



**REPUBLIC OF TURKEY
OSTİM TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING
COMPUTER ENGINEERING
MASTER'S PROGRAM**

**VIDEO ENCODING ON EMBEDDED LINUX SYSTEMS
PERFORMANCE MEASUREMENT AND LITERATURE
COMPARISON**

MASTER'S THESIS

**PREPARED BY
HÜSEYİN BÜYÜKGÜZEL**

**THESIS SUPERVISOR
DR. RAYAN ABRİ**

ANKARA-2024

THESIS ACCEPTANCE AND APPROVAL

This study, titled “Video Encoding on Embedded Linux Systems Performance Measurement and Literature Comparison” and submitted by Hüseyin BÜYÜKGÜZEL was found successful as a result of the thesis defense on 18/01/2024 and accepted as a Master's Thesis by our jury.

Date of Approval: 18/01/2024

Jury Member: Doç. Dr. Harun Artuner
Hacettepe University

Jury Member: Dr. Ahmet Özdil
Ostim Technical University

Thesis Supervisor: Dr. Rayan Abri
Ostim Technical University

APPROVAL

I hereby approve that this study, accepted by the jury, fulfills the requirements for being a Master's Thesis

...../...../20....

Dr. Pelin Tören ÖZGÜN
Institute Director

DECLARATION

I hereby declare that I have given Ostim Technical University the permission to archive all or any part of my Master's thesis approved by the Institute in printed or digital format and make it accessible under the conditions specified below. With this permission, all intellectual property rights other than the usage rights granted to the University will remain with me and the usage rights of all or a part of my thesis for future studies (article, book, license, patent, etc.) will belong to me alone. I declare and undertake that my thesis is entirely my own work, that I do not violate the rights of others and that I am the sole authorized owner of my thesis. I undertake that I use the copyrighted resources with written permission, which must be used with written permission from their owners, and to submit copies of the permissions to the University upon request. Within the scope of the "Directive on Collecting, Organizing and Opening to Access The Theses and Dissertations in Electronic Environment" published by the Council of Higher Education, my thesis is made available on the YÖK National Thesis Center and Ostim Technical University Open Access System, except for the following conditions.

- ☐ With the decision of the Institute / Faculty Administrative Board, the open access of my thesis has been postponed for 2 years from my graduation date.¹
- ☐ With the reasoned decision of the Graduate School / Faculty Administrative Board, the opening of my thesis has been postponed for **maximum of 6 months** from the date of my graduation.²
- ☐ A confidentiality decision has been made regarding my thesis.^{3 4}

Date :

Signature :

¹ ARTICLE 6(1) In the event that a patent application is made for a graduate thesis or the patenting process is ongoing, upon the recommendation of the thesis advisor and the approval of the graduate school department, the graduate school or faculty executive board may decide to postpone the access to the thesis for two years.

² ARTICLE 6(2) For theses that use new techniques, materials and methods, that have not yet turned into articles or are not protected by methods such as patents, and that contain information and findings that may create unfair gain for third parties or institutions if shared on the internet, the thesis may be prevented from being accessed for a period not exceeding six months upon the recommendation of the thesis advisor and the approval of the institute department, with the reasoned decision of the institute or faculty board of directors.

³ ARTICLE 7(1) The decision of confidentiality regarding graduate theses related to national interests or security, security, intelligence, defense and security, health, etc. is made by the institution where the thesis is conducted. The confidentiality decision regarding graduate theses prepared within the framework of a cooperation protocol with institutions and organizations is made by the university board of directors upon the recommendation of the relevant institution and organization and the approval of the institute or faculty. Theses for which a confidentiality decision is made are notified to the Council of Higher Education.

⁴ ARTICLE 7(2) Theses for which a confidentiality decision has been made are kept by the institute or faculty within the framework of confidentiality rules during the confidentiality period, and are uploaded to the Thesis Automation System if the confidentiality decision is lifted.

OSTIM TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF APPLIED SCIENCES MASTER'S THESIS
ORIGINALITY REPORT

Thesis Title: Video encoding on Embedded Linux Systems Performance measurement and literature comparison

Student Name Surname: Hüseyin BÜYÜKGÜZEL

Thesis Supervisor Title Name Surname: Dr. Rayan ABRİ

Department: Computer Engineering

Program: Computer Engineering

Date: 25 / 01 / 2024

The part of my Master's thesis, which consists of Introduction, Main Chapters and Conclusion and consists of 67 pages in total, was examined by my thesis advisor and myself on 25/01/2024 by the Turnitin plagiarism detection program.

According to the originality report, the similarity score of my thesis is 8%.

Applied filters:

1. Bibliography (excluding)
2. Citations (excluding)
3. Parts of text with less than five (5) words overlapping (excluding)

I declare that my thesis does not contain any plagiarism; that I accept all kinds of legal responsibility that may arise in the event that the contrary is detected and that the information I have given above is correct.

Student Signature:

APPROVAL

Date: / /.....

Supervisor's Title Name Surname and Signature

ETHICAL DECLARATION

I declare that this study is an original study, that I act in accordance with scientific ethics and rules in the preparation, data collection, analysis, presentation of information and all other stages of the study, that I have obtained all the document information in the study within the framework of academic ethics and rules, that I have presented all visual, audio, and written information and results in accordance with the scientific rules of ethics, that I have not made any falsifications in the data I have used, that I have referred to the sources I have used in accordance with scientific norms, that my thesis has been written by me and is original, except for the cases where I cited sources, that it has been produced by me under the supervision of my thesis advisor Dr. Rayan ABRI and written in accordance with Ostim Technical University thesis writing guide.

Student Signature

Hüseyin BÜYÜKGÜZEL

.../.../20..

ACKNOWLEDGEMENTS

Firstly, I would like to thank to Dr. Rayan ABRİ my supervisor for all his effort and knowledge about this thesis. He was always polite and supportive to me.

I extend my thanks to Dr. Ahmet Çağrı ARLI, my director at work, for being a friend even if he is an executive. He always tried to be best for collages and when needed he stand with them. The most difficult phase, writing a thesis while working is become possible with him.

I am so grateful that I have such awesome friends Anıl KARACA and Fahri ESEN, whose also supported the project and helped whenever needed.

I would also like to thank my wife Ayşenur Habibe BÜYÜKGÜZEL for being the most supportive person in my life and always tried to help me whatever life brings.

I express my love and thanks to my father who never lost hope on me, my mother who always made me feel love and my aunt who always be my best friend and life coach, my collages whose are always made me feel better through this process.

.../.../20..

Hüseyin BÜYÜKGÜZEL

ÖZ

Yazar Adı ve Soyadı : Hüseyin Büyükgüzel
Üniversite : OSTİM Teknik Üniversitesi
Enstitü : Fen Bilimleri Enstitüsü
Program Adı : Bilgisayar Mühendisliği
Tezin Türü : Yüksek Lisans Tezi
Sayfa Sayısı : 67
Tarihi : 2024

GÖMÜLÜ LINUX SİSTEMLERİ ÜZERİNDE VIDEO KODLAMA ÖLÇÜMÜ VE LİTERARÜR KARŞILAŞTIRMASI

Bu tezin konusu, dünyanın en popüler endüstriyel üretim yöntemlerinden biri olan Embedded Linux geliştirme ortamında video sıkıştırma ile aynı işlemin bilgisayar ile gerçekleştirilmesi arasındaki performans farklılıklarını benzersiz bir şekilde ortaya koymaktadır. Bu tez ile gömülü linux ve bilgisayar arasındaki fark gösterilecektir. Test ve araştırmalardan çıkacak çıktılar, belirli amaçlar için hangisinin daha karlı ve verimli olduğu konusuna ışık tutacaktır. Tez yazılırken literatür taraması, gerçek veriler yardımıyla karşılaştırma, test ortamı oluşturma ve elde edilen verilerden çıktı yazma yöntemleri kullanılmıştır.

Anahtar Sözcükler: Yocto projesi, Petalinux, h264, Gömülü Linux, Video kodlama

ABSTRACT

Thesis : Hüseyin BÜYÜKGÜZEL
University : OSTİM Technical University
Institute : Graduate School of Natural and Applied Sciences
Program's Name : Computer Engineering
Thesis Type: : Master
Pages : 67
Year : 2024

VIDEO ENCODING ON EMBEDDED LINUX SYSTEMS PERFORMANCE MEASUREMENT AND LITERATURE COMPARISON

The subject of this thesis reveals the performance differences between video compression in an embedded Linux development environment by using the Video Codec Unit and performing the same process with a computer. In this manner, differences will be shown between embedded Linux and computers in the comparison of inputs and outputs. The output that will come from tests and research will enlighten the topic and, for specific purposes, determine which one is more profitable and efficient. For this purpose, a test environment has been created by designing a carrier card that holds a System on Chip (SoC), carries out needed peripherals, and supplies power for it. Then we built a Petalinux environment for controlling the flow between digital design and the Linux environment. Then we encoded and decoded a video for losses on this system. In the last phase, we compared results and analyzed data with known algorithms for quality compression.

Keywords: Petalinux Project, Video Codec Unit, Embedded Linux Development, Video compression

CONTENTS

THESIS ACCEPTANCE AND APPROVAL	i
DECLARATION	ii
ETHICAL DECLARATION	iv
ACKNOWLEDGEMENTS	v
ÖZ	vi
ABSTRACT	vii
CONTENTS	viii
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
ABBREVIATIONS.....	xiii
1. INTRODUCTION	1
2. BACKGROUND	2
2.1. Application-specific integrated circuit (ASIC).....	2
2.2. Field Programmable Gate Array (FPGA).....	3
2.3. Microprocessors.....	4
2.4. System on Chip (SoC)	5
2.5. ARM Architecture	6
2.6. Video Compressing	7
2.7. Video Codec Unit	17
2.8. Video Quality Metrics	18
2.8.1. Peak signal-to-noise ratio (PSNR).....	18
2.8.2. Video Multimethod Assessment Fusion (VMAF).....	19
2.9. Embedded Linux.....	20
2.10. Petalinux Project.....	23
2.11. Brief Linux Terminologies	24
2.11.1. Kernel	24
2.11.2. Unix Shell	24
3. METHOD	30
3.1. Hardware	30
3.2. Digital Design.....	30
3.3. Gstreamer Pipeline	33

4. COMPARISON BETWEEN EMBEDDED LINUX AND COMPUTER.....	38
5. CONCLUSION	43
REFERENCES	46



LIST OF TABLES

Table 2.1. Comparison of PSNR values of codecs with different videos [25].....	16
---	----



LIST OF FIGURES

Figure 2.1. Apple 2e Personal Computer from 1980s [2].....	2
Figure 2.2. Altera EP-300 with upper crystal hole. [4]	3
Figure 2.3. The Open Form of Intel 4004 [7]	5
Figure 2.4. Apple Silicon M1 SoC. [8].....	6
Figure 2.5. Lifecycle of video compression [10].....	7
Figure 2.6. Results of compressing image using DPCM [12]	8
Figure 2.7. Run Length Encoding Representation [13].....	9
Figure 2.8. Different rates of I, P and B frames that skipping on not changing states of frame [18].....	11
Figure 2.9. The coding loop of Encoder[21]	11
Figure 2.10. 9 prediction methods of H.264[22]	12
Figure 2.11. Inter prediction demonstration is given for the B frame.[23]	13
Figure 2.12. The arithmetic coding demonstration[24]	14
Figure 2.13. The decoding flow demonstration.....	15
Figure 2.14. Comparison of video codecs with different videos [25]	15
Figure 2.15. Decoding speed graph with respect to codecs.[28]	17
Figure 2.16. Bit rate comparisons between most recent codecs. [31]	17
Figure 2.17. Visual Difference Caused by PSNR Change [33].....	19
Figure 2.18. An example terminal window that runs Busybox.[39]	21
Figure 2.19. U-Boot running on XPedite5501 [47].....	22
Figure 2.20. Example Device Tree file for syntax [51].....	23
Figure 2.21. Example Unix Shell in Command Line Interface [55]	25
Figure 2.22. Symmetrical encryption figure that demonstrate process. [65].....	26
Figure 2.23. The demonstration of Asymmetric Encryption [66]	28
Figure 2.24. Example usage of Diffie-Hellman method [68]	29
Figure 3.1. Digital Design for implementing Video Codec Unit	32
Figure 3.2. Generated pipeline which is used for encoding in project.	33
Figure 3.3. Representation of Encoder Pipeline	35
Figure 3.4. Generated pipeline which is used for decoding in project.	36
Figure 3.5. Representation of Decoding Pipeline.....	37

Figure 4.1. The raw original media on the left and h264 compressed by Petalinux at the right-hand side.[72]	38
Figure 4.2. Graph of PSNR values for comparing Computer (FFmpeg) and EmbeddedLinux (Petalinux)	40
Figure 4.3. Graph of PSNR value for every frame and parameter on the video Crew that encoded and decoded through Embedded Linux System	41
Figure 4.4. Graph of VMAF values for 5 different videos for comparing computer (FFmpeg) and embedded Linux (Petalinux).....	42



ABBREVIATIONS

AES	Advanced Encryption Standard
ARM	Acorn RISC Machine
ASIC	Application Specific Integrated Circuits
BASH	Bourne Again Shell
BSP	Board Support Package
CCTV	Closed-Circuit Television
CLB	Configurable Logic Blocks
CLI	Command Line Interface
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DPCM	Differential Pulse-Code Modulation
DST	Discrete Sine Transform
DTB	Device Tree Blob
DTSI	Device Tree Source Include
DTS	Device Tree Source
DVR	Digital Video Recorder
FPGA	Field Programmable Gate Array
FSBL	First Stage Boot Loader
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
HID	Human Interface Device
HMAC	Hash-based Message Authentication Code
IC	Integrated Circuit
IO	Input Output
ISA	Instruction Set Architectures
JPEG	Joint Photographic Experts Group
LAB	Logic Array Blocks
LRP	Linux Router Project
MB	Mega Byte
MMU	Memory Management Unit
MPEG	Moving Picture Experts Group
PLD	Programmable Logic Devices

PROM	Programmable Read-Only Memory
RISC	Reduced Instruction Set Computer
RLE	Run Length Encoding
SoC	System on Chip
SSBL	Second Stage Boot Loader
SSH	Secure Shell
USB	Universal Serial Bus
VCU	Video Codec Unit
VMAF	Video Multi-Method Assessment Fusion



1. INTRODUCTION

Nowadays, humanity is used to the digital world. With the new technology era, the increasing demand for media usage created a problem for the social industry that transmits and stores media files. Then, the need to reduce the size of media forced the invention of a concept called video compression. With video compression, people can easily connect to each other in real time, store enormous amounts of data that can be streamed whenever people would like to see them, and store closed-circuit television (CCTV) records in a much more efficient manner. And in today's world, social media is growing exponentially every day. Even with algorithms that compress media, it still cannot overcome needs. Thus, experiments and developments continue about newer algorithms and methods. These algorithms and methods occasionally use software for compression. Using software can reduce disk space and bandwidth usage a lot, but at the cost of power and the hardware resource itself. The use of a digitally designed non-CPU video codec unit (VCU) can reduce the cost of CPU usage caused by software encoding and decrease power usage. With this methodology, people placed dedicated VCU units in System on Chip (SoC) modules. This system on-chip module can handle embedded Linux for driver developments, and these systems can easily be placed in industrial applications. This thesis employs the KR260 SoC for an embedded device running Petalinux, utilizing the Video Codec Unit (VCU) for video compression. A comparative analysis is conducted between hardware compression on embedded Linux with VCU and software compression on a Personal Computer (PC). Five different videos are compressed on both platforms, and the resulting video quality is assessed using various metrics. This comparison aims to highlight differences in performance, emphasizing the hardware acceleration aspect introduced by the VCU in the embedded Linux setup. The evaluation provides insights into the efficiency and effectiveness of both hardware and software-based video compression approaches.

2. BACKGROUND

2.1. Application-specific integrated circuit (ASIC)

An application-specific circuit (ASIC) is a circuit that was developed for a specific purpose and can only be changed by the capabilities given by the producer. Such as an update function for parameters, changing frequency, etc. The history of ASIC started in 1967 by Ferranti and Interdesign [1]. At that time, ASIC was at the beginning, and applications were made with low amounts of components and used in the space and flight industries because of their expensiveness. At the end of the 1970s, with improved technology in transistor production, the industry could produce more transistors at a lower cost. This improvement has increased the popularity of ASIC by making it capable of industrial automation. At the time of the 1980s, the ASIC growth reached its peak because personal computers had arrived. One of the most popular ASIC personal computers from the 1980s is given in “Fig. 2.1.” The components for building a personal computer, such as graphics cards and sound cards, are based on ASIC. In the 1980s, it was also the arrival of Programmable Logic Devices (PLD), which are the ancestors of the Video Codec Unit that is used on these projects. The ASIC industry's growth increased every single day, from mobile devices in the 1990s to artificial intelligence in today's world.



Figure 2.1. Apple 2e Personal Computer from 1980s [2]

2.2. Field Programmable Gate Array (FPGA)

With increasing demand for speed in industry, people developed an integrated circuit for programming the gates for accessing enormous speeds and had the ability to reprogram. This circuit is called the Field Programmable Gate Array[3] (FPGA). Field Programmable Gate Arrays are one of the fastest reprogrammable solutions for electronic applications because they are not executing code but instead rearranging gates to perform as a circuit that works a lot faster. FPGAs are also good sandboxes for application-specific integrated circuits (ASIC) that are produced for only specific applications and are not reprogrammable gate arrays. These circuits are generally produced in bulk, and if an error comes out, it cannot be fixed unless it reproduces all the units. Whereas, with FPGAs, people can easily try their codes and applications and make their verifications before they go into production. The beginning of this technology started with the evaluation of programmable read-only memory (PROM) and programmable logic devices (PLD). Both of these can be programmed on-site or in the factory. In 1983, Altera was founded and produced the first logic device that can be programmed multiple times, the EP300. The EP300 was a chip that had a crystal hole on it for erasing when reflected ultraviolet light was inside, as given in “Fig. 2.2.”



Figure 2.2. Altera EP-300 with upper crystal hole. [4]

EP300 was a great success for the industry, but the first commercial product about FPGAs created by Xilinx. In 1985, XC2064 started to occur, and this technology merged gates with each other to accomplish tasks faster too. XC2064 had 64 configurable logic blocks (CLB) and 800 equivalent gates, built in 2 μm , and sold for \$55. With the funding of the Naval Surface Warfare Center, Steve Casselman made a computer that had 600,000 gates in 1992. The competition between Altera and Xilinx made growth faster, and FPGAs that were previously used mostly in networking and telecommunications have become industry standards in many branches of technology. Also, performance and functionality increased by reaching gate counts of millions at the start of the 2000s and over 50 million in 2013. The market share has also increased from \$14 million in 1987 to over \$9.8 billion in 2020. FPGAs separated into logic array blocks (LABs), also known as configurable logic blocks (CLBs), can be found in nearly all FPGAs for being able to program the gates for needs and hard blocks that are premanufactured by the vendor to accomplish tasks faster for common usages like PCI Express or memory controllers. With FPGAs, people can make tasks faster and independent of the sequence of processors.

2.3. Microprocessors

Microprocessors[5] are considerably one of the most important inventions in the world. In today's world, almost all electronic devices use microprocessors, microcontrollers, or chips. A microprocessor is an integrated circuit (IC) that is capable of making logic operations and controlling the needed components. Integrated circuits can have variety of different functions, such as the ability to execute programs or the capability of making arithmetic operations. Microprocessors need clock signals for working, registers for being able to handle data for execution, and memory for getting instructions in binary. With these resources, a microprocessor can get instructions and give output (again in binary) for the desired application. Instead of microcontrollers that were produced in 1969 by Four-Phase Systems (AL1) and in 1970 with Garrett AiResearch (CADIC), the first commercially published microprocessor was initially produced by Federico Faggin in 1971 with the name of Intel 4004[6], which has representation in "Fig. 2.3."

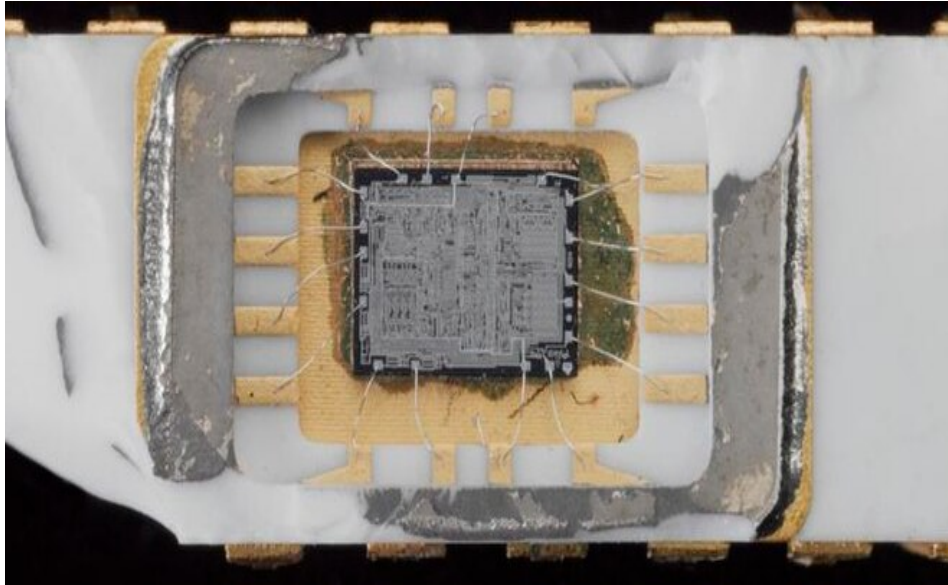


Figure 2.3. The Open Form of Intel 4004 [7]

The Intel 4004 had a 4-bit data width, a maximum CPU clock speed of 750 kHz, a 12-bit multiplexed address width, 10 μm production technology, and 2300 transistors. Its package also has 16 pins for peripherals and power. With this initiation, microprocessors improved every other day, which people use in their computers, cars, elevators, planes, railways, or all kinds of computer-based places in the modern world.

2.4. System on Chip (SoC)

System-on-Chip is an integrated circuit that unites all commonly used peripherals like storage components, memory interfaces, IO systems, Graphical Processing Units (GPUs), and even radio telecommunication systems in one pack. For example, the Apple Silicon M1 SoC chip is given in “Fig. 2.4.”



Figure 2.4. Apple Silicon M1 SoC. [8]

The packing of all needed components into one single chip comes with the pros of increasing performance, overall efficiency increases by reducing power consumption, costing less space for production, and the cons of the inability to change a faulted component or replace it for a performance increase. The benefits of SoC and the ability to make compact devices made the industry move from standard microprocessors to systems on chips.

2.5. ARM Architecture

The ARM architecture is an improved version of the open-source RISC-V instruction set architecture (ISA), and that is why it was named Acorn RISC Machine. ARM is also produced by the same name, but the company changed its name afterwards to Advanced RISC Machines. The success behind the ARM architecture is the reduced power consumption, lowering costs, and dissipating less heat under the same conditions as other architectures. These benefits include the advantage of portability and ease of producing battery-powered systems like phones, televisions, laptops, cameras, cars, and non-battery-powered systems like refrigerators, vacuum cleaners, or even keyboards.

2.6. Video Compressing

Video compression[9] is a core technology that is at the heart of media storage and broadcasting because of its efficiency and transmission speed. The basic idea behind video compression is that it uses an algorithm to not store unnecessary parts of different frames and, while doing that, makes the eyes not aware of it as much as possible. As said using the algorithm, compression starts with “encoding” for removing unnecessary parts of frames and adding information to reproduce missing parts of frames at the source, then decoding to use that media as efficiently as possible for the destination. The brief process demonstration is given at “Fig. 2.5.”

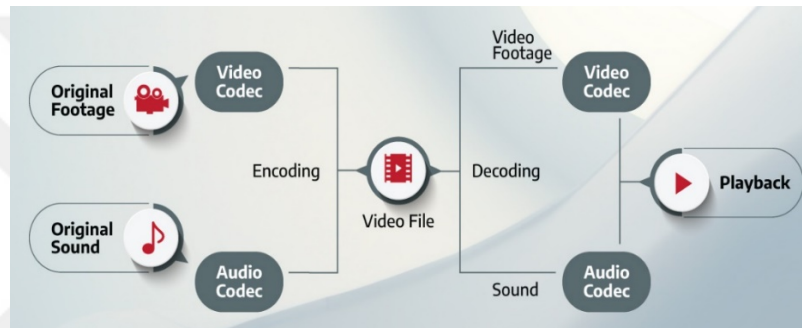


Figure 2.5. Lifecycle of video compression [10]

The evolution of video compression starts with the early ages of analog videos. In 1929, the “interframe comparison” method started to be discussed by researchers. The idea behind this approach is to compare the differences between frames. The algorithm takes a frame that will be the base of the difference, and with every new frame, it stores the difference. In this manner, people did not store all the frames with their full resolution but instead just stored a frame and a bunch of differences. After this method became familiar to people, Raymond Davis Kell said that it could be used on analog video signals. But even in today’s digital videos, this technique can be used in several different scenarios. Then people continued to research and find better and more efficient ways to compress videos [11]. In 1952, two researchers from Bell Laboratories developed a new method for efficiently compressing video. B.M. Oliver and C.W. Harrison discovered that videos could compress with prediction. After more research, they came up with the idea of using Differential Pulse-Code Modulation (DPCM)[12] on videos.

Differential pulse-code modulation was previously used on only audio, and the idea behind that was again taking an audio sample at the beginning and then storing the differences for continuous frames. But the difference between interframe compression is that it also has another feature that is not storing predictable places and lets them predict by algorithm when the file starts to play. Example DPCM compression is given in “Fig. 2.6.”

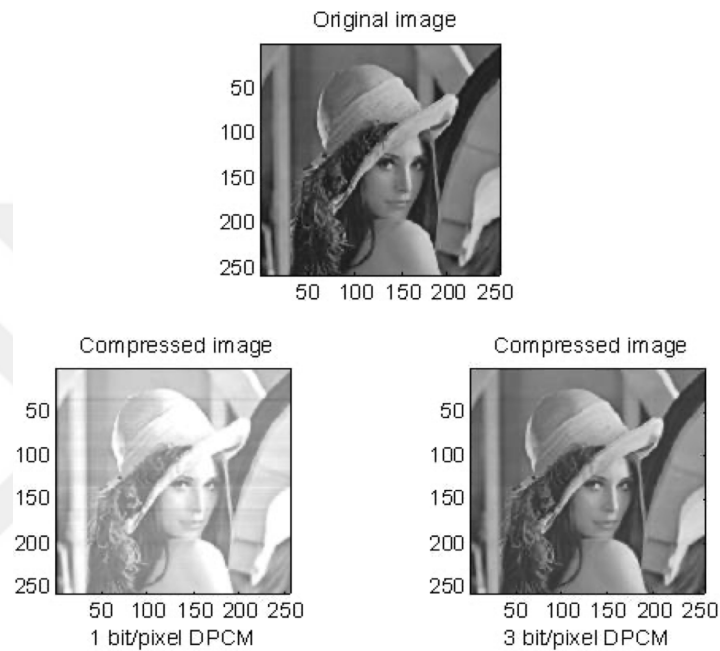


Figure 2.6. Results of compressing image using DPCM [12]

They adapted DPCM to work on videos and store them with the same idea. DPCM was a great improvement, but technology is always involved, and there is always potential for improvement. After long-term research, the Run Length Encoding (RLE) approach has been found. Run Length Encoding is a concept that stores only frames that occur for a while and how many frames reoccurred. The demonstration of run length encoding is given in “Fig. 2.7.”

Lossless Pixel Compression

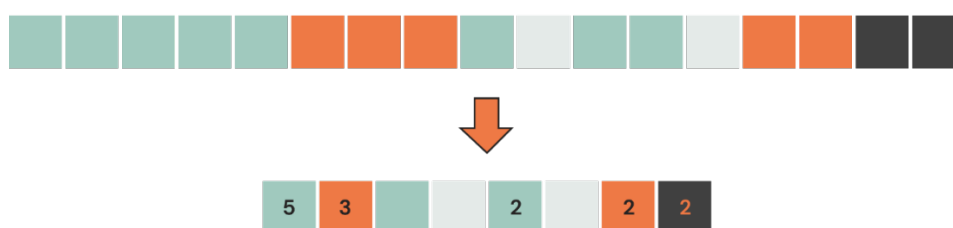


Figure 2.7. Run Length Encoding Representation [13]

But RLE is not always the best method for compression. If recurrence is low, it will be a lot less efficient than normal video because there will be a count added to all frames, and it will take more space than before. After Nasir Ahmed founded the Discrete Cosine Theorem (DCT) in 1972 at Kansas State University, a new approach has been achieved. In this method, important frequencies of an image are kept, and less important frequencies are not stored.

In 1973, at the University of Texas, Nasir Ahmed, T. Natarajan, and K.R. Rao came together and used the DCT technique for video compression. In 1974, at the University of Southern California, Ali Habibi made a mix of DCT and predictive video coding approaches, but only in intraframes (every image individually) and not in the interframe (between images) format. And after more research, Guner S. Robinson and John A. Roesse developed this approach for the interframe format in 1975. After all these experiments and theories, in 1984, a standard for digital video compression, H.120[14], occurred. This standard was good for only individual frames and was not very popular in its first days, but after optimization in 1988, it became famous and was used widely on video conferences. 1988 was also the birth of the first commercial digital video compressing algorithm, H.261[15]. H.261 was the most famous and successful compression standard in interframe and intraframe video formats in those days. The main benefit of H.261 was making a nice start for all of today's video compressing techniques (MPEG-1, H.262, H.263, H.264, and HEVC). At that time, H.261 was used; the maximum resolution was only 352x288; and it was created for only decoding. The system did not contain encoding. After the occurrence of Motion JPEG (Joint Photographic Experts Group), which saves every video frame as a JPEG image for use

with a computer's multimedia content in 1992, MPEG-1[16] occurred a year later. MPEG-1 was previously used for compression on audio files. Then again, with the research and improvements, it adapted to video compression and became a real success. It had a 352x288 maximum resolution and was designed for VHS-quality raw videos in digital format. In 1994, MPEG-2, with the other name H.262[17], arrived. H.262 came with numerous benefits, such as picture sampling, which multiplies frame rate with no bandwidth increase by getting lines numbered odd and even. Every frame showed only half of the image, but the method of changing the lines very quickly does not make too much difference for the human eye. This type of video is also known as interlaced video. Another method that eyes cannot see much difference in H.262 is chroma sampling. Chroma sampling stores less information about color because human eyes cannot feel a lot of difference between brightness and color changes. Instead, it stores smaller-sized brightness information. The H.262 also comes with another method, which is I frames, P frames, and B frame separation. I-frames stand for intra-coded frames. Intra-coded frames indicate that this frame is the result of compressing the raw frame individually. It's a lot of the same as a JPEG-coded image. It also has a good methodology; bare eyes cannot see a lot of difference. P-frames stand for predictive frames. Predictive frames use their diversity from frames before. For producing P-frames, a frame before the P-frame should be regenerated. The B-frames are also a lot the same as the P-frames, but instead B-frames get information about the next and previous (bi-directional) frames. But B-frames can never be used as reference frames on H.262 video. The brief example is given in "Figure 2.8."



Figure 2.8. Different rates of I, P and B frames that skipping on not changing states of frame [18]

H.261 and H.262 led to H.263[19] in 1996, and it became a world requirement for 3GPP for IP Multimedia Service. It also spread over video-sharing platforms like YouTube. With some other updates, it became a key feature on surveillance cameras with the name of MPEG-4 Part 2 in 1999.

In 2003, the most known and still-used video compression standard, H.264[20], arrived. H.264 has become the most used compression standard even today. The methodology lying behind H.264 is an algorithm chain of couple operations, as seen in “Fig. 2.9.”.

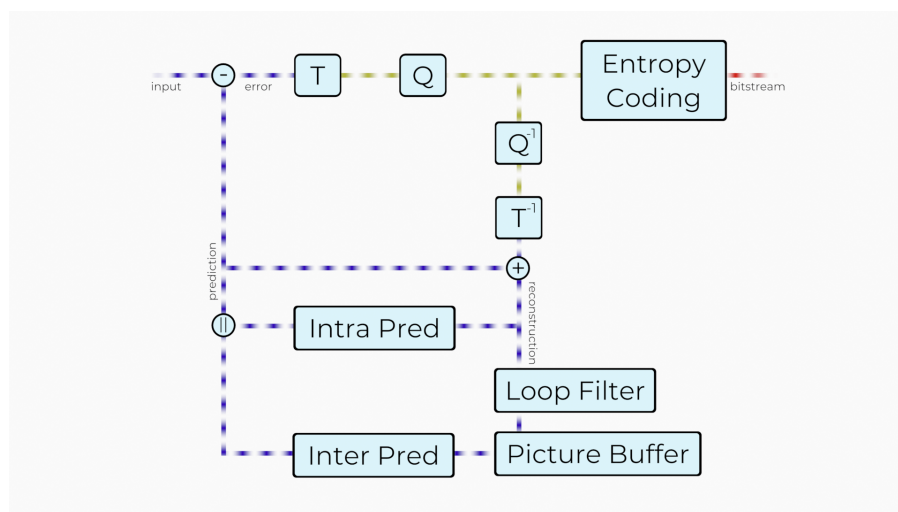


Figure 2.9. The coding loop of Encoder[21]

The first step of encoding is getting predictions from the intra- or inter-prediction phase. The intra prediction means that the prediction was made inside a specific frame. This prediction method uses nine prediction methods for H.264 and 35 prediction methods for H.265. As a sum of explanations, the DC method takes the mean of neighboring cells and places it on the inside. The other technique represented by the arrows in “Fig. 2.10.” means that the pixels or blocks are transformed in the direction of the arrows. When intra prediction is made, the encoder tries to find the best match between these methods and uses them to achieve the best encoding performance.

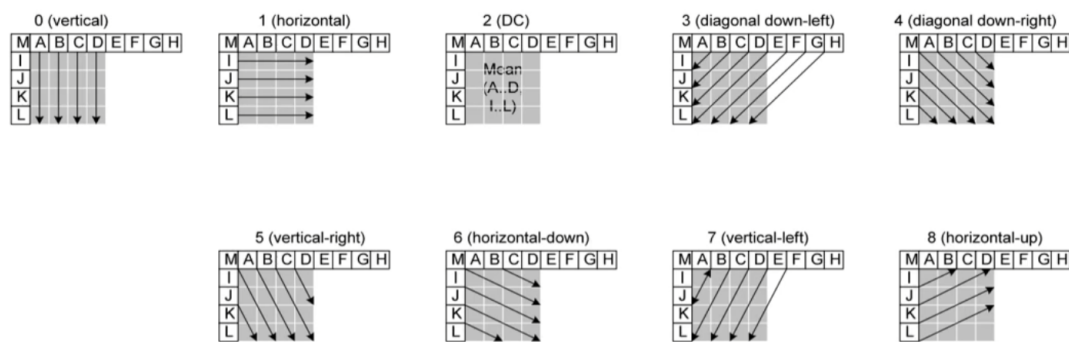


Figure 2.10. 9 prediction methods of H.264[22]

Inter prediction is a method of prediction between frames. The interframe method uses frames from before (P-frame) or after and before (Bi-directional “B” frame) of the current frame. The encoder gets the current frame and compares it with the frame before. It identifies similarities between frames to obtain a motion vector, representing a portion of the frame that has been moved to another location. In this manner, the encoder stores the vector that points to the direction in which that specific part moved and references the previous frame. The specified process demonstration is given for the B frame in “Fig. 2.11.”

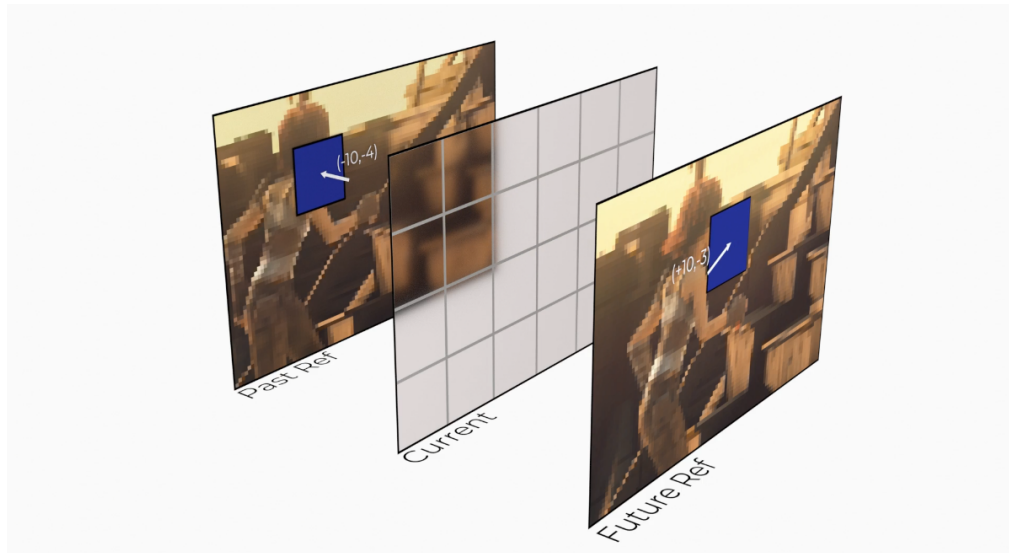


Figure 2.11. Inter prediction demonstration is given for the B frame.[23]

After obtaining inter- or intra-predicted data, the encoder continues by subtracting it from the original image.. The result from subtraction is called the “residual frame,” and it means the error rate of prediction. After having a residual frame, the transformation step is coming. The transformation step is needed for grouping similar pixels in the video. This process is done by using the discrete cosine transformation (DCT) or the discrete sine transformation (DST). The results of transformation are grouped by correlation. After the transformation phase, the quantization phase steps in. The quantization phase applies the given mathematical formula about how much loss on video the algorithm wanted. For example, dividing by 10 and rounding is one brief way of quantizing, and rounding and dividing operations cancel details from the frame when there is a need to regenerate. For this reason, if quantity increases, the loss also increases, and because of that, quality decreases. The encoder has to know the decoder because, in the future, it needs to decode as effectively as possible. Because of that, after this step, whether the encoding can output data, the generated frame needs to be reverse quantized and reverse transformed again. By making these steps, the encoder gets a lossy error of frame (residual frame) and previously made prediction (intra or inter prediction) to decode the frame again. In this manner, the encoder learns how frames will be decoded in the future. The last step from encoding before sending to bitstream is the entropy coding phase. In H.264, arithmetic coding is used for the entropy coding method. This method is used in encoding very commonly because of the lossless compression opportunity. The

arithmetic coding procedure starts with the assignment of probabilities with respect to their occurrence in the data. Then the number, which is 0 to 1, is given to the probabilities using the rule that the sum of all probabilities will be 1. After this process, the data we want to compress is starting to be divided into partitions with respect to their probabilities. Until given times of executing the same dividing into sections process, there is a floating-point number that is encoded and ready to transmit. In this approach, there is a great wisdom that gives the more frequent data more probability. Every time a dividing process occurs, the number (which is 0 to 1) will decrease by a certain amount of probability. The lower the probability, the smaller the sections will be divided. This makes the data size smaller every time, and this costs more data because of the increasing number of digits after 0. In this idea, the less occurring data has little probability, which makes the division not from a too small section. The arithmetic coding demonstration is given in "Fig. 2.12."

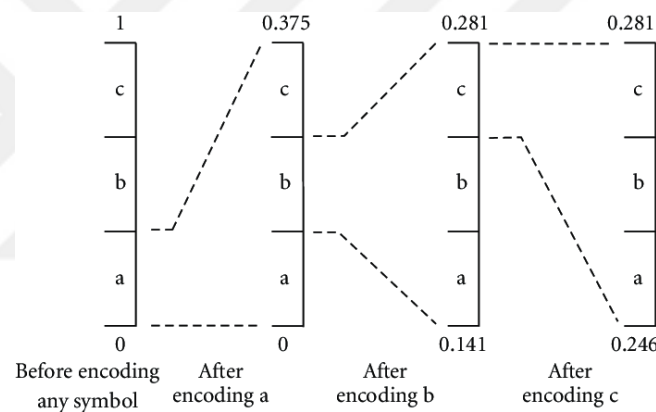


Figure 2.12. The arithmetic coding demonstration[24]

When we came to the decoding phase, the process started with entropy coding, which is arithmetic coding in this case. After reverting the lossless computation of arithmetic coding, there will be reverse quantization and reverse transformation steps for reconstructing the error metric. After reconstruction of the error metric with intra prediction and inter prediction, the frame is reconstructed, and after the loop filtering phase, which aligns the reconstructed pixels (deblocking filter), the decoded content is ready to emit from the picture buffer. The decoding flow is demonstrated in "Fig. 2.13."

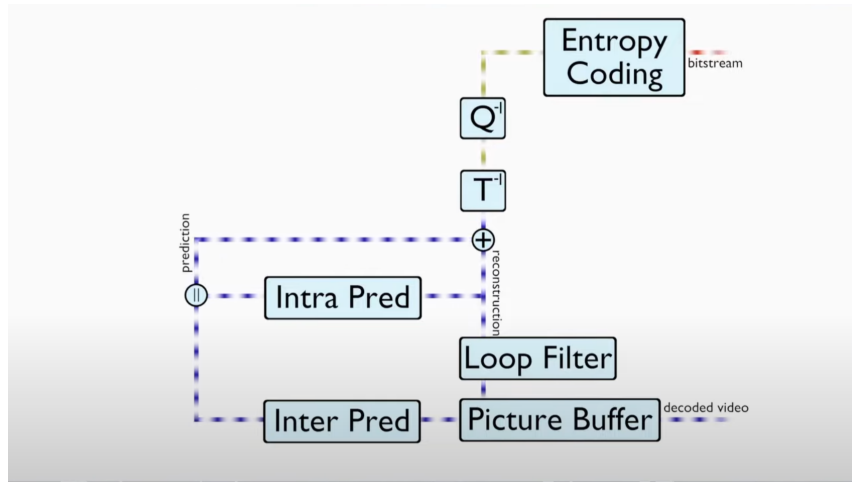


Figure 2.13. The decoding flow demonstration

The comparison between video codecs by bitrate and PSNR is given in “Fig. 2.14.” and “Table 2.1.”. The graphs show that with the improvement of intraframe prediction methods, entropy coding and quantization algorithms have made great quality improvements in video compression. The success of the H.264 video codec in terms of popularity can be explained by several benefits: it is free to use over the internet; it includes all successful compression methods; even 20 years later, it still has good resolution support (4096x2048); and it has a flexible structure.

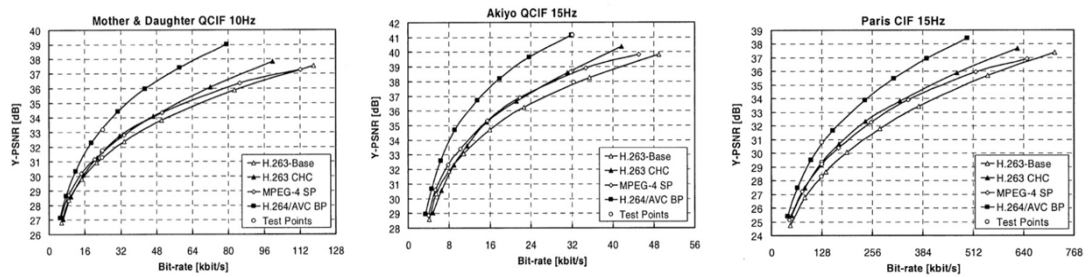


Figure 2.14. Comparison of video codecs with different videos [25]

Table 2.1. Comparison of PSNR values of codecs with different videos [25]

	H.263 Baseline				H.263 CHC				MPEG-4 SP				H.264/AVC Baseline			
Sequence	Rate	PSNR-Y	PSNR-U	PSNR-V	Rate	PSNR-Y	PSNR-U	PSNR-V	Rate	PSNR-Y	PSNR-U	PSNR-V	Rate	PSNR-Y	PSNR-U	PSNR-V
A: QCIF, 10 Hz, 24 kbit/s																
Akiyo	24.14	37.34	39.73	41.31	24.06	38.54	41.89	42.93	24.19	38.01	40.24	41.95	24.00	40.68	42.90	43.58
Foreman	24.21	27.73	35.39	34.95	24.25	28.52	37.39	37.37	24.09	29.10	36.27	35.95	23.87	30.08	37.45	37.58
Mother & Daughter	23.78	31.27	36.49	36.32	23.82	31.68	37.80	37.65	23.97	31.75	36.62	36.37	24.08	33.19	37.96	37.71
Silent	24.08	31.12	35.44	36.93	23.90	32.31	37.28	38.83	24.14	31.68	35.51	37.02	24.09	32.42	36.34	38.07

After increasing needs for more resolution and sphere-style video files (360°), compression needed to be more efficient. To respond to this, H.265[26] was created in 2013. H.265 also supports both sphere-style videos and higher resolutions up to 8K.

Despite efficient video storage, H.265 is not used as much as H.264 because of several disadvantages, like the fact that license prices are high for using this standard and because of the long encoding times, the latency increases. Also in 2013, VP9[27] was created by Google for contesting H.265 with nearly the same experience without any price. The performance comparison between codecs is given in “Fig. 2.15.” As mentioned before, the methods and algorithms on codecs differ due to technological improvements. The couple of explanations behind the H.265 algorithm based on intraframe prediction method count increase, with better algorithms and the block size increase for more resolution on videos.

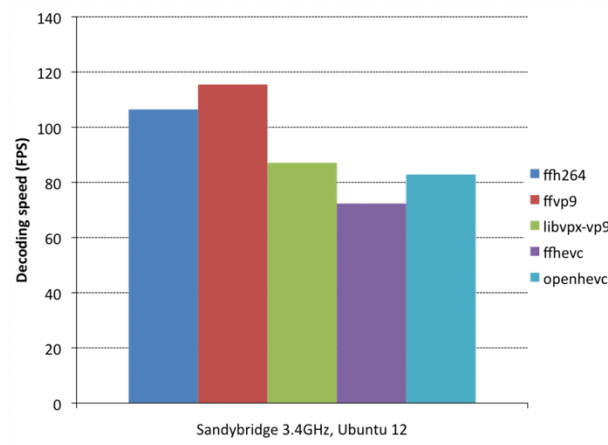


Figure 2.15. Decoding speed graph with respect to codecs.[28]

In 2018, Netflix, Google, Mozilla, Cisco, AMD, Apple, Adobe, Nvidia, VideoLAN, Hulu, Amazon, Facebook, Samsung, ARM, Vimeo, Microsoft, and Intel united with the same purpose, creating a new compression standard for real-time projects with high resolution. After this collaboration, they improved algorithms from previous projects, and AV1[29] occurred. It was also free, open-source, and had no copyrights. The most recent video compression standard, H.266[30](also known as VVC), was finalized in 2020. Although there has been a performance increase, the license fee problem has come back. One of the most important metrics on codecs, bitrate savings compared between VVC, AV1, and HEVC, is given in “Fig. 2.16.”

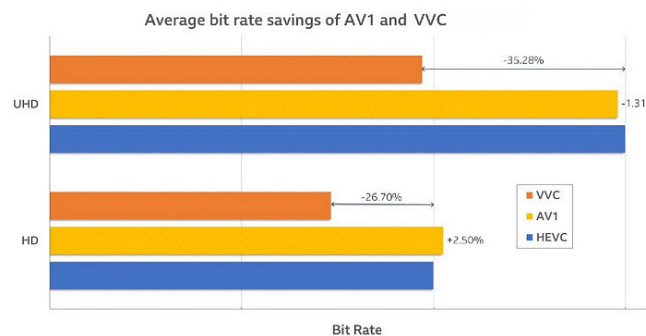


Figure 2.16. Bit rate comparisons between most recent codecs. [31]

2.7.Video Codec Unit

In the FPGA industry, vendors also create integrated programmable logic blocks that would be hard for end users to produce themselves. One of these programmable logic

blocks that comes with the device is the video codec unit. The Video Codec Unit is basically programmable logic that is specifically designed for encoding and decoding video. The main benefit of this terminology is the ability to decode and encode without requiring large amounts of processor power and time. Integrated VCUs can also decode video while encoding. With this capability, it becomes possible to watch data while encoding and recording, especially for systems like surveillance cameras.

2.8.Video Quality Metrics

Video quality metrics are metric representations of what quality means for actual people. In this work, two metrics are used for comparison.

2.8.1. Peak signal-to-noise ratio (PSNR)

Peak signal-to-noise ratio (PSNR) [32] is a metric that calculates the maximum signal power for unwanted noise. PSNR is widely used for compressed and decompressed media quality comparisons. The PSNR in the dB range calculated with logarithms because of the variety of signals can differ too much. Below represents how the PSNR value was calculated by the given mathematical formula in “Equation 2.1.”.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I_{ij} - K_{ij})^2 \quad (2.1)$$

In this equation, MSE stands for Mean Squared Error, which is calculated by I, the form of image in m x n that does not contain noise, and another form of K, which includes approximated noise. Then the PSNR value in dB format can be calculated with the given “Equation 2.2.”.

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned} \quad (2.2)$$

The MAXi value in the given equation means the maximum pixel amount of a frame that exists. (e.g., 255 for black and white pixels) The difference in image corresponding to PSNR values can be seen in "Fig. 2.17." The figure can also show us that the value above 40 is not a difference that can be seen with bare eyes.

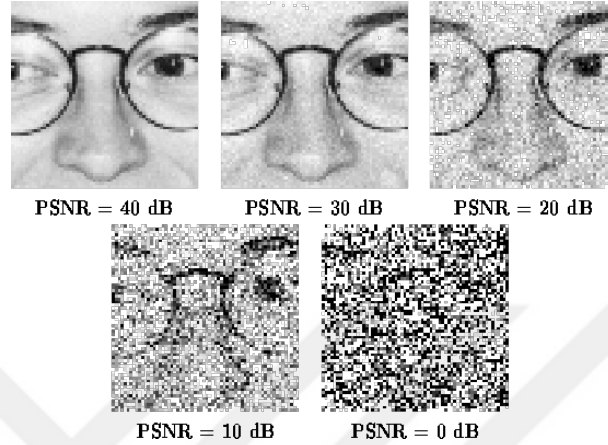


Figure 2.17. Visual Difference Caused by PSNR Change [33]

2.8.2. Video Multimethod Assessment Fusion (VMAF)

VMAF (Video Multi-Method Assessment Fusion) is a metric that is used for calculating the success of compression or quality. It calculates value by using three metrics. Firstly, VMAF calculates Visual Information Fidelity (VIF). VIF is a quality of image metric that is based on natural scene statistics and a model that is extracted from the human eye[34]. The second metric is the Detail Loss Metric (DLM), which is based on detail loss and distortion that distract the viewer. The third is the mean co-located pixel difference (MCPD), which is also a metric that calculates temporary changes based on components of luminance. This video quality metric does not have a publicly shared formula because it is based mainly on model training. The numeric quality representation of this metric was scaled between 0-100. There is also information that the values of this metric can be close to 100, even if the encoding and decoding are done. This behavior is caused by this metric, which is trained by the human visual system, and it means decoding is good enough to not make a real difference. This metric was also used effectively when compared to other VMAF results instead of individual values.

2.9. Embedded Linux

To understand embedded Linux, there is a need to know what embedded systems are. Embedded systems are systems that have less purpose and are mostly produced for specific usages. They have inexpensive production costs, low power usage, are less powerful, and do not have as many system resources as computers that are used today. In the history of this terminology, when Linus Torvalds created [35] Linux 0.01 in 1991, he never thought it would support another device that he did not have [36]. Then Linux was a great success over time. Its benefits and openness make people work for it and use it day by day. In 1995, Linux came with version 1.2. Because of the more platform support added to Linux, it rang a bell among the developers that it was not hard to port to other devices. With this awareness, Bruce Perens started to work on a project named Busybox. Busybox is also named “Swiss Army Knife for Embedded Linux” [37]. The reason for giving it a name like that is that Busybox makes a lot of Linux shell commands available with just the necessary functions [38]. With this approach, less-needed functions of commands are discarded to save storage space. Busybox was normally created for saving the image of Debian to floppy disks of 1.44 MB with the purpose of helping recovery by booting Debian distro from a floppy disk. Then they realized that this kind of shrinking can also make successful work on embedded systems. An example Busybox terminal is given in “Fig. 2.18.”

```

BusyBox v0.61.pre (2006.04.11-10:04+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# help

Built-in commands:
-----
. : bg break builtin cd chdir continue eval exec exit export
false fc fg hash help jobs kill local pwd read readonly return
set setvar shift times trap true type ulimit umask unset wait

#
#

```

Figure 2.18. An example terminal window that runs Busybox.[39]

In 1996, with Linux 2.0, more architectures were added, like the Motorola 68k and PowerPC. At the same time, the foundation started for one of the pioneering companies for embedded Linux services, Timesys [40]. Even though Timesys made successful processes because they made their service proprietary, there is no work contributed to the mainline kernel on Linux [41]. In 1997, Dave Cinege started the Linux Router Project (LRP). LRP was a project that was made to turn a normal computer into a router. It also used Busybox for using a floppy disk, which will boot from. This approach is called the first embedded Linux for some communities [42]. In 1998, Busybox was upgraded by Dave Cinege for higher systems, and Kenneth Albanowski started to develop uClinux [43] (also known as Clinux) with D. Jeff Dionne. This was a remarkable update in the embedded world because it allows people to run Linux on devices that have no memory management unit (MMU). It also gave birth to uClibc [44](also known as today's uClibc-ng), which is a tiny C library for embedded systems, and a build tool named Buildroot [45]. In 1999, with the Linux version 2.2.0, ARM architecture started to be part of Linux, and embedded Linux got more shares on the market with digital videorecording (DVR) systems and web cameras. Also in the same year, 8xxROM was released, which is the name of the 8xx PowerPC bootloader, but when it's needed to upload on sourceforge.net, under the rule that a project name cannot start with a number, it was renamed to

PPCBoot. In late 2002, PPCBoot version 2.0 changed its name, and U-Boot (also known as Das U-Boot) [46] was born. U-Boot is a great success and a very important utility for embedded Linux development; hence, it has still been used on new systems since it's been developed. It serves as both the first-stage boot loader (FSBL) and the second-stage boot loader (SSBL). While it has been developing, U-boot has supported other architectures such as PowerPC, Motorola Coldfire, and ARM. An example U-Boot terminal is given in “Fig. 2.19.”

```

U-Boot 2010.12-xes_r3 (Aug 25 2011 - 11:04:04)

CPU0: P2020E, Version: 1.0, (0x80ea0010)
Core: E500, Version: 4.0, (0x80211040)
Clock Configuration:
  CPU0:1066.667 MHz, CPU1:1066.667 MHz,
  CCB:533.333 MHz,
  DDR:400 MHz (800 MT/s data rate) (Asynchronous), LBC:133.333 MHz
L1:  D-cache 32 kB enabled
     I-cache 32 kB enabled
Board: X-ES XPedite5501 PMC/XMC SBC
      Rev SA, Serial# 36093001, Cfg 90015130-1
I2C:  ready
DRAM: 2 GiB (DDR3, 64-bit, CL=6, ECC on)
FLASH: Executed from FLASH1
POST memory PASSED
FLASH: 256 MiB
L2:   512 KB enabled
NAND: 4096 MiB
PCIE1: connected as Root Complex (no link)
PCIE1: Bus 00 - 00
PCIE2: disabled
PCIE3: disabled
In:    serial
Out:   serial
Err:   serial
DTT:   37 C local / 59 C remote (adt7461@4c)
DTT:   37 C local (lm75@48)
Net:   eTSEC1 connected to Broadcom BCM54825
       eTSEC2 connected to Broadcom BCM54825
       eTSEC1, eTSEC2
POST i2c PASSED
Hit any key to stop autoboot: 0
=>

```

Figure 2.19. U-Boot running on XPedite5501 [47]

With these improvements in technology, the WRT54G [48] wireless router was produced by Linksys, and it started an era of producing routers based on the Linux environment. After routers, there are a lot of different devices that have been started to produce.

But then another milestone was achieved by Intel with the introduction of the Yocto Project by changing from Poky Linux and giving its rights to the Linux Foundation in 2010. The Yocto Project also created the basis for the Petalinux Project, which is also the build system used on this project.

2.10. Petalinux Project

Petalinux Project [49] is a set of tools that combine to build a custom Linux distribution for Xilinx-based FPGA-Soc Systems. Petalinux includes Board Support Packages (BSP) tailored for Xilinx-produced Eval boards and emphasizes the ease of managing complexities through abstraction. When generating an output, Petalinux requires device configuration information, such as details about hardware and communication buses. This information is provided by a device tree [50]. It can come from either the board support package (BSP), typically supplied by the vendor and containing hardware and driver details, or a specific device tree source (DTS) file.

Subsequently, the Device Tree Compiler comes into play, compiling the device tree source (DTS) file into the device tree blob (DTB) file. This makes the information understandable for the device to be used. An illustrative example of a device tree block is presented in “Fig. 2.20.”

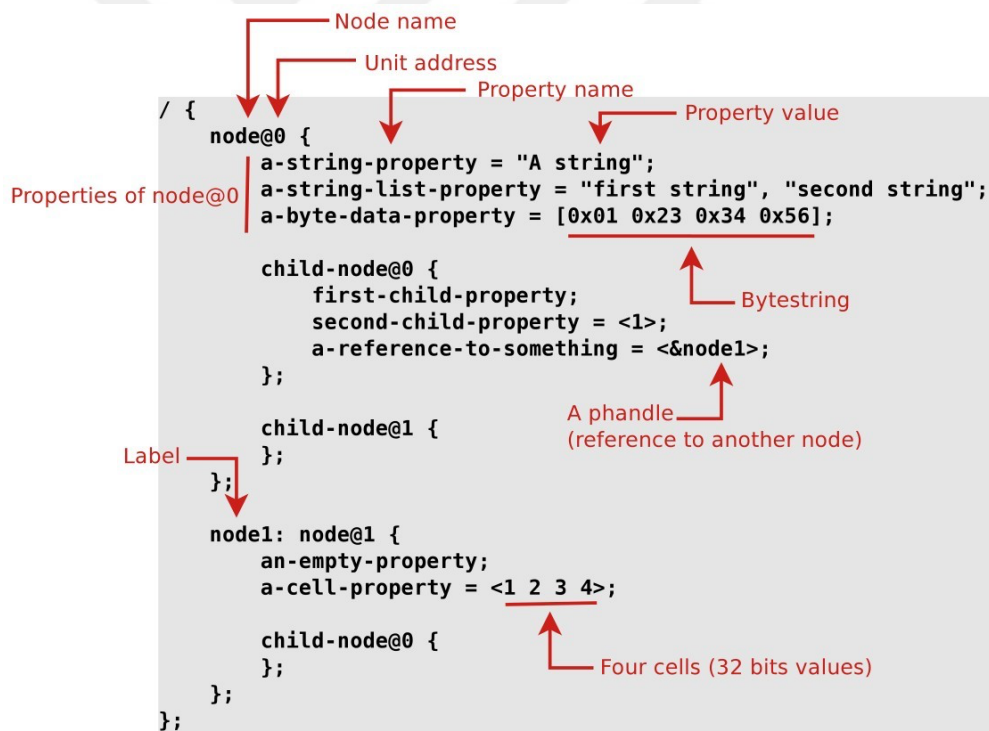


Figure 2.20. Example Device Tree file for syntax [51]

After collecting the needed files, we start our Petalinux project by creating it. Then we can add the needed functions or remove not-in-use features from KConfig, which is an easy way to configure components with a selection mechanism. Currently, there are six

KConfig-compatible configuration sections: kernel, u-boot, rootfs, device-tree, pmufw, and bootloader. With these configurations, Petalinux will build all the files and settings for producing output files. The files are created by Bitbake, which is a well-known component of Yocto Project's tool for creating boot images, and usually have three important files: BOOT.bin, which is used for initializing FPGA configuration (bitstream), the first stage bootloader (FSBL), and image.ub, which include the device tree and kernel. By booting from the output files of the build, the environment is ready to use [52].

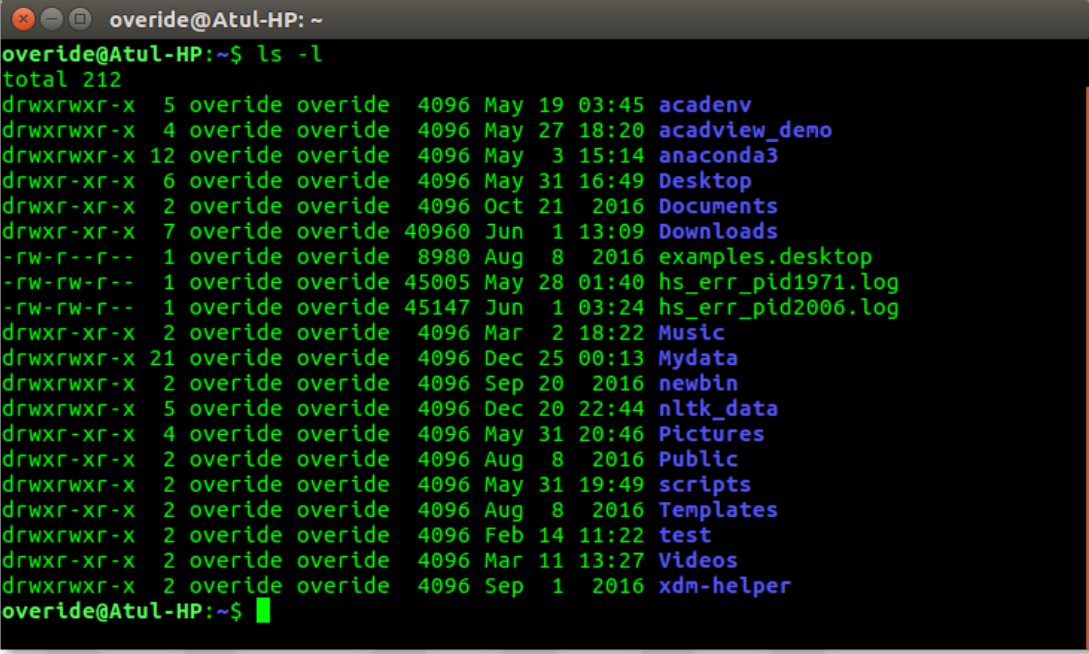
2.11. Brief Linux Terminologies

2.11.1. Kernel

Kernel [53] is a program that lies under every process of the computer. Whenever the user interacts with a computer, or even not in the interaction with it, the kernel works under all software. Despite being just another piece of software, it handles all the interactions between hardware and the needed components and resources for other running software. It also facilitates the administration of resources such as hard drives, Ethernet peripherals, cameras, and more, making the process easier. For this administration, a method called system calls was developed, and it generally uses interrupts for cutting processes on the CPU, getting needed component data, and continuing from where it left off. For security reasons, the kernel runs on the protected memory area, which is named kernel space, and it also separates memory from other software data sections, which are also called user space. With this separation, unauthorized access to the kernel is prevented, and any misbehavior in user space is not a cause of fatal failure. System calls are also used for safely getting the required information from kernel space.

2.11.2. Unix Shell

The Unix Shell [54] is the Command Line Interface (CLI) that can facilitate interaction between the operating system and the user.



```
override@Atul-HP: ~  
override@Atul-HP:~$ ls -l  
total 212  
drwxrwxr-x  5 override override  4096 May 19 03:45 acadenv  
drwxrwxr-x  4 override override  4096 May 27 18:20 acadview_demo  
drwxrwxr-x 12 override override  4096 May  3 15:14 anaconda3  
drwxr-xr-x  6 override override  4096 May 31 16:49 Desktop  
drwxr-xr-x  2 override override  4096 Oct 21  2016 Documents  
drwxr-xr-x  7 override override 40960 Jun  1 13:09 Downloads  
-rw-r--r--  1 override override  8980 Aug  8  2016 examples.desktop  
-rw-rw-r--  1 override override 45005 May 28 01:40 hs_err_pid1971.log  
-rw-rw-r--  1 override override 45147 Jun  1 03:24 hs_err_pid2006.log  
drwxr-xr-x  2 override override  4096 Mar  2 18:22 Music  
drwxrwxr-x 21 override override  4096 Dec 25 00:13 Mydata  
drwxrwxr-x  2 override override  4096 Sep 20  2016 newbin  
drwxrwxr-x  5 override override  4096 Dec 20 22:44 nltk_data  
drwxr-xr-x  4 override override  4096 May 31 20:46 Pictures  
drwxr-xr-x  2 override override  4096 Aug  8  2016 Public  
drwxrwxr-x  2 override override  4096 May 31 19:49 scripts  
drwxr-xr-x  2 override override  4096 Aug  8  2016 Templates  
drwxrwxr-x  2 override override  4096 Feb 14 11:22 test  
drwxr-xr-x  2 override override  4096 Mar 11 13:27 Videos  
drwxrwxr-x  2 override override  4096 Sep  1  2016 xdm-helper  
override@Atul-HP:~$
```

Figure 2.21. Example Unix Shell in Command Line Interface [55]

With Unix Shell, users can execute tasks or programs from a terminal and get outputs from executed content. An example of Unix Shell in Command Line is shown in “Fig. 2.21.”

The need to interact with the kernel and perform tasks arose at the inception of Unix development. In 1971, Ken Thompson, renowned as the developer of UNIX, introduced the Thompson Shell[56]. This marked the genesis of UNIX shells, featuring a basic command-line interface without support for scripting. The interface introduced essential elements such as pipes (“|”), straightforward input-output management using (“<” and “>”), a built-in command separator (“:”), a “goto” statement for command navigation, and easy utilization of globbing features. Then, in 1973, PWB Shell[57] (also known as Mashey Shell) was developed by John Mashey. This shell had several extensive features that passed to other generations, which are “if”, “then”, “else”, “endif”, “while”, “switch,” supporting operation execution, variable support (although it supports single letters and some reserved words, which are ancestors of environmental variables, for avoiding hard-coding like \$HOME for the home path of every desired system dynamically), and scripting. In 1979, Stephen Richard “Steve” Bourne created Bourne Shell[58], which is also today's bash.

After “The Unix Programming Environment” was published in 1984, people realized shell could be used like a programming language, and Bourne shell became famous. There were various shells after Bourne Shell, but as a part of the GNU project, BASH [59] (Bourne Again Shell) was created in 1989.

Even today, most Linux operating systems use BASH for command execution and bash scripting. With increasing demand for embedded Linux systems, there was a need for shells for use in embedded systems. For this purpose, as mentioned before, Bruce Perens Developed BusyBox. Also, the need to access shells through networks created Berkeley r-commands [60], Telnet [61], and for file-transferring purposes, FTP [62](File Transfer Protocol). These protocols for accessing the network took the risk of cyber security with them. For this purpose, the SSH [63] (Secure Shell) protocol was developed by Tatu Ylönen in 1995. The SSH protocol also uses client-server methodology for connections. But unlike the other protocols, it sends data in encrypted format instead of plain text format. SSH uses techniques for safely transmitting data from one point to another. The first technique is symmetrical encryption. Symmetrical encryption [64] is a way of encryption that uses only one key for decrypting and encrypting data. This means a person who owns a key can easily decrypt and encrypt data without the permission or knowledge of the owner. An example of a symmetrical encryption demonstration is given in “Fig. 2.22.”

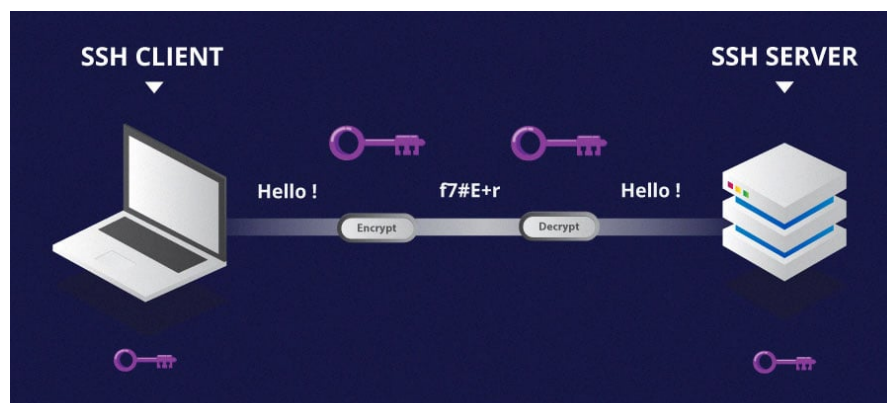


Figure 2.22. Symmetrical encryption figure that demonstrate process. [65]

Symmetrical encryption is normally used on the connection security side of SSH. The server and client negotiate on a key, and after that, both use the same key to encrypt transmitted and received data. SSH is also configurable with many ciphers, like AES

(Advanced Encryption Standard), Arcfour, 3DES, Blowfish, etc. Whenever an SSH connection starts, the server and client also need to negotiate the cipher standard. For the reference, on MacOS Ventura 13.5.1 the following command can be used for listing default ciphers:

```
%~ openssl ciphers -v | awk '{print $5}' | sort | uniq
```

```
Enc=3DES(168)
Enc=AES(128)
Enc=AES(256)
Enc=AESGCM(128)
Enc=AESGCM(256)
Enc=Camellia(128)
Enc=Camellia(256)
Enc=ChaCha20-Poly1305
Enc=GOST-28178-89-CNT
Enc=RC4(128)
```

The output provided means the supported encryption method and the bit size of the key to encryption or description.

The second technique is asymmetrical encryption. Asymmetrical encryption is an encryption technique that requires two keys for each transaction of data. These keys are also known as public and private keys. A public key is generally an encryption key that can easily be sent to anywhere, nearly without security risk. But the private key is usually used for decryption, and it should be kept “private”. The methodology behind this approach is that, after the mathematical equations, the result cannot be recovered by public key. The encrypted data is only decrypted with a private key.

Asymmetrical encryption is generally used for user authentication, but there is one more well-known usage: signing documents. Previously, it was mentioned that the private key is usually used as a decryption key, but while signing documents, people wanted to know who created the file, and it cannot be easily replaced instead of encryption. For this purpose, people used private keys to encrypt a file and shared a public key. In this way, whoever gets the file can access the public key, and if the public key decrypts the file, that indicates it is unique and safe to use. The demonstration of asymmetric encryption is given in "Fig. 2.23."

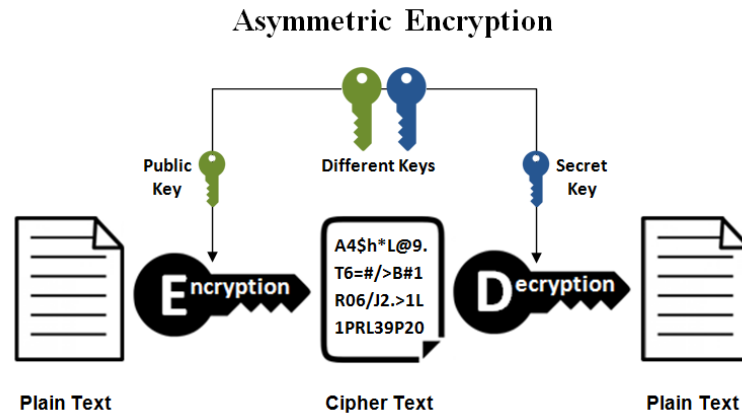


Figure 2.23. The demonstration of Asymmetric Encryption [66]

Another technique that SSH uses is hashing [67]. Hashing is mainly produced for the need to transfer unique, uncorrupted, and unchanged data through the network. The SSH protocol uses HMAC (hash-based message authentication code) methodology for this purpose, and every message is controlled for integrity and security after it is received. Hashing involves a series of tasks where each bit of data is processed through complex equations, resulting in a unique hash by comparing and equating each bit to one another. After these equations, the data cannot be reversible, and if any bit inside of it changes, the hash result will be different. Hashing is also used regularly when storing passwords for privacy. With this methodology, passwords cannot be seen by anyone, including service providers. After gaining knowledge of these concepts, the combination of them, and more, SSH becomes usable. For establishing a connection, the server and client declare a great prime number. This is publicly transferrable and will be used as "seed value." Then, as mentioned before, the server and client need to agree on an "encryption generator" that is supported by both devices. And the last needed component is another great prime number that is not known by each of the devices. With these components, the Diffie-Hellman method is used for generating a not-transferred private sacred key for symmetrical encryption. This method works by sharing prime numbers that are initially declared, then generating encrypted values by using ciphers with these prime numbers and secret prime numbers, transferring them to each other, and then calculating the same asymmetric sacred key value by sharing prime numbers, shared encrypted values, and private prime numbers. With this method, keys can be calculated with no security risk of

being transferred over the network. The example usage of the Diffie-Hellman method is given in “Fig. 2.24.”

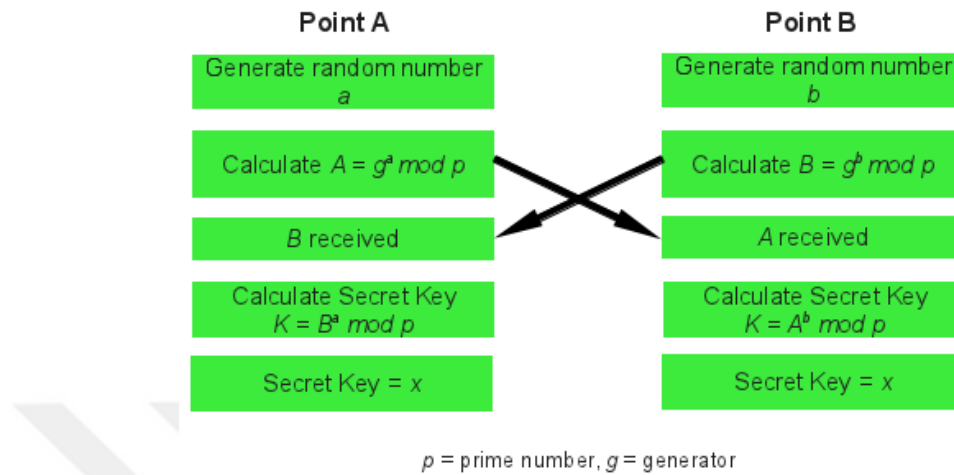


Figure 2.24. Example usage of Diffie-Hellman method [68]

After establishing a secure connection, SSH initiates the authentication feature. One of the best practices for secure authentication is using the asymmetrical encryption method. Using public and private keys for authentication is a good way to establish a connection because even if an attacker knows your username and password, it will not be enough to establish a connection. The SSH protocol is the barebone for even computer-to-computer communication, but it is also a heavy process for embedded devices, even if it is needed for embedded systems too. For this purpose, Dropbear [69], created by Matt Johnston in 2004, is required. Dropbear is a replacement for OpenSSH due to insufficient resources. With this replacement, it has been possible to use the SSH protocol over embedded systems.

3. METHOD

3.1. Hardware

The initialization of the project started with searching for hardware that could implement VCU. The supply shortage of chips and cost effectiveness led this idea to keep going with the Xilinx Kria Robotics 260 System on Module (KR260 SOM) [70]. This module is a good starter pack for making robotic applications on Xilinx's Zynq systems. It comes with gigabit Ethernet ports for connecting and transferring data, Universal Serial Bus (USB) ports for connecting storage devices or using Human Interface Devices (HID), a DisplayPort 1.2a port for outputting video, SFP+ optics for 10 Gbps data transfer, and universal connectors for aftermarket kits. The system on the module has a Primary boot memory of 512 MB for the boot file system, 4 GB of RAM for fluently performing operations over Linux, and an SDHC card slot for inserting more storage as needed. The dedicated Video Codec Unit (VCU) can also do decoding and encoding on media from hardware without bothering the CPU while compressing, giving it the ability to perform needed operations on this project.

3.2. Digital Design

For getting hardware ready after producing carrier cards and placing electronic components, the system needs a digital design that will configure the Video Codec Unit (VCU), timing components, buffering mechanism for media stream, and support of needed peripherals (USB, Ethernet, storage, etc.) for input and output operations. For this purpose, AMD Xilinx's Vivado 2022.1 is used with the built-in board support package of the Kria Robotics 260 development board. After getting the board support package, the block named Zynq Ultrascale+ MpSoC comes with default settings. In this process, a Video Codec Unit (VCU) block named Zynq Ultrascale+ VCU (Pre-production) was inserted, and its clock, encoder, and decoder settings were configured. Subsequently, at the output of the VCU, AXI Interconnect blocks were connected for data buffering. Processor system reset blocks were added, and the clock was adjusted using the clocking wizard. The resulting block diagram is now ready and can be observed in "Fig. 3.1.". The next step is generating output. For this purpose, the run-synthesis process should be initiated. The synthesis step gets the high-level description of the

design and turns it into a low-level netlist, which turns code into connections between components. The next step is run implementation.

The run implementation command is a set of processes that are specific to a board or device. The command basically makes placing and routing, which places the logic gates with respect to the netlist; clock planning, which is made for clock synchronization; static timing analysis, which is making sure that the design will work at a certain frequency; and power analysis, which manages power usage of the FPGA while it is working. After these processes, the program creates a bitstream for storing inside the FPGA's programmable memory and is ready to be executed.



3.3.Gstreamer Pipeline

One of the challenging operations is creating a gstreamer pipeline [71] for encoding and decoding. The created pipeline for encoding video is given in “Fig. 3.2.”

```
GST_DEBUG="GST_TRACER[ss1]:7"
GST_TRACERS="proctime"
GST_DEBUG_FILE=./debug_files/proctime.txt
GST_DEBUG_DUMP_DOT_DIR="./dot_files/"
gst-launch-1.0 filesrc location="input.yuv"
! rawvideoparse format=y42b
width=352height=288 !
videoconvert n-threads=4
! omxh264enc control-rate=2
target-bitrate=8000 ! queue !
filesink location=output_encoded.mp4
```

Figure 3.2. Generated pipeline which is used for encoding in project.

In the context of debugging within GStreamer, the variable "GST_DEBUG" signifies a debugging level of 7, specifically tracer-level debugging. The "GST_TRACERS" variable specifies which parameters are to be logged. Additionally, "GST_DEBUG_FILE" and "GST_DEBUG_DUMP_DOT_DIR" indicate the files where the debug information and GStreamer pipeline dot file, which illustrates the used pipeline schema, are stored. The command "gst-launch-1.0" is utilized to construct the desired GStreamer pipeline. The "filesrc location" designates the input file to be encoded, while "filesink location" represents the location where the resulting file will be saved.

The keyword "rawvideoparse" is employed for timestamping data streams and delivering buffers individually. The "format" parameter specifies that the video format is YUV 4:2:2 planar, while "width" and "height" denote the media frame sizes. The "videoconvert" command in the pipeline converts video frames to the specified video format, and "n-thread" indicates the maximum number of threads used during operation.

The presence of "omxh264enc" in the pipeline signifies the utilization of the OpenMax H.264 Hardware Accelerated Encoder. The "control-rate" parameter is set to constant bit rate (CBR), and "target-bitrate" specifies the desired bit-rate value. The "queue" command is responsible for buffering data until certain conditions, such as the completion of one buffer instance or meeting byte-size and time limits, are met.

The resulting pipeline is illustrated in "Figure 3.3." In this diagram, the process initiates with the GstFileSrc statement, where the system retrieves the file from the source. Subsequently, GstRawVideoParse processes the raw data stream, passing it incrementally to the next stage. GStreamer then initiates the Quality-of-Service procedure, configuring it to use four threads with GstVideoConvert for conversion. After encoding with GstOMXH264Enc-omxh264enc, the buffered data is managed by GstQueue before being written to the file. Finally, the frames are ready for output, orchestrated by GstFileSink.

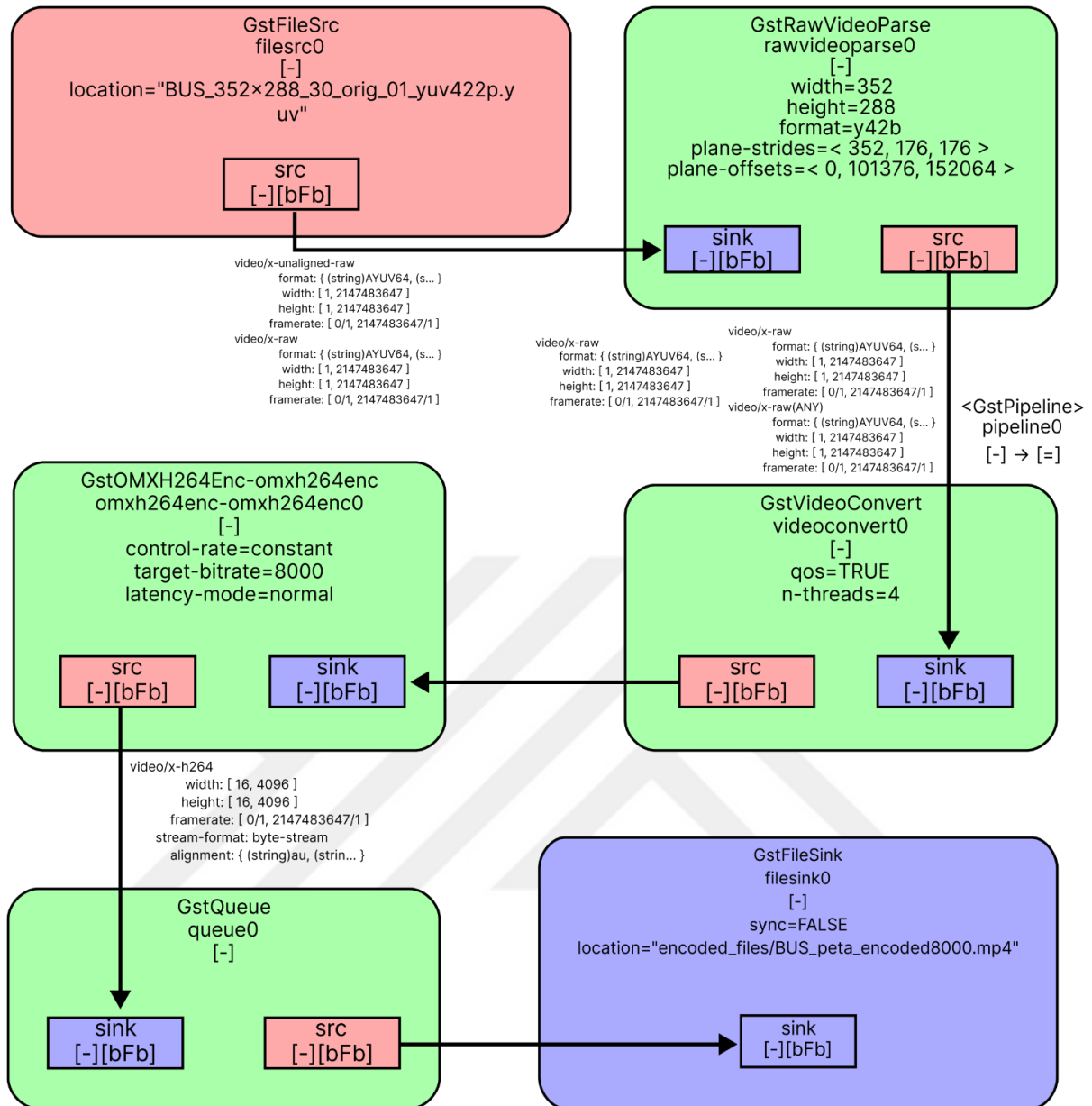


Figure 3.3. Representation of Encoder Pipeline

And for decoding video pipeline is given at “Fig. 3.4.”

```
GST_DEBUG="GST_TRACER[ss1]:7"
GST_TRACERS="proctime"
GST_DEBUG_FILE=./debug_files/proctime.txt
GST_DEBUG_DUMP_DOT_DIR="./dot_files/"
gst-launch-1.0 filesrc location="input.mp4"
! h264parse ! video/x-h264 ! omxh264dec
! videoconvert ! video/x-raw, width=352,
height=288, framerate=25/1, format=Y42B
! filesink location="output.yuv"
```

Figure 3.4. Generated pipeline which is used for decoding in project.

Here, alongside the preceding elements, "h264parse" assumes the role of parsing h264-encoded media. "video/x-h264" denotes the video format used for encoding, while "video/x-raw" specifies the raw video parameters for the output file. The "framerate" parameter establishes the number of frames per second. Following the completion of the aforementioned processes, the system is prepared for both encoding and decoding. The final decoding pipeline representation is depicted in "Figure 3.5." In this illustration, the process commences with the “GstFileSrc” statement, where the system retrieves the file from the source and incorporates it into the system. The subsequent step, “GstH264Parse”, processes the encoded data stream, inserting it incrementally into the subsequent stage. After parsing, “GstCapsFilter” intervenes to extract only H.264-encoded content. The ensuing step, “GstOMXH264Dec-omxh264dec”, is responsible for the hardware decoding function. Following decoding, GStreamer handles the output, initiating the Quality-of-Service procedure with “GstVideoConvert”. Subsequently, “GstCapsFilter” steps in to format the media according to the specified color and size. Following these stages, the frames are prepared for output, facilitated by “GstFileSink”.

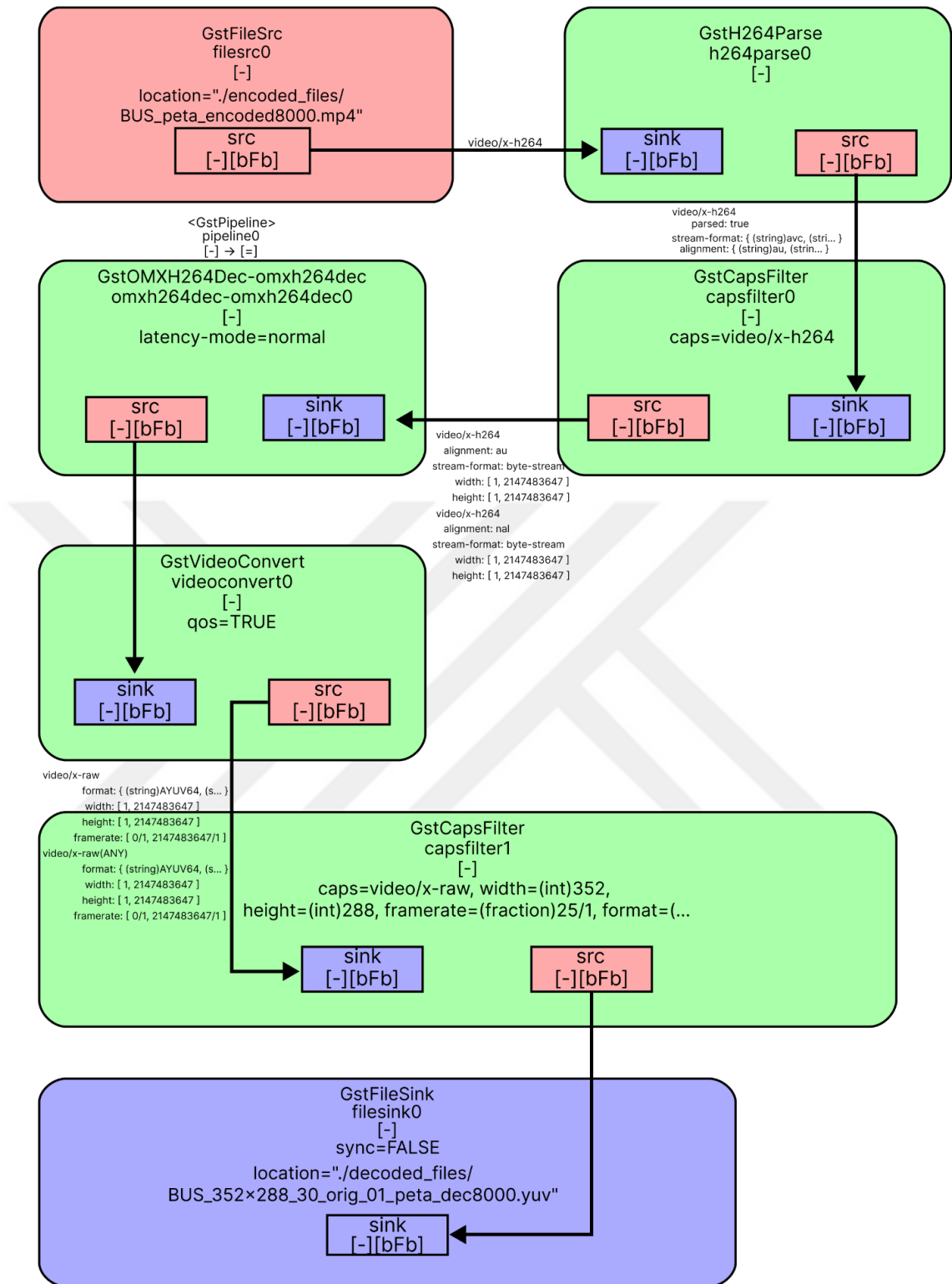


Figure 3.5. Representation of Decoding Pipeline

4. COMPARISON BETWEEN EMBEDDED LINUX AND COMPUTER

The completion of the preparation phase initiated the device initialization, and the compiled operating system integrated the configurations added and adjusted in the digital design. Connectivity to the internet was established through Ethernet, and Secure Shell (SSH) facilitated the device control. Consequently, the system became operational, ready to receive commands. Upon establishing the connection, raw media files were transmitted to the device using the Secure Copy Protocol (SCP). These media files underwent compression through the prepared pipeline commands. Following compression, the compressed files were decompressed, repeating the entire process for five media files. The hardware accelerator, powered by GStreamer on the embedded Linux system, and FFmpeg on the computer featuring Intel Xeon W-1350, 32 GB of RAM, and Ubuntu 22.04 Kernel 6.2.0-36-generic, executed this iterative process. At the top, the bitrate was configured at 8000 kbit/s, and the video, initially encoded and subsequently decoded, remained imperceptible to the human eye. This outcome is visually represented in “Fig. 4.1.”



Figure 4.1. The raw original media on the left and h264 compressed by Petalinux at the right-hand side.[72]

This operation resulted in a video size reduction by a ratio of 79.1 percent (from 60.3 MB to 12.6 MB for the given sample video "crew") through encoding.

Subsequently, the obtained media files underwent analysis using PSNR, a previously mentioned quality metric. PSNR values were acquired for each processed medium, and these values were compared across the five videos. The differences are depicted in "Fig. 4.2." as the average PSNR of the three pixel parameters for all frames, namely Y (brightness or luma), blue minus brightness (Cb), and red minus brightness (Cr), averaged across all frames of the video. As illustrated in the figure, the PSNR difference falls between 3 and 5. This indicates that the human eye is unlikely to discern any difference, even when recognizing images akin to "Fig. 2.17.," which possesses a PSNR value of 40 at its most visually appealing photo. On the graph, it is evident that every video file compressed by VCU exhibits a PSNR value above 40. The formula for calculating this difference is provided in Equation 4.1.

$$\text{PSNR}_Y = 10 \cdot \log_{10} \left(\frac{\text{Max}^2}{\text{MSE}_Y} \right)$$

$$\text{PSNR}_U = 10 \cdot \log_{10} \left(\frac{\text{Max}^2}{\text{MSE}_U} \right)$$

$$\text{PSNR}_V = 10 \cdot \log_{10} \left(\frac{\text{Max}^2}{\text{MSE}_V} \right)$$

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (I_i - K_i)^2$$

$$\text{Total PSNR} = \text{PSNR}_Y + \text{PSNR}_U + \text{PSNR}_V$$

$$\text{Average PSNR} = \frac{\text{Total PSNR}}{3} \quad (4.1)$$

The MSE value and calculation of PSNR were discussed previously in Equation 2.1. With this equation, overall performance and noise ratio can be shown.

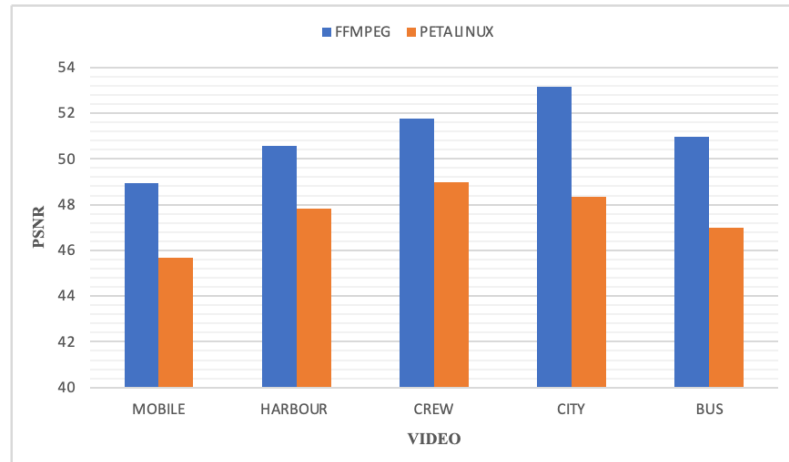


Figure 4.2. Graph of PSNR values for comparing Computer (FFmpeg) and EmbeddedLinux (Petalinux)

The graph illustrates the subtle disparity in quality between computer and hardware-accelerated embedded Linux, with corresponding values. Despite the computer exhibiting slightly superior performance, the distinction is imperceptible to the naked eye, as evidenced in “Fig. 4.1.” Factors such as compactness, productivity, and simultaneous display recording during encoding and saving can be pivotal game-changers. During compression, it is imperative to emphasize that there is no discernible difference in compressed media sizes between computers and embedded Linux (Petalinux). This parity is maintained because both adhere to identical bitrates and encoding standards. Post the average output from the provided media, a frame-by-frame comparison becomes essential for enhanced comprehension. The graph in “Fig. 4.3.” delineates 150 video frames, comparing different components and presenting their average PSNR values. Within this specific graph, the success of the Cr value surpasses other metrics. This success can be attributed to the encoding and decoding processes effectively preserving the red-blue color difference value (Cr) compared to Cb and Y values. If Cb were to exhibit better performance, it would signify improved preservation of the blue and green color differences, whereas an emphasis on Y would prioritize luminance.

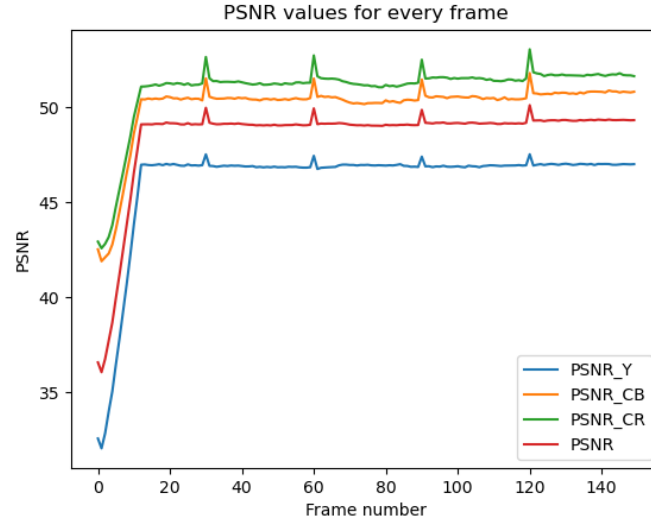


Figure 4.3. Graph of PSNR value for every frame and parameter on the video Crew that encoded and decoded through Embedded Linux System

An alternative approach to assess video decoding quality involves utilizing the VMAF metric, as previously mentioned. VMAF, closely aligned with human visual perception, indicates that the distinction between the two devices is not significant. Consequently, the variance from the original is minimal in both videos, as observed in “Fig. 4.4.”. In accordance with the details provided in “Section 2.8.2”, VMAF relies on three metrics for its calculations. Unlike PSNR, which is rooted in a mathematical formula, VMAF utilizes a machine learning model inspired by the human eye. The first metric is Visual Information Fidelity (VIF), an image quality metric based on statistical analysis and a model derived from human vision. The second metric, Detail Loss Metric (DLM), centers around the loss of detail. The third metric, mean co-located pixel difference (MCPD), hinges on luminance components. In light of these insights, the disparity between the two systems is not substantial, and there is little discernible difference in terms of the VMAF score.

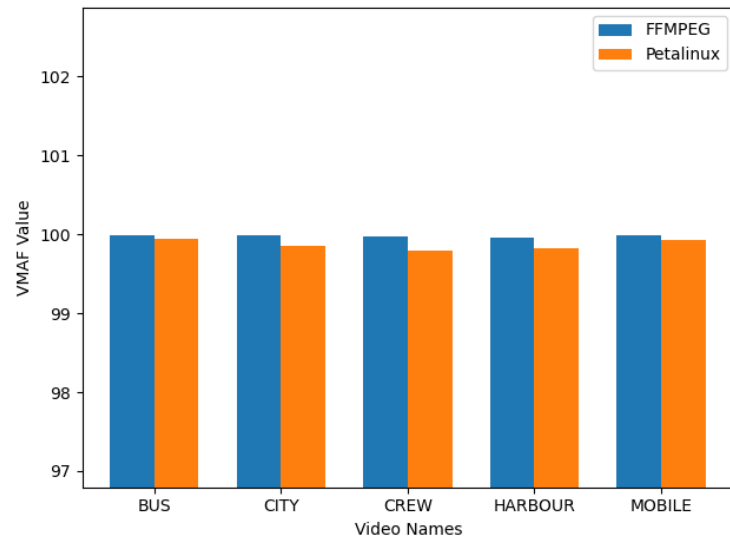


Figure 4.4. Graph of VMAF values for 5 different videos for comparing computer (FFmpeg) and embedded Linux (Petalinux)

5. CONCLUSION

The rapid growth of the digital era, driven by widespread social media use, has led to a significant increase in the digital industry. However, this surge in digital content size has created challenges in transferring and storing files. To address this, encoding algorithms have become crucial, allowing videos to be compressed during transfer or storage and then decoded in real-time on target devices.

In the landscape of video encoding, the evolution from traditional software-based encoding to the emergence of specialized Video Codec Units (VCUs) signifies a substantial development in the industry. In the earlier stages, video encoding predominantly relied on software-based approaches executed on general-purpose Central Processing Units (CPUs). This method, while functional, often struggled with limitations in terms of speed, efficiency, and cost-effectiveness.

The advent of specialized Video Codec Units marked a pivotal shift in the encoding paradigm. Unlike their software counterparts, VCUs are dedicated hardware units specifically designed to handle encoding and decoding tasks with enhanced efficiency and performance. This transition has significantly improved the speed and effectiveness of video encoding processes, addressing the demand for higher throughput and lower latency in multimedia applications.

Older software encoding methods were characterized by their dependence on the computational power of CPUs alone. This approach posed challenges, particularly as video resolutions and complexities increased over time. The dedicated hardware acceleration introduced by Video Codec Units, on the other hand, optimizes encoding tasks by offloading them from the general-purpose CPU to a specialized unit. This not only enhances the speed of the encoding process but also allows the CPU to focus on other essential tasks concurrently.

Embedded Linux, with its robust kernel, has proven to be an ideal platform for managing tasks related to video codec units. This work aims to showcase the capabilities of embedded Linux in encoding and decoding, providing insights for those seeking to implement a video codec unit. The development of a digital design for a Video Codec Unit and peripherals, along with the configuration of necessary parameters, is explored.

The central contribution of this research resides in its investigation of Embedded Linux Systems, an area of technology that has received comparatively less attention. By creating a Petalinux environment dedicated to resource management, kernel configuration, and Video Codec Unit implementation, this study offers valuable insights for researchers and practitioners. A significant focus is placed on advocating for a thorough digital design approach in the implementation of a Video Codec Unit and associated peripherals. This comprehensive methodology is underscored by the examination of quality trade-offs, facilitated by established metrics such as Video Multimethod Assessment Fusion (VMAF) and Peak Signal-to-Noise Ratio (PSNR).

This thesis addresses the gap in literature regarding the use of embedded Linux in the context of Video Codec Unit implementation and its impact on media compression. By building a Petalinux environment and utilizing the Gstreamer media library, the research not only contributes to the field but also serves as a valuable resource for those aiming to understand and adopt these technologies.

Additionally, the thesis explores how quality loss occurs during compression, offering practical information for decision-makers in the industry. The results obtained from computers with standardized metrics are presented quantitatively and validated with visual support. This dual approach provides a comprehensive understanding of the encoding and decoding process.

Moving forward, this work opens avenues for future research in the optimization of embedded Linux systems for various applications. It emphasizes the significance of considering quality trade-offs and the practical aspects of implementation when deploying Video Codec Units in various contexts. The thesis stands as a testament to the potential of embedded Linux in modern digital workflows, offering a bridge between high-performance encoding demands and cost-effective solutions.

The novelty of this work lies in its integration of embedded Linux with Video Codec Unit implementation, bringing together two crucial technologies. The establishment of a Petalinux environment adds a new dimension, simplifying resource management and configuration for users. Additionally, the research employs VMAF and PSNR metrics to

quantify the quality differences, providing a thorough assessment of the encoding and decoding process.

Future works in the realm of video encoding, embedded systems, and digital media processing hold promising directions. One avenue for exploration involves optimizing Embedded Linux systems for specific applications, tailoring kernel configurations and resource management to address the unique requirements of real-time video streaming or high-resolution content processing. Another area of potential advancement lies in enhancing Video Codec Unit implementations, focusing on the customization of hardware architectures and the integration of advanced features to meet evolving industry standards. Further research could delve into the real-time adaptability of embedded systems, investigating algorithms within Embedded Linux that dynamically adjust parameters based on varying processing loads or network conditions during video encoding and decoding. Additionally, considering the growing importance of energy efficiency in digital media applications, exploring strategies to optimize power consumption and improve energy efficiency in video processing on embedded systems emerges as a critical and relevant research direction. These endeavors collectively aim to contribute to the ongoing evolution of embedded systems and video processing, addressing the diverse needs of modern multimedia applications.

REFERENCES

- [1] “Application-Specific Integrated Circuit.” Wikipedia, 5 July 2021, en.wikipedia.org/wiki/Application-specific_integrated_circuit.
- [2] Fessenden, Marissa. “The World of Personal Computers in the 1980s Was a Wacky, Wonderful Place.” Smithsonian Magazine, www.smithsonianmag.com/smart-news/world-personal-computers-1980s-was-wacky-wonderful-place-180953274/.
- [3] Wikipedia Contributors, “Field-programmable gate array,” Wikipedia, Apr. 07, 2019. https://en.wikipedia.org/wiki/Field-programmable_gate_array
- [4] "G. Braad, “Altera EP300-DC,” Flickr, May 04, 2010. <https://www.flickr.com/photos/gbraad/4578576323/in/photostream/> (accessed Sep. 12, 2023).
- [5] Wikipedia Contributors, “Microcontroller,” Wikipedia, Mar. 14, 2019. <https://en.wikipedia.org/wiki/Microcontroller>
- [6] Wikipedia Contributors, “Intel 4004,” Wikipedia, Jan. 31, 2020. https://en.wikipedia.org/wiki/Intel_4004
- [7] “File:Intel 4004 open.jpg - Wikipedia,” commons.wikimedia.org. https://en.wikipedia.org/wiki/File:Intel_4004_open.jpg (accessed Sep. 17, 2023).
- [8] “Apple M1,” Wikipedia, Sep. 11, 2023. https://en.wikipedia.org/wiki/Apple_M1#/media/File:Mac_Mini_M1_chip.jpg (accessed Sep. 17, 2023).
- [9] “Video Compression,” Haivision. <https://www.haivision.com/resources/streaming-video-definitions/compression/>
- [10] U. Stanimirovic, “Best Video Codecs to Use and How to Choose the Right One,” BRID TV, Sep. 30, 2020. <https://www.brid.tv/best-video-codecs-to-use-and-how-to-choose-the-right-one/>
- [11] “Video coding format,” Wikipedia, Jun. 26, 2023. https://en.wikipedia.org/wiki/Video_coding_format (accessed Aug. 06, 2023).

- [12] R. Sharma and S. Agrawal, "Image Compression using DPCM," GRD Journal for Engineering, vol. 2, 2017, Accessed: Aug. 06, 2023. [Online]. Available: <https://grdjournals.com/uploads/article/GRDJE/V02/I04/0044/GRDJEV02I040044.pdf>
- [13] "stoimen's web log," May 03, 2012. <http://www.stoimen.com/tag/run-length-encoding/> (accessed Jan. 08, 2024).
- [14] "H.120," Wikipedia, May 30, 2020. <https://en.wikipedia.org/wiki/H.120> (accessed Aug. 06, 2023).
- [15] "H.261," Wikipedia, May 10, 2023. <https://en.wikipedia.org/wiki/H.261#:~:text=in%20practical%20terms.-> (accessed Aug. 06, 2023).
- [16] "What is MPEG-1? – Accurate Secretarial LLC." <http://www accuratesecretarial.com/2019/05/04/what-is-mpeg-1/> (accessed Aug. 06, 2023).
- [17] "H.262/MPEG-2 Part 2," Wikipedia, Jan. 21, 2023. https://en.wikipedia.org/wiki/H.262/MPEG-2_Part_2 (accessed Aug. 06, 2023).
- [18] "Inter frame," Wikipedia, Sep. 21, 2022. https://en.wikipedia.org/wiki/Inter_frame (accessed Aug. 06, 2023).
- [19] "H.263," Wikipedia, May 20, 2022. <https://en.wikipedia.org/wiki/H.263> (accessed Aug. 06, 2023).
- [20] "H.264 Video Compression | Video Streaming Definition," Haivision. <https://www.haivision.com/resources/streaming-video-definitions/h-264/>
- [21] Christian. "ChristianFeldmann/PresentationMaterial." GitHub, 24 Nov. 2023, github.com/ChristianFeldmann/PresentationMaterial?tab=CC0-1.0-1-ov-file#readme. Accessed 11 Jan. 2024.
- [22] Zhu, Albert. "Process of H.264 Encoding -- Intra Inter Prediction, Macroblock, Frame Type..." Wwww.youtube.com, 3 Feb. 2023, www.youtube.com/watch?v=ZXXDXZfEcAQ. Accessed 11 Jan. 2024.
- [23] Feldmann, Christian. "Video Coding Basics - How Is This so Efficient?" Wwww.youtube.com, 20 May 2022, www.youtube.com/watch?v=LDeL7-49qm4. Accessed 11 Jan. 2024.

- [24] Secure Arithmetic Coding with Error Detection Capability - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/An-example-of-arithmetic-coding-the-source-symbols-are-a-b-c-with-pa-0011-pb_fig2_47696414 [accessed 11 Jan, 2024]
- [25] Wiegand, T. et al. "Rate-Constrained Coder Control and Comparison of Video Coding Standards." IEEE Transactions on Circuits and Systems for Video Technology 13.7 (2003): 688–703. Web.
- [26] "6 Points of Comparison for VP9 or H.265," Red5 Pro, Sep. 02, 2020. <https://www.red5pro.com/blog/6-points-of-comparison-for-vp9-or-h265/> (accessed Aug. 06, 2023).
- [27] "VP9," Wikipedia, Mar. 01, 2021. <https://en.wikipedia.org/wiki/VP9>
- [28] R. S. Bultje, "VP9 encoding/decoding performance vs. HEVC/H.264 | Ronald S. Bultje," Gnome. <https://blogs.gnome.org/rbultje/2015/09/28/vp9-encodingdecoding-performance-vs-hevch-264/> (accessed Aug. 10, 2023).
- [29] "AV1," Wikipedia, Feb. 25, 2023. <https://tr.wikipedia.org/wiki/AV1> (accessed Aug. 09, 2023).
- [30] "Versatile Video Coding," Wikipedia, Mar. 27, 2023. https://en.wikipedia.org/wiki/Versatile_Video_Coding
- [31] K. Torres, "H.266 VVC vs AV1, Which is Better?," www.winxdvd.com. <https://www.winxdvd.com/video-transcoder/h266-vvc-vs-av1.htm> (accessed Aug. 11, 2023).
- [32] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 2010, pp. 2366-2369, doi: 10.1109/ICPR.2010.579.
- [33] "Measures of image quality," homepages.inf.ed.ac.uk. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.html
- [34] Li, Z., Bampis, C., Novak, J., Aaron, A., Swanson, K., Moorthy, A., Cock, J. D. (2018). VMAF: The journey continues. Netflix Technology Blog, 25(1).
- [35] "What would you like to see most in minix?," groups.google.com. <https://groups.google.com/g/comp.os.minix/c/dlNtH7RRrGA/m/SwRavCzVE7gJ>

- [36] “What is embedded Linux? Part I,” Ubuntu. <https://ubuntu.com/blog/what-is-embedded-linux>
- [37] “How to Use the BusyBox Docker Official Image | Docker,” www.docker.com, Jul. 14, 2022. <https://www.docker.com/blog/use-cases-and-tips-for-using-the-busybox-docker-official-image/#:~:text=What%27s%20BusyBox%3F> (accessed Aug. 15, 2023).
- [38] “BusyBox,” Wikipedia, Nov. 23, 2020. <https://en.wikipedia.org/wiki/BusyBox>
- [39] T. original uploader was R. at E. Wikipedia, “English: BusyBox running under the Microsoft Telnet Client from my Netgear Router,” Wikimedia Commons, Mar. 23, 2007. https://commons.wikimedia.org/wiki/File:Busybox_DG834Gt.PNG#/media/File:Busybox_DG834Gt.PNG (accessed Aug. 15, 2023).
- [40] “Timesys - We build, secure maintain embedded Linux, Android, open source OS software platforms,” Timesys. <http://www.timesys.com/> (accessed Aug. 16, 2023).
- [41] “Embedded Linux timeline | 2net.co.uk,” 2net.co.uk. <https://2net.co.uk/embedded-history> (accessed Aug. 16, 2023).
- [42] “Postcards from the Bleeding Edge: Who invented the embedded Linux based wireless router?” <http://the-edge.blogspot.com/2010/10/who-invented-embedded-linux-based.html> (accessed Aug. 16, 2023).
- [43] “UClinux,” community.intel.com, Jun. 26, 2019. <https://community.intel.com/t5/FPGA-Wiki/UClinux/ta-p/735614> (accessed Aug. 16, 2023).
- [44] “Welcome to uClibc-ng! - Embedded C library,” uclibc-ng.org. <https://uclibc-ng.org> (accessed Aug. 16, 2023).
- [45] “Buildroot - Making Embedded Linux Easy,” buildroot.org. <https://buildroot.org>
- [46] “Das U-Boot,” Wikipedia, Mar. 04, 2020. https://en.wikipedia.org/wiki/Das_U-Boot
- [47] “File:U-boot.png - Wikipedia,” commons.wikimedia.org, Sep. 06, 2011. <https://en.wikipedia.org/wiki/File:U-boot.png> (accessed Aug. 21, 2023).

- [48] “Linksys WRT54G series,” Wikipedia, Jul. 27, 2023. https://en.wikipedia.org/wiki/Linksys_WRT54G_series (accessed Aug. 21, 2023).
- [49] “PetaLinux Tutorial+Demo.” Accessed: Aug. 21, 2023. [Online]. Available: https://indico.cern.ch/event/799275/contributions/3413736/attachments/1861349/3059191/Petalinux_TutorialDemo.pdf
- [50] “Device tree overlays,” Android Open Source Project. <https://source.android.com/docs/core/architecture/dto> (accessed Aug. 21, 2023).
- [51] “Linux Device tree,” wiki.dreamrunner.org. http://wiki.dreamrunner.org/public_html/Embedded-System/Linux-Device-Tree.html (accessed Aug. 21, 2023).
- [52] “AMD Adaptive Computing Documentation Portal,” docs.xilinx.com. <https://docs.xilinx.com/v/u/en-US/ug1157-petalinux-tools-command-line-guide> (accessed Aug. 22, 2023).
- [53] Wikipedia Contributors, “Kernel (operating system),” Wikipedia, Sep. 04, 2019. [https://en.wikipedia.org/wiki/Kernel_\(operating_system\)](https://en.wikipedia.org/wiki/Kernel_(operating_system))
- [54] “Unix shell,” Wikipedia, Nov. 30, 2020. https://en.wikipedia.org/wiki/Unix_shell
- [55] “Introduction to Linux Shell and Shell Scripting,” GeeksforGeeks, Jun. 02, 2017. <https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/>
- [56] “Thompson shell,” Wikipedia, Jun. 08, 2023. https://en.wikipedia.org/wiki/Thompson_shell (accessed Sep. 05, 2023).
- [57] “PWB shell,” Wikipedia, Aug. 24, 2023. https://en.wikipedia.org/wiki/PWB_shell (accessed Sep. 05, 2023).
- [58] “Bourne shell,” Wikipedia, Jul. 31, 2021. https://en.wikipedia.org/wiki/Bourne_shell
- [59] Wikipedia Contributors, “Bash (Unix shell),” Wikipedia, Oct. 12, 2019. [https://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](https://en.wikipedia.org/wiki/Bash_(Unix_shell))
- [60] “Berkeley r-commands,” Wikipedia, Apr. 13, 2023. https://en.wikipedia.org/wiki/Berkeley_r-commands
- [61] Wikipedia Contributors, “Telnet,” Wikipedia, Dec. 13, 2019. <https://en.wikipedia.org/wiki/Telnet>

- [62] Wikipedia Contributors, “File Transfer Protocol,” Wikipedia, Aug. 20, 2019.
https://en.wikipedia.org/wiki/File_Transfer_Protocol
- [63] Wikipedia Contributors, “Secure Shell,” Wikipedia, Jun. 20, 2019.
https://en.wikipedia.org/wiki/Secure_Shell
- [64] J. Ellingwood, “Understanding the SSH Encryption and Connection Process,” DigitalOcean, Oct. 22, 2014.
<https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process>
- [65] E. D, “SSH nedir? SSH nasıl çalışır?,” Hostinger Rehberler, Nov. 09, 2017.
<https://www.hostinger.web.tr/rehberler/ssh-nedir>
- [66] SSL2BUY, “Symmetric vs. Asymmetric Encryption – What Are differences?” SSL2BUY Wiki - Get Solution for SSL Certificate Queries, Feb. 07, 2019. <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>
- [67] J. Nduati, “Understanding Hashing in Cryptography,” Engineering Education (EngEd) Program | Section, Jan. 05, 2021. <https://www.section.io/engineering-education/understand-hashing-in-cryptography/>
- [68] M. Karalar, “Medium,” Medium, Jul. 17, 2018.
<https://medium.com/@muhammedkaralar/diffie-hellman-anahtar-de> (accessed Sep. 07, 2023).
- [69] “Dropbear (software),” Wikipedia, Dec. 31, 2022.
[https://en.wikipedia.org/wiki/Dropbear_\(software\)](https://en.wikipedia.org/wiki/Dropbear_(software)) (accessed Sep. 07, 2023).
- [70] Xilinx. (2022, May 17). Kria KR260 Robotics Starter Kit Data Sheet (DS988). Retrieved November 3, 2023, from docs.xilinx.com website:
`\url{https://docs.xilinx.com/r/en-US/ds988-kr260-starter-kit}`
- [71] “MPEG-4 Part 2,” Techopedia, Apr. 30, 2020.
<https://www.techopedia.com/definition/10756/mpeg-4-part-2> (accessed Aug. 06, 2023).
- [72] Purdue.edu, 2024,
engineering.purdue.edu/~reibman/ece634/Videos/video_files.htm. Accessed 8 Jan. 2024.

- [73] “Motion-Compensated Hybrid Video Encoding - Video Compression,” Coursera.
<https://www.coursera.org/lecture/digital/motion-compensated-hybrid-video-encoding-k1MRZ> (accessed Aug. 06, 2023).
- [74] “System on a chip,” Wikipedia, Apr. 18, 2020.
https://en.wikipedia.org/wiki/System_on_a_chip
- [75] G. Sundari, T. Bernatin and P. Somani, "H. 264 encoder using Gstreamer," 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015], Nagercoil, India, 2015, pp. 1-4, doi: 10.1109/ICCPCT.2015.7159511.
- [76] Dubey, Ayush. “Image Compression.” Medium, 25 July 2021, medium.com/@ayush.pegasus2801/image-compression-f3c5f5320630.