Hardware-efficient implementations for discrete function transforms using LUT-based FPGAs

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

Abstract: The multiplier-free design of transforms implemented in LUT-based FPGAs is presented. To fit bit-level grain size in the FPGA device at algorithm level the authors use modified distributed arithmetic (DA) and a named adder-based DA to formulate bit-level transform expressions, then they further minimise hardware cost by the proposed vertical subexpression sharing. For implementation, the required input buffer design is also considered by employing FPGA device characteristics and cyclic formulation. The proposed design can offer savings in excess of two-thirds of hardware cost compared with ROM-based DA.

1 Introduction

Transforms are widely used in digital signal processing applications, such as multimedia, wireless and communication systems, as basic computation units for further processing. Transforms such as discrete cosine transform (DCT) [1] compute multiple inner products and the data inputs to the transforms are independent of each other. Direct computation of inner products often uses multipliers. However, since the multiplier costs too much in terms of silicon area, a more efficient way is to use distributed arithmetic (DA) [2], ROM-based DA, to implement these fixed coefficient computations with ROM tables. Due to its bit-level reformulation DA is potentially suitable for the fine grain architecture of FPGA hardware.

Field programmable gate arrays (FPGAs) are modern logic devices that can be programmed by the users to implement their own logic functions, in which the look-up table (LUT)-based architecture is the most popular one. A FPGA chip consists of many configurable k-LUT's (called CLB in Xilinx FPGA chips) that can implement an arbitrary function with up to k inputs. For example, in Xilinx XC4000 series, k is equal to 5, and equivalent to memory size of $16\text{w} \times 2\text{b}$ [3]. However, such limited memory size constrains the available filter tap numbers to implement ROM-based DA in a single FPGA chip. Besides, the routing channels in FPGA are often very limited. These limitations often result in inefficient hardware utilisation of the FPGA chip for ROM-based DA.

To eliminate these drawbacks we adopt a modified DA formulation, called adder-based DA, first proposed in [4], to implement these transforms and filtering methods on FPGA chips. The adder-based DA preserves the advantages of the bit-level computation held by the ROM-based DA, but uses an adder network instead of ROM to on-line compute the summation. This bit-slice datapath logic is more suitable for FPGAs and can efficiently utilise the FPGA routing channels.

Although it has no advantage precomputation and storage in ROM, the adder-based DA has the benefit of sharing by scheduling the computation. Here we adopt and further modify the concept of common subexpression sharing [5–7] to efficiently minimise the numbers of adder. Common subexpression sharing shares common addition and subtraction between different constant coefficient multiplications. Previous techniques based on word-level are not suitable for bit-level grain FPGA device. Thus, we use DA bit-level input and adopt bit-level sharing, i.e. vertical subexpression sharing. With these combining techniques, inner products can be hardware efficiently implemented in the FPGAs.

Though the above techniques can efficiently reduce hardware cost, transform design still needs special consideration towards its input interface circuits. These interface circuits, such as input delay chains and parallel to serial (P/S) converters, still entail large area costs for transform designs. Two techniques are proposed to overcome this problem. In the first technique we propose a skewed buffer design to implement the P/S converter. In the second, for transforms that can be formulated to be filter-like operations, we proposed a cyclic formulation to reformulate transform equations to filter-like operations to solve the problem.

2 Adder-based DA and subexpression sharing

In the following, we assume that the size of the inner product is L, word length of the variable input X is W_x and word length of the coefficient A is W_c.

2.1 Review of adder-based DA

The conventional DA, ROM-based DA, decomposes the variable input X_i of the inner products into bit-level, that is

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

IEE Proceedings online no. 19990739

DOI: 10.1049/ip-cdt:19990739

Paper received 19th January 1999 and in revised form 18th August 1999

The authors are with the Department of Electronics Engineering, National Chiao-Tung University, 1001 Ta-Hsueh Rd, Hsinchu, Taiwan, R.O.C.

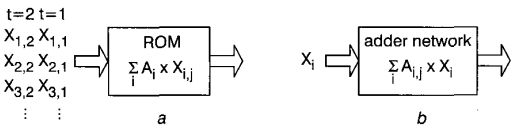


Fig. 1 *Computation of ROM-based DA and adder-based DA*
a ROM-based; *b* Adder-based

Without loss of generality we use unsigned fraction numbers to keep equations simple. The term $\sum_{i=1}^{L} A_i X_{i,k}$ is precomputed, stored in a ROM table and addressed by the input $X_{i,k}$

Our recently proposed adder-based DA [4], in contrast to the ROM-based DA, decomposes the coefficient A_i into bit-level and thus results in bit-slice equation, i.e.

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

Adder-based DA constructs an adder network to compute the term $\sum_{i=1}^{L} A_{i,j}X_i$. The major difference between the conventional ROM-based DA and our proposed DA is illustrated in Fig. 1. To reduce the routing requirement we reformulate the above equation into bit-serial form by decomposing S_j and exchanging the summation, where $S_j \equiv \sum_{i=1}^{L} A_{i,j}X_i$, i.e.

$$y = \sum_{j=1}^{W_c} S_j 2^{-j} = \sum_{j=1}^{W_c}\left(\sum_{t=1}^{P} S_{j,t}2^{-t}\right)2^{-j} = \sum_{t=1}^{P}\left(\sum_{i=1}^{W_c} S_{j,t}2^{-j}\right)2^{-t} \quad (3)$$

The bit-serial design accumulates and shifts the term $\sum_{i=1}^{W_c} S_{j,t}2^{-j}$ at each cycle t to obtain the inner product. The implementation of eqn. 3 depends on both L and W_x. Adder-based DA only computes the non-zero bits. Designs with fewer nonzero bits will save more area.

2.2 Review of previous common subexpression sharing approaches

Computation of adder-based DA can be further minimised by common subexpression sharing [5–7] which shares the common subexpression among several multiplication-accumulation operations so that total operation count is reduced. For example, Fig. 2 shows the FIR filter coefficients represented by the canonical signed digit (CSD). The circled groups of digits have the same subexpression. The filtering operation is

$$y = x[0] + x[0] \ll 1 + x[0] \ll 3 + x[1]$$
$$+ x[1] \ll 2 + x[2] + x[2] \ll 1, \quad (4)$$

	2^3	2^2	2^1	2^0
H_0	1	0	1	1
H_1	0	1	0	1
H_2	0	0	1	1

Fig. 2 *A common subexpression example*

where $x[a] \ll b$ denotes 'a' sample delay and 'b' digit left shifts of x. To illustrate the maximum possible sharing we adopt the following sharing pattern. If we define

$$w[i] = x[i] << 1 + x[i+1], \quad (5)$$

we can rewrite the filtering operation as

$$y = w[0] + w[1] + x[0] + x[0] << 3 + x[2]. \quad (6)$$

Thus, by sharing the common subexpression $w[i]$, the number of additions is reduced from six to four.

The drawbacks of previous common subexpression sharing techniques are: longer delay word length, additional intermediate delays and word-level input. Longer I/O tap delay word length is due to the transposed direct form architecture. Additional intermediate delays that are due to sharing between different delayed version of input will easily offset the advantages of subexpression sharing. Word-level input results in word-level subexpression, which is not suitable for FPGA.

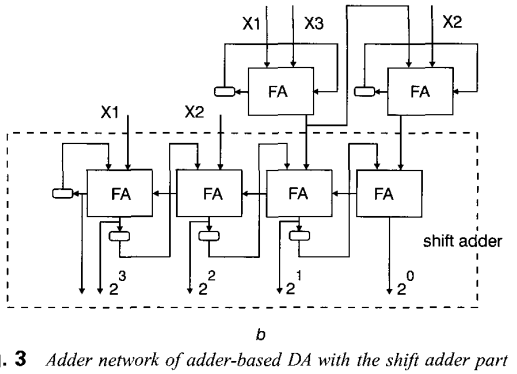
2.3 The proposed vertical subexpression sharing

To eliminate the above drawbacks we propose vertical subexpression sharing. Fig. 3 shows an example of vertical subexpression sharing, where the corresponding computation is $Y = X1 \times 1011_2 + X2 \times 0101_2 + X3 \times 0011_2$. With adder-based DA we can formulate bit-level output $Y[3] = X1$, $Y[2] = X2$, $Y[1] = X1 + X3$, $Y[0] = X1 + X2 + X3$, where Y[i] is the i-th bit of Y and each input to the expression is bit-serial input. Now with vertical common subexpression sharing $(X1 + X3)$ between $Y[1]$ and $Y[0]$, we get the final adder network design as shown in Fig. 3. In this figure the carry output of each FA in adder network is routed back to its own carry input for bit-level operation. Vertical sharing ensures bit-level computations and communications and is suitable for bit-level grain FPGA. Besides, with vertical sharing, summation of the tap result is shared with the computation of tap multiplication. Thus lower total hardware cost is attained.

To show the advantage of the proposed method we use the library data of XC4000E-3 [3] for delay and area calculations. For the same example with 8-bits input the hardware cost to implement Fig. 3 by using the proposed approach is three CLBs and the delay time is 6.25ns cycle time with eight computation cycles for one output. The

	2^3	2^2	2^1	2^0
H_0	1	0	1	1
H_1	0	1	0	1
H_2	0	0	1	1

a



b

Fig. 3 *Adder network of adder-based DA with the shift adder part*
a Common subexpression
b Adder network

word-level design in Fig. 2, using the previous approaches, uses 26 CLBs and needs 36.5ns cycle time and one cycle to compute one output. Both designs use ripple carry adders for fair comparison and minimum cost. Compared with previous approaches, the proposed method can achieve lower area-time complexity for the bit-level FPGA environment.

3 FPGA-based transform design

3.1 DA architecture for transforms

To simplify the explanation, we will use the 1-D 8-point DCT shown in Fig. 4 as our transform example. This architecture can also be made suitable for the ROM-based DA by replacing the adder network with ROM. The input buffer or P/S converter is a shift register chain, which converts the word-serial bit-parallel input to word-parallel bit-serial output. The computation result of the adder network is out through an accumulator to get the final product.

In the following, we will use the Xilinx XC4000-3 [3] to calculate the hardware cost and delay.

3.2 Adder network design

The structure of the adder network is like that in Fig. 3. To implement the adder network, half of the CLB is used for one bit adder and the output D flip-flops (DFFs) in each CLB store the carry output of the bit-serial adder. Most of the interconnections in the adder network are locally connected due to its tree structure.

The hardware saving due to the adder network depends on the coefficients. For the 1-D DCT example, we first use the split kernel method, i.e.

$$\begin{bmatrix} Y_0 \\ Y_2 \\ Y_4 \\ Y_6 \end{bmatrix} = \begin{bmatrix} \cos 4\theta & \cos 4\theta & \cos 4\theta & \cos 4\theta \\ \cos 2\theta & \cos 6\theta & -\cos 6\theta & -\cos 2\theta \\ \cos 4\theta & -\cos 4\theta & -\cos 4\theta & \cos 4\theta \\ \cos 6\theta & -\cos 2\theta & \cos 2\theta & -\cos 6\theta \end{bmatrix} \times \begin{bmatrix} X_0 + X_7 \\ X_1 + X_6 \\ X_2 + X_5 \\ X_3 + X_4 \end{bmatrix} \tag{7}$$

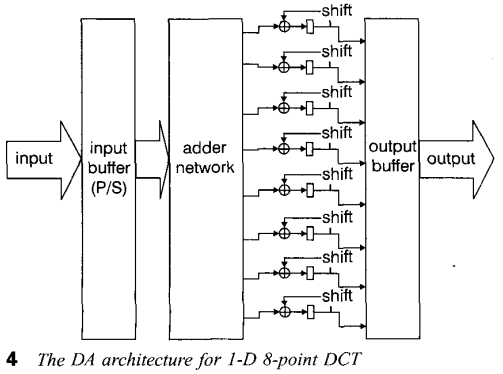


Fig. 4 *The DA architecture for 1-D 8-point DCT*

$$\begin{bmatrix} Y_1 \\ Y_3 \\ Y_5 \\ Y_7 \end{bmatrix} = \begin{bmatrix} \cos\theta & \cos 3\theta & \cos 5\theta & \cos 7\theta \\ \cos 3\theta & -\cos 7\theta & -\cos\theta & -\cos 5\theta \\ \cos 5\theta & -\cos\theta & \cos 7\theta & \cos 3\theta \\ \cos 7\theta & -\cos 5\theta & \cos 3\theta & -\cos\theta \end{bmatrix} \times \begin{bmatrix} X_0 - X_7 \\ X_1 - X_6 \\ X_2 - X_5 \\ X_3 - X_4 \end{bmatrix} \tag{8}$$

where $\theta = \pi/16$, X_i is the input sequences and the Y_i is the output sequences. Expanding the coefficients into bit-level, scaling with $1/\sqrt{2}$, and combining with vertical subexpression sharing, we can get only 16 addition terms for total computation. The scaling factor is easily removed if the 1-D DCT is applied to 2-D DCT.

The vertical subexpression sharing of DCT coefficients is shown in Fig. 5 for each output. From the tables, the network for output Y_0, Y_2, Y_4 and Y_6 needs six bit-serial adders and the network for output Y_1, Y_3, Y_5 and Y_7 needs 10 bit-serial adders. The common subexpression sharing in this transform is different from that in the filter shown in the previous section. First, there is no relationship between the inputs so we cannot share the similar addition patterns between different inputs, such as the two terms of Y_0 in Fig. 5. Second, we use the adder-based DA formulation, so all the shared terms are in a vertical direction, and the remaining nonzero bits will be directly fed into the final shift-adders without using more adders. This will embed the summation of each tap result in the sharing term and thus save hardware.

Table 1 shows the hardware comparisons of two DA methods for the DCT example. The adder network cost of the proposed design is reduced to just 12.5% of the cost of ROMs in ROM-based DA. The savings mainly come from the bit-level formulation and subexpression sharing. Also, the two DFFs in the CLB are very useful to construct the bit-serial adders without increasing cost.

The delay time of ROM-based DA design including the final accumulator is 10.4ns from the ROM address input to the accumulator output. The delay time of adder-based DA with vertical subexpression sharing is 8.35ns from network input to the accumulator output. The cycle numbers of the two designs are the same for the case with the same input and output precision. The proposed design is faster.

3.3 Hardware cost analysis

Table 2 shows the hardware cost comparisons of different approaches for transform length N, where K is the address numbers of each partitioned ROM, B is word length of the ROM table ($B \geq W_x$) and $\lceil . \rceil$ is the ceiling function. We choose $K = 5$ since one CLB can implement 32w × 1b RAM. Only the cost of the adder network and the ROM table is shown here, since other parts are identical. The hardware cost of the adder network, which varies according to coefficient distribution, is estimated by assuming uniformly distributed coefficients.

Fig. 6 shows that the proposed adder network requires only 4% of the ROM table cost. Exponential growth of the ROM table cost inhibits its applicability. Fig. 7 shows that the proposed network with vertical subexpression sharing can reduce the hardware cost by up to 70% compared with that without sharing. Compared with previous subexpression sharing, Fig. 8 shows that up to 70% of hardware cost can be saved, in which the proposed design has combined the P/S converter cost for fair comparison. The main

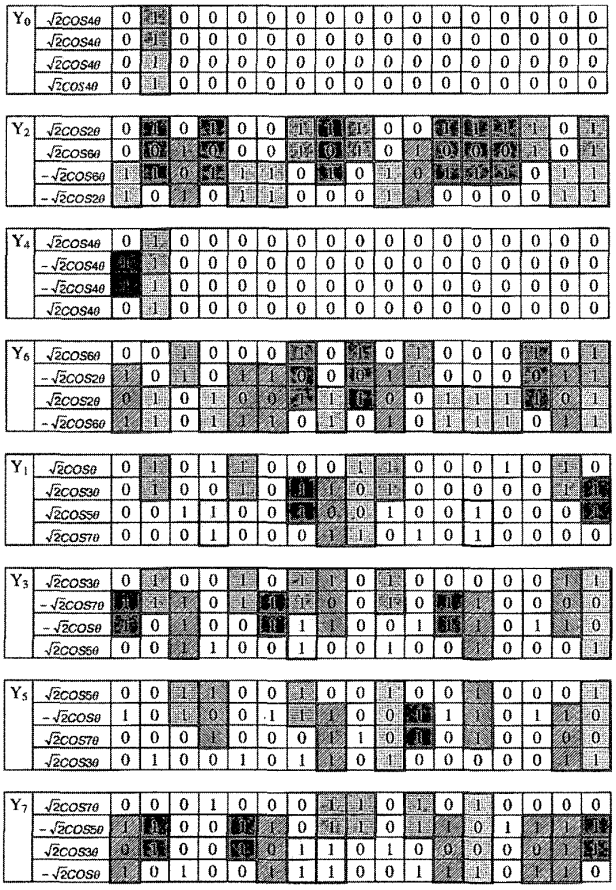


Fig. 5 *Common subexpressions of DCT computation*

Table 1: Hardware comparisons of two DA methods for 1-D split kernel DCT

Method	ROM-based DA	Adder based DA	
	ROM table	network without vertical subexpression sharing	network with vertical subexpression sharing
Hardware Item	*eight* $16W \times 16$ bit memory	*102 bit-serial adders for* two networks	*16 bit-serial adders for the* two networks
CLB no. in Xilinx 4000 series	1 CLB = $16W \times 2$ bit Total $8 \times 8 = 64$ CLBs	1 CLB = 2 bit-serial adders Total 51 CLBs	1 CLB = 2 bit-serial adders Total 8 CLBs

The two methods use 16-bits precision

Table 2: Hardware cost comparisons of adder network and ROM table using different approaches

Method		CLB number
ROM-based DA		$N \times (\lceil N/K \rceil \times 2^K \times B/32 + (\lceil N/K \rceil - 1) \times B/2)$
Previous sharing		$N \times W_x/2$ for $W_c = 8$, $N \times W_x$ for $W_c = 16$
Adder-based DA	without sharing	$\lceil W_c/2 \rceil \times N/2$
	with sharing	$(N + W_c)/2$ for $W_c = 8$, $(2 \times N + W_c)/2$ for $W_c = 16$

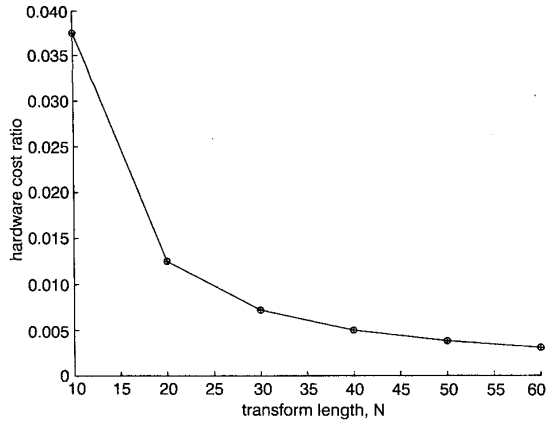


Fig. 6 *Hardware cost ratio of adder network with vertical subexpression against ROM table*

+ $W_X = W_C = 8$
○ $W_X = W_C = 16$

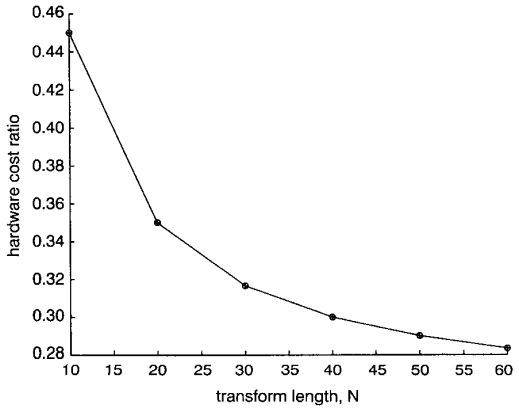


Fig. 7 *Hardware cost ratio of adder network with vertical subexpression sharing against adder network without vertical subexpression sharing*

+ $W_X = W_C = 8$
○ $W_X = W_C = 16$

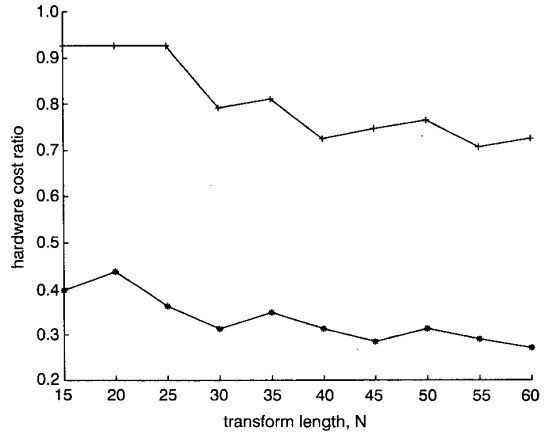


Fig. 8 *Hardware cost ratio of adder network with vertical subexpression sharing against adder network with common subexpression sharing*

+ $W_C = W_X = 8$
* $W_C = W_W = 16$

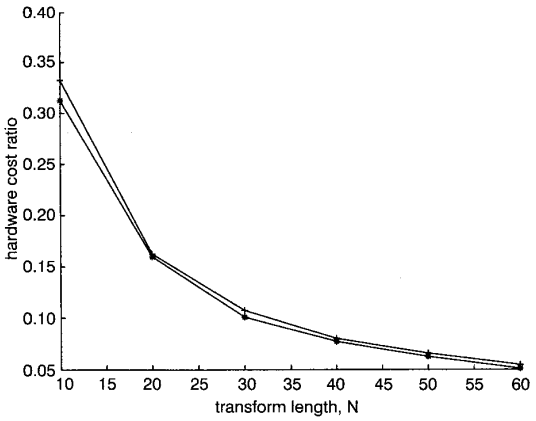


Fig. 9 *Hardware cost ratio between proposed design and ROM-based DA*

+ $W_C = W_X = 8$
* $W_C = W_X = 16$

savings come from the DA bit-level formulation instead of word-level one.

The hardware complexity of peripherals in the proposed transform design is: P/S converter: $W_x/2 \times \lceil \log_2 W_x \rceil \times (\lceil N/W_x \rceil + 1) + \lceil N/W_x \rceil \times W_x/2$; accumulation adder: $N \times B/2$; and S/P converter: $N \times W_x/2$. The P/S converter used the skewed buffer design introduced later and controller cost is ignored. Fig. 9 shows the overall hardware cost ratio between the proposed design and ROM-based DA, where B is assumed to be W_x and has the same partition strategy as previous paragraph. At least two-thirds of the hardware cost can be saved.

4 Interface circuit design

4.1 Input buffer (P/S converter) design with skewed buffer approach

In previous designs [8–10] of P/S converter, direct implementation of P/S converter with DFFs seems to be the only way for FPGA based transform designs. Thus, they can not be efficiently implemented with RAM, as proposed in [11, 12], for filters. However, DFF implementation is very inefficient since DFF is a very scarce resource in FPGAs (only two DFFs in one CLB).

Fig. 10 shows the proposed skewed buffer design. To illustrate the process, Fig. 11 shows the RAM contents for $L = 4$ and $W_c = 4$. Each small block surrounded by double lines is one RAM bank implemented by a CLB. The 4×4 regions constitute one RAM block in Fig. 10. At the initial point, input is first rotated shifted by a W_x-bits barrel shifter, and stored in the RAM banks. These inputs are skewed by 1-bit. Then the RAM content is accessed by a diagonal line order to implement P/S conversion. After the content is read out, new input can be written to the same address, since old data is not used any more. Half-cycle read and half-cycle write can perform this read-write operation. The read out data is rotated to match the input

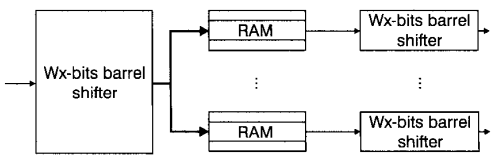


Fig. 10 *The skewed buffer design*

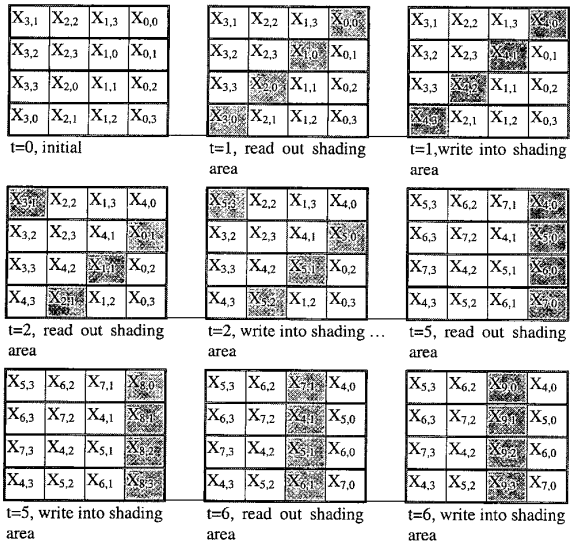


Fig. 11 *RAM contents on different time slots, where $X_{i,j}$ is the j-th bit of the i-th input*

of the adder network. The address generation for each RAM is just a simple counter and can be easily implemented by SRAM-based CLB. Thus, with this architecture, we can implement P/S converter with RAM instead of DFFs.

The cost of the skewed buffer design includes the barrel shifter: $W_x/2 \times \lceil \log_2 W_x \rceil$, and RAM: $W_x/2$. While for direct DFF-based design, P/S converter requires $N \times W_x/2$ CLB for equal throughput as a skewed buffer design. Fig. 12 shows that at least half the CLB count can be saved by using the presented approach.

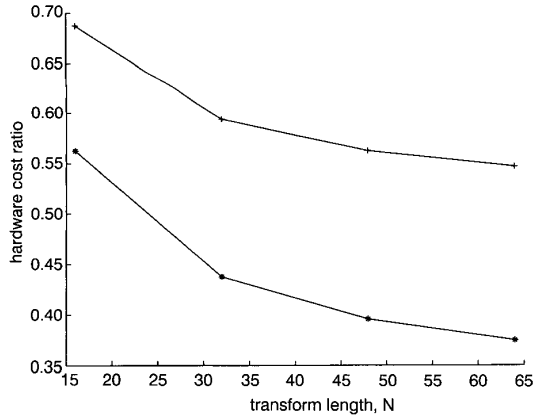


Fig. 12 *Hardware cost ratio of P/S converter design with skewed buffer versus DFF-based design*

\+ $W_X = 8$
* $W_X = 16$

4.2 Transform design with cyclic formulations

By creating pseudo–input delay dependency, P/S converter design of the transform can be efficiently implemented by RAM-based delay. This can be done by reformulating transform equations into cyclic convolution form. Let's take 1-D 5-point DFT as an example. The input sequence $\{x(n), n = 0,1,2,3,4\}$ is expressed as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 \\ 1 & W^2 & W^4 & W^6 & W^8 \\ 1 & W^3 & W^6 & W^9 & W^{12} \\ 1 & W^4 & W^8 & W^{12} & W^{16} \end{bmatrix} \times \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \end{bmatrix} \quad (9)$$

where $W = \exp(-j2\pi/5)$. By using the approach in [13], a cyclic convolution form can be expressed as:

$$Y(0) = \sum_{n=0}^{4} x(n)$$

$$\begin{bmatrix} Y(2) \\ Y(4) \\ Y(3) \\ Y(1) \end{bmatrix} = \begin{bmatrix} x(1) & x(2) & x(4) & x(3) \\ x(3) & x(1) & x(2) & x(4) \\ x(4) & x(3) & x(1) & x(2) \\ x(2) & x(4) & x(3) & x(1) \end{bmatrix} \times \begin{bmatrix} W^2 \\ W^4 \\ W^3 \\ W^1 \end{bmatrix} + \begin{bmatrix} x(0) \\ x(0) \\ x(0) \\ x(0) \end{bmatrix} \quad (10)$$

Fig. 13 shows the transform architecture with cyclic formulation. This design is also an adder-based DA design but with as added I/O permutation stage to reorder the input for cyclic convolution. Thus, the P/S converter can be efficiently implemented by RAM-based CLB technique as proposed in [11,12], where one bit in RAM can be a delay without extra circuitry.

The overall cost for a prime length DFT is

Input permutation: $\lceil (N-1)/16 \rceil/2 \times W_x$,
P/S converter: $(N-1) \times \lceil W_x/16 \rceil/2$,
Adder network: $(N + W_c)/2$ for $W_c = 8$-bits, $(2 \times N + W_c)/2$ for $W_c = 16$-bits
Accumulation adder: $B/2$,
Output permutation: $\lceil (N-1)/16 \rceil/2 \times W_x$,
Accumulation adder for $Y(0)$: $B/2$,

where B is the output word length ($B \geq W_x$), control cost is ignored and the adder network cost is a typical estimation. Fig. 14 shows that the proposed design can save at least two-thirds of the hardware cost due to saving in a P/S converter and accumulation adder. The cost of this design is also lower than that of the skewed buffer approach. However, this architecture has longer latency because of block input of transform and bit-serial input. The latency for one block is $(2N-1) \times W_x$ for the architecture, and is W_x for direct implementation. The latency can be reduced by bit-parallel input.

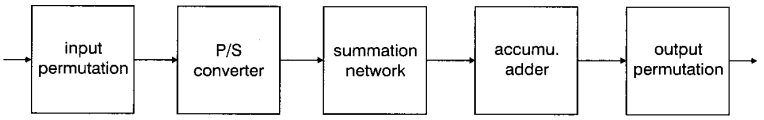


Fig. 13 *Transform architecture with cyclic formulation*

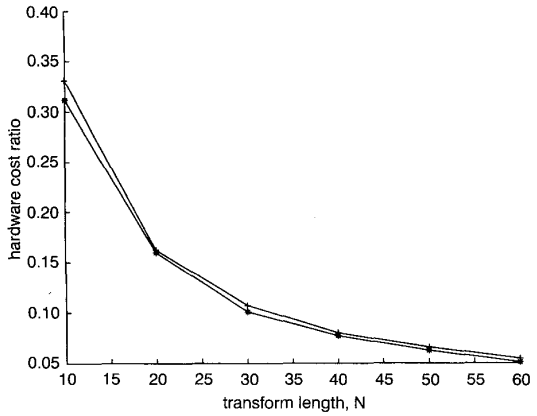


Fig. 14 *Ratio of total hardware cost with cyclic formulation against that with direct implementation*

$+ \ W_C = W_X = 8$
$* \ W_C = W_X = 16$

5 Conclusion

In this paper, we have proposed the hardware-efficient FPGA design and implementations for transforms by considering both algorithm and architecture level. At algorithm level, we use adder-based DA instead of ROM-based DA, and propose a modified common subexpression sharing technique for more hardware sharing. This leads to large savings. The bit-level algorithm formulation matches the FPGA bit-level gain feature well. As for interface considerations, by exploiting the features of FPGA architecture, we proposed two transform designs: a skewed buffer design and cyclic formulated transform. At least two-thirds of the hardware cost is saved, compared with ROM-based DA. For different tradeoff considerations, the first design is suitable for high-speed design while the second design is suitable for area-critical design.

6 Acknowledgments

This paper was supported by National Science Council, R.O.C., under the grant NSC-87-2215-E009-039.

7 References

1 RAO, K.R., and YIP, P.: 'Discrete cosine transforms - algorithms, advantages, applications' (Academic, Boston, MA, 1990)
2 WHITE, S.A.: 'Applications of distributed arithmetic to digital sequence processing: A tutorial review', *IEEE ASSP Mag.*, 1989, **3**, (6), pp. 5–19
3 Xilinx. The Programmable Logic Data Book. 1996
4 CHEN, C.-S., CHANG, T.-S., and JEN, C.-W.: 'The IDCT Processor on the adder-based distributed arithmetic', *Symp. VLSI Circ.*, 1996, pp. 36–37
5 HARTLEY, R.I.: 'Subexpression sharing in filters using canonic signed digit multiplier', *IEEE Trans.*, 1996, **CAS-43**, (10), pp. 677–688
6 DEMPSTER, A.G., and MACLEOD, M.D.: 'Use of minimum-adder multiplier blocks in FIR filters', *IEEE Trans.*, 1994, **CAS-42**, (9), pp. 569–577
7 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.*, 1996, **CAD-15**, (2), pp. 151–165
8 DICK, C.H.: 'FPGA based systolic array architectures for computing the discrete Fourier transform', IEEE ISCAS, 1996, **(2)**, pp. 465–468
9 HERON, J., TRAINOR D., WOODS, R., FARGUES, M.P., and HIPPENSTIEL, R.D.: 'Image compression algorithms using re-configurable logic', 31st Asilomar Conference on *Signals, Systems & Computers*, 1997, **(1)**, pp. 399–403
10 BERGMANN, N.W., and CHUNG, Y.Y.: 'Video compression with custom computers' *IEEE Trans.*, 1997, **CE-43**, (3), pp. 925–933
11 BULL, D.R., and WACEY, G.: 'Bit-serial digital filter architecture using RAM-based delay operators', *IEE Proc. Circuits, Devices, Syst.*, 1994, **CDS-141**, (5), pp. 371–376
12 GOSLIN, G.R., and NEWGARD, B.: '16-Tap, 8-Bit FIR Filter Application Guide' (Xilinx Publications, 1994)
13 GUO, J.I., LIU, C.-M., and JEN, C.-W.: 'The efficient memory-based VLSI array designs for DFT and DCT', *IEEE Trans.*, 1992, **CAS-39**, (10), pp. 723–733