Parallel Algorithms and Architectures for Low Power Video Decoding

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
OCT 26 2010
LIBRARIES

by

Vivienne Sze

B.A.Sc. in Electrical Engineering, University of Toronto, 2004
S.M. in Electrical Engineering, Massachusetts Institute of Technology, 2006

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

ARCHIVES

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

© Massachusetts Institute of Technology 2010. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by .....
Anantha P. Chandrakasan
Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Parallel Algorithms and Architectures for Low Power Video Decoding

by

Vivienne Sze

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Parallelism coupled with voltage scaling is an effective approach to achieve high processing performance with low power consumption. This thesis presents parallel architectures and algorithms designed to deliver the power and performance required for current and next generation video coding. Coding efficiency, area cost and scalability are also addressed.

First, a low power video decoder is presented for the current state-of-the-art video coding standard H.264/AVC. Parallel architectures are used along with voltage scaling to deliver high definition (HD) decoding at low power levels. Additional architectural optimizations such as reducing memory accesses and multiple frequency/voltage domains are also described. An H.264/AVC Baseline decoder test chip was fabricated in 65-nm CMOS. It can operate at 0.7 V for HD (720p, 30 fps) video decoding and with a measured power of 1.8 mW. The highly scalable decoder can tradeoff power and performance across >100x range.

Second, this thesis demonstrates how serial algorithms, such as Context-based Adaptive Binary Arithmetic Coding (CABAC), can be redesigned for parallel architectures to enable high throughput with low coding efficiency cost. A parallel algorithm called the Massively Parallel CABAC (MP-CABAC) is presented that uses syntax element partitions and interleaved entropy slices to achieve better throughput-coding efficiency and throughput-area tradeoffs than H.264/AVC. The parallel algorithm also improves scalability by providing a third dimension to tradeoff coding efficiency for power and performance.

Finally, joint algorithm-architecture optimizations are used to increase performance and reduce area with almost no coding penalty. The MP-CABAC is mapped to a highly parallel architecture with 80 parallel engines, which together delivers >10x higher throughput than existing H.264/AVC CABAC implementations. A MP-CABAC test chip was fabricated in 65-nm CMOS to demonstrate the power-performance-coding efficiency tradeoff.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Joseph F. and Nancy P. Keithley Professor of Electrical Engineering

Acknowledgments

First, I would like to thank Professor Anantha Chandrakasan for his guidance and mentorship throughout the past six years. I greatly appreciate the opportunities that you have created for me and the exposure you have given me through industry interactions. Thank you for being a strong advocate of my work and always keeping my best interest in mind. I am grateful that you always made yourself available for discussion and advice despite your busy schedule - I've very much enjoyed our talks in the lab and at the Z. It has truly been a rewarding experience to work in the collaborative environment that you have fostered in your lab. Finally, thank you for pioneering the use of voltage scaling with parallelism, which set the foundation for the work presented in this thesis (as well as my masters thesis).

I would like to acknowledge Professor Vivek Goyal and Dr. Madhukar Budagavi for serving on my thesis committee and providing invaluable research advice. I'd also like to extend my thanks to Vivek for serving on my RQE committee. Much of what I learned about video coding came from the many engaging discussions I had with Madhukar during my summers at Texas Instruments (TI). I look forward to many more in the future.

I'd like to thank Dennis Buss and Alice Wang for their help in making the test chips featured in this thesis possible - especially with surprises we faced with the last one. I'd also like to thank Alice for the many highlights (IV REAL!) during my time in Dallas. Thanks to Minhua Zhou, Bruce Flinchbaugh and Martin Izzard for providing me the opportunity to do several internships at TI, where I ramped up on video and got to experience R & D in an industry setting. I'd also like to thank Minhua and Umut Demircin for technical discussions.

I'd like to thank Professor Peter Ottensmeyer for giving me the opportunity to work in his lab over a decade ago, where I got my first taste of research. He taught me how to take a complex and seemingly insurmountable problem, and break it into smaller more manageable parts.

My time at MIT was truly enriched by the people I interacted with on a daily basis. Thanks to the past and present members of *ananthagroup* for making the lab an incredibly enjoyable and interesting place to work, and for the many friendships and laughs. For all the

hours we spend in the lab, I'm glad it's with you folks. I had great fun collaborating with Daniel Finchelstein and Mahmut Ersin Sinangil on the video decoder chip. Thanks to Yogesh Ramadass for helping us fix those 1000+ antenna violations two days before tapeout. I'd like to thank Raul Blazquez for his continued mentorship post-graduation; Manish Bhardwaj for his insight on all things technical and non-technical; Joyce Kwong for discussions on all things digital; and Nathan Ickes for coming to the rescue during soldering mishaps and answering all my PCB and FPGA related questions. To Fred (Chen and Lee), Brian Ginsburg, Raul, Alice, Manish, and Daniel thanks for all your career advice and for answering my many questions during the whole job hunting hoopla. I'd like to thank Alex, Ben, Brian, Daniel, Dave, Denis, Fred, Joyce, Mahmut, Manish, Marcus, Masood, Nathan, Naveen, Pat, Payam, Phil, Raul, and Yogesh, among others for the great lunchtime conversation that often provided a nice break from research. I'd like to thank Joyce, Denis, Raul and Bonnie for taking the time to proof-read various portions of this thesis and providing valuable feedback.

Thanks to Margaret Flaherty for her help with logistics ranging from rushed purchase orders, to scheduling appointments, to finding a room for the defense. Thanks to Danielle Guichard-Ashbrook for always being incredibly patient and helpful during my *many* visits to the ISO. I'd also like to thank the MTL Compute Team for their quick response whenever the computers or tools when down or a file needed to be recovered.

Finally, I'd like to thank those who provided invaluable support outside the lab. To my SATC gals, thanks for the brunches, trips, and memories. Joyce, thanks for being great buddy both in and out of the lab. Rumi Chunara, thanks for keeping me in shape on our morning runs and swims. Maryam Modir Shanechi, I couldn't have asked for a better friend and roommate. Thanks for being my family in Boston throughout the past six years. And to my family in Toronto, Mom, Jen and TeeTee, thank you for your support and encouragement throughout this journey. Your unconditional love and support mean the world to me.

Done and done!

This work was funded by Texas Instruments, who also provided fabrication services, and in part by an Natural Sciences and Engineering Research Council of Canada (NSERC) Fellowship.

Contents

List of Figures

List of Tables

Acronyms

AD	arithmetic decoders
ASIC	application specific integrated circuits
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context-based Adaptive Variable Length Coding
DB	deblocking filter
DVFS	dynamic voltage and frequency scaling
ED	entropy decoding
FB	frame buffer
FIFO	first-in-first-out queue
fps	frames per second
FSM	finite state machine
GOP	group of pictures
GPU	Graphics Processing Unit
HD	high definition
HDTV	high definition television
HRD	hypothetical reference decoder
IES	Interleaved Entropy Slices
INTRA	intra-prediction
IT	inverse transform
JCT-VC	Joint Collaborative Team for Video Coding

LCD	liquid crystal display
LPS	least probable symbol
LSB	least significant bit
LUT	look up table
LZ	leading zeros
MC	motion compensation
MEM	memory controller
MP-CABAC	Massively Parallel CABAC
MPEG	Moving Picture Experts Group
MPS	most probable symbol
MSB	most significant bit
MVC	Multiview Video Coding
mvd	motion vector difference
OLED	organic light-emitting device
PGA	pin grid array
PPS	picture parameter set
PSNR	peak signal-to-noise ratio
QD-OLED	quantum dot-organic light-emitting device
QP	quantization parameter
RGB	Red Green Blue
SD	standard definition
SEP	Syntax Element Partitions
SPS	sequence parameter set
SVC	Scalable Video Coding
TSV	through-silicon via
UDVS	Ultra-dynamic voltage scaling
VCEG	Video Coding Experts Group

Chapter 1

Introduction

Four decades of semiconductor process scaling driven by Moore's law [1] has enabled the integration of more than a billion transistors on a single chip [2]. A key challenge is how to link this technology to the market. In other words, how can we best use these billions of transistors to meet the needs of tomorrow? Parallelism is an effective method of leveraging these transistors to deliver the power and performance required for current and future applications. Given the growing pervasiveness of multimedia in recent years, one important application that merits study is next generation video coding (i.e. video compression).

Video codecs (i.e. *co*der and *dec*oder), used to compress and decompress video data, can be loosely classified into two categories, low power and high performance, both of which can benefit from parallelism. In this thesis, we will show that by exposing parallelism in *both* architecture and algorithm design we can meet these power and performance requirements and establish efficient tradeoffs across these metrics. Furthermore, we will demonstrate how accounting for architecture implementation during algorithm design can also result in low area and complexity cost. This chapter begins by establishing the requirements for next generation video coding in Section 1.1. Section 1.2 describes how parallel architectures and algorithms can be used to increase throughput to meet performance requirements.[1] Section 1.3 discusses how this increased throughput can be translated to power savings

[1]In this thesis, *throughput* will refer to operations per cycle, while *performance* will refer to operations per second.

using voltage scaling. Section 1.4 motivates the importance of scalability in video decoding.

1.1 Next Generation Video Coding

Video anytime, anywhere. In recent years, video has become widely available through many different sources, from broadcast to the web, across wired and wireless channels. Accordingly, devices that can receive and playback video have expanded beyond television sets to include personal computers and cellphones. In fact, the use of video is becoming ever more pervasive on battery-operated handheld devices such as camera phones, digital still cameras, personal media players, etc. Annual shipment of such devices already exceeds a hundred million units and continues to grow [3].

The amount of video content available on-line has grown substantially thanks to websites like YouTube [4] and Hulu [5]. As WiFi becomes increasingly ubiquitous along with 3G cellular networks and upcoming 4G LTE, portable devices will be increasingly used to access video content. Furthermore, there is a strong demand for these devices to support higher resolutions up to high definition (HD) [6]; HD implies 720p resolution (1280x720 pixels per frame) and beyond, which is well over twice as many pixels per frame than standard definition (SD). The battery operated iPad, released by Apple in 2010, can already support up to 720p video decoding, and YouTube and Hulu have recently begun offering video content in HD 720p. However, the iPad display resolution is only XGA (1024x768) [7]. New display technologies such as the pico projector [8], which enable display sizes to extend beyond the limited form factor of handheld devices, will further drive the demand for portable HD video in the future.

Video coding is required to overcome the limitations and costs of transmission bandwidth and data storage. The advancement of video coding technology has played a key role in enabling *video anytime, anywhere.* Traditionally, the focus of video codec design has been to improve coding efficiency (i.e. higher video fidelity with fewer bits). However, for mobile devices, power consumption is a key consideration and must be minimized to reduce the device's size, weight and cost. Therefore, the need to support portable HD video makes

power efficiency a key requirement in next generation video codec design.

From a power perspective, HD video on battery-operated devices has several implications for the video codec. First, the algorithms that are used in the codec need to provide high coding efficiency to substantially reduce the size of the video streams. This implies the use of state-of-the-art video coding standards such as H.264/AVC. The high coding efficiency usually comes at a cost of increased complexity in the algorithm. The complexity is typically evaluated based on the number of operations and memory accesses. For instance, while H.264/AVC has a 50% improvement in coding efficiency as compared to MPEG-2 (currently used for high definition television (HDTV)), it increases complexity by 4x at the decoder [9]. This increase in complexity translates into increase in power consumption which is problematic for battery-operated devices.

Second, HD videos have high resolution, which implies that many pixels need to be processed per second (e.g. 27.6 Mpixels/s for 720p at 30 frames per second (fps)). For real-time compression and playback, the codec must operate at a speed that can meet the performance requirement. Consequently, not only does HD video coding require more operations per frame, but each operation must be done at a faster rate, both of which pose a significant challenge when there is a limitation on power as in the case of battery-operated devices.

Video codec performance challenges also exist for non-battery powered devices. Displays reaching cinema quality resolutions of 4kx2k (4096x2160 pixels per frame - 25.6x more pixels than SD) have been showcased at the Consumer Electronics Show in recent years and will soon be coming to the market [10–12]. Furthermore, frame rates up to 120 fps are targeted for high action sports. Thus, even in devices without stringent power constraints, such as set-top boxes, next generation video codecs will face challenges in achieving the performance required to deliver 4kx2k resolution at frame rates up to 120 fps. Researchers have already begun looking into developing 8kx4k (7680x4320 pixels per frame) video cameras [13]. In the future, portables devices with pico projectors may also be required to support these high resolutions and frame rates.

In summary, next generation video codecs will support higher resolution and frame rates

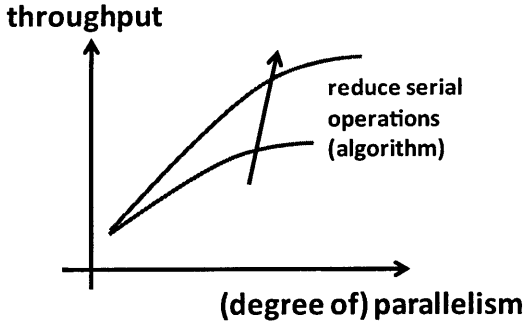
which require high processing performance. At the same time, many of these codecs will be located on battery-operated devices which require low power consumption. High coding efficiency will also be required to reduce storage and transmission costs. Finally, scalability is also desired given the many different use cases that exist for video coding. This thesis will show that these requirements can be addressed through the effective use of parallelism.

1.2 Enabling Parallelism

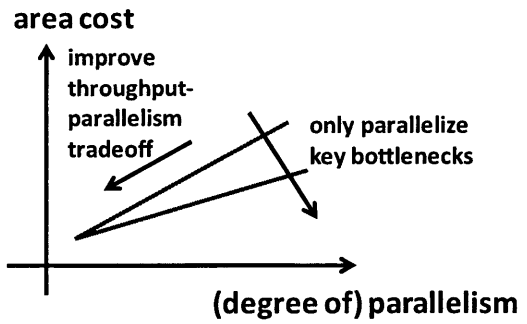
Increasing performance can be achieved by increasing throughput (i.e. number of operations per cycle) and/or operating frequency (i.e. number of cycles per second). However, high operating frequency comes at the cost of power consumption and heat dissipation. Furthermore, the maximum frequency is limited by the critical path of the hardware, and therefore cannot always be increased to meet a desired performance. Parallelism provides a good alternative by enabling more concurrent operations to increase processing throughput.

Parallelism can be enabled at various stages of the design. During architecture definition, hardware modules are replicated so that they can operate at the same time. The granularity of the parallelism can vary. For instance, the entire processor core is replicated in multi- and many-core solutions. Today's processors can contain two to four cores, while road maps point to hundreds and thousands of cores per chip in the future [14]. Graphics Processing Units (GPUs) are another variant of this form of parallelism. For hardwired solutions (i.e. application specific integrated circuits (ASIC)), parallelism is often exposed at the micro-architecture level; this involves replicating logic (e.g. adders, multipliers) in datapaths. To reduce area cost, parallel architectures should be mostly applied to the bottlenecks of the design (Fig. 1-1b). Furthermore, when selecting datapaths to parallelize, it is important to consider the impact on the control. In this thesis, we will demonstrate how parallel architectures can be used to increase the throughput of an H.264/AVC video decoder ASIC with little cost to area and control complexity.

However, replicating hardware is not enough to guarantee improved throughput. The throughput increase (speed up) that can be achieved with parallelism is limited by the



(a) The throughput is limited by the fraction of serial operations in the algorithm. Reducing serial operations can enable more throughput for a given degree of parallelism.



(b) The area cost increases with degree of parallelism. Identifying key bottlenecks to parallelize can reduce the area cost per degree of parallelism.

Figure 1-1: Relationship between throughput, parallelism and area cost. Increasing the throughput improvement per degree of parallelism (Fig 1-1a) also reduces the area cost (Fig. 1-1b).

algorithm. Amdahl's law states that the speed up of a program using multiple processors in parallel computing is limited by the sequential portion of the program [15] (Fig. 1-1a). The speed up limit is shown in equation 1.1, where S is the fraction of serial operations, $(1 - S)$ is the fraction of parallel operations and N is the number of processors,

$$Speed\ up = \frac{1}{S + \frac{(1-S)}{N}} \tag{1.1}$$

Therefore, algorithms must also be designed with reduced serial operations (S) such that they can fully utilize the available parallel hardware (i.e. "Amdahl-friendly" algorithms). For instance, the entropy coding in video coding is inherently serial and often poses a challenge for parallel processing. Consequently, these modules typically need to operate at very high frequencies in order to achieve the performance necessary for HD video. In H.264/AVC, two forms of entropy coding are used: Context-based Adaptive Variable Length Coding (CAVLC) and Context-based Adaptive Binary Arithmetic Coding (CABAC). CABAC has better coding efficiency, but at a cost of higher complexity and increased serial dependencies. CABAC is a key bottleneck in the H.264/AVC decoder. This thesis presents an entropy coding algorithm called the Massively Parallel CABAC (MP-CABAC) that was designed

with parallelization in mind and thus can easily be mapped to parallel hardware in an effective manner. Furthermore, this parallel algorithm offers an additional dimension of scalability, and enables a tradeoff between throughput and coding efficiency which will be discussed in Section 1.4. CABAC parallelism eliminates a key bottleneck and enables a *fully parallel video decoder.*

1.3 Voltage Scaling

Parallelism can be combined with voltage scaling to achieve an efficient power-performance tradeoff [16]. In traditional digital CMOS circuits, power consumption is composed of a dynamic and a leakage component as shown in equation 1.2.

$$P_{total} = P_{dyn} + P_{leak} \tag{1.2}$$

Dynamic energy is consumed whenever a capacitor is charged and is proportional to the capacitance that is switched and the supply voltage to which the capacitor is charged.[2] The switching activity describes the frequency with which the capacitor is charged and is the product of the probability that the capacitor will be charged in a cycle (α) and the number of cycles per second (i.e. the operating frequency (f)). Thus, the dynamic power consumption (P_{dyn}) is proportional to the switching activity ($\alpha \times f$), capacitance (C_L), and supply voltage (V_{DD}):

$$P_{dyn} = \alpha \times f \times C_L \times V_{DD}^2 \tag{1.3}$$

Leakage power consumption is the power that is consumed in the absence of switching activity. The leakage power is due to leakage currents through the channel, substrate and gate [17]. These leakage currents scale with supply voltage (V_{DD}). For instance, the subthreshold current through the channel that dominates the leakage power reduces exponentially with

[2]We assume that the capacitor is charged to the supply voltage (i.e. full rail switching).

V_{DD} as shown in equation 1.4, where V_{GS} ($\leq V_{DD}$) is the gate-source voltage, V_{DS} (equal to V_{DD}) is the drain-source voltage, I_o is from the leakage current model, V_t is the threshold voltage, n is the sub-threshold slope factor,[3] η is the drain induced barrier lowering factor, and V_{th} is the thermal voltage.

$$I_{sub} = I_o e^{\frac{(V_{GS}-V_t+\eta V_{DS})}{nV_{th}}}(1 - e^{-\frac{V_{DS}}{V_{th}}}) \tag{1.4}$$

The leakage power consumption (P_{leak}) is proportional to the leakage current (I_{leak}) and the supply voltage from which the current is drawn (V_{DD}).

$$P_{leak} = I_{leak} \times V_{DD} \tag{1.5}$$

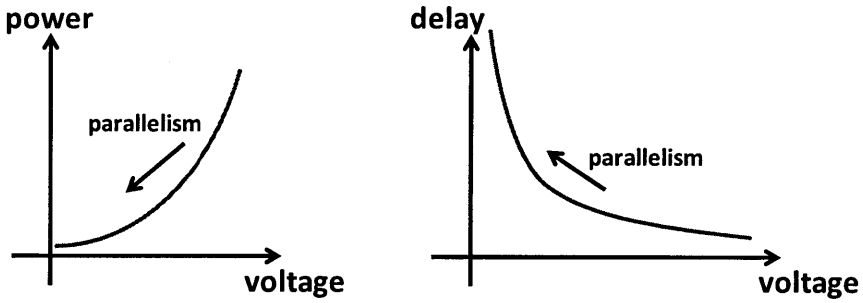
Based on the above relationships, the power consumption can be reduced by lowering the supply voltage (Fig. 1-2a). Voltage scaling on the order of 0.1 to 0.2 V below nominal supply is used in commercial mobile processors as a means to reduce power consumption [19]. Aggressive voltage scaling well below half the nominal supply, and even below the threshold voltage of the devices, has been demonstrated in various test chips. Some examples include a subthreshold FFT processor that can operate down to 180-mV, a tenth of the nominal supply [20], a subthreshold microcontroller with on-chip SRAM and integrated DC-DC power converter that can operate down to 300-mV [21], and subthreshold sensor processors that can operate down to 200-mV [22, 23].

Reducing the supply voltage reduces the speed of the circuit. Specifically, decreasing the supply voltage reduces the switching current (I_D), which increases the time it takes (t_p) to charge a capacitor. The propagation delay of a characteristic inverter with output capacitance C_L is shown in equation 1.6, where κ is the delay fitting parameter.

$$t_p = \frac{\kappa C_L \times V_{DD}}{I_D} \tag{1.6}$$

The behavior of the switching current relative to supply voltage (V_{DD}), and consequently

[3] n is also known as the sub-threshold swing parameter in the BSIM4 model [18]



(a) Reducing power with voltage scaling.

(b) Delay increases at lower voltage.

Figure 1-2: Relationship between voltage, power and delay. Parallelism can be used to mitigate delay to enable lower voltage and consequently lower power consumption.

the delay t_p, changes depending on whether V_{DD} is above or below the threshold voltage (V_t) of the device. When V_{DD} is above V_t, the delay increases approximately linearly with decreasing V_{DD}. When V_{DD} is below V_t (subthreshold), the delay to increases exponentially with decreasing V_{DD}. Fig. 1-2b illustrates the relationship between delay and supply voltage.

In performance constrained applications, parallelism can be used to mitigate the increase in delay by enabling more operations to be done concurrently (i.e. increasing the throughput). As a result, the same overall performance can be achieved at a lower supply voltage and consequently lower power (Fig. 1-2). An example of this approach can be found in [24], where 620 parallel correlators are used to deliver 100-Mbps ultra-wideband demodulation at 400-mV.

The power consumption can be further reduced by over-designing for throughput, and then scaling it back to the target performance with voltage scaling. In this thesis, we will demonstrate how parallel architectures can improve the power-performance tradeoff for an H.264/AVC video decoder.

1.4 Scalability

In video decoding, the frame rate and resolution of the playback video dictates the performance requirement of the video decoder hardware. Over the past years, video has become

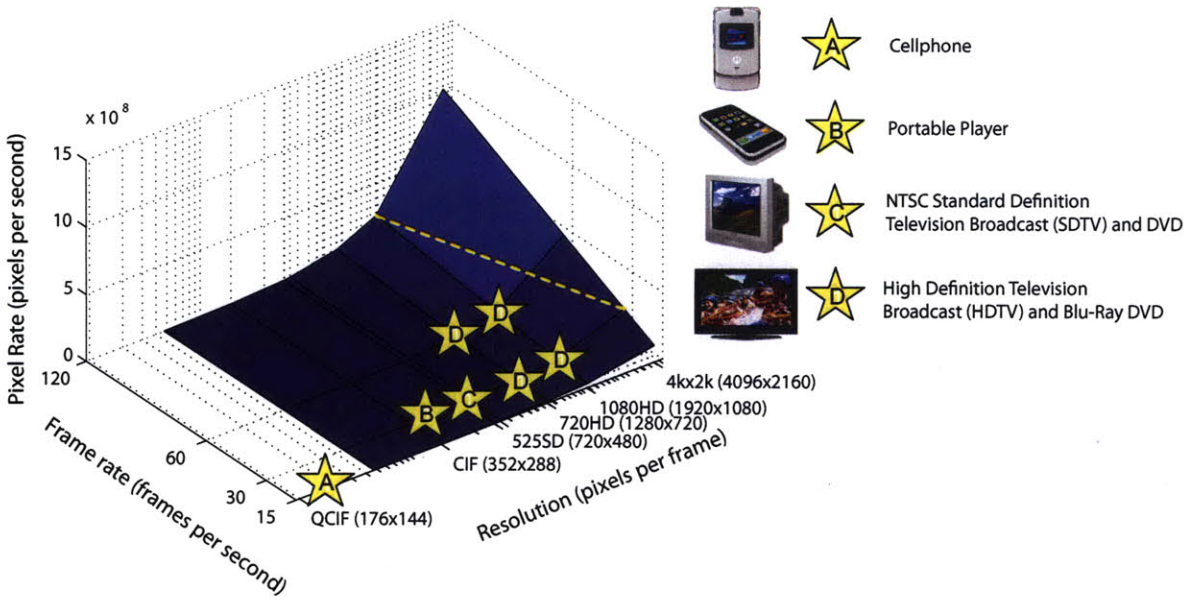


Figure 1-3: Performance requirements for various applications based on frame rate and resolution [25]. Yellow dashed line shows limit of H.264/AVC standard. The next generation standard 'H.265' could potentially reach above this line.

increasingly ubiquitous due to the reduction in storage and transmission costs. The number of different types of video content has been growing rapidly ranging from professional cinema to news reports to, most recently, user-generated content. In addition, the numerous modes of transmission of the video have also expanded from broadcast and playback of local storage material (e.g. DVD), to streaming across the internet and cellular network. Both of these factors cause the frame rate and resolution of today's video content to vary widely. Fig. 1-3 shows the resolutions and frame rates for different applications and their corresponding performance requirement (in terms of pixels/s). For instance, high definition (e.g. 720HD(1280x720) or 1080HD(1920x1080)) is used for playback movies and broadcast television on a high resolution monitor. A higher frame rate (e.g. 60 or 120 fps) is used for high-action sports. Video conferencing and streaming media can be done at lower resolutions (e.g. CIF(352x288) or VGA(640x480)) and frame rates (e.g. 15 or 30 fps) for display on a phone.

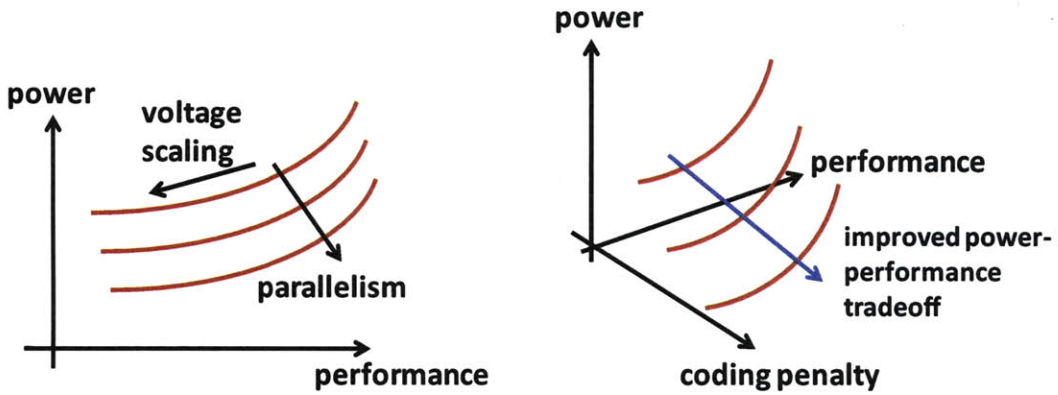
Accordingly, H.264/AVC [26] was designed to support a wide range of resolutions and frame rates as seen in Fig. 1-3. H.264/AVC supports videos from QCIF (176x144) at 15

fps [380 kpixels/s] up to 4kx2k (4096x2160) at 30 fps [265 Mpixels/s]; the performance requirement for 4kx2k at 30 fps is 698x greater than QCIF at 15 fps. It is likely that in the next generation standard [27, 28], both the lower and upper bound of this range will be increased, supporting QVGA (320x240) at 24 fps [1.92 Mpixels/s] up to 4kx2k (4096x2160) at 120 fps [1 Gpixels/s], which covers a performance range of more than 500x. A highly scalable video decoder is needed to support the wide variety of encoded sequences.

The use of video playback on handheld battery-operated devices is increasingly common. It is expected that a video decoder on a cellphone can playback different types of video under various use cases. For instance, it should be able to playback low to medium resolution/frame rate videos locally on the phone that perhaps were transmitted over a low bandwidth network; with the growing popularity of video capture on a cellphone, it may also be convenient to be able to connect the phone to a monitor, or use a pico projector on the phone, and playback high resolution and fast frame rate sequences. Having a single video decoder ASIC that is scalable and can be used for all these applications is convenient and cost effective. Consequently, it is important to minimize and scale the power across this wide range. Ultra-dynamic voltage scaling (UDVS), which involves voltage scaling from nominal down to subthreshold, is an effective method to support the more than 100x workload variation due to video content in an energy efficient manner [29]. In this thesis, we will describe how exposing parallelism in *both* the architecture and algorithm can improve the power-performance tradeoff (Fig. 1-4a) for video decoding. Furthermore, the parallel algorithm enables a third dimension of scalability with coding efficiency as shown in Fig. 1-4b. This approach can also be applied to the video encoding hardware where rather than having the video dictate the performance requirement for video decoding, the user has the ability to select the power-performance-coding efficiency point depending on the desired application.

1.5 Thesis Contributions

This thesis describes the use of parallel architectures and algorithms to meet the low power, high performance, and high coding efficiency requirements outlined in Section 1.1 for next



(a) A more efficient power-performance tradeoff is achieved with parallelism.

(b) Power-performance-coding efficiency tradeoff enabled by parallel algorithm.

Figure 1-4: Tradeoff enabled with parallel architectures and algorithms.

generation video coding. It also explores how efficient tradeoffs can be established across these metrics. In addition, while architecture and algorithms were optimized separately in previous work, this thesis demonstrates how architectures and algorithms can be jointly optimized to deliver improved power and performance at reduced area and coding efficiency cost.

The main contributions in this thesis include

1. **Parallel Architectures.** In Chapter 2, this thesis exposes parallel architectures within the constraints of the H.264/AVC algorithm to reduce the power of an HD video decoder. Other architecture optimizations such as pipelining with first-in-first-out queues (FIFOs), multiple voltage/frequency domains, and voltage-scalable design are also explored. Caching strategies are used to reduce off-chip memory bandwidth. The proposed techniques are demonstrated through the implementation of an H.264/AVC video decoder test chip in 65-nm CMOS [30].

   Three graduate students were involved in the design of the H.264/AVC video decoder. The main architects of the decoder were myself and Daniel Finchelstein. The deblocking filter (DB) unit was my own design, while the inverse transform (IT), intra-prediction (INTRA) and entropy decoding (ED) units were designed by Daniel

Finchelstein. The motion compensation (MC) unit and the system level architecture were a joint effort. The low voltage SRAMs were designed by Mahmut Ersin Sinangil.

2. **Parallel Algorithms.** Entropy coding, in particular, Context-based Adaptive Binary Arithmetic Coding (CABAC), is known to be a highly serial process and a key throughput bottleneck in video coding. In Chapter 3, this thesis presents a Massively Parallel CABAC (MP-CABAC) algorithm to enable real-time low power HD video coding. It leverages a combination of several forms of parallelism [31–33] to enable better coding efficiency-throughput tradeoff and area cost-throughput tradeoff than H.264/AVC. The MP-CABAC is evaluated under common conditions [34] recommended by the Video Coding Experts Group (VCEG) standards body to demonstrate its throughput improvement and coding penalty reduction compared with H.264/AVC CABAC as well as other parallel CABAC approaches. The MP-CABAC has been proposed to VCEG for the next generation video coding standard 'H.265' [35].

   Daniel Finchelstein was involved in the initial conception of interleaved entropy slices (IES) discussed in Section 3.3.3; he integrated the IES approach into the RTL of our H.264/AVC video decoder (i.e. IES for CAVLC) to demonstrate multi-core processing, and performed simulations to evaluate its impact on power and performance at the decoder system level.

3. **Joint Algorithm/Architecture Optimizations.** In Chapter 4, this thesis describes how the MP-CABAC algorithm is mapped to a highly parallel architecture which can support up to 4kx2k. Various architectural optimizations are performed to reduce its critical path delay. Joint algorithm/architecture optimizations enable additional critical path and area reduction with zero and negligible cost in coding efficiency, respectively. Configurability and clock gating strategies are applied to enable scalability with low power cost. The tradeoffs between power, performance and

coding efficiency of the MP-CABAC are demonstrated on a highly scalable test chip in 65-nm CMOS in Chapter 5.

Details on all video sequences used in this thesis can be found in Appendix E.

Chapter 2

Low Power Architecture for H.264/AVC Video Decoder

This chapter describes the design of an H.264/AVC video decoder that uses parallel architectures to enable low voltage operation for reduced power consumption [36]. In order to achieve the performance required for real-time HD video decoding (i.e. maintain a target frame rate), architectural optimizations such as parallelism and pipelining were used to reduce the number of cycles required per frame and enable low voltage operation. Additional power savings were achieved by dividing the decoder into multiple voltage/frequency domains and applying dynamic voltage and frequency scaling (DVFS) on each domain to efficiently adapt to the varying workloads.

2.1 Overview of Video Coding Standard H.264/AVC

Video compression is achieved by removing redundant information in the video sequence. In May 2003, ISO/IEC Moving Picture Experts Group (MPEG) and the ITU-T VCEG introduced a video coding standard called H.264/AVC. It is the latest standard in a series of video coding standards (MPEG-1, MPEG-2, MPEG-4) and (H.261, H.263) released by each group respectively. Fig. 2-1 shows the video encoding and decoding structure.

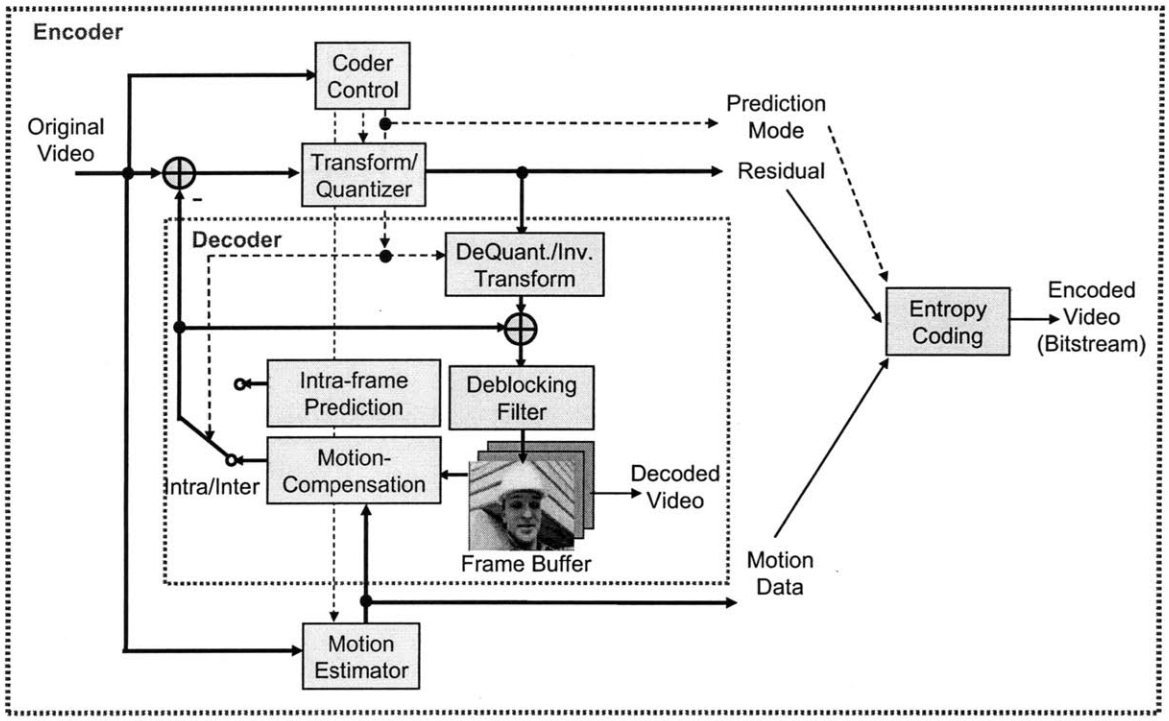


Figure 2-1: Basic video coding structure for H.264/AVC, the state-of-the-art video coding standard. Note that the decoder is embedded within the encoder. This figure is adopted from [37].

2.1.1 Partitioning the Video Sequence

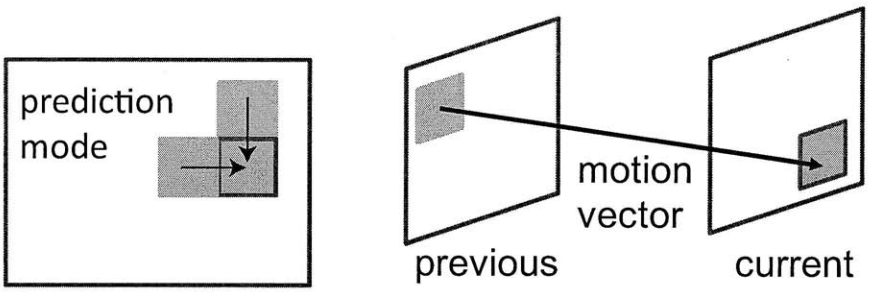
Each pixel in the video stream is composed of one luma and two chroma components. This format is often referred to as YUV or YCbCr. Luma represents the brightness of the image, while chroma provides the color information. These three components can be transformed into Red Green Blue (RGB) components that are used for display. Since the human visual system is less sensitive to color than brightness, the chroma components can be down-sampled by 2 in both the vertical and horizontal direction; this is known as the 4:2:0 scheme.

In H.264/AVC, each frame of the video can be broken into several slices. Slices are self-contained such that they can be decoded without knowledge of other slices (enables resynchronization). The slices are then divided into blocks of 16x16 pixels called macroblocks, which can then be further divided into blocks of 8x16, 16x8, 8x8, 4x8, 8x4 down to 4x4 pixels.

2.1.2 Prediction

Video coding algorithms exploit temporal and spatial redundancy in order to reduce the size of the video stream (Fig. 2-2). Motion compensation (also known as inter-prediction) exploits temporal redundancy by predicting a macroblock in the current frame from previous (and/or future) frames. At the encoder, motion estimation is performed to determine the vertical and horizontal translation (motion vector) of a given macroblock, or block, relative to a previous frame. The previous frames are stored in a frame buffer. The intra-prediction exploits spatial redundancy by predicting a macroblock in the current frame from surrounding pixels in the same frame. The encoder must decide which surrounding pixels to use and the prediction direction (e.g. vertical, horizontal, diagonal, etc.); there are 9 possible prediction modes (directions) in H.264/AVC [26].

Each frame can be classified as an I-frame for intra-prediction, P-frame for motion compensation (with only previous frames), and B-frame for bi-directional motion compensation (with both previous and future frames). While I frames contain only intra-predicted macroblocks, P-frames can contain both intra- and inter- predicted macroblocks, and B-frames can contain intra-, inter-, and bi-directional inter-predicted macroblocks. Intra-prediction



(a) spatial prediction (I-frame)

(b) temporal prediction (P-frame)

Figure 2-2: Spatial and temporal prediction.

typically has poorer coding efficiency than motion compensation.

Rather than transmitting a 16x16 block of pixels for every macroblock, motion vectors and the index of the previous frame (also known as reference index) are sent for the motion compensated macroblocks, and the intra-prediction modes are sent for the intra-predicted macroblocks. Using this information, the decoder can generate the macroblock prediction.

At the encoder, the coder control determines whether inter- or intra- prediction is used by performing rate-distortion optimization.

2.1.3 Transform and Quantization

Note that the error of the predicted macroblock relative to the original macroblock, called the residual, must also be transmitted. The residual can be compressed by transforming it into a different domain (e.g. frequency domain) that has good energy compaction properties, where the majority of the energy lies in a few coefficients. These coefficients are quantized to provide the desired encoded bit-rate. Quantization results in lossy video compression. Following quantization, the number of non-zero coefficients is much lower than the number of pixels, which means that fewer bits need to be transmitted to represent the residual. Fig. 2-3 illustrates this process. At the decoder, an inverse transform is performed on the coefficients to recover the quantized residual, which is then added to the prediction to form a reconstructed macroblock.

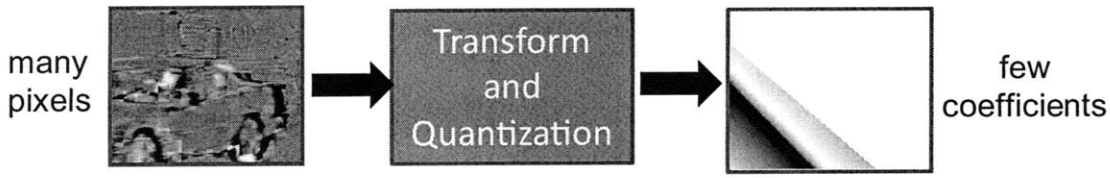


Figure 2-3: Transform and quantization

At the encoder, the coder control selects the desired amount of quantization for a target bit-rate using rate-control.

2.1.4 Entropy Coding

The data to be transmitted (e.g. motion vectors, reference indexes, intra-prediction modes, residual coefficients) is referred to as the syntax elements. The syntax elements undergo entropy coding, which leverages their probabilistic distribution to deliver additional lossless compression. The two forms of entropy coding used in H.264/AVC are Context-based Adaptive Variable Length Coding (CAVLC) and Context-based Adaptive Binary Arithmetic Coding (CABAC). CABAC is highly adaptive which enables it to provide better coding efficiency at the cost of increased complexity. CABAC will be discussed in detail in Chapter 3.

2.1.5 Deblocking

Deblocking is necessary to remove blocking artifacts that can arise between the reconstructed blocks as a result of the lossy block-based processing done by the H.264/AVC encoder. The challenge for the deblocking filter is that it needs to remove blocking artifacts due to quantization, while retaining true edges in the video. Consequently, an adaptive filter is used and the strength of the filter across boundary edges depends on macroblock and pixel information.

2.1.6 GOP and Prediction Structure

A series of frames (pictures) are grouped together and referred to as a group of pictures (GOP). The prediction structure indicates the type of prediction used between the frames in a GOP as shown in Fig. 2-4. For instance, H.264/AVC uses

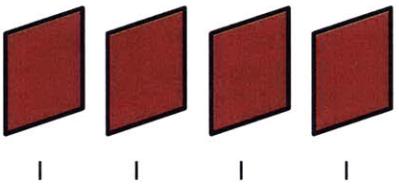
- **Ionly**: Only I-frames.
- **IPPP**: An I-frame followed by a series of P-frames. The first P-frame is predicted from the I-frame, while the subsequent P-frames are predicted by the previous P-frame within the GOP.
- **IBBP**: An I-frame followed by a series of B- and P-frames. The P-frame is predicted from the I-frame. The B-frames are predicted from the I- and P-frame.

A GOP always begins with an I-frame. The GOP size refers to the number of frames in the GOP.

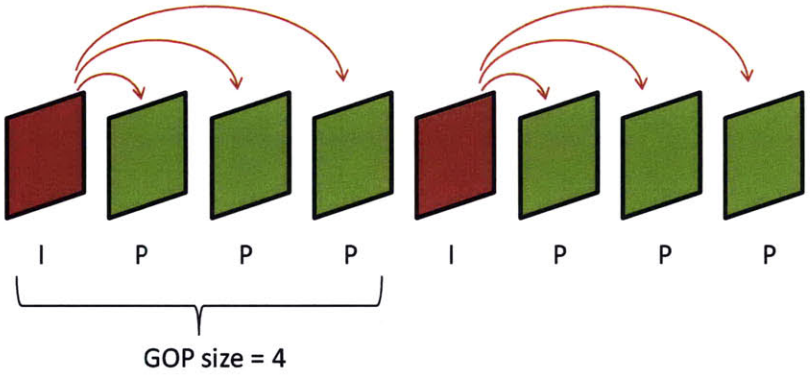
2.1.7 Profiles and Levels

H.264/AVC has various profiles and levels settings so that it can cover a wide range of applications and performance needs. The profiles (Baseline, Main, High, and Extended) define a set of coding tools: Baseline uses the simplest tools with the lowest coding efficiency and Extended uses the most complex tools with the highest coding efficiency. Currently, consumer products have been using the Baseline and High profiles. The levels describe the resolution and frame rate of the video that can be supported.

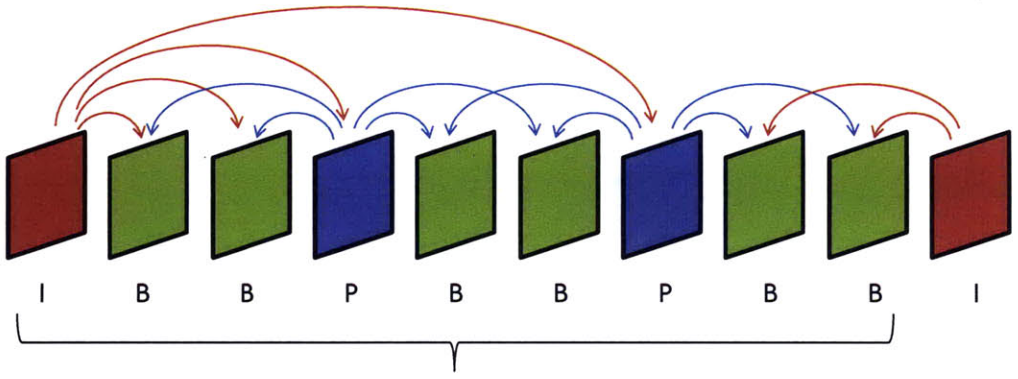
In this chapter, the focus will be on the design of an H.264/AVC Baseline decoder supporting up to level 3.2, which includes 720p (1280x720 pixels) decoding at 30 fps. The CABAC engine used in the H.264/AVC High Profile will be investigated in the latter half of the thesis.



(a) Ionly



(b) IPPP



(c) IBBP

Figure 2-4: Different group of pictures (GOP) structures.

2.2 Related Work

State-of-the-art ASIC H.264/AVC decoders [38–41] have used various micro-architecture techniques to reduce the *number of operations*, which lowers the operating frequency and consequently power consumption. In this work, *additional* architecture optimizations are introduced that enable aggressive voltage scaling to lower the *energy per operation*, which further reduces the power consumption of HD decoding [36].

In [40, 41], the cycle count required for motion compensation and deblocking filtering is reduced by optimizing the processing order to eliminate redundant memory accesses. In this work, the cycle count is further reduced by identifying inputs to these units, as well as others, that can be processed in parallel. A hybrid 4x4-block/macroblock pipeline scheme is adopted in [38–41]. In this work, a 4x4 block pipelining scheme with optimally sized FIFOs between stages is used to adapt for workload variability and consequently increase the decoder throughput. Finally, [38–41] as well as this work reduce external (off-chip) memory bandwidth by maximizing data reuse and exploiting various different caching schemes. Internal memories consume a significant portion of the core power [40]. In this work, the caches are implemented with custom voltage scalable SRAMs to further minimize memory access power.

2.3 Decoder Pipeline Architecture

We will now discuss our low power H.264/AVC Baseline decoder. The top-level architecture of the decoder is shown in Fig. 2-5. At the system level of the decoder, FIFOs of varying depths connect the major processing units: entropy decoding (ED),[1] inverse transform (IT), motion compensation (MC), intra-prediction (INTRA), deblocking filter (DB), memory controller (MEM) and frame buffer (FB). The pipelined architecture allows the decoder to process several 4x4 blocks of pixels simultaneously, requiring fewer cycles to decode each frame.

[1]Note that in this section, CAVLC is used for ED as a Baseline profile H.264/AVC decoder was implemented. The next section will discuss the case where CABAC is used for ED in High profile.

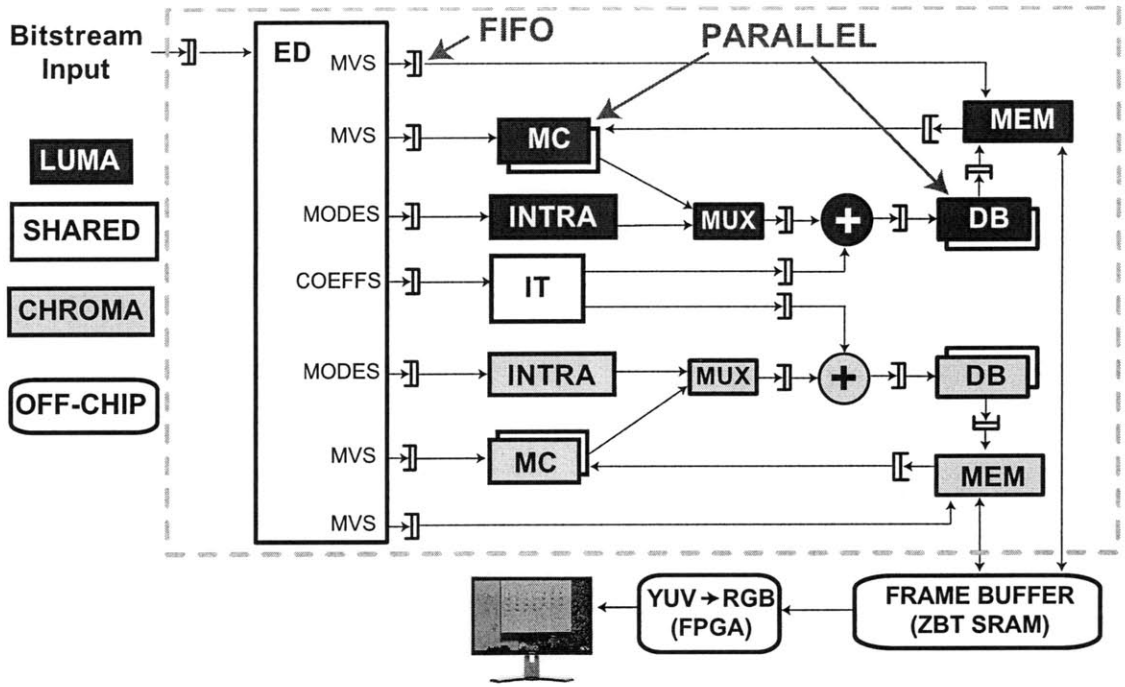


Figure 2-5: H.264/AVC decoder architecture.

The number of cycles required to process each 4x4 block varies for each unit as shown in Table 2.1. The table describes the pipeline performance for decoding P-frames (temporal prediction). Most of the optimization efforts were focused on P-frame performance, since they occur more frequently than I-frames (spatial prediction) in highly compressed videos.

The luma and chroma components have separate pipelines that share minimal hardware and are mostly decoupled from each other. In most cases, the luma and chroma components of each 4x4 block are processed at the same time which enables further cycle count reduction. However, the two pipelines do have dependencies on each other, which sometimes prevents them from running at the same time. For example, both pipelines use the same ED at the start, since this operation is inherently serial and produces syntax elements for both pipelines. To reduce hardware costs, the luma and chroma pipelines also share the IT unit, since this unit has a low cycle count per block relative to the rest of the units as shown in Table 2.1.

The workload of each unit varies depending on the prediction mode as well as the content of the video sequence. To adapt for the workload variation of each unit, FIFOs were inserted

Table 2.1: Cycles per 4x4 block for each unit in P-frame pipeline of Fig. 2-5 assuming no stalling. The values were measured for 300 frames of the mobcal sequence. Each 4x4 block include a single 4x4 luma block and two 2x2 chroma blocks. [ ] is performance after Section 2.4 optimizations.

Pipeline Unit		Min Cycles	Max Cycles	Avg Cycles
ED		0	33	4.6
IT		0	4	1.6
Luma	MC	4 [2]	9 [4.5]	4.6 [2.3]
	DB	8 [2]	12 [6]	8.9 [2.9]
	MEM	8	31	18
Chroma	MC	8 [2]	8 [2]	8 [2]
	DB	5 [2.5]	8 [4]	6.6 [3.3]
	MEM	10	10	10

between each unit. The FIFO issues full and empty signals to the units connected to its input and output respectively, to prevent overflow and underflow. These units will stall until the full or empty signal is lowered. These FIFOs also distribute the pipeline control and allow the units to operate out of lockstep to adapt to the variable latency of the units [42]. The FIFOs help to average out the cycle variations which increases the throughput of the decoder by reducing the number of stalls, as described in [43]. Increasing the depth of the FIFOs enables more averaging and higher throughput. Fig. 2-6 shows that the pipeline performance can be improved by up to 45% by increasing the depths of the 4x4 block FIFOs in Fig. 2-5. For very large FIFO depths, all variation-related stalls are eliminated and the pipeline performance approaches the rate of the unit with the largest average cycle count. This performance improvement must be traded off against the additional area and power overhead introduced by larger FIFOs.

In the simulation results presented in Fig. 2-6, all FIFOs were set to equal depths. However, deeper FIFOs should ideally be used only when they provide a significant performance improvement. For example, placing a deeper FIFO between the ED and IT unit reduces many stalls, but a minimum-sized FIFO between the DB and MEM units is sufficient. In the decoder test chip, FIFO depths between 1 and 4 were chosen in order to reduce FIFO

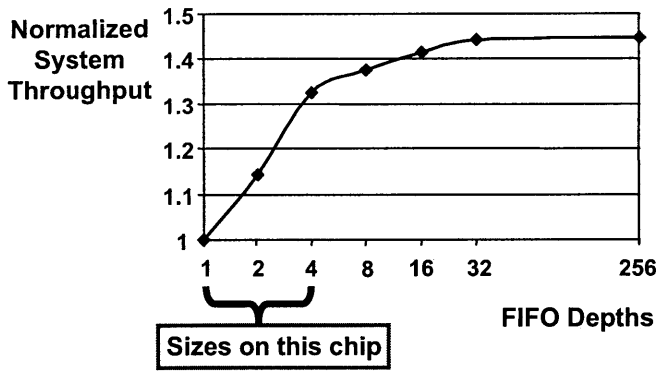


Figure 2-6: Longer FIFOs average out workload variations to minimize pipeline stalls. Performance simulated for mobcal sequence.

area while still reducing pipeline stalls.

2.4 Parallelism

Parallelism can be used within each processing unit to reduce the number of cycles required to process each 4x4 block and balance out the cycles in each stage of the pipeline. This is particularly applicable to the MC and DB units, which were found to be key bottlenecks in the system pipeline. This is not surprising as they were identified as the units of greatest complexity in the Baseline profile of H.264/AVC [44].

2.4.1 Motion Compensation (MC)

Given a motion vector, the MC unit predicts a 4x4 block in the current frame from pixels in the previous frames to exploit temporal redundancy. The previous frames are stored in the frame buffer. When the motion vector is integer-valued (full-pel), the predicted 4x4 block can be found in its entirety in a previous frame, as shown in the left part of Fig. 2-7. For increased coding efficiency, motion vectors in H.264/AVC can have up to quarter-pel resolution. When either the X or Y component of the motion vector is fractional, the predicted 4x4 block must be interpolated from pixels at full-pel locations in previous frames, as shown in the right part of Fig. 2-7. Accordingly, the main operation in motion compensation involves

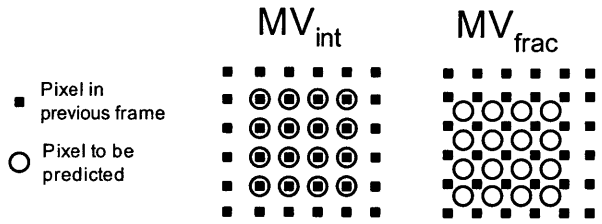


Figure 2-7: Integer and fractional motion vectors.

interpolation and filtering. The luma and chroma components are interpolated differently. Luma interpolation involves using a 1-D 6-tap filter to generate the half-pel locations. A 1-D bilinear filter is then used to generate the quarter-pel locations [26]. Thus, a 4x4 luma block is predicted from an area of at most 9x9 pixels in a previous frame. Chroma interpolation involves the use of a 2-D bilinear filter and each 2x2 chroma block is predicted from an area of 3x3 pixels.

The luma interpolator architecture is shown in Fig. 2-8 and is similar to the design in [45]. The datapath of the luma interpolator is made up of (6x5) 8-bit registers (full-pel), (6x4) 15-bit registers (half-pel), (4+9) 6-tap filters, and four bilinear filters. The interpolator uses a 6-stage pipeline. At the input of the pipeline, for vertical interpolation, a column of 9 pixels is read from the frame buffer and used to interpolate a column of 4 pixels. A total of 9 pixels, representing the full and half-pel locations, are stored at every stage of the interpolator pipeline; specifically, the 4 interpolated half-pel pixels and the 5 center (positions 3 to 7) full-pel pixels of the 9 pixels from the frame buffer are stored. The 9 registers from the 6 stages are fed to 9 horizontal interpolators. Finally, 9:2 muxes are used to select two pixels located at full or half-pixel locations as inputs to the bilinear interpolator for quarter-pel resolution.

To improve the throughput of MC, a second identical interpolator is added in parallel as shown in Fig. 2-9. The first interpolator predicts 4x4 blocks on the even rows of a macroblock, while second predicts 4x4 blocks on the odd rows. This parallel structure can double the throughput of the MC unit if during each cycle, both motion vectors are available and two new columns of 9 pixels from the frame buffer are available at the inputs of both interpolators. The chroma interpolator is also replicated four times such that it can predict a 2x2 block

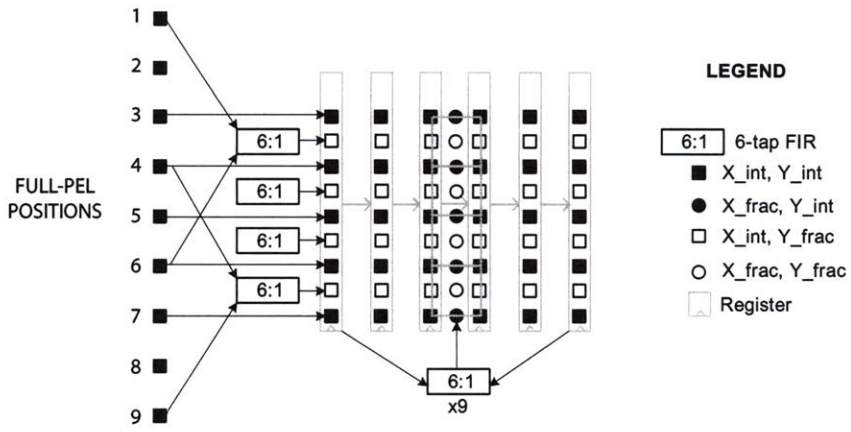
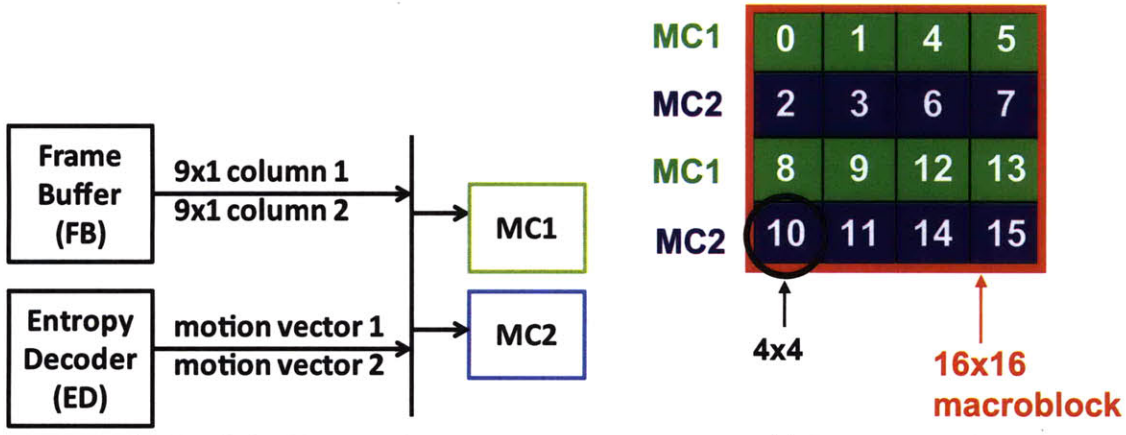


Figure 2-8: Luma interpolator pipelined architecture.



(a) Parallel MC interpolators

(b) Processing order.

Figure 2-9: Parallel MC interpolators. Note: the numbered 4x4 blocks reflect the processing order within a macroblock defined by the H.264/AVC standard.

every cycle. Additional interpolators can be used in parallel as discussed in [46], where the use of four luma interpolators is shown to be optimal. However, for this decoder, Table 2.1 shows that using two luma interpolators is sufficient for alleviating the MC bottleneck.

2.4.2 Deblocking Filter (DB)

The deblocking filter is applied using four pixels on either side of an edge, as shown in Fig. 2-10. The strength of the adaptive deblocking filter depends on both slice, block and pixel (sample) level information [47]. At a slice level, the filter can be enabled/disabled for

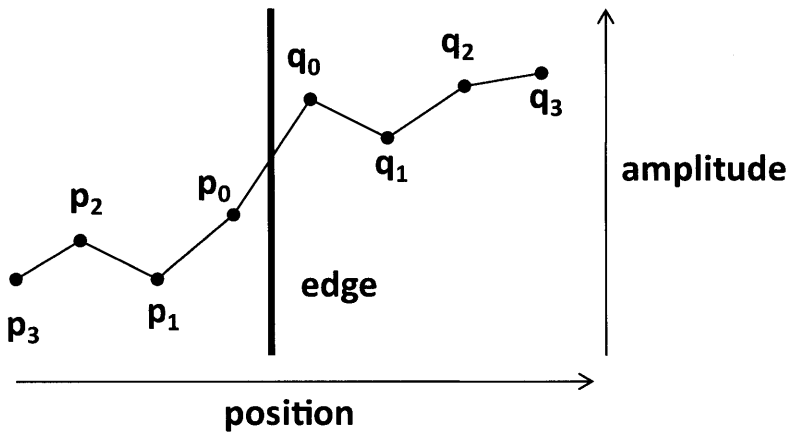


Figure 2-10: The four pixels on either side of the edge are used as inputs to the deblocking filter.

all macroblocks. At the block level, a boundary strength (Bs) is computed for an edge on a 4x4 block basis. Bs is assigned a value from 4(strong) to 0(none) based on whether the edge is a macroblock edge, intra prediction is used, non-zero coding residual exists, and whether the motion vectors and reference frames are different. Finally, at a pixel level, if Bs is not 0, the pixels around the edges are compared to various thresholds (α and β), which depend on QP, to determine whether the edge is likely an artifact due to quantization or a true edge. Different filters are used for each pixel. Fig. 2-11 shows the architecture of one such filter.

The boundary strength (Bs) of the adaptive filter is the same for all edges on a given side of a 4x4 block. Accordingly, the DB is designed to have 4 luma and 2 chroma filters running in parallel, which share the same boundary strength calculation, and filter an edge of a 4x4 block every cycle. The luma architecture is shown in Fig. 2-12. For additional cycle reduction, the luma and chroma filters operate at the same time, assuming the input data and configuration parameters are available.

A luma macroblock (16 4x4 blocks) has 128 pixel edges that need to be filtered, so with 4 luma filters a macroblock takes 32 clock cycles to complete. Unlike previous implementations [48], filtering on 4x4 blocks begins before the entire macroblock is reconstructed. To minimize stalls, the edge filtering order, shown in Fig. 2-13, was carefully chosen to account for the 4x4 block processing order (shown in Fig. 2-9b) while adhering to the left-right-top-bottom

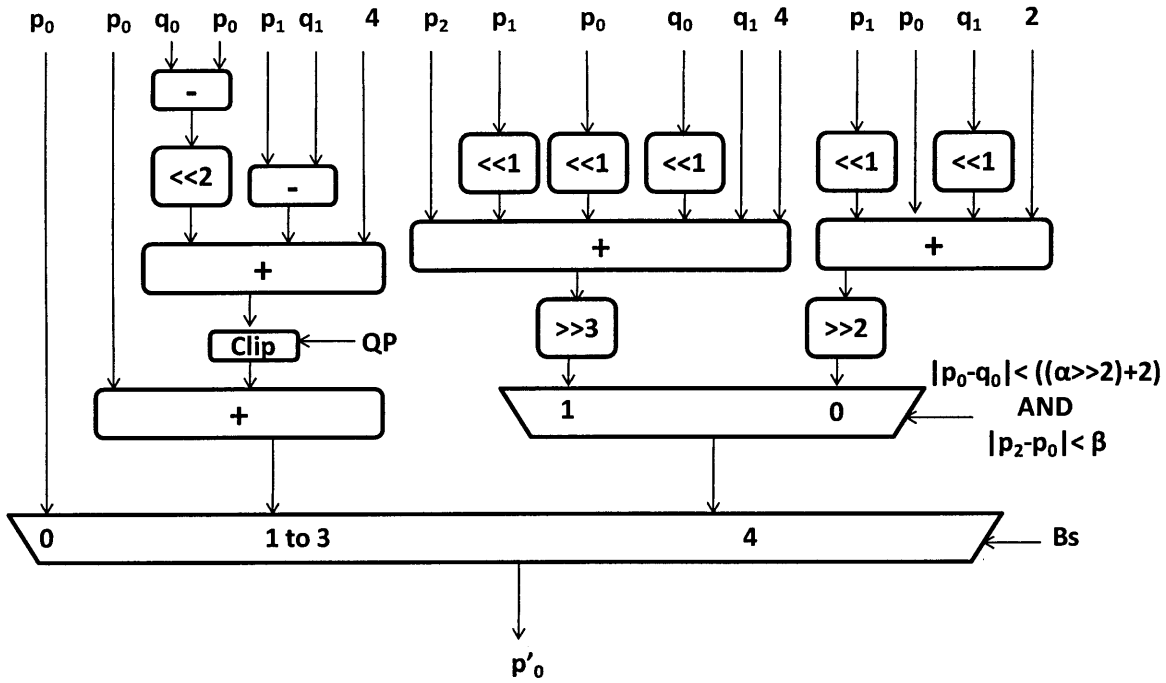


Figure 2-11: Architecture of luma filter for pixel p_0 (q_0 uses a similar filter). Different filters are used for pixels p_1, q_1, p_2, and q_2. Filters are defined in [26]. Pixels p_3 and q_3 are not modified.

edge order specified in H.264/AVC [26].

A single-port on-chip SRAM cache with 4x4x8-bit data-width is used to store pixels from the top and left macroblocks. Due to the 4x4 block processing order and the edge order constraints, certain 4x4 blocks need to be stored temporarily before all four of its edges are filtered and it can be written out. These partially filtered blocks are either stored in the on-chip cache or a 'scratch pad' made of flip-flops that can hold up to four 4x4 blocks. This 'scratch pad' along with the chosen edge filtering order minimize the number of stalls from read/write conflicts to the on-chip cache. The on-chip cache has a 2-cycle read latency resulting in a few extra stall cycles when prefetching is not possible. Taking into account this overhead, the average number of cycles required by DB for a luma 4x4 block is about 2.9.

Each of the two chroma components of a macroblock has 32 pixel edges to be filtered. Using 2 filters per cycle results in slightly more than 32 clock cycles per macroblock. When

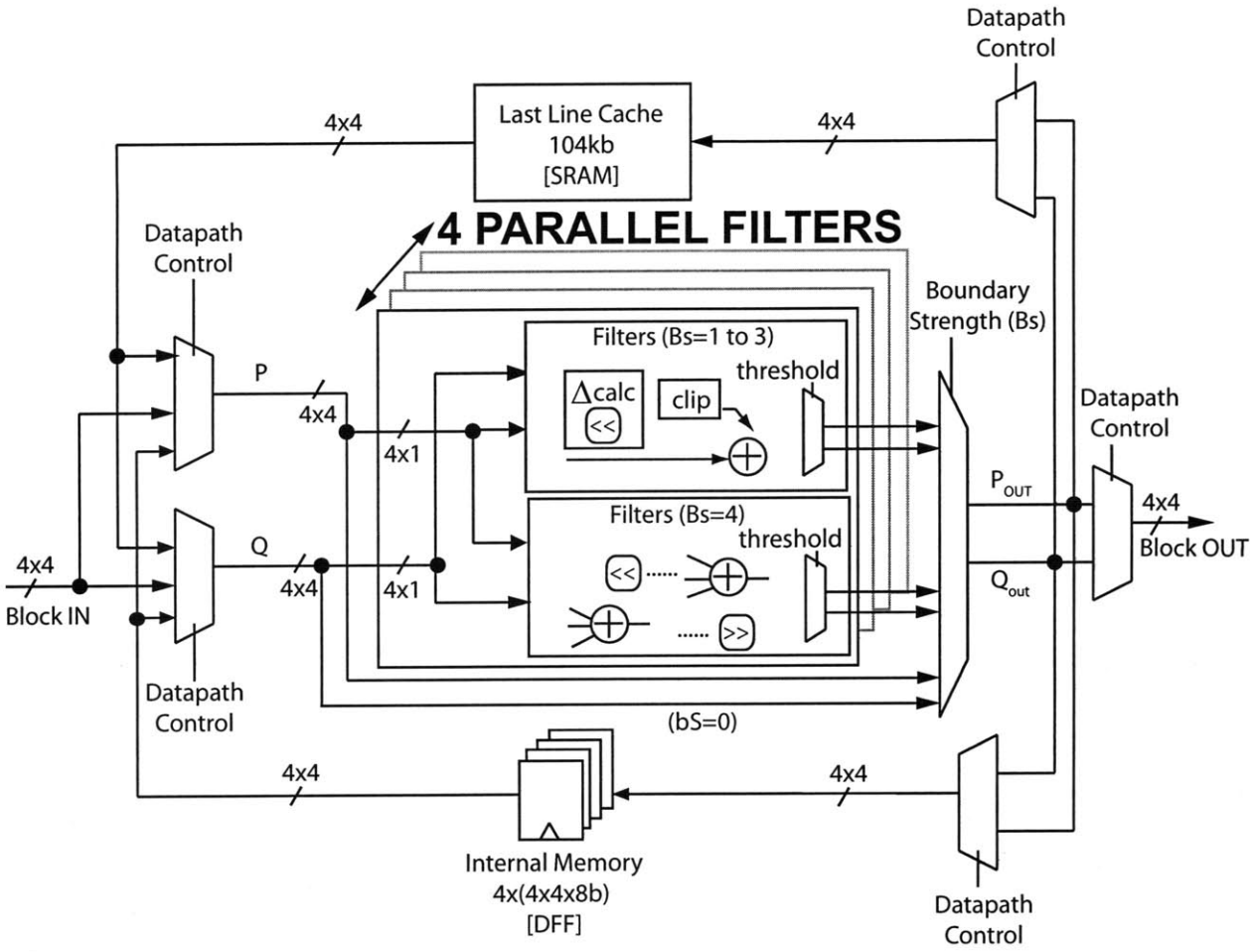


Figure 2-12: Parallel deblocking filter architecture for luma filtering.

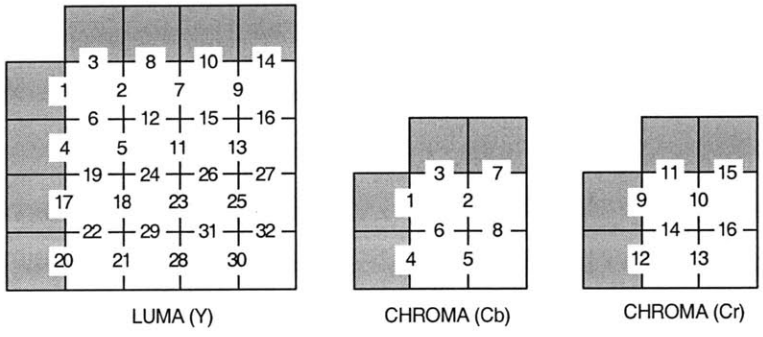


Figure 2-13: Deblocking edge filtering order.

accounting for stalls, the number of cycles per macroblock is about the same as luma. Overall, the number of cycles required by DB is 52 cycles per macroblock, which is less than other deblocking filter implementations that take between 204 to 6144 cycles per macroblock [48].

2.5 Multiple Voltage/Frequency Domains

The decoder interfaces with two 32-bit off-chip SRAMs which serve as the frame buffer (FB). To avoid increasing the number of I/O pads, the MEM unit requires approximately 3x more cycles per 4x4 block than the other processing units, as shown in Table 2.1. In a single domain design, MEM would be the bottleneck of the pipeline and cause many stalls, requiring the whole system to operate at a high frequency in order to maintain performance. This section describes how the architecture can be partitioned into multiple frequency and voltage domains.

Partitioning the decoder into two domains (memory controller and core domain) enables the frequency and voltage to be independently tailored for each domain. Consequently, the core domain, which can be highly parallelized, fully benefits from the reduced frequency and is not restricted by the memory controller with limited parallelism (due to limitations in the number of I/O pads).

The two domains are completely independent and are separated by asynchronous FIFOs as shown in Fig. 2-14. Voltage level-shifters are used for signals going from a low to high voltage. Table 2.1 shows that there could be further benefit to also placing the ED unit on a separate third domain. The ED is difficult to speed up with parallelism because it uses variable length coding which is inherently serial.[2]

Table 2.2, shows a comparison of the estimated power consumed by the single domain design versus a multiple (two and three) domain design. The frequency ratios are derived from Table 2.1 and assume no stalls. For a single domain design, the voltage and frequency must be set at the maximum dictated by the worst case processing unit in the system. It can be seen that the power is significantly reduced when moving from one to two domains.

[2]We will be addressing this challenge at the algorithm level in Chapter 3.

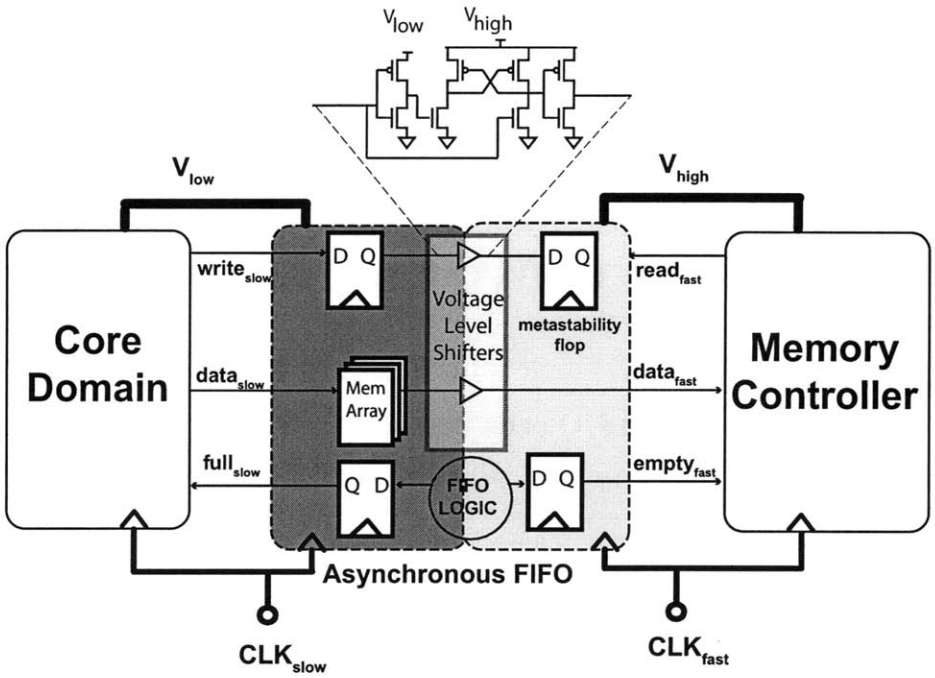


Figure 2-14: Independent voltage/frequency domains are separated by asynchronous FIFOs and level-converters.

The additional reduction for moving to three domains is less; thus, for our design we chose to use two domains.

In summary, several approaches are used to increase the performance of the video decoder. First, pipelining is used to enabled multiple blocks of pixels to be simultaneously processed by different units. Second, FIFOs are used to average out workload variation across different units to increase the average throughput. Third, the throughput of bottleneck units is increased using parallelism. Finally, if the throughput is still not enough to meet the performance requirements, or the unit cannot be parallelized, then the unit is placed on a separate voltage/frequency domain, where it runs at a higher operating frequency.

2.6 Dynamic Voltage and Frequency Scaling (DVFS)

The video decoder has a highly variable workload due to the varying prediction modes that enable high coding efficiency. While FIFOs are used in Section 2.3 to address workload

Table 2.2: Estimated impact of multiple domains on power for decoding a P-frame.

				720p				1080p			
	Frequency Ratio			Voltage			Power	Voltage			Power
Domains	ED	Core	MEM	ED	Core	MEM	[%]	ED	Core	MEM	[%]
One	1.00	1.00	1.00	0.81	0.81	0.81	100	0.98	0.98	0.98	100
Two	0.26	0.26	1.00	0.66	0.66	0.78	75	0.73	0.73	0.94	67
Three	0.26	0.18	1.00	0.66	0.63	0.78	71	0.73	0.69	0.94	63

variation at the 4x4 block level, DVFS and frame averaging allow the decoder to address the varying workload at the frame level in a power efficient manner [49].

DVFS involves adjusting the voltage and frequency based on the varying workload to minimize power. This is done under the constraint that the decoder must meet the deadline of one frame every 33 ms to achieve real-time decoding at 30 fps. The two requirements for effective DVFS include accurate workload prediction and the voltage/frequency scalability of the decoder. Several techniques are proposed in [50–53] to predict the varying workload during video decoding.

DVFS can be performed independently on the core domain and memory controller as their workloads vary widely and differently depending on whether the decoder is working on I-frames or P-frames as shown in Fig. 2-16. For example, the memory controller requires a higher frequency for P-frames versus I-frames. Conversely, the core domain requires a higher frequency during I-frames since more residual coefficients are present and they are processed by the ED unit. Note that the workload variation between I-frame and P-frame is much larger than the variation across different P-frames and different I-frames as shown in Fig. 2-16; thus, we investigate the impact of DVFS when changing between an I-frame and P-frame but not across P-frames or I-frames (i.e. for our analysis, we assume all I-frames, and all P-frames, have the same workload). Table 2.3 shows the required frequencies and voltages of each domain for an I-frame and P-frame of the mobcal sequence.[3]

[3]In this work, the frequency and voltage are manually determined for each sequence. Workload prediction techniques are needed to close the loop and automatically determine (estimate) the workload of a sequence, which can then be mapped to a frequency and voltage.

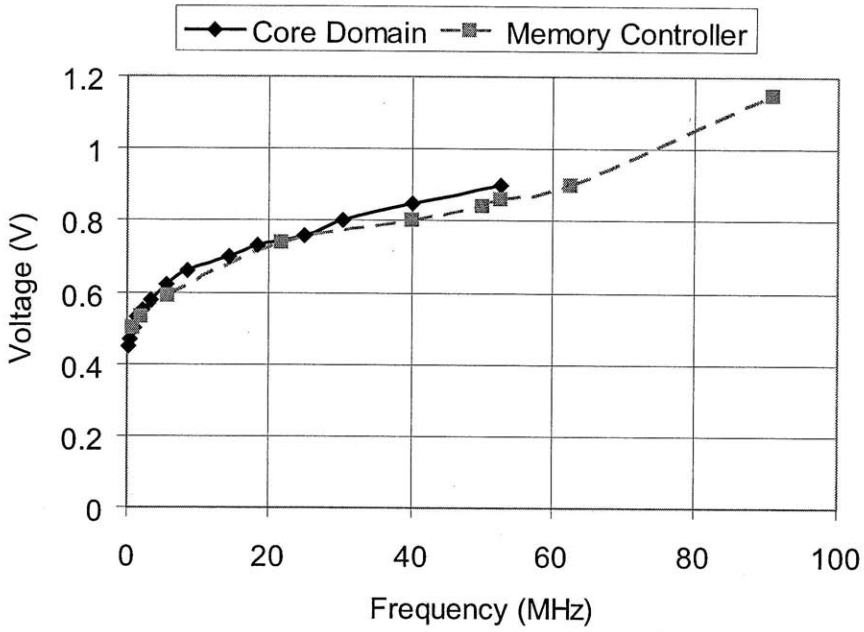


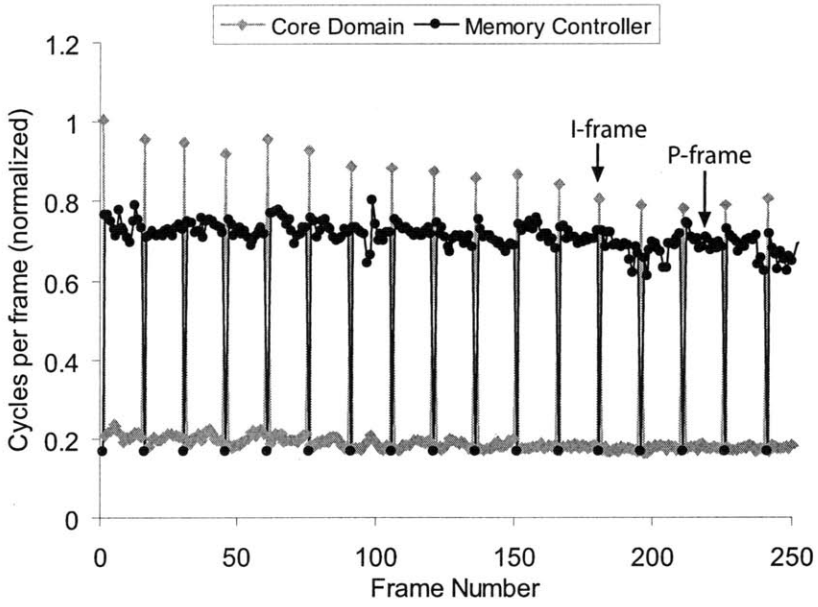
Figure 2-15: Measured frequency versus voltage for core domain and memory controller. Use this plot to determine maximum frequency for given voltage.

Fig. 2-15 shows the frequency and voltage range of the two domains in the decoder. Once the desired frequency is determined for a given workload, the minimum voltage can be selected from this graph.

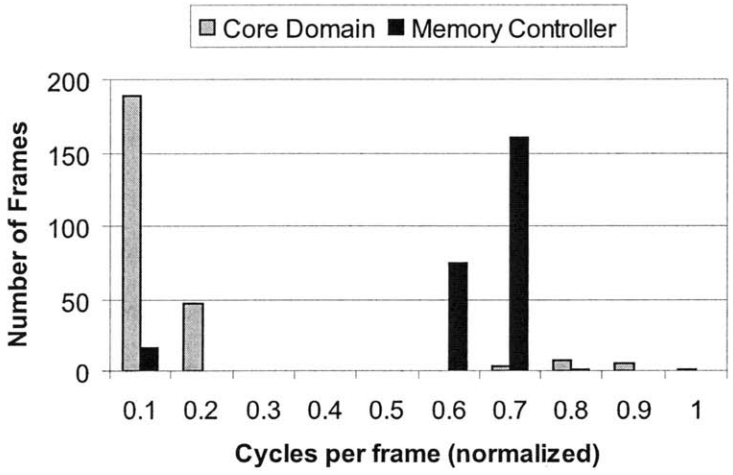
If latency can be tolerated, frame averaging can also reduce the frequency requirement and consequently power consumption [49]. Frame averaging does not have the overhead switching power that is consumed during DVFS. Table 2.4 shows the impact of DVFS and frame averaging for a GOP size of 2 with a GOP structure of IPPP (i.e. I-frame followed by

Table 2.3: Measured voltage/frequency for each domain for I-frame and P-frame for 720p sequence.

Frame Type	Core		Memory Controller	
	Frequency [MHz]	Voltage [V]	Frequency [MHz]	Voltage [V]
P	14	0.70	50	0.84
I	53	0.90	25	0.76



(a) Cycles per frame (workload) across sequence.



(b) Distribution of cycle variation.

Figure 2-16: Workload variation across 250 frames of mobcal sequence.

Table 2.4: Estimated impact of DVFS and frame averaging (FA) for GOP structure of IPPP and size 2.

Method	Core		Memory Controller		Relative Power [%]
	Frequency [MHz]	Voltage [V]	Frequency [MHz]	Voltage [V]	
No DVFS/FA	53	0.90	50	0.84	100
DVFS	14 or 53	0.70 or 0.90	25 or 50	0.76 or 0.84	85
FA	33.5	0.82	37.5	0.79	84

a series of P-frames; the GOP size is the period of I-frames). As the GOP size increases, the power savings of DVFS approaches that of frame averaging. For DVFS, the total power is computed as the average of the I-frame and P-frame decoding powers based on the I/P ratio in the GOP; for frame averaging, the total power is computed using the average frequency for I-frame and P-frame decoding based on the I/P ratio in the GOP. DVFS is done at the frame level, and the frame averaging is done across the entire GOP. DVFS can be done in combination with frame averaging for improved workload prediction and additional power savings [53, 54].

2.7 Memory Optimization

Video processing involves movement of a large amount of data. For high definition, each frame is on the order of megabytes (MB), which is too large to place on-chip (e.g. 1.4 MB/frame for 720p and 2.1 MB/frame for 1080p). Consequently, the frame buffer (FB) used to store the previous frames required for motion compensation is located off-chip. It is important to minimize the off-chip memory bandwidth in order to reduce overall system power.

Two key techniques are used to reduce this memory bandwidth. The first reduces both reads and writes such that only the DB unit writes to the frame buffer and only the MC unit reads from it. The second reduces the number of reads by the MC unit. The impact of the two approaches on the overall off-chip memory bandwidth can be seen in Fig. 2-17.

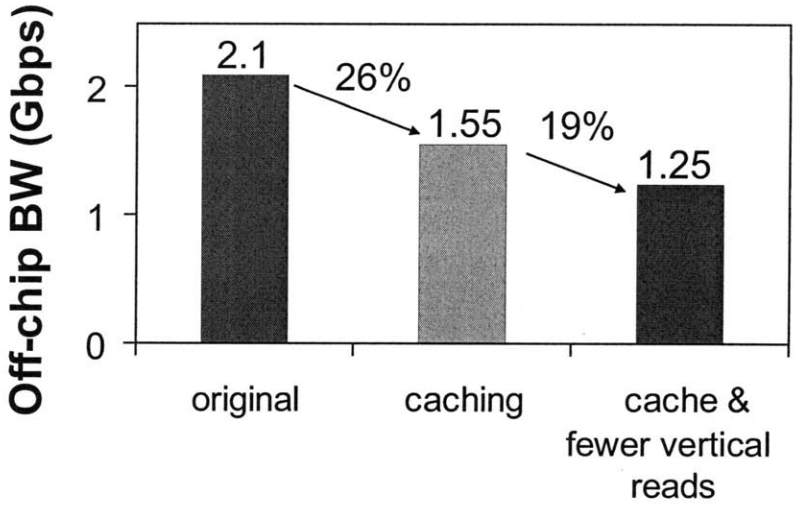


Figure 2-17: Reduction in overall memory bandwidth from caching and reuse MC data.

2.7.1 On Chip Caching

To leverage spatial correlation, decoding a block/macroblock often requires information about its left and top neighboring blocks/macroblocks. Data about the most recently decoded row (line) must be stored in order to satisfy the top dependencies. This includes data such as motion vectors and the last four lines of pixels that are required by the deblocking filter. To minimize off-chip memory bandwidth, this data is stored in several separate on-chip caches as shown in Fig. 2-18. These caches are often referred to as the last line buffer. For a P-frame, this caching scheme reduces total off-chip bandwidth by 26% relative to the case where no caches are used. The memory bandwidth and size of each cache is shown in Table 2.5.

Since conventional 6-transistor SRAMs cannot operate at low voltages, custom low-voltage 8-transistor SRAMs based on [55] were designed to operate at the desired core voltage and frequency.

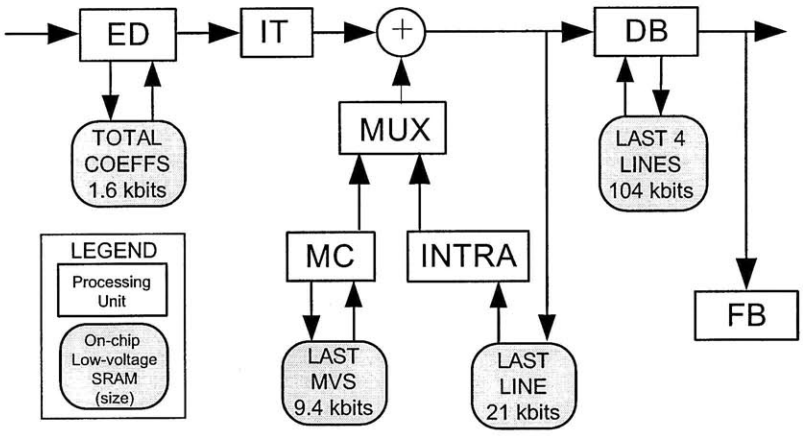


Figure 2-18: On-chip caches to reduce off-chip memory bandwidth.

2.7.2 Reducing MC Redundant Reads

The off-chip frame buffer used in the system implementation has a 32-bit data interface. Decoded pixels are written out in columns of 4, so writing out a 4x4 block requires 4 writes to consecutive addresses. When interpolating pixels for motion compensation, a column of 9 pixels is required during each MC cycle. This requires three 32-bit reads from the off-chip frame buffer.

During MC, some of the redundant reads are recognized and avoided. This happens when there is an overlap in the vertical or horizontal direction and the neighboring 4x4 blocks (within the same macroblock) have motion vectors with identical integer components [41]. As discussed in Section 2.4.1, the MC interpolators have a 6-stage pipeline architecture which inherently takes advantage of the horizontal overlap. The reuse of data that overlap in the horizontal direction helps to reduce the cycle count of the MC unit since those pixels do not have to be re-interpolated.

The two MC interpolators are synchronized to take advantage of the vertical overlap. Specifically, any redundant reads in the vertical overlap between rows 0 and 1 and between rows 2 and 3 (in Fig. 2-9b) are avoided. As a result, the total off-chip memory bandwidth is reduced by an additional 19%, as shown in Fig. 2-17.

In future implementations, a more general caching scheme can be used to further reduce redundant reads if it takes into account:

Table 2.5: Memory bandwidth of caches for 720p at 30 fps.

Cache	Size [kb]	Dimensions address x word	I-frame Bandwidth [Mbps]	P-frame Bandwidth [Mbps]
Deblocking (Last 4 lines + Left 4 columns)	104	324x158 (luma)	510	510
		324x158 (chroma)	297	297
Intra prediction (Last line + Left column)	21	324x32 (luma)	61.7	32.4
		162x32 (chroma x2)	57	35.4
Motion Vector	9	80x118	0	25.5
Total Coefficient Count	3	80x40	8.5	7.8
Macroblock Parameters	1	80x7 (luma)	6.1	6.1
		80x7 (chroma)	6.8	6.8
Intra Prediction Mode	1	80x16	3.1	1.7
Total	**138**	**n/a**	**579**	**922**

1. adjacent 4x4 blocks with slightly different motion vectors
2. overlap in read areas between nearby macroblocks on the same macroblock row
3. overlap in read areas between nearby macroblocks on two consecutive macroblock rows

A small cache of 512-Bytes can reduce the read bandwidth by another 33% by taking advantage of the first two types of redundancies in the above list. A larger cache of 32-kBytes is required to address the third redundancy on the list to achieve a read bandwidth reduction of 56% with close to no repeated reads. Finally, a cache on the order of several hundred kBytes can be used to reduce not only the redundant reads, but the majority of all MC reads from the off-chip frame buffer using either a last frame or direct forwarding caching schemes during frame parallel multi-core processing [33].

2.8 Test Setup

A real-time system was implemented in order to verify and characterize the H.264/AVC decoder test chip. Fig. 2-19 shows the key blocks in the system. The off-chip frame buffer was implemented using two 32-bit-wide 4MB SRAMs [56]. Separate SRAMs are used for

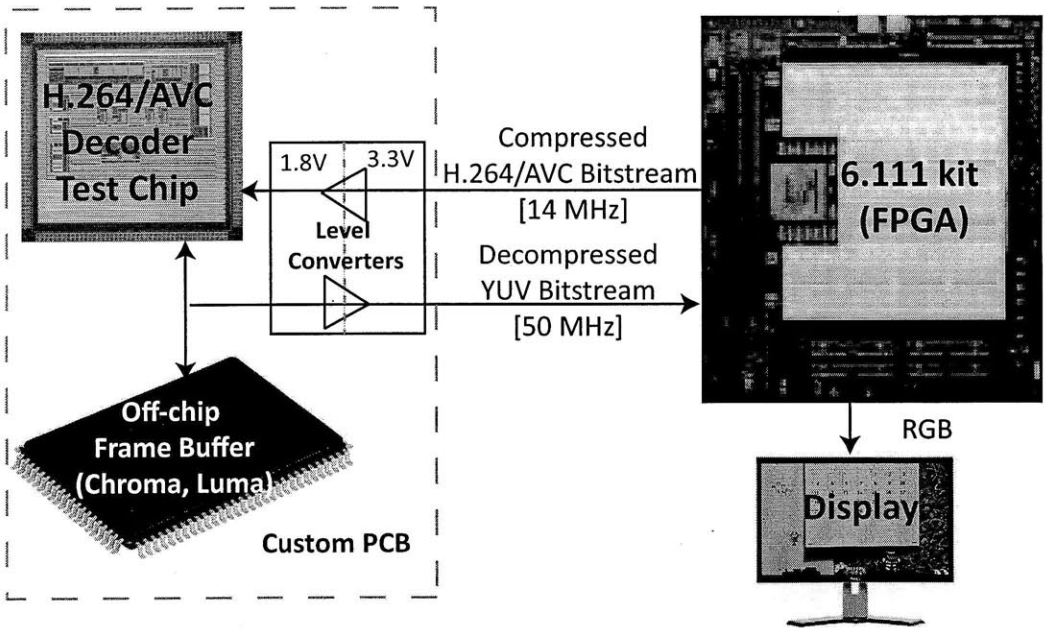


Figure 2-19: Connections in test setup for H.264/AVC decoder.

luma and chroma components. The FPGA and VGA driver on the 6.111 lab kit [57] were used to interface the ASIC to the display. Level converters [58] were required to connect the 1.8V ASIC I/O to the 3.3V FPGA I/O. The test chip, frame buffer and level converter were connected on a custom PCB shown in Fig. 2-20. A photo of the test setup is shown in Fig. 2-21.

The key steps in the system were

1. Send compressed video to the decoder
2. Decode video
3. Read the decompressed video from the test chip
4. Display the video

The encoded H.264/AVC bitstream was stored in a flash memory on the 6.111 lab kit and read into a FIFO on the FPGA. The FPGA then sends the encoded bitstream to the test chip in 8-bit words. During decoding, the test chip simultaneously writes the decoded YUV pixels to the off-chip frame buffer as well as input FIFOs on the FPGA in 32-bits words (4 pixels per word). The Virtex-2 FPGA reorders the YUV pixels, from the vertical 4x1 columns (luma) and 2x2 blocks (chroma) used in MC, to raster scan order and writes

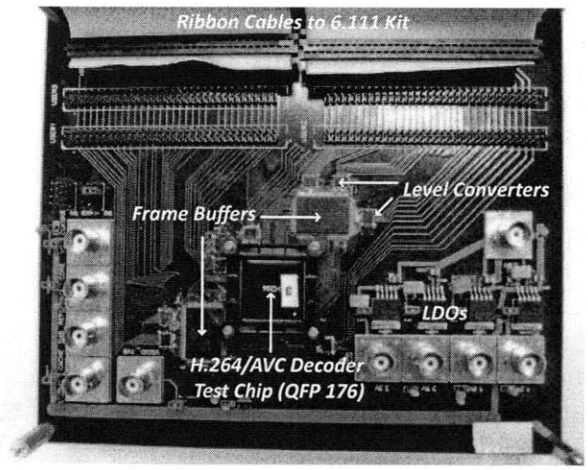


Figure 2-20: Photo of custom PCB designed to connect the H.264/AVC decoder test chip with the rest of the system.

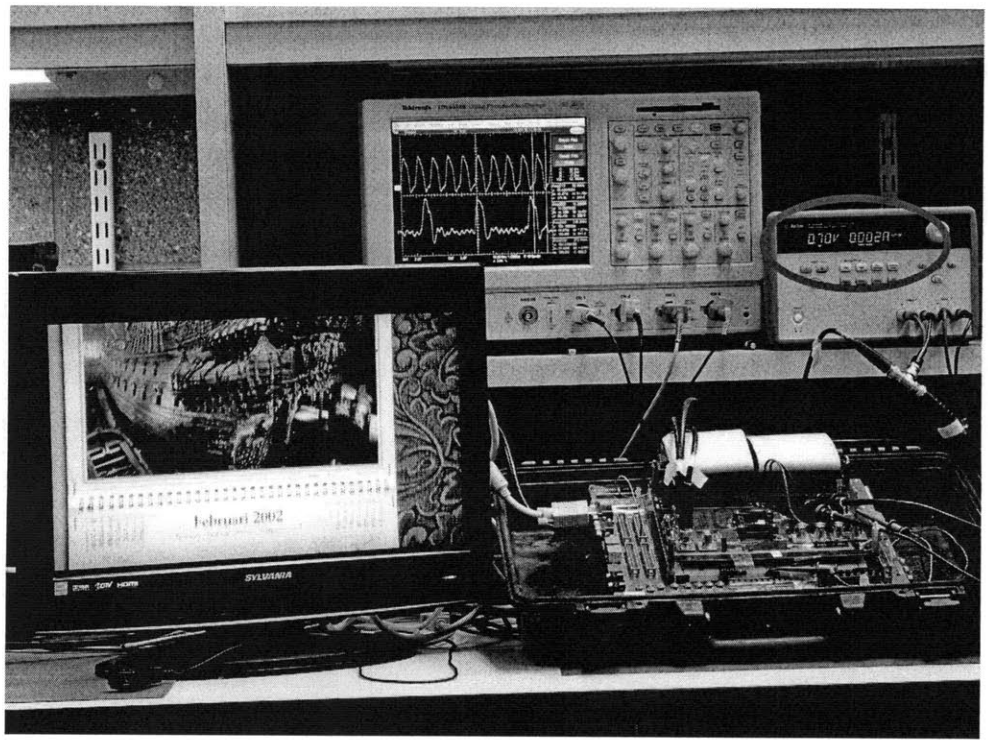


Figure 2-21: Photo of real-time system demo. Voltage and current measurements of the core domain can be seen in the upper right corner.

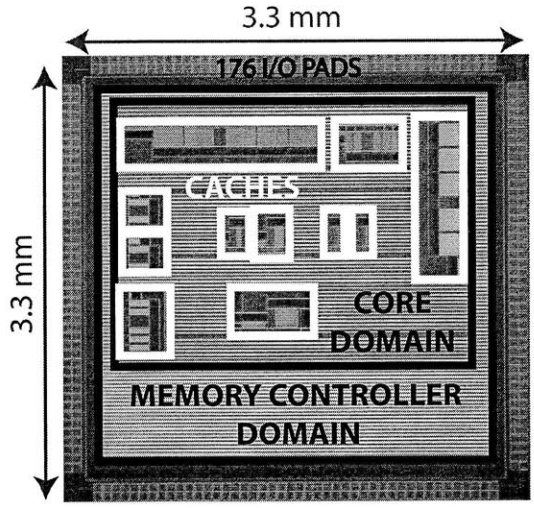


Figure 2-22: Die photo of H.264/AVC video decoder (domains and caches are highlighted).

them into the display buffer. The display buffer is implemented using 32-bit 16MB SRAMs, which allows them to store up to 8 frames. The YUV pixels are read from the display buffer, converted to RGB and sent to the display using the VGA driver. Additional details on the FPGA code can be found in [46].

2.9 Results and Measurements

The H.264/AVC Baseline level 3.2 decoder, shown in Fig. 2-22, was implemented in 65-nm CMOS and the power was measured when performing real-time decoding of several 720p video streams at 30 frames per second (fps) (Table 2.7) [36]. The video streams were encoded with x264 software [59] with a GOP size of 150 (P-frames dominate). Fig. 2-23a shows a comparison of our ASIC with other decoders [38–41, 60]. To obtain the power measurements of the decoder at various performance points, the frame rate of the video sequence was adjusted to achieve the equivalent Mpixels/s of the various resolutions.

At 720p, the decoder also has lower power and frequency relative to D1 of [38]. Fig. 2-23a shows that the decoder can operate down to 0.5 V for QCIF at 15 fps and achieves up to 1080p at 30fps at a core voltage of 0.85 V. The power scales accordingly ranging from 29 μW to 8 mW, a 280x range, to cover the 164x workload range. The power of the I/O pads

Table 2.6: Summary of H.264/AVC decoder chip implementation.

Video Coding Standard	H.264/AVC Baseline level 3.2
Technology	65-nm
Core Area	2.76 mm x 2.76 mm
Logic Gate Count (NAND-2)	314.8k
Package	176-pin QFP
On-Chip Memory (SRAM)	17 kB
Supply Voltage (720p at 30fps)	0.7 V (core) 0.84 V (memory controller) 1.8 V (I/O)
Operating Frequency (720p at 30fps)	14 MHz (core) 50 MHz (memory controller)
Core Power Consumption	1.8 mW

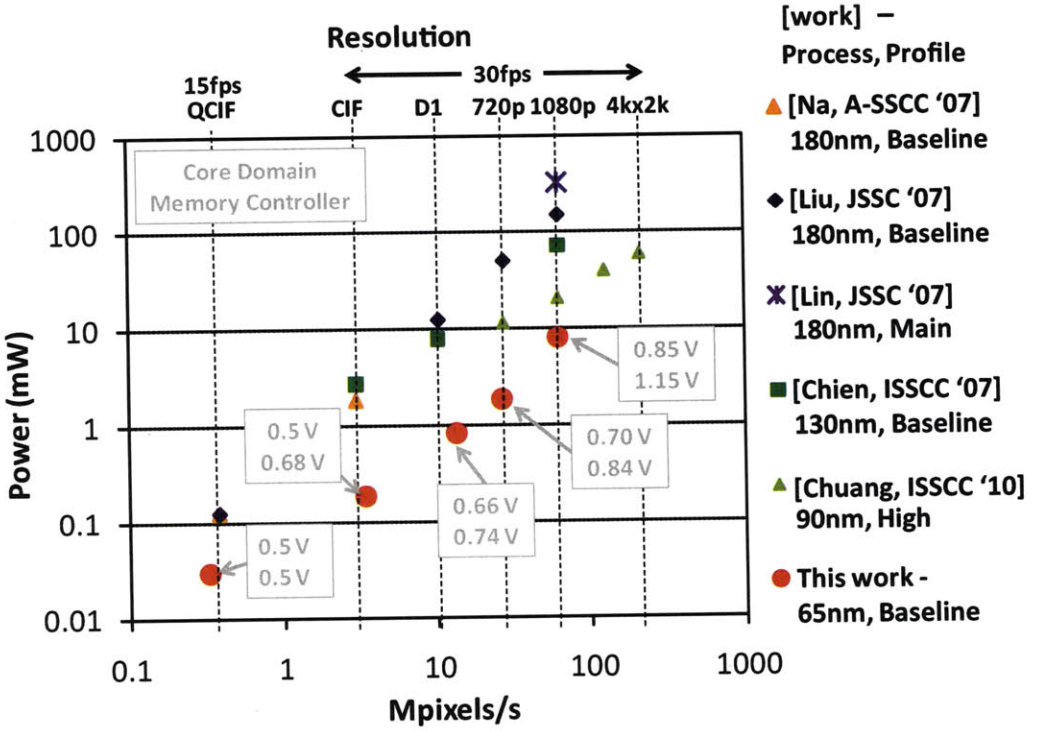
Table 2.7: Measured performance numbers for 720p at 30 fps.

Video	**mobcal**	**shields**	**parkrun**
# of Frames	300	300	144
Bitrate [Mbps]	5.4	7.0	26
Off-chip Bandwidth [Gbps]	1.2	1.1	1.2
Core Frequency [MHz]	14	14	25
Memory Controller Frequency [MHz]	50	50	50
Core Supply [V]	0.7	0.7	0.8
Memory Controller Supply [V]	0.84	0.84	0.84
Power [mW]	**1.8**	**1.8**	**3.2**

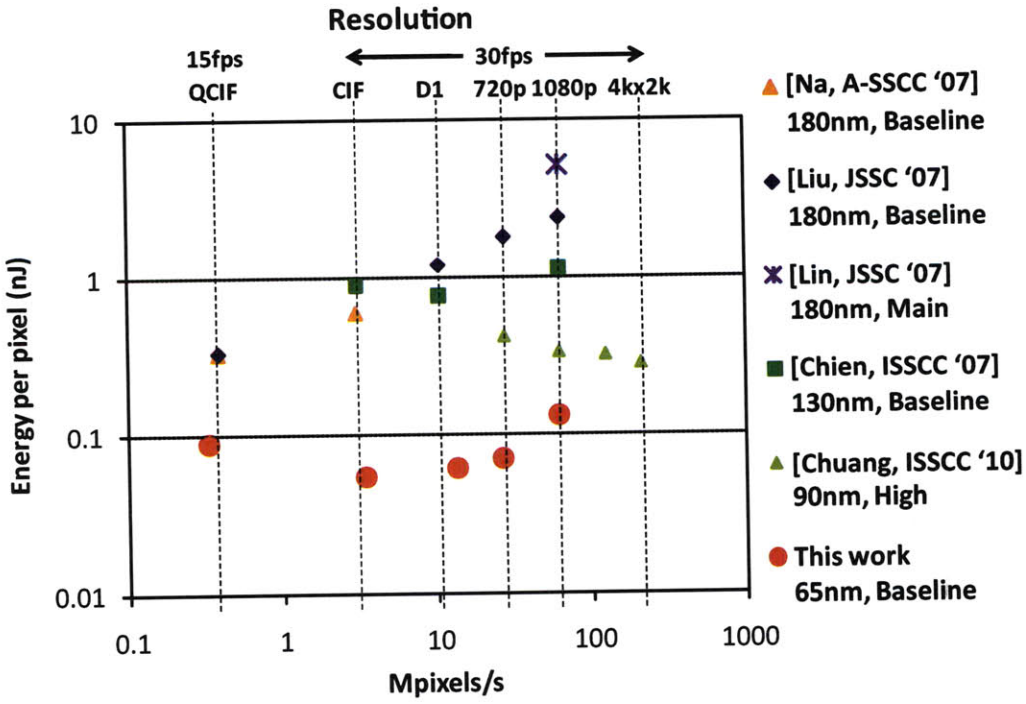
was not included in the measurement comparisons. The reduction in power over the other reported decoders can be attributed to a combination of using the low-power techniques described in this chapter and a more advanced process.

A comparison of the energy per pixel is shown in Fig. 2-23b. The decoder has a minimum energy per pixel of 53 pJ at 0.55 V for CIF decoding; for 720p, the energy per pixel is 69 pJ at 0.7 V. The concurrency in the decoder enables it to process 1.84 pixels per cycle, which is over 80% higher than the throughput of previously published decoders [38–41, 60].

The variation in performance across 15 dies is shown in Fig. 2-24. The majority of the dies operate at 0.7 V for 720p at 30fps decoding.



(a) Power Consumption



(b) Energy per pixel

Figure 2-23: Comparison with other H.264/AVC decoders [38–41, 60].

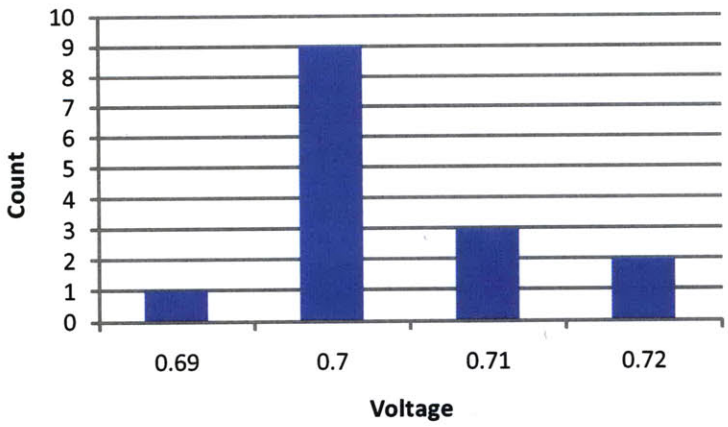


Figure 2-24: Variation of minimum core voltage supply for 720p decoding across test chips.

2.9.1 Power Breakdown and Switching Activity

This section shows the simulated power breakdown during P-frame decoding in the mobcal sequence. The power of P-frames is dominated by MC (42%) and DB (26%), as seen on the left chart of Fig. 2-25.

About 75% of the MC power (32% of total power) is consumed by the MEM read logic, as illustrated by the pie chart on the right of the same figure. The memory controller is the largest power consumer since it runs at a higher voltage than the core domain, its clock tree runs at a higher frequency, and the MC read bandwidth is large (approximately 2 luma pixels are read for every decoded pixel). At 0.7 V, the on-chip caches consume 0.15 mW.

The leakage breakdown across each module is shown in Fig. 2-26. The total leakage of our chip at 0.7 V is 25 μW which is approximately 1% of the 1.8 mW total power for decoding 720p at 30 fps. At 0.5 V, the leakage is 8.6 μW which is approximately 28% of the 29 μW total power for decoding QCIF at 15 fps. 64% of the total leakage power is due to the caches. The leakage of the caches could have been reduced by power gating unused banks during QCIF decoding for additional power savings.

The switching activity measures the probability that the average net will toggle during a given clock cycle. Fig. 2-27 shows the switching activity for each module for P-frame and I-frame with and without clock buffers. The clock buffers account for 5% and 19% of the

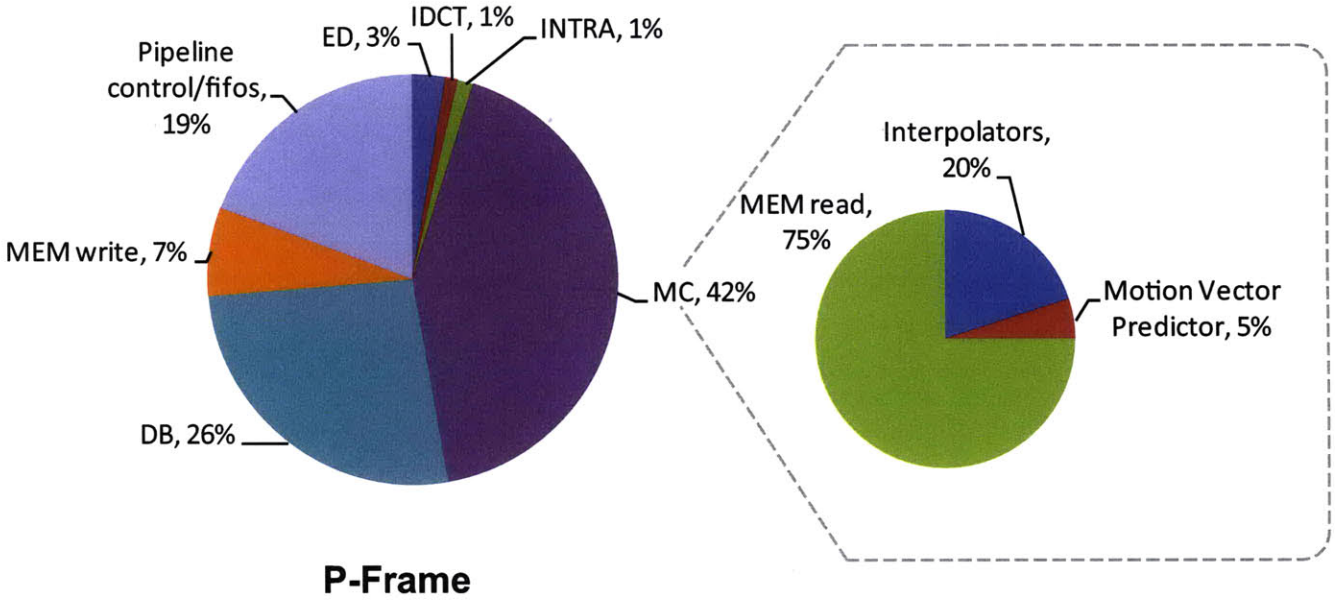


Figure 2-25: Simulated (post-layout) power breakdown during P-frame decoding.

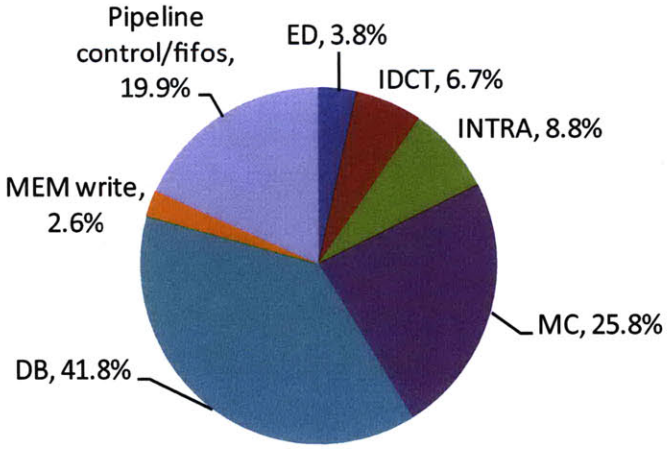
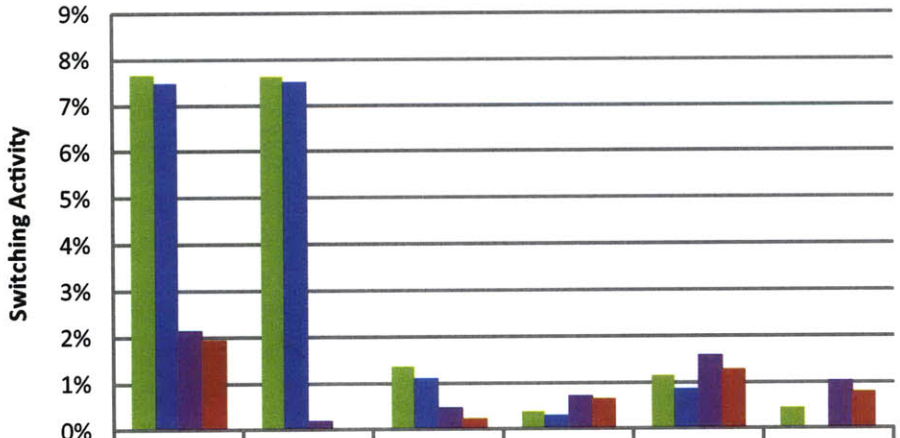


Figure 2-26: Post-layout simulated leakage power breakdown.



	DB	MC	MEM	IT	ED	INTRA
P-frame	7.68%	7.62%	1.35%	0.37%	1.13%	0.44%
P-frame (no clocks)	7.49%	7.53%	1.12%	0.31%	0.86%	0.04%
I-frame	2.13%	2.10%	0.48%	0.73%	1.59%	1.05%
I-frame (no clocks)	1.94%	0.00%	0.25%	0.66%	1.29%	0.80%

Figure 2-27: The switching activity of each module in the decoder, simulated with the mobcal sequence. Note: these switching activity numbers are for logic only; the SRAM caches are not included.

switching in the P-frame and I-frame respectively. These values were obtained through gate level simulations of the mobcal sequence over a time interval of 71,532 ns (5,946 MEM cycles and 991 CORE cycles) for the P-frame and 79,040 ns (3,293 MEM cycles and 4,960 CORE cycles) for the I-frame. The total toggle counts (number of 0 to 1 transitions) for these time intervals were obtained with PrimePower for each module, and then divided by the number of nets in the module along with the number of clock cycles within the time interval. The toggle counts do not include the switching in the on-chip SRAM caches. Ignoring the clock buffers, there is no switching activity in MC during an I-frame which reflects the fact that only spatial prediction is used in I-frames. Accordingly, the switching activity in INTRA is higher for I-frames than P-frames. Note that the switching activity in IT is less for P-frames than I-frames, which is consistent with the fact that P-frames typically have less residual and consequently fewer coefficients to process. The low switching activity can be attributed to the idle cycles where there is no switching, only 0 to 1 transitions are counted towards switching activity, and data dependencies that cause only a subset of nets to toggle.

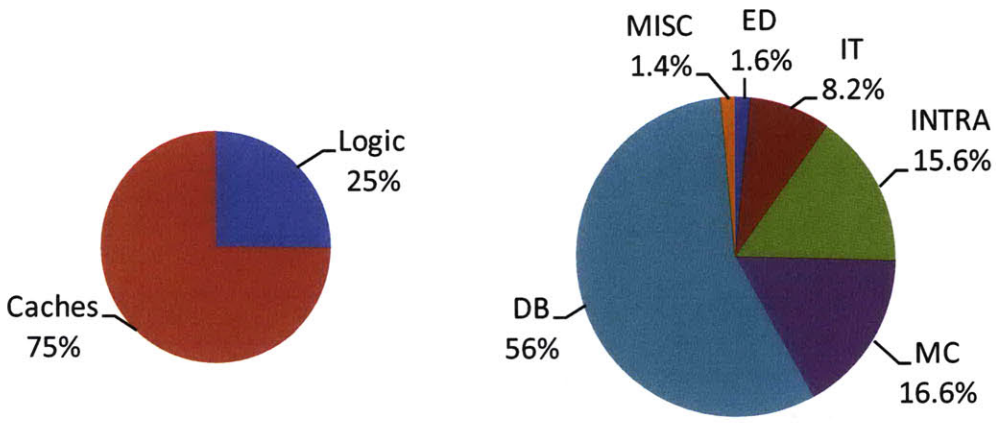


Figure 2-28: Post-layout area breakdown (includes logic and memory).

2.9.2 Area Breakdown

Fig. 2-28 shows the post-layout area breakdown by decoder unit. The pre-layout logic gate count from synthesis along with area of the memory in each decoder unit is reported in Table 2.8. The area is dominated by DB due the SRAM caches which occupy 91% of the DB area.

The cost of parallelism is primarily an increase in area. The parallelism in the MC and DB units described in Section 2.4 increases the area by about 12%. When compared to the entire decoder area (including on-chip memory) the area overhead is less than 3%.

2.9.3 System Level Issues

It is important to consider the impact of this work at the system level. As voltage scaling techniques can reduce the decoder power below 10 mW for HD decoding, the system power is then dominated by the off-chip frame buffer memory. In [40], the memory power using an off-the-shelf DRAM is on the order of 30 mW for QCIF at 15fps which would scale to *hundreds* of milliwatts for high definition. However, new low power DRAMs such as [61], can deliver 51.2 Gbps at 39 mW. For 720p decoding, the required bandwidth is 1.25 Gbps after memory optimizations in Section 2.7, which corresponds to a frame buffer power of 1 mW (a linear estimate from [61]). Furthermore, off-chip interconnect power can be reduced

Table 2.8: Logic gate count (pre-layout) and memory area for each decoder unit. Note that the area of each unit also includes the pipeline FIFO control for the unit. MISC includes the top level control, pipeline FIFOs, slice/NAL header parsing logic, and adders for reconstruction.

Decoder Unit	**Gate Count [kgates (2-input NAND)]**	**SRAM area** $[mm^2]$
ED	23.7	0
INTRA	25.2	0.35
MC	102.5	0.17
IT	41.3	0.07
DB	78.3	1.01
MEM	6.5	0
MISC	37.4	0
Total	**314.8**	**1.6**

by using embedded DRAM or system in package (i.e. stacking the DRAM die on top of the decoder die within a package) [62]. Alternatively, the entire frame buffer can be cached on-chip in low voltage SRAM for low power consumption, as proposed in [33]; however, this would result in significant area cost.

The display typically consumes around a third of the power on a mobile platform [63]. Upcoming technologies such as organic light-emitting device (OLED) and quantum dot-organic light-emitting device (QD-OLED) can significantly reduce the display power. OLED displays do not require a backlight, which accounts for a considerable amount of the liquid crystal display (LCD) power consumption [64].

2.10 Summary and Conclusions

A full video decoder system was implemented that demonstrates high definition real-time decoding while operating at 0.7 V and consuming 1.8 mW. Several techniques, summarized in Table 2.9, were leveraged to make this low power decoder possible. The decoder processing units were pipelined and isolated by FIFOs to increase concurrency. Luma and chroma components were mostly processed in parallel. The MC interpolators and DB fil-

Table 2.9: Summary of Low Power Techniques

Section(s)	Technique(s)	Impact
2.3, 2.4	Increased concurrency with pipelining and parallelism	Enable Low Voltage (0.7 V) and Frequency (14 MHz)
2.5	Multiple Voltage/ Frequency Domains	25% Power Reduction on P-frame
2.6	Frame level DVFS	25% Power Reduction
2.7	Memory Optimization	Reduce Bandwidth by 40%

ters were replicated for increased throughput. The decoder was partitioned into multiple voltage/frequency domains to enable lower voltage/frequency operation for some of the processing blocks. The wide operating voltage range of the decoder allowed for effective use of DVFS for additional power reduction. Finally, voltage-scalable on-chip caches helped reduce both on-chip and off-chip memory power. The decoder is highly scalable and can tradeoff up to 164x in performance for a 280x reduction in power.

In this chapter, we demonstrated that parallel architectures combined with voltage scaling is effective in reducing the power of a H.264/AVC Baseline profile decoder. It was also highlighted that the entropy decoding (ED) unit is inherently serial, and thus it is difficult to reduce its cycle count. Fortunately, the ED cycle count was not much larger than that of the other decoder units, as shown in Table 2.1. In the Baseline profile of H.264/AVC, the ED was done using CAVLC. For increased coding efficiency, the High profile of H.264/AVC includes CABAC for entropy coding. Unfortunately, CABAC has much stronger serial dependencies than CAVLC, which consequently leads to much higher cycle counts than the other units. This challenge will be addressed in the remaining chapters of the thesis.

Chapter 3

Massively Parallel CABAC Algorithm

Low power and high frame rate/resolution requirements for future video coding applications make the need for parallelism in the video ever more important [65]. The CABAC entropy coding engine has been identified as a key bottleneck in the H.264/AVC video decoder [66]. Parallelism is difficult to achieve with the existing H.264/AVC CABAC due to its inherent serial nature.

This chapter describes the development of a new CABAC algorithm that demonstrates how enabling parallelism in algorithms can lead to increased throughput with minimal coding efficiency cost. This new algorithm is not H.264/AVC standard compliant and is intended for the next generation video coding standard 'H.265'. It should be noted that the serial dependencies are more severe at the decoder, and thus CABAC *decoding* will be the focus of this work.

3.1 Overview of CABAC

Arithmetic coding is a form of entropy coding found in many compression algorithms due to its high coding efficiency. For instance, in H.264/AVC, the CABAC provides a 9-14% improvement over the Huffman-based CAVLC for standard definition (720x480 and 720x576) sequences [67]. For HD sequences, simulations show an average improvement of 16% (see

Section B.1). Arithmetic coding is used for a wide variety of applications, and is found in standards such as H.264/AVC, H.263 and China AVS for video; JPEG-2000 and JPEG-LS for image; and MPEG-4 SNHC for 3D animation.

3.1.1 Entropy Coding

Entropy coding is a form of lossless compression that attempts to represent data with a number of bits equal to the entropy of the data, a bound set by Shannon's source coding theorem. It is used at the last stage of video encoding (and first stage of video decoding), after the video has been reduced to a series of syntax elements (e.g. motion vectors, prediction modes, coefficients, etc.). Arithmetic coding performs better than Huffman coding, as it can compress closer to the entropy of a sequence by effectively mapping the syntax elements to codewords with non-integer number of bits; this is important when probabilities are greater than 0.5 and the entropy is a fraction of a bit. In fact, arithmetic coding can potentially achieve near optimal compression efficiency [68].

The CABAC involves three main functions: binarization, arithmetic coding, probability (context) modeling and estimation. Fig. 3-1 illustrates the connections between these functions in a block diagram of the CABAC encoder. Table 3.1 summarizes CABAC terms that are defined in this section.

3.1.2 Binarization

During binarization, any non-binary valued syntax elements are mapped to binary symbols referred to as *bins*. There are several forms of binarization used in CABAC: unary, truncated unary, exp-golomb, and fixed length. The type of binarization to use for each syntax element depends on its statistics and is dictated by the H.264/AVC standard [26]. This mapping is done in a similar fashion as variable-length-coding. These bins, as well as any binary valued syntax elements, are fed to the arithmetic coding engine for additional compression.

Note that CAVLC is serial at the syntax element level and the ED unit can typically process a syntax element per cycle. On the other hand, CABAC is serial on the bin level

Table 3.1: Summary of CABAC terms.

syntax element	Describes how to decode a compressed macroblock (e.g. prediction mode, motion vector, coefficients, etc.)
bins (binary symbols)	Syntax elements are mapped to bins that are fed to the arithmetic coding engine for compression.
context	Probability model of bins. Bins can have a different context depending on the syntax element, prediction, etc. Over 400 contexts are used in H.264/AVC. The context may be updated after every bin is encoded or decoded for to achieve an accurate probability estimate.
range	Size of current interval that is divided into subintervals based on the probability of the bins. It is updated after every bin is encoded or decoded and requires renormalization to prevent underflow.
offset	A binary fraction described by the encoded bits received at the decoder. A bin is decoded by comparing the offset to the subintervals. It requires renormalization to prevent underflow.
regular coding	Arithmetic coding using contexts (estimated probabilities).
bypass coding	Arithmetic coding using a uniform distribution. It has lower coding efficiency than regular coding, but also lower complexity since interval division and renormalization only requires a single left shift.
terminate coding	Arithmetic coding using highly skewed probability. It is only performed on last bin in a macroblock. It is used to flush range bits at the end of a slice.

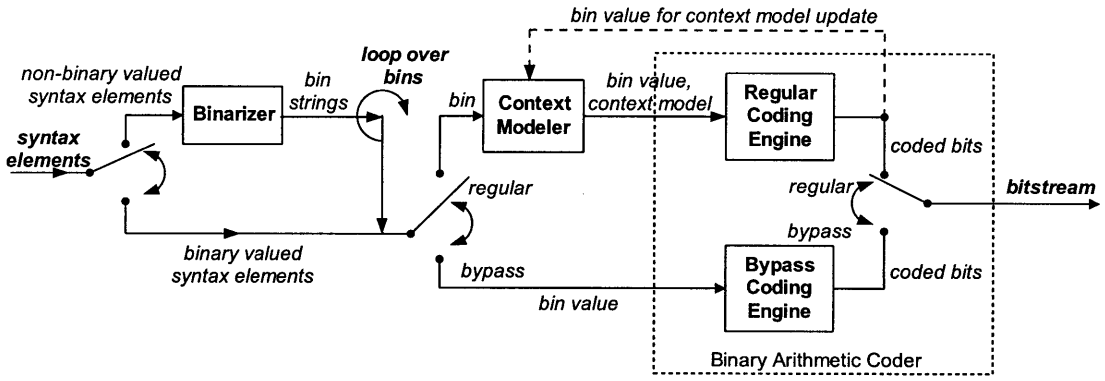


Figure 3-1: Block diagram of CABAC encoder [67]. Non-binary syntax elements pass through the binarizer to be mapped to binary symbols (bins). The majority of the bins are compressed with two forms of arithmetic coding: bypass and regular. Bypass coding assumes a uniform distribution for bins, while regular coding requires context modeling to estimate the bin distribution.

and the ED unit can typically process a bin per cycle. Since each syntax element can be mapped to a large number of bins (up to 33 in the worst case), the CABAC requires many more cycles than the CAVLC. This is one of the main reasons for the limited throughput of the CABAC engine.

3.1.3 Binary Arithmetic Coding

Arithmetic coding is based on recursive interval division. Binary arithmetic coding refers to the case where the alphabet of the symbol is restricted to zero and one (i.e. bins). The sizes of the intervals are determined by multiplying the current interval by the probabilities of the bin. At the encoder, a subinterval is selected based on the value of the bin. The range and lower bound of the interval are updated after every selection. At the decoder, the value of the bin depends on which subinterval the offset is located. The range and lower bound of the current interval have limited bit-precision, so renormalization is required whenever the range falls below a certain value to prevent underflow. There are two main forms of arithmetic coding used in the CABAC engine: regular and bypass. Regular uses the probabilities of the bin to divide the interval, while bypass assumes that the bin has uniform distribution.

Bypass results in simpler encoding and decoding than regular, since the interval division and renormalization is implemented with a single left shift; however this comes at a cost of poorer coding efficiency. A third form called terminate assumes a very skewed probability, and is primarily used at the end of every macroblock to check if it is the last macroblock in a slice.

The CABAC engine used in H.264/AVC leverages a modulo coder (M coder) to calculate the subinterval based on the product of the current range and the probability of the symbol. The M coder involves using a look up table (LUT) rather than a true multiplier to reduce implementation complexity [69].

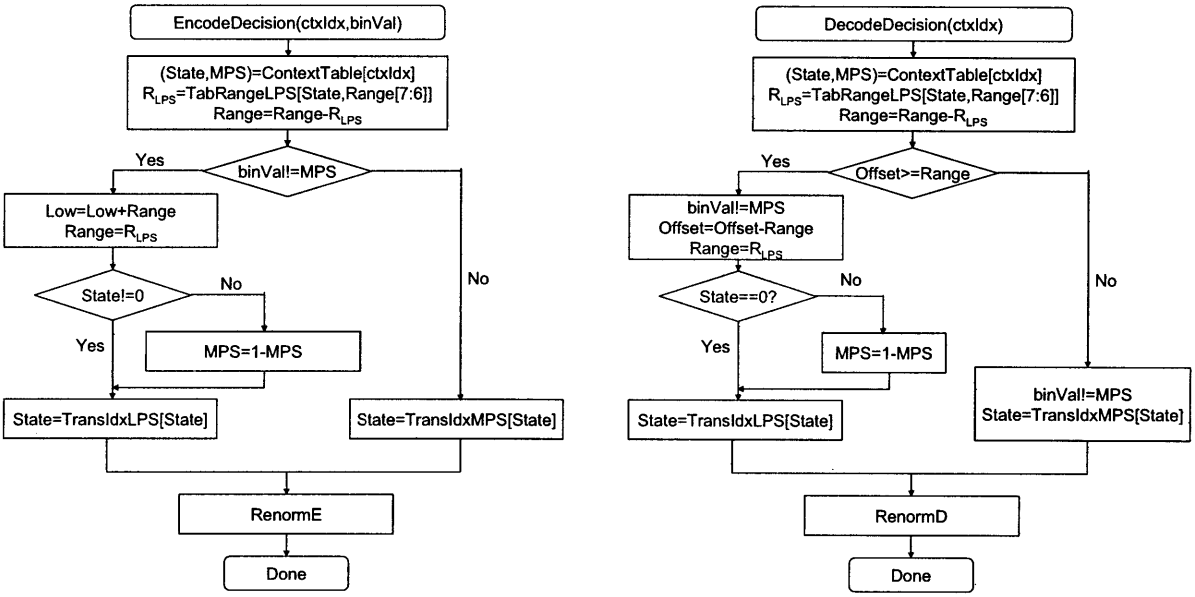
The flowchart of the arithmetic coding engine is shown in Fig. 3-2. The arithmetic coding engine typically contains the critical path in H.264/AVC CABAC. Section 4.2.2 will discuss various optimizations that can be used to reduce the delay of this critical path.

3.1.4 Probability (Context) Modeling and Estimation

In order to achieve optimal compression efficiency, the 'correct' probabilities must be used to code each bin. For High profile in H.264/AVC, 468 different probability models are required to achieve the significant coding gains over CAVLC. All bins of the same type (i.e. with the same probability distribution and characteristics) are grouped together and use the same model known as a context. Accordingly, the context of a bin dictates the probability with which it is coded.

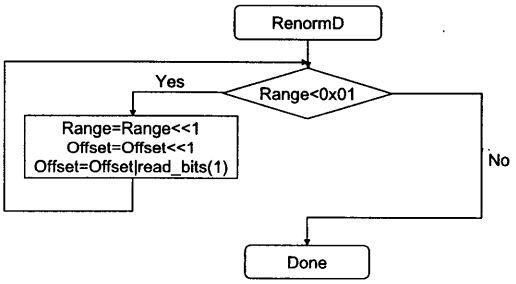
For each bin, the context is selected based on:

- the syntax element that bin belongs to, since syntax elements have different distributions
- the bin's position within the syntax element (e.g. the least significant bit (LSB) and most significant bit (MSB) are typically coded with different contexts)
- properties of the macroblock that the bin belongs to, since I, P and B macroblocks have different statistics
- the value of the syntax elements of the macroblocks (or blocks) located to the top and



(a) Encoding

(b) Decoding



(c) Renormalization for CABAC decoding. Encoding has a similar structure.

Figure 3-2: Data flow of the arithmetic coding engine.

left, since distributions in neighbouring macroblocks are correlated

For instance, all bins that are located at the MSB of a vertical motion vector difference (mvd) syntax element, and have neighbors with large mvds, use the same context.

Consequently, context switching can occur at every bin. A context selection finite state machine (FSM) is required to determine which context to use for each bin. The probabilities used for each context must be accurately modeled; this process of determining the probability of a bin is called source modeling. Since the bins have non-stationary distributions, the probabilities are continuously updated by the context modeler making the engine adaptive. The context can take on one of 64 possible probabilities (states). Updating the probabilities involve a state transition. For each context, the CABAC engine must store the state as well as the value of the most probable symbol (MPS). An example of the CABAC encoding/decoding process can be found in the next section.

3.1.5 CABAC Example

We will now walk through an example of encoding and decoding with CABAC.

Encoding

To encode the syntax element `coeff_abs_level_minus1` (coefficient level) with the value 3:

Step 1 - Binarize the value using truncated unary[1] from 2 to '110'. Let bin_0=1, bin_1=1 and bin_2=0.

Step 2 - Look up the probability state for each context (Table 3.2). The first bin uses a different context (context A) than the second and third bins (context B). Pr_{ctxA} is applied to bin_0 and Pr_{ctxB} is applied to bin_1 and bin_2. For simplicity, this example omits the probability state update.

Step 3 - Arithmetic encoding is performed on the binary symbols (bins) (Fig. 3-3), which generates the encoded bits. During this process, the encoder keeps track of the range

[1]For unsigned integer value x, unary coding maps to a string of x ones plus a terminating zero (e.g. x=5 maps to 111110). Truncated unary limits $x \leq S$ and when $x = S$ the terminating zero is neglected (e.g. x=5 and S=5 maps to 11111).

Bin	Probability
0	$Pr_{ctxA}(0) = 0.3$
1	$Pr_{ctxA}(1) = (1 - Pr_{ctxA}(0)) = 0.7$

(a) Context A. MPS is '1'

Bin	Probability
0	$Pr_{ctxB}(0) = 0.6$
1	$Pr_{ctxB}(1) = (1 - Pr_{ctxB}(0)) = 0.4$

(b) Context B. MPS is '0'

Table 3.2: Probability tables for binary arithmetic coding for context A and B.

and lower bound. Note that the larger (i.e. MPS) interval is always on the lower half of the range. Renormalization can occur after encoding each bin if the range falls below 0.5. For simplicity, we have omitted this step in our numerical calculations. The effect of renormalization is illustrated by *zooming in* on the selected interval in Fig. 3-3.

The range is initialized to 1, and low to 0. First, to encode bin_0, the range is divided into two intervals based on Pr_{ctxA}. In practice, this is done using an LUT. The intervals are: 0 to 0.7 for '1', and 0.7 to 1 for '0'. bin_0=1, so range is updated to 0.7 and low remains unchanged at 0.

Next, to encode bin_1, the range is divided into two intervals based on Pr_{ctxB}. The intervals are: 0 to 0.42 for '0', and 0.42 to 0.7 for '1'. bin_1=1, so range and low are updated to 0.28 and 0.42, respectively.

Finally, to encode bin_2, the range is divided into two intervals based on Pr_{ctxB}. The intervals are: 0.42 to 0.588 for '0', and 0.588 to 0.7 for '1'. bin_2=0, so range and low are updated to 0.168 and 0.42, respectively.

Any value between 0.42 and 0.588 can be used for the encoded bits to represent the bins '110'. For the decoder to make the correct decision when comparing to 0.588, the encoded bits must contain at least 2 digits of the binary fraction.[2] Thus, 0.5 is transmitted as .10. Note that we only require 2 bits to represent 3 bins.

Decoding

The decoder receives the encoded bits '10'. This binary fraction maps to 0.5 in decimal form.

[2]Note: 0.588 in the encoder is equivalent to 0.168 in the decoder, which can be seen by comparing Fig. 3-3 and Fig. 3-4. The offset is compared to 0.168.

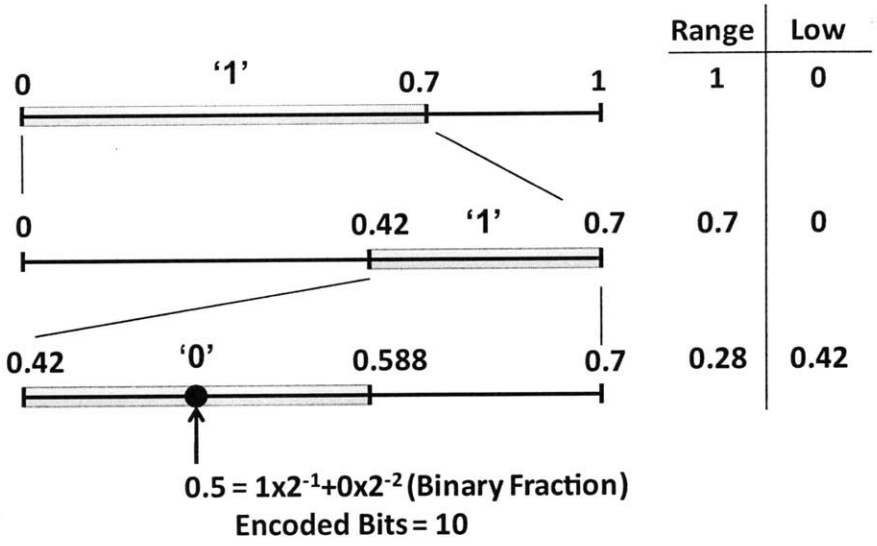


Figure 3-3: Example of arithmetic coding of sequence '110'.

Step 1 - The context selection FSM determines that it should decode `coeff_abs_level_minus1`. This information is useful in two ways. First, `coeff_abs_level_minus1` uses truncated unary for binarization, so a zero indicates that all bins in the syntax element have been decoded. Second, it indicates which context models should be used to decode the next bin.

Step 2 - Look up the probability state for each context. In this example, the probabilities are Pr_{ctxA} and Pr_{ctxB}. Their values are given in Table 3.2.

Step 3 - Arithmetic decoding is performed on the encoded bits (Fig. 3-4) to resolve the binary symbols (bins). At the same time, the decoder keeps track of the range and offset. Note that the larger interval is always on the lower half of the range. In the decoder, the lower bound is always fixed at 0; thus whenever the smaller upper interval is selected (i.e. least probable symbol (LPS)), the width of the larger interval must be subtracted from range and offset. Renormalization can occur after decoding each bin if the range falls below 0.5. For simplicity, we have omitted this step in our numerical calculations. The effect of renormalization is illustrated by *zooming in* on the selected interval in Fig. 3-4.

The range is initialized to 1, and the offset is set to 0.5 (the value of the encoded bits). To decode bin_0, the range is divided into two intervals based on Pr_{ctxA}. The intervals are:

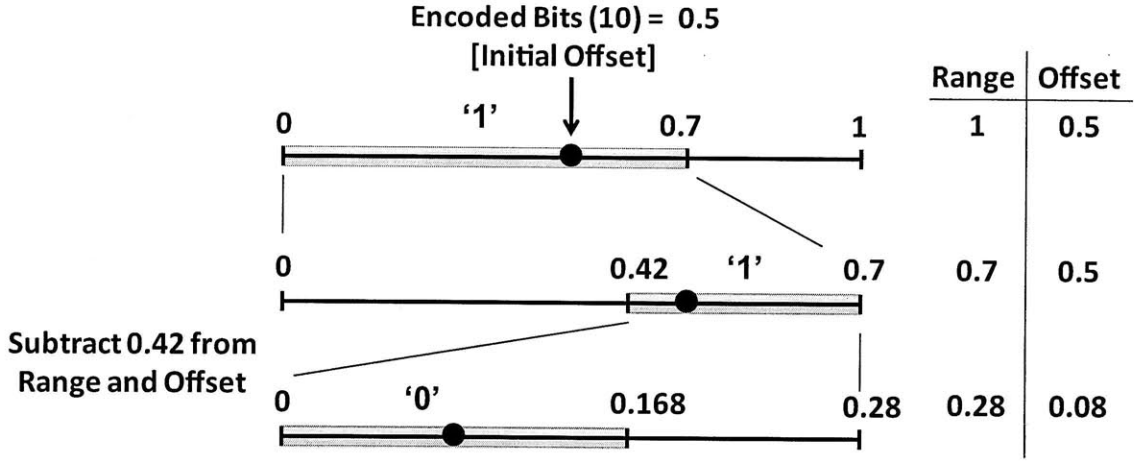


Figure 3-4: Example of arithmetic decoding of sequence '110'.

0 to 0.7 for '1', and 0.7 to 1 for '0'. The offset is *less* than 0.7, so bin_0=1, and range is updated to 0.7 while offset remains unchanged. Since bin_0 is *not* zero, we continue decoding the another bin for the syntax element.

To decode bin_1, the range is divided into two intervals based on Pr_{ctxB}. The intervals are: 0 to 0.42 for '0', and 0.42 to 0.7 for '1'.[3] The offset is *greater* than 0.42, so bin_1=1, and 0.42 is subtracted from range and offset. Range and offset are updated to 0.28 and 0.08, respectively. Since bin_1 is *not* zero, we continue decoding the another bin.

To decode bin_2, the range is divided into two intervals based on Pr_{ctxB}. The intervals are: 0 to 0.168 for '0', and 0.168 to 0.28 for '1'. The offset is *less* than 0.168, so bin_2=0, and range is updated to 0.168 while offset remains unchanged. As bin_2 is a zero, we have reached the end of the syntax element.

Step 4 - De-binarization for truncated unary works by counting the number of decoded bins. We decoded three bins ('110') for this syntax element, so the value of `coeff_abs_level_minus1` is 2.

[3] 0.42=range×probability=0.7×0.6

Table 3.3: Peak bin-rate requirements for real-time decoding of worst case frame at various high definition levels.

Level	Max Frame Rate [fps]	Max Bins per picture [Mbins]	Max Bit-rate [Mbits/s]	Peak Bin-rate [Mbins/s]
4.0	30	9.2	25	275
5.1	26.7	17.6	300	2107

3.1.6 Performance Requirements

It is important to understand the performance requirements for real-timing decoding applications such as video conferencing. To achieve real-time low-delay decoding, the processing deadline is dictated by the time required to decode each frame to achieve a certain fps performance.

It is important to note that the interval division and context modeling are tied to the bins, not the bits. Thus, the number of cycles required to process a given sequence is proportional to the number of bins rather than bits. While a highly compressed sequence may have a low bit-rate, its bin-rate may be quite high since each bit can represent multiple bins resulting in a high cycle count. Accordingly, the performance of CABAC is dictated by the bin-rate rather than bit-rate.

Table 3.3 shows the peak bin-rate requirements for a frame to be decoded instantaneously based on the specifications of the H.264/AVC standard [26]. They are calculated by multiplying the maximum number of bins per frame by the frame rate for the largest frame size. Details on the calculations can be found in Appendix A.1. For Level 5.1, the highest level defined for H.264/AVC, the peak bin-rate is in the Gbins/s; without concurrency, decoding 1 bin/cycle requires multi-GHz frequencies, which leads to high power consumption and is difficult to achieve even in an ASIC. Existing H.264/AVC CABAC hardware implementations such as [60] are in the 200 MHz range; the maximum frequency is limited by the critical path, and thus parallelism is necessary to meet next generation performance requirements.

3.2 Related Work

There are several methods of either reducing the peak performance requirement or increasing the performance of CABAC; however, they come at the cost of decreased coding efficiency, increased power consumption and/or increased delay. This section will discuss the approaches that are both standard compliant and non-compliant.

3.2.1 Frame Workload Averaging (Buffering)

Buffering allows for averaging which can be used to reduce the peak performance requirements at the cost of increased latency. Specifically, averaging the workload of several frames (i.e. on the order of one second as used in the hypothetical reference decoder (HRD) for rate control [70] defined in H.264/AVC) can decrease the peak bin-rate requirements to be within the range of the maximum bit-rate. Table 3.3 shows that the bit-rate requirement is much lower than the peak bin-rate requirement. However, buffering results in increased delay and storage costs. For low-delay applications such as video conferencing, where the maximum end-to-end delay (including network) is around 200-ms, an additional delay of several frames may not be tolerated. The buffer also has implications on the memory bandwidth. The location of the buffer within the CABAC engine can impact the required memory size and bandwidth [42].

3.2.2 Bin Parallel Processing

Due to the strong data dependencies from bin to bin, speculative computation is required for bin parallelism. This approach has been proposed in papers for both H.264/AVC compliant [71,72] and non-compliant [73] high throughput CABAC solutions. During our initial investigation on parallelizing CABAC, we also developed a bin parallel algorithm for a high throughput CABAC for the next generation standard [31,74]; this algorithm is outlined in detail in Appendix A.3. Speculation requires additional computations which may increase power consumption. Furthermore, the critical path delay increases with each additional bin,

since all computations cannot be done entirely in parallel (e.g. each bin needs to wait for updated range from the previous bin, which at the very minimum requires a subtraction and a shift; this can increase the critical path by approximately 30%, depending on the architecture). This reduces the overall performance improvement that can be achieved. The reported bin-rates for these approaches are in the low hundreds of Mbins/s. Additional discussion about the architecture of previously published bin parallel approaches can be found in Appendix A.2

3.2.3 Frame and/or Slice Parallel Processing

In H.264/AVC, frames can be broken into slices that can be encoded and decoded completely independently for each other. H.264/AVC slices contain contiguous groups of macroblocks as shown in Fig. 3-5. Accordingly, CABAC parameters such as range, offset and context states are reset every slice and slices can be processed in parallel. Each frame has a minimum of one slice so, at the very least, parallelism can be achieved across several frames. The complete CABAC engine would have to be replicated based on the desired degree of parallelism and additional control would be necessary to synchronize the engines.

Unfortunately, frame parallelism requires additional buffering, as inter-frame prediction prevents several frames from being fully decoded in parallel. The decoded bins would have to be stored before being processed by the rest of the decoder. Since we are dealing with *frames* worth of data, it would require a significant amount of memory and would have to be stored off-chip.

The storage costs could be reduced if there are several slices per frame as the amount of additional storage needed per increasing degree of parallelism is reduced. However, this cannot be guaranteed since the H.264/AVC standard does not have a minimum number of slices per frame constraint. Increasing the number of slices per frame also reduces the coding efficiency since it limits the number of macroblocks that can be used for prediction (i.e. redundancy cannot be leveraged across slices), reduces the training period for the probability estimation, and increases the number of slice headers and start code prefixes. Fig. 3-6 shows

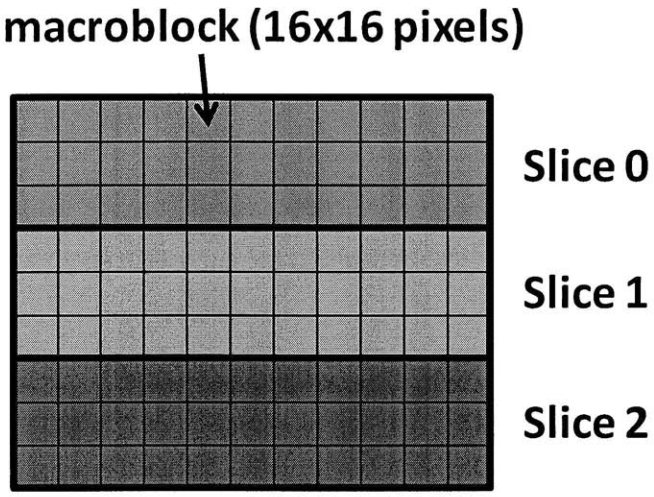


Figure 3-5: Example of three H.264/AVC slices in a frame. H.264/AVC slices are contiguous groups of macroblocks that can be encoded/decoded independently from each other. This enables parallelism, but reduces coding efficiency since redundancy across slices cannot be leveraged for prediction.

how the coding efficiency penalty increases with more H.264/AVC slices per frame.

3.2.4 Entropy Slices

As shown in the previous sections, increasing the performance of the CABAC is challenging when constrained by the H.264/AVC standard. An alternative is to modify the algorithm itself. In recent years, Sharp [75, 76] and MediaTek [77] have proposed new CABAC algorithms for the next generation standard that seek to address this critical problem. These contributions looked at various ways of using a new approach called entropy slices to increase parallel processing for CABAC. Entropy slices are similar to H.264/AVC slices in that contiguous macroblocks are allocated to different slices [75, 76] as shown in Fig. 3-5. However, unlike H.264/AVC slices, which are completely independent of one another, some dependency is allowed for entropy slices. While entropy slices do not share information for entropy (de)coding (to enable parallel processing), motion vector reconstruction and intra prediction are allowed across entropy slices, resulting in better coding efficiency than H.264/AVC slices (Fig. 3-6). However, entropy slices still suffer coding efficiency penalty versus H.264/AVC with single slice per frame. A significant portion of the coding penalty can be attributed to

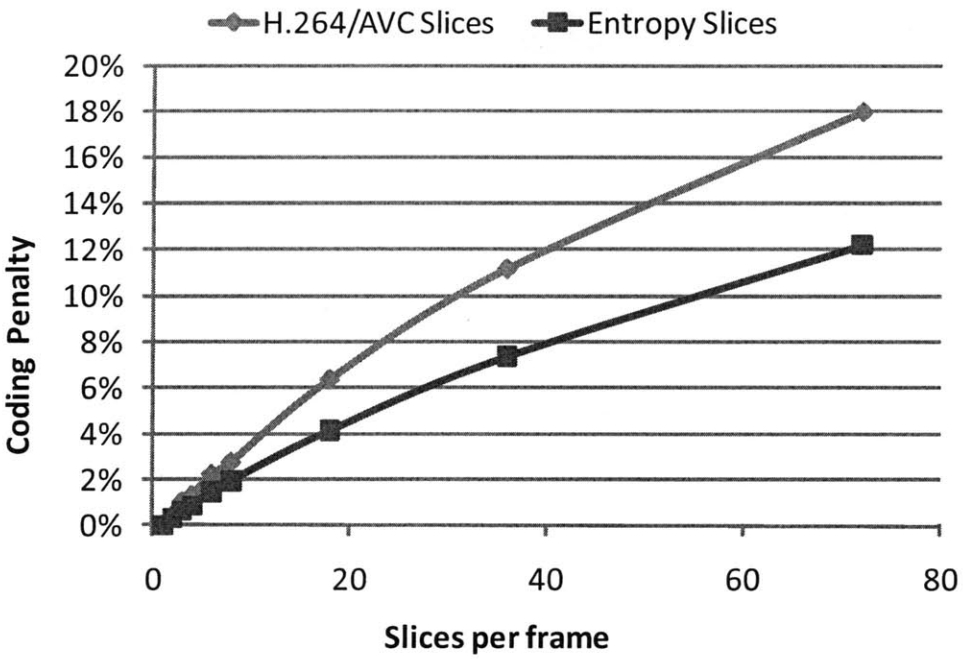


Figure 3-6: Coding penalty versus slices per frame. Sequence bigships, QP=27, under common conditions [34].

the reduction in context training.

As described in Section 3.1.4, one of the features that gives CABAC its high coding efficiency is that the contexts are adaptive. While encoding/decoding, the contexts undergo training to achieve an accurate estimate of the syntax element probabilities. A better estimate of the probabilities results in better coding efficiency. A drawback of breaking up a picture into several entropy slices is that there are fewer macroblocks, and consequently fewer syntax elements, per slice. Since the entropy engine is reset every entropy slice, the context undergoes less training and can results in a poorer estimate of the probabilities.

Ordered entropy slices propose processing in macroblocks in zig-zag order within a slice to minimize memory bandwidth costs from syntax element buffering [77]. Furthermore, it allows for context selection dependencies across entropy slices which improves coding efficiency. However, the zig-zag order results in increased latency and does not provide a favorable memory access pattern necessary for effective caching.

3.3 Massively Parallel CABAC (MP-CABAC)

In this section, we propose a parallel algorithm called Massively Parallel CABAC (MP-CABAC) which has an improved tradeoff between coding efficiency and throughput. It can also be easily implemented in hardware and with low area cost. The MP-CABAC leverages a combination of two forms of parallelism. First, it uses syntax element parallelism, presented in Section 3.3.2, by simultaneously processing different syntax element partitions, allowing the context training to be performed across all instances of the elements, thus improving the coding efficiency. Second, macroblock/slice parallelism is achieved by simultaneously processing interleaved entropy slices, presented in Section 3.3.3, with simple synchronization and minimal impact on coding efficiency. Note that the MP-CABAC can also be combined with bin parallelism techniques previously described in Section 3.2.2.

3.3.1 Improving Tradeoffs

The goal of this work is to increase the throughput of the CABAC at minimal cost to coding efficiency and area. Thus, the various parallel CABAC approaches (H.264/AVC Slices, Entropy Slices, Ordered Entropy Slices, MP-CABAC) are evaluated and compared across two important metrics/tradeoffs:

- Coding Efficiency vs. Throughput
- Area Cost vs. Throughput

It should be noted that while throughput is correlated with degree of parallelism, the two are not equal. It depends strongly on the workload balance between the parallel engines. If the workload is not equally distributed, some engines will be idle, and the throughput is reduced. Thus, we chose throughput as the target objective rather than degree of parallelism.

3.3.2 Syntax Element Partitions (SEP)

Syntax element partitions is a different method distributing the bins across parallel entropy engines. To avoid reducing the training, rather than processing various macroblocks/slices in

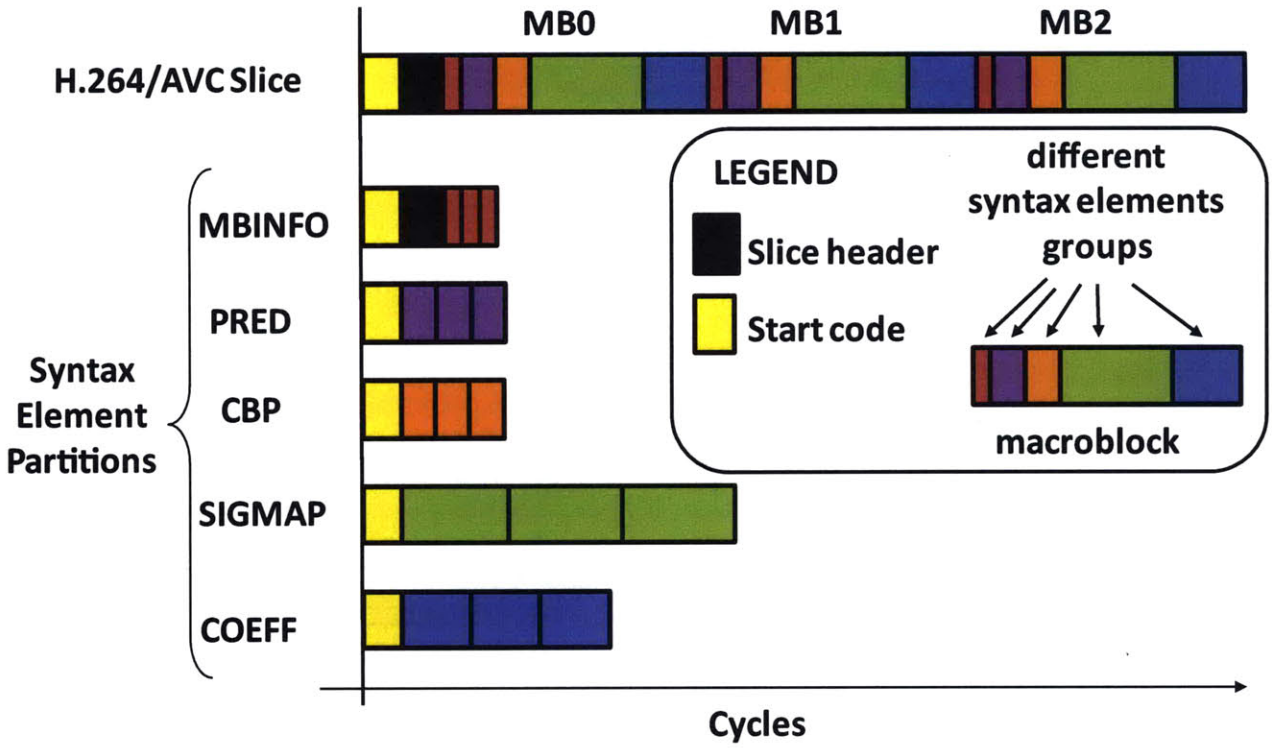


Figure 3-7: Concurrency with syntax element partitioning.

parallel, syntax elements are processed in parallel [32]. In other words, rather than grouping bins by macroblock and placing them in different slices, bins are grouped based on syntax element and placed in different partitions which are then processed in parallel (Fig. 3-7). As a result, each partition contains all the bins of a given syntax element, and the context can then undergo the maximum amount of training (i.e. across all occurrences of the element in the frame) to achieve the best possible probability estimate and eliminate the coding efficiency penalty from reduced training. Table 3.4 shows the five different groups of syntax elements. The syntax elements were assigned to groups based on the bin distribution in order to achieve a balanced workload. Each group of elements can be assigned to a different partition. A start code prefix for demarcation is required at the beginning of each partition.

This syntax element partitions scheme is similar to slice data partitions in the extended profile of H.264/AVC. However, slice data partitions in H.264/AVC is limited to CAVLC and is done primarily for error resilience purposes. Syntax element partitions for CABAC can also benefit in terms of error resilience, however, it is done primarily to increase throughput

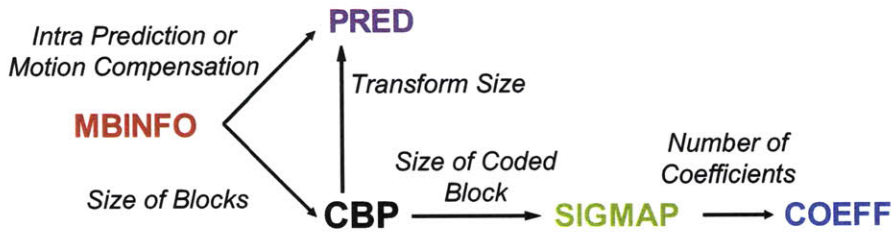


Figure 3-8: Dependencies between syntax element groups.

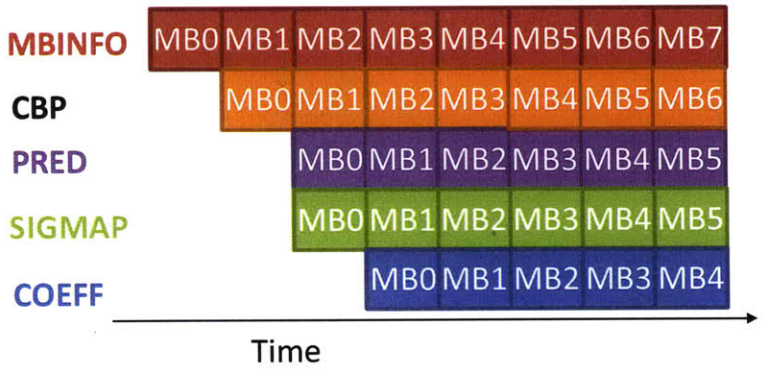


Figure 3-9: Partitions are processed in a pipelined manner such that different macroblocks from each partition are processed in parallel.

and the partitions are assigned accordingly.

Dependencies between each group are shown in Fig. 3-8. For instance, the MBINFO group for a given macroblock (MB0) must be decoded before the PRED group for the same macroblock (MB0). However, the MBINFO group of the next macroblock (MB1) can be decoded in parallel with the PRED group of MB0 as shown in Fig. 3-9. Thus, the processing of each partition must be synchronized. Synchronization can be done using data driven FIFOs between engines, similar to the ones used in [36] between processing units.

Coding Efficiency and Throughput

The syntax element partitions approach was evaluated using JM12.0 reference software provided by the standards body, under common conditions [34]. The coding efficiency and throughput were compared against H.264/AVC slices and entropy slices (Table 3.5). The coding efficiency is measured with the Bjøntegaard ΔBitrate (BD-rate) as described in Appendix B. To account for any workload imbalance, the partition with the largest number

Table 3.4: Syntax Element Groups.

Group	Syntax Element
MBINFO	mb_skip_flag, mb_type, sub_mb_type, mb_field_decoded_flag, end_of_slice_flag
PRED	prev_intra4x4_pred_mode_flag, rem_intra4x4_pred_mode, prev_intra8x8_pred_mode_flag, rem_intra8x8_pred_mode, intra_chroma_pred_mode, ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1
CBP	transform_size_8x8_flag, mb_qp_delta coded_block_pattern, coded_block_flag
SIGMAP	significant_coeff_flag, last_significant_coeff_flag
COEFF	coeff_abs_level_minus1, coeff_sign_flag

Table 3.5: Comparison of various parallel processing techniques. The coding efficiency was computed by evaluating the Bjøntegaard ΔBitrate (BD-rate) [78] against H.264/AVC with single slice per frame. The speed up was computed relative to serial 1 bin/cycle decoding. Results are averaged across bigships, city, crew, night, and shuttle. The area cost was computed based on the increased gate count relative to a serial 1 bin/cycle CABAC.

	H.264/AVC Slices		Entropy Slices		Syntax Element Partitions	
Area Cost	3x		3x		1.7x	
Prediction Structure	BD-rate	speed up	BD-rate	speed up	BD-rate	speed up
Ionly	0.87	2.43	0.25	2.43	0.06	2.60
IPPP	1.44	2.42	0.55	2.44	0.32	2.72
IBBP	1.71	2.46	0.69	2.47	0.37	2.76

of bins in a frame was used to compute the throughput. An average throughput speed up of ~2.7x can be achieved with negligible impact (0.06% to 0.37%) on coding efficiency. To achieve similar throughput requires at least three H.264/AVC or entropy slices per frame which have coding penalty of 0.87% to 1.71% and 0.25% to 0.69% respectively. Thus, syntax element partitions provides 2 to 4x reduction in coding penalty relative to these other approaches.

Area Cost

Implementations for parallel H.264/AVC slices and entropy slices processing require that the entire CABAC be replicated which can lead to significant area cost. An important benefit to syntax element parallelism is that the area cost is quite low since the FSM used for context selection, and the context memory do not need to be replicated. Only the arithmetic coding engine needs to be replicated, which accounts for a small percentage of the total area. Syntax Element Partitions (SEP) FIFOs need to be included to synchronize the partitions.[4] Overall the SEP engine area is approximately 70% larger than the estimated[5] H.264/AVC CABAC area as shown in Fig. 3-10. To achieve the throughput in Table 3.5, H.264/AVC slices and entropy slices require a 3x replication of the CABAC area, whereas syntax element partitions only increase the area by 70%.

Note that the area cost for SEP may be even less than 70% if we account for storage of the last line data. If the last line data is stored in an on-chip cache, then it also needs to be replicated for the H.264/AVC and entropy slices approach which results in significant additional area cost. Alternatively, the last line data can be stored off-chip but this will

[4]The architecture of SEP will be presented in Section 4.3.

[5]The H.264/AVC CABAC was not implemented. Its area was estimated from the synthesis results of the syntax element partitions implementation. We assume that the area of the control and context selection remain the same for both implementations. The areas of the arithmetic decoder and read bitstream control are divided by 5 for the H.264/AVC CABAC. SEP FIFOs area is not included for H.264/AVC CABAC. Finally, we assume that the combinational logic in the context memory is primarily due to the muxes for the read address port. To scale this area, we take the ratio of port-widths of H.264/AVC CABAC and SEP, $log_2307/(log_239 + log_259 + log_237 + log_220 + log_2146)$=0.27, and multiply it by the combinational logic area of the SEP context memory. The non-combinational logic area (registers) remain the same for both - we assume that both have 307 contexts (no interlace).

Table 3.6: Group allocation to partitions.

Mode	MBINFO	PRED	CBP	SIGMAP	COEFF
Low QP	0	0	0	1	2
High QP	0	1	2	2	2

increase the off-chip memory bandwidth. SEP does not require this cache to be replicated, which either reduces area cost or off-chip memory bandwidth.

Adaptive Bin Allocation for Varying Quantization

There is a fixed overhead of approximately 50 bits per partition that can be attributed to additional start code prefix, flushing/termination of the partition, and byte alignment bits. In the previous analysis, each syntax element group was assigned to a different partition. Certain syntax element groups can be allocated to the same partition, such that only three partitions are used instead of five. This reduces the fixed overhead per partition and the number of arithmetic decoding engines (i.e. area cost).

The overall throughput depends on how well the number of bins per partition are balanced. The distribution of the bins per syntax element group changes depending on the quantization parameter (QP) as shown in Fig. 3-11. To maximize throughput for varying QP, the allocation of groups to each partition should be adaptive.

A threshold QP is used to distinguish the two QP modes of bin allocation. Table 3.6 shows which partition (0,1,2) each group is allocated to for a given QP mode. Adaptive QP is only necessary for B and P frames. In I frames, SIGMAP and COEFF tend to dominate regardless of QP, and thus the low QP mode is always used. The QP threshold can be different for each sequence or frame and transmitted in the sequence parameter set (SPS) or picture parameter set (PPS). Smart encoder algorithms can be developed to determine the threshold that would provide balanced partitions for high throughput. For instance, a threshold can be set based on the number of non-zero coefficients.

Fig. 3-12 shows the throughput impact of adaptive syntax element partitioning over

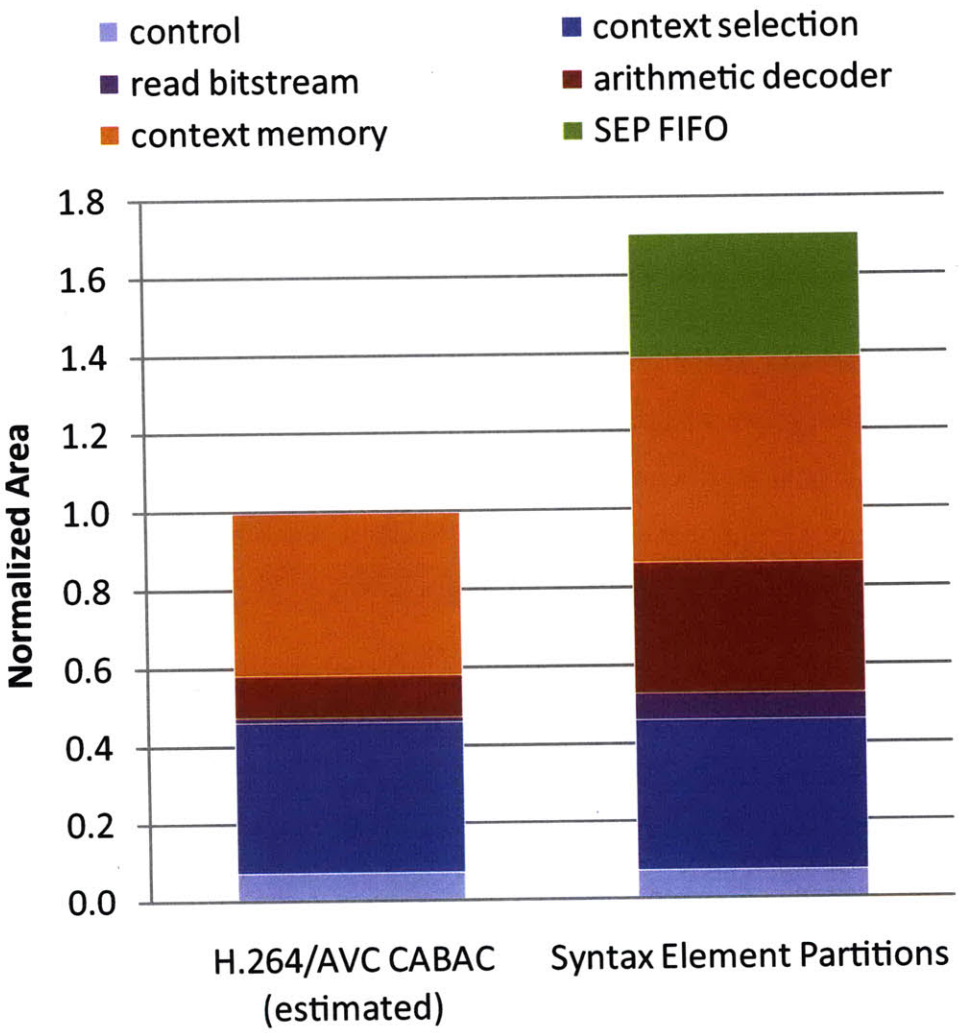
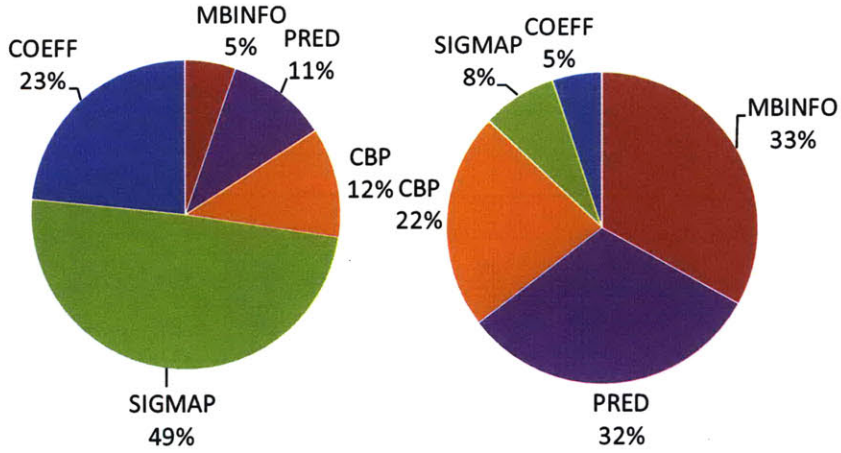


Figure 3-10: A comparison of the H.264/AVC CABAC area versus SEP engine area. The latter is approximately 70% larger due to the replication of the arithmetic decoder and the SEP FIFOs.



(a) QP=22 (b) QP=37

Figure 3-11: Average bin distribution per frame.

serial H.264/AVC CABAC determined from simulations under common conditions [34] using modified JM12.0 software. The average throughput is increased from 1.72x to 2.41x with adaptive QP. For most of the sequences, the QP threshold should be set somewhere between 27 and 32. However, for the shuttle sequence, the QP threshold should be between 22 and 27.

Achieving Additional Parallelism

To reach bin-rate in the Gbins/s range, syntax element partitions can be combined with the other approaches presented in Section 3.2 to achieve additional throughput at lower cost. For instance, a 6x throughput increase can be achieved by combining syntax element partitions with 4 entropy slices, which results in better coding efficiency and lower area costs than just using 8 entropy slices or 8 H.264/AVC slices as shown in Fig. 3-13. The slice and/or partition that has the most number of bins in a frame dictates the number of cycles required to decode that frame. Thus, the throughput improvement in Fig. 3-13 is determined by comparing the total number of bins in the frame with the slice and/or partition that has the most number of bins in that frame.

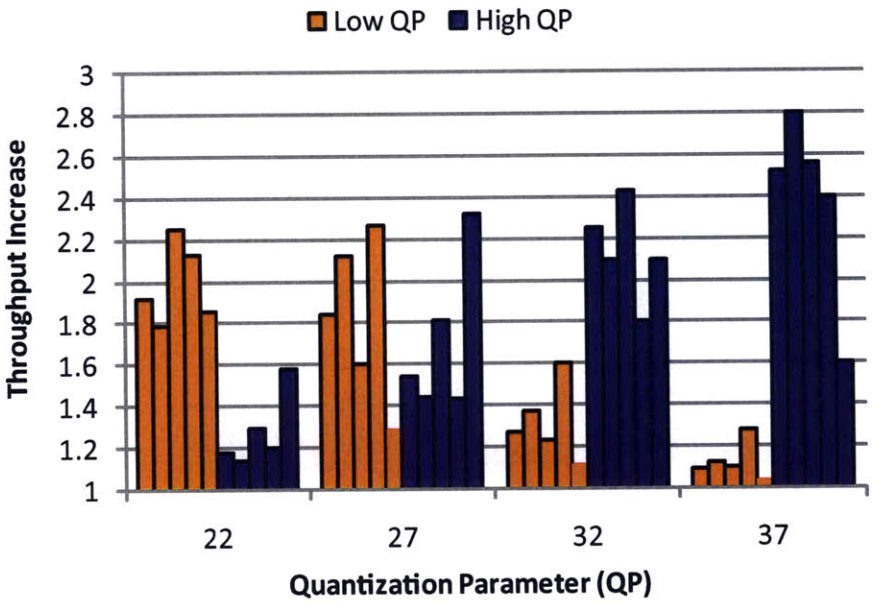


Figure 3-12: High QP (blue); Low QP (orange). Sequences (left to right): bigships, city, crew, night, and shuttle

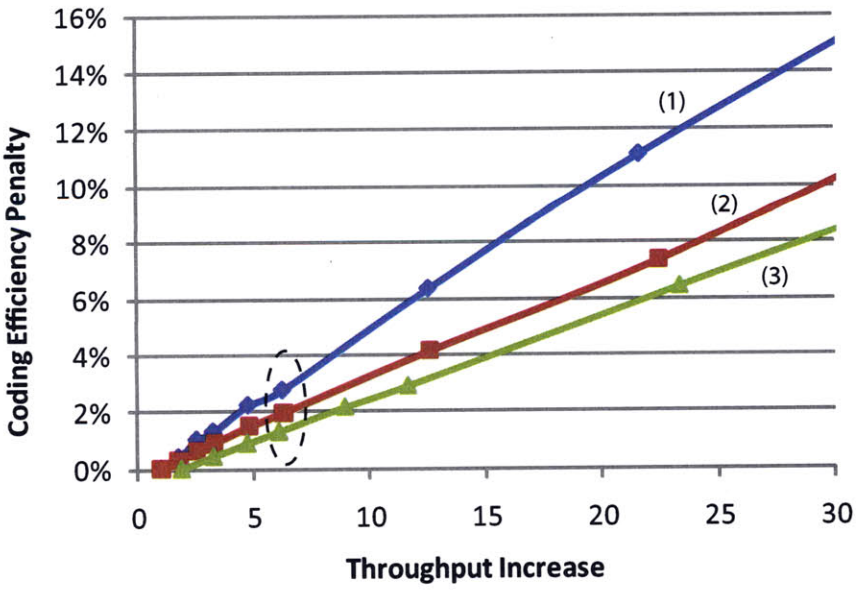


Figure 3-13: Coding efficiency vs. throughput for (1) H.264/AVC slices, (2) entropy slices and (3) entropy slices with syntax element partitions. Sequence bigships (QP=27) [34].

3.3.3 Interleaved Entropy Slices (IES)

To achieve additional throughput improvement, the previous approach (as well as bin parallelism) can be combined with slice parallelism such as entropy slices. As mentioned in Section 3.2.4, entropy slices can undergo independent entropy decoding in the 'front-end' of the decoder [33]. However, to achieve better coding efficiency than fully independent slices (i.e. H.264/AVC slices), there remains dependencies between the entropy slices for spatial and motion vector prediction in the 'back-end' of the decoder.

In the entropy slice proposals [75–77], the spatial location of the macroblocks allocated to each entropy slice is the same as in H.264/AVC (Fig. 3-14), i.e. contiguous groups of macroblocks. Due to the existing dependencies between entropy slices, back-end processing of slice 1 in Fig. 3-14 cannot begin until the last line of slice 0 has been fully decoded when using regular entropy slices. As a result, the decoded syntax elements of slice 1 need to be buffered as shown in Fig. 3-15, which adds to memory costs - on the order of several hundred megabytes per second for HD. In this work, we propose the use of *interleaved* entropy slices where macroblocks are allocated as shown in Fig. 3-16, i.e. for two slices, even rows are assigned to one slice, while odd rows are assigned to the other. Within each slice, the raster scan order processing is retained. Benefits of interleaved entropy slices include

1. cross slice context selection
2. simple synchronization
3. reduction in memory bandwidth
4. low latency
5. improved workload balance

In interleaved entropy slices, as long as the slice 0 is one macroblock ahead of slice 1, the top-macroblock dependency is retained, which enables cross slice context selection during parallel processing (i.e. spatial correlation can be utilized for better context selection) resulting in improved coding efficiency. This is not possible with regular entropy slices.

Synchronization between entropy slices can easily be implemented through the use of FIFOs between the slices (Fig. 3-17). Furthermore, both the front-end entropy processing

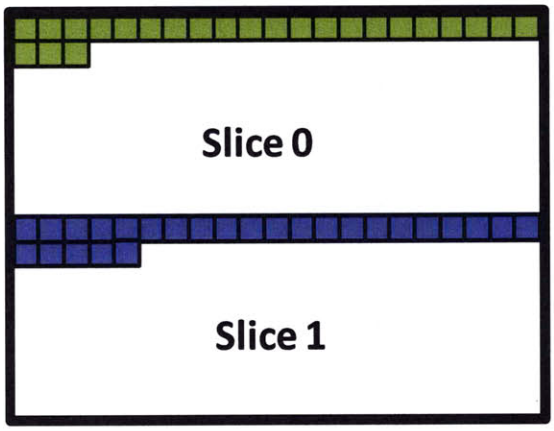


Figure 3-14: Macroblock allocation for entropy slices [75–77].

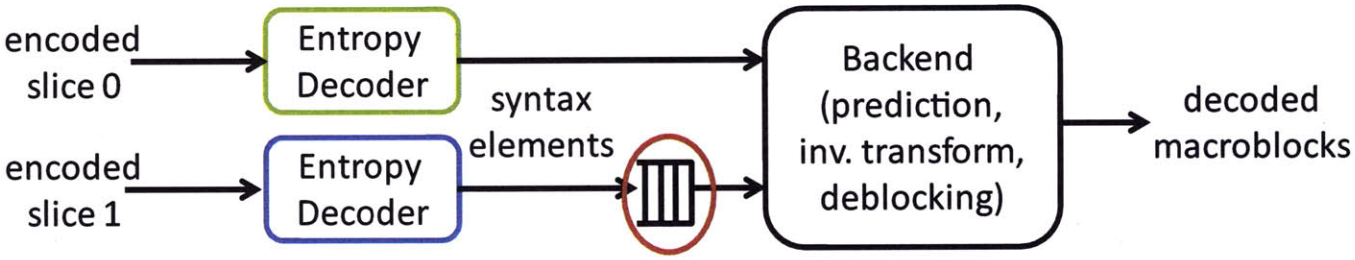


Figure 3-15: Decoded syntax elements need to be buffered.

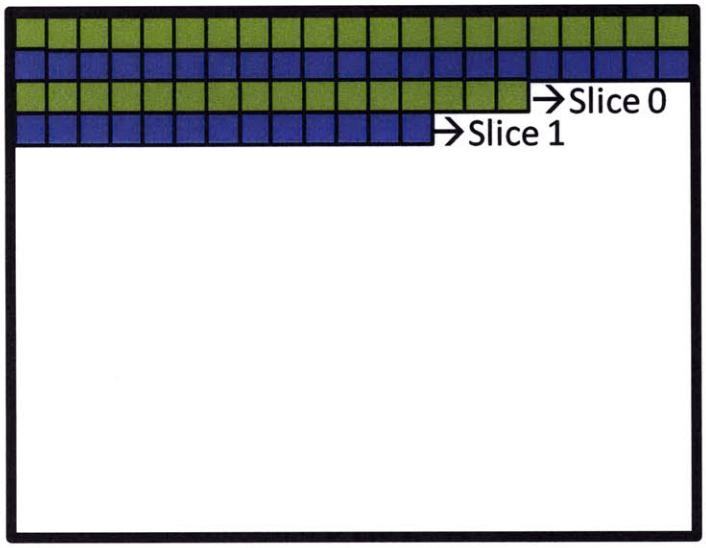


Figure 3-16: Macroblock allocation for interleaved entropy slices.

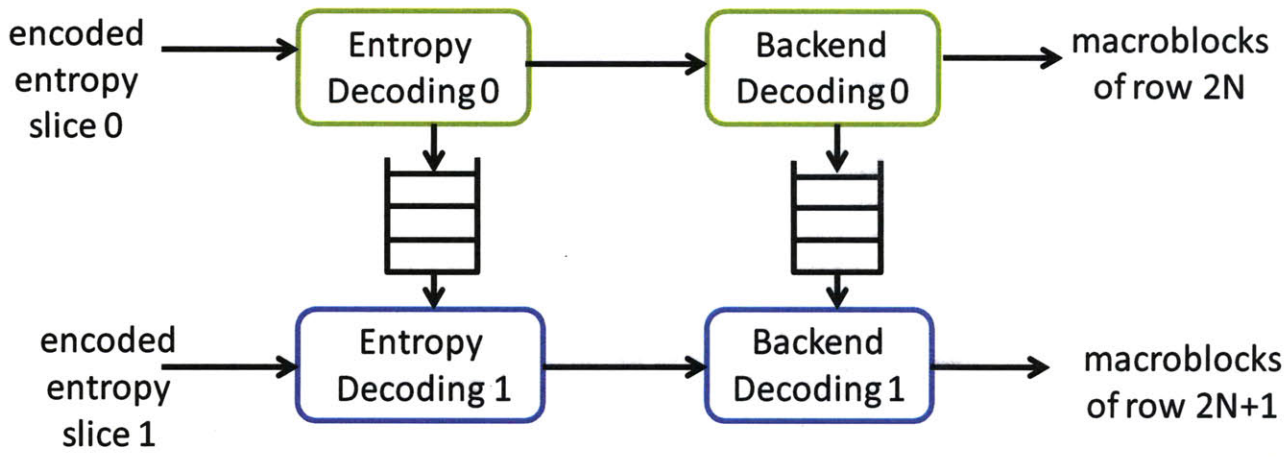


Figure 3-17: Interleaved Entropy Slices architecture example for 2x parallel decoding. Note that the *entire* decode path can be parallelized.

and the back-end prediction processing can be done in this order (i.e. the entire decoder path is parallelized), which allows the decoded syntax elements to be immediately processed by the back-end. Consequently, no buffering is required to store the decoded syntax elements, which reduces memory costs. This can have benefits in terms of reducing system power and possibly improving performance (by avoiding read conflicts in shared memory). In Appendix C, we examine how using interleaved entropy slices to parallelize the entire decoder path impacts the overall throughput and power of the entire video decoder.

The number of accesses to the large last line buffer is reduced for interleaved entropy slices. In Fig. 3-16, the last line buffer (which stores an entire macroblock row) is only accessed by slice 0. Since slice 0 is only several macroblocks ahead of slice 1, slice 1 only needs to access a small cache, which stores only a few macroblocks, for its last line (top) data. Thus out of N slices, N-1 will access small FIFOs for the last line data, and only one will access the large last line buffer. If the last line buffer is stored on-chip, interleaved entropy slices reduces area cost since it does not need to be replicated for every slice as with H.264/AVC and entropy slices. Alternatively, if the last line buffer is stored off-chip, the off-chip memory bandwidth for last line access is reduced by 1/N.

Unlike ordered entropy slices, interleaved entropy slices retains raster scan order processing within each entropy slice which provides a favorable memory access pattern for caching

techniques that enable further bandwidth reduction. Finally, in interleaved entropy slices the number of bins per slice tends to be more equally balanced; consequently, a higher throughput can be achieved for the same amount of parallelism.

At the limit, when the number of entropy slices equals the number of macroblock rows in a frame, interleaved entropy slices process macroblocks in the same order as regular entropy slices. In this case, a minimum latency in macroblocks of D=2 (as defined in [77]) is achieved. Another benefit of interleaved entropy slices is that its latency D is always equal to 2 regardless of the number of slices per frame. Thus, interleaved entropy slices are suitable for low latency applications (e.g. video conferencing).

Coding Efficiency and Throughput

We measured the throughput of interleaved entropy slice alone as well as in combination with syntax element partitions, which we call the MP-CABAC. Note that syntax element partitions can also be combined with any of the other entropy slice proposals. Fig. 3-18 compares their coding efficiency and throughput against regular and ordered entropy slices as well as H.264/AVC slices. Table 3.7 shows the coding efficiency across various sequences and prediction structures for throughput increase (speed up) of around 10x over serial 1 bin/cycle H.264/AVC CABAC. MP-CABAC offers an overall average 1.2x, 3.0x, and 4.1x coding penalty (BD-rate) reduction compared with ordered entropy slices, entropy slices, and H.264/AVC respectively. Data was obtained across different degrees of parallelism and plotted in Fig. 3-18. The throughput provided in Fig. 3-18 is averaged across five sequences, prediction structures (Ionly, IPPP, IBBP) and QP (22, 27, 32, 37). Note that the coding penalty should not exceed 16% since there would not longer be any coding advantage of CABAC over CAVLC.

To account for any workload imbalance, the slice with the largest number of bins in a frame was used to compute the throughput. The BD-rates for entropy slices and ordered entropy slices are taken directly from [77]. Since the macroblock allocation for these proposals are the same, the workload imbalance should also be the same. The workload imbalance was

measured based on simulations with JM12.0. The number of bins for each macroblock and consequently each entropy slice was determined and the throughput was calculated from the entropy slice in each frame with the greatest number of bins. Total number of bins processed by MP-CABAC, interleaved entropy slices, entropy slices and ordered entropy slices are the same; however, the number of bins for the H.264/AVC slices increases since the prediction modes and consequently syntax elements are different; this impact is included in the throughput calculations.

It should be noted that the coding efficiency for entropy slices and ordered entropy slices was obtained from implementations on top of the KTA2.1 software, which includes next generation video coding tools, while their throughput and the coding efficiency/throughput of interleaved entropy slices and MP-CABAC were obtained from implementations on top of the JM12.0 software, which contains only H.264/AVC tools. This accounts for the slight discrepancy in coding efficiency between the ordered entropy slices and interleaved entropy slices at 45x parallelism. In theory, they should be an exact match in terms of both coding efficiency and throughput.

Area Cost

As in the case of the entropy slices and ordered entropy slices, the area of the entire CABAC (including the context memory) must be replicated for Interleaved Entropy Slices (IES). Thus the total CABAC area increases linearly with parallelism as shown in Fig. 3-19. For the same throughput, interleaved entropy slices require less area increase since it needs fewer parallel engines due to its better workload balance. Table 3.8 shows that for a 10x throughput increase over serial 1 bin/cycle CABAC, interleaved entropy slices reduced area cost by 20%, while MP-CABAC reduces area cost by 60%. Furthermore, no buffering is required to store syntax elements and easy synchronization can be performed with FIFOs between the interleaved entropy slices. The simplicity of this approach allows the whole decoder to be parallelized. Note that a significant area cost reduction is achieved for MP-CABAC, when interleaved entropy slices are combined with syntax element partitions. As mentioned

Table 3.7: A comparison of the coding efficiency penalty (BD-rate) versus throughput for Parallel CABAC proposals. Speed up and BD-rates are measured against serial 1 bin/cycle one slice per frame H.264/AVC CABAC.

		H.264/AVC slices		Entropy Slices		Ordered Entropy Slices		IES only		MP-CABAC (IES + syntax element partitioning)	
	Video Sequence	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up	BD-rate	Speed up
Ionly	bigships	3.29	8.49	1.21	8.61	0.38	8.61	1.04	10.73	0.51	9.03
	city	3.02	11.89	0.63	12.12	-0.01	12.12	0.81	10.28	0.43	8.88
	crew	8.47	9.29	2.05	9.48	0.24	9.48	1.80	9.32	0.97	10.24
	night	4.10	9.66	0.68	9.83	-0.09	9.83	0.62	10.65	0.37	9.62
	shuttle	7.55	8.80	1.97	9.17	0.84	9.17	3.34	9.58	1.81	11.42
	Average	**5.29**	**9.62**	**1.31**	**9.84**	**0.27**	**9.84**	**1.52**	**10.11**	**0.82**	**9.85**
IPPP	bigships	5.65	9.95	5.69	10.31	2.01	10.31	2.37	9.91	1.91	9.77
	city	9.01	10.67	4.86	11.69	1.27	11.69	2.19	9.93	1.99	9.73
	crew	10.00	10.01	12.96	10.63	8.49	10.63	2.34	9.91	1.90	10.07
	night	4.87	8.27	2.14	8.50	0.56	8.50	1.67	10.02	1.30	10.44
	shuttle	1.95	8.47	9.63	8.84	2.93	8.84	5.06	9.96	3.99	10.70
	Average	**8.09**	**9.48**	**7.06**	**9.99**	**3.05**	**9.99**	**2.72**	**9.95**	**2.22**	**10.14**
IBBP	bigships	7.50	10.32	2.88	10.59	3.30	10.59	2.76	9.89	2.15	9.79
	city	11.31	11.53	5.69	12.20	1.67	12.20	2.73	10.16	2.40	9.83
	crew	11.01	10.28	4.86	10.79	6.00	10.79	2.58	10.13	1.99	10.49
	night	5.50	8.51	12.96	8.72	1.91	8.72	2.27	10.13	1.61	11.14
	shuttle	13.36	8.99	13.41	9.23	5.94	9.23	5.68	10.09	4.68	10.39
	Average	**9.73**	**9.93**	**8.75**	**10.30**	**3.76**	**10.30**	**3.21**	**10.08**	**2.57**	**10.33**

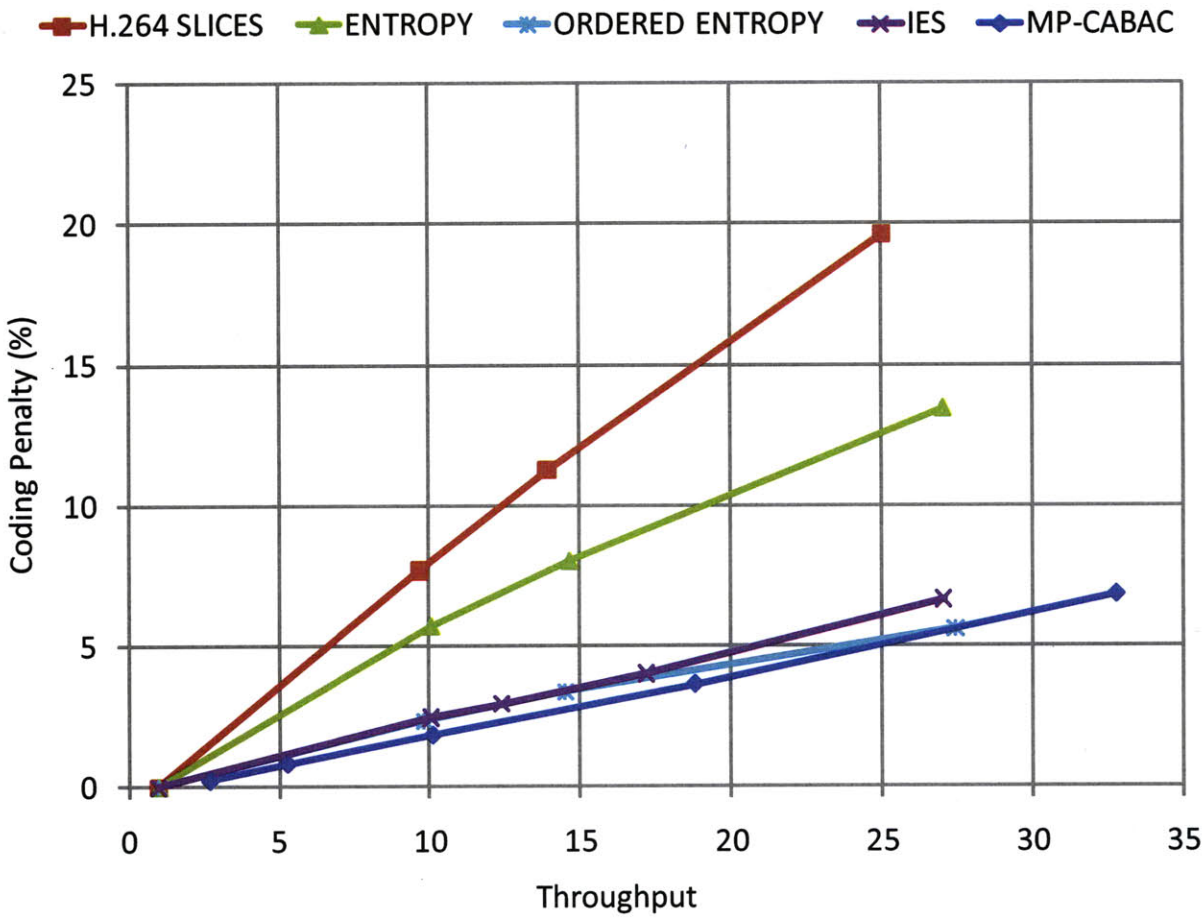


Figure 3-18: Tradeoff between coding efficiency and throughput for various parallel CABAC approaches. Coding efficiency and throughput are averaged across prediction structures, sequences and quantization. Note that the area cost versus throughput tradeoff for the entropy slices and ordered entropy slices are the same since they have the same throughput.

Table 3.8: A comparison of features of Parallel CABAC proposals

	H.264/AVC slices	Entropy Slices	Ordered Entropy Slices	IES only	MP-CABAC (IES + syntax element partitioning)
Contribution	Anchor	[76]	[77]	[35]	[35]
Software	JM12.0	KTA2.1	KTA2.1	JM12.0	JM12.0
Average BD-rate	7.7%	5.71%	2.36%	2.48%	1.87%
Area Cost	15x	15x	15x	12x	6x
Context Selection across Entropy Slices	No	No	Yes	Yes	Yes
Latency (for last slice to begin)	0	0	126 macroblocks	22 macroblocks	6 macroblocks
Syntax Element Memory BW Cost	No	Yes	No	No	No
MC cache size	same	same	increase(zig-zag)	same	same
Minimum Number of Slices	1	1	15	1	1

earlier, if the last line buffer is stored on-chip, interleaved entropy slices provides additional area savings since the buffer does not need to be replicated for every slice.

3.4 Standardization Effort

The video standards are required to enable compatibility between encoder and decoders. The VCEG standards body, which developed H.264/AVC, is currently investigating tools for the next generation video coding standard, unofficially termed 'H.265'. Proposals with tools which the body identifies as promising are incorporated into their working software known as KTA. The MP-CABAC was proposed to VCEG in July 2009 [35], and after cross-verification was adopted into the software in November 2009. The software was released to the public in March 2010 as KTA2.7 [79]. As part of this effort, the MP-CABAC had to demonstrate compatibility with other tools deemed promising for the next generation standard. This includes tools such as

Figure 3-19: Area cost versus throughput tradeoff for the various parallel CABAC approaches.

- enhanced adaptive interpolation filter
- mode-dependent transform customization for intra coding
- high precision filter
- extended macroblock size
- rate-distortion optimized quantization
- new offset for weighted prediction
- quadtree-based adaptive loop filtering
- competition for motion vector prediction

3.5 Summary and Conclusions

In this chapter, the Massively Parallel CABAC (MP-CABAC) algorithm was presented which leverages both syntax element and slice parallelism. MP-CABAC involved reorganizing the data (syntax elements) in an encoded bitstream such that the bins (workload) can be distributed across different parallel processors and multiple bins can be decoded simultaneously without significant increase in coding penalty and implementation cost.

Benefits of the MP-CABAC include

1. high throughput
2. low area cost
3. good coding efficiency
4. reduced memory bandwidth
5. simple synchronization and implementation
6. low latency
7. enables full decoder parallelism

For a 2.7x increase in throughput, syntax element partitions were shown to provide between 2 to 4x reduction in coding penalty when compared to slice parallel approaches, and close to 2x reduction in area cost. The reduction in coding penalty can be attributed to the fact that, unlike slice parallelism, the probability estimate of the contexts is not degraded by breaking the frame into syntax element partitions. The reduction in area costs is due to

the fact that, unlike slice parallelism, the context memory and context selection do not have to be replicated for syntax element partitions. When combined with interleaved entropy slices to form the MP-CABAC, additional throughput improvement can be achieved with low coding penalty and area cost. For a 10x increase in throughput, the coding penalty was reduced by 1.2x, 3x and 4x relative to ordered entropy, entropy and H.264/AVC slices respectively. Over a 2x reduction in area cost was achieved. The impact of IES on the overall decoder power and performance was also presented. Finally, the MP-CABAC was adopted into the KTA software, where it showed compatibility with other tools being considered for 'H.265'.

Chapter 4

Architecture of Massively Parallel CABAC

This chapter describes the architecture of the MP-CABAC and demonstrates how joint optimization of algorithm and architecture can improve performance and reduce area with little to no coding penalty. The use of data and clock gating techniques to reduce power consumption will be described. In cases where algorithm changes are required, coding efficiency costs are evaluated.

Note that while both syntax element partitions and interleaved entropy slices are supported in the architecture, bin parallelism and adaptive bin allocation are not included in the implementation.

4.1 Data Structure

Fig. 4-1 shows the structure of the encoded data which describes the frames in the video sequence. The encoded bitstream is generated using the MP-CABAC algorithm described in Chapter 3. General information pertaining to the video sequence and frame are transmitted at the beginning of the bitstream, before any macroblock specific information is sent, as in the case of H.264/AVC [26]. Specifically, the SPS is transmitted which describes sequence

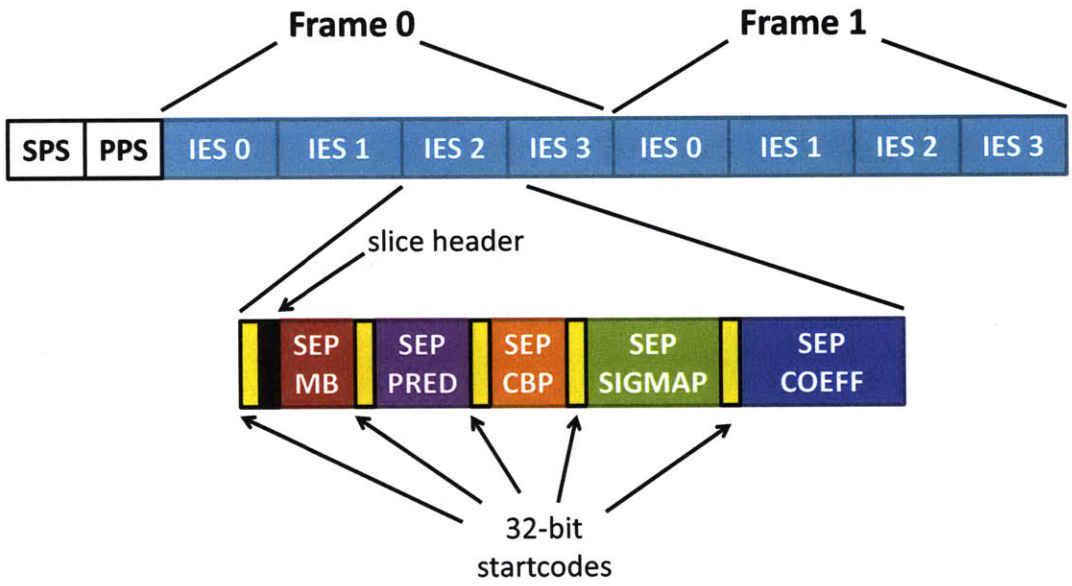


Figure 4-1: Data structure of the encoded bitstream with MP-CABAC. In this example, four IES are used per frame.

related information, such as dimension of the frame in macroblocks, the chroma format (e.g. 4:2:0), etc. The SPS is followed by PPS which describes the properties of the different frames (pictures) in the sequence, such as the number of slices per frame, the entropy coding mode of the frame, etc. Rather than transmitting this information in every slice, the slice header will simply contain an index that points to the stored PPS, and each PPS points to a stored SPS.

For the MP-CABAC, each frame is composed of several IES and each IES is composed of five SEP. A 32-bit startcode is inserted at the beginning of each partition to enable the parser to access any partition within the bitstream. The slice header information, such as slice type (I, P, B), slice quantization, etc., is inserted at the beginning of the MBINFO partition. The partitions can then be distributed across several engines to be processed in parallel. Fig. 4-2

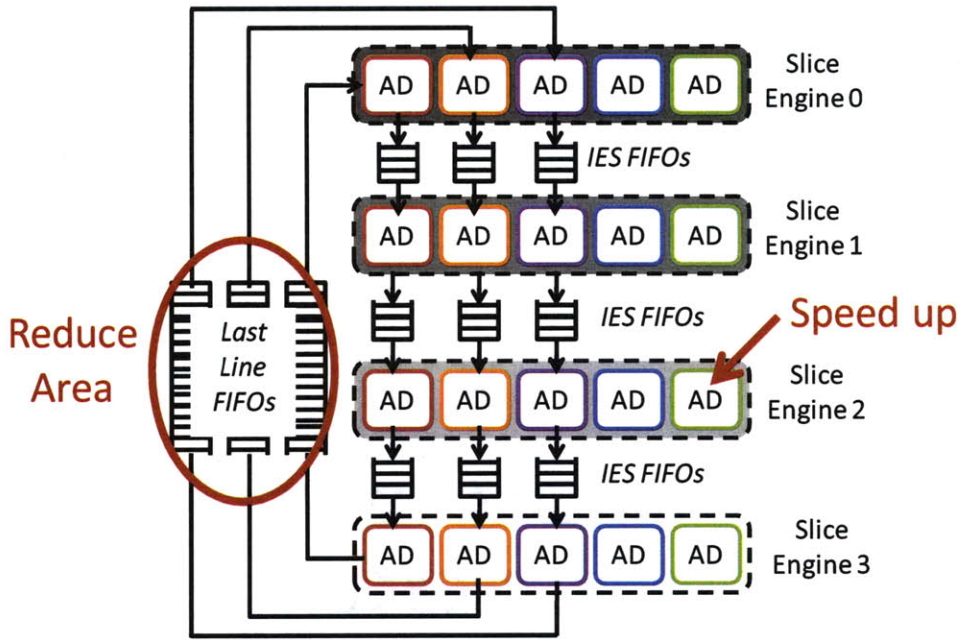


Figure 4-2: Top level architecture of MP-CABAC. Joint architecture/algorithm optimizations were performed to speed up the AD and reduce the area of the last line FIFO.

shows the top level MP-CABAC architecture used to decode the data structure in Fig. 4-1. The following sections will discuss how several arithmetic decoders (AD) were connected through FIFOs to form a slice engine, and how these slice engines were connected through FIFOs to form the MP-CABAC. Joint architecture/algorithm optimizations were performed to speed up the AD and reduce the area of the last line FIFO.

4.2 CABAC Engine

The CABAC engine is used to decode bins from encoded bits and map the bins to syntax elements which describe how to reconstruct the video sequence. The main steps in the CABAC engine can be grouped into two modules: context selection[1] and binary arithmetic decoder. Fig. 4-3 shows the connections between these two modules.

To decode a bin, the CABAC engine uses its context selection module to determine the appropriate context model, and sends its probability to the binary arithmetic decoder. A

[1]We have included de-binarization in the context selection module.

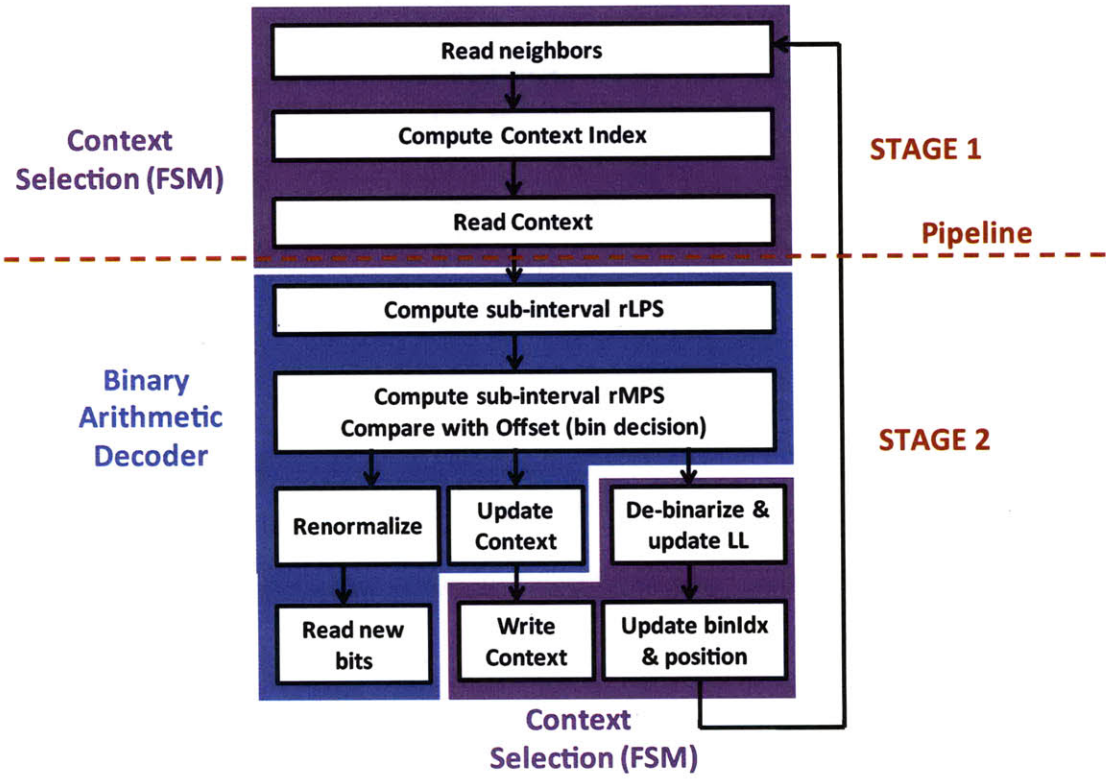


Figure 4-3: Main steps in CABAC decoding

total of 307 context models are used in the MP-CABAC.[2] The context model is identified by its context index, which is computed based on several factors:

1. slice type (I, P, B)
2. syntax element (e.g. coefficient, motion vector, mb_type, etc.)
3. bin position (binIdx) within the syntax element codeword
4. properties of neighboring left and top macroblocks (e.g. type of prediction - intra/inter)
5. block type (e.g. luma/chroma, AC/DC, 4x4/8x8/16x16)

These properties enable bins with correlated probabilities to be grouped together and share the same context model.

The current probabilities of the context models are stored in memory. The probabilities are represented by a 7-bit value (6-bits for the state, and 1-bit for the MPS). Once the context model has been selected, its probability is read from the context memory using the context index as the address.

The arithmetic decoder uses the probability to divide the range into two intervals (the range of LPS, *rLPS*, and the range of MPS, *rMPS*) compares the offset to the intervals, and makes a decision about the decoded bin. It then updates and renormalizes the range and sends the updated context probability back to the context memory.

The decoded bin is fed back to the context selection module, where the de-binarizer maps the decoded bin to the syntax element which is then used to determine which context model should be used next. The offset is updated with new bits from the encoded bitstream based on the number of bits shifted during the range renormalization.

Various architecture and algorithm optimizations were applied within each of the two modules. In the cases where the optimizations requires modifications to the algorithm, simulations were performed under common conditions to measure the impact on coding efficiency.

[2]For the MP-CABAC, the number of contexts is reduced from 468 to 307 since interlaced is unlikely to be supported in the next generation standard.

4.2.1 Context Selection

An FSM is used to select the appropriate context model for each bin based on properties such as slice type, syntax element, etc. The FSM keeps track of the state information such as current syntax element, binIdx, number of decoded syntax elements in current macroblock, etc. It also performs de-binarization to map the bins to a syntax element. Based on the state of the FSM, the address of the context (i.e. context index) is computed and its current state (and MPS) is read from the context memory.

Pipelining

Pipelining is a popular technique used to increase performance by reducing the critical path delay and increasing concurrency. It involves breaking up the datapath into several stages that can operate concurrently at a higher frequency [17]. The feedback loop in the CABAC engine makes it challenging to pipeline.

In this work, we propose pipelining the CABAC engine as shown in Fig. 4-3, such that context selection for the *next* bin can be performed at the same time as the arithmetic decoding of the *current* bin. However, the context index of the next bin depends on the value of the current bin being decoded. To address this data dependency, while the arithmetic decoder (stage 2) is decoding the current bin, the two context candidates for the next bin are computed by the context selection module (stage 1). The power overhead of computing the additional context candidate is around 14%.[3] Once the current bin is decoded, it is used to select between the two context candidates for the next bin. The context index of the next bin is compared with the context index of the current bin. If they are the same, then the updated context state is used for the next bin. Otherwise, the context state is read from the memory; renormalization and de-binarization are performed at the same time. Fig. 4-4 shows the interaction between the two pipeline stages that is required due to the feedback loop.

[3]Post-synthesis simulations show that 24% of the total power goes towards the context calculation for two candidates. We assume that this power is reduced by half if only one context is calculated. Thus the total overhead of the additional context calculation is 1-100/(100-24*0.5)=13.6%.

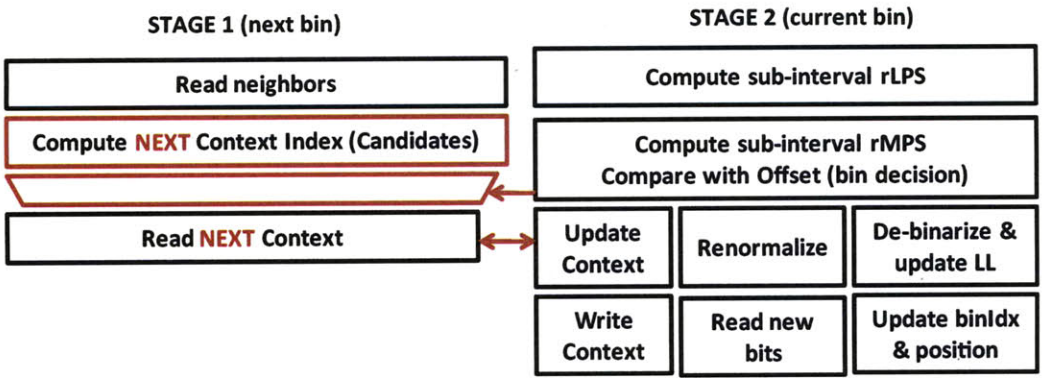


Figure 4-4: Pipelining CABAC engine to speed up performance. Connections between the two pipeline stages required due to feedback loop are highlighted in red.

Pipelining the CABAC in this manner reduces its critical path by approximately 40%. The area costs for the additional context candidate computation logic is only around 1.5 kgates, which is less than 3% of the total CABAC engine area.

Modified mvd Context Selection

To make use of the spatial correlation of neighboring pixels, context selection can depend on the values of the top and left blocks. A last line buffer is required in the CABAC engine to store information pertaining to the previously decoded row. The depth of this buffer depends on the width of the frame being decoded which can be quite large for high resolution (e.g. 4kx2k) sequences. The bit-width of the buffer depends on the type of information that needs to be stored per block or macroblock in the previous row. We propose reducing the bit-width of this data to reduce the overall last line buffer size.

Specifically, we propose modifying the context selection for motion vector difference (mvd). mvd is used to reduce the number of bits required to represent motion information. Rather than transmitting the motion vector, the motion vector is predicted from its neighboring 4x4 blocks and only the difference between motion vector prediction (mvp) and motion vector (mv), referred to as mvd, is transmitted.

$$\text{mvd} = \text{mv} - \text{mvp}$$

A separate mvd is transmitted for the vertical and horizontal components. The context

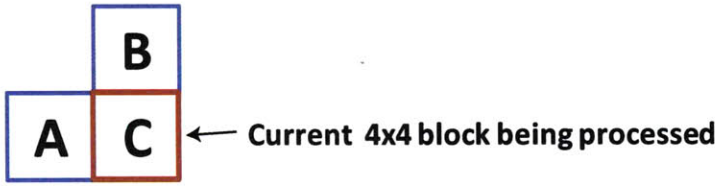


Figure 4-5: Last line dependencies for mvd.

selection for mvd depends on its left (A) and top (B) 4x4 neighbors as shown in Fig. 4-5. Consequently, a last line buffer is used to store the mvds of the previous line.

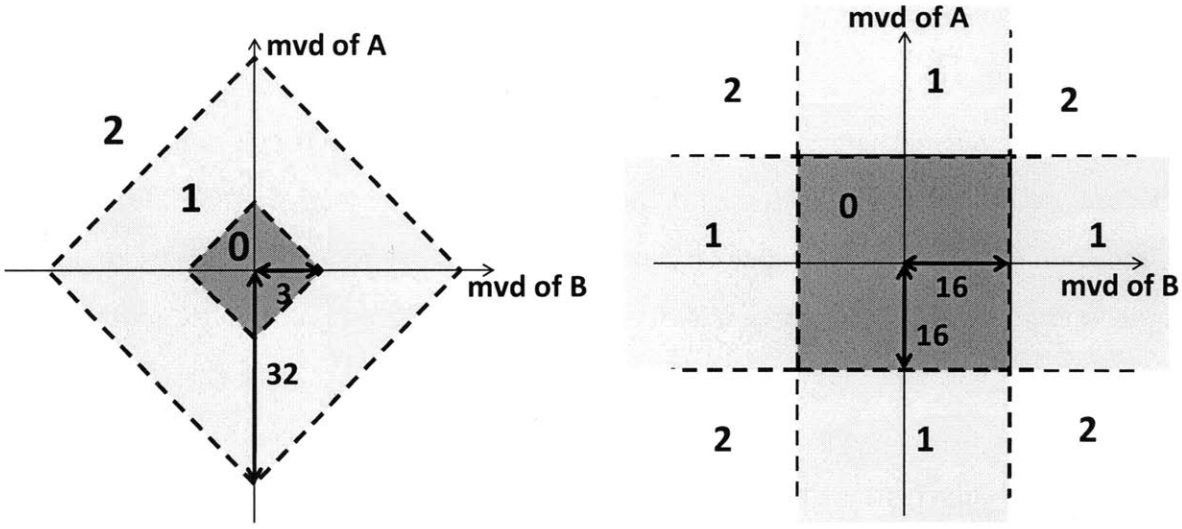
In H.264/AVC, neighboring information is incorporated into the context selection by adding a context index increment (typically between 0 to 3) to the calculation of the context index. The mvd context index increment, χ_{mvd}, is computed by taking the sum of the neighboring mvd and comparing it to thresholds of 3 and 32 as shown below [67].

$$\chi_{mvd} \begin{cases} 0, & \text{if e(A,B,cmp)}<3 \\ 1, & \text{if } 3 \leq \text{e(A,B,cmp)} \leq 32 \\ 2, & \text{if e(A,B,cmp)}>32 \end{cases}$$

where e(A,B,cmp)=|mvd(A,cmp)|+|mvd(B,cmp)|; A and B represent the left and top neighbor and *cmp* indicates whether it is a vertical or horizontal component. Fig. 4-6a illustrates how the above equation maps the mvd of A and B to different χ_{mvd}. In a given slice, all blocks surrounded by large mvds will use the same probability model (χ_{mvd}=2). Blocks surrounded by small mvds will use another probability model (χ_{mvd}=0 or χ_{mvd}=1).

With the upper threshold set to 32, a minimum of 6-bits of the mvd has to be stored per component per 4x4 block in the last line buffer. For 4kx2k, there are 1024 4x4 blocks per row, which implies 12,228 bits are required for mvd storage.

To reduce the memory size, rather than summing the components and then comparing to a threshold, each component can be separately compared to a threshold and their results can be summed.



(a) H.264/AVC (b) 1-bit per component per 4x4.

Figure 4-6: Context increments χ_{mvd} for different mvd in top (A) and left (B) neighboring 4x4 blocks.

i.e. χ_{mvd} = (|mvd(A,cmp)| >16) + (|mvd(B,cmp)| >16) or equivalently,

$$\chi_{mvd} \begin{cases} 0, & \text{if } |\text{mvd(A,cmp)}| \leq 16 \text{ AND } |\text{mvd(B,cmp)}| \leq 16 \\ 1, & \text{if } (|\text{mvd(A,cmp)}| \leq 16 \text{ AND } |\text{mvd(B,cmp)}| > 16) \text{ OR} \\ & (|\text{mvd(A,cmp)}| > 16 \text{ AND } |\text{mvd(B,cmp)}| \leq 16) \\ 2, & \text{if } |\text{mvd(A,cmp)}| > 16 \text{ AND } |\text{mvd(B,cmp)}| > 16 \end{cases}$$

Fig. 4-6b illustrates how the above equation maps the mvd of A and B to different χ_{mvd}. A single threshold of 16 is used. Consequently, only a single bit is required to be stored per component per 4x4 block; the size of the last line buffer for mvd is reduced to 2,048 bits. This reduces the overall last line buffer size of the CABAC by 50%, from 20,480 bits to 10,240 bits. The coding penalty of this approach was verified across common conditions to be 0.02%.

4.2.2 Binary Arithmetic Decoder

Fig. 4-7 shows the architecture of the binary arithmetic decoder. The inputs to the arithmetic decoder include current context state (and MPS), next bits, number of next bits (previous shift), and decoding mode. The outputs include updated context state (and MPS), decoded bin and number of shifted bits due to renormalization. The range and offset are stored as internal states. CABAC uses three arithmetic coding modes: regular, bypass, and terminate.

Data gating is a well known technique used to reduce switching activity when there are multiple modes. It involves inserting registers at the inputs to some combinational logic such that the inputs only switch when the outputs of the combinational logic will be used. It was used in the binary arithmetic decoder to reduce switching in the regular mode logic when either the bypass or terminate mode was being used. While the switching in the regular mode logic decreased by 12%, the control for the additional registers caused additional switching and the registers themselves increased the number of nets being switched on the clock path. As a result, the overall reduction in switching activity was less than 7%.

Bypass and terminate do not require context models and thus have a simpler data flow. Their architectures are shown in Fig. 4-8 and Fig. 4-9. The critical path lies in the regular mode as it uses the context models. Fig. 4-10 shows the architecture for the regular mode in the binary arithmetic decoder. This architecture was mapped directly from the data flow diagram, shown in Fig. 4-11, taken from the H.264/AVC standards document [26]. Four optimizations were performed on the architecture to increase concurrency and shorten the critical path as shown in Fig. 4-12. The aggregate impact of these optimizations was a 22% reduction in the critical path delay. We will now discuss each of these optimizations in detail.

(1) Range Comparison Reordering

In H.264/AVC, the rMPS is compared to the offset to determine whether the bin is MPS or LPS. The rMPS interval is computed by first obtaining rLPS from a 64x4 LUT (using bits [7:6] of the current 9-bit range and the 6-bit probability state from the context) and then subtracting it from the current range. The LUT contains constant values and is implemented

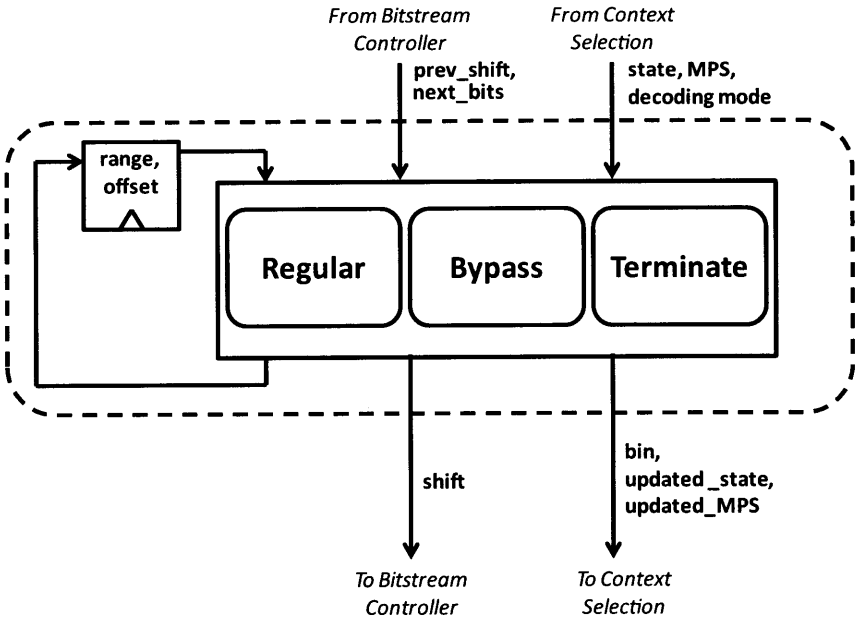


Figure 4-7: Architecture of arithmetic decoder with three different decoding mode.

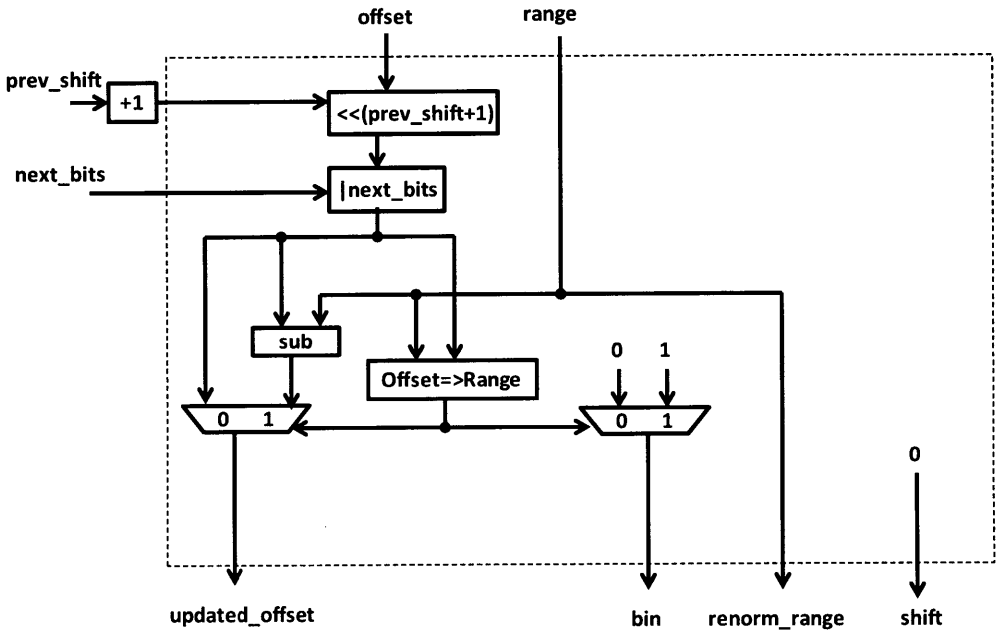


Figure 4-8: Bypass decoding path.

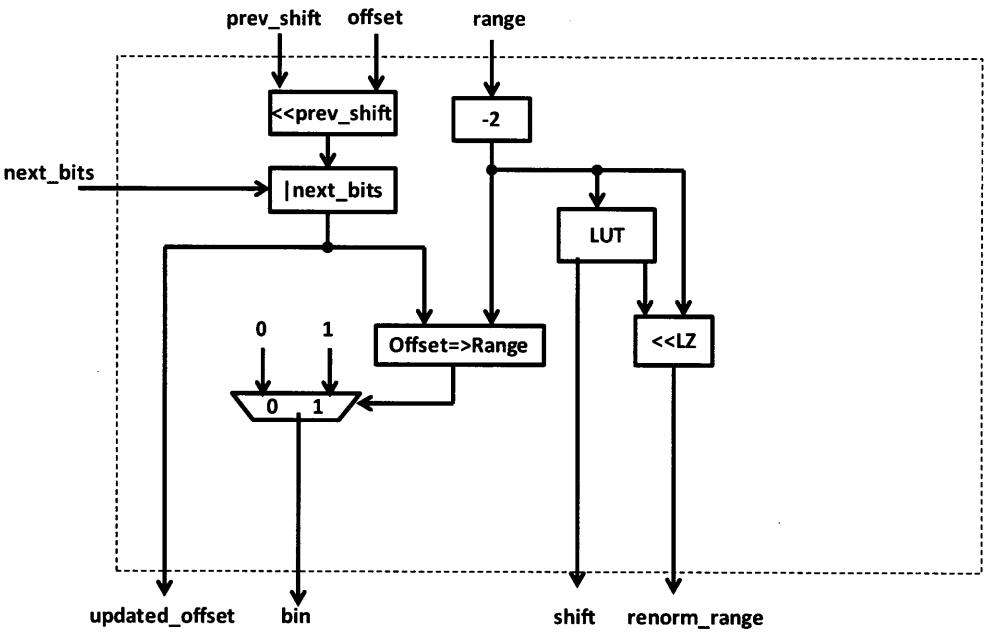


Figure 4-9: Terminate decoding path.

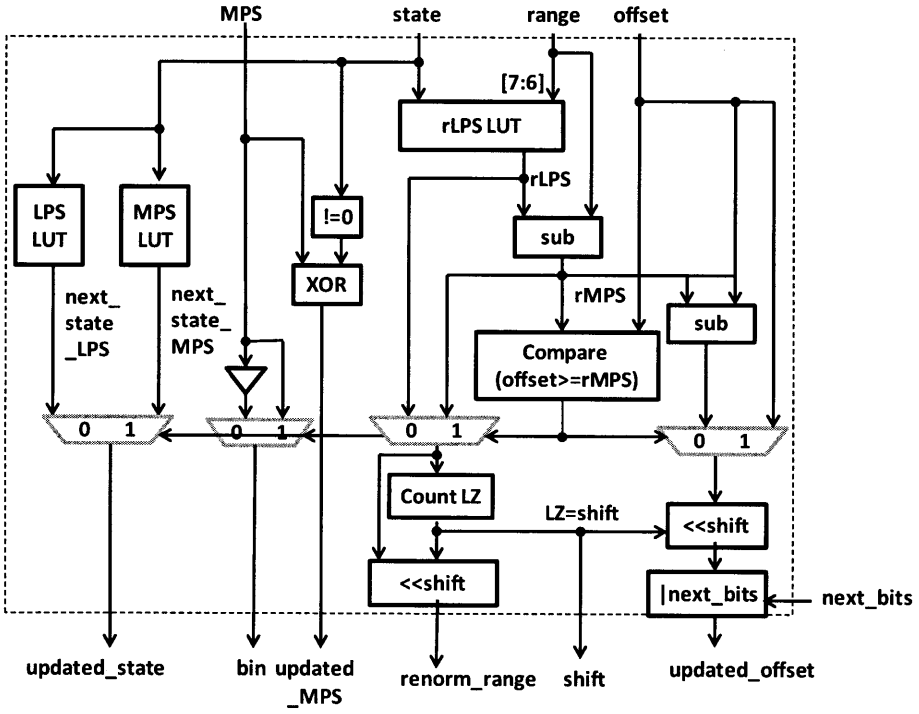


Figure 4-10: Binary arithmetic decoding architecture mapped directly from data flow in Fig. 4-11

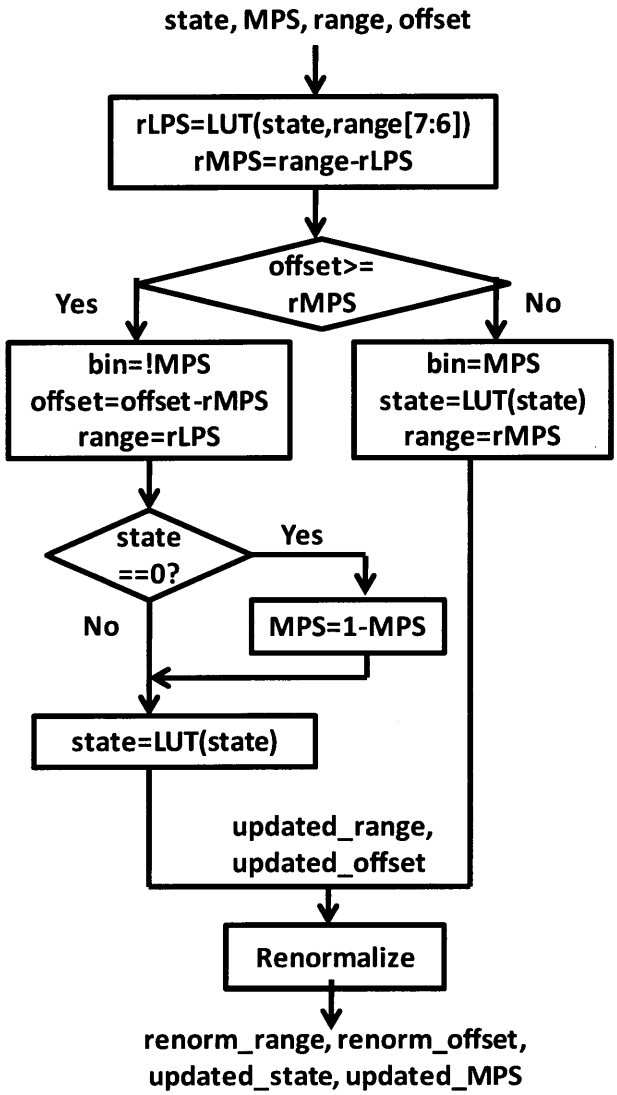


Figure 4-11: Data flow in binary arithmetic decoder.

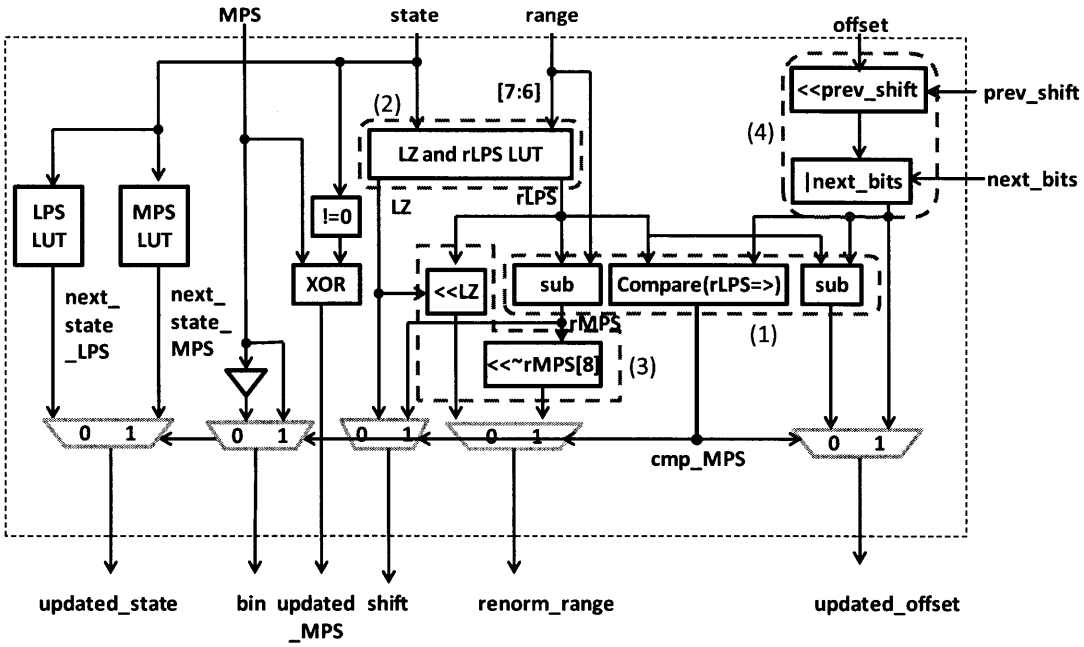
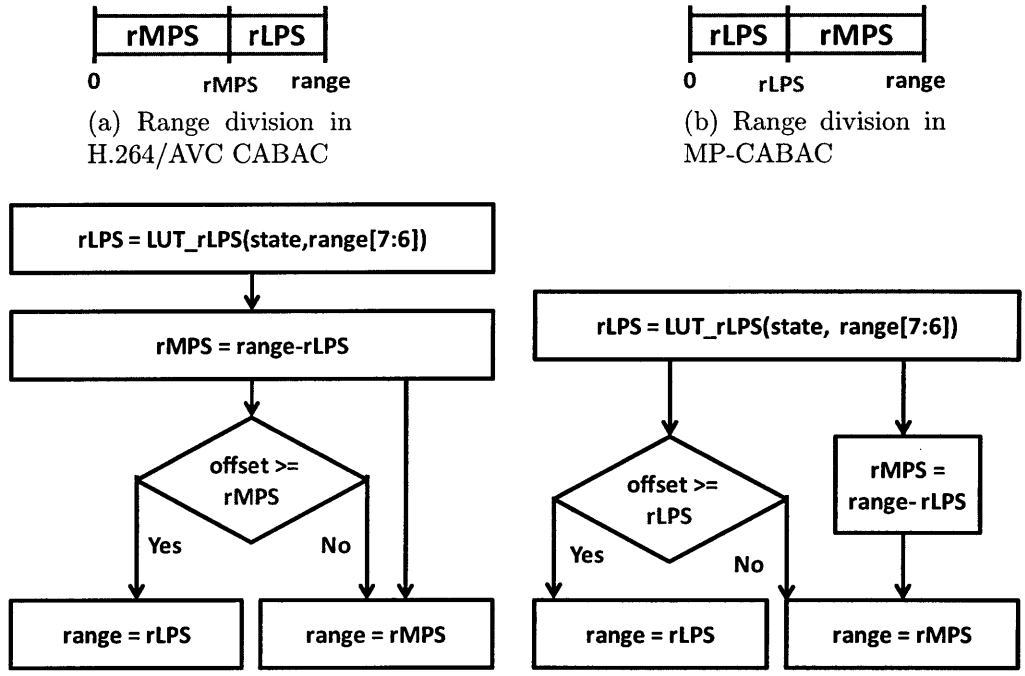


Figure 4-12: Binary arithmetic decoding architecture optimized for reduced critical path delay. Highlighted optimizations include (1) Range Comparison Reordering (2) Leading Zero LUT (3) Early Range Shifting (4) Next Cycle Offset Renormalization

with muxes. Depending on whether an LPS or MPS is decoded, the range is updated with their respective intervals. To summarize, the range division steps in the arithmetic decoder are

1. obtain rLPS from the 64x4 LUT
2. compute rMPS by subtracting rLPS from current range
3. compare rMPS with offset to make bin decoding decision
4. update range based on bin decision.

If the offset was compared to rLPS rather than rMPS, then the comparison and subtraction to compute rMPS can occur at the same time. Fig. 4-13 shows the difference between the range order of H.246/AVC CABAC and MP-CABAC. The two orderings of the intervals (i.e. which interval begins at zero, as illustrated in Fig. 4-13a and Fig. 4-13b) are mathematically equivalent in arithmetic coding and thus changing the order has no impact on coding efficiency. With this change, the updated offset is computed by subtracting rLPS from offset rather than rMPS. Since rLPS is available before rMPS, this subtraction can also be done in



(a) Range division in H.264/AVC CABAC

(b) Range division in MP-CABAC

(c) Range to offset comparison in H.264/AVC CABAC

(d) Range to offset comparison in MP-CABAC

Figure 4-13: Range comparison reordering for critical path reduction.

parallel with range-offset comparison. Changing the order of rLPS and rMPS requires the algorithm to be modified and is not H.264/AVC standard-compliant (i.e. it is a proposed modification for the next generation standard). It was verified through simulation that there is no coding penalty for this change. This optimization accounts for half of the overall 22% critical path reduction.

(2) Leading Zero LUT

After the range is updated based on the bin decision, renormalization may be necessary for the range and offset due to the use of finite bit precision. Renormalization involves determining the number of leading zeros (LZ) in the updated range and shifting the range accordingly (Fig. 4-14). LZ can be determined through the use of muxes in the form of a priority encoder. However, using a serial search for the first non-zero can increase the critical path.

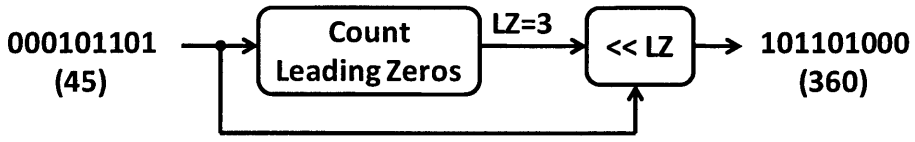


Figure 4-14: Range renormalization flow.

If an LPS is decoded, the updated range is rLPS and renormalization must occur. Recall that rLPS is stored in a 64x4 LUT implemented with muxes, indexed by the probability state and bits [7:6] of the original range. Since every rLPS can be mapped to a given LZ, an LZ LUT can be generated that is also indexed by the probability state and original range. This enables LZ to be determined in parallel with rLPS and reduces the critical path by avoiding the serial priority encoder.

If an MPS is decoded, LZ can only be determined after the rMPS subtraction. However, LZ can be quickly determined from the MSB of the updated range rMPS and thus has little impact on the critical path.

(3) Early Range Shifting

After a decision is made on the decoded bin, and LZ is determined, the range is shifted to the left based on LZ.

The shifting can be implemented using shift registers; however, this approach of moving one bit per cycle results in up to 7 clock cycles per renormalization (for the minimum range value is 2). Alternatively, the shifting can be done through the use of a 9:1 mux which can be done in a single cycle, but may increase the critical path.

To mitigate this, the shifting can be done *before* the decoded bin is resolved. Specifically, rLPS is shifted in parallel with the range-offset comparison and rMPS subtraction described earlier. rMPS is shifted by a maximum of one, which can be done quickly after the rMPS subtraction. Once the decoded bin is resolved, the range can be immediately updated with the renormalized rLPS or rMPS.

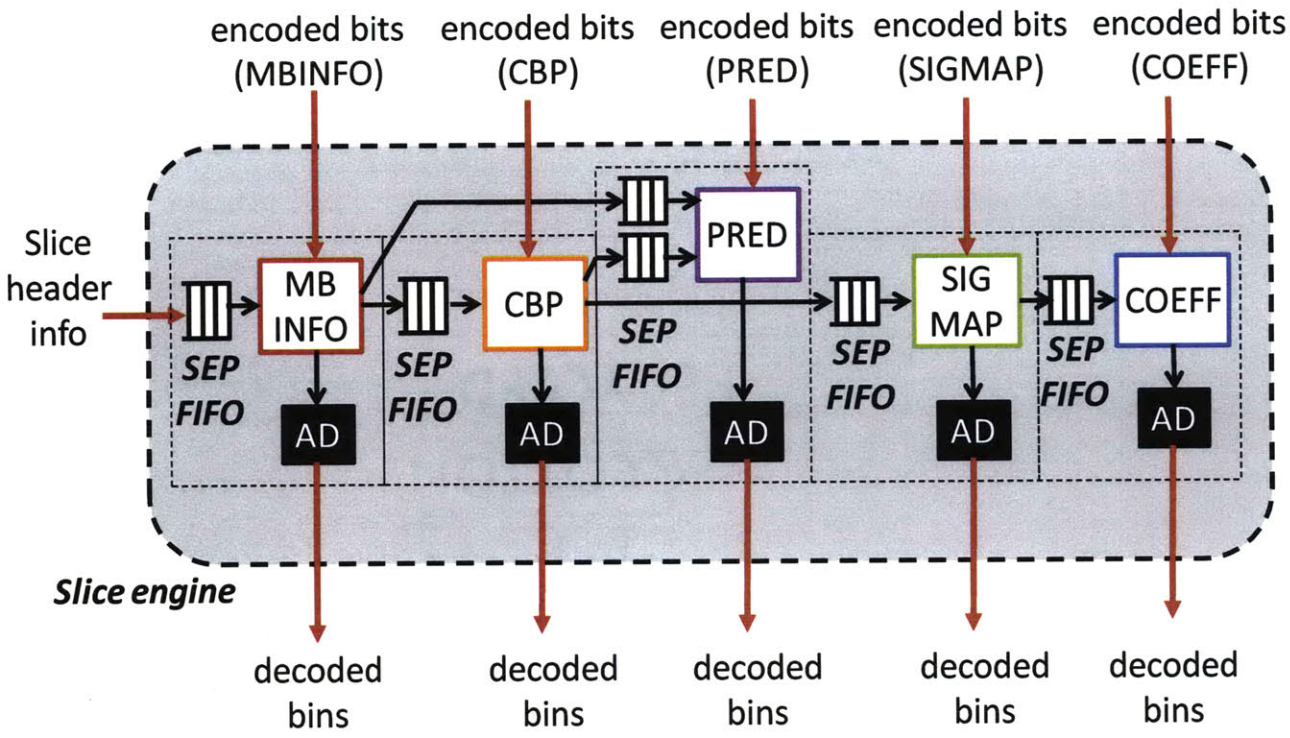


Figure 4-15: The slices engines architecture used to decode SEP in parallel.

(4) Next Cycle Offset Renormalization

Offset renormalization involves a left shift by the same amount based on the LZ of the range, and new bits are appended to the right. The offset is not used until after rLPS is determined. Therefore, rather than performing this shift in the current cycle after the bin is resolved, the shifting operation can be moved to the *next* cycle where it is done in parallel with the rLPS look up.

4.3 Syntax Element Partitions Processing

The CABAC engines are mapped to partition engines which are used to decode the bins of the five different SEP in parallel. This section describes how the CABAC engine is decomposed into five partition engines, and how these partition engines are connected with FIFOs to form a slice engine as shown in Fig. 4-15.

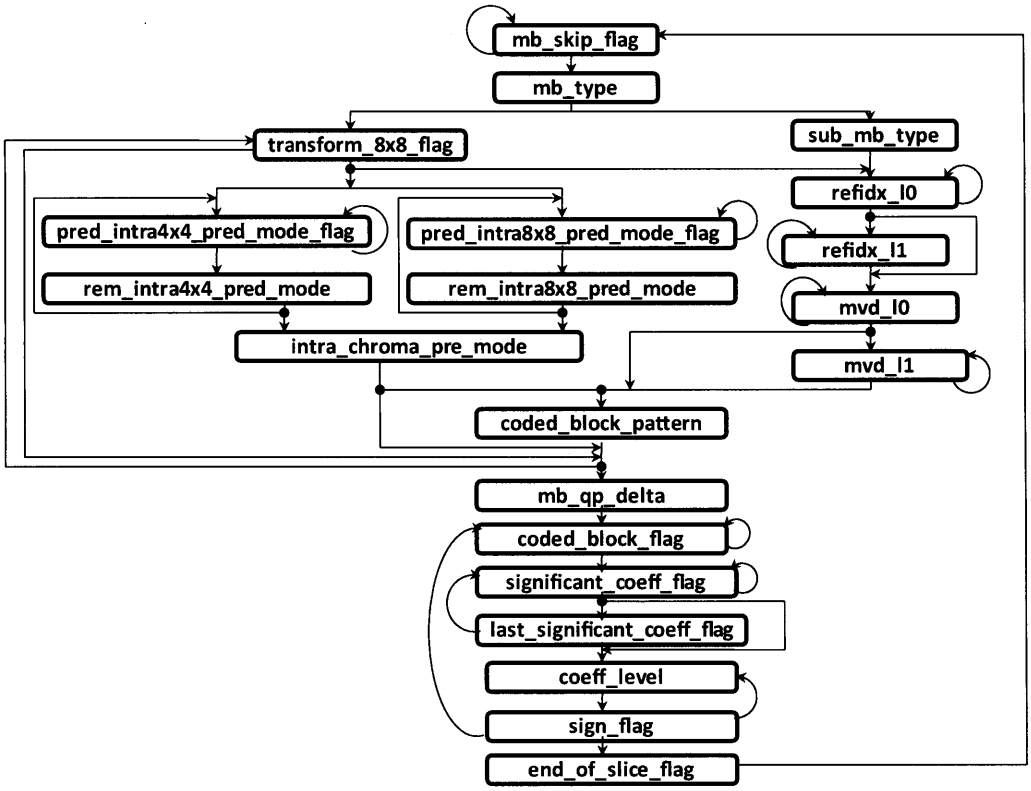


Figure 4-16: Context selection FSM.

4.3.1 Division of the Context Selection FSM

The context selection FSM is broken up into five different partition FSMs, one for each SEP. Each partition FSM is assigned to a different partition engine. Fig. 4-16 and Fig. 4-17 show how the FSM is divided amongst the five partitions. In Fig. 4-17, the states of the FSM are color-coded to indicate how they are mapped to the different partition engines in Fig. 4-15 (MBINFO - red, PRED - purple, CBP - orange, SIGMAP - green, COEFF - blue). The remaining connections between the partition FSMs were highlighted in the previous chapter in Fig. 3-8. These dependencies can be managed using SEP FIFOs between each of the partition FSM. Note that the partition FSMs can be recombined in a hierarchal manner resulting in an architecture that can easily be reconfigured to be backward compatible to support H.264/AVC CABAC.

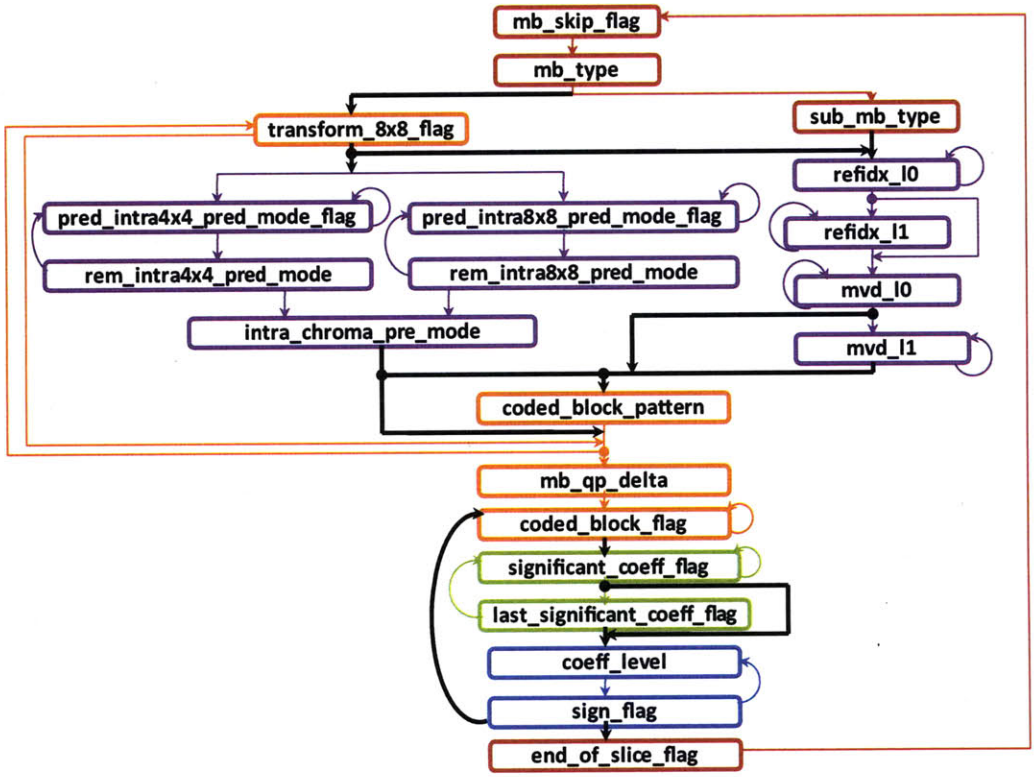


Figure 4-17: FSM for SEP. The states of the FSM in Fig. 4-17 are color-coded to map to the different partition engines in Fig. 4-15 (MBINFO - red, PRED - purple, CBP - orange, SIGMAP - green, COEFF - blue).

Table 4.1: Context memory Sizes

Partition	Number of Contexts
MBINFO	37
PRED (inter)	20
PRED (intra)	6
CBP	39
SIGMAP	146
COEFF	59
total	**307**

4.3.2 Distributed Context Memory

The context memory is broken up into smaller memories for each partition as shown in Table 4.1. Each of these memories is allocated to one of the five partition engines. Due to the small size of the distributed memories, registers instead of SRAMs were used to store the context states, and muxes were used to implement the ports. This increased the overall context memory area by approximately 2x. The smaller memories reduce the read access time (the worst case is a mux of 146:1 rather than 307:1). However, the area increases by around 30% due to the additional ports. This difference in area cost was factored into the estimated context memory area of the H.264/AVC CABAC engine in Fig. 3-10.

4.3.3 SEP FIFOs

As mentioned earlier, the different partitions have dependencies and can be synchronized using FIFOs. We refer to the group of FIFOs at the input of each partition as their SEP FIFO. For instance, the SEP FIFO for the COEFF partition transfers the necessary information from SIGMAP (e.g. number of coefficients), such that it can decode `coeff_abs_level_minus1` and `coeff_sign_flag`. Table 4.2 lists the memory size and number of read/write ports of each SEP FIFO. The data stored in these FIFOs is described in more detail in Appendix D. Each SEP FIFO can store up to 4 macroblocks worth of data. Increasing the FIFO depth can improve the throughput of the slice engine by up to 10% at

Table 4.2: SEP FIFO sizes

Partition	Size (bits)	Number of ports
MBINFO	28	1
PRED	2276	13
CBP	60	9
SIGMAP	20	3
COEFF	48	3
total	**2432**	**29**

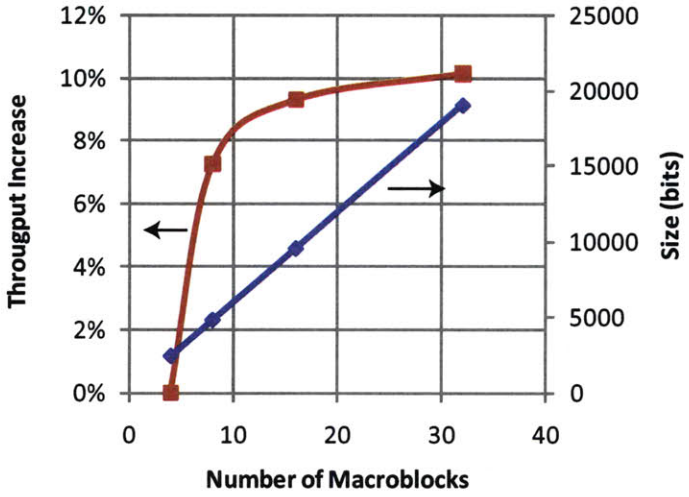


Figure 4-18: Tradeoffs between depth of SEP FIFOs in macroblocks versus improvement in slice engine throughput and memory size. Throughput results are averaged across parkrun and bigships sequences with various QP and prediction structures.

the cost of increased memory size. Fig. 4-18 shows the simulated results of this tradeoff.

When a partition's SEP FIFO is empty, its engine will stall. Custom clock gating was used to reduce clock power in the stalled engine.

4.4 Interleaved Entropy Slices Processing

The slice engines can be combined to process several IES in a given frame. Fig. 4-19 shows how the slice engines are connected using FIFOs.

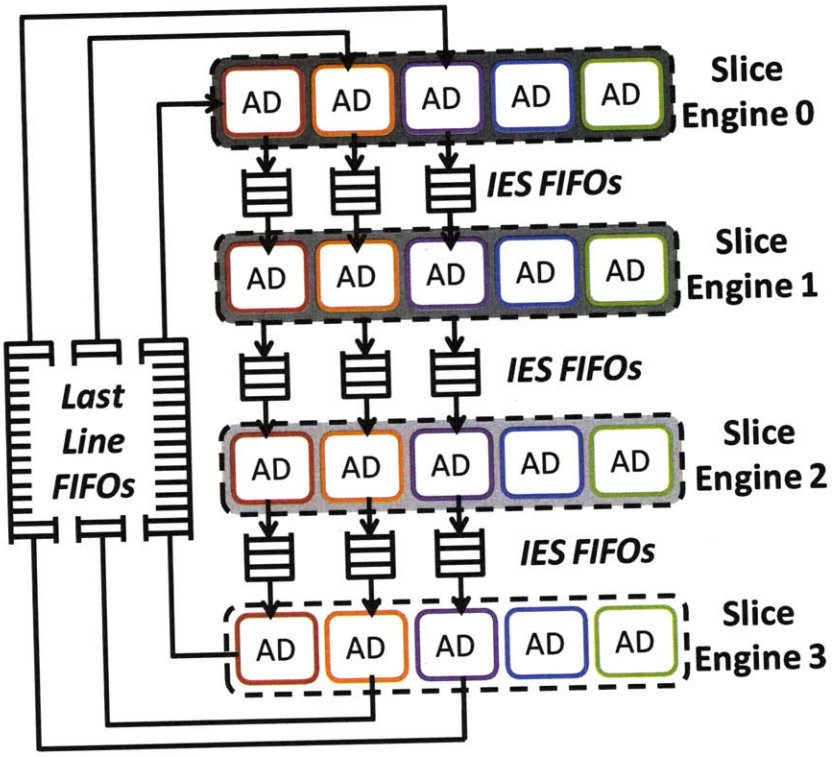


Figure 4-19: The slices engines are connected using FIFOs to process IES in parallel.

4.4.1 IES FIFOs

The slice engines are used to process the IES shown in Fig. 4-20. Due to the last line dependencies, slice 0 must always be one or two macroblocks ahead of slice 1. Similarly, slice 1 must be ahead of slice 2, and slice 2 must be ahead of slice 3. The slice engines are synchronized using IES FIFOs with depths of only a few macroblocks. The last line buffer needs to be inserted between slice engine 0 and 3, such that slice engine 0 can access the row above after wrapping to the next line as shown in Fig. 4-20. The last line buffer can be implemented using a FIFO since macroblocks are processed in a sequential order within a slice.

Three separate FIFOs are used for each slice engine to support the last line dependencies of partitions MBINFO, PRED and CBP. As a result, each partition engine within the slice engine can process a different macroblock. Table 4.3 shows the data and bit-width of each of the IES FIFOs. The bit-width was reduced by half using the mvd context selection

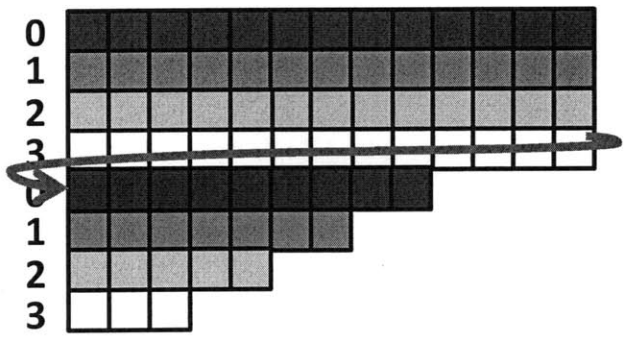


Figure 4-20: Last line buffer is inserted between slice engine 0 and 3, so that slice engine 0 can access the row above, which was decoded by slice engine 3, after wrapping to the next line.

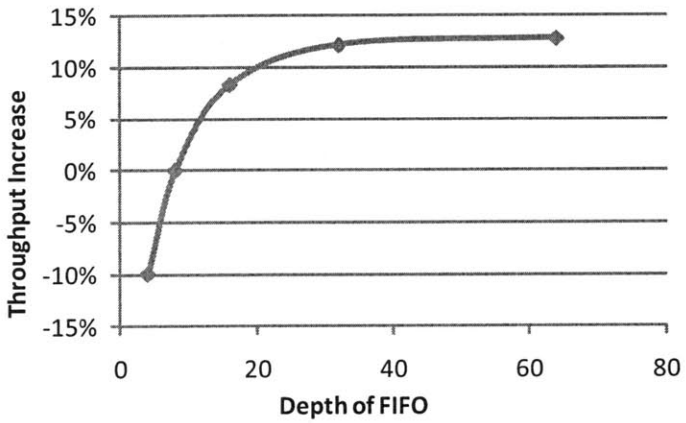


Figure 4-21: Tradeoff between depth of IES FIFOs in macroblocks versus improvement in throughput based on simulation of bigships QP=27, IBBP, IES=8.

optimization described in Section 4.2.1. When a FIFO is empty, the slice engine will stall. Custom clock gating was used to reduce clock power in the stalled slice engines.

All IES FIFOs have a depth of 8 macroblocks. The throughput of the connected slice engines can be improved by increasing the FIFO depth as shown in Fig. 4-21. A depth of 8 provides a 10% throughput increase over a depth of 4. A throughput increase of an additional 13% can be achieved by increasing the depth to 64. Each additional macroblock requires 40-bits. Note that increasing the FIFO depth also increases the critical path since the delay through the read mux increases as well.

Table 4.3: IES FIFO sizes

partition	info	width
MBINFO	mb_type	1
	mb_skip_flag	1
	total	**2**
PRED	refIdxL0	2
	refIdxL1	2
	mvdL0_h	4
	mvdL0_v	4
	mvdL1_h	4
	mvdL1_v	4
	mb_skip_flag	1
	total	**21**
CBP	cb_flag_luma_dc	1
	cb_flag_luma_ac	4
	cb_flag_chroma_dc	2
	cb_flag_chroma_ac	4
	coded_block_patter_luma	2
	coded_block_patter_chroma	2
	transform_8x8_mode_flag	1
	Intra_16x16	1
	total	**17**
	total	**40**

4.4.2 Row Balancing

The throughput depends strongly on how well the workload is balanced. As mentioned in Section 3.3.3, one of the benefits of IES is that it offers better workload balance than entropy slices. However, IES can still have imbalanced workload when the total number of macroblock rows in a frame is not a multiple of the number of IES. This causes some slice engines to be idle when the last few rows of the frame are being processed.

To address this, the row assigned to each IES rotates for every frame as shown in Fig. 4-22. The theoretical throughput improvement can be calculated as follows:

$$\text{max rows per IES} = \left\lceil \frac{R_{frame}}{N_{IES}} \right\rceil \tag{4.1}$$

$$N_{idle} = R_{frame} \,\%\, N_{IES} \tag{4.2}$$

$$\text{potential loss in throughput} = \frac{N_{idle}}{\text{max rows per IES} \times N_{IES}} \tag{4.3}$$

$$\text{throughput improvement} = 1 - \frac{1}{\text{potential loss in throughput}} \tag{4.4}$$

where N_{IES} is the number of IES, N_{idle} is the number of idle slice engines during processing of last rows without row balancing, and R_{frame} is the number of macroblock rows in the frame. From the above equations, we can see that the throughput improvement of row balancing depends on the resolution of the frame (i.e. R_{frame}).

Table 4.4 shows the calculated throughput improvement for different HD resolutions and number of IES per frame. Up to a 17% improvement can be achieved using the row balancing technique.

4.5 Simulation Results

The MP-CABAC architecture was implemented in RTL with 16 slice engines for a total of 80 partition engines. Each slice engine contains a distributed FSM for context selection and five performance optimized binary arithmetic decoders. A depth of 4 was used for the SEP

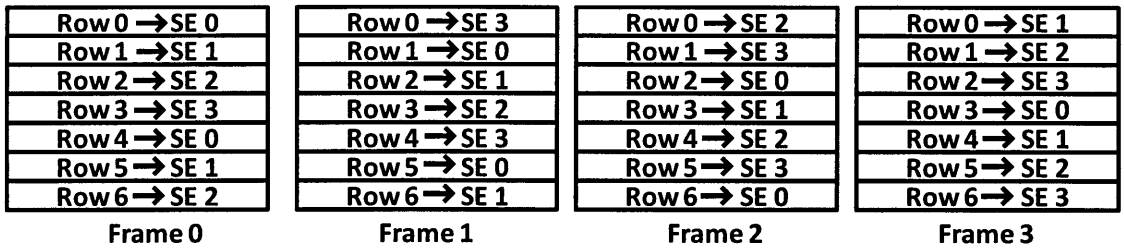


Figure 4-22: Rotate slice engine that process first row in frame to prevent idle slice engines when the number of rows in frame is not a multiple of the number of IES.

Table 4.4: Throughput improvement from row balancing.

Number of IES	Throughput Improvement (%)		
	720p[1280x720]	1080p[1920x1088]	4kx2k[4096x2160]
2	2.22	0	0.74
4	6.67	0	0.74
8	6.67	5.88	0.74
16	6.67	17.65	6.67

FIFOs, while a depth of 8 was used for IES FIFOs.

4.5.1 Throughput

Fig. 4-23 shows how the number of decoded bins changes from cycle to cycle. The number of decoded bins per cycle varies based on the workload balance and dependencies across the engines.

The workload distribution across partitions is affected by the sequence, QP and prediction structures. Fig. 4-24 shows the bin per cycle distribution across different sequences, QP and prediction structures. The average bin per cycle ranges between 2.12 to 2.52 across different sequences (Table 4.5), 2.13 to 2.58 across prediction structures (Table 4.6) and 2.08 to 2.60 across QP (Table 4.7).

The throughput can be increased by using more IES. As we increase the number of IES from 1 to 16, the number of bins per cycle increases from 2.52 to 23.72 and 2.16 to 24.50 for bigships and rally respectively, as shown in Table 4.8 and Table 4.9. For 16 IES per frame, IBBP and QP=27, MP-CABAC decodes an average of 24.11 bins per cycle. This drops to

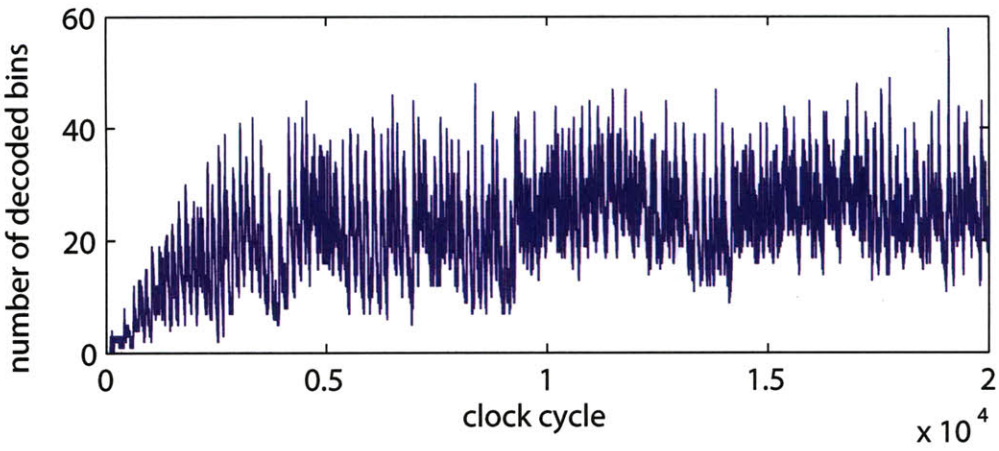


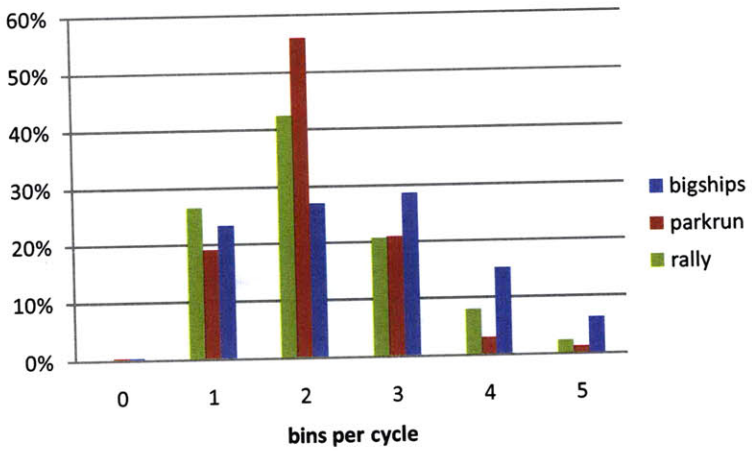
Figure 4-23: Bins per cycle variation for first 20,000 decoding cycles of bigships, QP=27, IES=16.

Table 4.5: Bins per cycle for different sequences

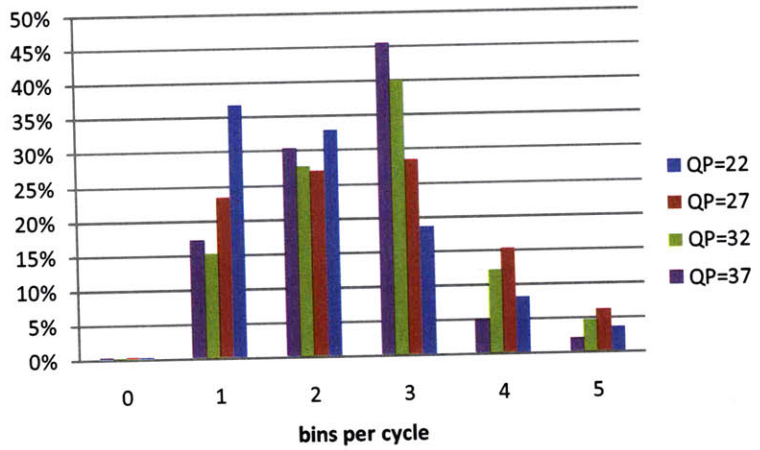
Sequence	**Resolution**	**Prediction Structure**	**Bit-rate for 30fps [Mbits/s]**	**QP**	**Number of IES**	**Number of frames**	**Average bins/cycle**
bigships	1280×720	IBBP	0.42	27	1	297	**2.52**
parkrun	1280×720	IBBP	1.76	27	1	69	**2.12**
rally	4096×2160	IBBP	9.04	27	1	234	**2.16**

Table 4.6: Bins per cycle for different prediction modes

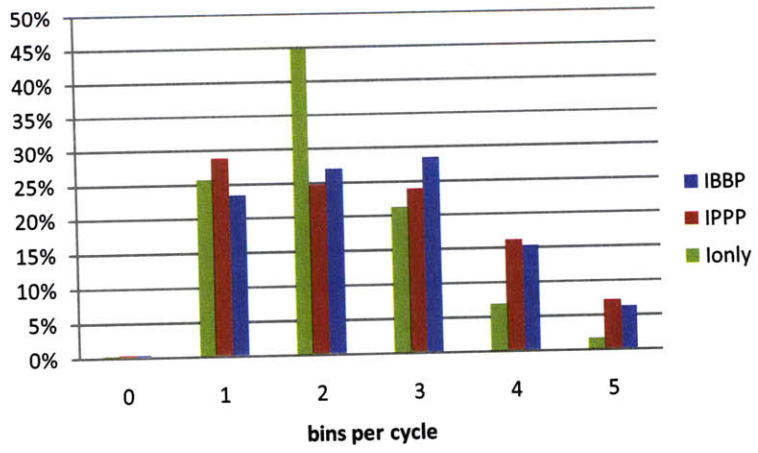
Sequence	**Resolution**	**Prediction Structure**	**Bit-rate for 30fps [Mbits/s]**	**QP**	**Number of IES**	**Number of frames**	**Average bins/cycle**
bigships	1280×720	Ionly	2.26	27	1	99	**2.13**
bigships	1280×720	IPPP	0.40	27	1	99	**2.44**
bigships	1280×720	IBBP	0.28	27	1	99	**2.58**



(a) For various sequences (QP=27, IBBP, IES=1)



(b) For various QP (bigships, IBBP, IES=1)



(c) For various prediction mode (bigships, QP=27, IES=1)

Figure 4-24: Bins per cycle distributions for different sequences, QP and prediction structures.

Table 4.7: Bins per cycle for different QP

Sequence	Resolution	Prediction Structure	Bit-rate for 30fps [Mbits/s]	QP	Number of IES	Number of frames	Average bins/cycle
bigships	1280×720	IBBP	1.16	22	1	297	**2.08**
bigships	1280×720	IBBP	0.42	27	1	297	**2.52**
bigships	1280×720	IBBP	0.18	32	1	297	**2.60**
bigships	1280×720	IBBP	0.09	37	1	297	**2.35**

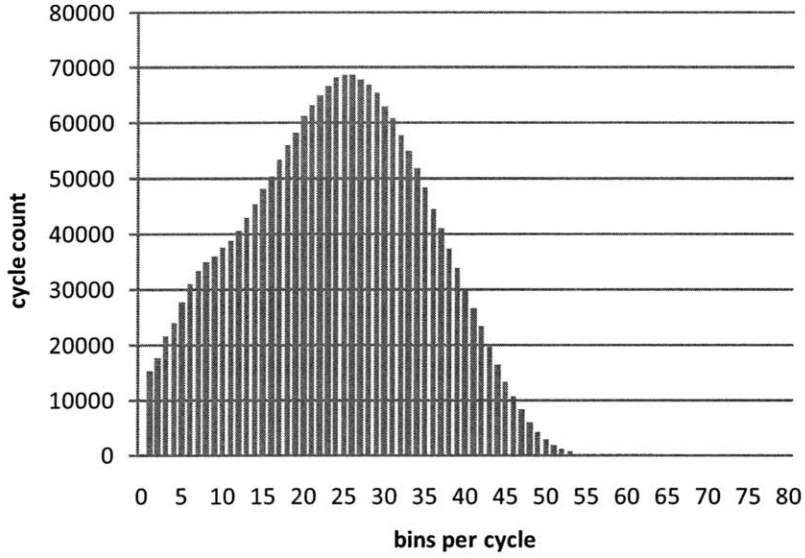
Table 4.8: Bins per cycle for different number of IES for bigships

Sequence	Resolution	Prediction Mode	Bit-rate for 30fps	QP	number of IES	number of frames	average bins/cycle
bigships	1280×720	IBBP	0.42	27	1	297	**2.52**
bigships	1280×720	IBBP	0.43	27	2	297	**4.44**
bigships	1280×720	IBBP	0.43	27	4	297	**7.78**
bigships	1280×720	IBBP	0.44	27	8	297	**13.81**
bigships	1280×720	IBBP	0.45	27	16	297	**23.72**

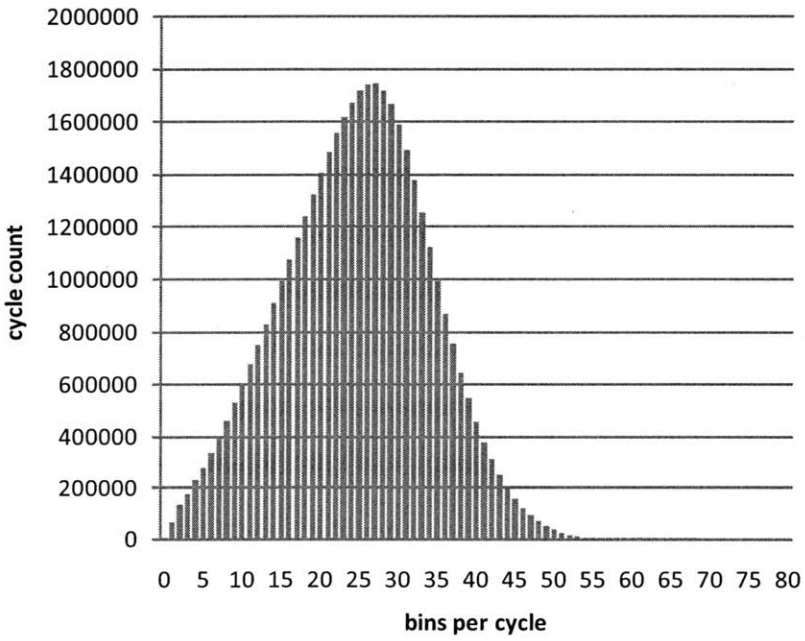
21.28 bins per cycle in the worst case frame of the sequence(i.e. the frame that takes the maximum number of cycles to decode). The distribution of the bins per cycle are shown for bigships and rally in Fig. 4-25.

4.5.2 Area Breakdown

The MP-CABAC with 16 slice engines has a gate count of 2,502 kgates after synthesis. This includes the interface logic and input/output FIFOs that will be discussed in the next chapter. The context memory and last line FIFO account for 499 and 104 kgates respectively. Each slice engine has a gate count of 101 kgates. The area breakdown of the MP-CABAC is shown in Fig. 4-26. Fig. 4-26a shows the area breakdown between the slice engines, the last line and IES FIFOs, the interface logic and input/output FIFOs. Fig. 4-26b shows the area breakdown within the slice engine. The data flow control includes logic that computes the macroblock position of the partitions, determines when to read and write from the SEP



(a) sequence bigships, 720p, QP=27, IES=16 (average 23.72 bins per cycle)



(b) sequence rally, 4kx2k, QP=27, IES=16 (average 24.50 bins per cycle)

Figure 4-25: Bins per cycle distributions for 16 IES per frame.

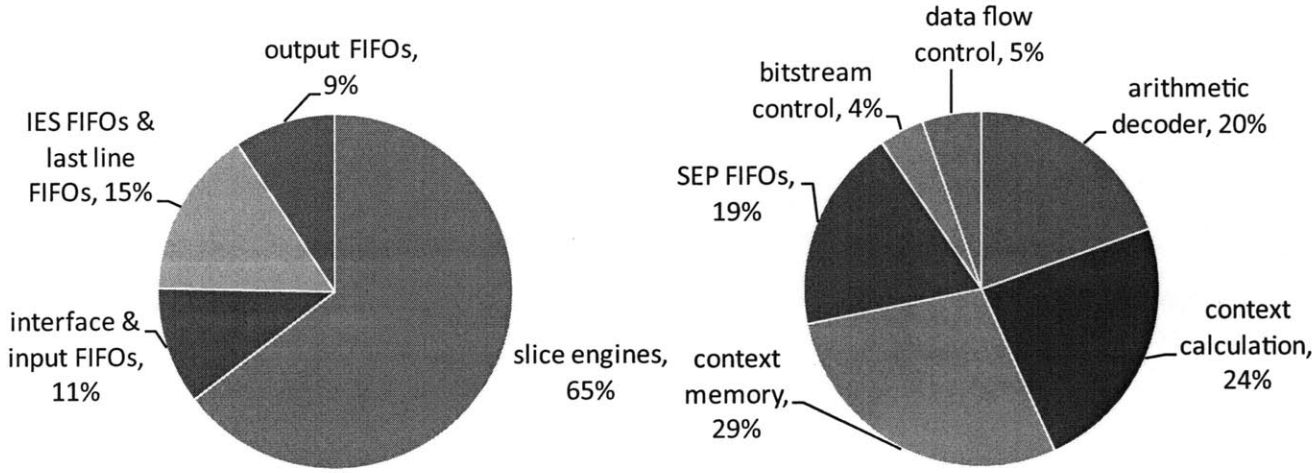
Table 4.9: Bins per cycle for different number of IES for rally

Sequence	Resolution	Prediction Mode	Bit-rate for 30fps	QP	number of IES	number of frames	average bins/cycle
rally	4096×2160	IBBP	9.04	27	1	234	**2.16**
rally	4096×2160	IBBP	9.04	27	2	234	**4.03**
rally	4096×2160	IBBP	9.05	27	4	234	**7.42**
rally	4096×2160	IBBP	9.07	27	8	234	**13.52**
rally	4096×2160	IBBP	9.10	27	16	234	**24.50**

and IES FIFOs and when engines should stall. Fig. 4-26c shows the area breakdown across different partitions. The PRED and SIGMAP partitions consume a significant portion of the area. The SIGMAP engine area is dominated by the 146 entry context memory. The PRED engine is dominates by the context selection logic.

4.5.3 Comparison

Table 4.10 shows a comparison of the MP-CABAC with previously published results for H.264/AVC CABAC. With the exception of [60] which was measured, all results were reported based on synthesis. Accordingly, the maximum frequency from synthesis as well as after place and route are reported for the MP-CABAC. Results were simulated at 1.2V in the nominal process corner. The MP-CABAC provides well over an order of magnitude increase in bin-rate. The majority of these H.264/AVC CABAC engines exploit bin parallelism, which typically has a lower area overhead than slice parallelism. To compare this overhead, or measure how efficiently we use the additional gates, we divide the area by by the bin-rate. The area efficiency for the H.264/AVC CABAC implementations ranges from 0.09 to 1.42 kgates per Mbins/s . The area increase of the MP-CABAC relative to the increase in bin-rate is within the same order of magnitude as bin parallelism with an area efficiency of 0.36 kgates per Mbins/s using synthesis result. Note that the reported MP-CABAC area in Table 4.10 includes the interface logic and FIFOs. If only the slice engine area is used, then the MP-CABAC has an area efficiency of 0.11 kgates per Mbins/s. The MP-CABAC



(a) Breakdown for entire MP-CABAC

(b) Breakdown within a slice engine

(c) Breakdown across partitions

Figure 4-26: Post layout area breakdown of MP-CABAC.

can be combined with the bin parallelism approaches used in the other H.264/AVC CABAC implementations for additional throughput improvement. To a first order, the resulting throughput will be the product of their average bins per cycle.

4.6 Summary and Conclusions

This chapter presents performance, area and power optimizations in the MP-CABAC architecture. To increase the performance, we pipelined the CABAC engine to reduce the critical path by 40%. Several optimizations in the binary arithmetic decoder including range comparison reordering, leading zero LUT, early range shifting and next cycle offset renormalization resulted in an additional 22% critical path reduction. A row balancing technique was used avoid idle slice engines regardless of the number of rows in a frame.

To reduce area costs, we modified the context index computation for mvd such that the size of the last line buffer was reduced by 50%. SEP and IES FIFOs were used to manage dependencies and synchronize the parallel engines. Tradeoffs between area and throughput for these FIFOs were evaluated. Custom clock gating was used to reduce the clock power of engines that stalled when their input SEP or IES FIFO was empty.

Finally, this chapter concludes by presenting the RTL simulation and synthesis results for the MP-CABAC architecture. The performance of the architecture was characterized across different sequences, prediction structures, QP and number of IES. It was shown that the MP-CABAC can deliver over 10x increase in performance when compared with H.264/AVC CABAC implementations when 16 IES are used, which results in a coding penalty of 4.9% at QP=27 for the bigships sequence. The area breakdown and area efficiency of the MP-CABAC was also discussed.

The range comparison reordering and modified mvd context selection are key examples of how minor changes identified during co-optimization of architecture and algorithm can make a significant impact in increasing performance and reducing area cost.

Table 4.10: Comparison of results with other published H.264/AVC CABAC implementations. Most results were obtained from synthesis. LL=last line buffer. *calculated based on values given in the paper. **measured result.

Publication	**Peak bins/ cycle**	**Avg. bins/ cycle**	**Maximum Frequency (MHz)**	**Bin-rate (Mbins/s)**	**Gate Count (kgates) [w/ memory]**	**Memory (bytes)**	**Proc. (nm)**
Yu, TCE, 2005 [71]	2(reg)+ 1(bypass)	N/A	149	N/A	39.9*	420	180
Kim, ISCAS, 2006 [80]	2(reg)	0.41	303	124	N/A	N/A	180
Yang, ICME, 2006 [81]	2(reg)	1.24	120	149	37.0[83.2]	504	180
Yi, TCSVT, 2007 [82]	0.25	0.25	225	57	81.2	662+ 11,520 (LL)	180
Shi, ICCSC, 2008 [83]	1(reg) or 2(bypass)	1.27	200	254	29.0	10,810 (includes LL)	180
Yang, TCSVT, 2009 [84]	2(reg) or 4(bypass)	0.86	105	90	35.0[76.3]	441	180
Zhang, TVLSI, 2009 [85]	16(reg)	2.27*	45	102	42.0	349	180
Chen, TCE, 2009 [86]	2(reg)	1.32	238	314	43.6	1,033	180
Liao, ISCAS, 2010 [87]	2(reg)	1.83*	267	483	42.4	N/A	90
Chuang, ISSCC, 2010** [60]	2(reg)	1.95	210	410	N/A	N/A	90
This Work	**80(reg)**	**24.11**	**307 (syn) 200 (PR)**	**7,398 4,822**	**1,900.1 [2,502.5]**	**4,298+ 10,240 (LL)**	**65**

Chapter 5

Massively Parallel CABAC Test Chip

This chapter describes the implementation of the test chip used to evaluate the power trade-offs of the MP-CABAC algorithm and architecture described in the previous two chapters. In particular, it will address the challenges faced with getting data on- and off-chip for highly parallel architectures. The test setup will be discussed and the measured results of the MP-CABAC test chip will be presented.

5.1 Functionality

The top level architecture of the MP-CABAC test chip is shown in Fig. 5-1. The MP-CABAC test chip supports 5 syntax element partitions with up to 16 interleaved entropy slices; accordingly, the test chip contains a total of 80 partition engines. Bin parallelism and adaptive QP allocation were not implemented in the test chip. The partition engines are based on the 1 bin per cycle CABAC engine described in the previous chapter. Thus, the test chip can achieve a peak bins per cycle of 80. The last line buffer is sized to support up to 4kx2k resolution (4096 horizontal pixels).

The key functions in the MP-CABAC test chip include reading encoded bits, distributing these bits to the parallel engines, decoding the bits to bins, and writing out the decoded bins. Note that the bitstream parser is not included on the test chip. There are several

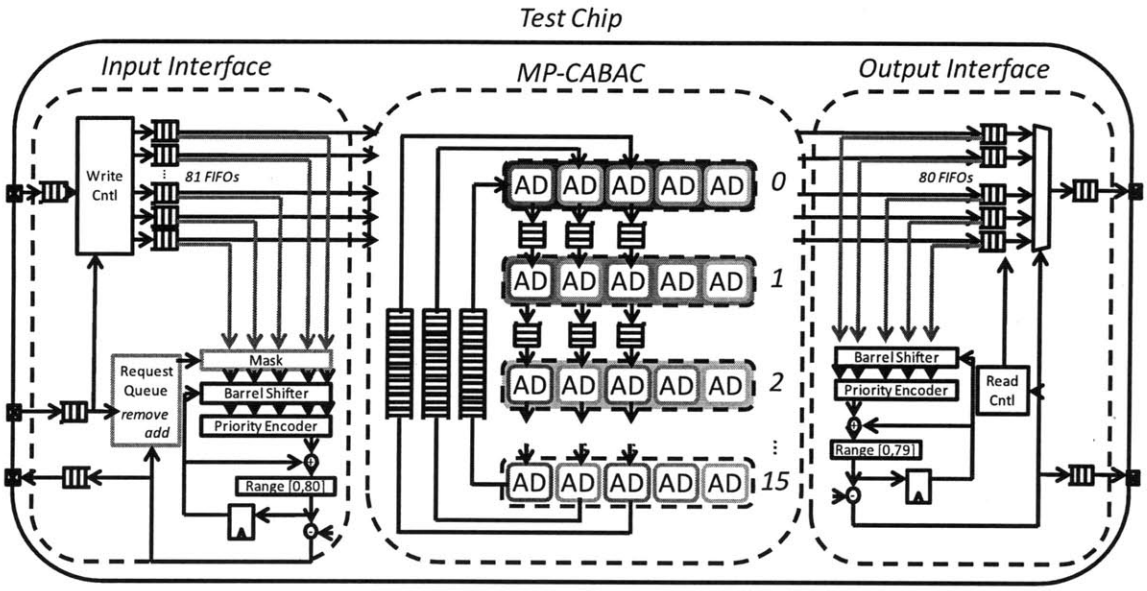


Figure 5-1: Top level architecture of MP-CABAC test chip. MP-CABAC is implemented with 16 slice engines.

practical issues that must be addressed in the test chip implementation. First, interface logic is required to share the limited I/O across the 80 parallel engines. Second, this interface logic should be placed on a separate voltage/frequency domain to limit its impact on the power and performance characterization of the MP-CABAC. Finally, the test chip needs to be reconfigurable in order to support a variable number of IES. These challenges as well as other implementations consideration will be addressed in the following sections.

5.2 Off-chip Interface

This section describes the interface logic used to move the data in and out of the test chip to feed all 80 partition engines. The number of I/O pads on the test chip is limited by the perimeter, and consequently area, of the chip. As a result, each engine cannot be directly connected to the I/O of the chip. The I/O can be shared amongst the engines using a round robin strategy. Round robin involves checking whether the engines are ready to read encoded bits (or write decoded bins) in a sequential manner. If engine N receives (or transmits) data in the current cycle, then in the next cycle we start checking beginning with engine N+1. In

other words, the engine that most recently made a transfer has the lowest priority. Separate round robins are used for reading encoded bits and writing decoded bins. Since the engines need to wait to transfer data to and from the I/O pads, FIFOs are used to buffer their inputs and outputs. Both input and output FIFOs have a depth of 4.

Fig. 5-2 shows the architecture of the output interface. Its goal is to prevent the output FIFOs from becoming full. As soon as there is any data in any of the FIFOs, it should try to write it off-chip. This is achieved using the following steps:

1. The empty signals of the output FIFOs in all 80 engines are monitored.
2. A barrel shifter shifts the empty signals by *ptr*. *ptr* indicates which position to start checking from.
3. The priority encoder searches for the next *not* empty. *incr_ptr* indicates its position relative to *ptr*.
4. *next_ptr* is computed for the next cycle. The *ptr* is wrapped in the range of 0 to 79.
5. The FIFO's index is computed. A read enable signal is sent to that FIFO and its output is muxed to the test chip's output FIFO.

Fig. 5-3 shows the architecture of the input interface. The round robin scheme is the same as the output interface except for several key differences. First, its goal is to prevent the input FIFOs from becoming *empty* by monitoring their full signals. This is done for the input FIFOs to all 80 engines as well as the slice info FIFO. When a FIFO is *not* full, the FIFO's index is sent off-chip to request for data to feed that FIFO. The full signal remains low during the several cycles needed for the requested data to reach the chip. However, if several requests are made for the same FIFO and that FIFO only has one available slot, then deadlock will occur since the FIFO cannot hold any additional requested data, and this additional data blocks other requested data from reaching their destination. To prevent deadlock from occurring, each FIFO is limited to one request at a time. The FIFO's index is placed in a queue when it issues a request; when the requested data returns to the chip, the index is removed from the queue. This enables the chip to keep track of which FIFOs have issued requests that have already gone out, and ignore any subsequent requests from

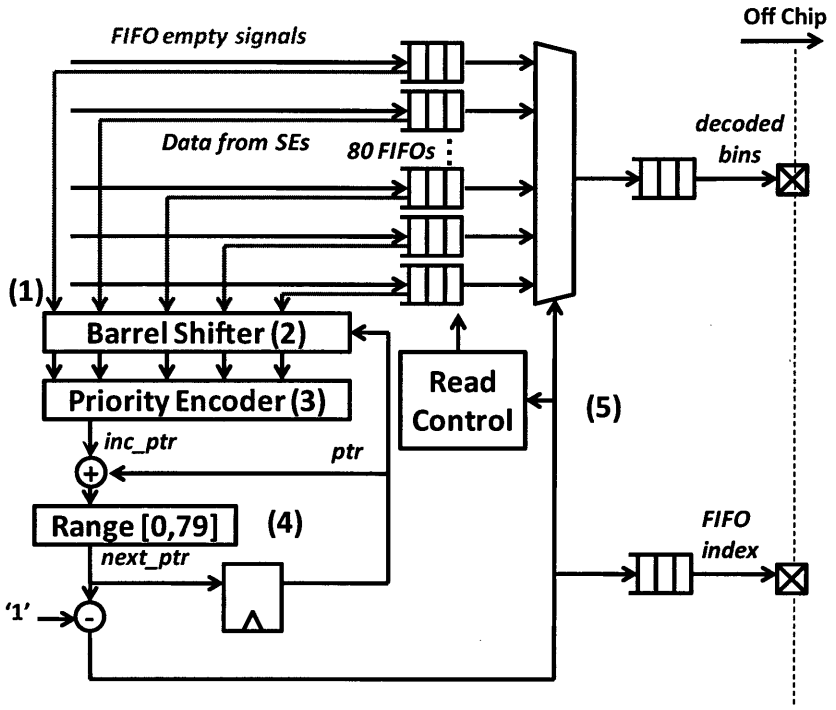


Figure 5-2: Round robin control for output of MP-CABAC.

those FIFOs until the requested data has returned.

The input logic steps are summarized as follows:

1. The full signals of the input FIFOs in all 80 engines as well as the shared slice info FIFO are monitored.
2. Check which input FIFOs are in the request queue and ignore (mask) their full signals.
3. A barrel shifter shifts the full signals by *ptr*. *ptr* indicates which position to start checking from.
4. The priority encoder searches for the next *not* full. *incr_ptr* indicates its position relative to *ptr*.
5. *next_ptr* is computed for the next cycle. The *ptr* is wrapped in the range of 0 to 80.
6. The FIFO's index is computed and sent to the request queue. The requested index is also sent off-chip.
7. The requested data (encoded bits) is written to the input FIFO with the matching index. The index is removed from the request queue.

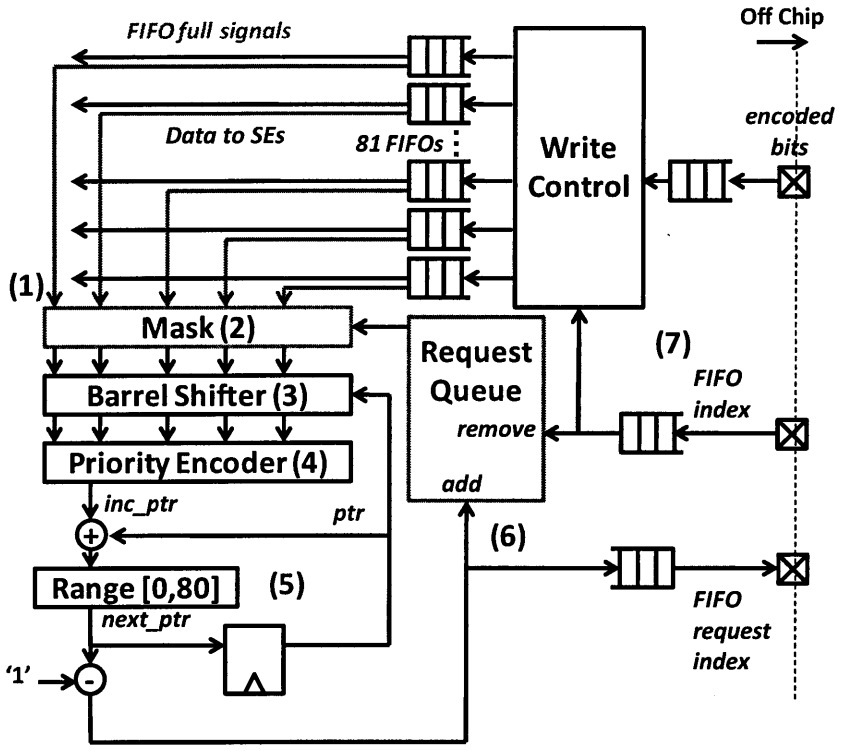


Figure 5-3: Round robin control for input of MP-CABAC.

Based on the number of I/O pads that can fit on the perimeter of the chip, both input and output FIFOs use a bit-width of 64. 7-bits are required for the FIFO index. Including reset, clocks, test control and status signals, the test chip has a total of 172 signal I/Os.

The round robin strategy only checks if a FIFO is full or empty. An alternative approach is to take the fullness (or emptiness) of the FIFOs into account and assign priority based on remaining available space. This can result in higher throughput, as the FIFOs are less likely to become full (empty). However, processing multiple bits per FIFO to assign priority would increase the complexity of the priority encoder, which may increase power consumption.

5.3 Bitstream Control

The arithmetic decoder reads in new bits during offset renormalization. A sliding window approach, shown in Fig. 5-4a, is used to read these bits from the input FIFO. To account for the case where the window straddles two entries, the input FIFO allows read access to

two entries at a time. During normal operation the number of new bits range from 0 to 7 per cycle. When a new slice occurs, the window byte aligns itself and reads in 9 new bits for the offset. The bitstream control architecture is shown in Fig. 5-4b.

5.4 Configurability

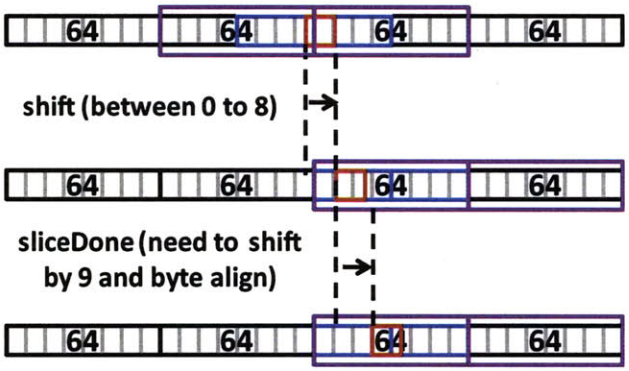
For scalability, the architecture needs to be configurable to support different numbers of IES. This involves muxing the input to the last line FIFOs as shown in Fig. 5-5. Custom clock gating was used to reduce the clock power of disabled slice engines.

5.5 Global Signals

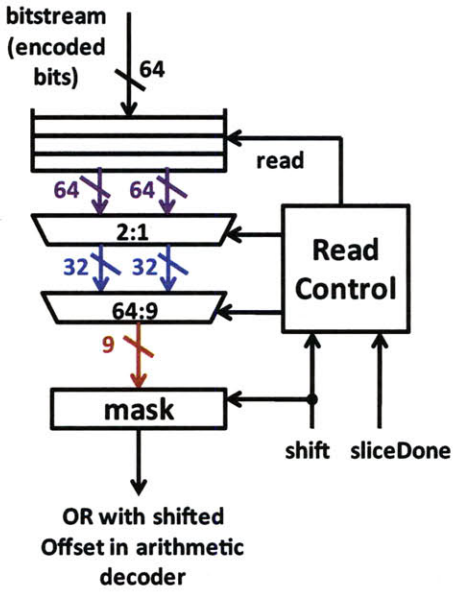
Global signals are routed to all slice engines in the test chip (Fig. 5-6). During initialization the SPS bits are loaded including `chroma_format_idc`, `direct_8x8_inference_flag`, `width_in_mbs`, and `height_in_mbs`. The `number_of_slice_engines` is also sent. The other global signals are loaded from the slice info FIFO. These signals include context initialization parameters (`cabac_init_idc`, `sliceQP` and `slice_type`), and slice header info (`transform_8x8_mode_flag`, `slice_type`, `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1`) used by the MBINFO engine. Whenever a frame is completed, new data is read from the slice info FIFO. At each new frame, the slice info FIFO sends a new set of parameters to the context initialization table where the initialization values of all 307 contexts are computed and sent to the slice engines.

5.6 Separate Voltage/Frequency Domains

Certain portions of the interface logic, particularly the movement of data off-chip (i.e. output interface logic), would not be required if the MP-CABAC was integrated into a full decoder chip. Consequently, separate voltage domains are used for the slice engines (i.e. core domain) and the interface logic (i.e. interface domain) for power measurements. Furthermore, the core



(a) Sliding window used to select new bit from the encoded bitstream.



(b) Architecture for bitstream control.

Figure 5-4: Bitstream control used to map new bits from input FIFOs to arithmetic decoder.

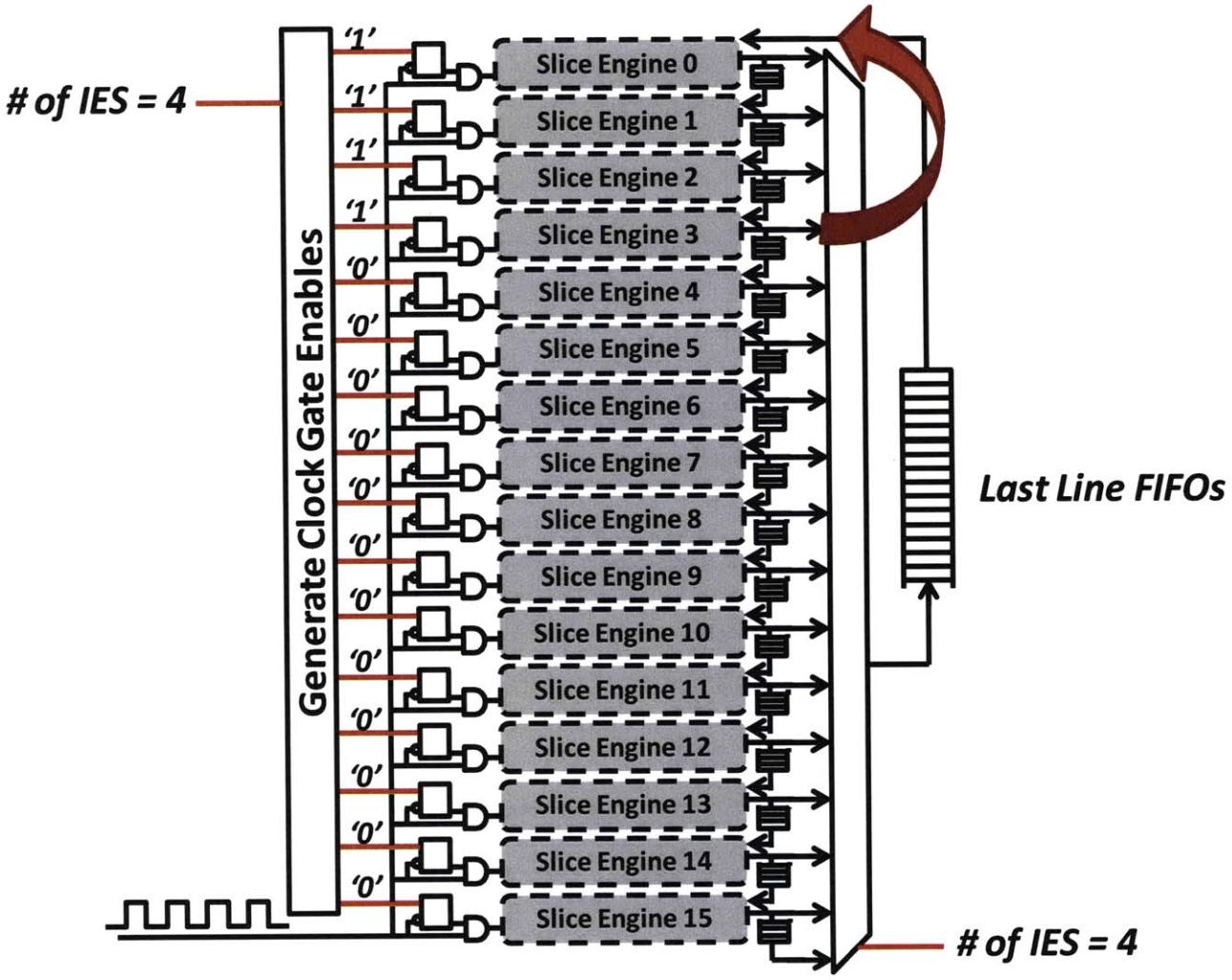


Figure 5-5: The architecture is configurable to support different number of IES. The disabled slice engines are clock gated. In the above example, the architecture can support up to 16 IES and is configured to process four.

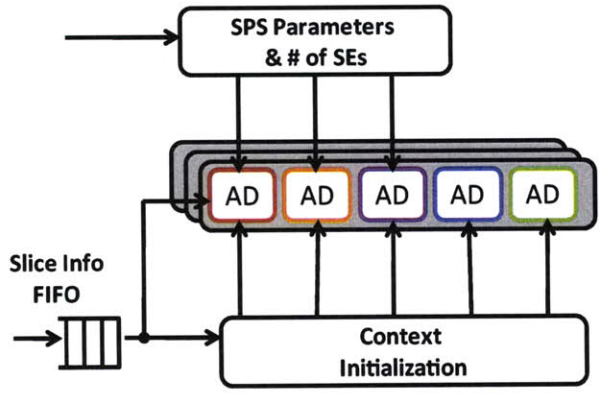
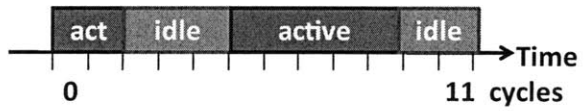
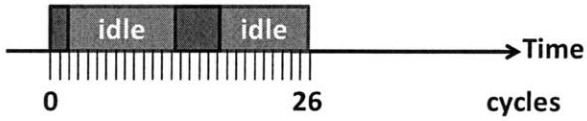


Figure 5-6: Global signals routed to all slice engines.



(a) Core and interface have the same frequency.



(b) Core has a 4x higher frequency of interface.

Figure 5-7: A illustration of how bins per cycle and bin-rate changes when core frequency is increased. The idle time remains the same regardless of core frequency since it is dictated by the interface bottleneck. However, active time is reduced by 4x since the time per cycle is reduced. Thus, at higher frequency the proportion of idle cycles to total cycles increases, which reduced the bins per cycle. However, the time to decode all bins decreases, so the overall bin-rate increases.

and interface domains are driven by separate clocks to prevent the off-chip interface from limiting the slice engine frequency. The interface frequency may be limited by the I/O pads and off-chip components (e.g. FPGA).

Fig. 5-7 illustrates how bins per cycle and bin-rate changes when core frequency is increased. Running the slice engines (core domain) at a higher frequency than the interface reduces the average decoded bins per core domain cycles. This is due to the fact that more idle cycles will occur when the input/output interface FIFOs become empty/full. For instance, when both domains run at the same frequency, if the input FIFOs are empty for one interface cycle, the engines are idle for one core cycle. If the core clock runs at 4x the frequency of the interface clock, then the engines are idle for four core cycles. Thus, total number of idle cycles increases. However, the time for each core cycle is shorter due to the high frequency. When the engines are not idle, they decode the bins at a faster rate, and thus the overall bin-rate still increases. The overall impact on bit-rate depending on the ratio between idle and active cycles in the original one frequency scenario. Simulation results in Fig. 5-8 show that the slice engines can run at a 4x higher frequency than the interface and achieve a 2.6x increase in bin-rate compared to the case where they use the same frequency.

Asynchronous FIFOs, described in [88], are used to connect the interface and core do-

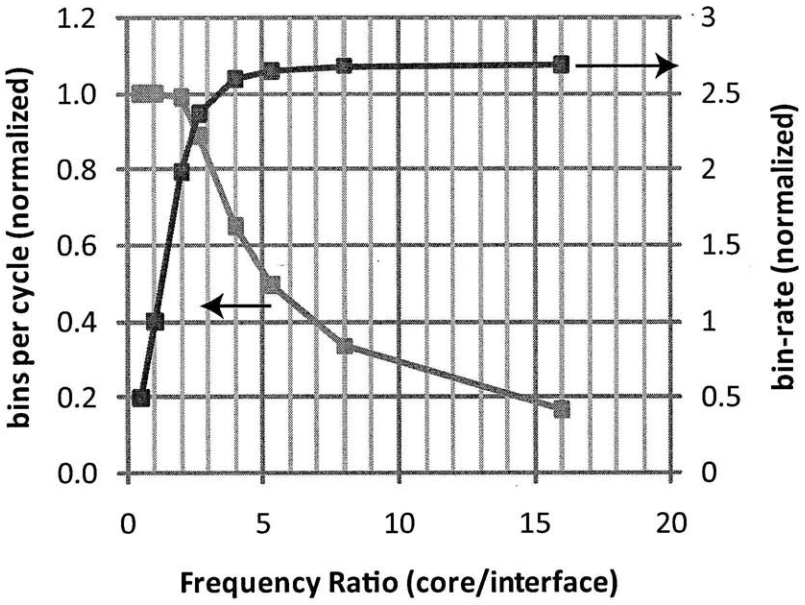


Figure 5-8: Impact of increasing the core frequency above the interface frequency on bin-rate and bins per cycle. (Simulated on bigships, IES=16, IBBP, QP=27)

mains. Fig. 5-9 shows the architecture of the asynchronous FIFO and how it is divided across voltage domains. Note that the FIFO memory is placed on the voltage domain of the write clock to prevent setup/hold time violations to the registers. Accordingly, the input FIFO memory is on the interface domain while the output FIFO memory is on the core domain. The input and output interface logic also run on separate clocks; however, since there is no direct connection between the input and output logic, no additional asynchronous FIFOs are required.

There are a total of 10,304 signals crossing the two voltage domains. Due to routing issues, it was decided that level converters would not be used between the domains. Consequently, the difference between the supply voltages should remain in the range of 100 to 200 mV.

5.7 Floorplan

The floorplan of the MP-CABAC is shown in Fig. 5-10. Three levels of hierarchy were used to place and route the design; going from bottom to top, the levels are slice engine, core

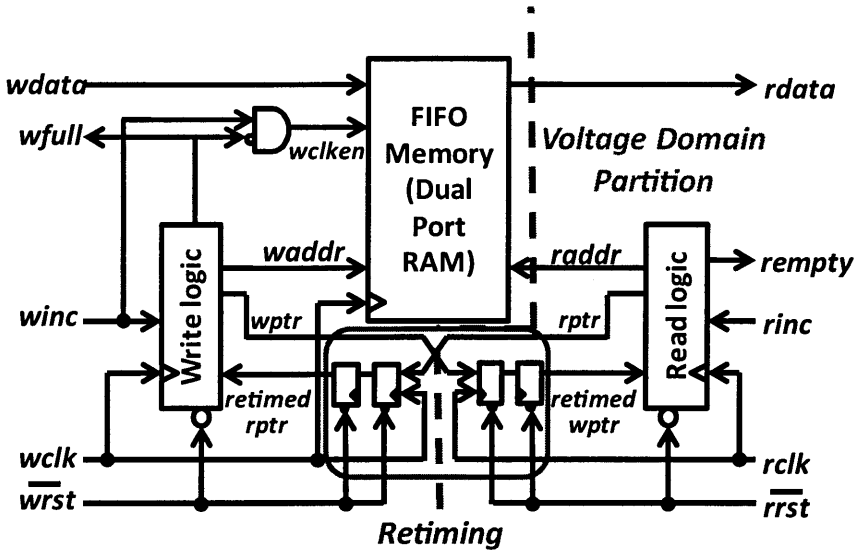


Figure 5-9: Asynchronous FIFO used between interface and core domain. This figure is adopted from [88].

domain and interface domain. Several steps were taken to ease routing congestion in the design. Routing in the slice engine was restricted to a subset of metal layers, such that an entire metal layer could be dedicated to the top level for connections between the engines. Gates for the the context initialization table and the last line FIFOs were placed in groups and allocated to specific regions within the core domain as shown in Fig. 5-10.

5.8 Test Setup

The MP-CABAC test chip is verified by feeding it various encoded bits and checking that the decoded bins were correct. For HD sequences, the encoded bits and decoded bins are on the order of several tens of megabytes, which requires them to be stored in an off-chip SDRAM. A Spartan-3 FPGA is used as an interface between the SDRAM and the test chip; both the FPGA and the SDRAM are located on the Opal Kelly XEM3050 board [89].

Fig. 5-11 shows the connections of the test setup. The test chip is connected to two Opal Kelly boards; one to provide the encoded bits (input) and the other to read and check the decoded bins (output). The architecture of the logic on the input FPGA and output FPGA are shown in Fig. 5-12 and Fig. 5-13 respectively. The two 16-bit-wide SDRAMs on

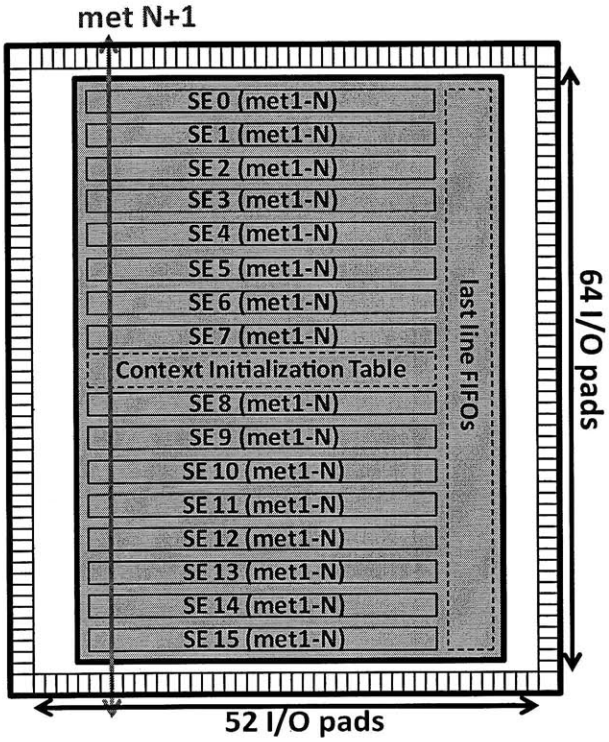


Figure 5-10: Floorplan for MP-CABAC test chip. Routing within the slice engine was restricted to layers 1-N, such that layer N could be dedicated to connecting the slice engines. Shaded area represents the core domain. White area represents the interface domain.

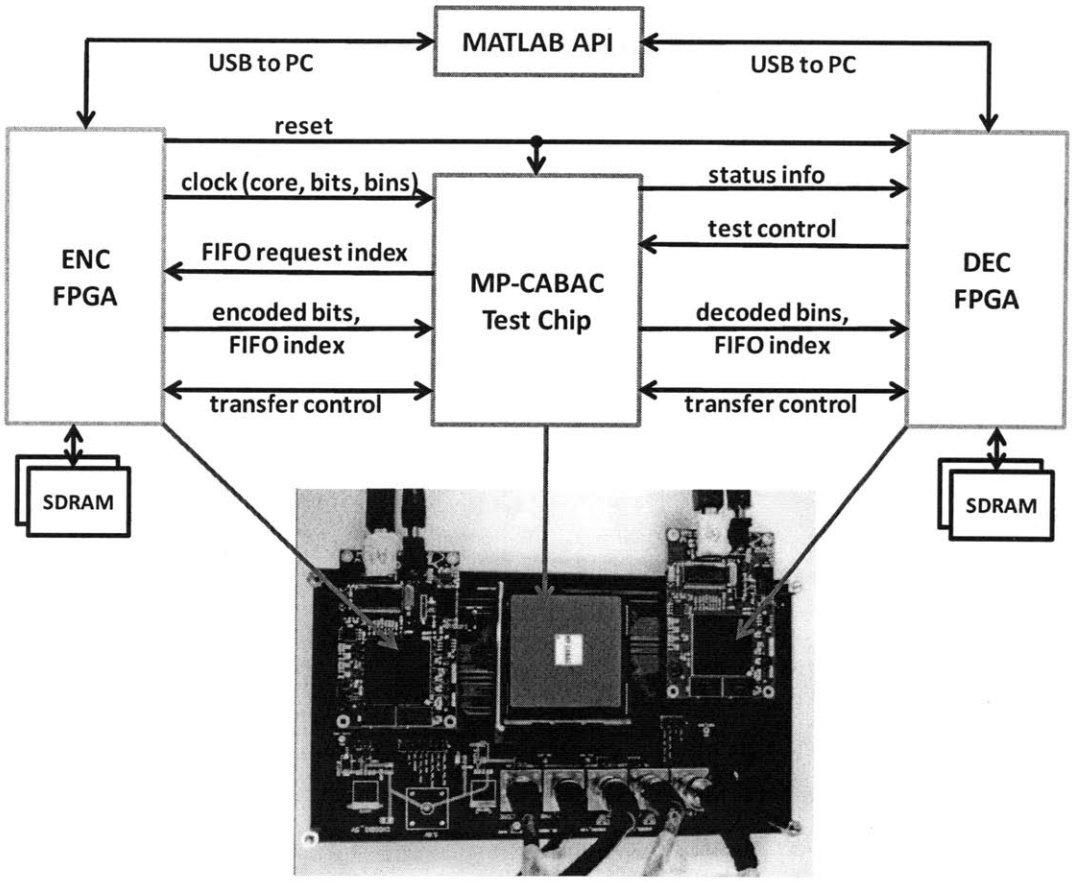


Figure 5-11: Test setup for MP-CABAC test chip.

the Opal Kelly boards are used as a single 32-bit-wide memory. Both the encoded bits (or decoded bins) and their corresponding FIFO indices are stored in the SDRAMs. The data is interleaved by storing four 7-bit FIFO indices in a 32-bit word followed by four 64-bit entries which are stored as eight 32-bits words in the SDRAM. Separate clocks are used for the SDRAM control logic and the rest of the FPGA (interface) logic; the SDRAM is clocked at a higher frequency since it deals in increments of 32-bits, while the rest of the FPGA operates on 64-bits.

The large number of I/O pads and the pad to chip area ratio significantly limited the packaging options. Details on the packaging challenges and the bonding diagram can be found in Appendix F.

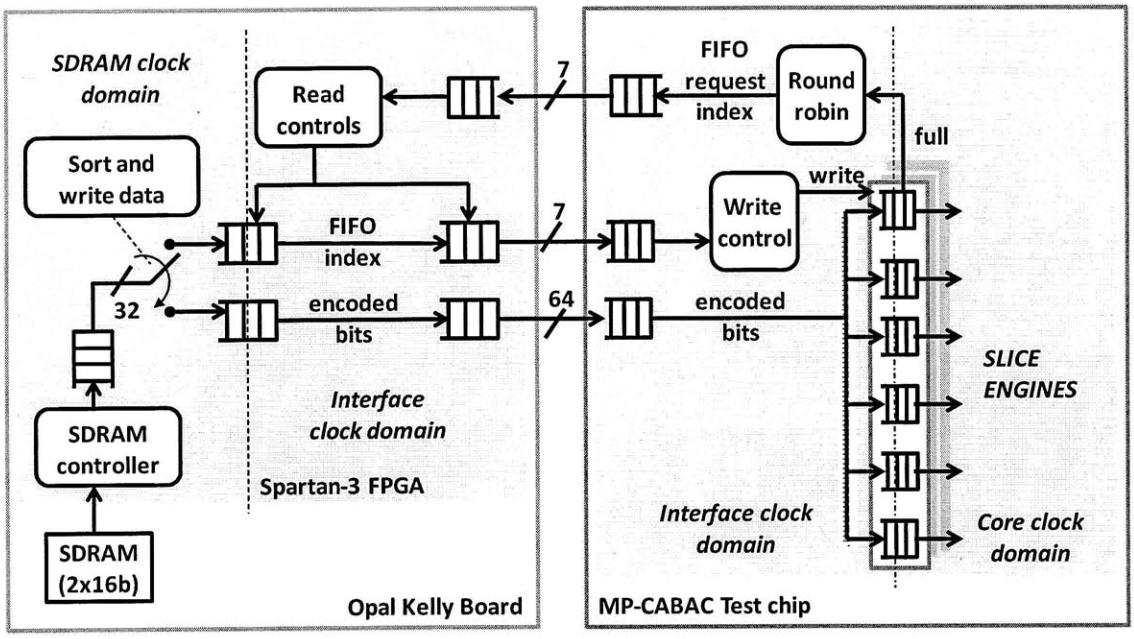


Figure 5-12: FPGA ASIC interface logic for input

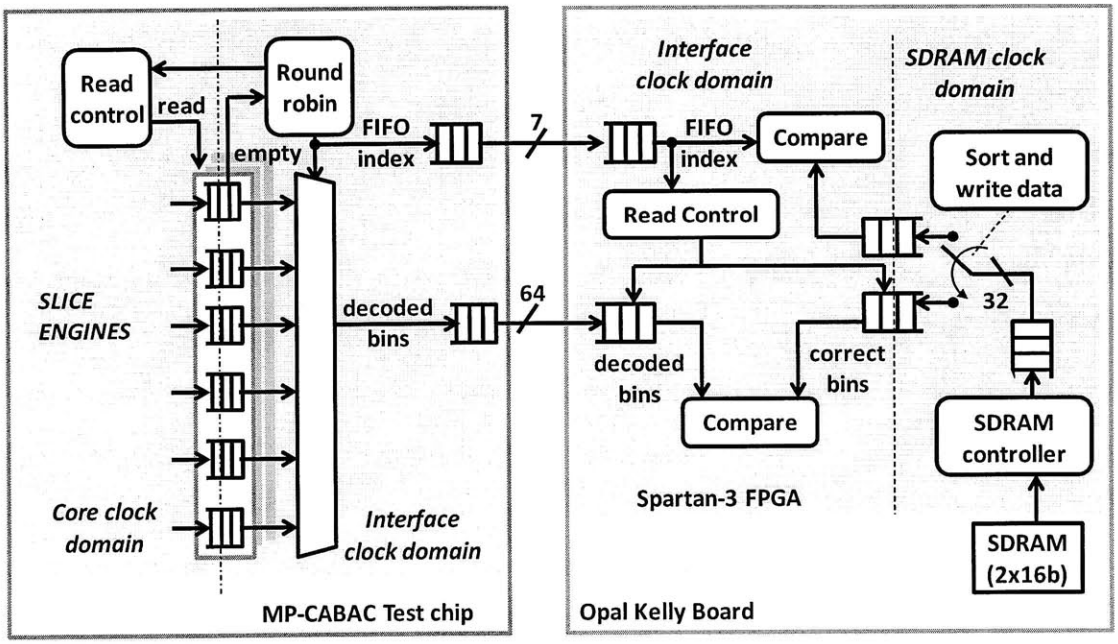


Figure 5-13: FPGA ASIC interface logic for output

5.9 Results and Measurements

The MP-CABAC test chip, shown in Fig. 5-14, was implemented in 65-nm CMOS. A summary of the chip statistics is shown in Table 5.1. The power was measured when performing real-time decoding of various 720p bitstreams with different number of IES (Table 5.2).

Simulations of the test chip at 0.9V indicate that it should support a core frequency up to 114 MHz and an interface frequency up to 90 MHz; however, the chip was tested at a maximum core and interface frequency of 67.2 MHz at 0.9V. This reduction in frequency was caused by the delay of the off-chip level converters and I/O pads of the test chip and the FPGA.

For simplicity, the interface logic on the FPGA assumes a specific order of FIFO index requests, which was determined from simulations. When the core and interface clocks are not aligned, the request order during testing becomes different from simulation. Consequently, during testing both interface and core clocks where driven by the same external clock to ensure phase alignment. A more flexible test setup which can adapt to different request orders can be used to mitigate this sensitivity to clock phase; a fast FPGA may be necessary to speed up the more complex interface logic.

The MP-CABAC test chip can achieve up to 1,594 Mbins/s at 0.9V. For the bigships (720p) sequence with QP=27, IBBP, this maps to a real-time (worst-case) frame rate of 1,807 fps and an average frame rate of 9,845 fps across 297 frames with a bit-rate of 146 Mbits/s. For the rally (4kx2k) sequence with QP=27, IBBP, this maps to a real-time (worst-case) frame rate of 245 fps and an average frame rate of 382 fps across 234 frames with a bit-rate of 116 Mbits/s. This is the highest report frame rate for 720p and 4kx2k resolutions. Based on these results, we can estimate to a first order that the MP-CABAC can achieve around 60 fps for 8kx4k. Frame rates of up to 120 fps are useful for high motion video. The rest can be traded off for power reduction as discussed in the next section.

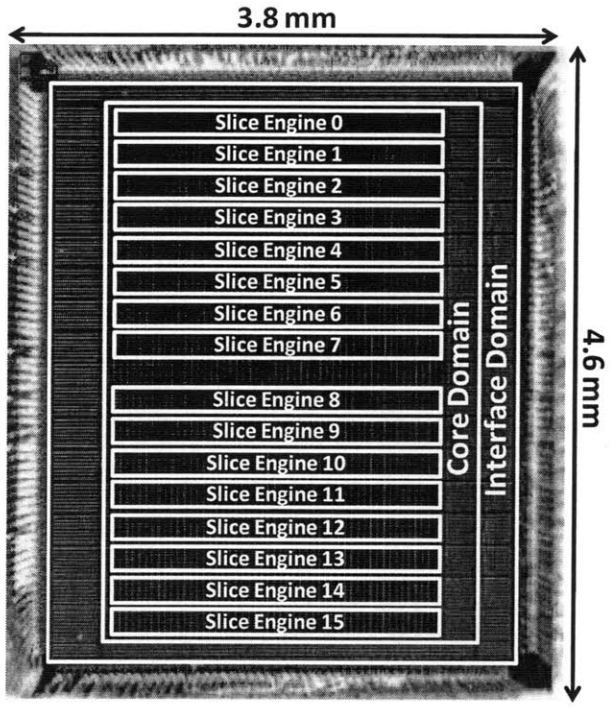


Figure 5-14: Die photo of MP-CABAC test chip (domains and slice engines are highlighted).

Table 5.1: Summary of MP-CABAC chip implementation.

Video Coding Standard	MP-CABAC for 'H.265'
Technology	65-nm
Core Area	3.4 mm × 4.2 mm
Logic Gate Count (NAND-2)	2502k
I/O Pads (package)	232 (223-pin PGA)
Supply Voltage	0.6 to 0.9 V (core) 1.8 V (I/O)
Operating Frequency	5.5 to 67.2 MHz
Core Power Consumption	0.6 to 28.8 mW

Table 5.2: Measured performance for varying number of IES per frame.

Supply Voltage [V]		**0.6**	**0.7**	**0.8**	**0.9**
Frequency [MHz]		**5.5**	**18.0**	**42.0**	**67.2**
IES=1	**Core Power [mW]**	0.6	1.4	3.2	5.4
	Bin-rate [Mbins/s]	14	45	106	169
IES=2	**Core Power [mW]**	0.7	1.7	4.0	7.7
	Bin-rate [Mbins/s]	25	80	186	298
IES=4	**Core Power [mW]**	0.8	2.5	6.0	11.3
	Bin-rate [Mbins/s]	44	140	327	523
IES=8	**Core Power [mW]**	1.1	3.5	9.2	18.0
	Bin-rate [Mbins/s]	78	249	580	928
IES=16	**Core Power [mW]**	1.4	6.3	14.4	28.8
	Bin-rate [Mbins/s]	130	512	996	1594

5.9.1 Tradeoffs

Fig. 5-15 shows the power-performance-coding efficiency tradeoff across the various bit-streams. The power-performance tradeoff is shown in Fig. 5-16 on a log scale for different number of IES. As the number of IES increases, the power-performance tradeoff improves. For instance, for 5 mW, 420 Mbins/s is achieved with 16 IES whereas only 160 Mbins/s is achieved with one IES, a 2.6x improvement. Alternatively, for 150 Mbins/s, this chip requires 4.7 mW with one IES and only 1.6 mW with 16 IES, a 2.9x reduction in power. This improved tradeoff comes at cost of 4.9% in coding efficiency.

The power consumption of the core domain ranges from 0.6 mW at 0.6 V to 28.8 mW at 0.9V when decoding between 14 to 1,594 Mbins/s respectively. The energy per bin is shown in Fig. 5-17. Increasing the number of IES can reduce the energy per bin to minimum of 10.5 pJ with 16 IES. The leakage energy dominates the energy per bin as we decrease the number of IES and supply voltage. The leakage could have been reduced by power gating the disabled slice engines.

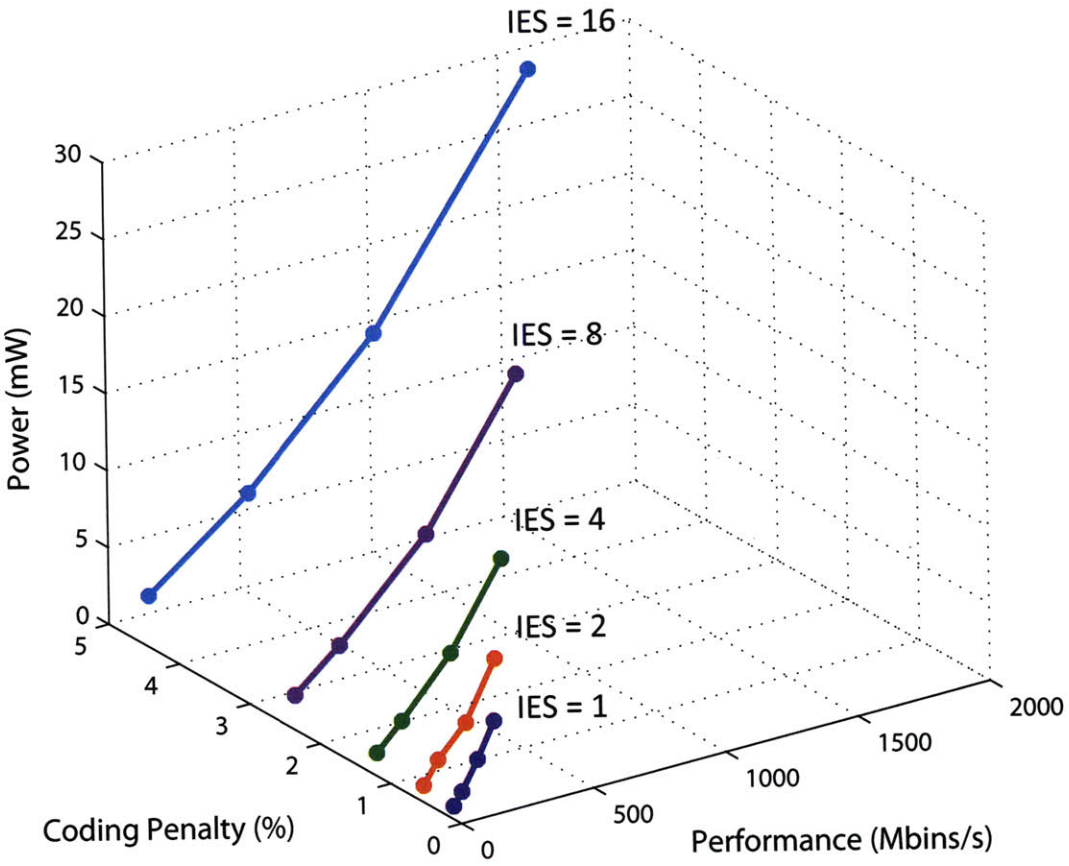


Figure 5-15: Tradeoff between power-performance-coding penalty for bigships, QP=27, IBBP. Each line represents a fixed number of IES and consequently a fixed coding penalty.

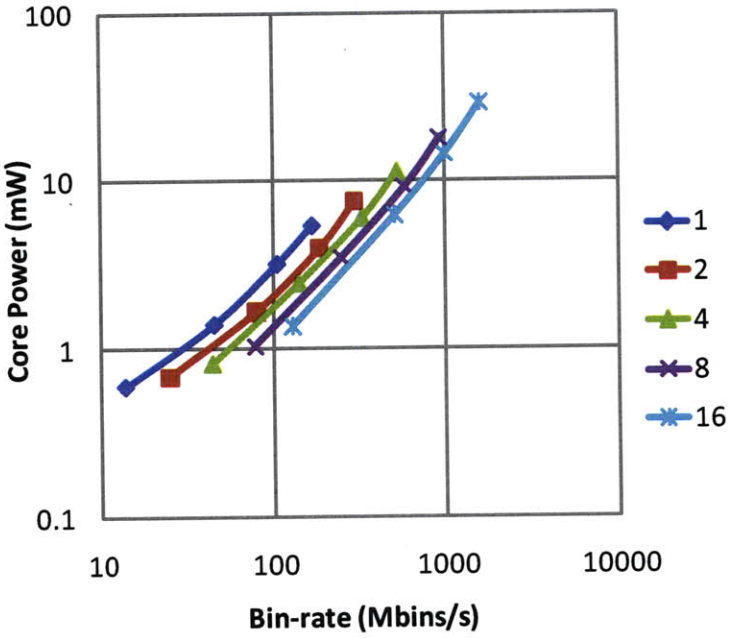
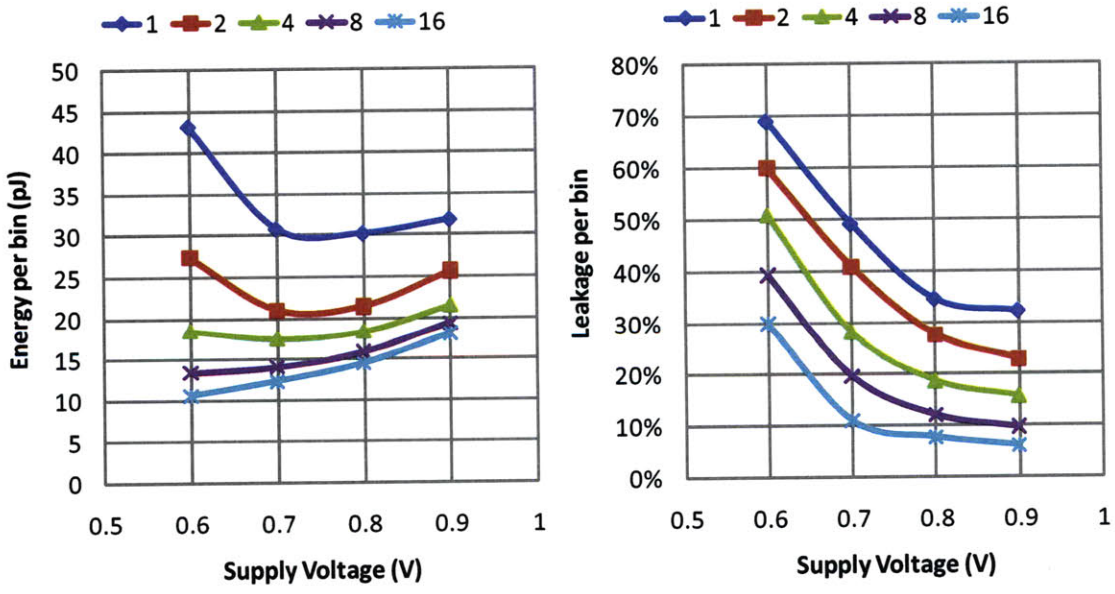


Figure 5-16: Power-performance tradeoff for various numbers of IES in bigships, QP=27, IBBP.



(a) Energy per bin

(b) Percentage of leakage energy

Figure 5-17: Energy per bin versus supply voltage for various number of IES in bigships, QP=27, IBBP.

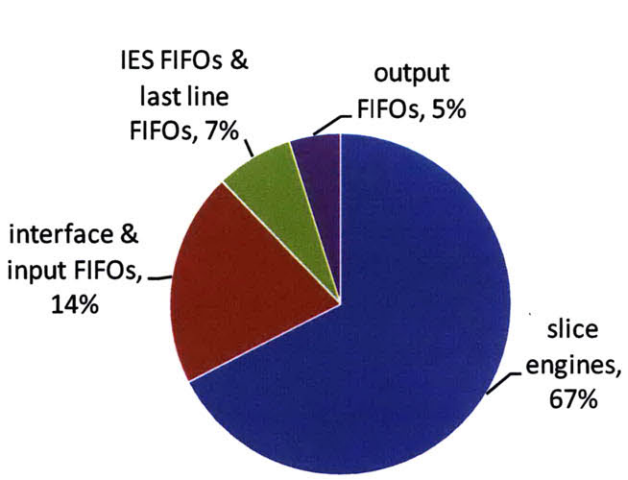
5.9.2 Power Breakdown and Switching Activity

Power breakdown was determined from simulation with the nominal corner, operating at 1.2V at 25 °C using the switching activity of bigships (IES=16, QP=27, IBBP). The slice engines only consume 67% of the total chip power as shown in Fig. 5-18a. The remaining power is consumed by the last line and IES FIFOs, interface logic and input/output FIFOs. Within the slice engine, a significant amount of power goes to the arithmetic decoder, context selection, context memory and the data flow control (Fig. 5-18b). Finally, Fig. 5-18c shows the power breakdown across partitions. SIGMAP dominates since it decodes the most bins and must read from a large context memory. Fig. 5-19 shows a breakdown of the leakage power.

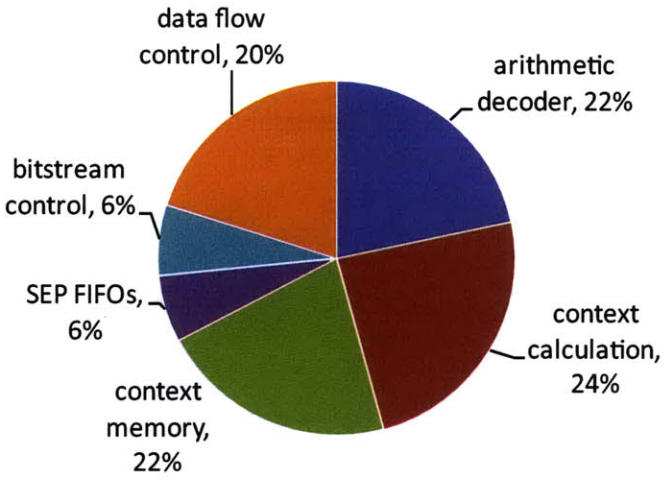
Fig. 5-20 shows the switching activity in different modules of a slice engine in the MP-CABAC. These values were obtained through gate level simulations of the bigships sequence over a time interval of 5,000 ns (625 CORE cycles) for a B-frame. Note that the switching activity in SIGMAP and COEFF engine are high, which is consistent with the fact that more bins are in the SIGMAP and COEFF partitions.

5.10 Summary

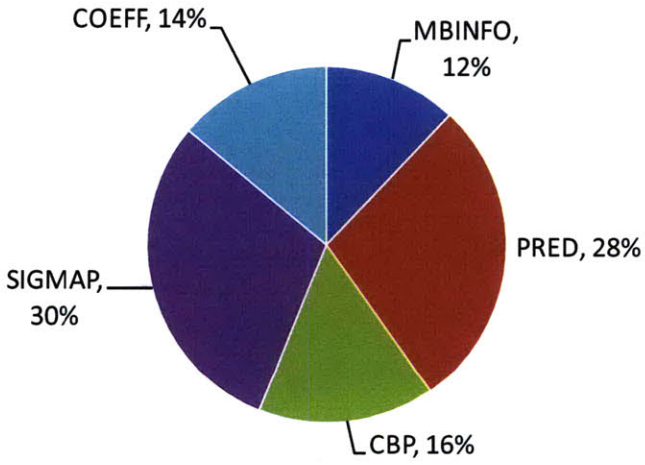
This chapter presents a highly scalable MP-CABAC test chip with 80 parallel engines. The performance of the test chip can be scaled by adjusting the number of IES, the supply voltage, and the operating frequency, enabling it to tradeoff power-performance-coding efficiency depending on the target application. The measured power of the test chip ranges between 0.6 mW to 28.8 mW and can achieve a bin-rate up to 1,594 Mbins/s, for a real-time frame rate of 1,807 fps for 720p (bigships) or 245 for 4kx2k (rally). A minimum energy per bin of 10.5 pJ can be achieved with 16 IES per frame at a cost of 4.9% in coding penalty. A round robin scheme was used to transfer data on/off-chip for each of the 80 engines. Custom clock gating was used to reduce clock power of disabled slice engines when the number of IES is less than 16. The test setup used to verify and characterize the test chip was also described.



(a) Breakdown for entire MP-CABAC

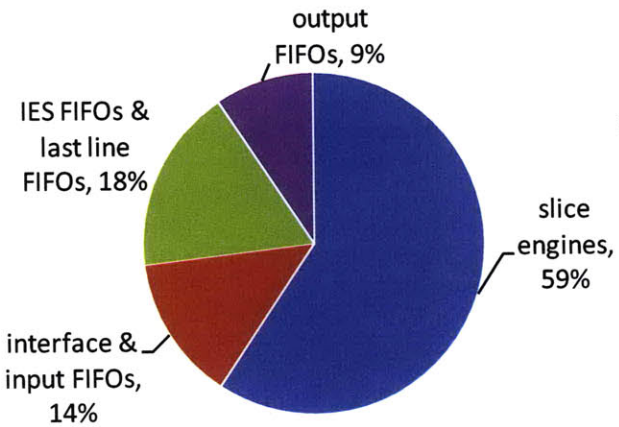


(b) Breakdown within a slice engine

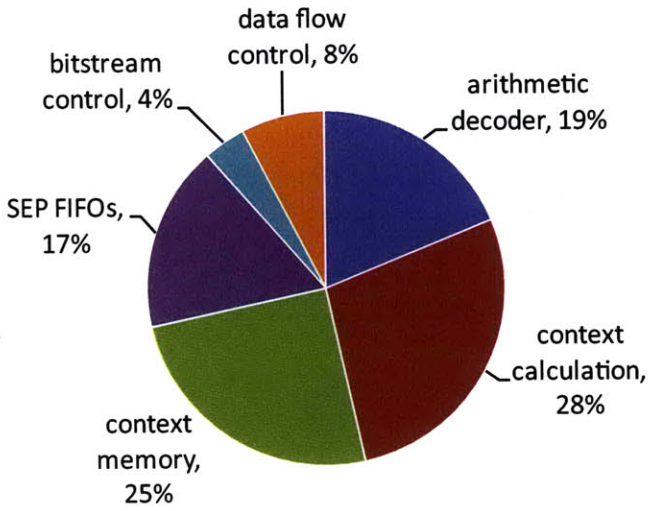


(c) Breakdown across partitions

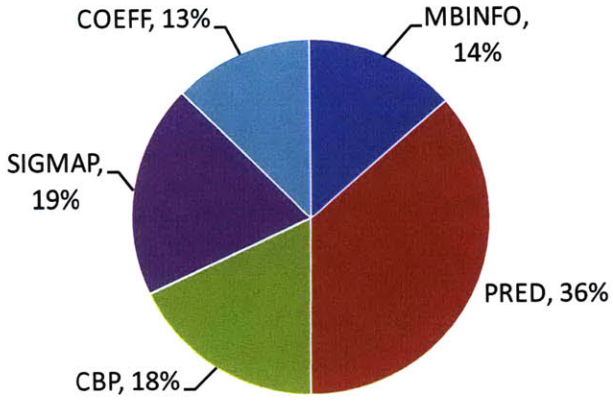
Figure 5-18: Simulated (post-layout) power breakdown of MP-CABAC.



(a) Breakdown for entire MP-CABAC



(b) Breakdown within a slice engine



(c) Breakdown across partitions

Figure 5-19: Leakage power breakdown of MP-CABAC.

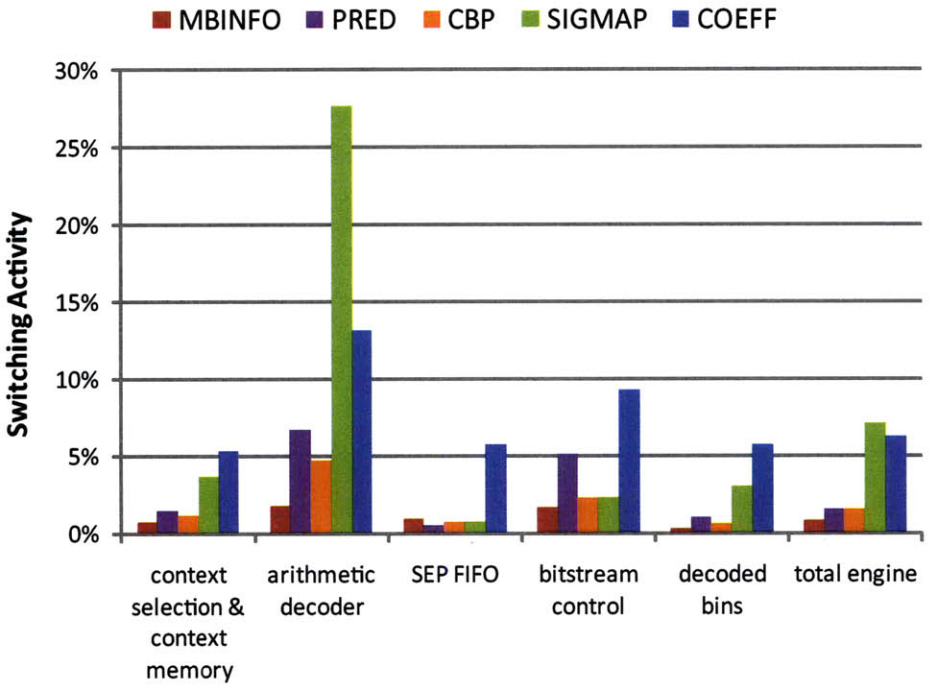


Figure 5-20: The switching activity of different components in a slice engine of MP-CABAC, simulated with the bigships, QP=27, IES=16, IBBP.

Chapter 6

Conclusions and Future Work

This thesis presented several key optimizations that illustrate how parallelism can meet the power-performance requirements for current and next generation video coding in a cost effective manner. First, applying parallel architectures to key bottlenecks can increase throughput; when done carefully, area cost can be kept to a minimum. Second, aggressive voltage scaling can be used to tradeoff the high throughput for significant power savings. Third, Amdahl's law shows that parallel architectures are not enough; parallel algorithms are necessary to enabled effective utilization of the available hardware afforded by Moore's law. Parallel algorithms, designed with parallel architecture in mind, can increase throughput with reduced area and coding efficiency cost relative to H.264/AVC. These algorithms also enable scalability across power-performance-coding efficiency. Finally, joint optimization of algorithm and architecture enables us to identify minor changes that have a significant impact on increasing performance and reducing area cost without affecting coding efficiency. While parallel architectures and algorithms improve tradeoffs, joint architecture and algorithm optimizations can provide significant benefits with no tradeoff in cost.

In this thesis, we presented several methods (bin, syntax element, and slice parallelism) of increasing the CABAC throughput with different costs such as power, coding penalty and area. Thus, the method should be selected based on the constraints of the application. For instance, if power is of utmost importance, then syntax element and slice parallelism

should be used instead of bin parallelism which requires extra computations for speculation. Alternatively, if coding efficiency is the focus, then bin parallelism should be used. For area constrained applications, bin and syntax element parallelism should be use.

6.1 Summary of Contributions

In this section, we will summarize several architecture and algorithm optimizations that demonstrate the key ideas presented in this thesis.

6.1.1 Parallel Architectures

- **Deblocking Filter (DB) and Motion Compensation(MC)** were identified as key bottlenecks that required parallelization in the H.264/AVC decoder. The modified DB uses four parallel filters for a 2 to 3x throughput improvement, while the modified MC uses two luma interpolators and four chroma interpolators for a 2 to 4x throughput improvement. In both DB and MC, luma and chroma components are processed in parallel. Parallelizing DB and MC only increased the total logic area in the decoder by 12%.
- **Partition and slice engines** enable binary arithmetic decoders to operate in parallel. Each slice engine contains 5 parallel partition engines, and each frame can be decoded with 16 parallel slice engines. This results in a total of 80 binary arithmetic decoders that can operate in parallel, which provides a 24.11x increase in throughput relative to a single decoder. The parallel engines are synchronized using FIFOs.

6.1.2 Parallel Algorithms

- **Syntax Element Partitions (SEP)** enable bins of different syntax elements to be processed in parallel. It reduces the coding penalty by 2 to 4x compared to H.264/AVC for the same 2.7x throughput improvement.

- **Interleaved Entropy Slices (IES)** enable bins of different entropy slices to be processed in parallel. It reduces the coding penalty by 2 to 3x compared to H.264/AVC for the same 10x throughput improvement.
- **Massively Parallel CABAC (MP-CABAC)** leverages SEP and IES to provide high throughput with low coding efficiency and area cost (improved tradeoff). It can be used in conjunction with bin parallelism for additional increase in throughput. The algorithm is scalable such that it can be used for various applications. Architectural considerations such as memory bandwidth, synchronization, area cost, etc. are accounted for in the algorithm design.

6.1.3 Joint Algorithm/Architecture Optimization

- **Modified mvd Context Selection** reduces last line buffer size by 50% at a cost of 0.02% in coding penalty.
- **Range Comparison Reordering** provides a 11% reduction in the critical path with no impact on coding efficiency.

6.1.4 Test Chips

Two test chips were fabricated to demonstrate the impact of the aforementioned algorithm and architectural optimizations.

A **low power H.264/AVC video decoder** was implemented in 65-nm CMOS that utilized architecture techniques such as parallelism, pipelining, and domain partitioning to maximize voltage scaling and power savings, while maintaining performance. Off-chip memory bandwidth was reduced to address system power. A complete real-time decoding system was implemented. The test chip can decode HD 720p at 30fps with measured power of 1.8mW at 0.7 V, which is over 6x lower that previously published results. This can be attributed to the parallel architectures that enable a high throughput of 1.84 pixels per cycle.

A **Massively Parallel CABAC (MP-CABAC)** test chip in 65-nm CMOS was designed and implemented with a high performance architecture that can provide scalability

in three dimensions (power-performance-coding efficiency). Measured performance of 1,594 Mbins/s at 0.9 V is achieved, for a real-time frame rate of 1,807 fps for 720p (bigships) or 245 fps for 4kx2k (rally). Co-optimization of algorithm and architecture enabled additional speed up with no impact on coding efficiency, and reduction in area with negligible coding penalty. Additional architectural strategies were used to increase the performance by reducing critical path and balancing workload across engines. Round robin interface was used to share the limited I/O across all 80 engines. Custom clock gating was used to reduce the clock power of the engines.

6.2 Future Work

There are many exciting challenges that lie ahead for next generation video coding.

- *Adoption of MP-CABAC into 'H.265'.* A new working group, Joint Collaborative Team for Video Coding (JCT-VC), has recently formed to begin formal standardization of 'H.265'. Rather than only focusing on coding efficiency, this next-generation coding standard intends to also address the needs of mobile applications and focus on the complexity of the video codec algorithms and the energy cost of the video codec circuits. The MP-CABAC has been submitted in a joint proposal by Texas Instruments and Massachusetts Institute of Technology [90].
- *Fully Parallel CODEC.* To evaluate the impact of the MP-CABAC, a fully parallel codec should be implemented. In addition to an ASIC implementation, this highly parallel codec could also be mapped to a GPU or a many-core processor.
- *Optimize MP-CABAC.* Additional optimizations of the MP-CABAC should be explored. For instance, to increase throughput, smart encoding algorithms that adaptively map syntax elements to different partitions based on workload can be developed. To reduce power, syntax elements should be partitioned to exploit low switching activity rate (e.g. bins which often have the same context or state would be placed in the same partition such that the switching in the context selection and memory is reduced.)

- *Closing the loop on DVFS.* To fully leverage the highly voltage-scalable architectures and maximize the impact of DVFS, effective workload prediction strategies are required to set the correct voltage and frequency for a given workload.
- *Integration of off-chip memory.* Off-chip memory bandwidth consumes a significant portion of the system power. This power can be reduced by leveraging existing advanced technologies such as System-In-Package, where both separately fabricated DRAM and decoder die are vertically stacked and connected in a package, and embedded DRAM, where DRAM and decoder are integrated on the same die. Upcoming technologies include through-silicon via (TSV), which integrates off-chip memory vertically on top of the decoder die and connecting them through specially fabricated vias to further reduce interconnect length and power. TSV can also increase available memory bandwidth since reducing interconnect length enables higher speed.
- *Power-Rate-Distortion Optimization.* Existing encoders can tradeoff quality for reduced power consumption by adjusting motion estimation parameters and intra mode search [91]. Encoders can also reduce power by extending rate-distortion optimization to include encoder power; power-rate-distortion optimization can account for the encoding power when making a decision on the prediction mode [92]. The difficulty lies in developing a model that can estimate encoding complexity and power in a fast and simple way while meeting real-time encoding requirements. A potential variation of this approach would be to include *decoder* power in the optimization for broadcast applications where the energy constraint is more likely to be in the decoder than the encoder.
- *Parallelizing Multiview Video Coding (MVC) and Scalable Video Coding (SVC).* The first drafts of the standards document for Multi-view Video Coding (MVC) for 3-D video and Scalable Video Coding (SVC) have recently been completed. Both standards can involve processing multiple streams of data and would benefit from the application of parallel architectures and low power techniques. MVC and SVC encoding also continues to be an area of research.

- *Multi-standard CODEC.* Video content can now come from a variety of different sources. As a result, they can be coded in numerous different standards. Supporting multiple codecs poses an interesting challenge as it requires a balance between reconfigurability, power and performance. Furthermore, software/hardware partitioning of the codec should be examined.

Appendix A

Additional Details on CABAC

A.1 Performance Requirements Calculations

It is important to understand the performance requirements for real-timing decoding applications such as video conferencing. To achieve real-time low-delay decoding, the processing deadline is dictated by the time required to decode each frame to achieve a certain frames per second (fps) performance.

The performance requirement of the arithmetic coding engine, and thus operating frequency, is dictated by the rate of the bins that need to be encoded and decoded (i.e. bin-rate), and not the bits of the compressed data (i.e. bit-rate). While the H.264/AVC standard restricts the bit-rate at each level, there is no equivalent restriction on the bin-rate; instead the number of bins per frame is restricted. This was done to reduce the complexity of the rate-control at the encoder.

Table A.1 shows the peak bin-rate requirements for a frame to be decoded instantaneously based on the specifications of the H.264/AVC standard [26]. They are calculated by multiplying the maximum number of bins per frame by the frame rate for the largest frame size.

Section 7.4.2.10 of the H.264/AVC recommendation [26] limits the maximum number of binary symbols in a coded picture as follows:

`BinCountsInNALunits<(32/3)*NumBytesInVclNALunits+(RawMbBits*PicSizeInMbs)/32`

where `BinCountsInNALunits` is the coded picture size in bins. `NumBytesInVclNALunits` is defined as the sum of `NumBytesInNALunit` values for all VCL NAL units of a coded picture and it is limited for the main profile in Section A.3.1 in [26]:

$\sum$ `NumBytesNALunit(n>0)=384*MaxMBPS*(tr(n)-tr(n-1))/MinCR`

Maximum value of the `BinCountsInNALunits` can be computed using the level specific limits in Table A-1 in [26]. Consequently, the CABAC decoder performance requirement for a worst case coded picture can be computed by multiplying the `BinCountsInNALunits` with the maximum frame rate allowed at a given level.

For high definition (level 3.1 to 5.1) in H.264/AVC, the maximum bin rate ranges from 121 Mbins/s up to 2.11 Gbins/s [26]. To guarantee real-time decoding with no latency, the CABAC engine must meet this performance. Without concurrency, decoding 1 bin/cycle requires multi-GHz frequencies, which leads to high power consumption and is difficult to achieve even in an ASIC. Existing H.264/AVC CABAC hardware implementations such as [71] only go up to 149 MHz; the maximum frequency is limited by the critical path, and thus parallelism is necessary to meet next generation performance requirements.

A.2 Bin Parallelism for H.264/AVC CABAC

Bin parallelism is particularly challenging at the decoder due to the strong data dependencies from bin to bin. For instance, typically the context to be used on a bin depends on the value of the previous bin. Thus, to decode two bins in parallel, the context for the second bin is unknown until the first bin has been decoded. Consequently, it is a significant challenge to decode both bins at the same time, since it is necessary to know which context to use in order to correctly decode the compressed data. One method of achieving parallelism involves some speculative computation. This enables the engine to start decoding the second (third, fourth, etc.) bin before the first bin has fully been resolved. However, speculation requires

Table A.1: Peak bin-rate requirements for real-time decoding of worst case frame at various high definition levels.

Level	Video Resolution [width x height]	Maximum Frame Rate [fps]	Maximum Bit-rate [Mbits/s]	Maximum Bins per frame [Mbins]	Peak Bin-rate [Mbins/s]
3.1	1280 x 720	30	17.5	4.0	121
3.2	1280 x 720	60	25	4.0	242
4.0	1920 x 1080	30	25	9.2	275
4.1	1920 x 1080	30	50	17.6	527
4.2	1920 x 1080	60	50	17.6	1116
5.0	1920 x 1080	72	169	17.6	1261
5.1	1920 x 1080	120	300	17.6	2107

Table A.2: A comparison of the various H.264/AVC CABAC architectures. *Estimated based on independent simulations of bin distribution

Ref.	Approach	Bins in Parallel	Cycles per bin	Tech.	Freq. [MHz]	Bin-rate [Mbins/sec]	
						Range	Average
[71]	Precompute	1 to 3	1	0.18μm	149	149 to 447	218*
[80]	Assume MPS	2	3	0.18μm	303	101 to 202	124
[85]	Variable Bin-Rate	1 to 16	1	0.18μm	45	45 to 720	129*

additional computations which may increase power consumption. Furthermore, the critical path increases with each additional bin, since all computations cannot be done entirely in parallel (e.g. each bin needs to wait for 'codIRangeLPS' from the previous bin). This reduces the overall performance improvement that can be achieved. Table A.2 summarizes several variations of this approach. Note that the reported bin-rates for these approaches are in the low hundreds of Mbins/s. Additional parallelism is needed to reach the Gbins/s required for 4kx2k.

It is important to note that as the degree of bin parallelism increases, the number of precomputed values as well as the complexity of the context management increases substantially.

Precompute In [71], Yu proposes an architecture that decodes one to three bins per cycle. Specifically, for each possible outcome of the first bin, it precomputes some values necessary for second bin decoding while the first bin is still begin decoded (Fig. A-3a). Once the first bin is resolved, the appropriate set of precomputed values is selected, and the second bin can then be resolved much faster. If the third bin is coded with a bypass mode (i.e. doesn't require a context), then it can also begin decoding before the second bin is resolved. This approach is applied to a subset of the syntax elements. The critical path is increased due to the additional time for the second bin, and potentially third bin, to resolve. Note that in this approach, as the number of parallel bins increases linearly, the number of precomputations and context fetches from memory increase *exponentially.*

Assume MPS In [80], Kim proposes decoding two bins in parallel by assuming that the first decoded bin is always the most probable symbol (MPS) (Fig. A-3b). Thus, it only precomputes one set of values for the second bin while the first bin is still being decoded. This approach is applied to all syntax elements. The critical path is increased due to the additional time for the second bin to resolve. The increase is less than [71] though since renormalization is not required for the second bin when MPS is assumed. However, it produces two bins less frequently than [71] since it relies on the probability of MPS. Unlike [71], as the number of parallel bins increases linearly, the number of precomputations and context fetches from memory also increase approximately linearly.

Variable Bin-Rate In [85], Zhang extends [80]'s "Assume MPS" approach to 16 bins (Fig. A-3c). This is applied to a subset of syntax elements. The authors claim that this guarantees a fixed bit-rate of decoding, with a variable bin-rate. However, this approach increases the critical path significantly and the performance becomes quite low if there is a short string of MPS. Short strings of MPS are much more likely than long strings of MPS as shown in Fig. A-1. Fig. A-2 shows how the bin-rate will scale with increase number of bins. We can see that if each addition bin increases the critical path by 7.7%, then the maximum bin-rate increase of 1.84x is achieved at 5 bins. It is likely that the additional delay per

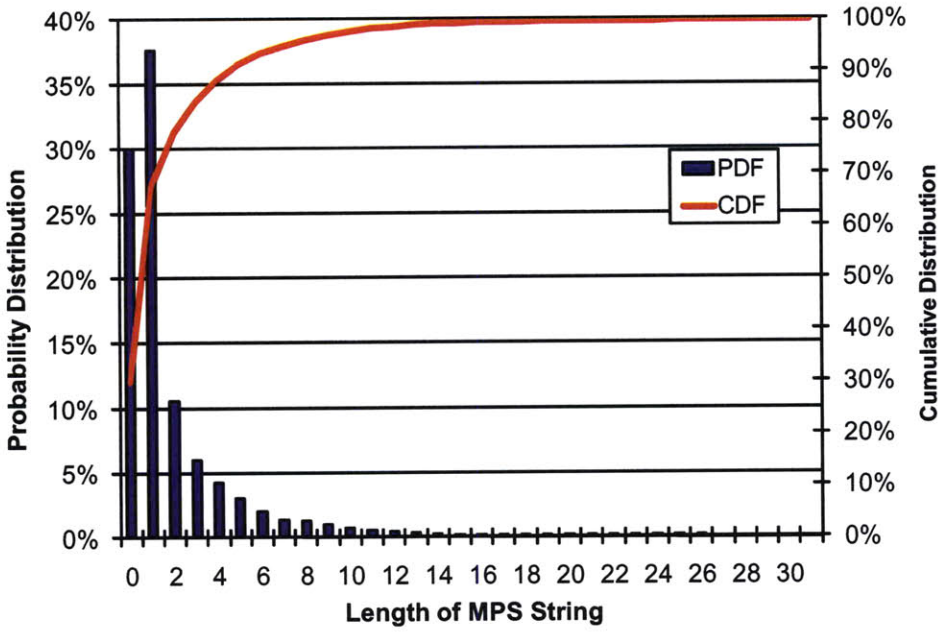


Figure A-1: Distribution for length of MPS string (averaged across five common condition sequences). As the length of the MPS string increases, it is less liklely to occur.

bin is greater than this, which would move the peak to the left. Finally, it is important to note that the bit-rate does fluctuate and is only constant when averaging across a certain window of time; thus, this approach will have problems meeting performance requirements if the bit-rate has a large peak.

A.3 Bin Parallelism for 'H.265' CABAC

The MP-CABAC can be used in conjunction with the bin parallelism to further increase the throughput. One of the main challenges of bin parallelism occurs at the decoder. Typically the context to be used on a bin depends on the value of the previous bin. Thus, to decode two bins in parallel, the context for the second bin is unknown until the first bin has been decoded. Consequently, it is a significant challenge to decode the second bin at the same time as the first since it is necessary to know which context to use in order to correctly decode the compressed data.

Various approaches for bin parallelism in H.264/AVC are presented in Appendix A.2.

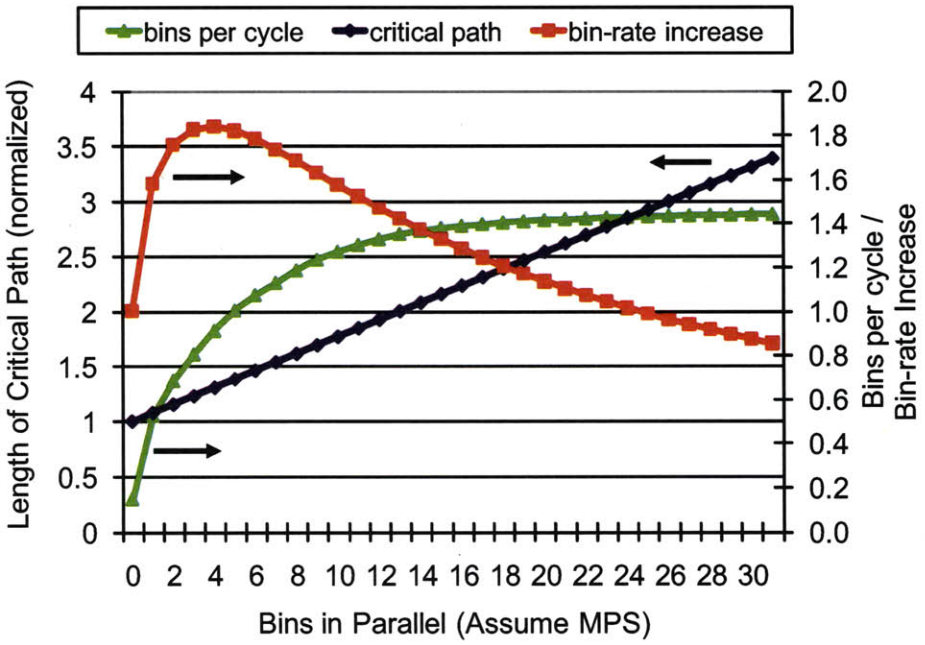
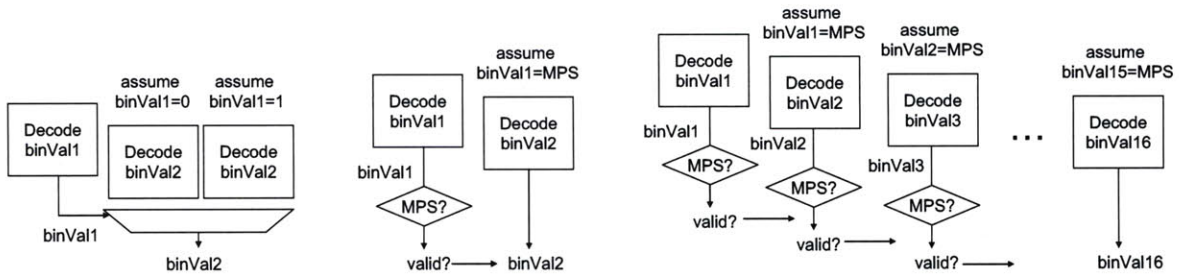


Figure A-2: Impact of number of bins on throughput, critical path and bin-rate. Plot assumes that each additional bin increases the critical path by 7.7%.



(a) Precompute (b) Assume MPS (c) Variable Bin Rate

Figure A-3: Various forms of bin parallelism in H.264/AVC CABAC.

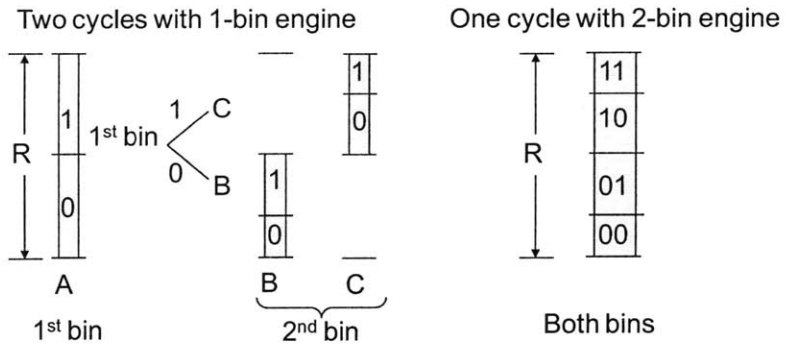


Figure A-4: Conceptual illustration of bin parallel decoding. The first bin is encoded/decoded with context A, while the second bin can be encoded/decoded with context B or C. The range is divided into four intervals rather than two.

Several of these approaches utilize predictive schemes to guess the context of the second bin, which would still have limited throughput in the worst case. An non-standard compliant bin parallel scheme is presented in [73]; however, it does not account for the fact that the context can change for each bin. In [31,74], we proposed an algorithmic approach of enabling bin parallelism for the next generation standard, which is fundamentally parallel and can deterministically decode several bins with different contexts at the same time.

Deterministic decoding that accounts for the different contexts is achieved through the use of *conditional* probabilities. At any given time, there are only two possible outcomes for the first bin (one or zero) and thus there are only two contexts that could be used for the second bin based on the value of the first bin. Let's assume they are context B and C respectively. The conditional probabilities of B are C are shown in equations below.

$$\begin{aligned} Pr_B(2^{nd}Bin) &= Pr(2^{nd}Bin|1^{st}Bin = 0) \\ Pr_C(2^{nd}Bin) &= Pr(2^{nd}Bin|1^{st}Bin = 1) \end{aligned}$$

Table A.3 can be constructed using these probabilities. Rather than dividing the range into two intervals every cycle, we divide the range into four intervals based on Table A.3. Consequently, picking one interval based on the offset will decode 2-bins. This form of 2-bin decoding is illustrated in Fig. A-4.

Table A.3: Probability table to construct four intervals for 2-bin decode.

$1^{st}bin$	$2^{nd}bin$	**Probability**
0	0	$Pr_A(0) \times Pr_B(0)$
0	1	$Pr_A(0) \times Pr_B(1)$
1	0	$Pr_A(1) \times Pr_C(0)$
1	1	$Pr_A(1) \times Pr_C(1)$

A.3.1 Coding Efficiency and Throughput

This approach has negligible impact on coding efficiency. For a 2-bin per cycle implementation, the average coding efficiency penalty (BD-rate) was 0.76% relative to H.264/AVC, with a throughput improvement of 1.79 to 1.98x.

A.3.2 Area Cost

In terms of architecture, the context memory which typically dominates the area does not need to be replicated; only a decision tree needs to be added to select context B and C. The arithmetic coding engine also needs to be extended from 9-bit to 14-bits. A true multiplier can be used to avoid a large look up table.

Appendix B

Calculating Coding Efficiency

In this thesis, we often compare the coding efficiency of various algorithms. In this appendix, we will discuss how this metric is computed.

The VCEG standards body provides a set of common conditions under which a set of reference sequences should be coded in order to compare the coding efficiencies [34] of various algorithms. There are five 720p HD reference sequences: bigships, city, crew, night, and shuttle. Details on these sequences can be found in Appendix E. The common conditions set encoding parameters such as motion estimation search range, QP, GOP size, etc.

To determine the coding efficiency, each of these video sequences is encoded at with varying QP to create the rate-distortion (RD) curve. The common conditions [34] set by VCEG uses QP of 22, 27, 32, and 37. The RD-curve shows how the video fidelity, measured in peak signal-to-noise ratio (PSNR), changes with bit-rate. Fig. B-1 shows an example RD-curves of CAVLC and CABAC for reference sequence bigships with prediction structure IBBP. The coding efficiency is measured using the Bjøntegaard ΔBitrate (BD-rate) [78], which computes the average difference between the RD-curves. The difference can also be measured in terms of BD-PSNR. An excel spreadsheet plug-in is provided in [93].

Finally, the reported coding efficiency is typically the average across all five 720p reference sequences and prediction structures (IBBP, IPPP, Ionly). In the next section, we will provide the results from experiments comparing the coding efficiency of CABAC and CAVLC.

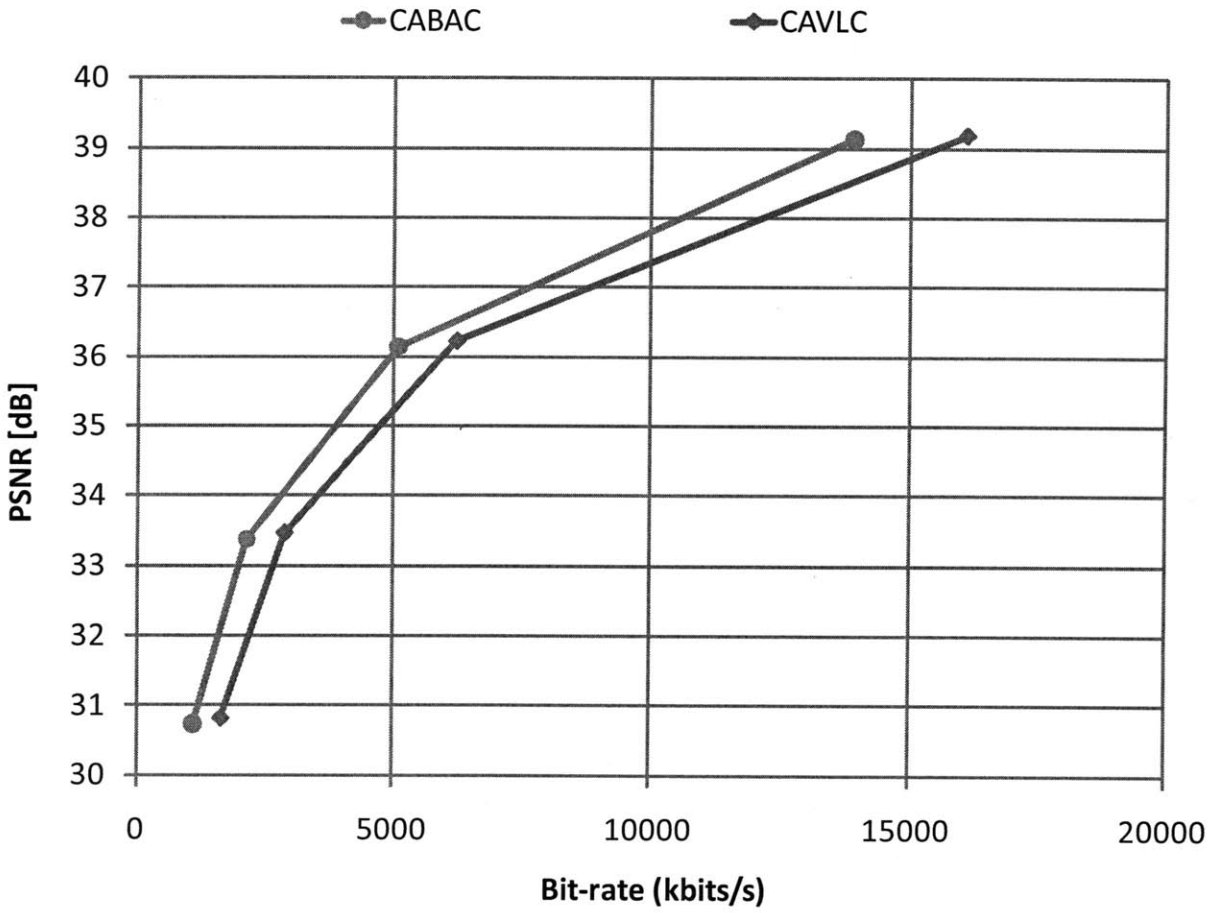


Figure B-1: Rate-distortion (RD) curves used to evaluate coding efficiency of CABAC versus CAVLC for sequence bigships, with IBBP prediction structure. The BD-rate measures the average difference between the RD-curves. From the above two curves, we can see that CABAC has better coding efficiency since it provides higher PSNR at the same bit-rate, or equivalently lower bit-rate at the same PSNR.

Table B.1: A comparison of the coding efficiency of CAVLC and CABAC in H.264/AVC for common conditions.

Prediction Structure	Video	BD-PSNR	BD-rate
Ionly	bigships	-0.5	9.85
	city	-0.43	7.01
	crew	-0.41	11.33
	night	-0.35	5.49
	shuttle	-0.60	16.03
	Average	**-0.46**	**9.94**
IPPP	bigships	-0.49	15.95
	city	-0.42	13.51
	crew	-0.43	15.08
	night	-0.37	9.99
	shuttle	-0.59	16.93
	Average	**-0.46**	**14.29**
IBBP	bigships	-0.73	25.49
	city	-0.74	26.68
	crew	-0.59	21.11
	night	-0.58	17.27
	shuttle	-0.97	30.24
	Average	**-0.72**	**24.16**

B.1 CAVLC vs. CABAC for HD

Five 720p video sequences were encoded under common conditions to quantify the impact of CABAC entropy coding versus CAVLC. The BD-PSNR and BD-rates were computed for different prediction structures as shown in Table B.1. The CABAC gives between 5.49% to 30.24% improvement in coding efficiency (BD-rate) over CAVLC. The average improvement is 16.13%.

Appendix C

Enabling Multi-Core Processing with IES

In Chapter 2, parallelism was applied at the micro-architecture level. IES enable the entire decoding path to be parallelized; subsequently, parallelism can be achieved at the core level by replicating units of the entire decoder, and all units can be used to decode a single video sequence (Fig. C-1). This is beneficial in terms of minimizing design costs, since it can be done with existing decoder cores. IES can be used for both CABAC as well as CAVLC. To demonstrate the impact of IES at the decoder level, the IES approach was applied to the decoder discussed in Chapter 2. Note that the use of IES renders the decoder non-standard compliant. An RTL implementation was simulated[1] across several video sequences and degrees of parallelism to evaluate its impact on performance and power savings [33].

As the number of IES increases, the operating frequency required to hit a performance point reduces; this allows the voltage to be lowered resulting in power savings. Fig. C-2 shows the tradeoff between the number of IES and the estimated power savings for a given performance point (the number of IES on the x-axis will scale for different resolutions/frame rates). Another benefit of multi-core IES is that the last line buffer does not need to be replicated across cores. The area cost increases linearly with a slope less than 1 relative to

[1]The RTL implementation of IES for CAVLC and the subsequent simulations were done by Daniel Finchelstein

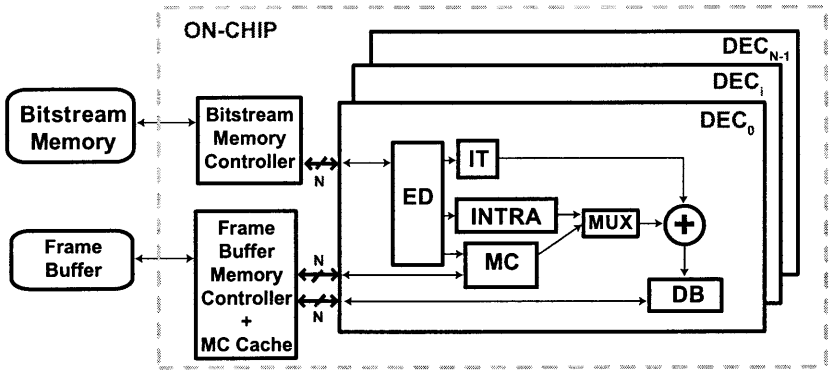


Figure C-1: Muti-core decoder architecture.

the degree of parallelism as shown in Fig. C-2.

Fig. C-3 shows that the IES approach results in a higher throughput-parallelism trade-off when compared with other multi-core approaches, namely H.264/AVC slices and frame parallelism. The higher IES performance can be attributed to better workload balance and better coding efficiency (i.e. reduces workload since fewer bits need to be processed).

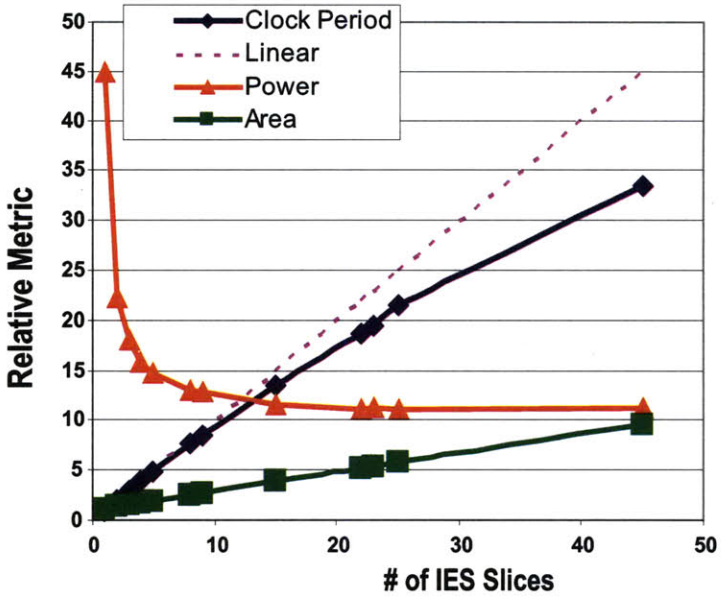


Figure C-2: Performance of IES multi-core decoding. The power is normalized relative to a single decoder running at the nominal supply voltage. The area increase assumes caches make up 75% of the area of a single decoder core and the memory controller takes 23% of the logic [30].

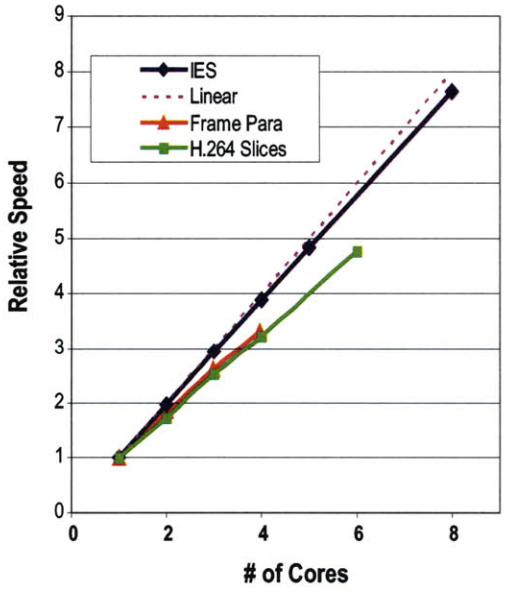


Figure C-3: Three different multi-core architectures show nearly-linear performance gains. The multi-core performance of H.264/AVC slices is slightly lower because of the extra processing required by the CAVLC and also the unbalanced slice workload due to uneven image characteristics across the slices.

Appendix D

Content of SEP FIFOs

This chapter describes the data stored in each SEP FIFO. Tables D.1, D.2, D.3, D.4, and D.5 list the data in each FIFO for partitions MBINFO, PRED, CBP, SIGMAP, and COEFF respectively. The size of the data is also provided along with the frequency with which the data is accessed (e.g. every slice, macroblock, 4x4 block). Careful effort was made to reduce the data widths and consequently the entry size of the SEP FIFOs. In hindsight, the PRED FIFOs could be significantly reduced if the `mb_type` and `sub_mb_type` were sent from MBINFO rather than the number of motion vectors and reference index and their locations.

Table D.1: Input SEP FIFOs to MBINFO partition.

Data	Width	Read/Write Frequency
slice_type	2	slice
transform_8x8_mode_flag	1	slice
num_ref_idx_l0_gt1	1	slice
num_ref_idx_l1_gt1	1	slice
num_ref_idx_l0_lt2	1	slice
num_ref_idx_l1_lt2	1	slice
direct_8x8_inference_flag	1	slice

Table D.2: Input SEP FIFOs to PRED partition.

Data	Width	Read/Write Frequency
slice_type	2	slice
end_of_slice_flag	1	macroblock
mb_skip_flag	1	macroblock
predType	1	macroblock
Intra_16x16	1	macroblock
transform_8x8_mode_flag	1	macroblock
B_Direct_16x16	1	macroblock
numRefIdxL0_perMb	3	macroblock
numRefIdxL1_perMb	3	macroblock
numMvdL0_perMb	5	macroblock
numMvdL1_perMb	5	macroblock
block_skip_l0	4	4x4 block
block_skip_l1	4	4x4 block
block_num0RefIdxL0	2	4x4 block
block_num1RefIdxL0	2	4x4 block
block_num0RefIdxL1	2	4x4 block
block_num1RefIdxL1	2	4x4 block
block_num0MvdL0	4	4x4 block
block_num1MvdL0	4	4x4 block
block_num2MvdL0	4	4x4 block
block_num3MvdL0	4	4x4 block
block_num0MvdL1	4	4x4 block
block_num1MvdL1	4	4x4 block
block_num2MvdL1	4	4x4 block
block_num3MvdL1	4	4x4 block

Table D.3: Input SEP FIFOs to CBP partition.

Data	Width	Read/Write Frequency
slice_type	2	slice
cbp_start_state	2	4x4 block
predType	1	macroblock
CodedBlockPatternLuma	4	macroblock
CodedBlockPatternChroma	2	macroblock
partial_condition_for_transform_8x8_flag	1	macroblock
mb_skip_flag	1	macroblock
end_of_slice_flag	1	macroblock
Intra_16x16	1	macroblock

Table D.4: Input SEP FIFOs to SIGMAP partition.

Data	Width	Read/Write Frequency
empty_slice	1	slice
ctxBlockCat	3	4x4 block
last_ctxBlockCat_in_slice	1	4x4 block

Table D.5: Input SEP FIFOs to COEFF partition.

Data	Width	Read/Write Frequency
empty_slice	1	slice
ctxBlockCat	3	4x4 block
numCoeff	7	4x4 block
last_ctxBlockCat_in_slice	1	4x4 block

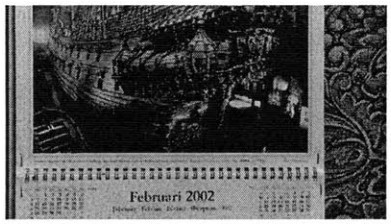
Appendix E

Video Test Bitstreams

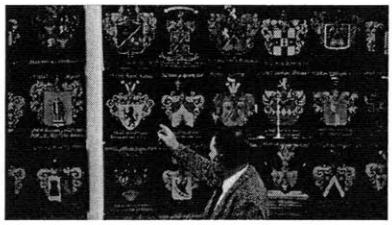
The nine HD video sequences used as test bitstreams in this thesis are described in this chapter of the appendix.

Table E.1: Properties of various encoded video sequences used as test bitstreams in this thesis.

Video Sequence	**Resolution**	**Frame Rate**	**Common Conditions**	**Bit-rate [Mbits/s]**	**Encoder**	**QP**	**PSNR [dB]**	**Profile Used**	**Chapter**
mobcal	1280x720	30	No	5.4	x264	23,26	n/a	baseline	2
parkrun	1280x720	30	No	26	x264	23,26	n/a	baseline	2
				0.44-4.64	JM12.0	22-37	36.54	high	4,5
shields	1280x720	30	No	7	x264	23,26	n/a	baseline	2
bigships	1280x720	30	Yes	1.16	JM12.0	22	39.13	high	3,4, 5
				0.42-0.45		27	36.14		3,4, 5
				0.18		32	33.37		3,4, 5
				0.09		37	30.73		3,4, 5
city	1280x720	30	Yes	0.65-38.91	JM12.0	22-37	29.93-38.77	high	3
crew	1280x720	30	Yes	0.90-18.54	JM12.0	22-37	33.85-40.48	high	3
night	1280x720	30	Yes	1.10-36.17	JM12.0	22-37	30.81-39.45	high	3
shuttle	1280x720	30	Yes	0.37-11.77	JM12.0	22-37	35.46-42.70	high	3
rally	4096x2160	30	Yes	9.04-9.10	JM12.0	27	41.78	high	3, 4, 5

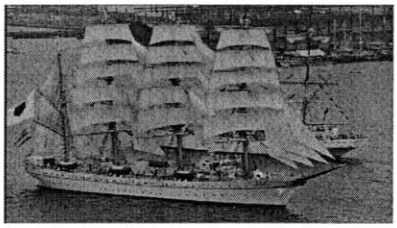


(a) mobcal



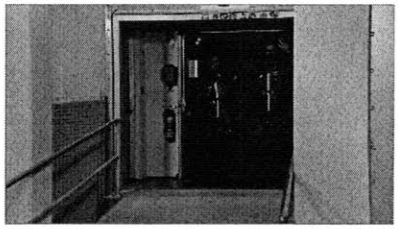
(b) shields

(c) parkrun



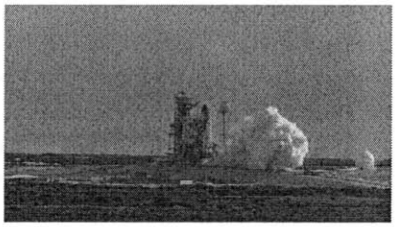
(d) bigships

(e) city

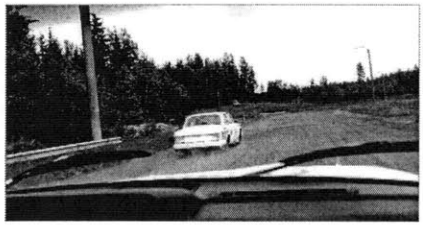


(f) crew

(g) night



(h) shuttle



(i) rally

Figure E-1: Screen captures of video test bitstreams.

Appendix F

Packaging for MP-CABAC

Due to the pad to chip area ratio, a 223-pin pin grid array (PGA) package was used. A QFP (quad flat package) could not be used due to the large cavity size (which occurs when the number of pins exceeds 208) resulting in long bond wires (lengths should be no more than 150-200 mils). A ZIF (zero insertion force) socket is used. The bonding diagram is shown in Fig. F-1. The 232 pins are reduced to 223 pins by sharing several power and ground connections.

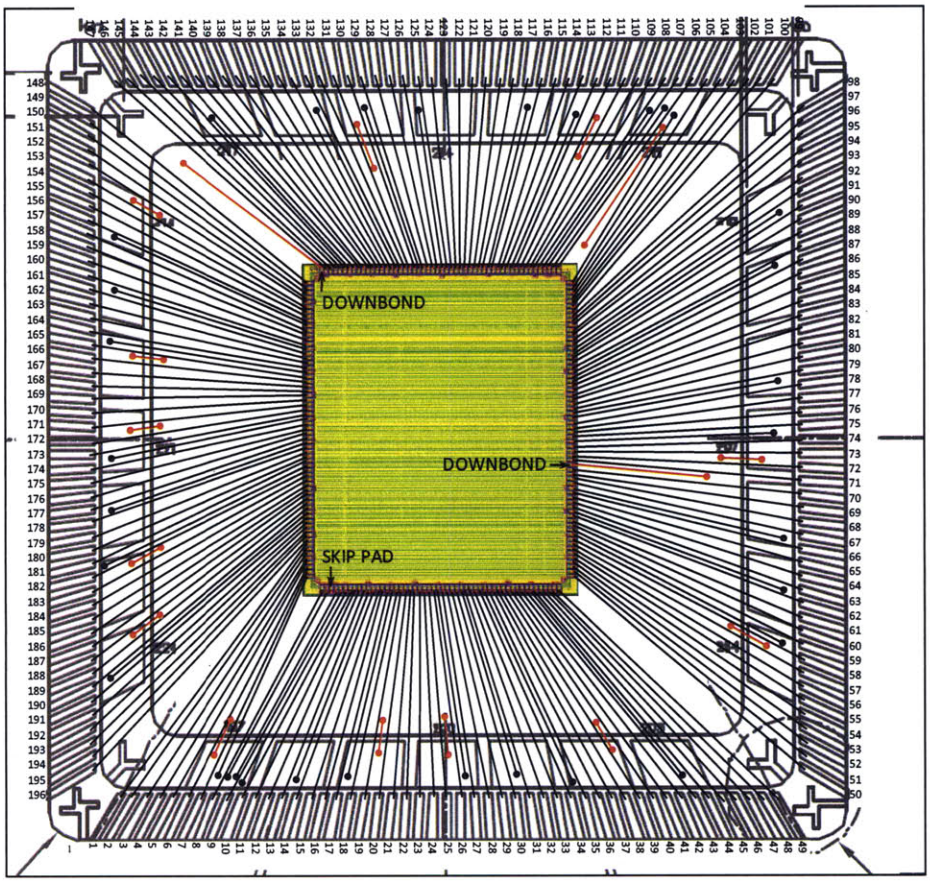


Figure F-1: Bonding diagram for the 223-PGA package. The package has two rings. The internal ring contains the power and ground pads, some of which are shared to reduce number of pins. The downbonds to the package substrate are highlighted in red.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, p. 114117, 1965.

[2] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles, "A 65nm 2-Billion-Transistor Quad-Core Itanium Processor," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2008, pp. 92 –598.

[3] "Emerging Markets Driving Growth in Worldwide Camera Phone Market ." [Online]. Available: http://www.infotrends.com/public/Content/Press/2008/07.29.2008.html

[4] "YouTube." [Online]. Available: www.youtube.com

[5] "Hulu." [Online]. Available: www.hulu.com

[6] S. Gupta, "High-Resolution Television Coming To A Cell-Phone Near You," January 2007. [Online]. Available: http://electronicdesign.com/article/digital/high-resolution-television-coming-to-a-cell-phone-.aspx

[7] "iPad Specifications." [Online]. Available: http://www.apple.com/ipad/specs/

[8] A. Hang, E. Rukzio, and A. Greaves, "Projector phone: a study of using mobile phones with integrated projector for interaction with maps," in *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services.* New York, NY, USA: ACM, 2008, pp. 207–216.

[9] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: Tools, Performance, and Complexity," *IEEE Circuits and Systems Magazine*, vol. 4, pp. 7– 28, 2004.

[10] "Panasonic Develops World's Largest 152-Inch Full HD 3D Plasma Display," January 2010. [Online]. Available: http://www.digitimes.com/news/a20100112PR207.html

[11] "Video Thing: Sharp demos 4Kx2K 64" experimental LCD," October 2006. [Online]. Available: http://videothing.blogspot.com/2006/10/sharp-demos-4kx2k-64-experimental-lcd.html

[12] "Toshiba Cell-based 4k x 2k TV," January 2009. [Online]. Available: http://www.itechnews.net/2009/01/08/toshiba-cell-based-4k-x-2k-tv/

[13] K. Mitani, M. Sugawara, H. Shimamoto, T. Yamashita, and F. Okano, "Ultrahigh-definition color video camera system with 8K x 4K pixels," *Journal of Electronic Imaging*, vol. 17, no. 2, p. 023014, 2008.

[14] S. Borkar, "Thousand core chips: a technology perspective," in *ACM IEEE Design Automation Conference*. New York, NY, USA: ACM, 2007, pp. 746–749.

[15] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," in *AFIPS Conference Proceedings*, 1967, p. 483485.

[16] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, April 1992.

[17] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, C. G. Sodini, Ed. Pearson Education, Inc., 2003.

[18] "BSIM4 Manual." [Online]. Available: http://www-device.eecs.berkeley.edu/bsim3/~bsim4.html

[19] Intel, "Embedded Voltage Regulator-Down (EmVRD) 11.0," January 2007. [Online]. Available: http://download.intel.com/design/intarch/designgd/31139505.pdf

[20] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 310 – 319, January 2005.

[21] J. Kwong, Y. K. Ramadass, N. Verma, and A. Chandrakasan, "A 65nm Sub-Vt Microcontroller with Integrated SRAM and Switched Capacitor DC-DC Converter," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 115–126, January 2009.

[22] B. Zhai, L. Nazhandali, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, D. Blaauw, and T. Austin, "A 2.60pJ/Inst subthreshold sensor processor for optimal energy efficiency," in *Symp. VLSI Circuits Dig. Tech. Papers*, June 2006, pp. 154–155.

[23] S. Hanson, B. Zhai, M. Seok, B. Cline, K. Zhou, M. Singhal, M. Minuth, J. Olson, L. Nazhandali, T. Austin, D. Sylvester, and D. S. Blaauw, "Performance and variability optimization strategies in a sub-200mV, 3.5pJ/inst, 11nW subthreshold processor," in *Symp. VLSI Circuits Dig. Tech. Papers*, June 2007, pp. 152–153.

[24] V. Sze and A. Chandrakasan, "A 0.4-V UWB Baseband Processor," in *IEEE International Symposium on Low Power Electronics and Design*, August 2007, pp. 262–267.

[25] "Technical Note TN2188 Exporting Movies for iPod, Apple TV and iPhone." [Online]. Available: http://developer.apple.com/technotes/tn2007/tn2188.html

[26] "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," ITU-T, Tech. Rep., 2003.

[27] "N10176: Call for test materials for future high-performance video coding standards development," *ISO/IEC JTC1/SC29/WG11*, October 2008.

[28] "VCEG-AL93:Draft "Request for Video Test Sequences for Enhanced Performance Video Coding Work" (for approval)," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), July 2009.

[29] A. Chandrakasan, D. Daly, D. Finchelstein, J. Kwong, Y. Ramadass, M. Sinangil, V. Sze, and N. Verma, "Technologies for Ultradynamic Voltage Scaling," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 191 –214, February 2010.

[30] V. Sze, D. F. Finchelstein, M. E. Sinangil, and A. P. Chandrakasan, "A 0.7-V 1.8mW H.264/AVC 720p Video Decoder," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 2943–2956, November 2009.

[31] V. Sze, M. Budagavi, A. P. Chandrakasan, and M. Zhou, "Parallel CABAC for Low Power Video Coding," in *IEEE International Conference on Image Processing*, October 2008, pp. 2096–2099.

[32] V. Sze and A. P. Chandrakasan, "A High Throughput CABAC Algorithm Using Syntax Element Partitioning," in *IEEE International Conference on Image Processing*, November 2009, pp. 773–776.

[33] D. Finchelstein, V. Sze, and A. Chandrakasan, "Multicore Processing and Efficient On-Chip Caching for H.264 and Future Video Decoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 11, pp. 1704–1713, November 2009.

[34] T. Tan, G. Sullivan, and T. Wedi, "VCEG-AE010: Recommended Simulation Common Conditions for Coding Efficiency Experiments Rev. 1," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), January 2007.

[35] V. Sze, M. Budagavi, and A. Chandrakasan, "VCEG-AL21: Massively Parallel CABAC," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), July 2009.

[36] D. F. Finchelstein, V. Sze, M. E. Sinangil, Y. Koken, and A. P. Chandrakasan, "A Low-Power 0.7-V H.264 720p Video Decoder," in *IEEE Asian Solid State Circuits Conference*, November 2008, pp. 173–176.

[37] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560– 576, July 2003.

[38] C.-D. Chien, C.-C. Lin, Y.-H. Shih, H.-C. Chen, C.-J. Huang, C.-Y. Yu, C.-L. Chen, C.-H. Cheng, and J.-I. Guo, "A 252kgate/71mW Multi-Standard Multi-Channel Video Decoder for High Definition Video Applications," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2007, pp. 282 – 283.

[39] S. Na, W. Hwangbo, J. Kim, S. Lee, and C.-M. Kyung, "1.8mW, Hybrid-Pipelined H.264/AVC Decoder For Mobile Devices," in *IEEE Asian Solid State Circuits Conference*, November 2007, pp. 192 – 195.

[40] T.-M. Liu, T.-A. Lin, S.-Z. Wang, W.-P. Lee, K.-C. Hou, J.-Y. Yang, and C.-Y. Lee, "A 125μW, Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 161–169, January 2007.

[41] C.-C. Lin, J.-W. Chen, H.-C. Chang, Y.-C. Yang, Y.-H. O. Yang, M.-C. Tsai, J.-I. Guo, and J.-S. Wang, "A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 170–182, January 2007.

[42] K. Kawakami, J. Takemura, M. Kuroda, H. Kawaguchi, and M. Yoshimoto, "A 50% Power Reduction in H.264/AVC HDTV Video Decoder LSI by Dynamic Voltage Scaling in Elastic Pipeline," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E89-A, no. 12, pp. 3642–3651, 2006.

[43] E. Fleming, C.-C. Lin, N. Dave, Arvind, G. Raghavan, and J. Hicks, "H.264 Decoder: A Case Study in Multiple Design Points," *Proceedings of Formal Methods and Models for Co-Design, (MEMOCODE)*, pp. 165–174, June 2008.

[44] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 704– 716, July 2003.

[45] S.-Z. Wang, T.-A. Lin, T.-M. Liu, and C.-Y. Lee, "A New Motion Compensation Design for H.264/AVC Decoder," in *IEEE International Symposium on Circuits and Systems*, vol. 5, May 2005, pp. 4558–4561.

[46] D. F. Finchelstein, "Low-Power Techniques for Video Decoding," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[47] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 614 –619, july 2003.

[48] K. Xu and C.-S. Choy, "A Five-Stage Pipeline, 204 Cycles/MB, Single-Port SRAM-Based Deblocking Filter for H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 363–374, March 2008.

[49] V. Gutnik and A. P. Chandrakasan, "Embedded Power Supply for Low-Power DSP," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, pp. 425–435, December 1997.

[50] K. Choi, K. Dantu, W. Cheng, and M. Pedram, "Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder," in *IEEE/ACM International Conference on Computer Aided Design*, November 2002, pp. 732 – 737.

[51] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips, "Power-aware video decoding," in *22nd Picture Coding Symposium*, 2001.

[52] A. C. Bavier, A. B. Montz, and L. L. Peterson, "Predicting MPEG execution times," in *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems.* New York, NY, USA: ACM, 1998, pp. 131–140.

[53] E. Akyol and M. van der Schaar, "Complexity Model Based Proactive Dynamic Voltage Scaling for Video Decoding Systems," *IEEE Transactions on Multimedia*, vol. 9, no. 7, pp. 1475–1492, November 2007.

[54] C. Im, H. Kim, and S. Ha, "Dynamic voltage scheduling technique for low-power multimedia applications using buffers," in *IEEE International Symposium on Low Power Electronics and Design*, August 2001, pp. 34–39.

[55] M. E. Sinangil, N. Verma, and A. P. Chandrakasan, "A Reconfigurable 65nm SRAM Achieving Voltage Scalability from 0.25-1.2V and Performance Scalability from 20kHz-200MHz," in *IEEE European Solid-State Circuits Conference*, September, 2008, pp. 282–285.

[56] "Cypress SRAM Datasheet." [Online]. Available: http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1470v25_8.pdf

[57] "6.111 Labkit Documentation." [Online]. Available: http://www-mtl.mit.edu/Courses/6.111/labkit/

[58] "FXL4245 Datasheet." [Online]. Available: www.fairchildsemi.com/pf/FX/FXL4245.html

[59] "x264 Encoder Software." [Online]. Available: http://www.videolan.org/developers/x264.html

[60] T.-D. Chuang, P.-K. Tsung, L.-M. C. Pin-Chih Lin, T.-C. Ma, Y.-H. Chen, and L.-G. Chen, "A 59.5 Scalable/Multi-View Video Decoder Chip for Quad/3D Full HDTV and Video Streaming Applications," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2010, pp. 330–331.

[61] K. Hardee, F. Jones, D. Butler, M. Parris, M. Mound, H. Calendar, G. Jones, L. Aldrich, C. Gruenschlaeger, M. Miyabayashil, K. Taniguchi, and I. Arakawa, "A 0.6V 205MHz 19.5ns tRC 16Mb embedded DRAM," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2004, pp. 200–201.

[62] Y. Kikuchi, M. Takahashi, T. Maeda, H. Hara, H. Arakida, H. Yamamoto, Y. Hagiwara, T. Fujita, M. Watanabe, T. Shimazawa, Y. Ohara, T. Miyamori, M. Hamada, M. Takahashi, and Y. Oowaki, "A 222mW H.264 Full-HD Decoding Application Processor with x512b Stacked DRAM in 40nm," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2010, pp. 326–327.

[63] "Mobile Platform Display Technology Advancements." [Online]. Available: http://developer.intel.ru/download/design/mobile/Presentations/MOB165PS.pdf

[64] O. Prache, "Active matrix molecular OLED microdisplays," *Displays*, vol. 22, no. 2, pp. 49–56, May 2001.

[65] Texas Instruments and Nokia and Polycom and Samsung AIT and Tandberg, "T05-SG16-C-0215: Desired features in future video coding standards," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), June 2007.

[66] V. Sze, M. Budagavi, and M. U. Demircin, "VCEG-AJ31: CABAC throughput requirements for real-time decoding," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), October 2008.

[67] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620– 636, July 2003.

[68] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[69] D. Marpe and T. Wiegand, "A highly efficient multiplication-free binary arithmetic coder and its application in video coding," in *IEEE International Conference on Image Processing*, vol. 2, September 2003, pp. II – 263–6 vol.3.

[70] J. Ribas-Corbera, P. Chou, and S. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 674 – 687, july 2003.

[71] W. Yu and Y. He, "A High Performance CABAC Decoding Architecture," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 4, pp. 1352–1359, November 2005.

[72] C.-H. Kim and I.-C. Park, "Parallel Decoding of Context-Based Adaptive Binary Arithmetic Codes Based on Most Probably Symbol Prediction," *IEICE - Trans. on Information and Systems*, vol. E90-D, no. 2, pp. 609–612, February 2007.

[73] J.-H. Lin and K. Parhi, "Parallelization of Context-Based Adaptive Binary Arithmetic Coders," *IEEE Transactions on Signal Processing*, vol. 54, no. 10, pp. 3702 –3711, October 2006.

[74] V. Sze and M. Budagavi, "T05-SG16-C-0334: Parallel CABAC," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), April 2008.

[75] J. Zhao and A. Segall, "COM16-C405: Entropy slices for parallel entropy decoding," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), April 2008.

[76] ——, "VCEG-AI32: New Results using Entropy Slices for Parallel Decoding," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), July 2008.

[77] X. Guo, Y.-W. Huang, and S. Lei, "VCEG-AK25: Ordered Entropy Slices for Parallel CABAC," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), April 2009.

[78] G. Bjøntegaard, "VCEG-M33: Calculation of Average PSNR Differences between RD curves," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), April 2001.

[79] "KTA Reference Software, kta2.7." [Online]. Available: http://iphome.hhi.de/suehring/tml/download/KTA/

[80] C.-H. Kim and I.-C. Park, "High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction," in *IEEE International Symposium on Circuits and Systems*, May 2006, pp. 1707–1710.

[81] Y.-C. Yang, C.-C. Lin, H.-C. Chang, C.-L. Su, and J.-I. Guo, "A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding with Look Ahead Parsing," in *IEEE International Conference on Multimedia and Expo*, July 2006, pp. 357 –360.

[82] Y. Yi and I.-C. Park, "High-Speed H.264/AVC CABAC Decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 4, pp. 490 –494, April 2007.

[83] B. Shi, W. Zheng, H.-S. Lee, D.-X. Li, and M. Zhang, "Pipelined Architecture Design of H.264/AVC CABAC Real-Time Decoding," in *IEEE International Conference on Circuits and Systems for Communications*, May 2008, pp. 492 –496.

[84] Y.-C. Yang and J.-I. Guo, "High-Throughput H.264/AVC High-Profile CABAC Decoder for HDTV Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 9, pp. 1395 –1399, September 2009.

[85] P. Zhang, D. Xie, and W. Gao, "Variable-bin-rate CABAC engine for H.264/AVC high definition real-time decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 3, pp. 417–426, 2009.

[86] J.-W. Chen and Y.-L. Lin, "A high-performance hardwired CABAC decoder for ultra-high resolution video," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1614 –1622, August 2009.

[87] Y.-H. Liao, G.-L. Li, and T.-S. Chang, "A High Throughput VLSI Design with Hybrid Memory Architecture for H.264/AVC CABAC Decoder," in *IEEE International Symposium on Circuits and Systems (Abstract Only)*, June 2010.

[88] C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design." [Online]. Available: www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

[89] "Opal Kelly XEM3050 Board." [Online]. Available: http://www.opalkelly.com/products/xem3050/

[90] M. Budagavi, V. Sze, M. U. Demircin, S. Dikbas, M. Zhou, and A. P. Chandrakasan, "JCTVC-A101:Description of video coding technology proposal by Texas Instruments Inc." Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, April 2010.

[91] H.-C. Chang, J.-W. Chen, B.-T. Wu, C.-L. Su, J.-S. Wang, and J.-I. Guo, "A Dynamic Quality-Adjustable H.264 Video Encoder for Power-Aware Video Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 12, pp. 1739 –1754, December 2009.

[92] Z. He, W. Cheng, and X. Chen, "Energy Minimization of Portable Video Communication Devices Based on Power-Rate-Distortion Optimization," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 5, pp. 596 –608, may 2008.

[93] S. Pateux and J. Jung, "VCEG-AE07: An Excel Add-In for Computing Bjntegaard Metric and Its Evolution," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), January 2007.