Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 425173, 17 pages
doi:10.1155/2009/425173

Research Article

Low-Power Bitstream-Residual Decoder for H.264/AVC Baseline Profile Decoding

Ke Xu and Chiu-Sing Choy

Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong

Correspondence should be addressed to Ke Xu, kexu@ee.cuhk.edu.hk

Received 11 July 2009; Revised 21 October 2009; Accepted 2 December 2009

Recommended by Leonel Sousa

We present the design and VLSI implementation of a novel low-power bitstream-residual decoder for H.264/AVC baseline profile. It comprises a syntax parser, a parameter decoder, and an Inverse Quantization Inverse Transform (IQIT) decoder. The syntax parser detects and decodes each incoming codeword in the bitstream under the control of a hierarchical Finite State Machine (FSM); the IQIT decoder performs inverse transform and quantization with pipelining and parallelism. Various power reduction techniques, such as data-driven based on statistic results, nonuniform partition, precomputation, guarded evaluation, hierarchical FSM decomposition, TAG method, zero-block skipping, and clock gating , are adopted and integrated throughout the bitstream-residual decoder. With innovative architecture, the proposed design is able to decode QCIF video sequences of 30 fps at a clock rate as low as 1.5 MHz. A prototype H.264/AVC baseline decoding chip utilizing the proposed decoder is fabricated in UMC 0.18 μm 1P6M CMOS technology. The proposed design is measured under 1 V $\sim$ 1.8 V supply with 0.1 V step. It dissipates 76 μW at 1 V and 253 μW at 1.8 V.

1. Introduction

The Motion Picture Experts Group and Video Coding Experts Group (MPEG and VCEG) have jointly developed a new standard named as H.264/AVC that outperforms the earlier MPEG-4 and H.263 standards, and provides much better compression of video images under the same bit rate. This significant coding gain is achieved at the expense of higher codec complexity.

One of the major differences between H.264/AVC and previous standards is the way that the quantized transform coefficients are handled. A more efficient method called Context-Adaptive Variable Length Coding (CAVLC) is employed. In this scheme, VLC tables are switched according to already transmitted syntax elements. Since these VLC tables are specifically designed to match the corresponding image statistic, the entropy coding performance is improved in comparison to schemes using only a single VLC table [1].

Besides CAVLC, H.264/AVC adopts variable block size motion prediction to provide more flexibility. The intra prediction can be applied either on 4×4 blocks individually or on entire 16×16 macroblock (MB) as a whole. The interprediction is based on a tree-structure where the motion vector and prediction can adopt various block sizes and partitions ranging from 16×16 MB to 4×4 block. To identify these prediction modes, motion vectors, and partitions, H.264/AVC specifies a very complex algorithm to derive them from their neighbors. The deblocking filter in H.264/AVC depends on a parameter called Boundary Strength (BS) to determine whether current block edge should be filtered. The derivation of the BS is highly adaptive which relies on the modes and coding conditions of the adjacent blocks.

The transform/inverse transform also operates on blocks of 4×4 pixels to match the smallest block size. The transform is still Discrete Cosine Transform (DCT) but with some fundamental differences compared to those in previous standards [2]. In addition, H.264/AVC employs a hierarchical transform architecture, as depicted in [3].

In general, H.264/AVC defines a hybrid block-based video codec with complex coding techniques to trade high

computational complexity for low bit rate. However, its intended applications, such as video transmission and play back on mobile terminals, place great demands on lowering power consumption to support real-time video decoding on battery-powered devices. Conventional implementations like μP or DSP cannot meet this requirement. For example, the design reported in [4] employs a 130 MHz ARM996 processor and is only capable of QCIF decoding at 7.5 fps. Even if some software solutions can achieve QCIF at 30 fps, the power consumption is relatively large and may not be suitable for battery-powered applications. A hardwired decoder targeting low power consumption is thus indispensable.

Although motion compensation and in-loop filter are the bottlenecks in the entire H.264/AVC decoding process, the bitstream-residual decoder which is the subject of this paper contributes to around 40% and 20% of total chip's area and power, respectively. A lot can still be gained if these costs are carefully optimized. In addition, the bitstream-residual decoder serves as a controller for the datapath to follow. Designs in [5–8] are representative examples of ASIC implementations of H.264/AVC decoder. However, they require an external RISC or host processors for syntax parsing and system control. Since we are only concerned with baseline profile, it is possible and advantageous to design a completely hardwired and self-contained decoder which does not need additional help from microprocessor. Furthermore, the proposed decoder is designed to function properly under various supply voltages that cannot be found in [6–8].

This paper shows how to integrate low-power strategies in the design of a power-efficient bitstream-residual decoder. Since there is no universal power saving approach that will work for all designs, power saving techniques suitable for various modules are proposed and integrated to achieve optimal results. Various low-power design techniques, such as statistic-based data driven decoding, hierarchical FSM decomposition, and nonuniform Look-Up-Table (LUT) partition giving rise to smaller tables are employed in the design of the syntax parser. A self-adaptive decoding sequence, coupled with zero-block-aware methodology, is applied in both the CAVLC and IQIT decoding. By making use of zero codewords, substantial amount of power can be saved. Both pipelining and parallelism are adopted in the IQIT decoder to increase the throughput and provide the headroom for voltage scaling. A TAG method is proposed for MV decoding. Signal and clock gating are widely adopted throughout the design of the whole bitstream-residual decoder.

The rest of the paper is organized as follows. The decoding pipeline and decoder architecture are described in Section 2. The design of the syntax parser is illustrated in Section 3. Basic algorithm and the parameter decoder design are presented in Section 4. Related background and the design of a parallel architecture for IQIT, together with the low-power techniques employed, are described in Section 5. We present the implementation details, measurement results, and comparisons in Section 6. Finally, we conclude the paper in Section 7.

2. System Architecture

2.1. System Block Diagram. The whole bitstream-residual decoder architecture is depicted in Figure 1. A hierarchical FSM resides in the syntax parser whose states are one-to-one mapped to each possible syntax element. Under the control of the FSM, the syntax parser processes the codewords of all levels in a proper and smooth order. The FSM also directs the decoding steps of the hybrid-length decoder and the IQIT decoder. The CAVLC decoder, which is part of the hybrid-length decoder, is only invoked when handling codewords at residual-level. Its outputs are fed to the IQIT decoder to construct residual blocks. According to current block type (DC or AC, luma or chroma) indicated by the FSM state, the IQIT decoder selects the appropriate butterfly algorithm and computes pixel residue on a 4×4 basis. According to the output of hybrid-length decoder, together with parameters of neighboring blocks, parameter decoder generates the necessary prediction control parameters, like Prediction Mode (PM), Motion Vector (MV), and Boundary Strength (BS).

2.2. Pipeline Architecture. The bitstream-residual decoder utilizes a 4×4 block level pipeline shown in Figure 2. Compared with a 16×16 macroblock level pipeline, it has three advantages. First, it matches the smallest block size specified in H.264/AVC. Second, temporary memory, such as buffer or SRAM for storing intermediate data, can be substantially smaller because one only needs to save one 4×4 block at a time instead of 16×16. Third, it greatly facilitates data reuse of neighboring blocks during prediction. These advantages make a substantial contribution to power savings. The IQIT decoder is also pipelined to increase the throughput. For pipeline of other blocks like prediction, please refer to [9].

3. Syntax Parser

3.1. Parser Structure. The syntax parser consists of several building blocks [10]. The circular bitstream buffer serves as a bridge between the off-chip bitstream RAM and the on-chip hybrid-length decoder. It holds the current 128-bit bitstream to be processed and a refilling mechanism helps to obtain new bitstream when necessary. The heading one detector detects the first appeared "1" in an encoded codeword for both the Exp-Golomb decoder and the CAVLC decoder. For syntax at residual level, CAVLC decoder is invoked; otherwise the syntax elements are processed either in fixed-length or Exp-Golomb decoder. The FSM orchestrates the decoding steps of the whole parser. Details of these building blocks are described in the following sections.

3.2. Circular Bitstream Buffer. Figure 3 describes the proposed circular buffer, interfacing between the off-chip bitstream RAM and the on-chip hybrid-length decoder. It keeps the current 128-bit bitstream to be processed. To accommodate codewords of hybrid-length without a fixed byte boundary in the input bitstream, the bitstream buffer is addressed in bit.

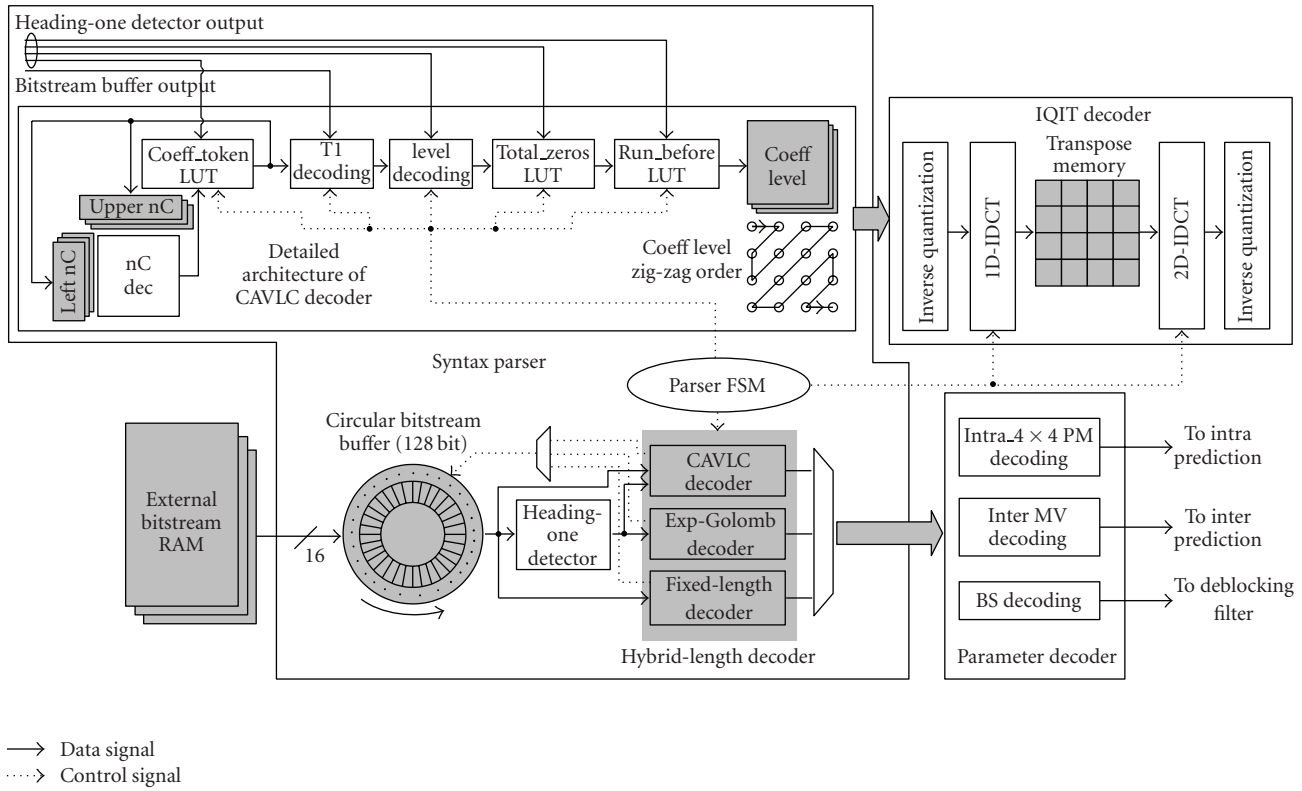


Figure 1: Bitstream-residual decoder block diagram.

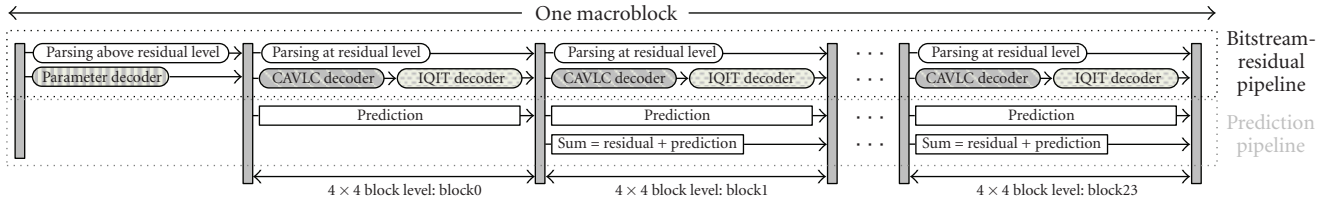


Figure 2: Pipeline architecture of entire decoder system.

The working principle of the circular buffer can be modeled by two hardware "threads" which manage the communication among the off-chip RAM, the bitstream buffer, and the hybrid-length decoder. Thread1 controls the bitstream flow from the RAM to the buffer, while thread2 controls the bitstream flow from the buffer to the decoder, operating like a sliding window. With independent operations of the two threads, buffer refill and read are independent to each other. Compared with the line-buffer architecture used in [11], the proposed design does not involve any data movement within the buffer; thus a great deal of power can be saved.

3.3. Heading One Detector. Based on the definition of Exp-Golomb code, if the first appeared "1" inside an encoded codeword is located, the whole codeword can be successfully extracted and decoded. A "first 1 decoder" was proposed in [12] which splits the 16-bit input equally into four parts and each part decides whether there is a "1" inside. However, no power optimization was realized since this scheme treated all the 16 bits with equal importance. From the SystemC modeling, statistical distribution of heading one's position was obtained as in Table 1, which indicates that the heading one appears within the first two positions (0∼1) with 80% possibility and the first six positions (0∼5) with 99% possibility.

According to the above analysis, a priority-based heading one detector with nonuniform input bits partition, depicted in Figure 4(b), is proposed in [13]. By selectively turning on the relatively larger decoder unit "dec4" (20% active) and "dec10" (1% active), power can be saved since there is often no need to search all the 16 input bits. By constructing [12], which is shown in Figure 4(a) with the same technology, postlayout power simulation shows that our design is three-times more power efficient.

3.4. Complex FSM Design. The syntax parser performs hybrid decoding of fixed-length and variable-length syntax

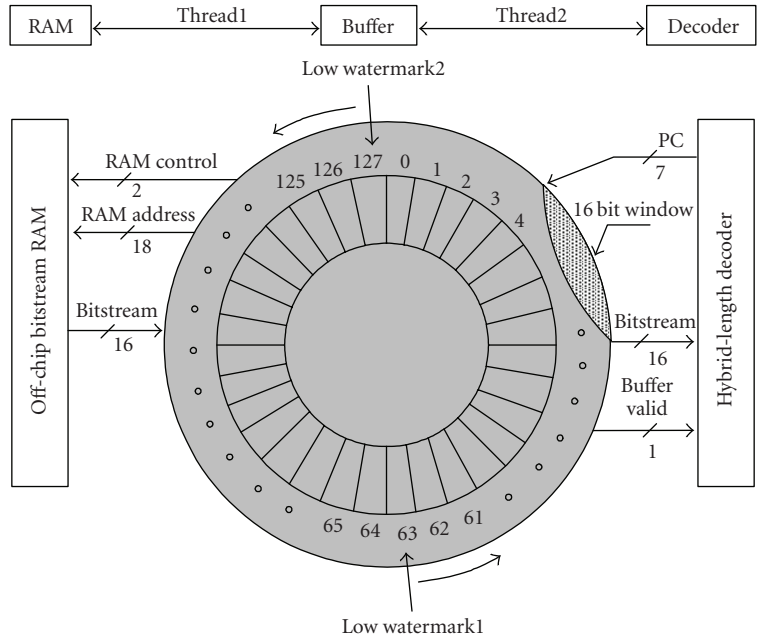


Figure 3: Circular bitstream buffer.

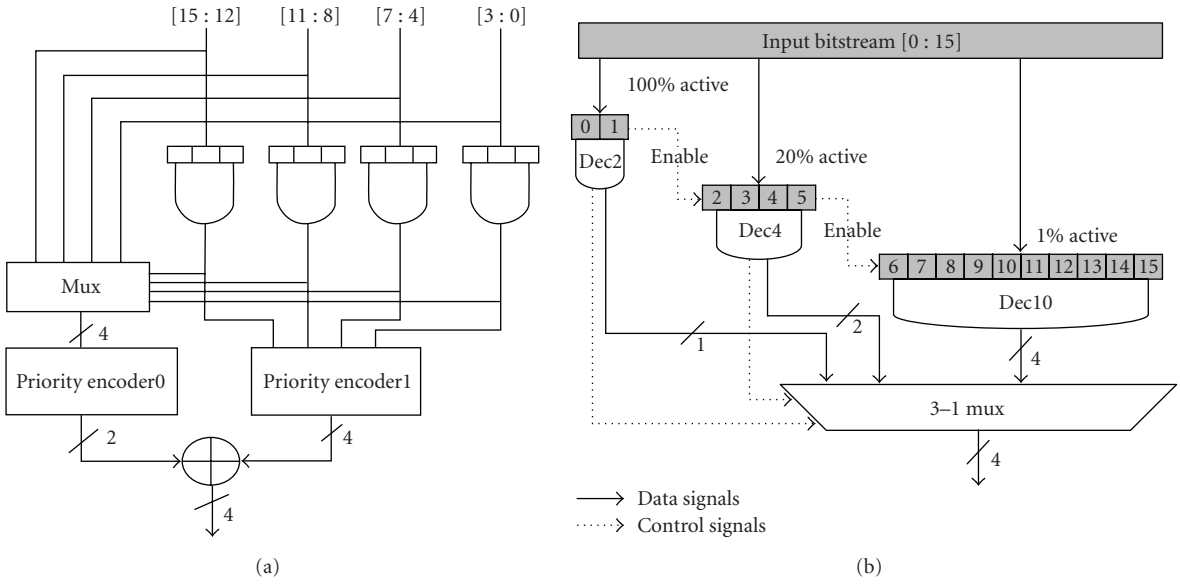


Figure 4: Heading one detector comparison.

elements. Designs in [5–8, 10] rely on external RISCs for codeword decoding higher than MB level, which lead to additional hardware cost, as well as IO power consumption. In our design, a complex FSM is utilized to orchestrate the syntax decoding at all levels. The type of each incoming codeword can be indicated by an FSM state. For baseline profile decoding, there are 25 syntax tables and 187 states needed to be addressed. A flattened FSM with many states is not a low power option because the whole FSM would switch all the time and the control/data flow is extremely complex and power consuming. A hierarchically decomposed FSM, consisting of 13 small sub-FSMs, is described in Figure 5. The total number of FSM states is reduced from original 187 to 107 (43% reduction) by FSM decomposition and states merging.

Further benefits can be gained by attending to some particularities in state transitions. We use probability-related Gray coding and variable-length state assignment to achieve minimum hamming distance between two possible adjacent FSM states. In addition, the FSM is clocked by nine gated clocks to avoid unnecessary switching activities. Figure 6(a) shows average active cycles of the nine gated clocks while Figure 6(b) indicates a power reduction of 37.6% by FSM gating. These results were obtained by feeding the decoder

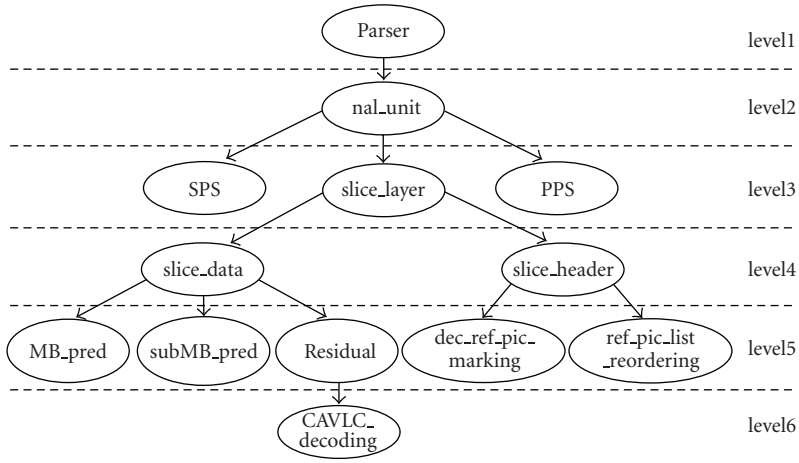


Figure 5: Hierarchically decomposed FSM.

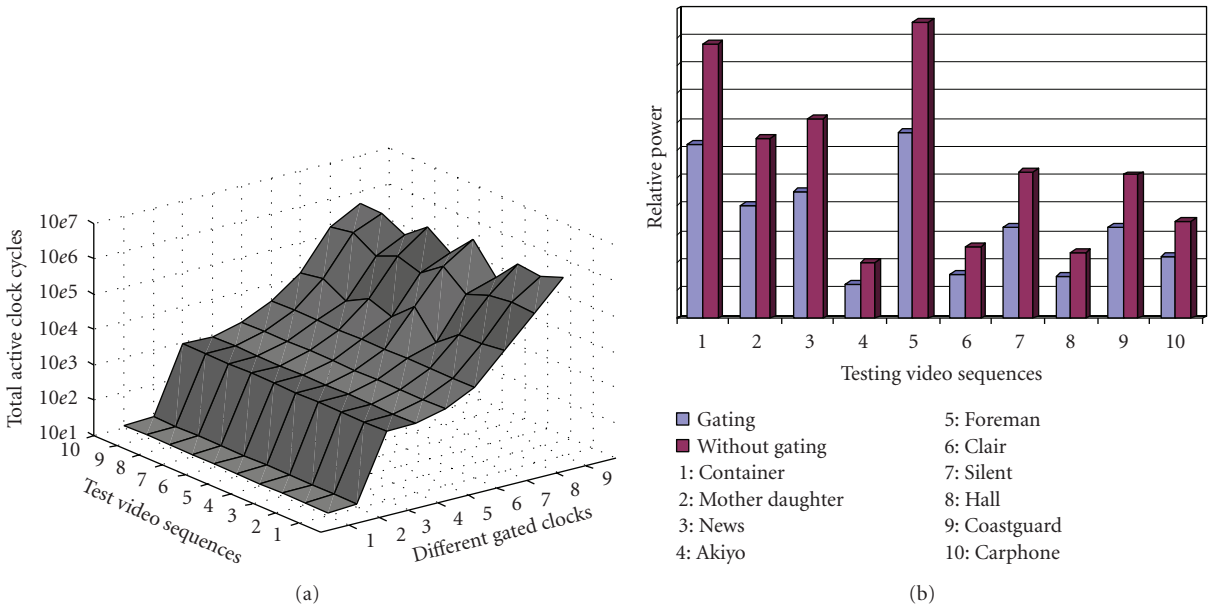


Figure 6: FSM clock gating.

with ten 300 frames video sequences at 30 fps, as shown in Table 2.

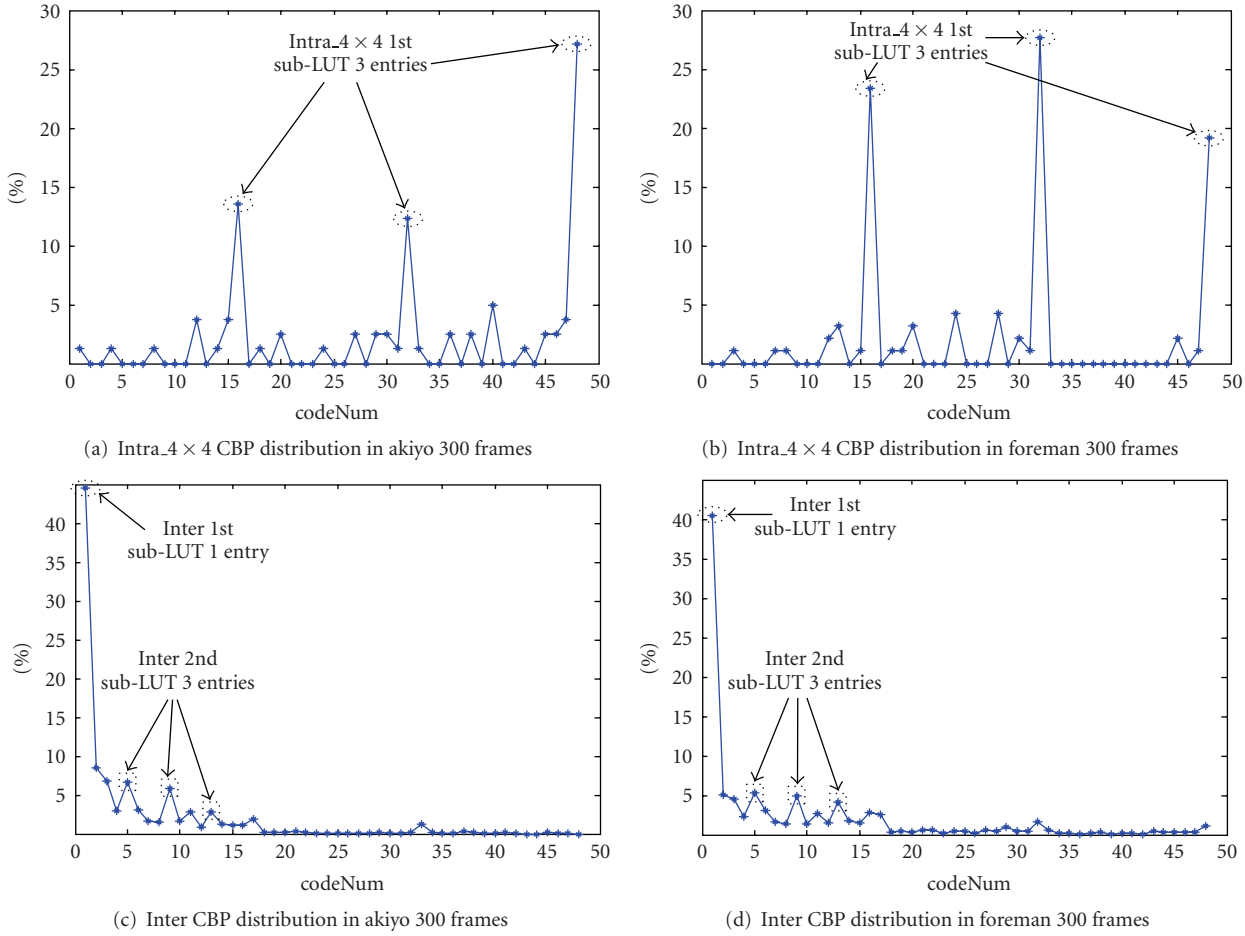
3.5. Data Driven LUT Partition. As specified in H.264/AVC, there are several LUTs used in bitstream parsing. For instance, codeNum (0~47) is mapped to coded_block_pattern (CBP, 0~47) in different ways for Intra_16 × 16, Intra_4 × 4, and Inter-macroblock. Except for Intra_16 × 16, Intra_4 × 4 and Inter-CBP decoding require one-to-one mapping without any arithmetic operations. Average energy consumption for decoding each codeNum for these two cases can be modeled as suggested in [14, 15]:

$$E_{\text{avg}} = \sum_{i=1}^{N} P_i E_i, \tag{1}$$

where P_i is the probability that codeNum $= i$ will occur, E_i is the energy required to decode such a codeNum, and N is the total number of codeNums in the table where $N = 48$ for both Intra_4 × 4 and Inter-macroblock.

In conventional approach, the energy required to decode the codeNum does not vary much over the codeNum's probability, because the whole table is implemented as a single LUT which is active all the time. This ignores the fact that a small amount of codeNums do occur more frequently compared to the majority. Therefore, the power consumed is predominantly in the decoding of codeNums with high occurrence probability. The statistical distribution of CBP for two typical 300 frame video sequences is plotted in Figure 7.

In our proposed approach, the decoding of codeNum for Intra_4 × 4 and Inter-macroblock is separated since they are independent.



(a) Intra_4 × 4 CBP distribution in akiyo 300 frames

(b) Intra_4 × 4 CBP distribution in foreman 300 frames

(c) Inter CBP distribution in akiyo 300 frames

(d) Inter CBP distribution in foreman 300 frames

Figure 7: CBP statistical distribution.

For Intra_4 × 4 prediction, from Figures 7(a) and 7(b), the most probable CBP lies at 15, 31, and 47, indicated by the three peaks. Their corresponding input codeNums are 2, 1, 0, respectively. They account for 50%~70% of all inputs. So the proposed LUT is divided into two tables, a smaller one for CBP 15/31/47 that is active most of the time and a larger one for other CBP values. Thus the energy model (1) can be rewritten as

$$E_{\text{avg}} = P_1 E_1 + (1 - P_1)E_2 + E_{\text{overhead}}, \tag{2}$$

where P_1 is the match probability of the first table; E_1 is the corresponding energy; E_2 is the energy consumption of the second table; E_{overhead} is the energy consumption by the circuit overhead resulting from partitioning the table. Like selection muxes, this power is very small and can be neglected. Without loss of generality, it is reasonable to assume that the complexity/energy of the LUT is proportional to the number of input codeNums since one-to-one mapping is required for decoding. We obtain

$$E_{\text{org}} = 48, \qquad E_1 = 3, \qquad E_2 = 45. \tag{3}$$

After partitioning, upper bound and lower bound of consumed energy are found when $P_1 = 50\%$ and 70%:

$$\begin{aligned} E_{\max} &= 0.5 \times E_1 + 0.5 \times E_2 = 24, \\ E_{\min} &= 0.7 \times E_1 + 0.3 \times E_2 = 15.6. \end{aligned} \tag{4}$$

Compared with the original single LUT energy consumption, the energy saving varies from 50% to 67.5%.

The LUT partition for interprediction can be derived in analogy. The ideal energy saving varies from 62% to 72%. From postlayout power simulation, the energy reduction is 51%.

3.6. Signal Gating. Signal gating refers to a class of general techniques to mask unwanted switching activity from propagating forward, avoiding unnecessary power dissipation. In this paper, we use precomputation logic to disable registers and to avoid inconsequential logic switching (guards). Illustration of these two signal gating techniques is shown in Figures 8 and 9, respectively. By trading area for power in signal gating, we add a small circuit (less than 7% of the total gate count) to minimize the switching activities of a large scale combinational logic. In the example shown, power

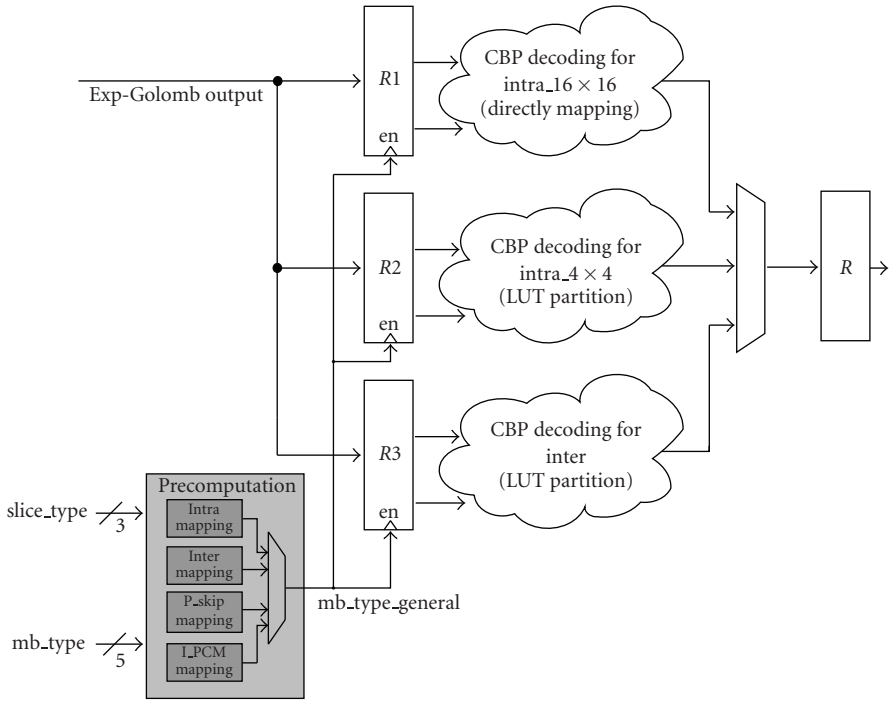


Figure 8: Precomputation for CodedBlockPattern decoding.

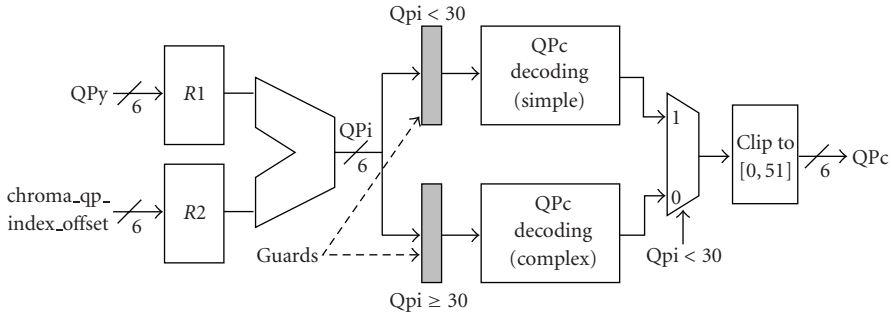


Figure 9: Guarded evaluation for Chroma Quantisation Parameter (QPc) decoding.

saving is achieved when only 1/2∼1/3 of the original logic is active at the same time.

3.7. CAVLC Decoder. CAVLC is a lossless entropy coding algorithm where VLC tables for various elements are switched according to previously coded elements. It is used to encode the residual data organized in 4×4 or 2×2 blocks of transform coefficients.

Since the system bottleneck usually lies in the prediction module, the CAVLC decoder, together with the serially connected IQIT decoder, runs as a parallel pipeline along with the prediction module. The input to the CAVLC decoder and the following IQIT decoder are decoded coefficient levels. They are zig-zag ordered at the CAVLC decoder outputs and should be reordered before entering the IQIT decoder.

3.7.1. LUT Partition. Similar to bitstream parsing, CAVLC decoding also needs to address several LUTs. Operation that is the most power critical is the mapping of coeff_token to TotalCoeff and TrailingOnes as it requires the decoding of a complex LUT (coeff_token LUT in Figure 1) with 270 entries. By exploiting the statistical distribution of coeff_token as plotted in Figure 10, and adopting a similar data-driven LUT partition scheme described in part E, the original LUT is divided into 16 smaller ones ordered in decreasing priority as shown in Figure 11. Table TX0 has the highest priority and is also much smaller than the others. Only when TX0 fails to decode input nC, TX1 will be activated. T40 is an exception since its decoding operation is mathematically based instead of table lookup. A power reduction as much as 67% is realized against the original single LUT solution.

3.7.2. Zero-Block-Aware. By simulating different video sequences of various resolutions and QP, we observed, as shown in Table 3, that there are many zero-valued blocks of residual coefficients, whose distributions are not directly

Table 1: Statistical results of heading one position.

	Whole input bitstream	Intra coded frame	Inter coded frame
Position = 0	55.36%	51.37%	56.61%
Position = 1	24.15%	24.36%	24.08%
Position = 2	11.16%	10.70%	11.31%
Position = 3	5.49%	6.21%	5.26%
Position = 4	2.17%	3.69%	1.72%
Position = 5	0.88%	1.6%	0.65%
Position = 6	0.41%	0.91%	0.25%
Position = 7	0.16%	0.4%	0.08%
Position = 8	0.08%	0.25%	0.03%
Position = 9	0.06%	0.24%	0.01%
Position = 10	0.04%	0.15%	Nearly 0
Position = 11	0.01%	0.06%	Nearly 0
Position = 12	Nearly 0	0.04%	Nearly 0
Position = 13	Nearly 0	0.03%	Nearly 0
Position = 14	Nearly 0	Nearly 0	Nearly 0
Position = 15	Nearly 0	Nearly 0	Nearly 0
Average position	0.81	1.12	0.74

Table 2: 10 QCIF 300 frames video sequences.

	QP	Bitrate (kb/s)	Bits/frame (Intra)	Bits/frame (Inter)
Mother & daughter	24	78.92	24,215	2,512
News	26	94.8	32,161	3,018
Akiyo	28	24.83	19,005	721
Claire	28	30.68	12,939	937
Foreman	28	130.24	21,911	4,177
Silent	30	65.06	21,350	2,058
Container	30	28.65	20,768	843
Hall	32	30.75	15,644	931
Coastguard	34	65.39	13,107	2,097
Carphone	36	45.19	10,418	1,431

related to video resolution. Skipping the decoding of these blocks and directly setting their corresponding output to zero would save a lot of power and improve the performance.

By combining zero block cases with normal non-zero cases, a fast zero-block-aware switching mechanism is adopted as in Figure 12.

Zero-block-aware can be exploited in three hierarchical levels.

(1) Slice level: at this level, we decide whether a 16×16 macroblock should be skipped.

Taking the decoding of the 10 QCIF 300 frames video sequences (Table 2) as an example, all these sequences have significant portion of skipped MB (64%) when targeting low-bit rate applications, as illustrated in Figure 13.

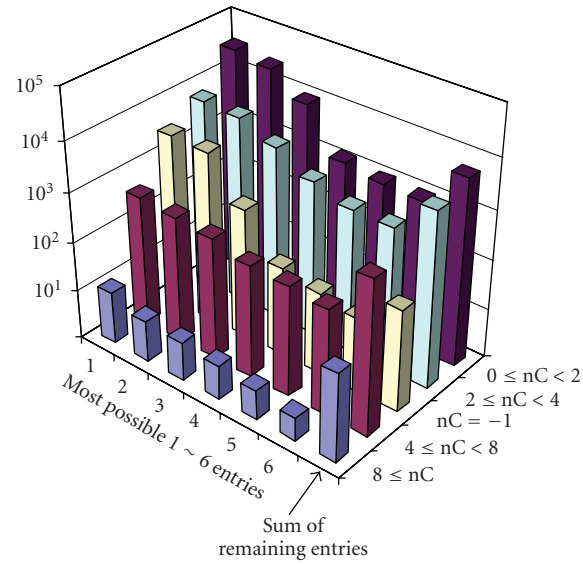


Figure 10: Coeff_token distribution.

Table 3: Zero block distribution.

		QP	Percentage of zero block MB	Bitrate (kb/s)
QCIF 176×144	Coastguard	34	42.5%	65.39
	Foreman	28	26.5%	130.24
CIF 352×288	Flower	22	23.7%	3,557
	Paris	26	52.1%	1,072
4CIF 704×576	Soccer	28	33.1%	2,879
	City	32	45.3%	1,359
HDTV 1280×720	Jets	24	54.6%	2,088
	Raven	32	65.7%	1,305
HDTV 1920×1088	Blue sky	20	33%	16,013
	Rush hour	32	51%	2,499

(2) Macroblock level: at this level, we decide whether an 8×8 block should be skipped. Figure 14 illustrates the distribution of CodedBlockPattern in the aforementioned 10 QCIF video sequences. For luma components, 72% of 4×4 blocks in an MB are set as all-zero residual block, while for chroma, 87% of 4×4 blocks contain only zero residual coefficients.

(3) 4×4 block level: at this level, we decide whether a 4×4 block should be skipped. Statistic results show that there is more than 50% chance that CAVLC decoding can be terminated immediately after decoding the TotalCoeff.

The control unit for the CAVLC decoder makes use of these zero-block-aware techniques to automatically recognize block skipping and to enable signal gating to avoid redundant operations.

4. Parameter Decoder

The parameter decoder includes three parts, Intra_4×4 PM (Prediction Mode) decoding, inter-MV (motion vector) decoding, and BS (Boundary Strength) decoding. Generally,

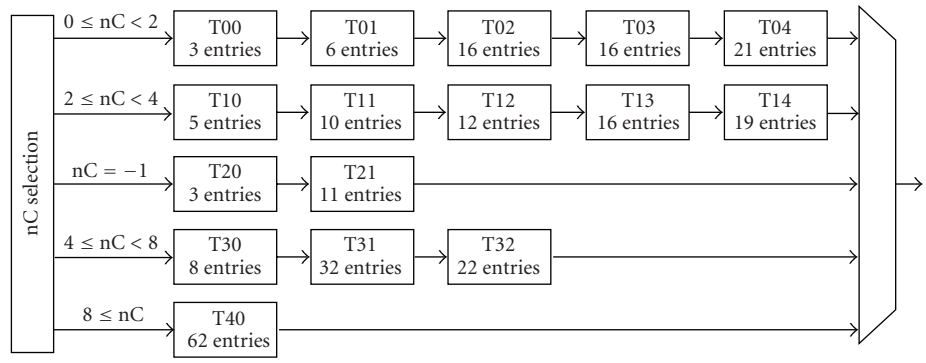


Figure 11: LUT partition based on statistics.

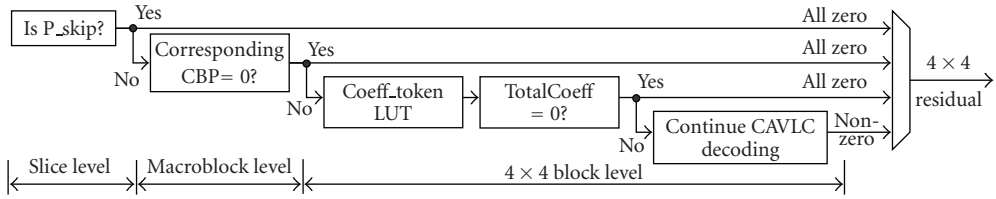


Figure 12: CAVLC decoder zero-block-aware.

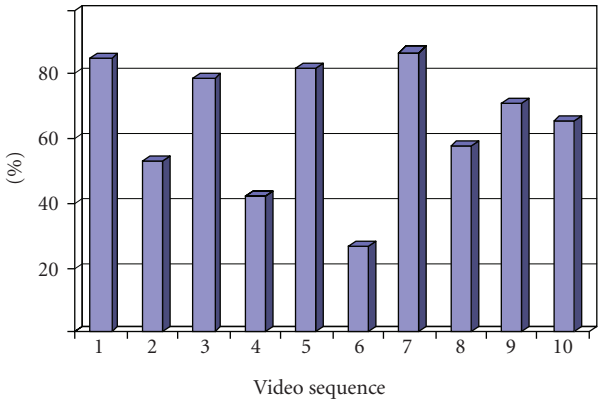


Figure 13: P_skip MB distribution.

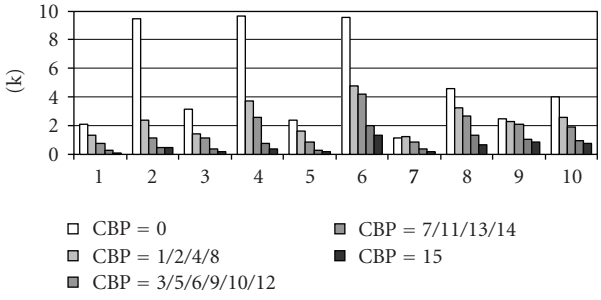


Figure 14: CodedBlockPattern distribution.

it can be treated either as part of prediction/deblocking filter module [9] or as a separate decoding block. Since the decoding of Intra_4 × 4 PM and BS are straightforward, we only discuss the design of Inter MV decoding here.

For the tree structured motion compensation, their corresponding motion vectors are also tree structured. Each MV should be coded and transmitted, as well as the choices of partitions must be signaled in the encoded bitstream. The proposed MV decoder, as shown in Figure 15, contains several parts. Neighboring MV Store Buffers (SBs), which include MV_LSB(Left SB) and MV_USB(Upper SB), as well as current MV store buffers MV_CSB(Current SB), provide related MVs to four distinct decoders for MV_addrA∼D decoding. We noticed that the size of MV_USB should be large enough to accommodate the entire frame row which is 88 byte for current QCIF resolution. However, if the resolution scales up to HDTV1080p, MV_USB becomes 960 byte which should be implemented in SRAM instead of simple DFF-based buffers.

The motion vector of each partition is 16 bits with 8 bits for x direction and 8 bits for y direction (although H.264/AVC standard allows a much larger motion vector range, our design only implements motion vector of $[-32, 31.75]$ in both x and y direction). For a single MB, its corresponding MV_LSB and MV_USB require 64-bit memory access if current MB is 4×4 partitioned. Since interprediction is the major prediction mode for entire video decoding, the frequently read and write operations on such memories cause significant access power. Based on the observation that a substantial amount of MBs inside a video sequence are predicted at a relatively larger partitions, that is, 16×16, 8×16, or 16×8 instead of 4×4, we propose an MV TAG method to reduce the memory access under such kind of large granularity cases.

The working principle of the basic TAG operation is described in Figure 16. We add 1 bit TAG signal for both MV_LSB and MV_USB. The TAG operations can be divided

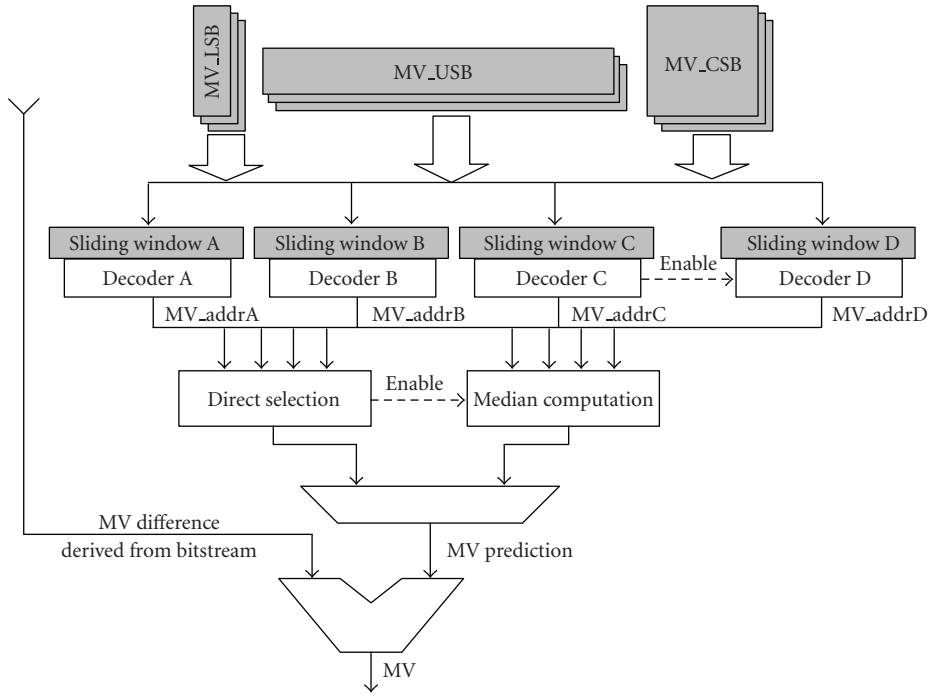


Figure 15: MV decoder.

into two categories, write and read, as described in the following.

Write. If the current MB is partitioned at a finer granularity of 4×4, each 4×4 block has its unique MV, which costs $16 \times (8 + 8)$ bit memory write plus 1-bit TAG update. However, if current MB is partitioned at a coarse granularity of 16×16 or 16×8, all the 4×4 blocks in the current MB of the same row are characterized with identical MV. Therefore, it is not necessary to update all the $4 \times (8 + 8)$ bit for the current 4 4×4 blocks in the same row with identical value. Therefore, only MV of first 4×4 block is updated (since the MV for the 2nd, 3rd, and 4th 4×4 blocks are the same) with TAG set to 1 that indicates same value for 4 blocks. This help to save $64 - 16 = 48$-bit memory write operations.

Read. when neighboring MV stored in MV_LSB is to be read out as prediction of current MV, or current MV in MV_CSB is to be read out for interpolation, corresponding TAG information is read out first instead of the MV itself. If TAG = 0, which means that 4 4 $\times$ 4 blocks in the same row have different MVs, their individual MVs have to be accessed one by one. However, if TAG = 1, which means that 4 4 $\times$ 4 blocks in the same row have the identical MV, only the MV of first 4×4 block needs to be read and treated as universal MV of all the 4 4 $\times$ 4 block in a row.

The TAG method for the 4×4 blocks in a column works similarly. The MV_LSB has only one TAG, and the MV_USB has W_MB TAGs where W_MB is the picture width in MB.

To further reduce the memory access power, a two-level TAG is adopted for MV_CSB. The "TAG" signal indicates the partition mode of the entire MB (16×16 or not), while "TAG0~TAG3" denote the individual partitions of 8×8 blocks of current MB. If the current macroblock is predicted in 16×16 as a whole, "TAG" is set to 1 and only the upper-left 16-bit MV of current MB is accessed and used as the MV for entire MB. TAG0~TAG3 and other MVs are left unchanged. If the current partition has a smaller partition like 16×8, 8×16, or 8×8, "TAG" equals 0 and TAG0~TAG3 are set to 1. Each upper-left 16 bit MV of 4 16 bit MV group is updated. Otherwise (partition smaller than 8×8), "TAG" and TAG0~TAG3 are assigned with 0 which indicates that all the 16 16-bit MV of the entire MB should be accessed.

The proposed TAG method significantly reduces MV store buffer access. The original MV access for one MB prediction is 384 bits (64 bit MV_LSB, 64 bit MV_USB, and 256 bit MV_CSB). By utilizing the TAG for three MV store buffers, MV access is reduced to 108 bits/MB in average with total 71.9% reduction.

5. IQIT Decoder

The proposed IQIT decoder works on the output from the CAVLC decoder and generates pixel residual data for each 4×4 block, which will be added with prediction results to reconstruct fully decoded pictures. According to the current block type indicated by the syntax parser FSM, different inverse transform modes and quantization parameters as well as quantization sequences are utilized.

5.1. Parallel Architecture. For a complete IQIT decoder, generally there are three steps needed to be carried out: inverse transform (IT), rescaling (IQ, inverse quantization), and final rounding (not the rounding inside IQ). The

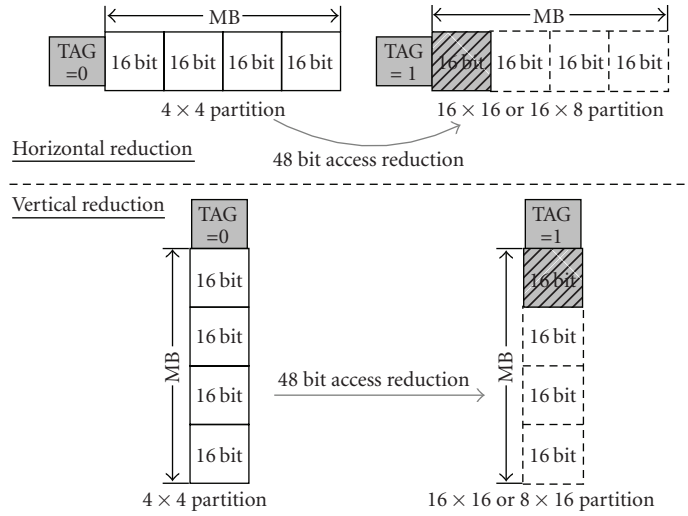


Figure 16: Basic TAG operation.

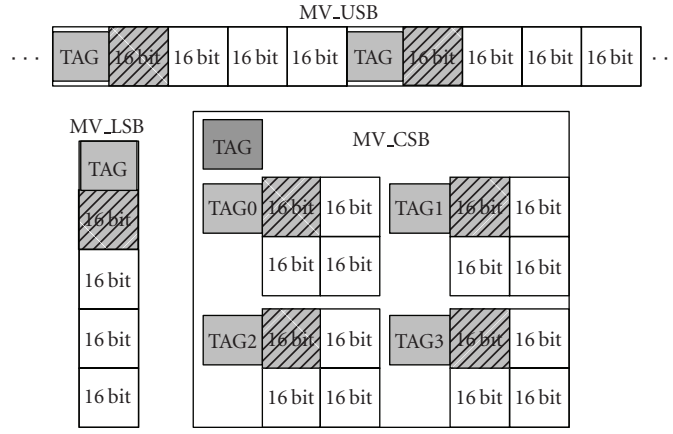


Figure 17: Overall MV store buffer with TAG method.

inverse transform can be further decomposed into two One-Dimensional (1D) transforms (1D-IDCT and 2D-IDCT, resp.) and a transpose memory. For DC coefficients, rescaling runs after 2D-IDCT and there is no rounding operation; for AC coefficients, rescaling comes first before 1D-IDCT and rounding follows 2D-IDCT. Serial decoding architectures are not suitable for high-throughput applications. In this paper, we propose a 4-parallel architecture which is capable of handling four input pixels simultaneously. The implementation of each of the three steps, IQ, IT, and rounding, is described below.

5.2. Inverse Quantization (Rescaling). The rescaling factor of each coefficient is chosen based on the coefficient position inside a 4×4 block. A three-dimensional LUT which is addressed by current quantization parameter and x/y coordinates of input coefficients is used to derive the rescale factor. The input coefficients are first multiplied by their individual rescaling factor, the results are then rounded (shifted) to achieve the final IQ results. Four-stage pipeline with 4-parallel Processing Element (PE) are adopted as shown in Figure 18.

In a single PE, QPy and QPc are selected by a 2-1 mux to get current QP which is used to address two LUTs: mod6 LUT and div6 LUT. The standard originally requires MODULAR and DIVISION operations for QP; however, it is not wise and power-efficient to use dedicated modular and divider as QP has only 52 possible values. The design reported in [16] adopts recursive subtraction which needs additional subtracters and causes long latency. We offer an alternative solution: two LUTs instead of two complex arithmetic modules. In comparison, LUT is faster, smaller, and more power efficient. The only disadvantage is the loss of flexibility which is not a concern in a dedicated decoder.

5.3. Inverse Transform. The inverse transform specified in H.264/AVC can be decomposed into a transpose memory sandwiched between two separate 1D transforms. For a 4×4 block, a total of 16 residual values need to be transformed

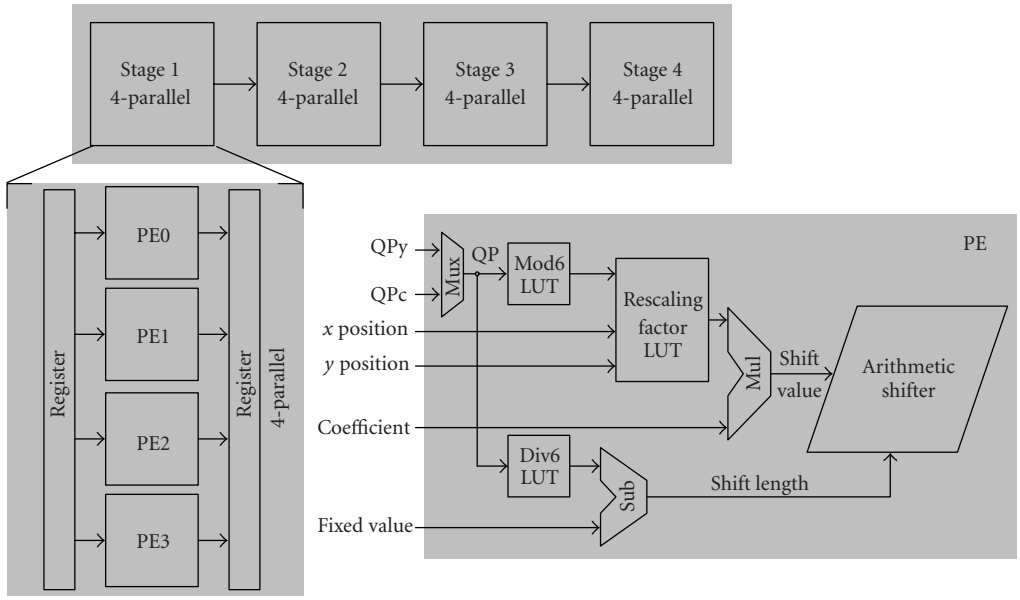


Figure 18: Pipelined inverse quantization.

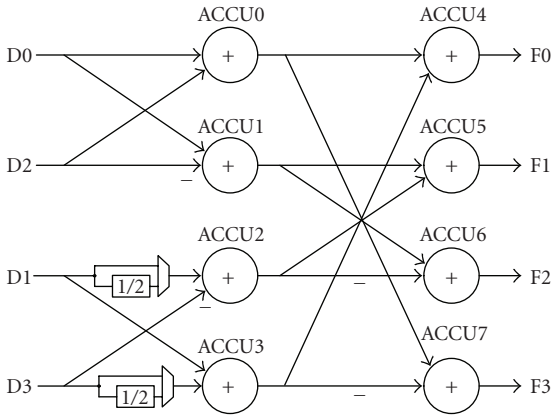


Figure 19: Hybrid 1D-IDCT architecture.

and they are organized in a 4-parallel subblock when going through a 4-stage pipeline.

5.3.1. Reconfigurable IDCT Unit. To reduce the hardware cost, a reconfigurable IDCT unit for all three transforms (4×4 AC, 4×4 DC, 2×2 DC) is proposed in Figure 19. For AC or 2×2 DC transform, the multiplexers before ACCU2 and ACCU3 select either the direct input or the 1/2 scaled version for the 4×4 DC transform. Thus a single reconfigurable IDCT processing unit is capable of handling all the three different inverse transforms.

5.4. Pipelining and Parallelism. The IQIT decoding clock cycles of individual tasks for each $4 \times 4/2 \times 2$ block are summarized in Table 4.

The cycle numbers in Table 4 represent clock cycles when 4-parallel architecture is utilized. For one macroblock coded as Intra_16 × 16 mode, it needs $(1 \times 12 + 2 \times 2 + 24 \times 16) = 400$ cycles, while for one macroblock coded other than Intra_16 × 16 mode, it requires $(2 \times 2 + 24 \times 16) = 388$ cycles. However, this throughput lags behind prediction part and can be further improved by pipelining. The proposed pipelining reduces 4×4 luma DC decoding from 12 cycles to 9 cycles and reduces 4×4 luma/chroma AC from 16 cycles to 10 cycles, as illustrated in Figure 20.

Table 4: Individual tasks decoding cycle (4-parallel, wO pipeline).

	IQ	IT	Final rounding	Total
Luma DC (4×4)	4	8	0	12
Luma AC (4×4)	4	8	4	16
Chroma DC (2×2)	1	1	0	2
Chroma AC (4×4)	4	8	4	16

Figure 21 summarizes that in general 36.8% and 37.1% clock cycles can be saved by pipelining for Intra_16 × 16 macroblock and other macroblocks, respectively.

5.5. Power Saving Techniques. It is observed that not all residual blocks need to run through all IQIT steps. All-zero blocks can skip the whole IQIT decoding while DC-only blocks do not need any inverse transform. Current designs like those in [17, 26] have not taken this into account and process all the residual blocks in the same manner. In the proposed design, residual blocks are handled differently according to their types.

For 4×4 blocks containing nonzero AC coefficient, they undergo the complete IQIT process, including inverse quantization and transform. For 4×4 blocks containing DC-only blocks, only their DC values are inverse quantized and rounded and then expanded to all coefficient levels; no inverse transform is invoked. For all-zero blocks, they skip

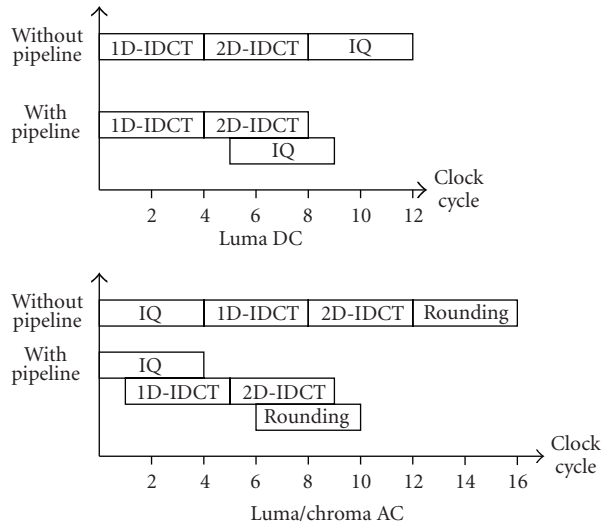


Figure 20: Pipeline improvements.

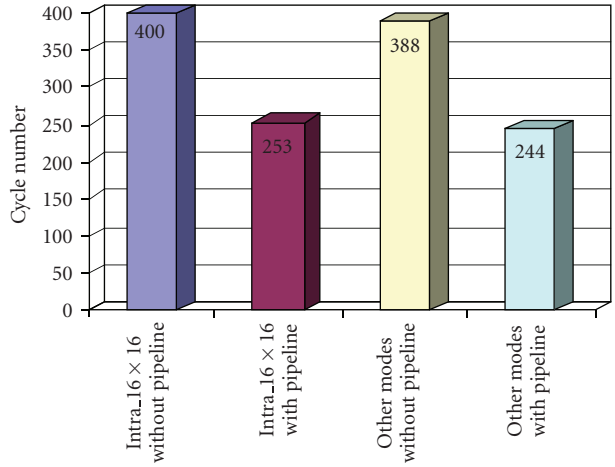


Figure 21: Clock cycle savings by pipelining.

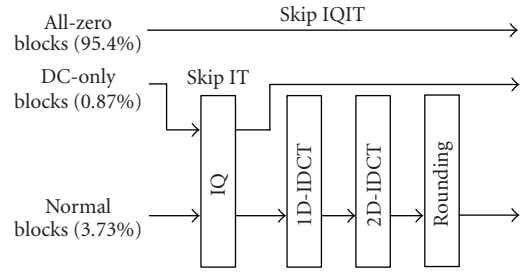


Figure 22: Skipping mechanism for different blocks.

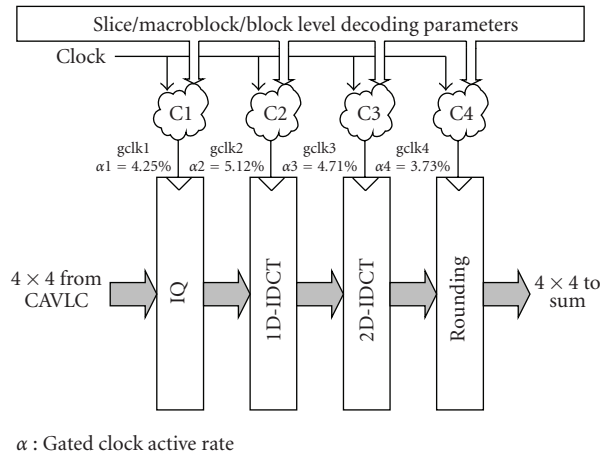


Figure 23: IQIT decoder clock gating scheme.

the whole IQIT module and an "all-zero" flag is used to signal the later sum module, which adds "residual" with "prediction" together. This technique not only reduces the power consumed by the IQIT module itself but also leads to savings in the sum module as well, since summation can be avoided when all-zero blocks are encountered.

The occurrence distribution of these three blocks, all-zero, DC-only, and normal block, is described as percentage in Figure 22. This distribution is obtained from simulation of same 10 video as mentioned before. The exact distribution percentage for other video sequence may be different but not by a lot. There are always a large percentage of all-zero blocks and some DC-only blocks.

The aforementioned "skip" characteristic, as well as the relatively regular structure of the IQIT decoder, provides the possibility of clock gating at each decoding stage.

A uniform clock gating scheme is not viable as not all IQIT components are active at the same time. Separate clock gating arrangement for IQ, 1D-IDCT, 2D-IDCT, and rounding allows each function unit to be selectively enabled or disabled. Each gating condition is precomputed one cycle ahead. These heterogeneous gated clocks are depicted in Figure 23; gclk1∼gclk4 are generated separately according to different conditions.

6. Design Implementation

6.1. Design Flow. The proposed bitstream-residual decoder is coverified with the prediction module to realize a complete H.264/AVC decoding system. High level SystemC model was first constructed to verify the behavior of the whole decoder system and to identify power reduction opportunities at system level. HDL design flow was then employed to reduce the design time and to obtain a technology independent design. The whole decoder was partitioned into several building blocks which were coded and verified individually before system integration. To facilitate debugging, we compared the intermediate results generated from the proposed decoder with the associated test patterns captured from the H.264/AVC reference software JM94 [18]. The proposed bitstream-residual decoder was demonstrated firstly in an FPGA, then in an ASIC implementation of a complete H.264/AVC baseline decoder. Finally, the chip was measured under various supply voltages.

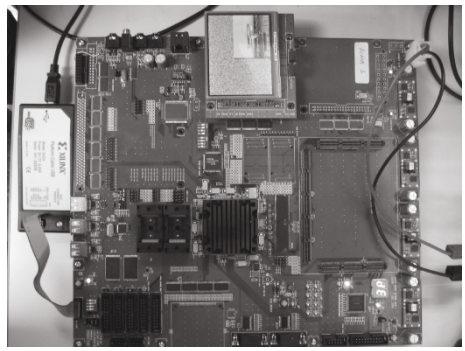
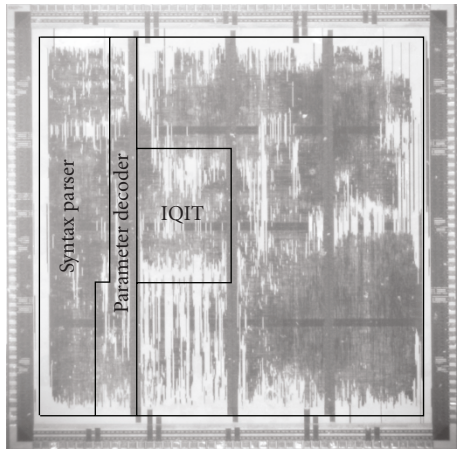


Figure 24: FPGA verification environment.



(a)

Technology	UMC 0.18 μm CMOS 1P6M
Area	6.5 mm^2
Gate count	61.6 k
Normal supply voltage	1.8 V
frequency	1.5 MHz
voltage	1.8 v
Measured Power	69 μW @ 1V 230 μW @ 1.8V

(b)

Figure 25: Chip photograph.

6.2. FPGA Prototyping. The whole decoder system has been realized on a Xilinx emulation board shown in Figure 24. The board consists of one Virtex4 [19] FPGA which implements the decoder logic and one Epson chip which controls the on-board TFT-LCD display. The FPGA-oriented RTL code is synthesized, translated, placed and routed by Xilinx ISE software, and simulated by ModelSim. The encoded video sequences are fed into the FPGA RAM and are sequentially fetched by the bitstream buffer of the bitstream-residual decoder. After decoding, YUV format video pictures are translated into RGB format and then are written to the display memory for real-time display at 30 fps. Noted that the test videos are all 300 frames of QCIF size, they occupy only part of the 320×240 LCD display. The decoder takes up 7,048 4-input LUTs with an equivalent gate count of about 337,109.

Table 5: Gate count and power profiling.

	Syntax parser	Parameter decoder	IQIT decoder
Gate count	23 k	24.3 k	14.3 k
Power (1.8 V)	148 μW	95.3 μW	9.7 μW

6.3. ASIC Implementation. The RTL-level ASIC-oriented Verilog code was synthesized using UMC 0.18 μm 1P6M standard cell technology by Synopsys Design Compiler. The layout was realized by Synopsys Astro, and the postlayout power consumption was obtained by Synopsys Prime Power. The chip photograph and the specifications of the bitstream-residual decoder are outlined in Figure 25. The implemented gate count and power profiling are described in Table 5.

In order to maximize the power efficiency, supply voltage scaling is needed since it has a quadratic effect on the power consumption. The architecture level pipelining and parallelism relax the constraint on the operating frequency and thus provide the possibility of scaling. Since the standard cell library was characterized at 1.8 V, the timing specifications for 1 V supply need to be translated into 1.8 V.

The datapath timing delay can be modeled as (5):

$$d = \frac{kV_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_{\mathrm{th}})^{\alpha}}, \tag{5}$$

where d is the path delay, k is an execution time constant, V_{dd} is the supply voltage, V_{th} is the transistor threshold, and α is the alpha power law constant [20]. For UMC 0.18 μm technology, V_{dd} =1.8 V, V_{th} =0.5 V, and $\alpha \approx 1.2$. Based on this model and the cell delay estimation from [21], the datapath delay under 1 V supply voltage is estimated to be 1.6~1.8 times larger than that under 1.8 V supply. To reserve a certain margin and to cope with process variation, we assume that the path delay is 2 times larger. Therefore, designing for 1.5 MHz at 1.0 V translates to 3 MHz equivalent timing constraint for logic synthesis and physical design under the normal case model (25°C, 1.8 V).

6.4. Comparison. As compared with designs in [8, 22], the proposed design has similar number of gates as in [22], while reducing gate count significantly over [8].

The proposed decoder is able to decode QCIF sequence of 30 fps at 1.5 MHz. Compared with [5, 7] in Table 7, this work is the most throughput-efficient.

The power consumption breakdown for the bitstream-residual decoder is described in Figure 26. The syntax parser consumes the most power due to its high switching activity. A large portion of the CAVLC decoder power is used in nC decoding, which requires complex memory access. Although the IQIT module accounts for more than 1/5 of the total gates, its power consumption is less than 4% of the total due to the clock gating and the innovative skipping mechanism. The gate count of the parameter decoder is the largest among the three blocks, but its power consumption is less than the syntax parser for its low activity and extensive clock gating in memory access.

The power comparisons of the VLC decoder, syntax parser, and IQIT decoder are described in Tables 8, 9, and 10,

Table 6: Gate count comparison.

	Syntax parser	Parameter decoder	IQIT decoder
Proposed	23 k	24.3 k	14.3 k
[22]	19.8 k	NA	19.8 k
[8]	52 k+2176 bit memory	29.8 k (only for inter MV)	9.3 k+7040 bit memory

Table 7: Performance comparison.

	[7]	[5]	Proposed
Capability	720 × 480 @ 30 fps	176 × 144 @ 15 fps	176 × 144 @ 30 fps
Frequency	25 MHz	1.15 MHz	1.5 MHz
Normalized throughput	0.414 Mpix/MHz	0.330 Mpix/MHz	0.506 Mpix/MHz

Table 8: VLC decoder power comparison.

	[23]	[24]	Proposed
Frequency	20 MHz	N/A	1.5 MHz
Power consumption	183 μW	75 μW	53 μW

Table 9: Syntax parser power comparison.

	[25]	Proposed
Frequency	27 MHz	1.5 MHz
Power consumption	2.87 mW	148 μW

Table 10: IQIT power comparison.

	[24]	[26]	[27]	Proposed
Frequency	N/A	80 MHz	50 MHz	1.5 MHz
Power consumption	113 μW	13.7 μW	4.7 μW	9.7 μW

respectively. All the reference designs are scaled to 0.18 μm technology with 1.8 V supply for a fair comparison. Note that [23, 25] handle standards simpler than H.264/AVC, while [26, 27] contain only IDCT which is half of a full IQIT decoder.

All the power comparison above of the proposed design was measured under 1.8 V supply voltage. The power can be further reduced when the voltage is scaled down to 1 V.

To summary, the power saving happens in all building blocks as shown in Figure 27. Original is our raw implementation without any power optimization. The IQIT decoder achieves the largest power reduction due to the skip mechanism with nonuniform clock gating. The main contribution of the syntax parser derives from the power optimized CAVLC decoder. Regarding the parameter decoder, power saving in MV decoding plays a major role. Besides these individual power reduction techniques used in the building blocks, voltage scaling on the whole decoder may contribute as much as 70% power saving. Under 1 V supply voltage, the power consumption of the proposed decoder is 76 μW for QCIF 30 fps decoding.

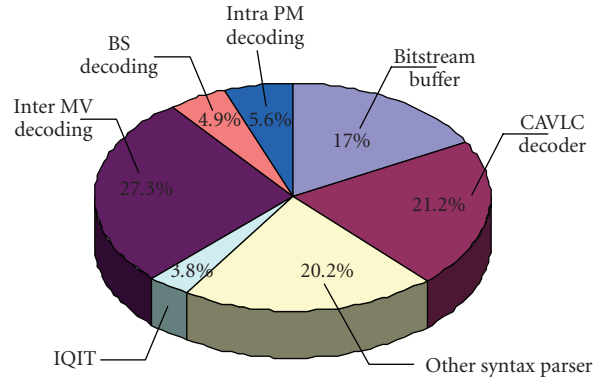


Figure 26: Residual decoder power consumption breakdown.

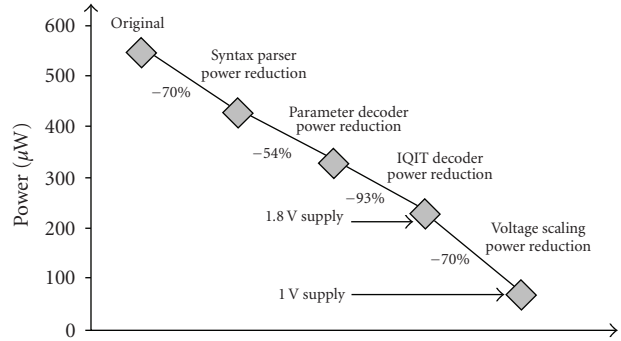


Figure 27: Power reduction overview.

7. Conclusion

In this paper, a new architecture of a low-power bitstream-residual decoder for H.264/AVC baseline decoding has been introduced. Novel low power building blocks are proposed and analyzed. Various power reduction techniques are utilized in the architecture level and the circuit level. No special process-dependent techniques like multi-Vth are adopted, thus making the proposed design easily portable.

The FPGA verification is realized with Xilinx Virtex4 device. The VLSI implementation demonstrates that it is readily realizable in an ASIC [28]. Implemented with the UMC 0.18 μm technology, the proposed residual decoder

uses 61.6 k gates and occupies 1/3 of the whole decoder area. When running at 1.5 MHz for real-time QCIF 30 fps decoding, this work consumes 253 μW at 1.8 V and 76 μW at 1 V supply. The entire decoder architecture is scalable, and all the techniques in this paper can be applied to larger resolutions other than QCIF. Currently we are implementing HDTV1080p decoder based on this architecture.

Acknowledgments

The authors thank Min Zhang for his help during FPGA implementation and thank ASTRI Company for providing the FPGA emulation board. The work is supported by a Hong Kong SAR Government of Direct Grant of no. 2050332.

References

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

[2] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*, John Willey & Sons, New York, NY, USA, 2003.

[3] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 598–603, 2003.

[4] S.-H. Wang, W.-H. Peng, Y. He, et al., "A platform-based MPEG-4 advanced video coding (AVC) decoder with block level pipelining," in *Proceedings of the International Conference on Information, Communications and Signal Processing*, vol. 1, pp. 51–55, December 2003.

[5] T.-M. Liu, T.-A. Lin, S.-Z. Wang, et al., "A 125 μW, fully scalable MPEG-2 and H.264/AVC video decoder for mobile applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 161–169, 2007.

[6] T.-M. Liu, T.-A. Lin, S.-Z. Wang, et al., "An 865-μW H.264/AVC video decoder for mobile applications," in *Proceedings of IEEE Asia Solid-State Circuits Conference*, pp. 301–304, November 2005.

[7] C.-C. Lin, J.-W Chen, H.-C. Chang, et al., "A 160K gates/4.5 KB SRAM H.264 video decoder for HDTV applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 170–182, 2007.

[8] S. Park, H. Cho, H. Jung, and D. Lee, "An implemented of H.264 video decoder using hardware and software," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 271–275, September 2005.

[9] K. Xu and C.-S. Choy, "A power-efficient and self-adaptive prediction engine for H.264/AVC decoding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 3, pp. 302–313, 2008.

[10] K. Xu, C.-S. Choy, C.-F. Chan, and K.-P. Pun, "Power-efficient VLSI implementation of bitstream parsing in H.264/AVC decoder," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 5339–5342, May 2006.

[11] J.-H. Li and N. Ling, "Architecture and bus-arbitration schemes for MPEG-2 video decoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 5, pp. 727–736, 1999.

[12] D. Wu, W. Gao, M. Z. Hu, and Z. Z. Ji, "An Exp-Golomb encoder and decoder architecture for JVT/AVS," in *Proceedings of the 5th International Conference on ASIC*, vol. 2, pp. 910–913, October 2003.

[13] K. Xu, C.-S. Choy, C.-F. Chan, and K.-P. Pun, "Priority-based heading one detector in H.264/AVC decoding," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 60834, 7 pages, 2007.

[14] S. H. Cho, T. Xanthopoulos, and A. P. Chandrakasan, "A low power variable length decoder for MPEG-2 based on nonuniform fine-grain table partitioning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 2, pp. 249–257, 1999.

[15] K. Xu, C.-S. Choy, C.-F. Chan, and K.-P. Pun, "A low-power bitstream controller for H.264/AVC baseline decoding," in *Proceedings of the 32nd European Solid-State Circuits Conference (ESSCIRC '06)*, pp. 162–165, Montreux, Switzerland, September 2006.

[16] I. Amer, W. Badawy, and G. Jullien, "Hardware prototyping for the H.264 4 × 4 transformation," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '04)*, vol. 5, pp. 77–80, May 2004.

[17] Y.-T. Kuo, T.-J. Lin, C.-W. Liu, and C.-W. Jen, "Architecture for area-efficient 2-D transform in H.264/AVC," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '05)*, pp. 1126–1129, July 2005.

[18] Joint Video Team (JVT) reference software JM9.4, http://iphome.hhi.de/suehring/tml/download.

[19] Xilinx, Inc., Xilinx Virtex4 Data Sheets, http://www.xilinx.com/support/documentation/index.htm.

[20] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl, "A physical alpha-power law MOSFET model," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 10, pp. 1410–1414, 1999.

[21] Virtual Silicon, "0.18 μm Standard Cell Library," revision 1.0, pp. 4–7, July 2004.

[22] T.-C. Chen, C. Lian Jr., and L.-G. Chen, "Hardware architecture design of an H.264/AVC video codec," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '06)*, pp. 750–757, January 2006.

[23] T.-H. Tsai, W.-C. Chen, and C.-N. Liu, "A low-power VLSI implementation for variable length decoder in MPEG-1 Layer III," in *Proceedings of the International Conference on Multimedia and Expo (ICME '03)*, vol. 1, pp. 133–136, July 2003.

[24] T.-A. Lin, T.-M. Liu, and C.-Y. Lee, "A low-power H.264/AVC decoder," in *Proceedings of IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT '05)*, pp. 283–286, April 2005.

[25] Y. C. Chang, C. C. Huang, W. M. Chao, and L. G. Chen, "An efficient embedded bitstream parsing processor for MPEG-4 video decoding system," in *Proceedings of the International Symposium on VLSI Technology, Systems and Applications*, pp. 168–171, 2003.

[26] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Parallel 4×4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 800–803, May 2003.

[27] K.-H. Chen, J.-I. Guo, K.-C. Chao, J.-S. Wang, and Y.-S. Chu, "A high-performance low power direct 2-D transform

coding IP design for MPEG-4 AVC/H.264 with a switching power suppression technique," in *Proceedings of IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test (VLSI-TSA-DAT '05)*, pp. 291–294, April 2005.

[28] K. Xu and C. S. Choy, "Low-power H.264/AVC baseline decoder for portable applications," in *Proceedings of the International Symposium on Low Power Design*, pp. 256–261, Portland, Ore, USA, September 2007.