Cost-Efficient $130nm$ TSMC Forward Transform and Quantization for H.264/AVC Encoders

Xuan-Tu Tran, Van-Huan Tran
SIS Laboratory, University of Engineering and Technology, VNU Hanoi.
144 Xuan Thuy road, Hanoi 10000, Vietnam. Email: {tutx, huantv}@vnu.edu.vn

***Abstract*—In this paper, we present a low cost Forward Transform and Quantization (FTQ) implementation for H.264/AVC encoders in mobile applications. To reduce the hardware implementation overhead, the proposed design uses only one unified architecture of 1-D transform engine to perform all required transform processes, including discrete cosine transform and Walsh Hadamard transform. This design also enables to share the common parts among multipliers that have the same multiplicands. The performance of the design is taken into consideration and improved by using a fast architecture of the multiplier in the quantizer, the most critical component in the design. Experimental results show that our architecture can completely finish transform and quantization processes for a 4:2:0 macroblock in 228 clock cycles and the achieved throughput is $445Msamples/s$ at $250MHz$ operating frequency while the area overhead is very small, $147755\mu m^2$ (approximate $15KGates$), with the $130nm$ TSMC CMOS technology.**

I. Introduction

The H.264 Advanced Video Coding (H.264/AVC) [1] is known as the lastest and most efficient video compression standard providing better video quality at a lower bit-rate than previous standards. The standard is recommended by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). It contains a rich set of video coding tools to support a variety of applications ranging from mobile services and video conferencing, digital broadcast, to IPTV, HDTV and digital storage media. To achieve high compression ratio, H.264/AVC has adopted several advances in coding technology but it makes the hardware implementation becomes more complex and costly. Many designers have paid much attention to the complexity reduction of motion estimation and/or entropy coding parts. However, the implementation of forward transforms and quantization (FTQ) part should also be considered for minimizing the system hardware overhead.

By specifying a set of integer transforms for small block-sizes, which are integer Discrete Cosine Transform (DCT) and Walsh Hadamard transform, it has really reduced the computational complexity as important as blocking artifacts [2]. In addition, thanks to the advanced procedures prensented in [3], these transforms can be easily realized with some required shifting and adding operations. In fact, in H.264/AVC standard, the size of transforms is variable depending on the profile used in the encoder, where 4×4 block-size and 2×2 block-size transforms are primitive components. Larger transforms, which used in adaptive block-size transforms (ABTs), are more suitable for High-Definition (HD) video. In this paper, only the 4×4 block-size transforms will be discussed but the same principle can be applied for the other sizes of transforms, or even larger transforms can be converted to the 4×4 block-size transforms [4].

Previous works have already been successes in hardware implementation of transforms and quantization. Chih-Peng Fan and Yu-Lin Cheng [5] proposed a design with a high through-put and low latency architecture using Canonical Signed Digit (CSD) multiplier for shared quantization/inverse-quantization. In [6], Yu-Ting Kuo presented an area-efficient architecture using direct 2-D transform method. Whereas, in [7], [8], [9] is proposed a multi-transform architecture used for variable adaptive block-size transforms. Generally, these works used two separate 1-D transforms in cascading to carry out a 2-D transform or to implement a direct 2-D transform. Obviously, the advantage of these methods is that they can achieve a high throughput in transforms. However, the disadvantage is that their hardware implementation area could not be significantly reduced, even in [9] the quantization parts are intently integrated into transform steps. In addition, the fact is that the bottleneck of encoders mostly comes from motion estimation and/or entropy coding modules rather than transforms and quantization. Optimizing the design for throughput is therefore less important than other objectives such as overal performance providing real-time processing capacity or hardware implementation area of the whole system.

In this work, we propose a transform architecture using only one unified 1-D transform module in order to trade-off between throughput and area overhead. With some improvements in control part, this architecture is able to perform integer DCT-based transforms as well as Hadamard transforms. In addition, to improve the system performance and get more area-efficiency, we also present a particular architecture of multiplier in the quantizer. In where, a shared module (called pre-multiplier) is intently used for multipliers have the same multiplicands. The FTQ architecture is designed to be used as part of a low power H.264/AVC encoder for mobile applications. The proposed architecture is then implemented using the $130nm$ TSMC CMOS technologies. It can completly finish the transform and quantization processes for a 4:2:0 macroblock in 228 clock cycles and therefore can achieve a throughput of $445Msamples/s$ at $250MHz$ operating frequency while the area overhead is very small, approximate $15KGates$.

The remaining part of this paper is organized as follows: Section II briefly recalls some backgrounds of transform and quantization algorithms in H.264/AVC coding. The proposed architecture for a forward transform and quantization will be presented in Section III. Experimental results will be presented and discussed in Section IV. Finally, conclusions and perspectives will be given in Section V.

II. Forward Transform and Quantization Algorithms in H.264/AVC

In H.264/AVC video coding standard, the residual frame of the prediction, which is the difference of the original frame and the predicted frame, is partitioned into fixed-size of macroblocks. As usually, a macroblock is composed of 16×16 luminance (Y) samples, 8×8 chroma blue (C_b) samples, and 8×8 chroma red (C_r) samples in the case of 4:2:0 chroma subsampling format. At a smaller level, macroblocks are subdivided into blocks of 4×4 samples for encoding. Each macroblock has its own information on quantization parameters (QPs), coded type (Intra mode or Inter mode) and prediction mode. The transform and quantization flow for those blocks can be illustrated in Figure 1.

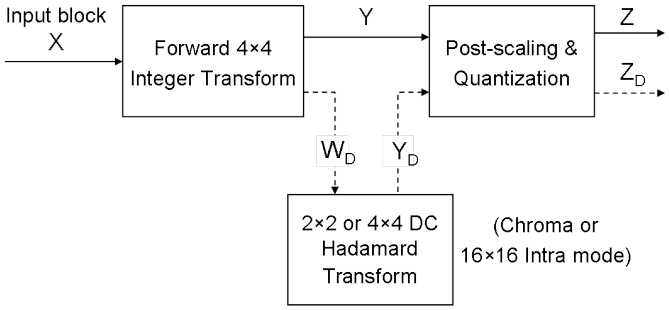

Fig. 1. Transform and quantization flow diagram.

According to this flow, the input block X is first transformed using integer DCT-base method. The transformed coefficients are then post-scaled and quantized. In the 16×16 Intra-prediction mode, DC coefficients of all transformed residual blocks are grouped into an array of 4×4 before being sent to Hadamard transform. Details of these processes are described in mathematical models in the following.

A. 4×4 *forward transforms*

- **Integer DCT-based transform**

The integer DCT-based transform, which applied to a residual 4×4 blocks (denoted by matrix X), is defined in H.264/AVC as the following:

$$Y = CXC^T = C(CX)^T \tag{1}$$

Where,

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & -1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

- **Hadamard transform**

The Hadamard transform which applied to a 4×4 luminance DC block (denoted by matrix W_D) in 16×16 Intra-prediction mode, is defined as the following:

$$Y_D = HW_DH^T = H(HW_D)^T \tag{2}$$

Where,

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} /2$$

In general, both integer DCT-based transform and Hadamard transform are normally formed by two duplicated *cores* of 1-D transform, where the *core* is a matrix multiplication either C^T or H^T. The 2-D transforms are carried out by applying the input block to the *core*; the immediate results are re-arranged by transposing operations and then re-applied to the *core*. Obviously, the specification of the matrixes C and H in which only coefficients of ± 1 and/or ± 2 are available. These transformations are multiplication-less and purely require a few of *add* and *logical shift* operations. On the other hand, the dynamic range of data is also estimated to reduce the overhead in computating processes. With 8-bit precision of the pixel data, the dynamic range of outcomes of integer DCT-based transform is 16-bit.

In here, we have already modified the matrix H by scaling of 1/2 to preserve the arithmetic operations of Hadamard transform in 16-bit precision as of integer DCT-based transform. Then, in the quantization process of the DC blocks, the result will be rescaled of 2. By this way, all 4×4 forward transforms are completely handled in 16-bit precision.

Figure 2 shows a hybrid and fast 1-D transform diagram for processing 4 samples. The diagram is in the shape of butterfly diagram and is used for two types of transform. There are some multiplexers to select the shift factors (or scaled factors) in computations of each transform type. This diagram is a great inspiration to design the architecture of transform module.

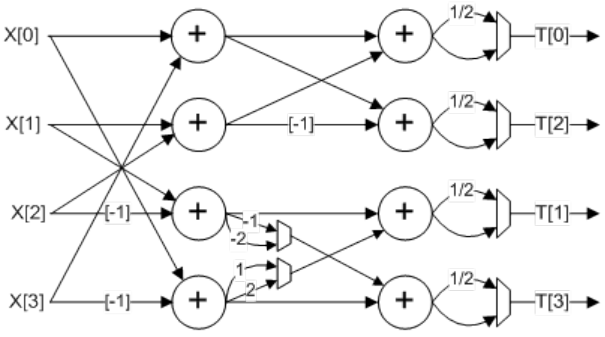

Fig. 2. Hybrid 1-D transform architecture for integer DCT-based and Hadamard transforms

B. Quantization

H.264/AVC standard defines a set of 52 values of quantization steps ($Qstep$). These values are indexed by QP and

TABLE I
MULTIPLICATION FACTOR (MF)

QP%6	Positions (0,0), (2,0), (2,2), (0,2)	Positions (1,1), (1,3), (3,1), (3,3)	Other positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

to be determined in the range of 0 to 51. As introduced above, the values of quantization parameters are associated with macroblocks (also blocks within a macroblock). Thanks to the wide range of QP, an encoder is able to accurately and flexibly control the trade-off between its bit-rate and quality [10].

Basically, forward quantization can be defined as follows:

$$Z_{ij} = round\left(\frac{Y_{ij}}{Qstep}\right), 0 \leq i,j \leq 3 \quad (3)$$

To avoid division operations, this equation can be represented in another way [3]:

$$Z_{ij} = round\left(W_{ij} \times \frac{MF}{2^{qbits}}\right), 0 \leq i,j \leq 3 \quad (4)$$

In consequence, the quantization can be computed as follows:

$$\begin{aligned} |Z_{ij}| &= \big(|W_{ij}| \times MF + f\big) \gg qbits, \\ sign\big(Z_{ij}\big) &= sign\big(W_{ij}\big), 0 \leq i,j \leq 3 \end{aligned} \quad (5)$$

In where, $qbits = 15 + floor(QP/6)$, MF is a multiplication factors matrix (see Table I) and f is an additional factor, $f = 2^{qbits}/3$ if the block is coded in Intra mode, and $f = 2^{qbits}/6$ if the block is coded in Inter mode.

Especially, the quantization for a DC block is implemented as follows (it has already rescaled by 2 due to scaling by 1/2 in transform):

$$\begin{aligned} |Z_{D(ij)}| &= \big(|Y_{D(ij)}| \times MF_{00} + 2f\big) \gg qbits, \\ sign\big(Z_{D(ij)}\big) &= sign\big(Y_{D(ij)}\big), 0 \leq i,j \leq 3 \end{aligned} \quad (6)$$

Where MF_{00} is the multiplication factor at position $(0,0)$.

The innovation of quantization in H.264/AVC is the definition of $Qstep$. In where, $Qstep$ is non-uniform (or non-linear according to QP) and doubled in size if QP increases by 6. Therefore, whenever QP is changed by the encoder, the values of MF factor matrix is also changed as consequence, but it absolutely depends on the value of $QP\%6$ (as shown in Table I). Besides, it does not require a lot of memory elements to store MF factors, only 18 values for full range of QP.

Similar to the transform part, the quantization has also been simplified to obtain low-complexity in a manner of avoiding division and floating point operations.

III. THE PROPOSED ARCHITECTURE

In this section, we present a hybrid architecture for DCT and Hadamard forward transforms and quantization of 4×4 blocks. While the design of transform is only intended to area-efficiency by using 1-D transform module for all transformations of 4×4 blocks, the design of quantizer is carefully taken into account in order to improve on both performance and implementation area. This design is capable to process 4 samples at the same time. The details will be described as follows:

A. Transform module

By sharing the only 1-D transform module, the second 1-D transform process could not be started until the first 1-D transform process had finished on the entire block. Therefore, it is necessary to have a memory buffer for storing and transposing the temporary data. Figure 3 shows the architecture of forward transform module with the sample-width of the datapath.

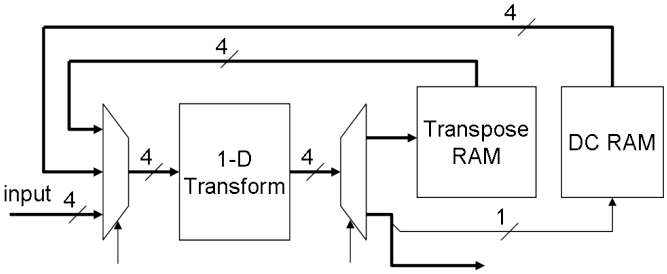

Fig. 3. Architecture of forward transform.

The architecture is simply composed of three main components: 1-D Transform module, Transpose RAM module, and DC RAM module. In addition, there are other components included in this architecture such as multiplexer and demultiplexer for arbitrating the dataflow. The input data and output data of the transform module are 4 samples, equivalent to 64-bit (4×16-bit). To have better view, some detail control signals have been hidden.

The activity of the module can be easily realized through the list of all states (corresponding to the dataflow):

- *State_1: Input ⇒ 1-D Transform ⇒ Transpose RAM*
- *State_2: Transpose RAM ⇒ 1-D Transform ⇒ Output*
- *State_3: DC RAM ⇒ 1-D Transform ⇒ Transpose RAM*

These states have a length of 4 clock cycles. By controlling the sequence of these states, a general block will be executed in the order of two states {*state_1*; *state_2*}, while a DC block will be executed in the order of two other states {*state_3*; *state_2*}.

- **1-D transform**

The 1-D transform module is a hybrid transform as illustrated in Figure 2. All multiplexers in this module are controlled by a selection signal which configures the module's activities as integer DCT-based transformation or Hadamard transformation. The responsibility of this module is one clock and the throughput is therefore $4samples/clock$. A higher throughput

can be easily obtained by using several 1-D transforms in parallel.

- **Transpose RAM and DC RAM**

The purpose of Transpose RAM is to store and transpose data for transformation processes. DC RAM is used to store luminance DC coefficients of a transformed macroblock. Basically, Transpose RAM and DC RAM are matrices of 4×4 16-bit registers (as shown in Figure 4). The input and output of Transpose RAM are 4 samples width for reading/writing accesses to a row/column data. Whereas the input of DC RAM is only one sample width for writing one DC coefficient at a time and the output is 4 samples width as Transpose RAM's.

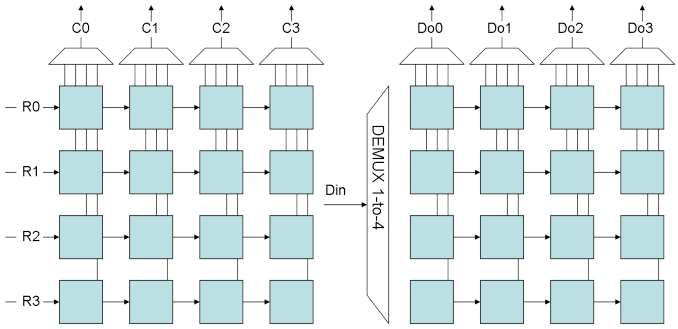

Fig. 4. Transpose RAM (*left*) and DC RAM (*right*).

The writing operations of Transpose RAM are enabled whenever valid data are ready at the output of 1-D transform module. These occur in 4 succesive clock cycles of the first 1-D transform process. The reading operations occur in next 4 clock cycles to get out the column-wise data for the second 1-D transform process. The registers in a row of Transpose RAM are connected in series. Thus, the data stored in a register will be automatically shifted into the back register in next clock. By this means, Transpose RAM will be filled up with new data in 4 clock cycles of the writing operations.

DC RAM is a bit different from Transpose RAM in their structures where the registers are independent from each other. This buffer is useful and will be active in 16×16 Intra prediction mode. In that case, the DC coefficients of any transformed luminance blocks are extracted and written into DC RAM. Therefore, the writing operations of DC RAM take place in only one clock cycle at the time where the earliest data is valid. The reading operations are enabled in 4 clock cycles when the last block of a luminance macroblock is completely transformed. Besides, the reading/writing address signals of DC RAM are directly controlled by FTQ controller.

Comparing with the cascading architecture using two separate 1-D transform modules [11], our architecture is required a bit challenging in designing the controller module but it has absolutely saved the hardware resource by the total cost of an 1-D transform module.

B. Quantizer module

The quantizer can be easily realized from equations 5 and 6, as depicted in Figure 5. It consists of four quantization cores and some common parts: MF_ROM module, DIVIDER_BY_6 module, and F_CALC module. These common modules are shared to the 4 quantization cores. Actually, DIVIDER_BY_6 module is possible to share with de-quantizer module.

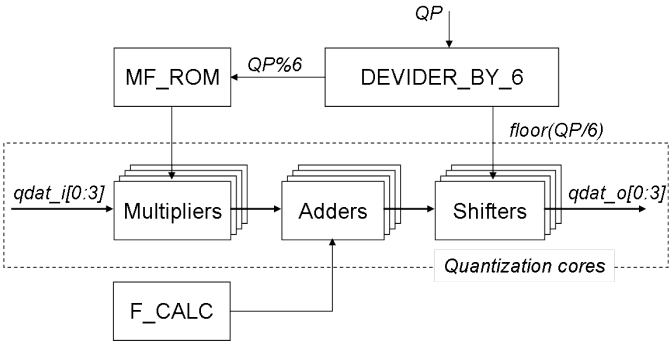

Fig. 5. Quantizer architecture.

DIVIDER_BY_6 module is a combinational block to calculate the value of $QP\%6$ and *floor* $(QP/6)$ as well. In some related works, it was designed as common look-up-tables (LUTs), such as [5]. This design may take lots of memory utilization because we have up to 52 values of QP. MF_ROM module is a ROM block for storing 18 constant values of MF factors. Accessing to a batch of MF factors is addressed by $QP\%6$ signal. F_CALC module is a combinational block to calculate the additional factor f based on the coded macroblock type (either Intra mode or Inter mode) and the block type (residual block or DC block).

Regarding to the multiplier design, when the size of multipliers are large (15-bit of $qdat\_i$ and 14-bit of MF), it can mostly impact to the performance of the quantizer as a result of large latency. For this reason, we have deeply investigated the design of multiplier, which will be presented in next paragraph to minimize the quantization latency.

- **A fast and highly shared multiplier**

The fast multiplier that we proposed is a conditional multiplier. The idea is to build a basic element (called pre-multiplier) which is multiplier of MF factor with all possible 3-bit numbers (as shown in Figure 6), where $A_i = i * MF$, with $0 \leq i \leq 7$. In fact, we do not need to carry out A0, A1, A2, and A4 on this module when these signals can be directly driven from MF signal.

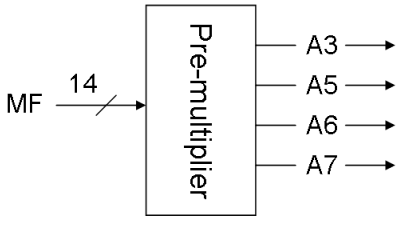

Fig. 6. The pre-multiplier element.

Consequently, the 15-bit multiplier using the pre-multiplier element is explored as Figure 7. Each group of 3-bit vector

(so it has 5 groups) is multiplied with the multiplicand W by controlling a multiplexer to select the equivalent result from the pre-multiplier element. There are some registers are inserted at the adders' outputs to cut down the combinational paths of the multiplier. By this way, we have improved the performance of the multiplier.

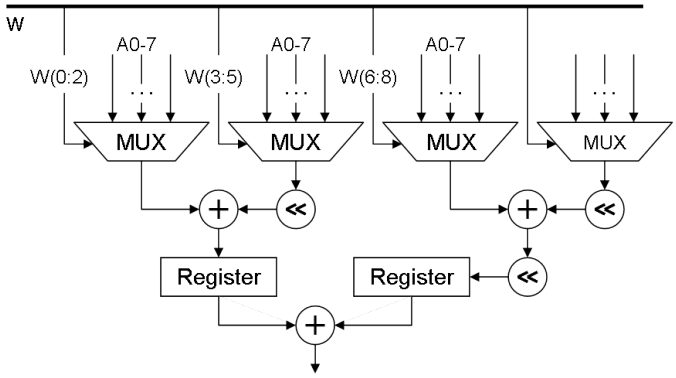

Fig. 7. Multiplier for quantizer using the pre-multiplier element.

Obviously, the pre-multiplier can be shared among 5 groups of 3-bit multipliers. Since the pre-multiplier takes 4 adder blocks, we save $(4 \times 4 = 16)$ adder blocks. In addition, from the MF factors table (Table I), $qdat\_i[0]$ and $qdat\_i[2]$ have the same MF factor, $qdat\_i[1]$ and $qdat\_i[3]$ have the same MF factor, therefore the pre-multiplier is also possible to share between two quantization multipliers which have the same MF factor. In consequence, our quantizer can be totally saved up to $2 \times (16 \times 2 + 4) = 72$ adder blocks. Certainly, several multiplexers will be required to replace these adders but the hardware utilization is significantly reduced.

C. Pipeline operation

To achieve better performance, the proposed architecture is controlled to operate in pipeline mode. Figure 8 shows the timeline of the whole coding process.

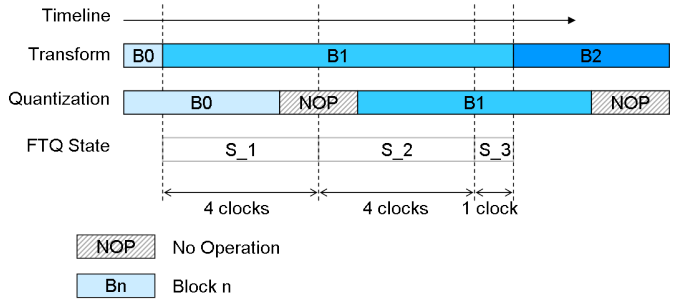

Fig. 8. Three states of pipeline operation.

The pipeline has three states as identified by S_1, S_2, and S_3. In where, S_1 (4 clock cycles) launches the first stage of 1-D transform on block B1 while previous block B0 is still quantized to the end. S_2 (4 clock cycles) launches the second stage of 1-D transform while starts quantizing block B1. S_3 (1 clock cycle) prepares loading new block B2 into transform module while block B1 is still quantized, valid data at the output are ready. Therefore, it normally takes 12 clock cycles to complete transform coding for a block.

Summary, by the pipelined schedule, our design will take 9 clock cycles on average to process a block and this is equivalent to 228 clock cycles to complete the transform and quantization processes for all 4×4 blocks within a 4:2:0 macroblock.

IV. Verification and Implementation

The architecture was modeled using VHDL language and simulated on Synopsys VCS tool. To verify the functionality of the design, we developed a simple simulation environment as described in Figure 9. Then, this environment is also used to verify the design before and after the layout implementation.

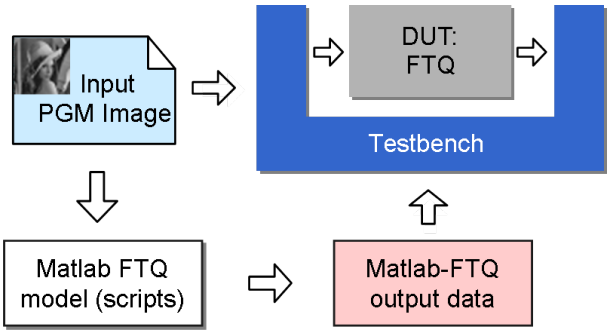

Fig. 9. Verification model for the design.

In this environment, we developed a software model of FTQ architecture on Matlab, which is used for testing purpose only. The input data used in this simulation is a PGM (portable graymap format) image. This image is provided to both Matlab model and hardware model. Then, the outputs of the both hardware and Matlab-based model will be compared to each other by using the developed testbench.

The proposed design was first prototyped on a Virtex-II Xilinx FPGA (XC2V1500-6) and then implemented using the $130nm$ CMOS technology from TSMC. The achieved operating frequency is $115MHz$ with the FPGA implementation, and $250MHz$ with the $130nm$ TSMC CMOS technology. In both cases, the design takes 228 clock cycles to complete the transform and quantization processes for the entire 4:2:0 macroblock. The transformation and quantization throughput is estimated of $204Msamples/s$ and $445Msamples/s$, respectively. The implementation area of the proposed design is $147755\mu m^2$ with the $130nm$ TSMC CMOS technology.

Table II shows the implementation report of different designs on hardware overhead, operating frequency, and throughput. From this report, our design has a much higher throughput and a lower hardware overhead compared to the design presented in [9], while both the designs have the same data width (4-bit). The designs presented in [5], [12], [8] have higher throughputs than our design but the data width of these designs are 4 times (even 8 times) larger than the data width of our design (16-bit, 16-bit, and 32-bit, respectively).

TABLE II
IMPLEMENTATION REPORT ON HARDWARE OVERHEAD, OPERATING FREQUENCY, AND THROUGHPUT BETWEEN DIFFERENT DESIGNS

Design name	**Technology**	**Data width** (*bit*)	**Operating freq.** (*MHz*)	**HW overhead** (*gates*)	**Throughput** (*Msamples/s*)
Pastuszak [8]	TOWER $0.18\mu m$	32	77	162,122	2,464
Chih-Peng Fan [5] (using CSD multipliers)	Xilinx XC2V1500	16	99.15	135,306	1,586
Kordasiewicz [12] (area-optimized)	TSMC $0.35\mu m$	16	68	51.619	644
Heng-Yao Lin [9]	TSMC $0.35\mu m$	4	32	30,785	273
Tasdizen [13]	ASIC $0.18\mu m$	1	210	23,162	21.5
This work (FPGA)	Xilinx XC2V1500	4	115	24,052	204
This work (ASIC)	TSMC $0.13\mu m$	4	250	15,033	445

Certainly, these designs therefore occupy much more hardware implementation areas than our design. In particular, the design presented in [13] with 1-bit data width but the hardware overhead is higher than ours while the throughput is much lower than our proposal. With the achieved throughput, our design totally responses to the need of H.264/AVC encoders/decoders while it is very suitable for efficient hardware implementation.

V. CONCLUSION

We presented in this paper a cost-efficient and high-performance forward transform and quantization hardware implementation for real-time H.264/AVC encoders in mobile applications. To minimize the hardware overhead, the proposed design used only one unified architecture of 1-D transform engine to perform all required transform processes, including discrete cosin transform and Walsh Hadamard transform. In addition, the detail architecture is also investigated carefully to optimize as much as possible both area cost and performance. The proposed architecture is then implemented on both FPGA and ASIC technologies. It is experimentally verified to work at $115MHz$ on a Xilinx Virtex II FPGA and to work at $250MHz$ on a TSMC $130nm$ CMOS implementation. The area overhead of this design is very small, $147755\mu m^2$ (approximate $15KGates$) with ASIC implementation. In addition, the design is able to complete transform and quantization processes for a macroblock in 228 clock cycles. Consequently, the achieved throughput is $204Msamples/s$ and $445Msamples/s$ for FPGA and ASIC implementations, respectively, while the data width is 4-bit. The proposed FTQ architecture is therefore proved to achieve a high performance with a lower area cost than the previous works.

ACKNOWLEDGMENT

This work is supported by Vietnam National University, Hanoi (VNU) through research project No. QGĐA.10.02 (VENGME). The authors would like to thank Synopsys' experts for their technical supports, TRIG-B project for travel grant.

REFERENCES

[1] ITU-T Recommendation and International Standard of Joint Video Specification. *ITU-T Rec. H.264/ISO/IEC 14496-10 AVC*, March 2005.

[2] Detlev Marpe, Thomas Wiegand, and Gary J. Sullivan. The H.264/MPEG4 Advanced Video Coding Standard and its Applications. *IEEE Communications Magazinel*, pages 134–143, August 2006.

[3] H.S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-Complexity Transform and Quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 598–603, 2003.

[4] Joo-Kyon Lee and Ki-Dong Chung. DCT Block Conversion for H.264/AVC Video Transcoding. In *Euro-Par 2005 Parallel Processing*, pages 919–927, Lisbon, Portugal, September 2005.

[5] Chil-Peng Fan and Yu-Lin Cheng. FPGA Implementations of Low Latency and High Throughput 4x4 Block Texture Coding Processor for H.264/AVC. *Journal of the Chinese Institute of Engineers*, 32(1):33–44, 2009.

[6] Yu-Ting Kuo, Tay-Jyi Lin, Chih-Wei Liu, and Chein-Wei Jen. Architecture for Area-Efficient 2-D Transform in H.264/AVC. In *Proceedings of the 2005 IEEE International Conference on Multimedia and Expo*, Amsterdam, Netherlands, July 2005.

[7] Javier D. Bruguera and Roberto R. Osorio. A Unified Architecture for H.264 Multiple Block-Size DCT with Fast and Low. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD)*, pages 407–414, Cavtat near Dubrovnik, Croatia, August 2006.

[8] Grzegorz Pastuszak. Transforms and Quantization in the High-Throughput H.264/AVC Encoder Based on Advanced Mode Selection. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 203–208, Montpellier, France, April 2008.

[9] Heng-Yao Lin, Yi-Chih Chao, Che-Hong Chen, Bin-Da Liu, and Jar-Ferr Yang. Combined 2-D Transform and Quantization Architectures for H.264 Video Coders. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1802–1805, May 2005.

[10] I.E.G. Richardson. H.264/MPEG-4 Part 10: Transform and Quantization. *VCodex Ltd White Paper*, March 2003.

[11] Tu-Chih Wung, Yu-Wen Huang, Hung-Chi Fang, and Liang-Gee Chen. Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264. In *Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS)*, pages 800–803, May 2003.

[12] R. C. Kordasiewicz and S. Shirani. ASIC and FPGA Implementations of H.264 DCT and Quantization Blocks. In *Proceedings of the 2005 IEEE International Conference on Image Processing (ICIP)*, pages III–1020–3, September 2005.

[13] Ozgur Tasdizen and Ilker Hamzaoglu. A high performance and low cost hardware architecture for h.264 transform and quantization algorithms. In *13th European Signal Processing Conference*, pages 4–8, September 2005.