Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Integration and evaluation of an FPGA-based accelerator card in the workflow of Opencast

Sven Dyhr

Course of Study:	Informatik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dipl.-Inf. Pascal Seeland
Commenced:	March 1, 2022
Completed:	September 1, 2022

Abstract

The open source software 'Opencast' offers a complete video processing system that includes all necessary components for recording, processing and distribution of video files like lecture recordings. In the course of a workflow, the files are encoded according to the requirements using a modern video compression standard. Typically, the algorithms for this are run on the processor of a machine, but various hardware accelerators exist that promise more efficient processing. This work deals with the integration of an acceleration card based on FPGA technology into an Opencast workflow. For this purpose, appropriate adaptations to an existing workflow as well as suitable encoding profiles are designed and implemented. An evaluation is then carried out that compares the FPGA acceleration card with a software-only implementation. Meaningful criteria and metrics are considered, presented and then applied to test files.

Kurzfassung

Die Open-Source Software 'Opencast' bietet ein komplettes Videoverarbeitungssystem, welches alle nötigen Bestandteile zur Aufnahme, Bearbeitung und Verteilung von Videodateien wie Vorlesungsaufzeichnungen beinhaltet. Im Laufe eines Workflows werden die Dateien passend zu den Anforderungen mit einem modernen Videokompressionsstandard kodiert. Üblicherweise werden die Algorithmen hierfür auf dem Prozessor ausgeführt, es existieren jedoch auch diverse Hardware-Beschleuniger, die eine effizientere Verarbeitung versprechen. Diese Arbeit beschäftigt sich damit, eine auf FPGA-Technologie basierende Beschleunigungskarte in einen Workflow von Opencast einzubinden. Hierfür werden entsprechende Anpassungen an einen bestehenden Workflow sowie passende Encoding-Profile entworfen und umgesetzt. Anschließend wird eine Evaluation durchgeführt, die die FPGA-Beschleunigungskarte mit einer reinen Software-Lösung vergleicht. Es werden sinnvolle Kriterien und Metriken überlegt, vorgestellt und anschließend an Testdateien angewendet.

Acronyms

ASIC Application-specific Integrated Circuit. 18

AWS Amazon Web Services. 33

CABAC Context-adaptive Binary Arithmetic Coding. 22

CAVLC Context-based Adaptive Variable Length Coding. 20

CBR Constant Bit Rate. 40

CLB Configurable Logic Block. 17

CPU Central Processing Unit. 15

CRF Constant Rate Factor. 40

DCT Discrete Cosine Transform. 21

DST Discrete Sine Transform. 22

EC2 Elastic Compute Cloud. 33

FPGA Field Programmable Gate Array. 15

fps Frames per second. 19

GPU Graphics Processing Unit. 15

HEVC High Efficiency Video Coding. 22

I/O Input/Output. 17

LUT Lookup Table. 17

MFX Multi-Format Codec. 7

MPSoC Multiprocessor System on a Chip. 35

MSE Mean Square Error. 51

NAL Network Abstraction Layer. 20

PSNR Peak Signal-to-Noise Ratio. 29

Px Pixel. 19

QSV Quick Sync Video. 7, 27

SSIM Structural Similarity Index Measure. 51

TDP Thermal Design Power. 55

UVD Unified Video Decoder. 28

VBR Variable Bit Rate. 40

VCE Video Coding Engine. 28

VCL Video Coding Layer. 20

vCPU virtual processor. 34

VCU Video Codec Unit. 35

VMAF Video Multi-Method Assessment Fusion. 52

XML Extensible Markup Language. 25

List of Figures

List of Tables

Listings

Contents

1 Introduction

The global pandemic caused by the coronavirus SARS-CoV-2, also known as COVID-19, changed how we treated each other in all areas of our lives. This also affected students throughout Germany, including the University of Stuttgart. Suddenly, lectures were no longer possible on site, prompting a shift to digital services such as live streams and lecture recordings. In order to give all students the chance to attend lectures in the best possible way even during this difficult time, it was suggested by the university's Digital Teaching Taskforce that appropriate recordings be made available to students. For this purpose, the task force 'defined interdisciplinary recommendations for online teaching events'. The Rectorate also 'emphatically supports these recommendations and views them as an important orientation tool during the current pandemic situation' [ILIAS20]. These recommendations and their implementation allowed students to continue with the curriculum as normal as possible. However, records also provide benefits to students independent of the pandemic, which is why the system is optimally maintained in a hybrid process: There will be a lecture on site, of which the recording will be made available afterwards.

In order to realize and implement this project, an appropriate infrastructure is required. In this context, the Technical Information and Communication Services of the University hosts and provides the following open-source solutions:

ILIAS. Developed since 1998, it is a learning management system, which offers a wide range of functions for both, students and lecturers, like Course management, learning modules, tests and assessments, portfolios, surveys, wikis and blogs [ILIAS-About].

Opencast. First released as 'Opencast Matterhorn' in 2010, it provides a 'flexible, reliable and scalable open-source video recording, management and distribution system for academic institutions [Opencast-About].

The video material is processed by the Opencast servers and, if configured to do so, automatically uploaded to ILIAS and other distribution channels. It needs to be transcoded as fast as possible without losing too much of its quality to ensure the availability of resources for the next lecture. There are several approaches for the transcoding process. We will take a look at the currently available and widely used solutions that use a computer's Central Processing Unit (CPU) or a Graphics Processing Unit (GPU) with hardware encoding circuits to transcode the video files in Chapter 3. We will also investigate a solution that uses hardware accelerator cards based on Field Programmable Gate Arrays (FPGAs) technology designed, programmed and optimized for video transcoding with support for state-of-the-art video compression standards like H.264 and H.265.

In this paper we address how easily the FPGA accelerator card can be integrated into the workflow of Opencast and how the hardware accelerator compares in terms of performance and quality to the pure CPU-based solution.

Methodology

In order to approach and solve the problem, we need to have a basic understanding of the files we will be working with in the course of this paper. Therefore, we will first have to analyze how video files are stored in an efficient way nowadays. We will explicitly look at standards that are also supported by the chosen FPGA acceleration card. In addition, current software and hardware solutions for converting video files and their advantages and disadvantages will be discussed.

Furthermore, it is important to get to know Opencast itself and other relevant tools. We will analyze how Opencast is structured, how to control and customize its operations and how to integrate the different software and hardware solutions for video conversion.

The integration performed will be based on the environment used and the subsequent evaluation. The data collected using the FPGA solution will be evaluated using meaningful and suitable criteria and metrics.

Last but not least, the thesis and its findings are summarized and considerations are made on how to further improve the implemented solution.

Structure of this paper

- **Chapter 2** addresses background information of this work. The technologies and tools used are described here.
- **Chapter 3** will present the current available state-of-art-solutions available for video transcoding using CPUs or GPUs.
- **Chapter 4** shows related work dealing with similar topics.
- **Chapter 5** includes the planning process for the Opencast workflow as well as an overview of the infrastructure and source materials used.
- **Chapter 6** gives an overview over the implemented workflows and encoding profiles.
- **Chapter 7** contains the evaluation criteria, the gathered data and the evaluation.
- **Chapter 8** summarizes the results of the thesis, gives an outlook on the usability of this solution and presents ways in which it could be further improved.

2 Background

2.1 Field Programmable Gate Array

As the name implies, a FPGA is an *field-programmable* integrated circuit with the ability to design and configure the logic behind it after manufacturing, consisting of an array of logic blocks. Because algorithms are not processed with instructions as in CPUs, but directly with an interconnection of logic gates in these logic blocks, a very efficient solution for specific applications is possible.

The most common FPGA architecture consists of three different main modules, as shown in Figure 2.1: An array of Configurable Logic Block (CLB), Input/Output (I/O) Pads and Interconnection Circuits.

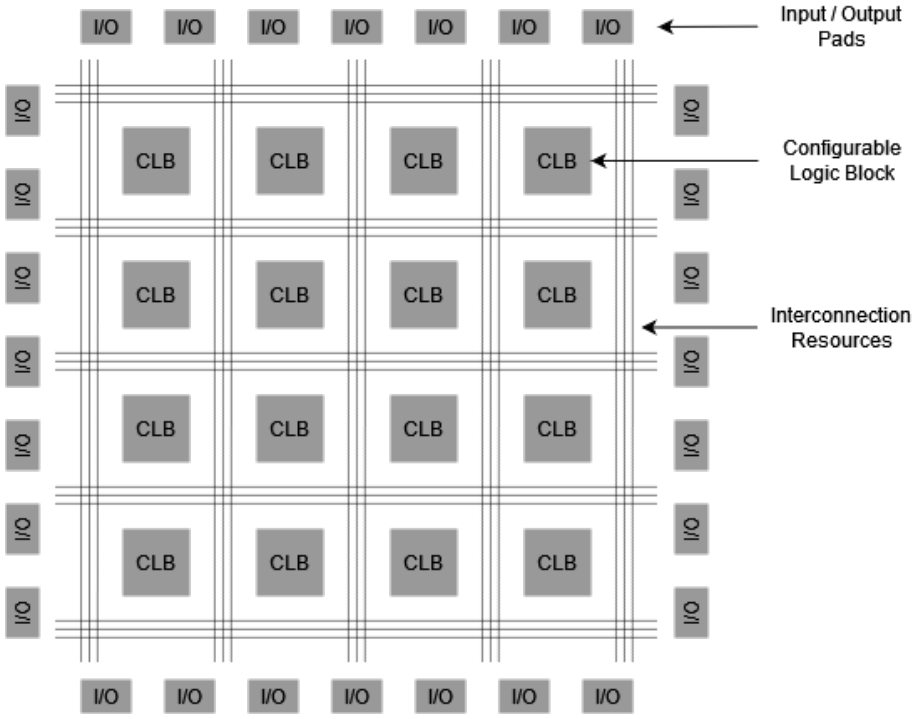

Figure 2.1: Simplified design of a basic FPGA

Each **CLB** represents a self-contained logic block that receives its input from and outputs to the routing infrastructure. They consist of one or more logic cells and each of these cells typically contains a Lookup Table (LUT) and multiple circuits like flip-flops, multiplexers and arithmetic circuitry to control the function of the logic block. The LUTs are usually denoted as *K*-LUT where K input signals can be used. The most common sizes for LUTs are 4 to 6 since they offer a good trade-off between the area needed and the performance of the logic cell.

I/O Pads offer direct input and output interfaces for the FPGA to communicate with external signals. These need to be routed to the correct CLBs and are usually grouped in multiple I/O banks for different voltages and connections. For example, one could implement a group for serial connections such as PCIe that can achieve speeds of several Gb/s with the help of additional circuitry.

The **Interconnection Circuits** connects the CLBs and I/O blocks. It consists not only of simple wires interconnecting all available CLBs, but also of programmable switches. By appropriate programming, any output of a CLB can be routed to an input of another CLB by using the available routing channels and switches.

In addition, modern FPGA designs not only use the existing FPGA blocks to implement all functionality, but offload common functions to an embedded microprocessor, forming a 'system on a programmable chip'. This reduces the space required on the chip and can greatly increase the processing speed of various functions.

While one of the biggest advantages of FPGAs is reconfigurability of the function circuitry, the engineering effort to implement the wanted functions is much higher than in instruction based circuits. A deep knowledge of hardware description languages and digital system fundamentals is necessary. Additionally, the overhead for compilation is higher because it doesn't boil down to a single instruction set. Above all, the translation between desired circuit and the available resources of the FPGAs with paths as short as possible plays an important role. It minimizes the latency between CLBs, therefore decreasing the final run time. Nevertheless, compared to other integrated circuits, especially Application-specific Integrated Circuits (ASICs), FPGAs offer lower recurring engineering cost and shorter time-to-market, since after the initial design of the logic, the 'physical design, layout, fabrication and verification stage can be skipped' [BB21]. Also, customization of the logic is easy by reprogramming, whereas with an ASIC the integrated circuit would have to be replaced.

2.2 Digital Video

A digital video is a sequence of images, called frames, and thus part of the spatial and temporal domain. Each frame is a still image with a given resolution, measured in the number of pixels along the width and height. Typical resolutions used today are 1280 x 720 (HD), 1920 x 1080 (FullHD) or 3840 x 2160 (4K). Each pixel stores a set of information defined by the color space used like RGB and YC_BC_R.

The RGB color space describes the ratio between the main colors Red, Blue and Green in each Pixel. A common amount of information for each of the color channels is 8 bits or 1 byte per channel, for a total of 24 bits or 3 bytes.

While RGB is intended for an approximately accurate representation of color information, YC_BC_R uses a weighted estimation of the individual color channels to match human color perception which is more sensitive to brightness than colors. To calculate YC_BC_R from RGB, first each value of RGB is normalized into the interval [0,1], giving us the values (R', G', B'). Y denotes the luminance and is a weighted sum of these channels.

$$Y = K_R * R' + K_G * G' + K_B * B'$$

The actual weights K_R, K_G, K_B differ between standards. To give an example, *ITU-R BT.601* [BT.601] specifies the tuple (K_R, K_G, K_B) as $(0.299, 0.587, 0.114)$, while *ITU-R BT.2020* [BT.2020] defines it as $(0.2627, 0.6780, 0.0593)$. The weights are based on the importance of the respective color for the human luminance perception.

Then, the color information for both, the blue-yellow chrominance and the red-green chrominance, is calculated with the following formulas to achieve the YP_BP_R representation used for analog images.

$$P_B = \frac{1}{2} * \frac{B'-Y'}{1-K_B}$$
$$P_R = \frac{1}{2} * \frac{R'-Y'}{1-K_R}$$

From there, an offset is added according to the used standard to receive the corresponding digital YC_BC_R colors.

In addition, we can save further space by taking advantage of the above-mentioned human perception. Since luminance is more important than colors, a method named 'chroma subsampling' is often used. While the luminance channel is rendered fully, the two color channels are stored in a lower resolution. Even though we reduce the information available, the video will still look very similar for the human eye. A subsample scheme is specified in a three-part ratio such as 4:2:0 or 4:4:4 and refers to a 4 Pixel (Px) wide and 2 Px high region. The first number, usually 4, refers to the width of the region. The second number indicates the number of different chromatic samples in the first row and the third number denotes the changes of the chromatic samples between the first and second row. The third value must be either 0 or the second value. For example for 4:2:0, the luminance data is stored for every pixel but the color data of the chromatic channels are only stored once every 2x2 pixel block, greatly reducing the total amount of information.

Still, video files contain a lot of data, stored in each pixel. Without proper compression, these frames can take a lot of storage space on the users devices and increasing the requirements for every infrastructure around video distribution. To give an very naive estimation of the file size purely from a video without any compression or encoding, we can use different properties of the video. If we are assuming that each pixel contains n color channels with m bytes of information each and they are all stored independently, the space required for one pixel equals $n * m$ bytes. Multiplying this value by the resolution, the frame rate (given in Frames per second (fps)) and the run time, we get the following formula:

$$\text{File Size}_{video} = time * framerate * width * height * n * m$$

To give an example, we assume a video file with the resolution of 1280 x 720, 24 fps and three color channels with 1 byte each. To store 60 seconds, this requires the following amount of space:

$$\begin{aligned}\text{File Size}_{video} &= 60s * 24\frac{frames}{s} * 1280 * 720 * 3 * 1\ \text{B} \\ &= 3.981.312.000\ \text{B} \approx 3.71\ \text{GiB}\end{aligned}$$

That's obviously a lot of space required for such a short video and doesn't even contain more information like audio signals, but a lot of redundant video material. Thus, compression and video coding formats were implemented. They reduce information, either lossless and lossy, by removing redundancies in the same frame as well as in successive frames.

Intra-frame- and inter-frame-prediction are often used to reduce redundant information. Intra-frame prediction exploits the fact that adjacent pixels often carry similar information and can therefore be combined. Inter-frame prediction, on the other hand, attempts to detect similarities in blocks of previous frames and use them to describe pixels in the current frame.

While lossless compression preserves the quality by storing all data needed to reconstruct the original data after decompression, lossy compression alters the given data while offering a higher compression rate. The compression algorithm is called 'encoding' while the decompression algorithm is called 'decoding'. The conversion from one video coding format into another one is called 'transcoding'. A 'codec' is a solution that encodes and decodes media files and can either be realized as software or hardware.

We will now introduce two video coding formats that are also supported by the accelerator card used later and therefore considered for this thesis.

2.2.1 H.264

H.264, also known as **Advanced Video Coding** or **MPEG-4 Part 10**, is a state-of-the-art video compression standard. Even though it got first published in 2004, it is still the most common used format for video material, used by 83% of video developers as of 2021 [Bit21]. Thanks to its general-purpose design, it supports a wide range of bit rates and video resolutions. For different target scenarios such as video conferencing or high-resolution video sharing, three profiles were initially defined (Baseline, Main and Extended), specifying a certain subset of the available features to produce the most suitable video for each use case. Over the years, extensions with additional profiles and levels have been implemented and incorporated, with which resolutions up to 8K are also possible. To give an example of the available functions of different profiles, all of them offer Context-based Adaptive Variable Length Coding (CAVLC), but only certain profiles offer a flexible macroblock order (Baseline and Extended) or quantization scaling matrices (profiles of the 'High' family).

In addition, different levels are defined which give an indication of the required performance of the decoder and primarily limits the bit rate as well as the resolution and frame rate.

Table 2.1 gives an overview of a selection of certain levels, specifying the maximum image size in samples and the maximum number of samples per second. With these two data we can calculate a resolution with typical sizes and the corresponding highest frame rate, given in width x height @ fps.

H.264 is built on a layered structure consisting of two major parts: The Network Abstraction Layer (NAL) and the Video Coding Layer (VCL). Since H.264 is designed for many different applications like television broadcasts, internet streaming services or storage on optical or magnetic drives, the NAL was introduced. It acts as a protocol for the data created by the VCL, making it usable for file storage and transmission over IP, and future-proofs the codec by allowing easy addition of new applications.

Level	Max frame size (samples)	Max samples per second	Example for high resolution @ frame rate (fps)
3	414.720	10.368.000	720x576@25.0
3.1	921.600	27.648.000	1280x720@30.0
3.2	1.310.720	55.296.000	1280x1024@42.2
4	2.097.152	62.914.560	2048x1024@30.0
4.1	2.097.152	62.914.560	2048x1024@30.0
4.2	2.228.224	133.693.440	2048x1088@60.0
5	5.652.480	150.994.944	3680x1536@26.7
5.1	9.437.184	251.658.240	4096x2304@26.7
5.2	9.437.184	530.841.600	4096x2304@56.3
6	35.651.584	1.069.547.520	8192x4320@30.2
6.1	35.651.584	2.139.095.040	8192x4320@60.4
6.2	35.651.584	4.278.190.080	8192x4320@120.9

Table 2.1: Overview of a selection of H.264 levels based on [H.264] (Chapter A.3.4)

The VCL takes care of the encoding task. To encode a given video with the H.264 video encoder (Figure 2.2), each frame of the video is divided into macroblocks. These blocks have a base size of 16 x 16 and can be subdivided into smaller blocks, down to 4 x 4. For each macroblock, the encoder can chose between intra- and inter-prediction coding mode which makes prediction in the same frame (intra-prediction with 9 different modes) or previously encoded frames (inter-prediction), removing several redundancies. Based on these predictions, the blocks are divided into 4 x 4 or 8 x 8 blocks, transformed by an Integer Discrete Cosine Transform (DCT), and then quantized. Since H.264 uses block-based transformations for both intra- and inter-prediction coding, the final material could have block artifacts at the boundaries of each macroblock. To overcome this problem, a deblocking filter is used.

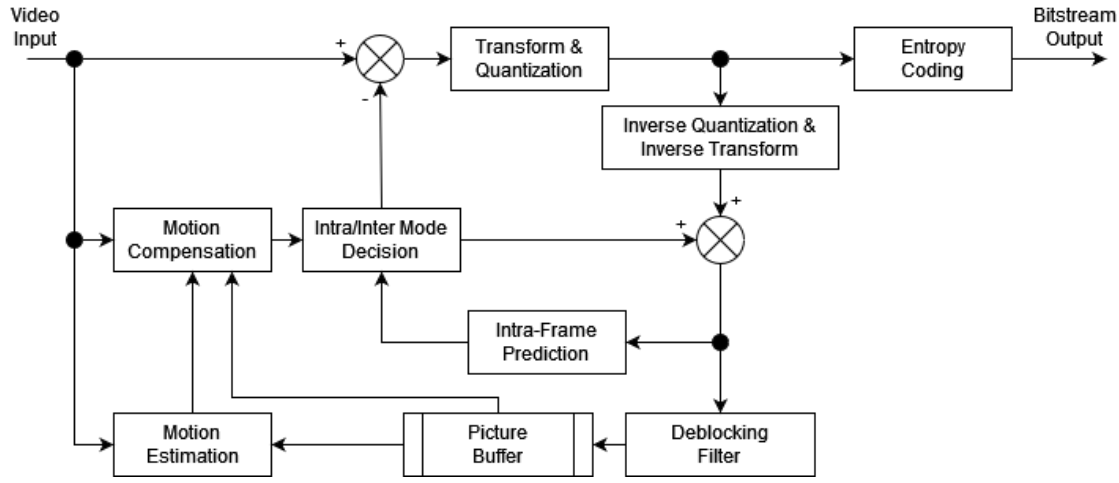

Figure 2.2: Block diagram of H.264 encoder, based on [KTR06]

Additionally, H.264 uses a entropy encoder based on fixed tables of variable length codes, similar to a Huffman code. Unlike the latter, however, H.264 uses a 'set of codewords based on the probability distributions of generic videos instead of exact Huffman code for the video sequences' [KTR06]. Depending on the chosen profile, we can use CAVLC or Context-adaptive Binary Arithmetic Coding (CABAC) for this.

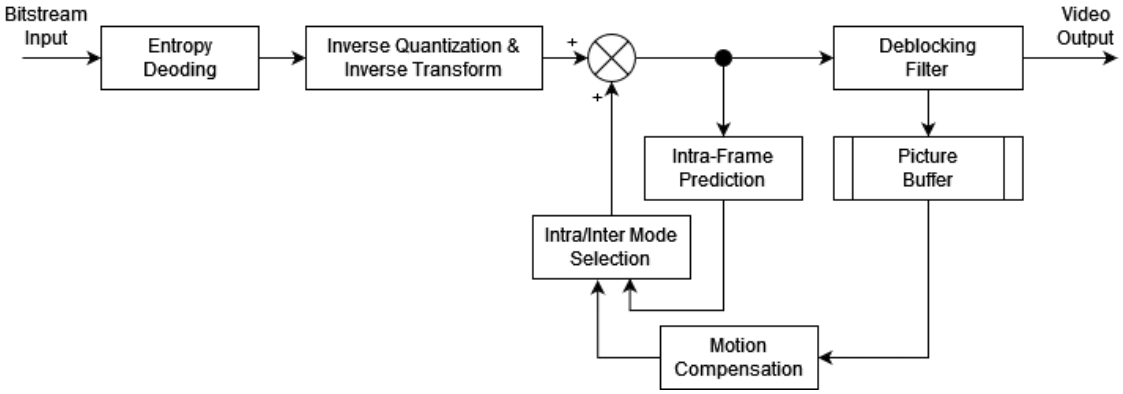

Figure 2.3: Block diagram of H.264 decoder, based on [KTR06]

Similar, the decoder (Figure 2.3) reverses the entropy encoding, applies inverse quantization and transformation and a inter-frame- and intra-frame-prediction to generate the video output from the stored bit stream.

2.2.2 H.265

H.265, also known as **High Efficiency Video Coding (HEVC)** or **MPEG-H Part 2**, is the successor of H.264. As demand for high-definition video grew steadily for both local media like Blu-ray discs and video streaming platforms like Netflix, it became clear that H.264 was a less effective solution for video files above 1920 x 1080 due to its 16 x 16 macroblocks. Therefore, a video compression standard based on H.264 was developed with better compression and higher supported resolutions.

Since it is based on H.264, it shares multiple parts with its predecessor. We still have the layered design with NAL and VCL and still use both, intra-frame- and inter-frame predictions to remove redundancies. While the coding process is very similar, H.264 splits the input videos into macroblocks of 16 x 16 size and H.265 is using a so called Coding Tree Unit with blocks of sizes 16 x 16, 32 x 32 or 64 x 64. Additionally, it uses Integer DCT and Discrete Sine Transform (DST) with variable block sizes between 4 x4 and 32 x 32. Especially for larger resolutions, the bigger block size increases the coding efficiency at the expense of higher computation complexity, since more pixels can be processed at a time. For intra-prediction mode, 33 different modes are available, 24 more than with H.264.

As with H.264, H.265 was originally released with multiple profiles and levels that got expanded over the years. While the main sample size and frame rate is similar to the ones of H.264, as seen as in Table 2.1, the maximum available bit rate differs. Two tiers (Main and High) were defined for each level beyond 4, with the Main tier as a general purpose solution and the High tier primarily for

demanding applications. Accordingly, the bit rates in the High tier are much higher than in the Main tier. For example, for the Main and Main 10 profiles at level 5, the Main tier is designed for 25 Mb/s and the High tier for 100 Mb/s.

Despite the advantages that H.265 offers over H.264, it was used by only 49% of all video developers in 2021 [Bit21]. One reason for this is the increased computational effort required to process the video files. Another important part is the licensing model around the H.265 standard. This leads to less decoders being deployed in various environments like mainstream web browsers.

2.3 FFmpeg

FFmpeg is an open-source software package consisting of various libraries and tools for de- and encoding multimedia data. The core parts of FFmpeg are the command line tool *ffmpeg*, *libavcodec* as a library to decode and encode various audio and video codecs and *libavformat* as a library with tools to mux and demux various container formats. For H.264, the open-source library x264 is used which implements the design of H.264. Similar, for H.265, the library x265 is used. They are both state-of-the-art codecs available for their corresponding compression standard.

2.3.1 ffmpeg, the command line tool

The general syntax of an ffmpeg command, according to the command line tool, is:

```
ffmpeg [options] [[infile options] -i infile]... [outfile options] outfile...
```

Infile and Outfile are used to define the input and output media file on the file system. For 'options', an overview of important flags and options used with ffmpeg and the H.264 and H.265 libraries is listed here:

- **-c** followed by a string is used to set the codec to be used. This can be done for both, audio and video signals, by specifying it in the call (-c:a for audio codec and -c:v for video codec)
- **-b** sets the target bit rate. Similar to flag 'c', it can be called for both audio and video streams separately by using -b:a and -b:v
- **-r** specifies the frame rate and can be used to change it to another value. Alternatively, `-fps` can be used in the context of the chosen video encoder
- **-s** similar to -r, it defines the frame size. Alternatively, `-vf scale=w,h` can be used in the context of the chosen video encoder as an filter operation
- **-profile** determines the used profile like 'Main' or 'High'
- **-level** sets the level and thus the maximum resolution and frame rate
- **-pix_fmt** specifies the color space used for the process

2.4 Opencast

Opencast offers a 'scalable video-capture, -management, and -distribution system for academic institutions' [Opencast-About] built on Java. It provides a complete video processing system based on the subsystems 'media ingestion', 'media processing' and 'media distribution' and offers multiple interfaces for users and administrators to communicate with the system, mainly an web interface and an API based on REST. In these interfaces, users can schedule recordings and even execute them directly in the web interface, provided that the appropriate hardware is connected to the system. Multiple streams are supported for each event, like a live recording of the presenter as well as a recording of the computer screen used by the presenter. After the recording, the video and metadata are packaged into 'Media Packages' and then used in the workflow system.

The core function of Opencast is the video processing system. Workflows determine exactly which available means are used to process a media. Multiple workflows can run in parallel, provided that enough resources are available on a node, and mapping of individual jobs to specific nodes is also possible. For example, video encoding could be outsourced to a machine with acceleration cards, while metadata processing could be handled on a general purpose server. Data for this processing system can be provided through REST APIs, the web interface or, if a suitable workflow is provided for a so-called 'inbox', by scanning the contents of a specific directory.

For the end user consuming the processed videos, several distribution channels as well as a built-in web player are available.

An Opencast setup may contain one or more nodes on different hardware, which also carry a different subset of the available functions. For example, there may be one node for the administration interface and multiple nodes for the workers. For a productive setup, at least two servers are recommended to reduce the load of the main system during a job by offloading processing workflows to a worker node.

According to Goyanes, González, Sanchez Bermudez, and Docampo, the 'Opencast core configures a complex workflow composed of 22 individual operations that can be grouped into [...] 9 functional the blocks' [GGSD17] seen in Figure 2.4.

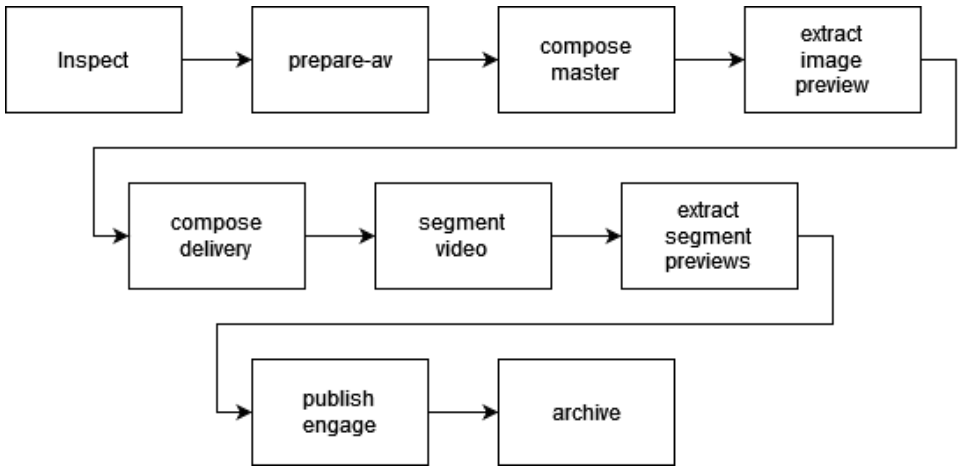

Figure 2.4: Standard Opencast Workflow, based on [GGSD17]

2.4.1 Workflows

Workflows in Opencast are written in Extensible Markup Language (XML) and define exactly which operations are performed in what order. The basic structure of a workflow can be seen in Listing 2.1. As part of the definition, the namespace to be used is 'http://workflow.opencastproject.org' to prevent name collisions with other XML documents.

```
<definition xmlns="http://workflow.opencastproject.org">
    <!-- Description of Workflow -->
    <id>example-workflow</id>
    <title>Example Workflow</title>
    <tags>
    </tags>
    <description>
        An example workflow to give an overview of the XML file used
    </description>

    <!-- Operations -->
    <operations>
        <operation>
            ...
        </operation>
        ...
    </operations>
</definition>
```

Listing 2.1: Structure of an Opencast workflow

Then, XML tags define the internal *id*, a *title*, optional *tags* and a *description*. The *id* is used by Opencast as a unique identifier for the workflow and tags can control where the defined workflow may be used. The *title* and *description* are only shown in the user interface.

The main part of a workflow are the operations used. An installation of Opencast brings many operations with it, like 'inspect' or 'encode'. Multiple configuration keys are available for each operation to allow developers to set up and control the operation. An example for the 'encode' operation can be seen in Listing 2.2. Each operation is using an *id* to denote the predefined operation control flow and the available configuration keys. After that the properties of the operation are set.

```
<operation
    id="encode">
    <configurations>
        <configuration key="source-flavor">*/prepared</configuration>
        <configuration key="target-flavor">*/delivery</configuration>
        <configuration key="target-tags">engage-download, engage-streaming</configuration>
        <configuration key="encoding-profile">sample-profile</configuration>
    </configurations>
</operation>
```

Listing 2.2: Structure of a workflow operation using the example of an 'encode' operation

2.4.2 Encoding Profiles

Unlike the workflows themselves which use XML, an encoding profile is just a set of key-value pairs of the following pattern, stored in a file with the extension `.properties`:

```
profile.<name>.<property> = <value>
```

For each profile exists a set of properties that should always be specified:

- **.name** is a description for the encoding profile
- **.input** defines the type of the source material
- **.output** defines the type of the target format and has the same available options as **.input**
- **.suffix** specifies the extension appended to the file
- **.mimetype** declares the media type of the file content
- **.ffmpeg.command** specifies the command line options for FFmpeg, which in the end control the options of the encoder

Examples of valid types for **.input** and **.output** include, but are not limited to, *audio*, *visual*, *stream* or *image*.

A media type is an identifier for files transmitted over the internet and is represented by a type and a subtype, separated by a '/'. For the allowed types named above for **.input** and **.output**, the relevant media types are audio, video or image, while for the subtype the container format is used. An example of this is *video/mp4*.

3 Transcoding on Software and Hardware

3.1 Transcoding with the CPU of a machine

Each encoding and decoding algorithm can be implemented in high level languages. The often used encoders, x264 and x265, are software implementations for H.264 and H.265. Depending on the profile, they offer high quality encodes at the expense of computational time.

The performance of a software transcode depends on different metrics. These metrics include, but are not limited to encoder settings, algorithm implementation, instructions per cycle, instruction set architecture, core frequency, single-core performance (based on frequency and instructions per cycle), core count (if the implementation uses multi-threading), transfer speed for storage and memory and the operating system.

3.2 Transcoding with accelerator hardware

Several ASICs are available for audio and video encoding. They allow the host system to offload processing from the CPU to a different chip, and thus freeing up resources for other purposes. Here we will discuss a selection of available and widely used ASICs from various vendors. They can be designed for encoding, decoding, or both.

3.2.1 Quick Sync Video (QSV)

Starting with 'Sandy Bridge' in 2011, Intel has integrated Intel QSV into the processor dies of the Intel Core series. It supports both encoding and decoding for various video codecs. H.264 can be used since 'Sandy Bridge' while H.265 is available since 'Skylake'.

Even though it is part of the processor, it is separate hardware and thus offloads encoding and decoding work, just like with additional acceleration hardware added to the system like GPUs. Intel processors add a hardware codec called MFX. A scheme for this can be seen in Figure 3.1.

MFX has hardware that can handle decoding entirely. For the video encoding process, Intel uses a hybrid software ('ENC') and hardware ('PAK') approach. 'ENC' uses a programmable execution unit array on the integrated GPU and takes care of rate control, motion estimation, intra estimation and mode decision. This allows for 'algorithm tuning and feature additions'. Then, 'PAK' uses the MFX hardware and handles motion compensation, intra-frame prediction, forward quantization, pixel reconstruction and entropy coding. To increase the throughput, both 'ENC' and 'PAC' can run on different frames at the same time [Jia11].

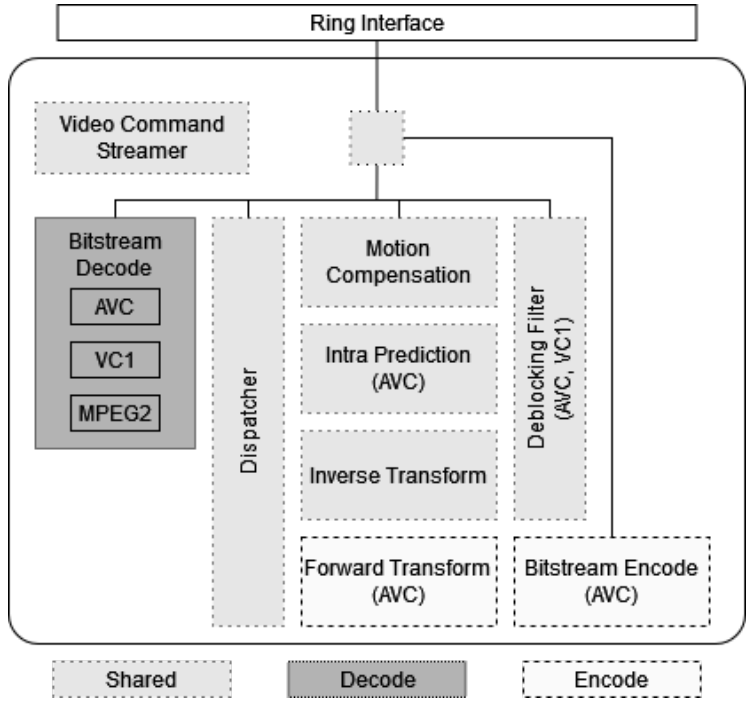

Figure 3.1: MFX in Intel Sandy Bridge, based on [Jia11]

3.2.2 Nvidia Encoder and Nvidia Decoder

Nvidia includes dedicated hardware for decoding (called NVDEC) and encoding (called NVENC) on their graphics cards. NVDEC is included since the 'Fermi' generation, released 2010, and supports a lot of different codecs used nowadays. In 2012, NVENC was included as part of their GPUs as well. Only H.264 was implemented initially, but H.265 encoding was supported in later iterations [PY16].

Since both NVDEC and NVENC are a dedicated parts of Nvidia's GPUs, the computational cores themselves are not used for encoding or decoding. This is very practical for live streamers. Since the recorded material is processed via NVENC, the GPU resources can be used by other active applications.

3.2.3 AMD Video Coding Engine (VCE) and Unified Video Decoder (UVD)

Similar to Nvidia, AMD included UVD ASICs since 2007 and VCE ASICs since 2011 in most of their GPUs, fully implementing the H.264 video codec and later on, H.265.

Since 2018, there is an official successor called 'Video Core Next', which combines the encoding and decoding functions of VCE and UVD in one ASIC on the GPU die, similar to Intel's QSV.

3.2.4 Advantages and disadvantages of using acceleration hardware

Patait and Young analyzed and presented the ratio between quality and performance in [PY16] for x264, QSV and NVENC. While quality is calculated and presented with Peak Signal-to-Noise Ratio (PSNR), which we will discuss further in Chapter 7, performance is measured in fps. It represents the number of frames that can be processed per second by the used solution.

As we can see in Figure 3.2, there is a slow and medium preset for each implementation. Going from slow to medium increases performance at the expense of quality. While the x264 software encoder delivers the highest quality, the performance is comparatively low. In contrast, Intel offers high performance with QSV, but lower quality with both presets. Nvidia's solution offers a good compromise and outperforms the other two solutions in terms of performance. The quality is lower than x264, but still comparable.

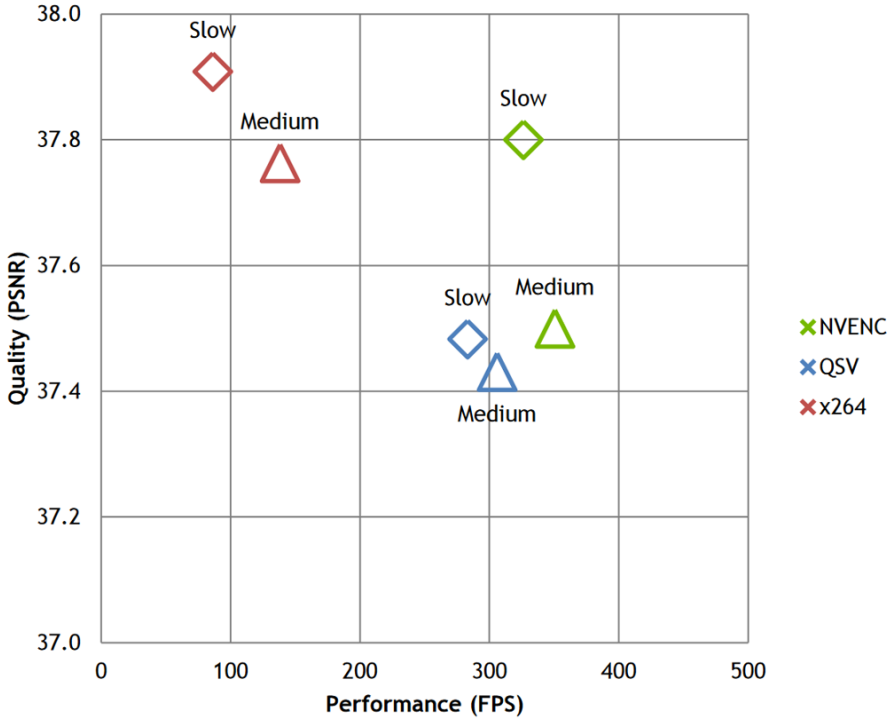

Figure 3.2: Quality vs Performance of NVENC, QSV and x264, from [PY16]

It is evident that the hardware solutions place more emphasis on performance than quality. Thus, a qualitative software encoder like x264 will deliver the best quality while compromising speed. In addition, hardware codecs usually don't implement all, but only a subset of the functionality of software codecs.

4 Related Work

There are several papers about efficient designs and implementations of the encoding process on FPGAs for both H.264 and H.265. While they have no direct relation to the integration of an FPGA-based accelerator card in Opencast, they deal with the codec implementation and performance benefits.

Atitallah et al. published a FPGA design for inter- and intra-prediction mode, an integer transform algorithm and a quantization algorithm based on H.264 that achieves performance good enough for real-time enoding [ALM11].

Kalali and Hamzaoglu implemented a H.265 intra prediction algorithm with High-Level Synthesis on a Xilinx FPGA. It can achieve 35 fps with a resolution of 1920x1080 [KH16]. *Azgin et al.* optimized an existing H.265 intra prediction algorithm on FPGA and achieved 55 fps at 1920x1080 in the worst case while reducing power consumption compared to a reference FPGA implementation, making it a viable solution 'in portable consumer electronics products that require a real-time HEVC encoder' [AMKH18].

Sjövall et al. presented 'a hardware-accelerated Kvazaar HEVC intra encoder for 4K real-time video coding at up to 120 fps' [SVV+18]. They achieved a speedup factor of 6.8 compared to the pure software implementation.

In addition to H.264 and H.265, other video compression methods and their implementations are discussed. The authors *Chen and Singh* performed an efficient real-time implementation of 'fractal video compression' with OpenCL and showed how it can be optimized for different platforms. They then ran the kernel on CPU, GPU and FPGA to get a comparison in terms of performance and 'demonstrate that the core computation implemented on the FPGA through OpenCL is 3× faster than a high-end GPU and 114× faster than a multi-core CPU, with significant power advantages' [CS13].

5 Considerations for the implementation and the used environment

5.1 Opencast workflows

To be able to compare results, we need to write several comparable encoding profiles and use them in similar workflows. Some of the profiles will only use the CPU of the machine while the others will use the FPGA acceleration card. The most important part of this workflow design will be finding and assembling the FFmpeg commands for the encoding profiles. We have to find commands for both H.264 and H.265.

While we can have a direct impact on the servers of self-hosted services, we cannot guarantee that every student will have access to high-speed Internet and powerful hardware. For this reason, it is also important that the final processed video files are available in several different resolutions to allow for smooth playback on all devices. A workflow should automatically transcode and scale a source video to different target resolutions. That material can then be used for adaptive streaming, which selects the appropriate resolution based on the users infrastructure.

5.2 Environment with FPGA accelerator card

FPGA acceleration cards for efficient video encoding can be implemented in suitable systems on-site or with cloud providers. We will use the latter for our environment. This gives us less control over the hardware, but allows us to quickly test the environment available with the cloud provider. For long-term integration into an existing server structure, on-site solutions are preferable, as it's independent of an external provider and the hardware is always accessible.

5.2.1 Amazon Web Services (AWS) Elastic Compute Cloud (EC2)

In 2002, Amazon began making the AWS platform available to the public. At that time, it was primarily intended as a 'platform for creating innovative web solutions and services designed specifically for developers and web site owners' [AWS02]. Over the next years, the company evolved from a web service provider to a cloud computing provider. In March 2006, Amazon S3 was made available, providing a 'simple storage service that offers software developers a highly scalable, reliable, and low-latency data storage infrastructure at very low costs' [AWS06]. In August of the same year, the limited beta of EC2 was released, providing customers with virtual servers for as little as 10 cents/hour [Bar06]. This provided a scalable and low-cost alternative to on-site hardware, which quickly attracted many interested customers. Just a year later, AWS introduced 'multiple

compute instance types for Amazon EC2 customers' that were 'up to eight times as powerful as those previously available'. Moreover, it was no longer a limited beta, but was made available to all developers that same year [AWS07].

Since then, EC2 has continued to evolve, providing users and developers worldwide with a variety of different instance types, so that the best possible service can be provided for any purpose. There are general purpose instances, compute optimized instances for applications that benefit from high performance processors, memory optimized instances for applications that process large amounts of data, accelerated computing instances that use hardware accelerators or co-processors to compute various functions more efficiently, and memory optimized instances for high read and write speeds to local storage [AWSEC2].

To evaluate the effectiveness of FPGA acceleration cards, we will use AWS EC2 instances of type 'VT1', which offers up to eight Xilinx Alveo U30 Data Center Accelerator Cards.

5.2.2 EC2 VT1 Instances

While instances of type 'VT1' are advertised for real-time transcoding for live streaming, we can also use them for Opencast.

Each instance is virtualized on a host with 2nd Generation Intel Xeon Scalable Processors (Cascade Lake P-8259CL). According to the specifications, a single CPU of this type offers 24 cores and 48 threads with a clock speed of 2.5 GHz and a turbo frequency of 3.5 GHz. As shown in Table 5.1, the number of virtual processor (vCPU) cores and number of accelerator cards available depends on instance size. Additionally, the available memory and the network bandwidth scales with size, but even the smallest instance offers 24 GiB of memory as well as 3.125 Gb/s network bandwidth. This ensures that the transcoding processes are not hindered by available memory or network bandwidth.

We will choose the smallest available instance type 'vt1.3xlarge', since one accelerator card is sufficient for our tests.

Instance Size	vt1.3xlarge	vt1.6xlarge	vt1.24xlarge
U30 Accelerator Cards	1	2	8
vCPU cores	12	24	96
Memory (GiB)	24	48	192
Network Bandwidth (Gb/s)	3.125	6.25	25
EBS Bandwidth (Gb/s)	Up to 4.75	4.75	19
1080p60 Streams	8	16	64
2160p60 Streams	2	4	16

Table 5.1: Features of VT1 instances

Xilinx Alveo U30 Data Center Accelerator Card

Instances of type 'VT1' contain one or more Alveo U30 accelerator cards. Each U30 offers a PCIe x8 interface, which is split into two sets of four lanes that allows for direct communication with the two 'XCU30' FPGAs on the cards, as seen in Figure 5.1.

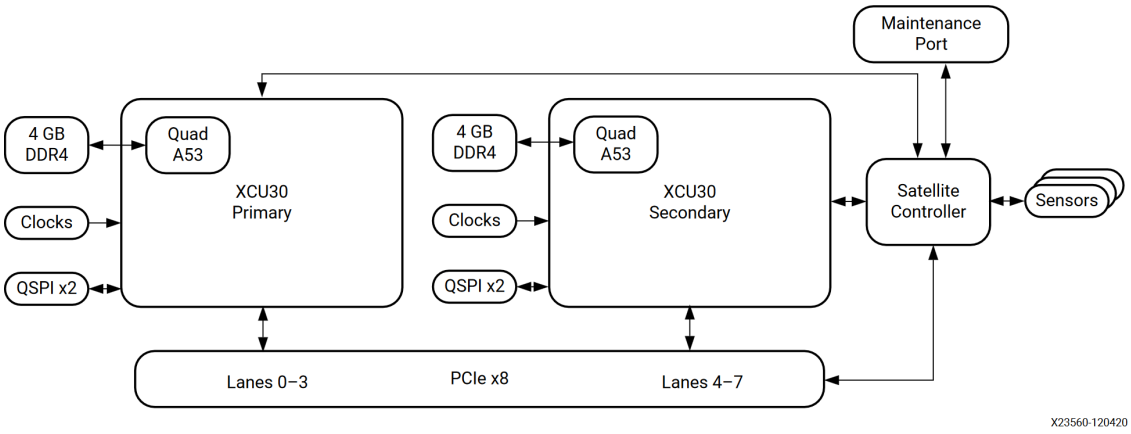

Figure 5.1: U30 Block Diagram from [DS970]

The FPGA 'XCU30', also known as 'XCZU7EV', is part of the 'Zynq UltraScale+ Multiprocessor System on a Chip (MPSoC): EV' family. According to the data sheet ([DS890] and [DS891]), each FPGA contains a Quad Core ARM Cortex-A53 as an Application Processing Unit and a Dual Core ARM Cortex-R5F as Real-Time Processing Unit, connected to 4 GiB of DDR4 memory. It offers 28800 CLBs. Each CLB contains 8 6-LUT and 16 flip-flops, 36 Kb block RAM, 288 Kb UltraRAM and DSP slices for faster arithmetic functions.

As part of the programmable logic, each XCU30 is equipped with a video encoder/decoder called Video Codec Unit (VCU), that can be addressed by both the ARM processors and the programmed CLBs. It offers simultaneous encoding and decoding through separate cores and supports H.264 up to 'High' profile level 5.2 as well as H.265 up to 'Main' or 'Main 10' profile level 5.1 (High tier). For the sample rates, it supports 4:2:0 and 4:2:2 chroma sampling.

Drivers and the manufacturer's toolkit are required to use the accelerator card. Xilinx offers the 'Xilinx Video SDK' which contains the necessary tools to interface with the card. They also provide a precompiled version of FFmpeg with the codecs needed to use the FPGA. The tools can be manually installed on an EC2 instance or an image provided by Xilinx called 'AMD-Xilinx Video SDK AMI for VT1 Instances' can be used. This image contains everything necessary to use the FPGA directly without further tinkering.

For licensing reasons, the included FFmpeg version does not include codecs and plugins that are available in the publicly distributed version. To use libraries such as x264 or x265, one can recompile FFmpeg with the Xilinx libraries.

5.3 Source material for the evaluation step

To evaluate the effectiveness of the solution, we are going to use a computer-animated short film with the title 'Big Buck Bunny'. This video file was made with Blender, a 'free and open source 3D creation suite' [Blender-About] and is publicly released under the Creative Commons Attribution 3.0 license. Although it is an animated short film and not a lecture recording, the choice of this source file offers the following advantages for this work:

Length

While lecture recordings are typically 90 minutes long, the short film is only 10 minutes long. This means that there are far fewer frames to compute, and we'll receive results faster. Since we are mainly interested in the ratio of the different conversion rates, it is still suitable for the comparison.

Multiple available resolutions

The selected video material is available in different resolutions and frame rates. This gives us the opportunity to collect more data by also making a comparison between different source resolutions transcoded to different target resolutions. An overview of relevant information about the video files can be found in Table 5.2. The bit rate only accounts for the video data, not the audio data.

We also have to differentiate between the original released files with 24 fps and the newer high definition releases with 30 or 60 fps. The latter differ not only in file size and profile, but also in the playback length. For the original release, the average and maximum bit rate matches since it uses constant bit rates, while the high definition releases use variable bit rates.

Width Px	**Height** Px	**Frame Rate** fps	**Avg. Bit rate** kb/s	**Max. Bit rate** Mb/s	**File Size** MiB	**Codec** Profile@Level
854	480	24	2.900	2.900	238	AVC Main@3
1280	720	24	5.147	5.147	397	AVC Main@3.1
1920	1080	24	9.283	9.283	692	AVC Main@4.1
1920	1080	30	3.000	16.7	263	AVC High@4.1
1920	1080	60	4.000	19.7	339	AVC High@4.2
3840	2160	30	7.500	37.8	604	AVC High@5.1
3840	2160	60	8.000	35.1	642	AVC High@5.1
4000	2250	60	10.000	46.1	793	AVC High@5.1

Table 5.2: Available 2D video files of Big Buck Bunny

In the rest of this document, we will name the video material in an abbreviated form corresponding to the pattern `<height>p` without the frame rate and `<height>p<fps>` with the frame rate. The **p** in the abbreviation stands for the 'progressive' scan mode, in which the images are rendered fully.

Width	Height	Short Form
3840	2160	2160p
1920	1080	1080p
1280	720	720p
854	480	480p
852	480	480p

Table 5.3: Short form of video resolutions and frame rates

In the last rows of Table 5.3, we use 480p for both 854x480 and 852x480 resolutions. While the source material comes with a width of 854, Section 6.1.3 mentions a filter requiring a height and width divisible by four, so 480p is used for both resolutions.

6 Implementation

6.1 Encoding Profiles

6.1.1 FFmpeg

We specify different encoding profiles for a selection of target resolutions. Each has a corresponding FFmpeg command. This section is intended to give an overview of the various encoding profiles.

To evaluate hardware and software implementations of H.264 and H.265, we have to use four different but comparable FFmpeg commands for each target resolution.

```
ffmpeg -i <infile> -c:a copy \
    -c:v (libx264|libx265|mpsoc_vcu_h264|mpsoc_vcu_hevc) \
    -profile:v (high|main) -level (40|51) -pix_fmt (yuv420p|nv12) \
    -s:v (854x480|1280x720|1920x1080|3840x2160) \
    -b:v (1500|2500|4000|7000)k <outfile>
```

Listing 6.1: Pattern for FFmpeg commands

As Listing 6.1 shows, variables change depending on the target resolution and codec used.

- **-c:a copy** The embedded audio stream is copied directly without conversion to minimize the impact during evaluation. If desired, one could also change the quality and size of the audio stream by specifying a codec and quality settings.
- **-c:v (libx264|libx265|mpsoc_vcu_h264|mpsoc_vcu_hevc)**. This specifies the codec used for conversion. `libx264` and `libx265` are software codecs while `mpsoc_vcu_h264` and `mpsoc_vcu_hevc` are the hardware codecs.
- **-profile:v (high|main)** For H.264 we use the 'High' profile and for H.265 the 'Main' profile.
- **-level (40|51)** sets the level of the codec. While we could omit this value and let FFmpeg select a fitting level, we set it to specify a consistent level. Target files up to 1080p30 are covered by level 4, while resolutions over 1080p30 and up to 2160p30 require at least level 5.1.
- **-pix_fmt (yuv420p|nv12)** sets the pixel format. `yuv420p` is for software codecs, while `nv12` is for hardware accelerators. Both are based on YC_BC_R and use chroma subsampling with a ratio of 4:2:0.
- **-s:v (854x480|1280x720|1920x1080|3840x2160)** specifies the target resolution.
- **-b:v (1500|2500|4000|7000)k** sets the target bit rate based on the resolution, respectively.

To achieve a specific target quality or file size, one would use Constant Rate Factor (CRF) or Variable Bit Rate (VBR) with an n-pass encoding. CRF is not supported by the VCU on the FPGA and VBR would require multiple passes for better results, greatly increasing the encoding time.

Therefore, Constant Bit Rate (CBR) is used for the evaluation. The encoder finds a quality for each frame with which the specified bit rate can be achieved. This does, however, provide a disadvantage. There can be fluctuations in quality depending on the complexity of the current frame. Since we are primarily interested in a direct performance comparison between the different codecs at comparable settings, this problem is negligible and can be addressed in further work.

6.1.2 Naming scheme

As stated in Section 2.4.2, each profile is a set of key-value pairs of the pattern `profile.<name>.<property> = <value>`. To distinguish between our profiles, we define `<standard>-<type>-<quality>` as the naming scheme for `<name>` with the options listed in Table 6.1.

Component	Options	Description
<standard>	h264, h265	video compression standard (H.264 / H.265)
<type>	cpu, fpga	software or hardware implementation
<quality>	480p, 720p, 1080p, 2160p, multlow, multhigh	target resolution or multi-transcoding

Table 6.1: Options defined for our naming scheme of encoding profiles

6.1.3 Encoding profile implementation

The following listings are examples of encoding profiles for single transcodes. Each example uses a different codec and transcodes to a resolution of 1080p. The information provided in Section 6.1.1 is used for specifying these examples and all other profiles.

```
profile.h264-cpu-1080p.name = h264 1080p encoding on cpu
profile.h264-cpu-1080p.input = visual
profile.h264-cpu-1080p.output = visual
profile.h264-cpu-1080p.suffix = -x264-1080p.mp4
profile.h264-cpu-1080p.mimetype = video/mp4
profile.h264-cpu-1080p.ffmpeg.command = -i #{in.video.path} -c:a copy \
  -c:v libx264 -profile:v high -level 40 -pix_fmt yuv420p -s:v 1920x1080 \
  -b:v 4000k #{out.dir}/#{out.name}#{out.suffix}
```

Listing 6.2: Encoding profile for 1080p with libx264

```
profile.h265-cpu-1080p.name = h265 1080p encoding on cpu
profile.h265-cpu-1080p.input = visual
profile.h265-cpu-1080p.output = visual
profile.h265-cpu-1080p.suffix = -x265-1080p.mp4
profile.h265-cpu-1080p.mimetype = video/mp4
profile.h265-cpu-1080p.ffmpeg.command = -i #{in.video.path} -c:a copy \
  -c:v libx265 -profile:v main -level 40 -pix_fmt yuv420p -s:v 1920x1080 \
  -b:v 4000k #{out.dir}/#{out.name}#{out.suffix}
```

Listing 6.3: Encoding profile for 1080p with libx265

```
profile.h264-fpga-1080p.name = h264 1080p encoding on fpga
profile.h264-fpga-1080p.input = visual
profile.h264-fpga-1080p.output = visual
profile.h264-fpga-1080p.suffix = -vcu-h264-1080p.mp4
profile.h264-fpga-1080p.mimetype = video/mp4
profile.h264-fpga-1080p.ffmpeg.command = -i #{in.video.path} -c:a copy \
  -c:v mpsoc_vcu_h264 -profile:v high -level 40 -pix_fmt nv12 -s:v 1920x1080 \
  -b:v 4000k #{out.dir}/#{out.name}#{out.suffix}
```

Listing 6.4: Encoding profile for 1080p with mpsoc_vcu_h264

```
profile.h265-fpga-1080p.name = h265 1080p encoding on fpga
profile.h265-fpga-1080p.input = visual
profile.h265-fpga-1080p.output = visual
profile.h265-fpga-1080p.suffix = -vcu-h265-1080p.mp4
profile.h265-fpga-1080p.mimetype = video/mp4
profile.h265-fpga-1080p.ffmpeg.command = -i #{in.video.path} -c:a copy \
  -c:v mpsoc_vcu_hevc -profile:v main -level 40 -pix_fmt nv12 -s:v 1920x1080 \
  -b:v 4000k #{out.dir}/#{out.name}#{out.suffix}
```

Listing 6.5: Encoding profile for 1080p with mpsoc_vcu_hevc

In addition to encoding to a single output file, FFmpeg allows the creation of multiple output files with different options. Listing 6.6 provides an example.

```
profile.h264-cpu-multlow.name = multiple h264 encodings to 1080p, 720p and 480p on cpu
profile.h264-cpu-multlow.input = visual
profile.h264-cpu-multlow.output = visual
profile.h264-cpu-multlow.suffix.1080p = -1080p.mp4
profile.h264-cpu-multlow.suffix.720p = -720p.mp4
profile.h264-cpu-multlow.suffix.480p = -480p.mp4
profile.h264-cpu-multlow.mimetype = video/mp4
profile.h264-cpu-multlow.ffmpeg.command = -i #{in.video.path} \
  -c:a copy -c:v libx264 -profile:v high -level 40 -pix_fmt yuv420p -s:v 1920x1080 \
    -b:v 4000k #{out.dir}/#{out.name}#{out.suffix.1080p}  \
  -c:a copy -c:v libx264 -profile:v high -level 40 -pix_fmt yuv420p -s:v 1280x720 \
    -b:v 2500k #{out.dir}/#{out.name}#{out.suffix.720p} \
  -c:a copy -c:v libx264 -profile:v high -level 40 -pix_fmt yuv420p -s:v 854x480 \
    -b:v 1500k #{out.dir}/#{out.name}#{out.suffix.480p}
```

Listing 6.6: Encoding profile for multi-transcoding to 1080p, 720p and 480p with libx264

With the U30 acceleration card, we also have the option of simultaneous scaling to different output resolutions using a multiscale filter. However, as shown in Table 7.3, the benchmark tests demonstrated that there is no noticeable difference for total time (rtime). Listing 6.7 shows an example profile that uses the multiscale filter. It specifies multiscale filtering from H.264-encoded material to H.264-encoded material with target resolutions of 1080p, 720p, and 480p. As mentioned earlier, a width of 852 pixels is used for 480p to make the pixel length divisible by 4.

```
profile.h264-fpga-multlow.ffmpeg.command = -c:v mpsoc_vcu_h264 -i <infile> \
  -filter_complex "multiscale_xma=outputs=3: \
  out_1_width=1920: out_1_height=1080: out_1_rate=full: \
  out_2_width=1280: out_2_height=720: out_2_rate=full: \
  out_3_width=852:  out_3_height=480: out_3_rate=full [a][b][c]" \
  -map "[a]" -c:a copy -c:v mpsoc_vcu_h264 -profile:v high -level 40 \
    -b:v 4000k -f mp4 #{out.dir}/#{out.name}#{out.suffix.1080p} \
  -map "[b]" -c:a copy -c:v mpsoc_vcu_h264 -profile:v high -level 40 \
    -b:v 2500k -f mp4 #{out.dir}/#{out.name}#{out.suffix.720p} \
  -map "[c]" -c:a copy -c:v mpsoc_vcu_h264 -profile:v high -level 40 \
    -b:v 1500k -f mp4 #{out.dir}/#{out.name}#{out.suffix.480p}
```

Listing 6.7: FFmpeg command for multi-transcoding to 1080p, 720p and 480p with a multiscale filter

6.2 Workflows

For our implementation, we modified the fast testing workflow shipped with Opencast to fit our needs. We add a configuration panel that allows the user to specify the target resolution and codec for encoding in the web interface. For each radio button field, we specify a name (*grpRes* and *grpEnc*) to limit the user to a single choice in each group.

```
<configuration_panel>
    <![CDATA[
      <div id="workflow-configuration">
        <fieldset>
          <legend>Video Quality</legend>
          <ul>
            <li>
              <input id="flagRes480p" name="grpRes"
                type="radio" class="configField" value="true" />
              <label for="flagRes480p">480p</label>
            </li>
            <li>
              <input id="flagRes720p" name="grpRes"
                type="radio" class="configField" value="true" />
              <label for="flagRes720p">720p</label>
            </li>
            <li>
              <input id="flagRes1080p" name="grpRes"
                type="radio" class="configField" value="true" checked="checked" />
              <label for="flagRes1080p">1080p</label>
            </li>
```

```
          <li>
            <input id="flagRes2160p" name="grpRes"
              type="radio" class="configField" value="true" />
            <label for="flagRes2160p">2160p</label>
          </li>
          <li>
            <input id="flagResMultLow" name="grpRes"
              type="radio" class="configField" value="true" />
            <label for="flagResMult">Multi-Transcoding to 1080p, 720p and 480p</label>
          </li>
          <li>
            <input id="flagResMultHigh" name="grpRes"
              type="radio" class="configField" value="true" />
            <label for="flagResMultHigh">Multi-Transcoding to 2160p and 1080p</label>
          </li>
        </ul>
      </fieldset>
      <fieldset>
        <legend>Encoder</legend>
        <ul>
          <li>
            <input id="flagEncH264-cpu" name="grpEnc"
              type="radio" class="configField" value="true" checked="checked" />
            <label for="flagEncH264-sw">libx264</label>
          </li>
          <li>
            <input id="flagEncH265-cpu" name="grpEnc"
              type="radio" class="configField" value="true" />
            <label for="flagEncH265-sw">libx265</label>
          </li>
          <li>
            <input id="flagEncH264-fpga" name="grpEnc"
              type="radio" class="configField" value="true" />
            <label for="flagEncH264-fpga">mpsoc_vcu_h264</label>
          </li>
          <li>
            <input id="flagEncH265-fpga" name="grpEnc"
              type="radio" class="configField" value="true" />
            <label for="flagEncH265-fpga">mpsoc_vcu_hevc</label>
          </li>
        </ul>
      </fieldset>
    </div>
  ]]>
</configuration_panel>
```

Listing 6.8: Configuration Panel Implementation

The implemented configuration panel visible in the web interface is presented in Figure 6.1. Before the workflow is processed, the corresponding values can also be found in the summary (Figure 6.2).

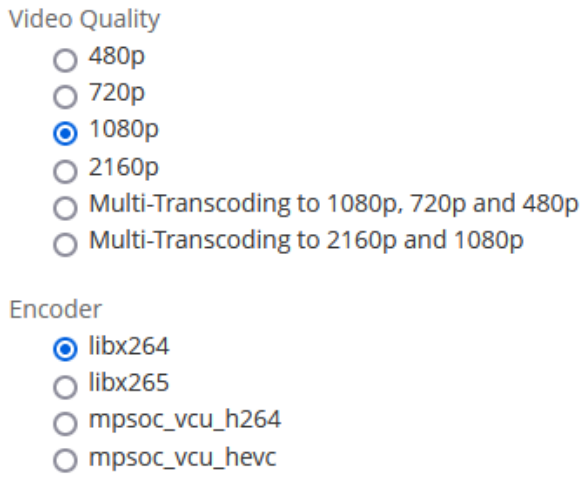

Figure 6.1: Web Interface Configuration Panel

Processing	
Workflow	Thesis Main Test Workflow
straightToPublishing	true
flagRes480p	false
flagRes720p	false
flagRes1080p	true
flagRes2160p	false
flagResMultLow	false
flagResMultHigh	false
flagEncH264-cpu	true
flagEncH265-cpu	false
flagEncH264-fpga	false
flagEncH265-fpga	false

Figure 6.2: Web Interface Flags

To avoid problems when using other interfaces than the web interface, like the REST API, default values are set for each option.

```
<operation
  id="defaults"
  description="Applying default values">
  <configurations>
    <configuration key="flagRes480p">false</configuration>
    <configuration key="flagRes720p">false</configuration>
    <configuration key="flagRes1080p">true</configuration>
    <configuration key="flagRes2160p">false</configuration>
    <configuration key="flagResMultLow">false</configuration>
    <configuration key="flagResMultHigh">false</configuration>
    <configuration key="flagEncH264-cpu">true</configuration>
    <configuration key="flagEncH265-cpu">false</configuration>
    <configuration key="flagEncH264-fpga">false</configuration>
    <configuration key="flagEncH265-fpga">false</configuration>
  </configurations>
</operation>
```

Listing 6.9: Specify default values for user variables

For the encode operations, we use a conditional to control which encoding profile is used. Since there are six quality and four codec options, we need to specify a total of 24 encode operations to cover all cases.

```
<operation
    id="encode"
    fail-on-error="true"
    if="${flagRes1080p} AND ${flagEncH264-cpu}"
    exception-handler-workflow="partial-error"
    description="Encoding videos to 1080p mp4 with H.264 (CPU)">
    <configurations>
        <configuration key="source-flavor">*/source</configuration>
        <configuration key="target-flavor">*/preview</configuration>
        <configuration key="target-tags">engage-download,engage-streaming</configuration>
        <configuration key="encoding-profile">h264-cpu-1080p</configuration>
    </configurations>
</operation>
```

Listing 6.10: Example for encoding operations with conditions

In the workflow operations overview of the web interface, all encoding operations and the selected profile are visible. Figure 6.3 shows an example of this, limited to the 720p and 1080p qualities. 'Skipped' is displayed next to the encoding operation if the conditional evaluates to false. Clicking on 'Details' displays the start and end time.

Skipped	encode	Encoding videos to 720p mp4 with H.264 (CPU)	Details ›
Skipped	encode	Encoding videos to 720p mp4 with H.265 (CPU)	Details ›
Skipped	encode	Encoding videos to 720p mp4 with H.264 (FPGA)	Details ›
Skipped	encode	Encoding videos to 720p mp4 with H.265 (FPGA)	Details ›
Succeeded	encode	Encoding videos to 1080p mp4 with H.264 (CPU)	Details ›
Skipped	encode	Encoding videos to 1080p mp4 with H.265 (CPU)	Details ›
Skipped	encode	Encoding videos to 1080p mp4 with H.264 (FPGA)	Details ›
Skipped	encode	Encoding videos to 1080p mp4 with H.265 (FPGA)	Details ›

Figure 6.3: Sample of the workflow operations overview in the web interface

7 Evaluation

7.1 Evaluation criteria

Since the core of an Opencast workflow is the encoding process and thus the FFmpeg commands used, we will execute the commands based on the pattern mentioned in Section 6.1.1 and compare the results. We also add the `-benchmark` ffmpeg flag to retrieve information about the time and resources used. The rest of the workflow also has a cost, but should take the same amount of time for both software and hardware solutions and thus has no major effect on the comparisons.

For our evaluation, we will consider the following criteria to compare the software and hardware solutions:

- **Computation time.** How fast are the video files transcoded?
- **Quality.** How does the quality compare with the same settings?
- **File size.** How big are the processed files in comparison?

The raw data for each criterion can be found in Section 7.5. The following evaluations have been created with it.

7.2 Selection of source material

For a meaningful comparison, both resolution and number of frames in the processed video should match the source video in order to remove other factors from the result. We perform the transcoding operations in Table 7.1 for each of the following cases: Software H.264, accelerator card H.264, software H.265 and accelerator card H.265. To ensure a certain output quality, we only consider input videos that are at least 1080p. This is representative of the 1080p30 lecture recordings at the University of Stuttgart. In addition, all transcodes will output the same frame rate as the input. We will only create files for which there is source material with the same target resolution and frame rate.

Since metrics can make frame-to-frame comparisons, we have to pay attention to the files with 24 fps and the files with 30 fps separately, since metrics can make frame-to-frame comparisons.

Optimally, uncompressed reference material would be used for quality comparisons to get an unadulterated measurement. This is not possible in this case, but since lecture recordings are already compressed, the source videos are more representative of the use case we're interested in. For the evaluation, we will use the video files specified in Table 5.2 as reference material.

Source Resolution	Target Resolution	FPS
1920x1080	854x480	24
1920x1080	1280x720	24
1920x1080	1920x1080	24
3840x2160	1920x1080	30
3840x2160	3840x2160	30

Table 7.1: Selected source material and target resolutions for successive evaluation.

7.3 Interpretation of Data

7.3.1 Time measurements

The first evaluation criterion is the **total time used** (rtime) to transcode one input file to one or multiple output files.

Single transcodes

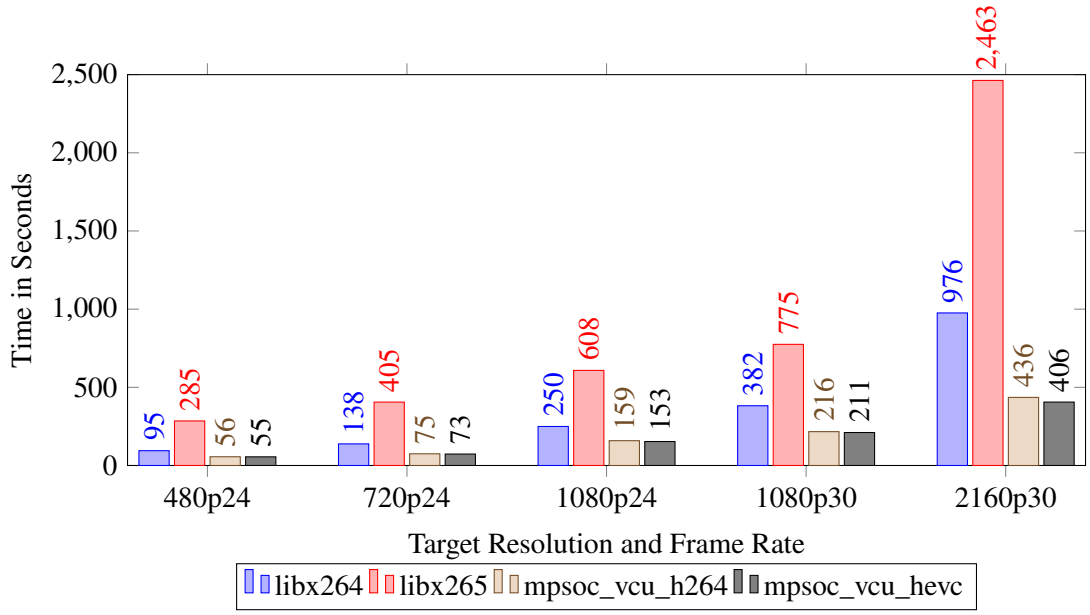

Figure 7.1: Time measurements for transcoding processes based on Table 7.2

Using the data shown in Figure 7.1, the effects of target resolution and frame rate on computation time are visible. The larger the target resolution, the greater the time required. The same is true for frame rate. The results show obvious differences between software and hardware codecs for the chosen specifications. The FPGA takes on average 45% less time than x264 and 78% less time than x265. In particular, the high computational requirements of software-implemented H.265 magnify this difference immensely.

Both H.264 and H.265 show similar processing times on the FPGA, with H.265 finishing slightly faster. In contrast, the software implementation of H.265 takes much longer than the software implementation of H.264. On average, the calculation time increases by 158% when comparing x264 to x265.

Using the total time, another metric can be calculated to show the performance of the codecs. If we divide the number of frames of a source video by the processing time, we get a measurement for the performance in fps. As presented in Figure 7.2, the performance for the hardware codecs is very close. They also have the highest performance, averaging in at 86% faster compared to x264 and at 380% faster compared to x265. For the software codecs, x264 operates twice to three times as fast as x265. On average, we gain a performance boost of 158% by using x264 instead of x265. In the opposite direction, we lose an average of 60% performance with the use of x265 compared to x264.

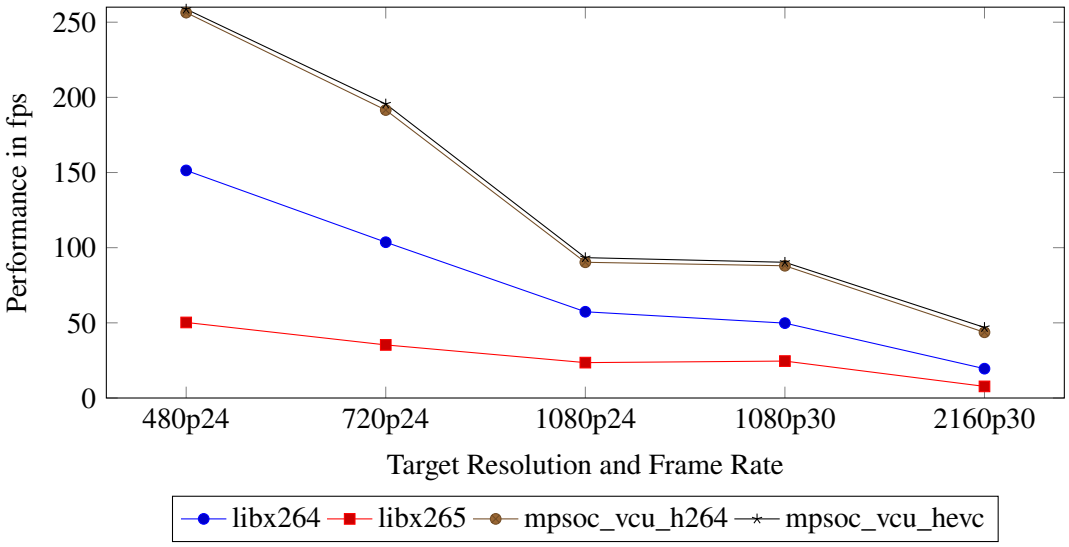

Figure 7.2: Performance measurements for transcoding processes based on Table 7.2

Multi-transcodes

Looking at the numbers for multiple transcoding, we see a similar effect to the single transcoding discussed previously. The codecs implemented via FPGA achieve the fastest result, while the software codecs are slower. Again, x265 is far slower than x264.

Figures 7.3 and 7.4 compare three different times. First, we take the individual times from Figure 7.1 and sum them up. This gives us a reference value that can be compared to the multi-transcoding solutions presented in Section 6.1.3. Then we ran the appropriate FFmpeg commands as in Listing 6.6 to generate multiple outputs from one input file. Lastly, the integrated multiscale filter was applied for the FPGA to obtain a comparison between the two multi-transcoding methods.

It is clear in both figures that the time for a multi-transcode roughly equals the sum of the individual transcoding times. We save some time because the source material only has to be read and decoded once. Both solutions, the direct command and the use of the multiscale filter, take about the same time, with the filter being slightly slower.

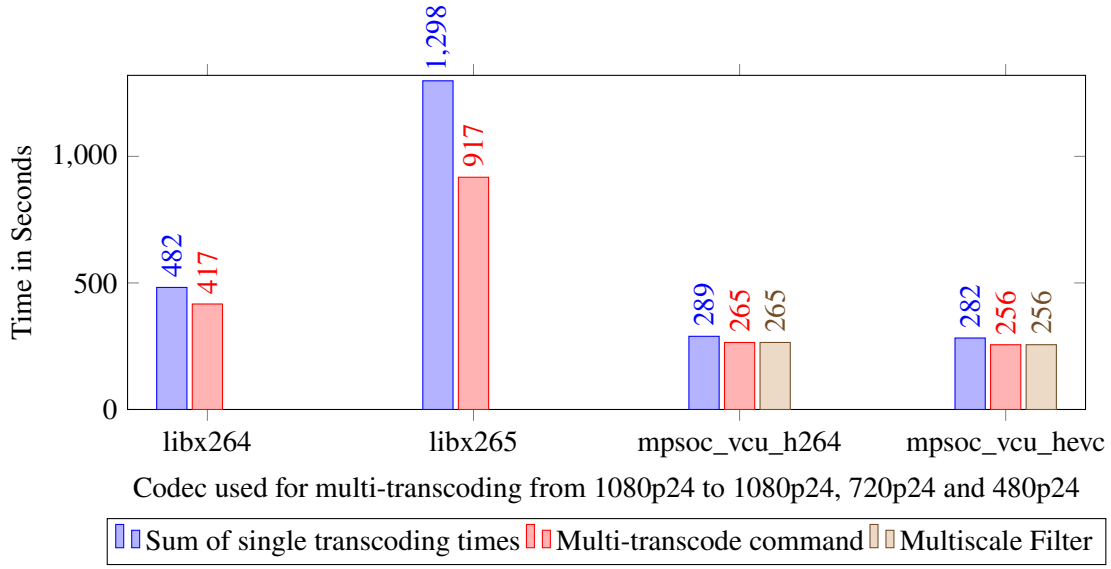

Figure 7.3: Time measurements for multi-transcoding processes based on Table 7.3 for 1080p24 source material transcoded to 1080p24, 720p24 and 480p24

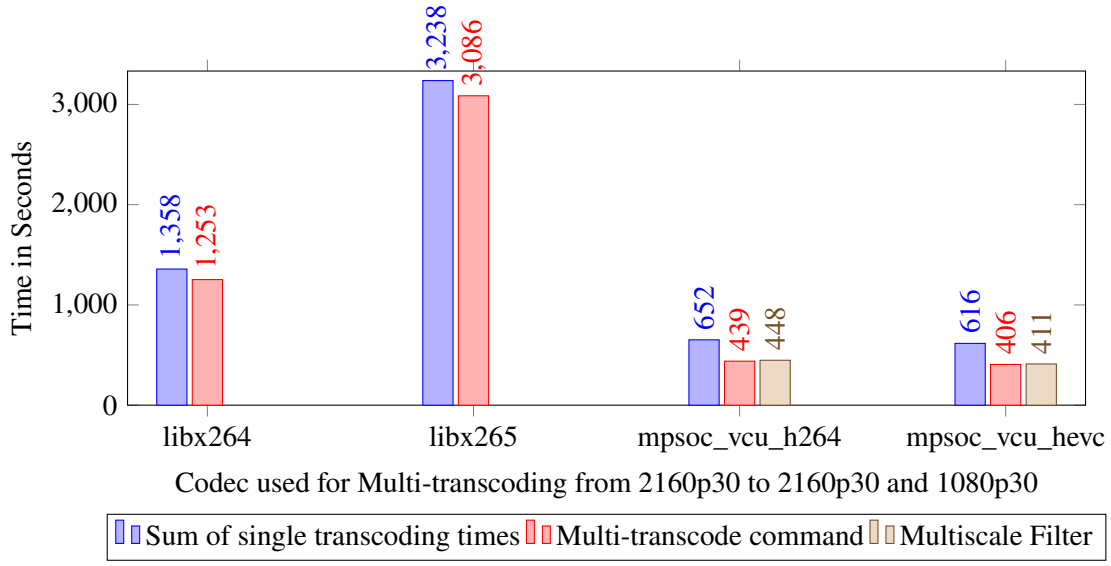

Figure 7.4: Time measurements for multi-transcoding processes based on Table 7.3 for 2160p30 source material transcoded to 2160p30 and 1080p30

7.3.2 Quality metrics

Several metrics have been created to evaluate the quality of a video compared to a given reference video. Even if these metrics result in an objectively calculated value, they do not necessarily correspond to subjective perception, which is why one should not rely on a single algorithm alone. Here, we will introduce three metrics and then use them to evaluate the quality of the different codecs.

Peak Signal-to-Noise Ratio

PSNR calculates the ratio between a maximum signal and unwanted noise that affects the quality. Intended for still pictures, it can be expanded to videos by calculating the PSNR for each frame and then averaging them.

The core of PSNR is Mean Square Error (MSE). For a frame of the source video S and the reference video R with the same size (Width W and Height H), MSE is calculated by summing the squared difference of the values of each pixel and then averaging the results.

$$\text{MSE} = \frac{1}{W*H} * \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} [S(x,y) - R(x,y)]^2$$

When handling multiple color channels, MSE is calculated for each channel individually, summed up and divided by the number of channels. With MSE and the maximum possible value of a pixel MAX_I, we can calculate the PSNR (in dB) of a frame with the following formula:

$$\text{PSNR} = 10 * \log_{10}(\frac{\text{MAX}_\text{I}^2}{\text{MSE}})$$

Structural Similarity Index Measure

Unlike PSNR, which is based on the MSE between two pictures to give a measurement, Structural Similarity Index Measure (SSIM) predicts the perceived visual quality of a picture using luminance, contrast and structure. It's the product of the three components:

$$\text{SSIM}(S,R) = l(S,R) * c(S,R) * s(S,R)$$

The components are defined with the following:

$$l(S,R) = \frac{2\mu_S\mu_R+C_1}{\mu_S^2+\mu_R^2+C_1},\ c(S,R) = \frac{2\sigma_S\sigma_R+C_2}{\sigma_S^2+\sigma_R^2+C_2},\ s(S,R) = \frac{\sigma_{RS}+C_3}{\sigma_S\sigma_R+C_3}$$

μ_x is the average value and σ_x is the standard derivation of an input image x. σ_{xy} is the covariance between input images x and y. The positive constants C_1, C_2 and C_3 are used to avoid null denominators [HZ10].

Similar to PSNR, it is designed for still images and we can expand it to videos in the same manner, by calculating the average SSIM value of all frames. The result of SSIM is a normalized value between 0.0 and 1.0, but it is also possible to output a value in dB.

Video Multi-Method Assessment Fusion

Video Multi-Method Assessment Fusion (VMAF) is a metric developed by Netflix in cooperation with various universities based on a machine learning model. It is intended to provide a subjective video quality perception 'focused on quality degradation due to compression and rescaling. VMAF estimates the perceived quality score by computing scores from multiple quality assessment algorithms and fusing them using a support vector machine' [Ras17]. As stated by Rassool in the same article, it currently uses 'three image fidelity metrics and one temporal signal' to compute a meaningful evaluation:

- Detail Loss Measure
- Visual Information Fidelity
- Mean Co-Located Pixel Difference
- Anti-noise Signal-to-Noise Ratio

The calculated VMAF score indicates the quality of the video compared to the reference on a scale from 0 to 100, where 100 means the best possible quality.

Data collection

To apply the quality metrics and collect the data, FFmpeg is used again. Instead of specifying a video codec, we pass the comparison and reference video to the appropriate filter, which calculates the metric for us. Since we are only interested in the result in this step and not in an output file, we also don't let FFmpeg generate a file.

We use the following pattern for FFmpeg:

```
ffmpeg -i <comparison_file> -i <reference_file> \
       -lavfi (psnr|ssim|libvmaf) -f null /dev/null
```

Listing 7.1: Pattern for FFmpeg evaluation command

While the filters for PSNR and SSIM are part of the FFmpeg package available on most distributions, VMAF must first be installed on the system and FFmpeg must be built with support for libvmaf.

Furthermore, there was a discrepancy in the 480p24 source material. The video file should have a resolution of 854x480 pixels. The MediaInfo for the file contains these properties:

```
Width: 853 pixels
Original width: 854 pixels
```

For comparison purposes, the video footage was scaled to 854x480 again and re-encoded with libx264 and a CRF of 0, the highest possible target quality. It should therefore be noted that a distorted result may be obtained for 480p.

Results

First, we will look at the results of PSNR. From the Figure 7.5, it can be concluded that the best result is achieved with software codecs, with x265 performing better than x264 at the same fixed bit rate. H.265 also achieves a better result than H.264 in the FPGA implementation, although the quality for both FPGA codecs is significantly worse according to the given numbers.

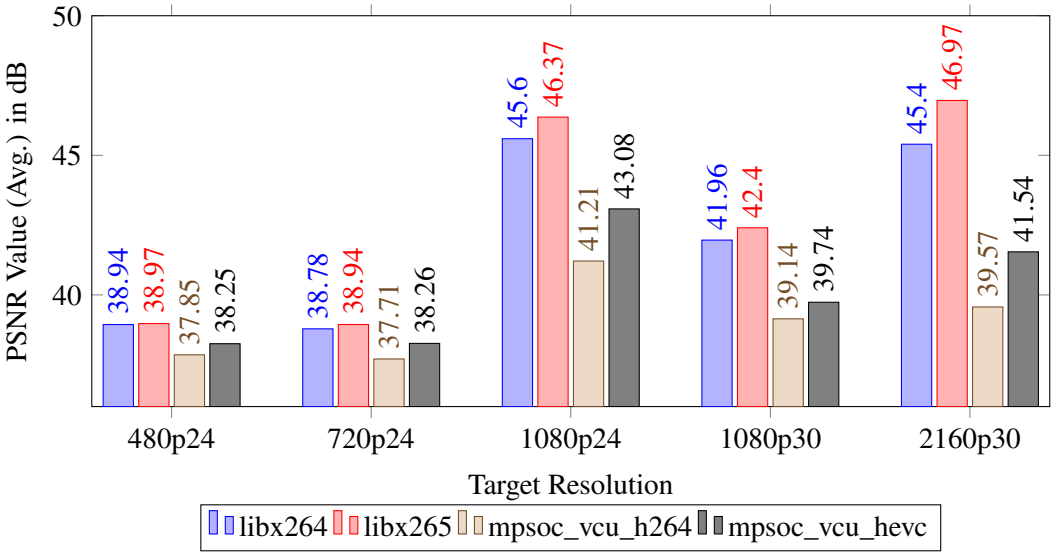

Figure 7.5: PSNR quality measurement based on Table 7.4

Another interesting factor results from the scaling. Video material that has been transcoded to the same resolution achieves the best quality measurements, whereas downscaled videos perform comparatively worse when evaluated against the corresponding lower resolution source material.

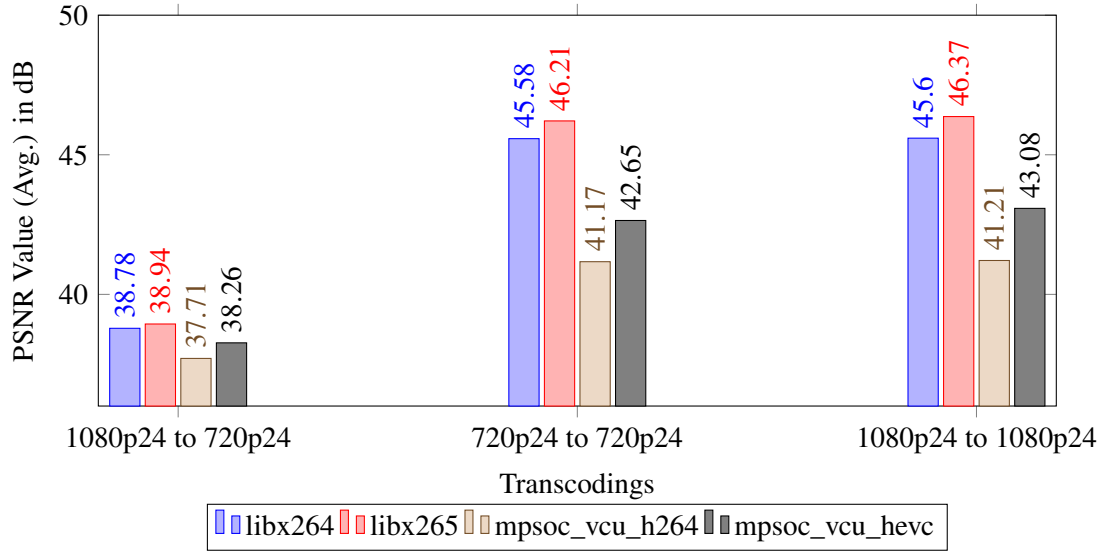

Figure 7.6: PSNR quality loss due to downscaling, based on Table 7.4

Since the target resolution is lower than the original resolution, information from the source must be bundled into the target frame size and is therefore partially lost. These scaling calculations may vary depending on the implementation, which can lead to different pixel values and thus a worse PSNR. To confirm that the discrepancy in quality was due to downsampling and not the source material, we ran another test transcoding directly from 720p24 to 720p24 with the same settings. The results in Figure 7.6 show that the quality is comparable to the 1080p24 to 1080p24 transcoding. This leads to the conclusion that the PSNR quality metric is not suitable for a direct quality comparison of downscaled material against material with the same source and target resolution. However, it still gives us an indication of the quality of the different codecs. This also applies to the other metrics like SSIM.

The results for the SSIM metric in Figure 7.7 look similar to PSNR. The quality difference between x264 and x265 is smaller than the difference between the hardware codecs. For the 2160p30 source video, the transcoded and downscaled 1080p30 video achieves a similar quality to the transcoded 2160p30 video. Again, the H.265-based codecs deliver better results than the H.264 codecs at the same setting, with the best results coming from the x265 software codec.

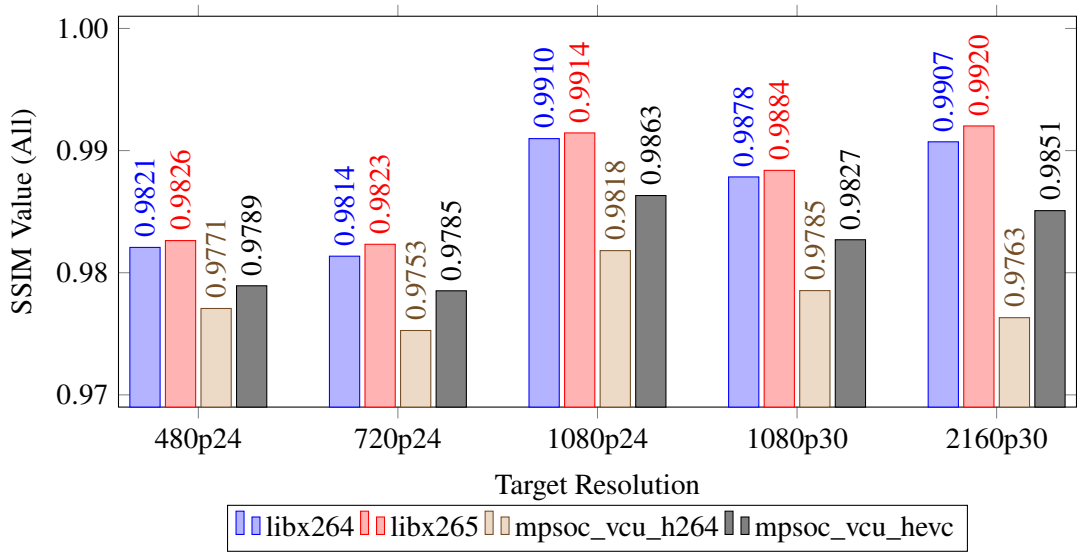

Figure 7.7: SSIM quality measurement based on Tables 7.5 and 7.6

Because VMAF was designed as a measure of subjective perception of video quality, the rating is different from the other two metrics used. VMAF also rates the worst quality at 480p24 and 720p24 in general, but the values for the same codec at different resolutions are closer together in Figure 7.8. Like with SSIM, the quality difference between software H.264 and H.265 is much smaller than the difference between the equivalent hardware codecs. Interestingly, transcoding from 2160p30 to 2160p30 with the hardware-based H.264 codec yields the worst result. It's even worse than the hardware-based H.264 transcoding for 480p and 720p. While the other metrics also rated this comparison low, VMAF delivered the worst rating.

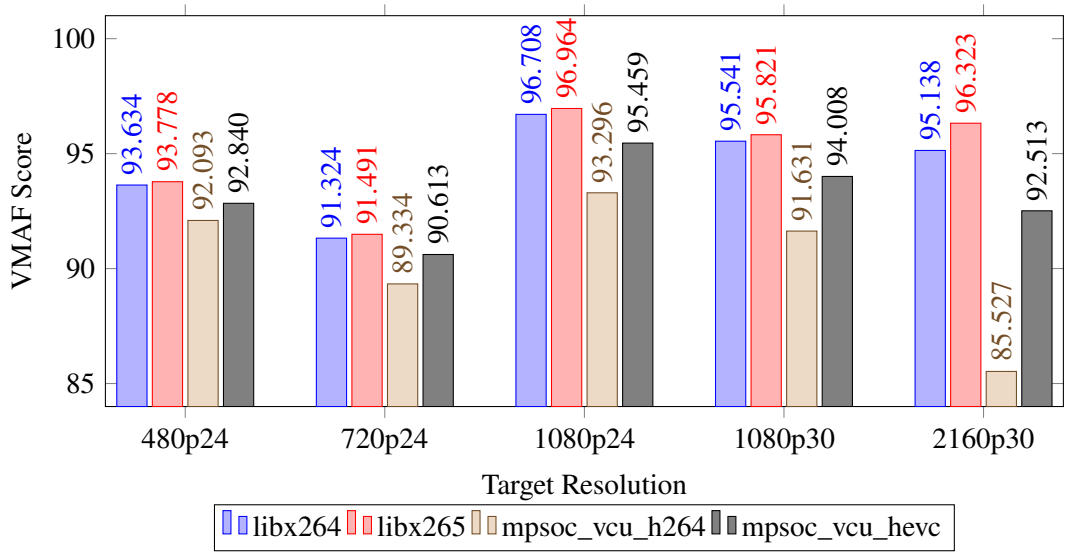

Figure 7.8: VMAF quality measurement based on Table 7.7

7.3.3 Assumptions on Power Usage

Because we used AWS to implement our solution, we do not have access to the hardware and cannot make accurate statements about power consumption. Virtualization also means that several customers can share a server, further reducing the consumption per customer as well.

Nevertheless, we can make assumptions about the power consumption. Assuming that power consumption increases proportionally to the run time, power consumption estimations should be derivable from the time evaluation in Section 7.3.1.

In reality, the power consumption of the FPGA codecs will be much lower than with the software implementations. This is because software codecs perform the reading and processing of the video file on the CPU, resulting in a high load. With the hardware implementation, the server deals with reading, decoding and writing, while the most complex part, the encoding, takes place on the FPGA. If we look at the benchmarks from Tables 7.2 and 7.3 and consider the time the process spends in user space (utime) and kernel space(stime), the results for the FPGA are much lower than the software codecs. One can also outsource the decoding to the FPGA as well, however, the results of Table 7.3 have shown that it has little influence on the total time.

Xilinx specifies a typical power consumption of 18-25 W for the U30 Accelerator Card, while modern CPUs can easily reach a multiple of that. For the processor in question in our AWS EC2 VT1 instance, Intel quotes a Thermal Design Power (TDP) of 210 W for its P-8259CL. The TDP is not a measurement for the actual power usage under load, but an indicator for the maximum heat development of a computer component. Hennessy and Patterson claims that the peak power consumption is 'often 1.5 times higher' than the TDP [HP11].

7.3.4 File size

Thanks to the use of comparable commands with CBR, the file size achieved from different codecs is almost identical for our evaluation, as seen in Table 7.8. The FPGA-generated video streams are usually a few MiB smaller, but the impact on the overall file size is small enough to be not relevant.

With the use of other quality settings, especially CRF, it is likely that there could be different results. We cannot confirm this, since CRF is not supported by the FPGA.

7.4 Discussion

For the comparison between the software implementation and the FPGA, we established various criteria and metrics to obtain the above results. To conclude the evaluation, we create a graph with the collected data to plot encoding profiles based on performance and quality (Figure 7.9). The performance in FPS is lower than in the reference Figure 3.2 because our evaluation is a real experiment while the reference states the total performance possible. In addition, it should be noted that PSNR is worse for downscaled material. Because of this, the individual score regarding quality can only be compared within appropriate groups, but in these groups, the ratio is comparable in each case. Nevertheless, our conclusion finds that the FPGA implementation is comparable to GPU solutions.

Similar to the existing GPU based solutions in Chapter 3, we gain speed by sacrificing quality. Overall, we found that the used FPGA is far more efficient than the software implementation for both standards. In addition, while there is a strong discrepancy between the computation times of H.264 and H.265 in the software implementation, both standards are computed in a similar time using the FPGA. While the performance of the software codecs also falls below the frame rate of the target video at higher resolutions, the performance of the hardware codecs was always above the frame rate. This implies that the FPGA is suitable for real-time transcoding at higher resolutions.

In terms of quality, H.265 achieves higher quality than H.264 with the same target parameters. The software codecs achieve higher quality overall than the FPGA codecs, which was confirmed by all quality metrics used. Transcodings with identical source and target resolutions produced the best quality scores compared to downscaled target video files.

In addition to speed, power consumption is also a significant advantage of an FPGA accelerator card over a CPU or dedicated GPU. As shown in Section 7.3.3, using the U30 accelerator card results in a much lower power consumption in comparison to using the processor. Even consumer processors are often specified with 65 W TDP or more, which is still worse than the typical power consumption of the U30.

On the other hand, if a GPU is used for transcoding, the total power consumption looks even worse. It is not uncommon for modern GPUs to consume 150 W or more. Nvidia, for example, specifies a power consumption of 320 W for an RTX 3080 alone. Thanks to NVENC, the power consumption of the encoding process does not correspond to full load, but it should not be underestimated.

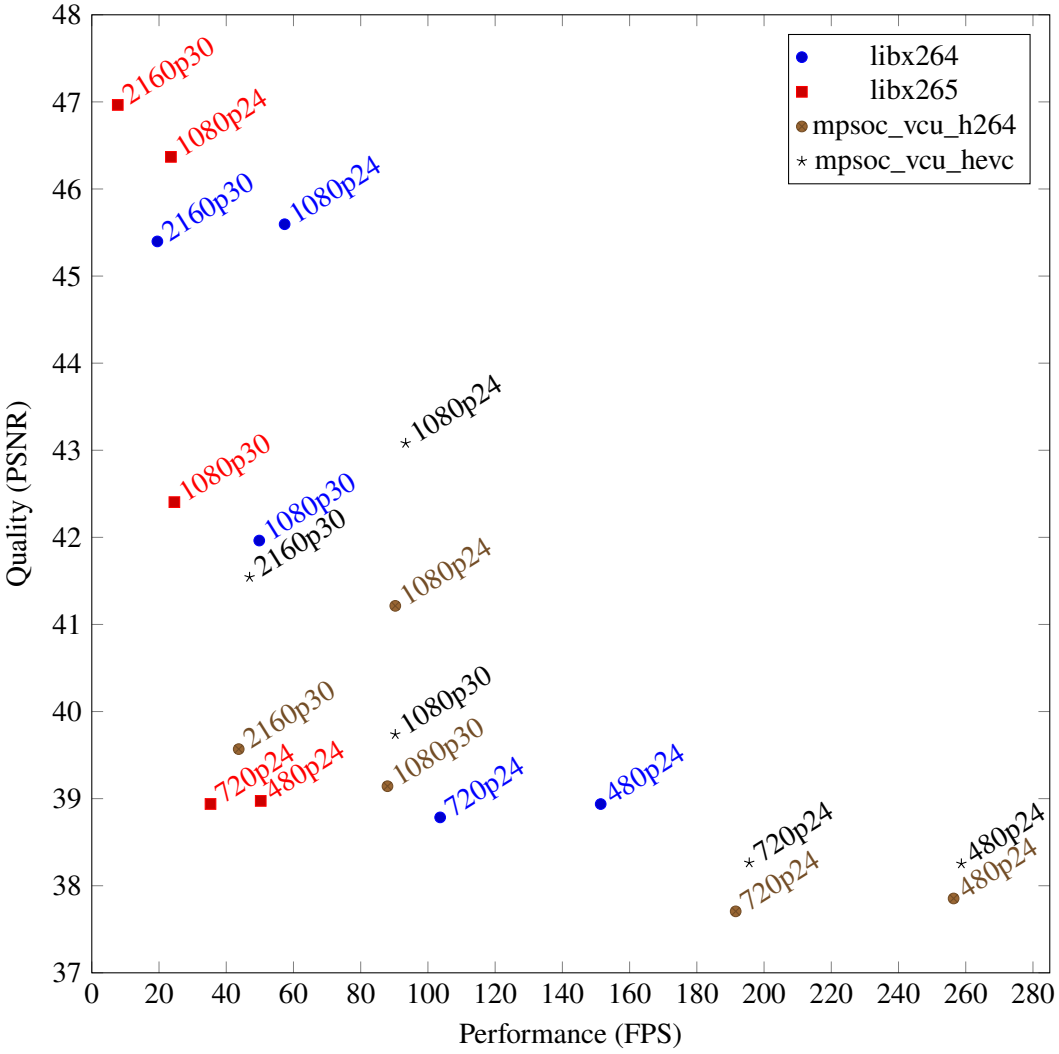

Figure 7.9: Performance evaluation of the different codecs based on single encodes and PSNR for the given target resolutions

7.5 Raw Data

This section contains the raw data collected during the evaluation step.

7.5.1 Encoding benchmarks

Single Encodes

utime, stime, rtime and maxrss are retrieved with the benchmark flag. Performance is calculated by dividing the amount of frames (14315 for 1080p24 source material, 19038 for 2160p30 source material) by the rtime, rounded to two digits after the delimiter.

Codec	**utime** (s)	**stime** (s)	**rtime** (s)	**maxrss** (kB)	**Performance** (fps)
1080p24 to 480p24					
libx264	729.725	7.454	94.552	340640	151.40
libx265	1213.939	10.969	284.878	353116	50.26
mpsoc_vcu_h264	134.929	1.513	55.833	155156	256.39
mpsoc_vcu_hevc	134.062	1.750	55.334	155904	258.70
1080p24 to 720p24					
libx264	1441.894	13.062	138.124	585904	103.64
libx265	2546.329	21.372	405.365	484132	35.32
mpsoc_vcu_h264	130.669	1.882	74.728	162636	191.56
mpsoc_vcu_hevc	131.916	1.780	73.188	163044	195.59
1080p24 to 1080p24					
libx264	2608.220	19.053	249.524	1016084	57.37
libx265	5457.986	27.416	608.212	742712	23.54
mpsoc_vcu_h264	97.870	2.269	158.547	176648	90.29
mpsoc_vcu_hevc	98.481	2.391	153.315	176500	93.37
2160p30 to 1080p30					
libx264	4184.173	24.334	382.109	1372368	49.82
libx265	7288.212	35.706	774.817	1099472	24.57
mpsoc_vcu_h264	707.442	3.628	216.419	528820	87.97
mpsoc_vcu_hevc	715.004	3.911	210.844	528544	90.29
2160p30 to 2160p30					
libx264	11017.833	52.295	975.683	3530696	19.51
libx265	23007.269	49.904	2462.973	2439336	7.73
mpsoc_vcu_h264	579.131	5.050	435.537	605112	43.71
mpsoc_vcu_hevc	579.017	5.623	405.559	607004	46.94

Table 7.2: Time measurements for single-transcoding processess

Multiple Encodes

Codec	**utime** (s)	**stime** (s)	**rtime** (s)	**maxrss** (kB)	**Performance** (fps)
Direct Multi-transcode from 1080p24 to 1080p24, 720p24 and 480p24					
libx264	4833.889	29.229	416.757	1612368	34.35
libx265	10186.023	51.357	917.484	1233936	15.60
mpsoc_vcu_h264	191.122	4.851	264.700	207136	54.08
mpsoc_vcu_hevc	190.847	4.818	255.810	207712	55.96
Multi-transcode with Multiscale filter from 1080p24 to 1080p24, 720p24 and 480p24					
mpsoc_vcu_h264	5.751	6.468	264.770	139636	54.07
mpsoc_vcu_hevc	5.230	6.716	255.969	139900	55.92
Direct Multi-transcode from 2160p30 to 2160p30 and 1080p30					
libx264	14678.454	55.027	1252.868	4386716	15.20
libx265	30180.522	77.513	3086.122	3016584	6.17
mpsoc_vcu_h264	737.500	7.235	438.870	637868	43.38
mpsoc_vcu_hevc	741.064	7.153	405.997	638472	46.89
Multi-transcode with Multiscale filter from 2160p30 to 2160p30 and 1080p30					
mpsoc_vcu_h264	6.078	8.014	448.151	399108	42.48
mpsoc_vcu_hevc	6.277	10.011	411.241	653052	46.39

Table 7.3: Time measurements for Multi-transcoding processess

7.5.2 PSNR

Codec	Y	U	V	Avg.	Min
1080p24 to 480p24					
libx264	37.687553	42.821806	44.841505	38.937732	28.423233
libx265	37.727176	42.811344	44.880770	38.973909	28.260874
mpsoc_vcu_h264	36.555473	42.104468	44.224241	37.853652	27.535665
mpsoc_vcu_hevc	36.971303	42.399126	44.396965	38.252210	27.945212
1080p24 to 720p24					
libx264	37.446004	43.513285	45.374139	38.784646	28.288207
libx265	37.606806	43.579765	45.494667	38.938625	28.171831
mpsoc_vcu_h264	36.336872	42.699577	44.670222	37.705784	23.859743
mpsoc_vcu_hevc	36.909342	43.159677	44.989729	38.263537	25.827590
1080p24 to 1080p24					
libx264	44.528372	48.620556	49.708389	45.595255	36.018695
libx265	45.371193	49.043310	50.064713	46.368727	37.552717
mpsoc_vcu_h264	39.949648	45.311504	47.069150	41.213480	25.305870
mpsoc_vcu_hevc	41.872720	46.928513	48.238236	43.080705	26.748632
2160p30 to 1080p30					
libx264	40.590604	47.579397	48.184075	41.963246	18.779636
libx265	41.095669	47.382357	48.007439	42.404561	18.780208
mpsoc_vcu_h264	37.718625	45.057756	46.314381	39.142575	18.779636
mpsoc_vcu_hevc	38.305434	45.928853	46.789579	39.737525	8.907356
2160p30 to 2160p30					
libx264	44.201368	49.210669	50.408853	45.397976	34.792628
libx265	45.968877	49.625499	50.662781	46.965040	35.546271
mpsoc_vcu_h264	38.145638	45.302095	46.965455	39.568604	24.887167
mpsoc_vcu_hevc	40.146732	47.113676	48.395311	41.542630	8.907356
Additional: 720p24 to 720p24					
libx264	44.500842	48.706221	49.685480	45.577313	37.094844
libx265	45.220993	48.876444	49.855594	46.212806	37.087479
mpsoc_vcu_h264	39.931105	45.125305	46.740647	41.170014	23.214672
mpsoc_vcu_hevc	41.418636	46.670774	47.916134	42.646466	26.116939

Table 7.4: PSNR quality comparison

7.5.3 SSIM

Codec	Y	U	V	All
1080p24 to 480p24				
libx264	0.982298 (17.519835)	0.979624 (16.908834)	0.983620 (17.856859)	0.982073 (17.464893)
libx265	0.983047 (17.707503)	0.979709 (16.927066)	0.983902 (17.932209)	0.982633 (17.602789)
mpsoc_vcu_h264	0.976425 (16.275474)	0.975565 (16.119878)	0.981170 (17.251557)	0.977073 (16.396441)
mpsoc_vcu_hevc	0.978592 (16.694243)	0.977390 (16.456929)	0.981829 (17.406294)	0.978931 (16.763604)
1080p24 to 720p24				
libx264	0.980869 (17.182652)	0.980304 (17.056311)	0.984349 (18.054559)	0.981355 (17.294377)
libx265	0.981997 (17.446557)	0.980929 (17.196369)	0.985069 (18.259024)	0.982331 (17.527898)
mpsoc_vcu_h264	0.973374 (15.746948)	0.976223 (16.238479)	0.981903 (17.424027)	0.975271 (16.067847)
mpsoc_vcu_hevc	0.977355 (16.450352)	0.978674 (16.710813)	0.983030 (17.703294)	0.978521 (16.679850)
1080p24 to 1080p24				
libx264	0.990193 (20.084506)	0.992186 (21.071272)	0.992928 (21.504734)	0.990981 (20.448356)
libx265	0.990678 (20.304783)	0.992630 (21.325544)	0.993339 (21.764574)	0.991447 (20.678682)
mpsoc_vcu_h264	0.979300 (16.840268)	0.985197 (18.296451)	0.988466 (19.380022)	0.981810 (17.401752)
mpsoc_vcu_hevc	0.984492 (18.094377)	0.989259 (19.689411)	0.990696 (20.313451)	0.986320 (18.639257)

Table 7.5: SSIM quality comparison for material with 24 fps

Codec	Y	U	V	All
2160p30 to 1080p30				
libx264	0.986455 (18.682195)	0.990322 (20.142125)	0.990922 (20.419947)	0.987844 (19.152072)
libx265	0.987401 (18.996778)	0.989982 (19.992107)	0.990712 (20.320631)	0.988383 (19.349141)
mpsoc_vcu_h264	0.975129 (16.043122)	0.983976 (17.952284)	0.986731 (18.771539)	0.978537 (16.683169)
mpsoc_vcu_hevc	0.980489 (17.097151)	0.986378 (18.657542)	0.987867 (19.160165)	0.982700 (17.619524)
2160p30 to 2160p30				
libx264	0.989465 (19.773755)	0.992699 (21.365911)	0.993757 (22.046214)	0.990719 (20.324280)
libx265	0.991270 (20.589729)	0.993080 (21.598888)	0.993948 (22.180677)	0.992018 (20.978756)
mpsoc_vcu_h264	0.971012 (15.377844)	0.985092 (18.265668)	0.988782 (19.500728)	0.976320 (16.256248)
mpsoc_vcu_hevc	0.982609 (17.596650)	0.989056 (19.608083)	0.991006 (20.460234)	0.985083 (18.263069)

Table 7.6: SSIM quality comparison for material with 30 fps

7.5.4 VMAF

Codec	VMAF Score	Codec	VMAF Score
1080p24 to 480p24			
libx264	93.633924	libx265	93.778391
mpsoc_vcu_h264	92.093309	mpsoc_vcu_hevc	92.839588
1080p24 to 720p24			
libx264	91.323754	libx265	91.491407
mpsoc_vcu_h264	89.333610	mpsoc_vcu_hevc	90.612638
1080p24 to 1080p24			
libx264	96.707754	libx265	96.963690
mpsoc_vcu_h264	93.296022	mpsoc_vcu_hevc	95.458523
2160p30 to 1080p30			
libx264	95.540954	libx265	95.821304
mpsoc_vcu_h264	91.631161	mpsoc_vcu_hevc	94.007749
2160p30 to 2160p30			
libx264	95.138342	libx265	96.322726
mpsoc_vcu_h264	85.526580	mpsoc_vcu_hevc	92.513364

Table 7.7: VMAF quality comparison

7.5.5 Video Stream Size

Codec	Video Stream Size	Codec	Video Stream Size
1080p24 to 480p24			
libx264	107 MiB (77%)	libx265	106 MiB (77%)
mpsoc_vcu_h264	106 MiB (77%)	mpsoc_vcu_hevc	105 MiB (77%)
1080p24 to 720p24			
libx264	178 MiB (85%)	libx265	176 MiB (85%)
mpsoc_vcu_h264	177 MiB (85%)	mpsoc_vcu_hevc	176 MiB (85%)
1080p24 to 1080p24			
libx264	284 MiB (90%)	libx265	284 MiB (90%)
mpsoc_vcu_h264	282 MiB (90%)	mpsoc_vcu_hevc	283 MiB (90%)
2160p30 to 1080p30			
libx264	302 MiB (92%)	libx265	301 MiB (92%)
mpsoc_vcu_h264	296 MiB (92%)	mpsoc_vcu_hevc	295 MiB (92%)
2160p30 to 2160p30			
libx264	521 MiB (95%)	libx265	527 MiB (96%)
mpsoc_vcu_h264	521 MiB (96%)	mpsoc_vcu_hevc	509 MiB (95%)

Table 7.8: Video stream size and percentage of total File Size

8 Conclusion and Outlook

The goal of this work was to find a way to integrate an FPGA-based accelerator card into an Opencast workflow to enable more efficient transcoding of video source material using current state-of-the-art video compression standards, and to evaluate this solution by comparing it to a purely CPU-based workflow using several metrics. Thanks to Xilinx' efforts, integrating the FPGA accelerator card is as easy as can be. All one has to do is install the manufacturer's appropriate drivers and tools, and then select the implemented video codec. A version of FFmpeg is also provided which already has the hardware codecs incorporated, making the effort required from the end user as small as possible.

The evaluation shows that significantly higher performance can be achieved with the FPGA at comparable settings, especially when using H.265. Furthermore, the power consumption is comparatively low compared to both, CPU and GPU solutions. Nevertheless, the FPGA implementation does come with disadvantages. For one, it does not come with the range of functionality provided by the software solution. Additionally, the software codecs produced higher quality output than the hardware codecs according to the quality metrics we used.

To achieve even better performance, more research can be performed.

On the **video compression standards** side, new standards can potentially achieve more efficient compression. Different approaches, some more common than others, already exist. As a successor to H.264 and H.265, the H.266 standard, also known as *Versatile Video Coding*, was published in 2020. This standard was not yet relevant enough to have a market share worth mentioning in 2021 according to the *Bitmovin Video Developer Report* [Bit21]. In the same report, the AV1 and VP9 video standards are also ahead of the curve when it comes to projected use in 2022. Provided there are efficient hardware implementations, these standards can also be worthwhile solutions for accelerating video transcoding.

On the other hand, several improvements can be made to the **hardware accelerators**. More standards can be implemented to achieve a greater variety of codecs. Additionally, current implementations can be further improved by increasing performance, increasing output quality or reducing power consumption.

For the **solution implemented in this work**, the quality of the individual encoding profiles can be further improved by using FFmpeg commands optimized for the individual codecs. While we have used comparable commands for evaluation here, various options like CRF can be used to maintain a quality level rather than relying on limiting the bit rate of the software codecs.

Bibliography

[ALM11] A. B. Atitallah, H. Loukil, N. Masmoudi. “Fpga design for h. 264/avc encoder”. In: *International Journal of Computer Science, Engineering and Applications* 1.5 (2011), p. 119 (cit. on p. 31).

[AMKH18] H. Azgin, A. C. Mert, E. Kalali, I. Hamzaoglu. “An efficient FPGA implementation of HEVC intra prediction”. In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. 2018, pp. 1–5. DOI: 10.1109/ICCE.2018.8326332 (cit. on p. 31).

[AWS02] AWS. *Amazon.com Launches Web Services; Developers Can Now Incorporate Amazon.com Content and Features into Their Own Web Sites; Extends "Welcome Mat" for Developers*. AWS. 2002. URL: https://press.aboutamazon.com/news-releases/news-release-details/amazoncom-launches-web-services (cit. on p. 33).

[AWS06] AWS. *S3 Provides Application Programming Interface for Highly Scalable Reliable, Low-Latency Storage at Very Low Costs*. 2006. URL: https://press.aboutamazon.com/news-releases/news-release-details/amazon-web-services-launches-amazon-s3-simple-storage-service (cit. on p. 33).

[AWS07] AWS. *Amazon Web Services Introduces Multiple Compute Instance Types for Amazon EC2 Customers*. 2007. URL: https://press.aboutamazon.com/news-releases/news-release-details/amazon-web-services-introduces-multiple-compute-instance-types (cit. on p. 34).

[AWSEC2] AWS. *Amazon EC2 Instance Types*. URL: https://aws.amazon.com/ec2/instance-types (cit. on p. 34).

[Bar06] J. Barr. *Amazon EC2 Beta*. AWS. 2006. URL: https://aws.amazon.com/de/blogs/aws/amazon_ec2_beta/ (cit. on p. 33).

[BB21] A. Boutros, V. Betz. “FPGA Architecture: Principles and Progression”. In: *IEEE Circuits and Systems Magazine* 21.2 (2021), pp. 4–29. DOI: 10.1109/MCAS.2021.3071607 (cit. on p. 18).

[Bit21] Bitmovin. *Bitmovin Video Developer Report*. 2021. URL: https://go.bitmovin.com/video-developer-report-2021 (cit. on pp. 20, 23, 65).

[Blender-About] *Blender - The Software*. Blender Foundation. URL: https://www.blender.org/about/ (cit. on p. 36).

[BT.2020] *Parameter values for ultra-high definition television systems for production and international programme exchange*. ITU-R. Oct. 2015. URL: https://www.itu.int/rec/R-REC-BT.2020 (cit. on p. 19).

[BT.601] *Studio encoding parameters of digital television for standard 4:3 and widescreen 16:9 aspect ratios*. ITU-R. Mar. 2011. URL: https://www.itu.int/rec/R-REC-BT.601 (cit. on p. 19).

[CS13] D. Chen, D. Singh. "Fractal video compression in OpenCL: An evaluation of CPUs, GPUs, and FPGAs as acceleration platforms". In: *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2013, pp. 297–304. DOI: 10.1109/ASPDAC.2013.6509612 (cit. on p. 31).

[DS890] *UltraScale Architecture and Product Data Sheet: Overview*. DS890. v4.2. Xilinx. May 2022. URL: https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview (cit. on p. 35).

[DS891] *Zynq UltraScale+ MPSoC Data Sheet: Overview*. DS891. v1.9. Xilinx. May 2021. URL: https://docs.xilinx.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview (cit. on p. 35).

[DS970] *Alveo U30 Data Center Accelerator Card Data Sheet*. v1.3. Xilinx. Dec. 2020. URL: https://docs.xilinx.com/v/u/en-US/ds970-u30 (cit. on p. 35).

[GGSD17] V. Goyanes, R. González, A. Sanchez Bermudez, D. Docampo. "On the use of smart recording agents in Opencast lecture capture systems". In: (Feb. 2017). DOI: 10.13140/RG.2.2.23764.40325 (cit. on p. 24).

[H.264] *Advanced video coding for generic audiovisual services*. ITU-T. Aug. 2021. URL: https://www.itu.int/rec/T-REC-H.264 (cit. on p. 21).

[HP11] J. L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Fifth. 2011, p. 22. ISBN: 9780123838735 (cit. on p. 55).

[HZ10] A. Horé, D. Ziou. "Image Quality Metrics: PSNR vs. SSIM". In: *2010 20th International Conference on Pattern Recognition*. 2010, pp. 2366–2369. DOI: 10.1109/ICPR.2010.579 (cit. on p. 51).

[ILIAS-About] *About the Open Source LMS ILIAS*. ILIAS open source e-Learning e. V. URL: https://www.ilias.de/en/about-ilias/ (cit. on p. 15).

[ILIAS20] *Recommendations for online teaching events*. Digital Teaching Taskforce, University of Stuttgart. 2020. URL: https://ilias3.uni-stuttgart.de/goto_Uni_Stuttgart_lm_2147629.html (cit. on p. 15).

[Jia11] H. Jiang. "The Intel® Quick Sync Video technology in the 2nd-generation Intel Core processor family". In: *2011 IEEE Hot Chips 23 Symposium (HCS)*. 2011, pp. 1–23. DOI: 10.1109/HOTCHIPS.2011.7477508 (cit. on pp. 27, 28).

[KH16] E. Kalali, I. Hamzaoglu. "FPGA implementation of HEVC intra prediction using high-level synthesis". In: *2016 IEEE 6th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 2016, pp. 163–166. DOI: 10.1109/ICCE-Berlin.2016.7684745 (cit. on p. 31).

[KTR06] S.-k. Kwon, A. Tamhankar, K. Rao. "Overview of H.264/MPEG-4 part 10". In: *Journal of Visual Communication and Image Representation* 17.2 (2006). Introduction: Special Issue on emerging H.264/AVC video coding standard, pp. 186–216. ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2005.05.010. URL: https://www.sciencedirect.com/science/article/pii/S1047320305000696 (cit. on pp. 21, 22).

[Opencast-About] *Opencast - Open-Source Video Management*. URL: https://opencast.org/ (cit. on pp. 15, 24).

[PY16] A. Patait, E. Young. "High performance video encoding with NVIDIA GPUs". In: *2016 GPU Technology Conference (https://goo.gl/Bdjdgm)*. 2016 (cit. on pp. 28, 29).

[Ras17] R. Rassool. "VMAF reproducibility: Validating a perceptual practical video quality metric". In: *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 2017, pp. 1–2. DOI: 10.1109/BMSB.2017.7986143 (cit. on p. 52).

[SVV+18] P. Sjövall, V. Viitamäki, J. Vanne, T. D. Hämäläinen, A. Kulmala. "FPGA-Powered 4K120p HEVC Intra Encoder". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018, pp. 1–5. DOI: 10.1109/ISCAS.2018.8351873 (cit. on p. 31).

All links were last followed on August 29, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature