New distributed arithmetic algorithm and its application to IDCT

T.-S.Chang, C.Chen and C.-W.Jen

Abstract: Distributed arithmetic (DA) has been widely used to implement inner product computations with a fixed input. Conventional ROM-based DA suffers from large ROM requirements. A new DA algorithm is proposed that expands the fixed input instead of the variable input into bit level as in ROM-based DA. Thus the new DA algorithm can take advantage of shared partial sum-of-products and sparse nonzero bits in the fixed input to reduce the number of computations. Unlike ROM-based DA that stores the precomputed results the new DA algorithm uses a predefined structure to compute results. When applied to a 1-D eight-point DCT system the new DA algorithm only needs 30% of hardware area and has faster speed as compared with ROM-based DA. To illustrate the efficiency of the proposed algorithm a 2-D IDCT chip was implemented using 0.8μm SPDM CMOS technology. The chip with size 4575 × 5525μm can deliver a processing rate of 50 Mpixels per second.

1 Introduction

Computation of inner products dominates computation cost in many digital signal processing (DSP) applications. Though inner-product designs using multipliers and accumulators (MAC) are fast, the associated cost is intolerable when long-length inner-product computation is considered. Instead of using MAC, distributed arithmetic (DA) [1] uses ROM that stores the precomputed partial sum of inner products. This computational efficiency makes DA popular in various DSP applications in which one of the multiplication operands is fixed, including filters, convolution and video processing applications like discrete cosine transform (DCT) and inverse DCT (IDCT) [2, 3].

DCT and IDCT [4] has been selected as an important approach in many video codec standards [5] to reduce the spatial redundancies induced from the correlation of signals. Due to the inherent high computation complexity requirement, high-speed and low-cost designs are inevitable in many real-time video applications. To achieve the goals, most of the VLSI implementations of 2-D DCT/IDCT use row–column decomposition to convert a 2-D transform into consecutive 1-D ones and adopt DA to compute 1-D DCT/IDCT.

The DA technique distributes arithmetic operations rather than lumps them as multipliers do. Conventional DA [1] called ROM-based DA decomposes the variable input of the inner product into bit level to generate precomputed data. ROM-based DA uses a ROM table to store the precomputed data, which makes it regular and efficient in silicon area in VLSI implementation. However, when the size of the inner product increases the ROM area increases exponentially and becomes impractically large, even using ROM partition [6].

We present a new DA algorithm [7] called adder-based DA for solving the problem. This algorithm, in contrast to conventional DA, decomposes the other operand of inner product into bit level, distributes the multiplication operation, and shares the common summation terms. The adder-based DA exploits the distribution of binary value pattern and may maximise the hardware sharing possibility in the implementation. Therefore the adder-based DA requires less hardware area and smaller computation cycle time than ROM-based DA. Due to its inherent sharing property, the proposed adder-based DA technique is very suitable for multiple inner-product calculations like DCT and IDCT [7]. Since the adder-based DA shares common summation terms between computations, it can be regarded as one of the class of the common subexpression algorithms [7–10]. Unlike previous word-level sharing algorithms, the bit-level DA formulation provides implementation benefits for efficient bit-serial hardware designs.

2 Algorithms and architectures of DA

An inner product of length L is defined as

$$y = \sum_{i=1}^{L} A_i X_i \tag{1}$$

where A_i is a fixed coefficient, and X_i is a variable input. To keep equations simple, A_i and X_i are expressed in unsigned fraction (two's complement form can also be used) as follows:

$$A_i = \sum_{j=1}^{M} A_{i,j} 2^{-j} \tag{2}$$

$$X_i = \sum_{k=1}^{N} X_{i,k} 2^{-k} \tag{3}$$

IEE Proceedings online no. 19990537
DOI: 10.1049/ip-cds:19990537
Paper first received 11th May 1998 and in revised form 29th March 1999

The authors are with the Department of Electronics Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, Republic of China

where M is the word length of A_i and N is the word length of X_i. To realise the inner-product computation in eqn. 1 efficiently, using DA is a good strategy from the hardware cost and computing speed points of view. To help the derivation and illustrate the difference of different DA approaches, we use both mathematical forms and the dependence graph (DG) [11] to represent the two DA approaches. The DG of the two approaches can be derived by proper projection and scheduling of the bit-level DG shown in Fig. 1.

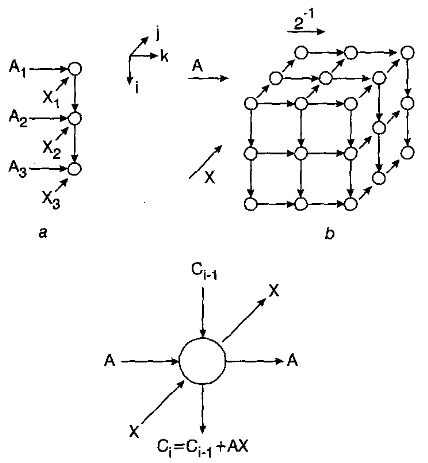

Fig. 1 *DG of inner product*
a DG in word level form
b DG in bit level form
c Node operations

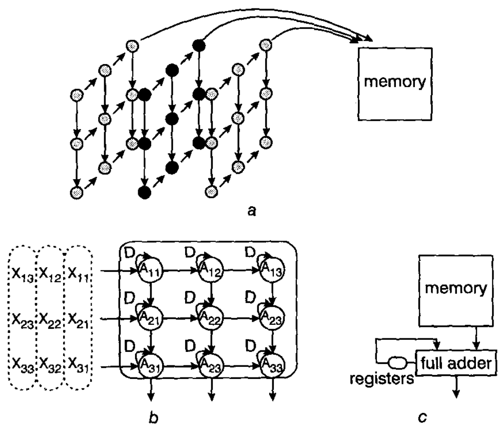

Fig. 2 *ROM-based DA design*
a Projection along [j k i] = [0 1 0]
b SFG
c Hardware implementation

2.1 ROM-based DA

ROM-based DA decomposes the variable input X_i into bit level. Substituting eqn. 3 into eqn. 1 obtains

$$y = A \cdot X = \sum_{i=1}^{L} A_i X_i = \sum_{i=1}^{L} A_i \left(\sum_{k=1}^{N} X_{i,k} 2^{-k} \right)$$
$$= \sum_{k=1}^{N} \left(\sum_{i=1}^{L} A_i X_{i,k} \right) 2^{-k} \quad (4)$$

It precomputes these partial products $\Sigma_{i=1}^{L} A_i X_{i,k}$ and stores them in a lookup table ROM for all possible combinations of $X_{i,k}$ value. The precomputed values are accessed by the input data and accumulated by shifts. Fig. 2 illustrates the signal flow graph (SFG) of the ROM-based DA design derived by projection along [0 1 0] and the corresponding implementation. Since all nodes along k-planes are constant, their values can be precomputed and stored in a ROM table. The size of the required ROM tables is proportional to 2^L. As the length of inner products L becomes large, the ROM-based DA suffers from extremely large ROM requirements.

2.2 Adder-based DA

Adder-based DA decomposes the fixed coefficients A_i instead of variable input X_i into bit level. Substituting eqn. 2 into eqn. 1 obtains

$$y = A \cdot X = \sum_{i=1}^{L} A_i X_i = \sum_{i=1}^{L} \left(\sum_{j=1}^{M} A_{i,j} 2^{-j} \right) X_i$$
$$= \sum_{j=1}^{M} \left(\sum_{i=1}^{L} A_{i,j} X_i \right) 2^{-j} \quad (5)$$

The term $\Sigma_{i=1}^{L} A_{i,j} X_i \equiv S_j$ is a combination of X_i since $A_{i,j}$ is only 0 or 1. If X_i is a serially input one can obtain S_j bit by bit via serial adders, that is

$$S_j = \sum_{t=1}^{P} S_{j,t} 2^{-t} \quad (6)$$

Thus eqn. 5 can be rewritten

$$y = \sum_{j=1}^{M} S_j 2^{-j} = \sum_{j=1}^{M} \left(\sum_{t=1}^{P} S_{j,t} 2^{-t} \right) 2^{-j}$$
$$= \sum_{t=1}^{P} \left(\sum_{i=1}^{M} S_{j,t} 2^{-j} \right) 2^{-t} \quad (7)$$

Thus, one can shift and then accumulate the term $\Sigma_{i=1}^{M} S_{j,t} 2^{-j}$ at each cycle t to obtain the inner product. Since S_j is computed using adders the proposed DA algorithm is called adder-based DA. There are two representations of the adder-based DA for two implementation styles. One is bit-parallel form of eqn. 5 and the other is bit-serial form of eqn. 7. Fig. 3 shows the SFG of the adder-based DA design and the corresponding implementation. All j-planes are combined and collapsed into a summation network, in which only unique and nonzero nodes have to be computed.

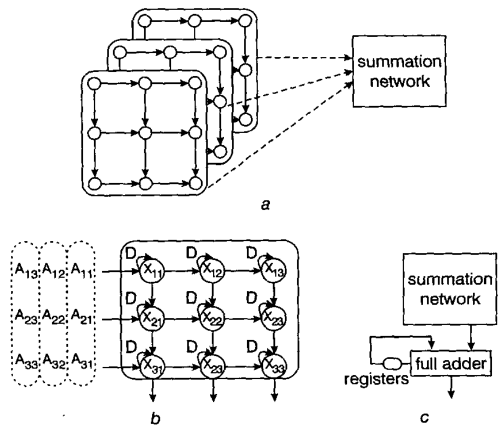

Fig. 3 *Adder-based DA design*
a Projection along [j k i] = [1 0 0]
b SFG
c Hardware implementation

Fig. 4 shows an adder-based DA example to illustrate how adder-based DA works. The adder-based DA algorithm first decomposes fixed coefficients into bit level. After rearranging these terms, one finds that additions are needed only at the nonzero bits of fixed coefficients. This is called zero-one pattern property. In addition, the summation term $X_1 + X_3$ can be shared between bit weights 2^1 and 2^0. This is called common term sharing property. Thus, hardware area is saved by exploiting these two properties.

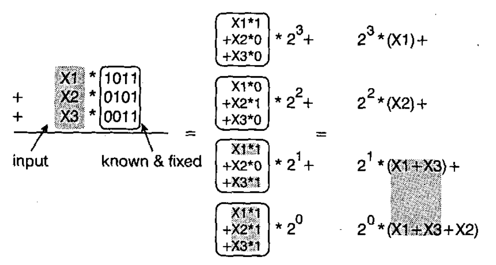

Fig. 4 *Example of adder-based DA*

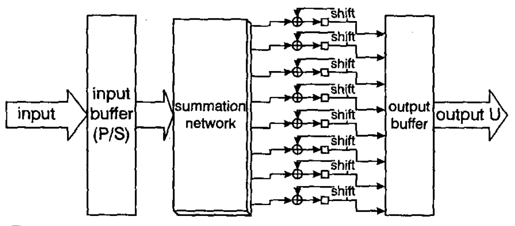

Fig. 5 *Adder-based DA architecture realising bit-serial form of adder-based DA algorithm*

2.3 Architecture design

Fig. 5 shows the adder-based DA architecture realising the bit-serial form of eqn. 5. The architecture consists of an input buffer (parallel-to-serial converter), a summation network, and a shift-add part. The input buffer converts parallel inputs and serially outputs bits of each input. The parallel-to-serial converter can be omitted in the architecture realising the bit-parallel form of eqn. 7. The summation network is a tree structure that connects the required input and generates their summation terms. The summation network of the example is shown in Fig. 6. The shift-add part shifts and accumulates the output of the summation network to generate final inner-product results.

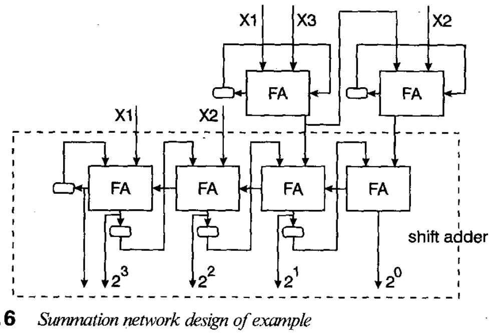

Fig. 6 *Summation network design of example*

2.4 Precision analysis

Given the error constraint, one can derive the required word length of coefficient A_i. If the coefficient A_i^∞ has infinite precision, A_i has finite word length M and $|A_i^\infty - A_i| \leq 2^{-M}$, the maximum error of the adder-based DA design is

$$
\begin{aligned}
|y - y^\infty| &= \left| 2^{-M} \sum_{i=1}^{L} \left[A_{i,M} - A_{i,M}^\infty \right] X_i \right. \\
&\quad \left. + \sum_{j=M+1}^{\infty} 2^{-j} \left[\sum_{i=1}^{L} A_{i,j}^\infty X_i \right] \right| \\
&\leq L \max(X_i) 2^{-M} \left| A_{i,M} - A_{i,M}^\infty \right| \\
&\quad + L \max(X_i) \sum_{j=M+1}^{\infty} A_{i,j}^\infty 2^{-j} \\
&= L \max(X_i) \left| A_i - A_i^\infty \right| \\
&\leq L \max(X_i) 2^{-M} \qquad (8)
\end{aligned}
$$

3 Comparisons with relevant approaches

3.1 Comparisons of DA algorithms

To illustrate the area and speed advantages we used 1-D eight-point DCT as an example whose results are shown in Table 1. We only compare ROM tables and summation networks since the two DA methods mainly differ on this part. All the data are estimated by using a 0.8μm SPDM CMOS standard-cell library. The summation network only requires 10% of the transistor counts and 30% of hardware area compared with ROM tables in the ROM-based DA design. The gate area in the summation network is 0.378mm^2 that occupies 47.25% of the summation network area. From the Table the adder-based DA design consumes less hardware area and is faster than the ROM-based DA design.

Table 1: Comparisons of ROM and summation network in split kernel eight-point DCT design

	Hardware cost	Transistor count	Estimated area (include routing)	Estimated delay time
ROM	Eight 16w × 18b ROM tables	≥ 6000	2.6mm^2	3.6ns
Summation network	22 serial adders	≅ 750	0.8mm^2	2.4ns

Utilising different function units, the ROM and summation network, results in the different requirements of word length and numbers of accumulation cycle in shift-adders. From eqn. 4, the maximum value of the ROM data in the ROM-based DA design is $\max[\Sigma_{i=1}^{L} A_i X_i k] = \Sigma_{i=1}^{L} |A_i|$. So the output of the ROM should be at least $\lfloor \log_2(\Sigma_{i=1}^{L} |A_i|) \rfloor + 1$ bits wide and the width of the accumulator should also be the same. In the adder-based DA design the accumulator is M-bits wide because the summation network generates M-bits output. Hence the adder-based DA design needs shorter word length of shift adders in generating final inner-product results.

The number of accumulation cycles in the ROM-based DA design is smaller than that in the adder-based DA design. It needs N accumulation cycles in the ROM-based DA design. However, the accumulators in the adder-based DA design need additional cycles to add the carry from the summation network. The additional number of cycles is $(\lfloor \log_2 \max|\Sigma_{i=1}^{L} A_{i,j} X_i| \rfloor + 1) - N \leq \log_2 M$, which depends on the application specifications. Considering the 2-D IDCT [13] design example, the input is 12 bits wide and the output will not exceed 12 bits. In this situation no additional cycles are needed.

3.2 Comparison with other subexpression sharing approaches

The proposed DA-based algorithm can be regarded as one of the classes of subexpression sharing techniques which can be classified according to how the shared common term is generated, as illustrated in Fig. 7. It shows the computation of the equation $Y = a^*x_1 + b^*x_2 + c^*x_3 + d^*x_4$, where $a = 0010101_b$, $b = 0101010_b$, $c = 1110101_b$, and $d = 1100101_b$. For each sharing type the input to the common term should be available simultaneously for computations. Approaches in [8, 9] generated the shared terms in the word-serial bit-parallel input direction (horizontal circles in Fig. 7). The shared-term generation in [10] was extended to be in the skewed direction (diagonal circles in Fig. 7). Our proposed approach shared the common terms in the word-parallel bit-serial direction (vertical gray circle in Fig. 7), which is very suitable for the hardware implementation when exploiting the parallel processing features. DA-based approaches can separate the low transition probability MSBs from the high transition probability LSBs, which is beneficial to the power management.

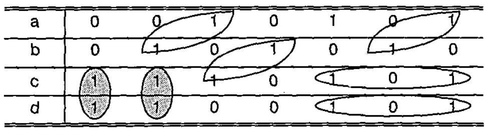

Fig. 7 *Sharing term formulations by different subexpression sharing approaches*

4 Issues of adder-based DA design

For the implementation of the adder-based DA design, eqn. 5 or eqn. 7 can be utilised. Eqn. 5 is suitable for software implementation while eqn. 7 is suitable for VLSI implementation due to the DA bit-serial nature. Software implementation of eqn. 5 is suitable for simple programmable processor without multipliers since the multiple multiplication and additions are reduced to just a few operations of shift and addition. Further reduction of operation numbers can be achieved by combining adder-based DA with fast algorithms. Software implementation of eqn. 5 also enables adder-based DA to apply to the adaptive designs since we can examine coefficient bits dynamically.

Due to the inherent sharing property, adder-based DA is very suitable for calculations of multiple inner products. Adder-based DA can share the common computations among multiple inner products and combine them into a summation network. In contrast, the ROM-based DA design has to store-separated ROM tables for each set of coefficients. The coefficients of multiple inner products can be either multiple dimensional coefficients like DCT or different sets of coefficients like DCT and DST. To calculate multiple inner products, just use a summation network to generate the desired subexpressions. If the coefficient sets are different like DCT and IDCT, use a shuffle network to select the terms required. Fig. 8 shows the architecture for multiple inner product calculations. The drawback of such design is extra shuffle network area and delay.

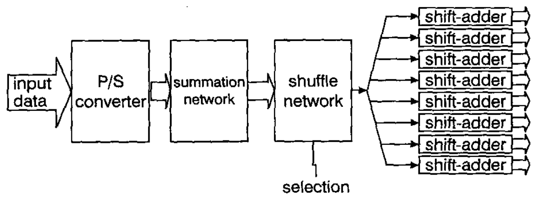

Fig. 8 *Architecture for multiple inner product calculations*

The optimisation problem of the proposed adder-based design is how to find the common terms from the nonzero subexpressions to reduce the summation network area. This is analogous to the logic minimisation problem that extracts the common terms, i.e. a NP complete problem. Fortunately, in some real systems like DCT and IDCT, the search space is not large such that one can find the optimal solution by exhaustive search. For more complicated cases, logic minimisation tools can be used to find the solution. Algorithms and optimisation techniques developed in previous subexpression techniques [8–10] can be modified to find efficient common subexpressions by considering the type of adder-based DA subexpressions.

In addition to direct optimisation, data partition provides a tradeoff between performance and area as the design specification changes. Data partition can be either data-independent or data-dependent. Data independent method directly partitions vector size L and input word length N, which has been used in previous work on ROM-based DA designs [6]. Data-dependent partition exploits bit patterns of coefficients. The rule for this type of partition is to group the coefficients with similar zero-one patterns into one partition. Fig. 9 shows an example on DCT coefficients. Depending on the bit patterns, different realisation methods can achieve lower hardware cost.

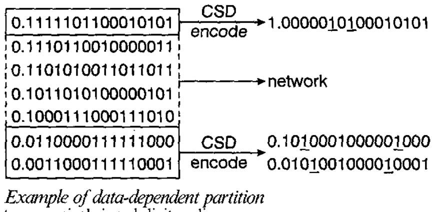

Fig. 9 *Example of data-dependent partition*
CSD denotes canonical signed digit coding

5 Application to 2-D IDCT processor

We illustrate a design example of 2-D 8 × 8 IDCT processor based on proposed adder-based DA approach. This IDCT chip adopts the row–column decomposition to compute 2-D IDCT via 1-D IDCT. The 1-D eight-point IDCT is first reformulated by the split-kernel method to split the 8 × 8 kernel into two 4 × 4 kernels to reduce computations. The split kernel equations of 1-D eight-point IDCT are

$$\begin{bmatrix} V_0 + V_7 \\ V_1 + V_6 \\ V_2 + V_5 \\ V_3 + V_4 \end{bmatrix} = \begin{bmatrix} \cos 4\theta & \cos 4\theta & \cos 4\theta & \cos 4\theta \\ \cos 2\theta & \cos 6\theta & -\cos 6\theta & -\cos 2\theta \\ \cos 4\theta & -\cos 4\theta & -\cos 4\theta & \cos 4\theta \\ \cos 6\theta & -\cos 2\theta & \cos 2\theta & -\cos 6\theta \end{bmatrix}^{-1} \begin{bmatrix} U_0 \\ U_2 \\ U_4 \\ U_6 \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} V_0 - V_7 \\ V_1 - V_6 \\ V_2 - V_5 \\ V_3 - V_4 \end{bmatrix} = \begin{bmatrix} \cos \theta & \cos 3\theta & \cos 5\theta & \cos 7\theta \\ \cos 3\theta & \cos 9\theta & \cos 15\theta & \cos 21\theta \\ \cos 5\theta & \cos 15\theta & \cos 25\theta & \cos 35\theta \\ \cos 7\theta & \cos 21\theta & \cos 35\theta & \cos 49\theta \end{bmatrix}^{-1} \begin{bmatrix} U_1 \\ U_3 \\ U_5 \\ U_7 \end{bmatrix} \tag{10}$$

Fig. 10 shows the block diagram of 1-D eight-point IDCT. The target throughput rate is one pixel per cycle, that is, the design has to complete 1-D eight-point IDCT computation in eight cycles. To attain such throughput, the speed of summation networks is designed to do a two-bits addition per cycle since the word length of the input is 12 bits. The common terms of summation networks are searched by the exhaustive method. The IDCT coefficients are first scaled by $\sqrt{2}$ to reduce the nonzero bits. The scaling factor is easily recovered by a shift since row-wise IDCT following by columnwise IDCT will make the scale factor be two. The serial adder in the summation network is composed of two full adders and a D-flip-flop (DFF) with reset. The two networks contain 22 full adders, 11 DFFs and 30 output latches. The total gate count of the network is 481, while 40% of the gate count is needed for output latches. The final shift-adders accumulate two 16-bit words per cycle by using a carry save adder and a BLC adder [12].

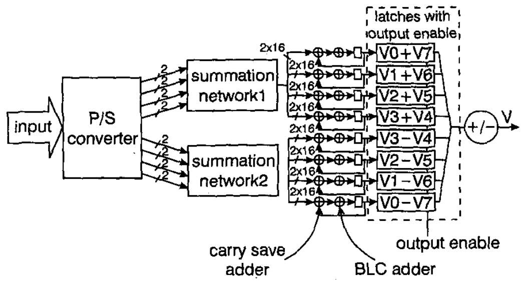

Fig. 10 *Block diagram of 1-D IDCT design*

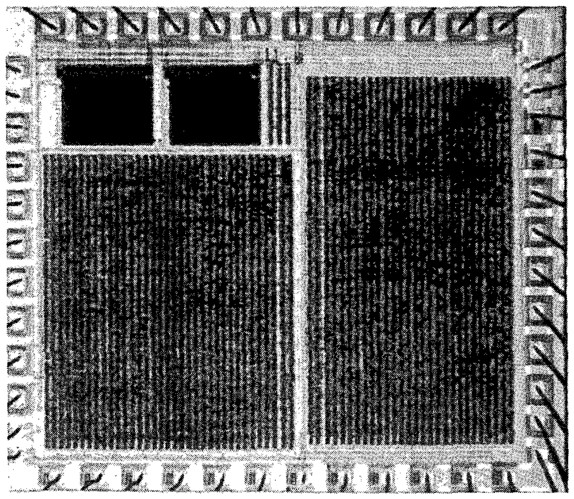

Fig. 11 *Microphotograph of 2-D IDCT chip*

Fig. 11 shows the microphotograph of the chip fabricated using 0.8μm SPDM CMOS technology. The chip size is 4575 × 5525μm and it achieves 50MHz working frequency. The precision of the chip meets the requirement of the IEEE standard [13], as listed in Table 2.

6 Conclusion

We have presented a new DA algorithm called the adder-based DA and its application to 2-D IDCT processor. This algorithm decomposed the fixed coefficients into bit level instead of decomposing variable input into bit level. Thus, one can exploit the constant and numerical characteristics of the fixed input to share and save hardware cost. This effectiveness makes the adder-based DA be a superior design choice over the ROM-based DA in current DA applications. Considering a 1-D DCT design, the adder-based DA only needs 30% of ROM area as compared with the ROM-based DA approach. A 2-D IDCT chip was designed and implemented based on the proposed adder-based DA approach to illustrate the efficiency associated with the proposed approach.

Table 2: Precision requirements of IDCT and simulation result

Items	Standard specification	Max value of chip
Pixel peak error	1	1
Peak mean square error	0.06	0.0224
Peak mean error	0.015	0.0145
Overall mean square error	0.02	0.0148
Overall mean error	0.0015	0.0012
All zero in	all zero out	all zero out

7 Acknowledgment

This work is supported by National Science Council, R.O.C., under the grant NSC-86-2221-E-009-014.

8 References

1 WHITE, S.A.: 'Applications of distributed arithmetic to digital sequence processing: a tutorial review', *IEEE ASSP Mag.*, 1989, **6**, (3), pp. 5–19
2 SUN, M.T., CHEN, T.C., and GOTTLIEB, A.M.: 'VLSI implementations of a 16×16 discrete cosine transform', *IEEE Trans. Circuits Syst.*, 1989, **CAS-36**, (4), pp. 610–617
3 URAMOTO, S., INOUE, Y., TAKABATAKE, A., TAKEDA, J., YAMASHITA, H., TERANE, H., and YOSHIMOTO, M.: 'A 100 Mhz 2-D discrete cosine transform core processor', *IEEE J. Solid-State Circuits*, 1992, **27**, (4), pp. 492–499
4 RAO, K.R. and YIP, P.: 'Discrete cosine transforms – Algorithms, advantages, applications' (Academic, Boston, MA, 1990)
5 RAO, K.R. and HWANG, J.J.: 'Techniques and standards for image, video and audio coding' (Prentice Hall, New Jersey, 1996)
6 NOURJI, K., and DEMASSIEUX, N.: 'Optimization of real-time VLSI architectures for distributed arithmetic-based algorithms: application to HDTV filters'. Proceedings of IEEE international symposium on *Circuits and systems*, 1994, Vol. 4, pp. 223–226
7 CHEN, C.-S., CHANG, T.-S., and JEN, C.-W.: 'The IDCT processor on the adder-based distributed arithmetic'. Proceedings of symposium on *VLSI circuits*, June 1996, pp. 36–37
8 DEMPSTER, A.G., and MACLEOD, M.D.: 'Use of minimum-adder multiplier blocks in FIR filters', *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, 1994, **42**, (9), pp. 569–577
9 POTKONJAK, M., SRIVASTAVA, M.B., and CHANDRAKASAN, A.P.: 'Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1996, **15**, (2), pp. 151–165
10 HARTLEY, R.I.: 'Subexpression sharing in filters using canonic signed digit multiplier', *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, 1996, **43**, (10), pp. 677–688
11 KUNG, S.Y.: 'VLSI array processors' (Prentice-Hall, New Jersey, 1988)
12 SUZUKI, K., YAMASHINA, M., GOTO, J., INOUE, T., KOSEKI, Y., HORIUCHI, T., HAMATAKE, N., KUMAGAI, K., ENOMOTO, T., and YAMADA, H.: 'A 2.4 ns, 16-bit, 0.5 μm, CMOS arithmetic logic unit for microprogrammable video sequence processor LSIs'. Proceedings of IEEE CICC, May 1993, pp. 12.4.1–12.4.4
13 IEEE Std 1180-1990: 'IEEE standard specifications for the implementation of 8×8 inverse discrete cosine transform'