

# ASIP-controlled Inverse Integer Transform for H.264/AVC Compression

N.T. Ngo, T.T.T. Do, T.M. Le *SM-IEEE*, Y.S. Kadam, A. Bermak (\*) *SM-IEEE*  
 Department of ECE, National University of Singapore  
 Department of ECE, Hong Kong University of Science and Technology (\*)  
 elelmt@nus.edu.sg

**Abstract**— In this paper, an Application-Specific Instruction Set Processor (ASIP) -controlled inverse integer transform IP block on a System-on-Chip (SoC) platform is proposed. The proposed design is implemented as an independently operated IP block connected to the ASIP via the Wishbone SoC bus. It features both 4x4 and 8x8 inverse integer transform with additional support for 2x2 and 4x4 Hadamard transforms of DC coefficients. Design portability can be achieved by using the open Wishbone standard for the system bus with a moderate increase in system area. The IP block is controlled by an ASIP which allows functional testability and design flexibility. Compared with existing designs in its class, the circuit area of this design is considerably minimal due to the embodiment of 4x4 circuit in the 8x8 circuit, while achieving a speed of 176MHz.

## I. INTRODUCTION

The new H.264/AVC standard [1-2] is developed by ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) known as Joint Video Team (JVT). The objective of the H.264/AVC is to deliver high quality video at lower bit rates than previous standards by the adoption of effective coding algorithms. One of the tools being adopted is integer transform. The previous video coding standards such as MPEG-2 [3] and MPEG-4 [4] use 8x8 Discrete Cosine Transform (DCT) as the basic transform. To avoid mismatch problem and reduce the expensive floating-point arithmetic implementation, the new approach carries out all transformations using just integer number. The improved DCT-based 4x4 transform can be calculated simply using additions and shift operations, thus significantly reduce the computational complexity for the coding standard. With a maximum of 16-bit integer arithmetic requirement, a small penalty in video quality is traded off for simpler hardware implementation. In addition, the H.264/AVC also adopts second level transforms. Array of luma DC coefficients in intra 16x16 prediction mode, and array of chroma DC coefficients are encoded using 4x4 and 2x2 Hadamard transform correspondingly.

In the updated proposal for fidelity range extensions (FRExt) [5], the new 8x8 integer transform is introduced for coding high resolution video sequences. For improved encoder performance and flexibility, the transform size can

be chosen dynamically, either 4x4 or 8x8, for inter predicted macroblocks. This achieves an average bit-rate reduction of around 10% [5].

In this paper, an Application-Specific Instruction Set Processor (ASIP) -controlled inverse integer transform IP block on a Wishbone-based [6] System-on-Chip (SoC) platform is proposed. It features both 4x4 and 8x8 inverse integer transforms with additional support for 2x2 and 4x4 Hadamard transforms of DC coefficients. The IP block is controlled by an ASIP which allows functional testability and design flexibility.

The remaining of the paper is organized as follows. The H.264/AVC Transform and Quantization is reviewed in Section II followed by the Complexity Analysis in Section III. The proposed Inverse Integer Transform block and the System-Level ASIP Architecture are presented in Section IV. Circuit Simulation and Pre-layout Synthesis Results are discussed in Section V. The paper ends with conclusions in Section VI.

## II. H.264/AVC TRANSFORM AND QUANTIZATION

In the H.264/AVC standard, the residual data will be coded using four different operations: 4x4 integer transform, 8x8 integer transform [5], 4x4 and 2x2 Hadamard transforms [7]. The macroblocks with their scanning order are summarized and depicted in Fig. 1 [8].

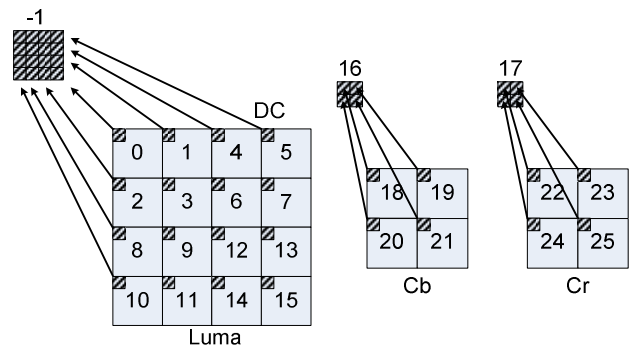


Figure 1: Scanning order of residual blocks within a macroblock

The first type of transform is applied to 4x4 luma blocks 0-15 and 18-25. The second transform use 8x8 block size instead of 4x4. The third and fourth types are applied to DC

coefficients block (-1, 16, 17) when the macroblock is encoded in 16x16 intra-prediction mode [8].

A. *The Integer Transform:*

$$Y = (C_4 X C_4^T) \odot E_f, \quad (1)$$

where  $C_4$  and  $E_f$  are given by:

$$C_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad (2)$$

$$E_f = \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix} \quad (3)$$

and  $X$  is input matrix.

The  $\odot$  symbol indicates that each element of  $(C_4 X C_4^T)$  is multiplied by the scaling factor in the same position in matrix  $E_f$ .  $C_4^T$  is the transpose of matrix  $C_4$ .

The 4x4 inverse integer transform can be written as:

$$X = C_4^i (Y \odot E_i) C_4^{iT}, \quad (4)$$

where  $C_4^i$  and  $E_i$  are given by

$$C_4^i = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \quad (5)$$

$$E_i = \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \quad (6)$$

The 8x8 forward and inverse integer transforms can be performed in a similar manner [5].

The 4x4 Hadamard forward and inverse transforms are given by:

$$Y = (HX H^T) / 2; \quad (7)$$

$$X = (HY H^T), \quad (8)$$

$$\text{where } H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (9)$$

The 2x2 Hadamard transform use the same formula for forward and inverse:

$$Y = HX H^T, \quad (10)$$

$$\text{with } H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (11)$$

B. *Quantization and Rescaling*

The scalar multiplication in the core forward and inverse transforms can be absorbed into the scalar quantization step. The basic quantization operation for forward (12) and inverse (13) transform can be defined as follows:

$$Z_{ij} = \text{round} ( Y_{ij} / Q_{\text{step}} ) \quad (12)$$

$$Y_{ij} = Z_{ij} * Q_{\text{step}} \quad (13)$$

There are 52 values of  $Q_{\text{step}}$  supported by H.264/AVC and each has a Quantization Parameter (QP) associated with it.

To integrate the scalar multiplication by matrix  $E_f$  or  $E_i$  as shown above, and to avoid any division operations, the equation is changed to:

$$Z_{ij} = \text{round} ( Y_{ij} * (PF / Q_{\text{step}}) ) \quad (14)$$

$$Y_{ij} = Z_{ij} * PF * Q_{\text{step}} * 64 \quad (15)$$

where:

$$PF / Q_{\text{step}} = MF / 2\text{qbits} \quad (16)$$

$$\text{qbits} = 15 + \text{floor} ((QP) / 6) \quad (17)$$

$$V_{ij} = (Q_{\text{step}} * PF * 64) \quad (18)$$

And PF is the value from matrix  $E_f$  or  $E_i$  depending on the location of  $Y_{ij}$

Finally, (14) can be rewritten as follows:

$$|Z_{ij}| = (( |Y_{ij}| * MF + f )) \gg \text{qbits} \quad (19)$$

$$\text{sign} (Z_{ij}) = \text{sign} (Y_{ij}) \quad (20)$$

The Offset parameter,  $f$  is  $2^{\text{qbits}/3}$  for Intra Blocks or  $2^{\text{qbits}/6}$  for Inter Blocks.

### III. COMPLEXITY ANALYSIS

The H.264/AVC reference software JM 13.0 [9] is analyzed using a program analysis tool PIN [10] to measure instruction level complexity. Performance complexity analysis is conducted according to the procedure outlined in [11]. The main focuses are arithmetic instructions such as addition, subtraction, shifting, multiplication, and division; and memory access instructions. Three videos sequences are analyzed: *Mother and Daughter* - low motion content; *Foreman* - medium motion content; and *Coastguard* - high motion content. Baseline profile with I-P-P-P mode at 30 fps was selected.

Using PIN tool and *Coastguard* video sequence, the total number of instructions executed per second for forward integer transform is reported to be  $9.14e+9$ , while the total number of memory accesses per second is  $5.67e+9$ , or 62%. On the other hand, the total number of instructions executed per second for inverse integer transform is reported to be  $4.97e+7$ , while the total number of memory accesses per second is  $2.61e+7$ , or 53%. The large percentage of memory accesses implies that the memory interface between the integer transform codec with the system memory must be designed to handle high level of memory accesses. As a result, pairs of 64-b master/slave system-on-chip interfaces were deployed, and its implementation will be discussed in details in Section IV.

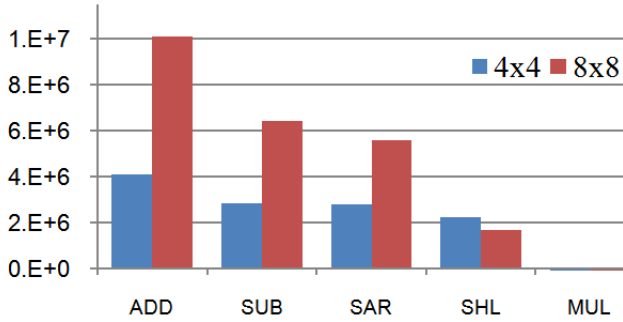


Figure 2: Inverse integer transform - profiles of arithmetic instructions for 4x4 and 8x8 transforms

In a separate analysis where data movement and control instructions are omitted, ADD (addition) and SUB (subtraction) are the most frequently used instructions, with more than 58% of the total in 4x4 transform, and 65% in 8x8 transform. Shift instructions (SAR, SHL) contribute about 30%, MUL (multiplication) and other logic instructions take up a small portion, while DIV (division) is insignificant (Fig. 2). Based on these relationships, more adder and shifter were designed, while multiplier is limited to a minimum.

TABLE I. RATIO OF INVERSE TRANSFORM INSTRUCTIONS OVER TOTAL INSTRUCTIONS

Test sequences(CIF)	Total instr/sec	Inv transform instr/sec	Ratio
Mother Daughter	6.86E+08	1.34E+07	1.95%
Foreman	1.01E+09	3.05E+07	3.04%
Coastguard	1.19E+09	6.54E+07	5.51%

In Table I, the ratio of inverse integer transform instructions over the total number of video encoding/decoding instructions is capped at 5.5%. This draws a clear picture that the computational complexity of integer transform is not critical compared to the overall video codec. Therefore, we have the flexibility to cater additional functions such as the system-on-chip bus to provide the added testability and portability.

#### IV. CIRCUIT IMPLEMENTATION

##### A. The Proposed Inverse Integer Transform

In the H.264/AVC standard, the 2-D forward transform is computed using row-column decomposition technique. That means the 2-D transform is performed as a 1-D vertical (column) transform followed by a 1-D horizontal (row) transform. Since column and row transformations use exactly the same algorithm, the identical 1-D core can be used. Hence, hardware reusability can significantly reduce the circuit area. Consequently, in all the transforms, when calculating  $Y = CXC^T$ , the first transform involves  $Z = CX$ , followed by  $Y = ZC^T$ .

In order to save circuit area, the 4x4 transform circuit is embedded inside the 8x8 transform circuit. Since the bit-depth requirement for H.264/AVC standard [1] is 12 bit to the input of the inverse integer transform, implementation of a 128-bit data bus allows transfer of maximum 8 coefficients per clock cycle (if 16 bits are assigned to each coefficient).

This means 1 column of 8x8 matrix or 2 columns of 4x4 matrix can be processed at a time. Compared to 64-bit, the 128-bit data bus helps to minimize the bottleneck and provides nearly 2 times increase in throughput, with slightly more gate count.

In the proposed inverse transform block (Fig. 3), there are 1 multiplier block, 1 shifter, 4 adder blocks (1, 1', 2, and 3), 1 rounding, 1 matrix transposer, and 2 FIFOs. Two 1-D transform operations are implemented in two individual stages. When the inverse transform is executed, inputs are initially fetched from the input FIFO to the rescaling multipliers. Based on the value of QP, multiplicative factors are generated and used inside the multiplier for rescaling data. Next, data are passed through a multiplexer to the adder blocks for first 1-D matrix multiplication. After that, data are bypassed in the rounding block and stored into the matrix transposer. At this point, rows and columns are exchanged for the second 1-D transform in the next stage. In the second stage, a multiplexer selects data coming from the transposer and passes them through processing steps including core transforming (adder block), rounding, and saving results to the output FIFO.

Detailed operations of the inverse integer transform are described as follows. The input data is first rescaled. This is an important step for all transform modes where the input matrix must be multiplied by the scaling factor in the same position in matrix  $E_f$ . This multiplier block contains 8 parallel multipliers so as to take full advantage of the 128-bit wide data bus. Pipelined multi-stage multipliers are employed to decrease the combinational delay in one clock cycle. This block is also responsible for  $\{QP \text{ DIV } 6\}$  and  $\{QP \text{ mod } 6\}$  operations, and the corresponding multiplicative factors for the transform. In inverse transform, input data is scaled by a factor of maximum 58. Therefore, the multiplier requires only 6 bits. This helps minimize the hardware cost.

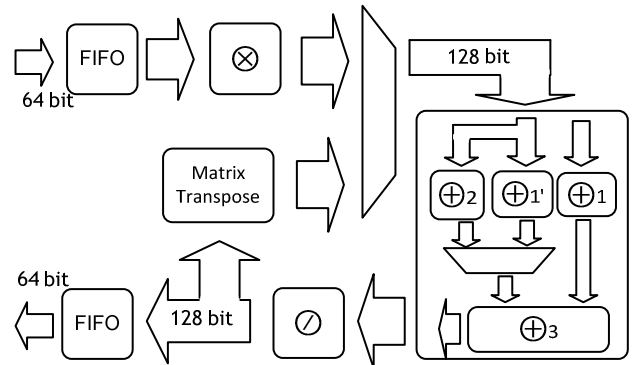


Figure 3: Inverse Integer Transform (IT) Block

After being rescaled in the multiplication block, results are passed onto the adder blocks. In Figs. 4a and 4b, adder block 1 is used in all the 4x4 transform types. There are 8 adders and 2 shifters to perform a 1-D transform for 4x4 data input. As can be seen, depending on the transform mode, proper coefficients ( $\frac{1}{2}$  for integer and 1 for Hadamard transform) are selected accordingly. In order to take advantage of 128 bit data bus, adder block 1 is duplicated

(adder block 1') as shown in Fig.3. In 4x4 mode, 128 bits are divided into two parts, with 64 bits fed to adder block 1, the other fed to adder block 1'. At this step, two 4x4 matrices are computed in parallel by adder blocks 1 and 1'. Afterwards, results from adder block 1' are passed through multiplexer, together with another the results from adder block 1, are passed to adder block 3, and finally moved to the rounding block.

In 8x8 transform mode (Fig. 4c), adder blocks 1, 2 and 3 are all enabled where adder blocks 2 and 3 are specifically designed for 8x8 transform. Adder block 1 which was previously used to compute 4x4 matrix, now is reused to compute 4 pixels of even rows. The other 4 pixels of odd rows are processed in adder block 2. Adder block 3 receives data from adder blocks 1 and 2, compute the final step and send data to the rounding block.

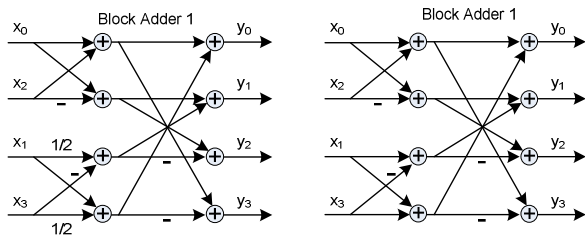


Figure 4: a) Dataflow diagram for 1-D 4x4 inverse integer transform; b) Data flow diagram for 1-D 4x4 inverse Hadamard transform, or 2x2 inverse Hadamard transform

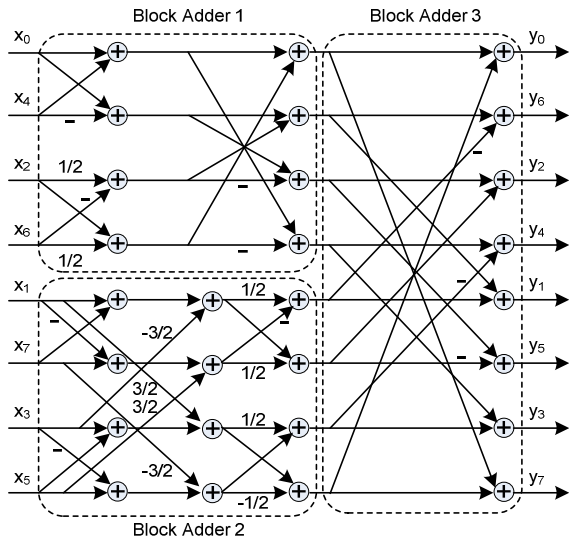


Figure 4c: Dataflow diagram for 8x8 1-D inverse integer transform

The final computation is a division which is implemented as a rounding block. This involves division of the values by 64. This can be done using arithmetic right shifting by 6. Only in 4x4 integer mode that this block is enabled and a simple multiplexer can be used to bypass division in other transform configurations. From here we get the final coefficients of the inverse integer transform.

In total, 8 multipliers, 40 adders, and 20 shifters are required for the whole inverse transform and inverse quantization process.

The transposer requires a total of 64 x 16-bit registers in order to transpose a maximum of 64 pixels in 8x8 matrix. This block is shared between all 4x4 and 8x8 transforms to minimize the number of registers and thus reduce circuit area. With some additional multiplexers, row and column pixels are exchanged. Since the second stage 1-D transform requires 1 row to operate, 4 clock cycles delay are expected for performing 4x4 transform, 8 cycles for 8x8 transform and the same number of cycles is required to completely send out data. To utilize the system and reduce number of cycles for these transforms, pipelining is used to increase the throughput of the whole system.

Since the Wishbone system bus only supports up to 64 bit data bus, while the internal data bus width is 128 bit, there is a need for 2 FIFO buffers to store temporary data. The input buffer must be large enough to store at least 2 matrices for continuous computation of data in the transform stage. That means while the first matrix is being calculated, the second one is transferred from the external memory into the buffer. At the output, the same FIFO is applied so that no data is overwritten. Therefore two 128 x 16-bit FIFOs are implemented.

As we can see, Hadamard transform can be achieved by removing the shift operation in block adder 1 and bypassing rounding block. These can be achieved easily using multiplexers. Especially in 2x2 Hadamard transform, 1-D transform is sufficient to get the Chroma DC coefficients.

## B. System Architecture

### 1) Top-level system architecture

The top-level system architecture – depicted by Fig. 5a – includes an ASIP, an external memory, and the proposed inverse transform with its DMA controller. The data, address, and control buses are regulated by an Arbiter. The Wishbone system bus signals at the master and slave interfaces are shown in Fig. 5b.

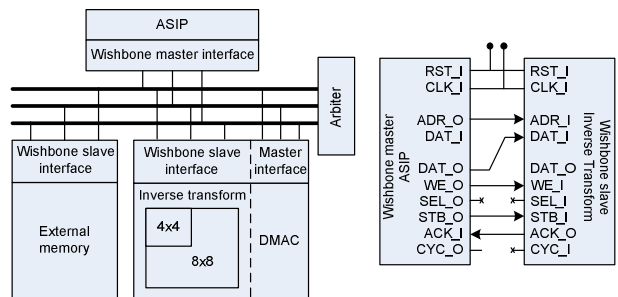


Figure 5: a) Top-level system architecture; b) Wishbone interface signals

### 2) ASIP and instruction set

The inverse integer transform block is controlled by an ASIP. The ASIP architecture is based on the RISC stored program machine (SPM) [12] with specific instruction set to control up to 4 IP blocks.

### a) Architecture of the ASIP

Fig. 6 shows the architecture of the supporting ASIP. The ASIP consists of 3 parts: a controller, a datapath and an internal memory. The controller generates signals to orchestrate the operation of the ASIP. The datapath comprises an ALU, 10 registers, and 2 multiplexers. The ten registers include 4 general-purpose registers R0, R1, R2, R3; Zero flag Reg\_Z, temporary register Reg\_Y, program counter PC; instruction register IR; and address register Add\_R. The Add\_R is used to store memory address information prior to memory write or memory read.

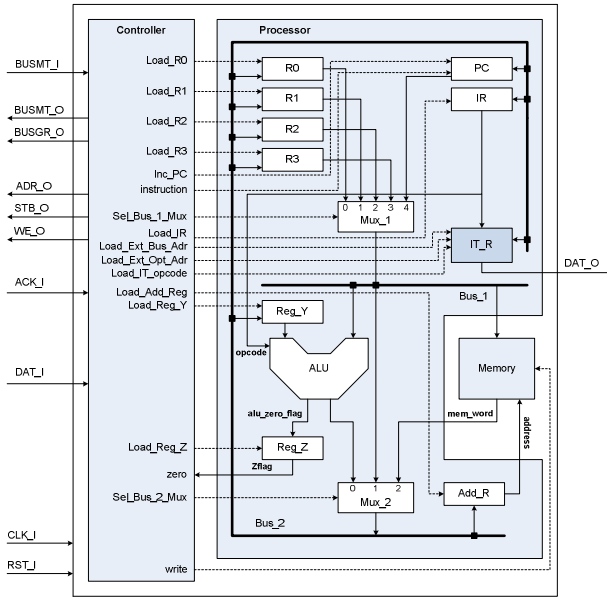


Figure 6: ASIP architecture

The modified ASIP has 10 instructions, in which the newly define instruction IIT is used to invoke the inverse integer transform function.

The ASIP operates through three phases: fetch, decode, and execute. In the fetch phase, it retrieves an instruction from memory using 2 clock cycles. In the decode phase, it decodes the instruction, manipulates the datapath, and loads registers using 1 cycle. In the execution phase, it executes and generates the results for 0 to 5 cycles.

### b) Instruction set

The RISC SPM has three types of instructions: the 8-bit, 16-bit, and 24-bit long. The 8-bit instructions are intended for basic arithmetic operations. The 16-bit instructions are for accessing internal memory, and the 24-bit instructions are for accessing external memory. A typical 24-bit instruction is shown in Table II.

TABLE II. 24-BIT LONG INSTRUCTION

Opcode	Src	Dst	Ext. mem. src. addr.	Ext. mem. dst. addr.
0 0 1 0	0 1	1 0	0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0

TABLE III. NEWLY DEFINED INSTRUCTIONS FOR INVERSE TRANSFORMS

Instr.	Machine byte 1		byte 2	byte 3	Action	
	Opcode	Src	Dst	E-src		E-dest
IIT	1010	src	??	E-src	E-dest	Exec inverse int. transform

In this instruction, Src is used to indicate the four types for inverse integer transform: 4x4 integer transform, 4x4 Hadamard transform, 2x2 Hadamard transform and 8x8 integer transform.

### 3) The inverse transform block

The inverse transform (IT) block has an integer transform component with Wishbone [6] slave interface to communicate with its Wishbone master interface of the ASIP, and a DMAC with Wishbone master interface to directly transfer image data with the external memory.

### 4) System-on-Chip Shared Bus

The ASIP, external memory, and the IT blocks are connected using a SoC bus, which is controlled by an arbiter. The arbiter is to decide which master will transfer data at a certain time.

In the proposed system, the ASIP is a bus master while the external memory and the IP block are bus slaves. However, this special IP block is designed with a built-in DMAC which helps improve the transfer speed of image data from the external memory to IP block, and vice versa. The DMAC itself also is a bus master of its own block. As a result, the system has 2 bus masters and 2 bus slaves in total.

### 5) ASIP and Hand-shaking Issues with IP block(s)

The ASIP and IT block are connected via a SoC-based Wishbone system bus, whose master and slave interfaces are shown in Fig. 5b.

In the interface, ASIP sends the strobe signal STB\_O, write enable signal WE\_O, IT slave address through ADR\_R and command through DAT\_O to all the slaves in the system. The IT block then reads the slave address in the address bus and decides if it is the intended block. This slave immediately sends back acknowledge signal to the ACK\_I of the ASIP to indicate that it is ready, and read all the necessary data in the bus. After receiving the acknowledge signal, the ASIP sends BUSMT\_O signal with the new bus master code – which represents the above IT block – to the arbiter. Because the IT block has Wishbone master and slave interface at the same time, so at the next cycle, the above IT block will become the bus master of the system and has the full right to use the system bus. And the ASIP continues its operation on its internal memory and registers.

Upon completion, the master interface of IT block will send CYC\_O to the arbiter to signal its completion. After receiving this signal, the arbiter will re-assign the ASIP as the default system bus master.

## V. SIMULATION AND PRE-LAYOUT SYNTHESIS RESULTS

### A. Functionality, testability, and portability

The proposed design has been implemented with 4x4 inverse transform for the residual data; 4x4 and 2x2 inverse transform for DC coefficients; and the newly added 8x8 inverse transform.

Quantization block is also integrated into our design. The hardware sharing methodology of different types of transform helps reduce the system area at the cost of a few extra clock cycles. Based on the instruction analysis, this cost is more affordable and timing requirement can be met at slightly higher operating frequency.

Video test sequences can be converted into bit streams and complete tests are to be conducted on the ASIP. Since the proposed IP block of the inverse integer transform was designed on an SoC environment assuming Wishbone bus, it can be ported to other SoC platforms of similar system bus characteristics.

### B. Performance

The proposed inverse integer transform and quantization architecture were implemented in Verilog, and verified with RTL simulations using Mentor Graphics ModelSim. System-level functional verification was also performed by passing different input patterns to the architecture and comparing the output with the results obtained by passing the same inputs to the defining Eqs 1, 4, 7, and 8 in Section II. It was pre-layout synthesized using AMS 0.35  $\mu\text{m}$  technology library by Synopsys Design Compiler. The gate count is 21.5k for inverse integer transform in combination with quantization at 150 MHz. Table IV lists the reported designs with their performances.

TABLE IV. PERFORMANCE COMPARISON OF PRE-LAYOUT SYNTHESIS RESULTS

Design	Type of transf	Tech ( $\mu\text{m}$ )	Gate <sup>(*)</sup> ( $10^3$ )	Speed (MHz)	Throughput (ppc)	Quant.
Wang03 [13]	4x4	0.35	6.5	80	4	No
Raja05 [14]	4x4	Virtex II	3.2K slcs + 4.1K ff	127	- <sup>(**)</sup>	Yes
Amer05 [15]	8x8 only	Virtex II	29K LUT	68	-	Yes
Kord05 [16]	4x4	Virtex II	1.6K slcs + 0.5K ff	97	16	Yes
Kord05 [16]	4x4	0.18	51.6	68 <sup>(***)</sup>	16	Yes
Chen06 [17]	4x4	0.18	6.4	100	8	No
Shi07 [18]	4x4	0.18	5.0	166	8	No
Chao07 [19]	All <sup>(****)</sup>	0.18	18.5	125	8	No
Proposed	All	Virtex 4	2.0K slcs + 1.2Kff	145	8	Yes
Proposed	All	0.35	21.5	150	8	Yes

(\*) Gate count is computed as the total combinational area normalized by the area of a 2-input NAND gate; (\*\*) unable to obtain from the reference; (\*\*\*) Speed is computed based on the critical delay of speed-optimized quantization circuit. This is the slower of the two DCT and quantization circuits; (\*\*\*\*) 'All' means all 4x4, 8x8 integer transforms, and 4x4, 2x2 Hadamard transform are supported.

In Table IV, the reported designs are listed in chronological order, and if two designs are reported on the same year, the FPGA design is listed first, followed by the ASIC design. We try to list the reported designs, but not all designs can be compared due to different reported parameters.

In the designs reported by [13, 14, 16, 17, and 18], the 8x8 integer transform is not supported. Highest speed of 166MHz is achieved by [18], while highest throughput of 16 pixels per cycle is achieved by [16] for a relatively large circuit area of 51.6K gates. We note here that the high throughput also implies a very wide data bus of as large as 256 bits (16-bit by 16 pixels). However, there is not a standard technique to report overall circuit area including buses.

Both designs by [14, 16] provide support for quantization. While [16] was implemented using both Xilinx Virtex II and 0.18  $\mu\text{m}$  TSMC technology, [14] was implemented using only Xilinx Virtex II. Among designs reported by [17] and [18], design in [18] is the most area-efficient assuming 0.18  $\mu\text{m}$  technology, using only 5K gates. Since throughput is heavily dependent on the data bus width and its supporting circuitries (multiplexers and drivers) which are not commonly reported, the data throughput per unit area – of combinatorial circuit - (DTUA) cannot yet be used as a performance parameter.

In [15], the implementation of 8x8 integer transform followed by quantization was reported. However, support for complete coding of residual data was not implemented.

Finally, in [19] and this paper, complete support for coding of residual data is implemented. The throughput of [19] is comparable to the proposed design at 8 pixels per cycle. In terms of circuit area, the proposed design outperforms [19] in its relatively smaller area (21.5K using 0.35 $\mu\text{m}$  technology vs. 18.5K using 0.18 $\mu\text{m}$  technology), including additional circuitries for quantization.

For real-time requirement, the proposed architecture can process all existed frame sizes [1-2]. For example, if the clock frequency is operated at 118 MHz, it achieves the super high-definition in 4:2:0 format, 4096 x 2304 resolution at 26.7 frames/sec in either 8x8 or 4x4 mode.

## VI. CONCLUSION

In this paper the most functionally complete inverse integer transform and quantization block is proposed. The design is implemented on an ASIP-controlled SoC platform for testability and portability. The resulting circuit area is considerably minimal compared to the design in its class due to the embodiment of 4x4 circuit in the 8x8 circuit with quantization, and achieves a speed of 150MHz.

## ACKNOWLEDGMENT

This work is supported by the Faculty Research Committee grant (R-263-000-405-112 and R-263-000-405-133), Faculty of Engineering, National University of Singapore.

## REFERENCES

- [1] "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC," in Joint Video Team of ISO/IEC JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-G050, Mar. 2003.
- [2] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on*

- Circuits and Systems For Video Technology*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
- [3] ISO/IEC 13818, “Coding of Moving Pictures and Audio”, Nov. 1993.
- [4] ISO/IEC 14496, “Coding of Moving Pictures and Audio”, Mar. 2002.
- [5] S. Gordon, D. Marple, and T. Wiegand, “Simplified use of 8×8 transforms – updated proposal and results,” Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 11th Meeting, JVT-K028, Munich, Germany, Mar. 2004.
- [6] Wishbone Specifications: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision: B.3, Sep. 2002.
- [7] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-Complexity Transform and Quantization in H.264/AVC,” *IEEE Transactions on Circuits and Systems For Video Technology*, Vol. 13, No. 7, pp. 598-603, Jul. 2003.
- [8] I.E.G. Richardson, *H.264 and MPEG-4 Video Compression*, Ed. Jon Wiley & Son, ISBN: 0-470-84837-5, Sep. 2003.
- [9] Reference video codec software: JM 13.0. Available at: <http://iphome.hhi.de/suehring/tml/>
- [10] C-K Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, and K. Hazelwood, “PIN: Building Customized Program Analysis Tools with Dynamic Instrumentation,” *Programming Language Design and Implementation (PLDI)*, pp. 190-200, Jun. 2005.
- [11] T.M. Le, X.H. Tian, B.L. Ho, J. Nankoo, and Y. Lian, “System-on-Chip Design Methodology for a Statistical Coder,” *Proc. of the 17th IEEE Inter. Symp. on Rapid System Prototyping*, pp.82-88, 2006.
- [12] M.D. Ciletti, *Advanced Digital Design with the Verilog HDL*, Chapter 7, Prentice Hall, ISBN 0-13-121974-X, 2003.
- [13] T-C. Wang, Y-W Huang, H-C. Fang, and L-G. Chen, “Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264”, *Proceedings of ISCAS’03*, Vol. 2, pp.II-800 - II-803, May 2003.
- [14] G. Raja, S. Khan, and M.J. Mirza, “VLSI Architecture and Implementation of H.264 Integer Transform”, *Proceedings of the 17<sup>th</sup> International Conference on Microelectronis*, pp.218-233, Dec. 2005.
- [15] I. Amer, W. Badawy, and G. Jullien, “A High-performance Hardware Implementation of the H.264 Simplified 8x8 Transformation and Quantization”, *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp.1137-1140, 2005.
- [16] R.C. Kordasiewicz, S. Shirani, “ASIC and FPGA Implementations of H. 264 DCT and Quantization Blocks”, *IEEE ICIP*, pp.1020-1023, Sep. 2005.
- [17] K-H. Chen, J-I. Guo, and J-S. Wang, “A High-Performance Direct 2-D Transform Coding IP Design for MPEG-4 AVC/H.264”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, No. 4, pp. 472-483, Apr. 2006.
- [18] B. Shi, W. Zheng, D. Li, and M. Zhang “Fast Algorithm and Architecture Design for H.264/AVC Multiple Transforms”, *Proc. of IEEE International Conference on Multimedia and Expo’07*, pp.2086-2089, Jul. 2007.
- [19] Y-C. Chao, H-H. Tsai, Y-H. Lin, J-F. Yang, and B-D. Liu, “A Novel Design for Computation of all Transforms in H.264/AVC Decoders”, *Proc. of IEEE International Conference on Multimedia and Expo’07*, pp.11914-1917, Jul. 2007.