Low-Power Techniques for Video Decoding

by

Daniel Frederic Finchelstein

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

Doctor of Philosophy in Electrical and Computer Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2009

Certified by .....
Anantha Chandrakasan
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

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

Acknowledgments

I want to thank the following people and organizations. If I forgot anyone, please give me a call.

- my family, in order of increasing age, for shaping me and helping me during the time we lived together: "Little-Little", "Fuxea", "Beige", Marica, "Ruby", Ina, mama, "Tica"
- the rest of the family whom I have had the pleasure of associating with, also in order of increasing age: Sanskaar, Kanako, Irina, Theo, Iolanda, Misae, Tesfaye
- all the friends I've made while at MIT, hoping that we won't lose touch
- all the team members from volleyball and soccer, for chasing together with me a simple dream (the ball)
- all the members of Anantha's group for their friendship and technical advice
- Anantha for creating such a collaborative and non-competitive culture among his students
- Arvind and his students for their stimulating discussions on computer architecture and hardware design languages
- Nokia and Texas Instruments for research funding and chip fabrication facilities
- people at Nokia Research Cambridge for their guidance (Jamey, John, and Gopal)
- people throughout Texas Instruments for listening to many of my presentations (Dennis, Alice, ...)

- Vivienne Sze who was involved in many of the ideas described in this thesis, and she is acknowledged in each of the chapters containing her contributions
- Ersin Sinangil who designed the low-voltage SRAMs used for the on-chip caches of Chapter 5

Contents

List of Figures

List of Tables

Acronyms

ADD addition of residual to prediction

ASIC application-specific integrated circuit

BW bandwidth

CABAC context-adaptive binary arithmetic coding

CAVLC context-adaptive variable-length coding

CAVLD context-adaptive variable-length decoding

CMOS complementary metal-oxide semiconductor

DB de-blocking filter

DCT discrete cosine transform

DEC video decoder

DFC data-forwarding cache

DRAM dynamic random-access memory

DVFS dynamic voltage and frequency scaling

ED entropy decoder

eDRAM embedded dynamic random-access memory

ENC video encoder

FB frame buffer

FIFO first-in-first-out register

FIR finite-impulse-response filter

FLLC full-last-line cache

fps frames per second

FPGA field-programmable gate array

FO4 fanout-of-4 delay

GOP group of pictures

IDCT inverse discrete cosine transform

IES interleaved entropy slice

INTRA spatial prediction

IT inverse transform

LCD liquid crystal display

LFC last-frame cache

LSB least significant bit

LUT look-up table

MB macroblock

MC motion compensation

ME motion estimation

MEM memory controller

MV motion vector

OCFB off-chip frame buffer

OLED organic light-emitting device

PLL phase-locked loop

PSNR peak signal-to-noise ratio

QD-OLED quantum dot-organic light-emitting device

QP quantization parameter

ROM read-only memory

SRAM static random-access memory

VLC variable-length coding

WB write-back buffer

Low-Power Techniques for Video Decoding

by

Daniel Frederic Finchelstein

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Abstract

The H.264 video coding standard can deliver high compression efficiency at a cost of large complexity and power. The increasing popularity of video capture and playback on portable devices requires that the energy of the video processing be kept to a minimum. This work implements several architecture optimizations that reduce the system power of a high-definition video decoder.

In order to decode high resolutions at low voltages and low frequencies, we employ techniques such as pipelining, unit parallelism, multiple cores, and multiple voltage/frequency domains. For example, a 3-core decoder can reduce the required clock frequency by 2.91 $\times$, which enables a power reduction of 61% relative to a full-voltage single-core decoder. To reduce the total memory system power, several caching techniques are demonstrated that can dramatically reduce the off-chip memory bandwidth and power at the cost of increased chip area. A 123 kB data-forwarding cache can reduce the read bandwidth from external memory by 53%, which leads to 44% power savings in the memory reads.

To demonstrate these low-power ideas, a H.264/AVC Baseline Level 3.2 decoder ASIC was fabricated in 65 nm CMOS and verified. It operates down to 0.7 V and has a measured power down to 1.8 mW when decoding a high definition 720p video at 30 frames per second, which is over an order of magnitude lower than previously published results.

Thesis Supervisor: Anantha Chandrakasan
Title: Professor of Electrical Engineering and Computer Science

Chapter 1

Introduction

We begin by describing why low-power is important for certain video applications. We also give a preview of how to reduce the power for a hardware video decoder. We then identify the key contributions of this work. To help introduce the reader to some of the foundations of video decoding, we describe the basic blocks of the H.264 video coding standard. Finally, this introductory chapter concludes with a literature survey of published works related to this thesis.

1.1 Motivation for Low-Power Video

Mobile multimedia devices such as smart phones are energy-constrained, so reducing their power is critical for extending video playback times. The goal of this thesis is to explore different power saving techniques for video decoders and demonstrate them on a hardware application-specific integrated circuit (ASIC) architecture. The first of these techniques is to use pipelining and parallelism to enable lower frequencies and supply voltages. The second is the efficient scheduling and caching of memory operations to reduce the access power of on-chip and off-chip memories. This chapter introduces these techniques and also describes how they relate to previously-published ideas.

1.1.1 Voltage Scaling for Low Power

The power usage of a given digital system can be minimized by lowering the supply voltage [1]. First, the decoder's clock frequency is set to the lowest value that still guarantees that the current computation workload can be met. Next, the supply voltage is reduced to the minimal value that still allows the circuit to operate at the chosen frequency. Equation 1.1 shows the energy required for a digital computation. The total energy E_{tot} is broken down into the dynamic energy E_{dyn} and the leakage energy E_{leak}. Voltage scaling reduces dynamic energy consumption by a quadratic factor, as shown in Equation 1.1; E_{dyn} is dynamic energy, C_{eff} is the effective total switched capacitance and V_{DD} is the supply voltage. Leakage energy is computed as the leakage power integrated over the total time of the computation T_{comp}. The leakage power is obtained from the subthreshold current formula with the gate-source voltage V_{GS} set to 0 and the drain-source voltage V_{DS} set to V_{DD}; I_S is the maximum leakage current and V_{th} is the thermal voltage. The leakage power also decreases with V_{DD}, but the computation time T_{COMP} varies inversely proportional with V_{DD}. Therefore, as V_{DD} decreases, the leakage energy first decreases slightly, but then begins to increase since T_{COMP} eventually grows faster than the leakage power decays.

$$
\begin{aligned}
E_{dyn} &= C_{eff} \times V_{DD}^2 \\
E_{leak} &= T_{comp}(V_{DD}) \times V_{DD} \times I_S \left(1 - e^{-\frac{V_{DD}}{V_{th}}}\right) \\
E_{tot} &= E_{dyn} + E_{leak}
\end{aligned}
\tag{1.1}
$$

Equation 1.2 shows how the propagation delay of static complementary metal-oxide semiconductor (CMOS) circuit varies with the supply voltage V_{DD}. The delay t_P is proportional to the supply voltage since V_{DD} is the amount of voltage that must be charged or drained to signal a 1 or a 0. The delay t_P is also directly proportional to the total signal capacitance C being switched. Finally, the delay varies inversely proportional with the switching current I_D, since a larger current speeds up the switching operation.

$$t_p = \frac{CV_{DD}}{I_D(V_{DD})} \tag{1.2}$$

The main cost of scaling down the voltage is an increased circuit delay, as the currents decrease with supply voltage. Specifically, the circuit suffers a linear increase in delay above the threshold voltage, as shown by the current dependence of Equation 1.3 ([2]); I_{D-max} is the maximum transistor on-current, υ_{sat} is the velocity-saturated mobility, C_{OX} is the unit oxide capacitance, W is the transistor width, V_T is the transistor threshold voltage, V_{DSAT} is the velocity-saturation voltage, and λ is the channel length modulation coefficient. As the supply voltage approaches the sub-threshold region and below ($V_{DD} < V_T$), the circuit begins to experience an exponential increase in delay, as shown in Equation 1.4; I_S and n are fitting parameters, and V_{th} is the thermal voltage. This can be seen on the left side of Figure 1-1, where the circuit delay increases exponentially along with the leakage component of total energy. This decreased speed can be a challenge for real-time applications such as video decoding where on average a new frame must be computed every 33 ms for frame rates of 30 fps.

$$I_{D-max} = \upsilon_{sat} C_{OX} W (V_{DD} - V_T - \frac{V_{DSAT}}{2})(1 + \lambda V_{DD}) \tag{1.3}$$

$$I_{D-max} = I_S e^{\frac{V_{DD}}{nV_{th}}} \left(1 - e^{-\frac{V_{DD}}{V_{th}}}\right) \tag{1.4}$$

Pipelining and parallelism, two well-known hardware architecture techniques, can be used to maximize concurrency. This increased performance can be exploited to lower the supply voltage, bringing the circuit back to the original performance but drawing less dynamic energy [1]. This is the key concept used in this thesis to lower the voltage and power of a video decoder. Pipelining increases computation concurrency by reducing the datapath between registers. This allows a circuit to be clocked at a higher frequency, and thus process data faster. One disadvantage of pipelining is the increase in pipeline registers and control complexity. Parallelism increases concurrency by distributing computation among several identical hardware units. For example, if a hardware unit is duplicated, the latency of each

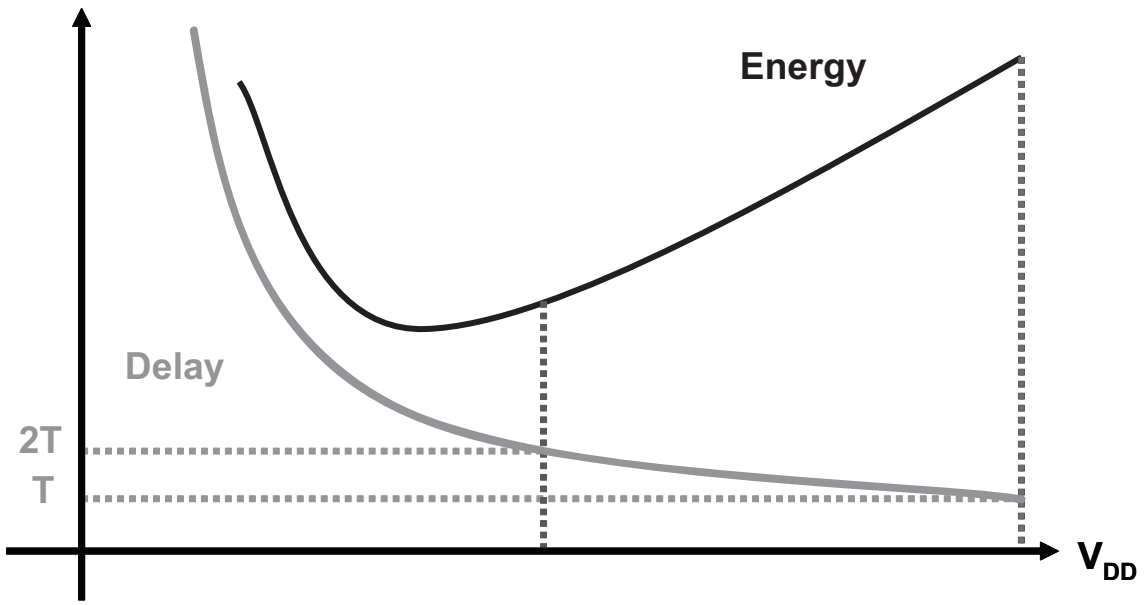


Figure 1-1: Parallelism of 2 blocks allows each block to tolerate double the latency for a given throughput and run at a lower voltage.

individual unit can increase by a factor of 2; this allows each of the units to run at a lower voltage, as shown in Figure 1-1. The main cost of parallelism is an increase in chip area and additional muxing/de-muxing logic to feed all the units and collect their results.

1.1.2 Memory Optimization for Low Power

Video processing also requires a significant amount of on-chip and off-chip memory bandwidth, for both motion compensation (MC) and last-line accessing. Therefore, memory system optimization can reduce total power in the video decoder (DEC) system, which includes both the decoder ASIC and the off-chip frame buffer (OCFB) memory. One effective way to reduce memory power is the use of on-chip caching. This technique trades off an increase in chip area for a reduction in more power-hungry off-chip accesses. The cache hit rate must be high enough so that the added power overhead of cache lookups and cache writes does not outweigh the saving in off-chip memory power.

1.2 Outline of Main Contributions

This section outlines the main contributions and distinguishing ideas of this thesis. Many of the ideas are complementary and can be used together on the same video decoder (DEC) implementation. A portion of the work presented in this thesis was done in collaboration with another doctoral student Vivienne Sze. She was heavily involved with the design of the H.264 ASIC as well as the development of some of the multi-core and caching ideas. The sections containing her contributions will be cited at the beginning of each chapter.

Pipelining and Unit-Level Parallelism (Chapter 2)

This thesis presents a pipelined architecture which separates the luma and chroma processing into two different pipelines and operates on 4x4 blocks of 16 pixels. This architecture allows the different hardware units in the DEC to be active during most clock cycles. Within this pipeline, we study the effect of varying the FIFO depths between the different DEC units. We find that deeper FIFOs can increase performance by 25% over using single-stage FIFOs, because they reduce stalls by averaging the workload variation within the pipeline stages. Parallelism is demonstrated within all the pipeline units to reduce cycles per 4x4 pixel block and speed up the pipeline throughput. For example, parallel architectures of up to 20 MC interpolators and 4 de-blocking filter (DB) filters are presented. We present a modified inverse transform (IT) algorithm (not compatible with H.264) that uses precision-scaling arithmetic. This allows power to be used as a third knob next to bitrate and image distortion for future video coding.

Multi-Core Decoding (Chapter 4)

This thesis presents three different multi-core DEC architectures. Due to the regularity of the designs, video performance can be achieved with very little design time by instantiating and connecting multiple copies of the same DEC. When replicating N DEC instances, the total cycle count goes down by approximately a factor of N. The parallel DECs can work on either multiple slices in one frame, or multiple consecutive frames. For slice processing, each

frame can either be broken up into H.264 slices or into interleaved entropy slices (IESs), the latter providing several advantages in terms of area and memory efficiency.

Memory Optimization (Chapter 5)

This work shows how memory accesses to typical full-last-line caches (FLLCs) can be reduced when using interleaved entropy slice (IES) processing. We also show several categories of on-chip MC caches which trade off cache area for total memory power savings. For example, a last-frame cache (LFC) can eliminate most off-chip reads by keeping a large on-chip cache, while data-forwarding caches (DFCs) together with N parallel frame DECs can eliminate up to $(N-1)/N$ of the of-chip reads. Similarly, using N parallel IES DECs replaces $(N-1)/N$ of the accesses to the FLLC with reads and writes to very small FIFOs. We use a memory power model to estimate and compare the power savings of the different on-chip caching techniques we present.

Prototype Video Decoder ASIC (Chapter 6)

Based on the low-power ideas described in this thesis, we built and demonstrated a real-time H.264 720p video decoder ASIC. This chip uses over 10x less power than previously-published results. The ASIC shows the benefits of splitting the design into multiple voltage and frequency domains. This allows each domain to operate at its minimum voltage and frequency. As a result we can reduce the power by 25% and 29% when separating one domain into two or three domains respectively. We also show how running dynamic voltage and frequency scaling (DVFS) on the DEC can reduce the operating power for videos with varying workloads. This shows a 25% improvement over a static control scheme where the DEC voltage and frequency are set to handle the maximum possible workload. The ASIC does not use any of the multi-core techniques described in Chapter 4 and Chapter 5.

1.3 The H.264 Video Codec

Before delving into low-power implementation details for DECs, it is important to give a basic description of the video algorithm.

1.3.1 H.264 Overview

The H.264 video standard was introduced in 2004 [3]. Its main purpose is to provide an increase in compression efficiency over previous standards such as MPEG2 [4]. The H.264 decoding flowchart, shown in Figure 1-2, is very similar to previous standards (MPEG-1, MPEG-2), and operates on units as small as 4x4 pixels.

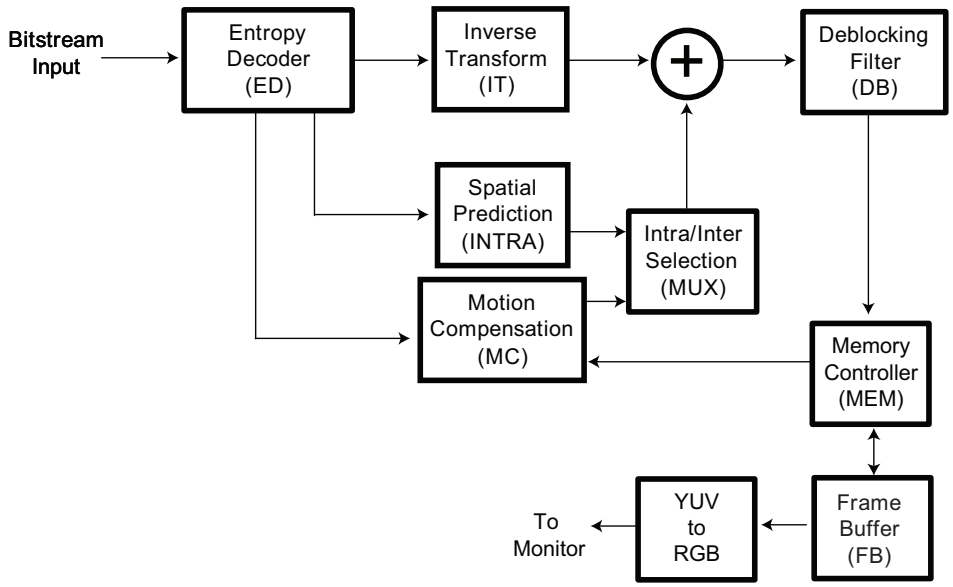


Figure 1-2: H.264 algorithm flowchart

Each pixel has three components: one luma (Y) and two chroma (U and V), each of which is processed separately. This format is different than Red/Green/Blue (RGB), which is used to display pixels, for example on a liquid crystal display (LCD). Therefore, before displaying the pixels, they must be converted from YUV to RGB using a matrix transform.

This thesis will focus on a video decoder that can handle various resolutions, from low to high definition, as shown in Table 1.1. There are also several variations (called profiles) of

the H.264 standard, which target different applications [5]. For example, the baseline profile targets low-cost applications, such as video-conferencing and mobile devices, which have limited computing resources. Another variation, the high profile, is intended for broadcasting and storage applications, where compression efficiency is the most important concern. To achieve the extra compression efficiency, the high profile uses more computation-intensive techniques such as context-adaptive binary arithmetic coding (CABAC), interlaced coding, monochrome format, bidirectional slices, and 8x8 transforms. The decoder discussed in this thesis only targets baseline profile videos, so it will not support some of the more advanced features.

Table 1.1: Video resolutions and frame rates

Resolution Name	**Frames per second**	**Width [Pixels]**	**Height [Pixels]**	**MegaPixels per second**	**Normalized Throughput**
QCIF	15	176	144	0.38	1
CIF	30	352	288	3.04	8
D1	30	720	480	10.4	27
720p	30	1280	720	27.6	73
1080p	30	1920	1080	62.2	164

The following sections briefly describe the function of each of the major components of the H.264 video decoder. This should provide the reader with enough details of the video codec so that the power-saving techniques described later will be more easily understood.

1.3.2 Entropy Decoder (ED)

In a H.264 video decoder (DEC), the encoded bitstream is serially parsed by the entropy decoder (ED), which produces configuration parameters, discrete cosine transform (DCT) coefficients, and prediction modes (spatial or temporal) for each 4x4 block of pixels. There are two entropy coding options for the H.264 standard: context-adaptive variable-length coding (CAVLC) and context-adaptive binary arithmetic coding (CABAC). The baseline profile of the H.264 standard uses CAVLC. The high profile uses CABAC, which offers a 10-15% coding gain over CAVLC, at the cost of increased computation complexity. This

thesis only explores the baseline profile, so only context-adaptive variable-length decoding (CAVLD) is implemented.

To illustrate the operation of the CAVLD, a sample set of DCT coefficients is shown in Figure 1-3. These coefficients have been quantized at the video encoder (ENC), so that many of the coefficients received by the DEC are zero-valued. Also note that most of the DCT coefficients are clustered at the lower frequencies, or lower indices of k_1 and k_2. This is generally true because typical images have most of their energy content located in lower frequency bands (there are fewer edges than smooth areas). This yields longer runs of zero coefficients at the higher frequencies, which can be more efficiently coded. The non-zero coefficients of Figure 1-3 are broken up into trailing coefficients with absolute value of 1, and the rest. Each non-zero coefficient is encoded into the bitstream, in the order given by the zig-zag scanning order (gray winding arrow). The locations of the coefficients (frequency indices of k_1 and k_2) are encoded into the bitstream by transmitting the run-length of zeros between consecutive non-zero coefficients.

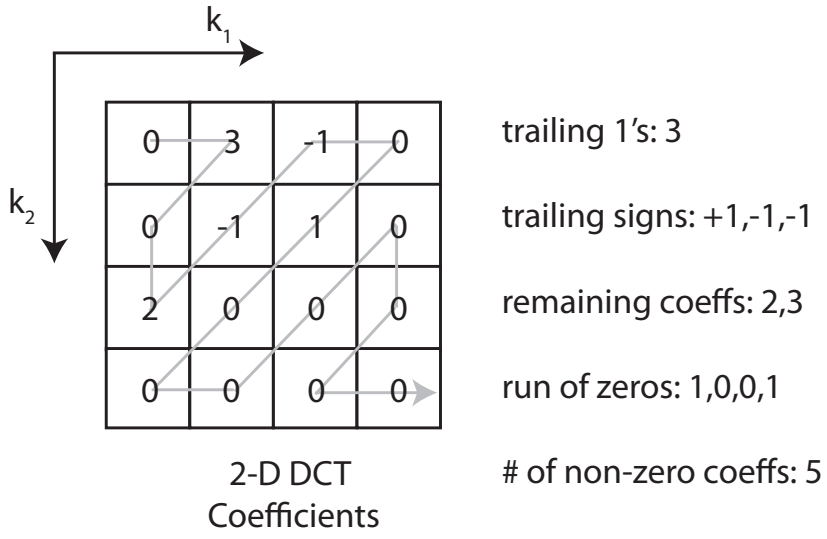


Figure 1-3: Decoding quantized DCT coefficients

CAVLC uses variable-length coding (VLC) to provide good compression by assigning shorter codes to more probable symbols. These codewords are either computed according to a fixed algorithm or are stored in code tables. For coding DCT coefficients, CAVLC uses

different code tables depending on the context, hence the context-adaptive name. For coding other syntax elements, such as motion vectors (MVs), CAVLC uses a fixed (non-adaptive) algorithm called exp-Golomb, which is described in Table 1.2. The symbols are enumerated in order of decreasing probability, with $symbol_0$ being the most probable.

Table 1.2: Exp-Golomb mapping between symbols and variable-length codes

Bit String Form	**Symbol Index**	**Size of Range**
1	0	1
$01x_0$	1-2	2
$001x_1x_0$	3-6	4
$0001x_2x_1x_0$	7-14	8
$00001x_3x_2x_1x_0$	15-30	16

1.3.3 Inverse Transform and Quantization (IT)

The inverse transform (IT) unit takes in a set of 4x4 discrete cosine transform (DCT) coefficients, as shown in Figure 1-3, and performs the inverse discrete cosine transform (IDCT) along with some pre-scaling and post-scaling. It produces a 4x4 block of pixels which can be added to the predicted block to get the final decoded block. The transformation is done using an integer-based approximation of the IDCT, as shown in Equation 1.5. The 4x4 matrix X represents a scaled version of the residual in the 2-dimensional space domain, while the 4x4 matrix Y is a scaled version of the coefficients in the 2-dimensional frequency domain. The two other 4x4 matrices are used to perform an approximate 2-dimensional IDCT.

$$X = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} Y_{00} & Y_{01} & Y_{02} & Y_{03} \\ Y_{10} & Y_{11} & Y_{12} & Y_{13} \\ Y_{20} & Y_{21} & Y_{22} & Y_{23} \\ Y_{30} & Y_{31} & Y_{32} & Y_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \tag{1.5}$$

1.3.4 Intra Spatial Prediction (INTRA)

The spatial prediction (INTRA) unit exploits the spatial redundancy found in still images to predict pixels in frames generally found at the start of a new scene which have no temporal redundancy. Using the already-decoded neighboring pixels above and to the left, it can predict luma blocks of size 4x4 and 16x16, and chroma blocks of size 8x8. The chroma resolution is half of the luma resolution in each dimension, as the 4:2:0 format is used. Since the blocks are processed in raster-scan order, the right and bottom neighboring pixels cannot be used for prediction since they have not yet been decoded. Figure 1-4 shows the neighboring pixels used to predict a 4x4 luma block.

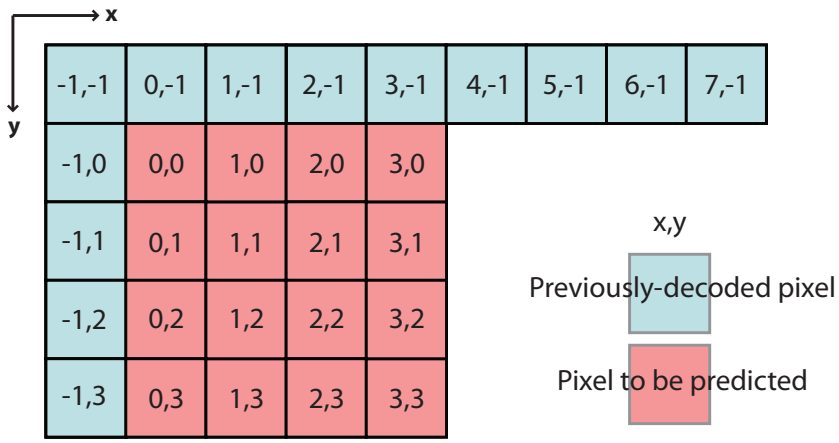


Figure 1-4: 4x4 luma block spatially predicted from its left, top-left, top, and top-right neighbors, which are already decoded

Each 4x4 luma block is predicted from its left, top, and top-right neighboring pixels. There are 9 directional prediction modes, such as vertical, horizontal, DC, and other directions. For example, the formula for the Diagonal-Down-Right (DDR) prediction mode uses a 3-tap finite-impulse-response filter (FIR), as shown in Equation 1.6. The 4x4 block is predicted from the previously-decoded pixels to the left (x-coordinate=-1) and above (y-coordinate=-1). Luma pixels can also be intra-predicted in blocks of 16x16, with one of four prediction modes: horizontal, vertical, planar, and a DC average. The chroma prediction modes for 8x8 blocks are identical to the modes of 16x16 luma intra.

if x>y,

$$pred4x4_L[x,y] = (p[x-y-2,-1]) + 2 \times p[x-y-1,-1] + p[x-y,-1] + 2)/4$$

if x<y,

$$pred4x4_L[x,y] = (p[-1,y-x-2]) + 2 \times p[-1,y-x-1] + p[-1,y-x] + 2)/4 \quad (1.6)$$

if x==y,

$$pred4x4_L[x,y] = (p[0,-1] + 2 \times p[-1,-1] + p[-1,0] + 2)/4$$

An example of a 4x4 luma intra prediction is shown in Figure 1-5. The prediction mode is DDR, so the prediction uses the equations of Equation 1.6. Note how the predicted pixels along every diagonal arrow are equal in value. Also note that their value is similar to the previously-decoded pixel on the same diagonal, since a weighted 3-tap FIR is used.

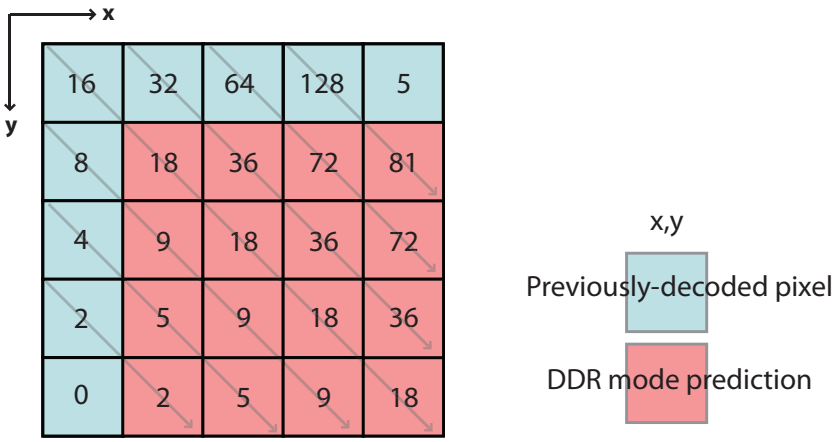


Figure 1-5: Example of 4x4 luma block intra predcited using Down-Down-Right mode

1.3.5 Motion Compensation (MC)

The motion compensation (MC) unit uses pixels from previously decoded frames along with corresponding motion vectors to predict the current 4x4 block. When the motion vectors are integer-valued, the predicted 4x4 block can be found in its entirety in a previous frame, as shown in the left part of Figure 1-6. However, when either the X or Y component of

the motion vector is fractional, the predicted 4x4 block must be interpolated from integer-location pixels in previous frames, as shown in the right part of Figure 1-6. The luma interpolating FIR (Equation 1.7) has 6 taps (better coding efficiency than a 2-tap FIR) so a 4x4 block is predicted from an area of at most 9x9 pixels. An invariant 6-tap Wiener FIR provides an improvement over a 2-tap FIR because it better approximates an ideal low-pass filter [6].

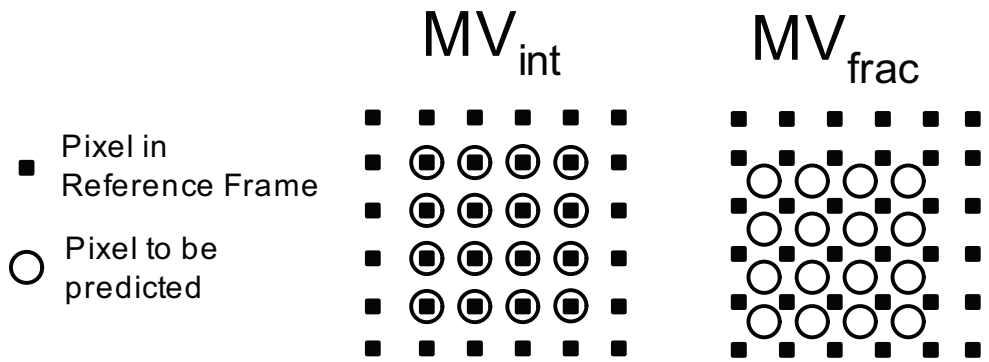


Figure 1-6: Integer and fractional motion vectors

$$pred[2.5] = (p[0] - 5 \times p[1] + 20 \times p[2] + 20 \times p[3] - 5 \times p[4] + p[5])/32 \tag{1.7}$$

The fractional chroma pixels are predicted using a simpler bidirectional filter. There is a separate MV for each 2x2 block of chroma pixels. Each fractional-location pixel (with 1/8th of an integer resolution) is interpolated from its integer-location neighbors: top-left (TL), top-right (TR), bottom-left (BL) and bottom-right (BR), as shown in Figure 1-7. The interpolation uses the equation in Equation 1.8, where dx and dy are 3-bit numbers representing the fractional portion of the MVs. To interpolate a block of 2x2 pixels, a 3x3 block is needed.

$$\begin{aligned} pred[dx, dy] = ((8 - dx) \times (8 - dy) \times TL + dx \times (8 - dy) \times TR + \\ (8 - dx) \times dy \times BL + dx \times dy \times BR + 32)/64 \end{aligned} \tag{1.8}$$

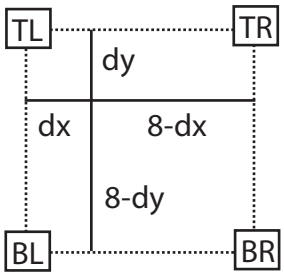


Figure 1-7: Fractional-location chroma pixel is interpolated from its integer-location neighbors

1.3.6 Deblocking Filter (DB)

A side effect of processing a frame in 4x4 blocks is that there can be some visible discontinuities along the edges of these small blocks. The de-blocking filter (DB) unit smooths these artificial discontinuities and thus improves the perceived image quality. The DB filter is adaptive by design [7]. The choice of whether to filter an edge or not depends on the pixel values across the edge. For example, if the gradient across the edge exceeds a certain threshold, it is assumed that the sharp edge is part of the original image and no filter is applied. This avoids unintended blurring in the original video. Alternatively, if the gradient across the edge is smaller, a filter operation is applied. The type of filtering applied across the edge depends on the type of edge. The most likely location for a blocking artifact is the boundary between two different intra-coded MBs.

The filtering operation is performed by a finite-impulse-response filter (FIR) with up to 5 taps, depending on the adaptive filter strength, as shown in Figure 1-8. The strongest of these filters, the 5-tap FIR, is shown in Equation 1.9. There are 6 of these FIRs, one for each of the 3 pixel values on either side of the 4x4 edge.

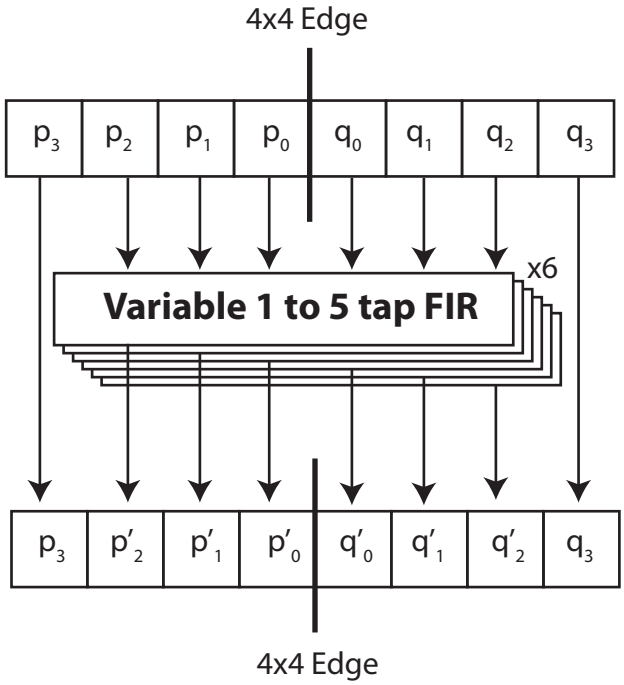


Figure 1-8: Deblocking filter smooths out artificial discontinuities across pixel edges

$$
\begin{aligned}
p'_0 &= (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4)/8 \\
p'_1 &= (p_2 + p_1 + p_0 + q_0 + 2)/4 \\
p'_2 &= (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4)/8 \\
q'_0 &= (q_2 + 2p_1 + 2p_0 + 2p_0 + p_1 + 4)/8 \\
q'_1 &= (q_2 + q_1 + q_0 + p_0 + 2)/4 \\
q'_2 &= (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4)/8
\end{aligned}
\tag{1.9}
$$

1.3.7 Frame Buffer (FB)

Decoded frames must be temporarily kept in a large cache called the frame buffer (FB), so that they can be used for temporal prediction during the decoding of future frames. The H.264 baseline profile level 3.2 (720p) requires the decoder to store the last 5 frames so that they can be used for predicting future frames. These five 720p frames use up about 6.9MB

of memory (720 (*height in pixels*) $\times$ 1280 (*width in pixels*) $\times$ 1.5 (*luma+chroma*) $\times$ 1 *Pixel Byte* $\times$ 5 *frames*). If they cannot be fit into on-chip caches, they must be kept in large off-chip memories with more storage capacity.

The FB is written to by the video decoder whenever an output pixel is produced. The decoder reads from the FB whenever it needs data for the MC unit. At the system-level, the decoded frames must also be sent to a display. In the absence of a separate display buffer memory, the frames to be displayed are also read from the FB.

1.4 Related Work

State-of-the-art H.264 ASIC video decoders (DECs) have used micro-architectural techniques such as pipelining and parallelism to increase throughput and thus reduce power consumption of the digital logic. Additionally, related papers have examined different ways to optimize the memory subsystem of video decoders and therefore increase performance and reduce off-chip memory accesses. Table 1.3 lists some of the recently published H.264 hardware decoders. There is a wide spread in their power efficiency, but most of them consume less than 1W when decoding 1080p videos at 30 fps.

Table 1.3: Survey of H.264 hardware video decoders

Paper Ref.	**Resolution W x H**	**Frame Rate**	**CMOS Process**	**Clock Frequency**	**Supply Voltage**	**Core Power**
[8]	1920 x 1080	30	130nm	130-170 MHz	N/A	554 mW
[9]	1920 x 1088	30	130nm	120 MHz	1.2 V	108 mW
[10]	720 x 480	30	180nm	16.6 MHz	1.2 V	12.4 mW
[11]	176 x 144	30	180nm	1.2 MHz	1.8 V	865 μW
[12]	1280 x 720	30	65nm	14-54 MHz	0.7-0.85 V	1.8 mW
[13]	352 x 288	30	180nm	6 MHz	1.65 V	1.8 mW
[14]	1920 x 1080	30	180nm	44.6 MHz	1.8 V	305 mW
[15]	1920 x 1080	15	65nm	162 MHz	1.2 V	172 mW

1.4.1 Related Work on Video Pipelining and Unit Parallelism

The authors of [14] and [16] pipeline the different decoder units using variable-depth FIFOs. The work in [17] provides an in-depth analysis of how FIFO sizing affects performance of the interconnect in a Network-on-Chip. They treat the whole decoder as a latency-insensitive pipeline, so that the units do not have to operate in lockstep. They also separate the chroma and luma pipelines, and argue that very little area overhead is incurred since the functional units are different for luma and chroma. The pipeline operates on 4-pixel wide data, which doubles the performance over a single-pixel pipeline while adding relatively little area. Within the DB unit, there are two 4x4 block-edge filters, one vertical and one horizontal. These two filters are pipelined and can thus run concurrently. A hybrid pipeline architecture is employed in [13] where most decoder units operate on 4x4 blocks, whereas the DB unit operates on a full MB.

Previous work proposes an architecture for 4x4 INTRA which is optimized for reducing area [9]. As a result, because parallelism is sacrificed for area savings, it can take up to 8 clock cycles to predict a 4x4 block. The work in [18] implements a parallel IDCT architecture using two 1-D IDCTs that can compute a 4x4 block in 4 cycles. They authors of [19] implement a pipelined and parallelized MC interpolator which can compute a 4x4 block in 4-9 clock cycles.

The authors of [14] and [16] show that up to four 4x4 block edges can be filtered by the DB unit in one cycle. This would require using 16 different 8-input filters, one for each of the pixel edges. They also proposes a scheme where multiple MC interpolators could be used in parallel.

The work of [20] used a hierarchical LUT for CAVLD in order to enable parallel lookup of multiple bits at once while reducing the total LUT size. Similarly, [21] explored several types of hierarchical LUT partitioning optimize the energy and performance of VLC. The work in [22] speeds up CAVLD by decoding two coefficient levels and more than two "run-of-zeros" in the same cycle. The work in [23] can improve the CAVLD throughput by 10% when identifying highly-probable patterns of coded 4x4 or 2x2 blocks and thereby avoiding the full CAVLD decoding process.

The author of [24] proposes using distributed arithmetic for implementing the IDCT function of the MPEG-2 video standard. Due to the successive approximation nature of distributed arithmetic, the author proposes using variable-precision arithmetic to perform the IDCT computations. This reduces the processing power of unnecessary precision bits while having only a minor impact on image quality.

This thesis presents several new ideas and analysis related to video pipelining and unit parallelism. Unlike previous work such as [14], we quantify the performance gain achieved by increasing the size of the FIFOs separating the different pipeline stages. We also explore luma interpolator parallelism within the MC unit, which we have not found in other papers. Finally, the variable-precision arithmetic study done for the computations in the IT unit of a modified H.264 decoder was not seen in any of the previous works.

1.4.2 Related Work on Multi-Core Video Decoding

The performance bottleneck of the DEC architecture described in [12] was identified to be the ED unit. This is because CAVLD processes an inherently serial bitstream and cannot be easily parallelized. This is also seen in [25] and [26], where everything but the ED unit was replicated by a factor of 8. Although [25] and [26] are software implementations for multi-core processors, the same ED bottleneck is seen in multi-core hardware DECs, as will be shown in Section 4.3. One way to overcome the ED performance bottleneck is to run it at a faster frequency, as suggested in [27, 15]. However, the ED unit must be run at a higher voltage than the rest of the system, so it will lower the overall energy efficiency. Also, even at the maximum frequency allowed by the underlying transistor technology, the ED unit might not be able to run fast enough to meet the highest performance demands.

A multi-core approach to increasing decoder throughput is to break the input stream into slices that can be processed in parallel, which has been proposed by [28] and [29]. The authors of [28] propose breaking up each frame into completely independent slices; this method was described for MPEG-2 but is also applicable within the H.264 standard at the cost of lower coding efficiency, as will be shown in Section 4.1. The work in [29] proposes breaking up each frame into "entropy" slices where only the ED portion is independent; this

method is not H.264 compliant.

Other approaches used to speed up video decoding in a multi-core system are the software implementations of [26] and [25]. In these works, the MBs are decoded in parallel along a diagonal, but the video bitstream is still parsed serially.

This thesis presents several new ideas and analysis related to multi-core parallelism. We extend the idea of [28] from the MPEG-2 standard to the H.264 standard and analyze the performance benefits, as well as the area costs. We also introduce frame parallelism, a technique which allows multiple decoders to process several consecutive P-frames at once. In addition, we implement the diagonal processing of [26] and [25] in a multi-core hardware decoders and also allow for the ED portion to be processed in parallel. Finally, we describe several ways of indexing multiple decoders into the same video stream at once.

1.4.3 Related Work on Video Memory Optimization

The work in [30] describes a tool called "ATOMIUM" that allows a designer to automatically find the optimal memory architecture for a given algorithm transformation. This tool is especially useful for multi-dimensional signal processing applications dealing with large sets of data, such as video and image processing. The work in [31] analyzes the memory architecture options for a motion estimation (ME) engine.

Previous work uses individual last-line caches for each of the decoder units to avoid accessing a large main memory [14, 16]. [10] uses a line-pixel look-ahead scheme to reduce the size and activity of this last-line cache.

The authors of [14, 16] place MC caches between the frame buffer controller (memory controller (MEM)) and the frame buffer store (FB). Separate MC caches are used for luma and chroma, and the authors conclude that two caches of size 1 kByte provide 46% and 30% reduction in OCFB MC reads for luma and chroma respectively.

The work in [32] and [13] uses local buffers to store the MC overlap data between neighboring 4x4 blocks. Additionally, [32] combines luma and chroma accesses to increase the burst length of the external dynamic random-access memory (DRAM) from 2.18 to 4.38. Based on these two techniques, [32] achieves 56% BW savings.

In addition to on-chip caching, another technique to reduce OCFB BW is to compress the reference frames ([33]). When storing a decoded frame, the DEC uses a simple fixed-length lossy compression to reduce the size of the frame sent to the OCFB. When reading back a reference frame for MC, the DEC must perform the inverse of that compression to recover a degraded copy of the reference pixels. This scheme can achieve a 25% reduction in both the size and BW of the OCFB when compressing pixels from 8 bits to 6 bits. The main cost is a degradation in either bitrate or PSNR and the extra computation required for compression and decompression. Specifically, this scheme leads to a 1.03% drop in bitrate or 0.043 drop in PSNR. The idea of [33] is not H.264 compliant and must be performed in the same way at the encoder and decoder.

The work in [26] demonstrates how memory BW can be reduced for a multi-core H.264 DEC software implementation. Two partitioning methods are considered for the multi-core processor architecture: by data (part of a frame) and by function (part of the algorithm). The data memory BW is found to be 65% smaller when each processor fully decodes part of a frame (data partitioning) versus when each processor performs part of the decoding for the entire frame (function partitioning).

The work in [9] uses two separate FBs, one for luma and one for chroma, in order to parallelize the MEM and allow it to operate at half the frequency. This is especially important for 1080p resolutions, when the MEM clock rate could be as high as 200-300MHz. [34] implements a video processor with embedded dynamic random-access memory (eDRAM), allowing a large reduction in processor I/O power.

This thesis presents several new ideas and analysis related to video memory optimization. We implement a last-frame cache (LFC) to help eliminate most off-chip memory reads in the video decoder. We also describe how data-forwarding caches (DFCs) can be used to increase the temporal locality of data written and read during two consecutive frames. Finally, we show how interleaved entropy slice (IES) caching improves the temporal locality of data written and read during consecutive MB lines.

Chapter 2

Pipelining and Unit-Level Parallelism

The decoder units of Figure 1-2 can be pipelined in order to increase concurrency and throughput. Pipeline registers can be inserted between the different units, as shown in Section 2.1 and Section 2.2, or within some of the units, as shown in Section 3.1 and Section 2.4.

Parallelism can also increase a video decoder's performance and allow it to operate at a lower voltage for a given performance requirement. Alternatively, the voltage can be fixed, and so parallelism can allow the decoder to achieve a higher throughput and decode higher resolutions. This chapter describes how parallelism can be used within the decoder units of Figure 1-2 in order to reduce the number of cycles used to process a 4x4 block of pixels. Chapter 4 will deal with multi-core parallelism.

As discussed in Section 1.4.1, pipelining and unit-level parallelism have been extensively explored for video decoders. In this chapter, we will describe which of the existing pipelining techniques we have used or improved, as well as introducing some new ideas.

The ideas presented in Section 2.1, Section 2.5, and Section 2.9 were developed together with Vivienne Sze.

2.1 Decoder Pipeline

The top-level pipelined architecture of the decoder hardware is shown in Figure 2-1. At the system level of the decoder, first-in-first-out registers (FIFOs) of varying depths connect the major processing units: entropy decoder (ED), inverse transform (IT), motion compensation (MC), spatial prediction (INTRA), de-blocking 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. Some pipeline dependencies can arise, which will require stalling in order to ensure correctness. For example, a block of pixels in the INTRA unit might have to wait for the previous block of pixels to be processed by the addition of residual to prediction (ADD) stage before it can proceed.

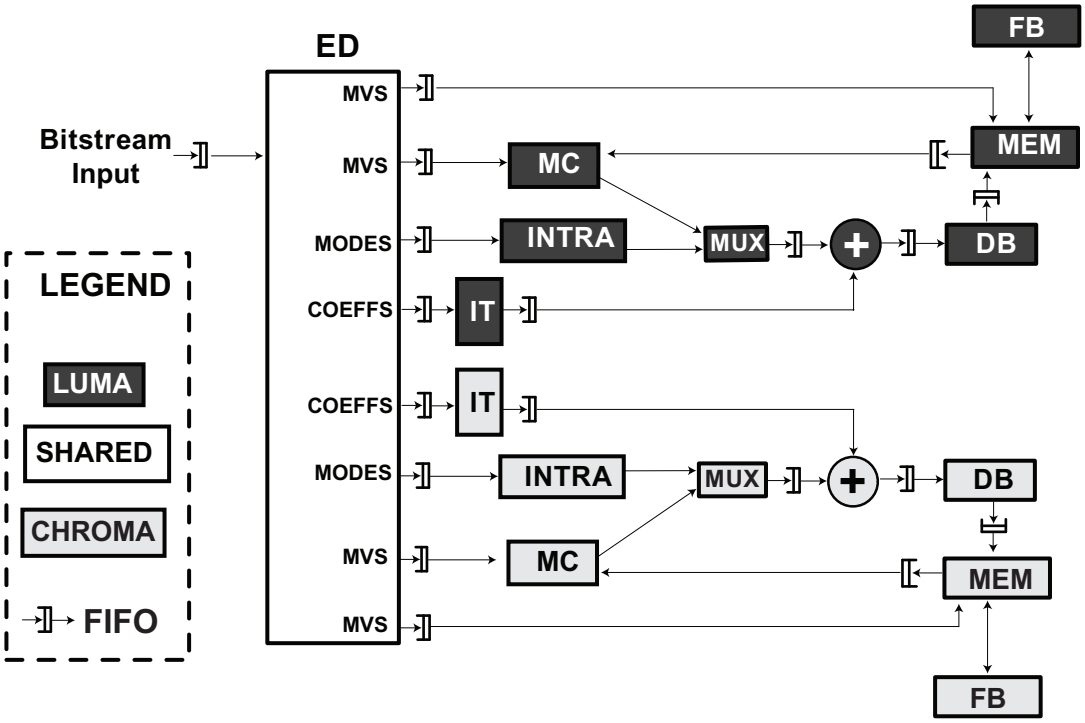


Figure 2-1: H.264 pipelined decoder architecture

An example of the luma pipeline operation for P-type macroblocks (MBs) is shown in Figure 2-2. P-type frames use temporal prediction from previously-decoded frames, while I-type frames use spatial prediction from previously decoded parts of the same frame. At any given time, 7 different luma 4x4 blocks can be in flight. Some stages can be idle due to

variable workloads or unbalanced cycle counts.

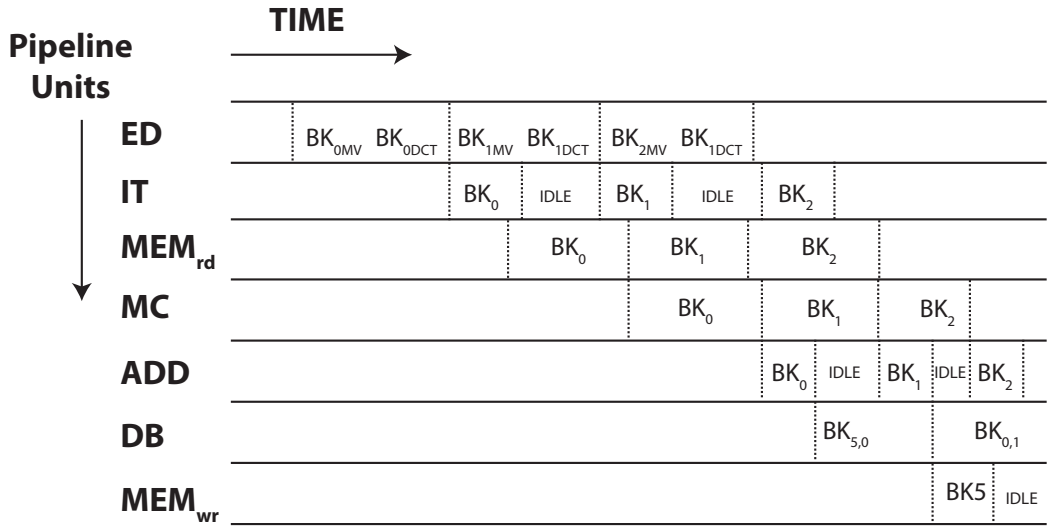


Figure 2-2: Pipeline timing example

Additional concurrency is achieved by processing the luma and chroma components with 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 simultaneously, 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 coefficients and motion vectors for both pipelines. To reduce hardware costs, the luma and chroma pipelines also share the IT unit, since this unit has a relatively low cycle count per block relative to the rest of the units.

2.2 FIFO Sizing

One of the challenges in the system design of the video decoder is that the number of cycles required to process each block of pixels changes from block to block (i.e. each unit has varying workload). Consequently, each decoder unit has a range of cycle counts. For instance, the number of cycles for the ED depends on the number of syntax elements (e.g. non-zero

coefficients in residual, motion vectors, etc.) and is typically proportional to the bitrate. As another example, the number of cycles for the MC unit depends on the corresponding motion vectors. An integer-only motion vector requires fewer cycles (4 cycles per 4x4 luma block) as compared to one which contains fractional components (9 cycles per 4x4 luma block).

To adapt for the workload variation of each unit, variable-depth FIFOs were inserted between each unit. These FIFOs also distribute the pipeline control and allow the units to operate out of lockstep. 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 [14]. For example, consider a simple example where the ED performance alternates between 1 cycle and 5 cycles per 4x4 block, with an average performance of 3 cycles/4x4. Also, suppose the IT unit following the ED always takes 3 cycles per 4x4 block. In an ideal pipeline, these two stages are balanced and the pipeline should have a throughput of one 4x4 block every 3 clock cycles. However, if the FIFO depth separating the two units is only one deep, the IT unit will stall for 2 cycles whenever the ED unit takes 5 cycles to produce a 4x4 block. In this case, the average throughput degrades to one 4x4 block every 4 clock cycles.

Figure 2-3 shows that the pipeline performance can be improved by up to 45% by increasing the depths of the 4x4 block FIFOs in Figure 2-1. Compared to a DEC running at full voltage, a 45% improvement in performance enables voltage scaling that corresponds to about a 37% savings in dynamic power. 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.

The FIFO depths considered in Figure 2-3 were fixed for all the FIFOs to be the same, and the performance was analyzed by varying the depths together. In reality, not all FIFOs have an equal impact on global performance, so their depths can be optimized independently. FIFOs with access patterns that have more variance and are more bursty should be deeper than FIFOs connecting units that have relatively constant instantaneous throughput. For a more in-depth analysis of FIFO sizing in the context of a Network-on-Chip interconnect, the reader is referred to [17].

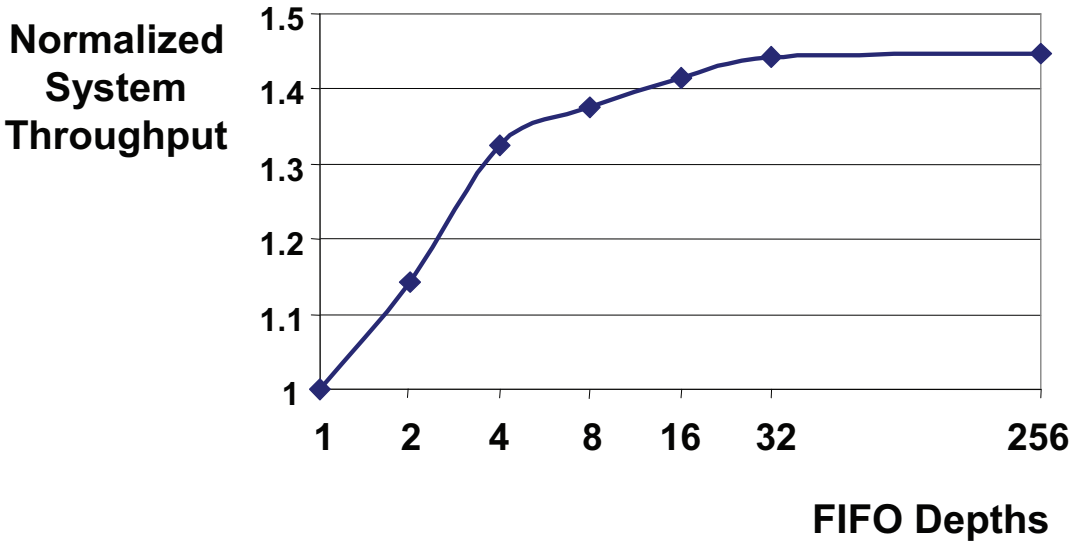


Figure 2-3: Longer FIFOs average out workload variations to minimize pipeline stalls. For this analysis, the depths of all DEC FIFOs are set to the same value, so the depths are varied together. One FIFO element corresponds to data representing a 4x4 block of pixels.

For maximum concurrency, the average cycles consumed by each stage of the pipeline, which is equivalent to each processing unit in this implementation, should be balanced. The remainder of this chapter describes how parallelism can be used to reduce cycle counts in the bottleneck units to help balance out the cycles in each stage of the pipeline.

2.3 Motion Compensation (MC) Architecture

This thesis provides a detailed description of the pipelining and parallelism optimizations implemented for the MC interpolator architecture. For this reason, this section will be dealt with separately in Chapter 3.

2.4 Inverse Transform (IT) Architecture

The inverse transform (IT) unit performs an inverse discrete cosine transform (IDCT) on the coefficients obtained from the entropy decoder (ED). A parallel implementation, shown in Figure 2-4, has 8 1-D butterflies running simultaneously in order to reduce the IT latency

and increase its throughput. Reducing the latency is important to minimize the pipeline stall cycles when there are many coded residual blocks (i.e. blocks with non-zero coefficients). Using this architecture, a full 4x4 residual block can be produced every cycle. The critical path includes pre-scaling, a 1D IDCT, transposing, another 1D IDCT, then post-scaling.

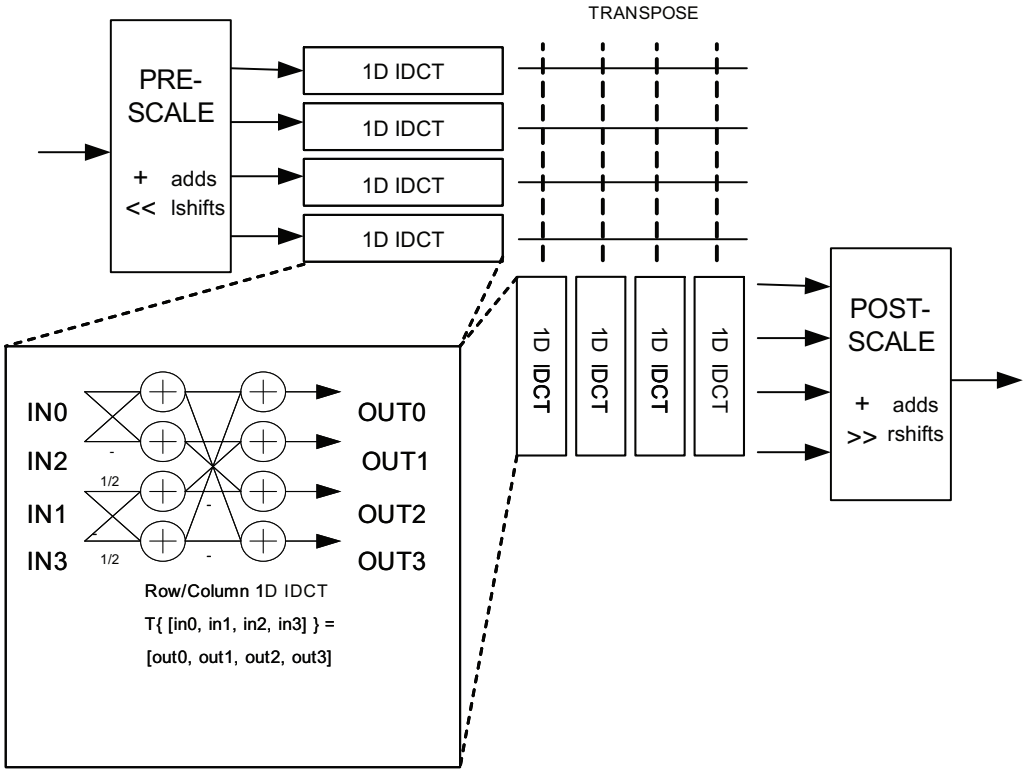
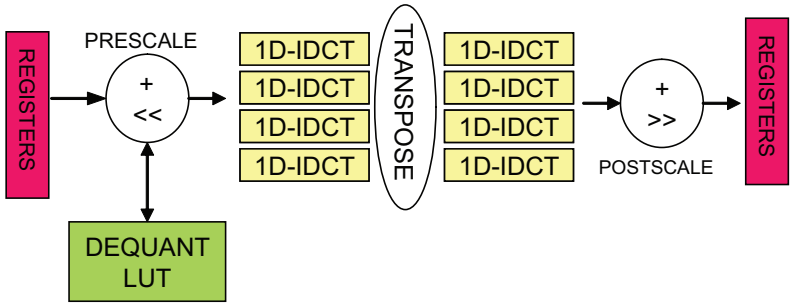


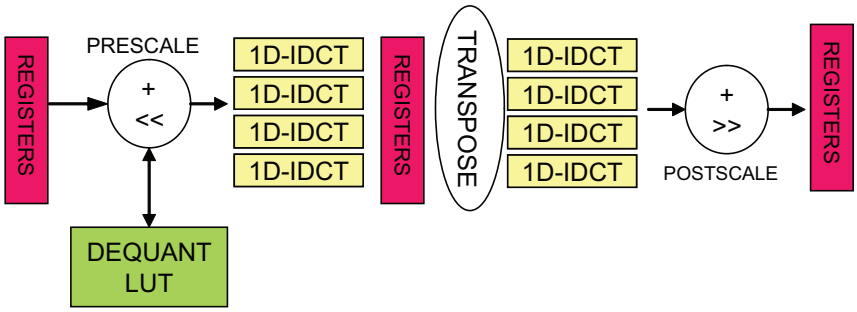
Figure 2-4: Parallel inverse transform architecture

To speed up the IT throughput, we can pipeline the parallel architecture of Figure 2-5a, by adding some registers before or after the transpose stage, as shown in Figure 2-5b. Alternatively, if we want to reduce the area and level of parallelism of the architecture in Figure 2-5a and Figure 2-5b, we can implement the IT as a folded pipeline, as shown in Figure 2-5c. This reduces the cycle time, but it now takes two cycles to transform a 4x4 block. Note that the cycle time is not reduced exactly by two, since there is the extra overhead of the mux and setup time of the registers.

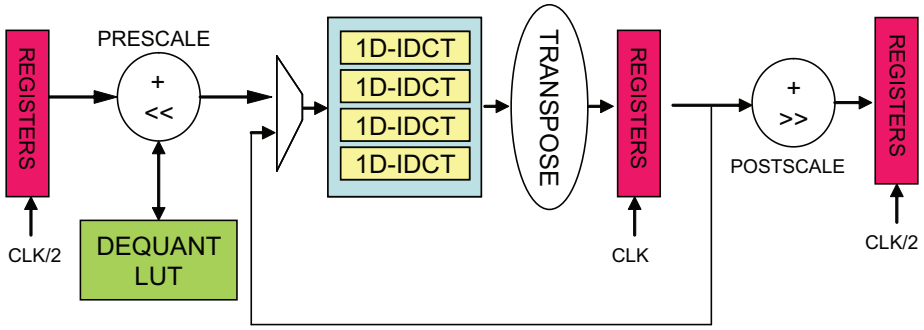
If there are no non-zero DCT coefficients, the IT is skipped since the residual is zero and does not need to be computed. The workload of the IT can therefore vary considerably,



(a) Unpipelined architecture, IMPL0



(b) Pipelined architecture, IMPL1



(c) Folded pipeline architecture, IMPL2

Figure 2-5: IDCT architectures

depending on the type of video, as shown in Figure 2-6. If the voltage and frequency of the IT unit can be dynamically adjusted based on the workload, the IT can operate at a lower voltage and frequency, and thus consume less energy. This is illustrated in Figure 2-7, which shows how delay and energy scale for each of the IT implementations of Figure 2-5.

For future video standards, we can trade off power versus coding efficiency by scaling the bit-accuracy of the computations in the IT block. One way to perform variable-resolution arithmetic is to "zero" out N least significant bits (LSBs), as shown in Figure 2-8. The motivation behind this is that for highly compressed videos, there is a great amount of

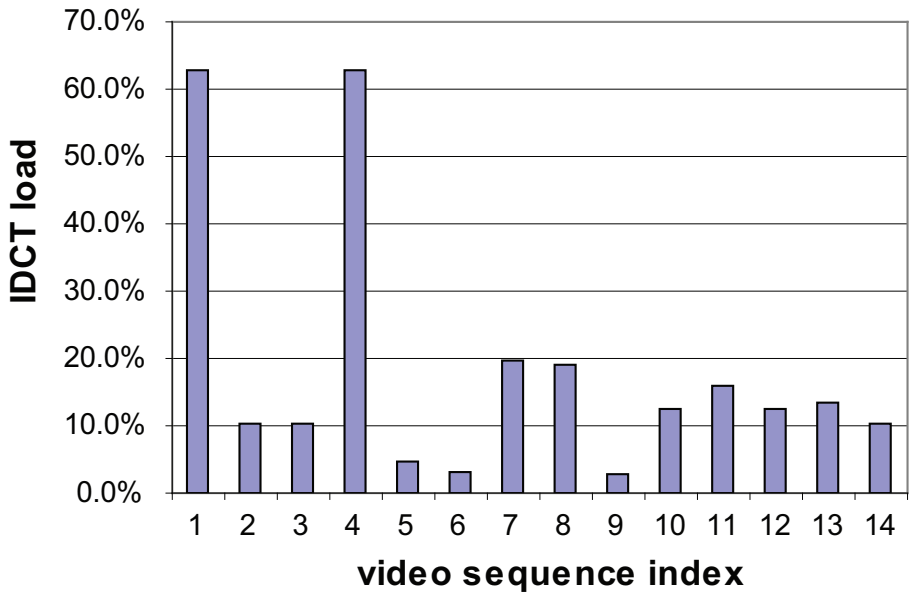
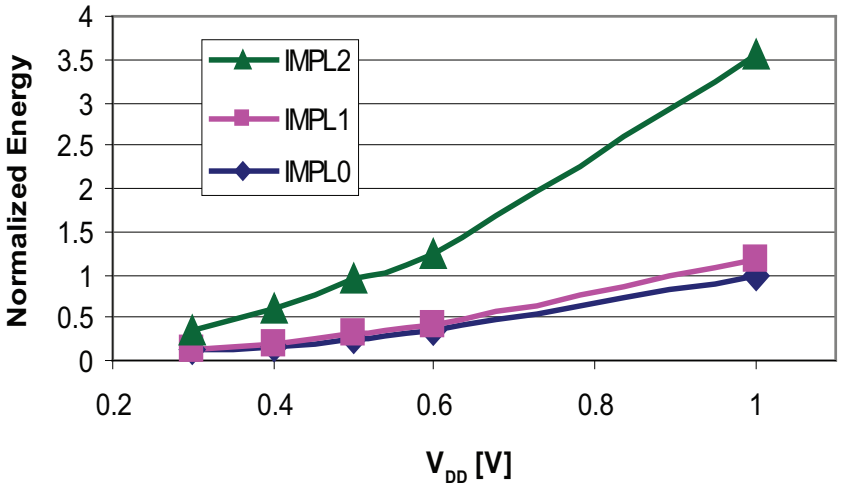


Figure 2-6: Interpolator pipeline

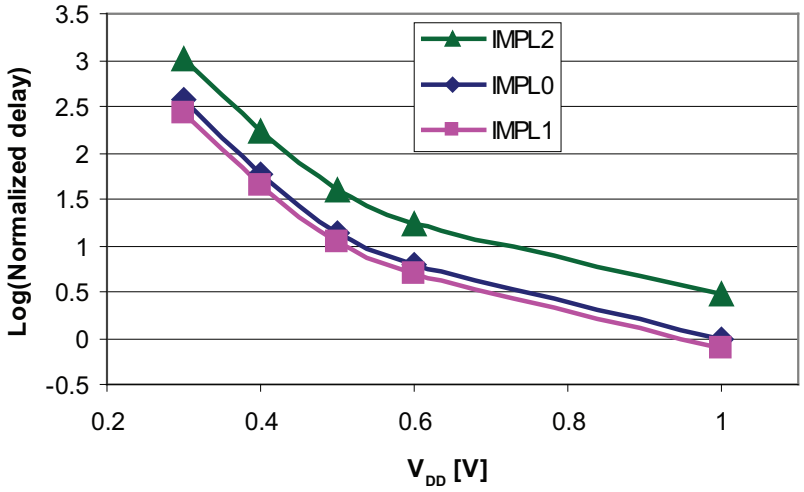
quantization noise present in the pixel values. As a result, the ratio of signal to quantization noise is low enough that additional truncation of the data will not have a great impact on the PSNR. This technique is not compliant with the H.264 standard.

Figure 2-9 shows how using less bits in the IT computation leads to lower dynamic power, but increases the image distortion. Figure 2-9a shows how the dynamic power savings increase as more bits are truncated. For larger quantization parameters (QPs), or higher quantization, more bits need to be truncated in order to notice an increase in power savings. This is because fewer LSBs are toggling for larger QPs. Figure 2-9b shows that with no truncation, the PSNR is inversely proportional to QP. As we begin to truncate bits, the video coded with the lower QP is the first to be affected. This is because the LSB contain more information for videos with lower QPs.

Figure 2-10 shows an example of truncating the 9 LSBs of the IT internal 16-bit data for a video coded with QP=40. From Figure 2-9, this can save 25% of the IT power while having only a minor impact on image quality (1.1dB). The visual impact of the 1.1dB drop in PSNR is almost negligible for this video, mainly because the PSNR was low to begin with before truncation.



(a) Normalized IDCT energy



(b) Normalized IDCT delay

Figure 2-7: Energy and delay comparisons between three different IDCT architectures

To support the claim that the truncation technique is more applicable to videos of low PSNR, consider trying to achieve 25% power savings when QP=10, with a PSNR of 55dB for this video. From Figure 2-9a, we would need to truncate about 6 bits. However, the impact of a 6-bit truncation is a PSNR drop of about 9dB, from Figure 2-9b. This might not be a satisfactory trade-off for a ENC to make.

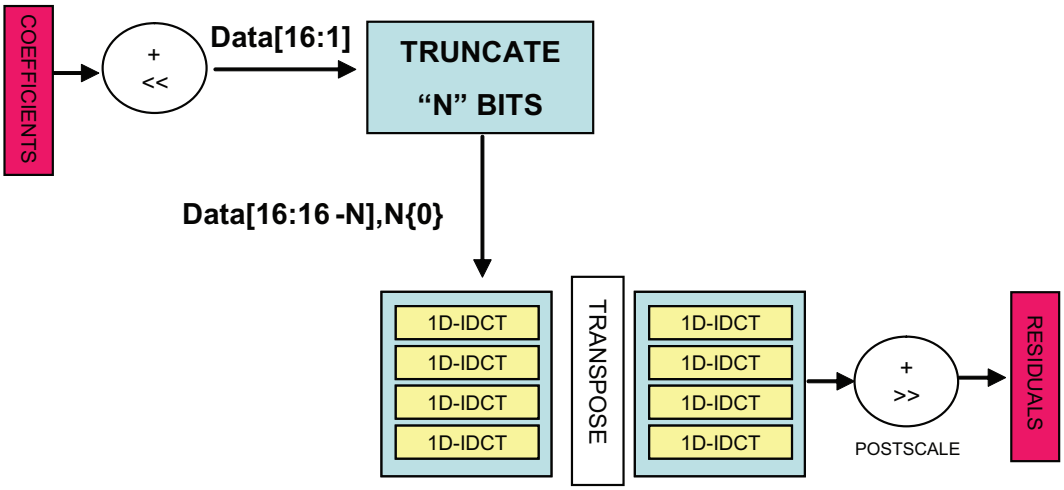
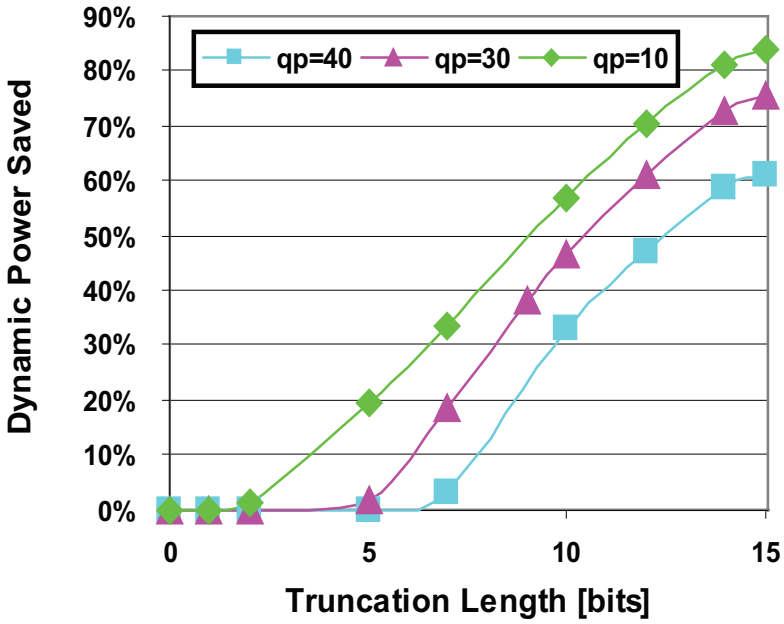
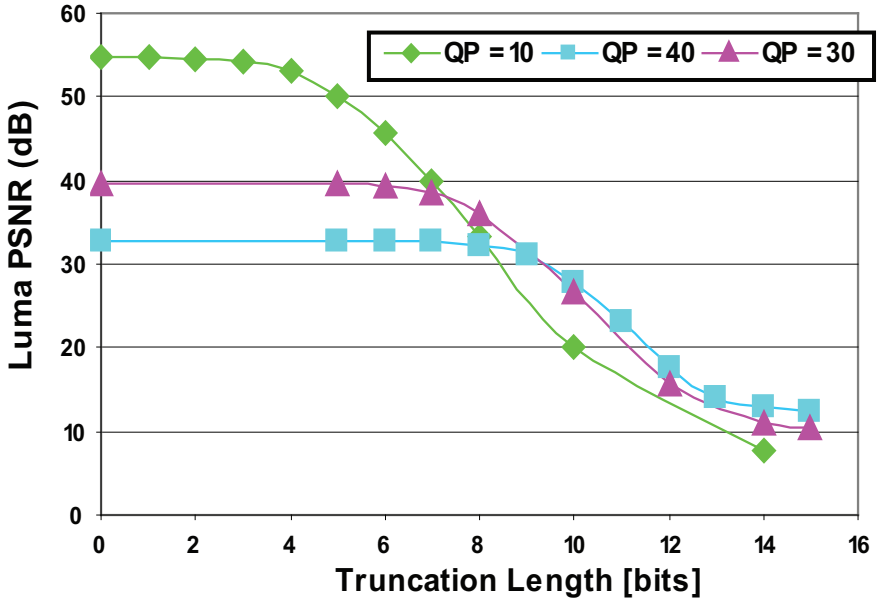


Figure 2-8: Scaling the bit-accuracy of the IT operation



(a) Effect of truncating IT bits on dynamic power



(b) Effect of truncating IT bits on luma PSNR

Figure 2-9: Power savings increase but peak signal-to-noise ratio (PSNR) decreases as the number of truncated bits in the IT unit increases, computed for the movie clip "You, Me, and Dupree"

(a) QP=40, 9-bit truncation, PSNR=29.4dB

(b) QP=40, No truncation, PSNR=30.5dB

Figure 2-10: Truncating the 9 LSBs of the IT data reduces the power by 25% and the PSNR by 1.1dB for the movie clip "You, Me, and Dupree", coded with quantization parameter (QP)=40

2.5 Deblocking Filter (DB) Architecture

The DB architecture was designed by Vivienne Sze, with contributions from the author.

The length and weightings in the de-blocking filter (DB) FIR are dependent on several parameters, including the coding type of the 4x4 blocks being filtered, as well as the pixel value values on either side of the 4x4 edge. These different parameters are combined into one DB parameter called the boundary strength. The boundary strength information of the adaptive FIR is the same for all edges on a given side of a 4x4 block. Accordingly, the DB can be designed to have 4 luma and 2 chroma FIRs running in parallel, and filter an edge of a 4x4 block every cycle. The luma architecture is shown in Figure 2-11. For additional cycle reduction, the luma and chroma FIRs operate at the same time, assuming the input data and configuration parameters are available.

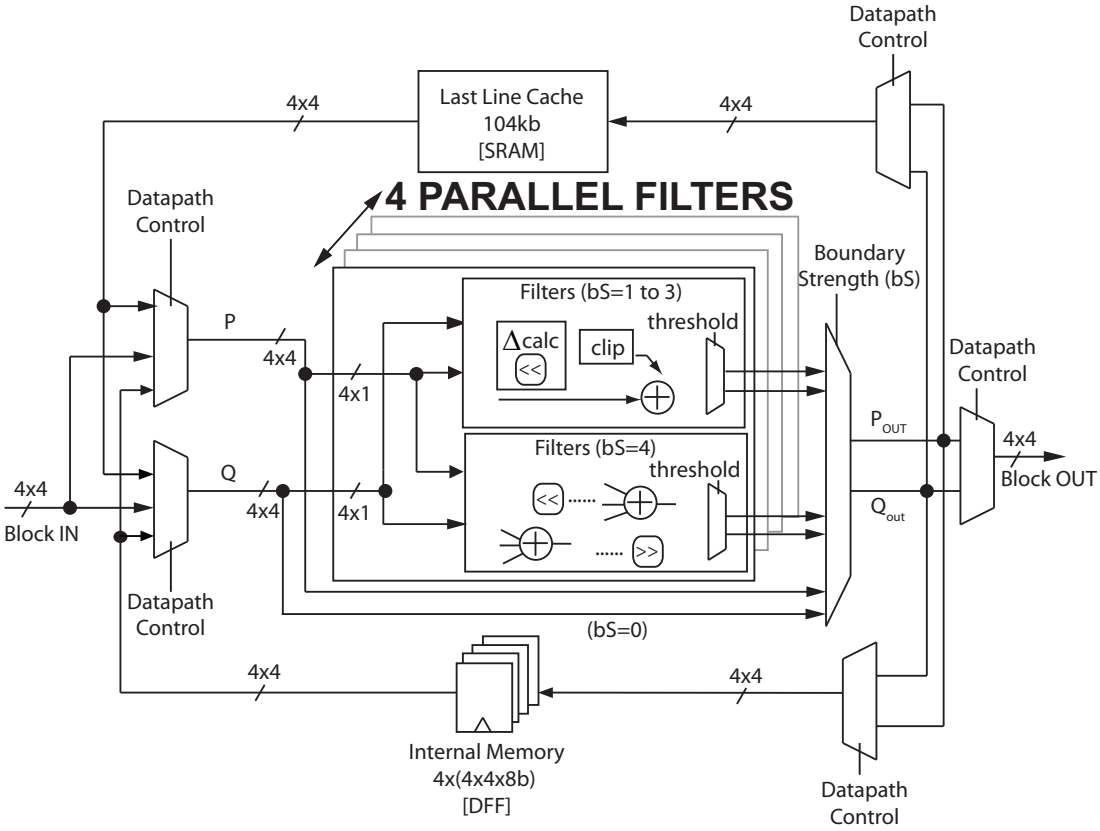


Figure 2-11: Deblocking filter architecture for luma filtering

2.6 Intra Prediction (INTRA) Architecture

The luma 4x4 prediction can be done in one cycle if all the filters are available in parallel. For example, to implement the Diagonal-Down-Right (DDR) prediction of Equation 1.6, 7 different 3-tap FIRs are necessary. From Equation 1.6, there are 7 total diagonals, and all the pixels along the diagonal have the same value, as the value only depends on $(x - y)$. Therefore, we do not need to instantiate 16 different FIRs, since the synthesis tool will recognize the common terms. Similarly, the other prediction modes can be implemented with additional parallel FIRs, and the synthesis tools can be used to extract any common terms amongst them.

Chroma prediction for 8x8 blocks and luma prediction for 16x16 blocks uses the same modes. As a result, these two units have some common hardware for doing averaging and plane prediction. If area optimization is more important for these units than performance, the common area can be multiplexed between the two predictors, and only one of the luma and chroma DEC pipelines can do spatial prediction at a time. To avoid this pipeline dependency, the common hardware can just be duplicated.

2.7 Entropy Decoding (ED) Architecture

Decoding a variable-length bitstream in parallel can be quite challenging, since the start of the next codeword is not known until the current element is fully decoded. As a result, the ED unit can be one of the bottlenecks of the decoder pipeline, especially for bitstreams with low compression ratios.

Although two codewords cannot be processed simultaneously, it is possible to speed up the decoding of an individual element. Instead of searching for a variable-length codeword using one bit per cycle, the entire code can be retrieved in parallel from a lookup table in one cycle. For example, the codeword that specifies the total number of coded coefficients and the total number of trailing ones has a maximum length of 16 bits. This would require a lookup table, or read-only memory (ROM) of size $2^{16} = 65536$ entries.

To reduce the memory requirements of the lookup table, a two-step lookup could be

used, as was demonstrated in [35] and shown in Figure 2-12. The 16-bit codeword could be split up into two 8-bit parts, which would be stored in two different lookup tables, each with up to $2^8 = 256$ entries. If the variable-length codewords are less than 8 bits, only the first lookup is needed and it takes one cycle. If it is longer than 8 bits, a second lookup is needed, so an extra clock cycle is needed. Since the most common codewords use fewer bits for good compression, the expected number of clock cycles needed would still be close to one. A similar approach was used in [20], where it was termed "Hierarchical logic for LUTs". The work in [21] also explored several types of hierarchical LUT partitioning, in order to optimize the energy and performance of VLC.

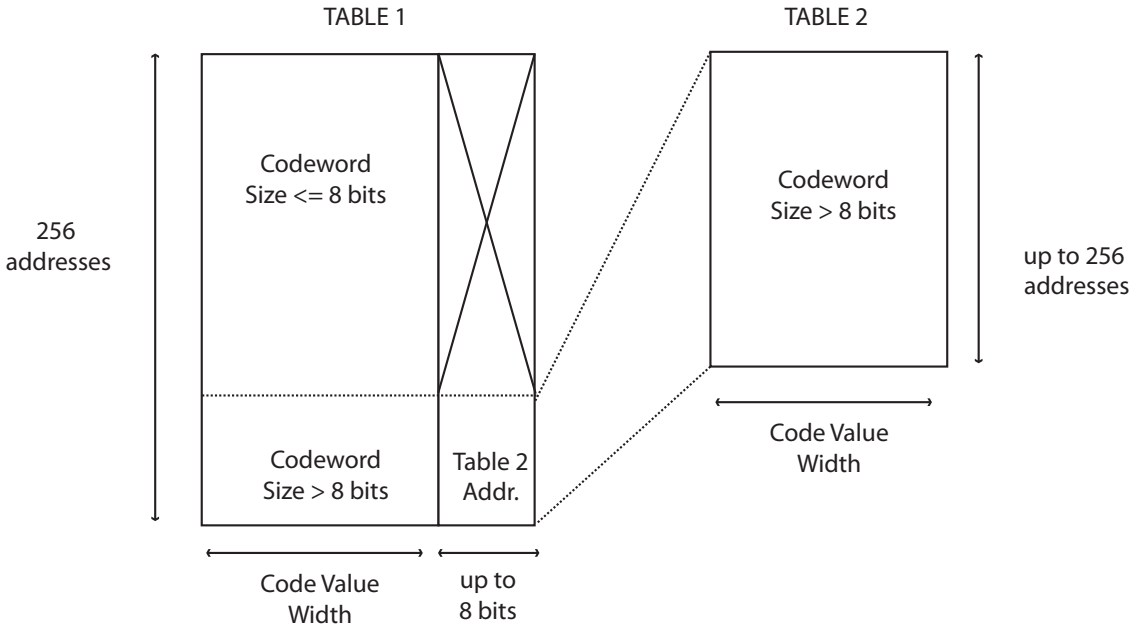


Figure 2-12: Hierarchical LUTs for ED

2.8 Reconstruction (ADD) Architecture

The reconstruction unit can easily be parallelized to perform as many additions in one cycle. For example, in order to reconstruct a 4x4 block in one cycle, 16 different 8-bit adders can be used. If all the other DEC pipeline stages take 4 cycles on average per 4x4, then having 4 different adders with muxed inputs and outputs is sufficient.

2.9 Memory Controller (MEM) Architecture

The memory controller unit can be divided into two components: I/O pads and logic. The I/O pads connect the video decoder to the external memory. The logic implements the memory interface protocol, such as address calculation, read or write selection, memory enable, generating output data, capturing input data.

The number of memory interface I/O pads might be restricted to the memory data width. For example, if a memory chip has a bidirectional data bus of 32 bits, this allows only 4 bytes of data to be read or written during every cycle by the decoder. Even if wider memory chips are an option, the number of memory I/O pads might still be limited by the total silicon area which places a maximum on the total number of I/O pads. Therefore, if the I/O pads and logic run off of the same clock, maximum parallelism is made possible by using as many memory interface I/O pads as possible. If further parallelism is desired, the logic used to generate the addresses could be placed on a slower clock domain and replicated. Smaller logic on a faster clock domain could be used to multiplex between the different parallel addresses.

2.10 Summary

This chapter described different techniques that can be used to speed up the video decoder units, such that a slower clock and therefore lower voltage can be used to decode each frame. The different video decoder units were first arranged into a non-interlocked pipeline, such that all the units can operate in parallel and increase performance. The pipeline units were separated by variable-depth FIFOs, whose depths were chosen to minimize stalls due to workload variation within the units. Each of the pipeline units was then optimized to reduce their cycle count, using parallelism whenever possible.

Chapter 3

Motion Compensation (MC) Architecture

Vivienne Sze designed the original pipelined luma interpolator of Section 3.1 and the parallel chroma interpolator of Section 3.3.

3.1 Luma Motion Compensation (MC) Pipeline

When one or both of the motion vectors are fractional, the luma interpolator predicts the current 4x4 block from a 9x9 block of pixels from the previous frame. The interpolator architecture is shown in Figure 3-1 and is similar to the design in [19]. It consists of a shift register of 6 columns, which get shifted to the right during each cycle. Each column has 9 registers, 5 (X_int,Y_int) and the 4 (X_int,Y_frac) that fit right between them. During each cycle, a column of 9 integer pixels (shown on the left) from the previous frame is input to the interpolator. The middle 5 of these inputs are directly fed to the first column's 5 (X_int,Y_int) registers. The 4 (X_int,Y_frac) registers are loaded with the outputs of the 4 vertical 6:1 FIRs shown on the left. After 6 clock cycles, the entire shift register has been populated with either integer or vertically-filtered pixels. A set of nine 6:1 FIRs is now used to obtain all the horizontally-filtered pixels at the half-way x-coordinate of the shift register. At this time, all the half-point pixels are available for that x-coordinate and a column of 4

predicted pixels can be output if the motion vectors had only half-point values. However, since quarter-point accuracy is possible, a set of 4 bilinear filters is used to predict the column of 4 pixels when one or both motion vectors have quarter-point components. In total, the datapath of the interpolator pipeline is made up of 54 different 8-bit registers, thirteen 6:1 FIRs, and four 4:1 Bilinear filters.

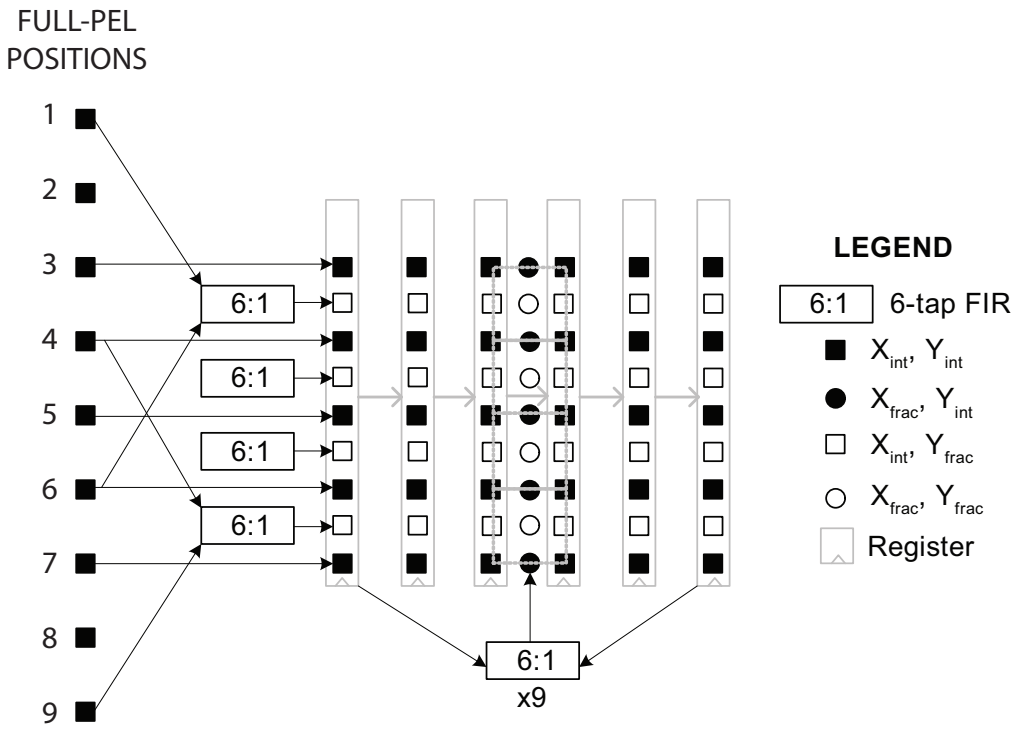


Figure 3-1: Interpolator pipeline

In order to minimze the total energy, the interpolator architecture of Figure 3-1 would be operated at the supply voltage that corresponds to the minimum energy point. This minimum is shown in Figure 3-2, which plots the total energy used for a 4x4 interpolation versus supply voltage. The minimum energy exists at around 50% of the supply voltage or 0.6V. However, at this voltage, the interpolator is quite slow and cannot meet the required performance. To make up for the increase in delay due to voltage scaling, we can use parallelism, as shown in the following section.

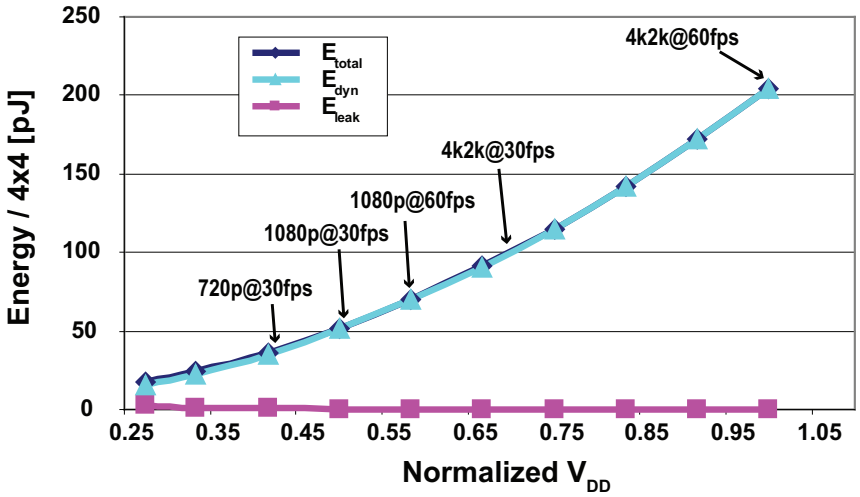


Figure 3-2: Energy of MC interpolation per 4x4 block plotted versus normalized supply voltage. Dynamic energy decreases with the supply voltage, whereas leakage energy is a small portion of the total energy due to the high activity factor.

3.2 Luma Interpolator Parallelism

The MC interpolator is a critical unit in the decoder pipeline. A single interpolator takes 4 to 9 cycles to compute a 4x4 block of pixels. For 65nm, the critical path in the interpolator is about 6ns at the maximum supply voltage of 1.2V. Therefore, assuming no stalls, a single interpolator can produce 733,108 4x4 blocks during every frame time period, which is 33ms for a 30fps frame rate. This is roughly the throughput needed for a 4k2k resolution frame, or 4096x2048 pixels.

Instead of having one interpolator running at the maximum voltage, the supply voltage can be lowered and parallelism of varying degrees can be used to make up for the loss in performance. As will be shown later, this can lower the MC power by as much as 72%. The parallel MC interpolator architecture is shown in Figure 3-3. Each of the interpolator blocks, MC_i, can be implemented with the architecture described in Figure 3-1.

There are two different inputs for each of the N interpolators during each cycle: a column of at most 9 pixels read by the MEM from the FB, and a motion vector coming from the ED. The 128-bit (16 pixels) output of each interpolator is stored in an output FIFO, and later sent down the DEC pipeline to the DB unit.

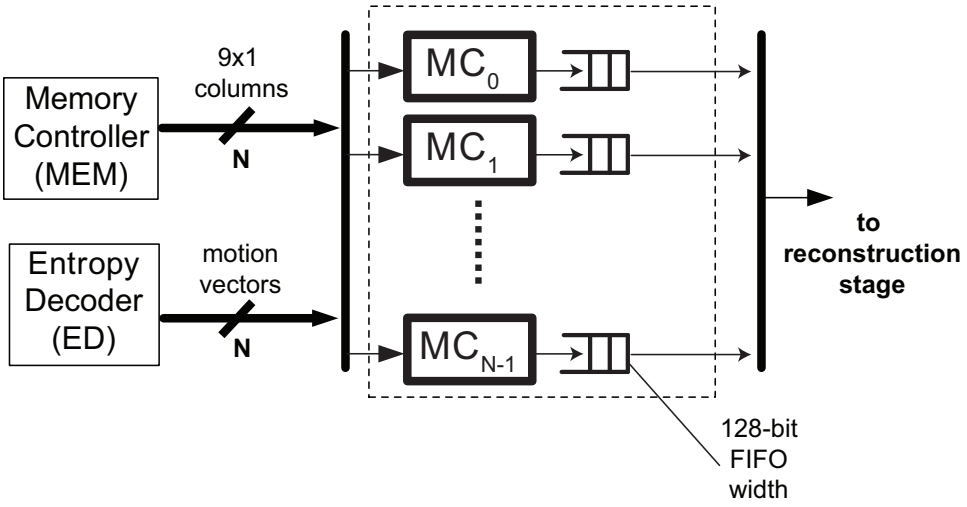


Figure 3-3: Parallel MC interpolator architecture

In the H.264 standard, MBs are transmitted and processed in raster-scan order. However, within a MB, 4x4 blocks are processed in a "nested zig-zag" order as shown in Figure 3-4. This 4x4 order, from 0 to 15, is referred to as the block index. There are several ways to assign these blocks to the MC interpolators.

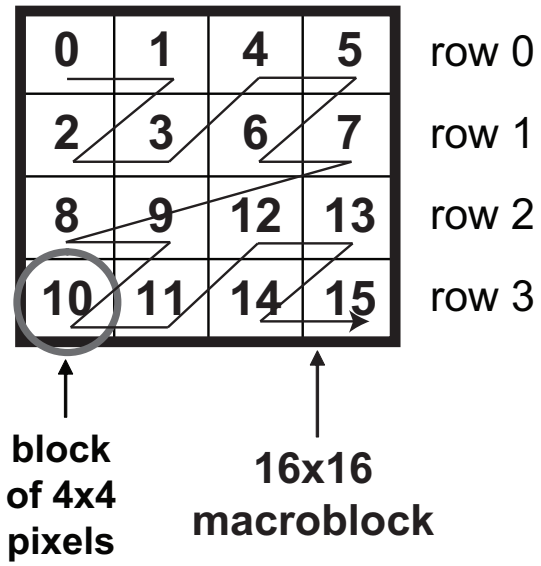


Figure 3-4: Scanning order for H.264 4x4 blocks.

One option is to always assign the next block to the first free interpolator. This way, each

of the interpolators could be assigned to any of the block indices. The problem with this approach is that there is no guarantee that horizontally-neighboring 4x4 blocks would be processed by the same interpolator. Therefore, the cycle savings and data reuse for adjacent blocks with the same horizontal integer MV, described in Section 3.1, would disappear and the performance and power would suffer.

An alternative is to assign fixed block indices to each of the parallel interpolators. This increases performance, allows better data reuse, and simplifies the control logic. The following list describes how the interpolators are assigned to the different block and MB indices, where N is the number of parallel interpolators. .

- *N=1*, interpolator processes in zigzag scan order
  - MC_0: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
- *N=2*, one interpolator for even block rows, the other for odd rows (see Figure 3-5)
  - MC_0: [0, 1, 4, 5, 8, 9, 12, 13]
  - MC_1: [2, 3, 6, 7, 10, 11, 14, 15]
- *N=4*, one interpolator for each block row (see Figure 3-5)
  - MC_0: [0, 1, 4, 5]
  - MC_1: [2, 3, 6, 7]
  - MC_2: [8, 9, 12, 13]
  - MC_3: [10, 11, 14, 15]
- *N=4*n* one interpolator for each block row, but processes every n^{th} MB (see Figure 3-5); MB with (index%n=j) is processed by:
  - MC_{0+4j}: [0, 1, 4, 5]
  - MC_{1+4j}: [2, 3, 6, 7]
  - MC_{2+4j}: [8, 9, 12, 13]

N=2

MC_0	0	1	4	5	0	1	4	5	MC_0
MC_1	2	3	6	7	2	3	6	7	MC_1
MC_0	8	9	12	13	8	9	12	13	MC_0
MC_1	10	11	14	15	10	11	14	15	MC_1

N=4

MC_0	0	1	4	5	0	1	4	5	MC_0
MC_1	2	3	6	7	2	3	6	7	MC_1
MC_2	8	9	12	13	8	9	12	13	MC_2
MC_3	10	11	14	15	10	11	14	15	MC_3

N=8

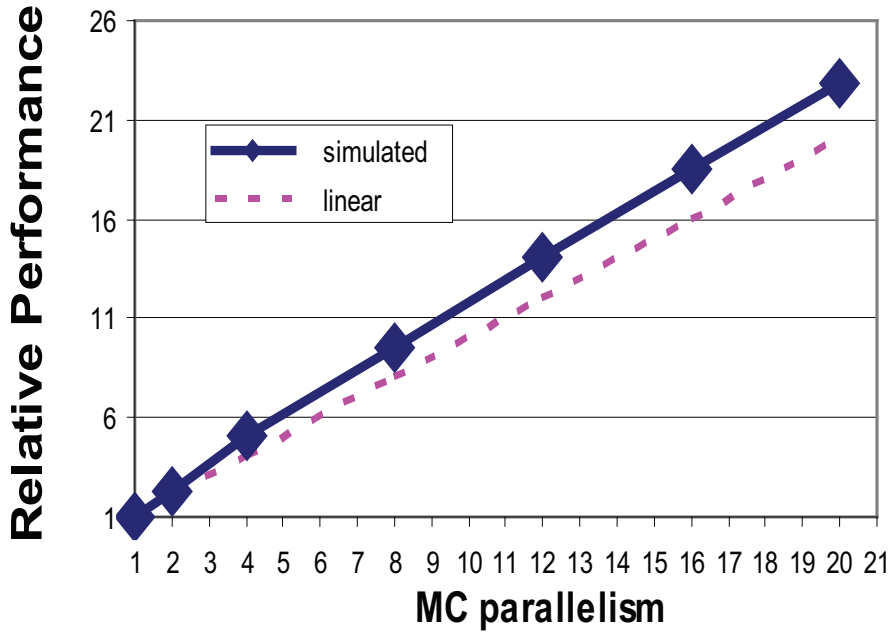
MC_4	0	1	4	5	0	1	4	5	MC_0
MC_5	2	3	6	7	2	3	6	7	MC_1
MC_6	8	9	12	13	8	9	12	13	MC_2
MC_7	10	11	14	15	10	11	14	15	MC_3

Figure 3-5: Parallel MC interpolator assignment to blocks and MBs for $N = 2$, $N = 4$ and $N = 8$.

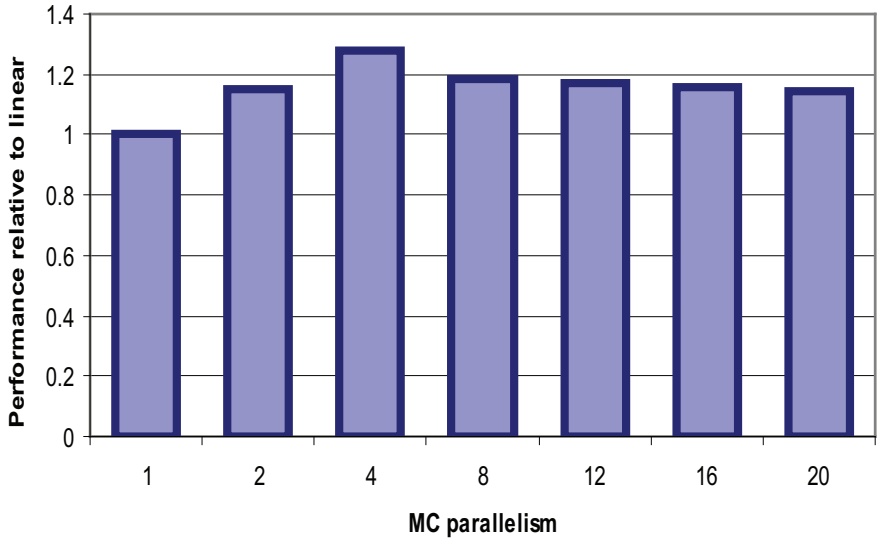
– MC_{3+4j}: [10, 11, 14, 15]

The performance increase achieved by interpolator parallelism is close to linear, as shown in Figure 3-6a. It can be seen that the simulated performance relative to one interpolator is slightly super-linear. This is because the single interpolator only processes at most two horizontally-adjacent 4x4 blocks, whereas all the other parallel interpolators process at least 4 adjacent blocks. If $N = 4$, the performance versus linear gain is the highest, as shown in Figure 3-6b. This is because the interpolators can process blocks along the entire frame width in the same row, thus avoiding redundant operations across the MB borders.

Another design variable is the depth of the FIFOs at the output of each interpolator. For



(a) Parallel MC performance normalized to $N = 1$



(b) Parallel performance normalized to linear gain. Super-linear performance growth is possible because the single-interpolator architecture follows the zig-zag processing order and fails to take advantage of overlapped computations across 4x4 block edges.

Figure 3-6: Simulated performance of parallel MC interpolators

example, consider the case when $N = 4$, the output FIFO of each interpolator has a depth of 1, and the outputs are processed in the typical zig-zag order. When MC_0 and MC_1 finish

processing block indices 5 and 7, interpolators MC_2 and MC_3 are also finishing blocks 9 and 11. However, because the output FIFOs of MC_2 and MC_3 are full with blocks 8 and 10, these two interpolators will stall and therefore lower performance. If we increase the FIFO depth to 2, the bottom two interpolators will mostly be able to operate without stalling, as shown in Figure 3-7. Table 3.1 shows the minimum output FIFO depth required to achieve a near-maximum performance for each of the parallelism options analyzed.

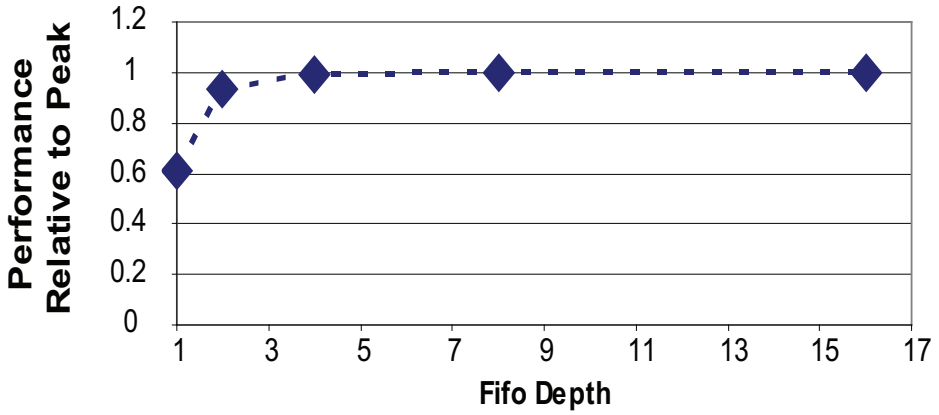


Figure 3-7: Parallel ($N = 4$) interpolator performance versus output FIFO depth

Table 3.1: Sufficient FIFO depths for different parallel interpolator architectures. The numbers represent the simulated performance for 100 frames of the "mobcal" 720p video.

Degree of Parallelism N	*Output FIFO depth for each interpolator*	*Performance relative to FIFOs of infinite depth*	*Total FIFO depth*
1	1	100%	1
2	1	99.89%	2
4	2	93.55%	8
8	4	98.89%	32
12	4	98.31%	48
16	4	98.32%	64
20	4	97.24%	80

To avoid stalling the parallelized interpolator by starving its inputs, the MEM and ED

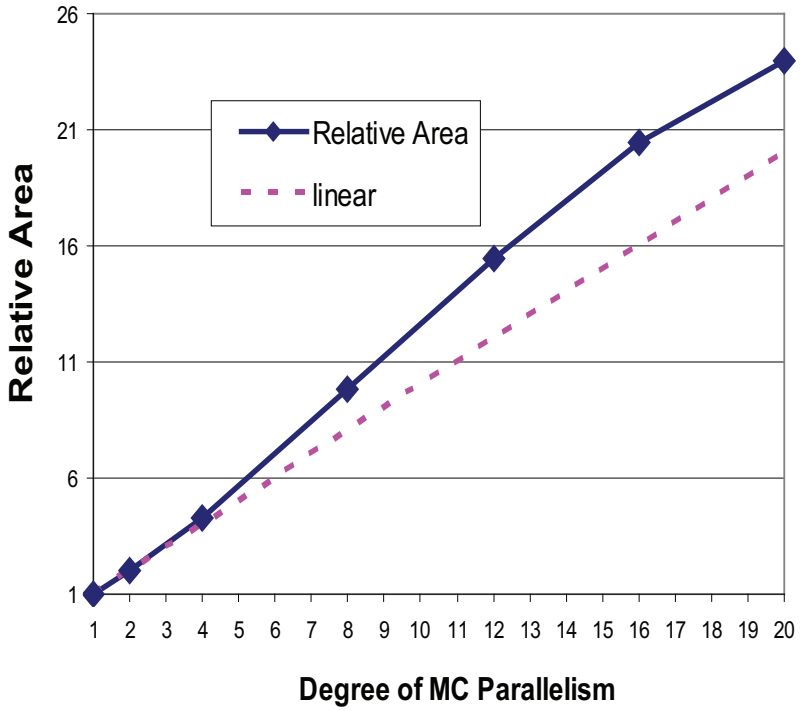
units should be parallelized to match the throughput. Alternatively, they could also run at a higher frequency than the MC unit, and have sufficient buffering of their outputs. The input FIFOs that hold the inputs for the parallel MC interpolators are not shown in Figure 3-3 and are absorbed within the MEM and ED units. Similarly, to avoid stalling the parallelized interpolator by filling its output FIFOs, the units that follow the MC must match its rate in blocks per second. Therefore, the DB, ADD, and MEM units must be also parallelized or be run at a higher frequency. In this analysis, the MC unit was simulated at a much lower frequency than the rest of the system, in order to avoid stalling generated by the other un-optimized units.

The main cost of interpolator parallelism is chip area. Figure 3-8a shows how the area grows with increased parallelism. The larger degrees of parallelism show super-linear area growth, mainly due to the increased output FIFO requirements shown in Table 3.1.

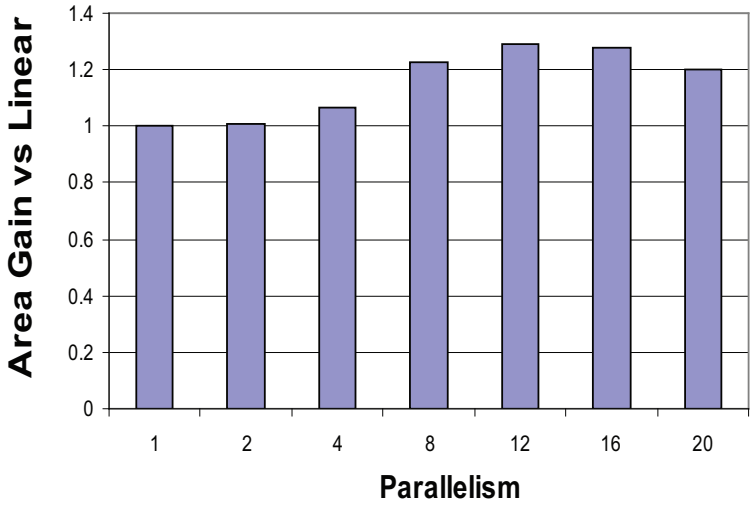
The resulting savings in energy due to increased parallelism are shown in Figure 3-9, normalized to the single-interpolator case running at full voltage. The energy is simulated using a post-layout netlist which includes wire parasitics. Note that even when the voltage is not scaled, total and dynamic energy is initially reduced due to a reduction in redundant MC computation.

For low degrees of parallelism, total energy decreases since it is dominated by dynamic energy, which decreases with voltage scaling. For higher degrees of parallelism, increased performance translates to smaller drops in voltage and therefore dynamic energy. This is because the current decreases faster at lower voltages. As a result, a minimum energy point of 58 pJ per 4x4 block can be seen for a parallelism of $N = 4$. This minimum energy point was also illustrated in Figure 3-2, where the minimum value was 68 pJ per 4x4 block. The difference in minimum values can be attributed to the fact that the 4-interpolator architecture eliminates many of the overlapping computations of the single-interpolator case. This difference was also seen in the super-linear performance gain of Figure 3-6b.

For higher degrees of MC parallelism, there is another factor that drives up the energy per 4x4 block. As the area gets larger, the routing complexity increases, since we need to distribute the inputs to more parallel processing elements and to collect the processed



(a) Parallel area normalized to $N = 1$



(b) Parallel area normalized to linear gain

Figure 3-8: Post-synthesis area overhead of MC interpolator parallelism

results and multiplex them onto the output. The increase in routing complexity leads to longer wires and has a direct effect on the power used in charging up the interconnect. This can be seen in Figure 3-11, which plots the normalized power of charging up the wires for

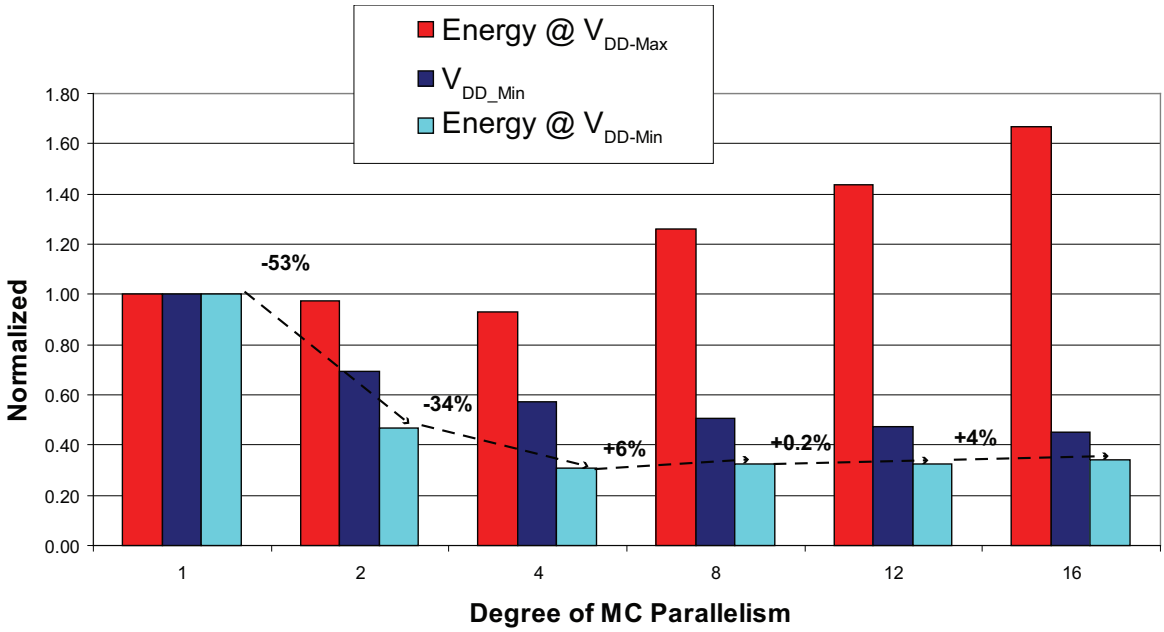


Figure 3-9: Energy savings of MC interpolator parallelism

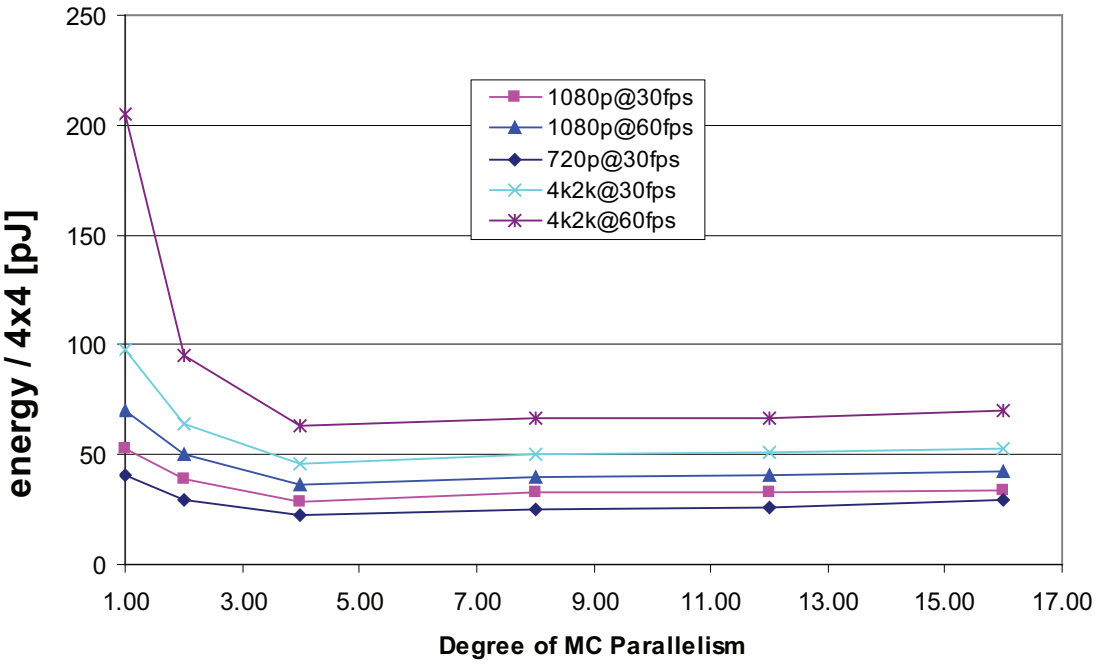


Figure 3-10: Energy of MC interpolation per 4x4 block. Dynamic energy decreases with parallelism initially, but then secondary effects such as wiring and muxing overhead drive the energy back up for further increases in parallelism.

various degrees of parallelism. The power numbers were obtained from the power report and normalized per cycle.

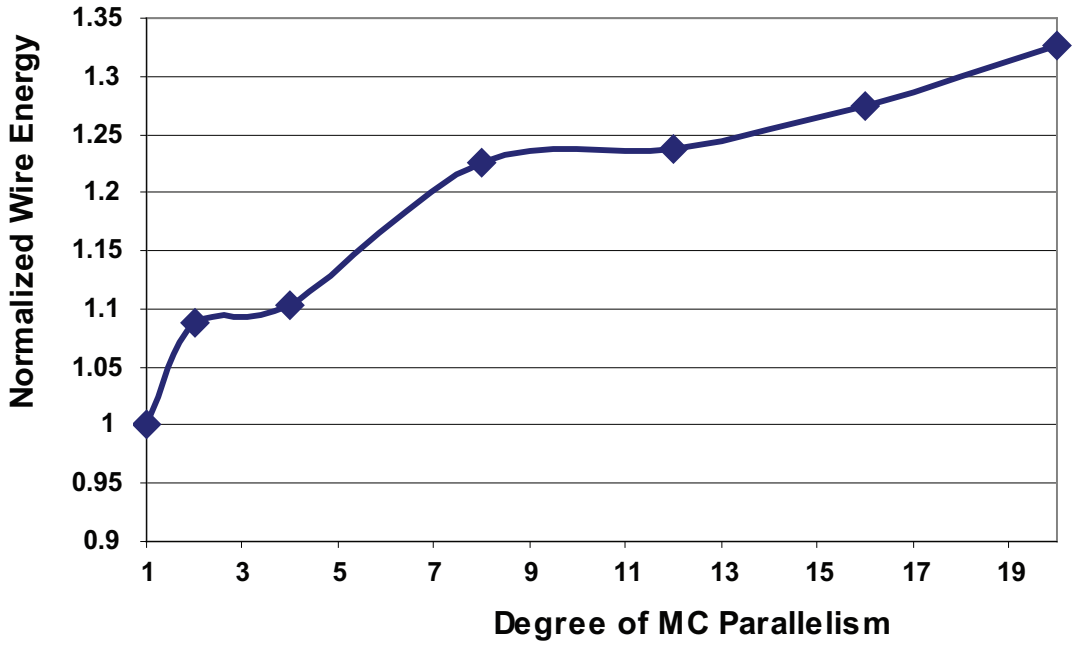
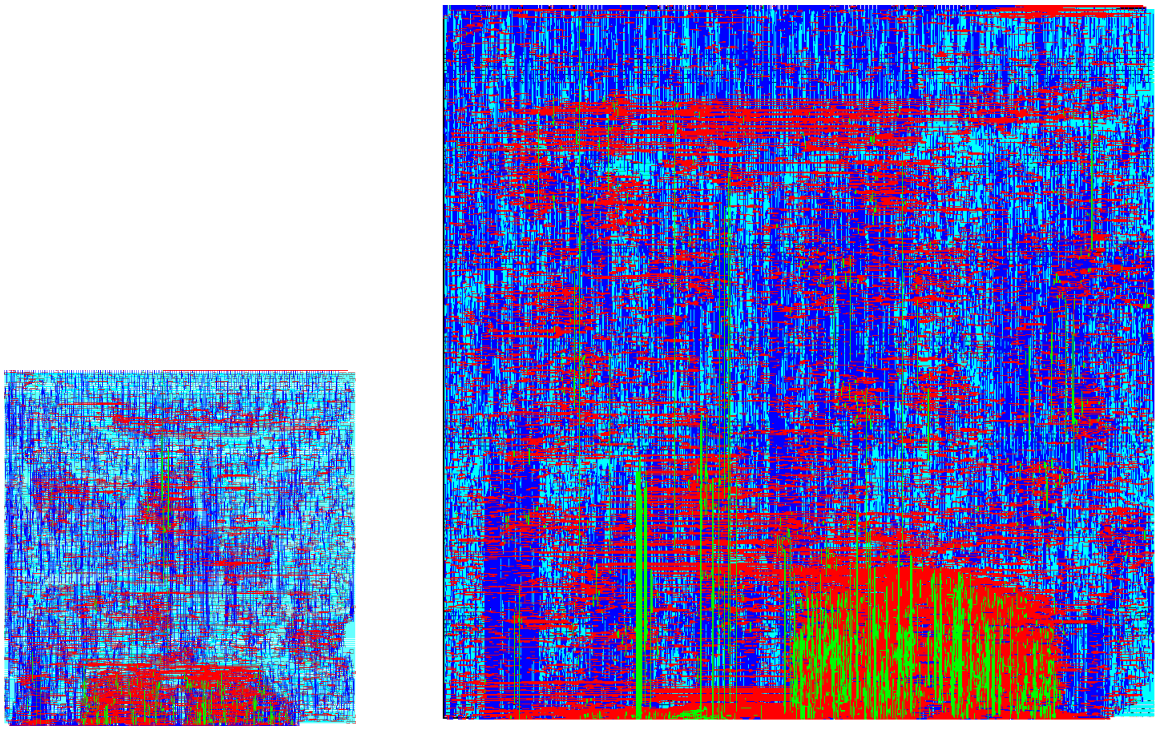


Figure 3-11: Normalized wire power for various degrees MC interpolator parallelism

Figure 3-12 shows the layout for a single MC interpolator and a parallel one with $N = 4$. The floorplan of the parallel interpolator is twice as large in each dimension and corresponds to the near-linear increase in standard-cell area.

3.3 Chroma Interpolator Parallelism

Chroma interpolation involves the use of a 2-D bilinear filter and each 2x2 chroma block is predicted from an area of 3x3 pixels. To speed up this operation, the chroma interpolator can be also be parallelized. For example, if it is replicated four times, a 2x2 block of pixels can be interpolated during every cycle, as shown in Figure 3-13. Each filter completes in one cycle and consists of four 8-bit multipliers and four 16-bit adders.



(a) MC interpolator layout for $N = 1$

(b) MC interpolator layout for $N = 4$

Figure 3-12: Comparison to scale of MC interpolator layouts, showing the nearly linear growth in area

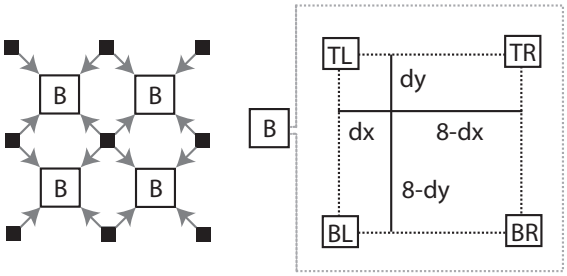


Figure 3-13: Chroma bilinear filter (B) is replicated 4 times

Chapter 4

Multi-Core Decoding

Chapter 2 described how parallelism can be applied within the video decoder (DEC) units (for example, MC or DB) to increase system performance. In this section, we will describe different ways in which two or more DECs can process a video in parallel and therefore increase system performance. The goal of these techniques is to enable N DECs to execute concurrently, in order to achieve a performance improvement of up to N. The added performance can be traded off for a lower operating voltage and power, as explained in Section 1.1.1. These techniques are also cumulative, so they could be used together to expose even more parallelism.

This section deals with both H.264-compliant video processing, as well as describing other ways to expose the desired parallelism by slightly modifying the H.264 algorithm. Specifically, the ideas of Section 4.1 and Section 4.2 are H.264 compliant, but the other ideas require slight changes to the H.264 standard.

Multi-core decoding consists of replicating an existing DEC architecture, as shown in Figure 4-1. Each of the parallel DECs parses different parts of the bitstream, and together they produce one output video. The frame buffer memory controller is shared between the parallel DECs, since they all share one off-chip memory. With enough buffering of memory reads and writes, the sharing of the memory controller should not introduce any stalls in the DEC cores. Similarly, the interface to the bitstream memory must also be shared by the different DECs, and it is assumed that this bitstream memory is randomly accessible.

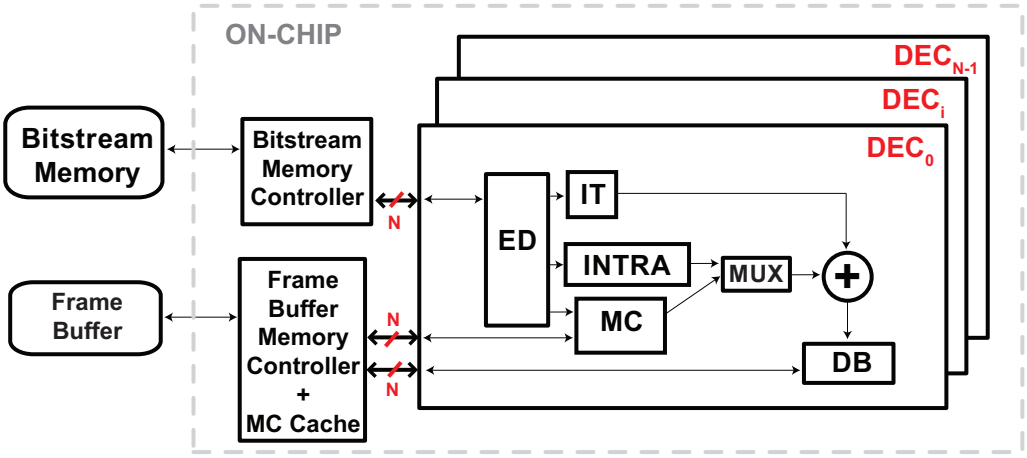


Figure 4-1: Parallel video decoder architecture

Section 4.1 shows how multiple DECs can parse several slices within one frame. Section 4.2 presents a way of decoding multiple H.264 frames simultaneously, while achieving a linear improvement in performance with no loss in coding efficiency. Section 4.3 introduces a new macroblock (MB) ordering that enables better DEC parallelism. Section 4.4 shows how to process slices in an interleaved way and thus greatly reduce the coding loss of H.264 slices. Section 4.5 proposes several bitstream controller architectures, including a new way of reducing the latency when buffering input slices or frames, which is required for all the parallel DEC techniques. Section 4.6 looks into the applicability of the multi-core decoding ideas to a multi-core software implementation. Section 4.7 summarizes and compares the different DEC parallelism techniques.

The proposed architectures were implemented using Verilog and the coding loss was simulated using the H.264 reference software [36]. The underlying DEC architecture used for all the analysis is based on the implementation of Chapter 6.

The development of the ideas presented in Section 4.1, Section 4.3, and Section 4.4 was done in collaboration with Vivienne Sze, who also performed the coding efficiency simulations featured in Section 4.1. I led the RTL implementations of the multi-core ideas of this chapter together with the performance, power and area analysis.

4.1 Slice Multi-Core Decoding

There is a simple scheme that enables multi-core H.264 decoding for increased performance or lower operating voltage. It consists of dividing a frame into two or more slices at the video encoder (ENC). Each slice can be processed by a separate DEC, as shown in Figure 4-2. Parallel slice processing relies on the ability of the DEC's entropy decoder (ED) to parse two or more slices simultaneously, and also assumes that the ENC divides each frame into enough slices to exploit parallelism at the DEC.

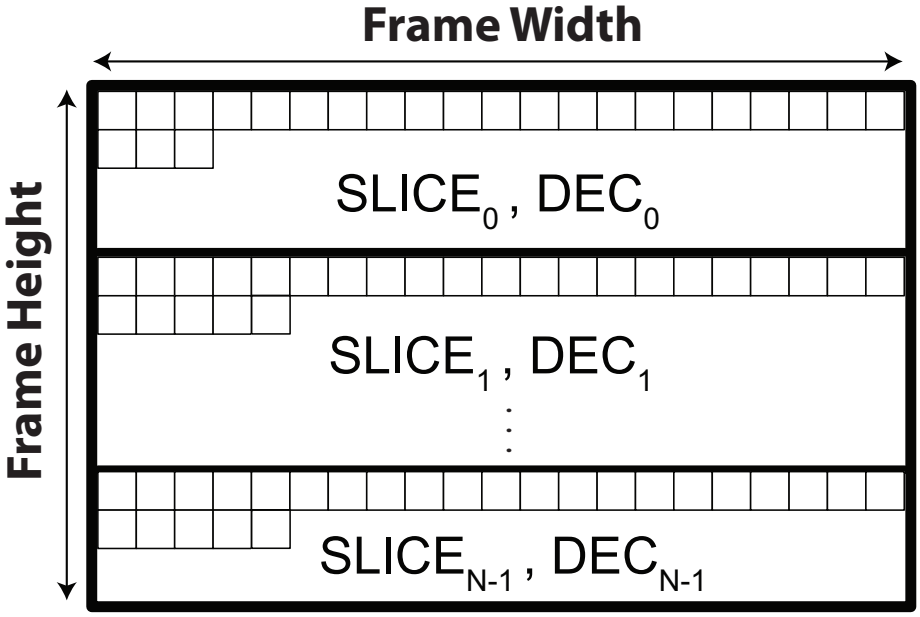


Figure 4-2: Dividing a frame into slices enables parallelism within a frame

Consider the case of slice parallelism for $N = 3$ and 30 fps. The corresponding timing diagram is shown in Figure 4-3. The three different DECs are staggered by approximately 11ms (one third of a frame period), such that each DEC finishes just in time for its part of the frame to be shown by the DISPLAY process.

In the H.264 standard [3], each slice is preceded by a small 32-bit delimiter code, as shown in Figure 4-4. If the DEC can afford to buffer an entire encoded frame of the input stream and quickly parse for the start code of all slices, then it can simultaneously read all the slices from this input buffer. This idea is similar to the parallel MPEG-2 decoder described in [28].

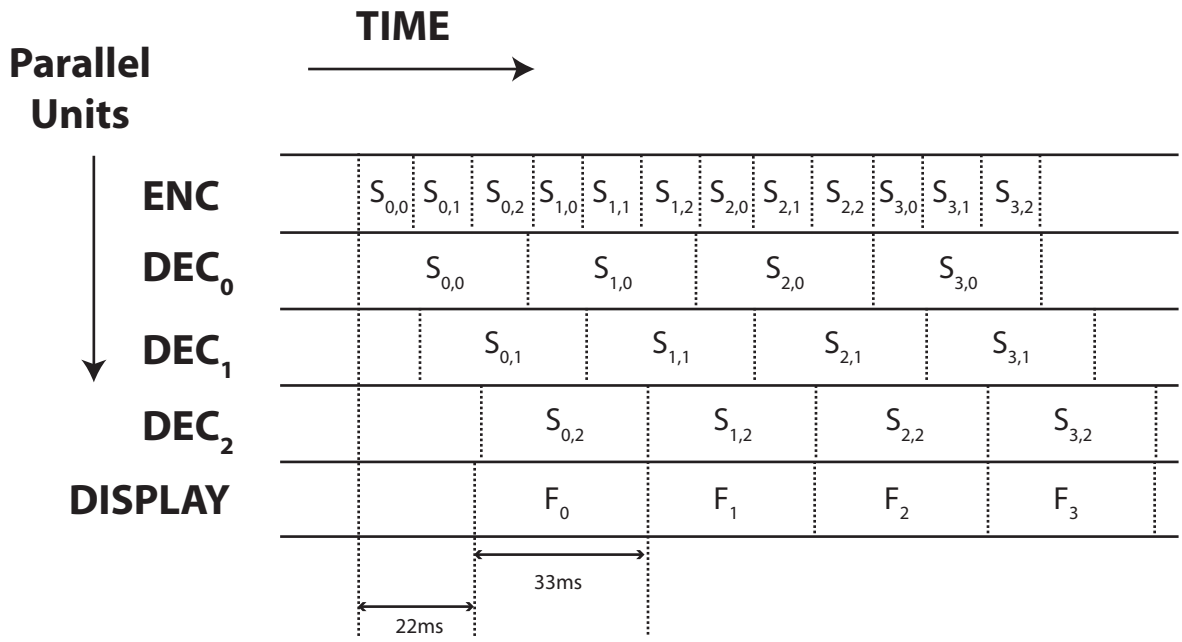


Figure 4-3: Timing diagram of slice parallelism for $N = 3$

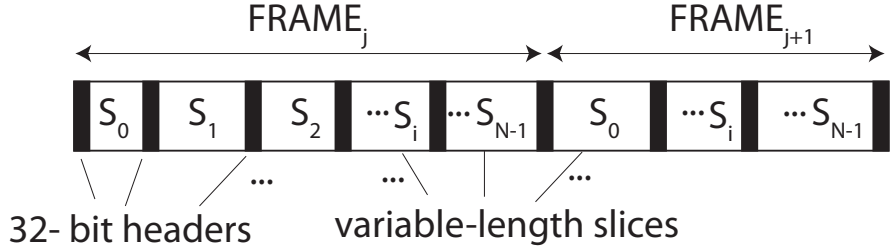


Figure 4-4: Start of slices can be found by parsing for headers. This figure shows each frame divided into N different slices.

The slice parallelism scheme is compatible with H.264 and trades off increased parallelism for a decrease in coding efficiency. We evaluated the impact of slice parallelism by encoding 150 frames of four different video sequences and separating each frame into a fixed number of slices, using the JM reference software [36] with QP=27. The result is shown in Figure 4-5. Relative to having single-slice frames, the coding efficiency decreases because the redundancy across the slice borders is not exploited by the ENC. Furthermore, the size of the slice header information is constant while the size of the slice body decreases because it contains fewer 16x16 pixel MBs. For example, when dividing a 720p video coded with QP=27 into 8 slices,

the CAVLC coding method suffers an average 1.54% coding loss, when measured under common conditions [37].

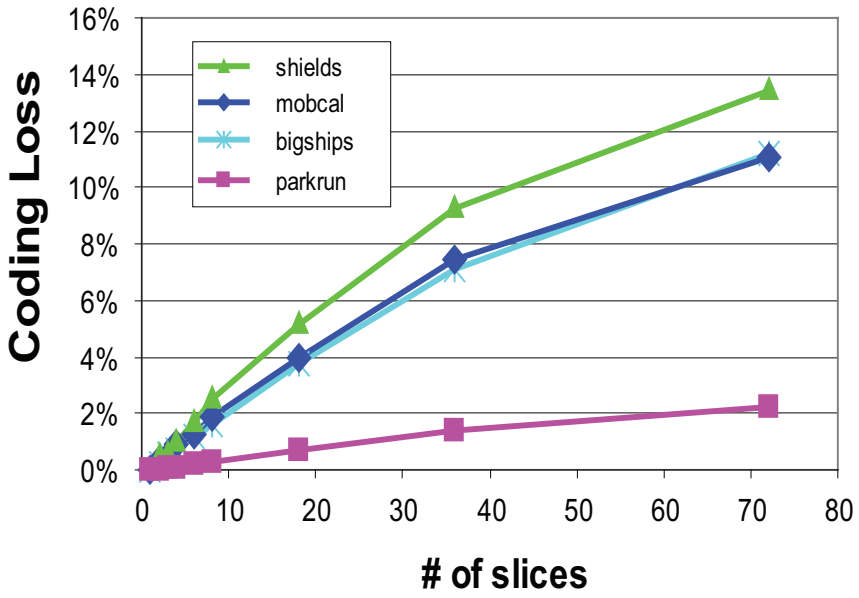


Figure 4-5: CAVLC coding loss increases with number of H.264 slices in a 720p frame

Beside the loss in coding efficiency, another disadvantage of the slice partitioning scheme is that the full-last-line caches (FLLCs) of Section 5.1 need to be replicated together with each DEC, since they operate on completely different regions of the frame. This causes the area overhead of parallelism to be nearly proportional to the degree of parallelism. In some DEC implementations the on-chip cache dominates the active area (75% as will be shown in Section 6.7), so replicating the FLLCs might be avoided if area is of critical importance. If the FLLCs are not replicated for each DEC, this increases off-chip BW and corresponding power, as discussed in Section 5.1.

Ideally, the performance improvement of slice parallelism with N decoders is at most N. However, there are two reasons why the performance does not reach this peak. First, the workload is not evenly distributed amongst the parallel slices, especially since they operate on disjoint regions of the frame which could have different coding characteristics. Second, the increase in total bits per MB due to loss in coding efficiency (more non-zero coefficients, for example) leads to an increase in ED computation cycles. Using the sizes of the encoded JM

slices as an estimate of ED performance, slice parallelism can only achieve a $2.51X$ relative performance for $N = 3$. The performance impovement of H.264 slice multi-core parallelism is shown in Figure 4-6. As the number of slices increase, the multi-core performance moves further away from the linear increase since the workload distribution across the different slices becomes more uneven and the number of compressed bits per frame also gets larger.

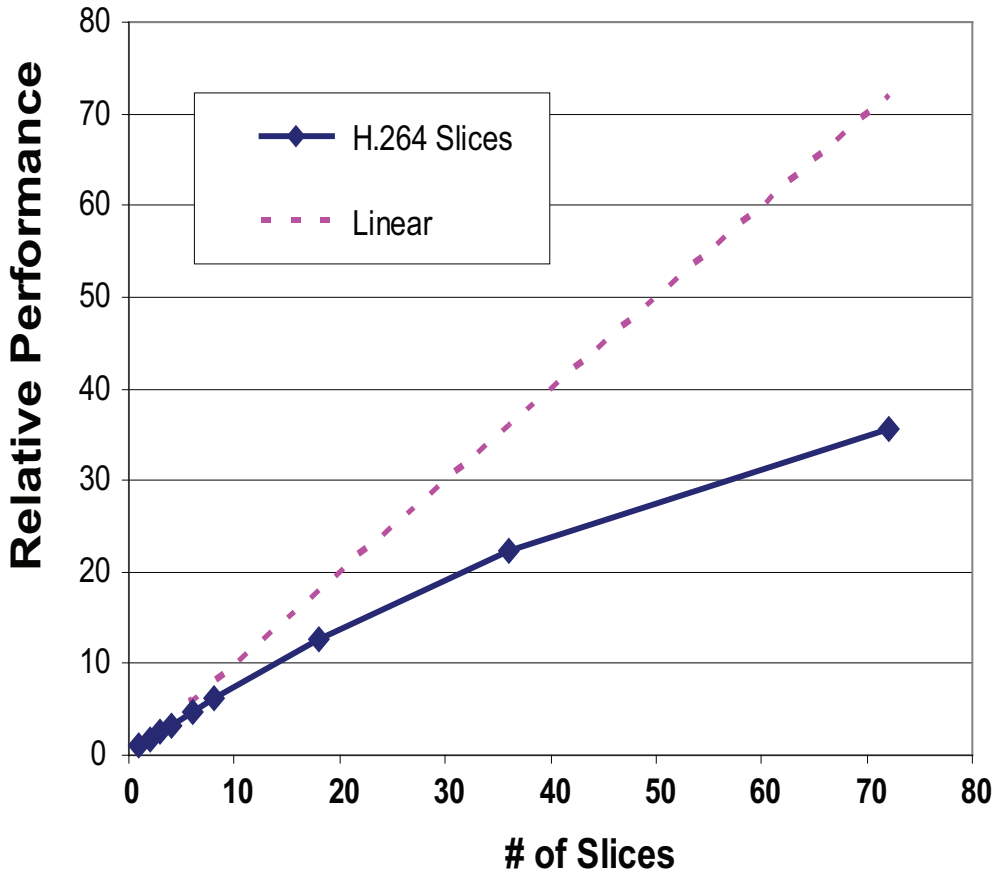


Figure 4-6: Performance of H.264 slice multi-core parallelism for 100 frames of the 720p "mobcal" video sequence. When many slices are used, the performance increase is not proportional due to uneven distribution across the slices and the extra CAVLC processing required for each slice.

4.2 Frame Multi-Core Decoding

In this section, we show how to process N consecutive H.264 frames in parallel, without requiring the ENC to perform any special operations, such as splitting up frames into N slices. Once again, the motivation for this parallelism is either increased performance or lower supply voltage. The simultaneous parsing of several frames relies on input buffering and searching for delimiters, similar to the discussion of Section 4.1. However, note that this technique requires buffering N frames, so it will incur a higher input latency than the buffering of N slices.

Several consecutive frames can be processed in parallel by N different DECs, as shown in Figure 4-7. The main cost of multi-frame processing is the area overhead of parallelism, which is proportional to the degree of parallelism, just as in Section 4.1. If these frames are all I-frames (spatially predicted), then they can be processed independently from each other. However, when these frames are P-frames (temporally predicted), DEC_i requires data from FB location FB_{i-1}, which was produced by DEC_{i-1}. If we synchronize all the parallel DECs, such that DEC_i lags sufficiently behind DEC_{i-1}, then the data from FB_{i-1} is usually valid.

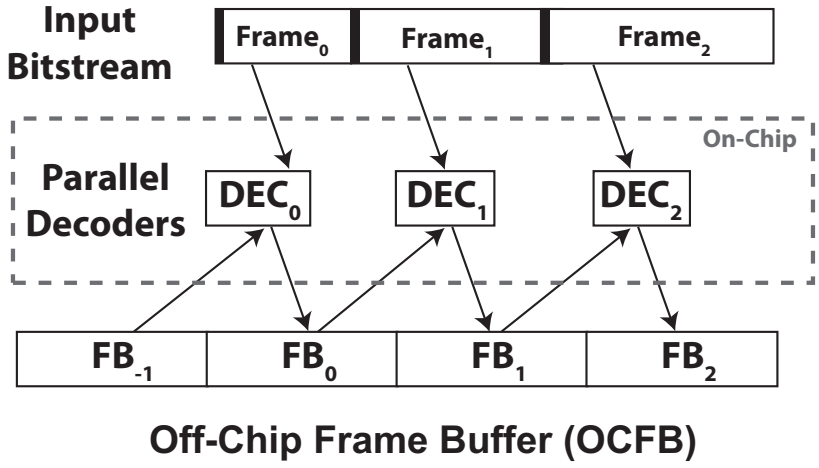


Figure 4-7: Three parallel video decoders processing 3 consecutive frames

Consider the case of frame parallelism for $N = 3$ and 30 fps. A corresponding timing diagram of the parallel units is shown in Figure 4-8. For 30 fps, a new frame must be

displayed every 33ms. Since there are 3 parallel DECs, each DEC can take about 100ms to decode one frame. Figure 4-8 shows that the input buffering latency is 66ms, from the time that $frame_i$ arrives from the ENC and begins to be decoded to the time that it begins to be displayed.

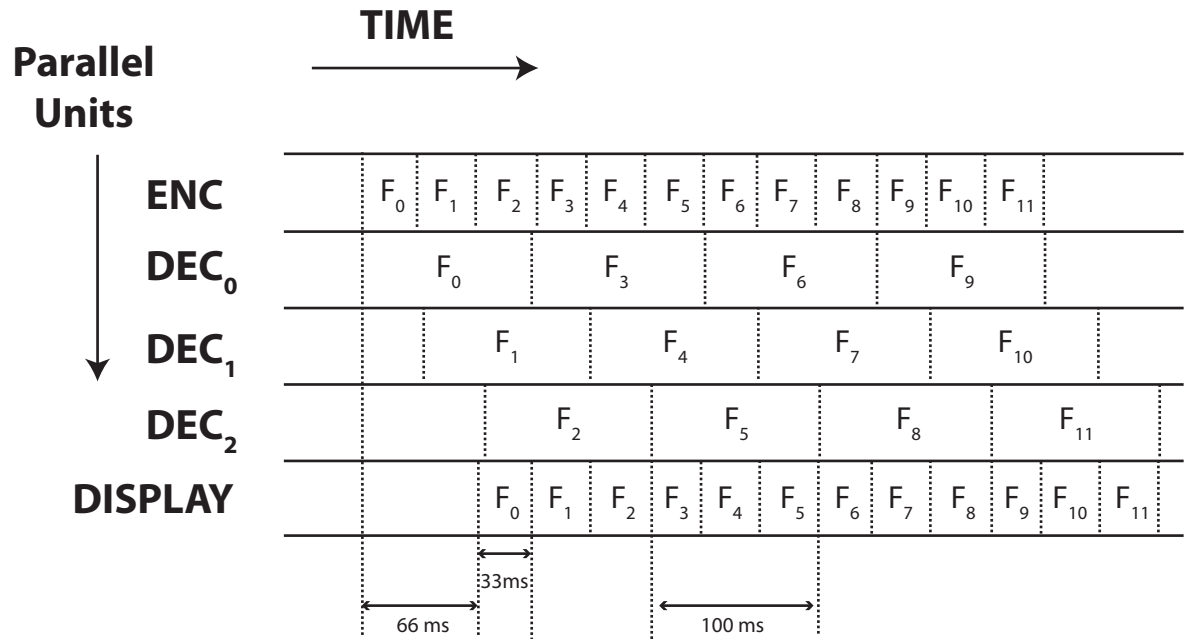


Figure 4-8: Timing diagram of frame parallelism for $N = 3$

This staggered arrangement of the DECs is also illustrated in Figure 4-9. If the motion vector in DEC_i requires pixels not yet decoded by DEC_{i-1}, then concurrency suffers and we must stall DEC_i. This could happen if the y-component of the MV is a large positive number. This stall is illustrated in Figure 4-9, which shows how DEC_1 wants to read data from a location in the previous frame. Since this location is not yet reached by DEC_0, which is processing the previous frame, DEC_1 must stall until DEC_0 reaches this area. These types of stalls can eventually propagate to the other decoders (DEC_2, ...), thereby degrading system performance.

This type of parallel processing can increase the DEC performance by up to a factor of N. The parallel frame processing architecture was implemented in Verilog using the core of Chapter 6 for each of the DECs. The architecture was then verified for different

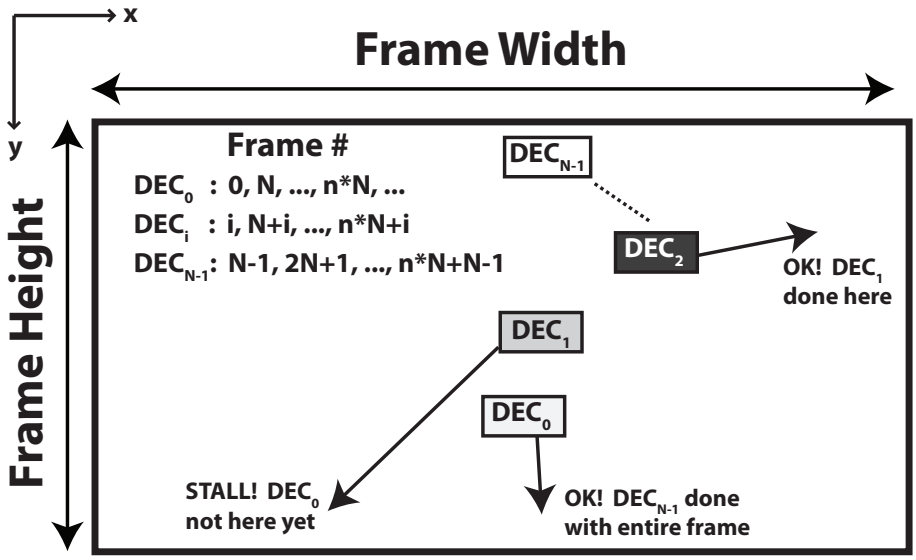


Figure 4-9: Snapshot of N parallel video decoders and their position in their respective frames

video sequences and varying degrees of parallelism. Figure 4-10 shows how the clock period increases for a given resolution as we process more frames in parallel. This increase is nearly linear, but is limited by the workload imbalance across the various sets of frames running on each of the parallel DECs.

The performance decrease due to the stalls described in Figure 4-9 was simulated to be less than 1% for $N = 3$, across 100 frames of a 720p "mobcal" video sequence. The relatively small number of stalls for the simulated videos can be understood by examining the statistics of their vertical motion vectors. As shown in Figure 4-11, the motion vectors for various videos are typically small and have a very tight spread, which minimizes stalling.

4.3 Diagonal Macroblock Processing

The H.264 coding standard processes the macroblocks (MBs) of video frames in raster-scan order. In order to exploit spatial redundancy, each MB is coded differentially with respect to its already-decoded neighbors to the left (L), top-left (TL), top (T), and top-right (TR), as seen in Figure 4-12. The redundancy between neighbors is present in both pixel values

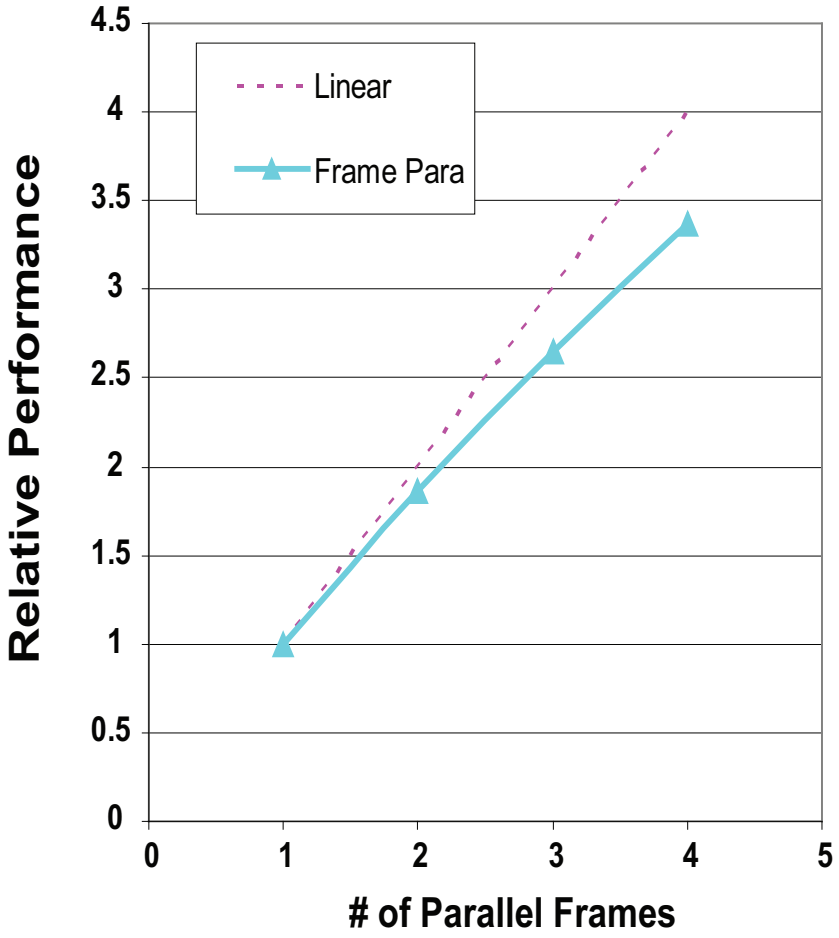
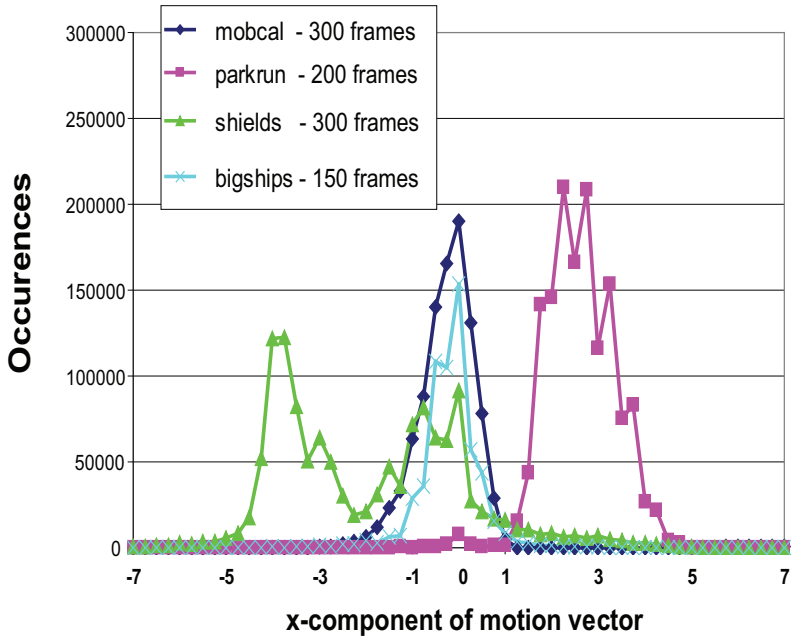


Figure 4-10: Performance of frame multi-core parallelism for 100 frames of the 720p "mobcal" video

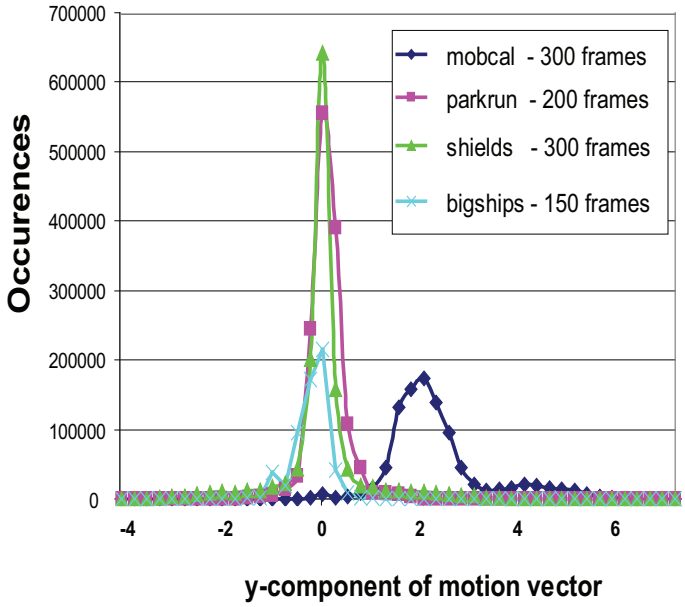
and control information (motion vectors, number of coded coefficients, etc).

In theory, we could instantiate two identical DECs to simultaneously process two consecutive MBs (for example, "Current" and "Left" in Figure 4-12). However, due to the dependency shown in Figure 4-12, the "Left" MB should be fully decoded before the "Current" one can be started. This means the two parallel DECs could not run at the same time for these two MBs without a lot of stalling.

As an alternative, the parallel DECs can process MBs on a 2:1 diagonal as shown in Figure 4-13. This is similar to the parallel software processing order described in [25]. The diagonal



(a) Horizontal motion vectors



(b) Vertical motion vectors

Figure 4-11: Distribution of vertical motion vectors for several conformance videos showing a tight spread

height D could be set to anywhere from 1 to H (frame height). The different diagonals are ordered from left to right. Setting $D = 1$ corresponds to the typical raster-scan processing order.

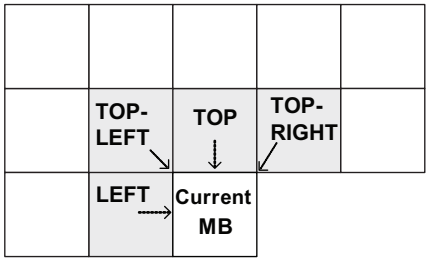


Figure 4-12: Spatial dependency on neighboring macroblocks

If diagonal processing is used, all the MBs on a diagonal can be decoded concurrently since there are no dependencies between them. If all MBs had similar processing workloads, the scheme described in this section could speed up the DEC by N (degree of DEC replication). In reality, the workload per MB does vary, so the performance improvement is lower than the increase in area. The diagonal height D of each region of diagonals can be set to N, since no further parallel DEC hardware is available. Note that the top line of MBs in each region of diagonals is still coded with respect to the MBs in the region of diagonals just above, in order to maintain good coding efficiency, as opposed to the slice parallelism of Section 4.1.

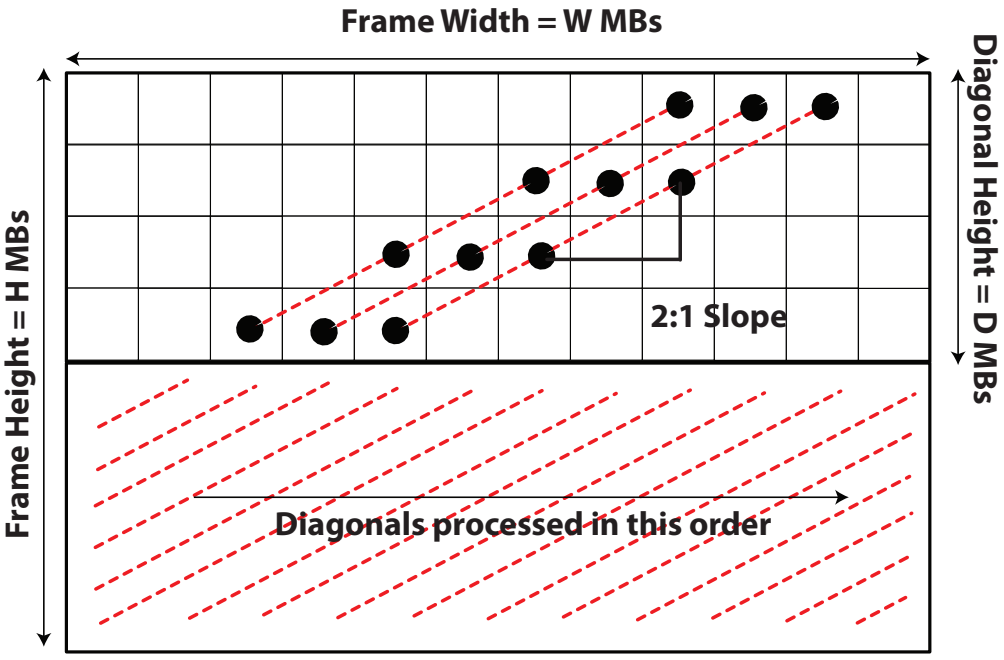


Figure 4-13: 2:1 diagonal processing order

A limitation to implementing this scheme is that the coded MBs in H.264 arrive in raster-scan order from the bitstream. One solution would be to modify the algorithm and reorder the MBs in a 2:1 diagonal order at the ENC. For example, the MBs in each diagonal could be transmitted from top-right to bottom-left in the bitstream. Another ordering could transmit MBs on even diagonals top-right to bottom-left and MBs on odd diagonals from bottom-left to top-right.

Diagonal reordering would require a change in the H.264 standard, and both the ENC and DEC would have to process MBs in a diagonal order. The CAVLC entropy coding efficiency would not suffer, since each MB can be coded in the same way as for the raster-scan ordering of H.264. Therefore, the reordered CAVLC bitstream would contain the same bits within the MBs, but the MBs would just be rearranged in a different order.

Even if diagonal reordering is used, the ED unit still cannot scan ahead to the next MB since the current MB has variable length and there are no MB delimiters. This critical challenge is addressed in the next section.

4.4 Interleaved Entropy Slice (IES) Multi-Core Decoding

In order to enable DEC parallelism when using the diagonal scanning order of Section 4.3, we propose the following solution. The bitstream can be split into N different interleaved entropy slices (IESs). An "entropy" slice refers to the fact that two adjacent slices are not completely decoupled, and coding can still be performed across the border. For example, if the slices in Figure 4-2 were entropy slices, the top row of $SLICE_1$ could be intra-predicted with respect to the bottom row of $SLICE_0$. The meaning of the word "entropy" in the IES acronym signifies that the slices do not change their entropy when a frame is split up, so the CAVLC coding efficiency is not affected by the partitioning of the slices.

Instead of splitting a frame into the slices of Figure 4-2, the slices can be interleaved among the MB lines, as shown in Figure 4-14. Each of N parallel DECs is then assigned to one of the IESs. Just as slices are separated for H.264, the compressed bitstream could be

split into different IESs.

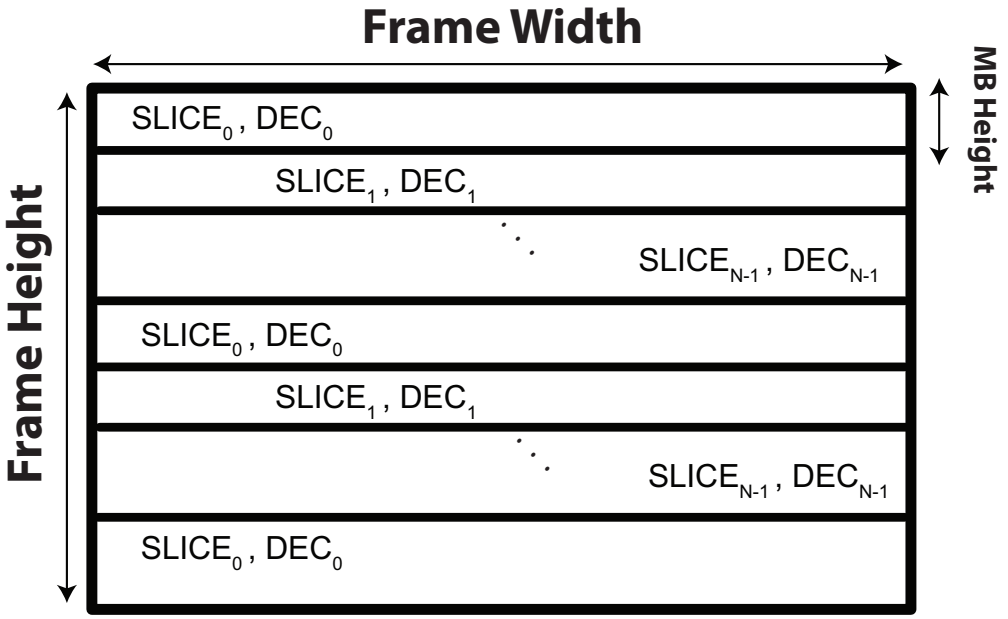


Figure 4-14: A frame can be divided into interleaved slices which alternate among the MB lines

There are two key differences between IESs and the entropy slices of [29]. IES processing order has no loss in coding efficiency due to border effects, whereas [29] loses some of the coding context across the slice borders. Additionally, IESs are interleaved to enable better parallel processing and memory locality.

For example, Figure 4-15 shows the IES processing method for $N = 2$, with the IESs split between DEC_0 and DEC_1. In this example, IES_0 is made up of all the even MB rows, while IES_1 is made up of all the odd MB rows. When DEC_0 finishes processing MB row 0, it starts processing MB row 2, and so on.

Consider the case of IES parallelism for $N = 3$ and 30 fps. The corresponding timing diagram is shown in Figure 4-16, where $S_{i,j}$ represents IES_j of frame_i. The index j varies from 0 to N-1. Since the slices are interleaved the processing of each of the N IESs begins and ends almost at the same time, with the difference being the time it takes to process a couple of MBs. Therefore, the maximum latency between the arrival of IES_0 and the start of its decoding by DEC_0 is 22 ms.

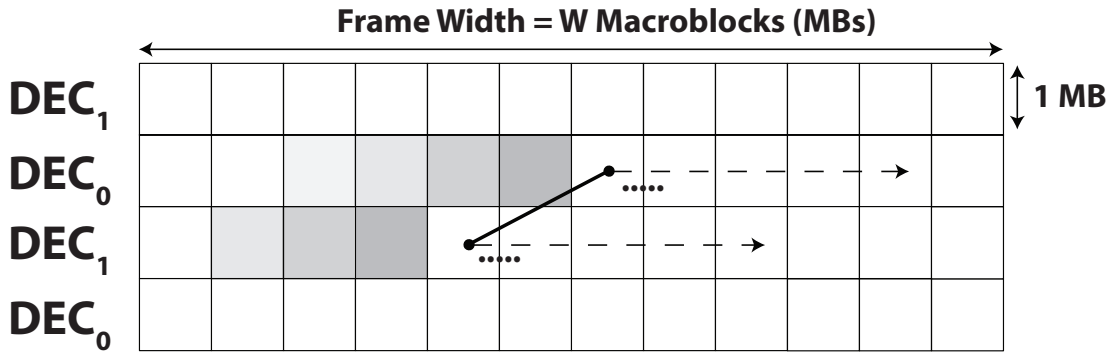


Figure 4-15: Interleaved entropy slices (IESs) with diagonal dependencies

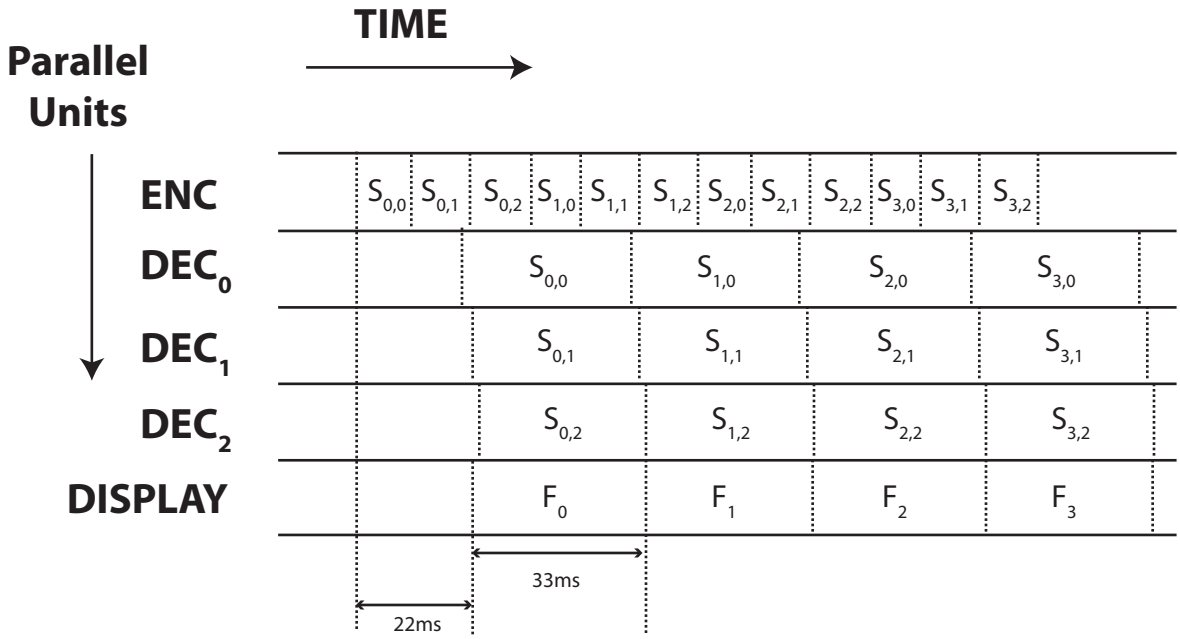


Figure 4-16: Timing diagram of IES parallelism for $N = 3$

The processing of IESs would be synchronized to ensure that the 2:1 diagonal order is maintained. As a result, each DEC must trail the DEC above. However, if one IES has a higher instantaneous processing workload than the IES above it, the DEC above can move forward and proceed further ahead, so that stalling is minimized.

This approach is different than the one used in [25]. In that work, the ED processing was done in the usual raster scan order and all the syntax elements were buffered for one frame. The diagonal processing could only start after the entire frame was processed by ED.

In the IES approach, which would be enabled by a change in the H.264 algorithm, even the ED processing is done in parallel on a diagonal, which speeds up the ED operation and does not require buffering any MB syntax elements.

This technique is similar to the dual macroblock pipeline of [15]. In that work, the authors duplicate the MB processing hardware at the encoder, whereas here we replicate the DECs at the decoder. While the encoder has the flexibility to process MBs in any order, interleaved processing at the decoder requires a change in the H.264 standard.

It is worth considering how the use of IESs affects the entropy coding efficiency. Once again, if the video uses CAVLC, the bitstream size will only be slightly affected, since the macroblocks are coded in the same way as the raster-scan order of H.264. The only coding overhead is the 32 bits used for the slice header and at most 7 extra bits for byte alignment between slices. As we see in Figure 4-17, this scheme offers much better coding efficiency than using CAVLC with H.264 slices, since there is no loss in coding efficiency at the borders between IESs.

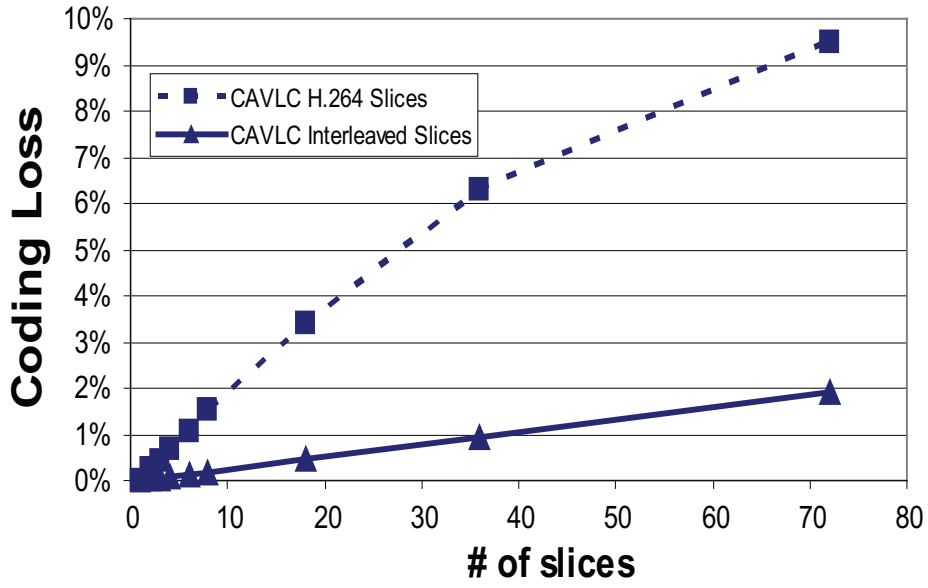


Figure 4-17: Average CAVLC coding efficiency of interleaved entropy slices (IESs) relative to parallel slice processing of Section 4.1 averaged over 150 frames of 4 different videos: "bigships", "mobcal", "shields" and "parkrun"

The performance of IES multi-core decoding is shown in Figure 4-18 for varying N. Ideally,

the IES multi-core technique can speed up the DEC performance by up to a factor of N. In reality, this cannot be achieved due to varying slice workloads (as discussed in Section 4.1) and stalls due to synchronization between the DECs. To evaluate the actual performance of a real system, we implemented the IES parallelism scheme in Verilog and evaluated it for several videos and degrees of parallelism. As an example, when $N = 3$, the relative performance is $2.91X$, which is close to the ideal of $3X$. There are two reasons why IESs perform better than regular H.264 slices ($2.51X$). First, the workload variation is not as large between interleaved slices since they cover similar regions of a frame; as N increases, however, the variation in interleaved slice workloads also gets larger. Second, IES parallelism does not suffer from a large coding penalty, so the ED performance does not decrease as a result.

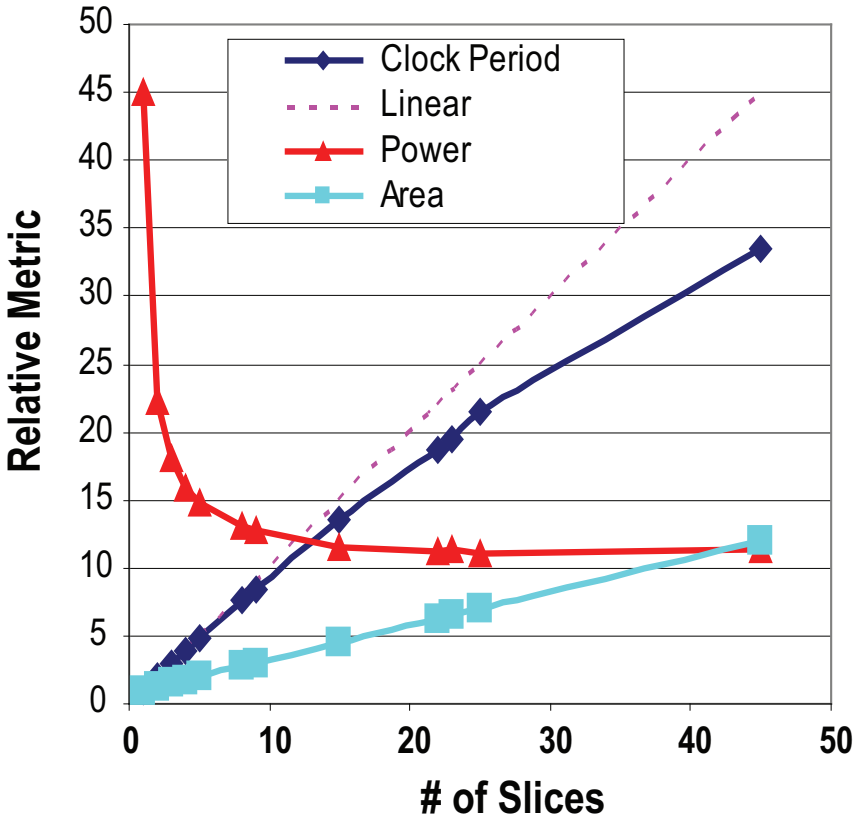


Figure 4-18: Performance of IES multi-core decoding. The power is normalized relative to a single decoder running at the nominal supply voltage. The area increase assumes caches make up 75% of the area of a single DEC (see Section 6.7).

4.5 Bitstream Controller

The video decoder (DEC) replication techniques described in Section 4.1, Section 4.2 and Section 4.4 rely on the ability of the DEC to parse two or more slices or frames in parallel. It is also assumed that the ENC and the DEC can agree on the number of slices per frame, or on the total number of DECs. One way the DEC can read from several slices at once is if the ENC serially orders the slices and separates them by slice delimiters, as was shown in Figure 4-4. During each cycle, the DEC reads from one of the several slice pointers, in a round robin fashion among the DECs requesting new input data.

When none of the DECs are reading from the input bitstream, the bitstream controller reads from the input in order to find the next slice header, as shown in Figure 4-19. If the frequency of this controller is too low, there will be no free cycles to read ahead for the next header, and parallel processing will have to stall. In order to avoid this, the bitstream controller should run at about twice the frequency relative to a non-parallel DEC since the bitstream is essentially parsed twice.

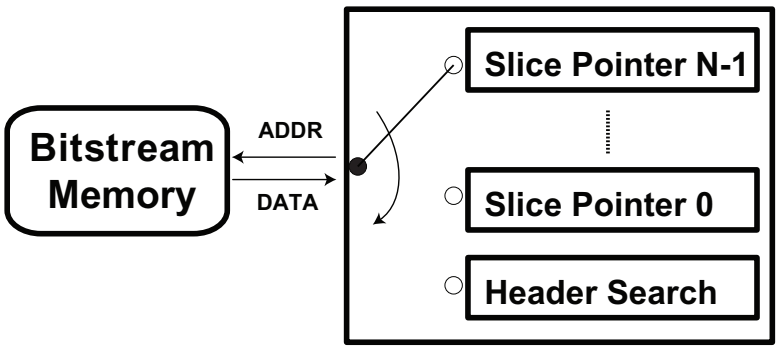


Figure 4-19: Bitstream controller supporting multiple slices and header search

An alternative to parsing for the slice delimiters is to also transmit the size of each slice at the start of a frame, as shown in Figure 4-20. This enables the DEC to easily find an index into the input buffer without having to scan the entire frame for slice headers. Encoding either the slice size or delimiter into the bitstream has a negligible effect on coding efficiency, as the coded size of each frame is quite large for the high resolutions targeted by parallel DECs. Note that the slice size does not need to be placed there by the ENC, but could also

be computed the first time the bitstream is received and placed into the bitstream memory.

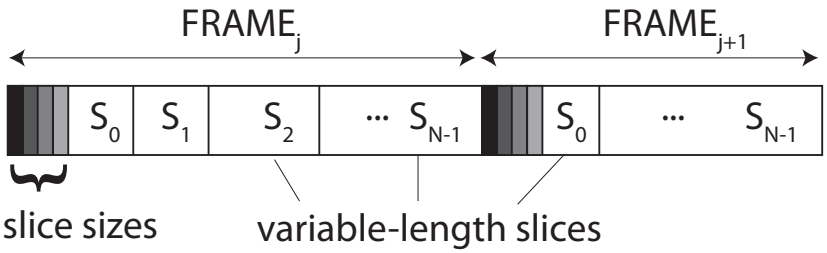


Figure 4-20: Size of all slices are encoded at the start of each frame

If a DEC can afford to buffer several slices or frames of the input stream and quickly parse for the start of each slice, it can simultaneously read and process all slices from this input buffer. The cost of this is an increase in latency from the time the frame is received by the DEC to the earliest time it can begin to be displayed. If the input buffering latency cannot be tolerated, a third scheme is proposed, and shown in Figure 4-21. At the ENC, each slice is chopped up into small segments, where segments in each slice have a fixed widths: $W_0, W_1, and W_2$. The stream then alternates between segments from each slice. The size of the segments for the different slices can be different, which ensures that the slices are synchronized even when their total sizes differ. Relative to the scheme of Figure 4-4, this method requires a much smaller input buffer to allow parallel slice processing.

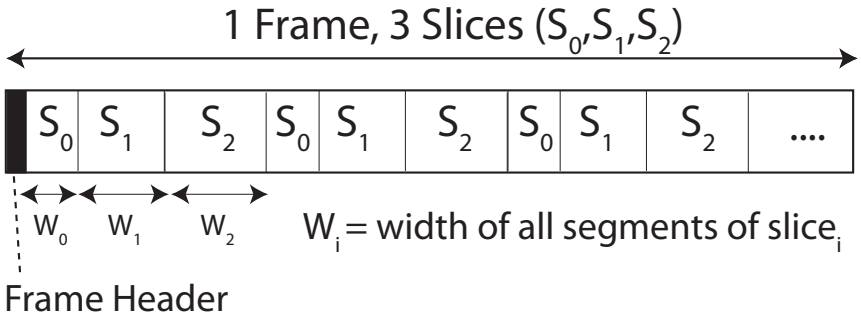


Figure 4-21: Splitting slices into fixed-length segments

4.6 Software Applicability of Multi-Core Decoding

This section discusses which of the multi-core techniques introduced in Chapter 4 are useful for a software implementation on a parallel processor machine similar to that of Figure 4-22.

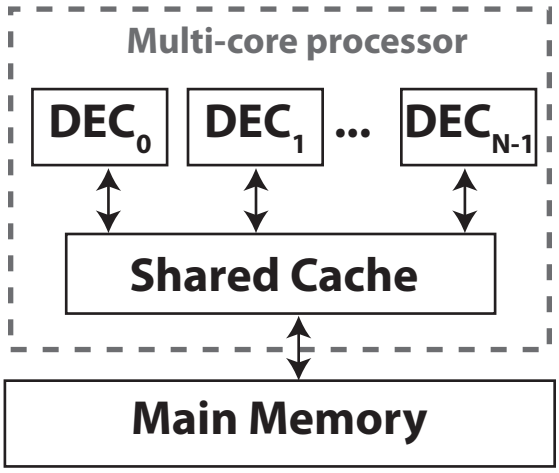


Figure 4-22: Running parallel software video decoders (DECs) on a multi-threaded machine

The slice parallelism technique of Section 4.1 can be applied to a N-core software implementation. If each of the N slice decoders is assigned to one of the processors, a speedup of up to N can be achieved. This was also demonstrated by the work in [26]. The work in [26] also showed that to minimize the inter-processor communication, each core should implement a full DEC instance, rather than dividing the DEC units among the different cores.

The frame parallelism technique of Section 4.2 is also applicable to a N-core software implementation. Each of the N frame decoders can run on one of the processors. Some overhead cycles will be needed to ensure that the synchronization is maintained between each pair of DEC_{i-1} and DEC_i of Figure 4-7. Additionally, there will also be some stall cycles whenever a vertical MV is positive and large, as was explained in Figure 4-9.

The idea of IES processing introduced in Section 4.4 is similarly applicable to a software implementation on a multi-threaded parallel processor. If each of the N threads runs an instance of the DEC, a software performance improvement of up to N is achievable.

4.7 Multi-Core Decoding Comparison

The multi-core decoding schemes described in the previous sections were implemented by replicating a particular DEC, though they should be applicable to most DEC implementations. The different architectures proposed were built, verified, and benchmarked in Verilog. Figure 4-23 shows that all multi-core architectures achieve a near-linear speedup and corresponding clock frequency reduction for a given resolution. However, as was shown in Figure 4-18, extending the level of multi-core parallelism to much higher than 3 achieves relatively small power savings at the cost of a much larger area. As a result, we compare these different multi-core architectures for $N = 3$, as shown in Table 4.1.

The following sections describe how the different fields in Table 4.1 were computed.

Relative Performance

Table 4.1 lists the performance achieved when the decoder is replicated 3 times, relative to the performance of a single decoder. The relative performance for slice multi-core is estimated from the relative size of the encoded slices, assuming performance is limited by the ED unit. The relative performance of frame and IES multi-core was simulated in Verilog.

Equivalent Dynamic Power Savings

The dynamic power savings are computed from voltage scaling according to Equation 1.1. The scaling is done from the maximum process voltage of 1.2 V down to the voltage that slows down the circuit by the same factor as the relative performance gain of Table 4.1. As discussed in Section 1.1.1, extra performance can be traded off for a slower clock and lower voltage. If the single DEC's operating voltage is lower than the full voltage (1.2 V), the power savings due to multi-core decoding decrease. For example, if multi-core provides a 2X increase in performance, this allows the supply voltage to scale from 1.2V to 0.83V, which yields 52% dynamic energy savings. However, if the starting voltage is 1.0V, a 2X increase in the clock period allows voltage scaling down to 0.77V, which only saves 41% of the dynamic energy.

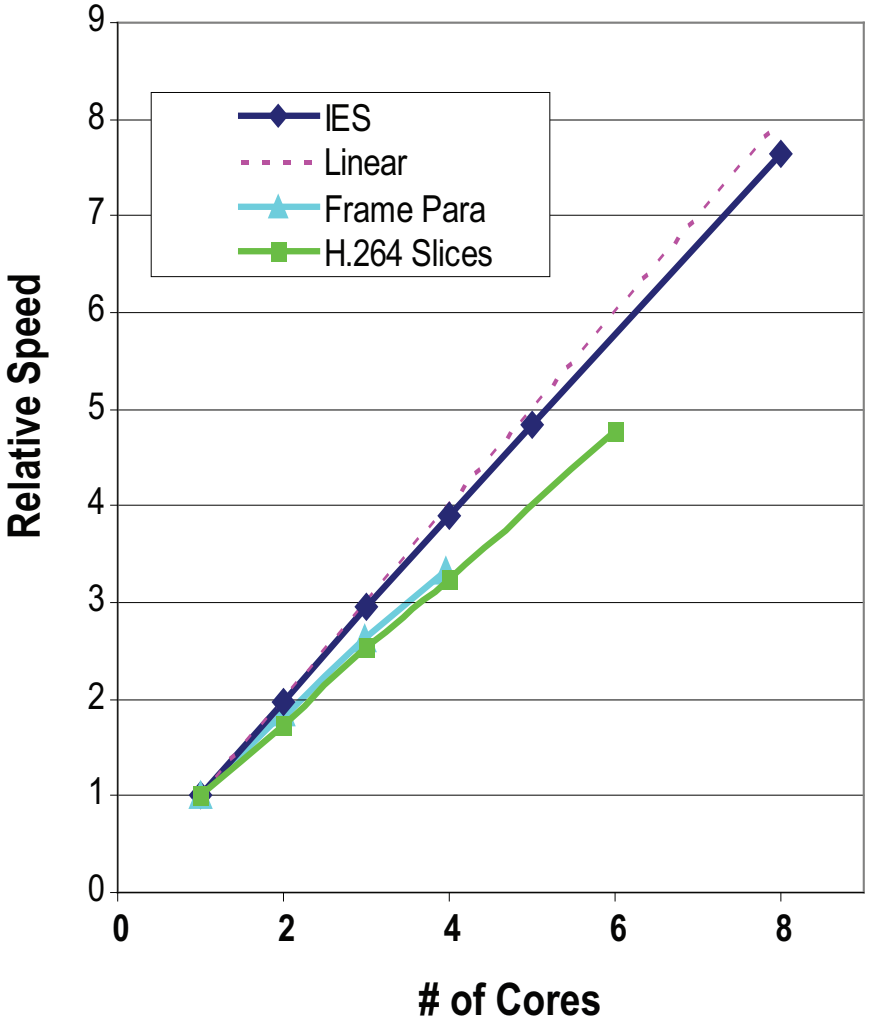


Figure 4-23: Three different multi-core architectures show nearly-linear performance gains. The multi-core performance of H.264 slices is slightly lower because of the extra processing required by the CAVLC and also the unbalanced slice workload due to uneven image characteristics across the slices.

CAVLC Coding Loss

The coding efficiency was quantified by the reference H.264 software using different slice configuration settings. The CAVLC coding loss was computed from the increase in compressed file size when the ENC breaks up the frames into slices. Frame multi-core does not suffer from any coding loss because the frames are not broken up into any slices.

Table 4.1: Video decoder multi-core ($N = 3$, 720p) comparison for different techniques relative to $N = 1$

Multi-Core Technique	*H.264 Slices*	*H.264 Frames*	*Interleaved Entropy Slices*
Thesis Section	4.1	4.2	4.4
Degree of Multi-Core	3	3	3
Relative Performance	2.51X	2.64X	2.91X
Equivalent Dynamic Power Savings	58%	59%	61%
CAVLC Coding Loss	0.41%	0%	0.05%
Relative Last-Line Size	3.00X	3.00X	1.03X
Relative Logic Area	3.00X	3.00X	3.00X
Input Buffering Latency (ms)	22	66	22
H.264 Compliance	Yes	Yes	No
Software Applicability	Yes	Yes	Yes

Relative Last-Line Size

The need for full-last-line caches (FLLCs) will be described in detail in Section 5.1. In Table 4.1, the size of the FLLCs is a direct multiple of the parallelism factor N for slice and frame multi-core, since the parallel DECs operate on independent areas of the video. For IES multi-core, the size of the FLLCs grows very slowly with N, as will be discussed in Section 5.2.

Relative Logic Area

The top-level logic required to integrate the N parallel DECs is much smaller than the logic within each of the DECs. Therefore, the total logic area increases almost linearly with N.

Input Buffering Latency

If the segmented scheme presented in Section 4.5 and Figure 4-21 is not used, the parallel DECs of this chapter suffer from input buffering latency. For slice and IES multi-core, this latency is $(N-1)$ slice periods, for a total latency given by the following equation.

$$InputLatency = 33ms \times (N-1)/N \tag{4.1}$$

For frame multi-core, the input latency is $(N-1)$ frame periods, for a total latency given by the following.

$$InputLatency = 33ms \times (N-1) \tag{4.2}$$

4.8 Summary

In this chapter, we presented several ways to enable multi-core decoding and provide a clear tradeoff between performance and area. If performance, power, area, coding efficiency and input latency are key concerns for the video decoder designer, we recommend choosing the proposed interleaved entropy slice (IES) architecture. In all of these metrics, IES processing provides comparable or better results relative to the other techniques, though it requires a slight change in the video standard.

If the decoder must remain H.264 compliant, then the choice is between the frame-level multi-core of Section 4.2 and the slice-level multi-core of Section 4.1. Frame multi-core outperforms slice multi-core in performance, power savings, and coding efficiency. However, slice multi-core has a lower input buffer latency, so it might be the better choice for applications such as video conferencing which have a hard limit on round-trip latency.

Note that frame multi-core is not mutually exclusive from either of the slice multi-core

techniques. For example, if we wish to have 9 parallel DECs to improve throughput, the input buffering latency for frame multi-core would be 264 ms. However, if the system cannot tolerate such a large input latency, we could combine frame multi-core with $N = 3$ and slice multi-core with $N = 3$, yielding similar performance to $N = 9$. In this case, the input latency would be (66 ms + 22 ms) = 88 ms.

Chapter 5

Memory Optimization

Video decoding requires a significant amount of memory activity, which can be broken down into the following categories:

- frame buffer (FB) writing
- FB reading
- reading and writing the last line of information
- reading from ROM tables for ED
- reading and writing from pipeline FIFOs between DEC units

The memory subsystem is critical for both performance and power. In general, on-chip memory accesses use less power and take less time than off-chip memory accesses. This is because on-chip caches are smaller than off-chip memories, and on-chip memory accesses avoid charging relatively long PCB traces. For on-chip memory, accesses to a smaller memory usually consume less power than those to a larger cache. In this chapter, we outline different techniques to help reduce the number of accesses or the size of the memory being accessed by the video decoder (DEC).

The ideas presented in Section 5.1 and Section 5.3 were developed in collaboration with Vivienne Sze. The initial concepts of Section 5.2 and Section 5.4 were identified independently, then were fleshed out together with Vivienne Sze.

5.1 Full-Last-Line Caching (FLLC)

The top-neighbor dependency shown in Figure 4-12 requires each MB to refer to the MBs in the last line above. The use of a full-last-line cache (FLLC) allows us to fetch this data from on-chip static random-access memorys (SRAMs) rather than getting the previously-processed data from a large off-chip memory. Only fully-processed pixels are stored in the off-chip memory.

Several independent on-chip caches can be used to store syntax elements or pixel data that have not been fully processed, as shown in Figure 5-1. This includes: the last four lines of pixels that are required by the DB, last line of pixels needed for INTRA prediction, INTRA prediction modes for each 4x4 in the last line, MVs for each 4x4 in the last line, total IDCT coefficient count for each 4x4 in the last line, and macroblock (MB) parameters for the last line of MBs. For 720p resolutions, the area cost of this technique is 138 kbits of on-chip SRAM [27], as shown in Table 5.1. For 1080p resolutions, the FLLC size increase to 207 kbits, which is obtained from $138kbits \times 1920/1280$.

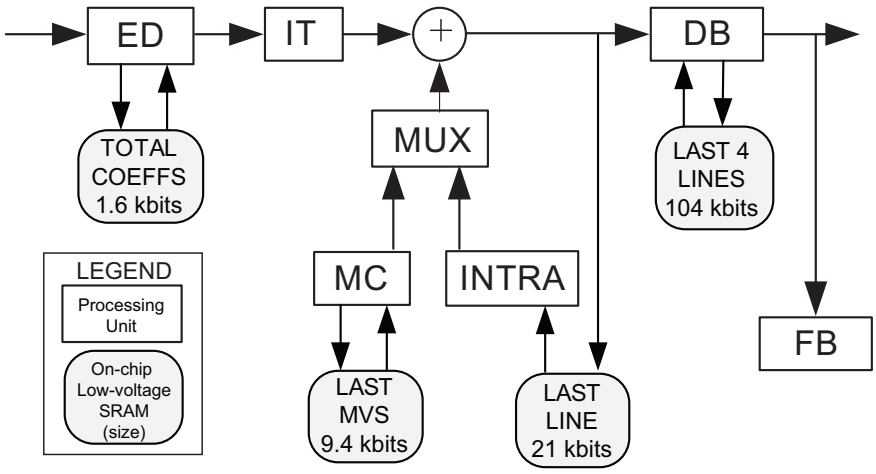


Figure 5-1: Full-last-line caches (FLLCs) reduce off-chip memory bandwidth (BW)

For a P-frame, this caching scheme reduces total off-chip BW by 26% relative to the case where no caches are used. The BW of each of the FLLCs is shown in Table 5.1. The FLLC is direct-mapped and does not need any tag bits since the address it caches is always implied to be from the last line. If the data and syntax elements for each MB are written to the

Table 5.1: Memory bandwidth (BW) of FLLCs for 720p at 30 fps

Cache	**Size [kb]**	**Dimensions addr x word**	**I-frame BW [Mbps]**	**P-frame BW [Mbps]**
Deblocking (Last 4 lines)	104	324x158 (luma)	510	510
		324x158 (chroma)	297	297
Intra prediction (Last line)	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**

FLLC, there is never the potential for a read miss.

5.2 Last-Line Caching for Interleaved Entropy Slices (IESs)

In addition to enabling parallel processing, the interleaved entropy slices (IESs) of Section 4.4 also allow for better memory efficiency than the raster-scan processing in H.264. This section shows how IES processing order can reduce accesses to the large full-last-line caches (FLLCs) discussed in Section 5.1. For example, when decoding B_i in Figure 5-2, the data from MBs $A_{i-1}, A_{i-2}, A_{i-3}$ can be kept in a much smaller cache since those MBs were recently processed by DEC_A and have high temporal locality.

The caches that pass data vertically between decoders, such as DEC_A to DEC_B in Figure 5-2, are implemented as FIFOs. A deeper FIFO could better handle workload variation between the IESs by allowing DEC_A to advance several MBs ahead of DEC_B and thus reduce stall cycles and increase throughput. The caches that pass data horizontally within each decoder only need to hold the information for 1 MB, and are unchanged from the H.264 raster-scan implementation. However, when we process A_i, the FLLC of Section 5.1 is still needed to hold the data that is passed from DEC_C to DEC_A, since DEC_C writes this data

long before DEC_A can read it. The depth of the FLLC FIFO should therefore be about as large as the frame width in order to prevent deadlock. The caching of data for IES processing is similar to the one used in the encoder of [15].

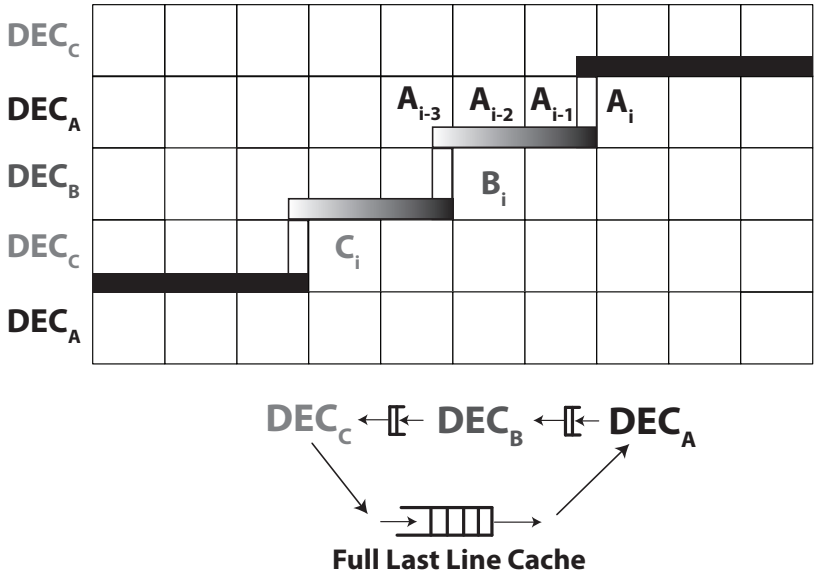


Figure 5-2: Caches used for interleaved entropy slice (IES) processing with 3 video decoders (DECs)

To evaluate the performance impact of sizing the FIFOs of Figure 5-2, we implemented the IES caches in Verilog and placed them together with the system of Section 4.4. When simulating INTRA frames for $N = 3$, we found that a FIFO depth of four 4x4 edges (one MB edge) only has a 3% performance penalty, whereas a minimally-sized FIFO reduces system performance by almost 25%. This trade-off is illustrated in Figure 5-3.

The FLLC FIFO is read by DEC_0 and written to by DEC_{N-1}, so if a single-ported memory is used, the accesses will need to be shared. The total size of the IES inter-slice caches is independent to first order of the degree of parallelism N, as the FLLC is not replicated with each DEC. This implies that the total area overhead of DEC parallelism with diagonal processing is not a factor of N, as was the case for the parallelism techniques in Section 4.1 and Section 4.2. As will be shown in Section 6.7, the area of the FLLC SRAMs can be 3 times larger than the rest of the DEC logic. As a result, for $N = 3$, the area increase due to parallelism would be about 50% and not 200%.

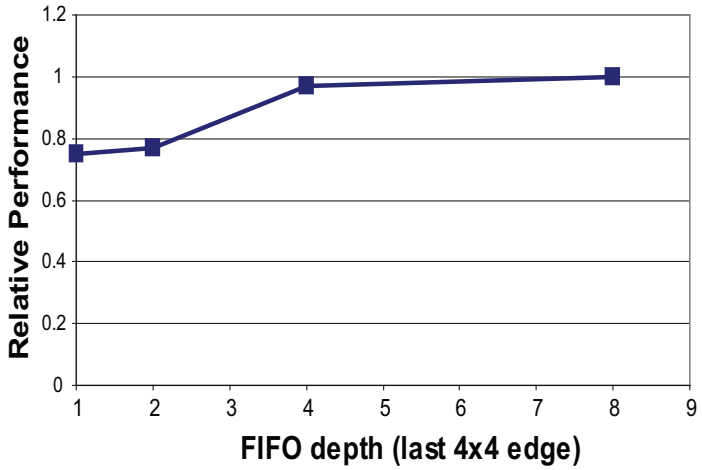


Figure 5-3: Impact of FIFO sizing on parallel interleaved entropy slice (IES) performance

If N is the number of parallel IES DECs, the number of accesses to the large FLLCs are reduced to $1/N$ of the original. These accesses are replaced with accesses to much smaller FIFOs that hold the information for about 1 MB. This uses much less energy than accessing a large memory that stores 80 MBs for 720p, or 120 MBs for 1080p. This reduction in FLLC accesses allows the designer to even eliminate the area-hungry FLLCs and just use the large off-chip memory where the FB is stored to keep the last-line information.

It is interesting to note that diagonal processing can reduce FLLC accesses even when only one DEC is used (no DEC replication). This would require the single DEC to alternate between different IESs whenever one of the FIFOs in Figure 5-2 stalls.

The small IES FIFOs are only 1MB deep, so each FIFO only needs to be 1/80 of the total FLLC size of Table 5.1, for a total size in kB of $(N-1) \times 138/80/8$. For really small caches (below 1kB), the memory storage can be implemented efficiently in flip-flops or latches. For the IES FIFOs of Section 5.2, the energy savings due to replacing FLLC accesses with FIFO accesses can be computed using the following formula.

$$E_{cache} = (N-1)/N \times E_{FIFO} + E_{FLLC}/N \tag{5.1}$$

The power savings relative to having no FIFO caches are computed as follows.

$$\%_{saved} = 100 \times (E_{FLLC} - E_{cache})/E_{FLLC} \tag{5.2}$$

Based on the memory size, the IES FIFOs are implemented using Flip-Flops.

5.3 Motion Compensation (MC) Caching for H.264

The off-chip frame buffer (OCFB) used in the system implementation of Chapter 6 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 OCFB.

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 MB) have MVs with identical integer components [9]. As discussed in Section 3.1, 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. If we predict 4 neighboring 4x4 blocks with identical motion vectors, we need to read 4x9x9 (324) pixels from the OCFB, as shown on the left side of Figure 5-4. However, the four 9x9 areas overlap significantly, so we should only have to read an area of 13x13 (169 pixels) from the OCFB, as shown on the right side of Figure 5-4.

If two parallel MC interpolators are used, as shown in Chapter 3, they can be 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 Figure 3-4) are avoided. Alternatively, a more general caching scheme can be used to further reduce redundant reads if it takes into account:

1. adjacent 4x4 blocks with slightly different motion vectors

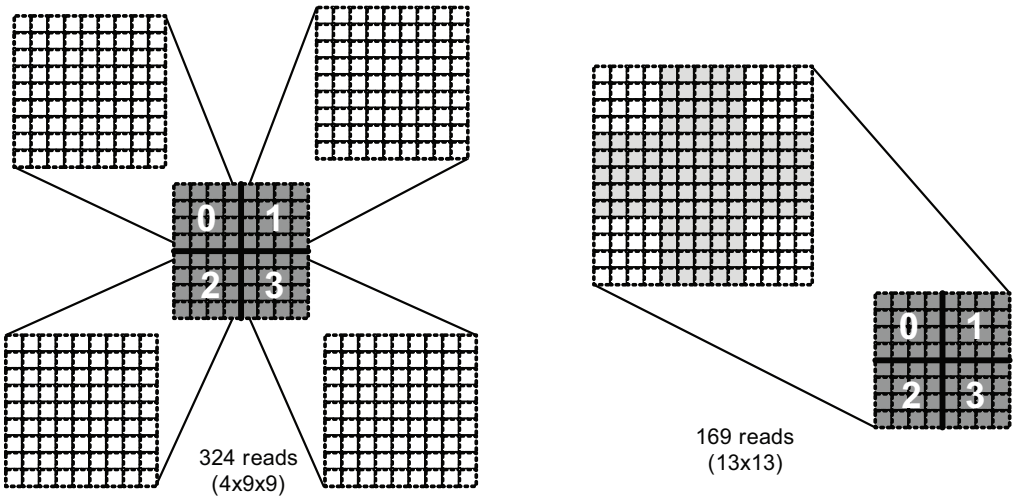
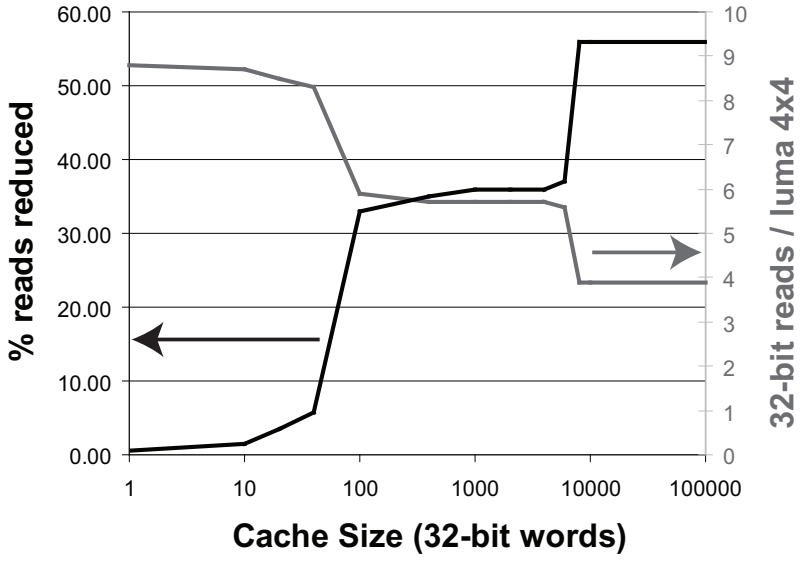


Figure 5-4: Eliminating motion compensation (MC) redundant reads

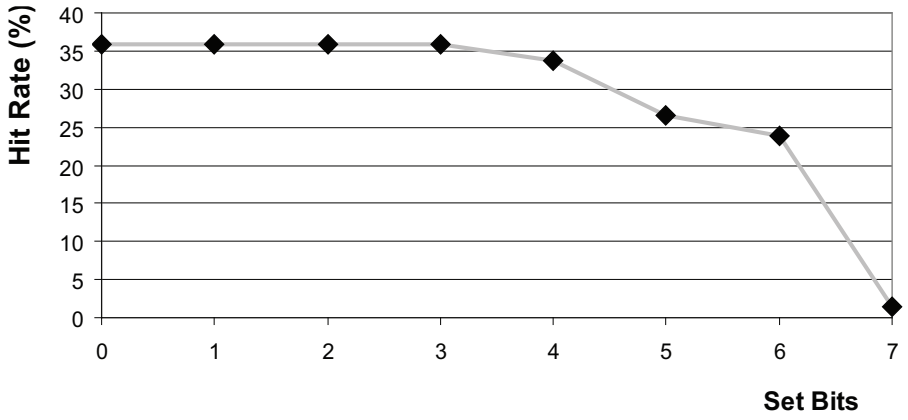
2. overlap in read areas between nearby macroblocks on the same macroblock line

3. overlap in read areas between nearby macroblocks on two consecutive macroblock lines

The potential benefits of this scheme can be evaluated with the help of a variable-sized fully-associative on-chip cache, as shown in Figure 5-5a. A small cache of 512-Bytes (128 addresses) can help reduce the off-chip read BW by a further 33% relative to the caching done between two parallel MC interpolators. This is achieved by taking advantage of the first two types of redundancies in the above list. In order to take advantage of the last redundancy in the list, a much larger cache is needed (32 kBytes) to achieve a read BW reduction of 56% relative to the caching of 2 parallel MC interpolators This larger MC cache achieves close to no repeated reads, as the average number of luma reads per 4x4 is about four 4-pixel words, or 16 pixels, as shown on the right axis of Figure 5-5a. The associativity of this cache also impacts the number of reads, due to the hit rate. As Figure 5-5b shows, a fully associative 512-Byte cache (0 set bits) provides the largest hit rate, while a direct-mapped scheme (7 set bits) has the lowest hit rate. The benefits of this MC cache must be weighed against the area overhead of data and address tags and the energy required to perform cache reads and writes.



(a) Effect of motion compensation cache size of read reductions



(b) Off-chip bandwidth (BW) reduction versus associativity

Figure 5-5: Motion compensation (MC) cache

5.4 Motion Compensation (MC) Caching for Interleaved Entropy Slices (IESs)

The MC cache described in Section 5.3 can have a higher hit-rate if a diagonal MB ordering is used. This is because the read area of the MBs above the current MB could fit inside a much smaller cache. The hit-rate of this type of MC cache was simulated for varying

cache sizes and degree of IES parallelism. We found that a moderately sized cache of 2kB reduces the OCFB read BW by 67%. This hit-rate was simulated to be 5% larger than for an equally-sized MC cache of a DEC that uses regular raster-scan MB ordering.

For $N = 3$, the IES MC cache hit rate plateaus when the size reaches around 2kB. For memory sizes between kBs and a few hundred kBs, SRAMs are a suitable choice. This grows about linearly with N, as more MBs are processed on a diagonal. A moderately sized MC cache for IES processing can eliminate a large fraction of the redundant MC reads, but will not cover the vertical overlap between MB *rows 0 and (N-1)* of parallel IES processing. The MC IES cache hit-rate will go up slowly with increasing N, and eliminate most overlaps between MB *rows 0 to (N-1)*. For IES MC caching, the average energy for a read is computed by the following formula, where EWR_{MC}, $ERDHIT_{MC}$ and $ERDMISS_{MC}$ are the write, read hit and read miss energies of the MC cache.

$$\begin{aligned} ERD_{cache} =& EWR_{MC} + HR \times ERDHIT_{MC} + \\ & (1 - HR) \times (ERD_{OCFB} + ERDMISS_{MC}) \end{aligned} \tag{5.3}$$

The power savings relative to having no MC cache are computed as follows.

$$\%_{saved} = 100 \times (ERD_{OCFB} - ERD_{cache})/ERD_{OCFB} \tag{5.4}$$

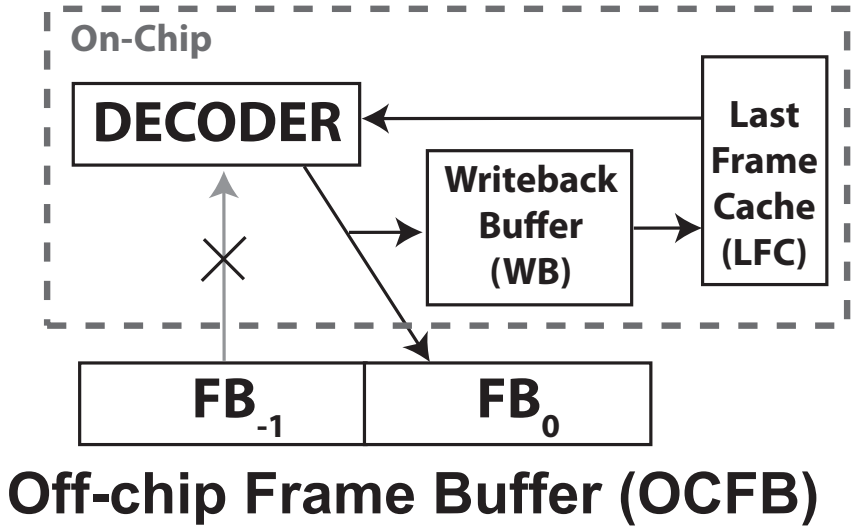
Based on the memory size, the MC cache is implemented as SRAM.

5.5 Last-Frame Cache (LFC) for Motion Compensation

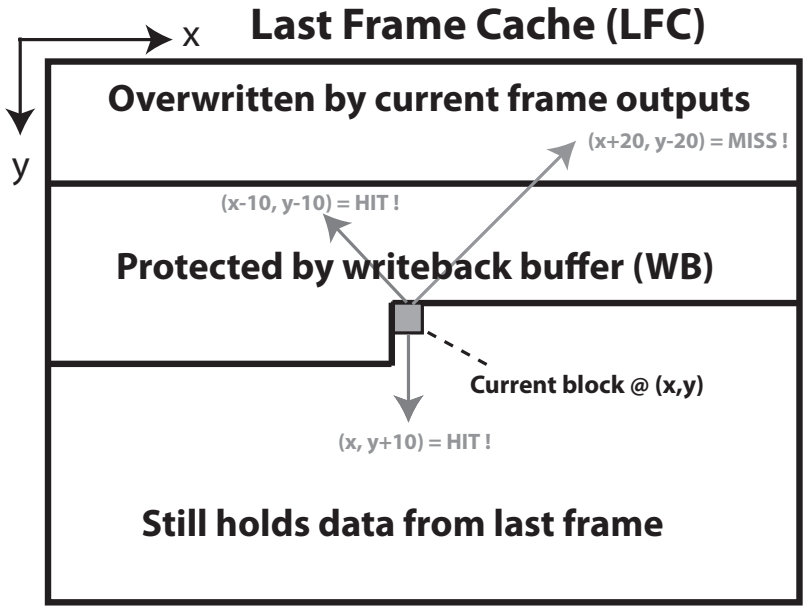
During motion compensation (MC), most of the pixels are read from the previous frame (FB_{-1}), as opposed to being read from even earlier frames (FB_{-2}, FB_{-3}, etc.). If we can store the last reference frame in an on-chip LFC, we can avoid going off-chip for the majority of MC reads. This caching architecture is described in Figure 5-6a, which shows how reads from FB_{-1} (the previously-decoded frame) are replaced with reads from the LFC. This

scheme requires a write-back buffer (WB) in order to not overwrite the data at the current location in the LFC, which is needed for MC.

To understand the need for a WB, let us assume that there is no WB and the MV for the current block at location (x, y) is $(-10, -10)$. In this case, the data from the last frame at location $(x-10, y-10)$ would no longer be found in the LFC, since it would have already been overwritten by the block at location $(x-10, y-10)$ from the current frame.



(a) Architecture



(b) Illustration of hits and misses in the LFC

Figure 5-6: Last-Frame Cache (LFC)

One overhead of this LFC scheme is the significant additional area of the WB and the LFC. There is also a power overhead, since each decoded pixel is now written to the LFC (as well as to the OCFB), written to and read from the WB, all of this just to avoid reading it back from the OCFB.

For 720p resolutions, the size of the LFC would be 1.4 MBytes, with an area of $2.7mm^2$ if implemented with high-density eDRAM [38]. The size of the WB depends on how many misses we are willing to tolerate in the LFC. Figure 5-7 shows how the hit rate of the LFC varies with the size of the WB. If there is no WB, the videos with more movement from left to right or up to down will have more LFC misses. For example, for the "shields" video, the LFC hitrate with no WB is 65% because the movement is from left to right. A small WB with the size of 1 MB can improve the hit-rate up to about 93%. To eliminate the remaining misses, the entire MB above must be buffered by the WB, which explains the last jump up to 100% when the WB size is 80 MBs. A miss occurs in the LFC when the block being fetched has a much smaller y-coordinate and/or x-coordinate than the current block being processed, as shown in Figure 5-6b. This type of miss happens when the MC data was already overwritten by a recently decoded block which was evicted from the WB cache and spilled into the LFC. If this happens, the data must be fetched from FB_{-1}.

If the reference frame is not the last frame, the LFC is also bypassed and the data is fetched from the OCFB. The frequency of this occurence depends on the choices made at the ENC. The work in [39] shows that the previous frame is chosen 80% of the time as the reference frame, as averaged over 10 different videos of CIF resolution. The ENC can choose to limit the search range to only the last frame, in order to reduce the search time, but this comes at a cost of decreased coding efficiency, since a less optimal prediction will be found in some cases.

The WB cache is a simple window buffer which is written to and read from in the same order that the pixels are decoded. There are no misses associated with the WB cache, since the data is read and written in a deterministic order. The LFC cache also stores data at deterministic addresses, since it has exactly the same form as a frame in the frame buffer. There is no need to store tag information in the LFC cache, since we can implicitly derive

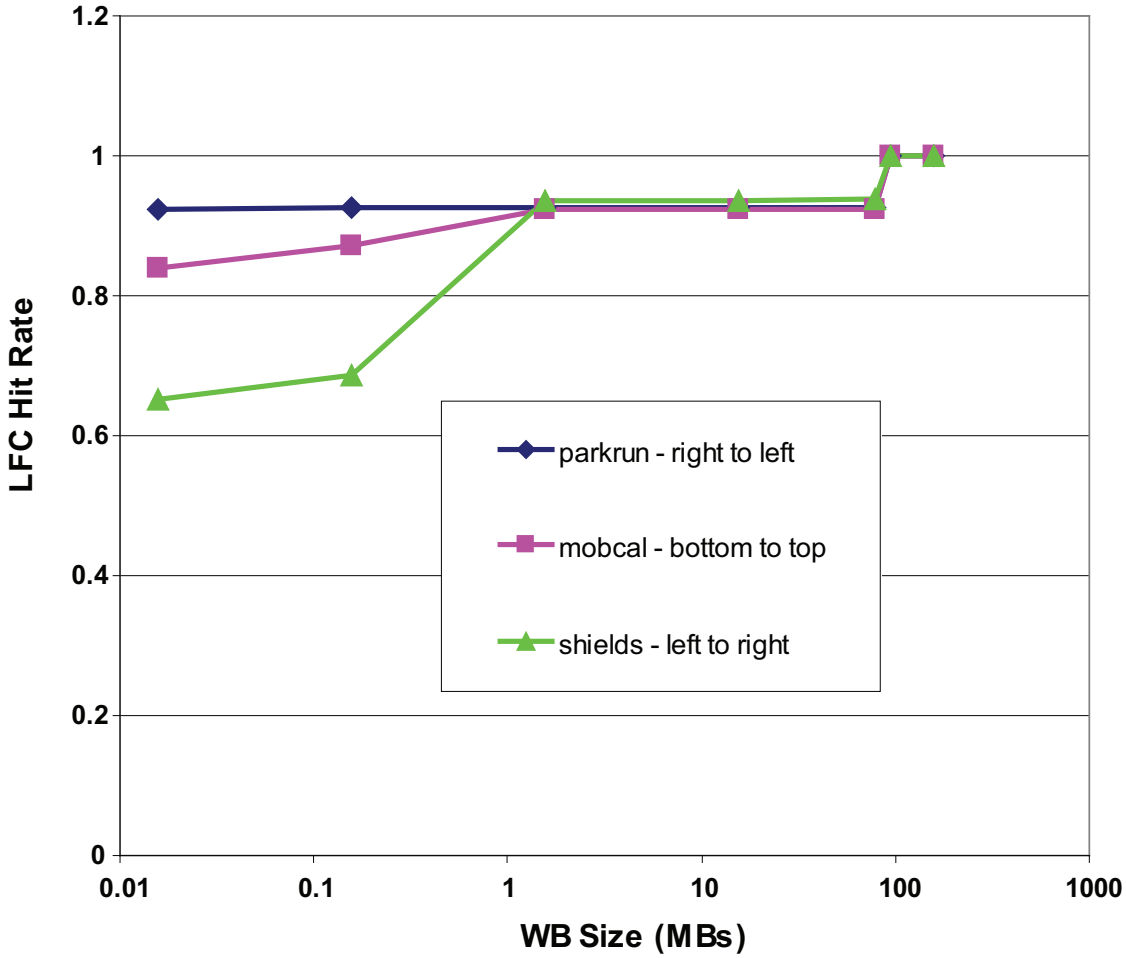


Figure 5-7: Hit rate of last-frame cache versus size of writeback cache for different 720p videos. For each video, the type of motion is described, in order to help explain the differences in hit rates.

this value. If we wish to read from a given address in the LFC, we first need to determine whether the data at that location is from the previous frame (cache hit) or the current frame (cache miss). To determine between a cache hit or miss, we simply need to look at the pointer that copies data from the WB into the LFC. If this pointer is smaller than the address we wish to read (modulo the size of the frame), the data from the previous frame is still in the LFC. If the pointer is larger than the address of the area we wish to MC predict from, we recognize this as a cache miss and fetch the data from the OCFB instead.

To measure the impact of the LFC, we can compute the different trade-offs of this tech-

nique. If a LFC with a 32-line WB is used, the size of the cache for 720p is (720+32) pixel lines, or (1080+32) lines for 1080p. Since eDRAM offers the highest area density, it is the most suitable for the large LFC cache, and might even fit the entire frame buffer (FB). The LFC cache can reduce up to 100% of the MC reads, if the WB is large enough and the reference frame is always the previously decoded frame. For the LFC, the energy of the cache was estimated by using the following formula, where HR refers to the cache hit-rate, ERD refers to read energy, and EWR refers to write energy.

$$\begin{aligned} ERD_{cache} =& EWR_{LFC} + HR \times ERD_{LFC} + (1 - HR) \times ERD_{OCFB} + \\ & EWR_{WB} + ERD_{WB} \end{aligned} \tag{5.5}$$

This is because LFC caching needs to write the data temporarily to the WB, and then transfer it from the WB to the LFC. In case of a hit, the data is read from the LFC, otherwise the data is read in from the OCFB. The power savings are computed using the following formula.

$$\%_{saved} = 100 \times (ERD_{OCFB} - ERD_{cache})/ERD_{OCFB} \tag{5.6}$$

Based on the memory sizes, the WB is implemented as SRAM, while the LFC uses eDRAM.

5.6 Motion Compensation Data-Forwarding Caches

If we allow N parallel DECs to operate concurrently on N consecutive frames, as in Section 4.2, we can forward the motion compensation (MC) data between them using on-chip data-forwarding caches (DFCs), as shown in Figure 5-8. This will avoid most off-chip MC reads for all DECs but DEC_0. For example, if $N = 3$, the DFCs can reduce the off-chip read BW by up to 67% or $(N - 1)/N$.

In general, DEC_i and DEC_{i-1} need to be synchronized, such that DEC_i lags sufficiently behind DEC_{i-1}, similar to the discussion in Section 4.2. Conversely, if DEC_{i-1} gets too far ahead of DEC_i, the temporal locality is lost, and DEC_i will read the MC data from the

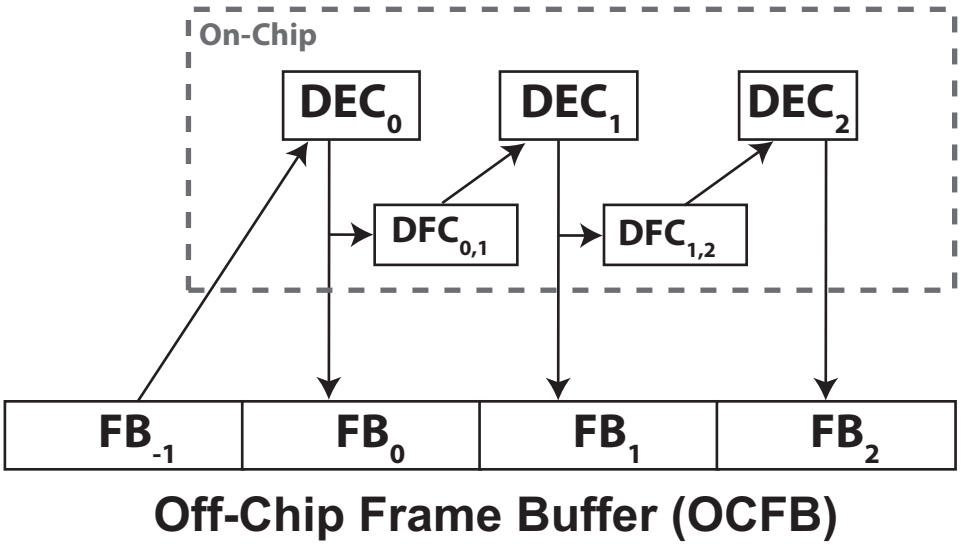


Figure 5-8: Motion compensation (MC) data-forwarding caches (DFCs) for $N = 3$

OCFB instead of from $DFC_{i-1,i}$. In that case, we can stall DEC_{i-1} in order to maximize the hit-rate of the DFCs. These two constraints can be handled with the help of low and high watermarks, as illustrated in Figure 5-9. A top-level controller is used to make sure $Dist_{0,1}$ remains between $WM_{lo-0,1}$ and $WM_{hi-0,1}$ and similarly that $Dist_{1,2}$ remains between $WM_{lo-1,2}$ and $WM_{hi-1,2}$. For example, if DEC_0 runs much faster than DEC_1, $Dist_{0,1}$ will eventually hit the watermark $WM_{hi-0,1}$ and DEC_0 will be stalled. Alternatively, if DEC_1 runs faster than DEC_0, $Dist_{0,1}$ will reach the value $WM_{lo-0,1}$ and DEC_1 will have to be stalled.

In order to evaluate the performance impact and hit-rate of these DFCs, we implemented the DFCs in Verilog and placed them between the DECs described in Section 4.2. The performance impact of stalling at these watermarks was simulated for a "mobcal" video sequence of 100 frames. The overall loss in throughput for $N = 3$ was less than 8%.

The DFCs need to store about 32-64 lines of pixels to minimize the cache miss rate, so their on-chip area can be quite large for high-resolution, highly-parallel DECs. To understand the trade-off between the size of the DFCs and the hit rate, we simulated the DFC system for 100 frames of the "mobcal" video. The result is shown in Figure 5-10. As expected, a really large cache will have near 100% hit rate, leading to 67% reduction in off-chip MC

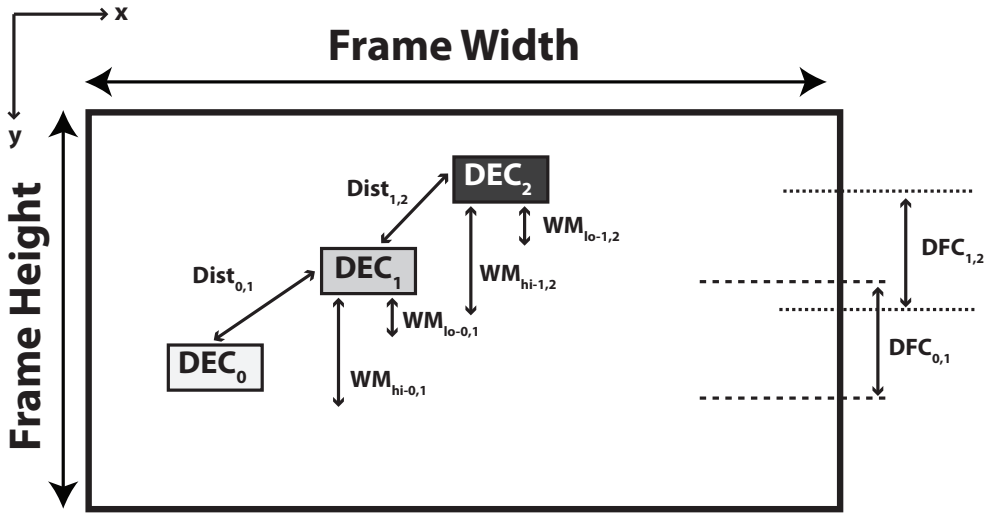


Figure 5-9: High and low watermarks for 3 DECs to maximize DFC hit-rate

reads for $N = 3$. The hit rate drops off significantly for DFC sizes of less than 32 lines, since the vertical MVs can easily fall outside this range. For $N = 3$ and 720p resolution, the total area of the two 64-line DFCs is about $1mm^2$, assuming high-density 65nm SRAMs.

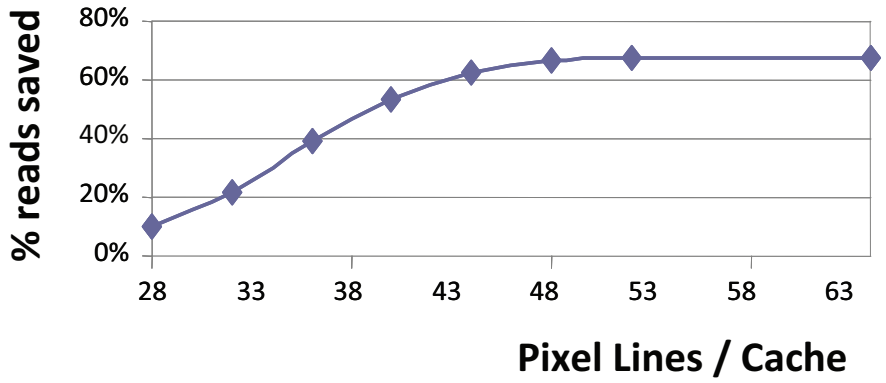


Figure 5-10: Reduction in off-chip reads versus size of motion compensation (MC) data-forwarding cache (DFC) for $N = 3$.

The hit rate of the cache is also dependent on how often the reference frame is chosen to be the last frame. For the simulations of Figure 5-10, a single reference frame was assumed. However, typical ENCs will use multiple reference frames in order to improve coding efficiency, so the maximum hit-rate of the DFCs can decrease to about 80%, assuming the statistics of [39].

A DFC cache with near-maximum hit-rate can be implemented with 48 pixel lines per DFC, for a total of $48 \times (N-1)$ pixel lines. If the DFC hit-rate can be maximized by synchronizing the parallel DECs and the MV variation is not large, the DFCs can eliminate $(N-1)/N$ of the MC reads, since all DEC but the first one will read from a DFC. For each of the $(N-1)$ DFCs, the energy per access is computed using the following formula.

$$ERD_{cache} = EWR_{DFC} + HR \times ERD_{DFC} + (1 - HR) \times ERD_{OCFB} \tag{5.7}$$

The average energy used for all MC reads is then given by the following equation, since the first DEC always reads from the OCFB.

$$ERD_{avg} = ERD_{cache} \times (N-1)/N + ERD_{OCFB}/N \tag{5.8}$$

The power savings relative to no DFCs are computed as follows.

$$\%_{saved} = 100 \times (ERD_{OCFB} - ERD_{avg})/ERD_{OCFB} \tag{5.9}$$

5.7 Software Applicability of Memory Optimization

This section discusses which of the on-chip caching techniques introduced previously are useful for a software implementation on a parallel processor machine similar to that of Figure 4-22.

FLLC cache accesses would also be reduced for a software IES DEC implementation (parallel or serial), as temporal locality would be better exploited than in the case of using the traditional raster-scan processing order. This is similar to the argument that was made in Section 5.2.

The DFC caching technique described in Section 5.6 also exploits the temporal locality in the processor's cache for a software DEC implementation. In this case, an increase in processor cache hit rate can be achieved for either a single or multi-core processor. For the case of a single-core processor, each of the parallel DECs can be assigned to a different

thread. Since the N threads must run on the same processor, they must be scheduled such that the DECs are properly synchronized and the cache hit rate is maximized. For the case of a multi-core processor, the DECs running on different processors should also be synchronized in the same way to obtain the same benefits.

5.8 Caching Summary

Several on-chip caching techniques were introduced that significantly reduce the off-chip memory BW requirements. The different caching ideas were implemented, verified, and benchmarked in Verilog. They are summarized and compared in Table 5.2. The first three techniques reduce OCFB BW by using large on-chip caches. The fourth technique takes advantage of interleaved entropy slice (IES) processing to provide better data locality and thus minimize accesses to the full-last-line cache (FLLC).

To calculate the exact energy savings based on the different cache hit rates, we simulated and estimated the energies for the different types of memories involved. The normalized energy per bit for each of the types of memories are as follows.

- *1* for a FIFO flip-flop, estimated using the synthesis libraries
- *19* for a large eDRAM, estimated using the numbers of [38]
- *51* for a large SRAM, estimated from the designs used in [12]
- *672* for an off-chip DRAM, assuming a 10pF pin capacitance and Micron's mobile SDRAM [40]

5.9 Summary

In this chapter, we described several memory optimizations that reduce the off-chip memory bandwidth and lead to overal power savings in a video decoder. If a very large eDRAM (1 MByte) can be fit on chip, the LFC technique described in Section 5.5 can save 60% of the MC memory read power relative to always doing off-chip reads. Using smaller caches

Table 5.2: Summary of different DEC caching techniques for 720p

Caching Technique	*LFC with 32-line WB*	*48-line DFCs N=3*	*MC cache for IESs N=3*	*1-MB FIFOs for IESs N=3*
Thesis Section	5.5	5.6	5.4	5.2
Cache Size (kB)	963	123	2	0.43
Cache Type	eDRAM	SRAM	SRAM	FIFO Flip-Flops
OCFB MC BW Reduction	80%	53%	67%	0%
FLLC BW Reduction	0%	0%	0%	67%
Memory Access Power Savings	60%	44%	37%	65%
H.264 Compliance	Yes	Yes	No	No
Software Applicability	Yes	Yes	Yes	Yes

(0.1 MByte) together with the frame parallelism of Section 4.2, the use of DFCs can lower MC memory read power by 53% relative to having no caches. A relatively smaller cache (2 kByte) can eliminate a lot of redundant reads for the MC interpolation of neighboring blocks, and thus reduce the MC read power by 37% relative to using no MC cache at all. Finally, accesses to the FLLC SRAM can also be reduced using the IES processing scheme of Section 4.4, saving about 65% of the FLLC access power relative to the case where each slice reads and writes to a large FLLC.

Some of these memory optimizations can be combined to yield further power savings, depending on the ratio of access energies between the different caches. For example, the MC cache of Section 5.3 could be placed between the large LFC cache and the decoder. This would replace some of the LFC reads with reads from the smaller MC cache. This would

only save power if the cost of writing and reading from the smaller MC cache is less than the power to read from the larger LFC cache.

Chapter 6

Prototype Video Decoder ASIC

This chapter describes the application-specific integrated circuit (ASIC) implementation of a video decoder (DEC), including the architecture, test setup, chip results, and chip statistics. Three graduate students were involved in the design of the H.264/AVC decoder. The main architects of the decoder were myself and Vivienne Sze. I was the lead designer of the IT, INTRA, ED and MEM units, while Vivienne was the lead designer of the DB unit. The design of the MC unit and the decoder pipeline architecture as well as the backend and testing of the chip were a joint effort. The MC interpolators were implemented by Vivienne Sze and I implemented the top-level parallel MC unit, the decoder pipeline, and the RTL for the test harness on the FPGA described in Section 6.4. The low-voltage SRAMs were designed by Mahmut Ersin Sinangil, and Vivienne and I helped integrate them into the rest of the DEC. Portions from this chapter, particularly Section 6.2, Section 6.3, Section 6.5, Section 6.6 and Section 6.7 appear in [27].

6.1 Video Decoder ASIC Architecture

The ASIC uses many of the pipelining techniques described in Chapter 2. The video decoder (DEC) architecture is shown in Figure 6-1. For this ASIC, the different 4x4 FIFO depths were chosen mostly to minimize chip area, so some performance was traded off as explained in Section 2.2. The MC interpolator pipeline used is the same as the one described in Section

3.1. The ASIC is fully compatible with the H.264 baseline profile standard.

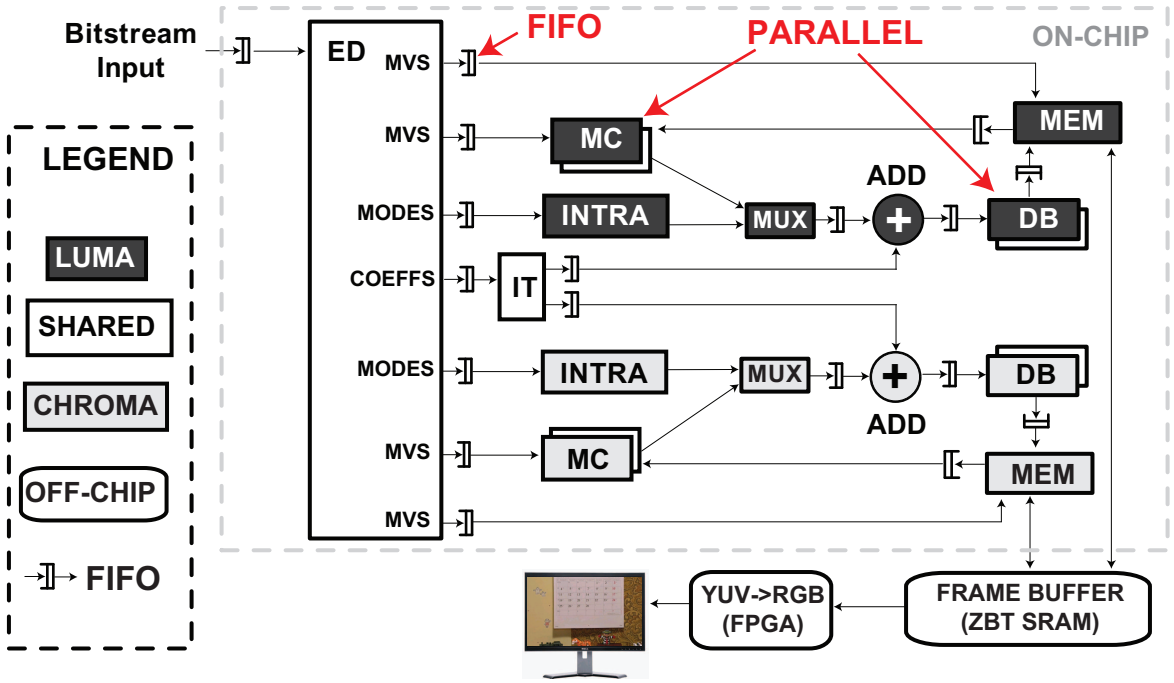


Figure 6-1: H.264 ASIC decoder architecture

The ASIC also uses parallelism within the decoder units whenever possible, as described in Chapter 2. The number of cycles required to process each 4x4 *luma* block varies for each unit as shown in Table 6.2. The cycles consumed for each 4x4 *chroma* block are also shown in Table 6.2. 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 MC interpolator was replicated by 2, as described in Section 3.2, so the average 4x4 block could be predicted in 2.3 cycles. The DB unit uses 4 different filters, as described in Section 2.5, so on average a 4x4 block is filtered every 2.9 cycles. The IT unit has 8 different 1-dimensional transform blocks, so a 4x4 block can be inverse transformed in one cycle, as described in Section 2.4. The reconstruction unit was parallelized by a factor of 16, so that a 4x4 residual and predicted block could be added in one cycle, as described in Section 2.8. Two different off-chip memories were used, one for chroma and one for luma, in order to enable more parallelism in the MEM unit, as was discussed in Section 2.9.

Table 6.1: FIFO sizes between different pipeline units

Source Unit	**Sink Unit**	**FIFO Depth**	**FIFO Width**	**Total Bits**	**Synch. FIFO ?**	**FIFO** element **description**
ED	IT	1	144	144	yes	16 luma coeffs
ED	IT	32	1	32	yes	any luma coeffs?
ED	IT	8	1	8	yes	any chroma coeffs?
IT	ADD	1	1	1	yes	no luma residual?
IT	ADD	8	1	8	yes	no chroma residual?
ADD	DB	1	1	1	yes	no luma residual?
IT	DB	4	4	16	yes	no chroma0 residual?
IT	DB	4	4	16	yes	no chroma1 residual?
IT	ADD	1	160	160	yes	16 luma residuals
IT	ADD	1	160	160	yes	16 chroma residuals
INTRA/MC_0	ADD	1	128	128	yes	luma prediction
MC_1	ADD	1	128	128	yes	luma prediction
MC	ADD	1	128	128	yes	chroma prediction
ADD	DB	1	128	128	yes	reconstructed luma
ADD	DB	1	128	128	yes	reconstructed chroma
ED	All	2	30	60	yes	MB parameters
ED	INTRA	16	4	64	yes	prediction modes
MEM	MC_0	4	72	288	no	luma from OCFB
MEM	MC_1	4	72	288	no	luma from OCFB
MEM	MC	4	72	288	no	chroma from OCFB
DB	MEM	1	151	151	no	16 luma outputs
DB	MEM	1	151	151	no	16 chroma outputs
ED	MEM	32	26	832	no	motion vectors
ED	MEM	8	3	24	no	reference indices

This ASIC did not use any of the multi-core parallelism techniques described in Chapter 4, but the ASIC RTL was used to evaluate those ideas. The ASIC RTL required some changes in order to implement those top-level parallelism ideas, some of which were not H.264 compliant.

Two key techniques from Chapter 5 are used to reduce the ASIC off-chip frame buffer (OCFB) memory BW. Making use of the full-last-line caches (FLLCs) discussed in Section 5.1 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 method used by the ASIC, discussed in Section 5.3, reduces the number of reads by the MC unit by reducing some of the horizontal and

Table 6.2: Cycles per 4x4 block for each unit in P-frame pipeline of Figure 1-2, assuming no stalling taken for 300 frames of the "mobcal" sequence. Each 4x4 block include a single 4x4 luma block and two 2x2 chroma blocks. [ ] is performance after Chapter 2 parallelism 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

vertical redundancies. The impact of the two approaches on the overall OCFB BW can be seen in Figure 6-2. The overall OCFB BW is reduced to 1.25 Gbps.

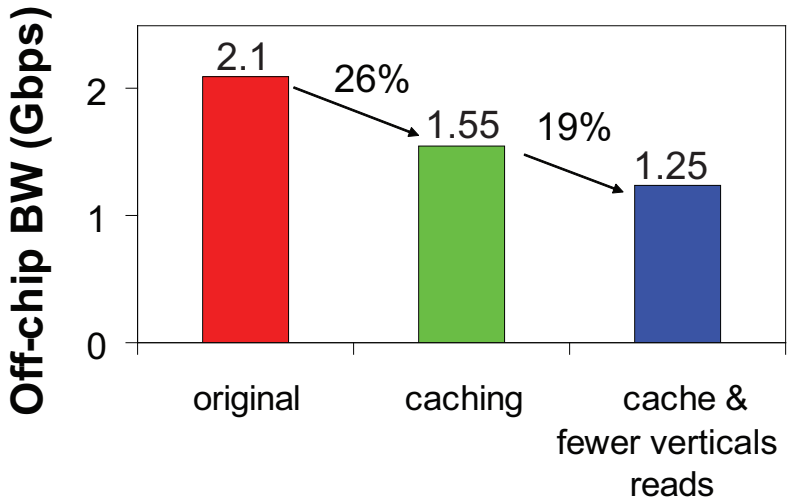


Figure 6-2: Reduction in overall memory bandwidth from caching and reuse MC data

6.2 Multiple Voltage and 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 6.2. 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 decoder architecture can be partitioned into multiple frequency and voltage domains.

Partitioning the decoder into two domains (MEM in the *memory controller* domain and the other processing units in the *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's limited parallelism.

The two domains are completely independent, and separated by asynchronous FIFOs as shown in Figure 6-3. Voltage level-shifters (using differential cascode voltage switch logic) are used for signals going from a low to a high voltage. The asynchronous FIFO shown in Figure 6-3 moves data from the core domain to the memory controller domain. This is similar to the multi-domain technique that was used in [41] and [42]. The FIFO contents are split across the two different clock/voltage domains. The general rule is to place all registers on the voltage domain corresponding to the clock of the register. This way, the clock tree is only contained to one voltage domain, making timing closure much simpler. For example, if the memory array registers of Figure 6-3 were instead placed on the memory controller domain, the clock CLK_{slow} would have to be routed using the memory controller voltage. Since the memory controller voltage can change independently of the core voltage, there is no way to guarantee that all the CLK_{slow} clock tree leaves could be de-skewed, and timing failures for the $data_{slow}$ signal would be unavoidable.

From Table 6.2, we can conclude that there could be a 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.

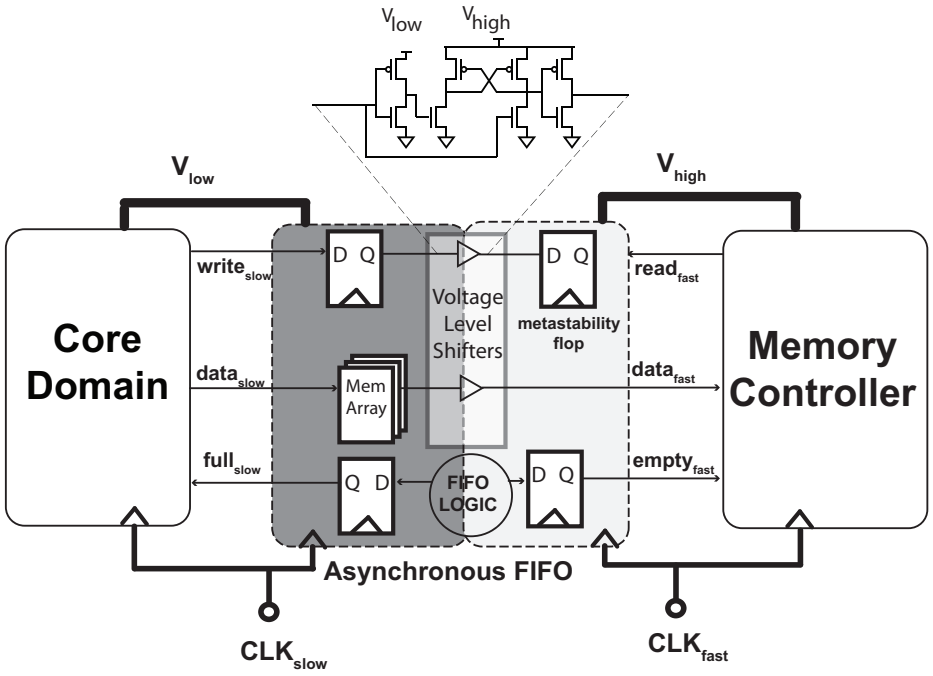


Figure 6-3: Independent voltage/frequency domains are separated by asynchronous FIFOs and level-converters

Table 6.3 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 6.2 and assume no stalls. For a single domain design the voltage and frequency must be set to 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. The additional power savings for moving to three domains is less significant since the impact of frequency reduction on voltage scaling is reduced as the operating point is nearing the threshold voltage. Therefore, a two-domain design was used for this ASIC.

Table 6.3: Estimated impact of multiple domains on power for decoding a P-frame.

	Frequency Ratio			**Voltage [V]**			**Power**
Domains	**ED**	**Core**	**MEM**	**ED**	**Core**	**MEM**	**[%]**
One	1.00	1.00	1.00	0.81	0.81	0.81	100
Two	0.26	0.26	1.00	0.66	0.66	0.78	75
Three	0.26	0.18	1.00	0.66	0.63	0.78	71

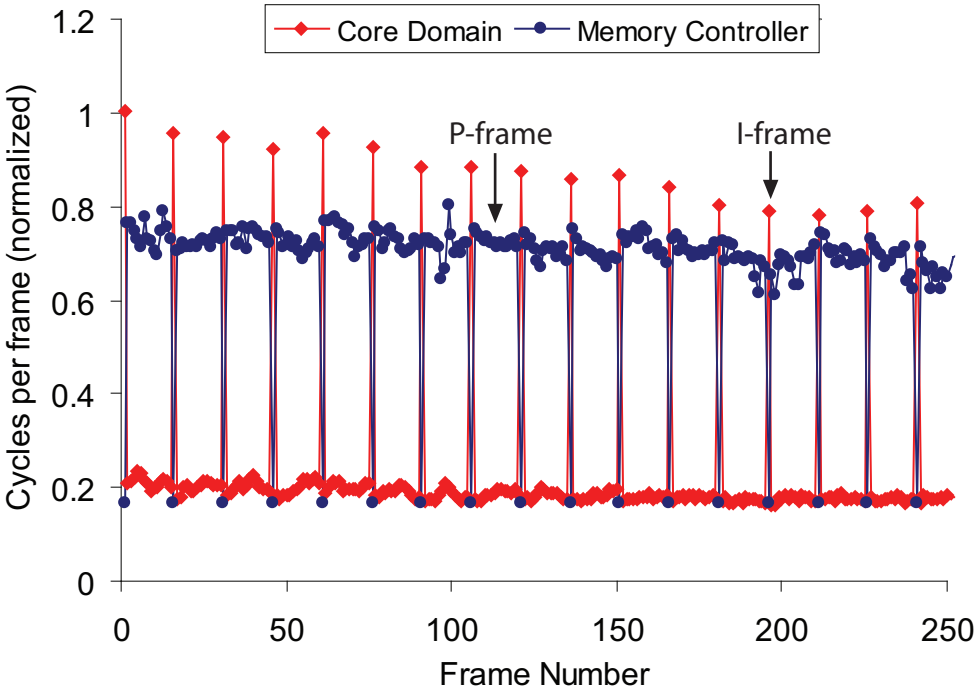
6.3 Dynamic Voltage and Frequency Scaling

Video decoders have a highly variable workload due to the varying prediction modes that enable high coding efficiency. While FIFOs are used in Section 2.2 to address workload variation at the 4x4 block level, dynamic voltage and frequency scaling (DVFS) allows the decoder to address the varying workload at the frame level in a power-efficient manner [43].

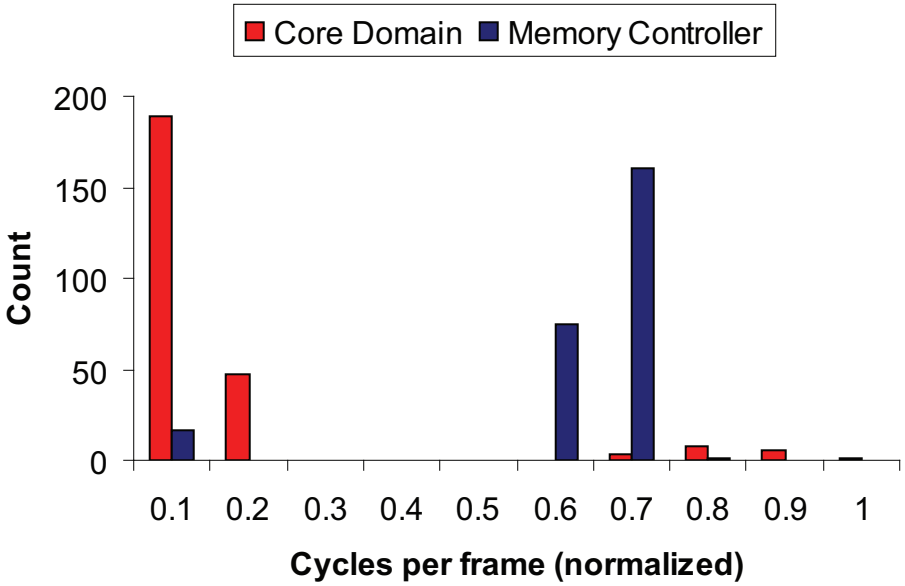
DVFS adjusts the voltage and frequency based on the varying workload to minimize power. This is done under the constraint that the decoder must meet the deadline of one frame every 33 ms to achieve real-time decoding at 30 fps. The two requirements for effective DVFS include accurate workload prediction and the voltage/frequency scalability of the decoder. Accurate workload prediction is ideal to maximize the efficiency of DVFS, but it can be quite challenging. One possible solution is to embed a measure of the workload at the ENC and transmit it in the bitstream. A second solution, which requires less changes to the ENC or the video standard, is to monitor the DEC's performance in real time and adjust the voltage and frequency several times per frame. This section only addresses the scalability of the decoder that enables DVFS.

DVFS can be performed independently on the core domain and memory controller domain as their workloads vary widely and differently depending on whether the decoder is working on I-frames or P-frames. 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. Figure 6-4 shows the workload variation across the "mobcal" sequence. Table 6.4 shows the required voltages and frequencies of each domain for an I-frame and P-frame. Figure 6-5 shows the measured frequency and voltage range of the two domains in the decoder ASIC. Once the desired frequency is determined for a given workload, the minimum voltage can be selected from this graph.

To estimate the power impact of DVFS, only the two operating points (P-frame and I-frame) shown in Table 6.4 are used. The power of the decoder was measured separately for each operating point using a mostly P-frame video and an I-frame-only video averaged over 300 frames. The frame type (I or P) can be determined from the slice header (assuming



(a) Cycles per frame (workload) across sequence



(b) Distribution of cycle variation

Figure 6-4: Workload variation across 250 frames of "mobcal" sequence.

one slice per frame). Table 6.5 shows the impact of DVFS for a group of pictures (GOP) size of 15 with a GOP structure of IPPP. This corresponds to one I-frame followed by a series of P-frames, where the GOP is the period of I-frame insertion among the P-frames.

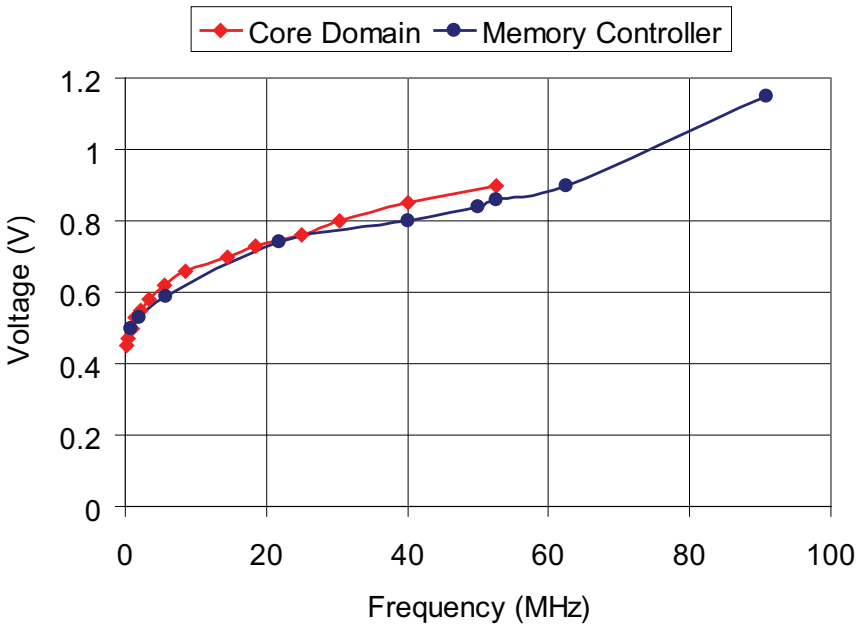


Figure 6-5: Measured frequency versus voltage for core domain and memory controller. Use this plot to determine maximum frequency for given voltage. Note: The rightmost measurement point has a higher voltage than expected due to limitations in the test setup.

DVFS can be done in combination with frame averaging for improved workload prediction and additional power savings [44, 45].

Table 6.4: Measured voltage/frequency for each domain for I-frame and P-frame for 720p sequence.

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

6.4 Real-Time ASIC Demonstration

Verifying and demonstrating the real-time operation of a video decoder can be a challenging task. There are many components involved in such a system, such as the ASIC, several

Table 6.5: Estimated impact of DVFS for GOP structure of IPPP and size 15.

Method	Core		Memory Controller		Relative Power [%]
	Freq [MHz]	Volt [V]	Freq [MHz]	Volt [V]	
No DVFS	53	0.90	50	0.84	100
DVFS	14 or 53	0.70 or 0.90	25 or 50	0.76 or 0.84	75

memories for storing input and output data, a test harness to generate the input clocks and data, and a monitor to display all the outputs.

The test setup is shown in Figure 6-6. The OCFB was implemented using two 32-bit-wide SRAMs [46], one for luma and one for chroma. A FPGA board based on [47] was used to interface the ASIC to the display. The FPGA board is slightly different than [47]. The main modification is the use of a larger SRAM size (16MB instead of 4MB), to allow the storage of eight 720p frames (split into luma and chroma).

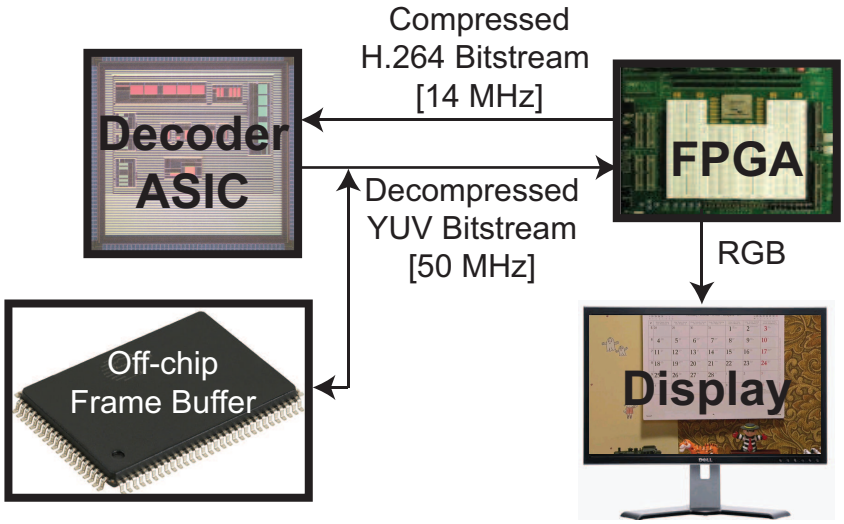


Figure 6-6: Test setup for H.264 decoder

An actual photograph of the lab setup that was used to demonstrate real-time video decoding is shown in Figure 6-7.

The architecture of the test harness on the FPGA is shown in Figure 6-8. The FPGA logic

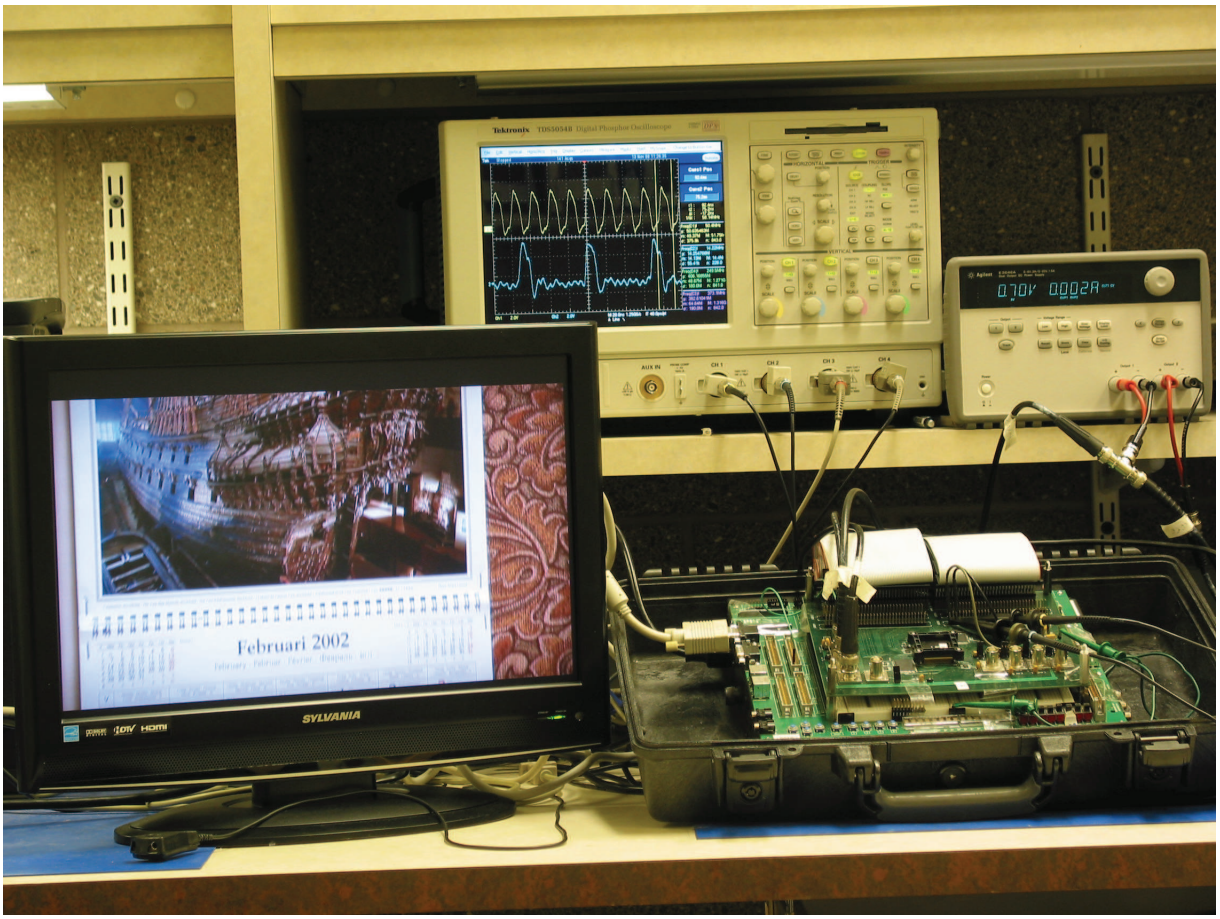


Figure 6-7: Photo of lab video demo

has several functions. A FIFO is used to read the compressed video, stored in flash memory, and feed it to the ASIC whenever it is requested. When the ASIC writes out the decoded pixels to the OCFB, the same data is also written into a FIFO on the FPGA. This data is then rearranged and written into the display buffers. At the same time, a separate process on the FPGA reads the pixel data (in YUV format) from the display buffers in raster scan order. The luma and chroma pixel components are then combined and converted into RGB format, and then sent out to the VGA controller. Equation 6.1 describes this conversion from three 8-bit YUV values to another three 8-bit RGB values. A Digital Clock Manager (DCM) on the FPGA uses several phase-locked loops (PLLs) to create all the different clocks needed by the system, and these clocks are listed in Figure 6-8.

$$
\begin{aligned}
R &= 1.164 \times (Y - 16) + 2.018 \times (V - 128) \\
G &= 1.164 \times (Y - 16) - 0.813 \times (U - 128) - 0.391 \times (V - 128) \\
B &= 1.164 \times (Y - 16) + 1.596 \times (U - 128)
\end{aligned}
\tag{6.1}
$$

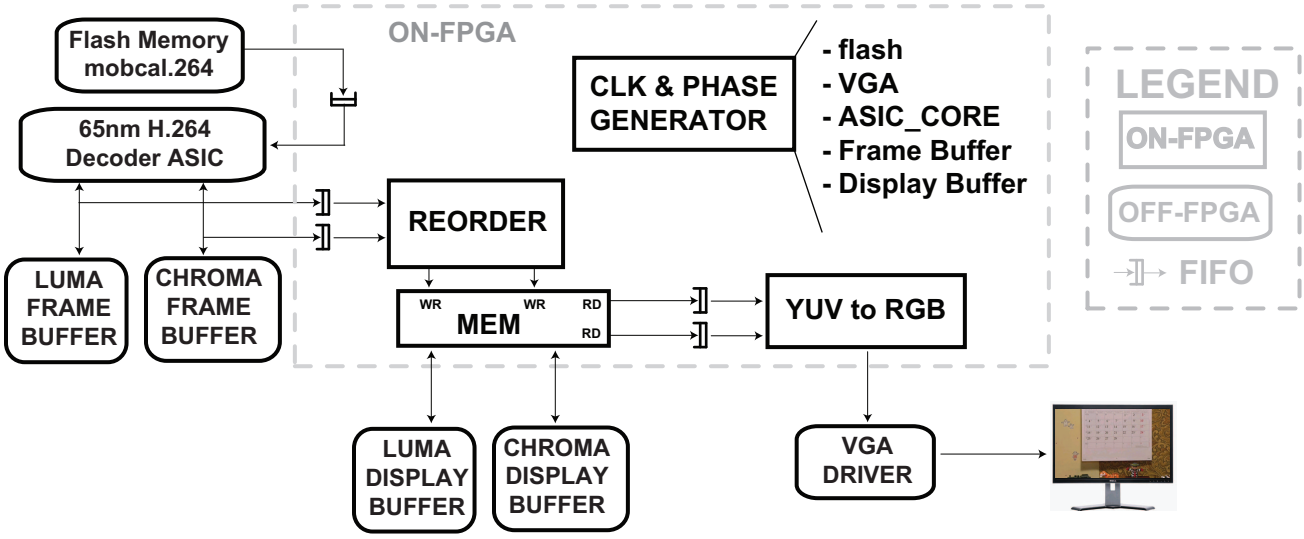


Figure 6-8: Test FPGA architecture

The 32-bit-wide frame and display buffer memories shown in Figure 6-8 store 4 pixel values at each address. Since the *luma* MC interpolator of Section 3.1 processes pixels in vertical columns, rather than horizontal rows, the data in the OCFB was arranged in columns of 4 pixels at each address, as shown in the top part of Figure 6-9. In order to display the data on a monitor, a raster-scan order is used, so it was more convenient to store the data in display buffer using horizontal rows of 4 pixels, as shown in the bottom part of Figure 6-9. The reordering was done at the FPGA by buffering each 4x4 block of pixels written to the OCFB (four 4x1 columns), and then writing its rows (1x4) into the display buffer.

The reordering from the frame buffer to the display buffer is also necessary for the *chroma* components. This rearrangement of pixels is shown in Figure 6-10. Since the chroma interpolator predicts a 2x2 block of pixels for each MV, it reads either a 2x2 or 3x3 area of pixels from the OCFB, depending on whether the MV is integer or fractional. If the address

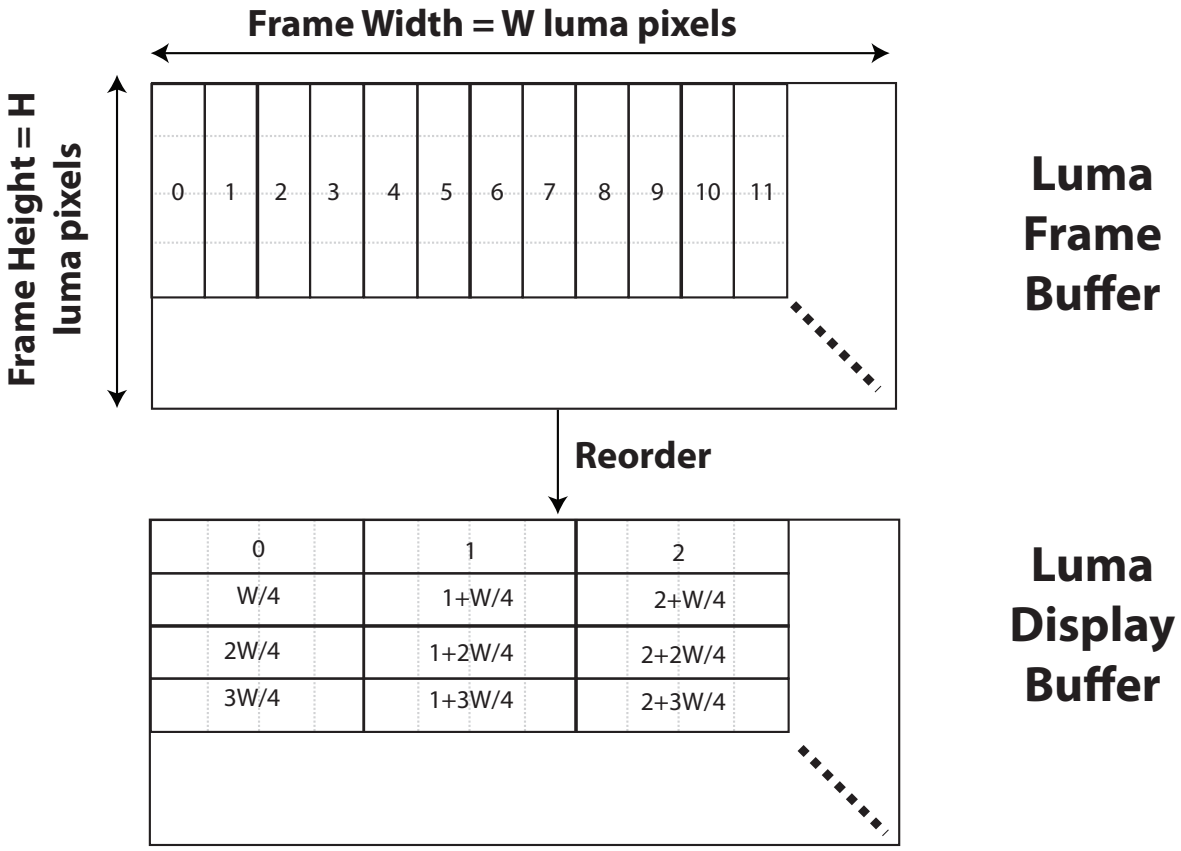


Figure 6-9: Reordering of luma pixels

of this block is aligned to the memory boundaries, this 2x2 block (4 pixels) can be stored in one memory location. Therefore, to minimize the total number of 32-bit reads, the 2x2 blocks are stored as a 2x2 box at each location in memory, as shown in the top part of Figure 6-10. In order to display the data on a monitor, a raster-scan order is used. Just as was the case for luma, it is more convenient to store the data in the display buffer using horizontal rows of 4 pixels, as shown in the bottom part of Figure 6-10.

The test setup in Figure 6-8 can first be used to validate the full correctness of individual decoded frames. Although this task can first be done by visual inspection of the frame's appearance on the monitor, the actual data bits stored in memory should be checked for any errors that the human eye cannot easily catch. For example, to verify frame_N, the FPGA can stop the ASIC clocks and stall the DEC once that frame has been fully decoded. The expected frame contents, generated by a software decoder such as [36], are then loaded into

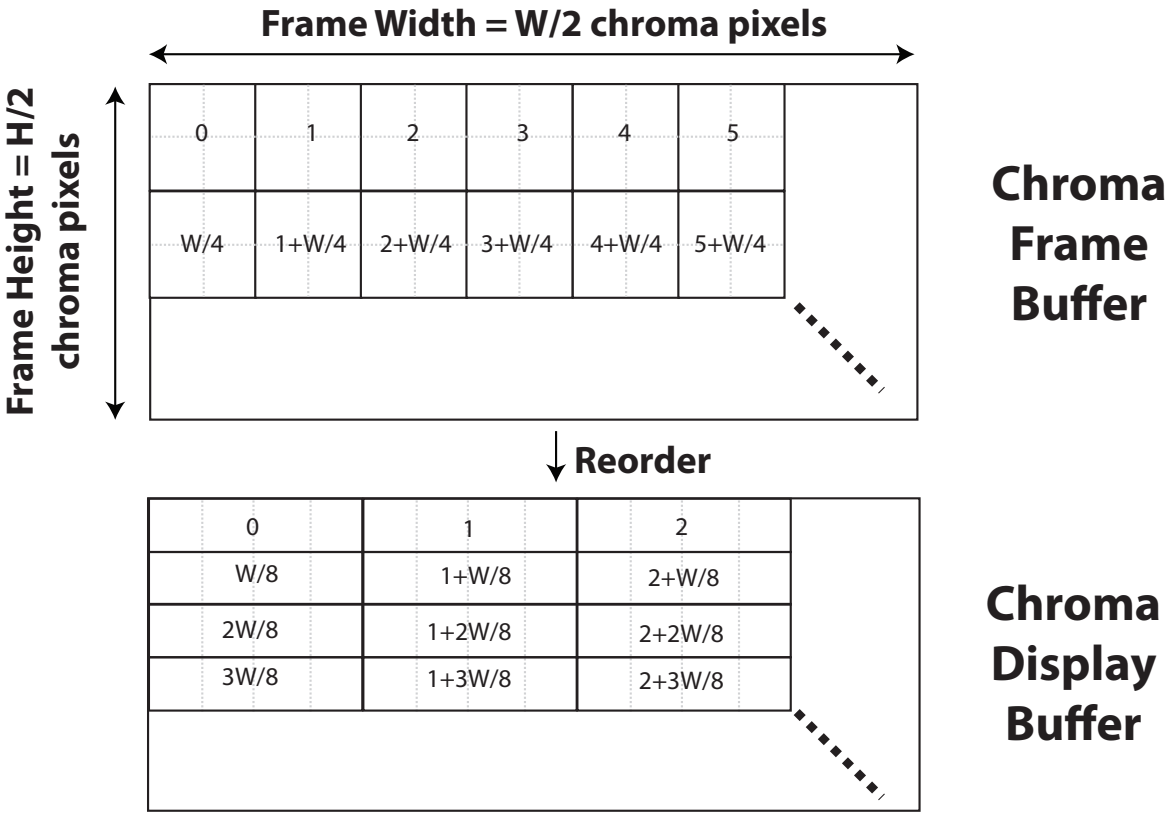


Figure 6-10: Reordering of chroma pixels

the flash memory on the FPGA board. A software program is then run on the FPGA's Micro-Blaze processor to read from both the reference and decoded frames and print out an error over UART whenever the contents differ. This process was run for several different frames and no errors were discovered.

To demonstrate real-time decoding, the compressed video was first loaded into the flash memory on the FPGA board. Since the flash memory has a fixed size, the total number of compressed frames that could fit inside depends on the video. For the three different 720p videos that were tested, the video memory could store 300 frames for "mobcal", 300 frames for "shields", and 144 frames for "parkrun". In order to perform continuous real-time decoding for an extended period of time, the input video was decoded in an infinite loop. This also made it easy to obtain stable power measurements for the ASIC. While the clock frequencies were set at compile time by the FPGA, the ASIC supply voltages were manually

adjusted via separate source-meters.

Video frames can have widely varying workloads, as described in Section 6.3. Since a new frame must be displayed every 33ms for 30fps, the display buffer must receive a new frame on average every 33ms. In the test setup of Figure 6-8, the display buffer has the capacity to store up to 8 different frames. If the ASIC DEC produces frames faster than 30fps on average, the display buffer will eventually be filled up and the clocks to the ASIC must be temporarily turned off. If the ASIC produces frames at less than 30fps, the display buffer eventually becomes empty. If there are no new frames to display, the FPGA simply repeats the last frame until a new frame appears. Note that as the size of the display buffer increases, the instantaneous decoder throughput in fps can be better averaged, thus potentially reducing the number of ASIC stalls and repeated frames. The number of repeated frames is counted for each video loop in order to obtain an accurate estimate of average effective frame rate, as shown in Equation 6.2. For example, if the display rate is 75fps (maximum for some LCD monitors) and there are 500 repeated frames out of the 300 total looped frames, the effective frame rate is 28.125fps.

$$fps_{effective} = fps_{display} \times TotalLoopFrames \div (Repeats + TotalLoopFrames) \qquad (6.2)$$

Although this method gives a good estimate of the average frame rate achieved by the ASIC DEC for a fixed voltage/frequency setting, it has a couple of drawbacks. First of all, it does not handle bursty activity very well, as the decoder can be stalled if the display buffer fills up, meaning that it has to run faster at all other times in order to make up for that loss in performance. Second, we can show that even when the effective average frame rate is the desired one, the instantaneous frame rate will be either higher or lower, and therefore the video playback will not be smooth. For example, if the ASIC frame rate temporarily exceeds the display rate, the video will appear to speed up, whereas in the converse case the video will appear to slow down and there will be many repeated frames. In a production-level implementation, the voltage and frequency would have to be controlled more dynamically,

Table 6.6: Equivalent 720p frame rates for different resolutions

Resolution Name	**Equivalent [frames per second]**	**MegaPixels per second**
QCIF	0.41	0.38
CIF	3.3	3.04
D1	11.3	10.4
720p	30	27.6
1080p	67.6	62.2

rather than be fixed for the entire duration of the video. This would ensure smooth playback and no repeated frames.

For example, a good workload prediction scheme as described in Section 6.3 would guarantee that the DEC runs just fast enough during each frame to guarantee completion within 33ms. Alternatively, some frame averaging could be done at the display buffer to maintain an average throughput of 30 fps, while guaranteeing that the display buffer never becomes empty. This could be done by monitoring the fullness and emptiness of the display buffer and setting the DEC frequency accordingly. If the display buffer becomes close to full, the DEC could be slowed down, whereas the DEC would be sped up when the display buffer is getting close to empty.

The ASIC was designed to process videos of only 720p resolutions, since the frame width and height were internally hard-coded to 1280 and 720. However, for the purpose of characterization, it is interesting to explore how the performance of the ASIC would vary for the different resolutions shown in Table 1.1. In order to emulate the other resolutions and frame rates, we could just operate at the 720p resolution, but vary the frame rate such that the same throughput in pixels per second is obtained. This leads to the equivalent frame rates shown in Table 6.6.

6.5 Results and Measurements

The H.264 Baseline Level 3.2 decoder, shown in Figure 6-11 was implemented in 65-nm CMOS and the power was measured when performing real-time decoding of several 720p

video streams at 30 fps (Table 6.7) [12]. The video streams were encoded with the x264 software [48] with a GOP size of 150 (P-frames dominate).

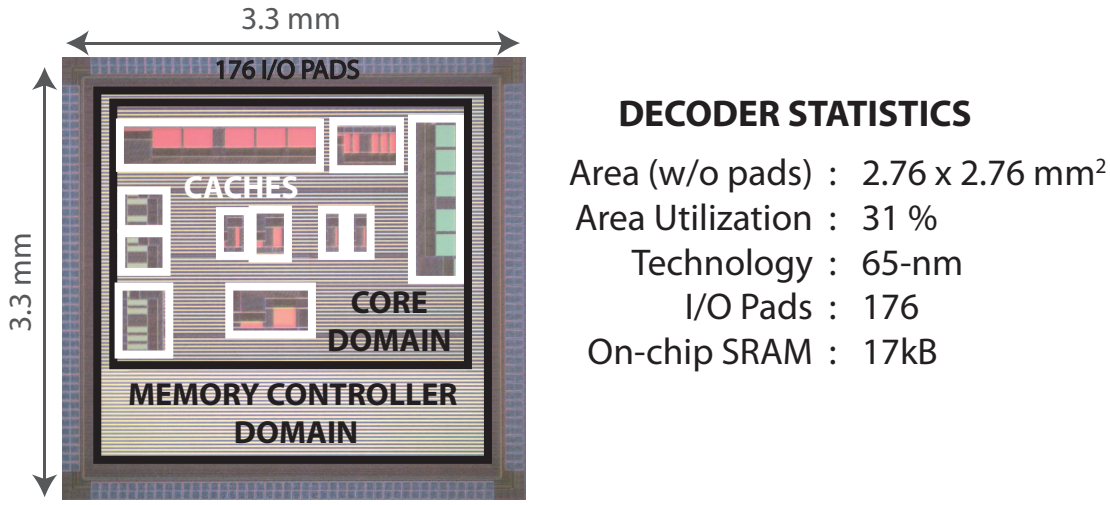


Figure 6-11: Die photo showing the different domains

Figure 6-12 shows a comparison of this ASIC with other decoders. 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 has lower power and frequency relative to D1 of [32]. The decoder can operate down to 0.5 V for QCIF at 15 fps for a measured power of 29 μW. The power of the I/O pads or the off-chip frame buffer (OCFB) was not included in the measurement comparisons.

The reduction in power over the other reported decoders can be attributed to using the low-power techniques described in this work and also benefits from using a more advanced silicon technology. To separate the two effects, we can try to estimate the power savings due to process scaling alone. Since our implementation uses the 65nm process, we can estimate what the power consumption of the other decoders of Figure 6-12 would be if they also used 65nm. For a given architecture, a more advanced process allows the circuits to operate at a lower voltage for the same throughput requirement. The voltage scaling factor can be estimated by simulating the fanout-of-4 delay (FO4) for different process technologies [49] and supply voltages, as shown in Figure 6-13. To compute the equivalent supply voltage at

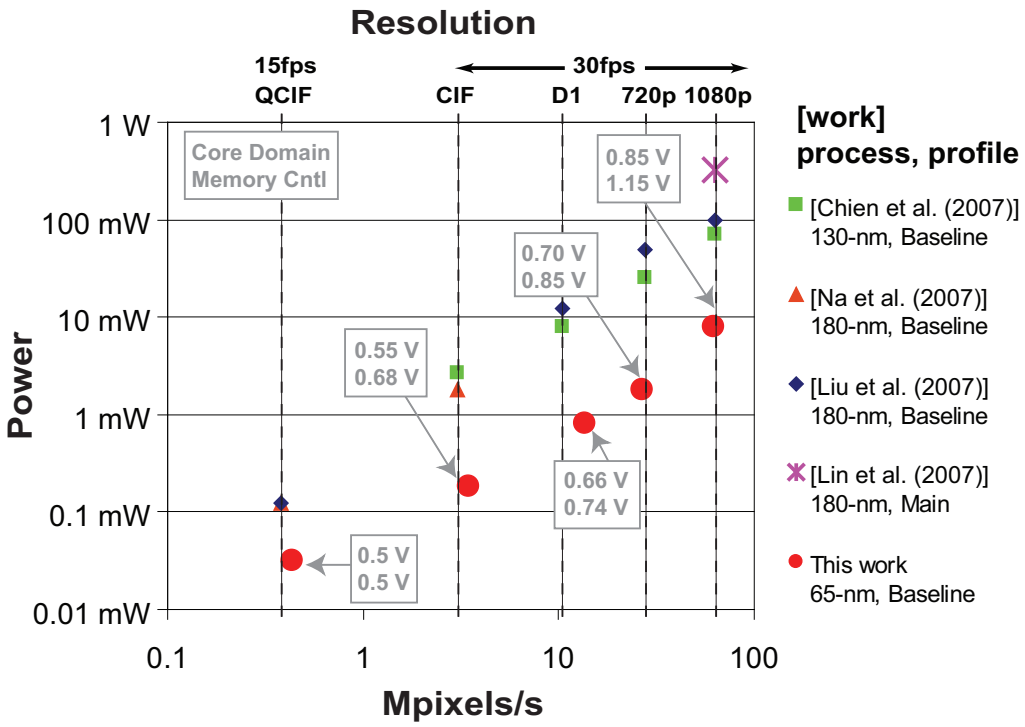


Figure 6-12: Comparison with other H.264/AVC decoders Chien et al. [32], Na et al. [13], Liu et al. [10], Lin et al. [9]

65nm, the operating voltage of the older process is mapped to a FO4, and the same FO4 is mapped back to a supply voltage for 65nm.

Figure 6-13 shows that advanced processes reduce power by allowing a throughput-constrained circuit to operate at a lower supply voltage. In addition, process scaling also provides a linear decrease in transistor capacitance to first order, since the transistor widths, lenghts and oxide thickness all scale down linearly. The length of interconnect reduces linearly, while coupling capacitance increases as the wires get taller and closer together. As a result, the decrease in wire capacitance is at most linear with process scaling. To compute the total power savings, we can scale the published results of Figure 6-12 by the linear decrease in effective capacitance and decrease in operating voltage enabled by process scaling. Figure 6-14 shows how our ASIC compares with other decoders, once the effects of process scaling have been accounted for. At 720p resolutions, our decoder draws almost half the power of [10]. For 1080p resolutions, our decoder draws slightly more power than [10], and

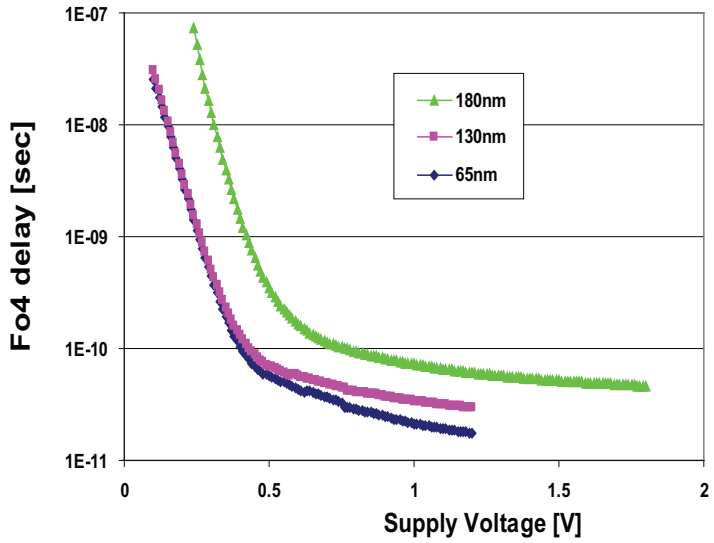


Figure 6-13: FO4 delays for different technologies across supply voltages using the predictive models of [49]

that can be attributed to a difficulty in raising the memory controller frequency high enough, which led to a larger requirement for the core frequency and voltage to compensate for the resulting stalls.

The variation in performance across 15 dies is shown in Figure 6-15. The majority of the dies operate at 0.7-V. Note that the spread in minimum supply voltage is only 30mV, so even if all chips were operated at the maximum of 0.72V, the increase in power would be minimal.

Table 6.7: 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 BW (Gbps)	1.2	1.1	1.2
Core Freq (MHz)	14	14	25
Mem Ctrl Freq (MHz)	50	50	50
Core Vdd [V]	0.7	0.7	0.8
Mem Ctrl Vdd [V]	0.84	0.84	0.84
Power (mW)	**1.8**	**1.8**	**3.2**

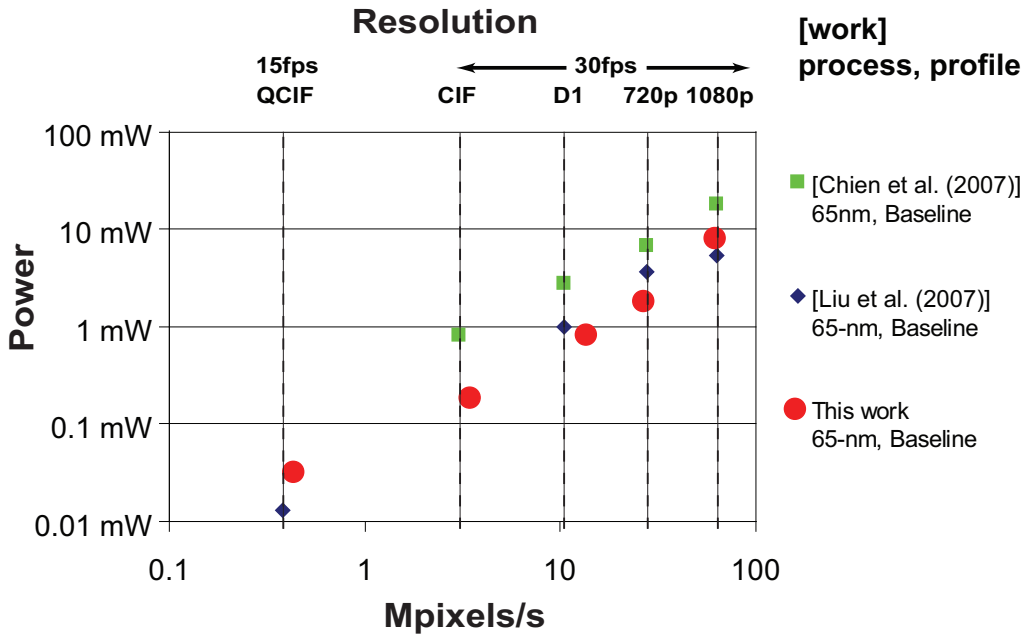


Figure 6-14: Comparison with other H.264/AVC decoders, estimated for the same 65nm process, Chien et al. [32], Liu et al. [10]

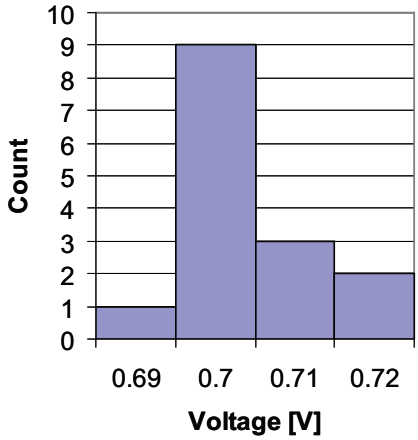


Figure 6-15: Voltage supply variation across test chips

It is important to consider the impact of this work at the system-level of a multimedia device. As voltage scaling techniques can reduce the decoder power below 10 mW for high definition decoding, the system power is then dominated by the OCFB. [10] shows that 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 [38], can deliver 51.2 Gbps at 39 mW. For 720p decoding, the required bandwidth is 1.25 Gbps after the memory optimizations of Figure 6-2. This corresponds to a frame buffer power of 1 mW, based on a linear estimate from [38]. Furthermore, off-chip interconnect power can be reduced by using embedded DRAM or system in package (i.e. stacking the DRAM die on top of the decoder die within a package).

The display typically consumes around a third of the power on a mobile platform [50]. With upcoming technologies such as organic light-emitting devices (OLEDs) and quantum dot-organic light-emitting devices (QD-OLEDs) the display power costs will be reduced. OLEDs have lower power compared to LCDs since there is no back-light required [51].

6.6 Power Breakdown

This section shows the simulated power breakdown during P-frame decoding of the "mobcal" sequence. The power of P-frames is dominated by MC (42%) and DB (26%), as seen on the left chart of Figure 6-16. About 75% of the MC power, or 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 (about 2 luma pixels are read for every decoded pixel). At 0.7-V, the on-chip caches consume 0.15 mW.

The total leakage of the ASIC at 0.7-V is 25 μW which is approximately 1% of the 1.8mW 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 leakage breakdown across decoder units is shown in Figure 6-17.

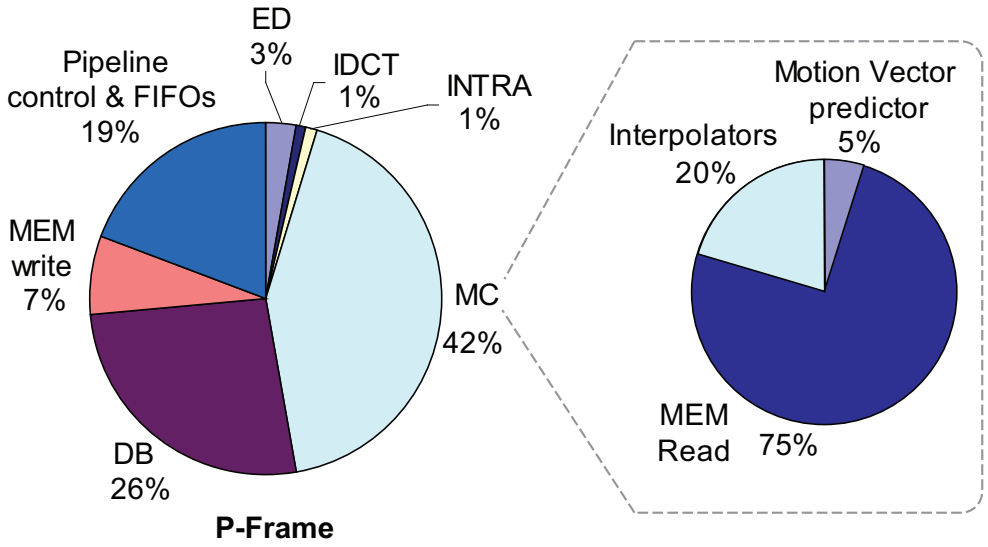


Figure 6-16: Post-layout simulated power breakdown during P-frame decoding

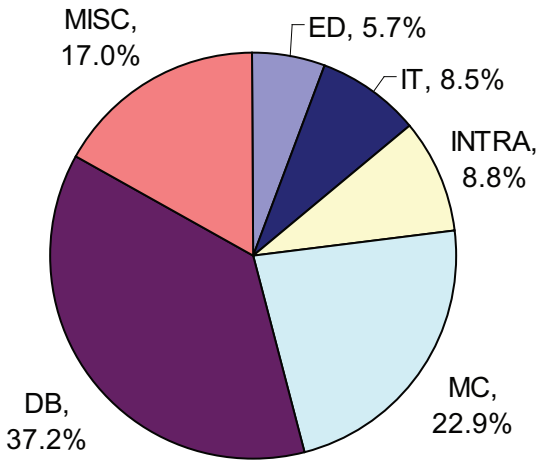


Figure 6-17: Post-layout simulated ASIC leakage power breakdown

6.7 Area Breakdown

The area breakdown by decoder unit is shown in of Figure 6-18. The area is dominated by the DB unit due the SRAM caches which occupy 91% of the DB area.

The cost of parallelism is primarily an increase in area. The increase in total logic area due to the parallelism in the MC unit (Chapter 3) and DB unit (Section 2.5) is about 10%. When compared to the entire decoder area (including on-chip memory) the area overhead is less than 3%.

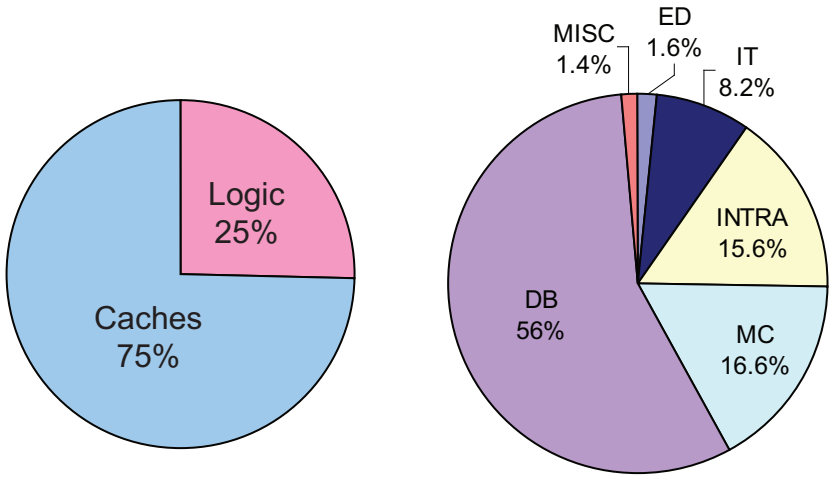


Figure 6-18: Post-layout area breakdown

6.8 Summary

This chapter presented the design of a H.264/AVC Baseline Level 3.1 video decoder ASIC that was fabricated in 65-nm CMOS and verified for real-time operation. The ASIC operated down to 0.7-V and had a measured power as low as 1.8 mW when decoding a high definition 720p video at 30 fps, which is over an order of magnitude lower than previously published results. The ASIC design made extensive use of the pipelining and parallelism techniques described in Chapter 2, and also optimized the memory system using the MC and FLLC caching techniques of Chapter 5.

Chapter 7

Conclusions

In this thesis we described how to build low-power video decoders by targeting low-voltage operation and efficient memory accessing. Low-voltage operation was enabled by using pipelining and parallelism techniques to reduce the number of cycles required to decode each frame. Memory optimization was focused on replacing power-hungry accesses to large memories with reads and writes to smaller on-chip caches. The proposed techniques were all validated by either ASIC implementation, RTL synthesis, layout, and/or simulation.

This work makes several key contributions to the field of video decoder hardware design, as listed below:

1. Non-interlocked 4x4 pipeline achieves nearly maximum concurrency for the different decoder units.

2. Arbitrary degree of parallelism demonstrated for the MC and DB units.

3. Demonstrated several multi-core video decoder architectures, for both existing and future video standards.

4. Proposed and evaluated several on-chip caching techniques for reducing power in the memory system.

5. Showed how domain partitioning and DVFS help reduce video power for a given architecture.

Based on our extensive prototyping and demonstration systems, we have shown that all of these ideas are feasible to implement in a complex system such as a video decoder. For each of the ideas, we also quantified the benefits (power and frequency reduction) versus the costs (area, design complexity). This thesis should therefore give future designers a set of tools which they can selectively use for their specific multimedia application.

7.1 Future Areas of Research

This section suggests several areas of video coding which could be investigated in future work. These topics were not directly addressed by this thesis.

7.1.1 Rate-Distortion-Power Video Coding

Typically, videos are coded while only considering the trade-off between rate and distortion. Rate is a measure of compression efficiency, while distortion is a measure of image quality relative to the original raw video. However, it would be useful if power was added as a third dimension to the coding process. For example, if the video size could be reduced by 1% while increasing the expected power of the mobile energy-constrained DEC by 50%, the ENC could consider this tradeoff in a quantitative way and decide whether it is worth the extra computation cost. If the decoder has a strict power budget, as shown in Figure 7-1, this sets the minimum achievable compression, and also maximizes the PSNR (minimizes distortion) given the power constraint. This type of analysis was done for a software video encoder in [52].

One way to achieve power-scalability in video coding is by using different coding options in the current H.264 video standard. For example, the high profile of H.264 can achieve an increase in coding efficiency over the baseline profile, but requires more computation and power. Other ways to trade off power versus coding efficiency can be thought of for future video standards. One example is performing variable-precision arithmetic as proposed in Section 2.4.

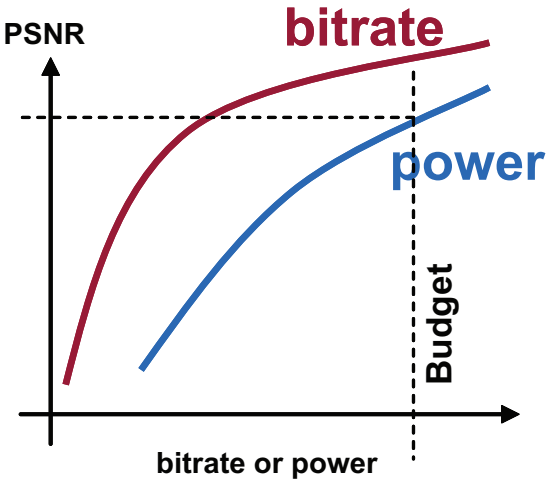


Figure 7-1: Illustration of a possible trade-off between bitrate, PSNR, and decoding power

7.1.2 Video System Integration

Further research could explore how the frame buffer (FB) can be stored closer to the DEC processor either using eDRAM or a stacked die approach. The on-chip caching impact then becomes lower since the off-chip power is no longer as large as before, so the trade-offs will look different. For example, [53] presents an architecture for 3D stacking of the memory above the video processor.

7.1.3 Multi-Standard Video Decoder ASICs

Most of the work explored in this thesis has focused on the H.264 video standard or proposed minor changes that can enable high-level parallelism. However, H.264 is not the only video decoder algorithm used by video players. There are many other popular standards [54], such as MPEG-1, MPEG-2, VC-1, RealVideo, Flash, and so on.

Since video content on mobile devices can be dynamically downloaded from any source, the compression algorithm used could be any of the ones in [54]. If the video stream is to be played back using the H.264 decoder hardware, the non-H.264 videos need to be trans-coded locally on the mobile device, using configurable software or hardware [55]. Although the transcoding does not necessarily need to take place in real time, it can use up as much or

even more power than decoding the actual video using the H.264 hardware, thus negating all the power savings of the decoder ASIC.

Ideally, the decoder ASIC could handle a group of the most popular standards. Furthermore, some of the less popular algorithms could be handled in software. The software implementations would draw more power and might not be able to handle the processing requirements of higher resolutions. This type of multi-standard ASIC is a challenge to design, because the chip area and design time increases linearly with the number of supported standards. This is an active area of research ([10, 56]), and an ideal solution would maximize the amount of hardware reuse amongst the different algorithms.

7.1.4 Video Encoder ASICs

This thesis mostly focuses on techniques for reducing the power in video decoder ASICs. However, encoders are also becoming popular on mobile devices such as digital video recorders, so there is a lot of active research in this area ([15, 57]). The processing required in a video encoder is arguably more complex, since the encoding process must also include a decoder to ensure identical reference frames.

Many of the pipelining, parallelism, and caching ideas discussed in this thesis can be ported to an encoder implementation. Additionally, because the standard only constrains the decoder, the encoder offers some flexibility over the decoder. This allows further architecture explorations, which could lead to more energy-efficient hardware designs.

7.1.5 Workload Prediction

The works in [44, 58, 59, 60] propose several techniques to predict the varying workload during video decoding. As discussed in Section 6.3, accurate workload prediction improves the power-efficiency of video decoders, since it allows them to operate at the lowest frequency and voltage and decode frames "just-in-time" to meet the deadlines. In order to achieve better prediction for H.264 and future video standards, further research is required to either integrate the prediction methodology in the standard, or to provide more predictable ways

in the standard to estimate a frame's required workload.

Bibliography

[1] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," in *Proceedings of the IEEE*, April 1995, pp. 498–523.

[2] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, 2nd ed. Prentice-Hall, 2002.

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

[4] 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.

[5] "Wikipedia H.264 entry." [Online]. Available: http://en.wikipedia.org/wiki/H.264

[6] T. Wedi and H. Musmann, " Motion- and Aliasing-Compensated Prediction for Hybrid Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 577–586, July 2003.

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

[8] H.-Y. Kang, K.-A. Jeong, J.-Y. Bae, Y.-S. Lee, and S.-H. Lee, "MPEG4 AVC/H.264 Decoder with Scalable Bus Architecture and Dual Memory Controller," in *IEEE International Symposium on Circuits and Systems*, May 2004, pp. II– 145–8 Vol.2.

[9] 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.

[10] T.-M. Liu, T.-A. Lin, S.-Z. Wang, W.-P. Lee, K.-C. Hou, J.-Y. Yang, and C.-Y. Lee, "A 125uW, 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.

[11] T.-M. Liu, T.-A. Lin, S.-Z. Wan, W.-P. Lee, K.-C. Hou, J.-Y. Yang, and C.-Y. Lee, "An 865uW H.264/AVC Video Decoder for Mobile Applications," in *IEEE Asian Solid State Circuits Conference*, November 2005, pp. 301–304.

[12] 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.

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

[14] 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.

[15] K. Iwata, S. Mochizuki, T. Shibayama, F. Izuhara, H. Ueda, K. Hosogi1, H. Nakata1, M. Ehama, T. Kengaku, T. Nakazawa, and H. Watanabe, "A 256mW Full-HD H.264 High-Profile CODEC Featuring Dual Macroblock-Pipeline Architecture in 65nm CMOS," in *Symp. VLSI Circuits Dig. Tech. Papers*, 2008.

[16] C.-C. Lin, "Implementation of H.264 Decoder in Bluespec System Verilog," *Master's Thesis, Massachusetts Institute of Technology*, February 2007.

[17] V. Chandra, A. Xu, H. Schmit, and L. Pileggi, "An Interconnect Channel Design Methodology for High Performance Integrated Circuits," in *Design, Automation and Test In Europe Conference (DATE)*, 2004.

[18] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Parallel 44 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2003.

[19] 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.

[20] H.-C. Chang, C.-C. Lin, and J.-I. Guo, "A novel low-cost high-performance VLSI architecture for MPEG-4 AVC/H.264 CAVLC decoding," in *IEEE International Symposium on Circuits and Systems*, vol. 6, May 2005, pp. 6110–6113.

[21] S. Cho, "Design of a Low Power Variable Length Decoder for MPEG-2 System," *Master's Thesis, Massachusetts Institute of Technology*, June 1997.

[22] T.-H. Tsa, D.-L. Fang, and Y.-N. Pan, "A Hybrid CAVLD Architecture Design with Low Complexity and Low Power Considerations," in *IEEE International Conference on Multimedia and Expo*, July 2007, pp. 1910–1913.

[23] S.-Y. Tseng, "A Pattern-Search Method for H.264/AVC CAVLC Decoding," in *IEEE International Conference on Multimedia and Expo*, July 2006, pp. 1073–1076.

[24] T. Xanthopoulos, "Low Power Data-Dependent Transform Video and Still Image Coding," *PhD Thesis, Massachusetts Institute of Technology*, February 1999.

[25] S. Nomura, F. Tachibana, T. Fujita, C. K. Teh, H. Usui, F. Yamane, Y. Miyamoto, C. Kumtornkittikul, H. Hara, T. Yamashita, J. Tanabe, M. Uchiyama, Y. Tsuboi, T. Miyamori, T. Kitahara, H. Sato, Y. Homma, S. Matsumoto, K. Seki, Y. Watanabe, M. Hamada, and M. Takahashi, "A 9.7mW AAC-Decoding, 620mW H.264 720p 60fps

Decoding, 8-Core Media Processor with Embedded Forward-Body-Biasing and Power-Gating Circuit in 65nm CMOS Technology ," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2008.

[26] E. van der Tol, E. Jaspers, and R. Gelderblom, ""Mapping of H.264 Decoding on a Multiprocessor Architecture"," in *Image and Video Communications and Processing*, 2003.

[27] V. Sze, D. F. Finchelstein, M. E. Sinangil, and A. P. Chandrakasan, "A 1.8-mW 0.7-V H.264 720p Video Decoder," *IEEE Journal of Solid-State Circuits*, to appear in November 2009.

[28] L. Philipps, S. V. Naimpally, R. Meyer, and S. Inoue, "Parallel architecture for a high definition television video decoder having multiple independent frame memories," U.S. Patent No. 5,510,842, issued to Matsushita Electric Corporation of America, April 1996.

[29] J. Zhao and A. Segall, "VCEG-AI32: New Results using Entropy Slices for Parallel Decoding," ITU-T Study Group 16 Question 6, Video Coding Experts Group (VCEG), 35th Meeting: Berlin, Germany, July 2008.

[30] L. Nachtergaele, F. Catthoor, F. Balasa, F. Franssen, E. D. Greef, H. Samsom, and H. D. Man, "Optimization of Memory Organization and Hierarchy for Decreased Size and Power in Video and Image Processing Systems," in *IEEE International Workshop on Memory Technology, Design, and Testing.*

[31] S. Dutta, W. Wolf, and A. Wolfe, " A Methodology to Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 1, pp. 36–53, February 1998.

[32] 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*, 2007.

[33] M. Budagavi and M. Zhou, "Video coding using compressed reference frames," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2008.

[34] T. Nishikawa, M. Takahashi, M. Hamada, T. Takayanagi, H. Arakida, N. Machida, H. Yamamoto, T. Fujiyoshi, Y. Maisumoto, O. Yamagishi, T. Samata, A. Asano, T. Terazawa, K. Ohmori, J. Shirakura, Y. Watanabe, H. Nakamura, S. Minami, T. Kuroda, and T. Furuyama, "A 60 MHz 240 mW MPEG-4 Video-phone LSI with 16 Mb Embedded DRAM," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2000.

[35] "Fffmpeg Open-Source Codec." [Online]. Available: http://www.ffmpeg.org/

[36] "H.264/AVC Reference Software, JM 12.0." [Online]. Available: http://iphome.hhi.de/suehring/tml/

[37] T. Tan, G. Sullivan, and T. Wedi, "VCEG-AE010: Recommended Simulation Common Conditions for Coding Efficiency Experiments Rev. 1," ITU-T SG16/Q6, January 2007.

[38] 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*, 2004.

[39] Y.-W. Huang, B.-Y. Hsieh, T.-C. Wang, S.-Y. Chien, S.-Y. Ma, C.-F. Shen, and L.-G. Chen, "Analysis and Reduction of Reference Frames for Motion Estimation in MPEG-4 AVC/JVT/H.264," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2003.

[40] M. Semiconductors, "Mobile DRAM, The Secret to Longer Life." [Online]. Available: www.google.com

[41] B.-G. Nam, J. Lee, K. Kim, S. J. Lee, and H.-J. Yoo, " 52.4mW 3D Graphics Processor with 141Mvertices/s Vertex Shader and 3 Power Domains of Dynamic Voltage and

Frequency Scaling," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2007.

[42] T. Fujiyoshi, S. Shiratake, S. Nomura, T. Nishikawa, Y. Kitasho, H. Arakida, Y. Okuda, Y. Tsuboi, M. Hamada, H. Hara, T. Fujita, F. Hatori, T. Shimazawa, K. Yahagi, H. Takeda, M. Murakata, F. Minami, N. Kawabe, T. Kitahara, K. Seta, M. Takahashi, and Y. Oowaki, "An H.264/MPEG-4 Audio/Visual Codec LSI with Module-Wise Dynamic Voltage/Frequency Scaling," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, February 2005.

[43] 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.

[44] 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.

[45] 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*. New York, NY, USA: ACM, 2001, pp. 34–39.

[46] "Cypress ZBT SRAMs." [Online]. Available: http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1470v25_8.pdf

[47] "MIT 6.111 Course Labkit." [Online]. Available: http://www-mtl.mit.edu/Courses/6.111/labkit/

[48] "X264 Open-Source Encoder." [Online]. Available: http://www.videolan.org/developers/x264.html

[49] "Predictive Technology Model." [Online]. Available: http://www.eas.asu.edu/~ptm

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

[51] O. Prache, "Active Matrix Molecular OLED Microdisplays," *Displays*, vol. 22, no. 2, pp. 49–56, May 2001.

[52] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu, " Power-Rate-Distortion Analysis for Wireless Video Communication under Energy Constraints," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 645–658, May 2005.

[53] A. Zeng, J.-Q. Lu, R. Gutman, and K. Rose, "Wafer-level 3D Manufacturing Issues for Streaming Video Processors," in *IEEE Advanced Semiconductor Manufacturing*, May 2004.

[54] "Wikipedia entry on commonly-used video codecs." [Online]. Available: http://en.wikipedia.org/wiki/Video_codec#Commonly_used_standards_and_codecs

[55] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, "Video Transcoding: an Overview of Various Techniques and Research Issues," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 793–804, October 2005.

[56] T.-M. Liu, W.-P. Lee, and C.-Y. Lee, " An Area-Efficient and High-Throughput De-Blocking Filter for Multi-Standard V Video Applications," in *International Conference on Image Processing*, September 2005.

[57] T.-C. Chen, Y.-H. Chen, C.-Y. Tsai, S.-F. Tsai, S.-Y. Chien, and L.-G. Chen, " 2.8 to 67.2mW Low-Power and Power-Aware H.264 Encoder for Mobile Applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 222–223, June 2007.

[58] K. Choi, K. Dantu, W. Cheng, and M. Pedram, "Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder," November 2002, pp. 732 – 737.

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

[60] 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.