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

# IMPLEMENTATION OF A SOC BY USING LOWRISC PROCESSOR ON AN FPGA FOR IMAGE FILTERING APPLICATIONS

**SENIOR DESIGN PROJECT**

**Bartu SÜRER**

**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT**

**JANUARY 2019**

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

# IMPLEMENTATION OF A SOC BY USING LOWRISC PROCESSOR ON AN FPGA FOR IMAGE FILTERING APPLICATIONS

**SENIOR DESIGN PROJECT**

**Bartu SÜRER**
**(040160706)**

**ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT**

**Project Advisor: Assoc. Prof. Dr. Sıddıka Berna ÖRS YALÇIN**

**JANUARY 2019**

We are submitting the Senior Design Project Report entitled as "**IMPLEMENTATION OF A SOC BY USING LOWRISC PROCESSOR ON AN FPGA FOR IMAGE FILTERING APPLICATIONS**". 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 work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity .

**Bartu SÜRER** .....
(040160706)

**Project Advisor:** **Assoc. Prof. Dr. Sıddıka Berna ÖRS YALÇIN**

.....

## FOREWORD

I would like to thank to my mentor Assoc. Prof. Dr. Sıddıka Berna Örs Yalçın who helped me to find this project and who supported me in all my mistakes, to reach flawless success and at least work as much as I to complete my project successfully. Secondly, I would like to offer my gratitude to my mentor Res. Assist. M.Sc. Latif Akçay, who gave his endless support and even sacrificed his own time in every stage of the project. Without him, I would never find my way in this project. Finally, I would like to emphasize that I owe to my friends, my girlfriend Serra Önder for her endless support and my family who has the biggest role on my successes for my entire life.

January 2019 Bartu SÜRER

**TABLE OF CONTENTS**

**Page**

## ABBREVIATIONS

| | |
|---|---|
| **ALU** | **:** Arithmetic Logic Unit |
| **CPU** | **:** Central Processing Unit |
| **FPGA** | **:** Field Programmable Gate Array |
| **FPU** | **:** Floating Point Unit |
| **ISA** | **:** Instruction Set Architecture |
| **OS** | **:** Operating System |
| **PC** | **:** Personal Computer |
| **RISC** | **:** Reduced Instruction Set Computing |
| **RTL** | **:** Register Transfer Level |
| **SoC** | **:** System on Chip |
| **VHDL** | **:** Very High –Speed Integrated Circuit Hardware Description Language |

**LIST OF TABLES**

**Page**

**LIST OF FIGURES**

**Page**

**IMPLEMENTATION OF LOWRISC SOC ON FPGA FOR IMAGE FILTERING APPLICATIONS**

## SUMMARY

The rapid increase in technology is one of the major factors that lead to a rapid increase in human needs. In every area of our lives, we now carry out our daily and professional life with the help of machines. For these reasons, the importance of processors that give mind and purpose to machines comes to the fore. But the biggest problem with all the increasing importance is the lack of a standard in the architecture of these processors. Although each company creates its own standard and architectural structure, today there is no architectural structure in which these companies have shared. In addition, large chip manufacturers develop their own chips and software and sell them at high prices. Because chip development requires years of knowledge and research, and at the same time, this process creates high costs. For this reason, countries and institutions that could not create their own chip architecture will benefit from using open source processors instead of purchasing high-priced and non-open-source firms' chips for their smart machines. In this context, open source processors are becoming more and more important in order to be able to benefit economically and to be highly effective in terms of performance / price. In this project, with using "RISC-V" instruction set architecture which was created eight years ago at the University of California, Berkeley and is closely followed by many large companies like Nvidia and Western Digital, based "lowRISC" SoC is aimed to be implemented and image processing algorithms will be executed on that SoC. This SoC created with Rocket cores.

For the implementation of the project, FPGA (field programmable gate series) technology, which is one of the most important blessings offered by the technology to the electronics engineers, is used. In general, the project first deals with the proper operation of the lowRISC chip system on the "Nexys 4 DDR" produced by Xilinx properly. Then, a Linux operating system version which works with lowRISC installed on this chip, certain filters have been applied to the images with pure "C" programming language without installing any additional libraries. For this purpose, Gauss blur filter, Sobel operation filter, Embossing filter, sharpening filter, black-white filter and many more are used in this project, which are essential for image processing applications and which are essential for image feature detection purpose.

As a result, the outcome of the project also shows us the working conditions and cost of a system developed using this processor. Even though we haven't developed an architecture as a country yet, open source processors are shown in this project which will be extremely helpful to us in the technological tools we produce. Here, it is seen that choosing these processors are extremely wise decision for developers in terms of time, price and labor, both for prototyping and technological studies.

# FPGA ÜZERİNDE GÖRÜNTÜ İŞLEME AMACIYLA LOWRISC ÇİP SİSTEMİNİN GERÇEKLEŞTİRİLMESİ

## ÖZET

Teknolojinin hızla artması, insan ihtiyaçlarının da hızla artmasına yol açan büyük etmenlerden biri. Hayatımızın her alanında artık makinelerin yardımlarıyla günlük ve profesyönel hayattaki işlerimizi gerçekleştiriyoruz. Tüm bu sebeple makinelere akıl ve amaç veren işlemcilerin önemi ön plana çıkmaktadır. Ancak tüm artan önemin getirdiği en büyük sıkıntı bu işlemcilerin mimarisinde bir standartın olmaması. Her firma kendi standartını ve mimari yapısını oluştursa da, günümüzde bu firmaların ortaklaştığı bir mimari yapı bulunmamakta. Ayrıca büyük çip üreticileri kendi çiplerini ve yazılımlarını geliştirmekte ve bunları yüksek fiyatlarda satmaktadır. Çünkü çip geliştirmek yılların bilgi birikimi ve araştırmasını gerektirmekte ve aynı zamanda bu süreçte yüksek maliyet ortaya çıkarmaktadır. Bu sebeple kendi çip mimarisini oluşturamamış ülkeler ve kurumlar, kendi geliştirecekleri makineler için yüksek fiyatlı ve açık kaynaklı olmayan firmaların çip mimarisini satın almak yerine, açık kaynak kodlu olan işlemcileri kullanmalarında büyük fayda sağlayacaklardır. Bu bağlamda özellikle ekonomik olarak da faydalanabilmek ve performans/fiyat açısından son derece efektif olması için, dünyanın birçok yerinde rövaşta olan açık kaynak kodlu işlemciler git gide önem kazanmakta. Bu projede ise sekiz yıl önce Berkeley'deki California Üniversitesi tarafından geliştirilen, günümüzde de bir çok büyük firma tarafından da (Nvidia, Western Digital) yakından takip edilen "RISC-V" komut seti mimarisi ile oluşturulmuş Rocket çekirdeği tabanlı olan "lowRISC" çip üzeri sistemi çip üzeri sistemi gerçeklenmesi ve üzerinde görüntü işleme algoritmalarının çalıştırılması hedeflenmiştir. Bu amaçla görüntü işleme uygulamalarında kullanılan olmazsa olmaz filtrelerden Gauss bulanıklaştırma filtresi, Sobel operasyonu filtresi, Kabartma filtresi, keskinleştirme filtresi, siyah-beyaz filtre ve daha pek çoğu, bu projede gerçekleştirilmiştir.

Projenin gerçekleştirilmesi için, teknolojinin elektronik mühendislerine sunduğu en önemli nimetlerden biri olan FPGA (alan programlanabilir geçit seri) teknolojisi kullanılmıştır. Proje genel olarak önce lowRISC çip üzeri sisteminin, Xilinx firması tarafından üretilen "Nexys 4 DDR" üzerinde kurulması ve çalıştırılmasını ele alıyor. Daha sonra bu çipe kurulan Linux işletim sistemi üzerinde, hiç bir ek kütüphane kurmadan saf "C" programlama dili ile görüntülere belirli filtreler uygulanmıştır. Bu amaçla, görüntü işleme uygulamalarında kullanılan ve görüntünün işlenmesi için gerekli olan özelliklerini ön plana çıkaran, olmazsa olmaz filtrelerden Gauss bulanıklaştırma filtresi, Sobel operasyonu filtresi, Kabartma filtresi, keskinleştirme filtresi, siyah-beyaz filtre ve daha pek çoğu bu projede gerçekleştirilmiştir.

Sonuç olarak, projenin getirisinin de bize gösterdiği sonuç, bu işlemci kullanılarak geliştirilen bir sistemin çalışma şartları ve maliyeti kolayca hesaplanabilir. Henüz ülke olarak bir mimari geliştirmemiş olsak bile açık kaynak kodlu işlemciler, kendi üreteceğimiz teknolojik araçlarda bize son derece yardımcı olacağı bu projede gözler önüne serilmiştir. Burada hem prototiplenmesi hem de teknolojik çalışmalara

ayrılacak zaman, fiyat ve emek açısından, bu işlemcilerin geliştiriciler için son derece akıllıca bir seçim olduğu görülmüştür.

# 1. INTRODUCTION

In this graduation project, firstly the "lowRISC" chip [1], which was created from the open source processors "RISCV" architecture [2], will be implemented on the Nexys4 DDR [3] which belongs to Xilinx and then the image processing algorithm will be run on the Linux operating system which will be run by this process. In addition, the "lowRISC" architecture in the project has been examined in detail and will be explained in this section.

Recently, the increasing trend and need for image processing and machine learning is the main reason for the use of this project and the processor. In this project, the use of image processing algorithms on the system and evaluation of the results will be discussed.

## 1.1 Basic Information And Concepts

A detailed description of all equipment, tools and software used in the project will be made in this section. It will be stated what each one is and what the purpose serves in the project..

### 1.1.1 Open source CPUs

Over the years, the word "open source" has only been talked about software in the world of technology [4]. The main reason for this is that developing a software requires less resources than developing hardware. These sources can be listed as basic economic, commercial, time and research studies. However, with the advancing technology nowadays, many developer groups are now less dependent on these resources. In addition, the development of their own architectural structures of large firms and the competition among themselves, these companies do not bring any standard to the chip world. For this reason, the importance and value of open source processors in the last decade has been understood by the big companies that develop other technologies. Some of thems are listed below.

#### 1.1.1.1 Amber

Amber is a processor core that supports the "ARM" architecture and has a 32 bit RISC calculation [5]. The processor is open source and is located on the website of the action called “OpenCores”. The Amber processor has an instruction set fully compatible with ARMv2a and has full support by the "GNU toolchain". The reason why the processor has this old version instruction set is that it is not patented by ARM [5].

#### 1.1.1.2 Leon

LEON is a 32-bit CPU microprocessor core, with used the SPARC-V8 RISC architecture and its instruction set designed by Sun Microsystems [6]. It was firstly designed by the European Space Research and Technology Centre, and after that designing the project was continued by Gaisler Research. LEON was written in synthesizable Verilog HDL (Hardware Description Language) [6]. LEON has two different license model, a LGPL/GPLFLOSS license that can be used without any payment for licensing, and other is a proprietary license that can be purchased for integration in a proprietary product [6].

#### 1.1.1.3 OpenSPARC

OpenSPARC is a 64-bit and 32-threaded processor written in verilog in 2005 by Sun Microsystems' register-transfer level (RTL) [7]. OpenSPARC with two different models, UltraSPARC T1 and UltraSPARC T2, has GNU General Public license [7].

### 1.1.2 RISC-V

RISC-V is an open-source, ISA (instruction set architecture) designed for high performance with great power efficiency which was created eight years ago at the University of California, Berkeley [8]. RISC-V is the fifth generation of the “reduced instruction set computer” type of architecture. What makes the "RISCV" architecture unique is not a great performance or technology. What makes this architecture unique is that it is completely redundant and therefore gives great promise to developers in terms of time and economy. RISC-V ISA delivers a new level of free open source which extensible hardware freedom on architecture, which makes this ISA innovative and futuristic. In addition, this ISA has many software tools, simulation tools, compiler tools, bootloaders, kenrels, OSs and debug tools avaible by the RISC-V foundation. Because of all these features, many developers and companies have created many

cores created by using RISC-V. Some of these cores are shown in the Table 1.1 and some other SoCs are shown in Table 1.2.

| Name | Links | Priv. spec | User spec | License | Maintainers |
|---|---|---|---|---|---|
| rocket | **GitHub** | 1.11-draft | 2.3-draft | BSD | SiFive, UCB Bar |
| freedom | **GitHub** | 1.11-draft | 2.3-draft | BSD | SiFive |
| Berkeley Out-of-Order Machine (BOOM) | **GitHub** | 1.11-draft | 2.3-draft | BSD | Esperanto, UCB Bar |
| ORCA | **GitHub** | | RV32IM | BSD | VectorBlox |
| RI5CY | **GitHub** | | RV32IMC | Solderpad Hardware License v. 0.51 | ETH Zurich, Università di Bologna |
| Zero-riscy | **GitHub** | | RV32IMC | Solderpad Hardware License v. 0.51 | ETH Zurich, Università di Bologna |
| Ariane | **Website**,**GitHub** | | RV64IMC | Solderpad Hardware License v. 0.51 | ETH Zurich, Università di Bologna |
| Riscy Processors | **Website**,**GitHub** | | | MIT | MIT CSAIL CSG |
| OPenV/mriscv | **GitHub** | | RV32I(?) | MIT | OnChipUIS |
| VexRiscv | **GitHub** | | RV32I[M][C] | MIT | SpinalHDL |
| Roa Logic RV12 | **GitHub** | 1.9.1 | 2.1 | Non-Commercial License | Roa Logic |
| SCR1 | **GitHub** | 1.10 | 2.2, RV32I/E[MC] | Solderpad Hardware License v. 0.51 | Syntacore |
| Hummingbird E200 | **GitHub** | 1.10 | 2.2, RV32IMAC | Apache 2.0 | Bob Hu |
| Shakti | **Website**,**GitHub** | 1.10 | 2.2, RV64IMAFD | BSD | IIT Madras |
| ReonV | **GitHub** | | | GPL v3 | |
| PicoRV32 | **GitHub** | | RV32I/E[MC] | ISC | Clifford Wolf |
| MR1 | **GitHub** | | RV32I | Unlicense | Tom Verbeure |

**Table 1.1 :** Some cores that used RISC-V ISA **[2]**

| Name | Links | Core | License | Maintainers |
|---|---|---|---|---|
| Rocket Chip | **GitHub**,**Simulator** | Rocket | BSD | SiFive, UCB BAR |
| LowRISC | **GitHub** | RV32IM | BSD | LowRISC CIC |
| PULPino | **Website**,**GitHub** | RI5CY, Zero-riscy, Ariane | Solderpad Hardware License v. 0.51 | ETH Zurich, Università di Bologna |
| PULPissimo | **Website**,**GitHub** | RI5CY, Zero-riscy, Ariane | Solderpad Hardware License v. 0.51 | ETH Zurich, Università di Bologna |
| Briey | **GitHub** | VexRiscv | MIT | SpinalHDL |
| Riscy | **GitHub** | RV64I | MIT | AleksandarKostovic |
| Raven | **GitHub** | PicoRV32 | ISC | RTimothyEdwards, mkkassem (efabless.com) |
| PicoSoC | **GitHub** | PicoRV32 | ISC | Clifford Wolf |
| Icicle | **GitHub** | RV32I | ISC | Graham Edgecombe |

**Table 1.2 :** Some SoCs that used RISC-V ISA **[2]**

In this project, LowRIS SoC (System on chip) is used which shown in Table 1.2.

Some basic RISC-V instructions are similar to old RISC ISA's like OpenRISC some of them are shown in Table 1.3

| 31 27 | 26 22 | 21 17 | 16 12 | 11 10 9 7 | 6 0 | |
|---|---|---|---|---|---|---|
| jump target | | | | | opcode | J-type |
| rd | upper immediate | | | | opcode | U-type |
| rd | rs1 | imm[11:7] | imm[6:0] | funct3 | opcode | I-type |
| imm[11:7] | rs1 | rs2 | imm[6:0] | funct3 | opcode | B-type |
| rd | rs1 | rs2 | funct10 | | opcode | R-type |
| rd | rs1 | rs2 | rs3 | funct5 | opcode | R4-type |

| | | | | | | |
|---|---|---|---|---|---|---|
| imm25 | | | | | 1101011 | J imm25 |
| imm25 | | | | | 1101111 | JAL imm25 |
| imm12hi | rs1 | rs2 | imm12lo | 000 | 1100011 | BEQ rs1,rs2,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 001 | 1100011 | BNE rs1,rs2,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 100 | 1100011 | BLT rs1,rs2,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 101 | 1100011 | BGE rs1,rs2,imm12 |
| rd | rs1 | imm12 | | 000 | 0000011 | LB rd,rs1,imm12 |
| rd | rs1 | imm12 | | 001 | 0000011 | LH rd,rs1,imm12 |
| rd | rs1 | imm12 | | 010 | 0000011 | LW rd,rs1,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 000 | 0100011 | SB rs1,rs2,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 001 | 0100011 | SH rs1,rs2,imm12 |
| imm12hi | rs1 | rs2 | imm12lo | 010 | 0100011 | SW rs1,rs2,imm12 |
| rd | rs1 | imm12 | | 000 | 0010011 | ADDI rd,rs1,imm12 |
| rd | rs1 | 000000 | shamt | 001 | 0010011 | SLLI rd,rs1,shamt |
| rd | rs1 | imm12 | | 010 | 0010011 | SLTI rd,rs1,imm12 |
| rd | rs1 | imm12 | | 100 | 0010011 | XORI rd,rs1,imm12 |
| rd | rs1 | 000000 | shamt | 101 | 0010011 | SRLI rd,rs1,shamt |
| rd | rs1 | 000001 | shamt | 101 | 0010011 | SRAI rd,rs1,shamt |
| rd | rs1 | imm12 | | 110 | 0010011 | ORI rd,rs1,imm12 |
| rd | rs1 | imm12 | | 111 | 0010011 | ANDI rd,rs1,imm12 |
| rd | rs1 | rs2 | 0000000 | 000 | 0110011 | ADD rd,rs1,rs2 |
| rd | rs1 | rs2 | 1000000 | 000 | 0110011 | SUB rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 001 | 0110011 | SLL rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 010 | 0110011 | SLT rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 011 | 0110011 | SLTU rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 100 | 0110011 | XOR rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 101 | 0110011 | SRL rd,rs1,rs2 |
| rd | rs1 | rs2 | 1000000 | 101 | 0110011 | SRA rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 110 | 0110011 | OR rd,rs1,rs2 |
| rd | rs1 | rs2 | 0000000 | 111 | 0110011 | AND rd,rs1,rs2 |

**Table 1.3 :** Some basic RISC instructions **[9]**.

#### 1.1.2.1 RISC-V compiling tools

RISC-V has a standart toolchain like other devices which is GNU cross compiler toolchain. It just reimplemented for RISC-V ISA. Compiling flow just like normal GCC tools except it produce a unique binary for RISC-V ISA. However in this project, lowRISV SoC is used. So this part is not applied on project because of these tools are for standalone RISC-V cores. LowRISC also using this GCC tools but with different terminal commands. Toolchain usage diagram shown in Figure 1.1 for general flow map.

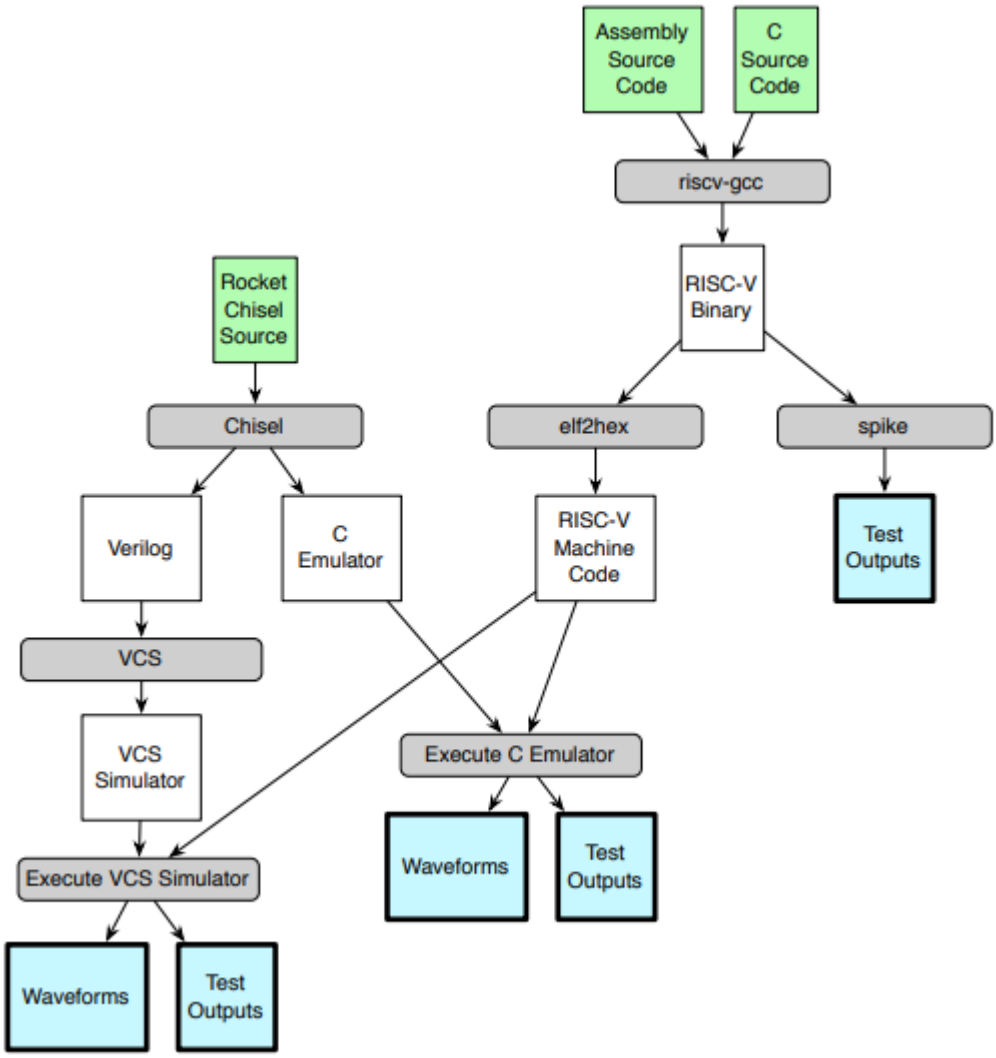


**Figure 1.1 :** RISC-V GCC toolflow. Greens are input files and blues are output files **[9]**.

This is just a standalone RISC-V coding toolflow. In our project, RISC-V core will run our applications on a Linux OS and will execute C language commands.

### 1.1.3 LowRISC

LowRISC an organisation that creating its own unique open sourced SoC based on 64 bit RISC-V ISA. This SoC is capable of running on a debian Linux OS. Organisation also working closely with University of Cambridge and with a community that aims to improve this SoC in terms of hardware and software design [1]. The LowRISC team is developing the system for the sake of three main objectives.

- Their designs are licensed and developed as allowed together with co-developers from many parts of the world. Free and open RISC-V ISA is being

implemented. Thus, a common open-source SoC will be completely standardized.

- The advantage of being open source is that it is a fully auditable system and it is aimed at being as safe as it is economically by using technologies tagged memory.
- By trying to reflect the flexibility and diversity of the software to the hardware, to develop a future SoC by creating a system that can be improved in real-time applications.

A high level top view of lowRISC chip can be seen in Figure 1.2.

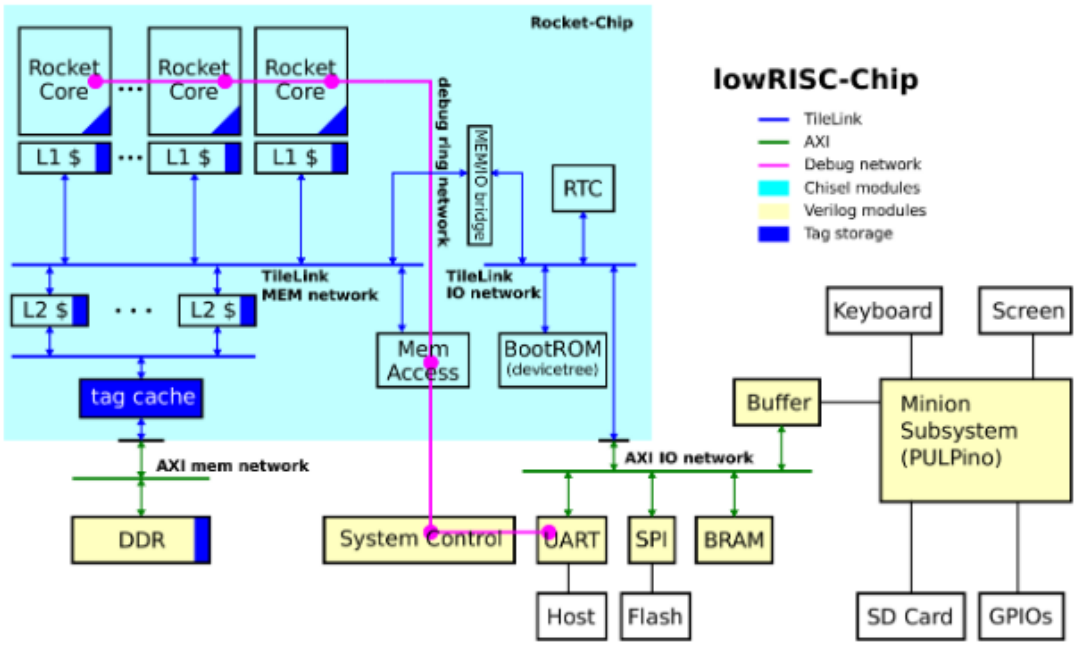


**Figure 1.2 :** Top view of lowRISC SoC **[10]**

However, version of lowRISC chip used in this project is 0.6 which does not include minion subsystem. Reason is that running Linux with that core makes Linux slow. In future releases it will come again. But still we can use SD Card, vga screen, ps2 keyboard. And also L2 caches removed from this version with upgrade of new version of TileLink network [11].

#### 1.1.3.1 Rocket-Core view

Rocket core contains a 5-stage pipeline and is built on RISCV64G, which includes one integer ALU and one optional FPU. In some aspects it is called 6-stage pipeline

because of "pcgen" stage is included with it. But lowRISC team differs this stage like a standalone stage. Pcgen and fetch stages are shown in Figure 1.3.

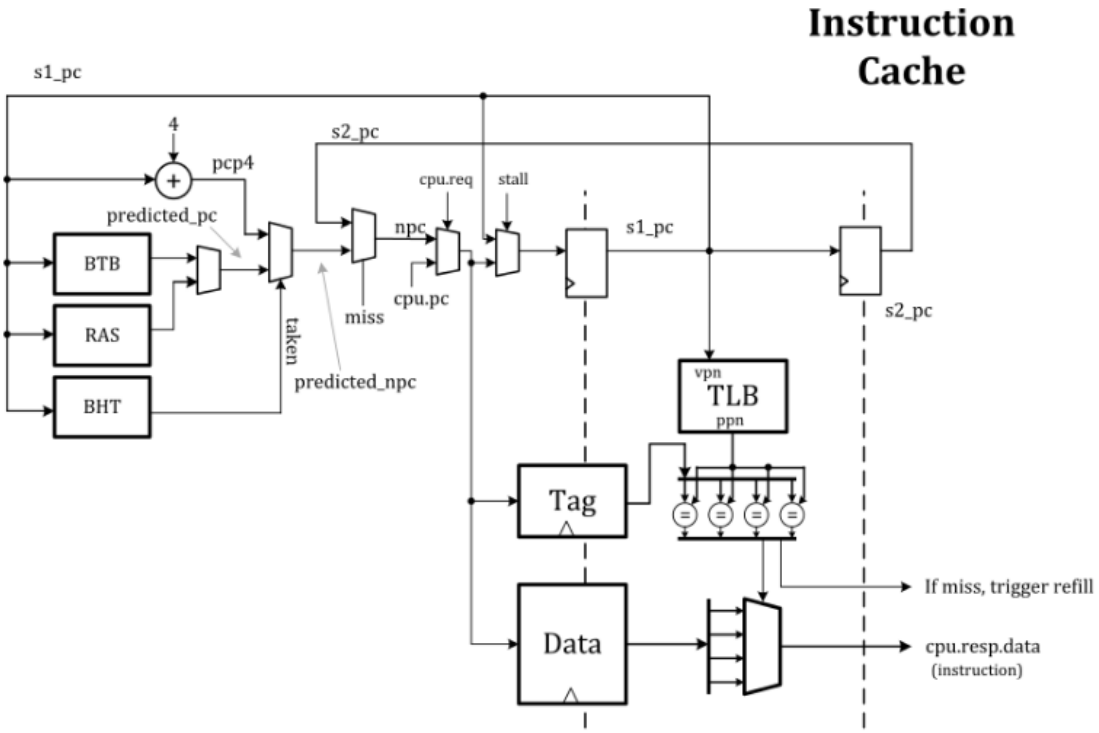


**Figure 1.3 :** Instruction cache which includes pcgen and fetch stages **[12]**.

Remained stages are shown in Figure 1.4.

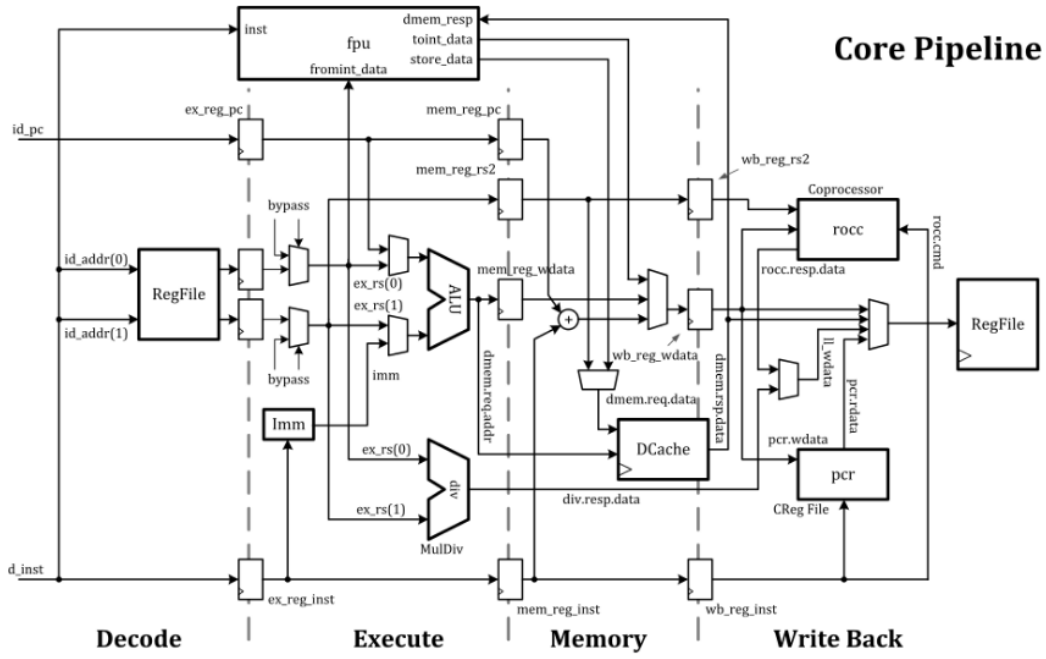


**Figure 1.4 :** Other stages of pipeline **[12]**.

General pipeline diagram can be shown in Figure 1.5.

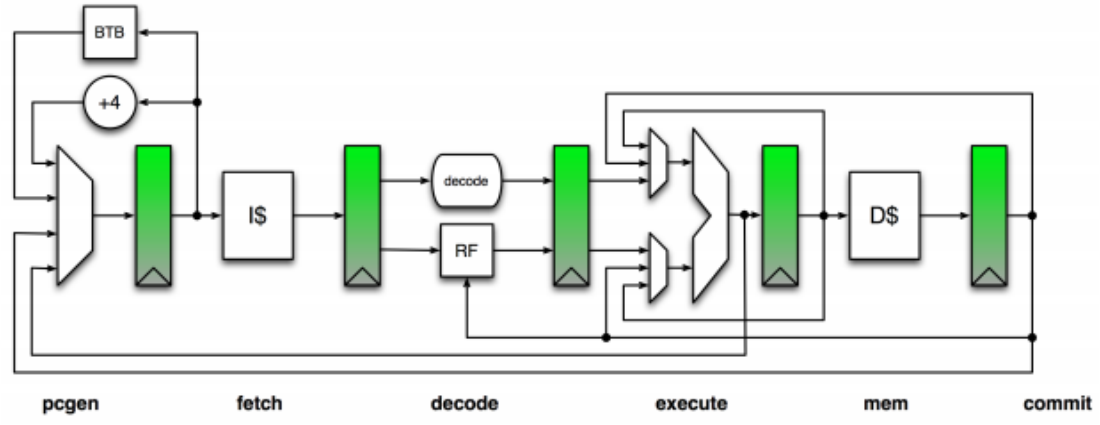


**Figure 1.5 :** All pipeline stages in one diagram **[9]**.

L1 data cache is in Figure 1.6.

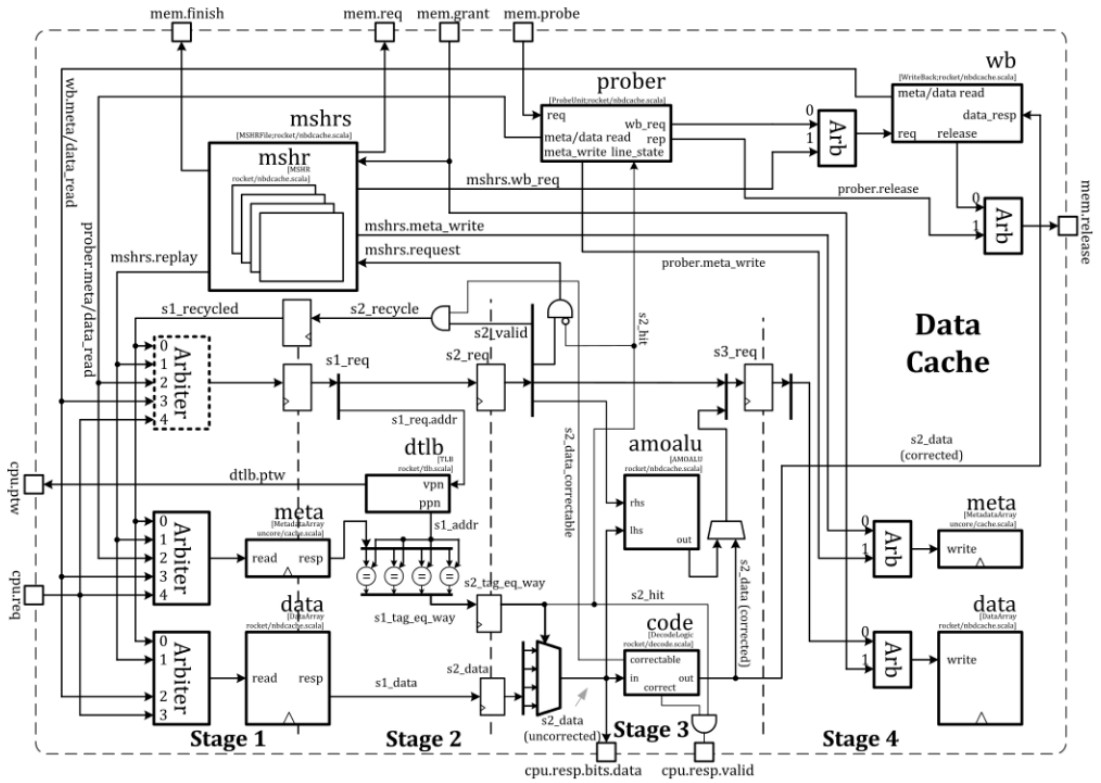


**Figure 1.6 :** L1 data cache overview **[12]**.

And all architecture of Rocket Core in Figure 1.7.

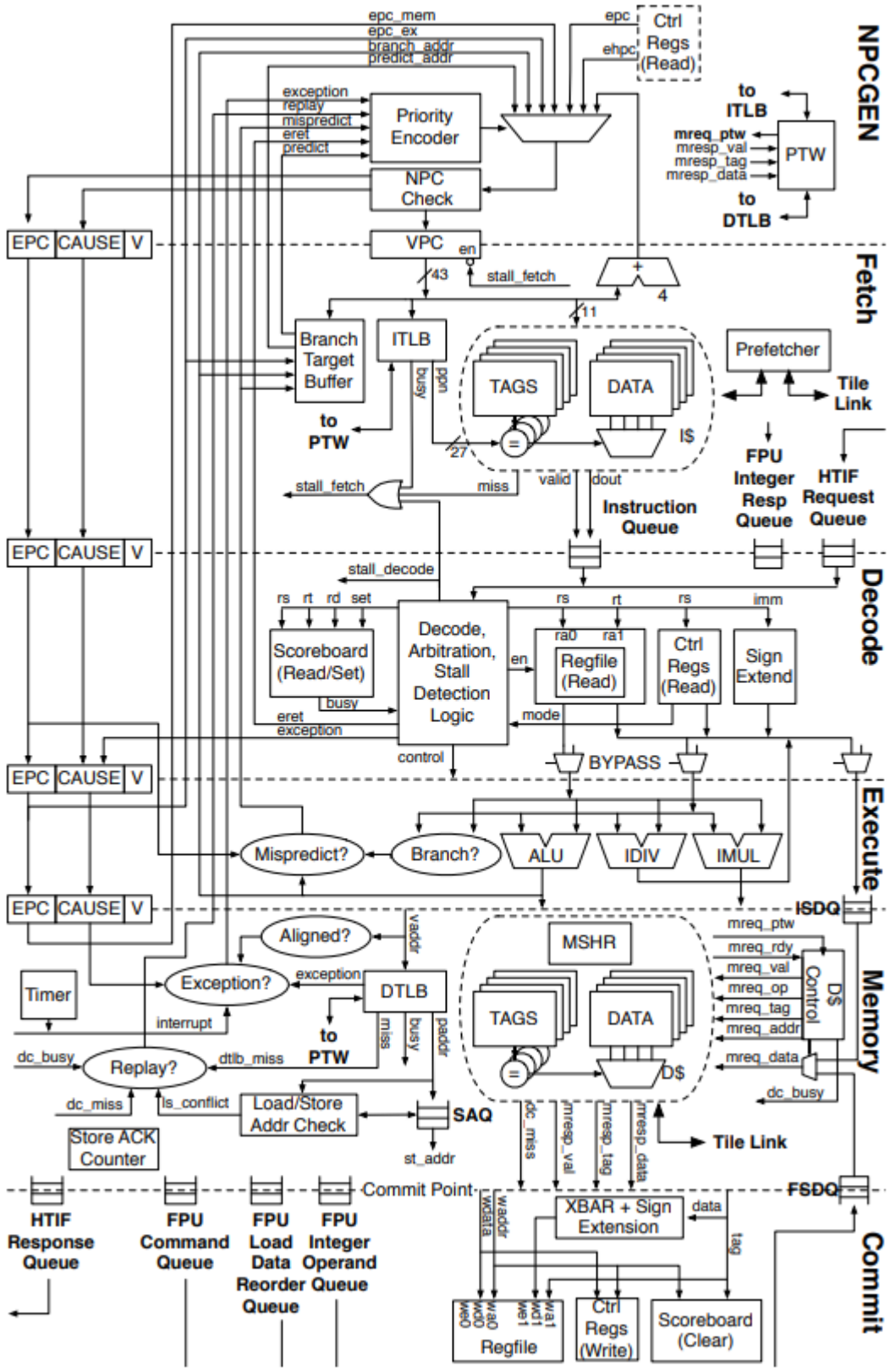


**Figure 1.7 :** General microarchitecture of all system **[9]**.

### 1.1.4 Image processing and filters

Image processing can be defined by extracting features to be used from any camera image, video image, or image, and making a result by using these features after rendering the computer to process [13]. Images can be considered mathematically two-dimensional arrays in digital media. On top of that, the signal processing algorithms can be processed just like a normal analog signal. For this reason, it has an important place in today's machine learning algorithms and many computer science. Today, in our world, from the medicine to the automotive sector, military and scientific research has a very important importance. This recent importance of image processing also plays an important role in what this project is working on. Although the algorithms are tried to be optimized by software, the importance of hardware comes to the fore as well. For these algorithms, many companies are very stable in designing their own equipment and transferring cost, time and labor to these designs. For this reason, it is appropriate to use the lowRISC project, which is open source in this graduation project, for image processing. A general flow for image processing algorith can be shown in Figure 1.8.

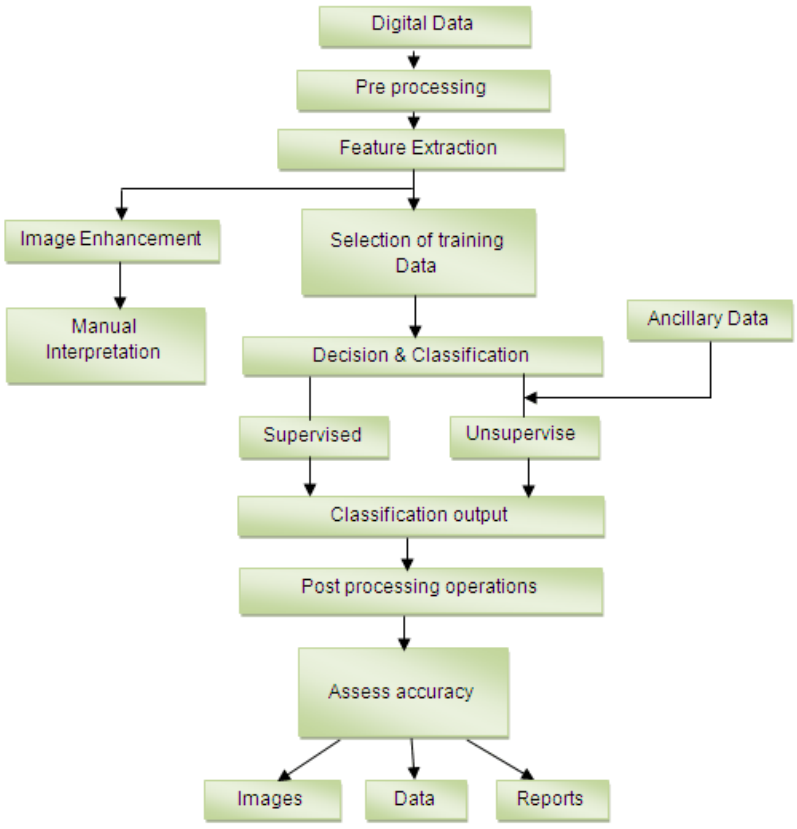


**Figure 1.8 :** Image processing algorithm general flow **[14]**.

### 1.1.4.1 Kernel

In image processing, the kernel can be defined as a square mask or convolution matrix, which multiplies certain pixels in the image with their weights and add these weighted pixels with each other and creates new pixels of the resulting image.

### 1.1.4.2 Convolution operation

Convolution operation is generally can be described as any pixel in an image will be additioned with pixels in the neighbor of central pixel values which multiplied by the kernel weights of each pixels, and every pixel in the picture sees this process in sequence. Briefly, it is an arithmetic that allows the new picture to be formed by sliding the kernel on to image from left to right and from top to bottom. Process can be shown in Figure 1.9.

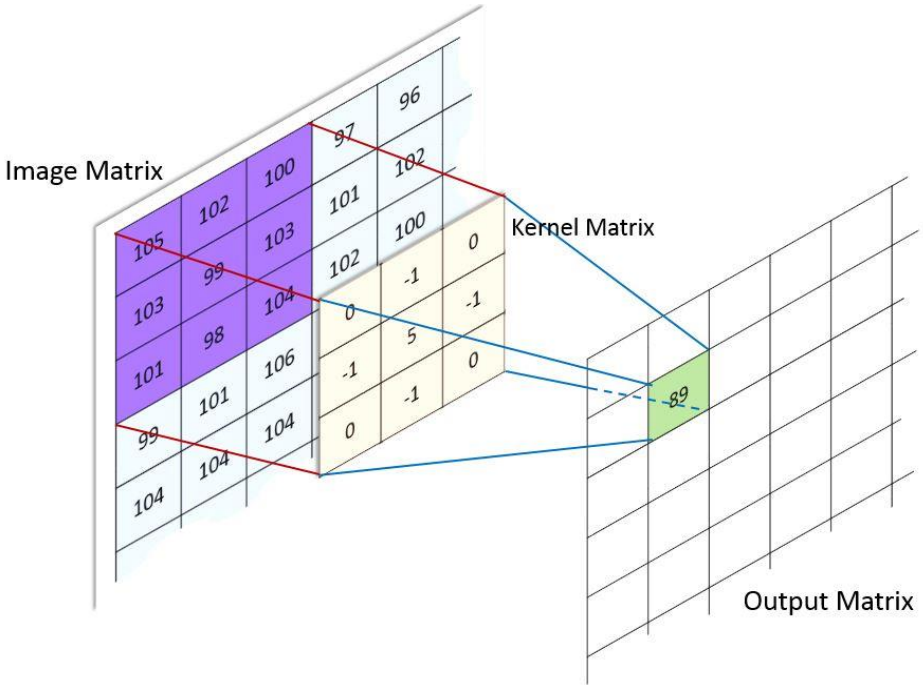


**Figure 1.9 :** Convolution operation.

### 1.1.5 General flow of project

The overall flow of the project will be covered under this heading. Each step written here will be explained in detail in the next chapter. What we need first is a computer and an external or internal disk for installing the Ubuntu operating system. Then, to run the lowRISC chip in the FPGA, the Vivado program of Xilinx will be installed. There may also need a monitor to connect the FPGA to the monitor, especially in terms of visual comfort. There is also a need for an ethernet cable for remote communication

with the lowRISC Soc in the FPGA. A minimum of 4 GB (16 GB recomended) SD card and a pc compatible card reader requires to install linux and some applications. Other option is connecting and using a keyboard through USB interface of FPGA and executes commands from there. Once the Ubuntu disk has been installed, the necessary software programs for the compilation of lowRISC and the execution of the software will be installed and some variables will be defined in the operating system. After this step, the compiled software and hardware files will be thrown into the FPGA and the lowRISC will be running. The format of the image to be processed will be changed for processing with C. The C codes of the image processing algorithm and the image to be processed will be transferred to the lowRISC SoC using the SSH protocol. The algorithms will be run on the lowRISC and the resulting image will be transferred to the host computer using the SSH protocol to be monitored again.

## 1.2 Literature Review

### 1.2.1 Literature review at Istanbul Technical University

- **(Güngör, Öndeş, Sarı, Uçkun; 2018)** ***"Instruction Set Extension Of Some Processors For Secure Iot Implementations"***

The main subject of this thesis is using AES and PRESENT cryptology algoritms with instartcion set extention (ISE) in open source processors and to increase the security of these processors for IoT applications. One of the core of the project is the Pulpino RI5CY which belongs to the ETH Zurich. The developer Talip Tolga Sarı first placed this core on the Nexys4 DDR development board which belongs to Xilinx and then observed that it was working by testing. After this stage, the main subject of the thesis AES application was implemented by using the instruction set extention method, successfully performed [15].

### 1.2.2 Literature review at Turkey

So far, no thesis work about this subject has been done in our country yet. However, one of Turkey's largest defense industrial firm ASELSAN is a member of RISC-V organisation ranked as silver member.

### 1.2.3 Global Literature review

- **(T. Liu, G. Shi, L. Chen, F. Zhang, Y. Yang and J. Zhang; 2018) "*TMDFI: Tagged Memory Assisted for Fine-Grained Data-Flow Integrity Towards Embedded Systems Against Software Exploitation*"**

Modern software attacks often benefit from problems caused by memory corruption. While the classical data flow integrity provides a good solution to these problems, they have a large impact on the speed of the applied system and create time problems because of origin of this algorithm is software based. For all these reasons, a new data stream integrity hardware, which supports the tagged memory system, is presented by the developers. In particular, the hardware prototype has been tested on the lowRISC with the RISCV-based core and major improvements in speed and time have been observed. According to the results, the data flow integrity hardware system in the real-time tests reduced the pressure of system from 104% to 39%, the space overhead shrinks from 50% to 12.5% [16].

- **(A. Ramos, A. Ullah, P. Reviriego and J. A. Maestro; 2018) "*Efficient Protection of the Register File in Soft-Processors Implemented on Xilinx FPGAs*"**

The main reason for this study is that software processors are designed on FPGA by using SRAM and also the register structure is one of the most critical information carrier elements for a processor. Regarding external factors, especially in the aviation and space sectors, the register values in the processors can be changed and data losses or errors are experienced. In this study, error tolerance system has been tried to be applied to register file. This system is used to add the register file to the memory element in the processor. The RISCV-based LowRISC SoC was used for testing. A parity based error finder and interchangeable logic system designed to eliminate single-bit errors. As a result, the designed system not only saves the faulty bits but also occupies much less space than the general "Triple Modular Redundancy" module [17].

# 2. IMPLEMENTING LowRISC PROCESSOR ON AN FPGA

## 2.1 Gathering Required Environment

First, the "Ubuntu 16.04.5 LTS" [18] operating system must be installed on an external or internal disk. A second method, the virtual machine method, has also been tried, but the use of virtual machines is not recommended because of some tools and hardware constraints [19].

Immediately after this process, it is necessary to install Vivado 2018.1 [20], which is owned by Xilinx and is required to place lowRISC in FPGA. In installation tab you must use WebPack edition of Vivado. It is also important that 2018.1 must be installed for the lowRISC v0.6. Because in feature releases of Vivado some IP Cores would change. This will create a conflict while creating lowRISC Vivado project After installation fallowing command must be entered at bottom of .bashrc file in your home directory.

- *source /opt/Xilinx/Vivado/2018.1/settings64.sh*

After enterence of that command start a new terminal and write Vivado. If program is starting than everything is until now is okey.

### 2.1.1 Installation of cable drivers

Installation of Vivado on Ubuntu needs also cable driver setup. If these drivers are not installed, Vivado program does not recognise FPGA board on bitstream process. Fallowing driver packages are delivered from Digilent. These are recommended drivers for board [19].

- Adept 2.16.1 Runtime, X64 DEB
- Adept 2.2.1 Utilities, X64 DEB

Drivers can be downloaded from Digilent's website [21]. After downloading them, do not forget to install these drivers.

After these driver installation JTAG driver also must be installed. Open a fresh terminal and enter fallowing codes.

- *cd /opt/Xilinx/Vivado/2018.1/data/xicom/cable_drivers/lin64/install_script/install_drivers*
- *sudo sh install_digilent.sh*

Now everything is set and ready to next installations.

### 2.1.2 Installation of Cmake

Cmake [22] must be installed to compile some sowftwares for lowRISC. To install cmake linux version 3.13.2, it can be downloaded from https://cmake.org/download/ [22]. After downloading the files, the commands below must be entered inside the download folder to the terminal

- *./bootstrap*
- *make*
- *make install*

### 2.1.3 Downloading LowRISC chip git repository

This repository contains all software and toolchains including with all LowRISC files. Before downloading this repo make sure all packages are updated. Following command update all packages [23].

- *sudo apt-get install autoconf automake autotools-dev curl \ libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison \ flex texinfo gperf libncurses5-dev libusb-1.0-0-dev libboost-dev \ swig git libtool libreadline-dev libelf-dev python-dev \ microcom chrpath gawk texinfo nfs-kernel-server xinetd pseudo \ libusb-1.0-0-dev hugo device-tree-compiler zlib1g-dev libssl-dev \ debootstrap debian-ports-archive-keyring qemu-user-static iverilog \ openjdk-8-jdk-headless iperf3 libglib2.0-dev libpixman-1-dev*

After these commands make sure Ubuntu has git. Git also can be installed with fallowing command;

- *sudo apt-get install git*

With proper installation of git, now lowRISC v0.6 could be downloaded from git repo. Following command must entered in work directory. In my case it is in /home/user/vivado/.

- *git clone -b refresh-v0.6 --recursive https://github.com/lowrisc/lowrisc-chip.git lowrisc-chip-refresh-v0.6 cd lowrisc-chip-refresh-v0.6*

When entering "lowrisc-chip-refresh-v0.6" directory, some main directories can be seen. Their purposes and usages are listed below [23];

- **fpga:** FPGA demo implementations
    - **board:** Demo projects for individual development boards.
        - **nexys4:** Files for the Nexys™4 DDR Artix-7 FPGA Board.
- **debian-riscv64:** Scripts to bootstrap a Debian Linux RISCV system
- **riscv-linux:** The Linux RISCV kernel with LowRISC device drivers
- **rocket-chip:** The Rocket core and its sub-systems.
    - **firrtl:** Hardware description intermediate language
    - **hardfloat:** Hardware floating-point arithmetic unit
    - **torture:** Tricky tests that stress the CPU
    - **riscv-tools:** The cross-compilation and simulation tool chain.
        - **riscv-fesvr:** The front-end server that serves system calls on the host machine.
        - **riscv-gnu-toolchain:** The GNU GCC cross-compiler for RISC-V ISA.
        - **riscv-isa-sim:** The RISC-V ISA simulator Spike
        - **riscv-opcodes:** The enumeration of all RISC-V opcodes executable by the Spike simulator.
        - **riscv-pk:** The proxy kernel need for running legacy programs in the Spike simulator.
        - **riscv-tests:** Tests for the Rocket core.
- **src:** The top level code of lowRISC chip.

    - **main:** The Verilog code for hardware implementation.
    - **test:** The Verilog/C++(DPI) test bench files
- **qemu:** User mode emulation of RISCV instruction set

### 2.1.4 Installing RISC-V GNU toolchain

Firstly, default gcc and g++ must be updated. Fallowing commands will work for it.

- *sudo apt update*
- *sudo apt upgrade*
- *sudo apt install build-essential*

Gcc version must be higher than 5.2.

- *gcc –version*
- *which gcc*

After these steps, Reopen "lowrisc-chip-refresh-v0.6" folder and open terminal at this folder.

- *cd /rocket-chip/riscv-tools*
- *./build.sh*

This process is really takes a long time depending computer system. Like 30min- 2 hours. If everything is correct next step is updating .bashrc commands. Open new terminal at "lowrisc-chip-refresh-v0.6" folder

- *./set_env.sh*

At command window result of bash commands will seen in terminal screen. Copy all variables paste to .bsahrc file. Final version of .bashrc can be seen in Figure 2.1

```
source /opt/Xilinx/Vivado/2018.1/settings64.sh

    export TOP=/home/bartu/Vivado/lowrisc-chip-refresh-v0.6
    export RISCV=/home/bartu/Vivado/lowrisc-chip-refresh-
v0.6/riscv
    export OSD_ROOT=/home/bartu/Vivado/lowrisc-chip-refresh-
v0.6/tools
    export PYTHONPATH=/home/bartu/Vivado/lowrisc-chip-refresh-
v0.6/tools/lib/python2.7/site-packages:
    export PATH=/home/bartu/Vivado/lowrisc-chip-refresh-
v0.6/tools/bin:/opt/Xilinx/SDK/2018.1/bin:/opt/Xilinx/SDK/2018
.1/gnu/microblaze/lin/bin:/opt/Xilinx/SDK/2018.1/gnu/arm/lin/b
in:/opt/Xilinx/SDK/2018.1/gnu/microblaze/linux_toolchain/lin64
_le/bin:/opt/Xilinx/SDK/2018.1/gnu/aarch32/lin/gcc-arm-linux-
gnueabi/bin:/opt/Xilinx/SDK/2018.1/gnu/aarch32/lin/gcc-arm-
none-eabi/bin:/opt/Xilinx/SDK/2018.1/gnu/aarch64/lin/aarch64-
linux/bin:/opt/Xilinx/SDK/2018.1/gnu/aarch64/lin/aarch64-
none/bin:/opt/Xilinx/SDK/2018.1/gnu/armr5/lin/gcc-arm-none-
eabi/bin:/opt/Xilinx/SDK/2018.1/tps/lnx64/cmake-3.3.2/bin:/opt
/Xilinx/DocNav:/opt/Xilinx/Vivado/2018.1/bin:/home/bartu/bin:/
home/bartu/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin
:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/ho
me/bartu/Vivado/lowrisc-chip-refresh-v0.6/riscv/bin
    export LD_LIBRARY_PATH=/home/bartu/Vivado/lowrisc-chip-
refresh-v0.6/tools/lib:/home/bartu/Vivado/lowrisc-chip-
refresh-v0.6/riscv/lib
    export PKG_CONFIG_PATH=/home/bartu/Vivado/lowrisc-chip-
refresh-v0.6/tools/lib/pkgconfig
    export FPGA BOARD=nexys4 ddr
```

**Figure 2.1 :** Bashrc commad window.

If getting response for fallowing command, toolchain has successfully installed.

- *which riscv64-unknown-elf-gcc*

### 2.2 Installing Linux to lowRISC

From now there are two ways to get Linux to lowRISC, first one is create your own Linux kernel from tools or, get binary release from lowRISC tutorial page [24]. Creation new Kernel is a more complex way and making mistakes at this stage is really critical. For now it is safe choğice to getting binary release and continue on uploading Linux system to lowRISC. A detailed guide could be seen on https://www.lowrisc.org/docs/download-install-debian/ to continue on building own Kernel [19].

To install binary release input fallowing codes to terminal

- *git clone* [*https://github.com/lowRISC/lowrisc-quickstart.git*](https://github.com/lowRISC/lowrisc-quickstart.git)

- *cd lowrisc-quickstart*
- *make getrelease*

After these commands three important files will be appear in directory [24].

- **boot.bin** – includes Linux kernel, Berkeley bootloader, and initial ramdisk.
- **chip_top.bit** - The FPGA bitstream containing the lowRISC SoC which includes RISCV processor and peripherals and the first-stage booter
- **rootfs.tar.xz -** The compressed tape archive containing the Debian root filing system for RISCV

After that process insert your SD card to card reader and to your computer. After recognized from your Ubuntu, type lsblk command to your command window too see all storage devices listed on your computer. **It is really dangerous that choosing right name on that list. If wrong choice has choosen, own PC data could be formatted without any chechkpoint.** In Figure 2.2 a default 16 GB SD Card shwon at lsblk list.

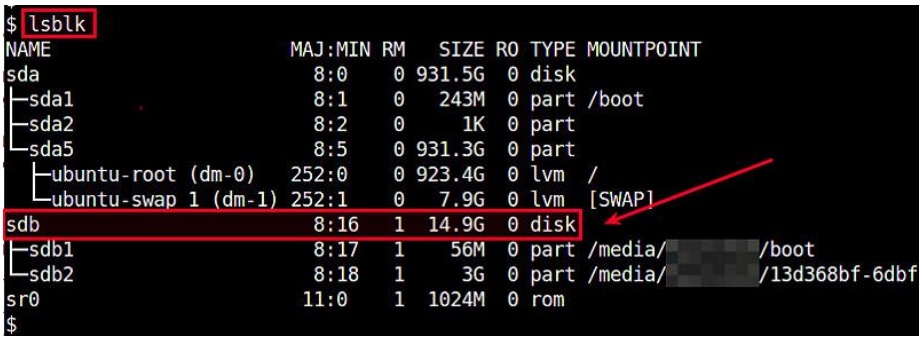


**Figure 2.2 :** 16 GB SD Card

A new fresh SD Card only been as sdx1 as 14.9 GB strorage. Check twice SD Card device name and never forget! In fallowing commands name of device entered as sdx for copy paste safety. X letter must be changed with correct SD Card name.

- *make umount USB=sdx*

If SD card used before type instead:

- *Make USB=sdx cleandisk partition*

If any error occurs during these processes repeat all steps two steps above independently after reinserting SD Card to PC. After operations run lsblk to see results at Figure 2.3.

```
sdc      8:32   1  14.9G  0 disk
├─sdc1   8:33   1    32M  0 part
├─sdc2   8:34   1     2G  0 part
├─sdc3   8:35   1   512M  0 part
└─sdc4   8:36   1  12.3G  0 part
```

**Figure 2.3 :** After commands worked properly.

After that typing fallowing command will install all 3 files above proper locations to SD Card.

- *make USB=sdx mkfs fatdisk extdisk*

Say yes on terminal if asks any question. First question is asking to user create new file system for lowRISC. Second step asks for write permission of kernel and BBL to DOS partition. After all these steps now unplug SD Card from computer and Plug it to FPGA. Also on FPGA board JP1 must stand at QSPI mode and JP2 must stand at SD mode as shown in Figure 2.4 [24].

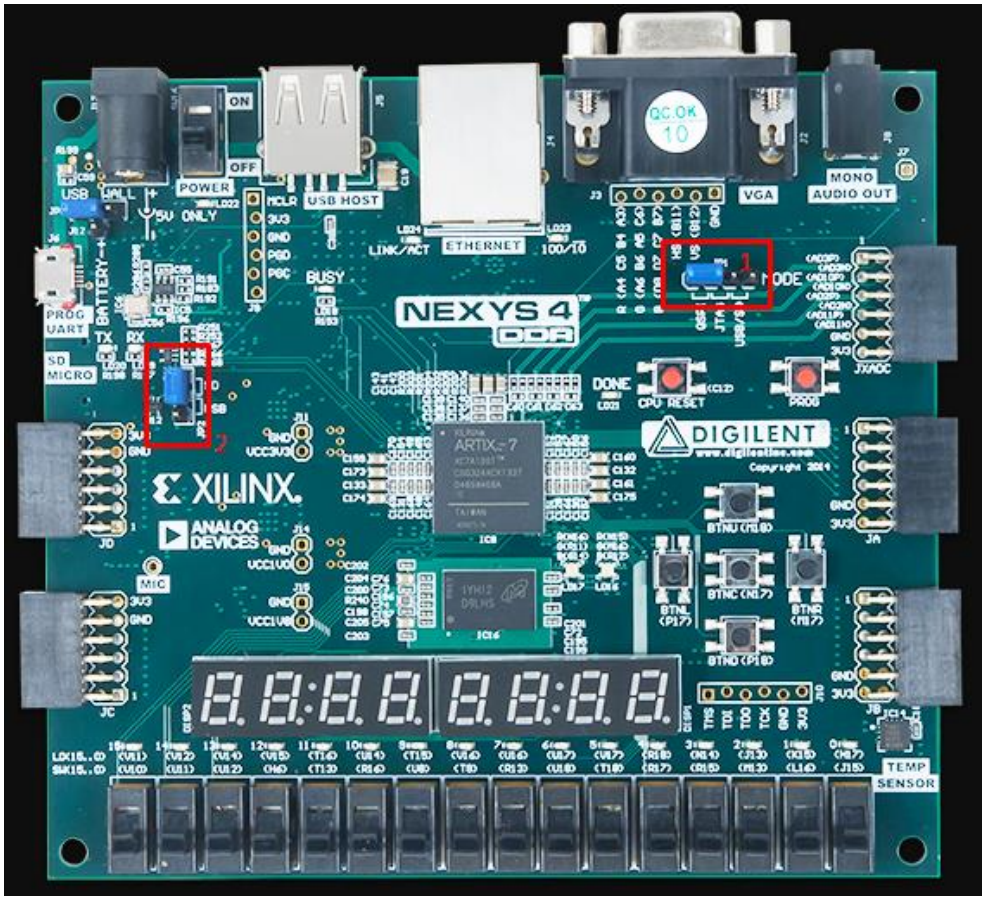


**Figure 2.4 :** Jumper positions to boot from SD card

From now its all set to boot lowRISC to FPGA. Fallowing code will install bitstream file to QSPI Flash.

- *make program-cfgmem*

After these process Vga screen conncected to the FPGA seems like at Figure 2.5.

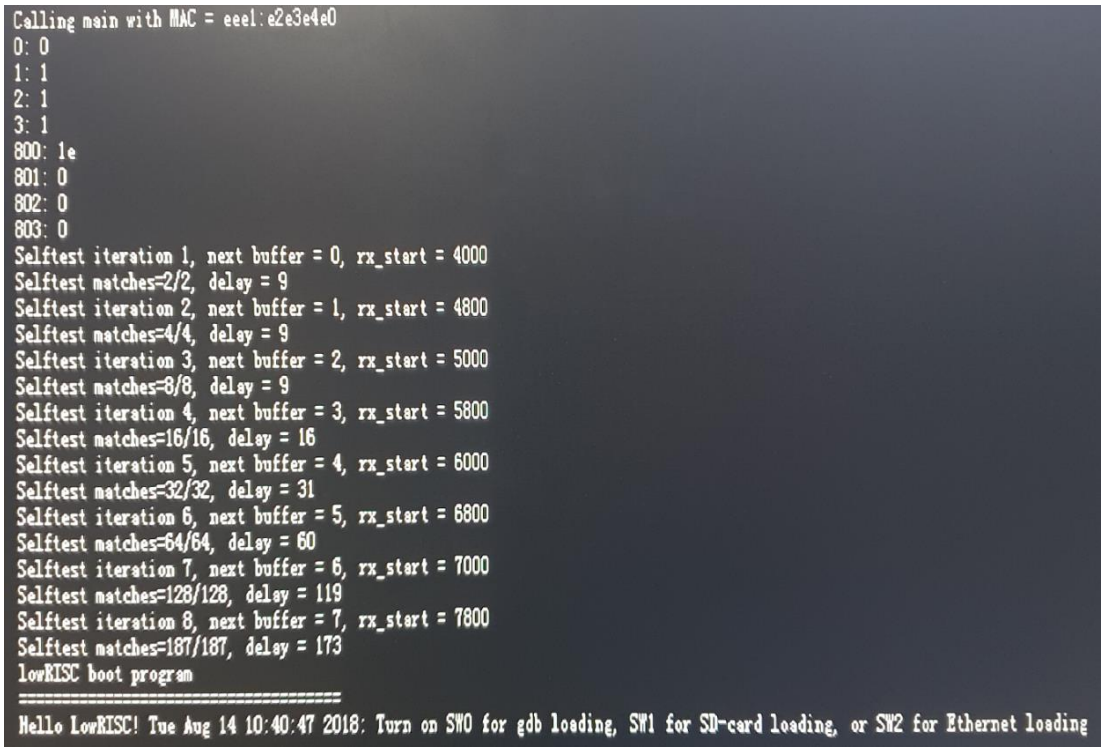


**Figure 2.5 :** Boot screen at VGA display.

And FPGA waits in waiting input mode at Figure 2.6.

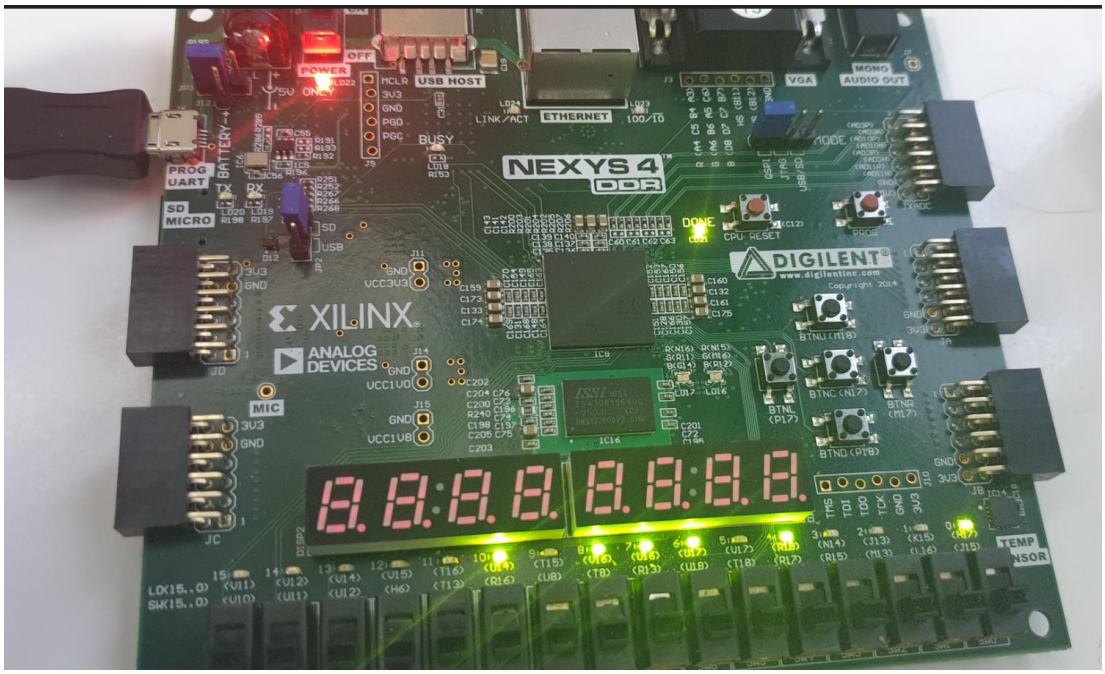


**Figure 2.6 :** Waiting input mode.

In this step, we should choose boot from SD Card and making switch 1 high will do job which shown in Figure 2.7.

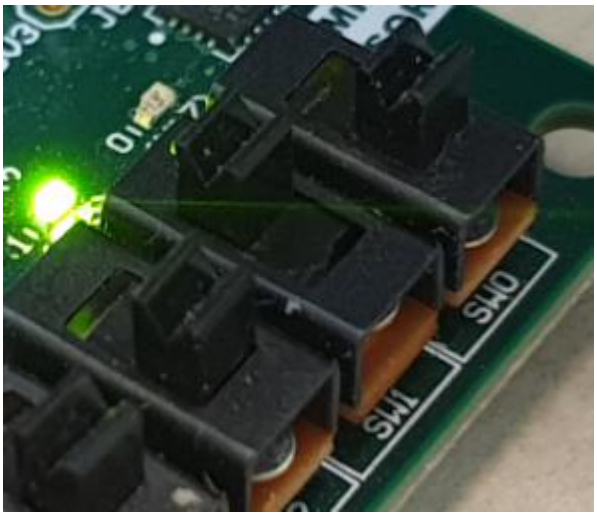


**Figure 2.7 :** Booting from SD card chosen

Choice must be done when before giving power FPGA. After restarting FPGA with choice which sw1 goes up, Figure 2.8, Figure 2.9 and 2.10 shows FPGA u-boot based operations.

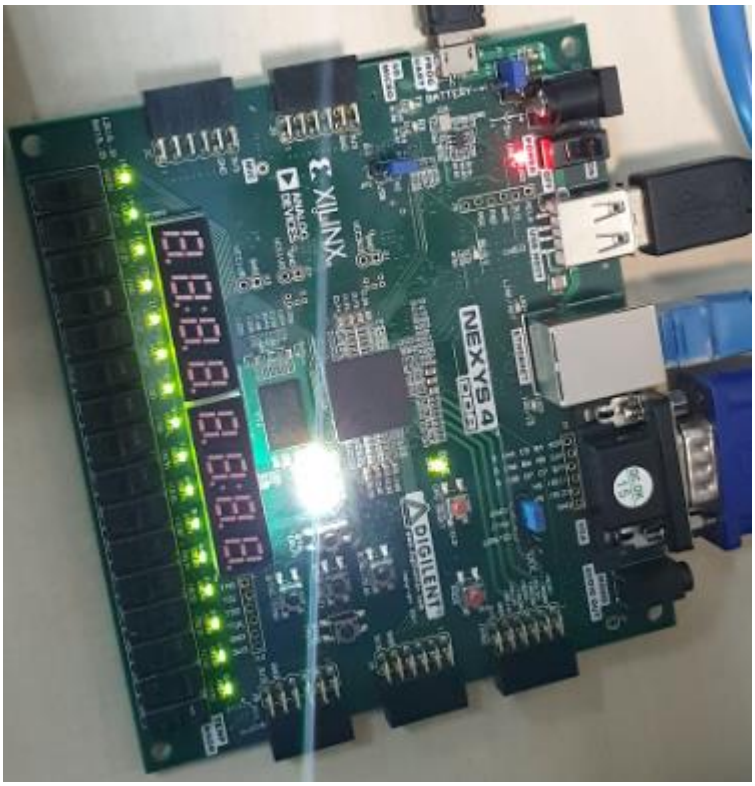


**Figure 2.8 :** Successfully boot operation started.

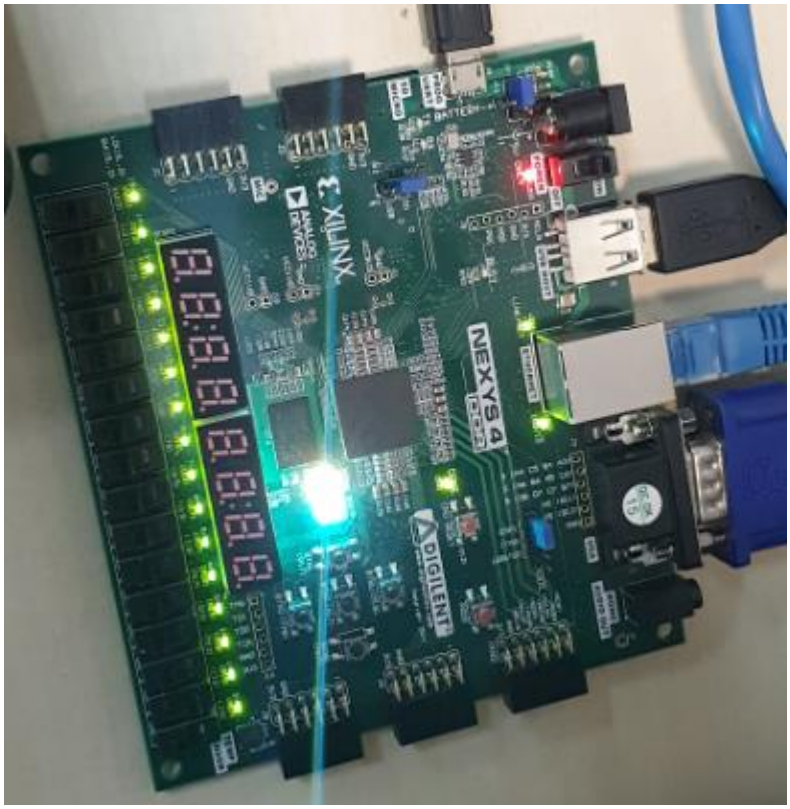


**Figure 2.9**

```
lowRISC boot program
====================================
Hello LowRISC! Tue Aug 14 10:40:47 2018: Booting from FLASH because SW1 is high ..
```

**Figure 2.10**

Output of First stage bootloader in Figure 2.11. If it does not write "Bus Width : 4 bit" it means your SD card does not set properly [24].

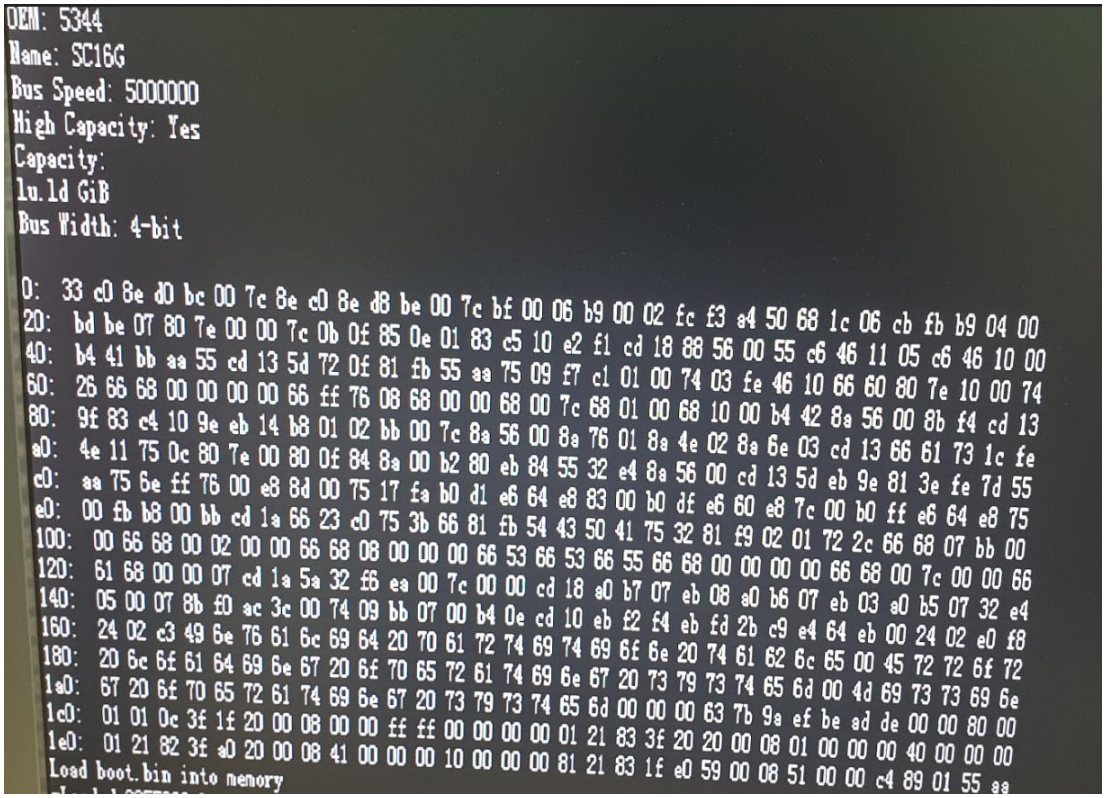


**Figure 2.11 :** First stage bootloader output.

From now fallowing figures which are Figure 2.12, 2.13, 2.14, 2.15 are serial stages of booting Linux to lowRISC.

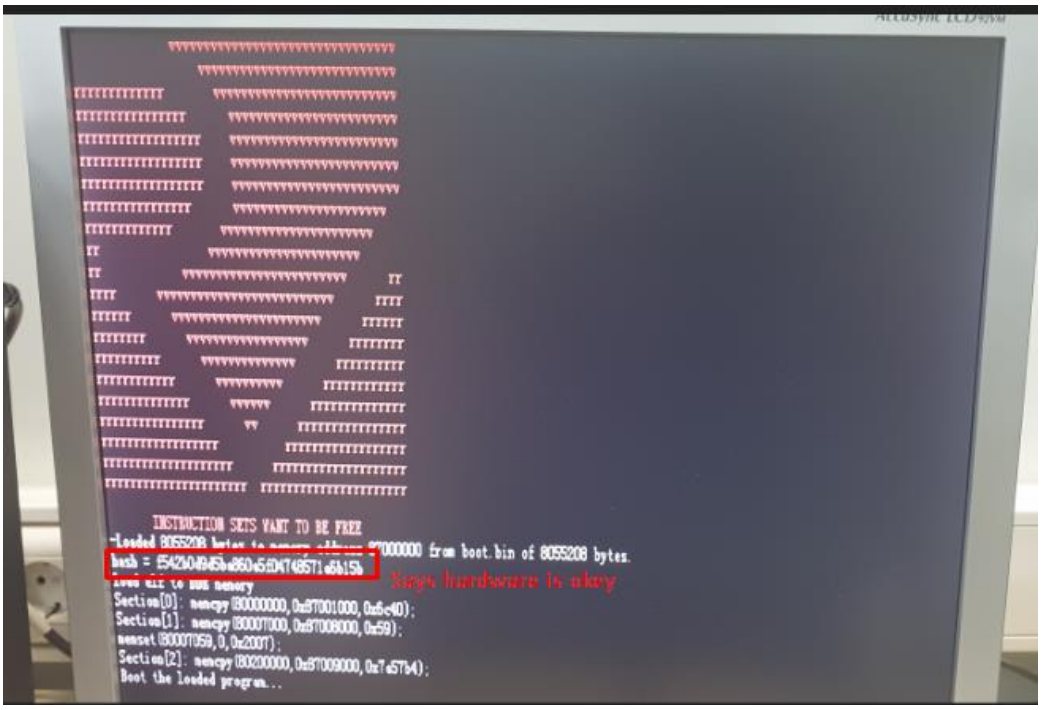


**Figure 2.12 :** Hash code of kernel.

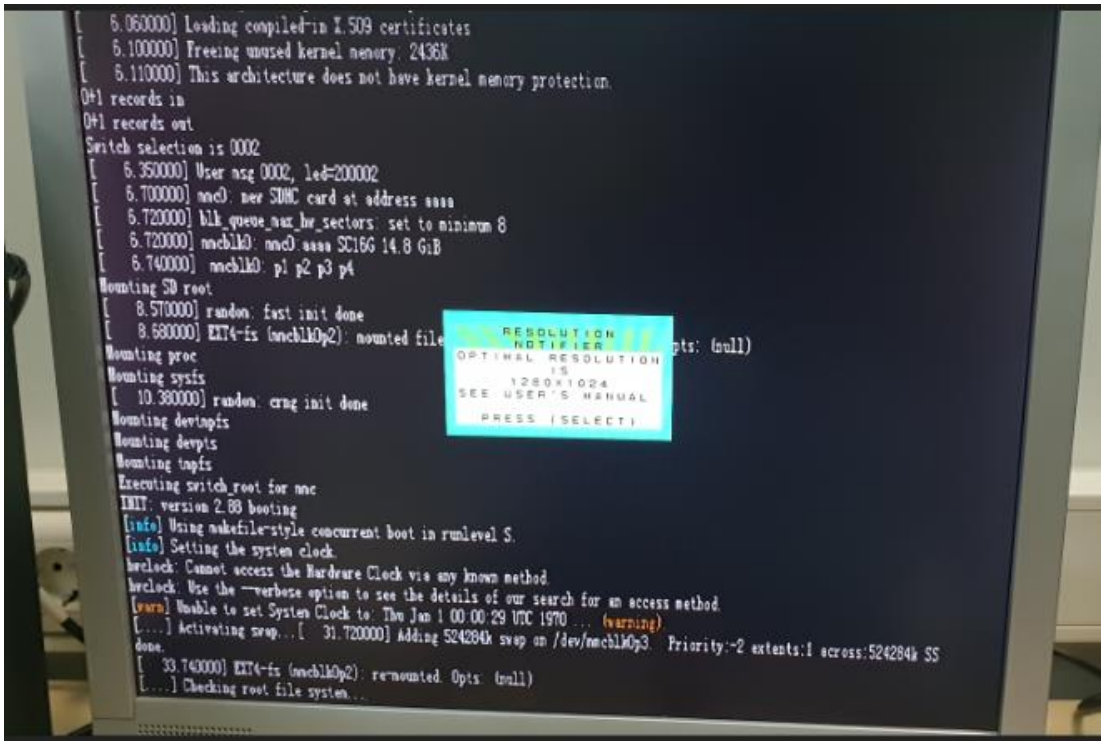


**Figure 2.13 :** There could be some warns. It will continue

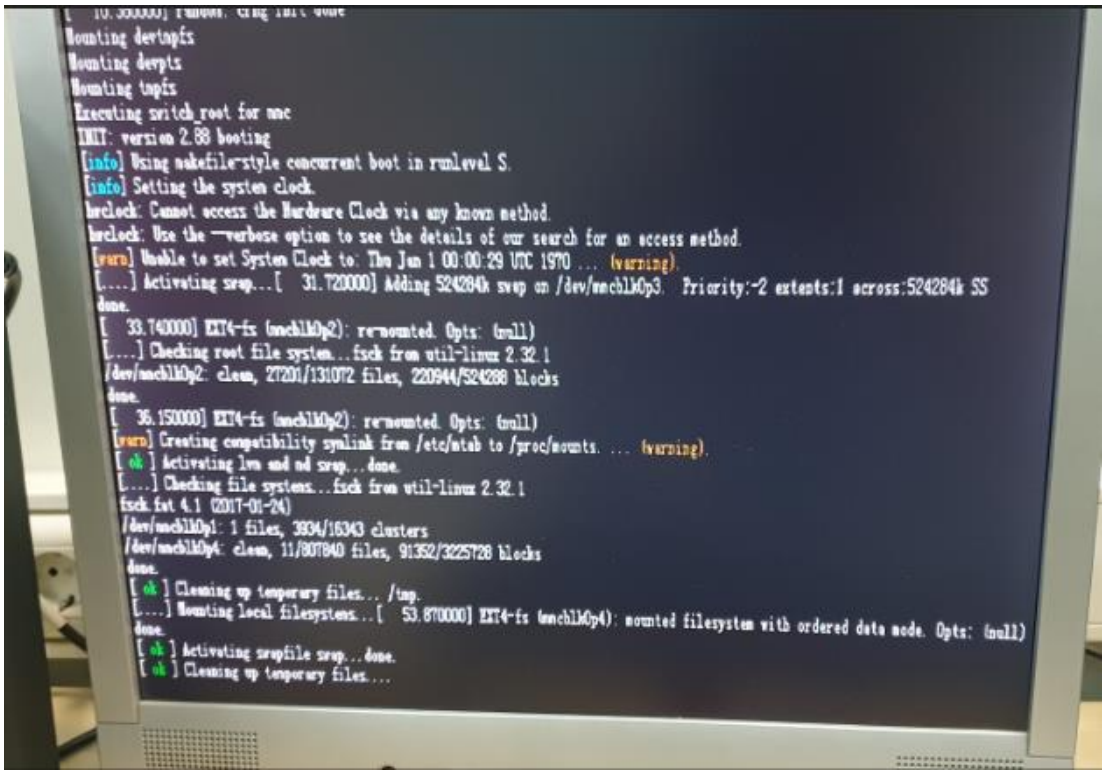


**Figure 2.14 :** Other boot informations

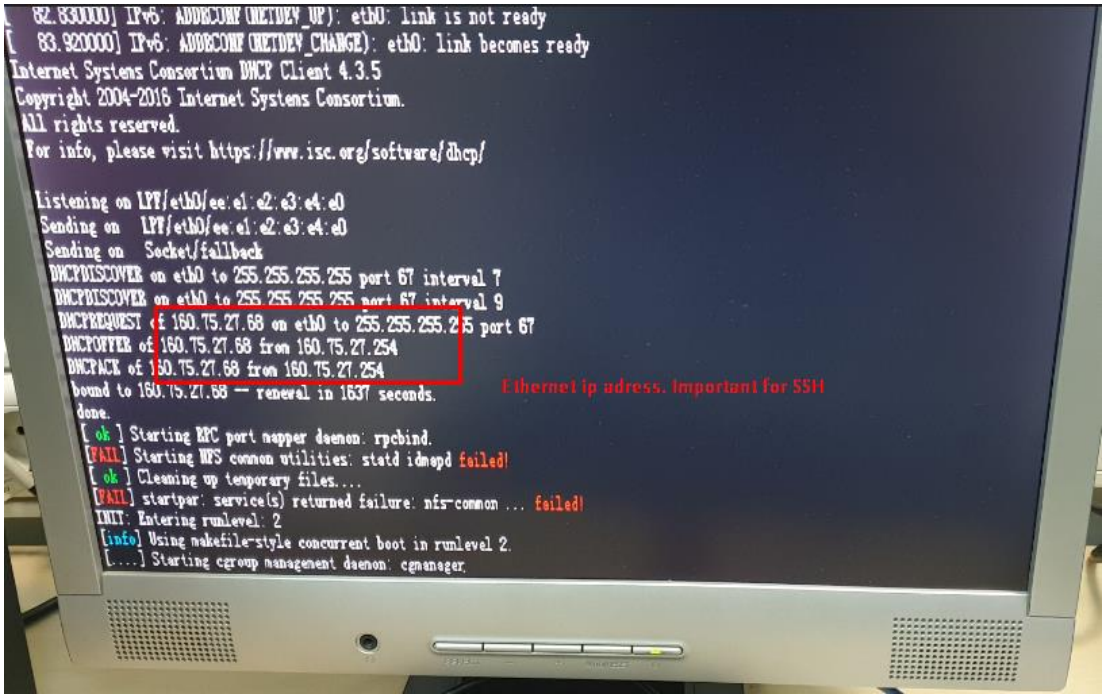


**Figure 2.15 :** Ethernet connection is okey too.

After these steps all boot operations are set and at the last screen user will see login screen like in Figure 2.16.

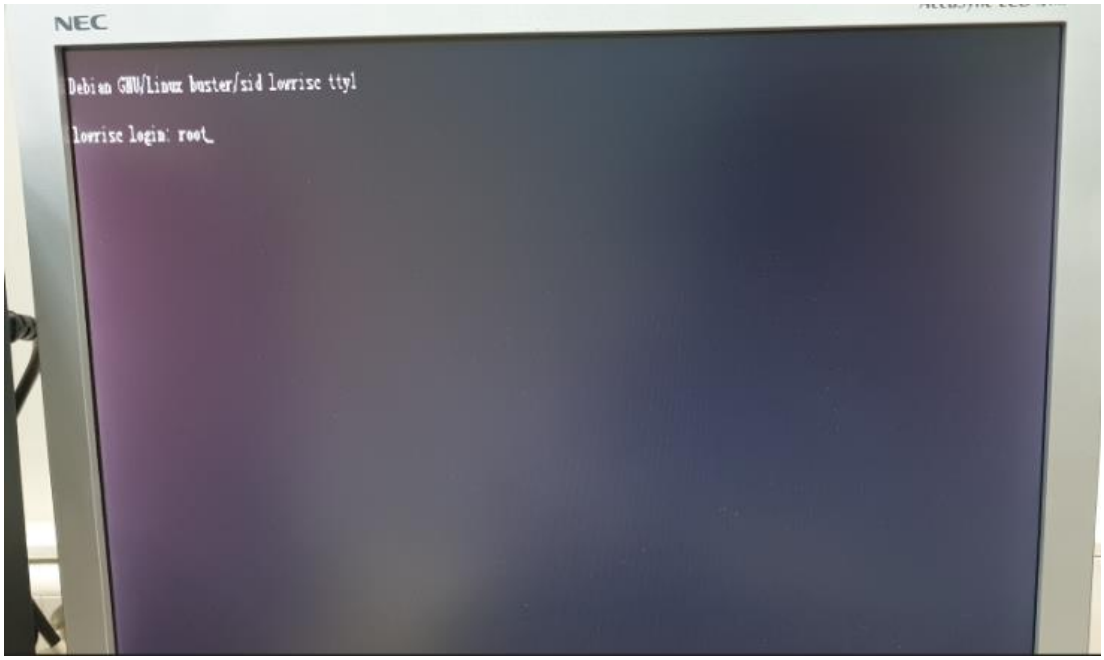


**Figure 2.16 :** Login screen and first login must be root.

First login must be root and givin any root password key is okey to root. After that lowRISC will prompt user for superuser set then creates a normal user which is also has same name of your Ubuntu computer name. In my example it is ***bartu.*** Also a password key must be entered to that user. Same password with root will work which shown in Figure 2.17.

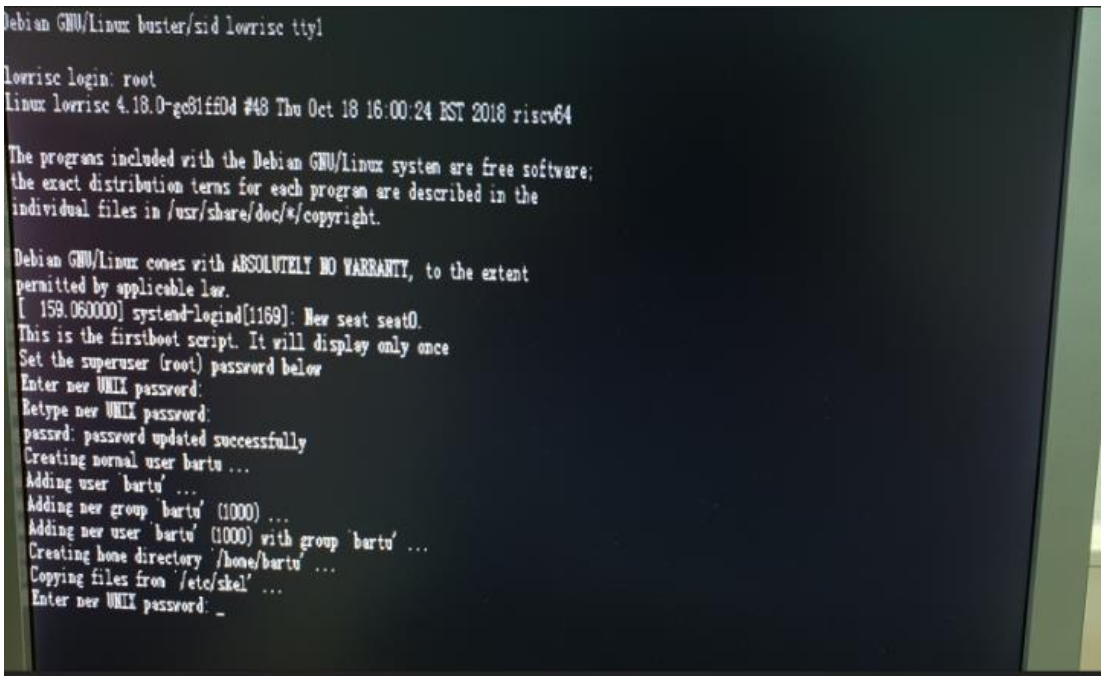


**Figure 2.17 :** Login screen with prompts.

After all these setups, now lowRISC will ask user name and password key to login machine. After entering either root or Ubuntu user, a bash command screen will appear at VGA display which shwon in Figure 2.18.

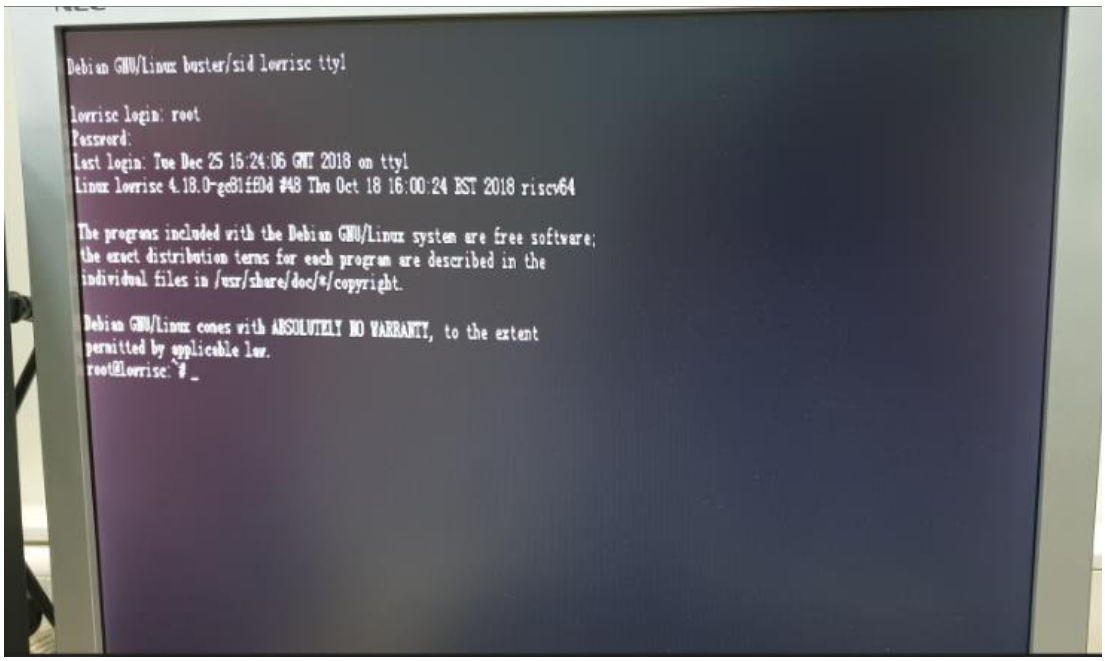


**Figure 2.18 :** Bash command screen.

From now, there is 2 way to use lowRISC. First way is using your Ubuntu machine and connect and control lowRISC with SSH protocol. Other is directly with PS2 FPGA keyboard and VGA monitor. It is recommanded that updating or installing any program to lowRISC must be from FPGA keyboard. SSH protocol gives shows some problems and errors at this stage.

### 2.3 Peripherals of system

LowRISC 0.6 version has some good peripherals to use system like standalone PC. One is a ps2 – USB connection keyboard which is easy to use lowRISC with bash commands. Second peripheral is ethernet cable support. This allows lowRISC to connect internet and get some new apps for ported RISCV architecture. These app list and app repository can be shown at this page [25]. And last peripheral is VGA display port. LowRISC team makes this VGA display as a text display which only encodes and writes some character with support of color. So if user want to show picture from lowRISC to VGA display for now it is not possible to do. Figure 2.19 shows all connected peripherals to FPGA.

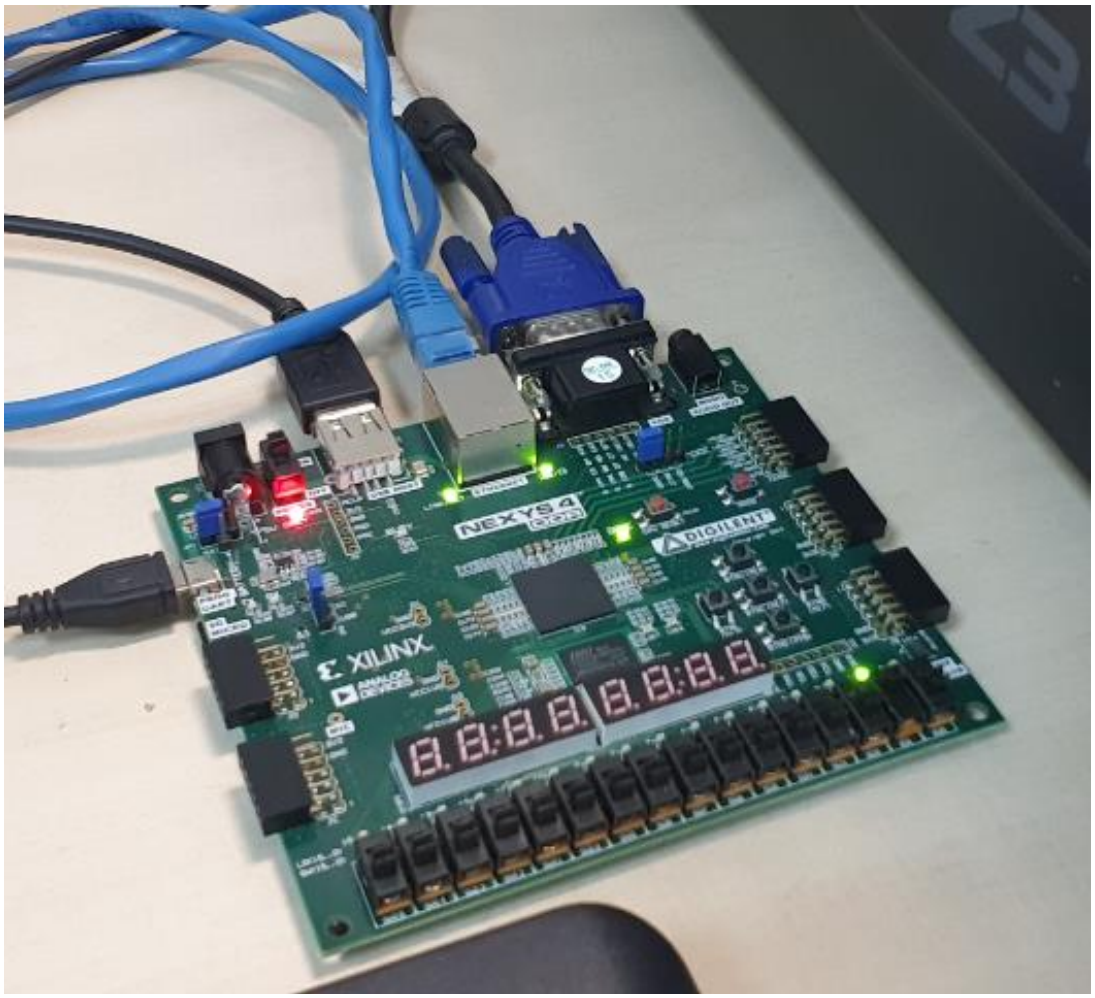


**Figure 2.19 :** Full peripheral connection.

### 2.4 Implementing image filter algorithms to lowRISC

Since package support for RISCV is limited for now, I decided to use pure C language to image filtering and processing operations. But the great constraint is pure C which has no additional library is not able to common image formats. For this purpose a little research shows us there is some formats even pure C could read an image and write a output image file. Project chose to use .ppm file.

#### 2.4.1 PPM format

This format is a lowest common denominator color image format. Format is so inefficient in terms of compressing and image quality. Also format does not include really little information about file. What makes this format good is format makes any image read easy and makes easy to process data of image which is essential for our system. Format in Hex reader shown at this format:

- A “magic number” that gives file type, which is “P6” for ppm.
- CR ASCII.
- Width information as ASCII.
- Whitespace.
- Height information as ASCII.
- CR ASCII.
- Maximum color value of a pixel. General maximum 255 and mininmum 0 in ASCII format.
- CR ASCII.
- And image vector values in ASCII format. Image starts From left to right and Upper Line is first line to start writing. There is no whitespace characters between pixels and rgb values. First value is Red of pixel 1, second is Green of pixel 1, third is blue of pixel 1 and after that goes pixel 2 rgb values and so on.

### 2.4.2 Converting PNG to PPM

Since ppm format is not common as like jpeg or png, input image firtly must converted on ppm from png. However for now there is no converter for lowRISC system, we should convert it at Ubuntu machine and after that give ppm image to lowRISC as input image. For this purpose a program named ImageMagick msut be installed on Ubuntu. To install enter fallowing code:

- *sudo apt-get install imagemagick*
- *sudo apt-get install gcc php-common*
- *sudo apt-get install php-imagick*
- *service apache2 restart*

After correct installation, faloowing command convert any png image to ppm file:

- *mogrify -format ppm input.png*

From now input image is ready to send to lowRISC Linux with SSH.

### 2.4.3 Sending input image to lowRISC

Since ethernet connection is enabled and it is easy to use ssh protocol to send or receive any data between two system at same LAN, it better send ppm data through this way. Only need for ssh connection from Ubuntu to lowRISC is ip address of lowRISC which already taken at Figure 2.15. However with using *ifconfig* command at bash screen of lowRISC will reveal ip address of ethernet again. After ssh command shold entered in bash screen of Ubuntu machine:

- *ssh 160.75.27.68*

It is just for controlling lowRISC bash remotely. Fallowing command must entered image location terminal, will send input.ppm image to lowRISC chosen directory:

- *scp input.ppm 160.75.27.68:/home/bartu/*

It will ask password. Type password and send file.

### 2.4.4 C code for filters

These codes are written in C language and general flow for running program on lowRISC should be in this order at bash command:

- *gcc sobel.c -o program –lm*
- *./program*

GCC first reads program and compile an executeble file which anmes as program. After next command, execution is occurs. But do not try to compile program in Ubuntu machine because exe files are system spesific. This means only running exe on lowRISC will not work because it was compiled an x64 machine. So all commands must be entered at lowRISC's bash comand screen. Figure 2.20 will show input image which is widely used for image processing application as subject.

**Figure 2.20 :** Example input image.

### 2.4.4.1 Common parts of filter in codes

Fallowing Figures which are 2.21, 2.22, 2.23, 2.24 are same for all filter codes.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

int main(void){

    FILE *file;
    FILE *fp = fopen("output.ppm", "wb"); // write output image file pointer
    file = fopen("input.ppm", "r"); // read input image file pointer
    int i = 0;
    int threshold; // some calcualtion variable
    printf("Enter threshold (Ex:50) => "); // get rgb difference to get red data
    scanf("%d",&threshold);
    int width; // image width
    int heigth; // image height

    char c;
    char strx[5];
    char stry[5];
    char strd[3];
```

**Figure 2.21 :** Reading input and writing output file pointers

```
//This part reads input.ppm and also same time creates same output image format.
while (c != 10)
{
        c = getc(file);
    fprintf(fp, "%c", c);
}
fprintf(fp, "%c", c);
int j = 0;
while(c != ' ')
{
    c = getc(file);
    fprintf(fp, "%c", c);
    strx[j] = c;
    j++;
}
width = atoi(strx); // getting image width data
j = 0;
c = strx[0];
while(c != 10)
{
    c = getc(file);
    fprintf(fp, "%c", c);
    stry[j] = c;
    j++;
}
heigth = atoi(stry); // getting image height data
c = getc(file);
fprintf(fp, "%c", c);

c = getc(file);
i++;
fprintf(fp, "%c", c);

c = getc(file);
fprintf(fp, "%c", c);

c = getc(file);
fprintf(fp, "%c", c);
```

**Figure 2.22 :** Reading image information and writing output file same informations.

```
unsigned char rgbc ;
int datar[width*heigth]; // all red pixels
int datag[width*heigth]; // all green pixels
int datab[width*heigth]; // all blue pixels
int datat[width*heigth]; // total pixel values
int gray;
int grayx[width*heigth];
int grayy[width*heigth];
int grayt[width*heigth];
i = 0;
j = 0;
int k = 0;
// Storing RGB datas to variables
while(i < width*heigth*3)
{
    rgbc = getc(file);
    if(i%3 == 0) datar[k] = rgbc;
    else if(i%3 == 1) datag[k] = rgbc;
    else if(i%3 == 2)
    {
        datab[k] = rgbc;
        k++;
    }
    i++;
}
```

**Figure 2.23 :** Some variables for rgb datas and reading image and storing to variables.

```
for (int i = 0; i < k; i++)
{
gray = (datar[i] + datag[i] + datab[i]) / 3; // grayscale calculation.
```

**Figure 2.24 :** Grayscale of image if needed.

### 2.4.4.2 Red Only filter

Fallowing Figure 2.25 shows Red mask algorithm and Figure 2.26 shows output of image.

```
if (datar[i] > datag[i] + threshold && datar[i] > datab[i] + threshold) // if upper than threshold write output.ppm all rgb datas
    fprintf(fp, "%c%c%c", datar[i],datag[i],datab[i]);

else fprintf(fp, "%c%c%c", gray,gray,gray); // if lower than threshold write output.ppm grayscale data
```

**Figure 2.25**

**Figure 2.26**

### 2.4.4.3 Green only filter

Fallowing Figure 2.27 shows Green mask algorithm and Figure 2.28 shows output of image.

```
if (datag[i] > datar[i] + threshold && datag[i] > datab[i] + threshold)
    fprintf(fp, "%c%c%c", datar[i],datag[i],datab[i]);

else fprintf(fp, "%c%c%c", gray,gray,gray);
```

**Figure 2.27**

**Figure 2.28**

#### 2.4.4.4 Blue Only filter

Fallowing Figure 2.29 shows Blue mask algorithm and Figure 2.30 shows output of image.

```
if (datab[i] > datar[i] + threshold && datab[i] > datag[i] + threshold)
    fprintf(fp, "%c%c%c", datar[i],datag[i],datab[i]);

else fprintf(fp, "%c%c%c", gray,gray,gray);
```

**Figure 2.29**

**Figure 2.30**

### 2.4.4.5 Sobel Operation filter

Sobel operation is used for edge detection. Two kernels is used to make filtering. First kernel is used for vertical lines and other is used for horizontal line detection. After both of them applied independantly each pixel will blend with ratio. Figure 2.31 and Figure 2.32 shows kernel values for convolution.

```
int kernelx11 = -1*p, kernelx12 = 0*p, kernelx13 = +1*p;
int kernelx21 = -2*p, kernelx22 = 0*p, kernelx23 = +2*p;
int kernelx31 = -1*p, kernelx32 = 0*p, kernelx33 = +1*p;
```

**Figure 2.31 :** Horizontal kernel

```
int kernely11 = -1*p, kernely12 = -2*p, kernely13 = -1*p;
int kernely21 = 0*p, kernely22 = 0*p, kernely23 = 0*p;
int kernely31 = 1*p, kernely32 = 2*p, kernely33 = +1*p;
```

**Figure 2.32 :** Vertical kernel

Next part is convole each kernel on image which shown in Figure 2.33

```
for (int i = 0; i < k; i++)
{
    if (i + width + width + 2 < k)
    {
        grayx[i] = (kernelx11 * gray[i] + kernelx12 * gray[i + 1] + kernelx13 * gray[i + 2]
+ kernelx21 * gray[i + width] + kernelx22 * gray[i + width + 1] + kernelx23 * gray[i + width + 2]
            + kernelx31 * gray[i + width + width] + kernelx32 * gray[i + width + width + 1] + kernelx33 * gray[i + width + width + 2]);
        grayy[i] = (kernely11 * gray[i] + kernely12 * gray[i + 1] + kernely13 * gray[i + 2]
            + kernely21 * gray[i + width] + kernely22 * gray[i + width + 1] + kernely23 * gray[i + width + 2]
            + kernely31 * gray[i + width + width] + kernely32 * gray[i + width + width + 1] + kernely33 * gray[i + width + width + 2]);
    }
    else
    {
        grayx[i] = 0;
    }
    grayt[i] = (int) sqrt((double)(grayx[i] * grayx[i] + grayy[i] * grayy[i]));
if(grayt[i] >= 255) datat[i] = 255;
else if(grayt[i] <= 0) datat[i] = 0;
else datat[i] = grayt[i];
fprintf(fp, "%c%c%c", datat[i],datat[i],datat[i]);
```

**Figure 2.33 : Convolution operation code and blending operation horizontal and vertical lines**

Output image shown in Figure 2.34.

**Figure 2.34 :** Sobel operation output.

### 2.4.4.6 RGB edge detection filter

It is same as sobel but in rgb mode. Kernel can be seen in Figure 2.35 Algorithm can be seen Figure at 2.36.

```
int kernelx11 = -1-p, kernelx12 = -1-p, kernelx13 = -1-p;
int kernelx21 = -1-p, kernelx22 = 8+8*p, kernelx23 = -1-p;
int kernelx31 = -1-p, kernelx32 = -1-p, kernelx33 = -1-p;
```

**Figure 2.35**

```
for (int i = 0; i < k; i++)
{
    if (i + width + width + 2 < k)
    {
        grayx[i] = (kernelx11 * datar[i] + kernelx12 * datar[i + 1] + kernelx13 * datar[i + 2]
+ kernelx21 * datar[i + width] + kernelx22 * datar[i + width + 1] + kernelx23 * datar[i + width + 2]
            + kernelx31 * datar[i + width + width] + kernelx32 * datar[i + width + width + 1] + kernelx33 * datar[i + width + width + 2]);
        grayy[i] = (kernelx11 * datag[i] + kernelx12 * datag[i + 1] + kernelx13 * datag[i + 2]
            + kernelx21 * datag[i + width] + kernelx22 * datag[i + width + 1] + kernelx23 * datag[i + width + 2]
            + kernelx31 * datag[i + width + width] + kernelx32 * datag[i + width + width + 1] + kernelx33 * datag[i + width + width + 2]);
        grayt[i] = (kernelx11 * datab[i] + kernelx12 * datab[i + 1] + kernelx13 * datab[i + 2]
+ kernelx21 * datab[i + width] + kernelx22 * datab[i + width + 1] + kernelx23 * datab[i + width + 2]
            + kernelx31 * datab[i + width + width] + kernelx32 * datab[i + width + width + 1] + kernelx33 * datab[i + width + width + 2]);
    }
    else
    {
        grayx[i] = 0;
        grayy[i] = 0;
        grayt[i] = 0;
    }
if(grayx[i] >= 255) grayx[i] = 255;
else if(grayx[i] <= 0) grayx[i] = 0;
else grayx[i] = grayx[i];

if(grayt[i] >= 255) grayt[i] = 255;
else if(grayt[i] <= 0) grayt[i] = 0;
else grayt[i] = grayt[i];

if(grayy[i] >= 255) grayy[i] = 255;
else if(grayy[i] <= 0) grayy[i] = 0;
else grayy[i] = grayy[i];
```

**Figure 2.36 :** Convulution and reduction of lower 0 and upper 255 values.

Output file can be seen in Figure 2.37.

**Figure 2.37**

### 2.4.4.7 Sharpen filter

This filter makes images looks sharper than normal. After RGB edge detection filter all convolution operations are same C code. They will not shown fallowing filters. Kernel of Sharpen is shown at Figure 2.38 and output shown in Figure 2.39.

```
int kernelx11 = 0, kernelx12 = -1-p, kernelx13 = 0;
int kernelx21 = -1-p, kernelx22 = 5+4*p, kernelx23 = -1-p;
int kernelx31 = 0, kernelx32 = -1-p, kernelx33 = 0;
```

**Figure 2.38**

**Figure 2.39**

### 2.4.4.8 Emboss filter

This filter makes images looks like embossed in 3D view. Kernel of emboss filter is shown at Figure 2.40 and output shown in Figure 2.41.

```
int kernelx11 = -2*p, kernelx12 = -1*p, kernelx13 = 0;
int kernelx21 = -1*p, kernelx22 = 1, kernelx23 = 1*p;
int kernelx31 = 0, kernelx32 = 1*p, kernelx33 = 2*p;
```

**Figure 2.40**

**Figure 2.41**

### 2.4.4.9 Gaussian Blur filter

This filter makes images blurrish. It is widely used after or before edge detection to get proper edges from image. Kernel of gaussian blur filter is shown at Figure 2.42 and output shown in Figure 2.43.

```
float kernelx11 = 0.0675, kernelx12 = 0.125, kernelx13 = 0.0675;
float kernelx21 = 0.125, kernelx22 = 0.25, kernelx23 = 0.125;
float kernelx31 = 0.0675, kernelx32 = 0.125, kernelx33 = 0.0675;
```

**Figure 2.42**

**Figure 2.43**

### 2.4.5 Autamatization of filtering image algorithm

Since converting and sending image and applying filter is easy to forget process, a general sh code written and can be run on Linux lowRISC. Fallowing code converts input.png to ppm, ask user for which filter will applied and writes time to bash screen to see starting time and applys filter. After that output image is written and again time is shown at bash to see how much time elapsed for processing part. It converts output image to png again. Then removes ppm files from location to better harware storage control.

```
cd /home/bartu/Desktop
mogrify -format ppm input.png
echo "Creating exe file"
echo "Enter 1 for Sobel"
echo "Enter 2 for OnlyRed"
echo "Enter 3 for OnlyGreen"
echo "Enter 4 for OnlyBlue"
echo "Enter 5 for RGBEdges"
echo "Enter 6 for Sharpen"
echo "Enter 7 for Emboss"
```

```
echo "Enter 8 for Gaussian Blur"
read choice
if [ "$choice" -eq 1 ];
then
   now=$(date +"%T")
   echo "Generating Sobel! - Time : $now"
   gcc sobel.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "Sobel operation finished - Time : $now"
elif [ "$choice" -eq 2 ];
then
   now=$(date +"%T")
   echo "Generating OnlyRed! - Time : $now"
   gcc only_red.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "OnlyRed operation finished - Time : $now"
elif [ "$choice" -eq 3 ];
then
   now=$(date +"%T")
   echo "Generating OnlyGreen! - Time : $now"
   gcc only_green.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "OnlyGreen operation finished - Time : $now"
elif [ "$choice" -eq 4 ];
then
   now=$(date +"%T")
   echo "Generating OnlyBlue! - Time : $now"
   gcc only_blue.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "OnlyBlue operation finished - Time : $now"
elif [ "$choice" -eq 5 ];
then
   now=$(date +"%T")
   echo "Generating RGBEdges! - Time : $now"
   gcc RGBEdgeDetector.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "RGBEdges operation finished - Time : $now"
elif [ "$choice" -eq 6 ];
then
   now=$(date +"%T")
   echo "Generating Sharpened! - Time : $now"
   gcc Sharpen.c -o program -lm
   ./program
   now=$(date +"%T")
   echo "Sharpen operation finished - Time : $now"
elif [ "$choice" -eq 7 ];
then
   now=$(date +"%T")
   echo "Generating Embossed! - Time : $now"
   gcc Emboss.c -o program -lm
```

```
    ./program
    now=$(date +"%T")
    echo "Emboss operation finished - Time : $now"
elif [ "$choice" -eq 8 ];
then
    now=$(date +"%T")
    echo "Generating Gaussian Blur! - Time : $now"
    gcc gb.c -o program -lm
    ./program
    now=$(date +"%T")
    echo "Gaussian Blur! operation finished - Time : $now"
fi;
mogrify -format png output.ppm
echo "output.png file created to current folder"
rm output.ppm
rm input.ppm
rm program
```

Usage on bash command screen can be seen on Figure 2.44.

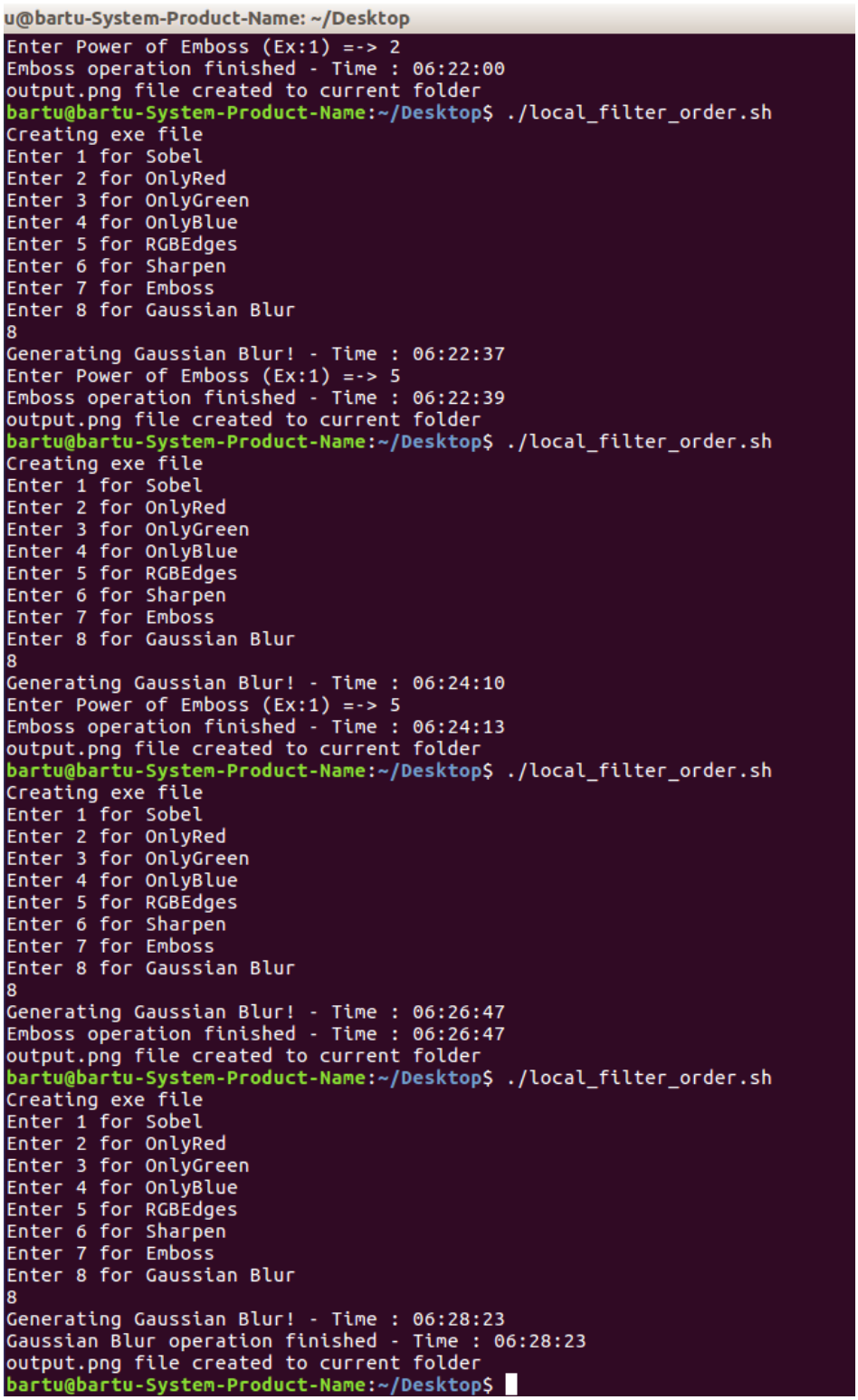


**Figure 2.44 :** Usage of Automatization code.

# 3. REALISTIC CONSTRAINTS AND CONCLUSIONS

## 3.1 Practical Application of this Project

In real life, lowRISC SoC can be used on FPGA as well as on prototyping and testing on FPGA and after design on a custom chip. In the current version of the open source coded SoC, there will be no access to the general purpose input output pins, but in the following versions it will come as update. In addition, the advantage of open sourced, chip developers as they want to customize the possibility that they want, they can use this chip some other spesific industries like automation and aviation.

## 3.2 Realistic Constraints

### 3.2.1 Social, environmental and economic impact

The biggest advantage of open source processors is that developers don't have to design a unique processor. This advantage will give the developers enormous benefits in terms of time and resources. Both countries that cannot produce their own chips and developers will benefit greatly in the advancement of technology. In addition, the standardization of being open source, thanks to the innovations offered by any company or developer has always been open to the opportunity to follow the global technology.

### 3.2.2 Cost analysis

First of the my necessary materials is FPGA evaluation boards (one for each member of group) that faculty management meets. But In addition to the FPGA evaluation boards, I also used Vivado development environment that is used for a implementation LowRISC chip. Since this project worked with full open sourced software and hardware and operating system like Linux Ubuntu, there is no cost for any of hardware or software environment.

Other costs for the project is similar to Form 3.

### 3.2.3 Standards

The purpose of this project is to show the level of applications that can be done using open source processors. Open source processors, which are still in development, are already in compliance with many standards with their low power consumption.

Moreover, since they are within the framework of legal standards, the standards here are completely free to the extent determined by the producers. Since the project is run by many developers, it is already designed to comply with IEEE standards. However, as there are no studies in our country, standards will be created over time and my project will be included in these standards.

### 3.2.4 Health and safety concerns

This project has not aimed to design and implement any harmful or risky product that may affect users. Created systems impelemented in FPGA and it is totally safe.

## 3.3 Future Work and Recommendations

- Since there is no "GPIO" access in this version, the camera cannot be connected. In future work, either wait for the new version and access the "GPIO", or connect the camera to the lowRISC without waiting for the new version with implementation of spesific camera interface.
- Any autonomous outgoing vehicle can be made using image processing filters and also using "GPIO" access. Thus, LowRISC, which is an open source, can be tested, observed and improved on such issues.
- Software applications for performing image filtering steps can also be performed hardware in terms of targeting certain customized applications. In this way, the hardware can be designed for a new purpose by integrating the lowRISC into a customized image processing chip.

## REFERENCES


[1] L. Team, "About LowRISC," LowRISC, 08 2018. [Online]. Available: https://www.lowrisc.org/about/. [Accessed 26 12 2018].

[2] R.-V. Foundation, "RISC-V Cores and SoC," RISC-V, 05 2017. [Online]. Available: https://riscv.org/risc-v-cores/#. [Accessed 28 12 2018].

[3] D. Inc., "Nexys 4 DDR," Digilent Inc., 2012. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start. [Accessed 12 11 2018].

[4] Opensource.com, "Opensource.com," 1 1 2019. [Online]. Available: https://opensource.com/resources/what-open-source.

[5] A. R. Core, "32bitMicro," 02 02 2015. [Online]. Available: https://web.archive.org/web/20150202024204/http://32bitmicro.com/component/content/article/457-soft-processor/973-amber-risc-core. [Accessed 25 12 2018].

[6] K. Afef, E. Wajih, R. Velazco and R. Tourki, "An exhaustive analysis of SEU effects in the SRAM memory of soft processor," vol. 13, pp. 58-68, 2018.

[7] Oracle, "About OpenSPARC," Oracle, December 2005. [Online]. Available: https://www.oracle.com/technetwork/systems/opensparc/opensparc-overview-1562924.html. [Accessed 25 12 2018].

[8] R.-V. Foundation, "About the RISC-V ISA," RISC-V Foundation, 25 11 2018. [Online]. Available: https://riscv.org/risc-v-isa/.

[9] B. Keller, "Handouts for CS250: VLSI Systems Design," 17 9 2013. [Online]. Available: http://www-inst.eecs.berkeley.edu/~cs250/fa13/handouts/lab2-riscv.pdf#13. [Accessed 24 12 2018].

[10] LowRISC, "Overview of the ethernet infrastructure," LowRISC, 12 2017. [Online]. Available: https://www.lowrisc.org/docs/ethernet-v0.5/overview/. [Accessed 28 12 2018].

[11] LowRISC, "Frequently asked questions," LowRISC, 6 2018. [Online]. Available: https://www.lowrisc.org/docs/current-release-faq/. [Accessed 27 12 2018].

[12] LowRISC, "Rocket core overview," LowRISC, 4 2015. [Online]. Available: https://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/. [Accessed 25 12 2018].

[13] A. P. D. G. A. (Shahab), "Digital Image Processing," University of Tartu, 1 2014. [Online]. Available: https://sisu.ut.ee/imageprocessing/book/1.

[14] EngineersGarage, "Introduction to Image Processing," EngineersGarage, 5 2011. [Online]. Available: https://www.engineersgarage.com/articles/image-processing-tutorial-applications. [Accessed 29 12 2018].

[15] C. B. Güngör, Y. Öndeş, T. T. Sarı and B. Uçkun, "Instructoıon Set Extension of Some Processors for Secure IoT Implementation," Istanbul, 2018.

[16] T. Liu, G. Shi, L. Chen, F. Zhang and Y. Y. a. J. Zhang, "TMDFI: Tagged Memory Assisted for Fine-Grained Data-Flow Integrity Towards Embedded Systems Against Software Exploitation," in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering*, New York, 2018.

[17] A. Ramos, A. Ullah and P. R. a. J. A. Maestro, "Efficient Protection of the Register File in Soft-Processors Implemented on Xilinx FPGAs," 1 2 2018. [Online]. Available:

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8008792&isnumber=8255532. [Accessed 1 1 2019].

[18] Ubuntu, "Ubuntu," Canonical, 28 12 2018. [Online]. Available: https://www.ubuntu.com/.

[19] LowRISC, "Download and install Debian," LowRISC, 5 2018. [Online]. Available: https://www.lowrisc.org/docs/download-install-debian/. [Accessed 2 11 2018].

[20] Xilinx, "Vivado," Xilinx, 30 12 2018. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html.

[21] Digilent, "Digilent Adept 2," Digilent, 2 2015. [Online]. Available: https://store.digilentinc.com/digilent-adept-2-download-only/.

[22] Kitware, "About CMake," Kitware, 25 12 2018. [Online]. Available: https://cmake.org/.

[23] LowRISC, "Structure of the git repository," LowRISC, 5 2018. [Online]. Available: https://www.lowrisc.org/docs/download-the-code/. [Accessed 10 10 2018].

[24] LowRISC, "Getting started," LowRISC, 5 2018. [Online]. Available: https://www.lowrisc.org/docs/getting-started/. [Accessed 20 12 2018].

[25] "Debian Ports," 23 03 2018. [Online]. Available: http://ftp.ports.debian.org/debian-ports/. [Accessed 22 12 2018].

**CURRICULUM VITAE**

**Name Surname** **:** Bartu Sürer

**Place and Date of Birth** **:** Sakarya 24.08.1994

**E-Mail** **:** surerb16@itu.edu.tr

Bartu Sürer finished primary school at Ahmet Akkoç primary school in Adapazarı and high school at Sakarya Anatolian High School in Adapazarı. He is currently senior year student at Electronics and Communication Engineering in Istanbul Technical University Electrical-Electronics Faculty. He completed his interships at Arçelik A.Ş. in Tuzla and Otokar Otomotiv ve Savunma Sanayi A.Ş in Sakarya.