Energy-Efficient Video Decoding Using Data Statistics

by

Mehul Tikekar

B.Tech., Indian Institute of Technology Bombay (2010)
S.M., Massachusetts Institute of Technology (2012)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author .... Signature redacted ....
Department of Electrical Engineering and Computer Science
August 29, 2017

Certified by .... Signature redacted ....
Anantha Chandrakasan
Vannevar Bush Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .... Signature redacted
Vivienne Sze
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .... Signature redacted ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Energy-Efficient Video Decoding Using Data Statistics

by

Mehul Tikekar

Submitted to the Department of Electrical Engineering and Computer Science
on August 29, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Video traffic over the Internet is growing rapidly and is projected to be about 82% of the total consumer Internet traffic by 2020. To address this, new video coding standards such as H.265/HEVC (High Efficiency Video Coding) provide better compression especially at Full HD and higher video resolutions. HEVC achieves this through a variety of algorithmic techniques such as larger transform sizes and more accurate inter-frame prediction. However, these techniques increase the complexity of software and hardware-based video decoders. In this thesis, we design a hardware-based video decoder chip that exploits the statistics of the video to reduce the energy/pixel cost in several ways. For example, we exploit the sparsity in transform coefficients to reduce the energy/pixel cost of inverse transform by 29%. With the proposed architecture, larger transforms have the same energy/pixel cost as smaller transforms owing to their higher sparsity thus addressing the increased complexity of HEVC's larger transform sizes. As a second example, the energy/pixel cost of inter-prediction is dominated by off-chip memory access. We eliminate off-chip memory access by using on-chip embedded DRAM (eDRAM). However, eDRAM banks spend 80% of their energy on frequent refresh operations to retain stored data retention. To reduce refresh energy, we compress the video data stored in the eDRAM by exploiting spatial correlation among pixels. Thus, unused eDRAM banks can be turned off to reduce refresh energy by 55%. This thesis presents measured results for a 40 nm CMOS test chip that can decode Full HD video at 20 - 50 frames per second while consuming only 25 - 31 mW of system power. The system power is 6 times lower than the state-of-the-art and can enable even extremely energy-constrained wearable devices to decode video without exceeding their power budgets. The inverse transform result can enable future coding standards to use even larger transform sizes to improve compression without sacrificing energy efficiency.

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

Thesis Supervisor: Vivienne Sze
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank Professor Anantha Chandrakasan and Professor Vivienne Sze for their invaluable support and guidance. They have been ever helpful and encouraged me through the tough times in the Ph.D. I have learned a lot in my time at MIT and grown as a student and a researcher thanks to both of them. I want to thank Professor Daniel Sanchez for serving on my thesis committee and providing useful feedback.

I would like to thank members of the Energy-Efficient Circuits and Systems group and the Energy Efficient Multimedia Systems group at MIT for their help, support and all the fun times. They are like a second family to me. I am especially grateful for the help provided by Arun Paidimarri, Michael Price and Priyanka Raina in the chip tapeout.

I would like to thank Chiraag Juvekar, Pulkit Agrawal, Shibani Santurkar, Luis Fernández, Diana Wofk, and Valerie Sarge for giving me the opportunity to mentor them.

Thanks to Margaret Flaherty and Janice Balzer for helping with the administrative stuff.

Thanks to the authors of all the free and open source software for sharing their work with everyone and asking only for the continued freedom of their work in return and often, not even that.

Finally, thanks to my family without whom I wouldn't be here.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The decade since the introduction of H.264/AVC in 2003 has seen an explosion in the use of video for entertainment and business. Standard Definition (SD) and High Definition (720p HD) broadcasts are making way for Full HD (1080p), which in turn, are expected to be replaced by Ultra HD resolutions like 4K and 8K. Video traffic over the internet is growing rapidly and is expected to be about 82% of the global consumer internet traffic by 2020 [1]. These factors motivated the development of a new video standard that provides high coding efficiency while supporting large resolutions. H.265/HEVC (High Efficiency Video Coding) [2] is the successor video standard to the popular H.264/AVC. The first version of HEVC was ratified in January 2013 and it aims to provide a 50% reduction in bit rate at the same visual quality [3]. HEVC achieves this compression by using larger and variable-sized pixel blocks, more prediction methods, better entropy coding, and better in-loop filters. These algorithmic improvements come at the cost of increased silicon area and power consumption in hardware implementations of the video codecs.

Power is a key constraint on portable devices such as smartphones, tablets and the newly developed wearable devices like smartwatches, fitness trackers, VR headsets. On wearable devices, the power budget can be as low as 50 mW [4] which makes it extremely challenging to perform video coding on fixed-function hardware accelerators let alone software implementations. Most of the previous work on hardware video decoders has focused on improving the energy efficiency of the decoder chip [5]–[7]. However, from a system perspective, the energy required for video decoding is dominated by memory access to off-chip DRAM. Some of the more recent work has demonstrated techniques such as caching, reference frame compression [8] and DRAM-aware memory mapping energy [9] to reduce off-chip memory power. However, even with these techniques, off-chip memory accesses require 2.8 to 6 times as much energy as the video decoder chip. Thus, it remains the main obstacle in the path of reducing energy consumption.

In this thesis, we demonstrate a fully-integrated video decoder using embedded DRAM (eDRAM) to eliminate off-chip memory. Compared to DRAM, eDRAM requires more frequent refresh to retain data in memory which makes refresh energy the major component of eDRAM power. We show techniques to reduce eDRAM refresh energy by exploiting redundancy in the data and the low latency and energy/access of eDRAM. We also demonstrate energy-efficient architectures in the core logic of the video decoder that exploit signal statistics, data dependencies and parallelism. The rest of this chapter provides a brief description of the video decoding process, the impact of algorithmic improvements in H.265/HEVC on silicon area and energy and an introduction to the contributions of this thesis.

1.1 Overview of Video Coding

Video coding uses spatial and temporal redundancy in video data and characteristics of the human visual perception system to compress the video. Each frame in the video stream is split into small pixel blocks for the purposes of coding. Using neighboring pixels in the same frame or similar pixels from previous frames, a prediction for each pixel block is generated. The difference between the pixel values and prediction, called residue, is transformed to spatial frequency domain using 2-dimensional discrete cosine transform and quantized to remove high-frequency content in the residue. Removing the high-frequency content provides lossy compression without significantly affecting the perceived quality. In this manner, the pixel data is converted to syntax elements such as:

1. Split flags, partition modes - these denote how a frame is broken down into a pixel block. The frame is initially divided into equal-sized square blocks called Macroblocks (MB) in H.264/AVC and Coding Tree Units (CTU) in H.265/HEVC. Each CTUs are hierarchically split into 4 smaller square blocks called Coding Units (CU) as denoted by split flags. CUs

in H.265/HEVC and MBs in H.264/AVC are partitioned into smaller square and rectangular blocks, called Prediction Unit (PU) in HEVC, for the purpose of generating a prediction. CUs are further split hierarchically into square transform units for the purpose of transform and quantization.

2. Motion vectors - these denote locations of pixels in previous frames to be used to generate a prediction. A motion vector has three components: reference frame index, horizontal motion vector, vertical motion vector. Reference frame index denotes the index of the previous frame from which to fetch the prediction. The horizontal and vertical components of the motion vector are the difference between the spatial location of current block and the that of the prediction in the reference frame. Motion vector can be specified at a quarter-pixel granularity. For integer motion vectors, the prediction value is simply copied from the reference frame. For fractional motion vectors, the prediction value needs to be interpolated.
3. Intra modes - these denote how to generate a prediction from neighboring pixels in the same frame. One row of pixels along the left and top edge of the current block is used as reference pixels for intra-prediction.
4. Transform coefficients - these are the transformed and quantized residue. The level of quantization used affects compression and video quality. A higher level of quantization results in better video quality at the cost of compression and vice versa.

These syntax elements undergo lossless compression to produce the final video bitstream. The decoding of the bitstream back to video is completely and unambiguously specified by the video coding standard. The decoder works as follows:

1. Unpack syntax elements from the bitstream. This happens in two steps: context-adaptive entropy decoding of bitstream into stream of binary symbols, followed by parsing of binary symbols into syntax elements. appendix A.1 provides more details on the parsing process.
2. Generate predictions using syntax elements like motion vectors and intra modes. Motion vectors are used for temporal prediction from previously decoded frames (inter-frame prediction) and intra modes are used for spatial prediction from previously decoded pixels in

the same frame (intra-frame prediction). Inter-frame prediction is described in Section 2.1.2 and intra-frame prediction is described in Section 2.1.3.

3. Scale and transform the coefficients back into residuals. The coefficients are first scaled to reverse the quantization performed by the encoder. Following that, HEVC uses inverse discrete cosine transforms and inverse discrete sine transform to obtain the residual pixel values. This process is described in chapter 3.
4. Add prediction to residuals
5. Apply a smoothing filter that removes coding artifacts while preserving details in the original frame. HEVC uses two filters: a deblocking filter and Sample Adaptive Offset which are described in more detail in appendix A.2.

Figure 1-1 shows a block diagram of this process.

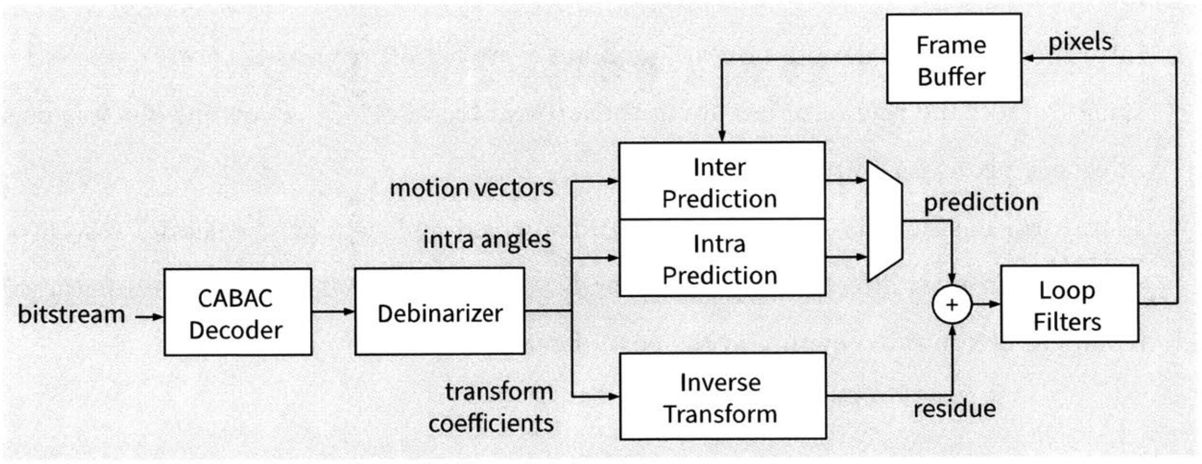


Figure 1-1: Block diagram of HEVC decoding process showing syntax elements and processing modules

1.2 Impact of H.265/HEVC Algorithms on Silicon Area and Energy

H.265/HEVC and H.264/AVC both use the basic scheme of inter-frame and intra-frame prediction, inverse transform, loop filter, and entropy coding explained in the previous section. However, HEVC improves upon AVC in several respects. With any coding standard, a range of compression ratios and video quality can be achieved by varying the quantization level. Improvements in coding efficiency are evaluated using the BD-Rate metric which is defined as the average bitrate over a particular range of PSNR [10]. The dual metric, BD-PSNR, defined as the average PSNR over a range of bitrates is also used.

HEVC achieves about 50% reduction in BD-Rate as compared to AVC over a variety of test video sequences [11]. High-resolution video sequences tend to show better improvements as HEVC is targeted towards Full HD and higher resolutions. These improvements come at the cost of complexity in both video encoders and decoders. For software decoders, the run-time for HEVC decoders can be as much as twice the run-time of AVC decoders [12]. A similar increase in complexity for hardware decoders can be expected, with complexity measured in terms of logic area, memory size and bandwidth, and energy consumption.

Table 1.1 summarizes the impact of the new algorithms introduced in HEVC on hardware design of video decoders. The rest of this section discusses the impact in more detail.

Table 1.1: The impact of new features in HEVC on hardware decoder implementations

HEVC feature	Impact on hardware decoder implementation
Large and hierarchical coding units	Large pipeline buffer, use SRAM instead of register arrays, more flexible architecture
Large prediction units	Less computation per pixel, more opportunities for computation reuse
More intra-prediction modes	Higher design complexity
Longer inter-prediction filters	More memory bandwidth from frame buffer
Larger transform units	More computation per pixel, use SRAM for transpose memory
Improved loop filters	Simpler deblocking filter design, new design for SAO filter
Entropy decoding	Simpler CABAC design for higher throughputs

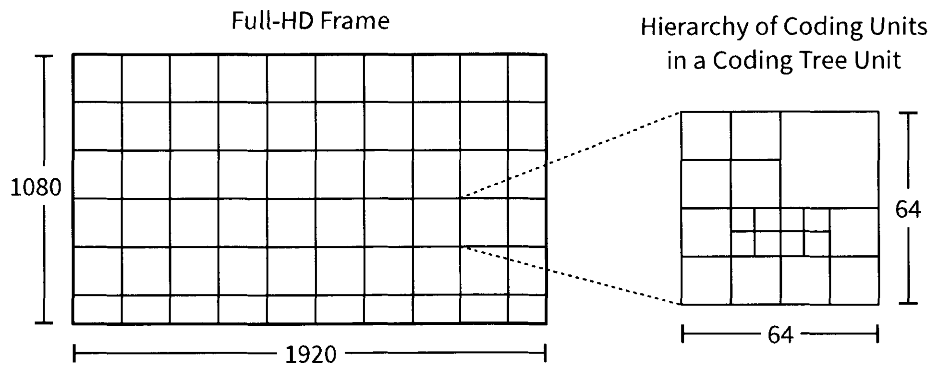


Figure 1-2: Example of splitting a Full-HD frame into 64×64 CTUs and splitting a CTU into a hierarchy of CUs

1.2.1 Larger and Hierarchical Coding Units

As shown in Figure 1-2, in HEVC, the frame is broken down into raster-scanned coding tree units (CTU) which are fixed to 64×64, 32×32 or 16×16 in each frame. The CTU may be split into four partitions in a recursive fashion down to coding units (CU) as small as 8×8. This recursive split into 4 partitions is called quad-tree. In comparison, AVC uses a fixed macroblock size of 16×16 with no Hierarchical split.

The larger sizes of CTUs and CUs in HEVC require larger pipeline buffers in hardware video decoders. In some cases, buffers in AVC decoders that could be efficiently implemented in register arrays need to be implemented in SRAMs in HEVC. SRAMs have lower flexibility in terms of read and write patters, and are more challenging to operate at lower voltages. On the other hand, the varied sizes of CUs within a CTU requires a much more flexible architecture. The CUs within a CTU may use a mix of intra and inter-prediction which introduces new dependencies between intra and inter-prediction processing.

1.2.2 Prediction Units (PU)

In HEVC, each CU may be partitioned into one, two or four prediction units. In all, up to 8 different types of partitions are possible as shown in Figure 1-3. Compared to AVC, HEVC introduces 4 new asymmetric partitionings. Of these, only the square PUs may use intra-prediction, and all PUs can use inter-prediction. The diversity in CU sizes combines multiplicatively with the diver-

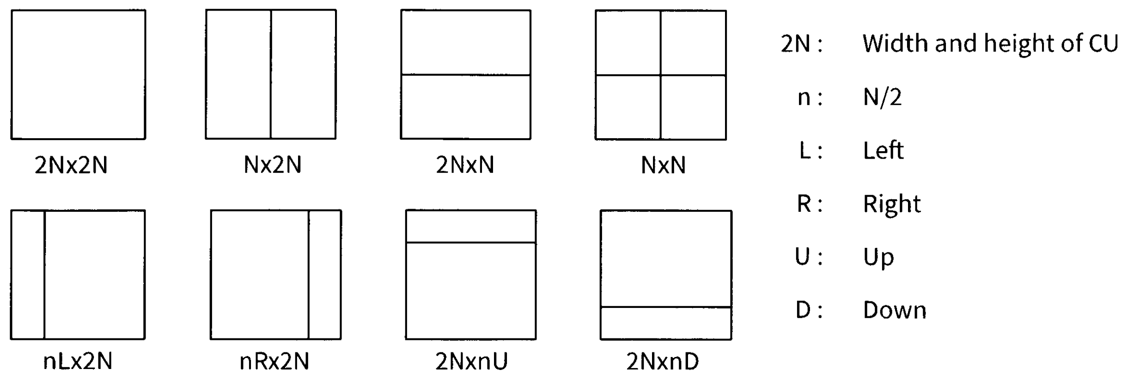


Figure 1-3: Eight possible types of partitions for a 2N×2N CU into PUs

sity in partitions thus giving 24 different PU sizes in HEVC Main Profile. On the other hand, the AVC macroblock can be partitioned into square blocks and symmetric blocks giving 7 types of macroblock partitions.

This variety of PU sizes needs to be addressed by flexible architectures for inter-prediction and intra-prediction and larger local memories. However, it is observed that the computational complexity per pixel is lower for larger PUs due to more opportunities for computation reuse.

1.2.3 Intra-Prediction

HEVC Main Profile uses 35 intra-prediction modes including planar, DC and 33 angular modes. In comparison, AVC uses 10 intra-prediction modes. HEVC intra prediction involves two steps. The first step is preparation of reference pixels which has $O(N)$ complexity for an N×N block, followed by the actual intra-prediction which has $O(N^2)$ complexity. As a result, smaller blocks have higher complexity per pixel which can make pipelining challenging. The large block sizes in HEVC require using a combination of SRAMs and registers to allow for flexible access at low area cost [13].

1.2.4 Inter-Prediction

PUs may be predicted from either one (uni-prediction) or two (bi-prediction) reference locations from up to 16 previously decoded frames. For luma prediction, HEVC uses an 8-tap interpolation filter compared to the 6-tap filter in AVC. The memory bandwidth overhead of longer interpolation filters is largest for the smallest PUs. The smallest PU in HEVC is 4×4, which would require up to two 11×11 reference pixel blocks. This is a 49% increase over the 9×9 reference block required for AVC. To alleviate this worst-case memory bandwidth increase, HEVC does not use 4×4 PUs for inter-prediction (only intra-prediction may use 4×4). Also, 8×4 and 4×8 PUs may only use uni-prediction (8×8 to 64×64 PUs may use uni-prediction or bi-prediction).

The large block size allows for more computation reuse but at the cost of more local memory. The longer interpolation filter can be efficiently implemented using multiple constant multiplication techniques [14].

1.2.5 Inverse Transform

In HEVC, each coding unit is recursively split into transform units. HEVC Main Profile uses square TUs from 32×32 down to 4×4 with discrete cosine transform for all sizes and discrete sine transform for 4×4. This compares to the 8×8 and 4×4 transforms in AVC. If a 2-D transform is expressed in terms of matrix-multiplication operations, the computational complexity of an N×N 2-D transform is $O(N^3)$. Further, to preserve precision of the computation for larger transforms, the constant transform weights are rounded to 8-bit integer values in HEVC as compared to 4-bit in AVC.

All of these factors make it very challenging to implement the large transforms in HEVC. In AVC hardware decoders [5], inverse transform consumed less than 1% of the total energy. Comparatively, HEVC inverse transform modules consumed 17% of the total energy in HEVC decoders [9]. A similar increase is seen in area costs as well.

1.2.6 Loop Filters

The purpose of the loop filters is to remove coding artifacts from decoded frames. The cleaned-up frame is then displayed and also used for inter-prediction for future frames. The cleaned-up frame provides a more accurate prediction for future frames, thus increasing the coding efficiency. The term, loop filter, is used to differentiate from any non-standard post-processing that may be applied to the video prior to display and is not part of the feedback loop of inter-prediction. HEVC has two loop filters: the deblocking filter and Sample Adaptive Offset (SAO).

The deblocking filter was first introduced in H.264/AVC for removing blocking artifacts. The deblocking filter in HEVC is significantly simpler than in AVC. Edges to be filtered are identified from CU, PU and TU edges and a smoothing filter is applied on up to 4 pixels on either side of the edge. To simplify the data dependencies, only those edges that lie on an 8×8 grid are considered. As a result, all the edges in a frame can be filtered independently.

SAO is a new loop filter introduced by HEVC [15]. Unlike the deblocking filter that operates edge-wise, SAO operates pixel-wise. Each pixel is classified into different categories based on the value of its neighbors and itself. The encoder signals an offset value for each category and the appropriate offset value is added to each pixel. Similar to deblocking filter, each pixel in SAO can be filtered independently.

Both the loop filters result in data dependencies between CTUs along the horizontal and vertical directions. Due to the horizontal raster scan order of processing, the horizontal dependencies can be managed by small CTU-sized local buffers, but the vertical dependencies need frame-width-sized line buffers.

1.2.7 Entropy Coding

AVC entropy coding uses either context-based adaptive binary arithmetic coding (CABAC) or context-adaptive variable length coding (CAVLC). In comparison, HEVC Main Profile uses only CABAC to simplify the design of compliant decoders. Further, the design of CABAC has been simplified to use fewer contexts and fewer context-coded bins to aid high-throughput designs. The CABAC hardware in this work was designed by Yu-Hsin Chen [16].

1.3 Considerations for Wearable Devices

Wearable devices such as smartwatches, smart-glasses and fitness trackers have stringent budgets for power (< 50mW) [4] and form factor. Previous work [6]–[9], [17] has focused on video resolutions of 3840×2160 and beyond, which are better suited for devices with larger power budgets like smartphones, tablets, set-top boxes, etc. The large frames need to be stored in DRAM, which dominates the overall system power. For example, [8], [9], [17] use DDR3 memory which has a background power of 92mW [18] for the smallest module.

This work targets the more power-constrained devices with < 50mW power budget. We demonstrate a HEVC decoder ASIC capable of decoding 1080p video at 24 frames per second (fps) in 25 mW power. The performance of the decoder is scalable down to 640x480 at 60 fps while consuming only 9.5 mW of power.

We use embedded DRAM as the main memory of the video decoder. eDRAM helps reduce system power and physical footprint but requires more frequent refreshes compared to DRAM. One unit of eDRAM memory is a 0.5 MB macro that can support a maximum bandwidth of 1.6 GB/s. For typical HEVC decoding workloads, tens of eDRAM macros are needed to meet the storage requirement, but just 1 to 2 macros are sufficient to meet the bandwidth requirement. As a result, very few macros are accessed for read/write at any point of time, but all macros need to be refreshed frequently to retain the stored data. As a result, the energy cost for refreshing the eDRAM far exceeds the cost accessing the memory.

SRAM with 3D stacking is another option for a fully-integrated solution for wearable devices. This has the benefits of both low access power like eDRAM and zero refresh power. However, SRAM has ~3x lower density than eDRAM and ~10x lower density than DRAM (see Chapter 4) which can make it very expensive to use for the sizes required in the video decoder.

1.4 Thesis Contributions

In this thesis, we demonstrate how to design an energy-efficient video decoder using a combination of algorithm and architecture techniques. The energy consumption for video decoding is dominated by off-chip memory access to DRAM. This motivates the use of embedded DRAM (eDRAM) to design a fully-integrated video decoder that does not require any external components. We identify that the main challenge with using eDRAM in video decoding is its high refresh power and we propose techniques to address this challenge.

Looking at the progression of video coding standards from MPEG, H.264/AVC and H.265/HEVC to H.266 in the future, we can see that the trend is to perform coding on blocks of pixels with ever-increasing sizes and variety [19]. The increasing pixel block size affects the complexity of inverse transform the most and we propose an architecture that exploits sparsity in large blocks to reduce the energy consumption of the inverse transform module.

Table 1.2 summarizes the contributions of some of the video decoders. The contributions are mainly in reducing off-chip memory power used by motion compensation, on-chip memory size used in pipelining, and parallel decoding to increase throughput. The next few subsections explain the contributions of this thesis.

Table 1.2: Summary of contributions of previous video decoders and this work.

Video decoder	Contributions
JSSC'14 [9]	4K HEVC decoder that addresses larger pixel blocks in HEVC through efficient pipelining, area cost of larger transforms through multiple-constant multiplications and DRAM access cost through caching and latency-aware memory mapping.
ISSCC'12 [17]	An 8K H.264 video decoder that uses frame-level parallelism to increase throughput, shared caching between parallel decoders to increase cache hitrate and reference frame compression to reduce DRAM power.
JSSC'17 [8]	An 8K HEVC decoder that uses elastic pipelining to address the variety of pixel block sizes and a high-throughput 16pixel/cycle pipeline to meet the high demand of 8K video.
A-SSCC'13 [6]	HEVC video decoder that uses reference frame compression to reduce DRAM power and shared line-buffer for reducing on-chip SRAM size
ESSCIRC'14 [7]	A 10-bit HEVC video decoder with efficient packing of 10-bit data, wavefront-parallel decoder cores and a cache with machine-learning based logic.
This work, VLSI'17 [20]	Fully-integrated HEVC video decoder with eDRAM. Uses reference frame compression to reduce eDRAM power wavefront-parallel decoder cores, and exploits data statistics to reduce power of inverse transform logic.

1.4.1 Exploiting Input Signal Statistics

Video coding and other applications process a lot of sparse data. We show how architectures can be designed to effectively use sparsity to reduce energy consumption. Exploiting sparsity results in workload variation which must be managed at the cost of design complexity. For example, one can use large FIFOs to smooth out the workload variation. This results in a energy versus area trade-off that is discussed in Section 3.2 on the inverse transform architecture for HEVC. We demonstrate a zero-column skip technique that achieves 29% energy/pixel reduction with small area overhead.

Video data possesses spatial and temporal redundancy which is used for lossy data compression. In this work, we demonstrate using the spatial redundancy for lossless compression within the decoder to reduce memory power. Traditionally, such methods have been used to reduce memory access power. In Section 4.3, we show how refresh power can also be effectively reduced using this approach. Using a lightweight compression algorithm and on-demand power up of eDRAM macros, we can reduce refresh power of eDRAM by 50%.

1.4.2 Managing Data Dependencies at Low Cost

Data dependencies in algorithms present fundamental constraints on how the circuit implementation can be pipelined and parallelized. For example, CABAC in entropy decoder uses a successive approximation technique involving updating probability contexts and range after decoding every binary symbol. The logic delay for computing the next state (probability contexts and range) places a upper bound on the speed of the CABAC decoder circuit in terms of number binary symbols per second.

Processing engines like intra-prediction and loop filter depend on neighboring pixels to the left and top. These pixels are best stored in local memory within the processing engines instead of fetching them from main memory. We show, in Section 2.1, how addressing schemes can be designed for these local memories based on the neighbor dependencies to minimize data movement and memory access.

Data dependencies can also be used to determine the smallest block of data that can be pipelined.

In this work, we identified that the data dependencies created by the inverse transform engine determine the data block size for the decoding pipeline. Compared to previous work, we are able to reduce the size of pipeline SRAM by 3x. This is explained in Section 2.1 in more detail.

1.4.3 Parallel Architectures

Strongly related to the issue of data dependencies is the design of parallel architectures. In general, parallelism can be exploited at the macro and micro level. A decoder core is built from processing engines such as CABAC, inverse transform, prediction and loop filters that operate in a pipelined fashion. Micro-level parallelism refers to parallelism within each processing engine and macro-level parallelism refers to using multiple decoder cores in parallel. This work uses both levels of parallelism and highlights their corresponding benefits and challenges.

Micro-level parallelism improves throughput of individual processing engines such as inverse transform, prediction and loop filter. Micro-level parallelism typically works by providing a processing engine with a block of pixels, say 16×16 where the data dependencies allow the engine to compute multiple output pixels per cycle. The order in which these output pixels are computed can be designed to:

- maximize computation reuse
- minimize memory access
- minimize logic switching

In this chip, the processing engines are designed for a throughput of at least 2 pixels per cycle which, at 100 MHz clock frequency, is sufficient for decoding 1920x1080 video at 60fps.

Macro-level parallelism uses multiple decoder cores built from these individual processing engines with each decoder core processing sections of the frame independently. The decoder cores can also process multiple frames in parallel [17]. In this chip, we exploit macro-level parallelism to use 4 decoder cores running at 1/4th the clock frequency. The order of pixels computed by each core is carefully controlled so that the dependencies across the cores can be handled by FIFOs instead of shared memories. This allows for a flexible architecture that can easily scale up to add more decoder cores. However, not all parts of the video decoder can be parallelized at the macro level. The serial part of the decoder, which is the entropy decoder, must be decoupled from the parallel parts to address the different processing orders of the two parts. We show how the decoupling buffer between the serial and parallel parts can exploit the compression inherent in video coding to reduce buffer size by 4×. Section 4.4 discusses this in more detail.

Parallelism enables voltage scaling for reducing energy consumption. The other benefits of micro-parallelism in the form of higher computation reuse, reduced memory accesses and reduced logic switching provide further energy savings. However, micro-parallel architectures are generally not scalable and require a significant redesign effort if the throughput needs to be changed. On the other hand, macro-parallel architectures are more scalable as explained in the last paragraph.

1.4.4 Chip Implementation and Directions for Future Standards

Measurement results for the test chip in 40nm CMOS technology are presented in Chapter 5. The chip can perform 1080p video decoding at 24 frames per second in 25 mW of power. The techniques used to improve energy-efficiency of this work can be applied to other current and future video coding standards as well as other applications that involving time-varying 2-D signals. For future video coding standards, we provide estimates of how the current approaches will scale and make observations on how the approaches and the standard can be co-designed to further improve energy-efficiency.

Chapter 2

Parallelism in HEVC Decoder

This chapter discusses the high-level architecture of the H.265/HEVC video decoder chip. It describes the clock domains used on the chip, considerations for the choice of eDRAM as main memory and comparison with DRAM. The chip's architecture exploits macro-level and micro-level parallelism afforded by the HEVC algorithms and this chapter describes them in detail.

Figure 2-1 shows the system block diagram of the H.265/HEVC decoder. The processing elements in the chip are separated into two clock domains:

1. **Frontend, memory and IO domain**: This domain contains the CABAC entropy decoder, frame buffer and memory access modules such as cache and reference frame compression (RFC). Chip IO consisting of bitstream input and decoded pixel output are also in this domain.
2. **Backend domain**: This domain contains four decoder cores and line buffers connecting them.

Having separate frontend and backend clock domains helps in keeping their workloads decoupled. The frontend workload is a function of the bitrate: the rate of the bitstream input measured in bits/s. The backend workload is a function of the pixel rate which depends on the frame size and frame rate. The ratio of the rates is the compression ratio of the video. Thus, having separate domains for frontend and backend potentially allows the decoder to adapt to different compression ratios by changing the two clock frequencies.

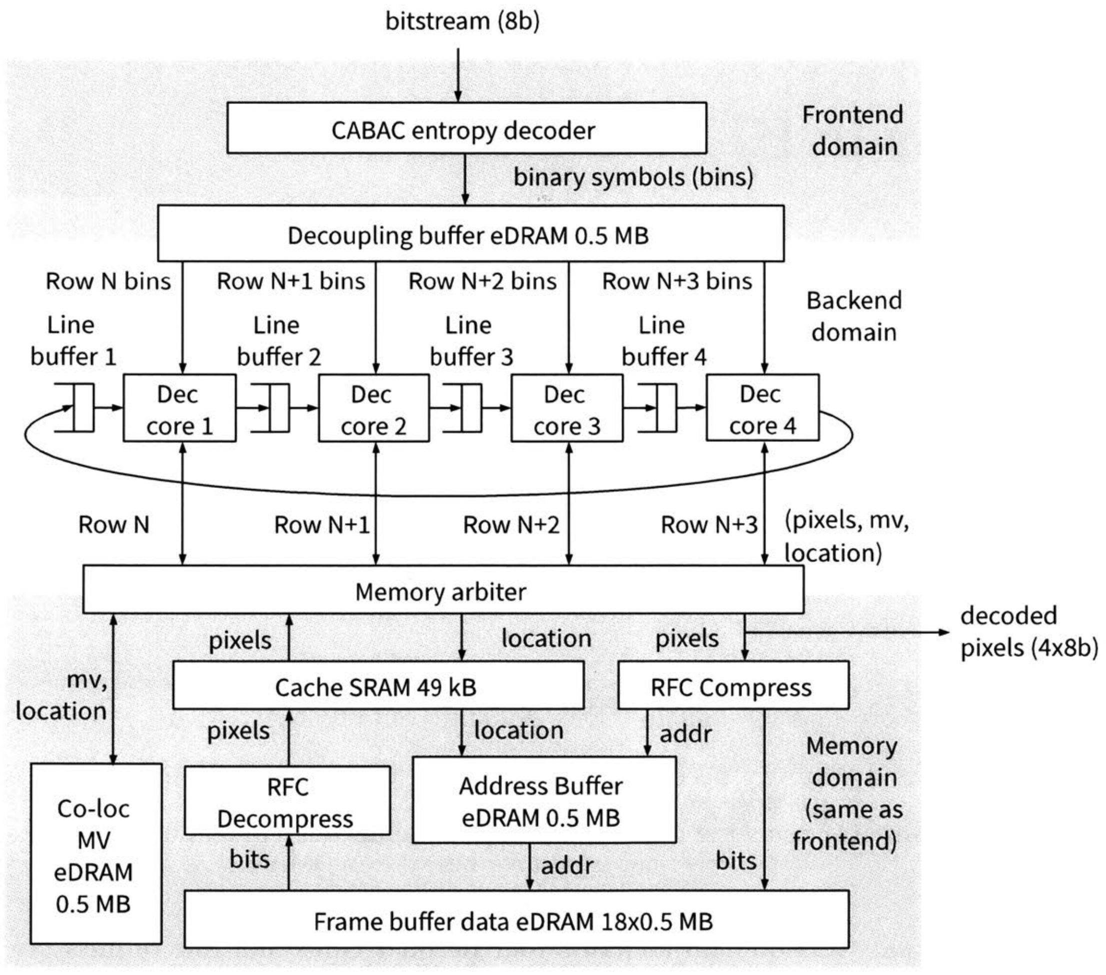


Figure 2-1: High-level architecture of hardware HEVC decoder showing separate clock domains and 4 parallel decoder cores

Clock crossings across the domains are implemented using synchronous FIFOs, i.e. the frontend and backend clocks are assumed to be phase-aligned. For typical workloads, the frontend clock should be 4× the backend clock to balance the throughputs of the two domains. But, in general, any two phase-aligned clocks can be fed to the chip to clock these domains.

2.1 Micro-Parallelism in Decoder Backend

In the backend, the four decoder cores are responsible for parsing binary symbols and reconstructing pixels. This process happens in four stages as shown in Figure 2-2.

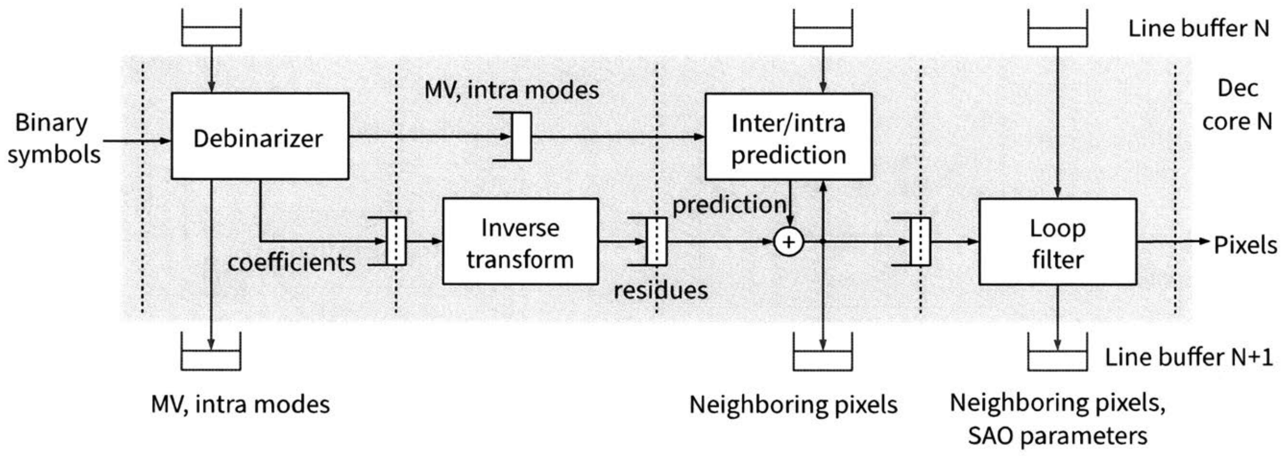


Figure 2-2: Internal architecture of a decoder core with 4 stages of processing engines and line buffers connecting to other decoder cores

1. Binary symbols are parsed by the Debinarizer into syntax elements such as intra modes, motion vectors and transform coefficients.
2. Transform coefficients are dequantized and converted from spatial frequency domain to pixel domain with inverse discrete cosine and sine transforms (IDCT and IDST). The pixel domain values are signed integers denoting the difference (residue) between actual pixel values and a prediction.
3. Prediction is generated by referring to either neighboring pixels in the current frame (intra-prediction) or pixels from previous frames (inter-prediction). Inter and intra-prediction in HEVC are tightly coupled by a feedback loop as the prediction for one block of pixels (with

the residual correction added in) is immediately used as intra reference for neighboring blocks. This is shown in Figure 2-3.

4. The residual corrected prediction frame exhibits artifacts of lossy coding in the form of edges along prediction and transform unit boundaries. These edges are filtered by a loop filter.

This pipeline operates on Coding Tree Units (CTUs), which are 16×16, 32×32 or 64×64 blocks of pixels in the frame, in a horizontal raster scan order. Debinarizer, intra-prediction and loop filter depend on partially decoded pixels and syntax elements from neighboring CTUs to the top and left. The left dependency is satisfied by buffers within the processing engines in the pipeline. The top dependency requires communication between the four decoder cores through line buffers.

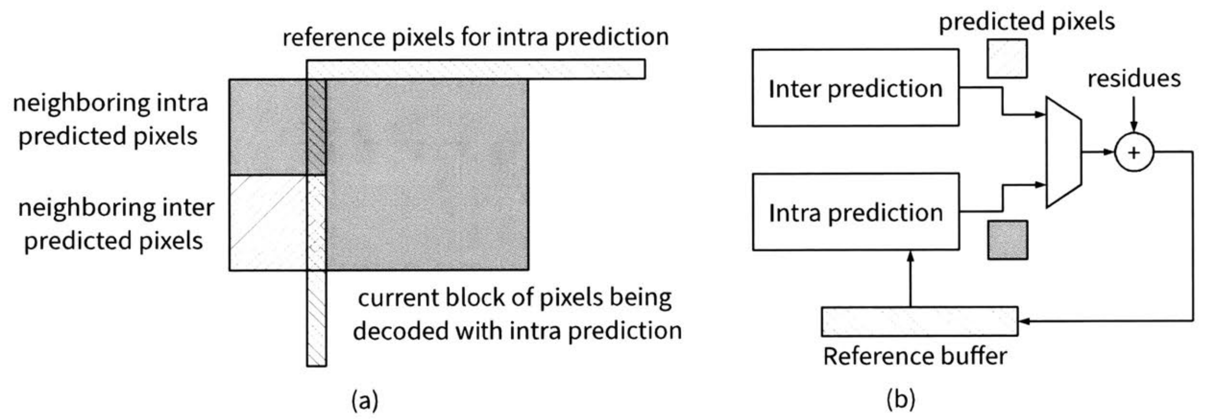

reference pixels for intra prediction
neighboring intra predicted pixels
neighboring inter predicted pixels
current block of pixels being decoded with intra prediction
(a)
predicted pixels
Inter prediction
residues
+
Intra prediction
Reference buffer
(b)

Figure 2-3: (a) Example of neighboring intra and inter predicted pixels used as reference for intra-prediction. (b) Tight feedback loop involving inter and intra-prediction and residual correction

2.1.1 Efficient Pipelining Through Analysis of Data Dependencies

The 4-stage pipeline described in the previous section is regulated through FIFOs without the use of any central pipeline control. In this work, we attempted to minimize the FIFO size by identified the smallest possible pixel block size that the pipeline can operate upon. The process of identifying the smallest pixel block for pipelining is based on understanding the data dependencies within each pipeline stage. The data dependencies are as follows:

1. Inverse transform needs to operate on block sizes that are at least as large as a TU. This is because each residue pixel in a TU depends on every coefficient in the TU.

2. Data dependency in intra-prediction is entirely determined by TU size. Intra PUs that span across multiple TUs are split into smaller units along TU boundaries. Each unit is given the same intra mode as the original PU. Intra PUs smaller than the TU are not allowed in the HEVC standard as it is practically impossible for the encoder to optimize both intra and transform parameters together. The encoder first determines the best intra mode for a PU and then determines the best TU configuration within that PU to encode the residue.
3. There is no data dependency within inter-prediction. In principle, each pixel can be given its own motion vector and its prediction value can be computed independent of other pixels.
4. The loop filter consists of two cascaded filters: Deblocking filter and Sample Adaptive Offset (SAO). Deblocking filter is a filter of range 8×8 and stride 8×8. As a result, 8×8 blocks of pixels can be independently deblocked. SAO can be treated as a filter of range 3×3 and stride 1×1. So, the smallest unit that the SAO can process independently is 1 pixel.

Based on this analysis of the data dependencies, it is clear that TU size determines the smallest pixel block for pipelining. The pipeline block size for this chip and compared to our previous work [9] is shown in Table 2.1

Table 2.1: Pipeline block size for [9] and current chip. Pipeline block size in current chip is equal to largest TU size.

Coding Tree Unit	Pipeline block in [9]	Pipeline block in this work
16×16	64×16	16×16
32×32	64×32	32×32
64×64	64×64	32×32

We shall now see how micro-parallelism is used in some of the processing engines in the decoder backend.

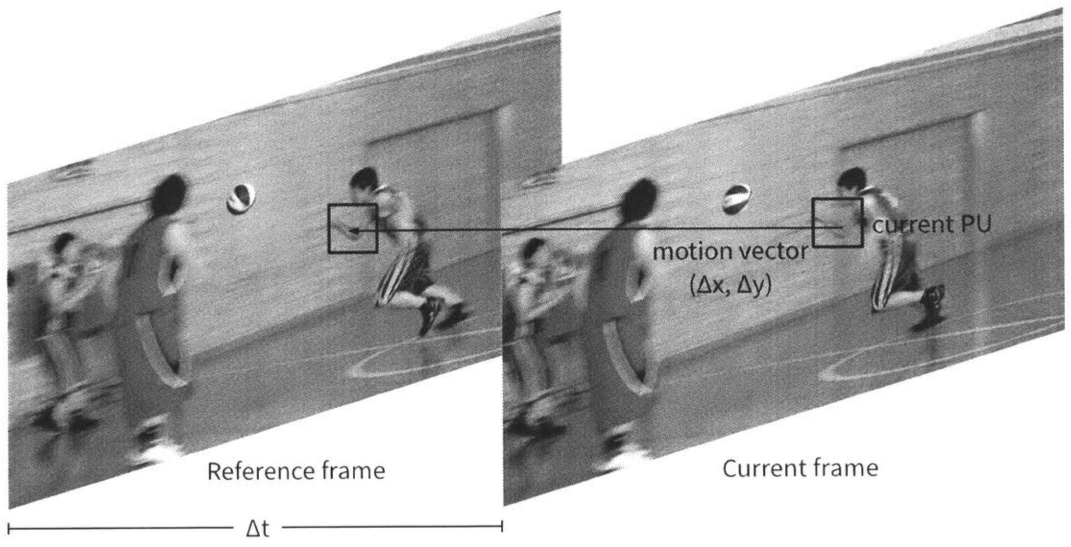


Figure 2-4: Inter-prediction of a Prediction Unit (PU) in current frame using motion vector (MV) pointing to reference frame

2.1.2 Micro-Parallelism in Inter-Prediction

Inter-prediction, also called motion compensation or inter-frame prediction, uses motion vectors (MV) to compute a prediction for a block of pixels in the current frame from a previously decoded reference frame. MV has three components: Δx, Δy and Δt as shown in Figure 2-4. Δx and Δy are differences in pixel coordinates of the current block and reference block, specified at quarter-pixel precision. Δt is specified as the difference in frame index of current frame and reference frame. For motion vectors with quarter-pixel and half-pixel offsets, the prediction pixels are computed using interpolation. The interpolation filter is a 2-D filter designed as two separable 1-D filters. An 8-tap FIR filter is used for luma and a 4-tap FIR filter is used for chroma interpolation. For integer-pixel offsets, the pixels from the reference frame are simply copied over.

Motion vectors are not specified for each pixel but at a Prediction Unit (PU) granularity. A large variety of PU sizes is available in HEVC to adapt to the level of detail in the picture. PUs are rectangular and may vary in size from 8×4 to 64×64 and in shapes of N×N, N×(N/2), N×(N/4), N×(3N/4) and their vertical counterparts. Each PU can have up to 2 motion vectors and when 2 motion vectors are specified, the prediction from both motion vectors is averaged to give the final prediction.

As the interpolation uses a FIR filter, each pixel can be computed independently providing max-

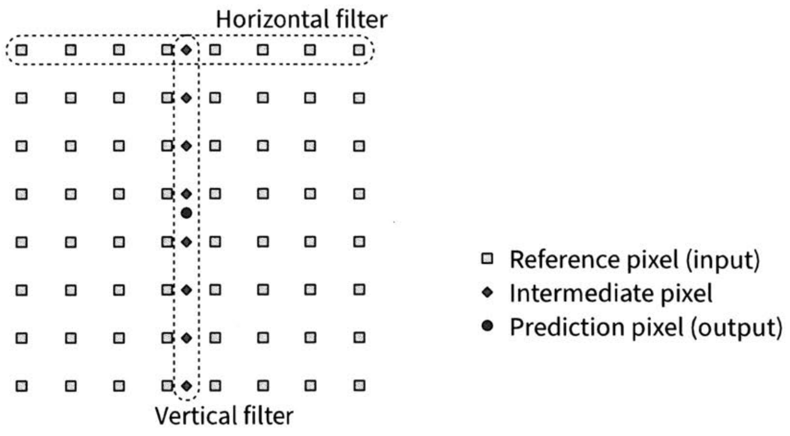


Figure 2-5: Horizontal and vertical filter computation for HEVC inter-prediction of luma pixels

imum parallelism. However, a naive implementation that fetches reference pixels and does the FIR computation for each pixel separately is very expensive. For each pixel output, it would require fetching an 8×8 block of reference pixels and computing 72 multiplications. The FIR filter structure and large PU sizes provide several opportunities for data and computation reuse:

1. Reuse reference pixel input across horizontal filters
2. Reuse intermediate pixels in a vertical filter
3. Computation reuse in filter using multiple constant multiplication (MCM) [21]

Efficiently exploiting reference pixel reuse reduces the cost of memory access to the reference frame buffer. Reusing intermediate pixels and applying MCM techniques help with reducing the cost of computation per pixel. The downside of increasing reuse is that it requires more local memory within the inter-prediction module to store reference pixels and intermediate pixels. Achieving a good trade-off between these costs is the main challenge in the design of the inter-prediction module.

In this work, we designed a 4 pixel/cycle architecture shown in Figure 2-6. This design uses two independent uni-prediction modules whose results are averaged at the end. The following describes the design of a uni-prediction module. The key features of the uni-prediction architecture are:

1. The PU is processed in vertical stripes that are 4 pixels wide. The vertical order allows better reuse in the vertical direction than the horizontal direction. This is desirable because the

reference pixels to be reused horizontally are 8 bit values while the intermediate pixels to be reused vertically are 16 bit values.

2. The horizontal filter is designed as a purely combinational circuit that takes 11 reference pixels as input and outputs 4 filtered pixels. This allows reuse of reference pixel input, i.e. instead of reading 8 input pixels per output pixel, we read 11 input pixels per 4 output pixels. Also, the horizontal filter can be optimized with MCM.
3. The 4 outputs of the horizontal filter are sent to 4 separate vertical filters. The traditional 1 pixel/cycle FIR filter architectures shown in Figure 2-7 can be used for the vertical filter. The design in Figure 2-7 (a) optimizes for register size by using registers of lower bitwidth on the input. The design in Figure 2-7 (b) optimizes for multiplier size as it allows for MCM. The first design was used in this chip.

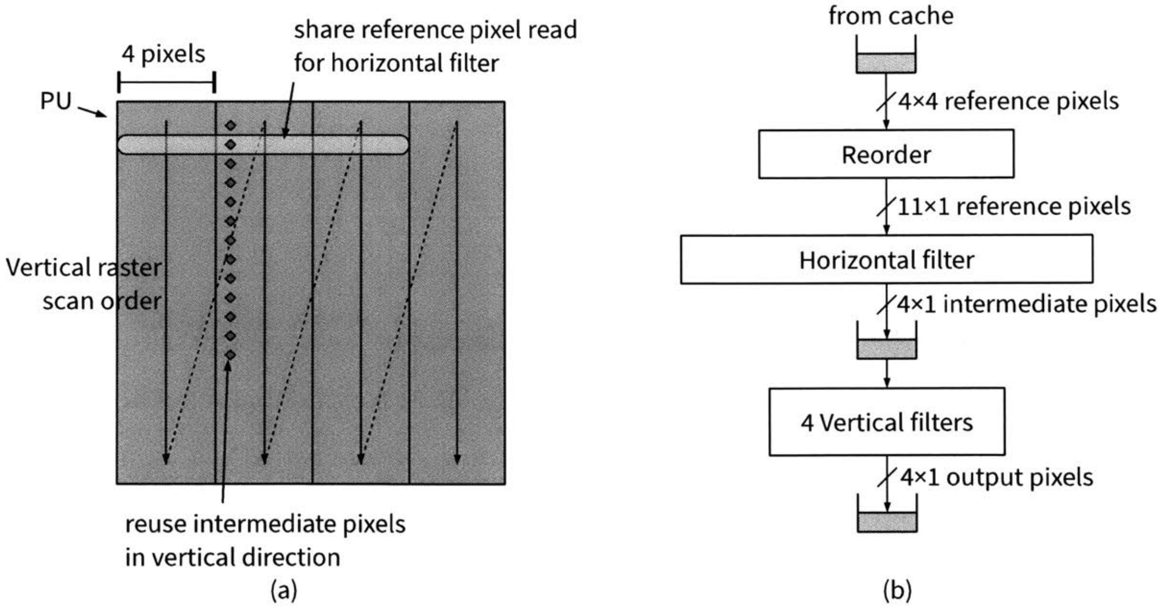


Figure 2-6: (a) Processing order of inter-prediction showing memory and computation reuse. (b) Pipeline architecture for inter prediction

This architecture does not fully exploit the reuse of reference pixels. When processing a vertical stripe, it is possible to reuse 4 reference pixels per row from the neighboring vertical stripe. We did not make use of this opportunity in the interest of a simpler design. Doing so would require 71×4 bytes of local storage as there are 71 rows in a 64×64 PU.

The complete architecture of the inter-prediction engine is shown in Figure 2-8. The inter-

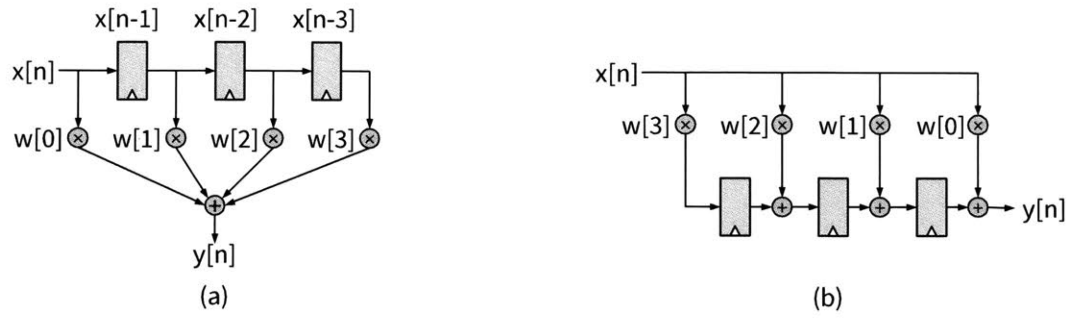


Figure 2-7: FIR filter architectures for 1 pixel/cycle throughput. (a) optimizes for register area while (b) optimizes for multiplier area

prediction engine was synthesized at 100 MHz in 40nm LP technology in which it uses 71 kgates of area, with the two uni-prediction modules taking up 34 kgates each and the remaining in the final average and input/output FIFOs. Within each uni-prediction, we have a reorder module (5 kgates), horizontal filter (8 kgates) and vertical filter (17 kgates). The throughput depends on integer versus fractional motion vector, PU size and cache hit rate. The typical throughput is sufficient for 1080p 60fps video decoding.

2.1.3 Micro-Parallelism in Intra-Prediction

Intra-prediction uses previously decoded neighboring pixels to generate a prediction. As shown in Figure 2-9, HEVC uses three forms of intra-prediction:

1. DC: The average of the neighboring pixels is used as the prediction value for all pixels.
2. Planar: Pixel values are interpolated from the edge.
3. Angular: Pixel values are copied along gradient lines from the edge. HEVC uses 32 angular modes. If the gradient line intersects the edge at a non-integer position, the sub-pixel value is interpolated from the integer neighbors with a bilinear filter.

The intra-prediction mode is specified at a PU granularity which can be squares from 4×4 to 32×32. However, the prediction is performed at a TU granularity, which is always smaller than the PU granularity, for more accurate prediction. The processing order of TUs must follow the Z-scan order shown in Figure 2-10. This order determines which neighboring pixels are to be used for the current TU based on their availability.

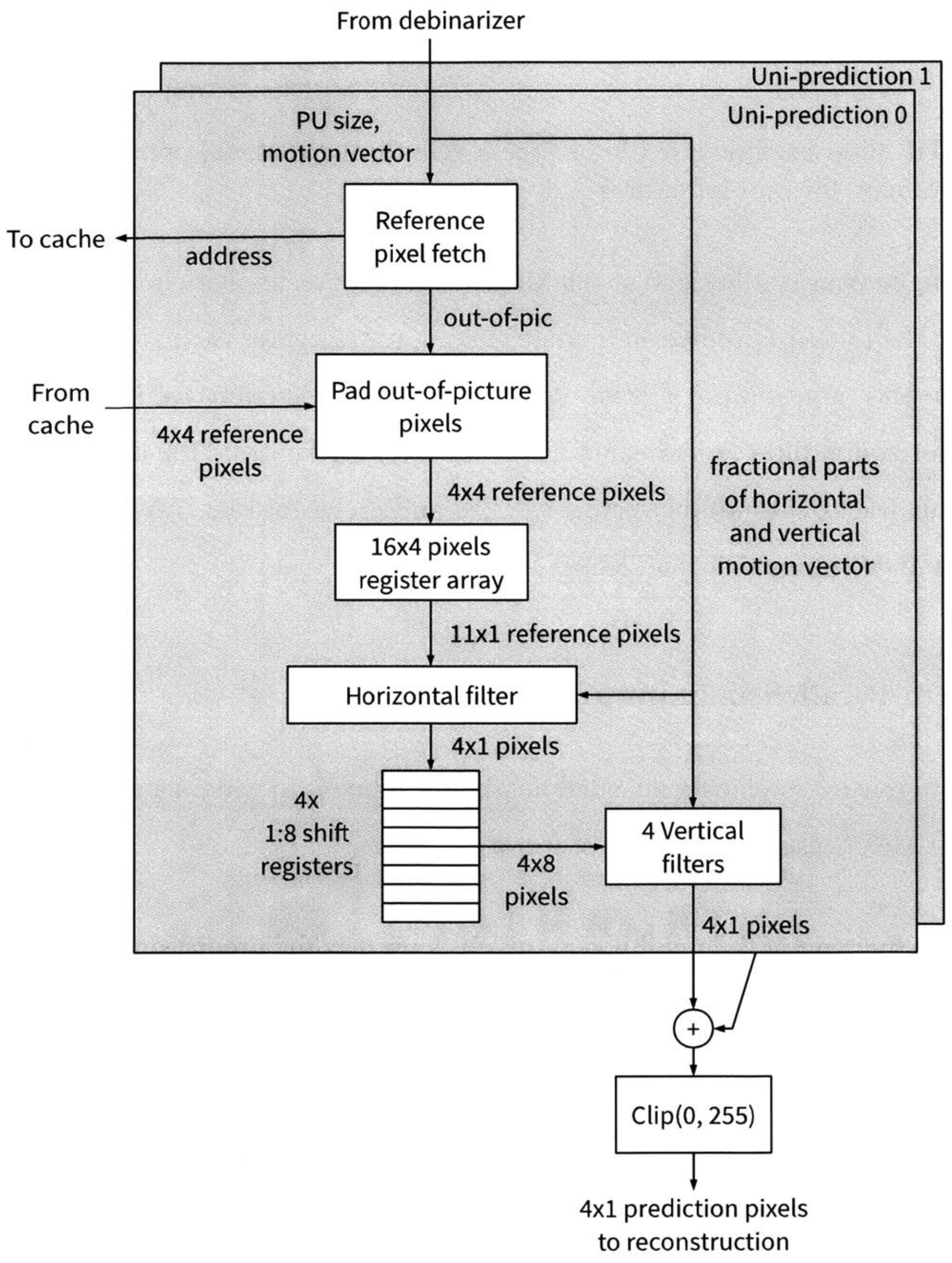


Figure 2-8: Architecture of inter-prediction engine showing two uni-prediction engines

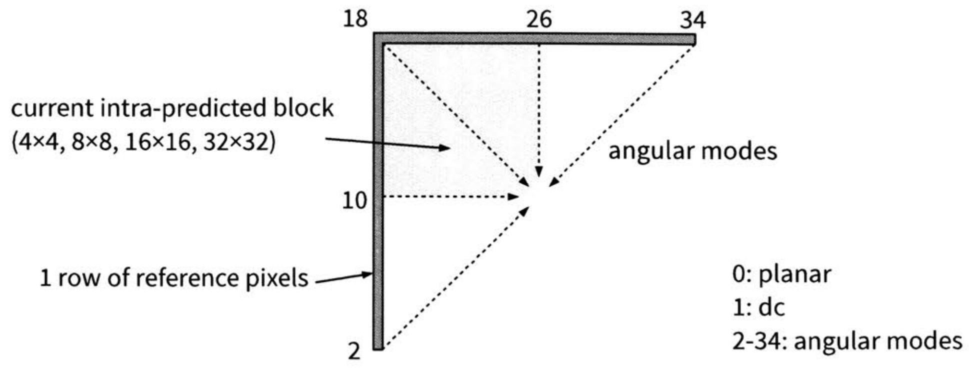


Figure 2-9: Block sizes, reference pixels and intra modes for HEVC intra-prediction

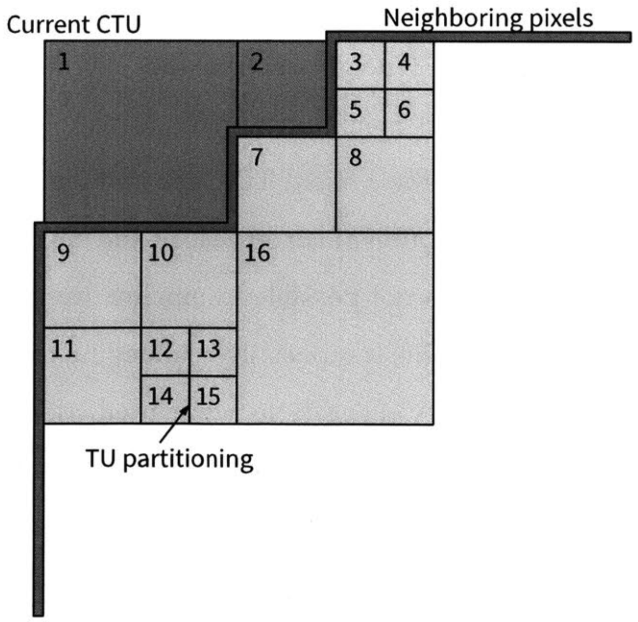


Figure 2-10: Z-scan order of processing for intra TUs for an example TU partitioning. The relevant neighboring pixels after the first two TUs have been processed are shown.

The intra-prediction process is as follows:

1. Reference pixel preparation stage:
    1. Determine available of neighboring pixels. Pixels may be unavailable as they are out of the frame, or not yet decoded.
    2. Fill in unavailable pixels. If all pixels are unavailable, like at the start of a frame, fill in 128 for all pixels. Else, use the nearest available pixel.
    3. Pre-filter the neighboring pixels with a 3-tap FIR filter to generate reference pixels.
    4. Extend reference pixels for angular prediction.
2. Perform prediction using the reference pixels according to the intra mode.

The main challenges for intra-prediction architecture are:

1. Managing neighboring pixel dependencies requires an hierarchy of memories from large SRAM-based line buffers for top neighbors from the row of CTUs above the current CTU, to SRAM-based local buffers for left neighbors from the CTU to the left, to SRAM or register based local buffer for top and left neighbor within a CTU, to even smaller register-based memory for reference pixels for the current TU. Such an hierarchy can be seen in [13].
2. Tight feedback in the intra-prediction of current block and next block makes it challenging to meet the cycle budget, as it is not possible to pipeline the reference pixel preparation stage and the prediction stage. This is especially challenging for smaller transforms as reference pixel preparation has $O(N)$ operations while prediction has $O(N^2)$, which causes the preparation stage to take up a significant portion of the cycle budget in smaller blocks.

Figure 2-11 shows the architecture designed for this chip. The main features of this architecture are:

1. Local buffers for left CTU and top/left pixels within current CTU are merged into a single Local Neighbor SRAM buffer. The address of a pixel into the buffer is (x - y) % SRAM size where x and y are pixel coordinates in the frame. The (x - y) operation allows the SRAM

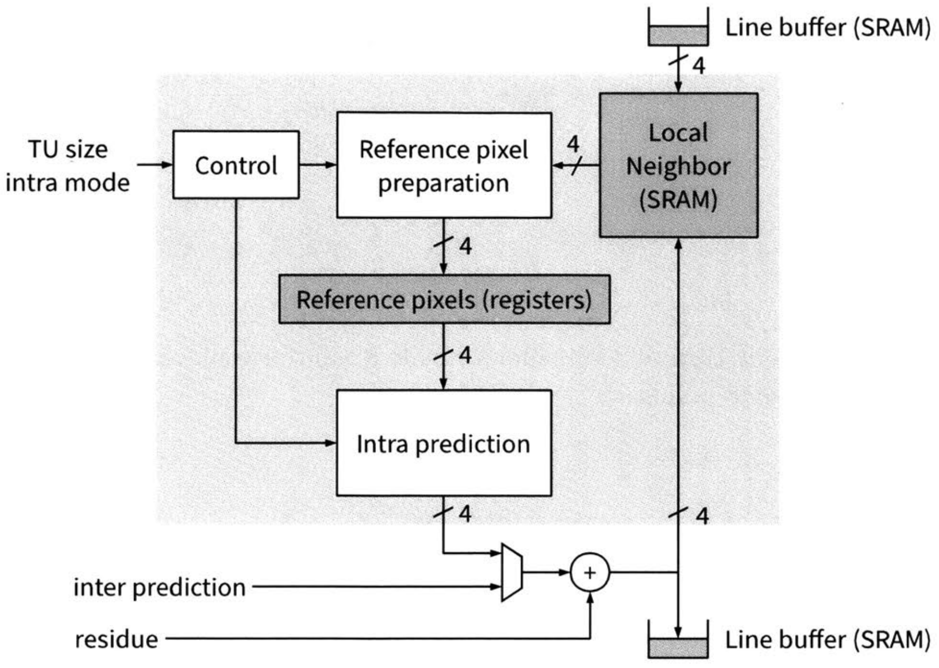
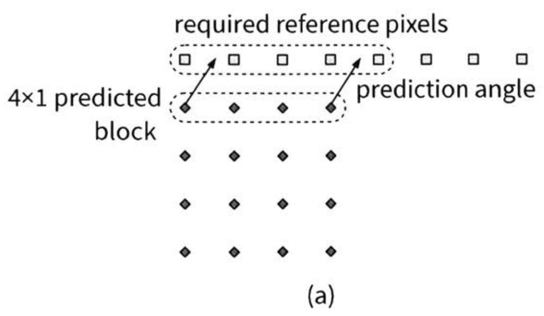


Figure 2-11: Architecture for intra-prediction showing memory hierarchy

index to follow the shape of the neighboring pixel boundary for the z-scan order shown in Figure 2-10. SRAM size of 512 pixels is sufficient to guarantee no aliasing due to the modulo operation. This technique minimizes data movement in the local buffers.

2. A 4 pixel/cycle reference pixel preparation module is designed to meet the cycle budget for smaller blocks.
3. A 4 pixel/cycle intra-prediction module is designed. We observed that the read pattern in the reference pixel buffer can be simplified if 2×2 pixel blocks are processed instead of a 4×1 block. For a prediction 4×1 block, we need to fetch up to 5 reference pixels but a 2×2 block needs only 4 pixels as shown in Figure 2-12. This enables a simpler design of the reference pixel buffer using four 17-pixel register arrays.

The intra-prediction block uses 14 kgates of logic and 4096 SRAM bits at 100 MHz. The minimum throughput is 2 pixel/cycle which is sufficient for 1080p 60fps video decoding.



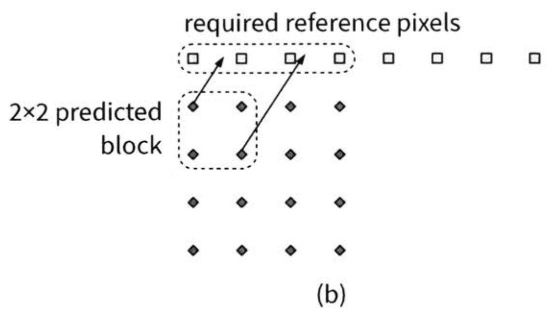


Figure 2-12: (a) Intra prediction of a 4×1 block needs 5 reference pixels (b) Intra prediction of a 2×2 block needs 4 reference pixels

2.2 Macro-Parallelism in Decoder Backend

The backend domain of the chip contains 4 decoder cores (Dec core 1-4). Each Dec core processes a row of Coding Tree Units (CTU) in the frame as shown in Figure 2-13. Due to data dependencies between CTU rows, Dec core N lags behind Dec core N+1 by two CTUs. This means that there are ramp-up and ramp-down times during with not all Dec cores are fully running. Another reason for inefficient utilization of all cores is that the frame height in CTUs is not a multiple of number of Dec cores. As a result, the decoder exhibits sub-linear speed-up with number of cores as shown in Figure 2-14.

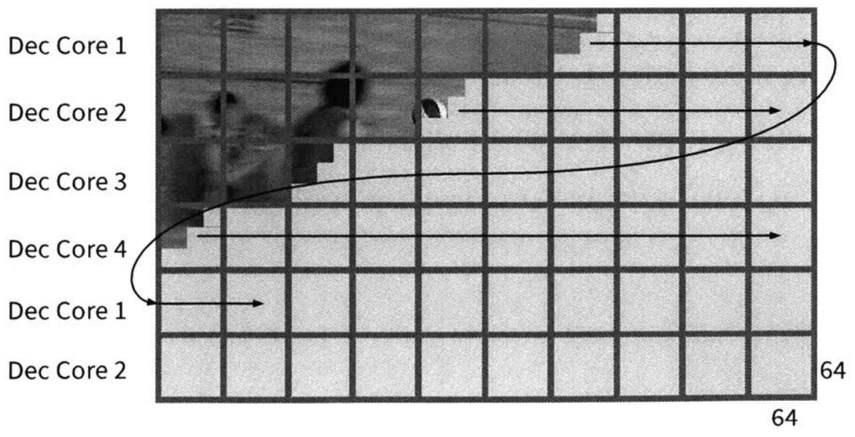


Figure 2-13: Parallel processing of video frame by 4 Dec Cores

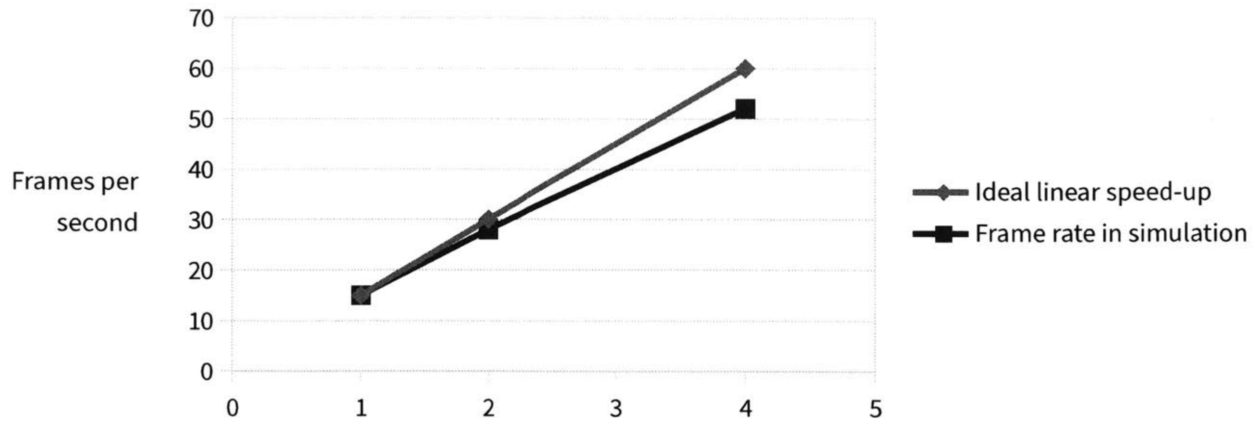


Figure 2-14: Frame rate for decoding 1920x1080 video with parallel decoder cores as observed in RTL simulation. Sub-linear speed-up is observed when more decoder cores are added

2.2.1 Line Buffer Design for Scalable and Reconfigurable Macro-Parallelism

In previous work [9], a single decoder core was used to decode the frame on a CTU-basis in horizontal raster-scan order. Data dependencies between CTU rows were managed by line buffers. The size of the line buffer is proportional to the width of the frame and is independent of number of decoder cores. However, when multiple decoders are used, using a single line buffer requires tracking the dependencies for all decoder cores:

1. Dec core N+1 must read a pixel from the line buffer only after Dec core N has written it.
2. Dec core N+1 must not read stale pixels from older CTU rows.
3. Dec core N must write new pixels to the line buffer only after Dec core N+1 is done with the old pixels in the same location.

In this chip, the Dec core is designed to have a regular access pattern on the line buffer irrespective of CU/PU/TU partitioning within a CTU. This allows the single line buffer to be replaced by FIFOs between adjacent cores. The FIFO empty/full logic guarantees that the above three conditions for dependency tracking are met. The total depth of the FIFOs is proportional to the width of the frame and SRAMs are used as the memory element in these FIFOs. This design is easily scalable with number of decoder cores. The difference between the two line buffer systems is shown in Figure 2-15.

The depths of the individual line buffers FIFOs can be designed to achieve appropriate scheduling

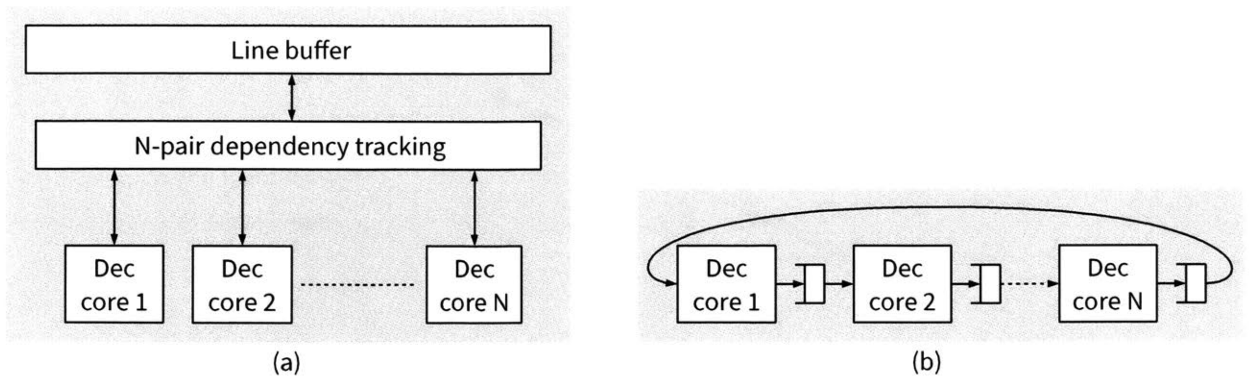


Figure 2-15: (a) Shared line buffer requires complex dependency tracking for all pairs of adjacent decoders. (b) Designing the decoder cores to work with FIFO-based line buffers results in a more scalable design.

of the decoder cores. The total depth of the FIFOs must be proportional to the frame width. If the FIFO depth is counted in units of CTUs, the total depth for 1080p video decoder with 64×64 CTUs is 30.

The closest that the decoder cores can operate to each other is 2 CTUs apart as shown in Figure 2-16 (a). The configuration of the line buffer depths that can achieve this scheduling is shown in Figure 2-16 (b). The configuration used in this chip uses line buffers with equal depths resulting in the schedule shown in Figure 2-16 (c). The configuration was chosen for its regularity to make floorplanning for chip layout easier.

The choice of line-buffer configuration affects the throughput of the decoder:

1. The first configuration has higher possibility of data reuse from frame buffer with a sufficiently large cache. With the 4 decoders only 2 CTUs apart at any point of time, read requests to the reference frame buffer by inter-prediction in the 4 decoders are likely to be spatially closer to each other in the frame. This improves hitrate in the cache as seen from the results in Table 2.2.
2. The second configuration has more flexibility in dealing with variable processing times for CTUs. In this case, the 4 decoders are operating as much as 8 CTUs apart. This allows for small variations in processing times for CTUs to get averaged out. In the first configuration, the 4 decoders are more tightly coupled. They operate in a lock-step fashion which means that all 4 decoders can move to their next CTU only when the slowest decoder is finished.

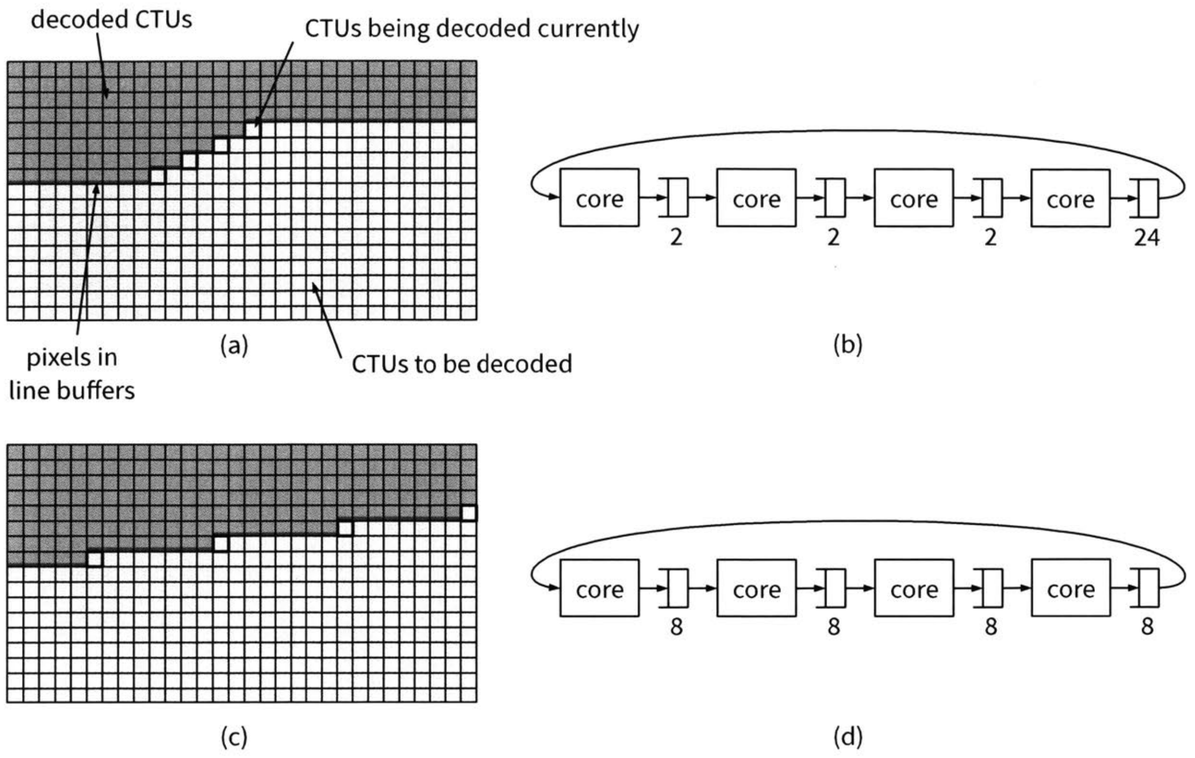


Figure 2-16: Two configurations for line buffers and the corresponding schedules for the decoder cores. The number under each line buffer FIFO is depth of the FIFO in units of 1 CTU.

Table 2.2: The effect of line-buffer configuration on decoder frame rate and cache hitrate.

Line buffer design	Cache hitrate	Frame rate
Figure 2-16 (b)	78.1%	48.0 fps
Figure 2-16 (d)	77.7%	50.5 fps

Table 2.2 shows the results of RTL simulations for the two line buffer configurations. As expected, the 2-CTU configuration has a higher cache hitrate. However, this improvement is not enough to offset the loss in flexibility of dealing with CTUs with variable processing time. As a result, the second configuration provides better frame rate. These simulations were done with a 49kB cache. Using a larger cache may turn the balance in favour of the first configuration. However, a larger cache will also have a correspondingly higher energy/access. So, that trade-off must also be taken into account when designing the linebuffer and cache configurations.

Chapter 3

Energy-Efficient Inverse Transform

This chapter describes the design of an energy-efficient inverse transform module. We describe the challenges in designing a hardware for HEVC's inverse transform methods brought about by the large and varied transform sizes. We show how data-gating can be used to reduce the energy/pixel for smaller transforms while maintaining a shared 1-D transform logic for small and large transforms. We show how the sparsity in transform coefficients can be used effectively to perform large inverse transform operations with low energy/pixel cost without increasing complexity of the design. We compare our approach to other approaches for sparse operations used in video coding and neural network applications. The main result of this work is that large transforms can be performed just as efficiently as smaller transforms which can enable future coding standards to use even larger transforms to achieve better coding gains without sacrificing on energy consumption.

HEVC inverse transform algorithm operates on square transform units (TU) of size 4×4, 8×8, 16×16 and 32×32. The transform coefficients in the input TUs first undergo a dequantization operation to reverse the quantization that was performed in the encoder. In the next step, a 2-D inverse transform is applied to obtain the residue pixels. This process is shown in Figure 3-1.

The 2-D inverse transform is separable into row and column transforms. HEVC uses inverse discrete cosine transforms (IDCT) for all TU sizes. 4×4 TUs corresponding to intra-predicted PUs with certain intra angle modes may also use inverse discrete sine transforms (IDST). The operations are performed with 16-bit integer precision for the transform coefficients and 8-bit

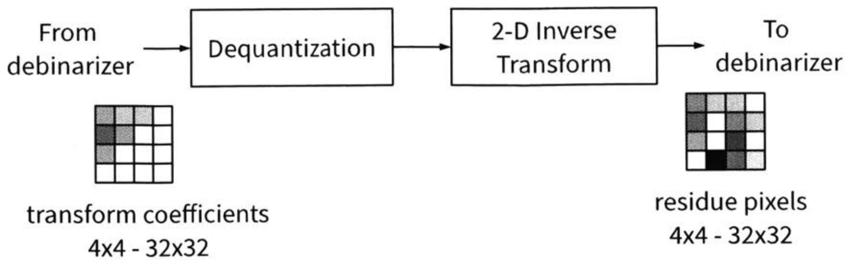


Figure 3-1: HEVC inverse transform algorithm

integer precision for the IDCT and IDST constants. After the multiply-and-accumulate for each transform output is complete, the result is scaled down to maintain the 16-bit precision. This ensures that the row and column transforms can use the same logic. The IDCT constants for the various transform sizes have been chosen by the standard to ensure that logic for the smaller transforms can be reused by larger transforms.

To summarize, HEVC uses 4×4 to 32×32 transforms with higher precision as compared to H.264/AVC's 4×4 and 8×8 transforms resulting in an increased hardware complexity. The main challenges for hardware implementation are:

1. Inverse transform logic: HEVC inverse transform matrices use 8-b precision constants as compared with 5-b constants for AVC. The constant multiplications can be implemented as shift-and-adds, where 8-b constants would need at most four adds while the 5-b constants need at most two. Further, the largest 1-D transform in HEVC is the 32-point IDCT, compared with the 8-point IDCT in AVC. These two factors result in an 8-fold complexity increase in the transform logic. Some of this complexity increase is alleviated by the fact that the 32-point IDCT can be recursively decomposed into smaller IDCTs in a partial butterfly structure. Further, the area of the IDCT logic can be reduced using multiple constant multiplication as shown in [9]. The high energy cost is addressed in this work.
2. Transpose memory: In AVC decoders, transpose memory for inverse transform is usually implemented as a register array with multiplexers for column-write and row-read. In HEVC, however, a 32×32 transpose memory using a register array takes about 125 kgates. SRAM-based transpose memory can help reduce area and power but the SRAM is less flexible than registers.

In this work, we develop an energy and area-efficient VLSI architecture of an HEVC-compliant inverse transform and dequantization engine. We use data-gating in the 1-D Inverse Discrete Cosine Transform engine to improve energy-efficiency for smaller transform sizes. We exploit sparsity in the input coefficients through zero-column skipping for improved throughput and energy efficiency. Future video coding standards are likely to use even larger transform sizes [19]. This chapter shows that the presented techniques scale well with transform size and presents directions for how the techniques may be augmented to perform even better.

3.1 High-throughput Pipelining Scheme for All TU Sizes

In general, two high-level architectures are possible for a 2 pixel/cycle inverse transform [22]. The first one, shown in Figure 3-2 uses separate blocks for row and column transforms. Each one has a throughput of 2 pixel/cycle and operates concurrently. The dependency between the row and column transforms (all columns of the TU must be processed before the row transform) means that the two must process different TUs concurrently. The two TUs could take different number of cycles thus causing pipeline stalls. For example, if a 4×4 TU follows a 8×8 TU, the column transform will stall after processing the 4×4 TU as it waits for the row transform to finish the 8×8 TU.

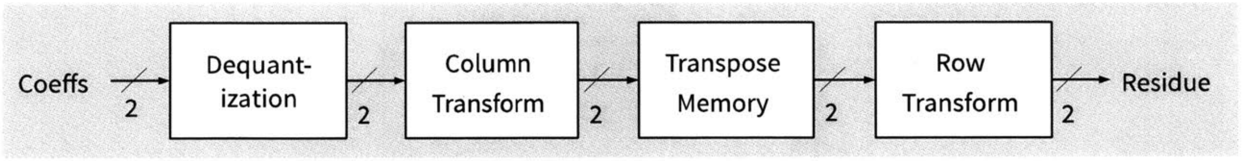


Figure 3-2: Transform architecture with separate transform logic

With these considerations, the second architecture, shown in Figure 3-3 is preferred. This uses a single transform block capable of 4 pixel/cycle for both row and column transform. The block works on a single TU at a time, processing all the columns first and then the rows. Hence, the transpose memory needs to hold only one TU and can be implemented with a single-port SRAM since row and column transforms do not occur concurrently.

The complete architecture of the inverse transform and dequantization engine is shown in Figure 3-4. The partial 1-D transform block includes the 4 pixel/cycle IDCT and IDST blocks. The

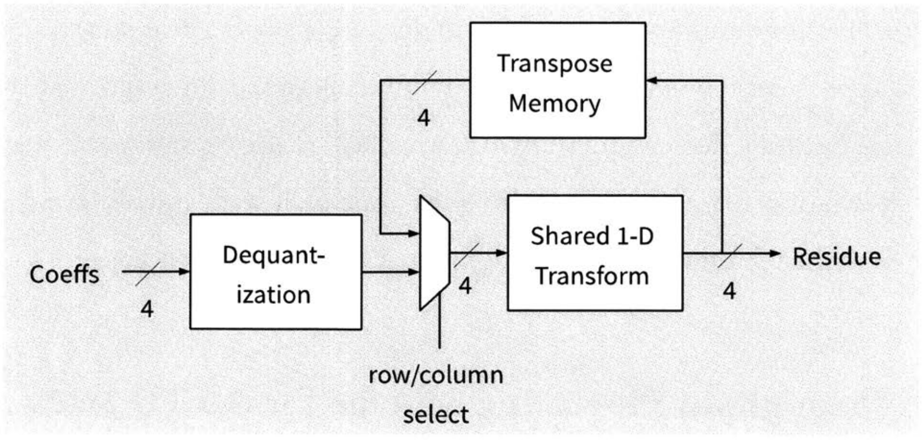


Figure 3-3: Transform architecture with shared transform logic

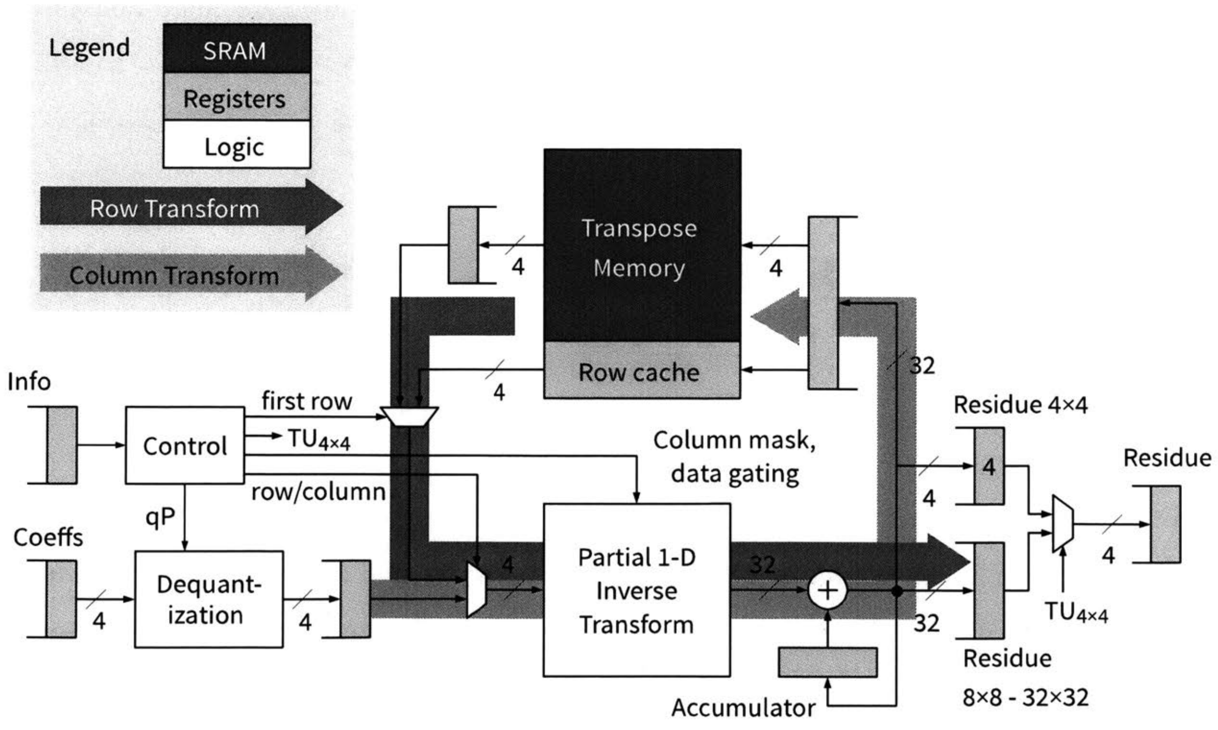


Figure 3-4: Block diagram of inverse transform and dequantization engine

transform coefficients and TU information (TU size, quantization parameter, luma/chroma) are read from the Coeffs and Info FIFOs respectively and the output is written to the Residue FIFO. Single-element FIFOs are used for pipelining.

3.2 Exploiting Sparsity With Zero-column Skip

An N×N 2-D inverse transform operation in HEVC can be represented as:

$$Y = A^T \cdot (X \cdot A)$$

where the variables are:

Variable	Size	Precision	Description
A	N×N	7-bit	N-point 1D IDCT or IDST constants
X	N×N	16-bit	Transform coefficients in spatial frequency domain
Y	N×N	9-bit	Residual pixels

After the first matrix multiplication ($X \cdot A$, which corresponds to 1-D column transforms), the precision grows to 16 + 7 + $\log_2$(N) = 29-bit for N=32. This intermediate result is scaled down to 16-bit precision which allows the next matrix multiplication for 1-D row transforms to use the same hardware. The intermediate results are stored in a transpose memory that gets written column-wise and read row-wise.

The coefficient matrix X is typically sparse as most of the residual energy is concentrated in small spatial frequencies. Figure 3-5 shows a histogram of TUs binned based on the density of non-zero coefficients. Most of the TUs have less than 20% non-zero coefficients. We also notice different trends based on TU sizes. The low density bins contain a greater fraction of 8×8 and larger TUs. The high density bins are dominated by smaller TUs, mainly 4×4. In short, larger TUs are more sparse than the smaller TUs and we exploit this observation to reduce the energy cost of processing larger TUs.

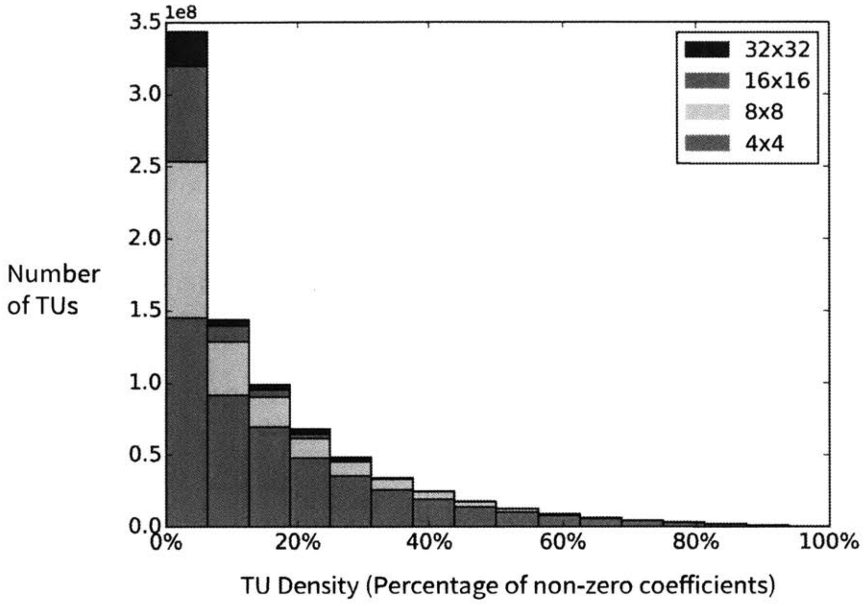


Figure 3-5: Histogram of TUs by percentage of non-zero coefficients. Most TUs have less than 20% density and larger TUs have lower density compared to smaller TUs.

Sparse matrix multiplication has been studied extensively in the software domain. In the hardware domain, a general implementation of sparse matrix × sparse vector multiplication can be found in [23]. The matrix is stored column-wise in compressed sparse column (CSC) format. Each non-zero elements of the sparse vector is multiplied with a column in the matrix and the resulting columns are accumulated together to generate the output column. This process is parallelized over multiple processing engines (PEs). The elements of the sparse vector are broadcast to all PEs, and each PE is given a section of the matrix column, to generate the corresponding section in the output column.

For HEVC, [24] presents an IDCT pruning technique with reduced software complexity. Among hardware video decoders, [25] presented a design for MPEG-2 8×8 IDCT, and more recently [26] presented a design for HEVC IDCT. The essential idea is to perform the N-point 1-D IDCT one input coefficient at a time while skipping zero coefficients.

Exploiting sparsity in matrix operations poses two main challenges:

1. **Detecting non-zero elements:** In the general implementation [23], a Leading Nzero Detect module is employed to find the indices of the non-zero elements. In video coding, the coefficients are stored in the bitstream in a compressed format that makes heavy use of

the sparsity. In HEVC, a 4×4 subblock of coefficients is stored as a 16-bit mask called significance map which denotes the positions of the non-zero coefficients followed by only the non-zero values.

2. **Load balancing**: In the general implementation, the PEs all need different number of cycles to produce their results depending on the sparsity of the inputs. Load balancing is achieved using input FIFOs before each PE. With a sufficiently deep FIFO, variation in the workload across PEs gets smoothed out. In the video decoders described before, the 1-D N-point transform takes as many cycles as the number of non-zero coefficients, but writing the result to transpose memory takes a constant number of cycles based on the IO width of the memory (see Figure 3-6 a). To avoid this imbalance, these decoders use a wide-IO transpose memory that can write the entire result column in one cycle as shown in Figure 3-6 b. This has implications on the design of the transpose memory that will be discussed later.

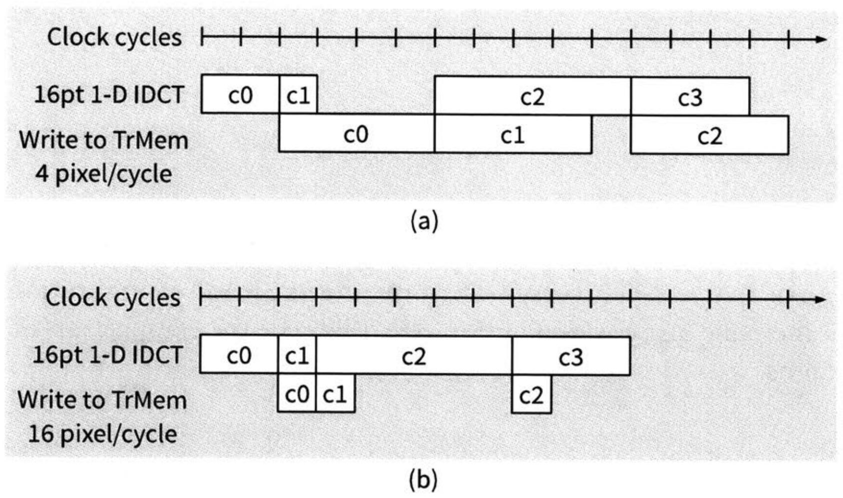


Figure 3-6: (a) Example of load imbalance between variable-time sparse IDCT and write to transpose memory (TrMem). (b) Load imbalance avoided by wide-IO transpose memory at the cost of memory area.

Exploiting sparsity for 2-D IDCT without load imbalance requires a transpose memory that is capable of writing an entire column per cycle. In [25], a register-based 8×8 transpose memory is used to achieve this high throughput for MPEG-2 decoding. This does not scale well for HEVC's 32×32 transform sizes, requiring 125 kgates of logic.

In [26], the transpose memory is built out of 32 banks of SRAM each containing 32 entries. By mapping a 32×32 matrix in a striped fashion, one can write a 32-entry column into the memory and read a 32-entry row from the memory. For such an SRAM configuration with few rows, the SRAM area and access energy are dominated by row/column circuits and not the bitcells. The area of this SRAM is comparable to the register-based array. For comparison, we propose using a transpose memory with only 4 pixel/cycle throughput that can be implemented with 4 banks of 256 entries, which has 5.5× lower area and 11× lower access energy as compared to the configuration in [26].

The high throughput transpose memory allows [25], [26] to skip all zero coefficients. They can also process coefficients in the same order as they are parsed by the entropy decoder. This eliminates a scan reorder buffer between the entropy decoder and inverse transform modules. In all, the choice of microarchitecture for exploiting sparsity affects the configuration of three SRAM buffers as shown in Figure 3-7.

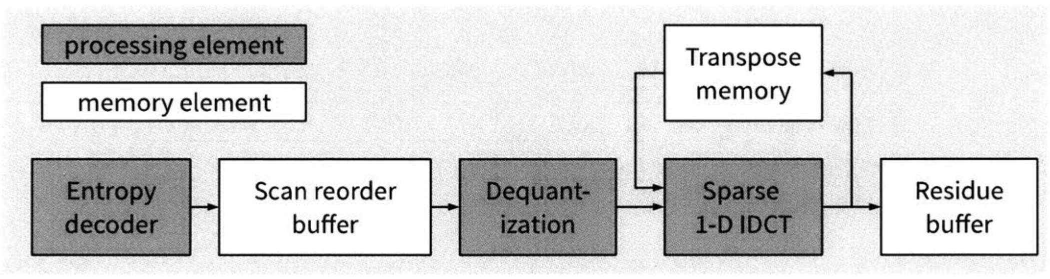


Figure 3-7: Design of sparse 1-D inverse transform affects SRAM configurations for scan reorder buffer, transpose memory and residue buffer. See Table 3.2 for comparison of SRAM configurations for two designs.

The proposed design solves the load imbalance problem by perfectly pipelining a 4 pixel/cycle 1-D IDCT with a 4 pixel/cycle transpose memory. This means that only columns with all zero coefficients can be skipped. If a column has even one non-zero coefficient, the transpose memory will take N/4 cycles to write the N-element column and the IDCT module must stall until then. This is shown in the Figure 3-8.

To support zero-column skipping, the dequantization and inverse transform modules need the additional input of an N-bit mask ($N = 4,8,16,32$) for an $N \times N$ transform denoting the positions of the non-zero columns. If this mask were to be generated by simply going through all the transform coefficients in order, the extra cycles required to generate the mask would negate the throughput savings achieved by using the mask in the following dequantization and inverse

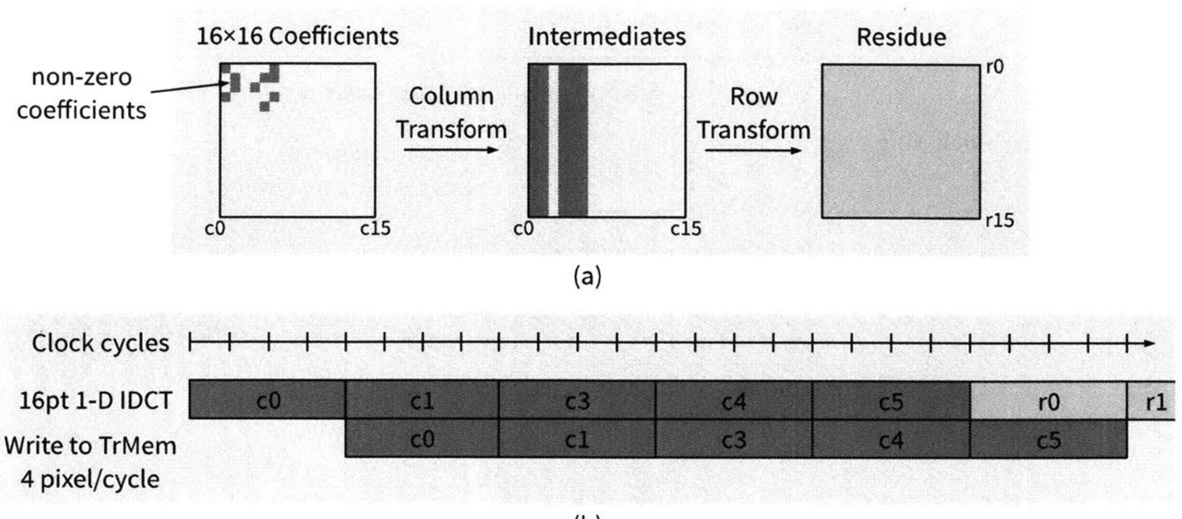


Figure 3-8: (a) Operation and (b) timeline of proposed design for zero column skipping on a 16x16 TU

transform modules. Instead, we generate the mask as the transform coeffients are parsed from the stream of binary symbols by the debinarizer. The binary format of transform coefficients makes heavy use of the sparsity to achieve better video compression. Here, we exploit this feature of HEVC to generate the column mask. The following syntax elements are used to generate the non-zero column mask:

1. **last_sig_coeff_x**, **last_sig_coeff_y**: These denote the position of last significant (i.e. non-zero) coefficient in the TU in the scan order chosen for the TU.
2. **coded_sub_block_flag**: This denotes whether a 4×4 sub-block in a TU has any significant coefficients.
3. **sig_coeff_flag**: This denotes whether a coefficient in a 4×4 sub-block is significant.

Figure 3-9 shows an example of the relevant syntax elements for an 8×8 TU. A 32-bit register maintains the non-zero column mask for a TU and gets updates as these syntax elements get parsed by the debinarizer. Once the parsing is finished, the column masks and the transform coeffients are passed on to dequantization and inverse transform.

On typical video sequences, zero-column skipping reduces cycle-count by 27% to 66%. TUs with larger sizes and higher quantization benefit more from zero-column skipping since they have

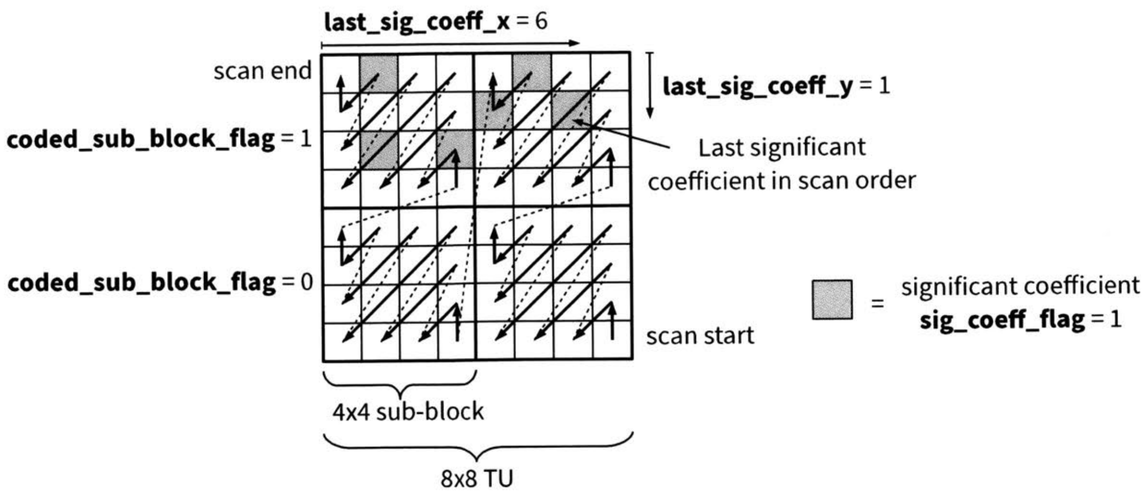


Figure 3-9: Example of syntax elements used to generate non-zero column mask for an 8×8 TU

a higher proportion of all-zero columns. Zero-column skipping also improves energy/pixel by avoiding any switching to zero. For example, when processing 2000 TUs (173360 pixels with a mixture of all TU sizes) from the ParkScene test sequence at QP 32, zero-column skipping reduces the cycle count by 38% and energy/pixel by 29%.

It should be noted that this work can guarantee a minimum throughput of 2 pixel/cycle on the 2-D IDCT, while achieving 3.26 pixel/cycle on typical video sequences. Table 3.2 provides a comparison between proposed design and [26].

Table 3.2: Comparison of two strategies for exploiting sparsity in HEVC inverse transform. SRAM configuration format is number of banks × number of entries × data IO width.

	[26]	Proposed design
Method	Zero coefficient skipping	Zero column skipping
Scan reorder buffer	0	1×512×64
Transpose memory	32×32×29	4×256×16
Residue buffer	1×64×288	1×512×36
SRAM bits	48 kbit	68 kbit
SRAM area (normalized)	2.1	1
Logic area	101 kgate	126 kgate
Minimum throughput	0.5 pixel/cycle	2.0 pixel/cycle
Typical throughput	12.44 pixel/cycle	3.26 pixel/cycle

3.3 Data-gating in 1-D IDCT

Our 1-D IDCT is based on the design from [9] which implements a single shared 4 pixel/cycle transform for 4-pt to 32-pt IDCT and uses multiple constant multiplication (MCM) for an area-efficient implementation. However, the sharing causes some spurious switching activity that reduces the power-efficiency, especially for smaller transforms. For example, when computing a 4-pt IDCT, due to the shared architecture shown in Figure 3-10, all 8 outputs of IDCT8 toggle even though only the first 4 outputs need to change. This extra switching cascades down to the outputs of the larger transforms through the subtraction blocks. In this design, all the subtraction blocks are data-gated as shown in Figure 3-11 so that only the necessary outputs have any switching activity for smaller transforms. Similarly, the 32 accumulators are explicitly clock-gated for smaller transforms to reduce clock switching power.

The data-gates require some energy for switching the gate inputs. But this switching can be amortized over many cycles as the gate inputs change only when the transform size changes. For example, when going from 8×8 to 16×16 transform, only gate8 changes from 0 to 1 as shown

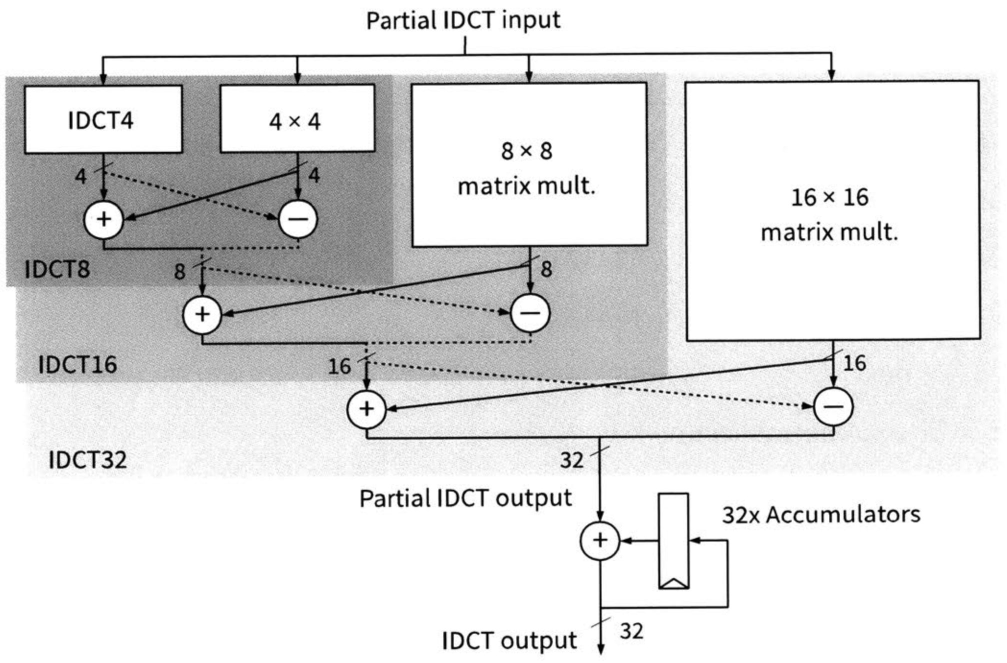


Figure 3-10: Shared 4-pt to 32-pt 1-D IDCT. Dotted lines denote the paths that need data-gating to reduce spurious switching when computing smaller transforms.

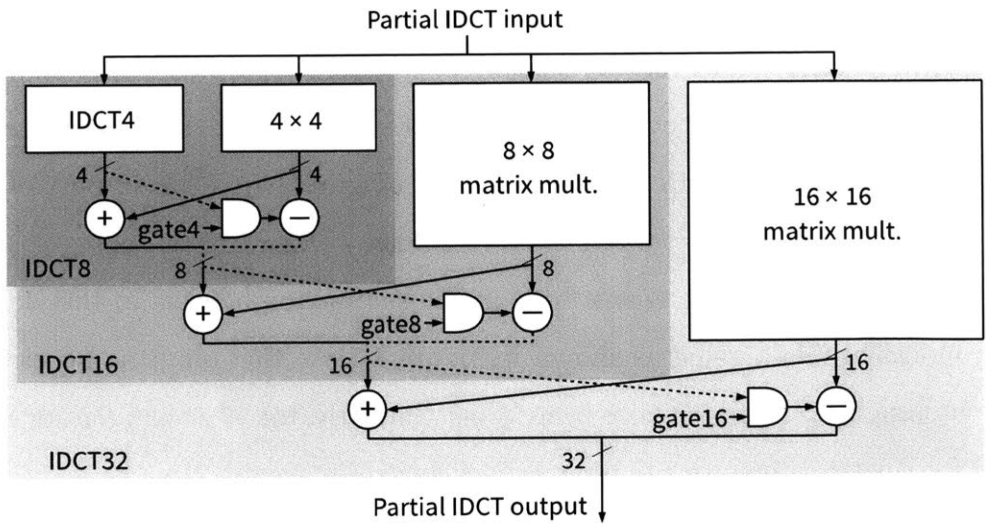


Figure 3-11: Shared 4-pt to 32-pt 1-D IDCT with data-gating.

in Table 3.3. The worst case TU sequence that causes fastest switching in gate inputs is 4×4, 8×8, 4×4, 8×8, ... in which case, gate4 has switching activity factor of 1/40. As a result, this overhead is negligible.

Table 3.3: Gate inputs (shown in Figure 3-11) for all TU sizes.

TU size	gate4	gate8	gate16
4×4	0	0	0
8×8	1	0	0
16×16	1	1	0
32×32	1	1	1

The benefits of gating are seen in Table 3.4 which compares the post-layout power for different transform sizes with and without gating. The gating circuit adds an overhead of 4.7 kgate (4%) to the logic area while reducing energy/pixel by 17%. As expected, the smaller transforms benefit more from data-gating.

Table 3.4: Energy saving in inverse transform engine using data-gating. Energy measured in pJ/pixel.

TU size	No gating energy	Gating energy	Energy saving
4×4	17.8	11.2	37%
8×8	32.4	22.2	31%
16×16	42.1	38.1	9%
32×32	50.9	57.1	-12%

We notice that the largest transform loses some energy efficiency. This is due to the added load of the AND data-gates. This suggests for every level of data-gates (gate4 to gate16), there is a trade-off with respect to energy savings for smaller transforms and added load for larger transforms. A closer analysis of this trade-off with energy simulations might yield a better configuration of data gates (e.g. removing gate4 as it only saves energy for 4×4 transforms but adds load for all other transforms).

Data-gating is expected to be more significant in future standards which are likely to add larger transform sizes to handle higher video resolutions. The shared 1-D transform architecture is well suited to support a larger variety of transforms in an area-efficient manner, and data-gating complements it by minimizing the switching energy cost for lower transforms.

3.4 Summary of Contributions

In this section, we presented the hardware design of an HEVC-compliant inverse transform engine capable of processing 4K Ultra-HD 30 frames/sec video in 40 nm technology. Zero-column skipping reduces cycle-count by 27%-66% over the worst case. Data and explicit clock-gating improves the energy efficiency of the shared transform logic. This design takes 126 kgates of logic and consumes 7.8 mW of power (or 11.9 pJ/pixel). The proposed techniques are summarized in Table 3.5.

Table 3.5: Summary of proposed techniques

Designs	Logic area (kgates)	Energy (pJ/pixel)	Throughput (pixel/cycle)
Base design [27]	118.5	20.94	2.00
Gating	123.1	17.62	2.00
Zero-column skip	121.6	14.79	3.26
Complete design [9]	125.8	11.92	3.26

The energy consumption of the transform engine depends on the statistics of input data. In typical video bitstreams, larger transforms have very sparse inputs because large transforms used for compressing pixel regions with less detail. The proposed transform engine is able to exploit this fact and perform large transforms just as efficiently as smaller transforms. Figure 3-12 shows this trend. Energy/pixel increases with increasing transform sizes if sparsity is artifically kept constant at 90%. But, with real video data, all transform sizes have similar energy/pixel as the higher computation cost of larger transforms is compensated for by sparsity in input data.

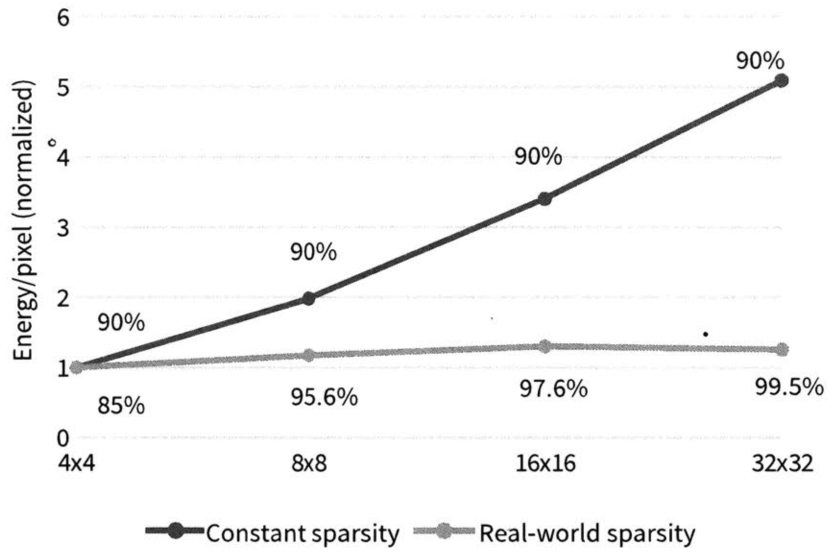


Figure 3-12: Energy/pixel cost of transforms at constant sparsity and real-world sparsity

3.5 Future Directions

Future standards are expected to use larger transform sizes to better compress features in very high resolution images. [19] presents coding gain results for the next coding standard after HEVC being developed the Joint Video Exploration Team. Among the many changes, is the use of 256×256 Coding Tree Units (CTUs) and 64×64 Transform Units (TUs) compared to 64×64 CTUs and 32×32 TUs in HEVC. All the changes have shown an overall 28% BD-Rate reduction compared to HEVC.

Inverse transform is different from the other coding tools such as inter-prediction, intra-prediction and loop filter in that its computational complexity per pixel increases with increasing transform size. Compared to that, inter-prediction, which is essentially a 2-D convolution, gets more efficient in terms of computation and memory accesses for larger block sizes due to boundary effects. As a result, it is expected that an area and energy-efficient inverse transform design will become even more important in future standards. The various aspects of this design remain relevant for larger transform sizes:

1. **1-D IDCT logic**: We should continue to use a shared multi-point 1-D IDCT logic for area efficiency. It might however be beneficial to separate out the smallest transforms to reduce switching energy in logic and memory at the cost of slightly increased area.

2. **Data gating:** The shared 1-D IDCT logic will continue to benefit from data gating to improve energy efficiency of small transforms. A closer analysis of the trade-offs in the energy efficiency of smaller and larger transforms might yield a better data gating configuration as explained earlier.
3. **Sparsity:** The higher sparsity in larger transforms is the main factor that keeps their energy efficiency competitive with smaller transforms. Zero-coefficient and zero-column skip techniques should be evaluated for their trade-offs to determine which works best for even larger transforms. The present design completely skips a TU with all zeros. It might be possible to also bypass the 2-D transform module for cases such as TUs with exactly 1 coefficient. For example, TUs with only a DC coefficient will have a constant residue that can be directly computed without going through column transform $\longrightarrow$ transpose memory $\longrightarrow$ row transform.
4. **Transpose memory:** As the SRAMs used in transpose memory are relatively small in terms of number of addresses, the area and access energy of the SRAMs depend more strongly on IO width than the number of bits. The present design has IO width proportional to the required throughput unlike other designs which need IO width proportional to transform size. So, the present transpose memory is expected to scale better for future standards.

Chapter 4

Energy-Efficient Use of eDRAM by Reducing Refresh Power

In this chip, we use eDRAM as main memory to eliminate the use of off-chip DRAM memory. In the hierarchy of memory technologies, eDRAM stands between SRAM and DRAM in terms of energy/access, density, maximum size, latency and bandwidth. For example, in 28 nm technology, eDRAM has 321× lower energy/access than DRAM [28]. Some of the other metrics are compared in the table below:

Table 4.1: Comparison of SRAM [29], [30], eDRAM [31]–[33] and DRAM [34], [35] memories in 32nm and smaller technology nodes

Memory technology	Density (Mb/mm^2)	Maximum size (MB)	Latency (ns)
SRAM	4.2	54	2.5
eDRAM	11.0	128	5.0
DRAM	56.8	1024	13.5

4.1 eDRAM as Main Memory

The chip entirely relies on eDRAM for the main memory. A total of 21 eDRAM macros, each 0.5 MB in size, are used for the following purposes:

1. Frame buffer: Two previous and one current frame are stored in a lossless compressed format in the frame buffer. To support the unlikely worst case that the lossless compression provides no data savings, the eDRAMs are provisioned to support fully uncompressed frames (3MB for 1080p frames). This requires 6 macros per frame. When compression provides data savings, the unused eDRAM macros are placed in deep-power-down mode to minimize their power consumption. A 3-way set-associative, 2× parallel cache of size 49kB is used to reduce the frame buffer bandwidth by 2.1×.
2. Address buffer: The lossless compression on the frame buffer necessitates the use of an address buffer to translate pixel location in frames to byte address in the frame buffer. A compressed format is used to store the addresses to reduce the number of macros needed for the address buffer from 5 to 1.
3. Co-located motion vector (MV) buffer: Parsing motion vectors for the current frame requires co-located motion vectors from previous frames. These take up 1 eDRAM macro.
4. Decoupling buffer: A 1 macro decoupling buffer is used to distribute data from the entropy decoder in the frontend to the 4 parallel decoder cores in the backend. The entropy decoding process happens in two stages: CABAC and Debinarizer. By splitting these stages into separate modules and moving the Debinarizer into the backend, this decoupling buffer is made to store partially decompressed binary symbols. This reduces the size of the decoupling buffer from 4 macros to 1, and bandwidth from 256 MB/s to 3.9 MB/s.

Of these, the frame buffer is the most prominent use of eDRAM taking up 18 macros out of the 21. The frame buffer also consumes 33% of the total chip power. In Chapter 4, we show that power consumption of the frame buffer is dominated by refresh power and propose techniques to minimize the amount of refresh needed.

This eDRAM-based frame buffer can store 2 previous frames. For fully compliant HEVC decoding, 16 frames are needed. Storing all 16 frames would increase area and refresh power but access power will not change significantly. The total chip area would increase by 3.75× and chip power by 2.1×.

The key challenge with eDRAM is that their bit cells require frequent refresh to retain data. Due to higher leakage and smaller bitcell capacitance of the eDRAM process technology as compared to DRAM process, refresh requests need to be made more frequently and so, refresh power is more significant on eDRAM than DRAM. SRAM does not need to be refreshed as the data is actively maintained by cross-coupled inverters. Further, for video decoding application, the instantaneous read-write bandwidth requirement can be satisfied by 2-3 eDRAM macros in the frame buffer, while the rest of the macros remain in self-refresh to retain data. As a result, the eDRAM energy is dominated by refresh power (80% of total eDRAM power and 40% of total chip power).

Accordingly, we focus on reducing refresh power as a means to reduce energy consumption of eDRAM. In this work, we show how reference frame compression can be used to place as many eDRAM macros in deep power-down as possible to minimize refresh power. Another use of eDRAM is in the decoupling buffer between CABAC and the 4 decoder cores. We show how the inherent compression in HEVC can be used to reduce the size and bandwidth of the decoupling buffer to reduce its power consumption.

eDRAM macros can operate in three main modes:

1. Active mode: frequent read/write requests and regular refresh requests are sent to the macro
2. Self-refresh mode: no read/write request is sent, but regular refresh requests are sent to retain stored data
3. Deep power-down mode: the macro is power-gated and all data is lost

The deep power-down mode is the only mode in which refresh is disabled. Accordingly, we propose techniques to keep as many macros in deep power-down mode.

4.2 Comparison with DDR Memories

Compared to DDR3/4 and LPDDR3/4 memories, eDRAM has the following benefits:

1. eDRAM does not require a controller to issue commands like activate/precharge in addition to the user commands of read and write. eDRAM uses standard rail-to-rail voltage signals to communicate with logic.

2. A large memory can be built with tens of eDRAM macros which can be individually controlled (read/write/refresh/power down). The bandwidth of the memory is proportional to the number of macros. DDR memories have comparatively few banks (typically 8). The banks can be individually controlled so that the latency of an operation on one bank can be hidden behind an operation on a different bank. DDR3 and LPDDR3/4 memories have a feature called Partial Array Self Refresh (PASR) which allows inactive banks to be selectively placed in Self Refresh mode. However, the banks cannot be individually powered down, resulting in large background power.
3. Low latency of read requests in eDRAM allows for simpler logic design. For example, caches can be designed with a higher miss rate because the penalty of cache miss is reduced.

The main benefit of DDR memories is their higher density. Also, even though they have a higher background power, in a System-on-Chip setting, the incremental energy for memory access from an accelerator can be very small. For comparison, consider a 1Gb DDR3L 1.35V SDRAM with 8 banks [36]. For 1080p HEVC decoding, 4 banks are needed at the most, which can be reduced to 3 by the use of reference frame compression.

As a simple back-of-the-envelope calculation, the standby power with refresh for this system is given by:

$$P = \frac{3}{8} \cdot V_{DD} \cdot \left(I_{DD3N} + (I_{DD5} - I_{DD3N}) \cdot \frac{t_{RFC}}{t_{REF}} \right) = 26 \text{ mW}$$

where the terms have their standard meanings in DRAM literature.

Table 4.2: Power specifications for DDR3L memory [36] used for basic refresh power calculation.

V_{DD}	Supply voltage	1.35V
I_{DD3N}	Active standby current	45mA
I_{DD5}	Refresh current	175mA
t_{RFC}	Refresh-refresh spacing	110ns
t_{REF}	Refresh interval	7800ns

For comparison, an equivalent system with eDRAM has a self-refresh power of 36 mW.

A more accurate DRAM power estimate can be obtained by using a DRAM power estimation tool [37]. Using the memory access trace for the decoder, we estimate the power for Micron's LPDDR3 memory with the following specifications:

Table 4.3: Specification of Micron's LPDDR3 memory used for power estimation

DRAM clock	800MHz (1600MT/s)
Size	4Gb
Rank	Single rank
Number of banks	8
Rows per bank	16384
Columns per row	1024
Column width	32b
Burst length	8 columns

The minimum access unit for this memory is 256b (32b×8). As the decoder is designed for the 128b access unit size of eDRAM, this analysis assumes that only half the data per access is used and the power estimate provided by the tool is halved. The total power of the DRAM memory was estimated to be 89.6 mW for decoding 10 frames of the Kimono test video (1920x1080 resolution).

A significant portion of this power was for ACT (activate) energy, which is the energy to read a new 32Kb row from a memory bank. To address this, [9] attempts to minimize the number of ACT commands by distributing accesses to all banks and increase temporal locality of accesses to the same row. This helps to reduce both energy and latency of accesses. The downside of this approach is that all memory banks are used which prevents bank-wise optimizations like PASR from being used.

For video decoders in applications like 4K TVs, DRAM is a clear choice as eDRAM would be prohibitively expensive in terms of both cost and power, especially refresh power. In smaller devices with 1080p and smaller screens, eDRAM can be much more energy-efficient than DRAM through reduction in active power. In either scenarios, reference frame compression (RFC) can be used to reduce memory power. In this work, we focus on RFC for reducing eDRAM refresh power.

4.3 Reference Frame Compression

Refrence frame compression (RFC) is a popular technique used on video decoders with DRAM main memory to reduce off-chip bandwidth [38]–[40]. Data compression for memory savings has also been explored for processor caches [41]–[43] but for specific applications like video coding, we can use prior knowledge of structure in the data to design better compression methods.

To maintain compliance with the video coding standard, RFC must use lossless compression which makes it a variable length compression. To maintain random accessibility of data, the compressed data is stored at fixed offsets corresponding to the maximum size of the compressed data (which, in case of lossless compression is at least as much as the size of uncompressed data). The size of the compressed data to be read is stored in a separate on-chip buffer, which is read before the off-chip data is read. In this way, bandwidth reduction is achieved, without storage size reduction. Hence, this method is not useful for reducing eDRAM refresh power.

Our approach is to store the compressed data in the memory in a fully packed form. As it is stored, the starting byte address and compressed size of the data is recorded in a separate address buffer. To read the pixel values at a given location in the image, the following process is used:

1. Convert pixel coordinates into an index into the address buffer
2. Read starting byte address and compressed size from address buffer. Use this to index into the data buffer
3. Read compressed data from the data buffer.
4. Decompress the data.

This process is shown in Figure 4-1.

4.3.1 Lightweight RFC Algorithm

In this work, a lightweight compression technique is applied on 4×4 pixel blocks. Each 4×4 block is compressed to three elements:

1. M: minimum of the 16 pixel values [8-bit: 0 to 255]

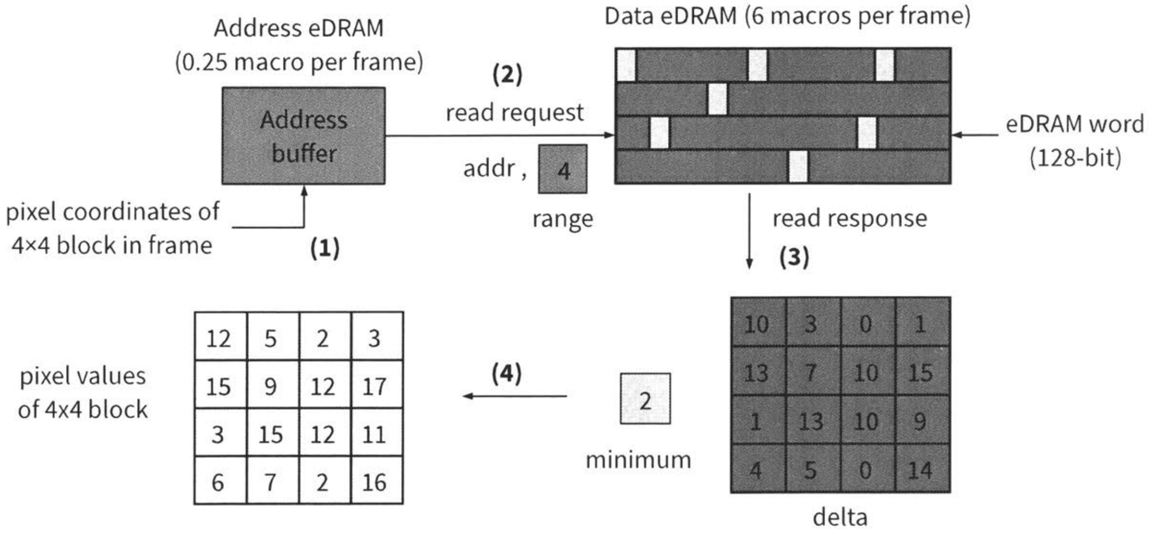


Figure 4-1: Complete process of reading a 4×4 block

2. **R**: (log-range) number of bits required to represent the delta above the minimum [4-bit: 0 to 8]
3. **D**: delta above M for 16 pixels using R bits (16R bits)

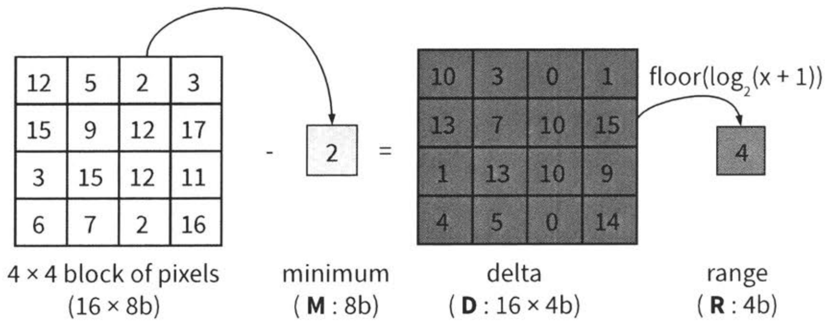


Figure 4-2: Example of lossless compression of 4×4 block

Figure 4-2 shows an example of the proposed RFC algorithm. The compressed size is 12 + 16R bits, for 128 bits of uncompressed data. In the special case when R = 8, no bit reduction is achieved by this algorithm. In this case, the minimum-delta representation is not used and the pixels are stored as their original 8-bit values. The R value which determines the size of the compressed data is stored separately in the address buffer and only M-D (or original values for R=8) are stored in the data buffer. The data block is always byte-aligned and its size in bytes is:

$$\text{byte-size}(R) = \begin{cases} 1 + 2R, & \text{for } R < 8 \\ 2R, & \text{for } R = 8 \end{cases}$$

Compression is achieved since the pixels in a 4×4 block are typically correlated and R is around 3 to 4 (as opposed to 8 for uncompressed data).

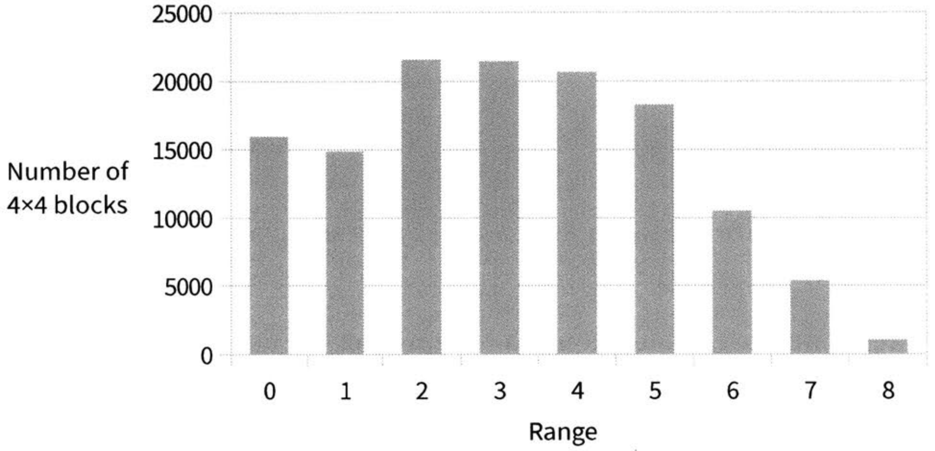


Figure 4-3: Histogram of 4×4 blocks according to their R values for one 1080p luma frame

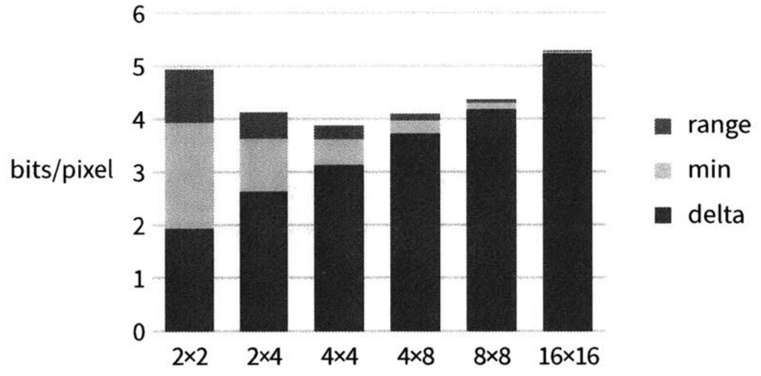


Figure 4-4: Motivating the choice of pixel block size for RFC

Implementing this algorithm in hardware takes up a total of 8 kgates of logic area for compression and decompression at a throughput of one 4×4 block per cycle at 100 MHz.

4.3.2 Compact Address Format for Address Buffer Size and Energy Savings

The data buffer is addressed by a 22-bit address, so the entry in the address buffer is 26-bit (22-bit address + 4-bit R). For 128-bit uncompressed data, this overhead is 20%, requiring 5 eDRAM macros for the address. To reduce the size and refresh power of the address buffer, the address entries are stored in a compact format as shown in Figure 4-5.

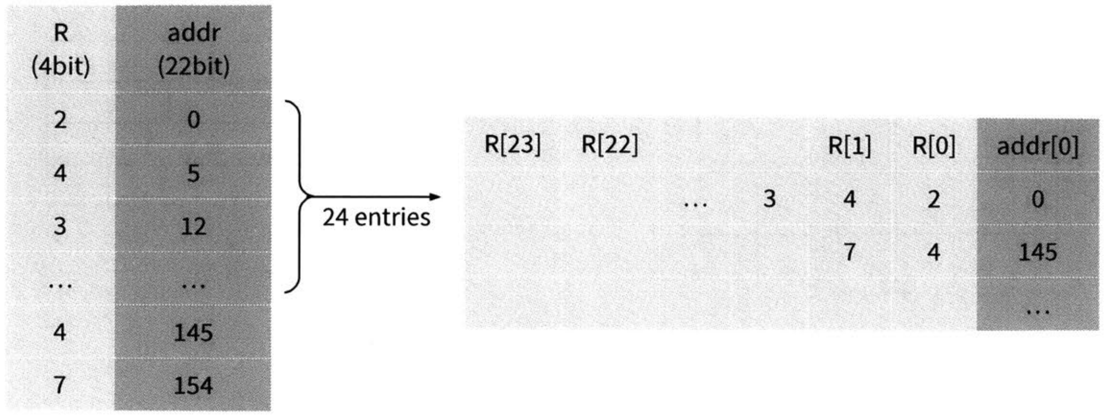


Figure 4-5: Reducing the size of Address buffer

24 consecutive address entries are packed in a 128-bit eDRAM word by storing the starting address of the first entry and 24 R values. Since the compressed data byte-size is a function of R, the starting addresses for the 2nd to 24th entries can be computed from all the R values as shown in Figure 4-6. This method enables storing the address buffer in a single eDRAM macro.

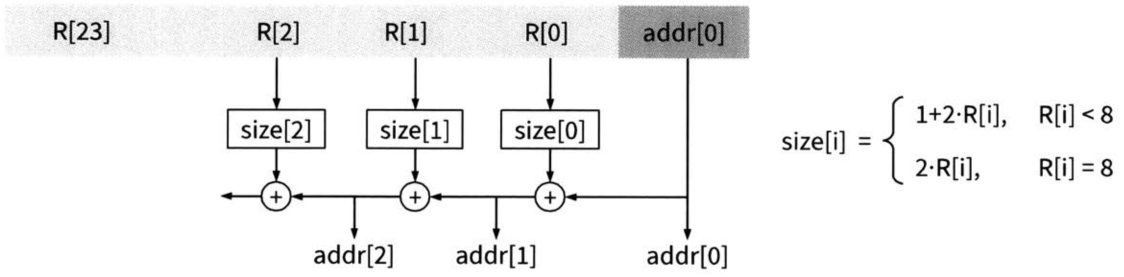


Figure 4-6: Computing starting addresses of all 4×4 blocks from compact adddress entry

The loop filter is designed to output pixels in units of 24 4×4 blocks composed of 16 luma and 8 chroma blocks as shown in Figure 4-7. When writing pixels to the frame buffer, the compressed data (minimum and deltas) is written to the data eDRAM after aligning to 128-bit eDRAM word. The starting address of the first 4×4 block in the group of 24 blocks is saved to an address entry register along with the 24 R values. When all the 24 blocks are written to the data eDRAM, the address entry register is written to the address eDRAM at an index computed from the pixel coordinates of the first 4×4 block. This process is shown in Figure 4-8. The corresponding read process is shown in Figure 4-9.

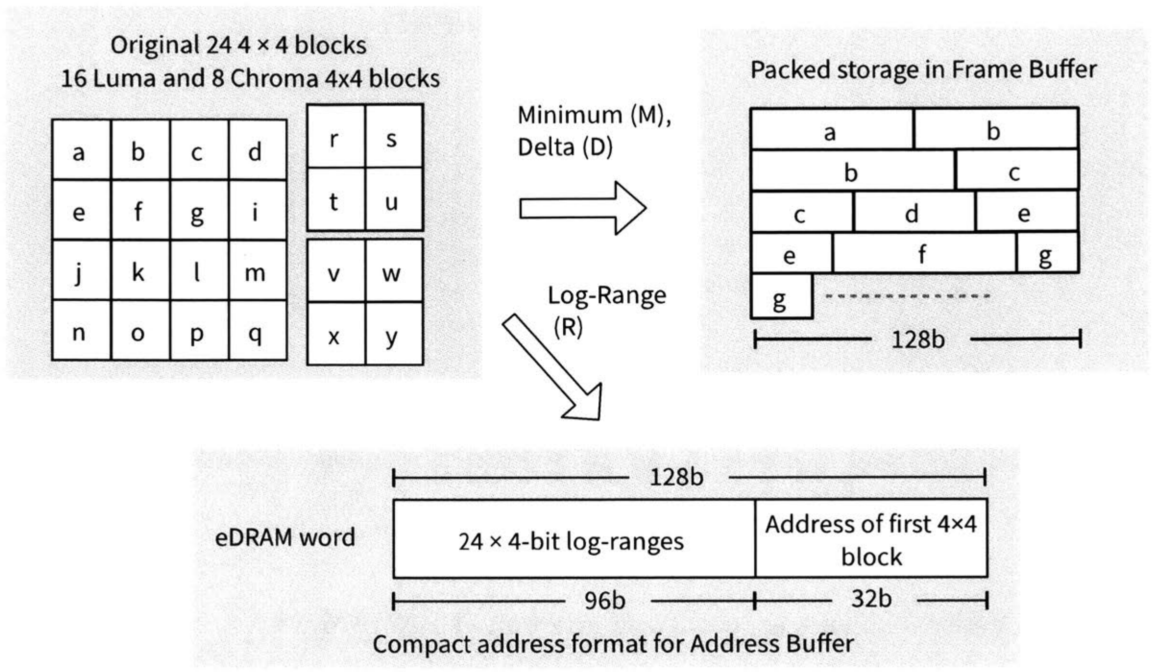

Original 24 4 × 4 blocks
16 Luma and 8 Chroma 4x4 blocks
a
b
c
d
e
f
g
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
Minimum (M), Delta (D)
Log-Range (R)
Packed storage in Frame Buffer
128b
eDRAM word
128b
24 × 4-bit log-ranges
Address of first 4×4 block
96b
32b
Compact address format for Address Buffer

Figure 4-7: Storing 24 4×4 blocks in consecutive positions in Frame data eDRAM and address buffer eDRAM

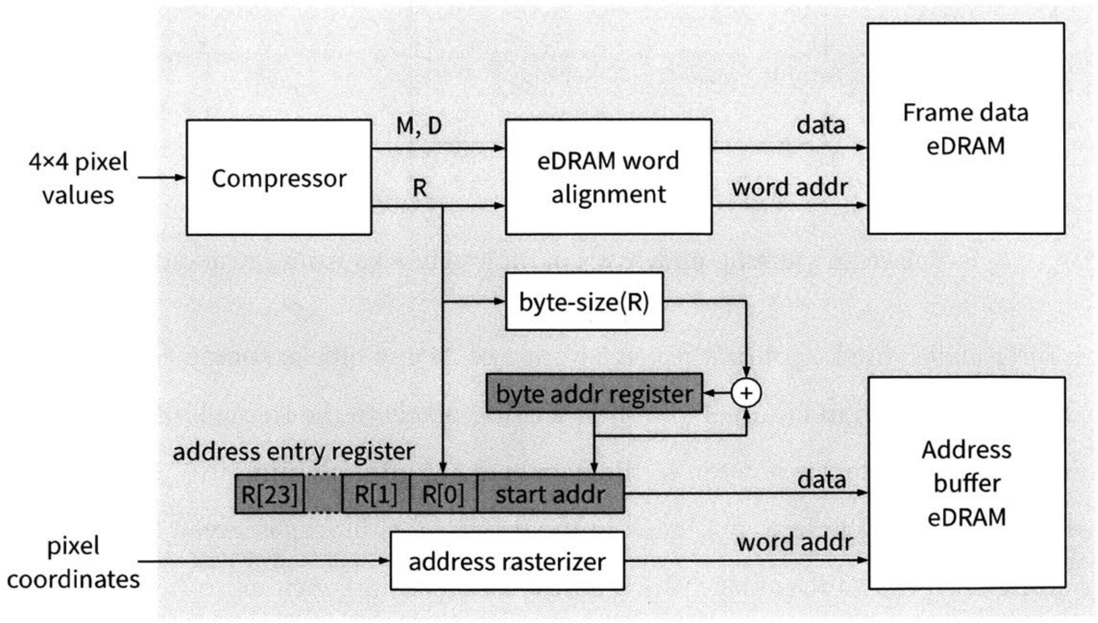


Figure 4-8: Write circuit for RFC showing data compressor and bookkeeping circuits for address lookup

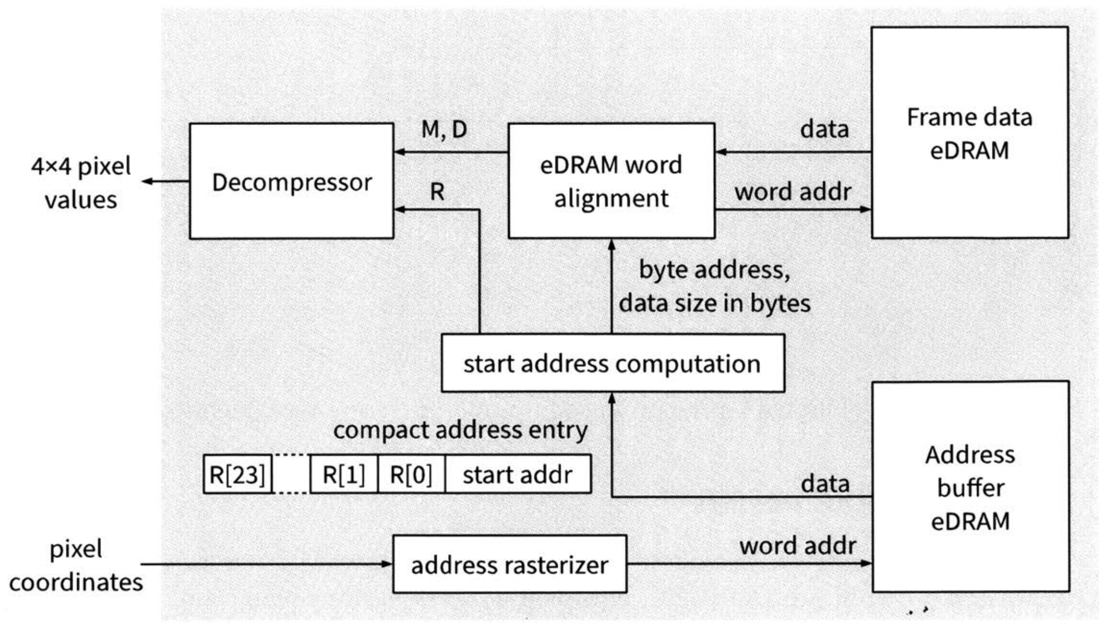


Figure 4-9: Read circuit for RFC showing address lookup from compact address format, start address computation and data decompressor

4.3.3 On-demand eDRAM Power-up

Due to data-dependent compression of RFC, the number of eDRAM macros needed to store a frame cannot be known a priori. Maximum number of macros is needed when no compression is achieved (R = 8 for all 4×4 blocks). For 1080p frames, this is 6 macros. In the best case (R = 0 for all blocks), the compressed frame needs less than 1 macro. In a simple scheme, the maximum number of macros needed for a frame are powered up at the start of decoding a new frame. When the frame is fully decoded, we can determine how many macros were actually used, and place the rest in deep power down mode.

To further reduce the number of macros used, we propose an on-demand power-up scheme. At the start of decoding a new frame, one eDRAM banks is powered up for writing. When the storage utilization of the bank reaches a predetermined threshold, a new bank is powered up. The threshold is designed to take into account eDRAM startup time. This on-demand scheme for powering up banks reduces eDRAM refresh power by 33% over the simple scheme and 55% over keeping all banks powered up always. Figure 4-10 shows the number of eDRAM banks powered up over time for each scheme.

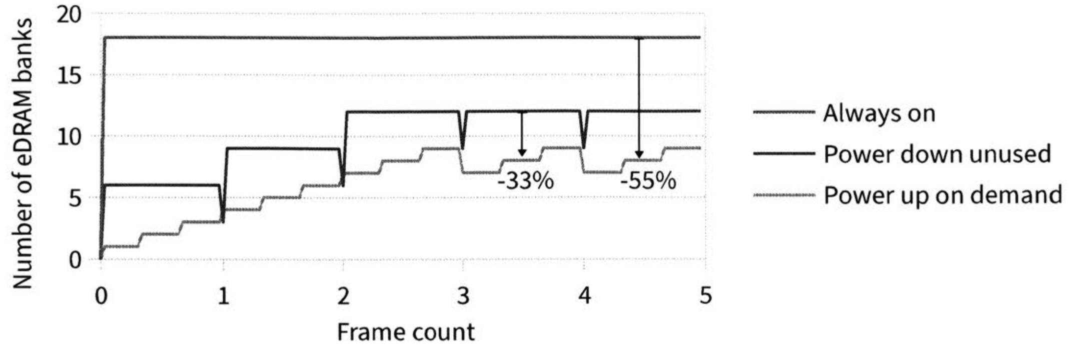


Figure 4-10: Number of eDRAM banks powered up over the course of decoding the first 5 frames

4.3.4 Summary of RFC Results

We demonstrated a combination of RFC, compact address buffer format and on-demand power-up of banks to reduce the number of active eDRAM banks. The unused banks are powered down to reduce refresh power by 50%.

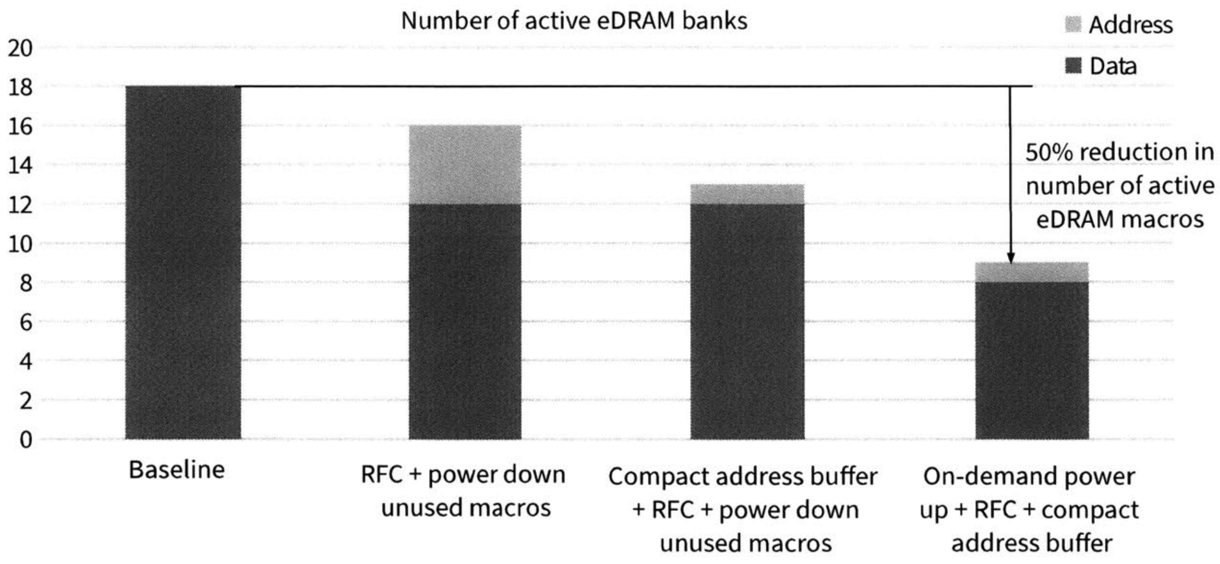


Figure 4-11: Reduction in number of active eDRAM banks through the use of RFC, compact address buffer format and on-demand power up of macros

The lightweight lossless RFC algorithm achieves a compression of 1.2×-5× over 384 video sequences in the HEVC test suite. The compression depends on factors such as level of detail in the video and quantization level. Video content with small spatial gradients (background sky, for example) and heavily compressed video with high quantization achieves better RFC compression. The average compression is 50%. Table 4.4 shows a comparison of the lightweight RFC algorithm with a state-of-the-art algorithm used for DRAM-based video decoders. We can see that the lightweight algorithm achieves a good cost-performance trade-off. Overall, RFC has <1% power and area overhead but saves total power by 16%.

Table 4.4: Comparison of lightweight RFC algorithm with a state-of-the-art algorithm.

	Lightweight algorithm	State-of-the-art [38]
Compression method	minimum-delta	intra-prediction + DPCM + coding
Data savings	50%	60%
Logic Area	8 kgates	80 kgates
Throughput	32 pixel/cycle	32 pixel/cycle

4.4 Exploiting HEVC Compression in Syntax Element Buffer

As explained in Chapter 2, the chip uses two clock domains for frontend and backend processing. The frontend consists of the entropy decoder and the backend consists of 4 pixel decoder cores (Dec core 1-4). The frontend and backend frequencies can be configured to balance throughputs; nominally frontend is run at 4 times the clock frequency as the backend. The decoder cores use macro-parallelism and process 4 consecutive rows of Coding Tree Units (CTUs) in the frame as shown in Figure 2-13. In HEVC, a CTU can be up to 64×64 pixels in size.

To address the mismatch in processing order of entropy decoder and the 4 Dec cores, a decoupling buffer is needed between them. To keep all 4 Dec cores running, the buffer needs to store entropy decoder output of 4 rows of CTUs. An additional 4 rows of buffer is needed to allow the entropy decoder to write its output.

The decoupling buffer is useful even when macro-parallelism is not used i.e. with 1 entropy decoder and 1 Dec core. The bit-level throughput of entropy decoder varies widely due to varying levels of quantization of transform coefficients. Comparatively, the Dec cores have a more regular pixel throughput. With a decoupling buffer capable of storing multiple CTUs of entropy decoder output, the throughput variation can be averaged out. Figure 4-12 shows the variation in the workload of entropy decoder as measured by number of binary symbols (bins) per CTU for one intra frame in a 1920x1080 video sequence (ParkScene encoded at QP27). The workload varies considerably from as low as 30 bins per CTU to as high as 8000 bins per CTU. The dotted line in Figure 4-12 shows the workload when averaged over one row of CTUs (30 CTUs) in a moving average. We observe that the variation is much more tolerable (1800 - 4000 bins per CTU).

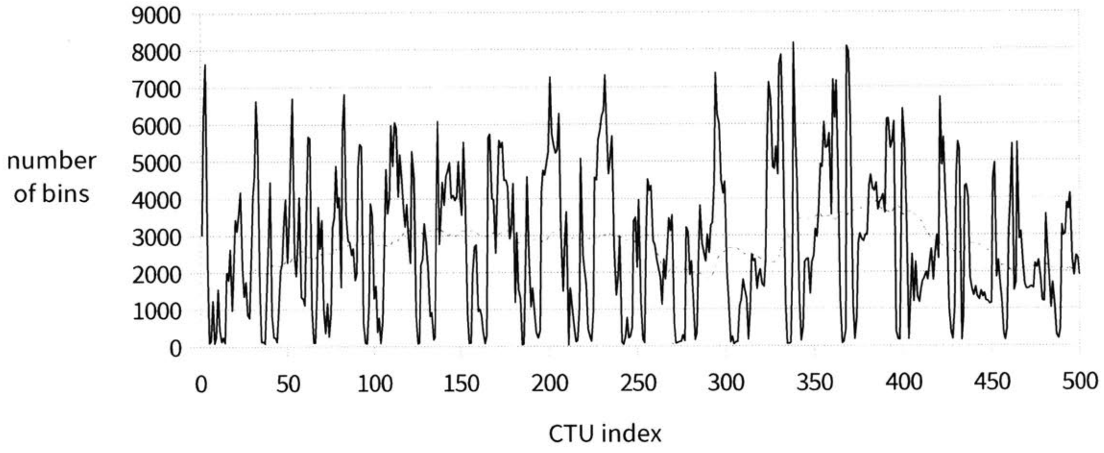


Figure 4-12: Variation in workload of entropy decoder over CTUs as measured by number of binary symbols. Dotted line shows moving average over 30 CTUs.

If syntax elements are stored in the decoupling buffer, most of the buffer space is taken up by transform coefficients (16 bit/pixel) as the other syntax elements such as motion vectors and intra modes are signaled at PU granularity. The buffer size for storing 8 rows of 64×64 CTUs in a 1080p image is 3 MB, or 6 eDRAM macros. To reduce this size, the entopy decoder is split into CABAC that outputs binary symbols (0s and 1s) and a Debinarizer that parses the stream of binary symbols for syntax elements as proposed in [44]. This is shown in Figure 4-13. The binary symbols are very compact representations of the syntax elements using a variety of lossless coding techniques such as Run-length coding and Huffman coding. The Debinarizer is moved into each Dec core so that the decoupling buffer can store the binary symbols instead of syntax elements.

This reduces bandwidth to the decoupling buffer by 66× and its power by 4×. Debinarizers are needed in each Dec Core to decode the bins to syntax elements, which add an overhead of 1mW power (4% of chip power), 102 kgates of logic area (10% of chip logic area) and 16 kB of SRAM (10% of chip SRAM bits). Overall, the area and power savings in the decoupling buffer are larger than the debinarizer overhead. The total chip area is reduced by 6% and power is reduced by 16%.

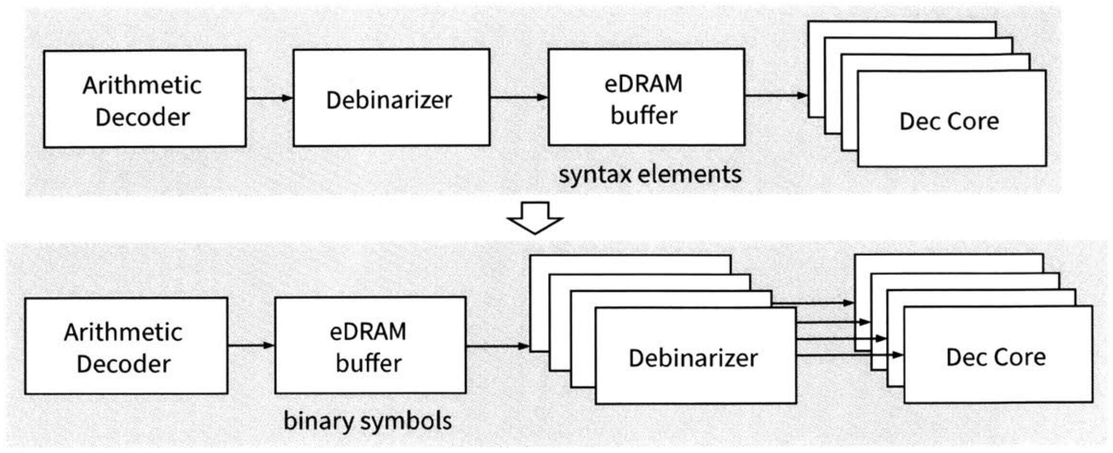


Figure 4-13: Decoupling buffer between CABAC and Dec Cores that stores binary symbols instead of syntax elements

Chapter 5

Test Chip Results

The test chip [20] shown in Figure 5-1 was taped out in 40nm CMOS process. The test chip can operate from 0.8V to 1.1V (eDRAM fixed at 1.1V) with the frontend/memory clock domain frequency ranging from 30.3MHz to 76.9MHz and a backend domain frequency ranging from 7.6MHz to 19.2MHz. At maximum frequency, the chip can decode 1920x1080 video at 24 - 50 frames per second depending on encoding parameters. The chip consumes 30.6mW (0.35nJ/ pixel) for I frames (only intra-prediction is used), and 24.9mW (0.77nJ/pixel) for B frames (both intra and inter-prediction is used). Table 5.1 summarizes the key specifications of the chip.

The difference in throughput based on encoding parameters is due to insufficient buffer sizes in memory arbiters and inter-prediction. As a result, the backend cores would send at the most 2 read requests to the cache which was designed to handle as many as 8 requests in a pipelined manner. Due to this, the decoder processes B frames much slower than it processes I frames (I frames do not make any read requests to the cache as they only use intra-frame prediction). Increasing the size of buffers in memory arbiters and inter-prediction allows the decoder to process all frames at 52 frames per second at 100 MHz. This was verified in RTL simulation. The extra buffer size is 2.5 kbits of registers.

The test chip was synthesized to operate at 100MHz frontend and 25MHz backend frequency at 1.1V supply. However, during testing, it was not possible to supply the two clocks signals with a phase-alignment tolerance of 200ps as set in the timing constraints. This resulted in hold-time violations between paths that crossed the clock domains. The hold-time violations were avoided by changing the phase alignment of the clock signals so that the positive edge of the slower clock (backend) coincided with the negative edge of the faster clock (frontend). However, this results in lower timing margin for setup on the clock-domain-crossing paths. This limited the frontend clock frequency to 76.9MHz during testing. For future work, the timing constraints should be set to better match the clock inputs that can be generated by the test setup. Alternatively, asynchronous clock-domain crossings can be used to address hold-time violation issues.

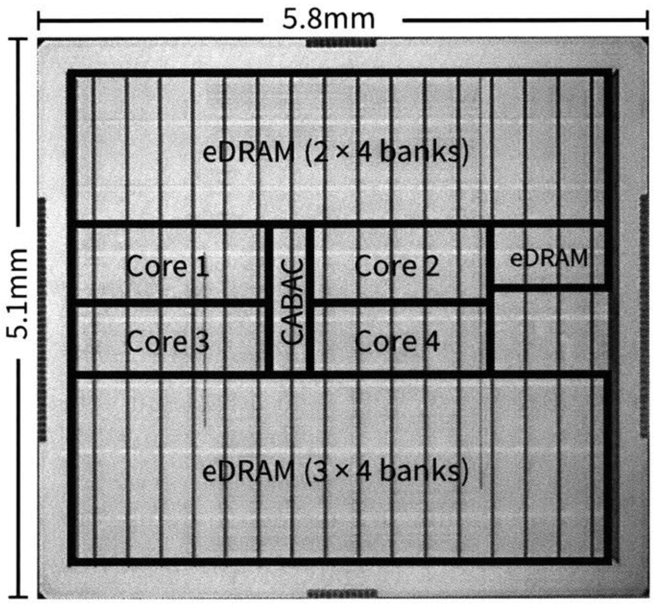


Figure 5-1: Die micrograph of HEVC decoder test chip showing memory and logic blocks

Figure 5-2 shows the voltage-frequency plot for the test chip. The frontend frequency is kept at 4× the backend frequency at all voltages. On the lowest voltage setting (0.8 V), the frequency is sufficent to decode 640x480 at 60 fps while consuming only 9.5 mW of power.

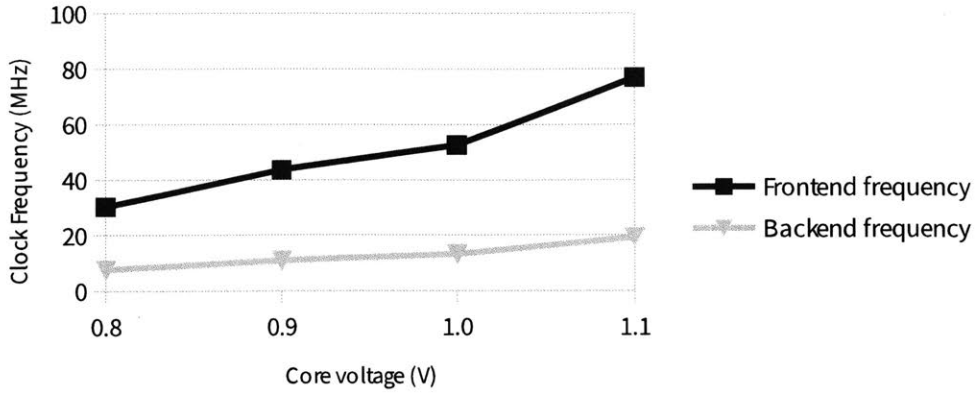


Figure 5-2: Measured voltage-frequency performance plot for test chip

Table 5.1: Summary of chip specifications

Technology	TSMC 40nm LP
Supply voltage	Core: 0.8 - 1.1V, eDRAM: 1.1V, IO: 2.5V
Video standard	H.265/HEVC (Main profile with 2 reference frames)
Chip size	5.8mm × 5.1mm
Core size	5.1mm × 4.3mm
Gate count	1122 kgates (NAND2 logic area only)
On-chip SRAM	162.75 kB
On-chip eDRAM	21 × 0.5 MB
Max resolution	1920 × 1080
Max throughput	47.9 MPixel/s
Power at 1.1V	30.6 mW (I frame) and 24.9 mW (B frame)

5.1 Test Setup

The test setup consists of a PG3A Digital Pattern Generator from The Moving Pixel Company and a TLA7012 Logic Analyzer from Tektronix. Input patterns to the chip were generated from Verilog RTL simulation and loaded into the Pattern Generator. The Pattern Generator has a limit of 32M input samples which limited testing to approximately 6 frames in each video sequence. Four 1080p video sequences (Kimono, ParkScene, Cactus, and BQTerrace) were encoded with 2 reference frames. Four different quantization parameters were used (22, 27, 32 and 37 respectively) for the video sequences.

The Logic Analyzer is limited to recording even fewer data samples (512K), so a script was written to capture and export consecutive sections of the output stream. The exported output samples were compared to the expected samples from RTL simulation in a Python script. Figure 5-3 shows a photograph of the assembled printed circuit board within the test setup.

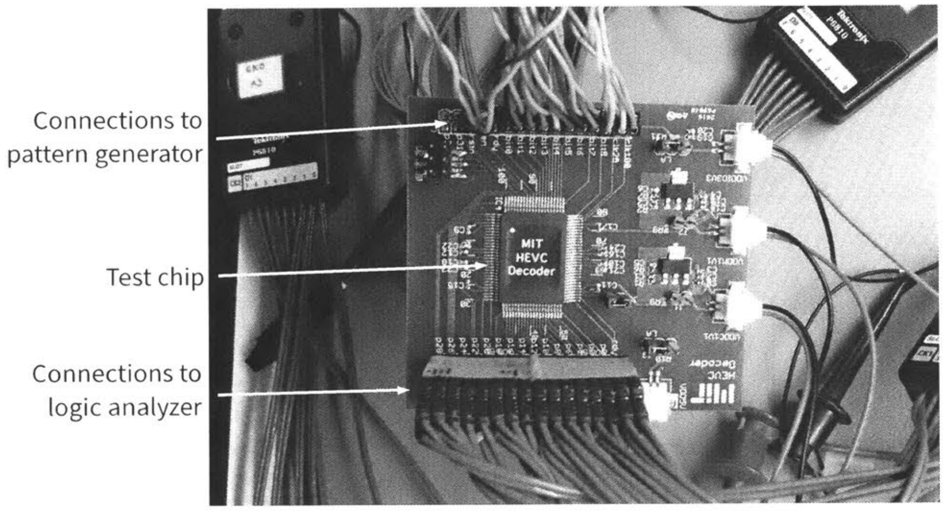

Connections to pattern generator
Test chip
Connections to logic analyzer
MIT HEVC Decoder

Figure 5-3: Photograph of the test board

5.2 Energy and Area Breakdown

Figure 5-4 shows breakdown of energy consumed for decoding 1080p frames. This includes the power for eDRAM macros in the frame buffer which take up 33% of the total power. The portion marked Memory controller in the breakdown contains the cache, RFC circuits and memory access arbiters with cache being the most dominant energy consumer.

Figure 5-5 shows the energy breakdown for an individual decoder core. This breakdown is obtained from post-synthesis analysis using 5 ms of switching activity. The main modules contributing to energy consumption in the decoder core are inter-prediction, pipeline buffers, debinarizer, and inverse transform. The energy consumption in inter-prediction is dominated by the interpolation filters. In debinarizer, the main energy consumer is the pipeline buffer between debinarizer and transform which, as explained in the previous paragraph, is counted as part of debinarizer. Overall, we observe that pipeline buffers take up a significant portion of the energy consumption (even more that line buffers, which are much larger in size). We expect that pipeline buffers will become even more important in future standards as they follow the trend of increasing pixel block sizes.

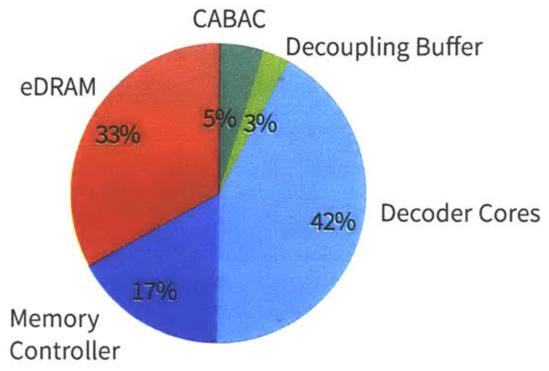


Figure 5-4: Energy breakdown of chip for decoding 1080p frames

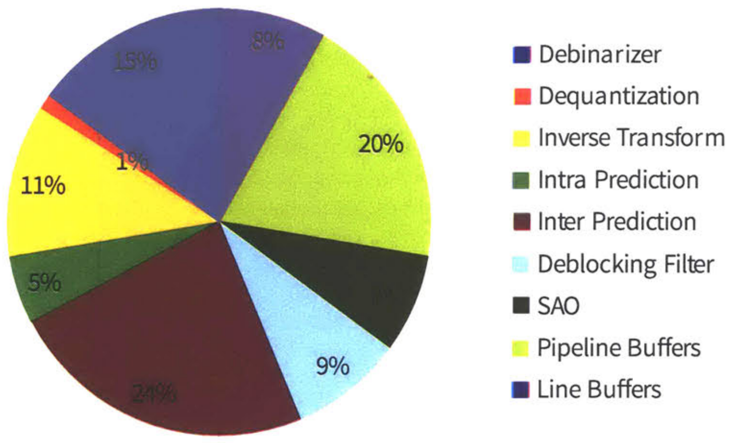


Figure 5-5: Energy breakdown for individual decoder core

Each Dec core uses 235 kgates of logic with a breakdown as shown in Figure 5-6. Inverse Transform and Inter Prediction contribute most to the gate count (31% and 29% respectively). In both modules, the area is dominated by constant multipliers in the data path. Multiple constant multiplication has been used to reduce the area cost of both modules.

The complete decoder uses 162.75 kB of on-chip SRAM with a breakdown shown in Figure 5-7. Memory controller consists of arbiter for writing back decoded pixels to the frame buffer and a cache which uses SRAM for both data and tags. Pipelining consists of buffers between:

1. debinarizer and prediction
2. inverse transform and prediction
3. prediction and deblocking filter

The transform coefficient buffer between debinarizer and inverse transform is also used to convert between Z-scan order and raster scan. That buffer is counted in the SRAM usage of the debinarizer module. In all, 100 SRAM macros , both single-port and dual-port, are used. Single-port SRAMs are preferred at larger sizes where most of the area is used by the bit-cells as opposed to row and column circuits. Dual-port SRAMs have the benefit of allowing simultaneous read and write, and so they are preferred especially at smaller sizes as 1 dual-port SRAM can be used instead of 2 single-port SRAMs.

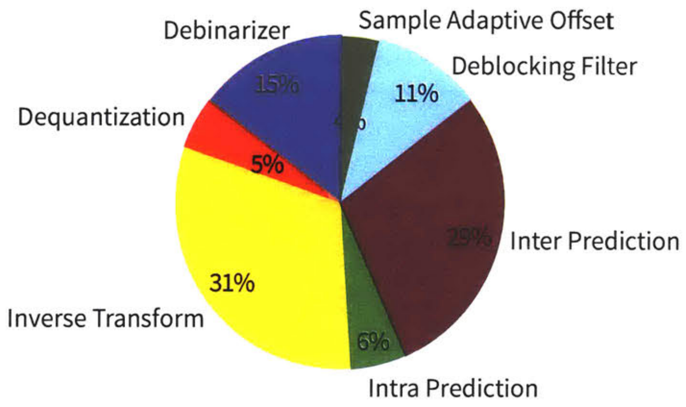


Figure 5-6: Logic area breakdown of pixel decoder core

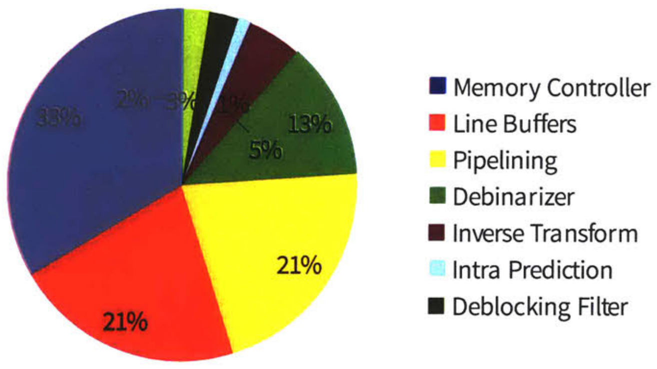
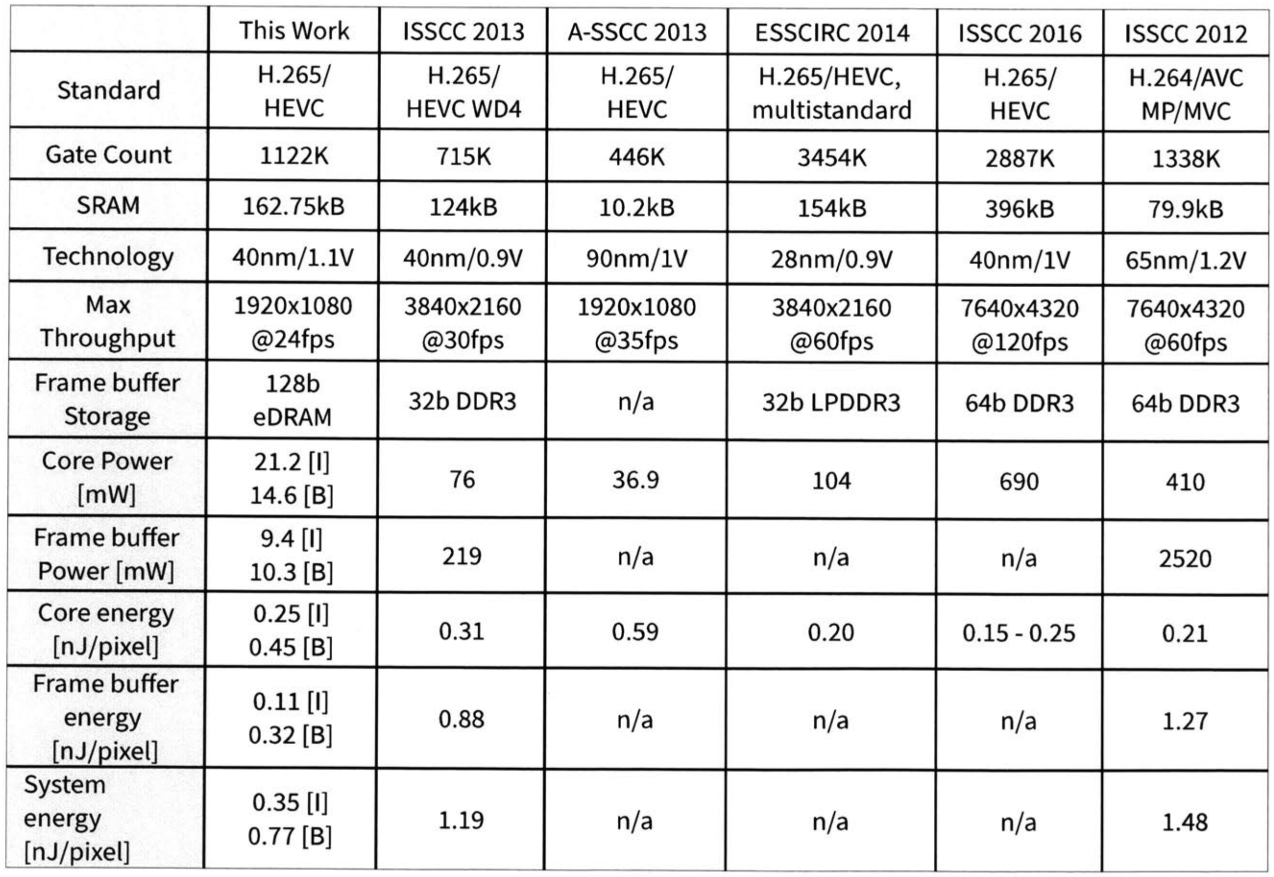


Figure 5-7: SRAM bits breakdown of chip

5.3 Comparison With State-of-the-Art

Figure 5-8 shows how this chip compares against state-of-the-art H.265/HEVC video decoders. All the other designs target higher resolutions for applications with > 50mW power budgets. The use of eDRAM allows this work to meet the stringent power budgets for wearable devices. We also compare this chip with our previous work [9] scaled to 1080p resolution in Table 5.2. The frequency of the previous chip is scaled down to 40 MHz to match the throughput of this chip. Voltage and technology scaling is not applied. For energy comparison, leakage power of the core and background power of DRAM are kept constant and only the active components of both are scaled down for the reduced throughput. We observe that the current work has 6× lower energy/pixel than the previous work.



	This Work	ISSCC 2013	A-SSCC 2013	ESSCIRC 2014	ISSCC 2016	ISSCC 2012
Standard	H.265/ HEVC	H.265/ HEVC WD4	H.265/ HEVC	H.265/HEVC, multistandard	H.265/ HEVC	H.264/AVC MP/MVC
Gate Count	1122K	715K	446K	3454K	2887K	1338K
SRAM	162.75kB	124kB	10.2kB	154kB	396kB	79.9kB
Technology	40nm/1.1V	40nm/0.9V	90nm/1V	28nm/0.9V	40nm/1V	65nm/1.2V
Max Throughput	1920x1080 @24fps	3840x2160 @30fps	1920x1080 @35fps	3840x2160 @60fps	7640x4320 @120fps	7640x4320 @60fps
Frame buffer Storage	128b eDRAM	32b DDR3	n/a	32b LPDDR3	64b DDR3	64b DDR3
Core Power [mW]	21.2 [I] 14.6 [B]	76	36.9	104	690	410
Frame buffer Power [mW]	9.4 [I] 10.3 [B]	219	n/a	n/a	n/a	2520
Core energy [nJ/pixel]	0.25 [I] 0.45 [B]	0.31	0.59	0.20	0.15 - 0.25	0.21
Frame buffer energy [nJ/pixel]	0.11 [I] 0.32 [B]	0.88	n/a	n/a	n/a	1.27
System energy [nJ/pixel]	0.35 [I] 0.77 [B]	1.19	n/a	n/a	n/a	1.48

Figure 5-8: Comparison with the state-of-the-art

Table 5.2: Comparison with previous work at 1080p 24 fps. * Power for previous work is estimated by scaling frequency.

	This work	[9]
Standard	H.265/HEVC	H.265/HEVC Working Draft 4
Logic gate count	1122 kgates	715 kgates
SRAM	162.75 kB	124 kB
Frame buffer	128b eDRAM	32b DDR3
Technology	40nm LP	40nm GP
Core voltage	1.1 V	0.9 V
Frequency	80 MHz/20 MHz	40 MHz
Core power	14.6 mW	36 mW *
Frame buffer power	10.3 mW	150 mW *
System power	24.9 mW	186 mW *

Chapter 6

Conclusions and Future Directions

In this thesis, we demonstrated several techniques to improve energy-efficiency of a fully-integrated H.265/HEVC video decoder in hardware. These techniques exploit some key features of video coding as follows:

1. **Exploit input statistics:** We saw that the move to larger transform sizes, which achieve greater coding efficiency, causes the inverse transform module to consume significantly higher energy in HEVC decoders than in AVC decoders. We addressed this by exploiting the higher sparsity of larger transforms to design an architecture that can perform larger transforms with the same energy/pixel as smaller transforms.

   We use eDRAM as the main memory for the video decoder to elimiate off-chip memory accesses that can take as much as 321 times more energy/access as eDRAM [28]. However, we identified that refresh operations take up most of the energy in eDRAM. To address this, we used reference frame compression, which exploits spatial redundancy in the pixel data, to reduce the number of eDRAMs being used for storing data at any point of time and power down the remaining eDRAMs.

   We exploit the probability distribution of syntax elements to reduce the energy cost associated with distributing them to multiple decoder cores. The decoupling buffer between entropy decoder and the multiple decoder cores needs to store syntax elements for several rows of CTUs. We store the syntax elements in the compact binary format designed by the

HEVC standard based on the probability distribution of the syntax elements. This reduces both bandwidth and storage cost for the decoupling buffer.

2. **Exploit parallelism:** We exploit macro-level parallelism in HEVC's algorithms to design a multi-core decoder. The dependencies between the cores are managed using FIFOs instead of a shared buffer which allows for an easily scalable and reconfigurable architecture.

   We exploit micro-level parallelism in the pixel-processing modules like inter-prediction and intra-prediction to increase reuse of computation through multiple-constant-multiplication and reuse of memory accesses.

3. **Manage data dependencies:** HEVC uses flexible coding units from 8×8 to 64×64 as compared to H.264/AVC's fixed macroblock size of 16×16. This requires large pipeline buffers and a flexible pipeline scheme to manage the various block sizes. We analyzed the data dependencies in HEVC algorithms to determine that the pipeline buffers between inverse transform, inter and intra prediction can be sized to the largest transform unit rather than the largest coding unit size. This reduces the size of the pipeline buffers by 4×.

Future video coding standards can be made aware of these techniques and be designed to further exploit them. Improving coding efficiency usually comes at the cost of increased encode-decoder complexity, but that is not necessarily the case for all aspects of the video coding process. For example, focusing on implementation complexity during the development of HEVC Entropy Coding resulted in a standard specification that performed better than AVC Entropy Coding in terms of both coding efficiency and throughput [45]. In this work, we showed that larger transform sizes need not consume more energy/pixel than smaller transforms as larger transforms are also more sparse.

The standard has to strike a balance between coding efficiency and implementation complexity in both software and hardware. However, recent trends [1] have shown that video is being consumed in increasingly higher proportion on energy-constrained devices. This arguably justifies prioritizing hardware costs over software.

We saw that almost 50% of the energy is spent in storing and accessing reference frames in eDRAM. We show here some approaches that can reduce this energy.

6.1 Restrict Memory Footprint

For DRAM-based memories, the energy is often dominated by refresh power rather than access power as we typically need to maintain 16 reference frames. A standard profile with restricted number of reference frames can be released for use with embedded devices. This will however restrict the choices for the encoder in motion estimation which impacts coding efficiency.

6.2 Lossy Compression of Reference Frames

In this work, we used lossless compression of reference frames to save memory power while maintaining standard compliance. Lossy fixed-length compression can provide similar benefits in memory power without the added complexity of address lookup. [40] uses a minimum-delta compression similar to what we used, except that the deltas are quantized and a fixed-length representation is used. Future coding standards can include a lossy compression algorithm into the specification. [40] showed that the frame buffer can be downsized by a fixed 25% with a 1.03% increase in bitrate for same quality. (For reference, HEVC achieves 50% decrease in bitrate for same quality).

6.3 Downsampled Reference Frames

A more extreme approach than lossy compression would be to downsample the reference frames. Single-image super-resolution algorithms such as [46] can be used to upsample the reference frame on-the-fly during inter-prediction. It is interesting to note that inter-prediction already has an element of upsampling in its interpolation filter for quarter-pixel motion compensation. The upsampling for compressed reference frame can be merged with this interpolation filter.

A similar approach has been used for a different application: super-resolution of videos [47]. The approach there is to use syntax elements of encoded video to amortize the cost of the super-resolution algorithm over multiple frames.

6.4 Learn the Compression Algorithm

Super-resolution algorithms such as [46] use a neural network trained on blocks of pixels and their downsampled versions. The training can be extended to the downsampling method so that both the compression and decompression methods can be learned from data. The complete scheme embedded inside a video decoder is shown in Figure 6-1. Image compression using learning has been explored for full images in [48] and can be extended to the small pixel blocks used by reference frame compression. Once again, the decompression algorithm can incorporate the interpolation filter needed by inter-prediction.

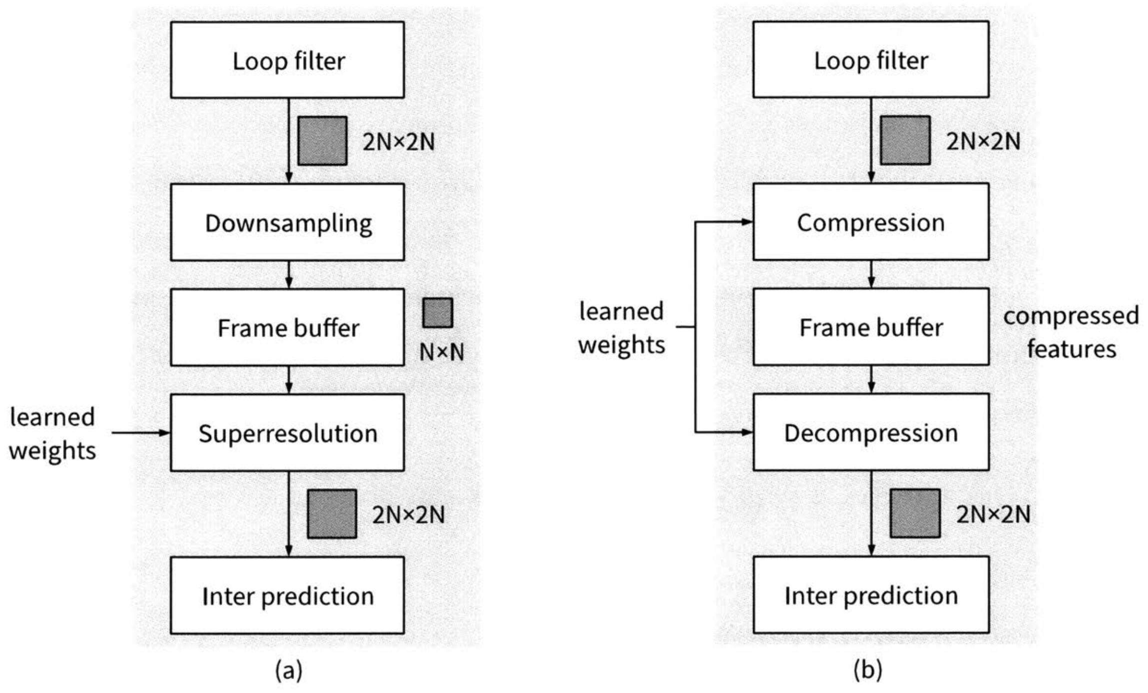


Figure 6-1: (a) Reference frame compression based on super-resolution (b) Learning both compression and decompression algorithms

It is important to remember that the super-resolution algorithm needs to be simple enough that the energy consumption of its implementation does not exceed the savings it achieves in the refresh or access energy of the reference frame buffer. As a result, simpler neural networks such as a single fully connected network might be preferable to deep networks.

Appendix A

Design of processing engines

The main chapters in the thesis described the design of inter-prediction, intra-prediction and inverse transform modules. This chapter provides details about the remaining processing engines.

A.1 Debinarizer

The debinarizer parses a stream of binary symbols from the arithmetic decoder to generate syntax elements. HEVC uses several binary formats to store the syntax elements. Some of these binary formats, called binarizations, are listed here:

- Fixed-length binary code: This is the simplest binarization for N-bit integer data where N is known.
- Mode-dependent Huffman codes: For example, the partitioning of CUs into PUs uses Huffman codes. The choice of the code depends on the size of the CU and whether the CU uses inter or intra-prediction.
- Unary code: This is a binarization for non-negative integers, where the integer value is denoted by the length of an unbroken string of 1's. (0: 0, 1: 10, 2: 110, 3: 1110, ...). If the maximum value is known, then the final 0 for the binarization of the maximum value is omitted.

- Truncated Rice codes: This is a combination of a prefix value encoded using a unary code followed by a fixed-length suffix value. The length of the suffix is decided by the prefix.
- Exponential-Golomb codes: These codes encode an N-bit non-negative integer with a binary string of length $O(log(N))$.

Following the parsing, some syntax elements require post-processing. For example, intra modes can be predicted from neighbors, motion vectors are predicted from neighbors and collocated regions in past frames using Advanced Motion Vector Prediction (AMVP), and transform coefficients need to be reordered from diagonal scan order to raster scan.

The main challenges with implementing the debinarizer are:

- **Large and complicated grammar the sequence of syntax elements in HEVC**: To address this, we used Bluespec SystemVerilog (BSV) to describe the Register Transfer Logic (RTL) for debinarizer. BSV provides a feature called Statement FSM which allows large finite state machines (FSMs) to be described as a sequence of actions. C-like constructs such as if-then-else, for and while loops are also supported. Also, parallel threads of execution can be described. A sequence of actions can be treated as a procedure and used in other sequences. The BSV compiler then infers a program counter to keep track of the current action and infers the logic for updating the program counter.
- **Maintaining the throughput for parsing transform coefficients**: The average throughput needed by inverse transform is 2 pixels/cycle. To support this throughput, the debinarizer is also designed to parse up to 2 transform coefficients. The input stream of binary symbols is stored in a FIFO with a readahead of 16 bits. This allows the parser to determine whether 2 transform coefficients are present in the 16 bits. If present, they are both parsed in the same cycle. Allowing a readahead of 16 bits also allows most other syntax elements to be parsed in 1 cycle.

The debinarizer was synthesized at 25 MHz with a logic area of 34 kgates and SRAM size of 5.3 kB. The main FSM for the HEVC syntax grammar takes 7 kgates, intra mode predcition takes 1.5 kgates, AMVP takes 16 kgates and reordering of transform coefficients takes 6 kgates. Input and output FIFOs take up rest of the area.

A.2 Loop Filter

HEVC uses two cascaded loop filters: deblocking filter and Sample Adaptive Offset (SAO). Deblocking filter operates on 8×8 block of pixels around TU and PU edges to reduce blocking artifacts. SAO is a new filter added to HEVC. It adds a small offset to each pixel value based on the value of its immediate neighbors. By removing coding artifacts, these filters improve the perceptual video quality of the current frame and provide a better prediction for future frames.

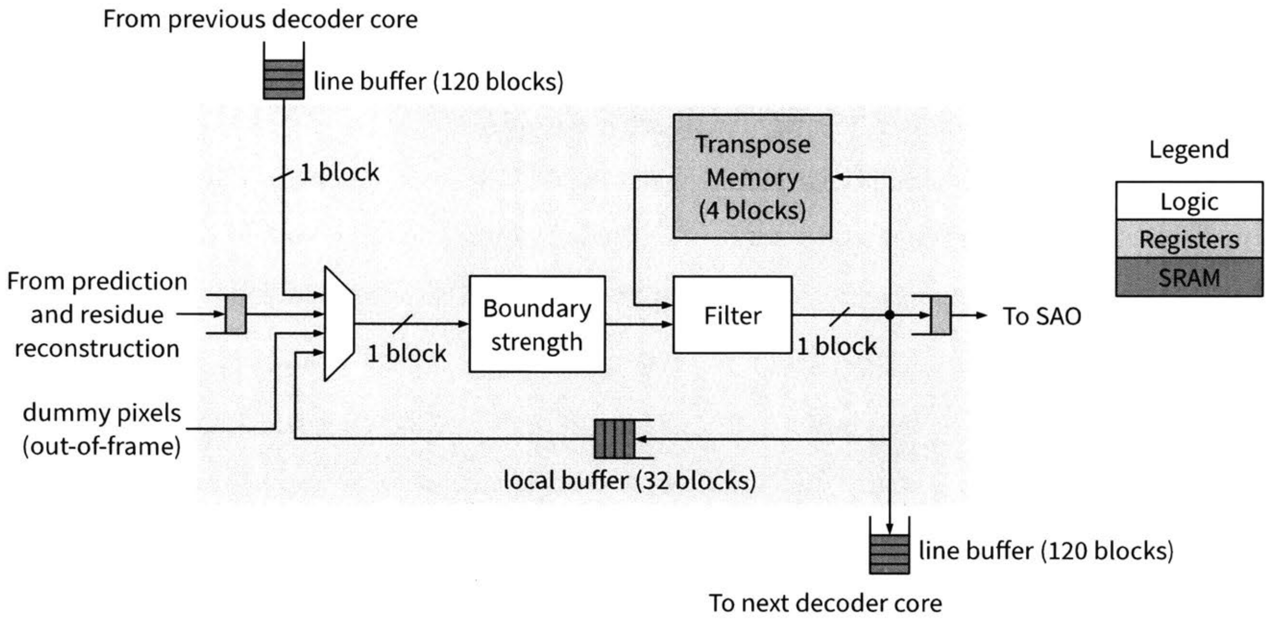


Figure A-1: Architecture of Deblocking filter. Block size is 4×4 pixels.

Figure A-1 shows the architecture of the deblocking filter. The deblocking filter reads 4×4 blocks of pixels from the previous pipeline stage in the decoder core pipeline. Pixel blocks from the upper row of CTUs are read from the line buffer connecting to previous decoder core. For deblocking edges that are also the edge of the frame, dummy out-of-frame pixels are provided to the deblocking filter.

The filter process is as follows:

1. Determination of boundary strength and filter parameters: Based on factors such as whether the edge is a TU or PU edge, quantization parameters of TUs on either side of the edge, values of motion vectors, etc. a determination is made about the strength of filter

and the threshold of pixel value that differentiates between a coding artifact and actual edge in the picture. Deblocking filter uses three types of filters: a strong luma filter, weak luma filter and a chroma filter.

2. Filter the vertical edges: Each 8-pixel row in the 8-pixel tall vertical edge is filtered independently.
3. Filter the horizontal edges: Each 8-pixel column in the 8-pixel wide horizontal edge is filtered independently. A transpose memory of four 4×4 pixel blocks forming an 8×8 pixel block is used to convert from the vertical edge to horizontal edge.

The output pixels are sent to SAO and also stored in a local buffer to be read as neighboring pixels for the next CTU. Pixels along the bottom edge of the CTU are stored in a line buffer to be read by the deblocking filter in the next decoder core. The local buffer and line buffers are implemented in SRAM while the transpose memory and input-output FIFOs are implemented in registers.

The deblocking filter is synthesized at 25 MHz and takes 26 kgates of logic area and 1.1 kB of SRAM, not including the line buffers. Boundary strength calculation takes 3 kgates, transpose memory takes about 4.5 kgates and the filter takes 8 kgates of area. Input-output FIFOs, and peripheral circuit around the SRAM for local buffer take up rest of the gates.

In contrast with the block-wise processing of the Deblocking filter, Sample Adaptive Offset (SAO) operates pixel-wise. The SAO parameters are specified at the CTU-level with the option to use parameters from left and top neighbor CTUs to reduce overhead of signaling the SAO parameters in the bitstream. The SAO parameters are as follows:

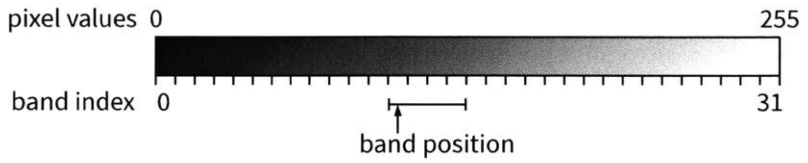


Figure A-2: SAO with band offsets. Offset values are specified for 4 contiguous bands starting at a specified band position

1. Offset type: Band offset and edge offset are the two types of offsets are used in HEVC.
2. Band position and offset values: When a band offset is used for a CTU, the range of pixel values from 0 to 255 is divided into 32 bands as shown in Figure A-2. Offset values are given

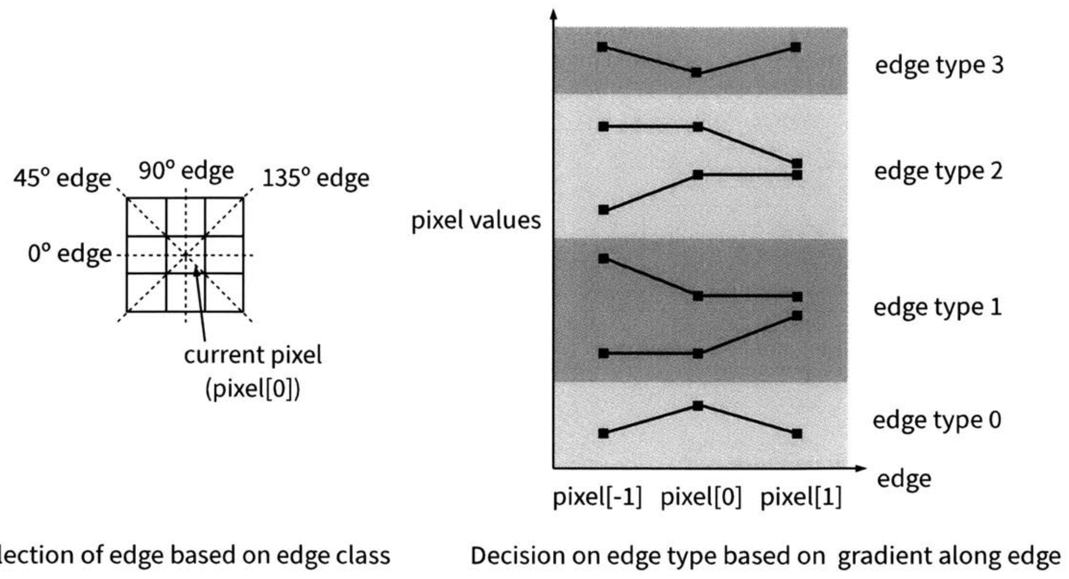


Figure A-3: SAO with edge offsets. Offset values are specified for 4 types of edges

for 4 contiguous bands. If the pixel value lies in any of these bands, the corresponding offset value is added. Band position is a 5-bit number indicating the first of the 4 contiguous bands. The 4 offset values are each 4-bit signed integers [-7 to 7].

3. Edge class and offset values: When edge offset is used, two immediate neighbors of the current pixel are used to determine what type of edge is present in the pixel values. The choice of the two neighbors is decided among one of four edge class (0°, 45°, 90°, 135°) as shown in Figure A-3. Based on the gradient in pixel values of the neighbors and the current pixel, one of four edge types is decided. Four offset values (3-bit unsigned integers) are specified, one for each edge type.

Figure A-4 shows the architecture of the SAO module. The core SAO computation module processes a 2×2 block per cycle using a 4×4 block of pixels around it as input. Local buffer is used to store within-CTU dependencies and dependencies between current CTU and the next CTU to the right. Line buffers communicate pixels and SAO parameters to and from the top and bottom neighbors. The SAO module was synthesized at 25 MHz and takes 9 kgates of area and 1 kB of SRAM.

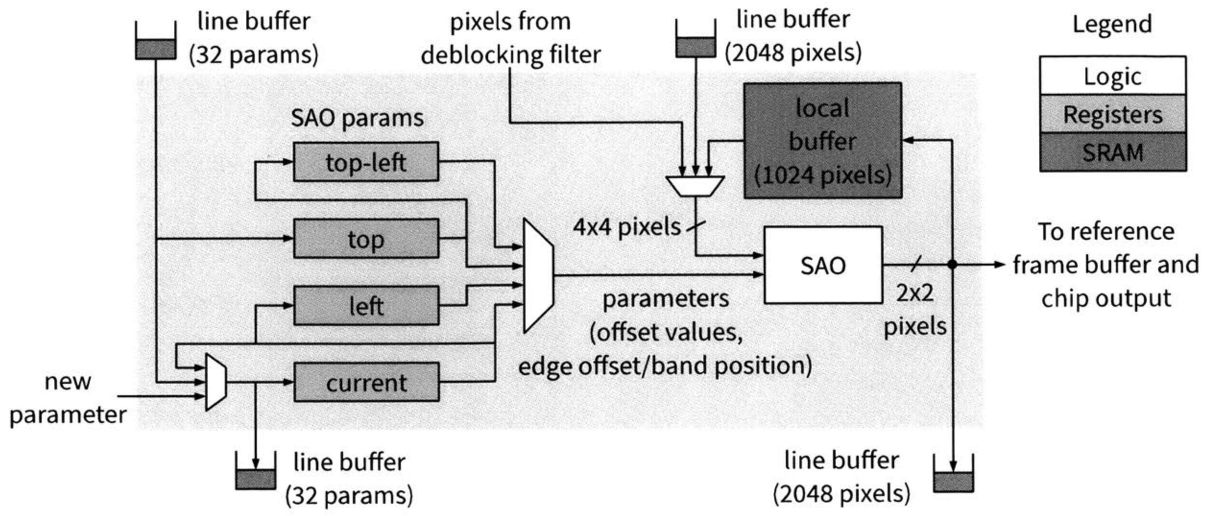

line buffer (32 params)
pixels from deblocking filter
line buffer (2048 pixels)
Legend
Logic
Registers
SRAM
SAO params
top-left
top
left
current
local buffer (1024 pixels)
4x4 pixels
SAO
parameters (offset values, edge offset/band position)
2x2 pixels
To reference frame buffer and chip output
new parameter
line buffer (32 params)
line buffer (2048 pixels)

Figure A-4: Architecture of SAO showing data movement of SAO parameters and pixels to/from neighbors

A.3 Interpolation Filter for Inter-Prediction

As explained in Section 2.1.2 and in chapter 5, the horizontal and vertical filters take up a significant portion of area and energy in the inter-prediction module. This section provides details about the interpolation filter design.

In previous work [14], a common interpolation filter was designed for luma and chroma pixels using hand-crafted multiplexed multiple constant multiplication. The filter took 8 luma or 4 chroma pixels as input to output a single pixel. To generate multiple pixels, parallel filters were used. As a result, this approach does not exploit computation reuse across filter outputs.

In this work, separate luma and chroma filters were designed. However, the filters output 4 filtered pixels allowing for computation reuse across filter outputs. The filter coefficients for fractional interpolation of luma pixels are as follows:

$$h_{0.25}[n] = \{-1, 4, -10, 58, 17, -5, 1\}$$

$$h_{0.5}[n] = \{-1, 4, -11, 40, 40, -11, 4, -1\}$$

$$h_{0.75}[n] = \{1, -5, 17, 58, -10, 4, -1\}$$

This gives the following constant multiplier factors: {1, 4, 5, 10, 11, 17, 40, 58}. These are genarated using a multiple constant multiplication (MCM) circuit generated through SPIRAL [21]. Only 4 adders are used as shown in Figure A-5. The circuit takes 6 kgates of area for 8-bit input and 16-bit output when synthesized at 25 MHz.

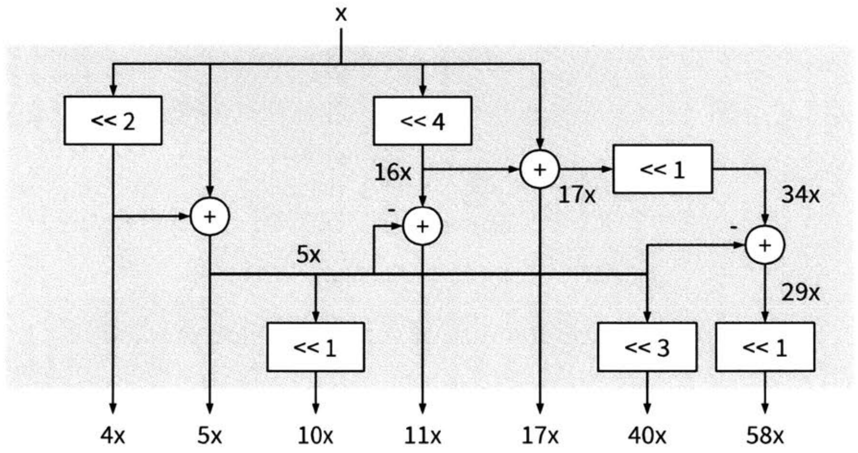


Figure A-5: implementation of multiplications for HEVC luma interpolation using MCM

A.4 Cache

This work uses a common cache for all inter-prediction requests coming from all 4 decoder cores. Figure A-6 shows the architecture of the cache. The cache is operated at 100 MHz while the decoder cores run at 25 MHz to ensure that the cache throughput is sufficient for all decoders.

The cache is composed of two parallel caches that store mutually exclusive regions in the main memory such that adjacent 4×4 pixel blocks always go to separate caches. Two 4×4 addresses are read from inter-prediction per cycle and, if their target caches are different, both addresses are dispatched to their respective caches. The tag-file in each cache is made of a 20kbit SRAM along with a hit/miss determination logic. The tag-file is set up for 3-way set associativity with FIFO update method. On a miss, the tag-file is immediately updated so that subsequent requests to the same address are evaluated as a hit. As all requests are processed sequentially, the subsequent

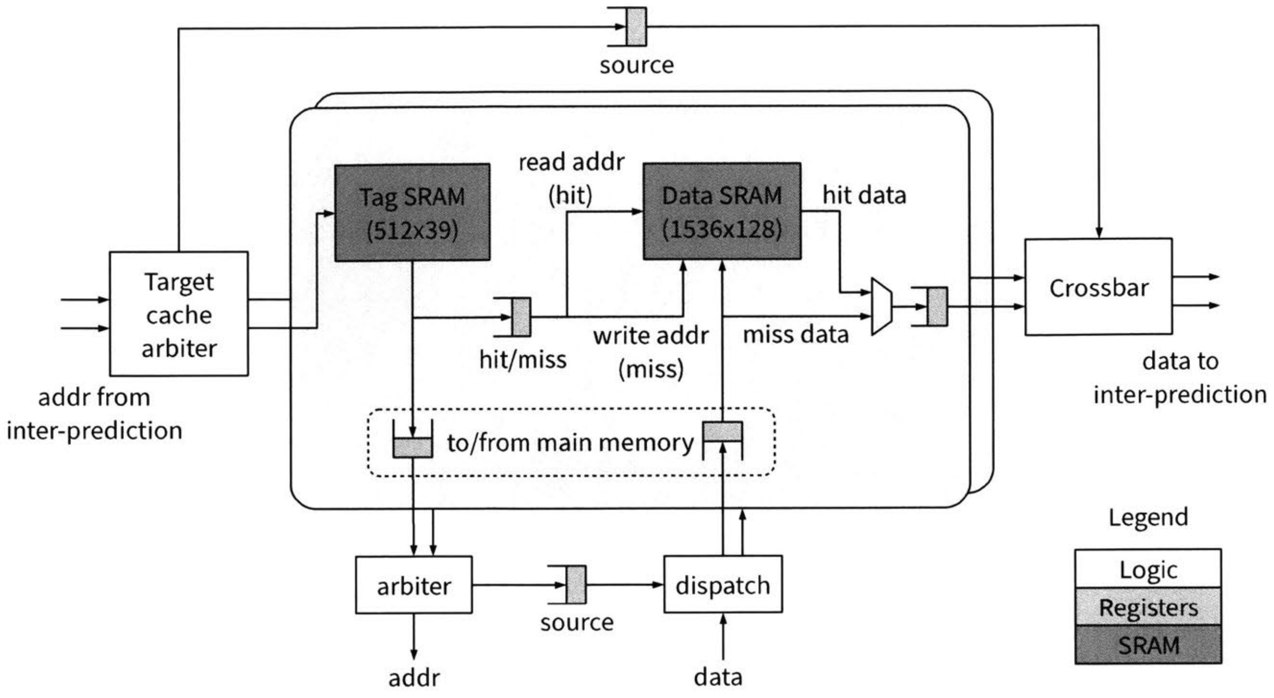


Figure A-6: Architecture of cache for inter-prediction

hits will wait until the first miss is resolved and correct data is available in the data SRAM. The cache line in the data SRAM is 128b corresponding to a 4×4 pixel block. A total of 3072 cache lines are stored in the two parallel caches.

Bypass FIFOs are used throughout the cache to minimize the hit latency. The minimum hit latency is 3 cycles and worst-case miss latency with eDRAM memory is 14 cycles. This is at the 100 MHz cache clock, so the 25 MHz inter-prediction modules see, at worst, a 4 cycle latency on read requests.

Bibliography

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2011 - 2016," May-2012. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[2] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012. DOI: 10.1109/TCSVT.2012.2221191.

[3] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012. DOI: 10.1109/TCSVT.2012.2221192.

[4] M. Alexsic, "Deep learning for mobile and embedded devices," In *Short Course on Machine Learning for Circuit Designers at 2017 Symposium on VLSI Circuits*, 2017.

[5] V. Sze, D. F. Finchelstein, M. E. Sinangil, and A. P. Chandrakasan, "A 0.7-V 1.8-mW H.264/AVC 720p Video Decoder," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 2943–2956, Nov. 2009, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5308724. DOI: 10.1109/JSSC.2009.2028933.

[6] C.-H. Tsai, H.-T. Wang, C.-L. Liu, Y. Li, and C.-Y. Lee, "A 446.6K-gates 0.55 - 1.2V H.265/HEVC decoder for next generation video applications," In *Solid-State Circuits Conference (A-SSCC), 2013 IEEE Asian*, 2013, pp. 305–308. DOI: 10.1109/ASSCC.2013.6691043.

[7] C.-C. Ju, T.-M. Liu, Y.-C. Chang, C.-M. Wang, H.-M. Lin, C.-Y. Cheng, C.-C. Chen, M.-H. Chiu, S.-J. Wang, P. Chao, M.-J. Hu, F.-C. Yeh, S.-H. Chuang, H.-Y. Lin, M.-L. Wu, C.-H. Chen, and C.-H.

Tsai, "A 0.2nJ/pixel 4K 60fps Main-10 HEVC decoder with multi-format capabilities for UHD-TV applications," In *European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014 - 40th*, 2014, pp. 195–198. DOI: 10.1109/ESSCIRC.2014.6942055.

[8] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, J. Zhou, S. Zhang, S. Kimura, T. Yoshimura, and S. Goto, "An 8K H.265/HEVC Video Decoder Chip With a New System Pipeline Design," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 113–126, Jan. 2017. DOI: 10.1109/JSSC.2016.2616362.

[9] M. Tikekar, C. T. Huang, C. Juvekar, V. Sze, and A. P. Chandrakasan, "A 249-Mpixel/s HEVC Video-Decoder Chip for 4K Ultra-HD Applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 61–72, Jan. 2014. DOI: 10.1109/JSSC.2013.2284362.

[10] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," ITU-T SG16 Q.6 Document, Austin, VCEG-M33, Apr. 2001.

[11] F. Bossen, "Common test conditions and software reference configurations," JCTVC, San Jose, CA, JCTVC-H1100, Feb. 2012.

[12] J. Vanne, M. Viitanen, T. Hamalainen, and A. Hallapuro, "Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1885–1898, 2012. DOI: 10.1109/TCSVT.2012.2223013.

[13] C. T. Huang, M. Tikekar, and A. P. Chandrakasan, "Memory-Hierarchical and Mode-Adaptive HEVC Intra Prediction Architecture for Quad Full HD Video Decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 7, pp. 1515–1525, Jul. 2014. DOI: 10.1109/TVLSI.2013.2275571.

[14] C. T. Huang, C. Juvekar, M. Tikekar, and A. P. Chandrakasan, "HEVC interpolation filter architecture for quad full HD decoding," In *2013 Visual Communications and Image Processing (VCIP)*, 2013, pp. 1–5. DOI: 10.1109/VCIP.2013.6706371.

[15] C. M. Fu, E. Alshina, A. Alshin, Y. W. Huang, C. Y. Chen, C. Y. Tsai, C. W. Hsu, S. M. Lei, J. H. Park, and W. J. Han, "Sample Adaptive Offset in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012. DOI:

10.1109/TCSVT.2012.2221529.

[16] Y. H. Chen and V. Sze, "A Deeply Pipelined CABAC Decoder for HEVC Supporting Level 6.2 High-Tier Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 5, pp. 856–868, May 2015. DOI: 10.1109/TCSVT.2014.2363748.

[17] D. Zhou, J. Zhou, J. Zhu, P. Liu, and S. Goto, "A 2Gpixel/s H.264/AVC HP/MVC video decoder chip for Super Hi-Vision and 3DTV/FTV applications," In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, 2012, pp. 224–226. DOI: 10.1109/ISSCC.2012.6176985.

[18] Micron, "DDR3 SDRAM system-power calculator." [Online]. Available: http://www.micron.com/products/support/power-calc.

[19] M. Karczewicz and E. Alshina, "JVET AHG report: Tool evaluation (AHG1)," 5th meeting of JVET, Geneva, CH, JVT-E0001, Jan. 2017.

[20] M. Tikekar, V. Sze, and A. Chandrakasan, "A Fully-Integrated Energy-Efficient H.265/HEVC Decoder with eDRAM for Wearable Devices," In *2017 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, 2017, pp. 1–2.

[21] M. Puschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo, "SPIRAL: code generation for DSP transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–275, 2005. DOI: 10.1109/JPROC.2004.840306.

[22] D. F. Finchelstein, "Low-power Techniques for Video Decoding," Thesis, Massachusetts Institute of Technology, 2009. DOI: 1721.1/52794.

[23] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," Feb. 2016, http://arxiv.org/abs/1602.01528.

[24] M. Budagavi and V. Sze, "IDCT pruning and scan dependent transform order," 6th meeting of ITU-T/ISO/IEC JCT-VC, Torino, Italy, JCTVC-F236, July 2011.

[25] T. Xanthopoulos and A. P. Chandrakasan, "A low-power IDCT macrocell for MPEG-2

MP@ML exploiting data distribution properties for minimal activity," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 5, pp. 693–703, May 1999. DOI: 10.1109/4.760381.

[26] M. Abeydeera and A. Pasqual, "HEVC inverse transform architecture utilizing coefficient sparsity," In *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 4848–4852. DOI: 10.1109/ICIP.2015.7351728.

[27] M. Tikekar, C. T. Huang, V. Sze, and A. Chandrakasan, "Energy and area-efficient hardware implementation of HEVC inverse transform and dequantization," In *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 2100–2104. DOI: 10.1109/ICIP.2014.7025421.

[28] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622. DOI: 10.1109/MICRO.2014.58.

[29] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang, and M. Bohr, "A 4.0 GHz 291Mb voltage-scalable SRAM design in 32nm high-K metal-gate CMOS with integrated power management," In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, 2009, pp. 456–457, 457a. DOI: 10.1109/ISSCC.2009.4977505.

[30] R. J. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski, "A 32nm 3.1 billion transistor 12-wide-issue Itanium processor for mission-critical servers," In *2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 84–86. DOI: 10.1109/ISSCC.2011.5746230.

[31] G. Wang, D. Anand, N. Butt, A. Cestero, M. Chudzik, J. Ervin, S. Fang, G. Freeman, H. Ho, B. Khan, B. Kim, W. Kong, R. Krishnan, S. Krishnan, O. Kwon, J. Liu, K. McStay, E. Nelson, K. Nummy, P. Parries, J. Sim, R. Takalkar, A. Tessier, R. M. Todi, R. Malik, S. Stiffler, and S. S. Iyer, "Scaling deep trench based eDRAM on SOI to 32nm and Beyond," In *2009 IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4. DOI: 10.1109/IEDM.2009.5424375.

[32] N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, M. Lal, A. Deval, J. Douglas, M. Elassal, A. Nalamalpu, T. M. Wilson, M. Merten, S. Chennupaty, W. Gomes, and R. Kumar, "Haswell: A family of IA 22nm processors," In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 112–113. DOI:

10.1109/ISSCC.2014.6757361.

[33] F. Hamzaoglu, U. Arslan, N. Bisnik, S. Ghosh, M. B. Lal, N. Lindert, M. Meterelliyoz, R. B. Osborne, J. Park, S. Tomishima, Y. Wang, and K. Zhang, "A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology," In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 230–231. DOI: 10.1109/ISSCC.2014.6757412.

[34] "30nm DRAM Comparison," *TechInsights*. [Online]. Available: https://www.techinsights.com/uploadedFiles/Public_Website/Content_-_Primary/Marketing/2012/Micron_makes_first_entry_into_30nm-class_SDRAM/30nm-DRAM-Comparison.pdf. [Accessed: 25-Jun-2017].

[35] "16Gb: X4, x8 TwinDie DDR4 SDRAM." [Online]. Available: https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr4/ddr4_16gb_1_2v_twindie_x4x8.pdf. [Accessed: 25-Jun-2017].

[36] "1Gb: x4, x8, x16 DDR3L SDRAM Description." [Online]. Available: https://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/1gb_1_35v_ddr3l.pdf.

[37] D. M. Mathew, É. F. Zulian, S. Kannoth, M. Jung, C. Weis, and N. Wehn, "A bank-wise dram power model for system simulations," In *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2017, pp. 5:1–5:7. DOI: 10.1145/3023973.3023978.

[38] L. Guo, D. Zhou, and S. Goto, "A New Reference Frame Recompression Algorithm and Its VLSI Architecture for UHDTV Video Codec," *IEEE Transactions on Multimedia*, vol. PP, no. 99, pp. 1–1, 2014. DOI: 10.1109/TMM.2014.2350256.

[39] D. Zhou, L. Guo, J. Zhou, and S. Goto, "Reducing power consumption of HEVC codec with lossless reference frame recompression," In *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 2120–2124. DOI: 10.1109/ICIP.2014.7025425.

[40] M. Budagavi and M. Zhou, "Video coding using compressed reference frames," In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2008, pp. 1165–1168. DOI: 10.1109/ICASSP.2008.4517822.

[41] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*,

2004, pp. 212–223. DOI: 10.1109/ISCA.2004.1310776.

[42] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed Compressed Caches," In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 331–342. DOI: 10.1109/MICRO.2014.41.

[43] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate Compression: Practical Data Compression for On-chip Caches," In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012, pp. 377–388. DOI: 10.1145/2370816.2370870.

[44] K. Kawakami, J. Takemura, M. Kuroda, H. Kawaguchi, and M. Yoshimoto, "A 50% Power Reduction in H.264/AVC HDTV Video Decoder LSI by Dynamic Voltage Scaling in Elastic Pipeline," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vols. E89-A, no. 12, pp. 3642–3651, Dec. 2006, http://dx.doi.org/10.1093/ietfec/e89-a.12.3642. DOI: 10.1093/ietfec/e89-a.12.3642.

[45] V. Sze and M. Budagavi, "High Throughput CABAC Entropy Coding in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1778–1791, Dec. 2012. DOI: 10.1109/TCSVT.2012.2221526.

[46] C. Dong, C. C. Loy, and X. Tang, "Accelerating the Super-Resolution Convolutional Neural Network," Aug. 2016, http://arxiv.org/abs/1608.00367.

[47] Z. Zhang and V. Sze, "FAST: Free Adaptive Super-Resolution via Transfer for Compressed Videos," Mar. 2016, http://arxiv.org/abs/1603.08968.

[48] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full Resolution Image Compression with Recurrent Neural Networks," Aug. 2016, http://arxiv.org/abs/1608.05148.