4K Real-Time HEVC Decoder on an FPGA

Maleen Abeydeera, Manupa Karunaratne, Geethan Karunaratne,
Kalana De Silva, and Ajith Pasqual, *Member, IEEE*

***Abstract*— With the popularization of a quad high-definition/ 4K video being dependent on the availability of real-time High Efficiency Video Coding (HEVC) decoders, hardware implementations have become more appealing due to their superior performance and low power consumption. In this paper, a field-programmable gate array (FPGA)-based hardware implementation of a 4K 30 frames/s real-time HEVC decoder is presented. An elastically pipelined decoder architecture is used to absorb variations in processing time and each pipeline stage is optimized to use available FPGA primitives. FPGA-specific challenges in managing critical path delays to achieve a target operating frequency of 150 MHz required many architectural novelties, such as exploitation of the sparsity of transformed coefficient matrix, single-cycle reference pixel processing in intra prediction, and flexible 8 × 8 block ordering in deblocking filter/ sample adaptive offset filter. A high-throughput latency-aware cache architecture was used to reduce the external dynamic RAM access bandwidth by 70%. This work is compliant with the HEVC main profile at level 5 of the HEVC standard and only consumes 126K lookup tables, 58K registers, and 335 18-kb block RAMs when implemented on Xilinx Zynq 7045.**

***Index Terms*— High Efficiency Video Coding (HEVC), motion compensation (MC) cache, ultrahigh-definition video, video decoding on field-programmable gate array (FPGA).**

I. Introduction

HIGH Efficiency Video Coding (HEVC) is the latest video compression standard introduced by International Organization for Standardization/ International Electrotechnical Commission and Telecommunication Standardization Sector of the International Telecommunications Union in early 2013, as an effort to address the issue of ever-growing demand for transmission and storage of a high-resolution video. HEVC promises up to 50% bitrate reduction over the incumbent H.264, but at the cost of extra computational complexity. With the video industry now moving toward quad high-definition (QHD/4K) (3840 × 2160), which contains four times more video data than high-definition (HD) video, it is expected that HEVC will become the video codec of choice in the future. With the growing popularity of HD and QHD video and the increased computational complexity, the added resource usage and power consumption of software HEVC decoders place them at a disadvantageous position. Therefore, hardware decoders are preferred especially in low-power mobile applications.

Hardware decoders can come in two forms as application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). In ASIC decoders, the silicon chip is custom made, which results in high performance, but the nonrecurrent engineering cost associated with such an ASIC development is extremely high, making them economical only for very large volume production. Also in a transition time like at present where the standard is still evolving and slow transition from H.264 to H.265 due to lack of content, there is an inherent danger in ASIC development unless extreme care is taken to perform exhaustive testing of all profiles.

In contrast, FPGA-based decoders use existing configurable hardware platforms to implement specific hardware architectures. Hence, the cost is much lower, but the performance is less than that of ASIC designs. Therefore, it is more cost effective for low-volume products such as specialized broadcast equipment, where the reconfigurability of the FPGA provides an additional advantage. Furthermore, the ability to easily implement the design and to port it across existing FPGA hardware platforms makes it an attractive choice for academic and industry researchers who are into video processing applications.

Several chip vendors and academic researchers have come up with real-time QHD capable hardware HEVC decoder designs, but they are mostly optimized for ASIC and the same hardware architecture runs much slower when implemented on FPGAs. (Kuon and Rose [1] found that the ratio of critical path delay, from FPGA to ASIC, is roughly three to four times.) It requires careful analysis of the capabilities of an FPGA to optimize a design so that it can efficiently utilize the available hardware resources of the FPGA while meeting the timing requirements for real-time operation. Hence, implementing a real-time 4K HEVC decoder on an FPGA remains a challenge.

In this paper, an optimized FPGA-based hardware decoder solution is presented for decoding of HEVC compliant bit streams. The architecture was designed with the objective of real-time decoding of QHD video at 30 frames/s, while using minimum hardware resources. The FPGA implementation was done on the Zynq 7000 platform.

The FPGA implementation of [2] is only targeted at 1080p30. Seunghyyun *et al.* [3] present a design that can be scaled for 4K 60 frames/s by running four cores of a primary decoder pipeline at 200 MHz. However, they have

Manuscript received November 16, 2014; revised April 2, 2015 and May 11, 2015; accepted July 14, 2015. Date of publication August 17, 2015; date of current version January 6, 2015. This paper was recommended by Associate Editor V. Sze.

M. Abeydeera, M. Karunaratne, G. Karunaratne, and K. De Silva were with the University of Moratuwa, Moratuwa 10400, Sri Lanka, and also with ParaQum Technologies, Colombo 00500, Sri Lanka (e-mail: maleen@paraqum.com; manupa@paraqum.com; geethan@paraqum.com; kalana@paraqum.com).

A. Pasqual is with the Department of Electronic and Telecommunication Engineering, University of Moratuwa, Moratuwa 10400, Sri Lanka (e-mail: pasqual@ent.mrt.ac.lk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2015.2469113

1051-8215 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

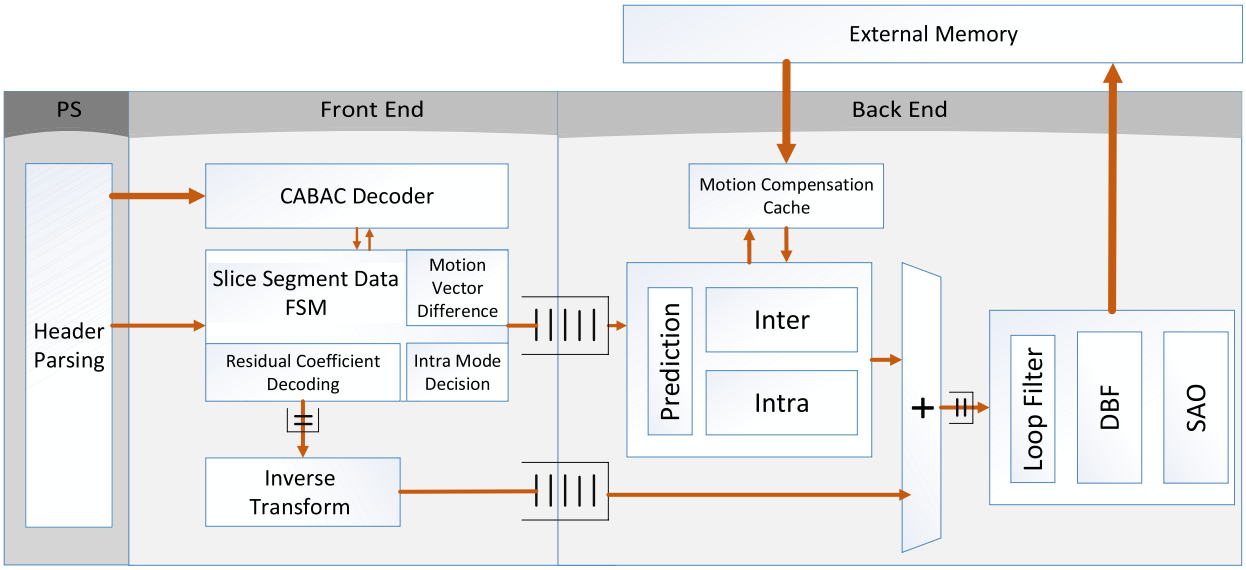

Fig. 1. Overall architecture.

only prototyped the design on an FPGA and does not claim that it can actually run at this frequency on the FPGA itself. Further their device, the Virtex 2000T, is four times the size of our platform and is typically used in prototyping large ASICs.

There have been several previous implementations of very large scale integration (VLSI) implementations of 4K capable HEVC decoders. Tikekar *et al.* [4] present a 40-nm ASIC that can decode 30 frames/s at 200 MHz. However, this work was done prior to the specification being finalized and hence is not compatible with the final standard. More recently, [5] used a data sharing dual core architecture to achieve 4kp60 on a 28-nm process.

Our work is the first to achieve real-time 4K HEVC decoding on a commercial grade FPGA. Achieving this required novel contributions that did not appear in the existing literature to the best of our knowledge. These include the following.

1) An inverse transform architecture that can be seamlessly integrated with the entropy decoder by consuming coefficients as they are produced and a scheduling strategy that inherently utilizes the sparseness of transformed coefficients.
2) Single-cycle reference pixel fetching and padding in the intra-prediction pipeline allowing headroom for interpolations, using eight-bank line buffers.
3) A unique way to handle constrained intra prediction using a special reference pixel update scheme at the end of the pipeline.
4) Single-bank motion compensation (MC) cache architecture with uninterrupted cache line scheduling (UCLS) mechanism to improve cache throughput.
5) A low-latency in-loop filter architecture that supports on the fly filtering of 8×8 pixel blocks in any valid decoding order, without using external memory.

Although this work focuses on an FPGA implementation, these contributions can be made use of in an ASIC environment as well.

The rest of this paper is organized as follows. Section II describes the high-level architecture with Section III elaborating on each module in detail. An MC cache is described in Section IV, and the results are presented in Section V.

II. Overall Architecture

Following along the main pipeline stages in HEVC, our design is built modular wise and then assembled to create a functional decoder (Fig. 1). On the Zynq system, the processing system (PS) is used for header parsing since the time consumed is very negligible compared with the others. The actual decoder pipeline stages are implemented in programmable logic (PL).

Based on the dependency of computational complexity, we categorize these pipeline stages into two main groups.

1) *Front End:* As it consists of the entropy decoder and the inverse transform, the complexity of this group is highly related to the video bitrate. A high bitrate implies a large number of residual coefficients and more time spent in these two stages.
2) *Back End:* It consists of prediction and loop filtering. In this case, the throughput is not related to the bitrate but the picture dimensions.

There can be significant differences between the processing speed of these groups. For an intra-coded block, the prediction accuracy is usually not perfect. This results in a large residual and the time taken by the front end will be significantly higher compared with the time taken by the back end. However, for an inter-coded block (which usually does a proper prediction and produces minimal residual), the time to retrieve the reference pixels from dynamic RAM (DRAM) far outweighs the time taken for the front end.

A first-in first-out (FIFO) queue (of size 112 kbits) is placed between these two groups to create an elastic pipeline. (Refer to Fig. 2 for an illustration on how elastic pipelining improves throughput.) Tikekar *et al.* [4] also use elastic pipelining between the entropy decoder and the inverse transform. However, our unique design of the inverse transform

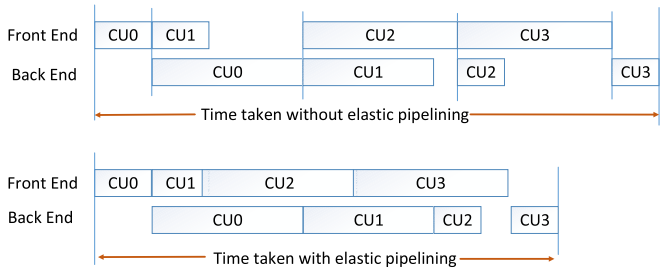

Fig. 2. Impact of using elastic pipelining between the two stages. Coding units (CUs) 0 and 1 are inter coded, and hence, the time taken for back end dominates that for the front end. CUs 2 and 3 are intra coded with a significant amount of time taken for front end.

module ensures that the complexity of the module is a function of the number of nonzero coefficients present (and hence related to the throughput of entropy decoder) and is able to shift this elastic pipeline one stage later. Further, we also place smaller FIFOs within each stage to connect their functional modules.

This usage of FIFOs is driven by the following observations.

1) Each pipeline stage can be made independent of others. A common format for passing data down the pipelines is established and each module can be developed and verified independently.
2) Independent clocking is possible for each stage.
3) The FPGA already contains a significant amount of block RAM (BRAM) with embedded FIFO controllers. Therefore, the resource overhead will be minimal.

The reader is referred to [6] for an extensive exploration of how elastic pipelining can be used in the design of an H.264 decoder.

Handling of Chroma: In the front end, chroma is serially processed with luma due to the order in which the syntax elements (SEs) occur in the bitstream. However, for back end, throughput is a major constraint, and therefore, separate hardware exists for parallel operation on luma and chroma.

III. Processing Engines

In this section, we detail the internal operation of individual pipeline stages.

A. Slice Segment Data FSM

The HEVC specification [7] provides the order in which SEs are placed in the bitstream, and a state machine [slice segment data (SSD) FSM] is used to traverse this ordering.

Apart from traversing the SSD, several additional processing steps are done in the FSM itself.

1) *Intra-Mode Decision:* The actual intra mode, which is not directly sent in the bitstream, is determined by reference to the mode of surrounding blocks as well as to several SEs.
2) *Motion Vector Difference (MVD):* The actual MVD is coded in the bitstream in three separate SEs. These are combined within the FSM.
3) *Residual Coefficient Decoding:* Residual coefficient is coded in the bitstream via several SEs. The FSM calculates the actual coefficient by combining these SEs and pushes it to the inverse transform block.

Whenever an SE encoded with context adaptive binary arithmetic coding (CABAC) needs to be decoded, the FSM sends a request to the CABAC decoder, and the return value is used by the FSM to determine the next state.

Decoding an SE may consist of decoding multiple CABAC bins and our implementation is limited to decoding 1 bin/cycle. Although works such as [8] describe VLSI architectures that can be used to decode multiple bins per cycle, they have a longer critical path that preclude their implementation to an FPGA platform.

There have been several recent works on highly efficient CABAC decoders. The architectures described in [9] and [10] are targeted for level 6.2 (8K/120 FPS) and level 6.0 (4K/120 FPS), respectively. However, for our requirement (4K at 30 frames/s), an implementation of lower complexity (and more amenable to an FPGA architecture) is preferred.

Although the CABAC engine can provide a throughput of 1 bin/cycle, the actual usage is less since separate states in the FSM are dedicated to peripheral work as above. However, as we demonstrate in Section V, this does not constrain the overall throughput as our implementation can handle bitrates up to 37 Mbits/s, which is comfortably above the mandated 25 Mbits/s for HEVC Level 5 main tier.

B. Inverse Transform

Our design of the inverse transform module is based on two observations that have rarely been considered in the work so far.

1) Sparse Matrix: The purpose of a transform from the spatial domain to the frequency domain is to represent the pixel information in as few basis vectors as possible. Hence, the transformed coefficients would mostly congregate at the upper left section of the matrix, with the others being zero. During decoding, these zero coefficients do not contribute to the final residual pixel values, and hence can be safely dropped.

2) Entropy Decoding Order: The order in which the coefficients come out of the entropy decoder can be either horizontal, vertical, or diagonal, but the traditional inverse transform designs consume coefficients column wise. To make the switch, a buffer of 4 kbits needs to be placed at the interface [4]. This design eliminates the need for this buffer by accepting coefficients as they come.

The inverse discrete cosine transform core in Fig. 3 can be time multiplexed to perform both the column and the row transforms. The number of iterations required for one transform block is dependent on the number of nonzero coefficients present.

1) During the column transform, the effect of a nonzero coefficient in the incoming matrix is limited to the column in which the coefficient is present. Therefore, an empty column in the input matrix always yields an empty column in the intermediate matrix. However, one nonzero coefficient is enough to trigger a fully filled column.

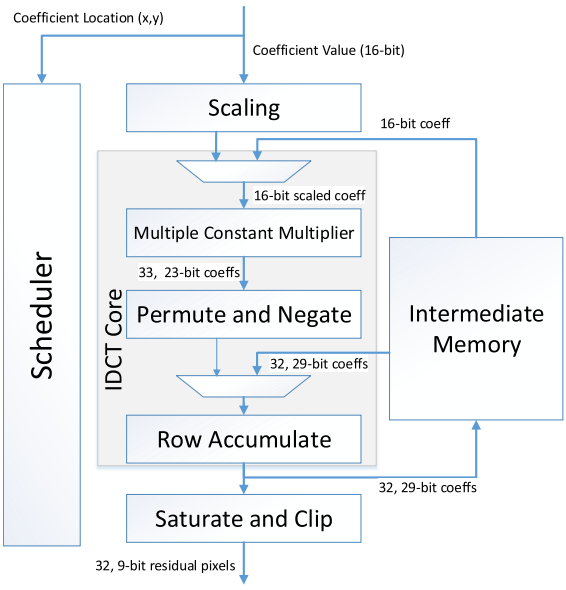

Fig. 3. Inverse transform design.

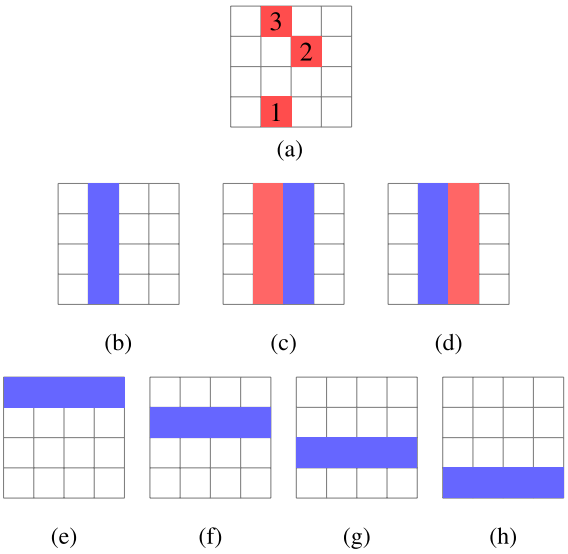

Fig. 4. Illustration of the proposed scheduling algorithm. The colored cells are nonzero with blue representing the cells updated during the current iteration. (a) Input coefficient matrix. (b) Iteration 1. (c) Iteration 2. (d) Iteration 3. (e) Iterations 4 and 5. (f) Iterations 6 and 7. (g) Iterations 8 and 9. (h) Iterations 10 and 11.

2) During the row transform stage, the number of iterations required per row is limited to the number of columns where nonzero coefficients are present in the intermediate matrix. This is equivalent to the number of columns with at least one nonzero coefficient in the original input matrix and will always be constant for each row.

These observations lead to a simple strategy for ensuring maximum utilization.

3) Strategy: During the column transform stage, the scheduler will keep track of which columns contain nonzero coefficients, and only these will be fed to the row transform stage.

4) Example: Consider a 4×4 discrete cosine transform transformed coefficient matrix with three nonzero coefficients [Fig. 4(a)]. Assuming a diagonal coefficient scanning method, the entropy decoder would produce the coefficients in the order marked. Fig. 4(b)–(d) represents column transform and Fig. 4(e)–(h) represents row transform. Since only two columns have nonzero coefficients in the input matrix, each row transform would consume two iterations.

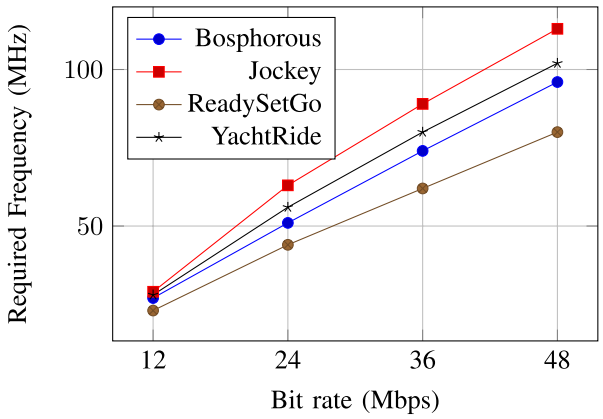

Fig. 5. Required frequency of operation of the inverse transform stage for QHD streams (obtained from [11]) encoded at different bitrates using x265.

Based on this strategy, a single $N \times N$ transform with c nonzero coefficients spanning across k ($k \leq c$ and $k \leq N$) columns consumes $c + Nk$ iterations.[1]

Unlike prior architectures where the throughput is fixed for a given resolution and frame rate, the minimum required frequency of the proposed scheme is highly dependent on the coefficient distribution. Simulations based on actual QHD sequences (Fig. 5) show that the minimum required operating frequency is comfortably within $3\times$ the bitrate for QHD streams.

Exploitation of the sparsity of coefficients is highly important for FPGA-based high throughput inverse transform architectures. For example, the traditional architectures presented in [13] and [14] require operating frequencies of 270 and 412 MHz, respectively, for real-time decoding of 4K at 30, which is very difficult to achieve on FPGA platforms.

In [12], a zero column skipping strategy is adopted to make use of some of the sparsity. However, this is limited to the skipping of column transform for columns that consist of all zero coefficients. Our approach, however, is able to skip each and every zero coefficient (both in column and row transforms), because we only operate on a single coefficient per cycle. This is in contrast to four coefficients per cycle as in [12], which leads it to not being able to take advantage of isolated zero coefficients unless they occur contiguously.

The proposed architecture is naturally capable of disregarding completely empty residual blocks. Although this feature is not present in [12], it is not difficult to add this at a higher level of architecture by looking at the *cbf* flag from entropy decoder. Hence, for a fair comparison of the two approaches,

[1]This is of course assuming that the entropy decoder can produce one nonzero coefficient per cycle. Even if it does not, the inverse transform does not consume inputs during the column transform stage and the FIFO placed between can be used to regulate the flow.

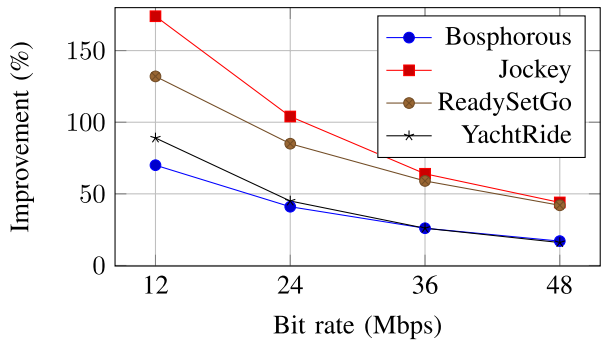

Fig. 6. Percentage improvement in throughput by the proposed architecture compared with that by the architecture in [12].

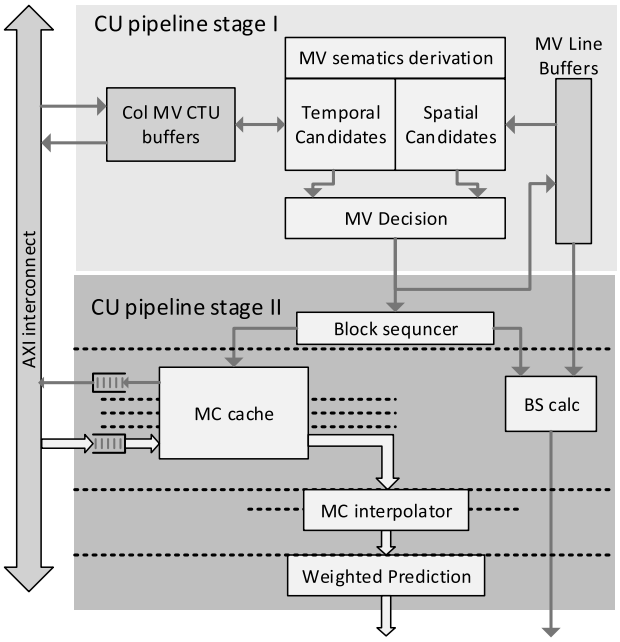

Fig. 7. Inter-prediction architecture. Depicted in dashed lines are 4×4 subpipeline stages.

we assume that this optimization is present and present the simulation results in Fig. 6.

C. Inter-Prediction Architecture

Inter-prediction architecture comprises two CU level pipeline modules: *motion vector (MV) computation engine* and *sample generation engine*. The latter implements a subpipeline of 4×4 block level for each CU that includes an *MC cache*, *MC interpolator*, and *weighted prediction* module. In addition, a separate *boundary strength calculator* module resides in *sample generation engine*, as shown in Fig. 7.

1) Motion Vector Computation Engine: MV computation engine takes relevant SEs in and derives semantics to produce MV components and reference indices for each PU in the decoding order. To guarantee required throughput of an *MV computation engine* even in a worst case quad-tree partitioning and prediction block (PB) partitioning, spatial and temporal candidates are computed in parallel despite that HEVC standard specifying temporal candidates would be

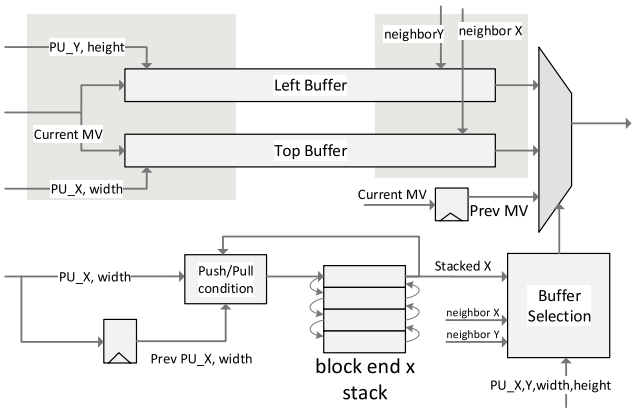

Fig. 8. MV line buffer architecture.

required only if there are no sufficient spatial candidates. The spatial candidate selection module interacts with MV line buffers, as shown in Fig. 7. The temporal candidate selection module interacts with a maximum coding tree unit (CTU) size MV buffer that has prefetched temporal MVs from DRAM. Due to prioritization order of spatial candidates, we sequentially check availability of candidates and fetch candidates from the MV line buffer in a pipelined fashion, reducing the resource requirement. The following are the novel architectural contributions for inter prediction.

a) Efficiently packed spatial MV candidate storage mechanism: HEVC standard dictates five spatial neighboring candidates to be referred to while decoding a PU. MV information of neighboring candidates can be predominantly stored in top and left line buffers. However, given the irregular block sizes of CU, it is possible that the MV location at the bottom right corner of a PB slots into both top and left line buffers, while some potential candidate MVs get prematurely erased if a naive approach is followed for line buffer management. A naive approach is where the MV for decoding block gets written into corresponding projected locations in top and left line buffers based on the height and width of PU. A separate maximum CTU size array would be required for storing of MVs that get displaced but has the potential to be referred again.

We propose a stack-based architecture for line buffer management as shown in Fig. 8. This technique eliminates some of the redundancies by introducing a new write policy to the top and left line buffers after completion of a PB, as shown in Fig. 9. The new policy helps to retain displaced MVs in the buffers themselves without having to maintain a separate memory space for them. This saves SRAM requirement by at least 18.5 kbits. The block end X coordinate of the immediate past PU and the current PU are checked against the top of the stack content to determine stack push or stack pull condition. If top of the stack content matches with neighbor's X coordinate for a top line buffer request, MV data read from the left line buffer is passed through instead.

b) Multipurpose scheduling for MV line buffers: Boundary strength calculation as specified by HEVC standard requires MV information on adjacent 8×8 blocks. We propose

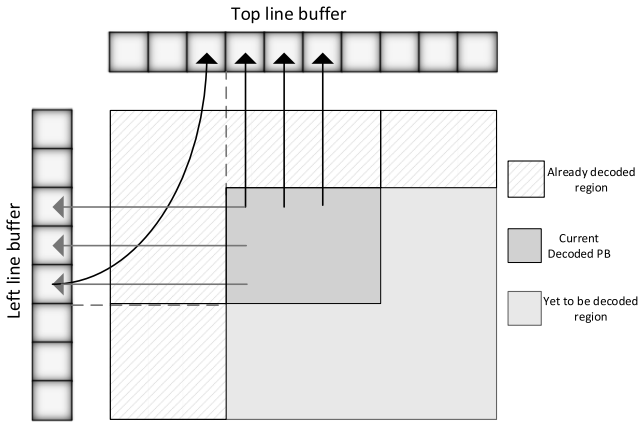

Fig. 9. New write policy for MV line buffers.

reusing the same MV line buffers used by the MV computation engine for this purpose without maintaining a duplicate buffer. Further, we eliminate the need for incurring a separate read interface to handle requests of the boundary strength module by introducing an efficient scheduling strategy for MV line buffers.

According to the proposed scheduling strategy, when the MV for a PB is determined, it is registered and written into PB's bordering locations in line buffers, while the next MV decoding is started. By employing a read-first-type memory, the locations being overwritten are read in the same cycle and sent to boundary strength module over a FIFO of depth 16. This process gets momentarily halted if next MV decoding requires to access spatial candidate MVs from the line buffers. This is because spatial candidate fetch is critical for the completion of MV decoding, while border location update can occupy until next MV is decoded. Since there is a risk that a spatial MV candidate could refer to immediate past PB and that location is not updated at the time it is read, all requests referring to immediate past PB are served by registered previous MV value, as shown in Fig. 8.

c) Temporal MV buffer in DDR3 and prefetch schedule: Temporal MV memory requirement is about 5 MB for level 5 decoding. Hence, it is stored on chip DRAM. As per HEVC standard, MV descriptors referred from previous frames are subsampled at 16×16 pixel blocks. Further, as far as one CTU is concerned, previous MVs are referenced from only one reference frame. This means a total of 20 descriptors (4 rows $\times$ 5 columns) are sufficient for decoding a CTU. (The four columns from the colocated CTU + rightmost column of the left neighbor CTU of the collocated CTU.) A CTU-level prefetch schedule is proposed in our architecture due to these reasons. The prefetch helps to negate the effect from longer DRAM access latency. By exploiting burst mode in advanced extensible interface (AXI), required MV descriptors could be fetched in 7 beats, which means the total cycle time for prefetch would not exceed 50 cycles on average (DRAM latency accounted). This is only a fraction of 2% compared with the total number of cycles allocated for CTU decoding. At the end of decoding of a CTU, if applicable, the 16 MV descriptors are written to DRAM for future reference.

2) Two-Pipeline-Stage MV Interpolator: To cope with the throughput requirement for QHD at 30 frames/s, it is necessary that MC interpolator should process 11×11 reference pixel blocks in four clock cycles for biprediction. Hence, we propose the interpolator architecture with two pipeline stages. By decomposing into two pipelined stages as in horizontal and vertical filtering, we make sure that the worst case throughput for a 4×4 block is improved from 14 to 5 cycles compared with the time-multiplexed reuse of horizontal and vertical filters. This implementation requires 11 8-bit eight-tap filters at the horizontal filter stage, while four 16-bit eight-tap filters for vertical filtering for luma. For chroma, it requires five 8-bit four-tap filters and two 16-bit four-tap filters.

D. Intra Prediction

The intra prediction consists two pipelines, as shown in Fig. 10, and both of them produce 4×4 pixel data with varied throughputs: the main pipeline is to produce blocks coded with angular and planar modes and the alternate pipeline is responsible for producing blocks coded with dc mode. There are several key prepipeline activities in order to reduce the latency in pipelines.

Intra prediction inherently could not be pipelined over transform units (TUs) as it may require the decoded pixels of the latest previous TU. In order to reduce the interim memory usage, a 4×4 block level pipeline is adhered to. According to [1], FPGA designs are usually 3.4–4.6 times slower compared with their counterpart, standard cells. In order to run at the desired frequency, factorization should be done to avoid redundant computations in the main pipeline. Therefore, this design introduces several key prepipeline activities as a result of such factorization.

The main prepipeline activity is the generation of valid ranges for the addresses of line buffers. The line buffer capable of producing eight consecutive pixels from a given address require these ranges to clip off the invalid addresses. The available range generation module will provide a maximum address that data are valid depending on the current decoding location. In a similar way, constrained range generation module will provide a minimum address. If the maximum address boundary is met within the 8 pixels, the pixels that exceed the boundary will present the maximum value instead of the data at their respective locations, using a multiplexing logic. A similar treatment would be done if the minimum valid address is met within the 8 pixels.

1) There Are Mainly Two Scenarios That Authors Wishes to Highlight in Regard to This Matter:

a) Constrained intra-prediction flag is set to zero: The above-mentioned flag that is set by the encoder notifies the decoder that the intra prediction will use decoded pixel data from both intra and inter modes. If this is the case, there would be a maximum valid address range for a given location (depending on the z-scan order), tile, and slice. The responsibility of the available range module is to calculate this range once for a TU and it would only consume a single clock cycle.

b) Constrained intra-prediction flag is set to one: In contrast, when the flag is one, it will only use decoded pixels from

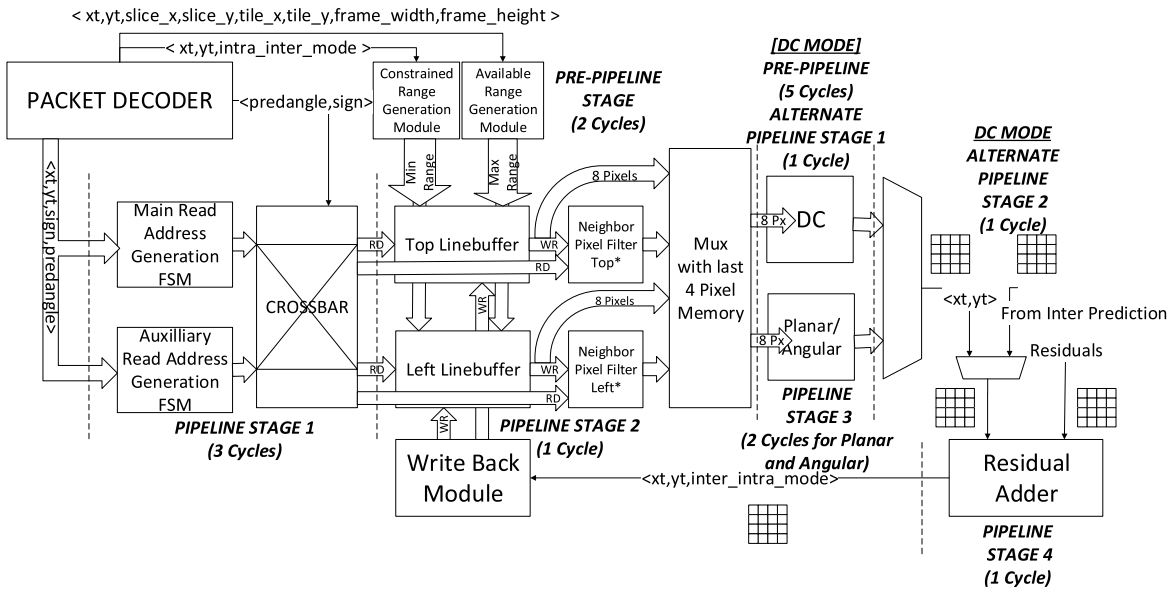

Fig. 10. Intra-prediction architecture.

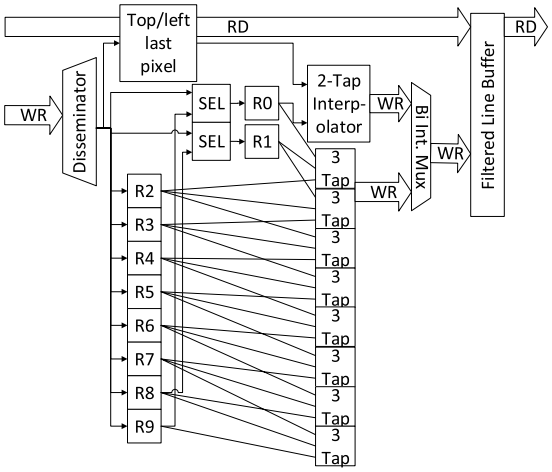

Fig. 11. Reference pixel filter.

TABLE I
SUMMARY OF CYCLE CONSUMPTION FOR INTRA PREDICTION

Block Size	Prediction Mode	Fixed Cycle Overhead	Variable Cycle Cost	Total Cycles	Total pixels	Pixel/ cycle
4	DC	10	1	11	16	1.4545
	P/A	8	4	12	16	1.3333
8	DC	10	4	14	64	4.5714
	P/A	8	16	24	64	2.6667
	P/A Filt	14	16	30	64	2.1333
16	DC	10	16	26	256	9.8461
	P/A	8	64	72	256	3.5556
	P/A Filt	14	65	79	256	3.2405
32	DC	10	64	74	1024	13.838
	P/A	8	256	264	1024	3.8378
	P/A Filt	14	259	273	1024	3.7509

intra mode. The write back module will pad the line buffers toward the corner pixel when the first intra block is received after several or one inter blocks are decoded. Afterward, it will extend the last pixel of intra blocks to the size of inter blocks opposing to the corner pixel direction. This will not affect the overall latency as it is done just after a 4×4 block is decoded in a pipelined manner.

2) Although the Line Buffer Is Intended to Give Pixel Data Straight Away Incorporating the Above, There Might Be Two Other Optional Activities to Be Carried Out:

a) Reference pixel filter: Depending on the mode and block sizes, the decoder will be required to do neighbor pixel filtering. This is implemented in a similar manner to the line buffer but with reduced storage and interpolation filters prior to the write interface, as shown in Fig. 11.

In dc mode, the average value needed to be calculated prior to producing a single pixel. An adder tree with 16 leaves is used to calculate the sum in a pipelined manner. This would consume $(N/8 + 4)$ clock cycles with N being the size of a dimension of the TU. The added delay in this mode is compensated for by the use of an alternate pipeline.

Afterward, the core computation of intra prediction is performed in two pipelines: the main pipeline having a throughput of one 4×4 block per 4 cycles and the alternate pipeline producing a 4×4 block per cycle. The alternate pipeline, used by the dc prediction mode, has an overhead of calculating the average value that is compensated for by its higher throughput. The core computations of angular or planar mode are a 2-tap interpolation of reference pixels based on the angle/mode.

It is important to note the above-mentioned prepipeline activities has rendered more headroom for core computations needed due to inherent routing delay prevalent in FPGAs, enabling the design to perform intra prediction at an average rate of 2 pixel/cycle rates, as shown in Tables I and II. The benchmark 4K at 30 frames/s, could be expressed as the need to decode 248 832 000 ($3840 \times 2160 \times 30$) pixels in 150 000 000 cycles; hence, it is needed to achieve a rate of 1.65888 pixels/clock. The only case according to Table I, which will fail to perform real-time decoding, is when the complete frame is encoded as 4×4 blocks. However, this

TABLE II
BLOCK SIZE AND PREDICTION MODE ANALYSIS

Block size		HoneyBee(4K) (3840x2160)	YachtRide(4K) (3840x2160)	Jockey(4K) (3840x2160)	ShakeNDry(4K) (3840x2160)	CrowdRun(4K) (3840x2160)	PeopleOnStreet (2560x1600)	BasketballDrive (1920x1080)
4	DC	10490	3363	7948	3461	3422	6490	750
	P/A	33406	26575	28004	11631	22270	58730	11518
8	DC	3766	3446	3050	1447	1843	2755	670
	P/A	16348	18691	12702	5708	13102	13250	7103
16	DC	1574	2054	1644	994	1536	1316	252
	P/A	6622	5738	5440	3466	7038	3499	1602
32	DC	1381	1453	1404	1962	1590	403	249
	P/A	1876	1793	2127	2217	3031	357	635
Total		75676	63547	62632	31417	53832	87070	22779
P/Cycle		2.3124	2.543	2.4135	3.0499	2.6855	1.8548	2.1365

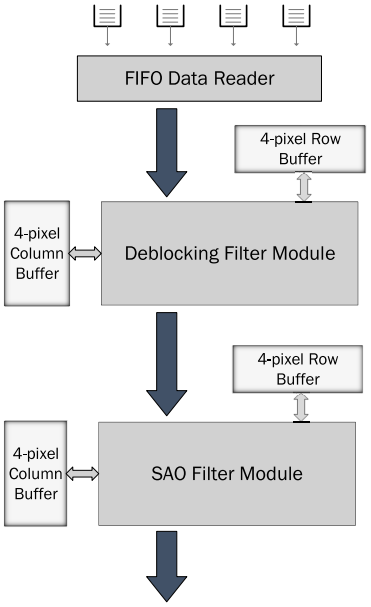

Fig. 12. In-loop filter architecture.

will not be a probable case with the intention of HEVC being bitrate reduction and could be justified with the analysis presented in Table II, which includes block statistics for I-only frames. According to our intra-prediction module, we will have, on average, a 2.601-pixel/clock decoding rate for 4K sequences.

E. In-Loop Filtering

The loop filters in HEVC consist of the deblocking filter (DBF) and sample adaptive offset filter (SAO). They are used to remove blocking and ringing artifacts introduced by the lossy encoding and decoding process. References [15] and [16] explain in detail the operation of the HEVC deblocking filter and sample adaptive offset filter, respectively. The high-level architecture of our loop filter implementation is shown in Fig. 12.

In keeping with our design methodology, the loop filter stage is implemented as a main (elastic) pipeline stage of the full decoding pipeline. It is connected to the prediction module through FIFOs that provide a flexible pipeline interface allowing both modules to run at their own speed. The loop filter stage is further subdivided into two stages as the DBF stage and SAO stage, respectively. These two modules are pipelined at 8×8 block level for maximum efficiency. Both modules have their own line buffers, which are used to store pixels until they can be filtered. This loop filter architecture is designed in such a way that it balances the pipeline latency and critical path delay so that it can support real time 4K decoding on the FPGA. It is able to handle all types of input streams that come in any valid decoding order, including support for multiple tiles without using any external memory. This architecture is able to achieve this performance due to the following reasons.

1) Highly flexible 8×8 block level filtering order.
2) Efficient use (and reuse) of line buffers that eliminates the need for CTU sized buffering.
3) FPGA-aware pipelined design to minimize critical path delay.

As shown in the architecture in Fig. 12, the prediction module fills the data FIFOs with data related to one 8×8 block at a time as they are decoded. Each 8×8 block contains a header with its (X, Y) coordinates, luma and chroma pixel values, and loop filter parameters. These 8×8 blocks can come in any valid decoding order. That is, when the current block is parsed, its left and top neighboring blocks should already be parsed to the filter at some point earlier in time. This is guaranteed by the standard under all decoding modes. This highly flexible block filtering order makes it easy for the prediction module, especially when dealing with variable sized CUs, since the loop filter does not impose any restriction on the processing order of 8×8 blocks.

Fig. 13 shows the usage of line buffers in the DBF module. As the DBF module reads an 8×8 block at a time from the data FIFOs, it also fetches the top and left neighboring 4×4 blocks from the top and left line buffers. The line buffers are four-pixel wide and have a length that matches the maximum picture dimensions. They contain pixel data as well as filtering parameters including tile/slice specific flags. Therefore, it has all the information needed for filtering. The line buffers are implemented using BRAMs in the FPGA. These are built-in on-chip RAMs available in the FPGA and hence consume no additional logic resources.

Once the DBF module has all the blocks in place, it filters the vertical edges followed by the horizontal edges. Once the filtering is done, the old blocks in the line buffers are overwritten with the newly received blocks, as shown in Fig. 13.

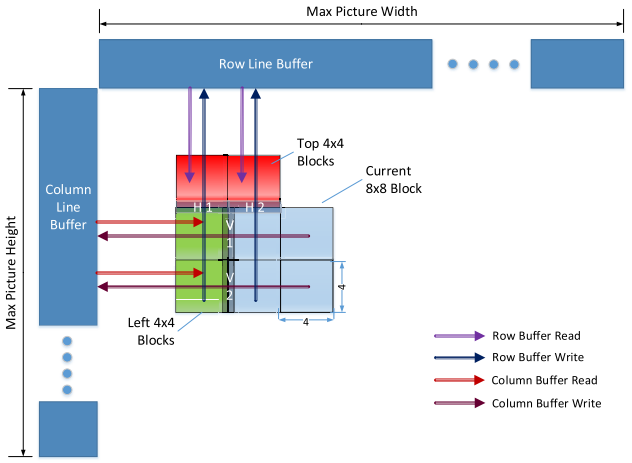

Fig. 13. DBF line buffer usage.

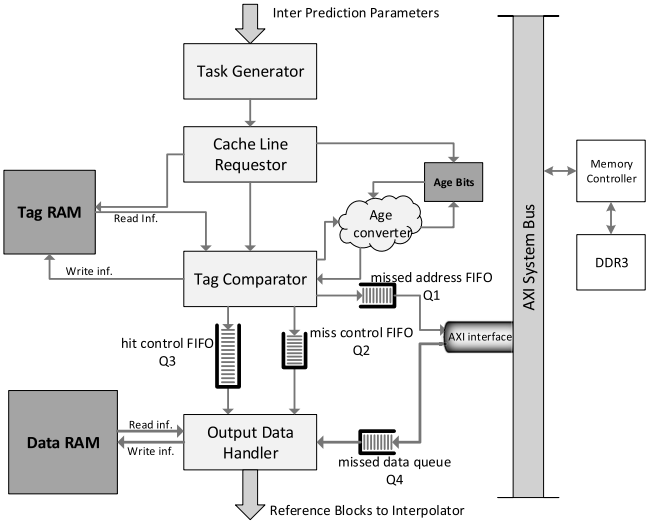

Fig. 14. MC cache architecture.

In this way, at any given point of time, the line buffers contain the bottom and right most pixels to be filtered in the current frame. This efficient reuse of existing line buffers eliminates the need for any additional buffering within the DBF module.

The SAO module operates very much the same way. The DBF and SAO modules are pipelined at the 8×8 block level. The output 8×8 block from the DBF module has a 4-pixel offset to the left and top directions, as shown in Fig. 13. The SAO module also has its own line buffers to store the deblocked pixels until they are ready for SAO filtering. These line buffers operate in the same way as described before.

In [2] and [17]–[18], the loop filter stage operates on a CTU (up to 64×64) basis, which requires a CTU sized buffer at each stage. This increases the amount of registers and memory resources required at each stage while also increasing the overall latency in the loop filter stage. With our efficient reuse of line buffers, we are able to run the pipeline at 8×8 block level, filtering the blocks on arrival (regardless of the order of arrival) without CTU level buffering. This approach saves resources while minimizing the latency in the loop filter stage. With regard to tile handling, the loop filter architectures in [18] and [19] do not support multiple tiles, while [17] requires an external memory connection for tile handling. The architecture in [2] supports tile handling using line buffers that cover the entire height of the picture. We too have a similar line buffer architecture that spans the entire height of the picture, so we are able to handle multiple tiles without discrimination and without the use of an external memory connection. This is important because an external memory connection to the DRAM has two adverse impacts. First, external memory transfers introduce a further unaccounted and unpredictable delay to the filter stage. Second, by increasing the data traffic on the memory bus, this would not only slow down the filter stage but also would slow down the operation of the inter-prediction module that requires significant memory bandwidth itself.

When designing both filters, special care was taken to minimize critical path delay. The filtering operations contain a long chain of addition and shifting operations, which could be done in one single clock cycle in an ASIC design. But the resource placement constraints and longer routing delays in an FPGA mean that this would result in a very long critical path resulting in the design failing to meet the timing closure when implemented on an FPGA. Therefore, such operations were broken down to a set of smaller steps with register stages in between. The result was a design that can run at over 150 MHz when actually implemented on an FPGA. It is able to operate on 4K 30 frames/s video streams in real time.

IV. Motion Compensation Cache

The decoder employs an MC cache to reduce off-chip DRAM traffic, which reaches data rates beyond 42 Gb/s otherwise. DRAMs also demonstrate high memory access latency, paving way to deplete the overall throughput. Parallel bank cache architectures have been proposed in [20] and [21] for H265, and [22] for H264. However, they use special mechanisms to take care of cache conflicts and out-of-order task completion. We propose a single bank cache architecture with capability to provide reference pixel throughput for 4K at 30 frames/s. The following are the key novel contributions in the MC cache.

A. Single Bank Cache Architecture

The cache is implemented on a single bank so that one cache line could be read or written at a time. As shown in Fig. 14, the data RAM and tag RAM have simple dual port interfaces.

The operation of the cache can be described as follows.

1) *Task Generator:* Admits prediction parameters and breaks down PB into smaller units called fundamental blocks (FBs) that are 4×4 pixels in size. A task is defined for each FB as serving the corresponding reference region known as reference block (RB).
2) *Cache Line Requestor (CLR):* Identifies a group of cacheable entries (CEs) that the RB of a particular task is distributed into and places set addresses in the read address port of tag RAM, for each CE sequentially.

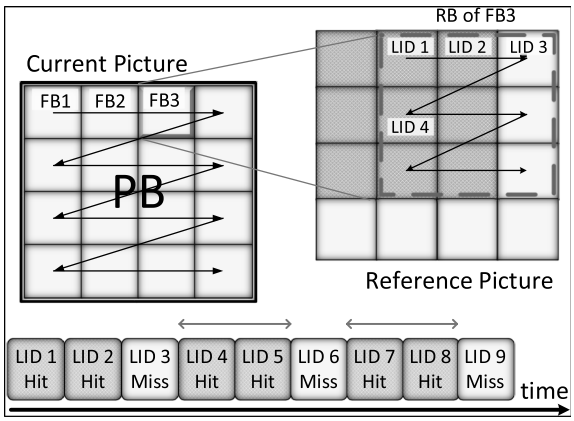

Fig. 15. Cache line ID ordering for maximum spacing between misses.

A CE could be any pixel region in the reference picture buffer (RPB) that fits into a single cache line together.

3) *Tag Comparator Unit:* Determines whether a CE is present in the cache, or else, the missing CE has to be fetched from DRAM, by initiating a read transaction via missed address FIFO connected to the main system bus. Based on hit/miss result, cache address and index information for mapping cache line to output block is passed either via miss control FIFO (Q2) or hit control FIFO (Q3), as in Fig. 14.
4) *Output Data Handler (ODH):* Manages inputs from control FIFOs, access cache data RAM, and miss data queue from DRAM interchangeably to dispatch pixels from cache lines into the output buffer. The reason for setting up two control queues for hits and misses is that read cycle latency to cache data RAM is different from that of write latency. To match this difference, read address needs to be put a cycle earlier.

Cache conflicts can be avoided by ensuring that ODH processes cache lines in the same order in which CLR initiated them. To support ODH in deciding which control FIFO out of Q2 and Q3 to read from at any given context, a task ID and a line ID are encoded and sent via the relevant queue. Line ID is used to form the access sequence of CEs in a given task.

By providing the miss cache line update controllability to ODH that solely manages cache data reads, as shown in Fig. 14, task completion order is maintained properly. Some of the earlier MC cache designs such as [21] suffer from out-of-order cache completion problem, costing more resources and clock cycles for additional checks.

When FBs in a larger PB are processed in a raster scan order, as shown in Fig. 15, raster scan line IDs ordering maximizes the spacing between consecutive misses. This is important because system bus at times fail to provide cache missed data from DRAM in consecutive cycles due to arbitration between the rest of the masters accessing the DRAM.

B. Cache Parameter Selection

The presented architecture aims to process one cache line per clock cycle. With declaration of a larger cache line size, higher throughput can be achieved, but there are hardware implications that come along with it. A larger cache line considerably downgrades circuit speed because of high fan-in path generated from cache line to output buffer of the cache.

TABLE III
EFFECT OF CACHE LINE SIZE ON PERFORMANCE PARAMETERS

Cache Line	Max. Cycles/RB	Min. Pixels/Cycle	LUT usage*	Max Frequency*
4x4 pixels	16	0.50	3993	619 MHz
4x8 pixels	12	0.66	8833	480 MHz
8x8 pixels	9	0.89	16577	410 MHz
8x16 pixels	6	1.33	33033	351 MHz

*Figures are based on Vivado Design Suite, and are applicable to Output Data Handle stage only

Based on Table III, which shows how worst case throughput and resource usage varies with the size of cache line, a cache line of 64 pixels arranged in a square of 8 pixels-a-side emerges as the best trade off. It has 33% worst case throughput increase over a 4×8 cache line, but only 49.8% of the area of a design with a 128-pixel cache line.

A cache size of 12 kbits is preferred since it is approximately the sufficient space to store two 64×64 regions and fully utilized when the largest PB size is decoded in bipredction mode. An 8×8 cache line thus results in 128 cache lines of size 96 bytes (64 bytes in luma layer and 16 bytes each from two chroma layers), which is arranged into 32 sets, each set with four cache lines. The four-way set associativity helps to avoid premature replacement of cache lines.

C. Uninterrupted Cache Line Scheduling

FIFO queues Q1, Q2, Q3, and Q4 facilitate UCLS, whereby all the key pipeline stages of the cache, namely, cache line requestor, tag comparator, and ODH operate at their full capacity without starving for input data or getting stalled by backlog of output data. Tag comparator unit (TCU) processes incoming block requests and pushes the result to either Q1 and Q2 (in case of a miss) or Q3. By exploiting multiple acceptance capability in Xilinx AXI interconnect and memory interface generator, it is possible to queue up to 32 outstanding read requests to DDR3 via Q1. With an average latency of 27 cycles and a throughput of 1 request per cycle from Q1 to Q4 (by which time a maximum of 27 outstanding transactions would have been queued in memory pipeline, in a very rare case of 27 consecutive misses), chances of TCU stalling due to backpressure from memory is minimum.

If TCU guarantees passing data to one of Q2 and Q3 at every clock cycle, it necessarily means that once the initial latency for arrival of first cache line through Q4 has elapsed, the ODH will not starve of data from Q2/Q3, while Q4 provides TCU-requested miss cache lines on time. At steady state, therefore, all pipeline stages will process one hit or miss cache line per every clock cycle. Thus incurring a zero miss penalty at steady state, meaning miss data become available as and when they need to be consumed.

However, it should be noted this behavior is observed as long as inter-PBs are decoded. Whenever an intra-block

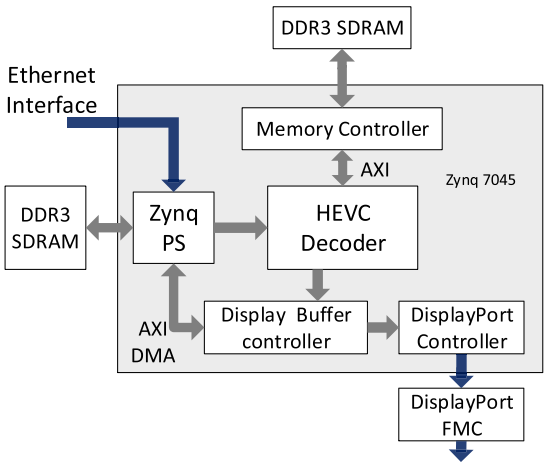

Fig. 16. FPGA test environment.

turns up for decoding, TCU become idle and gradually sinks the pipeline. Initial latency will be again observed at worst on resumption. However, it is expected that superior decoding rate in intra prediction will make up for the latency loss in MC engine.

D. DRAM Bandwidth Management

DDR3 synchronous dynamic random access memory (SDRAM) modules can be generally operated at a maximum of 800 MHz, which is much higher than the synthesized frequency of the FPGA design. Hence, we adopt a 4:1 PHY to controller clock ratio to enable higher data rate across the DRAM. Since DDR3 64-bit data transfers occur on dual edges, the effective throughput on the double data rate (DDR) memory bus is 8× the system bus. To match at the system side, AXI data bus width between memory controller to decoder core is configured to 512 bits.

V. Results

The fully hardware based HEVC decoder was designed using Verilog hardware description language (HDL). The decoder was synthesized and implemented using Vivado Design Suite along with peripheral IP cores that are required for an end-to-end system deployment. Fig. 16 shows the interconnection of all peripheral blocks along with the main decoder core in the FPGA test environment. The system was implemented on a Xilinx ZC706 development board that features a Zynq 7045 programmable system on chip (SoC). The Zynq SoC contains PL as in an FPGA as well as a PS, which has two Advanced Risc Machine cores, all in a single chip. The HEVC decoder was implemented on the PL side.

In this test configuration, the Ethernet interface on the PS side is connected to a PC that sends the HEVC encoded bitstream via Ethernet. The Ethernet packets are handled by the PS and HEVC bitstream is passed to the decoder. The decoded output frames are sent to an RPB that resides in the external DDR3 SDRAM on the PL side. In addition, for displaying purpose, the frames are also written to a display buffer that resides in the external DDR3 SDRAM on the PS side (which is accessible to both PS and PL). On the display side, a DisplayPort1.1 interface is used to send the decoded

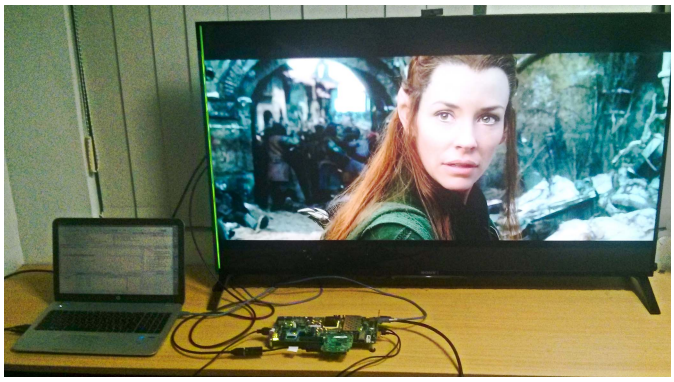

Fig. 17. Demo test setup.

4K video output to a display device (4K TV in this case). We have used a DisplayPort FPGA mezzanine card daughter board along with a DisplayPort controller core to implement the DisplayPort interface. A snapshot of test environment is given in Fig. 17.

A. Performance Results

Critical paths in the decoder are optimized to achieve a maximum runnable frequency of 180 MHz on the FPGA. Some of the FPGA-specific techniques used to improve timing include: setting max fanout limit contraint on several high fanout enable/reset signals, enforcing multicycle latency on DSP datapaths, and using high-speed AXI interconnects. This helps to avoid timing violations at a 150-MHz operating frequency. For real time decoding of 4K at 30 frames/s, the decoder have a clock cycle budget of 5 000 000 per frame based on the 150-MHz frequency. This is equivalent to 1.6588 pixels per cycle. All submodules in the decoder are designed to achieve this throughput. The decoder has been tested for performance on various sequences and Table V details the simulation results obtained for some of them. The *BasketBallDrive* and *PeopleOnStreet* sequences are from the Joint Collaborative Team on Video Coding common test conditions [24]. The *CrowdRun* sequence [25] was encoded from YUV using x265 at various bitrates (options—prest slow). Four pre-encoded 4K sequences from [11] were also analyzed.

1) Front End: The performance of the front end is directly related to the input bit rate and this is confirmed from Table V. The decoder can go up to a bitrate of 37 Mb/s, which is comfortably above the 25 Mb/s mandated for level 5 main tier.

2) Back End: Back-end decoder performance is mostly dependent on decoded slice type, which determines the amount of DRAM access bandwidth and the associated latency. The average 4K frame decoding time for intra-predicted slices is generally low and is in the range of 15–20 ms, since no DRAM access is involved. For unipredicted slices, frame decoding time is averaged at 25 ms and frame decoding time for bipredicted slices also comfortably sits below 33.33 ms demarcated by level 5 main tier of the standard for 4K real-time decoding. As shown in Table V, even for other test configurations, back-end modules satisfies the frames per second requirement.

TABLE IV
COMPARISON WITH THE PRIOR WORKS

	ISSCC [4]	**A-SSCC [23]**	**TCE [2]**	**ESSCC [5]**	**This Work**
Platform	40nm ASIC	90nm ASIC	65nm FPGA	28nm ASIC	Zynq 7045
Target Resolution	2160p30	1080p30	1080p30	2160p60	2160p30
Resources : Logic DSP Blocks Registers	715k gates	446k gates	74200 ALM* + 341 DSP + 113k	3454k gates	126k LUTs + 35 DSP + 58k
Memory (SRAM)	124 KB	10.2 KB	663 KB	154 KB	754 KB
Max. Freq. (MHz)	200	224	180	350	150

*One ALM (Adaptive Logic Module) in the Altera architecture can be used to implement up to 2 6-input LUTs.

TABLE V
PERFORMANCE RESULTS FOR SELECTED SEQUENCES

Sequence	Type	QP	Bitrate (kbps)	Cache				FPS	
				Pixels/Cycle	Hit rate	BW saving	BW (Mb/s)	Front End *	Back End *
BasketBallDrive	RA	22	10,192	2.31	0.89	0.69	1287.4	**103.9**	120.5
(1920x1080)		37	879	2.53	0.89	0.70	1297.1	830.2	**136.8**
	LD	22	11637	2.25	0.92	0.79	865.7	**91.3**	121.5
		37	943	2.54	0.92	0.79	767.5	772.4	**153.8**
PeopleOnStreet	RA	22	32065	3.58	0.85	0.56	2094.8	**34.2**	58.3
(2560x1600)		37	4528	3.24	0.88	0.63	2054.7	165.1	**71.3**
	LD	22	36683	3.41	0.89	0.66	1789.1	**30.2**	61.2
		37	4899	3.36	0.90	0.71	1400.5	150.8	**87.3**
QHD Sequences									
HoneyBee	RA		12000	3.12	0.87	0.61	4549.7	89.4	**35.5**
Jockey	RA		12000	2.34	0.92	0.74	4538.9	86.4	**33.9**
ReadySetGo	RA		12000	3.18	0.88	0.62	4701.4	86.6	**33.8**
YachtRide	RA		12000	2.45	0.90	0.70	4861.7	87.3	**34.0**
Crowd Run	RA		12000	2.95	0.89	0.67	4607.2	84.1	**38.3**
	RA		16000	2.85	0.89	0.67	4739.6	69.8	**37.2**
	RA		20000	2.89	0.88	0.67	4623.4	58.5	**37.6**
	RA		24000	2.92	0.88	0.67	4543.2	46.3	**37.4**

*Denoted in bold-face are the limiting FPS which determine the overal FPS

3) MC Cache Performance: Pixel throughput of the MC cache is directly related to MV information as it determines the number of cache lines an RB can be spread into. Hence, the throughput is statistically evaluated. A cache throughput of 2.87 pixels/cycle, hit ratio of over 89%, and bandwidth saving of more than 70% are comparable and at times superior to those in [4], [21], and [22].

B. Resource Utilization

Module-wise break down of resource utilization for main submodules of the decoder is given in Table VI. FIFOs used in interconnecting the modules are taken as a separate entry. Lookup table (LUT) usage for the decoder is 57.6% of the total available on the target FPGA. Motion interpolation filter chain datapath is technology mapped to DSP48 primitives to even out resource distribution. Other resource types are predominantly underutilized.

An analysis of how the proposed system compares to the prior work is in Table IV.

TABLE VI
MODULE-WISE RESOURCE USAGE

Module	LUTs	Registers	BRAMs(18Kb)	DSP48
Entropy Decoder	4415	1997	25	0
Inverse Transform	10840	2623	32	0
Inter Prediction	29496	12568	8	34
Intra Prediction	43166	22160	94	0
DBF/SAO	17332	10898	40	0
MC Cache	20502	6280	61	1
FIFOs	350	1041	64	0
Total	126101	57567	335	35

VI. CONCLUSION

An FPGA implementation of the HEVC decoder, capable of decoding 4K 30 frames/s video sequences, was presented. Challenges of HEVC, variable matrix sizes for inverse transform, increased number of prediction modes,

a large number of reference pixels for interpolation, and high bandwidth memory access via standard DDR3 memory interface amalgamated with inherent routing delay and placement constraints prevalent in FPGAs were addressed in this paper. The maximum pipeline width of 8×8 block sizes throughout the design incorporated with elastic buffers contributed to the seamless integration of the modules. Single-cycle reference pixel padding and fetching in intra prediction allowed the average of 2.579 pixel/clock in decoding for all-intra frames. Single-bank cache architecture with UCLS technique reduces DRAM bandwidth by 70% while achieving an average throughput of 2.87 pixels/clock barring input delays.

ACKNOWLEDGMENT

The authors would like to thank the Facilitation Center for Advanced Electronics Design at the Department of Electronic and Telecommunication Engineering at University of Moratuwa for providing the resources.

REFERENCES

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[2] D. Engelhardt, J. Moller, J. Hahlbeck, and B. Stabernack, "FPGA implementation of a full HD real-time HEVC main profile decoder," *IEEE Trans. Consum. Electron.*, vol. 60, no. 3, pp. 476–484, Aug. 2014.

[3] S. Cho, H. Kim, K. J. Byun, and N.-W. Eum, "HEVC hardware decoder implementation for UHD video applications," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2014, pp. 1–3. [Online]. Available: http://www.icassp2014.org/show_and_tell.html, accessed Nov. 2014.

[4] 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 J. Solid-State Circuits*, vol. 49, no. 1, pp. 61–72, Jan. 2014.

[5] C.-C. Ju *et al.*, "A 0.2 nJ/pixel 4 K 60 fps main-10 HEVC decoder with multi-format capabilities for UHD-TV applications," in *Proc. 40th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2014, pp. 195–198.

[6] K. Fleming, C.-C. Lin, N. Dave, Arvind, G. Raghavan, and J. Hicks, "H.264 decoder: A case study in multiple design points," in *Proc. 6th ACM/IEEE Int. Conf. Formal Methods Models Co-Design (MEMOCODE)*, Jun. 2008, pp. 165–174.

[7] *High Efficiency Video Coding*, Geneva, Switzerland, document ITU-T H.265, 2013.

[8] Y.-H. Liao, G.-L. Li, and T.-S. Chang, "A highly efficient VLSI architecture for H.264/AVC level 5.1 CABAC decoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 2, pp. 272–281, Feb. 2012.

[9] Y.-H. Chen and V. Sze, "A 2014 Mbin/s deeply pipelined CABAC decoder for HEVC," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 2110–2114.

[10] J. Hahlbeck and B. Stabernack, "A 4k capable FPGA based high throughput binary arithmetic decoder for H.265/MPEG-HEVC," in *Proc. IEEE 4th Int. Conf. Consum. Electron. (ICCE)*, Berlin, Germany, Sep. 2014, pp. 388–390.

[11] *Ultra Video Group*. [Online]. Available: http://ultravideo.cs.tut.fi/, accessed Sep. 2014.

[12] M. Tikekar, C.-T. Huang, V. Sze, and A. Chandrakasan, "Energy and area-efficient hardware implementation of HEVC inverse transform and dequantization," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 2100–2104.

[13] P.-T. Chiang and T. S. Chang, "A reconfigurable inverse transform architecture design for HEVC decoder," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2013, pp. 1006–1009.

[14] Y. Ziyou, H. Weifeng, H. Liang, H. Guanghui, and M. Zhigang, "Area and throughput efficient IDCT/IDST architecture for HEVC standard," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 2511–2514.

[15] A. Norkin *et al.*, "HEVC deblocking filter," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1746–1754, Dec. 2012.

[16] C.-M. Fu *et al.*, "Sample adaptive offset in the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[17] M. Mody, N. Nandan, and T. Hideo, "High throughput VLSI architecture supporting HEVC loop filter for ultra HDTV," in *Proc. IEEE 3rd Int. Conf. Consum. Electron. (ICCE)*, Berlin, Germany, Sep. 2013, pp. 54–57.

[18] E. Ozcan, Y. Adibelli, and I. Hamzaoglu, "A high performance deblocking filter hardware for High Efficiency Video Coding," *IEEE Trans. Consum. Electron.*, vol. 59, no. 3, pp. 714–720, Aug. 2013.

[19] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A pipelined VLSI architecture for sample adaptive offset (SAO) filter and deblocking filter of HEVC," *IEICE Electron. Exp.*, vol. 10, no. 11, pp. 1863–1869, 2013.

[20] M. Tikekar, "Circuit implementations for High-Efficiency Video Coding tools," M.S. thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2012.

[21] S. Wang, D. Zhou, and S. Goto, "Motion compensation architecture for 8K UHDTV HEVC decoder," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2014, pp. 1–6.

[22] J. Zhou, D. Zhou, G. He, and S. Goto, "Cache based motion compensation architecture for quad-HD H.264/AVC video decoder," *IEICE Trans. Electron.*, vol. E94-C, no. 4, pp. 439–447, 2011.

[23] C.-H. Tsai, H.-T. Wang, C.-L. Liu, Y. Li, and C.-Y. Lee, "A 446.6K-gates 0.55–1.2 V H.265/HEVC decoder for next generation video applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2013, pp. 305–308.

[24] *JCT-VC Common Test Conditions*. [Online]. Available: ftp://ftp.kw.bbc.co.uk/hevc/hm-11.0-anchors/bitstreams/, accessed Mar. 5, 2015.

[25] *Xiph.org Video Test Media*. [Online]. Available: http://media.xiph.org/video/derf/, accessed Mar. 5, 2015.

Maleen Abeydeera received the B.Sc.Eng. degree in electronics and telecommunication from University of Moratuwa, Moratuwa, Sri Lanka, in 2014. He is currently working toward the Ph.D. degree with Massachusetts Institute of Technology, Cambridge, MA, USA.

He was an Associate Architect with ParaQum Technologies, Colombo, Sri Lanka, a startup based on the final year thesis project, from 2014 to 2015. His research interests include high performance computer architecture, dataflow computing, and irregular algorithms.

Manupa Karunaratne received the B.Sc.Eng. degree in electronics and telecommunication engineering from University of Moratuwa, Moratuwa, Sri Lanka, in 2014. He is currently working toward the Ph.D. degree with National University of Singapore, Singapore.

He was an Associate System Architect with ParaQum Technologies, Colombo, Sri Lanka, a startup based on the final year thesis project, from 2014 to 2015. He was responsible for the development of H.265 decoder, advanced packet analyzer technologies, and application development for custom processors. His current research interests include computer architecture, compilers, and reconfigurable computing.

Geethan Karunaratne received the B.Sc.Eng. degree in electronics and telecommunication engineering from University of Moratuwa, Moratuwa, Sri Lanka, in 2014.

He has been an Associate Hardware Architect with ParaQum Technologies, Colombo, Sri Lanka, a startup based on the final year thesis project, since 2014. His research interests include high performance computing and reconfigurable computing.

Kalana De Silva received the B.Sc.Eng. degree in electronics and telecommunication from University of Moratuwa, Moratuwa, Sri Lanka, in 2014.

He has been an Associate Hardware Architect with ParaQum Technologies, Colombo, Sri Lanka, a startup based on the final year thesis project, since 2014. His research interests include hardware architectures for reconfigurable digital systems and hardware accelerated processor architectures.

Ajith Pasqual (M'11) received the B.Sc.Eng. (Hons.) degree in electronic and telecommunication engineering from University of Moratuwa, Moratuwa, Sri Lanka, in 1993, and the M.Eng. and Ph.D. degrees in computer vision from The University of Tokyo, Tokyo, Japan, in 1998 and 2001, respectively.

He is currently a Senior Lecturer and was the Head of the Department of Electronic and Telecommunication Engineering with University of Moratuwa. He leads the Reconfigurable Digital Systems Research Group with University of Moratuwa, where he is involved in the area of hardware acceleration, novel architectures for application specific processors, and systems-on-chip (SoC) to improve performance and power efficiency. He is the Founder of the first Semiconductor Startup Company, ParaQum Technologies, Colombo, Sri Lanka, where he is developing a high performance hardware decoder and encoder for the newest Video Compression Standard H.265/HEVC. His current research interests include computer vision, processor, and SoC architectures.