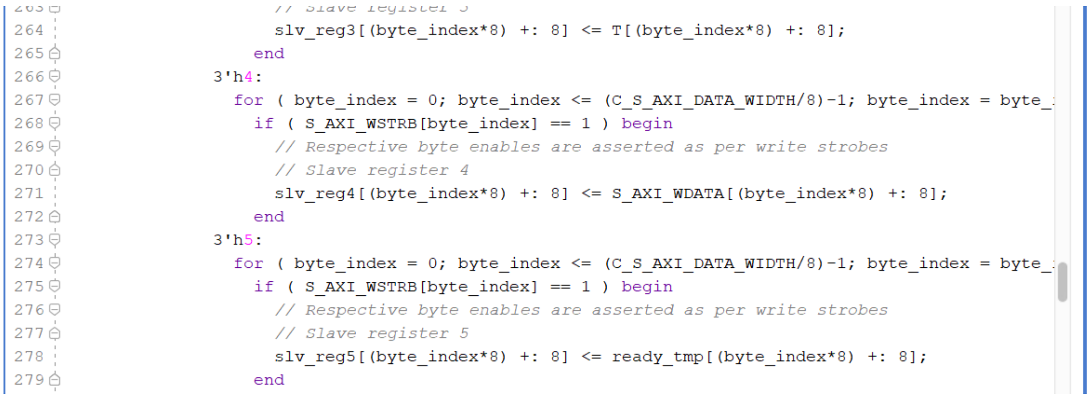
First, the clock signal, input signals, valid signals and reset signal are pulled to low. A clock pulse is created by changing the clock signal at 5ns intervals. When the simulation starts, the reset signal is pulled to high. Inputs are assigned float values. These values are oriented by converting from IEEE-754 floating point format to hexadecimal format. The variables X_valid, W_valid, H_valid and T_ready are set high so that the results of the operations are observed.



| Name | Value |
|---|---|
| aclk | 1 |
| aresetn | 1 |
| X[31:0] | 0.4386528134 |
| X_valid | 1 |
| X_ready | 1 |
| W[31:0] | 0.1979708373 |
| W_valid | 1 |
| W_ready | 1 |
| H[31:0] | 0.1979708373 |
| H_valid | 1 |
| H_ready | 1 |
| T[31:0] | 0.3994603753 |
| T_valid | 1 |
| T_ready | 1 |
| m_axis_result_tdata[31:0] | 0.0391924530 |

**Figure 5.14:** Simulation Results

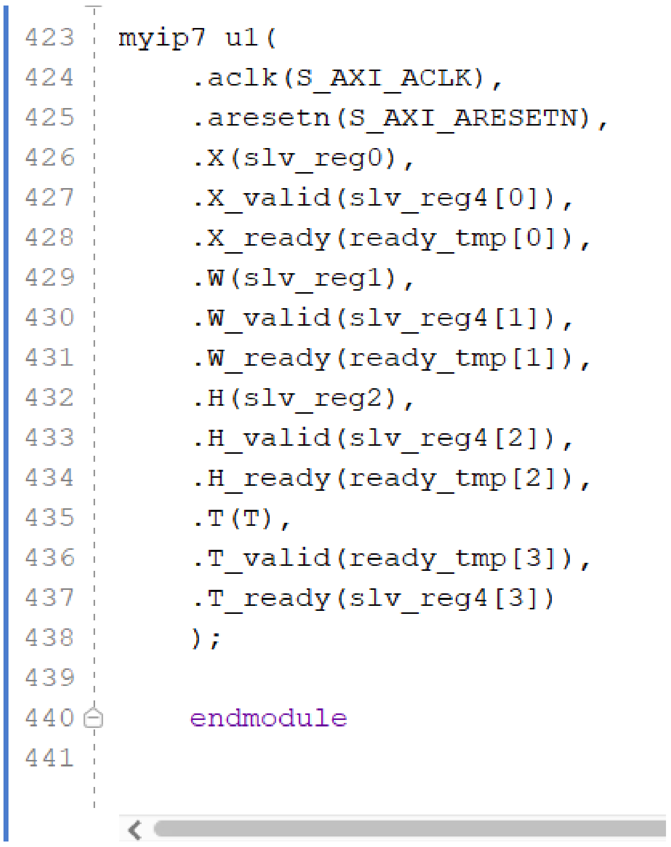
W is multiplied by H and written to the variable m_axis_result_data. This value is then subtracted from X. Simulation results with the sent values give correct results. Thus, it has been proven that the designed system works.



```
              // Slave register 3
              slv_reg3[(byte_index*8) +: 8] <= T[(byte_index*8) +: 8];
            end
        3'h4:
          for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_
            if ( S_AXI_WSTRB[byte_index] == 1 ) begin
              // Respective byte enables are asserted as per write strobes
              // Slave register 4
              slv_reg4[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
            end
        3'h5:
          for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_
            if ( S_AXI_WSTRB[byte_index] == 1 ) begin
              // Respective byte enables are asserted as per write strobes
              // Slave register 5
              slv_reg5[(byte_index*8) +: 8] <= ready_tmp[(byte_index*8) +: 8];
            end
```

**Figure 5.15:** Register Assignments

The variable T, which holds the process result, is assigned to register 3, and the variable ready_tmp, which holds the process result is ready, is assigned to register 5.



```
myip7 u1(
    .aclk(S_AXI_ACLK),
    .aresetn(S_AXI_ARESETN),
    .X(slv_reg0),
    .X_valid(slv_reg4[0]),
    .X_ready(ready_tmp[0]),
    .W(slv_reg1),
    .W_valid(slv_reg4[1]),
    .W_ready(ready_tmp[1]),
    .H(slv_reg2),
    .H_valid(slv_reg4[2]),
    .H_ready(ready_tmp[2]),
    .T(T),
    .T_valid(ready_tmp[3]),
    .T_ready(slv_reg4[3])
    );

    endmodule
```

**Figure 5.16:** Register Assignments Module

register 0 is directed to variable X, register 1 to variable W, register 2 to variable H, register 4 to valid variables. These register assignments are important when sending and receiving data in the Vitis interface.

All changes made are saved in the edit project and the IP is packaged. Currently designed IP is in use and operational.

## 5.3 Block Design of Project

A block design is created using the IP, designed in Section 5.2, and the Zynq-7000 processor. Since the designed IP is an AXI Peripheral, AXI Interconnect provides the connection between it and the Zynq-7000 processor. It comes automatically with the addition of Processor Reset System and AXI Interconnect Zynq-7000 processor. The frequency of the FCLK_CLK0 signal was changed to 10 MHz from the Clock Configuration → PL Fabric Clocks tab by double-clicking on the Zynq-7000 processor.

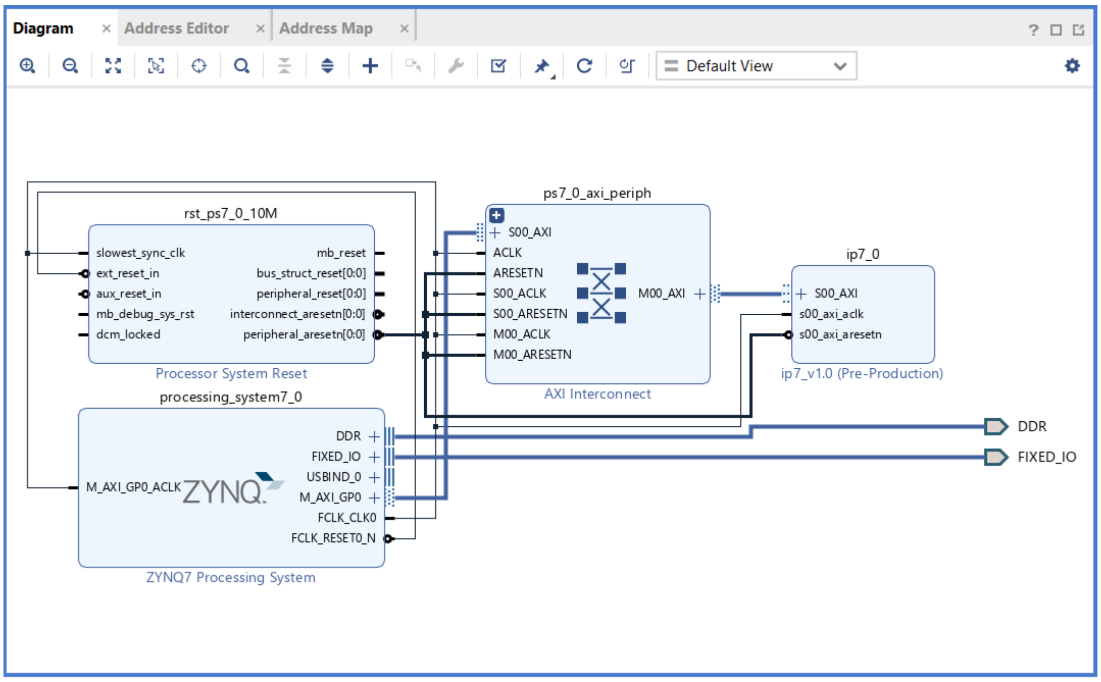


**Figure 5.17:** Block Design of Project

Output Products are produced after block design is evaluated. After the output products are successfully created for all blocks in the block design, HDL Wrapper is produced from the design. The steps of synthesis, implementation and bitstream file generation are done respectively. The designed hardware is exported with the Export Hardware option and a .xsa file is produced.

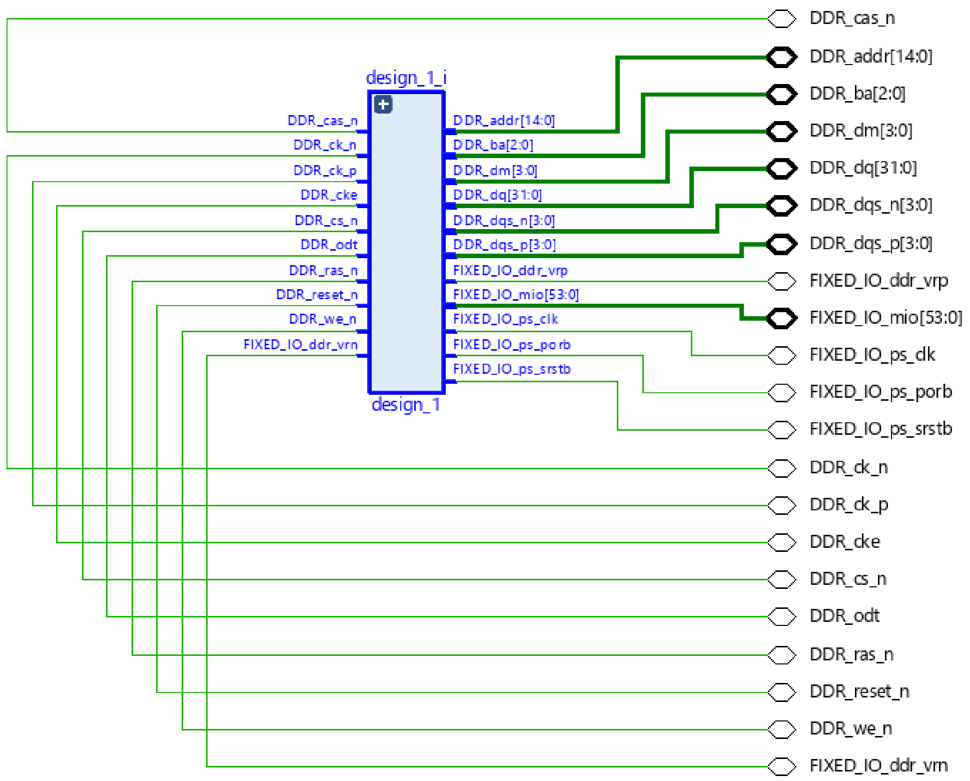
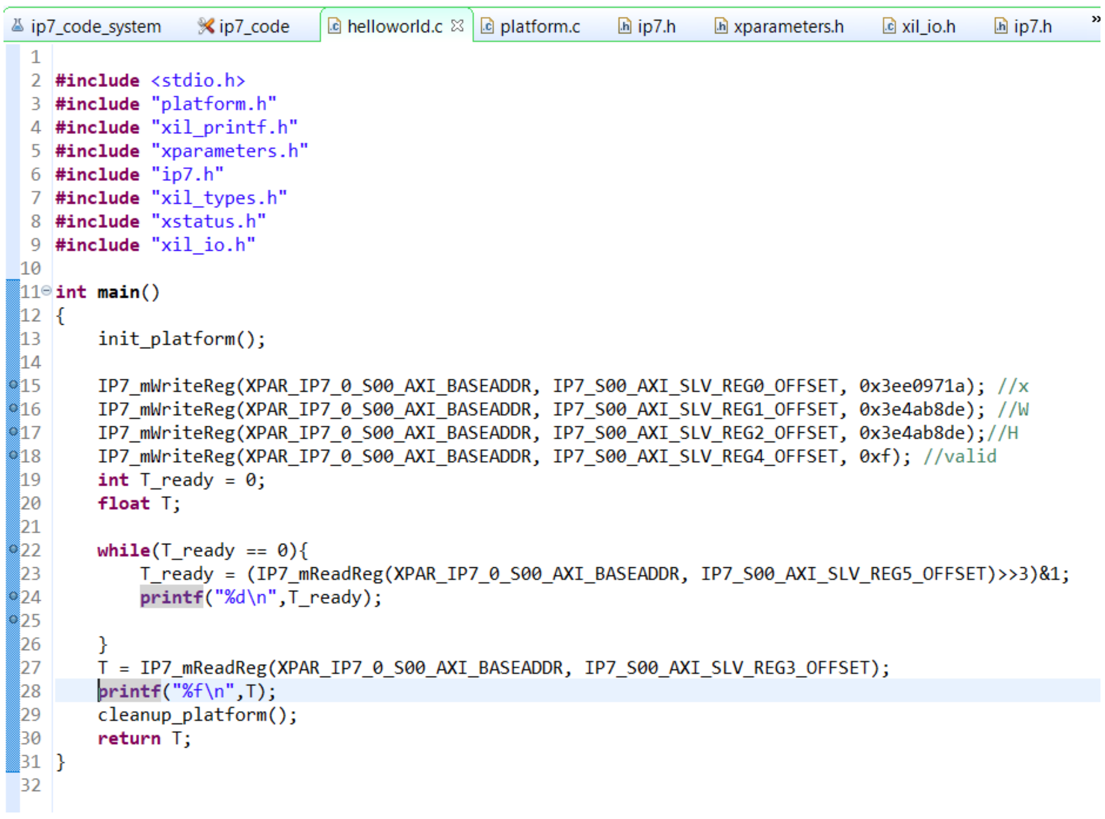


**Figure 5.18:** Schematic of Project

The schematic representation of the whole system is shown in Figure 5.18.

## 5.4 Vitis C Code of Project

By following the Vitis project creation steps described in Chapter 2, the hardware file is imported and a new project is created.



```

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "ip7.h"
#include "xil_types.h"
#include "xstatus.h"
#include "xil_io.h"

int main()
{
    init_platform();

    IP7_mWriteReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG0_OFFSET, 0x3ee0971a); //x
    IP7_mWriteReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG1_OFFSET, 0x3e4ab8de); //W
    IP7_mWriteReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG2_OFFSET, 0x3e4ab8de);//H
    IP7_mWriteReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG4_OFFSET, 0xf); //valid
    int T_ready = 0;
    float T;

    while(T_ready == 0){
        T_ready = (IP7_mReadReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG5_OFFSET)>>3)&1;
        printf("%d\n",T_ready);

    }
    T = IP7_mReadReg(XPAR_IP7_0_S00_AXI_BASEADDR, IP7_S00_AXI_SLV_REG3_OFFSET);
    printf("%f\n",T);
    cleanup_platform();
    return T;
}

```

**Figure 5.19:** Vitis C Code of Project

The "platform.h" header is included to initialize the platform, the "xparameters.h" header to use the XPAR_IP7_0_S00_AXI_BASEADDR variable of the ip file, the "ip7.h" header to access the ports of the designed IP, and the "xil_io.h" header for the WriteReg and ReadReg functions. In the main function, since the variables X, W, H are inputs in the equation, they are sent into the mWriteReg function according to the registers where the IP is directed in the hardware using the BaseAddr and Reg_Offset variables. The third variable in the function is the hexadecimal form of the data. When the valid variable was high when observed in the simulation, the data was processed and the result was produced. Therefore, the valid variable is sent as 0xf high with the appropriate register to the mWriteReg function. The correct result was observed when the T_ready variable was high in the simulation. For this, the state of T_ready is observed in the while loop until T_ready is high in the code. When T_ready is high, the result of T is retrieved from the mReadReg function with the appropriate register value.

# 6. REALISTIC CONSTRAINTS AND CONCLUSIONS

## 6.1 Practical Application of This Project

Ground penetrating radar, which can be the application area of the study, is used to find objects buried in the ground in many different sectors. Especially in the detection of anti-personnel mines, radars used in the military field need to process the image and clear the confusion for target detection. Matrix decompositions to be implemented in the project are relatively simple algorithms for FPGA implementation, which are widely used in the processing of ground penetrating radar images. By implementing a successful image processing algorithm such as robust negative matrix decomposition on FPGA, a portable, inexpensive and fast solution will be developed directly on the radar.

## 6.2 Realistic Constarints

Many realistic constraints were encountered during the design. The fact that the data used is of floating-point type has been the most difficult part of the project. It both made it difficult to receive data on the Vitis interface and caused some situations such as the insufficient number of DSPs on the card in the Vivado interface, requiring the use of a special IP. Therefore, the operation of the project has slowed down. In addition, the fact that the project is within the scope of TUBITAK 1001 projects caused time to be lost due to the preparation of extra reports and presentations.

### 6.2.1 Social, environmental and economic impact

Ground penetrating radar is an extremely safe measurement method that does not require digging, used to find buried objects. It emerges as a very useful and safe technique in mine exploration for military purposes.

With the addition of suitable (low transaction cost and robust clutter) clutter removal and detection methods to be developed for the mobile system, the whole system will find a wide market opportunity as a compact, mobile and low-cost product, and will appeal to civilian or military users from all walks of life.

The ability of these application to implement on FPGA and SoC will provide great advantages in terms of speed.

### 6.2.2 Cost analysis

In this project, all the steps implemented on the computer and hardware devices are planned to be used. The programs that will be used are provided by ITU and Xilinx. A junior engineer’s salary is assumed to be 5$/hour. The project will take 28 weeks with respect to EHB4901E and EHB4902E lectures AKTSs.
Salary = 5$ * 5.5 hour * 28 weeks = 770 $ for per student
ZedBoard Zynq-7000 ARM/FPGA SoC Development Board = 500$
Sum = 1270 $

### 6.2.3 Standards

Throughout the project, hardware designs were based on IEEE's Verilog and VHDL standards and IEEE-754 Floating-Point standard. Likewise, the C model was completed with reference to the C99 standard. In addition, TUBITAK Standards were complied with.

### 6.2.4 Health and safety concerns

OnSite Effective with a Low Cost, Portable Ground Penetrating Radar System Clutter Clearing and Targeting is an extremely safe measurement method for locating buried objects. It is a very useful and safe technique in military mining exploration.

## 6.3 Future Work and Recommendations

In order to improve this project, the steps applied with float value can also be applied with double values. In addition, in order to speed up the RNMF algorithm at the moment, the hardware part can be designed for operations in other lines, such as matrix multiplication, division, shifting, etc., which are performed in hardware. Thus, the system will be accelerated even more.

## 6.4 Conclusion

With this project, it is aimed to accelerate the system by realizing the matrix multiplication, which is a frequently performed operation in software, of RNMF, which is a clutter removal algorithm. For this, first of all, it was necessary to transfer the data from the radar to the computer. In the Vitis environment, the data was read from the text file and transferred to the computer and made ready for processing. In the Vitis environment, the RNMF algorithm was run with the data from the text file. It has been seen that the longest-running operation is matrix multiplication. Therefore, this line of operation is intended to be implemented in hardware. Implementation of floating-point numbers in hardware is quite difficult compared to other data types. Therefore, a special design is required. Methods to do this have been investigated. It was decided that the most suitable method for this project is the existing Floatimg-Point IP Generator. A new private IP was designed using this custom IP located in Xilinx's own environment. Afterwards, a block design was created using the designed IP and Zynq-7000 processor. After the block design was synthesized, implanted and the bit file was produced, the system designed in the software part was used. Thus, the execution of the main code is accelerated and the number of clock cycles is reduced.

## REFERENCES

**[1] Kuon, I., Tessier, R. and Rose, J.,** 2008. FPGA Architecture: Survey and Challenges.

**[2] Ashenden, P.J.,** 2007. Digital Design (VHDL): An Embedded Systems Approach Using VHDL, Morgan Kaufmann.

**[3] XILINX**, 2020, "Vivado Design Suite User Guide".

**[4] MathWorks,** Design digital FPGA, SoC FPGA, or ASIC hardware, https:// www.mathworks.com/discovery/hardware-design.html. (accessed Jan. 02, 2023).

**[5] Thomas, D.E. and Moorby, P.R**., (2002). The Verilog® Hardware Description Language, Kluwer Academic Publishers.

[6]**"ZedBoard - Digilent Reference**," *digilent.com.*, https://digilent.com/reference/programmable-logic/zedboard/start (accessed Dec. 27, 2023).

**[7] Ishtiaq, A., Khan, M. U., Ali, S. Z., Habib, K., Samer, S., & Hafeez, E**. (2021, January). A Review of System on Chip (SOC) Applications in Internet of Things (IOT) and Medical. In ICAME21, International Conference on Advances in Mechanical Engineering, Pakistan (pp. 1-10).

**[8] Zynq-7000 ARM/FPGA SoC** https://www.digikey.com/en/products/detail/amd-xilinx/XC7Z020-2CLG484I/3925763 (accessed Dec. 27, 2023).

**[9] Design, M. B. D.** (2012). Vivado Design Suite Reference Guide

**[10] L. Zhang, Z. Chen, M. Zheng, and X. He,** "Robust non-negative matrix factorization," *Frontiers of Electrical and Electronic Engineering in China*, vol. 6, no. 2, pp. 192–200, Feb. 2011, doi: 10.1007/s11460-011-0128-0.

**[11] Xilinx,** (2021). Vivado Design Suite Properties Reference Guide.

**[12] Kumlu, D. ve Erer, I**., (2018) "Clutter removal in GPR images using non-negative matrix factorization", Journal of Electromagnetic Waves and

Applications, vol 32, no. 16, pp. 2055–2066, doi: https://doi.org/10.1080/09205071.2018.1489740

**[13] "Zynq-7000 SoC,"** *Xilinx.* https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

**[14] Y.-X. Wang and Y.-J. Zhang,** "Nonnegative Matrix Factorization: A Comprehensive Review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1336–1353, Jun. 2013, doi: 10.1109/tkde.2012.51.

**[15] H. Zhou, Y. Wang, Q. Liu and Y. Wang,** "RNMF-Guided Deep Network for Signal Separation of GPR Without Labeled Data," in IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, 2022, Art no. 3507705, doi: 10.1109/LGRS.2021.3099161.

**[16] D. Kumlu and I. Erer,** (2020) "Improved Clutter Removal in GPR by Robust Nonnegative Matrix Factorization," in IEEE Geoscience and Remote Sensing Letters, vol. 17, no. 6, pp. 958-962, doi: 10.1109/LGRS.2019.2937749.

**[17] N. B. Erichson, A. Mendible, S. Wihlborn, and J. N. Kutz,** "Randomized nonnegative matrix factorization," *Pattern Recognition Letters*, vol. 104, pp. 1–7, Mar. 2018, doi: 10.1016/j.patrec.2018.01.007.

**[18] L. Du, X. Li and Y. -D. Shen,** "Robust Nonnegative Matrix Factorization via Half-Quadratic Minimization," 2012 IEEE 12th International Conference on Data Mining, 2012, pp. 201-210, doi: 10.1109/ICDM.2012.39.

**[19] APU, A. P. U.** (2012). XA Zynq-7000 All Programmable SoC First Generation Architecture.

**[20] "UART Protocol"**, 2021,https://techdestek.net/2021/07/15/uart-protokolu/

**[21] Fasi, M., & Mikaitis, M.** (2021). Algorithms for stochastically rounded elementary arithmetic operations in IEEE 754 floating-point arithmetic. *IEEE Transactions on Emerging Topics in Computing*, *9*(3), 1451-1466.

**[22] PG060, X.** (2017). Floating-Point Operator V7. 1 LogicCore IP Product Guide.

**[23] Campos, N., Edirisinghe, E., Fatima, S., Chesnokov, S., & Lluis, A.** (2023). Fpga implementation of a custom floating-point library. In *Proceedings of SAI Intelligent Systems Conference* (pp. 527-542). Springer, Cham.

**[24] Even, G., Mueller, S. M., & Seidel, P. M.** (2000). A dual precision IEEE floating-point multiplier. *Integration*, *29*(2), 167-180.

**[25] Louca, Cook and Johnson**, "Implementation of IEEE single precision floating point addition and multiplication on FPGAs," 1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 1996, pp. 107-116, doi: 10.1109/FPGA.1996.564761.

**[26] R. K. Kodali, S. K. Gundabathula and L. Boppana**, "FPGA implementation of IEEE-754 floating point Karatsuba multiplier," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 300-304, doi: 10.1109/ICCICCT.2014.6992974.

**[27] K. Manolopoulos, D. Reisis and V. A. Chouliaras**, "An efficient multiple precision floating-point multiplier," 2011 18th IEEE International Conference on Electronics, Circuits, and Systems, 2011, pp. 153-156, doi: 10.1109/ICECS.2011.6122237.

**[28] A. Akkas and M. J. Schulte,** "A quadruple precision and dual double precision floating-point multiplier," Euromicro Symposium on Digital System Design, 2003. Proceedings., 2003, pp. 76-81, doi: 10.1109/DSD.2003.1231903.

**[29] H. Yamada, T. Hotta, T. Nishiyama, F. Murabayashi, T. Yamauchi and H. Sawamoto,** "A 13.3ns double-precision floating-point ALU and multiplier," Proceedings of ICCD '95 International Conference on Computer Design. VLSI in Computers and Processors, 1995, pp. 466-470, doi: 10.1109/ICCD.1995.528909.

## CURRICULUM VITAE

**Name Surname** : Çisem KURT

**Place and Date of Birth** : Gönen -Turkey, 1999

**E-Mail** : cisemkurt99@gmail.com

**Education: • B.Sc.** : Istanbul Technical University- Electronics and Communication Engineering (2020-2023)

**Professional Experience** : 07.2021 – 08.2021 İTÜ Embedded System Design Lab. Internship

08.2021 – 09.2021 Tekhnelogos Software – Hardware Design Engineer Intern

10.2021 – 03.2022 Tekhnelogos Software – Hardware Part Time Design Engineer

03.2022 – Present Turkish Aerospace (TUSAŞ) – Part Time Design Engineer

## CURRICULUM VITAE

|  |  |
|---|---|
| **Name Surname** | **:** M. Furkan ERTURAL |
| **Place and Date of Birth** | **:** İstanbul -Turkey, 1997 |
| **E-Mail** | **:** fertural@gmail.com |

**Education: • B.Sc.** **:** Istanbul Technical University- Electronics and Communication Engineering (2018-2023)

**Professional Experience** **:** 07.2021 – 08.2021 ITU Embedded System Design Lab. Internship

2021 - 2022: Embedded Sofware Systems Developer at TUBITAK RUTE, Part-Time Research Engineer

2022 - 2022: Sofware Developer at SIEMENS-ADVANTA, Part-Time Software Developer