**Indonesian Journal of Electrical Engineering and Computer Science**
Vol. 12, No. 3, December 2018, pp. 995~1002
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v12.i3.pp995-1002


# Efficient H.264 Decoder Architecture Using External Memory and Pipelining

**G.R. Poornima[1], S C Prasanna Kumar[2]**
[1]Dept. of E & CE, Sri Venkateshwara College of Engineering, Bangalore
[2]Dept. of Electronics & Instrumentation Technology, R V College of Engineering, Bangalore

**Article Info**

***Article history:***

Received Apr 30, 2018
Revised Jul 14, 2018
Accepted Aug 21, 2018

***Keywords:***

H.264 decoder
Pipelining architecture
Memory reusability
Luma/Chroma

**ABSTRACT**

A H.264 standard is one of the most popular coding standard with significant improvement in video broadcasting and streaming application. However it's significant in compression but needs huge calculation and complex algorithm for providing better image quality and compression rate. In H.264 coding technique, designing of decoder is a key factor for efficient coding. In this paper we are designing a decoder using a complex input. We ensured several improvement like looping arrangement, buffer upgradation, buffer supplement, memory reusability and pipelining architecture. We have modified the memory structure also. Our designed decoder achieves a better frame decoding efficiency against state-of-art methods. The proposed approach also provides good area optimization with a maximum frequency of 355 MHz.



***Corresponding Author:***

G.R. Poornima,
Dept. of E & CE,
Sri Venkateshwara College of Engineering, Bangalore.
Email: poornima_g_r@yahoo.com

## 1. INTRODUCTION

H.264 is a most broadly used standard for video coding with significant improvement in video broadcasting, video streaming and optical disc. It is established by JVT (Joint video Team) of ITU-T and ISO/IEC and also known as MPEG-4 part 10 advance video coding. Most of the bits transmitted wirelessly in a communication networks uses MPEG-4 part 10 advanced video coding (AVC). H.264 video coding comprises high compression efficiency, so it is most frequently used in video coding. It has some new features including inter-prediction, intra prediction, variable block size and context-based adaptive entropy coding [1]. These all new feature needs complex compression algorithm and huge calculations to provide better image quality and compression rate. H.264 coding technique involves source code from different domain like computational physics, computer science and machine learning approach, which makes complex source code and challenges in synthesis. The complexity of the H.264 decoder is increased lot compare to MPEG-4 decoder.

Usually a H.264 decoder contains a pipelining architecture of 4*4 sub block. The data processing time and the complexity of each stage of pipelining architecture depends on the type of data and decoding methods. The timing of each stages should be known, so that the stages which requires more processing time can be normalized by efficient decoder architecture. An efficient decoder architecture can idealize all the time consuming stages by reducing the complexity up to certain level.It is observed by run time analysis that the motion compensation (MC) uses 55% of the decoding time. So, it's a crucial factor in designing a decoder architecture [2-3] considering performance. Reading pixel data with less complexity can increase the performance of the decoder. H.264 decoders usually have three units. One is the MC unit discussed above, second is the deblocking filter unit and the last is data out unit which helps in transferring image data on display device.

The size of code, complexity of data structure and function hierarchy should be limited as much as possible. There are different methodologies, which affects the efficiency of decoder. Some researchers followed bottom-up methodologies which includes block level design along with system-level designing. Whereas some researchers followed top-down methodologies which includes the whole function as a single unit. There is also difference between HLS generated hardware and standalone hardware. So, the different methodologies uses in different goal and application. A complex application may be more effective with approaches that breaks the code piecewise and optimize each piece of code where as a simple application may require more methods to get effective.so, it's completely depends to purpose of application.

To reduce the memory scope, several optimization are done by researchers. Some of the HEVC hardware interpolation is presented in [4-7], where they have compared different techniques. In their approach, they did not used memory based implementation. In [4], they have implemented three different 8-tap FIR filter by using a configurable path, which can evaluate single filter output at a time. Due to this reason, it can be used for only motion compensation. In [5], the presented hardware design uses multiplier less constant multiplication (MCM) approach for multiplying with constant factor in [6-7], they have used adders and shifters for generating FIR filters.

In this paper we have demonstrated H.264 video coding with its real application and the types-of complexity which we face on top down approaches. Designing of decoder core is very important in fast and power efficient decoding. Many online platform like YouTube and Facebook using h.264 technique for video coding. Popular manufacturing company like apple and snapdragon also uses H.264 video coding for their processors. This paper includes that how we are improving the design process and what difficulties we are facing while designing a decoder. We synthesized the code, optimized the code and achieved a throughput which outperforms the state-of-art techniques.

This paper is organized as following way. In section 2 we have demonstrated a brief related work of designing a decoders. In Section 3, an overview of H.264 decoder is presented. Section 4 describes our proposed optimization techniques of designing a H.64 decoders. The performance and result evaluation is shown in Section 5. In last section we conclude our paper.

## 2. RELATED WORK

HLS tools much hyped to achieve prototyping and rapid designing of hardware in register transfer level. J. Andrade et al [8] has proved this claim by using a complex application designing, which implements low-density parity-check (LDPC).HLS tool can help user to explore large space designing into multiple small designing, which make its productivity high. HLS tool can also explore micro architecture of the generated design. LDPC decoders are developed by using this HLS tools only, which has an average throughput.

S. Baldev et.al [9] has developed an efficient 5-stage pipelining architecture of daglocking filter for design of HEVC decoder. The luma/chroma samples are applied vertically on edge filters of the designs to get maximum throughput and minimizes the number of clock cycle. This proposed architecture is developed in FPGA and ASIC platform using 90-nm technique. The result of this propose architecture shows that the UHD videos are decoded at 200fps.

F. Leduc-Primeau et.al [10] has developed a design approach, known as quasi-synchronous design approach .LDPC decoder allows timing violation, which is modified through proper modeling by HLS. The new designed circuits can provide same performance with same area constraint but having an energy reduction of 32%.

Designing of Node Processing Units in LPDC decoder [11] is very important for both hardware resources and processing experiences. NPU architectures supports decoder in keeping low hardware utilization with maximum operating frequency. The synthesis outcome proves the hardware efficiency for proposed architecture.

T. Mallikarachchi et.al [12] has proposed a framework to reduce the complexity of decoding. By reducing the complexity of decoder, they have reduced the energy consumption during media playback. They also improved in bit-rate and video quality by designing this framework.

H. Kim et.al [13] has developed an efficient architecture of HEVC for supporting ultra-high definition content by multicore implementation. In standard HEVC technique there is issue of data dependencies, which makes it inefficient for parallel processing. The novel architecture of memory organization solves the problem of data dependencies and makes its efficient for parallel processing. They have implemented de-blocking filter with skip mode pipelining to achieve high performance of throughput.

## 3. H.264 STANDARD VIDEO DECODING

A H.264 standard has a number of profiles, which covers different encoding features, frame rates and resolutions. Thus, it's mandatory to define the profile, while designing a decoder, so that, they will support for that specific profile. In this paper, we are designing a decoder for the main profile, normally used in standard video streaming and broadcasting.

Input of the decoder is an encoded YUV video containing color spaced pixels, where Y represents luminance whereas U and V represents chrominance component. Human eyes are more sensitive to brightness than colors, for this reason chrominance data is considered for improving encoding efficiency.

In H.264 standard, video is encoded frame wise, where I-frames are encoded without any information of past and future frames. Encoding of P-frame requires information of previous frame whereas encoding of B-frame requires information of past frame as well as future frames also. Encoded video is stored in form of bit stream. File format of encoded video provides information about each frame's type. A H.264 input file format is shown in Figure 1.

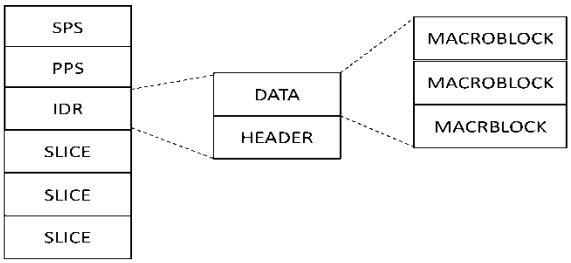


Figure 1. Structure of H.264 encoded file

SPS and PPS unit of input file (frame) contains information regarding decoding parameters and frame size. IDR unit of the input file is the first slice where frame is further divided into macro blocks. Each of the slice header having basic information of slice like slice identifier, number of macro blocks, quantization factor and frame configuration.

A Decoder gets compressed bit-stream from the encoded input file where entropy decoder decodes the input bit stream into a set of quantized coefficient. The residual image information can be obtained by using inverse quantization and inverse transformation unit. The combined information of residual data, pre-decoded data and prediction information is utilized in final decoded image.

In this paper, we mainly focus on getting higher resolution decoded image with a high frame rate. For this purpose we need to consider a design process, where we implement the parameterized buffer for temporary storage solution. A H.264 video decoder functional block diagram along with intra prediction unit is shown in Figure 2.

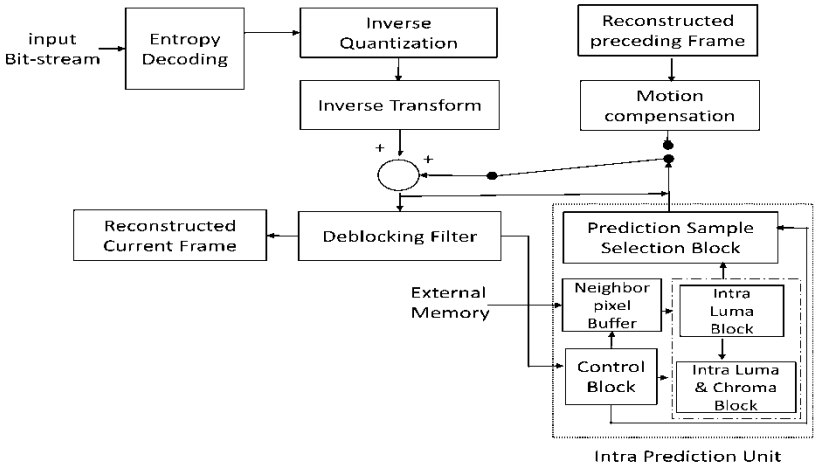


Figure 2. H.264 Decoder functional block diagram

## 4. IMPROVED H.264 DECODER WITH SIMULATION ARCHITECTURE

In this section we will discuss about the unique designing and methodology to get improved implementation of H.264 decoder. We are mainly concentrating on difficulties in performing to designing a module, which can decode our video frame efficiently. In total designing process, a use case has considered for the hardware module which includes requirement verification, code writing and iterative source optimization with single function or multiple function along with system level optimization. The simulated architecture of H.264 decoder is shown in Figure 3. The architecture shows the connectivity of external memory with shared bus and interface unit.

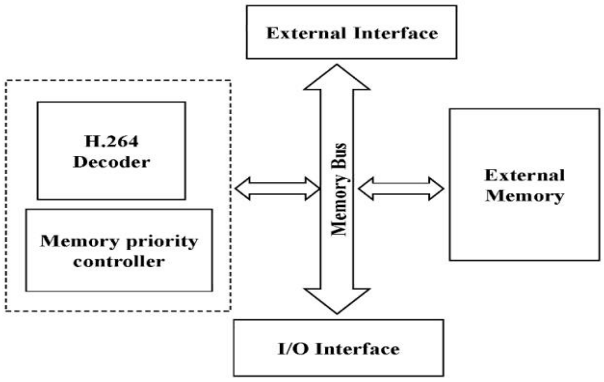


Figure 3. Simulated H.264 decoder architecture

### 4.1. Improved Synthesis Process

In this section we will start improving the application iteratively by improving the code. We will first improving the design for area minimizing by implementing maximally parallelize and pipelining architecture. Though, this architecture cannot be used in complex application, because applying global parallelization makes area cost too high. Due to this reason, we should more selective in optimizing area and optimizing performance in complex application.

We have written our code by the help of Xilinx ISE tool in Verilog hardware description language. It's a tool which gives environment to develop the design code, synthesize the code and simulate the developed design code.

#### 4.1.1 Building Individual Function

We start building every individual function in a fashion, which helps in getting performance on call. We have divided all single function into groups and performed several enhancement. We concentrated more on performance optimization rather than area optimization.

### 4.2. Memory Reusability with Pipelining Architecture

In order to make memory references efficient in reference software CPU implementation is a key factor. The hardware implementation of shared memory might be expensive, even in case of local BRAM block also, which may cause a performance bottleneck of the system. Thus, by applying splitted individual register or small local data array reusability can be an important factor in system performance. The decision of splitting registers or small local data array implementation depends on three factors. First, the functions which involved in array should be significant function individually. Second thing the functions which are involved in array should have data-parallelism and memory port limitation should not be there. The last thing is the implementation of BRAM. The reducing BRAM can be more effective than preventing the register use.

This strategy is applied in our design implementation. But, if the local buffer size is not completely partitioning feasible, in that case we create an additionally local buffers which can be reused efficiently. In our code, in each iteration we read 5-6 overlapped data items where only one data item is new. Therefore, by creating local buffer we read a new data in each iteration.

Till now, improvement is done for efficiency and memory reusability by low level function calling. Now, we can further improve our system for area constraints. A pipelining architecture helps a lot in improving performance as well as significant area optimization. We have pipelining based on the number of iterations to improve further performance of the system along with area optimization.

#### 4.2.1 Improving Through Cross Function Examination

We start developing function sequences any analyzed their importance in application. In descending order of significance, we can improve the leaf function sequence, which is more effective in enhancing important call sequences. Profiling data is updated and there after we find out the different sequences of function calls. These function call might be the improved leaf function or other several function including call stack. Each function latency is analyzed along with the number of calls on call stack. This data helps us in finding critical path in call stack.

#### 4.2.2luma and Chroma Parallel Data Flow Design

Figure 4 demonstrates the Luma and Chroma parallel data path unit, where buffers are utilized. Weight predictor estimates the weight for both Luma and Chroma data samples where data is stored in small buffers. Average sum of the predicted weight, buffer data and individual predicted weight is applied to multiplexer for finalizing the output.

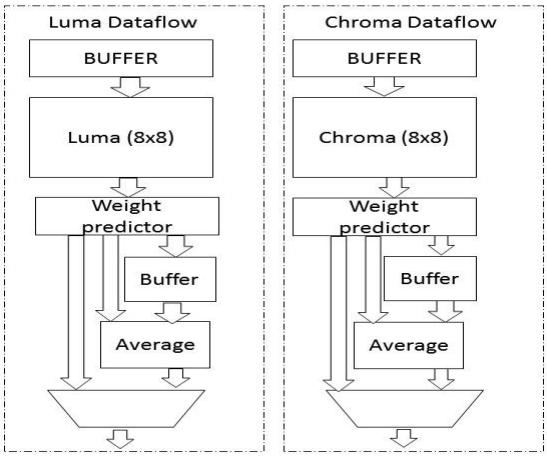


Figure 4. Luma-Chroma Data flow design

### 4.3. Looping Rearrangement and Function Inlining

There are some factors, which can stop the pipelining completely in loop. The control flow of the loop may degrade the performance by applying conditional check in each loop iteration. Due to these reason, we reorder every loop where optimization of both the latency is needed considering the pipelining suitability.

During single function optimization, it has been found in several cases, that there is a need of function inlining for both latency and area optimization. But it is not necessary that there is always a significant benefit of specialization would be there by function inlining. Due to duplication of resources, a tradeoff is there between function call overheads and increased area. By implementing inlining tool, we can achieve universally inline or we can prevent inlining of a function. In our case, we tried to get maximum benefit in each call site. However, in some cases it is beneficial only after inlining a subset of frequently used function call sites.

After identifying important positions of potential saving, we implemented function inlining. Using profile data and call site positions, it is observed that the significance of all call sites for candidate function. We open multiple alternative option of inlining directives which helps in finding latency saving and area cost, But we have finalized inline choices by inlining manually for the implemented function.

Buffer upgradation

As we discussed earlier, in improving of single function buffer role is very significant to execute parallelism algorithm. Initially, we start from local buffer insertion to improve single leaf function. Here, we are not considering the places, where to define the buffer for getting maximum benefit of parallelization during the call stack. We just evaluate the individual buffer to define in call stack. Buffer can affect parallelism of sub-functions at higher level of call stack, but at the same time it leads complexity in inter-block edges, which experiences overhead while copying form global edges. In this situation, a confliction comes in sharing of potential data, because multiple sub-functions reuses same buffer. We analyze the call stack of each local buffer and try to define it earliest places, where overhead is minimum and sub function gets maximum benefit during parallelism.

### 4.4. System Level Parallel Processing

Once call stack improvement is done, we improved every single function along with the important sequence of function in call stack. Now instead of optimizing less significant function, we start optimizing data buffering and data transportation across different portions of decoder, which is not directly connected with function call stack.

After data optimization we find out top-level dependencies in profiling data, which helps in finding data dependency between buffers of top level function in call stack.

#### 4.4.1 Buffer Supplement and Task Level Parallelism

As we discussed earlier, the input data stream is complex, so after call stack and local buffer insertion, the external data evaluation is critical factor in performance evaluation. That's why we want to develop a core system, which should be independent from target resolution, where some part of memory can be accessed through external bus for data transfer rather than using local memories. Moreover, implemented local buffer are well optimized for parallelism in local function. So, there is no need of having facilitate reusing throughout call state region.

By using the statistics of profile information, critical function is determined and clustering is developed for function call graph. Additionally, with the help of this clustering, a local data array is created for system-level, which is comparatively large and used by directly or indirectly. Directly use of this larger data can be possible when, there is no need of substantial parallelism whereas indirectly use of data can be possible by copying data into next level of buffer. Copying of data into next level buffer may experiences overhead, but it protects from latency produce by localization and reusability. Later we find out those sub function, which implements local buffer based on the observation of their buffering necessity and processing time.

In our designing process, we have optimized first individual function or call sequences and ignored task-level parallelism. Later we considered, task-level parallelism by implementing two methods which are (a) Buffer duplication and (b) interface duplication. We use buffer duplication method in case of using input data by multiple function whereas interface duplication method in case of partitioning data into two or more group accessed individually by function.

### 4.5. Runtime Memory Allocation

In order to minimize the memory allocation problem H.264 uses a dynamic memory allocation. In our use-case also, we have used dynamic memory allocation for internal buffers which will convert in parameterized static allocations. The usage of dynamic memory allocation depends on the size of the input file resolution. It's not good to design a decoder which will support only one maximum resolution. To overcome this problem, we have to redesign our model for each target case resolution which increases optimization challenges. A large amount of input buffer converts into top level memory interfaces, which will use for memory banks. It might need of transferring data using memory interface. That's why we optimizing kernel's without considering the video resolution.

## 5. RESULT AND ANALYSIS

We used Xilinx VCU1525 development kit as a synthesizable software for optimization of each intermediate stage of designing. We performed on-board verification using Omnitek Zynq 7000. ARM CPU is utilized for data travelling while Zynq acts as a standalone FPGA.

Vivado 2015.2 tool is applied to determine the occupied area and operating frequency for each design process. In order to measure performance of input video files and the corresponding board-level application, simulation has performed in H.264 video file. H.264 video file follows a repeating pattern like one frame is I-frame, next is P-frame and then B-frames and again P-frame. The latency of decoded frame depends on data for almost all frame type. We try to find out the worst case latency (maximum latency) of each frame type in each cycle. A weighted average is performed twice for B-frames compare to I or P frames. Later, the average latency of each cycle is multiplied with obtain frequency to check the average latency got per frame. Moreover, we have used Intel core i5-2310 CPU (2.9GHz) to complete the required modification.

Because, our core design is independent of resolution, we tested our design with multiple resolution like QCIF 144p and 480p of input files. We estimated system performance for different resolutions. We also covered the average latency for per micro block of the system. We have compared our performance with different pre-developed H.264 implementation.

### 5.1. Performance Evaluation and Analysis

We synthesized our design and evaluate performance while treating with QCIF input videos. At each optimization stage, occupied area is presented with performance for corresponding synthesized implementation. By implementing single function improvement, a little area gets optimized but it also improves the CPU performance, which indicates reusing function is well organized. Though, optimization of cross-function with the help of local buffer making the pipelining more effective rises performance excellently. The area depends primarily on FF, DSP and LUT units and SRAMs are utilized mainly in system-level buffer but rarely used in local buffers. Optimization in system level configuration makes the performance low in CPU level depicts a deviation in our desire result. In final step of parallelization, we achieve good throughput.

However, our simulation process do not design a memory bandwidth at system level. H.264 system requires a lot of computation process, so, it should not be a memory bounded application. In experiment, Xilinx kintex-7 FPGAs is demonstrated to get a good area optimization with maximum frequency.

Table 1. The number of luma and chroma intra prediction unit comparison is depicted

| Design | LUT | FF |
|---|---|---|
| luma [14] | 545 | 127 |
| luma[15] | 212 | 37 |
| luma [proposed] | 83 | 284 |
| chroma[14] | 216 | 59 |
| chroma[15] | 163 | 33 |
| chroma [proposed] | 284 | 105 |

Table 2. Comparison of H.264 decoders with other H.264 decoders and HEVC decoders are shown

| | Our proposed work | ESSCIRC 14[ 16] | ISSCC 13[17 ] | ASSCC 13[18] | ISSCC 12[19 ] | VLSI 10 [ 20] | ISSCC 10[ 21] | Work done[ 22] |
|---|---|---|---|---|---|---|---|---|
| Video-format | H.264 | HEVC + Multi-format | HEVC WD4 | HEVC | H.264 | H.264 | H.264 | HEVC |
| On-Chip SRAM | 102.5 KB | 154KB | 124KB | 10.2KB | 79.9KB | 59.6KB | 9.0KB | 396KB |
| Logic gates technology | 190k 28nm/ 0.9v | 3454k 28nm/ 0.9v | 715k 40nm/ 0.9v | 446k 90nm/ 01.0v | 1338k 65nm/1.2 v | 662k 90nm/ 1.0v | 414k 90nm/ 1.09v | 2887k 40nm/ 1.0v |
| Clock rate | 355 MHz | 350 MHz | 200 MHz | 224 MHz | 340 MHz | 175 MHz | 210 MHz | 300 MHz |
| DRAM config | DDR3L | 32bLPDD R3 | 32bDDR3 | n/a | 64bDDR2 | 64bDDR1 | n/a | 64bDDR3 |

## 6. CONCLUSION

This paper shows an efficient designing of an h.264 decoder with its intra prediction unit (luma and chroma). For this process, we have used Xilinx VCU1525 development kit with on board verification using ARM CPU. By using runtime memory allocation, individual function improvement and cross function verification, we have achieved complete improvement in designing. This improvement provides good area optimization with maximum frequency of 355 MHz. We have used different block types for intra prediction unit to reduce the area cost of the decoder. By applying various improvement, we have achieved a great throughput.

## REFERENCES

[1] L. V. Agostini, A. Azevedo, W. Staehler, V. Rosa, B. Zatt, A. C. Pinto, R. E. C. Porto, S. Bampi, A. Susin, "Design and FPGA Prototyping of a H.264/AVC Main Profile Decoder for HDTV", *Journal of the Brazilian Computer Society*, vol. 12, pp. 25-36, 2007.

[2] D. Indoonundon, T. P. Fowdur, K. M. S Soyjaudah, "A Concealment Aware UEP Scheme for H.264 using RS Codes", *Indonesian Journal of Electrical Engineering and Computer Science(IJEECS)* Vol. 6, No. 3, June 2017, pp. 671 ~ 681 DOI: 10.11591/ijeecs.v6.i3.pp671-681.

[3] Chuan-Yung Tsai, Tung-Chien Chen, To-Wei Chen and Liang-Gee Chen, "Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder," *48th Midwest Symposium on Circuits and Systems, 2005.*, Covington, KY, 2005, pp. 1199-1202 Vol. 2. doi: 10.1109/MWSCAS.2005.1594322

[4] E. Kalali, Y. Adibelli, I. Hamzaoglu, "A Reconfigurable HEVC Sub Pixel Interpolation Hardware", *IEEE Int. Conference on Consumer Electronics - Berlin, Sept*. 2013.

[5] E. Kalali, I. Hamzaoglu, "A low energy HEVC sub-pixel interpolation hardware," *IEEE Int. Conference on Image Processing*, pp. 1218-1222, Oct. 2014.
[6] Mengmeng Zhang, Jianfeng Qu, Huihui Bai, "Fast Intra Prediction Mode Decision Algorithm for HEVC", *TELKOMNIKA Indonesian Journal of Electrical Engineering*, Vol. 11, No. 10, October 2013, pp. 5703 ~ 5710 ISSN: 2302-4046.
[7] C. M. Diniz, M. Shafique, S. Bampi, J. Henkel, "A Reconfigurable Hardware Architecture for Fractional Pixel Interpolation in High Efficiency Video Coding," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 238-251, Feb. 2015.
[8] J. Andrade *et al*., "Design Space Exploration of LDPC Decoders using High-Level Synthesis," in *IEEE Access*, vol. PP, no. 99, pp. 1-1. doi: 10.1109/ACCESS.2017.2727221
[9] S. Baldev, K. Shukla, S. Gogoi, P. Rathore and R. Peesapati, "Design and Implementation of Efficient Streaming Deblocking and SAO Filter for HEVC Decoder," in *IEEE Transactions on Consumer Electronics*, vol. PP, no. 99, pp. 1-1.
doi: 10.1109/TCE.2018.2812518
[10] Zhao Han, M.R. Anjum, "A New Low-Costing QC-LDPC Decoder for FPGA", *TELKOMNIKA Indonesian Journal of Electrical Engineering* Vol. 12, No. 11, November 2014, pp. 7721 ~ 7727 DOI: 10.11591/telkomnika.v12i11.6512.
[11] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi and L. Hanzo, "Hardware-Efficient Node Processing Unit Architectures for Flexible LDPC Decoder Implementations," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 99, pp. 1-1. doi: 10.1109/TCSII.2018.2807362
[12] T. Mallikarachchi, D. S. Talagala, H. K. Arachchi and A. Fernando, "Decoding-Complexity-Aware HEVC Encoding Using a Complexity-Rate-Distortion Model," in *IEEE Transactions on Consumer Electronics*, vol. PP, no. 99, pp. 1-1. doi: 10.1109/TCE.2018.2810479
[13] H. Kim, J. Ko and S. Park, "An Efficient Architecture of In-Loop Filters for Multicore Scalable HEVC Hardware Decoders," in *IEEE Transactions on Multimedia*, vol. PP, no. 99, pp. 1-1 doi: 10.1109/TMM.2017.275950
[14] F. Palumbo et al., "Runtime energy versus quality tuning in motion compensation filters for HEVC," in *Proc. of the PDeS Conf*., 2016.
[15] C. Sau et al. "Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multi-frequency Approximate Computing IEEE Embedded Systems Letters" 2017.
[16] C.-C. Ju et al., "A 0.2 nJ/pixel 4K 60 fps Main-10 HEVC decoder with multi-format capabilities for UHD-TV applications," in *Proc. Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 2014, pp. 195–198.
[17] C.-T. Huang, M. Tikekar, C. Juvekar, V. Sze, and A. Chandrakasan, "A 249 Mpixel/s HEVC video-decoder chip for quad full HD applica-tions," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2013, pp. 162–164.
[18] C.-H. Tsai, H.-T. Wang, C.-L. Liu, Y. Li, and C.-Y. Lee, "A 446.6K-gates 0.55–1.2V H.265/HEVC decoder for next generation video appli-cations," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2013, pp. 305–308.
[19] D. Zhou, J. Zhou, J. Zhu, P. Liu, and S. Goto, "A 2 Gpixel/s H.264/AVC HP/MVC video decoder chip for Super Hi-Vision and 3DTV/FTV applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2012, pp. 224–225.
[20] D. Zhou et al., "A 530 Mpixels/s 4096 × 2160@60 fps H.264/AVC high profile video decoder chip," in *Proc. Symp. VLSI Circuits (VLSI)*, Honolulu, HI, USA, 2010, pp. 171–172.
[21] T.-D. Chuang et al., "A 59.5 mW scalable/multi-view video decoder chip for quad/3D full HDTV and video streaming applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2010, pp. 330–331.
[22] D. Zhou *et al*., "An 8K H.265/HEVC Video Decoder Chip With a New System Pipeline Design," in *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 113-126, Jan. 2017. doi: 10.1109/JSSC.2016.2616362.