ENERGY AND AREA-EFFICIENT HARDWARE IMPLEMENTATION OF HEVC INVERSE TRANSFORM AND DEQUANTIZATION

Mehul Tikekar[1], *Chao-Tsung Huang*[2], *Vivienne Sze*[1], *Anantha Chandrakasan*[1]

[1]Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[2]National Tsing Hua University, Hsinchu 30013, Taiwan

ABSTRACT

High Efficiency Video Coding (HEVC) inverse transform for residual coding uses 2-D 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. In this paper, an energy and area-efficient VLSI architecture of an HEVC-compliant inverse transform and dequantization engine is presented. We implement a pipelining scheme to process all transform sizes at a minimum throughput of 2 pixel/cycle with zero-column skipping for improved throughput. We use data-gating in the 1-D Inverse Discrete Cosine Transform engine to improve energy-efficiency for smaller transform sizes. A high-density SRAM-based transpose memory is used for an area-efficient design. This design supports decoding of 4K Ultra-HD (3840×2160) video at 30 frame/sec. The inverse transform engine takes 98.1 kgate logic, 16.4 kbit SRAM and 10.82 pJ/pixel while the dequantization engine takes 27.7 kgate logic, 8.2 kbit SRAM and 1.10 pJ/pixel in 40 nm CMOS technology. Although larger transforms require more computation per coefficient, they typically contain a smaller proportion of non-zero coefficients. Due to this trade-off, larger transforms can be more energy-efficient.

***Index Terms*—** HEVC, Inverse Discrete Cosine Transform, Transpose Memory, Data Gating

1. INTRODUCTION

High-Efficiency Video Coding (HEVC) achieves a 50% reduction in bit-rate over H.264/AVC at the same visual quality. A key feature in HEVC is the introduction of large 16×16 and 32×32 inverse discrete cosine transforms (IDCTs), a new 4×4 inverse discrete sine transform (IDST) and high-precision 4×4 and 8×8 IDCTs. The large transforms contribute to 6.7% - 10.1% bit-rate reduction for 1080p (1920×1080) and larger video sequences, and the increased precision in smaller transforms contribute 0.3% - 1.2% [1]. These new features of HEVC raises several challenges for hardware implementations:

1. HEVC uses Transform Units (TUs) of size 4×4, 8×8, 16×16, and 32×32 pixels. This variety of TU sizes complicates the design of control logic as TUs of different sizes take different number of cycles for processing.

2. Like H.264/AVC, the 2-D transforms in HEVC are separable into 1-D transforms along the columns and rows. The N-pt 1-D IDCT used in an $N \times N$ 2-D IDCT can be viewed as the product of a $N \times N$ transform matrix with $N \times 1$ input coefficients. This requires N multiplications per coefficient. Hence, the largest IDCT in HEVC (32×32) takes 4× the number of multiplications as the largest IDCT in H.264/AVC (8×8). Further, the increased precision in HEVC transforms doubles the cost of each multiplication. Combined together, HEVC transform logic has 8× the computational complexity of H.264/AVC which affects both area and energy.

3. An intermediate memory is needed to store the TU between the column and row transforms operation. This memory must perform a transposition i.e. columns are written to it and rows are read out. Previous designs for H.264/AVC used register arrays due to the small TU sizes. These do not scale very well to the higher TU sizes of HEVC and one must look to denser memories such as SRAM to achieve an area-efficient implementation. However, the higher density of SRAMs comes at the cost of lesser memory throughput and lesser flexibility in read-write patterns.

In this paper, we propose the following techniques to improve the throughput, energy and area of an HEVC inverse transform engine:

1. A pipelining scheme is devised to handle the variety of TU sizes. A high throughput is achieved using zero-column skipping and bypassing residues of smaller TUs.

The authors would like to thank Texas Instruments for supporting the project and TSMC University Shuttle Program for chip fabrication. Mehul Tikekar was supported by the NSF.

2. To address the computational complexity of the HEVC transform logic, an Multiple Constant Multiplication (MCM) [2] based method had previously been proposed where the area depends only on the number of unique constants in the IDCT matrices [3]. In this work, the design is extended using data-gating to improve energy-efficiency as well.

3. An SRAM-based transpose memory is proposed that provides the required throughput inspite of SRAM limitations mentioned previously using parallel SRAM banks and a register-based cache.

These techniques are used to implement an HEVC inverse transform engine with a worst-case throughput of 2 pixel/cycle. At 200 MHz, this is sufficient for decoding 4K Ultra-HD video at 30 frames/sec ($2 \times 200 \times 10^6 > 3840 \times 2160 \times 1.5 \times 30$). A dequantization engine supporting all HEVC scaling list types is briefly described and area and energy results for the complete design in 40 nm CMOS technology are presented finally.

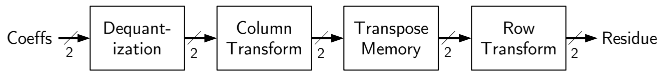
2. HIGH-THROUGHPUT PIPELINING SCHEME FOR ALL TU SIZES

In general, two high-level architectures are possible for a 2 pixel/cycle inverse transform [4]. The first one, shown in Fig. 1(a) 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.

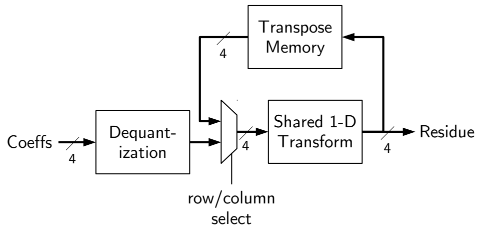
With these considerations, the second architecture, shown in Fig. 1(b) is preferred. This uses a single transform block capable of 4 pixel/cycle for both row and column transform. The block works on a single TU at a time, processing all the columns first and then the rows. Hence, the transpose memory 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 Fig. 2. The partial 1-D transform block includes the 4 pixel/cycle IDCT and IDST blocks. The 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.

The design includes a separate 4-element 4 pixel/element FIFO (marked “Residue 4×4” in Fig. 2) to store residues for



(a) Separate row and column transforms



(b) Shared transform block

Fig. 1: Possible high-level architectures for inverse transform (bus-widths are in coefficients/residue pixels). To achieve an overall 2 pixel/cycle throughput, the shared 1-D transform block must be designed for 4 pixel/cycle.

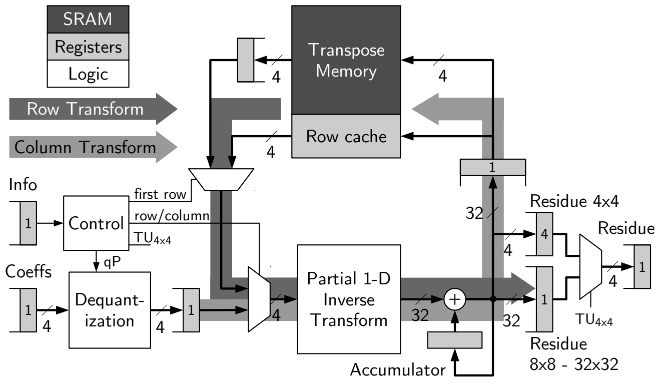


Fig. 2: Architecture of inverse transform based on Fig. 1(b)

4×4 TUs. This is to avoid a stall when a 4×4 TU immediately follows a 32×32 TU. After the last row of the 32×32 TU is computed, it takes 8 cycles to write it out. During those cycles, the partial 1-D transform block computes the column transforms of the 4×4 TU and is ready to write the first row out after 4 cycles itself. To avoid stalling the pipeline while the last row of the 32×32 TU is being written out, the residues of the 4×4 TU are saved in the separate residue FIFO.

The proposed design implements zero-column skipping based on IDCT pruning [5] in which the 1-D transform is skipped for columns that have all zero coefficients. This reduces cycle-count by 27% to 66%. TUs with larger sizes and higher quantization benefit more from zero-column skipping since they have 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%.

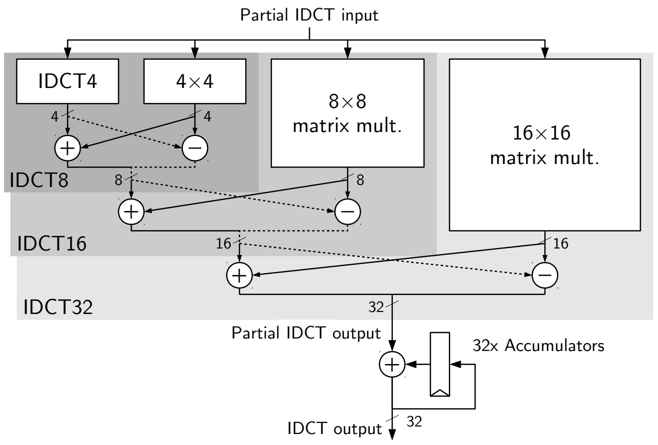


Fig. 3: Shared 4-pt to 32-pt 1-D IDCT from Fig. 2. Dotted lines denote the path which is data-gated to reduce spurious switching when computing smaller transforms.

TU size	Energy without gating (pJ/pixel)	Energy with gating (pJ/pixel)	Energy savings
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%

Table 1: Power reduction in inverse transform engine using data-gating

3. ENERGY-EFFICIENT 1-D INVERSE DISCRETE COSINE TRANSFORM

Our 1-D IDCT is based on the design from [6] 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 Fig. 3, 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 (by setting their outputs to 0) 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 benefits of gating are seen in Table 1 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.

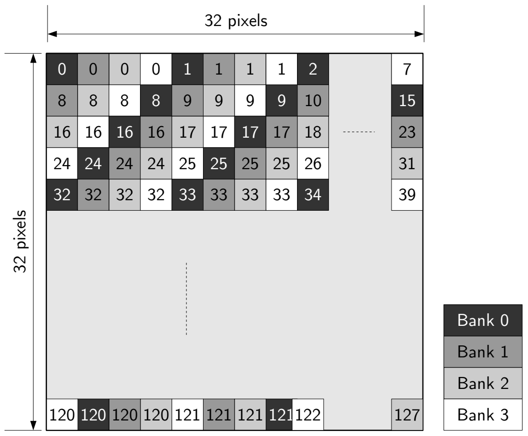


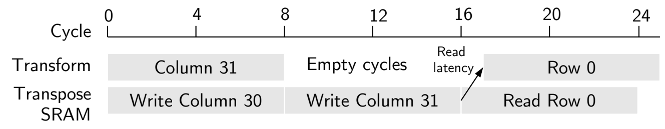
Fig. 4: Mapping a 32×32 TU to 4 SRAM banks for transpose operation. The color of each pixel denotes the bank and the number denotes the bank address.

4. HIGH-DENSITY AND HIGH-THROUGHPUT TRANSPOSE MEMORY

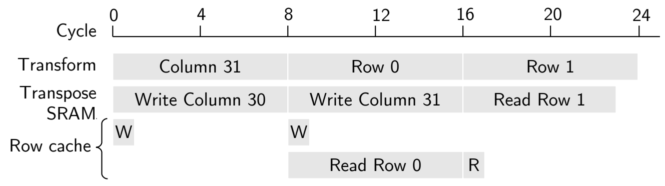
The transform block uses a 16-bit precision input for both row and column transforms. The transpose memory must be sized for 32×32 TU which means a total size of $16 \times 32 \times 32 = 16.4$ kbits. In comparison, H.264/AVC decoder designs require a much smaller transpose memory: $16 \times 8 \times 8 = 1$ kbits. A 16.4 kbit memory with the read circuit for transpose operation is prohibitively large (125 kgate) when implemented with registers and multiplexers. Hence, an SRAM implementation is needed. The main disadvantage of the SRAM is that it is less flexible than registers. A register array allows reading and writing to arbitrary number of bits at arbitrary locations - ideal for the transpose operation. In comparison, a single-port SRAM can access only one entry at a time. Multi-port SRAMs can access multiple entries simultaneously but they incur significant area penalties.

We implement the 4 pixel/cycle 32×32 transpose memory as 4 single-port banks of 256 entries. The pixels in a 32×32 TU are mapped to locations in the 4 banks as shown in Fig. 4. By ensuring that 4 adjacent pixels in any row or column are stored in different SRAM banks, it is possible to write along columns and read along rows by supplying different addresses to the 4 banks.

After a 32-pt column transform is computed, it takes 8 cycles for the result to be written to the transpose SRAM, during which time the transform block processes the next column. This is shown in cycles $0-7$ in Fig. 5(a) where result of column 30 is written to the SRAM while the transform block works on column 31. However, when the last column is processed, the transform block must wait for it to be written to the SRAM before it can begin processing the row. This results in a delay of 9 cycles for 32×32 TU. In general, for an $N \times N$ TU, this delay is equal to $N/4 + 1$ cycles which is 1.75% (for $N = 32$) to 25% (for $N = 4$) of the total $N^2/2$



(a) Pipeline stall due to transpose SRAM delay for 32×32 TU



(b) Row caching to avoid stall

Fig. 5: Eliminating stall cycles in SRAM transpose memory with a register-based row cache

cycles. This delay is avoided through the use of a row cache that stores the first $N + 4$ pixels in registers. This enables full concurrency as shown in Fig. 5(b). The first pixel in each column is saved to the row cache so that the first row can be read from the cache while the last column is being stored in the SRAM.

This transpose memory design using SRAM scales very well for lower throughputs. A p pixel/cycle transpose memory would need p banks each with $1024/p$ entries. For higher throughputs, more banks are needed with fewer entries in each bank. Such short SRAMs have an area overhead of sense-amplifiers, row decoders, etc., and a register-based transpose memory [7], [8] is more efficient.

5. DEQUANTIZATION ENGINE

This work includes a dequantization engine that supports all three scaling list types in HEVC - no scaling list, default scaling list and custom lists. Custom lists are stored in an 8 kbit single-port SRAM while the default scaling list is stored in a separate ROM to avoid unnecessary power consumption from SRAM accesses for default scaling.

6. RESULTS

Breakdown of the logic area at 200 MHz clock frequency in TSMC 40 nm technology is given in Table 2. The total area is 128 kgate of logic and 24.6 kbits of SRAM. Table 2 also shows the energy/pixel as estimated from post-layout simulation for processing 2000 TUs (173360 pixels with a mixture of all TU sizes) from the ParkScene test sequence at QP 32.

The switching power in a digital circuit depends on the amount of computation as well as the statistics of the data being computed upon. Larger TU sizes require much more computation per pixel which increases energy/pixel. However, larger TUs are also better at compressing video data. As a result, they typically have fewer non-zero coefficients (sparsity) which results in lower switching activity. This effect is seen in Table 3 where the energy/pixel with a fixed 10% sparsity and with a variable sparsity obtained from actual video are compared.

This data suggests that, from a power perspective, the increased computation in larger TU sizes can be offset by the fewer number of non-zero coefficients.

Module	Logic area (kgate)	SRAM (kbit)	Energy (pJ/pixel)
Partial 1-D transform	65.3	0	4.88
Transpose Memory	5.2	16.4	2.31
Row cache	5.2	0	0.22
Accumulator	13.4	0	1.84
FIFOs and control	9.0	0	1.57
Dequantization	27.7	8.2	1.10
Total	125.8	24.6	11.92

Table 2: Area and energy breakdown for complete design

TU size	Sparsity of TUs	Energy (pJ/pixel)	Sparsity of TUs	Energy (pJ/pixel)
4×4	10%	11.2	15.0%	9.7
8×8	10%	22.2	4.5%	11.4
16×16	10%	38.1	2.4%	12.6
32×32	10%	57.1	0.5%	12.2

Table 3: Energy for different TU sizes with fixed and variable sparsities.

7. CONCLUSION

In this paper, 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. A pipelining scheme is developed to manage all TU sizes in HEVC at a worst-case throughput of 2 pixel/cycle. Zero-column skipping reduces cycle-count by 27%-66% over the worst case. The design of a transpose memory using a combination of SRAM for high density and registers for high throughput is explained. Finally, a dequantization engine for all scaling list types is briefly described. This design takes 126 kgates of logic and and consumes 7.8 mW of power (or 11.9 pJ/pixel). Data and explicit clock-gating improves the energy efficiency of the shared transform logic. The proposed techniques are summarized in Table 4.

Designs	Logic area (kgates)	Energy (pJ/pixel)	Throughput (pixel/cycle)
Base design [6]	118.5	20.94	2.00
Gating	123.1	17.62	2.00
Zero-column skip	121.6	14.79	3.26
Complete design	125.8	11.92	3.26

Table 4: Summary of proposed techniques

8. REFERENCES

[1] M. Budagavi, A. Fuldseth, G. Bjontegaard, V. Sze, and M. Sadafale, “Core Transform Design for the High Efficiency Video Coding (HEVC) Standard,” *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 6, pp. 1029–1041, 2013.

[2] M. Potkonjak, M.B. Srivastava, and A. Chandrakasan, “Efficient Substitution of Multiple Constant Multiplications by Shifts and Additions using Iterative Pairwise Matching,” in *Design Automation, 1994. 31st Conference on*, june 1994, pp. 189 – 194.

[3] M. Tikekar, C.-T. Huang, C. Juvekar, and Chandrakasan A., “JCTVC-G265: Core transform property for practical throughput hardware design,” *Joint Collaborative Team on Video Coding (JCT-VC)*, 2011.

[4] D. F. Finchelstein, *Low-power Techniques for Video Decoding*, Thesis, Massachusetts Institute of Technology, 2009.

[5] M. Budagavi and V. Sze, “IDCT pruning and scan dependent transform order,” *Joint Collaborative Team on Video Coding (JCT-VC)*, 2011.

[6] M. Tikekar, Chao-Tsung Huang, C. Juvekar, V. Sze, and A.P. Chandrakasan, “A 249-Mpixel/s HEVC Video-Decoder Chip for 4K Ultra-HD Applications,” *Solid-State Circuits, IEEE Journal of*, vol. 49, no. 1, pp. 61–72, 2014.

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

[8] P.K. Meher, S.Y. Park, B.K. Mohanty, K.S. Lim, and C. Yeo, “Efficient Integer DCT Architectures for HEVC,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 24, no. 1, pp. 168–178, Jan 2014.