UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

TULIO PEREIRA BITENCOURT

Architecture exploration and VLSI design of multi-symbol arithmetic encoders for the AV1 coding format

Thesis presented in partial fulfillment
of the requirements for the degree of
Master in Microelectronics

Advisor: Prof. Dr. Sergio Bampi
Coadvisor: Prof. Dr. Fábio Luís Livi Ramos

Porto Alegre
April 2023

CIP — CATALOGING-IN-PUBLICATION

Pereira Bitencourt, Tulio

Architecture exploration and VLSI design of multi-symbol arithmetic encoders for the AV1 coding format / Tulio Pereira Bitencourt. – Porto Alegre: PGMICRO da UFRGS, 2023.

108 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2023. Advisor: Sergio Bampi; Coadvisor: Fábio Luís Livi Ramos.

1. AV1. 2. Video coding. 3. Arithmetic encoder. 4. Hardware design. 5. VLSI architectures. I. Bampi, Sergio. II. Livi Ramos, Fábio Luís. III. Título.

“One thing a leader does is to remove the stigma of mistakes.”

— GORDON MOORE

AGRADECIMENTOS

Iniciar um curso de mestrado durante uma pandemia global de Covid-19 foi um dos maiores desafios que já vivi em toda a minha vida. Durante os dois anos em que me dediquei incansavelmente à pesquisa e às aulas, fui sortudo o suficiente por poder contar com toda a minha famliia, que sempre esteve do meu lado e meu deu forças para seguir lutando pelo meu sonho de obter o título de Mestre.

Graças aos meus pais, Roselene Alves Pereira e Getulio de Oliveira Bitencourt, que fui capaz de chegar ao final deste curso e de me tornar a pessoa que sou hoje. Graças ao apoio especial dos meus tios maternos Rui Alves Pereira e Sérgio Alves Pereira, assim como o suporte dos meus tios paternos, que consegui me dedicar à pesquisa e aos eventos que participei. Meus irmãos, Gerson Barreto Bitencourt, Loriege Pessoal Bitencourt e Magnus Cesar Pessoal Bitencourt sempre foram a minha inspiração e me incentivaram a crescer, amadurecer e me atingir meus objetivos.

Além disso, cada uma das diversas pessoas que conheci durante esses últimos anos contribuiu de alguma maneira para esse momento. Sou extremamente grato à todas as pessoas que convivi por terem participado da minha jornada.

ABSTRACT

To reduce the impact of videos in the global Internet capacity, companies rely upon video coding standards and formats, also known as codecs, to reduce the overall sizes of videos before transmitting or storing them. AV1, which arises as a promising state-of-the-art and royalties-free video coding format first released in 2018, aims to reduce the sizes of videos by applying novel techniques to boost AV1's compression results.

Amongst its core components, AV1 comprises an entropy coding block, which is responsible for losslessly encoding symbols generated by other core modules (e.g., intra-prediction, motion compensation, etc.). The arithmetic encoder, which is part of the entropy encoder, is a bottleneck due to its difficulty to work with parallelizations, and relies upon two primary operations: *CDF Operation* and *Boolean Operation*, where CDF stands for Cumulative Distribution Function.

This thesis proposes a baseline VLSI design, which was named AE-AV1, as the first ever AV1 arithmetic encoder found in the literature, and capable of reaching ultra-high performance (i.e., processing of 8K@120fps videos in real-time). Moreover, additional versions of this architecture were proposed as AE-AV1-LP and AE-AV1-MB, which are, respectively, a low-power version and a novel design applying a Multi-Boolean technique also introduced in this thesis. All the herein proposed designs were synthesized using the Cadence™ RC tool and the ST 65nm PDK. As the AV1 is well-known for being an open-source alternative in the video coding industry, the AE-AV1 architecture was also synthesized from Verilog to GDSII layout using a fully open-source ASIC flow (i.e., OpenROAD tool, OpenLane flow, and ASAP7 and SkyWater 130nm PDKs).

The architectures were capable of reaching frequencies of 581 MHz, 563 MHz and 590 MHz for the versions AE-AV1, AE-AV1-LP and AE-AV1-MB 2-bool, respectively. With regard to throughput rates, all herein introduced architectures are capable of reaching 8K@120fps real-time video processing with rates of 1.032 Gbits/sec, 0.999 Gbits/sec and 1.117 Gbits/sec respectively.

Keywords: AV1. video coding. arithmetic encoder. hardware design. VLSI architectures.

RESUMO

Para reduzir o impacto dos vídeos na capacidade global de Internet, as empresas contam com padrões e formatos de codificação de vídeo, também conhecidos como *codecs*, para reduzir os tamanhos dos vídeos antes de transmiti-los ou armazená-los. O AV1, que surge como um promissor formato de codificação de vídeo de última geração e livre de *royalties* lançado pela primeira vez em 2018, visa reduzir os tamanhos dos vídeos aplicando técnicas inovadoras e aprimoradas para aumentar os resultados de compactação do AV1.
Entre seus componentes principais, o AV1 compreende um bloco de codificação de entropia, que é responsável pela codificação sem perdas de símbolos gerados por outros módulos (por exemplo, predição intra-quadro, compensação de movimento, etc.). O codificador aritmético, que faz parte do codificador de entropia, é um gargalo devido à sua dificuldade em trabalhar com paralelizações e conta com duas operações principais: *CDF Operation* e *Boolean Operation*, onde CDF representa *Cumulative Distribution Function*.
Esta dissertação propõe um projeto VLSI digital, nomeado AE-AV1, como o primeiro codificador aritmético AV1 encontrado na literatura e capaz de atingir desempenho *ultra-high* (ou seja, processamento de vídeos 8K@120fps em tempo real). Além disso, versões adicionais desta arquitetura foram propostas como AE-AV1-LP e AE-AV1-MB, que são, respectivamente, uma versão de baixo consumo (*low-power*) e um design inovador aplicando uma técnica *Multi-Boolean* também introduzida nesta dissertação. Todos os projetos aqui propostos foram sintetizados usando a ferramenta Cadence™ RC e o PDK ST 65nm. Como o AV1 é conhecido por ser uma alternativa de código aberto na indústria de codificação de vídeo, a arquitetura AE-AV1 também foi sintetizada de Verilog a *layout* GDSII usando um fluxo ASIC totalmente de código aberto (ou seja, ferramenta OpenROAD, fluxo OpenLane e PDKs ASAP7 e SkyWater 130nm).
As arquiteturas foram capazes de atingir frequências de 581 MHz, 563 MHz e 590 MHz nas versões AE-AV1, AE-AV1-LP e AE-AV1-MB 2-bool, respectivamente. Com relação às vazões, todas as arquiteturas são capazes de processar vídeos 8K@120fps em tempo real com taxas de 1.032 Gbits/seg, 0.999 Gbits/seg e 1.117 Gbits/seg respectivamente.

Palavras-chave: AV1, codificação de vídeo, codificador aritmético, projeto de hardware, arquiteturas VLSI.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ABNT	Associação Brasileira de Normas Técnicas
AC	Arithmetic Coding
ACM	Assiciation for Computing Machinery
AE-AV1	Arithmetic Encoder - AV1
AE-AV1-LP	Arithmetic Encoder - AV1 - Low-Power
AE-AV1-MB	Arithmetic Encoder - AV1 - Multi-Boolean
AOMedia	Alliance for Open Media
ASAP7	Arizona State ARM Predictive 7nm PDK
ASIC	Application-Specific Integrated Circuit
AV1	AOMedia Video 1
AVC	Advanced Video Coding
AVS	Audio Video Standard
BAE	Binary Arithmetic Encoder
CABAC	Context-Adaptive Binary Arithmetic Coding
CBAC	Context-based Binary Arithmetic Coding
CDEF	Constrained Directional Enhancement Filter
CDF	Cumulative Distribution Function
CMOS	Complementary Metal-Oxide-Semiconductor
CQ	Constraint of Quality
CQs	Constraints of Quality
CSV	Comma-Separated Values
DecCDF	Decoder for the Cumulative Distribution Function
EC	Entropy Coder
ET-BAE	Energy-Throughput Binary Arithmetic Encoder

FIFO	First In, First Out
FME	Fractional Motion Estimation
fps	frames-per-second
HDL	Hardware Description Language
HEVC	High Efficiency Video Coding
IEEE	Institute of Electrical and Electronics Engineers
IF	Interpolation Filter
INF	Instituto de Informática
LSB	Least Significant Bit
MaAD	Multi-alphabet Arithmetic Decoder
MACs	Multi-Alphabet Cores
MC	Motion Compensation
MHz	Mega Hertz
MRTR	Maximum Real-Time Resolution
MSB	Most Significant Bit
nm	nanometer
PDK	Process Design Kit
PGMICRO	Programa de Pós Graduação em Microeletrônica
RTL	Register Transfer Level
ST	STMicroelectronics, company
TCF	Toggle Count File
TSMC	Taiwan Semiconductor Manufacturing Company
UFRGS	Universidade Federal do Rio Grande do Sul
UHD	Ultra-High-Definition
UNIPAMPA	Universidade Federal do Pampa
VLSI	Very Large Scale Integration

VoD	Video-on-Demand
VP8	Video Codec VP8
VP9	Video Codec VP9

CONTENTS

1 INTRODUCTION

The Video-on-Demand (VoD) industry has seen, throughout the last few decades, a remarkable growth. Not only more costumers are actively joining the streaming services provided by companies such as Netflix, YouTube, PrimeVideo, Disney Plus, etc, but even more companies are starting to operate in this industry (e.g., Brazilian company Globo with its GloboPlay platform).

After decades of evolution in the opened network television market, well-established companies such as American Broadcasting Company (ABC) and National Broadcasting Company (NBC) started their move towards the Internet, as the technology finally created a good opportunity for these companies. The traditional television has observed a free fall in demand and costumers ever since the Internet have became wide-spread worldwide and the price for bandwidth became low enough to allow more people to broadly use this technology (ROSENBAUM, 2014).

As more people started widely using video-on-demand systems, which have become a backbone for industries based on videos (e.g., live television, movies, news, etc), videos became a burdensome kind of data to be managed, stored and transmitted. This is mainly caused by the huge complexity inherent to them, as they can be briefly defined as a combination of images presented in the right speed and order to a viewer, which causes a motion sensation in the screen. Hence, the importance of methodologies that allow data compression of these kind of data emerges as a key element of the ever-increasing VoD industry of the XXI century, i.e., video coding standards or formats.

Since 1988, industry and academia have combined efforts to develop an effective coding format that can compress videos, combining a manageable complexity with great data compression. The first official video codec to be introduced, the H.261 (ITU-T, 1988), was created as a joined effort between the International Telecommunication Union (ITU), sector Telecommunication Standardization Sector (ITU-T), and companies such as Hitachi, Toshiba and PictureTel (ITU-T, 1989).

After the H.261 coding format, newer and more enhanced codecs have been released until reaching the widely used Advanced Video Coding (H.264/AVC) (ITU-T; ISO/IEC, 2003) and High Efficiency Video Coding (H.265/HEVC) (ITU-T; ISO/IEC, 2013). State-of-art video coding formats are the recently released AOMedia Video 1 (AV1) (HAN et al., 2020) and Versatile Video Coding (VVC, H.266) (ITU-T; ISO/IEC, 2020). The former, which is the target of the acceleration presented throughout this

work, inherited its features from the previously released VP8 (FELLER et al., 2016), VP9 (GRANGE; RIVAZ; HUNT, 2016), VP10 (MUKHERJEE et al., 2015), Thor (ROSENBERG, 2015) and Daala (VALIN et al., 2016). Figure 1.1 depicts the timeline of video codecs from 1999 to 2018 (i.e., the releases of H.261 in 1988, and VVC in 2020, are not displayed).

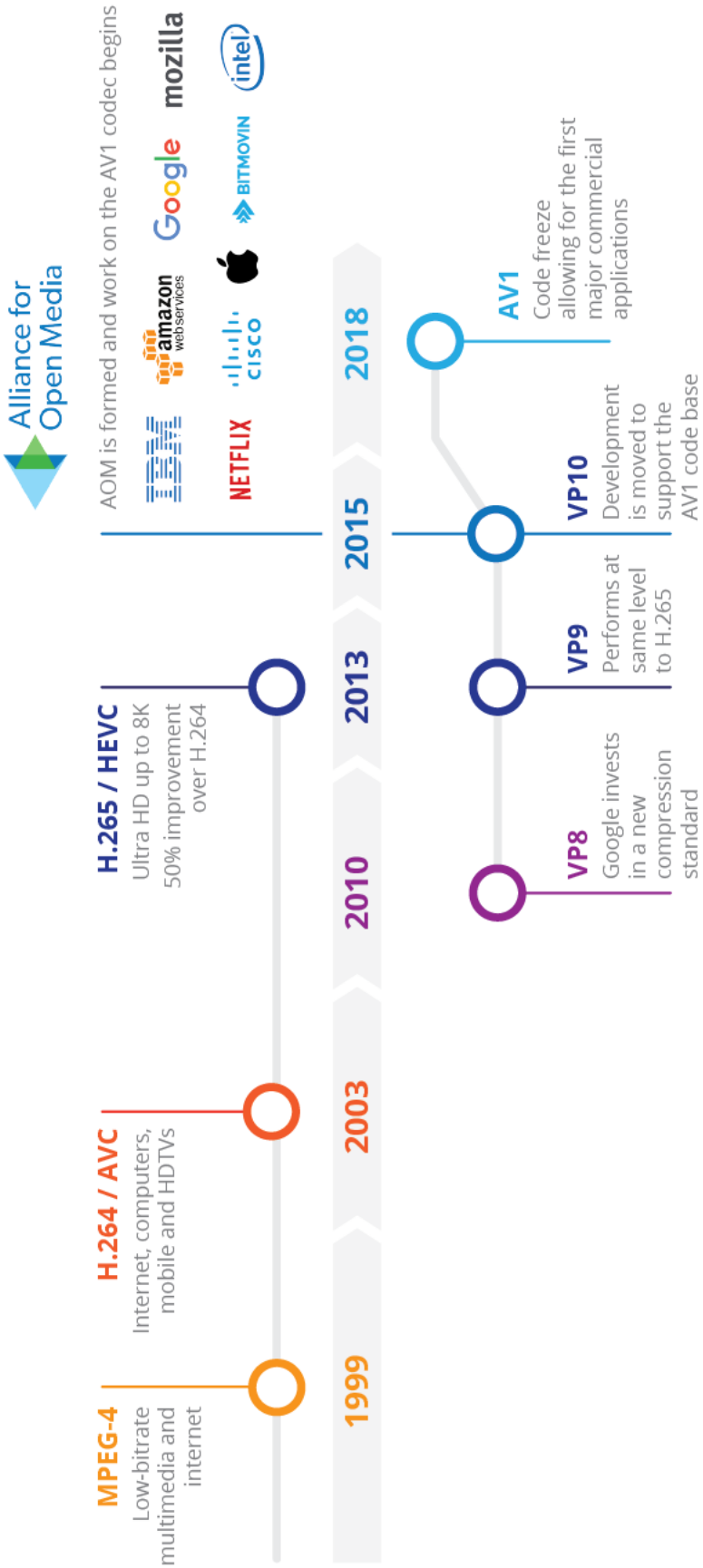
The AV1 video codec, which was released in 2018, can be tracked all the way to 2015, when the consortium of companies responsible for its development was formed. The Alliance for Open Media (AOMedia) (AOMedia, 2020) created the AV1 to become a royalties-free video coding format, and accomplished that by requesting that the member companies waived their algorithmic patent rights in favor of the AOMedia. Founding members of the AOMedia include Amazon, Cisco, Google, Intel, Microsoft, Mozilla and Netflix (SHANKLAND, 2015), with the general members group including companies such as Adobe, Vimeo and Xilinx (AOMedia, 2021).

The AV1 codec brought to the market a 50% higher compression rate when compared to the AVC codec and 30% when compared to its predecessor VP9 (HAN et al., 2020) while holding the royalties-free title. Furthermore, AV1 is considered to be a hybrid video codec (i.e., a codec composed of prediction combined with transforms), as its internal cores are organized in a way that leaves the entropy coding block as the final step in the codification process.

The arithmetic encoder of the AV1 codec, which is part of the entropy coding block, is located at the final part of the encoder implementation, receives the thus far generated symbols and generates bitstreams. This final step is responsible for generating a more optimal coded file, as it is capable of considerably reducing the overall size of the thus far generated symbols by applying a lossless codification procedure.

As already experienced in the arithmetic encoding blocks of prior video codecs (e.g., HEVC, AVC), this step has difficulties to deal with parallelization techniques because of its sequential structure and the re-feeding of value (e.g., $Range$ and Low, explained more thoroughly in Chapters 3 and Chapters 4), which emerges as problematic due to the complexity of its internal operations. A general-purpose arithmetic encoder relies upon computationally complex operations such as multipliers, which add considerable delay to the entire system.

Figure 1.1: Timeline of video codec releases from 1999 to 2018.



Source: (BHATIA, 2018).

1.1 Motivation and Problem Definition

With the ever-increasing demand for videos to be transmitted on the Internet, video codecs have become essential tools to tackle the massive traffic of data required to supply the modern society's addiction to videos. The AV1, as introduced in this chapter, is a royalties-free state-of-the-art video codec that is very effective on reducing the sizes of videos.

At the same time as the demand for videos is growing, streaming platforms, social networks and the academia is in a forever pursuit for video qualities that accurately reflect reality. Video resolutions have skyrocketed during the last decades, with current researches aiming to achieve 16K resolutions. Furthermore, as state-of-the-art video resolutions are reaching new levels (i.e., 8K and 16K), previous but still competent technology, which create less sizable videos are becoming cheaper (i.e., Full HD and 4K). As it is a tradition in many areas of the markets worldwide, when a given technology starts to become cheaper, more people start to have access to it.

Therefore, with the ever-increasing demand for videos, specially observed in social media networks such as TikTok, Facebook and YouTube, as well as streaming platforms such as Netflix, PrimeVideo and StarPlus, the importance of video coding in the XXI century has also skyrocketed. As video codecs now need to tackle incredibly sizable videos, such as the ones recorded in 8K resolutions, their complexity has also catapulted.

State-of-the-art video codecs such as AV1 and VVC, although very competent on reducing the sizes of videos, need to rely on very complex algorithms and methodologies to achieve the impressive compression rates demanded by the video industry in 2022. Hence, with their performance and power efficiency diminished, solutions allowing real-time video processing, using reasonable power, emerge as critical technologies, and, sometimes can only be achieved using ASICs (Architecture-Specific Integrated Circuit).

This work targets the AV1 arithmetic encoder block and proposes hardware designs capable of achieving real-time video encoding for 8K videos at up to 120 frames-per-second (fps). The herein presented research project resulted in three versions of the AV1 arithmetic encoder architecture, where (i) AE-AV1 is presented as high-throughput, (ii) AE-AV1-LP is presented as low-power, and (iii) AE-AV1-MB uses a multi-boolean approach also proposed by this work.

1.2 AV1 Related Works

Though very important in the overall evolution of the VoD industry, as it inserts an option in the coding formats market for a royalties-free solution, the AV1 codec is mostly untackled by academic work with regard to hardware designs. Among its core components, the AV1, as did previous coding formats such as HEVC (ITU-T; ISO/IEC, 2013) and VP9 (MUKHERJEE et al., 2013), presents several computationally complex blocks that are unable to achieve ultra-high performance (real-time coding for an 8K@120fps video, for instance) in a software-only solution. This section presents, in publication date order, works that target AV1 internal components for hardware accelerations and does not present, however, any work addressing AV1 arithmetic encoding as, to the best of the author's knowledge, this dissertation introduces the first ever proposed designs for the aforementioned step of the AV1 codec.

The next year after the AV1's formal release, Domanski et al. (2019), Correa et al. (2019b) and Correa et al. (2019a) proposed hardware designs for the AV1 codec. The former worked on the motion compensation capabilities of the codec, whereas the latter two explored the intra-prediction block. More specifically, the work proposed by Domanski et al. (2019) aims to present a multi-filter hardware design for the AV1 motion compensation (MC) operations. This work presented two different MC designs targeting 2160p@30fps and 4320p@30fps real-time processing.

The works by Correa et al. (2019b) and Correa et al. (2019a) presented solutions that tackled the AV1 intra-prediction block's computational complexity by designing efficient hardware architectures. The first publication introduces a high-throughput hardware architecture for four AV1 intra prediction modes (i.e., *Paeth*, *Smooth*, *Smooth Vertical* and *Smooth Horizontal*) and is capable of achieving ultra-high definition (UHD) 4K at up to 30fps in real-time. Differently, Correa et al. (2019a) proposed a highly parallelized hardware design for the AV1 Paeth intra predictor, which supports all block sizes allowed (i.e., 19). The latter work is capable of processing UHD 4K videos at up to 120fps.

The year of 2020 was the time for Zummach et al. (2020) and Correa et al. (2020) to propose their works on, respectively, the Constrained Directional Enhancement Filter (CDEF) of the AV1 decoder and the AV1 non-directional intra modes. The former presented an architecture capable of processing 4K@60fps, whereas the latter processes 4K@30fps.

In 2021, the works by Freitas et al. (2021), Domanski et al. (2021) and Neto

Table 1.1: Comparison between AV1 related works by Domanski et al. (2019), Correa et al. (2019b), Correa et al. (2019a), Zummach et al. (2020), Correa et al. (2020), Neto et al. (2020) Freitas et al. (2021) and Domanski et al. (2021).

Designs	Target Block	Tech. (nm)	Freq. (MHz)	Gates Count	Power (mW)	MRTR[6]
Domanski et al. (2019)	MC[1]		279.9	141.1 K	81.3	4K@30
Correa et al. (2019b)	Intra-	40[2]	648	109.6 K	16.1	4K@30
Correa et al. (2019a)	Prediction		315	247.3 K	268.3	4K@120
Zummach et al. (2020)	CDEF[3]		23	369 K	65.0	4K@60
Correa et al. (2020)	Intra-	40[2]	648	128.5 K	65.5	4K@30
Neto et al. (2020)	Prediction		476	691.7 K	382.0	1K@60
Freitas et al. (2021)	MC[1]	65[4]	345	270.4 K	130.7	2K@120
Domanski et al. (2021)	FME[5]	40[2]	686	72.6 K	26.79	8K@30

[1] Motion Compensation
[2] TSMC PDK
[3] Constrained Directional Enhancement Filter
[4] STMicro PDK
[5] Fractional Motion Estimation
[6] Maximum Real-Time Resolution

et al. (2020) were introduced into the literature as other solutions to deal with the AV1 core components' complexity. More specifically, Freitas et al. (2021) presented a design for the fractional-pixel interpolation filter, Domanski et al. (2021) proposed a low-power approximated architecture for the Fractional Motion Estimation (FME), and Neto et al. (2020) their highly parallelized ASIC solution for directional intra-frame prediction.

Table 1.1 briefly presents results extracted from the herein presented works. As one may notice, the majority of the works are capable of processing UHD videos (i.e., 4K and 8K), with some exceptions reaching less demanding resolutions (i.e., 1K and 2K). One may find this information in the Maximum Real-Time Resolution (MRTR) row of Table 1.1, which is presented as the most suitable way to compare each of the depicted works as they have different constraints with regards to throughput rates due to targeting different blocks of the codec. Regarding the additional information within Table 1.1, one may find feasible to conclude that frequency, area and power consumption cannot be defined within an specific range, as there are huge disparities between the works.

At the time this dissertation was being written, academic repositories lack works targeting the AV1 arithmetic coding block (both encoder and decoder). During the period in which this research was undergoing, the works (BITENCOURT; RAMOS; BAMPI, 2021; GOMES; RAMOS, 2021; BITENCOURT; RAMOS; BAMPI, 2022a; GOMES et al., 2022), all from the same research group, were published and became the first-ever published hardware acceleration for the AV1 entropy coding module. The works by Gomes e Ramos (2021) and Gomes et al. (2022) are presented, in-detail, in Subsection 3.3.1.

1.3 Objectives

Throughout the writing of this Master's thesis, all research, analysis and implementation developed during a period of 2 years was included. The major goals to be achieved during the research that originated this Master's thesis are listed below.

1. To analyze works connecting Video Coding to Digital Integrated Circuits design.
2. To explore and propose hardware accelerations for the AV1 arithmetic encoder block.
3. To introduce a novel technique to enhance the throughput rate of the AV1 arithmetic encoder.

1.4 Outline

This dissertation was organized in a way that all basic knowledge regarding video and video coding was introduced prior to the presentation of the proposed designs and results. Hence, this document's is as follows:

- In Chapter 2, video concepts were introduced. From the origins of videos, through the emergence of novel technologies such as digital and video coding, Chapter 2 provides insights on how videos became what their are in the XXI century.
- In Chapter 3, arithmetic coding is introduced. This chapter pose a critical role in this work, as it explains, in detail, how a general-purpose arithmetic encoder behaves and what to expect from the AV1 arithmetic encoding block, which is also depicted in Chapter 3.
- In Chapters 4 and 5, the author's contributions to the AV1 codec and arithmetic encoder are presented. The former chapter presents the baseline version of the circuitry created to accelerate the AV1 arithmetic encoder block proposed by the author, whereas the latter chapter presents the novel techniques and approaches used to enhance the baseline architecture's capabilities.
- In Chapter 6, the results obtained after detailed and thorough analysis are presented. Beyond just presenting the results, Chapter 6 provides insights on how the results

were obtained, and discusses the meaning of the results when comparing to related works that are already published in the literature.

2 VIDEO PROCESSING CONCEPTS

Throughout the XX century, the world saw an incredible evolution in video technology and mass communication tools (e.g., radio, television). One of the greatest products created in the second half of the XX century was the digital video technology, which can be briefly described as an electronic representation of moving visual images. The creation of digital videos changed the everyday lives of millions of people forever by allowing videos to be more easily managed (i.e., captured, edited, stored and transmitted) using computers.

Moreover, combined with the creation of digital videos and the ever-increasing pursuit for greater image qualities, another problem emerged: the massive quantity of data required to store and transmit videos. To tackle this problem, scientists developed different techniques and algorithms to encode and decode digital video with the goal of reducing the overall size of videos for a brief period of time, while ensuring a good trade-off between size reduction and quality loss (i.e., when the video is returned to its original state for display).

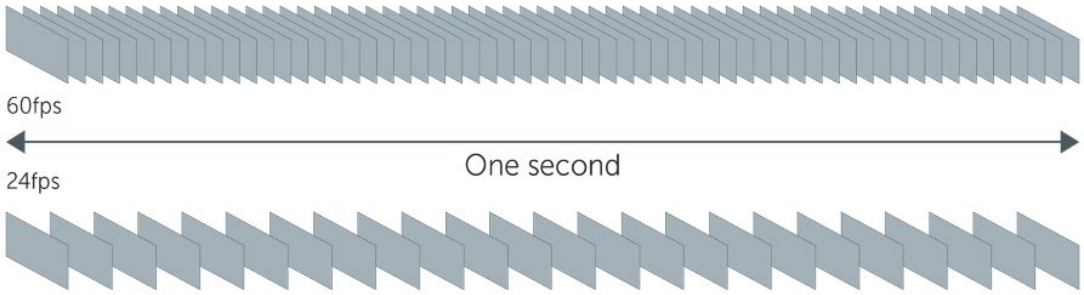
This chapter introduces the theory behind the creation and use of digital videos (Section 2.1) and video coding techniques (Section 2.2). Furthermore, this chapter also presents key components of the state-of-the-art video codec called AV1 (Section 2.3).

2.1 Digital Videos

Ever since humans developed a way to show and store the views of things, landscapes, etc, as images, videos became possible. The concept of videos is based on the sequential presentation of several images, the so-called *frames*, with enough speed (i.e., minimum of 24 frames-per-second – fps) (RICHARDSON, 2011) to give a viewer the impression of movement. Although the images themselves are static and unchangeable, they emulate movement when they are quickly swapped by another image. Figure 2.1 depicts, at the top, an arrangement with 60fps and, at the bottom, an arrangement with 24fps.

With the emergence of electronics, scientists found a way to represent videos by using analog signals, which originated analog videos (Maxim Integrated, 2002). In the 1980s decade, the first digital video format was introduced by Sony and receive the name of D-1 (REMLEY, 1986).

Figure 2.1: Arrangements with 60fps at the top and 24fps at the bottom.



Source: (TESSEROLI; SEIKE, 2021).

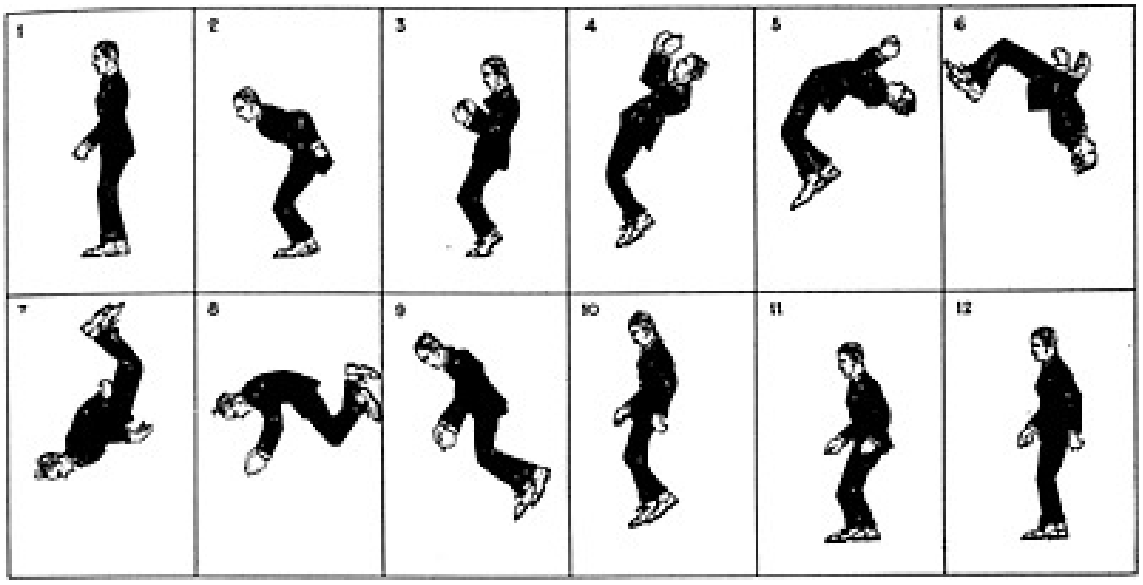
There are a few different concepts that are required to understand how video processing works. The following subsections present three concepts that are paramount for the clear understanding of digital video: frames (Subsection 2.1.1), pixels (Subsection 2.1.2), and color (Subsection 2.1.3).

2.1.1 Frames

The main goal of videos is to present a scene, which is composed by movement, in a static screen or device. In order to accomplish that, videos emulate movement by presenting several images in sequence. A good example for how videos emulate movement is a notebook with a drawing in each page, as shown in Figure 2.2. When all pages as quickly turned, in sequence, a feeling of movement emerges in the viewer.

The state-of-the-art video technologies still follow the same idea presented within Figure 2.2, with the only different that everything is digitally stored and displayed, instead of mechanically turning pages in a sketchbook. Additionally, the human eye needs, at least, 24 images, hereafter referred as frames, per second for the movement to appear smooth (RICHARDSON, 2011). If less that 24 frames-per-second (fps) are presented, the viewer will them feel the video lagging. If more than 24 fps is presented, then the video will appear even smoother and fluid. Figure 2.3 presents a sequence of frames from a given video to exemplify this concept.

Figure 2.2: Flip book animation. When the pages are quickly turned, in sequence, the character appears to be moving.

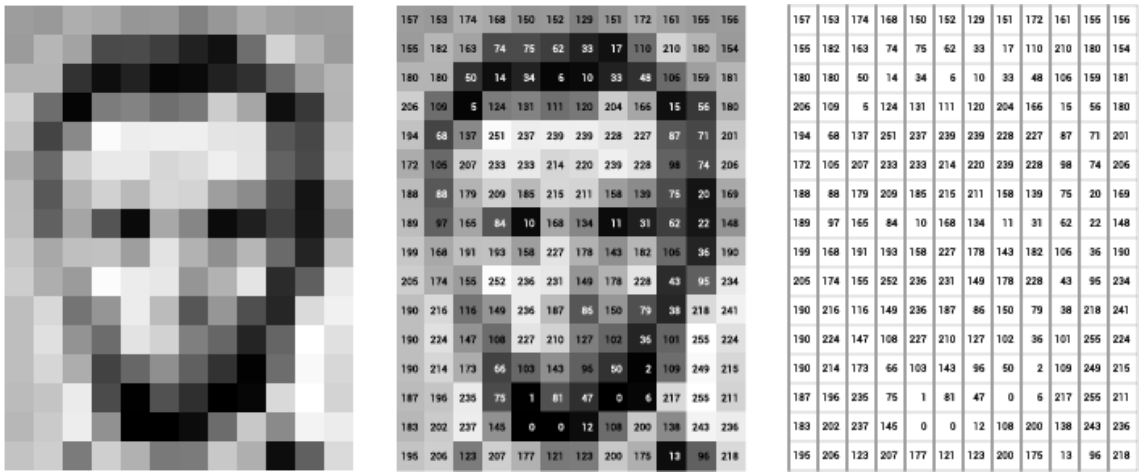


Source: (Graphic Communications, 2016).

Figure 2.3: Sequence of frames in a video sequence.

Source: (OWEN, 2012).

Figure 2.4: Black and White pixels decomposition.



Source: https://ai.stanford.edu/ syyeung/cvweb/tutorial1.html (Accessed in June 25th, 2022).

Table 2.1: Table presenting the commonly used video resolutions.

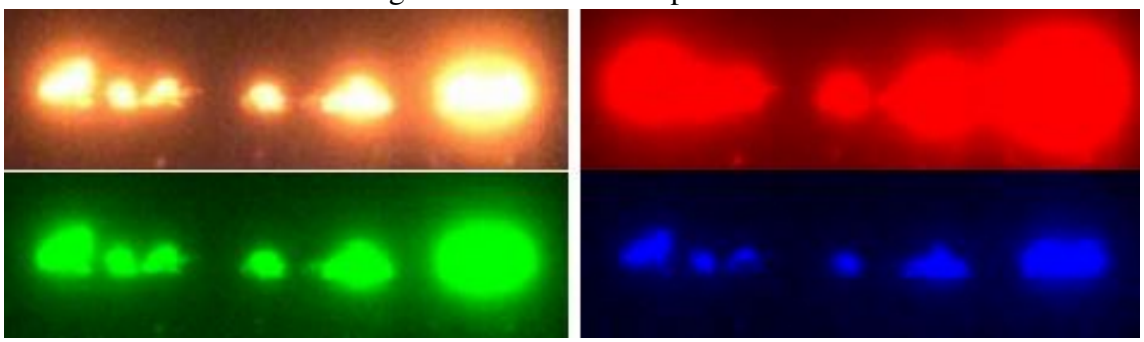
Resolution Type	Common Name	Aspect Ration	Pixels		Pixel Size
			Row	Column	
SD (Standard Definition)	480p	4:3	640	480	640x480
HD (High Definition)	720p	16:9	1280	720	1280x720
Full HD	1080p	16:9	1920	1080	1920x1080
QHD (Quad HD)	1440p	16:9	2560	1440	2560x1440
2K	1080p	1:1.77	2048	1080	2048x1080
UHD (Ultra HD)	4K	1:1.9	3840	2160	3840x2160
Full Ultra HD	8K	16:9	7680	4320	7680x4320

2.1.2 Pixels

Pixels were not created for videos, but for static images stored in a digital standard as they became reality before videos. If a screen of any device (e.g., television, cellphone, etc.) is analyzed very closely, one may notice the presence of tiny squares arranged in a matrix (i.e., with rows and columns). Each of the squares is called a pixel and, in modern devices, they can assume virtually any color found in the real world. Figure 2.4 presents an example of an image divided into several pixels.

One may notice, after analyzing Figure 2.4, that a combination of colors in pixels located within a given area creates an object (e.g., the head and face depicted in black and white in Figure 2.4). The number of pixels located in a row and in a column, when combined, create what is well-known as resolution of the images and videos. Table 2.1 associates the resolution type with its common name, along with the number of pixels in a row and in a column, and the combination of Row X Column.

Figure 2.5: RGB decomposition.



Source: (CHEN et al., 2017)

2.1.3 Colors

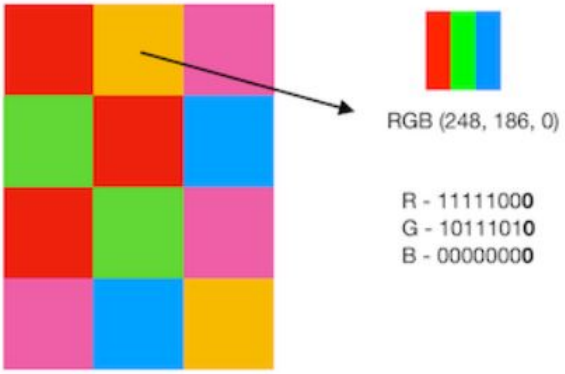
As previously mentioned, each pixel assumes a specified color in a given time, which allow the video to display objects and scenery according to the combination of colors in pixels located in a given area of the image. Each pixel's color definition arrives to a screen as code and can follow one of the following color spaces: RGB (Red Green Blue) or YCbCr (Luminance Chrominance Blue and Chrominance Red). The color spaces are explained in, respectively, 2.1.3.1 and 2.1.3.2.

2.1.3.1 RGB – Red Green Blue

The RGB color space, where RGB stands for Red Green Blue, is a color model broadly adopted in displays (YANG; YUHUA; ZHAOGUANG, 2007). As one may have guessed, after analyzing Figure 2.5, the RGB model uses nature's three fundamental colors (i.e., red, green and blue) to create any other color found in nature (NOWAK; KOŚCIELNIAK, 2019). In the depicted example, the original image (i.e., top-left corner) can be recreated by combining the three other images, which were created by taking into consideration only one of the color components each (i.e., bottom-left uses green, top-right uses red and bottom-right uses blue).

When working with digital images, each pixel receives an n-bit code for each color component of the RGB color model. The greater is the code's wirdth (i.e., n), the more precise are the colors presented by each pixel and the more colors are supported by the model. Figure 2.6 displays an arrangement comprising nine pixels, where pixel number 2 (i.e., top-middle) is being decomposed into RGB values 248, 186 and 0, respectively, using an 8-bit code for each color component, which totals a 24-bit code for the entire pixel.

Figure 2.6: RGB pixel decomposition.



Source: (GANGURDE; TIWARI, 2020).

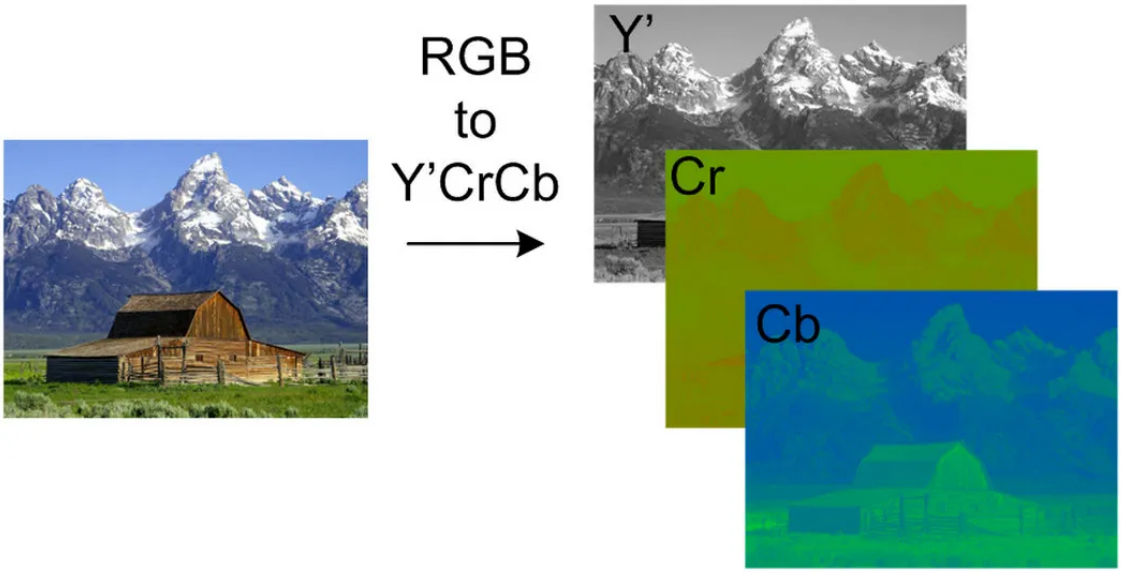
2.1.3.2 YCbCr – Luminance Chrominance Blue and Chrominance Red

The YCbCr, also known as YUV, is based on the Recommendation 601 (ITU-R, 2011), which specifies the Y as the luminance, and the Cb and Cr as, respectively, the blue-difference and red-difference chroma components (ROY; MAITI; GHOSH, 2015). The YCbCr model has each of its color components represented by an n-bit array, where the higher the n, the greater it will be the variety of colors supported (i.e., an 8-bit coding supports colors from 0 to 255 and, therefore, only 256 different colors can be expressed) (CHAI; BOUZERDOUM, 2000). Figure 2.7 depicts a YCbCr decomposition, where it is possible to identify each of the color components and how they impact on the original image on the left.

The YCbCr color space separates the RGB into luminance and chrominance, and is very useful in compression applications, even though it does not have intuitive color specifications (CHAI; BOUZERDOUM, 2000). As the YCbCr color model is broadly used in the video codecs and transmissions, and the RGB model is adopted in display, the conversion between them is an unavoidable task dealt by modern video systems (YANG; YUHUA; ZHAOGUANG, 2007).

Furthermore, the YCbCr allows for Chroma Subsampling, which is an encoding practice that allows the reduction of the chroma signals while keeping the luminance the same. The idea is that the human eye is more sensible to brightness than it is to color, which means that subsampling (i.e., reducing) the signals related to color would not affect the image quality as long as its luminance is kept the same (Xilinx Inc., 2021).

Figure 2.7: YCbCr decomposition.



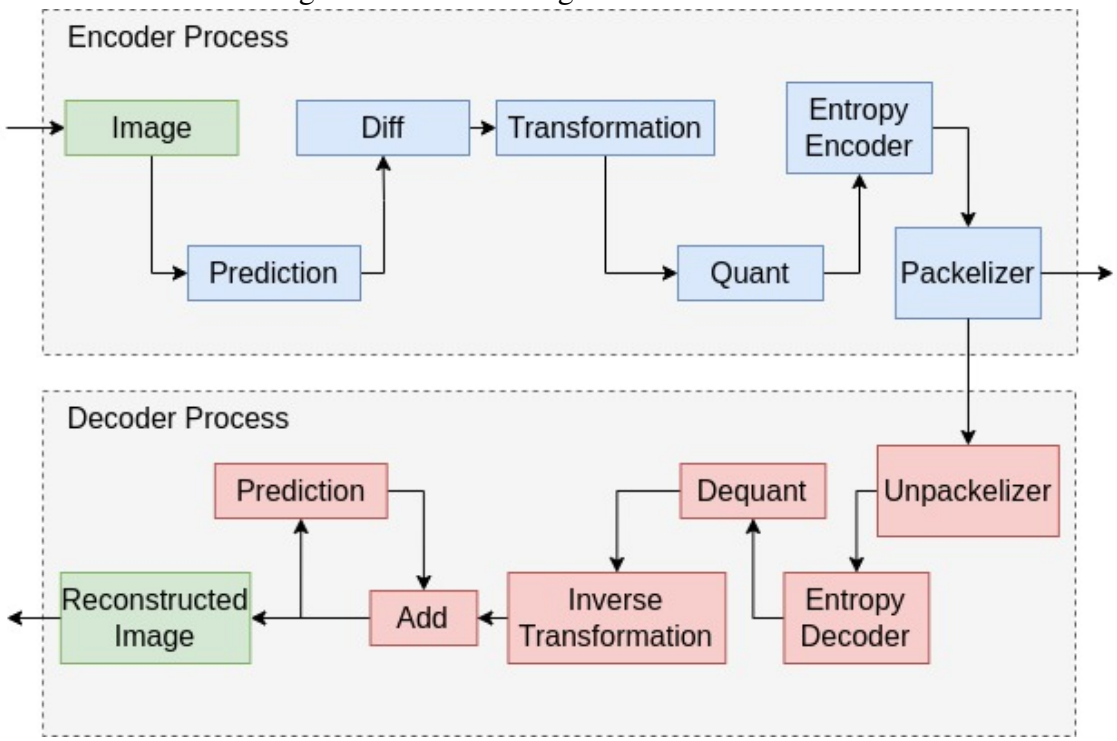
Source: https://www.hisour.com/ycbcr-color-spaces-26075/ (Accessed in June 25th 2022).

2.2 Video Coding

Videos are a challenging class of data to be transmitted and stored due to their huge sizes. This characteristic, as previously presented (i.e., in Chapter 1), is created by the combination of multiple frames depicted in each second, and the number of pixels within a single frame, which creates the video resolution. The precision of the color, which is defined by the number of bits used to encode a color in a single frame, also has a huge impact in a video size. As depicted in (2.1), the size of a raw digital video, S_{video}, in bytes, can be calculated by considering all the presented characteristics (i.e., F_{sec} is the number of frames-per-second, t is the time in seconds, P_{frame} is the number of pixels in a frame, and W_{pixel} is the width of the color code for a given pixel, in bits). Equation 2.2 defines W_{pixel} for a video that uses the RGB color model and, therefore, the width of the binary code that describes each of the three color components (i.e., W_{red}, W_{green} and W_{blue}) must be summated to create the complete color code (e.g., in a video where each RGB color component is defined by an 8-bit array, 24 bits are used to define the color for one pixel).

$$S_{video} = \frac{(F_{sec} \times t) \times (P_{frame} \times W_{pixel})}{8} \tag{2.1}$$

Figure 2.8: Blocks diagram of a video codec.



Source: (STAMENKOVIC et al., 2012).

$$W_{pixel} = W_{red} + W_{green} + W_{blue} \tag{2.2}$$

As the demand for videos is increasing, so is the necessity for effective video coding formats. One may use the example of a 60-fps 4K 10-minute long video using RGB colors to understand how costly it is to transmit and store a state-of-the-art video: around 895 Giga Bytes. Streaming companies, such as Netflix, YouTube, PrimeVideo, etc, have to deal with videos many times longer than 10 minutes and, perhaps, with more frames-per-second (i.e., 120-fps) and greater resolutions (e.g., 8K).

Figure 2.8 depicts the internal structure of a video codec on its encoder and decoder sides (i.e., top and bottom, respectively). As one may notice, the encoding process starts with the arrival of the image, which is followed by a prediction and a calculation of the difference to the original image. Then, the resulting residues pass through the transformation and quantization phases, to finally arrives at the entropy encoder. After the packing, the images is already encoded and ready to be transmitted or stored. The decoding process may start right away, or after some time, and it applied the invert set of process (i.e., starts with unpacking, then entropy decoder, dequantization, inverse transformation, to finally arrives at the loop between add and prediction where the image will be, in fact, reconstructed).

Figure 2.9: Spatial redundancy example.

Source:
http://lookingtothesky.com/2011/02/first-flight-to-salem-sle/blue-horizon/gallery

A general-purpose video codec is a specialized tool able to convert large amounts of video data into smaller pieces that can be more effectively transmitted through the Internet or stored locally. The main difference between a video codec and simple data compressors is that the former uses video-specific techniques capable of exploring distinct features of videos such as spatial redundancy, temporal redundancy, and entropic redundancy, whereas the latter usually relies only upon a powerful lossless algorithm. Each of the enumerated features (i.e., spatial redundancy, temporal redundancy, and entropic redundancy) are presented, respectively, in 2.2.1, 2.2.2, and 2.2.3.

2.2.1 Spatial Redundancy

The arrangements of pixels in an image, which was more broadly explored in Subsection 2.1.2, brings a more accurate depiction of the reality. However, the use of many pixels in an image sometimes might be redundant and may generate excessive amounts of unnecessary data to be stored.

A good example for spatial redundancy is an image showing a horizon such as in Figure 2.9, where half of the image presents the blue sky and the other half shows a scenary. One might expect to receive the color codes for each pixel from within the

figure and the corresponding pixels' coordinates in the figure's matrix. However, as more than half of the image has the same color (i.e., blue), an image compression tool would store the color code only once and combine it with the range of coordinates where the pixels with that color are located. Furthermore, slightly different colors might also be represented by manipulating the already stored color code (e.g., adding or subtract one) (TAGLIASACCHI et al., 2006).

2.2.2 Temporal Redundancy

When considering the same idea presented in Subsection 2.2.1, one might find feasible to apply it to a video. As a video is defined as an arrangement of images being presented quickly to give the viewer a impression of motion, spatial rendundancy can be applied for the images individually and for the video as a whole by combining the color codes and pixel coordinates for each image (i.e., frame) and creating a 3D coordinate, where the third number would represent the frame (i.e., or time it is presented) (TAGLIASACCHI et al., 2006). This process is called Temporal Redundancy and might be applied, for example, for a video show the scenary depicted in Figure 2.9.

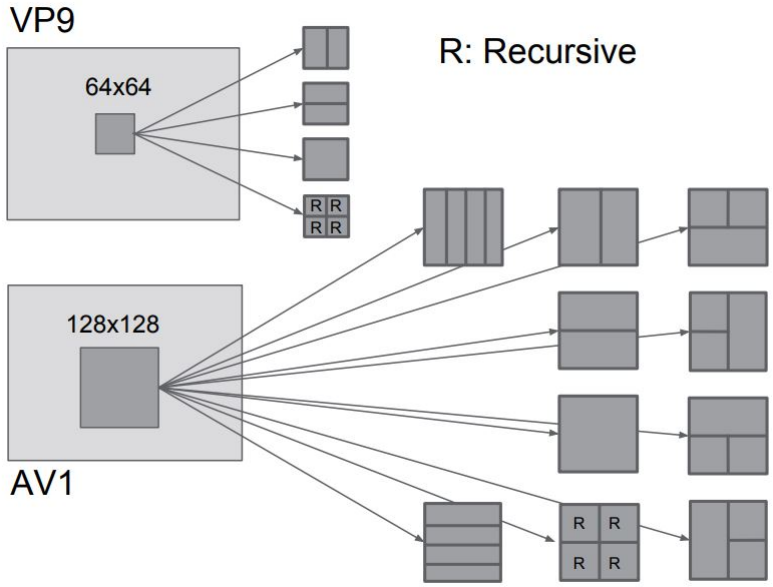
2.2.3 Entropic Redundancy

Entropic redundancy might be considered the final step in processing a video when using a video codec. As videos might present similar combinations of data on its extention (i.e., hereafter referred as symbols), one may use lossless algorithms to compress symbols based upon their frequency of appearence (BAMPIS et al., 2017). These lossless algorithms (e.g., Huffman coding (HUFFMAN, 1952)) are well-known in the academia and industry for being capable to considerably reducing the overall size of almost any kind of data, as long as it has repetitive symbols.

2.3 AV1 Codec

In late 2015 (i.e., September 2015), the Alliance for Open Media (AOMedia), headed by companies such as Amazon, Google, Netflix, Cisco and Microsoft (SHANKLAND, 2015), announced the development of a new video coding format: AV1. Based on

Figure 2.10: Comparison between the VP9 and AV1 partition trees.



Source: (CHEN et al., 2018).

previously released video codecs such as VP8 (FELLER et al., 2016), VP9 (GRANGE; RIVAZ; HUNT, 2016) and Daala (VALIN et al., 2016), AV1 was announced as a highly capable royalties-free video coding format able to encode videos in up to 8K@120 frames-per-second (fps) (Alliance for Open Media, 2020; CHEN et al., 2018). A few key features of the AV1 core include partitioning, intra prediction, inter prediction, transform coding and entropy coding (TROW, 2020).

One of the main advances introduced in AV1, when compared to VP9, is the different partition tree. VP9 comprises a 4-way partition tree, which starts in 64x64 and goes down to 4x4, whereas AV1 has a 10-way structure and also increases the largest size supported to 128x128 blocks, which is referred as a superblock (CHEN et al., 2018). Figure 2.10, which was initially introduced by Chen et al. (2018), depicts a brief comparison between the partition tree for VP9, at the top, and for AV1, at the bottom. Furthermore, as one may notice by analyzing Figure 2.10, AV1 also includes atomic (i.e., unable to be further divided) 4:1/1:4 rectangular partitions, and adds more flexibility for partitions below 8x8 (i.e., 2x2 partitions are now available).

Throughout all the other previously cited blocks, the AV1 development team worked to enhance capabilities, add different modes and optimize existing modules with a major goal of increasing the compression rates of the codec when compared to VP9. In the entropy coding specter, AV1 introduced a multi-symbol arithmetic coder, whereas VP9 used a tree-based boolean non-adaptive binary arithmetic encoder to encode all syntax elements (CHEN et al., 2018).

Therefore, as presented throughout this section, the AOMedia development teams focused its efforts on enhancing the AV1 capabilities by adding new functionalities (e.g., more partitioning possibilities) and by improving existing modules of the VP9. The VP9 codec provided the starting point for the AV1 development, as its skeleton was used as the basis for the novel AV1 codec.

In the next chapter (i.e., Chapter 3), the theory and examples of arithmetic coding, as well as the AV1 implementation of it, are presented. Related works that target arithmetic encoders and decoders are also introduced in Chapter 3.

3 ARITHMETIC CODING

The use of arithmetic coding (AC) within video coding formats has allowed developers to achieve impressive compression gains at a relatively low cost. State-of-the-art video coding formats such as VVC (ITU-T; ISO/IEC, 2020), AVS 3.0 (ZHANG et al., 2019) and AV1 (RIVAZ; HAUGHTON, 2019) use, respectively, the following three AC algorithms: CABAC (Context-Adaptive Binary Arithmetic Coding – (MARPE; SCHWARZ; WIEGAND, 2003)), CBAC (Context-based Binary Arithmetic Coding – (CHOI; LEE; CHAE, 2021)) and a royalties-free Multi-Alphabet AC (RIVAZ; HAUGHTON, 2019).

The AC algorithms are not, however, presented within the list of main blocks of the aforementioned video coding formats due to their usage being comprised within a more robust and complex block: the Entropy Coder (EC). Nevertheless, the AC is, in fact, the kernel of the entropy encoders of all of the cited standards and formats. A classical example of an EC is defined by Huffman (1952), which introduces a powerful and lossless data compression scheme that, among its many versions and implementations, compresses all symbols comprised within an array and generates a significantly less sizable coded array.

As will be explored throughout this chapter, a general-purpose AC aims to represent the most common symbol within an array (i.e., symbol with the largest probability) with less bits when compared to the least apparent symbol (refer to Section 3.1). In Section 3.2, the AV1 adaptation for the arithmetic encoding algorithm is presented. Section 3.3 presents and analyzes published works that tackle the arithmetic encoding complexity in other video coding formats, such as HEVC and AVS 2.0, and the arithmetic decoding complexity for the AV1 codec.

3.1 Basic Concepts of Arithmetic Coding

A general-purpose arithmetic coding (AC) algorithm is defined as a lossless codification process that converts a given set of symbols into a bitstreams (SAID, 2004). The overall goal of an AC algorithm is to represent the symbols of greater appearance probability by using less bits in the bitstream (sometimes even no bit). Furthermore, AC algorithms are usually close to the Shannon Optimal, which aims to reach an optimal choice of methods for coding and decoding information (DOBRUSHIN, 1961). This sec-

tion presents, within its Subsection 3.1.1, the encoding process of a general-purpose AC.

3.1.1 Encoder

A general-purpose arithmetic encoder relies mainly on four variables: $Range$, Low, symbol and probability. Equations 3.1 and 3.2 present the definition of the variables, respectively, $Range$ and Low.

$$Range_i = Range_{i-1} \times P_{\text{symbol, probability}} \tag{3.1}$$

$$Low_i = Low_{i-1} + (Range_{i-1} - C_{\text{range, symbol, probability}}) \tag{3.2}$$

As one may notice, the definition of $Range_i$, in (3.1), relies upon the usage of $Range_{i-1}$ (i.e., the last $Range$ value, defined in the previous round), and $P_{\text{symbol, nsyms}}$, which is defined as a general-purpose probability function that takes into consideration the encoding symbol and this symbol's probability. The Low_i definition in (3.2) follows the same logic of (3.1) and presents function $C_{\text{range, symbol, probability}}$, which is defined as a summation of all probabilities for the symbols that appeared before the current symbol.

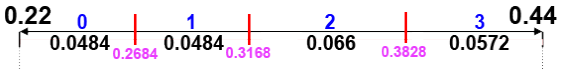
Arithmetic encoding is a tricky subject that relies on a well-defined encoding methodology based on the probability for a given encoding symbol. Although (3.1) and (3.2) already depicted the definition of the main variables of a general-purpose arithmetic encoder, there is no better way to explain a complex process than through an example, which is exactly what is depicted within Figure 3.1.

As one may notice, Figure 3.1 presents the three main phases of an arithmetic encoding process: Figure 3.1a is the initial bar found at the beginning of the process (i.e., when no symbol was encoded yet), Figure 3.1b represents the new bar created when symbol '1' is encoded and before the renormalization process (i.e., just after symbol '1' arrived), and Figure 3.1c represents the final bar (i.e., when the codification of '1' is completed). The relations between the aforementioned variables and the bar are as follows:

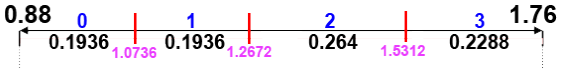
Low represents the left-most value (i.e., zero in Figure 3.1a) and is used to generate the bitstreams when a renormalization of $Range$ is needed. The Low variable represents the base for the interval and, when combine to $Range$, creates the bar presented in Figure 3.1.

Figure 3.1: Arithmetic encoding process step-by-step.
(a) Phase 1: initial bar.

(b) Phase 2: new bar prior to renormalization after receiving Symbol 1.



(c) Phase 3: final bar.



Source: (SAID, 2004).

Range is combined with Low to create the bar from Figure 3.1 (i.e., as $Range = 1$ in Figure 3.1a, then $Low + Range = 1$). Furthermore, $Range$ represents the summation of all symbols' probabilities at any given point of the codification process, and is used to define if whether a renormalization process is needed or not.

Symbols are represented by the numbers 0, 1, 2 and 3 above the bar. In Figure 3.1, the symbols are depicted as labels for each probability interval. In any arithmetic coding algorithm, the number of symbols is defined by the length of the alphabet (i.e., 4-symbol alphabet in Figure 3.1). Moreover, symbols are considered the "encoding items" for any arithmetic coding algorithm.

Probabilities are the numbers below the bar and represent the probability of appearance of each symbol. The probabilities might be adaptive, and change according to the appearance of the symbols, or fixed.

For example, if symbol number one is being encoded, as depicted in Figure 3.1, then the new bar (i.e., 3.1b) is created according to the symbol's probability when the '1' arrives as input for the arithmetic encoder using (3.1) and (3.2). The idea is to use the sub-bar within the red vertical dividers corresponding to symbol '1' (i.e., the probabilistic limits for symbol '1', which represents a 22% – 0.22 – probability) and create the new bar. As one may notice, the new bar also comprises all other symbols and their adjusted probabilities, which ensures that the correlation between the summation of all symbols

probabilities and $Range$ is kept according to (3.3).

$$\sum_{i=0}^{3} P_i = Range \tag{3.3}$$

There is, however, a rule that must be followed to achieve the final bar: $0.5 \leq Range \leq 1.0$. In order to accomplish this rule, the renormalization process must multiply both Low and $Range$ by 2 (i.e., a left-shift in computational solutions) concurrently, so their values are equally incremented. If Low eventually goes above one (i.e., $Low \geq 1.0$) after the multiplication, a bitstream '1' is generated and 1 is subtracted from the Low value. Otherwise, a bitstream '0' is generated and the value of Low is kept as the multiplication result.

Algorithm 1, which is an adaptation from the algorithm presented by Said (2004), depicts the behavior of the renormalization procedure of a general-purpose Arithmetic Encoder. As one may notice, loop keeps multiplying $Range$ by two until its value reaches the required interval (i.e., $0.5 \leq Range \leq 1.0$). Meanwhile, the Low variable generates a bit '1' if it goes about 1.0, or '0' if it does not.

Algorithm 1: Encoder renormalization process adapted from (SAID, 2004).

```
Data: Low, Range
Result: d[t]
{Renormalization loop};
while Range ≤ 0.5 do
    t ← t + 1; {Increment bit counter and scale interval length};
    Range ← 2 × Range;
    if Low ≥ 0.5 then
        {Test most significant bit of interval base};
        d_t ← 1; {Output bit 1};
        Low ← 2 × (Low0.5); {shift and scale interval base};
    else
        d_t ← 0; {Output bit 0};
        Low ← 2 × Low; {scale interval base};
    end
end
```

After executing two multiplications upon the $Range$ and Low values defined in the bar from Figure 3.1b, it is possible to achieve the targeted $Range$, which is displayed in Figure 3.1c. The outcome from this execution is the bitstream '00', as the Low value was never greater than one after the operations.

3.2 AV1 Arithmetic Coding

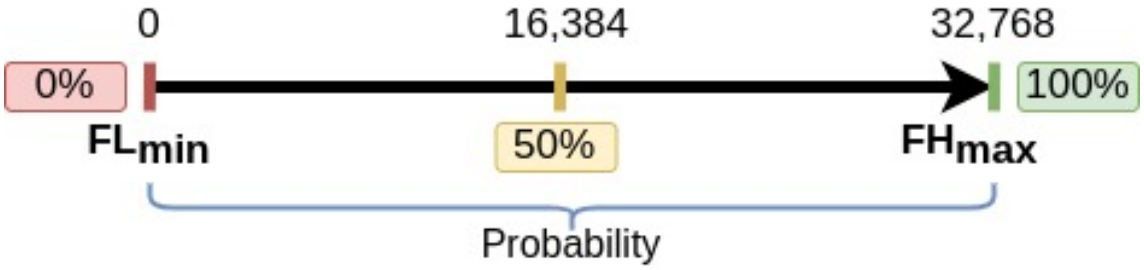
The AV1 arithmetic encoder, which is part of the entropy encoding block of the AV1 codec, is defined as a lossless multi-symbol multi-alphabet compressing algorithm (HAN et al., 2020; CHEN et al., 2018). It is responsible for converting the thus far generated syntax elements (i.e., the results from previous modules in the AV1 execution line) into less sizable bitstreams (SAID, 2004).

The AV1 arithmetic encoder operates under two different modes: CDF (Cumulative Distribution Function) and Boolean. The former is comprised of multi-symbol alphabets in which the number of symbols is defined by the incoming $nsyms$ variable. The lowest possible value for $nsyms$ is two, whereas the maximum is 16. The *CDF Operation* also differs from the *Boolean Operation* due to its adaptive probabilities for each of the possible symbols within a given alphabet, where a symbol $symbol$ must be comprised within the $0 \leq symbol < nsyms$ interval.

The latter mode, the *Boolean Operation*, is composed of a 2-symbol (i.e., '0' and '1') alphabet, where the probability for both symbols are equally set, and fixed, to 50%. When one considers that two 16-bit signals (i.e., FL and FH) are, together, responsible for representing the probability range for the encoding symbol, and that the 100% of probability is defined by the decimal number 32768, which is also the initial $Range$, one may conclude that 50% is, therefore, represented by the decimal number 16384. Furthermore, when considering a 2-symbol alphabet, previous modules of the AV1, which generate the syntax elements to be encoded, are responsible for choosing whether to use *CDF Operation* or *Boolean Operation* for the arithmetic encoding step (RIVAZ; HAUGHTON, 2019). This is chosen according to the type of syntax element that is being encoded, and not by the number of symbols in the alphabet (i.e., $nsyms$ is an input signal only for the *CDF Operation* and is not comprised within the *Boolean Operation*).

Figure 3.2 depicts the relation between variables FL and FH, and the 50% probability given for the boolean symbols, which are processed by the *Boolean Operation*. As one may notice, the minimum value for FL is defined as zero, whereas the maximum value for FH is defined as 32,768 (i.e., 16-bit array where MSB is set to '1'). The difference between FL and FH defines the probability for a given symbol (i.e., the entire arrow depicted).

Figure 3.2: Representation of 50% of probability based upon variables FL and FH.



Source: The Author (2022).

Table 3.1: Summary of all results for the related works that target arithmetic encoders and decoders (i.e., (GOMES; RAMOS, 2021; GOMES et al., 2022; ZHOU et al., 2015; PASTUSZAK, 2020a; RAMOS et al., 2021; CHOI; LEE; CHAE, 2021)).

	Video Codec	**Coder**	**Tech. (nm)**	**Freq. (MHz)**	**Gates Count**	$\frac{bins}{cycle}$	$\frac{Gbins}{sec}$	**Power (mW)**	**MRTR**[1]
□	AV1	AD[2]	65[3]	462	31.3 K	1.62	0.748	26.8	8K@60
◯	AV1	AD[2]	65[3]	473	35 K	1.62	0.766	14.3	8K@60
△	HEVC	CABAC	90[4]	420	64.1 K	4.37	1.836	-	8K@120
†	HEVC	CABAC	90[4]	700	120.4 K	9.65	6.755	111.7	8K@120
♣	HEVC	CABAC	65[3]	507	21.22 K	4.31	2.185	26.18	8K@120
⋆	AVS 2.0	CBAC	65[4]	735	133.5 K	4.78	3.51	79.6	8K

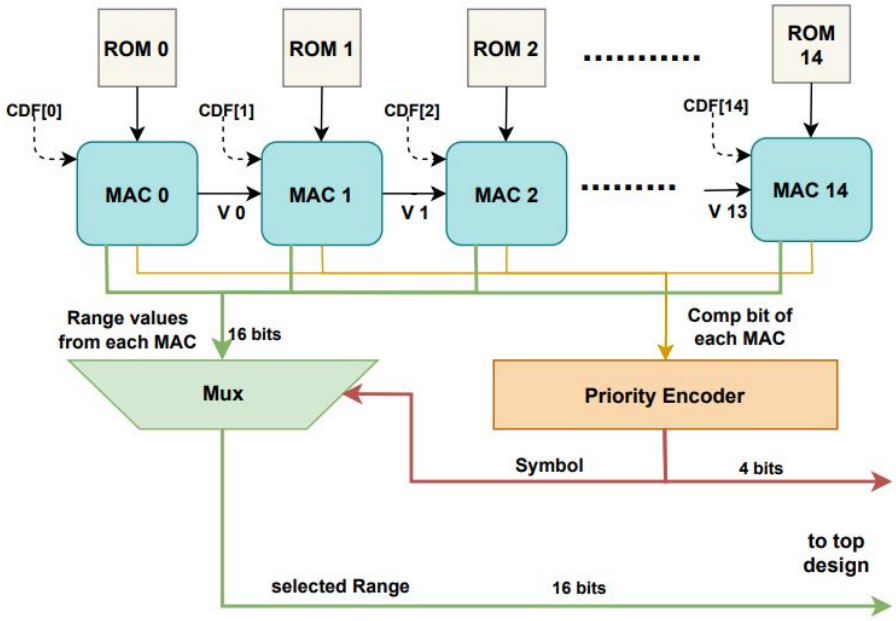
[1] Maximum Real-Time Resolution
[2] Arithmetic Decoder
[3] ST PDKs
[4] TSMC PDKs

□ (GOMES; RAMOS, 2021)
◯ (GOMES et al., 2022)
△ (ZHOU et al., 2015)
† (PASTUSZAK, 2020a)
♣ (RAMOS et al., 2021)
⋆ (CHOI; LEE; CHAE, 2021)

3.3 Related Works About Arithmetic Coding

On the time this section was being written, the literature was still lacking works on the AV1 arithmetic encoder, aside from the proposals presented within Chapters 4 and 5. This section analyzes the works by Gomes e Ramos (2021) and Gomes et al. (2022), which introduced a hardware designs to accelerate the AV1 arithmetic decoder, in Subsection 3.3.1, by Zhou et al. (2015), Pastuszak (2020a) and Ramos et al. (2021), which presented works related to the HEVC CABAC, in Subsection 3.3.2, and, finally, by Choi, Lee e Chae (2021), which targeted the AVS 2.0 CBAC, in Subsection 3.3.3. Table 3.1 presents a summary of all results obtained by the related works introduced and analyzed in this section.

Figure 3.3: DecCDF module proposed by Gomes e Ramos (2021).



Source: Gomes e Ramos (2021).

3.3.1 AV1 Arithmetic Decoders

Although the literature still lacks works targeting the AV1 arithmetic encoder, Gomes e Ramos (2021) and Gomes et al. (2022) proposed approaches to accelerate the AV1 arithmetic decoder side using hardware designs. The former work introduced the MaAD (i.e., Multi-alphabet Arithmetic Decoder) design, which was characterized by the author as mono-cycle architecture that was divided into two modules named DecCDF (i.e., responsible for multi-symbol codification and related to the AV1 CDF presented in Section 3.2), and DecBool (i.e., responsible for binary symbols codification) (GOMES; RAMOS, 2021).

The work by Gomes e Ramos (2021) aims to maximize the use of parallelism techniques as it explores a trial and error approach to decode symbols. As one may notice, after analyzing Figure 3.3, out of the 16 possible symbols available in the AV1 CDF codification side, the DecCDF block uses 15 parallel MACs (i.e., Multi-Alphabet Cores) to decode the symbols. For instance, if the first MAC does not recognize the decoding symbol, the symbol is surely not 0, and if none of the MACs is able to decode the symbol, then the symbol must only be the maximum value possible (i.e., 15). Therefore, by processing 15 different analysis at the same time, and by prioritizing the MACs in ascend order (i.e., from 0 to 14), the proposed design can more effectively decode a given symbol by pointing out which MAC comprises the right value at a given round.

The design proposed by Gomes et al. (2022), which is in line with (GOMES; RAMOS, 2021), introduces the addition of low-power techniques to reduce the power consumption of the architecture. More specifically, Gomes et al. (2022) applied Operand Isolation to MACs 5 through 14, which reduced the power consumption by, on average, 46.69% when compared to the original work (i.e., (GOMES; RAMOS, 2021)).

With regards to the results, both (GOMES; RAMOS, 2021) and (GOMES et al., 2022) were capable of achieving enough throughput rates for real-time decoding of 8K at 60 frames-per-second (fps) videos. Concerning frequency and area, Gomes e Ramos (2021) accomplished, respectively, 467 MHz and 34.3 K gates count, whereas Gomes et al. (2022) reached 478 MHz and 35.5 K gates count. Both works were synthesized to a 65nm technology

3.3.2 HEVC CABAC works

The works (ZHOU et al., 2015), (PASTUSZAK, 2020a) and (RAMOS et al., 2021) proposed architectural solutions to accelerate the HEVC CABAC (ITU-T; ISO/IEC, 2013) encoder. The work by Zhou et al. (2015) aimed ultra-high-throughput for UHDTV applications and proposed improvements on prenormalization, hybrid path coverage and lookahead rLPS with the goal of reducing the critical path delay of BAE (Binary Arithmetic Encoding).

Furthermore, Zhou et al. (2015) proposed the use of a pipeline arrangement, where the $Range$ and Low updating processes are located in two different stages. This methodology is discussed in more details in Chapter 4, as this is a critical designing feature used in the circuits proposed in this thesis.

Pastuszak (2020a) proposed a multi-symbol architecture of the entropy coder of the HEVC video coding standard. The design created by Pastuszak (2020a) works by processing the syntactic elements with the least probability of appearance, as their results are pre-calculated and stored in a Look-up Table (LUT), before processing the other elements. By using this approach, the (PASTUSZAK, 2020a) work was able to accomplish the greatest throughput rates, but is comprised within the largest area among the other related works.

Ramos et al. (2021) introduced a novel configurable high-throughput BAE design, which was named ET-BAE and relies upon a combination of a modified Multiple-Bypass Bins Scheme (MBBS) and upon a power-saving approach with a two-mode configura-

tion. Ramos et al. (2021) applied the low-power techniques clock gating and operand isolation, which were also applied in the novel designs introduced in this thesis. Therefore, this works targets low-power and high-throughput capabilities as critical features for the proposed architecture.

Regarding the results, all three works achieved enough throughput rates for 8K real-time video coding. Furthermore, the works presented frequency and area of, respectively, 420 MHz and 64.1 K gates count in 90nm technology for (ZHOU et al., 2015), 570 MHz and 273 K gates count in 90nm technology for (PASTUSZAK, 2020a), and 507 MHz and 21.22 K gates count in 65nm technology for (RAMOS et al., 2021).

3.3.3 AVS 2.0 CBAC work

The work developed by Choi, Lee e Chae (2021) targets an acceleration on the CBAC (Context-based Binary Arithmetic Coding) of the novel AVS 2.0 codec. The proposal is based on a bin-merging technique that converts a sub-sequence of bins into a merged bin without affecting the functionality of the CBAC encoder. The design comprises, according to the author, a 4-stage pipeline and lookup tables for context updates.

Choi, Lee e Chae (2021) was able to accomplish 735 MHz of frequency and 182.5 K gates count of area in a TSMC 65nm technology. Furthermore, the achieved throughput rate is high-enough to accomplish real-time encoding of an 8K video.

Therefore, the works presented in Subsections 3.3.1 and 3.3.2 achieved similar results when compared to (CHOI; LEE; CHAE, 2021). Although it is difficult to make a fair comparison when analyzing three distinct video codecs, all presented works were able to accomplish real-time coding capabilities of at least 8K videos.

Chapter 4 introduces the baseline proposal, which received the name AE-AV1. The methodology used to develop AE-AV1, and important designing decisions are also highlighted.

4 AV1 ARITHMETIC ENCODER HARDWARE DESIGN

The designing process of any hardware architecture is done by following a carefully defined methodology that provides the most certainty that the final product will be functional and work as expected. This chapter aims to go through the entire process of designing a hardware architecture from scratch to accelerate the AV1 arithmetic encoder.

The baseline architecture, which was named AE-AV1, is defined as a high-throughput design capable of executing, in real-time, videos with 8K resolution at 120 frames-per-second (fps). Throughout this chapter, the methodology used to design the architecture (Section 4.1), the hardware design created (Section 4.2) and the verification methodology (Section B) are presented. Furthermore, as additional reading for this chapter, the Appedixes A and B present an in-depth analysis on how the AV1 reference software works and how was the verification methodology applied to ensure the right functionality of the designs introduced.

4.1 Methodology for the Arithmetic Encoding Design

The AV1, as already described in previous chapters, is a well-defined coding format very capable to encode videos. However, one of its drawbacks is the fact that it uses lots of very complex and computationally demanding techniques that require (i) long studies and analysis to understand, and (ii) a very detail-oriented implementation to cover all the encoder's features.

The AOMedia, as a proof-of-concept, developed the AV1 reference software (Alliance for Open Media, 2020) comprising all the necessary features for the effective codification of videos using the codec. Hence, as the first step towards proposing a novel and enhanced technology is to acquire a good understanding of the problem, the AV1 reference software, combined with the AV1 specification (RIVAZ; HAUGHTON, 2019) also created by the AOMedia are seen as the starting point.

By analyzing the AV1 reference software and combining the features observed in the code with the AV1 specification, one shall be able to more accurately understand important details of the codec. Moreover, executing slightly modified versions of the AV1 reference software allows for a further understanding on how some changes might affect the overall codification process. The entire analysis upon the AV1 reference software is presented in Appendix A.

As this section's goal is to give an overall description of the designing process, one shall consider the analysis of the AV1 reference software and AV1 specification as the first step. The second step is defined by the acquisition of data from internal parts of the AV1 AV1 reference software, which was accomplished by writing values from internal variables into CSV (Comma-Separated Values) files. Besides serving as a baseline tool to be using during the designing process, these files were also used in the Verification Phase, which is out of the scope of this thesis, but is presented in detail in Appendix B.

Therefore, by studying the AV1 reference software and the AV1 specification, as well as extracting internal data from the AV1 AV1 reference software, one shall be able to acquire enough knowledge about the codec's functionality. The next step, which is deeply described in Section 4.2, is to design the de facto RTL (Register Transfer Level) circuit using the Verilog HDL (Hardware Description Language).

4.2 AE-AV1 Design

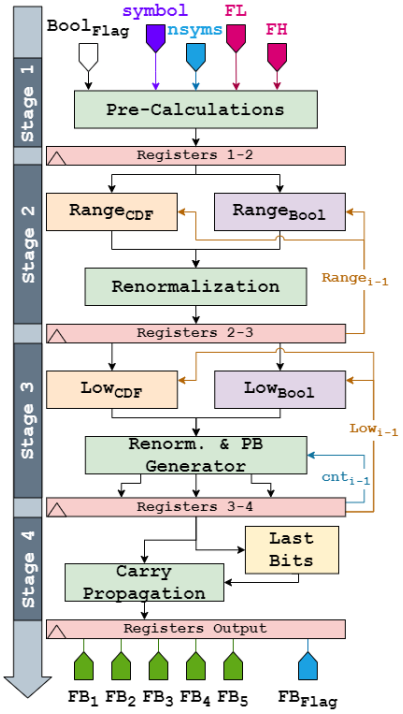
This section introduces the herein proposed AE-AV1 architecture, which is a high-throughput baseline design for the AV1 arithmetic encoder. This section's organization is as follows: introduction of the architecture arrangement in Subsection 4.2.1, the pipeline stages in Subsection 4.2.2 and a brief presentation of essential decisions that impacted the architecture's development in Subsection 4.2.3.

4.2.1 AE-AV1 Top-Entity

The AE-AV1, which is considered to be the baseline and first version of the AV1 Arithmetic Encoder architecture, is comprised of four stages of pipeline. This number was achieved after considering the minimum block division without creating bubbles on the pipeline and after taking into consideration the works by (ZHOU et al., 2015), (RAMOS et al., 2021) and (RAMOS et al., 2018), which target the HEVC CABAC and use the same pipeline structure since the update of the main variables follow a similar flow, no matter which arithmetic encoder proposal is being used.

Figure 4.1 briefly presents the top-level of the architecture and the connections between the stages. As one may notice, the variables Low, $Range$ and cnt are re-fed to the stages that generate them. This re-feeding is necessary because these variables are

Figure 4.1: Overview of the AE-AV1 pipeline stages (BITENCOURT; RAMOS; BAMPI, 2022a).



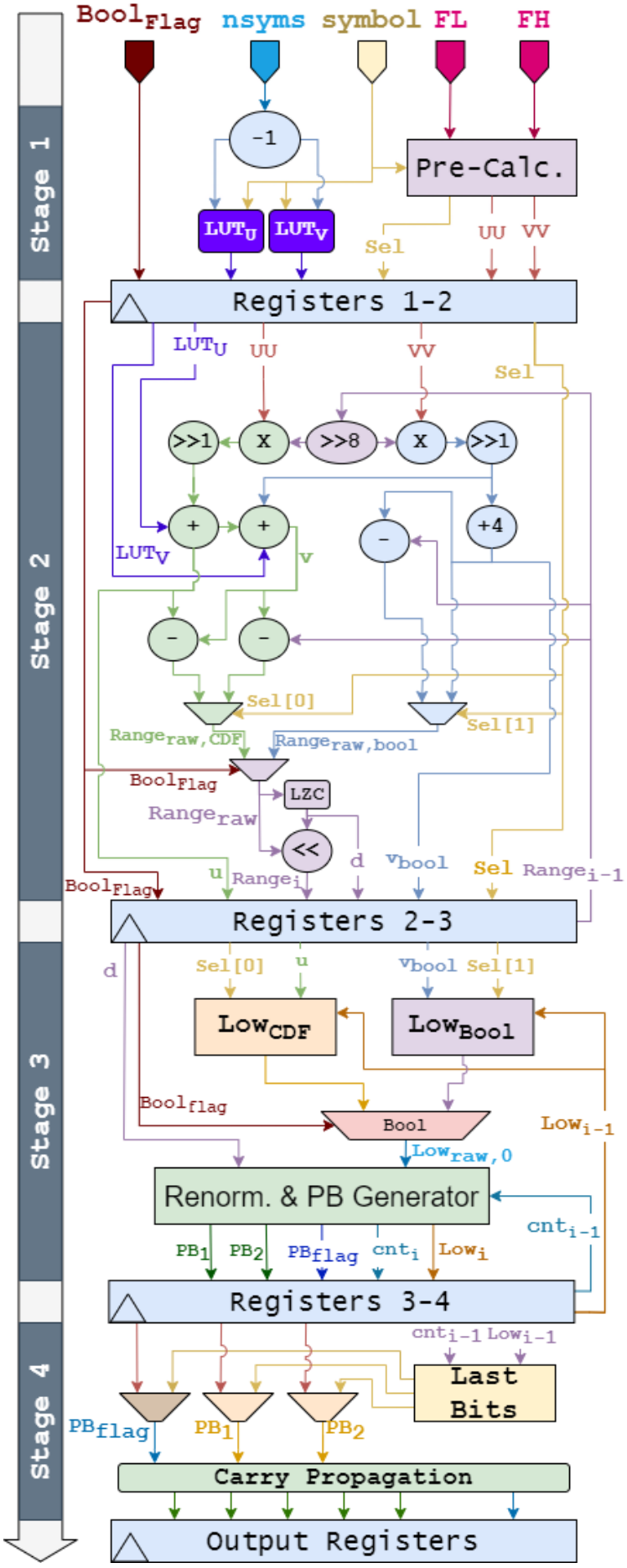
Source: The Author (2022).

directly dependent on their previous values, as they are in constant evolution throughout the execution.

The variables Low, $Range$ and cnt, therefore, are responsible for preventing the architecture from being broken into more pipeline stages. An uncareful breaking into more than four stages would create bubbles due to Stage 2 ($Range$ updating process) reliance on a previously generated range (i.e., $Range_{i-1}$) for the $Range_i$ definition. Although the split would significantly increase the architecture's frequency, the throughput rate would drop to half due to the inability of the architecture to update $Range$ in only one round and, therefore, external blocks would have to set inputs in every other round.

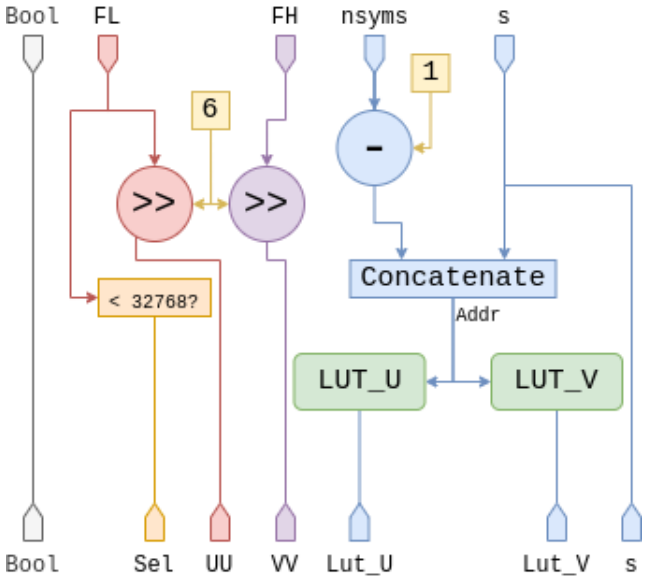
Figure 4.2 provides a more in-depth view of the architecture. As one may notice, Stages 2 and 3 are mainly composed by two blocks: *CDF Operation* and *Boolean Operation*, whereas Stage 4 is comprised by *Final Bits* and *Carry Propagation* (each block is independently described in Subsection 4.2.2).

Figure 4.2: In-depth view of the AE-AV1 design.



Source: The Author (2022).

Figure 4.3: Stage 1 design of AE-AV1.



Source: The Author (2022).

4.2.2 Pipeline Stages

4.2.2.1 Stage 1

Stage 1 of the AE-AV1 is mainly responsible for pre-calculations. The primary idea behind the Stage 1's creation is the fact that, as presented in 4.2.2.2, Stage 2 comprises the critical path for the entire architecture. Hence, the necessity to reduce the delay created by the critical path served as inspiration to create Stage 1, which is composed of operations from the $Range$ generation that could be accomplished without the use of $Range_{i-1}$.

As previously presented, and defined by Figures 4.1 and 4.2, the architecture inputs are: $bool\_flag$, FL, FH, $nsyms$ and $symbol$. As an additional information, which is also presented within Figure 4.2, the architecture receives $flag\_last$, which helps any external circuitry to learn the exact time when the final symbol is encoded and the *Final Bits* block already generated the last bitstreams. Other control variables, which are not depicted in Figure 4.2, are a clock and a reset signals that are propagated throughout the design.

Stage 1 comprises two look-up tables (LUTs), which are shown in Figure 4.3. Each LUT stores all possible results for (4.1) and (4.2), where $symbol$ is a 4-bit array and $nsyms$ is a 5-bit array with its maximum number set to 16 and suffering, upon its arrival on the architecture, a subtraction by one. Therefore, as the maximum value used

for $nsyms$ throughout the architecture is 15, $nsyms$ should be considered to be a 4-bit array, which allows the possibility of addressing the LUTs with an 8-bit array defined by the concatenation of $nsyms$ and $symbol$, and resulting in a 256-position memory.

$$LUT_U = 4 \times [(nsyms - 1) - (symbol - 1)] \tag{4.1}$$

$$LUT_V = 4 \times [(nsyms - 1) - (symbol + 0)] \tag{4.2}$$

Furthermore, Stage 1 also executes two right-shift operations upon the arrival of the FL and FH variables as shown in (4.3) and (4.4). The shifts calculated are by six on both FL and FH, which reduces the width of the signals and allows for more efficient multiplication in Stage 2 (see 4.2.2.2). The last and simplest functionality of Stage 1 is to propagate the $Bool\_Flag$, which defines whether to use *CDF Operation* or *Boolean Operation* in the upcoming Stages 2 and 3.

$$UU = FL \gg 6 \tag{4.3}$$

$$VV = FH \gg 6 \tag{4.4}$$

4.2.2.2 Stage 2

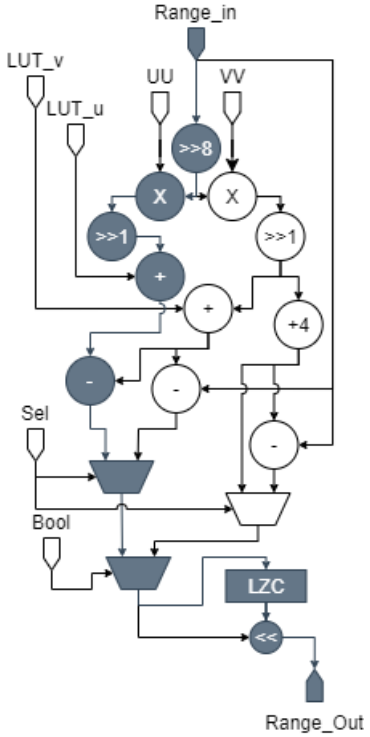
Once the results from Stage 1 are safely stored in the registers connected to Stage 2 inputs, the latter starts the $Range$ updating process. This process comprises the completion of the already-started (and presented) Equations (A.1) and (A.2), which were started by (4.1), (4.2), (4.3) and (4.4). Therefore, until this point, stage 1 already calculated the values for the signals UU, VV, LUT_U and LUT_V.

The Stage 2 has the responsibility to take the previously generated $Range$ (i.e., $Range_{i-1}$), multiply it by each UU and VV, and add LUT_U and LUT_V to the result. All these operations will generate the variables named u and v, which are represented by Equations (4.5) and (4.6), respectively.

$$u = [(Range_{i-1} \gg 8) \times UU \gg 1] + LUT_U \tag{4.5}$$

$$v = [(Range_{i-1} \gg 8) \times VV \gg 1] + LUT_V \tag{4.6}$$

Figure 4.4: Primary design of Stage 2 (BITENCOURT; RAMOS; BAMPI, 2021).



Source: The Author (2022).

Figure 4.4 depicts the main operations within Stage 2. One may notice that the figure highlights the critical path of the architecture in gray color, which is also the *CDF Operation* block of Stage 2.

The *CDF Operation*, which roughly defines half of the Stage 2 circuitry according to Figure 4.1 (i.e., in area, *CDF Operation* is around 3 to 4 times larger than *Boolean Operation*), comprises two 8-bit by 10-bit multiplications. Moreover, it is possible to notice the combination of the previously defined variables u and v with a multiplexer in which the selector is defined by the logical equation $FL < 32768$ to generate Sel.

The previously defined variables u and v are here combined to generate the so-called $Range_{raw}$ ($Range$ variable before the renormalization process). As presented by (4.7), the *CDF Operation* block uses the already generated Sel variable, which is created by Stage 1, to decide which sub=equation to use as $Range$ generation.

$$Range_{raw} = \begin{cases} u - v, & \text{if Sel} = \text{'1'} \\ Range_{i-1} - v, & \text{otherwise} \end{cases} \tag{4.7}$$

On the other hand, the *Boolean Operation* in Stage 2 is mainly defined by the combination between the variables $Range_{i-1}$ and v, as already shown in Figure 4.4. The already presented (4.6) is also valid for the *Boolean Operation*, as the fixed and single

probability defined for the *Boolean Operation* was configured to occupy the FH signal due to the similarities between the *CDF Operation*'s v equation and the *Boolean Operation* original equation.

However, as the probability for the *Boolean Operation* is fixed to 50%, which in the scale used by the AV1 reference software (i.e., 0 through 32768) and defined by Rivaz e Haughton (2019) represents 16,384, it is possible to assume a simplification for the equation. This simplification is presented in Equation 4.8, which defines the entire *Boolean Operation* $Range$ updating process. Moreover, this simplification allowed the so-called AE-AV1-MB version of the architecture, which is presented in Section 5.2.

$$Range_{raw} = \begin{cases} Range_{i-1} - [(Range_{i-1} \gg 8) \ll 7] + 4, & \textit{if symbol} = \text{'1'} \\ Range_{i-1}, & \text{otherwise} \end{cases} \tag{4.8}$$

The renormalization process is one of the most important parts of the arithmetic encoding process. According to the AV1 reference software, the renormalization process is responsible for the pre-bitstream generation and for ensuring that $Range_i$ ($Range$ final for round i) respects the $0.5 \leq Range \leq 1.0$ rule.

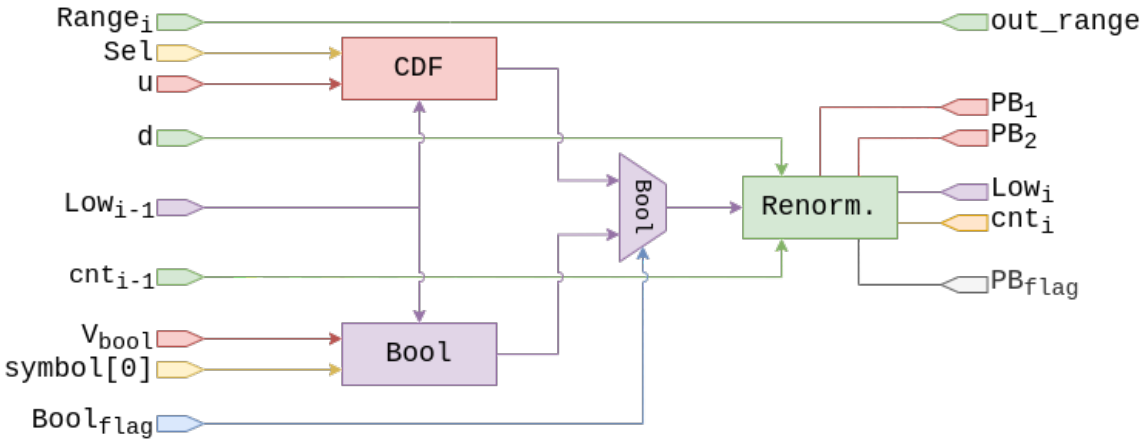
As previously mentioned, the $Range$ within the AV1 reference software is defined by a 16-bit array, where its maximum value is 65,535 in decimal. Hence, 50% of the maximum value of $Range$ is represented by 32768, which in binary is indicated by only the most significant bit (MSB) of a 16-bit array set to '1'.

Therefore, the critical goal of the renormalization process is to ensure that $Range$ MSB is set to one following a set of multiplications by two (or left-shifts). In order to accomplish that, the architecture was designed to use a low-power and high-speed Leading Zero Counter (LZC) proposed by Dimitrakopoulos et al. (2008), which was later swapped by Miao e Li (2018) proposal due to its reduced delay. This circuitry aims to define a value that represents the number of zeros before the first bit '1' in a 16-bit array (i.e., it defines the so-called d variable).

After establishing the correct number of zeros before the first bit '1' in $Range_{raw}$, the architecture executes a left-shift of $Range_{raw}$ by d to generate the $Range_i$. The result from the LZC circuitry is also used as an output to Stage 2, as Stage 3 relies on its value for the Low renormalization process and pre-bitstream generation.

The LZC circuitry is a fundamental part of the architecture and directly impacts the architecture's frequency, as its execution follows the multiplication used during the

Figure 4.5: Top-entity schematic for Stage 3.



Source: The Author (2022).

$Range$ updating process. Hence, it is crucial to find a good and effective LZC proposal capable of executing this task effectively. According to several analysis and tests, the LZC circuit proposed by Miao e Li (2018) was chosen, as it accomplishes a better result than the work by Dimitrakopoulos et al. (2008).

4.2.2.3 Stage 3

Stage 3 is mainly defined by its responsibility to update the Low variable. This variable, similarly to the $Range$ variable, is defined by two different operations, where each compose the *CDF* and *Boolean* blocks of Stage 3.

The structure of the herein presented stage is depicted in Figure 4.5, where one may notice the presence of each of the mentioned blocks. Furthermore, as previously explained in 4.2.2.2, Stage 3 does not comprise any leading zero counter (LZC) circuitry as it relies entirely on Stage 2 LZC for the Low renormalization process. It is possible to observe, in Figure 4.5, the definition of Low_{raw}, which represents the Low variable prior to its renormalization and suffers a left-shift by d to reach its final value (i.e., Low_i, where i identifies the current round of execution).

Stage 3 is, however, different from previous stages. Its main difference falls upon the fact that it is responsible for generating the pre-bitstream values every time Low goes above a certain number. As one may remember from Chapter 3, Algorithm 1, during the renormalization process, several multiplications by 2 (left-shifts) are executed upon the $Range$ and Low variables, and every time Low goes above 1.0, it suffers a subtraction by 1, which generates a bitstream.

Stage 3, as expected, follows the same logic using more efficient and hardware-friendly techniques. In order to accomplish the same functionality, Stage 3 keeps track of a counter, the so-called cnt variable, which is combined to the incoming d (otherwise known as the number of multiplications by two needed, or the LZC result), to decide whether to reduce Low or not. Therefore, Paragraphs 4.2.2.3.1 and 4.2.2.3.2 detail the process to generate the first Low (i.e., $Low_{initial}$), whereas everything from Paragraph 4.2.2.3.3 to the next subsection presents the definition of Low_{raw} and the pre-bitstreams generation.

4.2.2.3.1 The *CDF Operation* for the Low updating process uses the u variable defined by Stage 2, which is associated with the incoming $Range_{i-1}$ and Low_{i-1}, according to (4.9). As one may notice, (4.9) defines a combination of two sub-equations that are chosen to be the output by the previously presented Sel variable generated by Stage 1.

$$Low_{initial} = \begin{cases} Low_{i-1} + (Range_{i-1} - u), & \text{if Sel} = \text{'1'} \\ Low_{i-1}, & \text{otherwise} \end{cases} \tag{4.9}$$

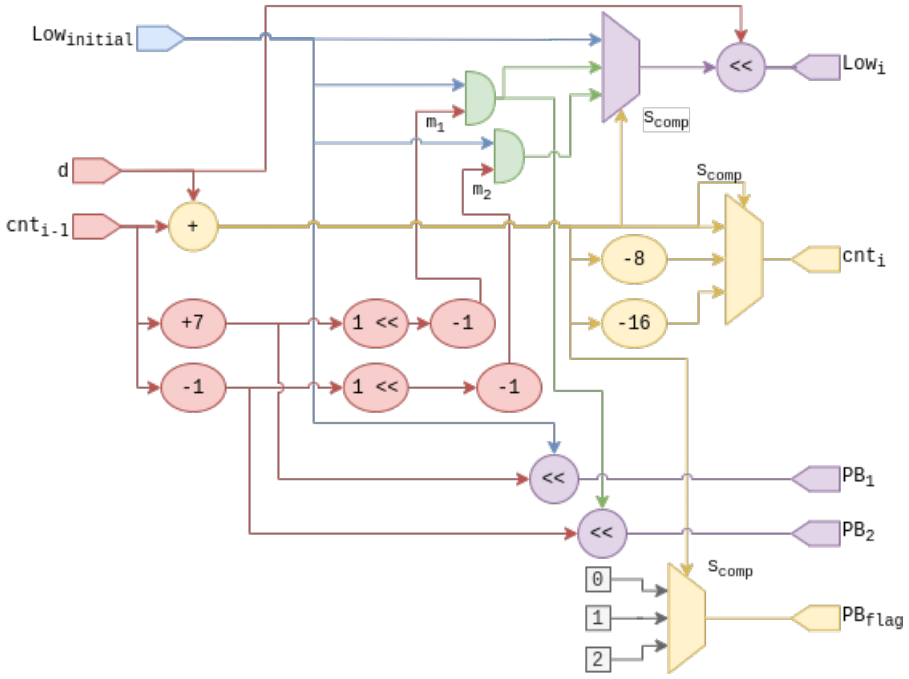
4.2.2.3.2 On the other side of Stage 3 there is the *Boolean Operation* block. This block is also a combination between a variable generated in Stage 2, in this case v_{bool} (also known as the v variable set by the Stage 2 *Boolean Operation* block). Equation 4.10, again using a combination of two sub-equations, defines the Low_{raw}, which is chosen by the incoming 1-bit symbol (i.e., the least significant bit of the 4-bit symbol).

$$Low_{initial} = \begin{cases} Low_{i-1} + (Range_{i-1} - V_{bool}), & \text{if symbol} = \text{'1'} \\ Low_{i-1}, & \text{otherwise} \end{cases} \tag{4.10}$$

4.2.2.3.3 The renormalization process of Stage 3 is quite tricky. Not only it is in charge of generating the Low_i variable, but it also needs to generate the pre-bitstreams, which are, as explained previously, 9-bit arrays that will later originate the 8-bit arrays released by the architecture as final bitstreams. Figure 4.6 defines the entire renormalization and pre-bitstream generation processes, which is presented after 4.2.2.3.4.

To critically analyze Figure 4.6, it is important to understand the equations behind that circuitry. Firstly, a combination between the incoming cnt_{i-1} and d (i.e., S_{comp},

Figure 4.6: Stage 3 renormalization and pre-bitstream generation processes.



Source: The Author (2022).

where '*comp*' represents '*comparison*') is done in (4.11). This combination is used to define whether to reduce the Low and generate pre-bitstreams or not.

$$S_{comp} = cnt_{i-1} + d \tag{4.11}$$

The next step is to define, in parallel, the three possible output values for cnt_i and Low_{raw}. The number three also represents the number of possible decisions taken in this process, according to the following list:

- $S_{comp} < 9$ prevents the Low reduction and does not allow any pre-bitstream generation;
- $9 \leq S_{comp} < 17$ defines the generation of only one pre-bitstream and a Low reduction;
- $S_{comp} \geq 17$ indicates the generation of two pre-bitstreams and a greater Low reduction.

The Low reduction happens by the definition of a mask m, which is calculated in parallel for each reduction case according to (4.12) and (4.13), where m_1 and m_2 represent, respectively, the cases $9 \leq S_{comp} < 17$ and $S_{comp} \geq 17$.

$$m_1 = [1 \ll (cnt_{i-1} + 7)] - 1 \tag{4.12}$$

$$m_2 = [1 \ll (cnt_{i-1} - 1)] - 1 \tag{4.13}$$

Each of the masks, m_1 and m_2, is used upon the $Low_{initial}$ variable, which originates the two possibilities for Low_{raw}, according to (4.14). The definition of cnt_i (i.e., the final value for the cnt counter) is done by (4.15), following the same three conditions.

$$Low_{raw} = \begin{cases} Low_{initial} \wedge m_1, & \text{if } 9 \leq S_{comp} < 17 \\ Low_{initial} \wedge m_2, & \text{if } S_{comp} \geq 17 \\ Low_{initial}, & \text{otherwise} \end{cases} \tag{4.14}$$

$$cnt_i = \begin{cases} S_{comp} - 8, & \text{if } 9 \leq S_{comp} < 17 \\ S_{comp} - 16, & \text{if } S_{comp} \geq 17 \\ S_{comp}, & \text{otherwise} \end{cases} \tag{4.15}$$

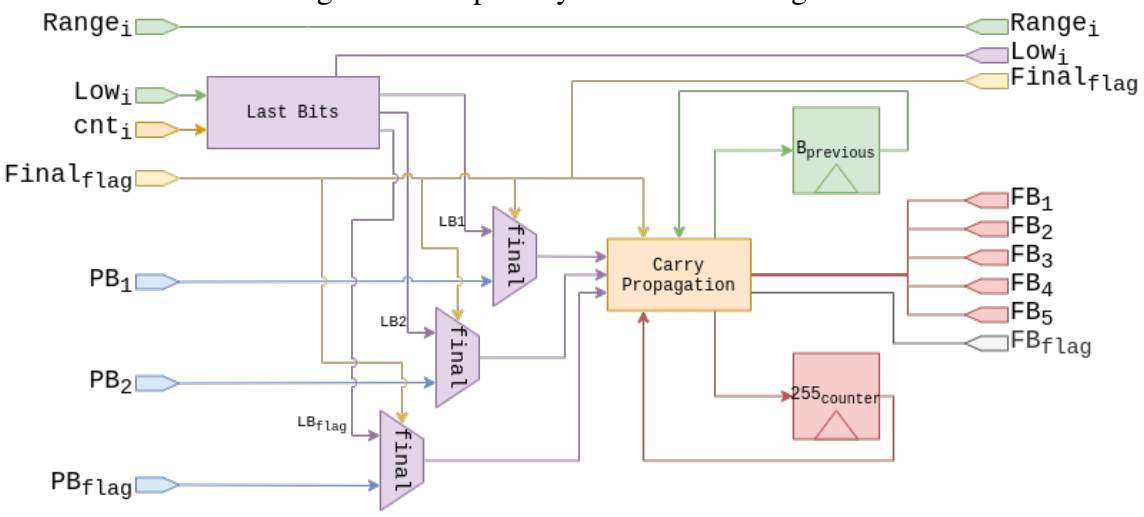
4.2.2.3.4 The pre-bitstream generation, which was already depicted in Figure 4.6, relies on the same cnt counter variable and on the three previously presented cases for the Low reduction. The pre-bitstream generation is mainly defined by (4.16), (4.17) and (4.18), where pre-bitstream 1 (PB_1), pre-bitstream 2 (PB_2) and the flag (PB_{flag}) are, respectively defined. PB_{flag} is the variable that identifies to any outside circuitry (i.e., testbench or a following module on the codec execution line) that (0) no pre-bitstream is valid, (1) only PB_1 is valid, or (2) both PB_1 and PB_2 are valid.

$$PB_1 = \begin{cases} Low_{initial} \gg (cnt_{i-1} + 7), & \text{if } S_{comp} \geq 9 \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

$$PB_2 = \begin{cases} (Low_{initial} \wedge m_1) \gg (cnt_{i-1} - 1), & \text{if } S_{comp} \geq 17 \\ 0, & \text{otherwise} \end{cases} \tag{4.17}$$

$$PB_{flag} = \begin{cases} 1, & \text{if } 9 \leq S_{comp} < 17 \\ 2, & \text{if } S_{comp} \geq 17 \\ 0, & \text{otherwise} \end{cases} \tag{4.18}$$

Figure 4.7: Top-entity schematic of Stage 4.



Source: The Author (2022).

4.2.2.4 Stage 4

The final stage of the architecture, the so-called Carry Propagation stage, is responsible for the carry propagation of the pre-bitstreams, which generates the 8-bit final bitstreams. As explained in previous sections, and shown in (4.19), (4.20) and (4.21), the carry propagation combines the 9-bit arrays PB_{i-1} with PB_i to generate an 8-bit final bitstream. Figure 4.7 shows the top schematic of Stage 4 and its sub-modules.

$$FB_1 = B_{previous} + PB_1[8] \tag{4.19}$$

$$FB_2 = PB_1[7:0] + PB_2[8] \tag{4.20}$$

$$B_{previous} = PB_2[7:0] \tag{4.21}$$

The situation depicted by (4.19), (4.20) and (4.21) considers a $PB_{flag} = 2$, which generates PB_1 and PB_2 at the same round. Another situation, presented by (4.22) and (4.23), defines the scenario where $PB_{flag} = 1$ and, therefore, only PB_1 is valid. If $PB_{flag} = 0$, the $B_{previous} = B_{previous-1}$, which is the same as affirming that $B_{previous}$ does not change.

$$FB_1 = B_{previous} + PB_1[8] \tag{4.22}$$

$$B_{previous} = PB_1[7:0] \tag{4.23}$$

As one may notice after analyzing (4.19), (4.20), (4.21), (4.22) and (4.23), the Verilog (Cadence, 2021) notation $[8]$ is used to express that only the most significant bit (MSB), in a 9-bit array (i.e., the 9th bit), is being used for the given equations. When $[7:0]$ is presented, then the equation considers the eight least significant bits (LSB) of the given array.

Although oversimplified by the previously presented equations, Stage 4 also has to deal with an exception that compromises the entire process described so far. The so-called *255 Exception*, which is identified in a scenario where $B_{previous} = 255$ and $PB_1 > 255$, forces the architecture to propagate the carry not only to $B_{previous}$, but also to $B_{previous-1}$, which is not kept in the registers.

In order to solve the *255 Exception*, a $255_{counter}$ is used, which counts the number of arrays representing the decimal number 255 received, in sequence, as pre-bitstream until the first $PB_1 \neq 255$ or $PB_2 \neq 255$ arrives. Once the *255 Exception* is broken, then the architecture releases four outgoing final bitstreams, according to (4.24), (4.25), (4.26), (4.27) and (4.28).

$$FB_1 = B_previous + PB_1[8] \tag{4.24}$$

$$FB_2 = 255 + PB_1[8] \tag{4.25}$$

$$FB_3 = 255_{counter} \tag{4.26}$$

$$FB_4 = PB_1[7:0] + PB_2[8] \tag{4.27}$$

$$B_{previous} = PB_2[7:0] \tag{4.28}$$

The scenario presented by (4.24), (4.25), (4.26), (4.27) and (4.28) considers $PB_{flag} = 2$, where $PB_1 \neq 255$ and $PB_2 \neq 255$. If $PB_1 = 255$, then $255_{counter}$ receives an increment, whereas if $PB_2 = 255$, then the *255 Exception* is restarted. Another possible scenario is $PB_{flag} = 1$, where FB_4 will not be generated and $B_{previous} = PB_1[7:0]$.

4.2.2.4.1 ***Last Bits*** **block** Within Figure 4.7, one may notice, besides the *Carry Propagation* block, the *Last Bits* blocks. This block is activated once for each frame being encoded and its use relies on the idea that any remaining Low needs to be transformed in the last bitstreams of the frames, so it can be accurately used in the decoding process.

The $Final_{flag}$, which is a flag set by outside circuitry or testbenches to identify the last symbol in a frame, is used to activate the *Last Bits* block. This block will then take the final Low and cnt values and apply (4.29), (4.30), (4.31) and (4.32), where $m = 65535$ in decimal.

$$e_1 = [(Low_i + m) \wedge \overline{m}] \vee (m + 1) \tag{4.29}$$

$$LB_1 = e_1 \gg (cnt_i + 7) \tag{4.30}$$

$$LB_2 = \{e_1 \wedge [1 \ll (cnt_i + 7) - 1]\} \gg (cnt_i - 1) \tag{4.31}$$

$$LB_{flag} = \begin{cases} 1, & \text{if } 9 < (cnt_i + 10) \leq 17 \\ 2, & \text{if } (cnt_i + 10) > 17 \\ 0, & \text{otherwise} \end{cases} \tag{4.32}$$

The LB_{flag}, LB_1 and LB_2 variables generated by (4.29), (4.30), (4.31) and (4.32) are then propagated by the multiplexers presented within Figure 4.7 until they reach the *Carry Propagation* block.

Moreover, another implication created by the $Final_{flag} = 1$ scenario is the release of the $B_{previous}$ as final bitstream. Usually, as the logic mandates for all rounds of codification, Stage 4 keeps the $B_{previous}$ stored for further carry propagation executions upon the arrival of upcoming pre-bitstreams. However, as the $Final_{flag}$ identifies the last round of execution before a complete reset of the architecture, the $B_{previous}$ must be released as final bitstream.

As one may guess at this point, the combination between the *255 Exception* with the $Final_{flag} = 1$ might be the only case where the output pin FB_5 is used. As this combination is considered rare, $B_{previous}$ always uses the next available FB pin, and a given flag is defined to describe the valid output pins. Table 4.1 defines every possible combination of flag generated by Stage 4.

Table 4.1: Map of the FB_{flag}.

Flag Binary	**Flag Decimal**	FB_1	FB_2	FB_3	FB_4	FB_5
000	**0**	-	-	-	-	-
001	**1**	Valid Bits	-	-	-	-
010	**2**	Valid Bits	Valid Bits			
011	**3**	Valid Bits	Valid Bits	Valid Bits		
100	**4**	-	-	-	-	-
101	**5**	Valid Bits	$255 + Carry$	$255_{counter}$	-	-
110	**6**	Valid Bits	$255 + Carry$	$255_{counter}$	Valid Bits	-
111	**7**	Valid Bits	$255 + Carry$	$255_{counter}$	Valid Bits	Valid Bits

4.2.3 Key hardware design decisions

This subsection has the goal of briefly describing some important decisions taken during the architecture's design and how they affected the outcome of the project. The first one, and perhaps the most important, is the definition of the number of pipeline stages.

As already presented in previous sections, the number of stages in the pipeline is a direct reflex of the possibility to differently divide the logic. The number four was reached after taking into considering (i) what was the critical path, and (ii) how to reduce the critical path without creating bubbles. With regards to (i), the critical path was locate around the multiplications in the $Range$ updating process, whereas regarding the (ii), bubbles would only be created if $Range$ or Low required more than one round (i.e., one clock cycle) to be generated.

Moreover, definition that the Low updating process would be located one stage after $Range$ (i.e., $Range$ in Stage 2 and Low in Stage 3) emerges as a good idea due to Low relying on some of the values generated during the $Range$ processing. Therefore, if Low was to be updated in in same stage as $Range$, then the critical path would be greater, as it would comprise, in sequence, the $Range$ and Low processes.

Another important decision to take into consideration was the carry propagation process. According to the AV1 reference software, the carry propagation occurs after the entire codification of a frame and the storage of all pre-bitstreams into a buffer. As the goal of this project is to design a hardware architecture, the buffer would me limited and, therefore, this approach would create a possibility for overflow.

As one may also notice, the use of a buffer prior to the carry propagation where all pre-bitstreams were to be stored would also create a synchronizing problem. Imagine *Frame 1* generated 100 pre-bitstreams and *Frame 2* comprises 50 symbols, for example.

Assuming it takes one round to propagate the carry for each pre-bitstream, *Frame 2* would be done with its encoding process before *Frame 1* finishes its carry propagation. The complexity created by this approach would be enormous and uncountable problems could emerge simply by encoding a larger video.

Therefore, the use of a logic that allows the carry propagation to be executed exactly one round after the generation of each pre-bitstream reduces the chances for problems and avoids reserving area in the chip for buffers that could never be used. Moreover, the predictability of storing only two values ($B_{previous}$ and $255_{counter}$) allows a simpler logic than by using the buffer approach.

The analysis herein presented of different scenarios and different implementations allow the creation of a more robust and closer to error-free architecture. Designing an architecture implies the use a different techniques and approaches for problems that can be tackle in a non-efficient way in software, but that need to be efficient in hardware. Dealing with memories and/or buffers that need to store an unpredictable amount of data is always complicated, as a large chip area might be allocated memory positions that will rarely be used. Therefore, avoiding memories as much as possible, and relying mostly on combinational logic are good practices when designing a hardware architecture.

Chapter 5 introduces the versions obtained during the architectural exploration of the AV1 Arithmetic Encoder. During that phase, the designs AE-AV1-LP, and AE-AV1-MB 1-, 2-, and 3-bool were implemented. As also mentioned in Chapter 5, a AE-AV1-MB 4-bool version was implemented, but found to be inefficient during to huge frequency and throughput rate drops when comparing to the baseline AE-AV1 design.

5 LOW-POWER AND MULTI-BOOLEAN APPROACHES/ARCHITECTURES

The architecture named AE-AV1 presented throughout Chapter 4 as the first ever published AV1 arithmetic encoder architecture. The AE-AV1, then, emerges as the most important version of this work due to its creation as the baseline design used for comparison with other versions, which aim to apply specific improvements into the architecture to achieve better results regarding specific aspects such as power consumption and throughput rate. The following sections present the additional versions the AV1 arithmetic encoder architecture and are organized as follows: low-power version (AE-AV1-LP) in 5.1, and a multi-boolean version (AE-AV1-MB) in 5.2.

5.1 AE-AV1-LP: Low-Power Version

AE-AV1-LP is defined as a low-power version of the baseline AE-AV1 architecture presented throughput Chapter 4. This section's main goal is to provide information regarding the methodology used to implement the low-power version (Subsection 5.1.1), the low-power techniques applied (Subsection 5.1.2) and the main changes between AE-AV1 and AE-AV1-LP (Subsection 5.1.3).

5.1.1 Statistical Analysis

In order to effectively add low-power techniques into the baseline architecture, which are presented within Subsection 5.1.2, it was necessary analyze the AV1 Arithmetic Encoder process to understand how it behaves when exposed to real-world data. During the entire designing process of the AE-AV1, the aim was to create a hardware solution able to accelerate real-world video sequences and still achieve enough throughput rates for a real-time coding of an UHD (Ultra-High Definition) video (i.e., 8K@120fps, where $800MBits/second$ is the minimum throughput rate established by the AV1 documentation (RIVAZ; HAUGHTON, 2019)). Thus, the creation of a low-power version of the architecture needs to follow the same principles and be based on real-world data.

In order to follow the presented principles, several video sequences from mainly two datasets (MERCAT; VIITANEN; VANNE, 2020; MONTGOMERY, 2005) were analyzed and their main behaviors and patters were observed. Among different analysis made

Table 5.1: Analysis upon the *Boolean Operation* calling rate for 6 120-frame video sequences in two Constraint of Quality (CQ) configurations (MERCAT; VIITANEN; VANNE, 2020).

CQ	Resolution	Video	*Boolean Operation* rate
20	1080p @ 120 8-bit	Beauty	17.75%
20	1080p @ 120 8-bit	Bosphorus	20.17%
20	1080p @ 120 8-bit	HoneyBee	19.58%
20	1080p @ 120 8-bit	Jockey	17.99%
20	4K @ 120 10-bit	ReadySetGo	19.47%
20	4K @ 120 10-bit	YachtRide	21.94%
55	1080p @ 120 8-bit	Beauty	12.33%
55	1080p @ 120 8-bit	Bosphorus	15.43%
55	1080p @ 120 8-bit	HoneyBee	17.63%
55	1080p @ 120 8-bit	Jockey	9.41%
55	4K @ 120 10-bit	ReadySetGo	11.02%
55	4K @ 120 10-bit	YachtRide	12.87%
		Average	**16.30%**

(i.e., symbols rate, $nsyms$ rate, etc), it was analyzed the frequency to which the *Boolean Operation* was used over the total number of encoded symbols.

As one may notice after analyzing Tables 5.1 and 5.2, the *Boolean Operation* calling rate is directly connected to the Constraint of Quality (CQ) configuration. Additionally, in Table 5.2, the *allintra* and *good* configurations do not display any apparent effect upon the calling rate of *Boolean Operation*.

Figure 5.1 depicts a graph with the compounded calling rate of the *Boolean Operation* divided by CQ group, which was calculated based on the average of the results presented in Tables 5.1 and 5.2. As one may notice, for the *allintra* configuration, the higher is the CQ, the lower will be the number of calls for the *Boolean Operation*. As one may notice, on the other hand, the *Boolean Operation* calling rate for the *good* configuration also decreases with the increase of the CQ, but with a less steep curve.

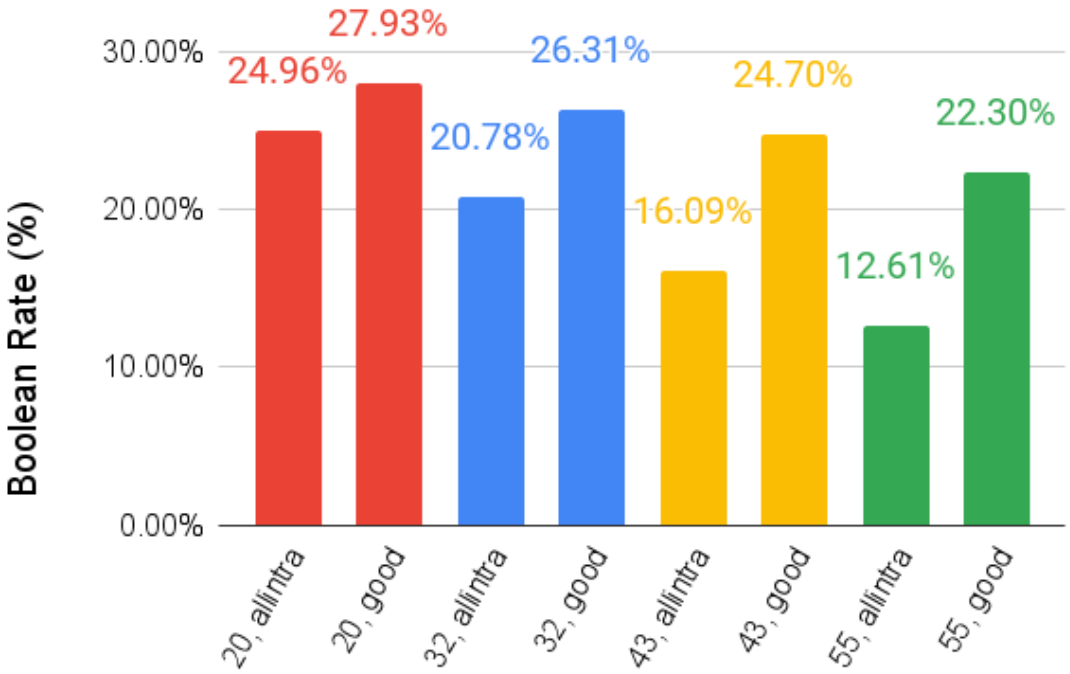
Moreover, despite the changes in the *Boolean Operation* rate according to the encoding configurations, none of the video sequence displays any *Boolean Operation* rate greater than 30%. As one may remember from Stages 2 and 3 descriptions (refer to 4.2.2.2 and 4.2.2.3), the entire AE-AV1 architecture is mainly divided into two circuitry: *CDF Operation* and *Boolean Operation*.

Therefore, as hardware architectures can be simply defined as several circuitry blocks connected by internal wires, one may expect that once the input pins are set to a value, all connected components shall generate output values even if their values are useless (i.e., when running a *CDF Operation*, *Boolean Operation* blocks will generate

Table 5.2: Analysis upon the *Boolean Operation* calling rate for 8 60-frame video sequences with a total of 8 configurations for each (MONTGOMERY, 2005).

CQ	Config.	Resolution	Video	***Boolean Operation* rate**
20	*allintra*	720p	Boat	20.50%
20	*allintra*	720p	Dark	23.44%
20	*allintra*	720p	Kristen And Sara	23.93%
20	*allintra*	720p	Netflix Driving	29.34%
20	*allintra*	720p	Netflix Roller Coaster	27.60%
20	*good*	720p	Boat	25.86%
20	*good*	720p	Dark	25.91%
20	*good*	720p	Kristen And Sara	26.61%
20	*good*	720p	Netflix Driving	32.73%
20	*good*	720p	Netflix Roller Coaster	28.54%
32	*allintra*	720p	Boat	16.14%
32	*allintra*	720p	Dark	17.51%
32	*allintra*	720p	Kristen And Sara	20.11%
32	*allintra*	720p	Netflix Driving	25.61%
32	*allintra*	720p	Netflix Roller Coaster	24.53%
32	*good*	720p	Boat	23.37%
32	*good*	720p	Dark	25.74%
32	*good*	720p	Kristen And Sara	23.34%
32	*good*	720p	Netflix Driving	31.05%
32	*good*	720p	Netflix Roller Coaster	28.07%
43	*allintra*	720p	Boat	12.51%
43	*allintra*	720p	Dark	12.65%
43	*allintra*	720p	Kristen And Sara	16.07%
43	*allintra*	720p	Netflix Driving	19.78%
43	*allintra*	720p	Netflix Roller Coaster	19.45%
43	*good*	720p	Boat	20.75%
43	*good*	720p	Dark	24.48%
43	*good*	720p	Kristen And Sara	22.52%
43	*good*	720p	Netflix Driving	29.01%
43	*good*	720p	Netflix Roller Coaster	26.72%
55	*allintra*	720p	Boat	16.14%
55	*allintra*	720p	Dark	8.49%
55	*allintra*	720p	Kristen And Sara	11.30%
55	*allintra*	720p	Netflix Driving	12.93%
55	*allintra*	720p	Netflix Roller Coaster	14.19%
55	*good*	720p	Boat	17.87%
55	*good*	720p	Dark	20.16%
55	*good*	720p	Kristen And Sara	23.31%
55	*good*	720p	Netflix Driving	25.95%
55	*good*	720p	Netflix Roller Coaster	24.19%
			Average	**21.96%**

Figure 5.1: Graph with the compounded calling rate for the *Boolean Operation* divided by the set of configurations.



Source: The Author (2022).

useless and invalid values, which shall be discarded). Hence, in order to reduce the overall power consumption of the architecture, emerges as an interesting option the isolation of the *Boolean Operation* blocks from their input signals to avoid processing and storing useless values.

5.1.2 Low-Power Techniques

With the goal of reducing power consumption already during the design phase (i.e., without relying on physical design techniques), clock gating and operand isolation emerge as effective low-power techniques to reduce dynamic power consumption. These techniques were explored in the (BITENCOURT; RAMOS; BAMPI, 2021) worked, which was written by the author, and included in this thesis in 5.1.2.1 and 5.1.2.2.

5.1.2.1 Clock gating

The application of the clock gating technique seeks to prevent registers from storing non-essential values. This action is achieved by restricting the clock signal from reaching the registers once the architecture detects that a specific component is not needed

at a given time (Qing Wu; PEDRAM; Xunwei Wu, 2000).

As one may notice after reading Chapter 4, the Stages 2 and 3 are basically divided in two different parallel circuitry: *CDF* and *Boolean* operations. Hence, the register barriers after the cited stages comprise registers only useful when executing *Boolean Operation* and, as the power analysis presented Subsection 5.1.1 shows, this specific operation occurs at a considerably lower rate than the *CDF Operation*. Therefore, applying clock gating on the registers only connected to the *Boolean Operation* creates a decrease in power consumption.

Furthermore, according to 4.2.2.3, the Stage 3 does not generate pre-bitstreams in every clock cycle and, hence, the architecture does not always release bitstreams. Therefore, the addition of clock gating into the registers responsible for pre-bitstream and bitstream storage (i.e., after Stages 3 and 4) also creates a significant reduction on power consumption.

5.1.2.2 Operand Isolation

On the other hand, the operand isolation technique aims to reduce power consumption by not allowing non-essential operations to be completed. This technique seeks to isolate each operand and force the outputs of logical or mathematical operations to be constant (preferably set to zero) when the operations are not critical (MUNCH et al., 2005). The use of operand isolation is important because some of the operations do not depend on registers. Thus, even though their results will not be stored, they still generate output and, hence, consume energy.

Again, as explained in Subsection 5.1.2.1, the distinction between *CDF* and *Boolean* operations allows for the application of low-power techniques. In this specific case, instead of targeting the registers, it is necessary to isolate the operands for each of the *Boolean Operation* sub-steps presented in 4.2.2.2 and 4.2.2.3. Moreover, the pre-bitstream and bitstream generation processes also allow for the application of operand isolation for the same reason explained in 5.1.2.1.

Operand isolation can also be added into the *Last Bits* block within Stage 4, as the block is only used upon the arrival of each frame's last symbol. Therefore, as none of the other symbols' arrival will generate significant results from *Last Bits* block, the technique can be added to each of the components presented within 4.2.2.4.

5.1.3 From AE-AV1 to AE-AV1-LP

The low-power version of AE-AV1, the so-called AE-AV1-LP, is featured as a solution that reduces the power consumption (i.e., by around 19.89% as is later presented in Chapter 6) while ensuring enough throughput rate for ultra-high performance (8K@120fps in real-time). As presented throughout previous subsections of this section (i.e., 5.1.1 and 5.1.2), the idea of the AE-AV1-LP is to include the Clock Gating and Operand Isolation low-power techniques into the original and baseline AE-AV1 design to save power by avoiding, respectively, the storage and calculation of useless values generated by the *Boolean Operation* block when executing a *CDF Operation*. Moreover, this is possible due to the analysis presented within Subsection 5.1.1, which shows that, on average, only 16.97% of the total clock cycles required to encode a video sequence uses the *Boolean Operation* block.

The following list presents the equations and registers that received each of the aforementioned low-power techniques:

Stage 1: Clock Gating applied into registers connected to (4.3) and (4.1). As all *Boolean Operation* variables are also used within the *CDF Operation*, the Clock Gating techniques aim to avoid storing results for the *CDF Operation*. No Operand Isolation was applied.

Stage 2: Clock Gating applied into registers connected to, in one side (4.5), and on the other (4.6). The reason is that Stage 3 uses or u or v_{bool}, never both. Operand Isolation was applied into (4.5).

Stage 3: As PB_1 and PB_2 are not generated in every clock cycle, Clock Gating was individually applied into each register for either variables and was triggered by PB_{flag}. Operand isolation was applied into all components connected to the *Boolean Operation*.

Stage 4: Clock Gating applied into all FB_1–FB_5 registers. Operand Isolation added into the entire *Last Bits* circuitry because it is only used for one round per frame.

5.2 AE-AV1-MB: Multi-Boolean Version

As AE-AV1-LP, the AE-AV1-MB is presented as an alternate version of the baseline AE-AV1 architecture. AE-AV1-MB aims to increase the throughput rate of the architecture by parallelizing the *Boolean Operation* in Stages 2 and 3. For that, a proposal for the $Range$ updating process is introduced in Subsection 5.2.1 and the actual architecture is elucidated in Subsection 5.2.2.

5.2.1 Multi-Boolean Proposal

As highlighted in Chapter 4, the AV1 arithmetic encoding process relies upon two operations: *CDF Operation* and *Boolean Operation*. For a hardware design of AV1 arithmetic encoding, (5.1) features part of the critical path for the entire process (i.e., the common equation for the $Range$ updating step used on both *CDF* and *Boolean* operations), where F, $Range_{i-1}$ and $Range_{raw}$ represent, respectively, the 16-bit probability, the incoming $Range$ and the $Range$ prior to renormalization. Equation (5.1) is analog to (4.7), which relies on (4.5) and (4.6).

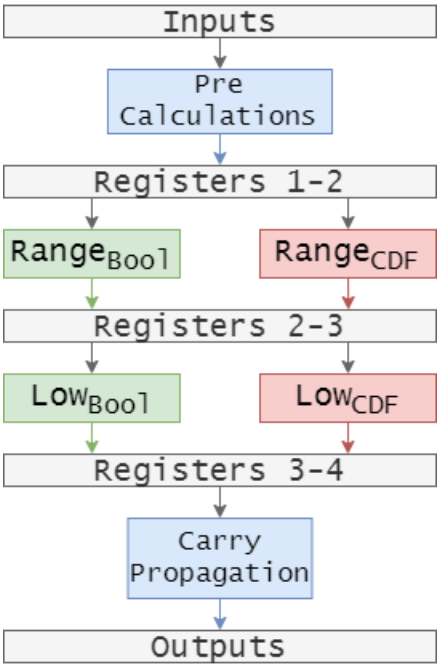
$$Range_{raw} = (Range_{i-1} \gg 8) \times (F \gg 6) \gg 1 \tag{5.1}$$

Considering F as a 16-bit variable and the maximum probability being 32768 (RIVAZ; HAUGHTON, 2019), the *Boolean Operation* probability, which is fixed in 50%, is represented by the decimal value 16384, thus reaching 256 after a right-shift by 6. As one may notice, $2^8 = 256$, which allows the removal of the multiplication from (5.1), obtaining (5.2).

$$Range_{raw} = (Range_{i-1} \gg 8) \ll 7 \tag{5.2}$$

When converting the new $Range$ updating process for the *Boolean Operation* into architecture, it is feasible to assume that the new *Boolean Operation* from (5.2) creates less delay than *CDF Operation*, which does not allow the same transformation due to its always-changing probabilities. Furthermore, if considered Figure 5.2 as being a simplified diagram for a 4-stage baseline AV1 arithmetic encoder architecture, one may find feasible to allocate multiple *Boolean* blocks into Stage 2 ($Range$ updating process), which will result in an increased number of *Boolean* symbols being encoded during the

Figure 5.2: Simplified diagram for a baseline AV1 arithmetic encoder.



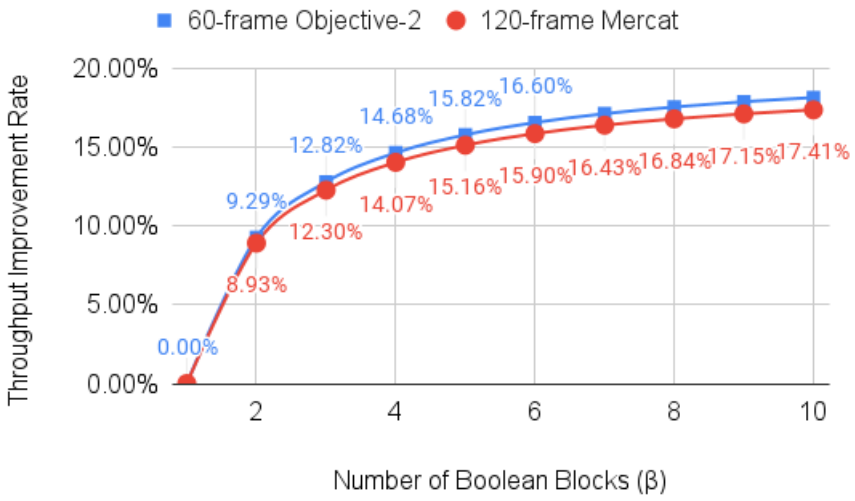
Source: The Author (2022).

same time taken by one *CDF* symbol. Stages 1 and 4 from Figure 5.2 comprise the pre-calculations and carry propagation blocks respectively (BITENCOURT; RAMOS; BAMPI, 2021), whereas Stage 3 is the Low updating process.

The herein proposed parallel approach for the *Boolean Operation* takes advantage of the *CDF Operation* complexity to increase the number of symbols encoded per round of execution. Figure 5.3 displays a graph resulting from analysis made using twelve video sequences from (MERCAT; VIITANEN; VANNE, 2020) (i.e., video sequences *Beauty*, *Bosphorus*, *HoneyBee*, *Jockey*, *ReadySetGo* and *YachtRide* encoded under Constraints of Quality – CQs – 20 and 55) and forty video encodings from (MONTGOMERY, 2005) (*Boat*, *Dark*, *KristenAndSara*, *Netflix_RollerCoaster* and *Netflix_Driver*, encoded under CQs 20, 32, 43 and 55, using configurations *allintra* and *good*), and considering different scenarios of β, where β represents the number of *Boolean* blocks in parallel. While disregarding any increase in delay, which is deeply analyzed in Subsection 5.2.2, the graph presents the throughput improvement rates for different versions of the architecture in which β hangs within $1 \leq \beta \leq 10$.

As one may conclude after analyzing Figure 5.3, the greatest improvements happen when $2 \leq \beta \leq 4$. When $\beta > 4$, the improvements turn smaller until reaching asymptotically the value of around 17%.

Figure 5.3: Number of *Boolean* blocks in parallel (β) *vs.* Throughput improvement rate.



Source: The Author (2022).

5.2.2 Multi-Boolean Architecture

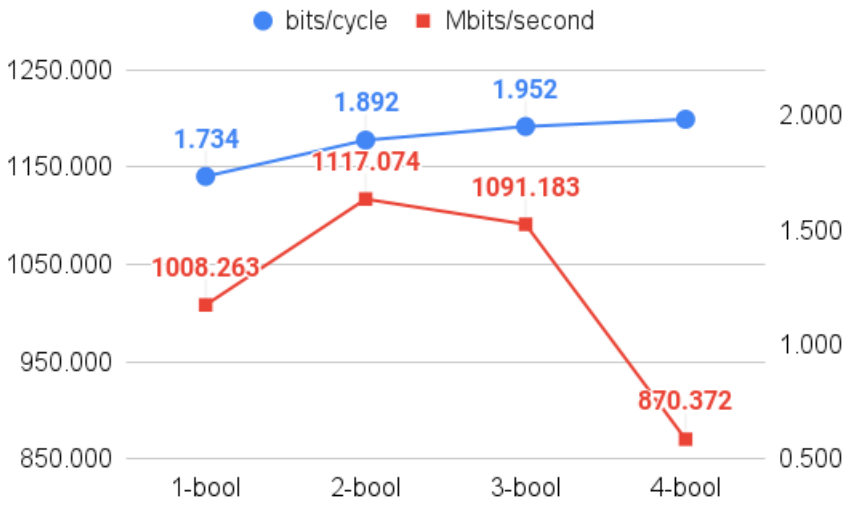
The association for the *Boolean Operation* presented in (5.2) makes it possible to add multiple *Boolean* blocks to be executed in parallel with only one *CDF* block and therefore create the proposed AE-AV1-MB architecture. As suggested by Figure 5.2, for each *Boolean* block or *CDF* block added into architecture's Stage 2, one corresponding block must be added into Stage 3 (Low updating process) likewise.

Moreover, the Carry Propagation block (i.e., pipeline Stage 4) is capable of dealing with up to two pre-bitstreams and one flag (PB_1, PB_2 and PB_{flag}) per round. Analysis upon the use of different values for β show that when $\beta \geq 4$, one extra Carry Propagation block must also be added.

The herein presented AE-AV1-MB architecture aims to include more $Range_{Bool}$ blocks, which forces the addition of an equal β number of Low_{Bool} blocks into Stage 3 and, perhaps, a few Carry Propagation blocks into Stage 4. Furthermore, assuming $Range_{CDF} \gg Range_{Bool}$ and $Range_{CDF} > Low_{Bool}$ with regards to delay, where Low_{Bool} is defined by a set of several sequential simple operations (RIVAZ; HAUGHTON, 2019; Alliance for Open Media, 2020), the most likely scenario to happen first is that when β reaches a certain λ, the critical path of the architecture migrates from Stage 2 to Stage 3.

As suggested by Figure 5.4, the λ value is four, which indicates that when $\beta \geq \lambda$, the critical path of the architecture becomes Stage 3 instead of Stage 2. Figure 5.4 displays the throughput per second ($Mbits/second$) for $1 \leq \beta \leq 4$ versions (i.e., 'β-

Figure 5.4: Comparison between the throughput rates for AE-AV1-MB versions.



Source: The Author (2022).

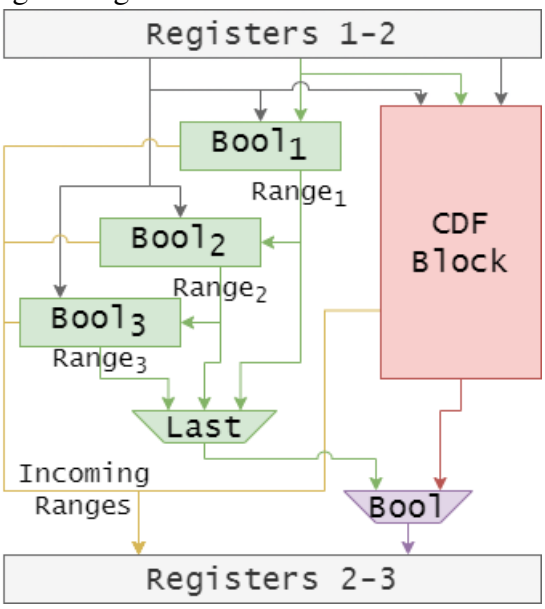
bool') and, therefore, takes into consideration the frequency results obtained from the synthesis process using Cadence™ RTL Compiler (RC) tool with the ST 65nm PDK.

Figure 5.5 displays the Stage 2 diagram for the $\beta = 3$ (i.e., 3-bool) version of the architecture. As one may conclude, each of the *Boolean* blocks receives one symbol from the sequence S_i, S_{i+1} and S_{i+2}, where S represents a symbol being encoded and i the current round of execution. Upon the end of the current round, a certain number of *Boolean* symbols Φ, where Φ is comprised within $0 \leq \Phi \leq \beta$, might arrive according to upcoming *Boolean* symbols found within the video sequence. The *Last* multiplexer represents a choice of the last valid *Boolean* block in a round to define $Range_i$ (i.e., final $Range$), whereas the *Bool* multiplexer chooses between the *CDF* and *Boolean* operations. The incoming $Range$ values for each *Boolean* block (e.g., $Range_1$ for $Bool_2$) must be used in Stage 3 and are, therefore, set into their respective output pins from Stage 2.

Stage 2 in Figure 5.5 is capable, therefore, of encoding up to three *Boolean* symbols in one clock cycle. Furthermore, it is important to notice that each block presented within Figure 5.5 comprises a renormalization process (i.e., a 3-input multiplexer for the *Boolean* blocks, and a 16-bit Leading Zero Counter, LZC, circuitry (MIAO; LI, 2018) on the *CDF* side). As the renormalization result d for (5.2) approach is never greater than 2, a 3-input multiplexer does the task more effectively than a LZC for the *Boolean* side.

As one may conclude, Stage 3 of the architecture's 3-bool version looks similar to what is displayed in Figure 5.5. The main differences rely upon the pre-bitstream generation process and the internal characteristics of each block, where the *CDF* and *Boolean* blocks need to calculate the Low value. The *Boolean* block in Stage 3 is represented by

Figure 5.5: Stage 2 diagram for the 3-bool version of the architecture.



Source: The Author (2022).

(5.3), where $PL = Range_{i-1} - (Range_{raw} + 4)$ calculated in Stage 2 and stands for '*PreLow*'. No extra Carry Propagation block needs to be added for this version as $\beta < 4$ (i.e., $\beta = 3$).

$$Low_{raw_{Bool}} = Low_{i-1} + PL \tag{5.3}$$

The herein described 3-bool version of the AE-AV1-MB was presented as an example to facilitate the understanding of the parallel proposal. If respecting the technology and hardware limitations with regards to area, delay, and power consumption, any β number of *Boolean* blocks can be added into Stages 2 and 3. Section 6 displays and analyzes the results for the AE-AV1-MB 1- (baseline version), 2- and 3-bool versions, which respect the already-presented $\beta < \lambda$ limitation for maximum optimization.

Chapter 6 present the methodology used to obtain the results for all designs presented in Chapters 4 and 5 (i.e., AE-AV1, AE-AV1-LP, and AE-AV1-MB 1-, 2- and 3-bool). Furthermore, the results are introduced and compared with related works.

6 RESULTS AND DISCUSSION

Throughout the Chapters 4 and 5, three versions of an AV1 arithmetic encoder architecture were presented. The first one, presented within Chapter 4, defines a baseline architecture for the AV1 arithmetic encoder block and was named AE-AV1. The second version, named AE-AV1-LP, is defined as a low-power version of the baseline, whereas the third version is a Multi-Boolean design named AE-AV1-MB.

The results presented in the following subsections were obtained by synthesizing all aforementioned architectures using the Cadence™ RTL Compiler (RC) tool with the ST 65nm PDK. Additional results for the AE-AV1, which are presented within Subsection 6.2.1, were achieved by synthesizing the design using open-source-only solutions (i.e., OpenROAD (OpenROAD, 2021; KAHNG; SPYROU, 2021) tool with SkyWater 130nm PDK (SkyWater; Google, 2021; ENE, 2020) and ASAP7 PDK (CLARK et al., 2016)).

This chapter is organized as follows: Section 6.1 presents the methodology used to achieve the herein presented results, Section 6.2 presents the information regarding AE-AV1, Section 6.3 shows the results for the additional versions designed (i.e., AE-AV1-LP in Subsection 6.3.1, and AE-AV1-MB in Subsection 6.3.2), and Section 6.4 depicts a variety of comparisons between the introduced versions of the AV1 arithmetic encoder architecture and related works.

6.1 Analysis Methodology

This section introduces the methodology used to accomplished the herein presented results. The Cadence™ RTL Compiler (RC) (RAMANI; AHMED, 2016) was the mainly used tool for synthesis. It was chosen due to its powerful capabilities, for being a highly capable tool used throughout the Microelectronics market, and for being capable of delivering realistic results. Additionally, the open-source-focused results used the OpenLane flow (GHAZY; SHALAN, 2020; EDWARDS; SHALAN; KASSEM, 2021), which comprises the OpenROAD (OpenROAD, 2021) tool as one of its core elements. The reason for using the OpenROAD toolset is because of its (i) relevance in the global microelectronics scenario, and (ii) for being a state-of-the-art open-source tool that is very efficient and straightforward to be used.

The choosing of an adequate standard-cell library, also known as PDK (Process Design Kit), is critical for the synthesis process and further procedures of the ASIC flow

(KOMMURU; MAHMOODI, 2009). For this process, the chosen PDKs were the ST 65nm (ST, 2015), SkyWater 130nm, nangate 45nm (TORNG, 2020), (EDWARDS, 2020; Google, 2022; TEODOR-DUMITRU, 2020) and ASAP7 (CLARK et al., 2016). The former was applied in the high-throughput designs (i.e., AE-AV1, AE-AV1-LP and AE-AV1-MB), whereas the latter three were used for the open-source analysis that targeted only the baseline AE-AV1 architecture.

Using the AV1 reference software (Alliance for Open Media, 2020), five video sequences from a datasets (MONTGOMERY, 2005) were encoded under different constraints of quality (CQs) and configuration modes. According to Rivaz e Haughton (2019), the AV1 comprises configuration modes such as *allintra* and *good*, which flags to the software the use of only the intra-prediction block or either inter- and intra-prediction, respectively.

The 60-frame 720p 8-bit depth videos *Boat*, *Dark*, *KristenAndSara*, *Netflix_Driving* and *Netflix_RollerCoaster*, the 60-frame 1080p 8-bit depth videos *Aspen* and *Rush_Hour*, and the 60-frame 4K 8-bit depth *Netflix_Narrator* and *Netflix_Dancers* were taken from the dataset named *objective-2* published by Montgomery (2005). Each of the aforementioned videos was encoded under the CQs 20, 32, 43 and 55, and the configuration modes *allintra* and *good*, totaling 72 videos encoded (i.e., videos encoded will be referred, from now on, as video sequences for the sake of simplification).

Moreover, in order to accurately estimate the throughput rates ($bits/cycle$ and $Gbits/second$), software analysis were created to emulate the architectural behavior of a given version while reading the encoded video sequences. For example, when analyzing the video sequences to acquire the throughput rate information for the AE-AV1-MB version of the architecture, the software eventually was required to read rows without increasing the clock cycle counter to accurately define the $bits/cycle$ throughput rate for the AE-AV1-MB version.

Furthermore, the $Gbits/second$ presented in following sections are a direct result from the combination of the aforementioned software analysis to estimate the $bits/cycle$ throughput rate with the frequency resulted by the synthesis process. This methodology was created to avoid increasing simulation complexity using the already-computationally heavy simulation tools (Cadence, 2021). The power estimation, however, relied upon simulation tools (i.e., Cadence™ *irun* utility (Cadence, 2021)), which generated the TCF (Toggle Count Format) used in the RC (INFN, 2021) to estimate power based upon the switch activity of the design when stimulated with real-world data.

Therefore, the herein presented methodology aims to reduce the analysis complexity while ensuring the achievement of accurate results. Seventy-two video codifications (i.e., also referred as video sequence for the sake of simplicity) were analyzed and generated the throughput rate results depicted in the following sections of this chapter. Although the throughput rate analysis did not rely on hardware simulation, the power analysis required such execution, which combined the Cadence™ *irun* utility with the RC power estimation methodology.

6.2 Baseline AE-AV1 Results

The baseline version of the AV1 arithmetic encoder architecture, the so-called AE-AV1, was capable of achieving 581 MHz when encoded using the ST 65nm PDK. Additionally, the AE-AV1 version accomplished 11.7k gates count of area and 9.072 mW of power. The herein presented results were calculated based on the average of the results obtained for each video codification (i.e., a video codification is defined as the encoding of one video sequence using a given set of configurations), which are presented in Table 6.4.

With regards to the throughput rates, the AV1 documentation stipulates, according to Table 6.1, that an 8K@120fps (i.e., 7680x4320@120fps) resolution requires at least 800 Mbits/seconds of throughput rate for real-time processing (RIVAZ; HAUGHTON, 2019). The AE-AV1 achieves, on average, and after analyzing all video sequences presented within Section 6.1, 1.775 bits/cycle and 1.032 Gbits/second of throughput rates. Therefore, the AE-AV1 version of the AV1 arithmetic encoder architecture is capable of 8K@120fps real-time processing when synthesized for the ST 65nm PDK.

Tables 6.4 and 6.5 presented within Subsection 6.3.1 depict, in detail, the throughput rate and power consumption for the AE-AV1 baseline design. The reason those tables are within Subsection 6.3.1 is due to the comparison they present between the results for the AE-AV1 and AE-AV1-LP designs.

6.2.1 Open-Source Results

The herein presented results for the AE-AV1 design were achieved using the OpenROAD (OpenROAD, 2021) in combination with the following Package Design Kits

Table 6.1: Maximum real-time processing according to throughput rates defined in Mbits/second (RIVAZ; HAUGHTON, 2019).

Level	MainMbps (Mbits/sec)	HighMbps (Mbits/sec)	Resolution
2.0	1.5	-	426x240@30fps
2.1	3.0	-	640x360@30fps
3.0	6.0	-	854x480@30fps
3.1	10.0	-	1280x720@30fps
4.0	12.0	30.0	1920x1080@30fps
4.1	20.0	50.0	1920x1080@60fps
5.0	30.0	100.0	3840x2160@30fps
5.1	40.0	160.0	3840x2160@60fps
5.2	60.0	240.0	3840x2160@120fps
5.3	60.0	240.0	3840x2160@120fps
6.0	60.0	240.0	7680x4320@30fps
6.1	100.0	480.0	7680x4320@60fps
6.2	160.0	800.0	7680x4320@120fps
6.3	160.0	800.0	7680x4320@120fps

(PDKs): Nangate 45nm (*nangate45*), SkyWater 130nm high density (*sky130hd*), SkyWater 130nm high speed (*sky130hs*) and, finally, the ASAP 7nm (*ASAP7*) (CLARK et al., 2016). Moreover, the AE-AV1 architecture used the entire OpenROAD flow as defined by (AJAYI et al., 2019) and classified as 'from RTL to GDSII'.

To effectively compare the results for each PDK, Table 6.2 displays the frequency, throughput rates, and area for each analyzed PDK. The gates count information (i.e., area) of the post-layout synthesis for all technologies was calculated by the actual area obtained by each circuit, and then divided by the smaller two-input gate available on the PDK (i.e., commonly a NAND-2 gate). One may notice that *ASAP7* is presented as the best alternative with regards to frequency, hence throughput rate, and area. As the number of symbols encoded per clock cycle is always one, the throughput rate per second is directly related to the achieved frequency. Furthermore, Fig. 6.1 depicts a graph presenting the correlation between area and frequency between the used PDKs.

As one may notice after analyzing Table 6.2 and Fig. 6.1, the frequency increases in opposition to the technology size, whereas the area decreases with the technology size. When compared to the *nangate45* PDK, the *ASAP7* technology presents a 83.07% increase in frequency and a 24.48% decrease in area. Both *nangate45* and *ASAP7* were able to accomplish enough throughput rate per second for real-time 8K@120fps processing, whereas *sky130hd* and *sky130hs* achieve, respectively, 4K@30fps and 4K@120fps.

When considering the *ASAP7* as having the best results among the presented

Table 6.2: Comparison between the results for for the ASAP7, Sky130hs, Sky130hd and Nangate45 PDKs.

PDKs	□	△	◯	**ASAP7**
Design	AE-AV1			
Tech.	130 nm	130 nm	45 nm	7 nm
Freq.[1]	97	152	513	938
Area[2]	7.7 K	8.7 K	8.2 K	6.2 K
bits/cycle	4.37			
Gbits/sec	0.17	0.27	0.91	1.67
MRTR[3]	4K@30	4K@120	8K@120	8K@120

□ sky130hd
△ sky130hs
◯ nangate45
[1] MHz
[2] Gates Count
[3] Maximum Real-Time Resolution

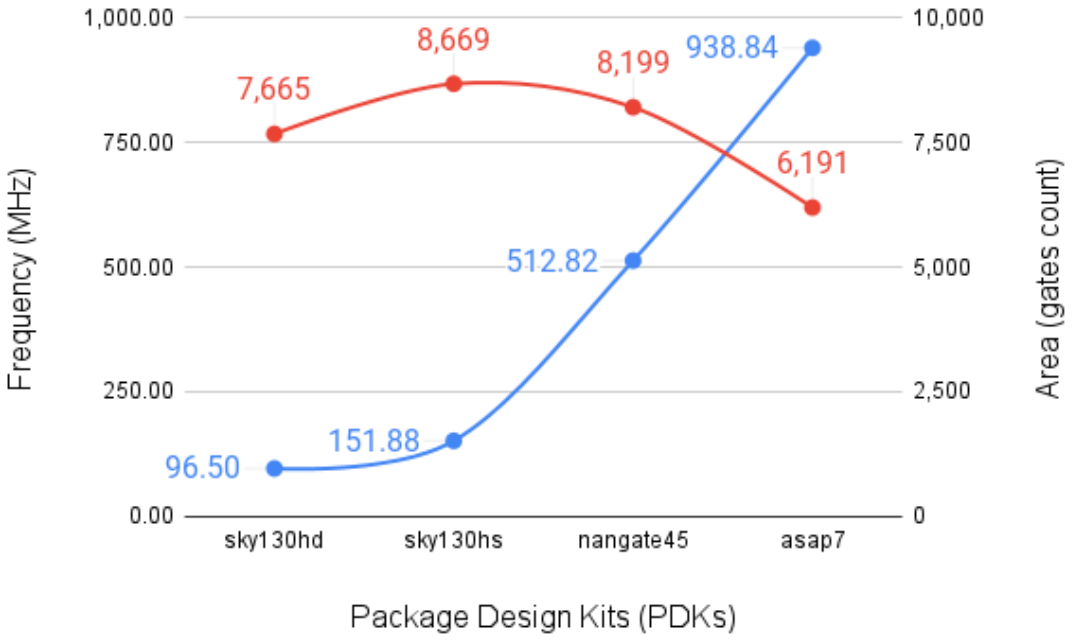
Table 6.3: Comparison between the results accomplished for ASAP7 in (BITENCOURT; RAMOS; BAMPI, 2022c), the baseline AE-AV1 (BITENCOURT; RAMOS; BAMPI, 2021) and other works (Dajiang Zhou et al., 2015; RAMOS et al., 2021; PASTUSZAK, 2020b; CHOI; LEE; CHAE, 2021).

Design	□	◯	△	†	♣	⋆
Video Codec	HEVC	HEVC	HEVC	AVS 2.0	AV1	AV1
Coder	CABAC	CABAC	CABAC	CBAC	Multi-Symbol	Multi-Symbol
Tech. (nm)	90[1]	65[2]	90[1]	65[1]	65[2]	7[3]
Freq. (MHz)	420	507	700	735	581	938
Gates Count	110.9 K	21.22 K	120.4 K	133.5 K	11.7 K	6.2 K
bins/cycle	4.37	4.31	4.67	3.94	[4]	[4]
Gbins/sec	1.836	2.185	3.269	2.90	1.032	1.67[5]
MRTR[6]	8K@120				8K@120	8K@120

[1] TSMC PDK
[2] ST PDK
[3] ASAP
[4] $bits/cycle$
[5] $Gbits/second$
[6] Maximum Real-Time Resolution

□ (Dajiang Zhou et al., 2015)
◯ (RAMOS et al., 2021)
△ (PASTUSZAK, 2020b)
† (CHOI; LEE; CHAE, 2021)
♣ (BITENCOURT; RAMOS; BAMPI, 2021)
⋆ (BITENCOURT; RAMOS; BAMPI, 2022c)

Figure 6.1: Frequency *vs.* Area for each used PDK.

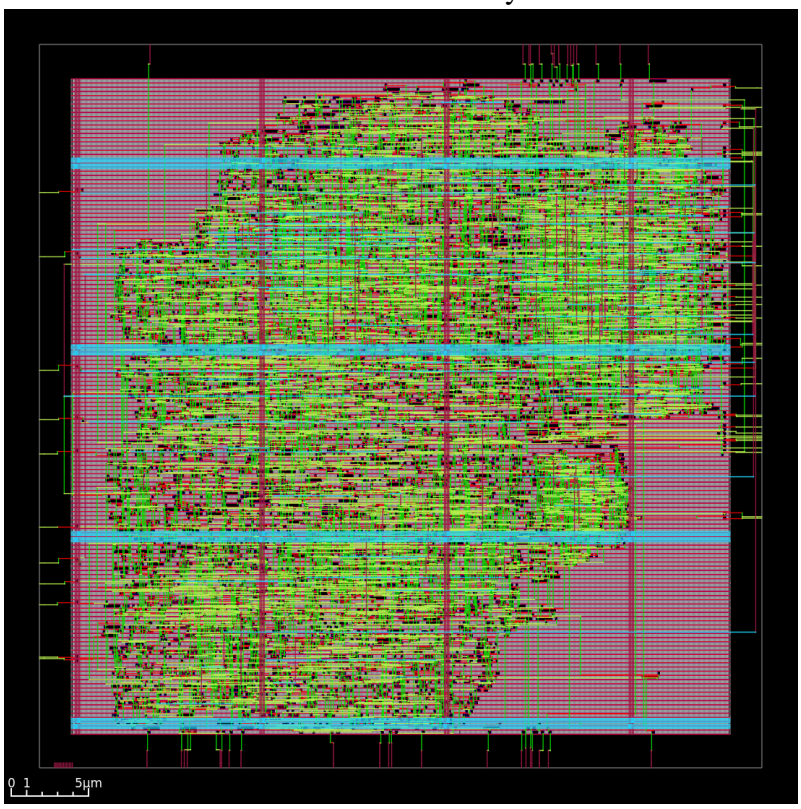


Source: The Author (2022).

PDKs, a comparison between related works from the literature can be made using such numbers. Table 6.3 depicts the results for the works (BITENCOURT; RAMOS; BAMPI, 2021) and (Dajiang Zhou et al., 2015; RAMOS et al., 2021; PASTUSZAK, 2020b; CHOI; LEE; CHAE, 2021), where the former represents the baseline AV1 arithmetic encoder design synthesized with ST 65nm and the latter four are CABAC architectures. As one may notice, the herein presented flow with ASAP7 has the best results regarding the area, frequency, and throughput rates, and can accomplish up to 8K@120fps real-time processing (i.e., highest resolution supported by AV1 (RIVAZ; HAUGHTON, 2019)).

Although very effective in depicting the overall capabilities of the designs we are proposing and analyzing in this manuscript, Table 6.3 does not provide a fair comparison due to the differences in the throughput rate units used for CABAC and CBAC when compared to AV1. While AV1 relies upon $bits/cycle$ and $Gbits/second$ units, CABAC and CBAC solutions use $bins/cycle$ and $Gbins/second$, which are not directly convertible and, therefore, do not allow a direct comparison. To go around the unfair comparison, we included the MRTR row in Table 6.3, which highlights and stands for the maximum real-time resolution for each of the analyzed designs. Hence, as one may notice, the results achieved for ASAP7, as well as for *sky130hs* and *nangate45* PDKs presented in Table 6.2, are in the same performance level when compared to the other designs displayed in Table 6.3 (i.e., real-time processing of 8K@120fps).

Figure 6.2: GDSII result for the AE-AV1 when synthesized for the ASAP7 PDK.



Source: The Author (2022).

The OpenROAD Flow arrangement (AJAYI et al., 2019) supports a flow starting at the RTL level and going all the way to the GDSII level. Fig. 6.2 presents the GDSII result for the AE-AV1 when synthesized for the ASAP7 PDK. As depicted in Table 6.2, the layout obtained 6.2 K gates count of area, and used six metal layers. Both VDD and GND are represented in layers of metals 1, 2, 5 and 6.

6.3 Low-Power and Multi-Boolean Approaches/Architectures

As presented throughout this work, a baseline version (i.e., AE-AV1) and additional versions of the AV1 arithmetic encoder architecture were introduced. This section displays the results for the AE-AV1-LP and AE-AV1-MB versions in, respectively, Subsections 6.3.1 and 6.3.2.

6.3.1 Low-Power AE-AV1-LP Results

The low-power adaptation of the baseline AE-AV1 architecture was named AE-AV1-LP and aims to include Clock Gating and Operand Isolation circuitry throughout the entire design to avoid, respectively, storing and calculating non-essential values. The AE-AV1-LP achieved a frequency of 563 MHz and an area of 11.2k gates count. Its average power consumption, taking into consideration all eighty-two presented video sequences coded, is 7.267 mW.

Tables 6.4 and 6.5, when combined, present an in-depth throughput rate and power consumption, respectively, analysis for each 720p video sequence taken from the *objective-2* dataset (MONTGOMERY, 2005) and encoded under constraints of quality 20, 32, 43 and 55, and configurations *good* and *allintra*. Table 6.6 present the same analysis for 1080p and 4K video sequences from Montgomery (2005).

The power results were accomplished by simulating all the synthesized designs with real-world data obtained from the encodings, using a lower frequency than their maximum (i.e., 400 MHz), the same testbench environment used for the verification and delays back-annotated to the netlist. After that, we executed analysis with the RC tool upon the architecture switches generated by the simulations. The *Boolean Rate* (i.e., column *Bool Rate*) is presented as a percentage of *Boolean* symbols over the total of encoded symbols, whereas the *Power Saving* column displays the percentage of power saved for each video when comparing AE-AV1 and AE-AV1-LP (e.g., 20, *allintra*, *Netflix Driving* consumes 8.926 mW in AE-AV1 and only 7.233 mW in AE-AV1-LP, representing 18.97% of power saving).

As one may notice, 4K video sequences lean towards requiring higher throughput rates than lower resolutions and, for the 4K sequences encoded under lower CQs, there is a tendency to present higher power-saving capabilities in the AE-AV1-LP design, which is highlighted by the tonality of the color green in the column *Power Saving* (i.e., the darker the green, the higher is the power saving).

The AE-AV1-LP version is capable of processing 8K@120fps videos, as it accomplishes 1.775 bits/cycle and 0.999 Gbits/second of throughput rates. Additionally, the AE-AV1-LP saves, on average, 19.89% of power when compared to AE-AV1.

Table 6.4: Power analysis for 720p videos from the *objective-2* dataset (MONTGOMERY, 2005) on the AE-AV1 and AE-AV1-LP designs.

CQ	Config.	Video	bits/cycle	AE-AV1 Power (mW)	AE-AV1-LP Power (mW)	Power Saving
20	*allintra*	Boat	1.827	9.367	7.569	19.20%
20	*allintra*	Dark	1.776	9.263	7.341	20.75%
20	*allintra*	KristenAndSara	1.787	8.984	7.222	19.61%
20	*allintra*	Netflix_Driving	1.746	8.926	7.233	18.97%
20	*allintra*	Netflix_RollerCoaster	1.782	8.692	6.946	20.09%
20	*good*	Boat	1.705	9.089	7.311	19.56%
20	*good*	Dark	1.668	9.004	7.333	18.56%
20	*good*	KristenAndSara	1.702	9.018	7.151	20.70%
20	*good*	Netflix_Driving	1.678	8.561	6.978	18.49%
20	*good*	Netflix_RollerCoaster	1.687	8.564	6.810	20.48%
32	*allintra*	Boat	1.876	8.557	6.767	20.92%
32	*allintra*	Dark	1.871	9.668	7.723	20.12%
32	*allintra*	KristenAndSara	1.831	9.139	7.339	19.70%
32	*allintra*	Netflix_Driving	1.785	8.987	7.203	19.85%
32	*allintra*	Netflix_RollerCoaster	1.828	9.050	7.251	19.88%
32	*good*	Boat	1.703	9.059	7.288	19.55%
32	*good*	Dark	1.673	9.481	7.670	19.10%
32	*good*	KristenAndSara	1.688	8.803	6.849	22.20%
32	*good*	Netflix_Driving	1.656	8.741	7.125	18.49%
32	*good*	Netflix_RollerCoaster	1.675	8.618	6.862	20.38%
43	*allintra*	Boat	1.924	8.441	6.683	20.83%
43	*allintra*	Dark	1.965	9.945	8.017	19.39%
43	*allintra*	KristenAndSara	1.882	9.356	7.511	19.72%
43	*allintra*	Netflix_Driving	1.846	9.156	7.283	20.46%
43	*allintra*	Netflix_RollerCoaster	1.886	9.403	7.556	19.64%
43	*good*	Boat	1.700	8.990	7.173	20.21%
43	*good*	Dark	1.687	9.542	7.577	20.59%
43	*good*	KristenAndSara	1.690	8.318	6.603	20.62%
43	*good*	Netflix_Driving	1.644	8.954	7.257	18.95%
43	*good*	Netflix_RollerCoaster	1.675	8.865	7.102	19.89%
55	*allintra*	Boat	1.876	8.557	6.767	20.92%
55	*allintra*	Dark	2.035	10.020	8.096	19.20%
55	*allintra*	KristenAndSara	1.948	9.387	7.539	19.69%
55	*allintra*	Netflix_Driving	1.936	9.459	7.629	19.35%
55	*allintra*	Netflix_RollerCoaster	1.961	9.709	7.823	19.43%
55	*good*	Boat	1.691	8.624	6.796	21.20%
55	*good*	Dark	1.706	9.444	7.497	20.62%
55	*good*	KristenAndSara	1.679	9.007	7.241	19.61%
55	*good*	Netflix_Driving	1.662	9.052	7.262	19.77%
55	*good*	Netflix_RollerCoaster	1.676	9.098	7.280	19.98%
Average			**1.775**	**9.072**	**7.267**	**19.91%**

Table 6.5: Throughput rate analysis for 720p videos from the *objective-2* dataset (MONTGOMERY, 2005) on the AE-AV1 and AE-AV1-LP designs.

CQ	Config.	Video	bits/cycle	AE-AV1 Gbits/sec	AE-AV1-LP Gbits/sec	Power Saving
20	*allintra*	Boat	1.827	1.062	1.028	19.20%
20	*allintra*	Dark	1.776	1.033	1.000	20.75%
20	*allintra*	KristenAndSara	1.787	1.039	1.005	19.61%
20	*allintra*	Netflix_Driving	1.746	1.015	0.982	18.97%
20	*allintra*	Netflix_RollerCoaster	1.782	1.036	1.003	20.09%
20	*good*	Boat	1.705	0.991	0.960	19.56%
20	*good*	Dark	1.668	0.970	0.939	18.56%
20	*good*	KristenAndSara	1.702	0.989	0.958	20.70%
20	*good*	Netflix_Driving	1.678	0.975	0.944	18.49%
20	*good*	Netflix_RollerCoaster	1.687	0.981	0.949	20.48%
32	*allintra*	Boat	1.876	1.090	1.055	20.92%
32	*allintra*	Dark	1.871	1.088	1.053	20.12%
32	*allintra*	KristenAndSara	1.831	1.065	1.030	19.70%
32	*allintra*	Netflix_Driving	1.785	1.038	1.005	19.85%
32	*allintra*	Netflix_RollerCoaster	1.828	1.063	1.029	19.88%
32	*good*	Boat	1.703	0.990	0.958	19.55%
32	*good*	Dark	1.673	0.973	0.942	19.10%
32	*good*	KristenAndSara	1.688	0.981	0.950	22.20%
32	*good*	Netflix_Driving	1.656	0.963	0.932	18.49%
32	*good*	Netflix_RollerCoaster	1.675	0.974	0.943	20.38%
43	*allintra*	Boat	1.924	1.119	1.083	20.83%
43	*allintra*	Dark	1.965	1.143	1.106	19.39%
43	*allintra*	KristenAndSara	1.882	1.094	1.059	19.72%
43	*allintra*	Netflix_Driving	1.846	1.073	1.039	20.46%
43	*allintra*	Netflix_RollerCoaster	1.886	1.097	1.061	19.64%
43	*good*	Boat	1.700	0.989	0.957	20.21%
43	*good*	Dark	1.687	0.981	0.950	20.59%
43	*good*	KristenAndSara	1.690	0.982	0.951	20.62%
43	*good*	Netflix_Driving	1.644	0.956	0.925	18.95%
43	*good*	Netflix_RollerCoaster	1.675	0.974	0.943	19.89%
55	*allintra*	Boat	1.876	1.090	1.055	20.92%
55	*allintra*	Dark	2.035	1.183	1.145	19.20%
55	*allintra*	KristenAndSara	1.948	1.132	1.096	19.69%
55	*allintra*	Netflix_Driving	1.936	1.126	1.089	19.35%
55	*allintra*	Netflix_RollerCoaster	1.961	1.140	1.103	19.43%
55	*good*	Boat	1.691	0.983	0.952	21.20%
55	*good*	Dark	1.706	0.992	0.960	20.62%
55	*good*	KristenAndSara	1.679	0.976	0.945	19.61%
55	*good*	Netflix_Driving	1.662	0.966	0.935	19.77%
55	*good*	Netflix_RollerCoaster	1.676	0.974	0.943	19.98%
Average			**1.775**	**1.032**	**0.999**	**19.91%**

Table 6.6: Throughput rate and power comparison between AE-AV1 and AE-AV1-LP for the videos *Aspen*, *Rush_Hour*, *Netflix_Narrator* and *Netflix_Dancers* from *objective-2* dataset (MONTGOMERY, 2005).

CQ	Config.	Resol.	Video	Bool Rate	$\frac{bits}{cycle}$	AE-AV1		AE-AV1-LP		Power Saving
						$\frac{Gbits}{sec}$	Power (mW)	$\frac{Gbits}{sec}$	Power (mW)	
20	*allintra*	1080p	♣	26.50%	1.763	1.025	9.131	0.993	7.32	19.83%
20	*allintra*	1080p	♡	25.26%	1.802	1.047	8.95	1.014	7.14	20.22%
20	*allintra*	4K	♦	15.06%	1.863	1.082	6.776	1.049	4.645	31.45%
20	*allintra*	4K	⋆	21.38%	1.820	1.057	8.04	1.025	6.245	22.33%
20	*good*	1080p	♣	20.68%	1.709	0.993	8.69	0.962	6.785	21.92%
20	*good*	1080p	♡	19.85%	1.700	0.988	8.41	0.957	6.44	23.42%
20	*good*	4K	♦	19.53%	1.763	1.024	8.926	0.993	7.143	19.98%
20	*good*	4K	⋆	16.21%	1.770	1.028	8.444	0.997	6.713	20.50%
32	*allintra*	1080p	♣	23.33%	1.793	1.042	9.065	1.009	7.189	20.69%
32	*allintra*	1080p	♡	19.45%	1.864	1.083	9.15	1.049	7.286	20.37%
32	*allintra*	4K	♦	11.46%	2.001	1.162	7.994	1.126	6.431	19.55%
32	*allintra*	4K	⋆	14.21%	1.921	1.116	9.329	1.082	7.457	20.07%
32	*good*	1080p	♣	16.77%	1.680	0.976	8.312	0.946	6.486	21.97%
32	*good*	1080p	♡	17.32%	1.681	0.977	8.346	0.947	6.969	16.50%
32	*good*	4K	♦	20.82%	1.729	1.004	8.876	0.973	6.575	25.92%
32	*good*	4K	⋆	14.57%	1.759	1.022	8.43	0.990	6.483	23.10%
43	*allintra*	1080p	♣	21.82%	1.830	1.063	8.945	1.030	7.029	21.42%
43	*allintra*	1080p	♡	17.86%	1.910	1.110	9.203	1.075	7.339	20.25%
43	*allintra*	4K	♦	6.63%	2.095	1.217	7.546	1.179	6.051	19.81%
43	*allintra*	4K	⋆	10.81%	2.001	1.163	9.848	1.127	7.989	18.88%
43	*good*	1080p	♣	13.50%	1.654	0.961	9.083	0.931	7.272	19.94%
43	*good*	1080p	♡	13.88%	1.657	0.963	8.682	0.933	6.888	20.66%
43	*good*	4K	♦	13.71%	1.739	1.010	8.056	0.979	5.897	26.80%
43	*good*	4K	⋆	12.31%	1.734	1.008	7.529	0.976	5.684	24.51%
55	*allintra*	1080p	♣	18.44%	1.903	1.106	8.849	1.072	6.943	21.54%
55	*allintra*	1080p	♡	15.12%	1.979	1.150	9.164	1.114	7.346	19.84%
55	*allintra*	4K	♦	2.87%	2.162	1.256	6.78	1.217	5.413	20.16%
55	*allintra*	4K	⋆	7.50%	2.096	1.218	9.658	1.180	7.872	18.49%
55	*good*	1080p	♣	9.40%	1.615	0.938	9.261	0.909	7.41	19.99%
55	*good*	1080p	♡	9.47%	1.591	0.924	8.888	0.896	7.047	20.71%
55	*good*	4K	♦	5.90%	1.485	0.863	8.774	0.836	6.941	20.89%
55	*good*	4K	⋆	11.19%	1.599	0.929	7.526	0.900	5.816	22.72%
Average				**15.40%**	**1.787**	**1.047**	**8.583**	**1.015**	**6.758**	**21.39%**

♣ Aspen
♡ Rush
♦ Netflix Dancers
⋆ Netflix Narrator

6.3.2 Multi-Boolean AE-AV1-MB Results

As already presented throughout Section 5.2, the AE-AV1-MB versions of the AV1 arithmetic encoder architecture were created as a successful attempt to increase the throughput rates when comparing to the baseline AE-AV1 version. The AE-AV1-MB, however, deals with constraints inherent to the number of *Boolean* blocks added into pipeline stages 2 and 3. For instance, when including four *Boolean* blocks into the cited stages, the design suffers a severe frequency drop (i.e., $\beta \geq \lambda$) that directly affects the throughput rate per second and making the architecture incapable of real-time processing of 8K@120fps. Hence, the herein presented results do not include the 4-bool version of the AE-AV1-MB design.

Table 6.7 depicts the 1-, 2- and 3-bool versions of the AE-AV1-MB design and their respective throughput rates per second in Gbits/second for the videos *Boat*, *Dark*, *KristenAndSara*, *Netflix_Driving* and *Netflix_RollerCoaster* from the *objective-2* dataset. On the other hand, Table 6.8 displays the throughput results for the remaining videos (i.e., *Aspen*, *Rush_Hour*, *Netflix_Narrator* and *Netflix_Dancers* from *objective-2* (MONTGOMERY, 2005), and *Beauty*, *Bosphorus*, *HoneyBee*, *Jockey*, *ReadySetGo* and *YachtRide* from Mercat, Viitanen e Vanne (2020) dataset).

As one may notice after analyzing Tables 6.7 and 6.8, there is a slightly difference between the 2-bool and 3-bool versions of the AE-AV1-MB (i.e., in the former table 2-bool has higher average throughput rate, whereas in the latter table the 3-bool gains). When adding the architectural information of each version, which is presented within Table 6.9, one may conclude that the 2-bool version is presented as the best trade-off between throughput gains and area and power consumption losses.

Therefore, when comparing the throughput rates of both 2- and 3-bool versions with the AE-AV1-MB 1-bool (similar to the baseline AE-AV1 design), one may find that they gain, respectively, 10.81% and 8.23%. The area and power consumption losses for 2-bool version are 39.31% and 51.18%, whereas the 3-bool version presents 73.50% and 82.32%. Finally, one may conclude the superiority of the 2-bool version over the 1-, 3- and 4-bool versions.

As explained in Section 6.1, the power consumption estimation is done by simulating a given architecture's behavior with real-world data (i.e., video sequences) and using the resulting TCF file in the Cadence™ RC tool. The simulations of the AE-AV1-MB versions, which are limited by time, are capable of encoding a different number of

Table 6.7: Throughput rate comparison between the AE-AV1-MB 1-, 2- and 3-bool versions for the video sequences *Boat*, *Dark*, *KristenAndSara*, *Netflix_Driving* and *Netflix_RollerCoaster* from the *objective-2* dataset (MONTGOMERY, 2005).

Video	CQ	Config.	Bool Rate	1-bool Gb/sec	2-bool Gb/sec	3-bool Gb/sec
Boat	20	allintra	19.83%	1.062	1.197	1.177
Dark	20	allintra	24.41%	1.033	1.194	1.186
KristenAndSara	20	allintra	22.99%	1.039	1.192	1.180
Netflix_Driving	20	allintra	26.32%	1.015	1.187	1.183
Netflix_RollerCoaster	20	allintra	24.17%	1.036	1.196	1.187
Boat	20	good	19.07%	0.991	1.113	1.092
Dark	20	good	22.32%	0.970	1.108	1.095
KristenAndSara	20	good	20.05%	0.989	1.117	1.098
Netflix_Driving	20	good	18.70%	0.975	1.092	1.071
Netflix_RollerCoaster	20	good	16.46%	0.981	1.085	1.059
Boat	32	allintra	16.36%	1.090	1.206	1.177
Dark	32	allintra	18.08%	1.088	1.214	1.189
KristenAndSara	32	allintra	19.53%	1.065	1.198	1.177
Netflix_Driving	32	allintra	23.18%	1.038	1.192	1.180
Netflix_RollerCoaster	32	allintra	21.27%	1.063	1.208	1.191
Boat	32	good	16.56%	0.990	1.096	1.070
Dark	32	good	19.06%	0.973	1.092	1.071
KristenAndSara	32	good	19.27%	0.981	1.102	1.082
Netflix_Driving	32	good	16.69%	0.963	1.067	1.042
Netflix_RollerCoaster	32	good	14.05%	0.974	1.064	1.033
Boat	43	allintra	12.79%	1.119	1.213	1.176
Dark	43	allintra	12.63%	1.143	1.238	1.199
KristenAndSara	43	allintra	15.75%	1.094	1.206	1.175
Netflix_Driving	43	allintra	18.79%	1.073	1.203	1.180
Netflix_RollerCoaster	43	allintra	17.67%	1.097	1.221	1.195
Boat	43	good	14.19%	0.989	1.081	1.050
Dark	43	good	19.47%	0.981	1.104	1.084
KristenAndSara	43	good	17.08%	0.982	1.090	1.066
Netflix_Driving	43	good	15.37%	0.956	1.051	1.024
Netflix_RollerCoaster	43	good	12.14%	0.974	1.053	1.019
Boat	55	allintra	16.34%	1.090	1.206	1.177
Dark	55	allintra	8.44%	1.183	1.254	1.205
KristenAndSara	55	allintra	10.81%	1.132	1.216	1.173
Netflix_Driving	55	allintra	12.90%	1.126	1.222	1.184
Netflix_RollerCoaster	55	allintra	12.81%	1.140	1.237	1.198
Boat	55	good	11.08%	0.983	1.057	1.021
Dark	55	good	14.55%	0.992	1.086	1.056
KristenAndSara	55	good	14.71%	0.976	1.070	1.041
Netflix_Driving	55	good	13.01%	0.966	1.050	1.017
Netflix_RollerCoaster	55	good	9.75%	0.974	1.040	1.002
Averages			**16.97%**	**1.032**	**1.145**	**1.120**

Table 6.8: Throughput rate comparison between the AE-AV1-MB 1-, 2- and 3-bool versions for the video sequences *Boat*, *Dark*, *KristenAndSara*, *Netflix_Driving* and *Netflix_RollerCoaster* from the *objective-2* dataset (MONTGOMERY, 2005).

Video	CQ	Config.	Bool Rate	1-bool mW	2-bool mW	3-bool mW
Boat	20	allintra	19.83%	9.278	13.515	16.206
Dark	20	allintra	24.41%	9.184	13.837	16.565
KristenAndSara	20	allintra	22.99%	8.971	13.657	16.518
Netflix_Driving	20	allintra	26.32%	8.964	14.126	17.248
Netflix_RollerCoaster	20	allintra	24.17%	8.655	13.501	16.512
Boat	20	good	19.07%	9.049	13.797	16.730
Dark	20	good	22.32%	8.997	13.779	16.569
KristenAndSara	20	good	20.05%	8.998	13.870	16.799
Netflix_Driving	20	good	18.70%	8.642	13.789	17.173
Netflix_RollerCoaster	20	good	16.46%	8.510	13.269	16.212
Boat	32	allintra	16.36%	8.381	12.388	14.741
Dark	32	allintra	18.08%	9.563	14.029	16.619
KristenAndSara	32	allintra	19.53%	9.065	13.594	16.348
Netflix_Driving	32	allintra	23.18%	8.951	13.775	16.707
Netflix_RollerCoaster	32	allintra	21.27%	9.007	13.869	16.785
Boat	32	good	16.56%	8.996	13.656	16.530
Dark	32	good	19.06%	9.416	14.156	16.948
KristenAndSara	32	good	19.27%	8.670	12.999	15.654
Netflix_Driving	32	good	16.69%	8.801	13.997	17.355
Netflix_RollerCoaster	32	good	14.05%	8.574	13.339	16.368
Boat	43	allintra	12.79%	8.279	11.908	14.034
Dark	43	allintra	12.63%	9.869	14.250	16.776
KristenAndSara	43	allintra	15.75%	9.263	13.609	16.195
Netflix_Driving	43	allintra	18.79%	9.052	13.489	16.197
Netflix_RollerCoaster	43	allintra	17.67%	9.336	13.939	16.701
Boat	43	good	14.19%	8.881	13.263	15.765
Dark	43	good	19.47%	9.473	14.094	16.790
KristenAndSara	43	good	17.08%	8.247	12.634	15.431
Netflix_Driving	43	good	15.37%	8.996	14.110	17.272
Netflix_RollerCoaster	43	good	12.14%	8.843	13.721	16.701
Boat	55	allintra	16.34%	8.381	12.388	14.741
Dark	55	allintra	8.44%	9.934	14.159	16.592
KristenAndSara	55	allintra	10.81%	9.265	13.317	15.667
Netflix_Driving	55	allintra	12.90%	9.367	13.480	15.856
Netflix_RollerCoaster	55	allintra	12.81%	9.623	13.951	16.450
Boat	55	good	11.08%	8.440	12.427	14.847
Dark	55	good	14.55%	9.339	13.848	16.507
KristenAndSara	55	good	14.71%	8.978	13.684	16.570
Netflix_Driving	55	good	13.01%	9.028	13.879	16.840
Netflix_RollerCoaster	55	good	9.75%	9.035	13.834	16.698
Averages			**16.97%**	**9.008**	**13.573**	**16.330**

Table 6.9: Frequency, area, power consumption, PC/EB (Power Consumption per Encoded Bit) and throughput rate per cycle comparison between the 1-, 2- and 3-bool versions of AE-AV1-MB.

	1-bool	**2-bool**	**3-bool**
Frequency (MHz)	581	590	559
Gates Count (K)	11.7k	16.3k	20.3k
bits/cycle	1.734	1.892	1.952
Gbits/second	1.008	1.117	1.091
Power (mW)	8.924	13.492	16.271
PC/EB[1]	1.236	1.678	1.948
MRTR[2]	8K@120fps	8K@120fps	8K@120fps

[1] Power Consumption per Encoded Bit ($\mu W/bit$)
[2] Maximum Real-Time Resolution

bits according to the AE-AV1-MB version used. Hence, in order to accurately compare the power consumption results for the AE-AV1-MB versions, Table 6.9 introduces the variable PC/EB (Power Consumption per Encoded Bit). When taking into consideration the PC/EB comparison between 2- and 3-bool against 1-bool, the power losses are, respectively, 35.76% and 57.60%.

6.4 Comparisons

As part of the effort to effectively design an AV1 arithmetic encoder architecture, one key step is to compare the solution created with related works. Comparisons allow the development team to understand what are the strengths and flaws of the designed solution to then, if feasible, improve the architecture. This section compares all versions of the AV1 arithmetic encoder architecture (i.e., AE-AV1, AE-AV1-LP, and AE-AV1-MB) among themselves and to other works found within the literature.

As presented in Subsections 1.2 and 3.3, some related works might be classified as, respectively, AV1 hardware designs targeting different blocks or arithmetic coding architectures targeting different video coding formats. Table 6.10 presents a comparison between the works introduced in this document and works that target other blocks of the AV1 codec. As presented in Table 6.11, the variable MRTR (Maximum Real-Time Resolution) was added into Table 6.10 to facilitate the comparison as the each block of the AV1 and each video codec have different throughput rate constraints. Moreover, for the sake of simplifying Tables 6.10 and 6.11, only the versions 2-bool and 3-bool of AE-AV1-MB were added due to their better trade-off (throughput gains *vs.* area and power

Table 6.10: Comparison between the herein presented works (i.e., AE-AV1, AE-AV1-LP, AE-AV1-MB 2- and 3-bool) and other works targeting the AV1 codec (i.e., (DOMANSKI et al., 2019; CORREA et al., 2019b; CORREA et al., 2019a; ZUMMACH et al., 2020; CORREA et al., 2020; NETO et al., 2020; FREITAS et al., 2021; DOMANSKI et al., 2021)).

	Coder	Tech. (nm)	Freq. (MHz)	Gates Count	bits/cycle	Gbits/sec	Power (mW)	MRTR[1]
◯	MC[5]	40[4]	279.9	141.1K	-	-	81.3	8K@30
△	Intra[6]	40[4]	648	109.6K	-	-	16.1	4K@30
†	Intra[6]	40[4]	315	247.3K	-	-	268.3	4K@120
♣	CDEF[7]	40[4]	23	369K	-	-	65	4K@60
⋆	Intra[6]	40[4]	648	128.5K	-	-	65.5	4K@30
♢	Intra[6]	40[4]	686	72.6K	-	-	26.79	8K@30
♠	IF[8]	40[4]	476	691.7K	-	-	382	1K@60
♦	FME[9]	65[3]	345	270.4K	-	-	130.7	2K@120
♡	AE[2]	65[3]	581	11.7k	1.775	1.032	9.072	8K@120
§	AE[2]	65[3]	563	11.2k	1.775	0.999	7.267	8K@120
♯	AE[2]	65[3]	590	16.3k	1.892	1.117	13.492	8K@120
□	AE[2]	65[3]	559	20.3k	1.952	1.091	16.271	8K@120

[1] Maximum Real-Time Resolution
[2] Arithmetic Encoder
[3] ST 65nm PDK
[4] TSMC 40nm PDK
[5] Motion Compensation (MC)
[6] Intra-prediction
[7] Constrained Directional Enhancement Filter (CDEF)
[8] Interpolation Filter (IF)
[9] Fractional Motion Estimation (FME)

◯ (DOMANSKI et al., 2019)
△ (CORREA et al., 2019b)
† (CORREA et al., 2019a)
♣ (ZUMMACH et al., 2020)
⋆ (CORREA et al., 2020)
♢ (NETO et al., 2020)
♠ (FREITAS et al., 2021)
♦ (DOMANSKI et al., 2021)
♡ AE-AV1
§ AE-AV1-LP
♯ AE-AV1-MB 2-bool
□ AE-AV1-MB 3-bool

consumption losses) than the 3-bool.

As one may notice, the AV1 related works presented within Table 6.10 cannot be fairly compared to the arithmetic encoder designs introduced in the thesis, but the authors claim as maximum real-time resolution (MRTR) supported by their designs 8K@30fps, 4K@(30, 60, and 120)fps, going as low as 1K@120fps resolutions. The herein introduced designs are all capable of reaching up to 8K@120fps, which is also the maximum supported quality for the AV1 format (RIVAZ; HAUGHTON, 2019). Furthermore, one may notice that the related works targeted an interesting variety of modules of the AV1 codec, but none of them targeted the Arithmetic Encoder, or even the Entropy Encoder, which makes the designs presented within this thesis unique at this time.

With regards to the arithmetic encoder and decoder works presented in Table 6.11, one will find works targeting the AV1 Decoder, and HEVC and AVS 2.0 Encoders. More-

Table 6.11: Comparison between the herein presented works (i.e., AE-AV1, AE-AV1-LP, AE-AV1-MB 2- and 3-bool) and other works targeting the Arithmetic Encoders/Decoders (i.e., (GOMES; RAMOS, 2021; GOMES et al., 2022; ZHOU et al., 2015; PASTUSZAK, 2020a; RAMOS et al., 2021; CHOI; LEE; CHAE, 2021)).

	Video Codec	**Coder**	**Tech. (nm)**	**Freq. (MHz)**	**Gates Count**	$\frac{bins}{cycle}$	$\frac{Gbins}{sec}$	**Power (mW)**	**MRTR**[1]
□	AV1	AD[10]	65[3]	462	31.3 K	1.62	0.748	26.8	8K@60
◯	AV1	AD[10]	65[3]	473	35 K	1.62	0.766	14.3	8K@60
△	HEVC	CABAC	90[4]	420	64.1 K	4.37	1.836	-	8K@120
†	HEVC	CABAC	90[4]	700	120.4 K	9.65	6.755	111.7	8K@120
♣	HEVC	CABAC	65[3]	507	21.22 K	4.31	2.185	26.18	8K@120
⋆	AVS 2.0	CBAC	65[4]	735	133.5 K	4.78	3.51	79.6	8K
♡	AV1	AE[2]	65[3]	581	11.7k	1.775[11]	1.032[12]	9.072	8K@120
§	AV1	AE[2]	65[3]	563	11.2k	1.775[11]	0.999[12]	7.267	8K@120
♯	AV1	AE[2]	65[3]	590	16.3k	1.892[11]	1.117[12]	13.492	8K@120
♦	AV1	AE[2]	65[3]	559	20.3k	1.952[11]	1.091[12]	16.271	8K@120

[1] Maximum Real-Time Resolution
[2] Arithmetic Encoder
[3] ST PDKs
[4] TSMC PDKs
[5] Motion Compensation (MC)
[6] Intra-prediction
[7] Constrained Directional Enhancement Filter (CDEF)
[9] Fractional Motion Estimation (FME)
[8] Interpolation Filter (IF)
[10] Arithmetic Decoder
[11] bits/cycle
[12] Gbits/sec

□ (GOMES; RAMOS, 2021)
◯ (GOMES et al., 2022)
△ (ZHOU et al., 2015)
† (PASTUSZAK, 2020a)
♣ (RAMOS et al., 2021)
⋆ (CHOI; LEE; CHAE, 2021)

♡ AE-AV1
§ AE-AV1-LP
♯ AE-AV1-MB 2-bool
♦ AE-AV1-MB 3-bool

over, all the works presented within Table 6.11 are capable of reaching at least 8K@60fps and have chip area and power consumption superiors than the designs proposed in this thesis.

Therefore, after analyzing the comparisons between the designs introduced in this thesis and related works, either targeting other blocks of the AV1 codec or Arithmetic Encoder/Decoder blocks of other coding formats, one shall conclude that all versions introduced here are capable of achieving similar results with the related works. Furthermore, the introduced designs consume less power and have lower gates count.

The works proposed by Gomes e Ramos (2021) and Gomes et al. (2022) present similar results to the versions introduced in this work, but have lower MRTR (i.e., 8K@60fps).

7 CONCLUSION

Video coding is a very complex and challenging technology for both industry and academia. In one side (i.e., industry), video codecs are critical tools for allowing the continuous growth in number of concurrent users, as well as video quality improvements in streaming providers (e.g., TikTok, YouTube, Netflix, etc.). On the other side (i.e., academia), video coding is a highly difficult concept that needs to be tackled to allow video technology to keep improving and to reach levels previously unimaginable (e.g., 16K).

The AV1 coding format, which was introduced by the Alliance for Open Media (AOMedia) in 2018, is a very capable codec that supports up to 8K@120fps video processing, and introduces novel and enhanced techniques for its core components (e.g., intra-prediction, inter-prediction). The AV1, however, has a sequential logic located at the final part of the codification process, which is the so-called Entropy Coding, that is seen as bottleneck due to the necessity for encoding all Syntax Elements and for the difficulty for this block to deal with parallelizations.

This work proposed a baseline design (i.e., AE-AV1) to accelerate the AV1 Arithmetic Encoder block, which is located inside the Entropy Encoder. By applying hardware designing techniques, such as pipelining, the AE-AV1 architecture was able to achieve ultra-high performance (i.e., 8K@120fps real-time processing).

Furthermore, this Master's thesis also developed a new and low-power design (i.e., AE-AV1-LP) and a new AE-AV1-MB architecture, where the latter comprises a novel parallelization strategy, which was also proposed in this work, called Multi-Boolean. With regards to the former, the low-power techniques clock gating and operand isolation were used to allow for a power consumption reduction, whereas the latter presents a trade-off between area and power increases, and improvements on throughput rates.

Therefore, this work introduced three novel designs (i.e., AE-AV1, AE-AV1-LP and AE-AV1-MB) to accelerate the AV1 Arithmetic Encoder block and compared these architectures with published related works that target (i) Arithmetic Encoders, and (ii) different blocks of the AV1 codec. This thesis, as well as the manuscripts published by the author, are the first ever works to tackle the AV1 Arithmetic Encoder.

REFERENCES

AJAYI, T. et al. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. **Government Microcircuit Applications and Critical Technology Conference**, p. 1105–1110, 2019.

Alliance for Open Media. **AV1 Reference Software**. 2020. Disponível em: <https://aomedia.googlesource.com/aom/>.

AOMedia. **Alliance for Open Media**. 2020. Disponível em: <http://aomedia.org/>.

AOMedia. **They Developed It. They Benefit From It. They Stand Behind It. | Alliance for Open Media**. 2021. Disponível em: <https://aomedia.org/membership/members/>.

BAMPIS, C. G. et al. SpEED-QA: Spatial Efficient Entropic Differencing for Image and Video Quality. **IEEE Signal Processing Letters**, v. 24, n. 9, p. 1333–1337, 9 2017. ISSN 1070-9908.

BHATIA, H. **Strong Winds Loom Against the Video Codec Successor HEVC, Challenging the Licensing Model**. 2018. Disponível em: <https://www.counterpointresearch.com/13270/>.

BITENCOURT, T. P.; RAMOS, F. L. L.; BAMPI, S. High-Throughput and Low-Power Architectures for the AV1 Arithmetic Encoder. In: **2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)**. IEEE, 2021. p. 1–6. ISBN 978-1-6654-2170-6. Disponível em: <https://ieeexplore.ieee.org/document/9529994/>.

BITENCOURT, T. P.; RAMOS, F. L. L.; BAMPI, S. Power-Saving 8K Real-Time AV1 Arithmetic Encoder Architecture. **IEEE Design & Test**, p. 1–1, 2022. ISSN 2168-2356. Disponível em: <https://ieeexplore.ieee.org/document/9800932/>.

BITENCOURT, T. P.; RAMOS, F. L. L.; BAMPI, S. Power-Throughput Trade-off Analysis for a Novel Multi-Boolean AV1 Arithmetic Encoder Design. In: **2022 Picture Coding Symposium (PCS)**. [S.l.]: IEEE, 2022. p. 25–29. ISBN 978-1-6654-9257-7.

BITENCOURT, T. P.; RAMOS, L.; BAMPI, S. AV1 Arithmetic Encoder Design on Open-Source EDA. **Journal of Integrated Circuits and Systems**, v. 17, n. 2, p. 1–9, 2022. Disponível em: <https://jics.org.br/ojs/index.php/JICS/article/view/564>.

Cadence. **Incisive SystemC, VHDL, and Verilog Simulation**. 2021. Disponível em: <https://www.cadence.com/en_US/home/training/all-courses/82115.html>.

CHAI, D.; BOUZERDOUM, A. A Bayesian approach to skin color classification in YCbCr color space. In: **2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No.00CH37119)**. IEEE, 2000. p. 421–424. ISBN 0-7803-6355-8. Disponível em: <https://ieeexplore.ieee.org/abstract/document/888774>.

CHEN, W. et al. RGB color decomposition and image feature extraction of flame image in rear of sintering machine. In: **2017 36th Chinese Control Conference (CCC)**. [S.l.]: IEEE, 2017. p. 5460–5463. ISBN 978-988-15639-3-4.

CHEN, Y. et al. An Overview of Core Coding Tools in the AV1 Video Codec. In: **2018 Picture Coding Symposium (PCS)**. [s.n.], 2018. p. 41–45. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8456249>.

CHOI, Y.-K.; LEE, H.-J.; CHAE, S.-I. High Throughput CBAC Hardware Encoder With Bin Merging for AVS 2.0 Video Coding. **IEEE Transactions on Circuits and Systems for Video Technology**, Institute of Electrical and Electronics Engineers Inc., v. 31, n. 11, p. 4439–4453, 11 2021. ISSN 1051-8215. Disponível em: <https://ieeexplore.ieee.org/document/9310296/>.

CLARK, L. T. et al. ASAP7: A 7-nm finFET predictive process design kit. **Microelectronics Journal**, Elsevier, v. 53, p. 105–115, 7 2016. ISSN 0026-2692.

CORREA, M. et al. A High Throughput Hardware Architecture Targeting the AV1 Paeth Intra Predictor. In: **2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)**. IEEE, 2019. p. 93–96. ISBN 978-1-7281-0453-9. Disponível em: <https://ieeexplore.ieee.org/document/8667544/>.

CORREA, M. et al. High Throughput Hardware Design for AV1 Paeth and Smooth Intra Modes. In: **2019 IEEE International Symposium on Circuits and Systems (ISCAS)**. IEEE, 2019. p. 1–5. ISBN 978-1-7281-0397-6. Disponível em: <https://ieeexplore.ieee.org/document/8702258/>.

CORREA, M. M. et al. A High-Throughput Hardware Architecture for AV1 Non-Directional Intra Modes. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 67, n. 5, p. 1481–1494, 5 2020. ISSN 1549-8328. Disponível em: <https://ieeexplore.ieee.org/document/8998190/>.

Dajiang Zhou et al. Ultra-High-Throughput VLSI Architecture of H.265/HEVC CABAC Encoder for UHDTV Applications. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 25, n. 3, p. 497–507, 3 2015. ISSN 1051-8215. Disponível em: <http://ieeexplore.ieee.org/document/6851145/>.

DENG, Z.; MOCCAGATTA, I. Hardware-friendly inter prediction techniques for AV1 video coding. In: **2017 IEEE International Conference on Image Processing (ICIP)**. IEEE, 2017. p. 948–952. ISBN 978-1-5090-2175-8. Disponível em: <http://ieeexplore.ieee.org/document/8296421/>.

DIMITRAKOPOULOS, G. et al. Low-Power Leading-Zero Counting and Anticipation Logic for High-Speed Floating Point Units. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 16, n. 7, p. 837–850, 7 2008. ISSN 1063-8210. Disponível em: <http://ieeexplore.ieee.org/document/4539802/>.

DOBRUSHIN, R. Mathematical problems in the Shannon theory of optimal coding of information. In: **Proc. 4th Berkeley Symp. Mathematics, Statistics, and Probability**. [S.l.: s.n.], 1961. p. 211–252.

DOMANSKI, R. et al. High-Throughput Multifilter Interpolation Architecture for AV1 Motion Compensation. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 66, n. 5, p. 883–887, 5 2019. ISSN 1549-7747. Disponível em: <https://ieeexplore.ieee.org/document/8682095/>.

DOMANSKI, R. et al. Low-Power and High-Throughput Approximated Architecture for AV1 FME Interpolation. In: **2021 IEEE International Symposium on Circuits and Systems (ISCAS)**. IEEE, 2021. v. 2021-May, p. 1–5. ISBN 978-1-7281-9201-7. ISSN 02714310. Disponível em: <https://ieeexplore.ieee.org/document/9401224/>.

EDWARDS, R. T. Google/SkyWater and the Promise of the Open PDK. 2020. Disponível em: <https://woset-workshop.github.io/PDFs/2020/a03.pdf>.

EDWARDS, R. T.; SHALAN, M.; KASSEM, M. Real Silicon Using Open-Source EDA. **IEEE Design & Test**, IEEE Computer Society, v. 38, n. 2, p. 38–44, 4 2021. ISSN 2168-2356. Disponível em: <https://ieeexplore.ieee.org/document/9336682/>.

ENE, T.-D. DESIGN AND CHARACTERIZATION OF A STANDARD CELL LIBRARY FOR THE SKYWATER 130NM PROCESS. 2020.

FELLER, C. et al. The VP8 video codec - overview and comparison to H.264/AVC. **2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)**, p. 57–61, 9 2016. ISSN 2166-6814. Disponível em: <https://api.semanticscholar.org/CorpusID:37800094#id-name=S2CID>.

FREITAS, D. et al. Hardware Architecture for the Regular Interpolation Filter of the AV1 Video Coding Standard. In: **2020 28th European Signal Processing Conference (EUSIPCO)**. IEEE, 2021. v. 2021-Janua, p. 560–564. ISBN 978-9-0827-9705-3. ISSN 22195491. Disponível em: <https://ieeexplore.ieee.org/document/9287551/>.

GANGURDE, S.; TIWARI, K. LSB Steganography Using Pixel Locator Sequence with AES. 12 2020. Disponível em: <http://arxiv.org/abs/2012.02494>.

GHAZY, A. A.; SHALAN, M. OpenLANE: The Open-Source Digital ASIC Implementation Flow. In: **Workshop on Open-Source EDA Technology (WOSET)**. [S.l.: s.n.], 2020.

GOMES, J. S. et al. Low-Power High-Throughput Architecture for AV1 Arithmetic Decoder. **IEEE Design & Test**, p. 1–1, 2022. ISSN 2168-2356. Disponível em: <https://ieeexplore.ieee.org/document/9807340/>.

GOMES, J. S.; RAMOS, F. L. L. High-Performance Design for the AV1 Multi - Alphabet Arithmetic Decoder. In: **2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)**. IEEE, 2021. p. 1–6. ISBN 978-1-6654-2170-6. Disponível em: <https://ieeexplore.ieee.org/document/9529970/>.

Google. **SkyWater PDK Github**. 2022. Disponível em: <https://github.com/google/skywater-pdk>.

GRANGE, A.; RIVAZ, P. de; HUNT, J. **VP9 Bitstream & Decoding Process Specification v0.6**. [S.l.], 2016. Disponível em: <https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.6-20160331-draft.pdf>.

Graphic Communications. **Flip Book Project**. 2016. Disponível em: <http://grid14.weebly.com/assignments/flip-book-project>.

HAN, J. et al. A Technical Overview of AV1. p. 1–25, 8 2020. Disponível em: <http://arxiv.org/abs/2008.06091>.

HUFFMAN, D. A. A Method for the Construction of Minimum-Redundancy Codes. **Proceedings of the IRE**, v. 40, n. 9, p. 1098–1101, 1952. ISSN 00968390.

INFN. **Power Analysis**. 2021. Disponível em: <https://wiki.to.infn.it/vlsi/workbook/digital/syn/power>.

ITU-R. **Recommendation ITU-R BT.601-7 Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios BT Series Broadcasting service (television)**. [S.l.], 2011. Disponível em: <https://www.itu.int/rec/R-REC-BT.601/>.

ITU-T. **H.261 : Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88)**. [s.n.], 1988. Disponível em: <http://www.itu.int/rec/T-REC-H.261-198811-S/en>.

ITU-T. **ITU-T Recommendation declared patent(s)**. [S.l.], 1989. Disponível em: <https://www.itu.int/ITU-T/recommendations/related_ps.aspx?id_prod=1088>.

ITU-T; ISO/IEC. **Advanced video coding for generic audiovisual services, ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC)**. [S.l.], 2003. Disponível em: <https://www.itu.int/rec/T-REC-H.264-201906-I/en>.

ITU-T; ISO/IEC. **High Efficiency Video Coding, ITU-T Recommendation H.265 and ISO/IEC 23008-2**. [S.l.], 2013. Disponível em: <https://www.itu.int/rec/T-REC-H.265-201304-S/en>.

ITU-T; ISO/IEC. **Infrastructure of audiovisual services – Coding of moving video (H.266)**. [S.l.], 2020.

KAHNG, A. B.; SPYROU, T. The OpenROAD Project: Unleashing Hardware Innovation. 2021. Disponível em: <https://theopenroadproject.org/>.

KOMMURU, H. B.; MAHMOODI, H. ASIC design flow tutorial using synopsys tools. **Nano-Electronics \& Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring**, 2009.

MARPE, D.; SCHWARZ, H.; WIEGAND, T. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 13, n. 7, p. 620–636, 7 2003. ISSN 1051-8215. Disponível em: <http://ieeexplore.ieee.org/document/1218195/>.

Maxim Integrated. **Understanding Analog Video Signals**. 2002. Disponível em: <https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1184.html>.

MERCAT, A.; VIITANEN, M.; VANNE, J. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In: **Proceedings of the 11th ACM Multimedia Systems Conference**. [S.l.: s.n.], 2020. p. 297–302.

MIAO, J.; LI, S. A design for high speed leading-zero counter. In: **Proceedings of the International Symposium on Consumer Electronics, ISCE**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. p. 22–23. ISBN 9781538654330.

MONTGOMERY, C. **Derf's Test Media Collection**. 2005. Disponível em: <https://media.xiph.org/video/derf/>.

MUKHERJEE, D. et al. The latest open-source video codec VP9 - An overview and preliminary results. In: **2013 Picture Coding Symposium (PCS)**. IEEE, 2013. p. 390–393. ISBN 978-1-4799-0294-1. Disponível em: <http://ieeexplore.ieee.org/document/6737765/>.

MUKHERJEE, D. et al. An overview of new video coding tools under consideration for VP10: the successor to VP9. In: TESCHER, A. G. (Ed.). [s.n.], 2015. p. 95991E. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2191104>.

MUNCH, M. et al. Automating RT-level operand isolation to minimize power consumption in datapaths. In: **Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)**. IEEE Comput. Soc, 2005. p. 624–631. ISBN 0-7695-0537-6. Disponível em: <http://ieeexplore.ieee.org/document/840850/>.

NETO, L. et al. Directional Intra Frame Prediction Architecture with Edge Filter and Upsampling for AV1 Video Coding. In: **2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)**. IEEE, 2020. p. 1–6. ISBN 978-1-7281-9625-1. Disponível em: <https://ieeexplore.ieee.org/document/9189902/>.

NOWAK, P. M.; KOŚCIELNIAK, P. What Color Is Your Method? Adaptation of the RGB Additive Color Model to Analytical Method Evaluation. **Analytical Chemistry**, v. 91, n. 16, p. 10343–10352, 8 2019. ISSN 0003-2700.

OpenROAD. **OpenROAD – Foundations and Realization of Open and Accessible Design**. 2021. Disponível em: <https://theopenroadproject.org/>.

OWEN, D. **Frame Rates**. 2012. Disponível em: <https://www.mediacollege.com/video/frame-rate/>.

PASTUSZAK, G. Generative Multi-Symbol Architecture of the Binary Arithmetic Coder for UHDTV Video Encoders. **IEEE Transactions on Circuits and Systems I: Regular Papers**, IEEE, v. 67, n. 3, p. 891–902, 3 2020. ISSN 1549-8328. Disponível em: <https://ieeexplore.ieee.org/document/8897118/>.

PASTUSZAK, G. Generative Multi-Symbol Architecture of the Binary Arithmetic Coder for UHDTV Video Encoders. **IEEE Transactions on Circuits and Systems I: Regular Papers**, IEEE, v. 67, n. 3, p. 891–902, 3 2020. ISSN 1549-8328. Disponível em: <https://ieeexplore.ieee.org/document/8897118/>.

Qing Wu; PEDRAM, M.; Xunwei Wu. Clock-gating and its application to low power design of sequential circuits. **IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications**, v. 47, n. 3, p. 415–420, 3 2000. ISSN 10577122. Disponível em: <http://ieeexplore.ieee.org/document/841927/>.

RAMANI, A. S.; AHMED, A. H. **Cadence Encounter RTL Compiler**. 2016. Disponível em: <https://sudip.ece.ubc.ca/rtl-compiler/>.

RAMOS, F. L. L. et al. High-Throughput Binary Arithmetic Encoder using Multiple-Bypass Bins Processing for HEVC CABAC. In: **2018 IEEE International Symposium on Circuits and Systems (ISCAS)**. IEEE, 2018. p. 1–5. ISBN 978-1-5386-4881-0. Disponível em: <https://ieeexplore.ieee.org/document/8350885/>.

RAMOS, F. L. L. et al. Energy-Throughput Configurable Design for Video Processing Binary Arithmetic Encoder. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 31, n. 3, p. 1163–1177, 3 2021. ISSN 1051-8215. Disponível em: <https://ieeexplore.ieee.org/document/9082025/>.

REMLEY, F. M. Introduction to the Papers on the Type D-1 Digital Video Recorder. In: **Digital Television Tape Recording and Other New Developments: 20th Annual SMPTE Television Conference**. IEEE, 1986. p. 11–12. ISBN 978-1-61482-915-7. Disponível em: <http://ieeexplore.ieee.org/document/7268266/>.

RICHARDSON, I. E. **The H. 264 advanced video compression standard**. [S.l.]: John Wiley & Sons, 2011.

RIVAZ, P. de; HAUGHTON, J. Av1 bitstream & decoding process specification. **The Alliance for Open Media**, p. 182, 2019. Disponível em: <https://aomediacodec.github.io/av1-spec/av1-spec.pdf>.

ROSENBAUM, S. **Why Television Is Dead**. 2014. Disponível em: <https://www.forbes.com/sites/stevenrosenbaum/2014/01/28/why-television-is-dead/?sh=368e3ec54262>.

ROSENBERG, J. **World, Meet Thor – a Project to Hammer Out a Royalty Free Video Codec**. 2015. Disponível em: <https://blogs.cisco.com/collaboration/world-meet-thor-a-project-to-hammer-out-a-royalty-free-video-codec>.

ROY, A.; MAITI, A. K.; GHOSH, K. A perception based color image adaptive watermarking scheme in YCbCr space. In: **2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)**. [S.l.]: IEEE, 2015. p. 537–543. ISBN 978-1-4799-5991-4.

SAID, A. **Introduction to Arithmetic Coding - Theory and Practice**. [S.l.], 2004.

SHANKLAND, S. **Tech giants join forces to hasten high-quality online video**. CNET, 2015. Disponível em: <https://www.cnet.com/news/tech-giants-join-forces-to-hasten-high-quality-online-video/>.

SkyWater; Google. **Welcome to SkyWater SKY130 PDK's documentation! — SkyWater SKY130 PDK 0.0.0-331-gf70d8ca documentation**. 2021. Disponível em: <https://skywater-pdk.readthedocs.io/en/main/>.

ST. **Rad hard 65nm CMOS technology platform for space applications**. [S.l.], 2015. Disponível em: <https://www.st.com/en/space-products/c65space.html>.

STAMENKOVIC, Z. et al. Rear view camera system for car driving assistance. In: **2012 28th International Conference on Microelectronics Proceedings**. [S.l.]: IEEE, 2012. p. 383–386. ISBN 978-1-4673-0238-8.

TAGLIASACCHI, M. et al. Exploiting Spatial Redundancy in Pixel Domain Wyner-Ziv Video Coding. In: **2006 International Conference on Image Processing**. [S.l.]: IEEE, 2006. p. 253–256. ISBN 1-4244-0480-0.

TEODOR-DUMITRU, E. **Design and Characterization of a Standard Cell Library for the SkyWater 130nm Process**. Tese (Doutorado), 2020. Disponível em: <http://www.akrabjuara.com/index.php/akrabjuara/article/view/919>.

TESSEROLI, C. R.; SEIKE, J. **Extração de frames em canais ao vivo**. 2021. Disponível em: <https://aws.amazon.com/pt/blogs/aws-brasil/extracao-de-frames-em-canais-ao-vivo/>.

TORNG, C. **mflowgen/freepdk-45nm: ASIC Design Kit for FreePDK45 + Nangate for use with mflowgen**. 2020. Disponível em: <https://github.com/mflowgen/freepdk-45nm>.

TROW, I. AV1: Implementation, Performance, and Application. **SMPTE Motion Imaging Journal**, Society of Motion Picture and Television Engineers, v. 129, n. 1, p. 51–56, 1 2020. ISSN 21602492.

VALIN, J.-M. et al. Daala: Building a next-generation video codec from unconventional technology. In: **2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP)**. IEEE, 2016. p. 1–6. ISBN 978-1-5090-3724-7. Disponível em: <http://ieeexplore.ieee.org/document/7813362/>.

Xilinx Inc. **Video Beginner Series 10: YCbCr Chroma subsampling/resampling**. 2021. Disponível em: <https://support.xilinx.com/s/article/887486?language=en_US>.

YANG, Y.; YUHUA, P.; ZHAOGUANG, L. A Fast Algorithm for YCbCr to RGB Conversion. **IEEE Transactions on Consumer Electronics**, v. 53, n. 4, p. 1490–1493, 11 2007. ISSN 0098-3063.

ZHANG, J. et al. Recent Development of AVS Video Coding Standard: AVS3. In: **2019 Picture Coding Symposium (PCS)**. IEEE, 2019. p. 1–5. ISBN 978-1-7281-4704-8. Disponível em: <https://ieeexplore.ieee.org/document/8954503/>.

ZHOU, D. et al. Ultra-High-Throughput VLSI Architecture of H.265/HEVC CABAC Encoder for UHDTV Applications. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 25, n. 3, p. 497–507, 2015. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6851145>.

ZUMMACH, E. et al. High-Throughput CDEF Architecture for the AV1 Decoder Targeting 4K@60fps Videos. In: **2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)**. IEEE, 2020. p. 1–4. ISBN 978-1-7281-3427-7. Disponível em: <https://ieeexplore.ieee.org/document/9068979/>.

ANNEX A — LIST OF PUBLICATIONS BY THE AUTHOR

A.1 Journal Publications

1. **Power-Saving 8K Real-Time AV1 Arithmetic Encoder Architecture**, *IEEE Design & Test*, (BITENCOURT; RAMOS; BAMPI, 2022a).

2. **AV1 Arithmetic Encoder Design on Open-Source EDA**, *Journal of Integrated Circuits and Systems (JICS)*, (BITENCOURT; RAMOS; BAMPI, 2022c).

3. **Low-Power High-Throughput Architecture for AV1 Arithmetic Decoder**, *IEEE Design & Test*, (GOMES et al., 2022).

A.2 Conference Publications

1. **High-Throughput and Low-Power Architectures for the AV1 Arithmetic Encoder**, *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, (BITENCOURT; RAMOS; BAMPI, 2021).

2. **Power-Throughput Trade-off Analysis for a Novel Multi-Boolean AV1 Arithmetic Encoder Design**, *2022 Picture Coding Symposium (PCS)*, (BITENCOURT; RAMOS; BAMPI, 2022b)

3. **Area and Power Efficient High-Throughput Design for AV1 Arithmetic Decoding**, *2022 Picture Coding Symposium (PCS)*, approved for publication.

APPENDIX A — METHODOLOGY FOR THE ARITHMETIC ENCODING DESIGN

The methodology used to design a new process, both software or hardware, has the primary goal of ensuring that the process is done only once. In order to create an architecture capable of tackling the problem at the first trial, it is vital to understand precisely how the process works, what its flaws are, and how to make the process as efficient as possible.

Accomplishing these goals takes a considerable amount of analysis (especially data analysis) and a well-defined project. These topics are presented in subsections organized as follows: data extraction (Section A.1), data analysis and understanding of the algorithm (Section A.2), and the catalog of essential information taken from the analysis made (Section A.3).

A.1 Data Extraction

The first task to be done, which is prior to the designing phase itself, was the data extraction from the AV1 reference software (Alliance for Open Media, 2020). As explained in previous chapters, the AV1 itself is a definition of methods, algorithms, and procedures that, when combined, are capable of effectively compressing videos. To demonstrate the effectiveness of the process, the developers responsible for the coding format definition created a software solution capable of encoding and decoding videos. This software is herein referred to as the AV1 reference software.

As the AV1 reference software is presumed to represent the exact behavior expected from the codecs, analyzing its implementation brings a certain level of confidence of understanding the video coding format itself. Moreover, as the AV1 is an open-source solution, its software source code is available on the Internet and can be modified to extract data from intermediate execution stages.

In order to accomplish that, it was taken a dataset of six video sequences with resolutions ranging from 720p up to 4K (MERCAT; VIITANEN; VANNE, 2020). These videos were then encoded using a modified entropy encoder process that allows the creation of CSV (Comma-Separated Values) files comprising the intermediate values of internal functions.

Each row of the the so-called “main_data" CSV file generated indicates one round of execution (i.e., the encoding of one symbol). As one may notice, the file comprises the following columns that represent internal variables within the AV1 reference software:

$bool\_flag$**:** represents the flag indication if the current round is (1) *Boolean Operation* or (0) *CDF Operation*;

$Range_{i-1}$**:** indicates the $Range$ value at the beginning of the round. When the row does not represent the first round of a frame, this value must match the $Range_{final}$ from the previous round;

Low_{i-1}**:** represents the Low at the beginning of the round and also should match the previous Low_{final};

FL**:** represents the lower probability of the current symbol;

FH**:** represents the higher probability of the current symbol;

$nsyms$**:** represents the number of symbols in the alphabet currently being encoded;

$symbol$**:** represents the symbol being encoded;

$Range_{raw}$**:** represents the $Range$ value sent as input for the renormalization function;

Low_{raw}**:** represents the Low value sent as input for the renormalization function;

$Range_i$**:** represents the final $Range$ value obtained after the renormalization;

Low_i**:** represents the final Low value obtained after the renormalization.

The analysis made upon the generated files allows the understanding of a few important information regarding the AV1 arithmetic encoding process:

1. $Range_{initial}$ and $Low_{initial}$ should always match their counterparts $Range_{final}$ and Low_{final} when there is not a frame changing or, in other words, a reset in the architecture;

2. FL and FH individually do not give anything information. However, when combined, they represent the red bars observed in Figure 3.1 and, hence, the probability of the current symbol of a *CDF Operation* round;

3. $nsyms$ is always greater than $symbol$. The former represents the number of symbols in the given alphabet, whereas the latter represents the symbol being encoded, which is comprised within the range $0 \leq symbol < nsyms$.

To understand the bitstream generation, which represents a crucial part of the entropy encoding process, the so-called “bitstream" CSV file was created. This file is written by the function responsible for the carry propagation process, and its structure is as simple as possible: one decimal number representing an 8-bit bitstream per row. The analysis of this file made it possible to understand how the bitstream generation works, which is analyzed in Section A.2.

Therefore, the extraction of data from the AV1 reference software allowed an in-depth comprehension of how the AV1 codec works and how it generates the output bitstreams. Moreover, this data extraction allowed a more precise verification process, which is explained in Section B.

A.2 Analysis and Understanding of the Algorithm

The analysis and comprehension of the AV1 reference software allow better hardware designing, which is the primary goal of this research. This subsection aims to explain the conclusions regarding the algorithm’s behavior and how they affected the architecture designing process.

In order to accomplish that, the following topics are presented: the $Range$ and Low updating process in A.2.0.1, and the bitstream generation including the carry propagation in A.2.0.2.

A.2.0.1 $Range$ and Low Updating Processes

The AV1 reference software comprises three main functions for the $Range$ and Low updating processes: *CDF Operation*, *Boolean Operation* and renormalization. The first two functions, *CDF Operation* and *Boolean Operation*, are the ones receiving the inputs (mainly FL, FH, $nsyms$ and $symbol$) for the arithmetic encoding process. These variables are then manipulated inside these functions by (A.1) and (A.2) in the case of

CDF Operation, and only (A.2) in the case of *Boolean Operation*.

$$u = [(Range_{i-1} \gg 8) \times (fl \gg 6) \gg 7] + 4 \times [(nsyms - 1) - (symbol - 1)] \quad \text{(A.1)}$$

$$v = [(Range_{i-1} \gg 8) \times (fh \gg 6) \gg 1] + 4 \times [(nsyms - 1) - (symbol + 0)] \quad \text{(A.2)}$$

As one may notice after analyzing equations (A.1) and (A.2), they are comprised of two multiplications each. The first multiplication in each one involves the $Range$ resulted from previous rounds, characterized as $Range_{i-1}$ in the "main_data" CSV file presented in Section A.1. This multiplication, unfortunately, has its operands ranging within $0 < Range \leq 65535$ and $0 < [(\text{FL or FH}) \gg 6] \leq 1023$, which makes it unfeasible to create a multiplication-free architecture by using a look-up table (LUT) (i.e., the presented multiplication is needed because there are too many possible results for a LUT).

On the other hand, the second multiplication is related to the values $nsyms$ and $symbol$, which rang within the range $2 \leq nsyms \leq 16$ and $0 \leq symbol \leq 15$, respectively. The combination of all possible values for these variables can be feasibly allocated into a fairly small LUT (i.e., 256 16-bit positions). Therefore, this multiplication can be avoided in hardware, thus reducing delays, area, and power consumption.

With regards to the renormalization process, this function, implemented in the C programming language, uses a built-in leading zero counter (LZC), followed by the pre-bitstreams (PB_1 and PB_2) generation. As one may remember from Chapter 3, the renormalization process in a general-purpose arithmetic encoder is done by multiplying by two both Low and $Range$. In binary, however, a multiplication by two can be effectively swapped by a left-shift and, as the renormalization aims to make $Range$ fit within the interval $0.5 \leq range \leq 1.0$, and assuming $1.0 = 65535$ due to the $Range$'s 16-bit array, $0.5 = 32768$. As one may conclude, 32768 in binary is represented by a 16-bit array in which only the most significant bit (MSB) is set to one. Therefore, the use of an LZC reduces the complexity of the renormalization process considerably, as only one left-shift is needed to ensure that $Range$ fits within the required final interval.

Furthermore, Algorithm 2 displays the so-called pre-bitstream generation method. This process generates a 9-bit array based upon the Low and LZC values, where the latter

is expressed by the letter d.

Algorithm 2: Low renormalization process and pre-bitstreams generation.

Data: $Low_{raw,0}$, cnt_{i-1}, d
Result: PB_{flag}, PB_1, PB_2, Low_i, cnt_i
$m_1 \leftarrow [1 \ll (cnt_{i-1} + 7)] - 1$; {Mask 1 for renorm.}
$m_2 \leftarrow [1 \ll (cnt_{i-1} - 1)] - 1$; {Mask 2 for renorm.}
$Low_{raw,1} \leftarrow Low_{raw,0} \wedge m_1$; {Reduced Low_1.}
$Low_{raw,2} \leftarrow Low_{raw,0} \wedge m_2$; {Reduced Low_2.}
$S_{comp} \leftarrow cnt_{i-1} + d$; {Cumulative renorm. checker.}
$PB_1 \leftarrow Low_{raw,0} \gg (cnt_{i-1} + 7)$; {Possible PB_1.}
$PB_2 \leftarrow Low_{raw,1} \gg (cnt_{i-1} - 1)$; {Possible PB_2.}
if $(S_{comp} \geq 9) \wedge (S_{comp} < 17)$ **then**
    $Low_i \leftarrow Low_{raw,1} \ll d$; {Use Low_1 and renorm.}
    $cnt_i \leftarrow cnt_{i-1} + d - 8$; {Cumulative renorm. 1.}
    $PB_{flag} \leftarrow 01_2$; {Generate PB_1 only.}
else if $S_{comp} \geq 17$ **then**
    $Low_i \leftarrow Low_{raw,2} \ll d$; {Use Low_2 and renorm..}
    $cnt_i \leftarrow cnt_{i-1} + d - 16$; {Cumulative renorm. 2.}
    $PB_{flag} \leftarrow 10_2$; {Generate PB_1 and PB_2.}
else
    $Low_i \leftarrow Low_{raw,0} \ll d$; {Just renorm. Low.}
    $cnt_i \leftarrow S_{comp}$; {Just update cnt}
    $PB_{flag} \leftarrow 00_2$; {Do not generate PB.}
end

A.2.0.2 Bitstream Generation and Carry Propagation

After the 9-bit pre-bitstream generation presented in A.2.0.1, the AV1 reference software stores the generated values into a list to be later processed by the carry propagation function upon the arrival of the last symbol in the encoding frame. This list, which might comprise a reasonably significant number of pre-bitstreams, is analyzed on its entirety using a loop and the analysis process is accomplished by the Algorithm 3.

Algorithm 3: Carry propagation behavior when $flag \leftarrow 2$.

Data: PB_{flag}, $PB_{previous}$, PB_1, PB_2, $flag\_last$
Result: FB_{flag}, FB_1, FB_2, FB_3, FB_4, FB_5
$FB_{flag} \leftarrow 2$;{Set output flag for the design.};
$FB_1 \leftarrow B_{previous}[7:0] + PB_1[8]$;{Set the FB_1 based on $B_{previous}$ and PB_1};
$FB_2 \leftarrow PB_1[7:0] + PB_2[8]$;{Set FB_2 based on PB_1 and PB_2};
$B_{previous} \leftarrow PB_2[7:0]$;{Set the next $B_{previous}$ based on the 8 LSB for PB_2};
if $flag\_last == 1$ **then**
    $FB_3 \leftarrow PB_2[7:0]$;{Set FB_3 when the $flag\_last$ is 1.};

A functionality designed in hardware to behave exactly like the AV1 reference software carry propagation process presented in the Algorithm 3 would imply the use of a buffer or a FIFO (First In, First Out) circuitry. Moreover, this circuitry would spend a considerable amount clock cycles digging through all pre-bitstreams generated for the just-encoded frame, which might be in the order of millions, while the rest of the architecture would be waiting without encoding new symbols. As one may conclude, this would create a critical bottleneck that could easily suffer from overflow when encoding ultra-high definition (UHD) videos, for example.

As it is presented in Subsection 4.2.2.4, the carry propagation process had to be changed and executed right after each pre-bitstream generation. For that, and after carefully analyzing pre-bitstreams and final bitstreams extracted by the AV1 reference software, it was possible to conclude that the carry propagation is defined by the addition of the MSB (the ninth bit in a 9-bit array) of the just generated pre-bitstream into the previously generated one.

Equation (A.3) defines the transcribed process. PB_{i-1} and PB_i represent the previously and currently generated bitstreams, respectively, whereas FB_{i-1} identifies the output for the final bitstream number $i-1$.

$$FB_{i-1} = PB_{i-1} + PB_i[8] \tag{A.3}$$

Another critical behavior of the carry propagation function is to generate the last bitstreams of a frame, which occurs upon the arrival of the current frame's last $symbol$ during a video encoding, and when the Low variable is not in its resetting value (i.e., 0). The main idea of this sub-function is to use the remaining Low to generate the frame's last bitstreams and ensure that all symbols of the frame are indeed being covered by the bitstreams generated throughout the codification. Algorithm 4 depicts the process of generating the last pre-bitstreams of a frame. These last pre-bitstreams are then passed through the carry propagation process to generate the last final bitstreams.

A.3 Key Informations About the Algorithm

The herein presented analysis upon the AV1 reference software allows some conclusions to be made. The first one, and perhaps the most important, is the fact that, although the entire structure of the AV1 codec is considered to be hardware-friendly

Algorithm 4: *Last Bits* sub-block of Stage 4.

Data: Low_{i-1}, cnt_{i-1}
Result: PB_{flag}, PB_1, PB_2
$m \leftarrow 3FFF_{16}$; {Set default mask.};
$n \leftarrow [1 \ll (cnt_{i-1} + 7)] - 1$; {Set cumulative mask.};
$e_1 \leftarrow [(Low_{i-1} + m) \wedge \overline{m}] \vee (m + 1)$;
$e_2 \leftarrow e_1 \wedge n$; {Apply cumulative mask.};
$S_{comp} \leftarrow cnt_{i-1} + 10$; {Renormalization checker.};
if $(S_{comp} > 9) \wedge (S_{comp} \leq 17)$ **then**
    $PB_{flag} \leftarrow 01_2$; {Generate PB_1 only.};
else if $S_{comp} > 17$ **then**
    $PB_{flag} \leftarrow 10_2$; {Generate PB_1 and PB_2.};
else
    $PB_{flag} \leftarrow 00_2$; {Do not generate PB.};
end
$PB_1 \leftarrow e_1 \gg (cnt_{i-1} + 7)$; {Set PB_1.};
$PB_2 \leftarrow e_2 \gg (cnt_{i-1} - 1)$; {Set PB_2.};

(DENG; MOCCAGATTA, 2017), the arithmetic encoder step is implemented with a lower rate of care for hardware adaptations. This hardware-unfriendly arithmetic encoder is confirmed due the carry propagation step reliance on storing all pre-bitstreams generated for a frame by the renormalization process before, finally, propagating the carry and releasing the final bitstreams.

Another important conclusion taken after analyzing the AV1 reference software is the necessity for the $Range$ and Low updating processes to rely upon previously generated values for these variables. Moreover, the Low step is also a continuation of the $Range$ process and can, therefore, be combined into the same pipeline stage, or divided into two stages, as presented in Subsections 4.2.2.2 and 4.2.2.3.

With regards to the $nsyms$ and the $symbol$ variables, it is accurate to assume that one combination of $symbol$, $nsyms$, FH and FL is received per round of execution. The AV1 reference software does not create any kind of parallelization upon the arithmetic encoder step, as the current set of inputs are directly dependent on results from the previous round of execution.

Taking into consideration the herein presented conclusions from the AV1 reference software and combining the knowledge acquired from previously designed architectures and already published works (RAMOS et al., 2018), it was feasible to predict the necessity for a 4-stage pipeline in the AE-AV1 baseline design. This number of stages is defined based on the $Range$ multiplication, which creates the architecture's critical path and is explained in Subsection 4.2.2.2. As explained in Subsection 4.2.1, the division into more

than four stages of pipeline is unfeasible due to the $Range$ and Low processes relying on their previous results, whereas a lower number of stages would result in a greater critical path, which would, therefore, increase the architecture delay and reduce the throughput rate per second.

APPENDIX B — VERIFICATION METHODOLOGY

Section 4.2 introduced the AE-AV1 architecture and all important decisions taken to accomplish the best possible architecture with regards to frequency, area and throughput rate. However, a crucial part of the designing process is the verification phase, where the architecture receives random and/or real-world data be ensure that all its functionalities are working properly.

Usually, when working for companies as part of teams, there are mainly two different teams: design and verification. The former has the goal of designing an architecture capable of executing a given procedure or functionally, whilst the latter has to, independently, prepare a verification environment and test-cases for the given problem. As each team works in its solution separately and based on the same parameters and definitions (i.e., specification), there is a greater chance that both teams, when combined, will reach a solution that is exactly what was proposed initially.

The verification phase is where both of the presented teams get closer and connect their solutions (architecture and testbenches). The testbench stimulates the architecture and analyzes its outputs to ensure that the architecture works as planned for all possible scenarios (defined by the testbench functional and code coverages). Once a mismatch occurs, the designers and verification engineers need to establish which of the teams implemented the wrong solution according to the testbench analysis.

In an academic research, as normally the human resources are limited, the same team or person will be in charge of designing the architecture and creating the testbenches. Hence, the use of real-would data emerges as a solution to break that person's biases to commit the same error on both sides, which will create a unexpected match for a wrong result.

For the AE-AV1 verification process, it were extracted, from the AV1 reference software, several input and output CSV files containing all stimuli for the architecture and all final bitstreams generated by the software. Each combination of input and output files represented a single video sequence from selected datasets (MERCAT; VIITANEN; VANNE, 2020; MONTGOMERY, 2005).

Therefore, in order to created a valid testbench that allowed the verification of the architecture without any bias to commit an error on both sides, it was defined that real-world data would be used. In total, round 500 million symbols were encoded using the architecture and each output bitstream was analyzed to ensure total compatibility between

expected and generated. Moreover, it is interesting to note that some problems occurred in an extremely small rate, which makes then fairly hard to find using not-big-enough datasets.