Highly Efficient, Limited Range Multipliers for LUT-Based FPGA Architectures

R. H. Turner and R. F. Woods

***Abstract*—A novel design technique for deriving highly efficient multipliers that operate on a limited range of multiplier values is presented. Using the technique, Xilinx Virtex field programmable gate array (FPGA) implementations for a discrete cosine transform and poly-phase filter were derived with area reductions of 31%–70% and speed increases of 5%–35% when compared to designs using general-purpose multipliers. The technique gives superior results over other fixed coefficient methods and is applicable to a range of FPGA technologies.**

***Index Terms*—Discrete cosine transform (DCT), multiplier-less multiplier blocks, poly-phase filters, reconfigurable multipliers, signed digit encoding.**

I. Introduction

In digital signal processing (DSP) algorithms, many fixed transforms such as the discrete cosine transform (DCT) do not require the flexibility of a general-purpose multiplier as the multiplicand has a limited number of values [1], [2]. In these cases, a constant coefficient multiplier (KCM) can be derived using a number of techniques such as distributed arithmetic (DA) [3], string encoding [4] and common subexpression elimination [5]. KCMs are particularly attractive for ROM-based field programmable gate arrays (FPGAs) such as the Altera Stratix [6] and the Xilinx Virtex [7], as all possible outcomes can be precomputed and stored, resulting in efficient solutions for digital filters [3] and digital downconverters [8]. These approaches, however, are applied in cases where the applications have fixed coefficients and cannot be used efficiently in applications where a limited number of coefficients is needed. To date, no method has been presented to tradeoff multiplier complexity efficiently with computational needs. A technique is presented here for designing "limited number" or reduced coefficient multipliers (RCMs) which exploit circuit redundancy, naturally created when mapping algorithms to FPGAs, to allow highly efficient implementations to be achieved.

The paper is organized as follows. Section II presents the basic concept leading to, in Section III, a general methodology for generating the most efficient RCM structures based on performance needs. In Section IV, the technique is applied to a two-dimensional (2-D) DCT circuit and a poly-phase filter. Conclusions are given in Section V.

II. Basic Concept of the Reconfiguration Mux

One approach to implementing "limited number" coefficient capability is to develop separate KCMs for each single coefficient and swap in and out the KCMs as needed, i.e., reconfigure. This process can be viewed as connecting individual circuits using a *mux* or *rc_mux* [9] which is conceptual and is not implemented. With reconfiguration times reaching 200 μs, however, this approach is proving prohibitive as this constitutes a downtime of 20 000 clock cycles for a digital signal processing (DSP) system with a clock rate of 100 MHz. In this brief, these *rc_muxes* are physically implemented by exploiting unused hardware thereby allowing circuit functionality to be changed without the negative impact of these reconfiguration times.

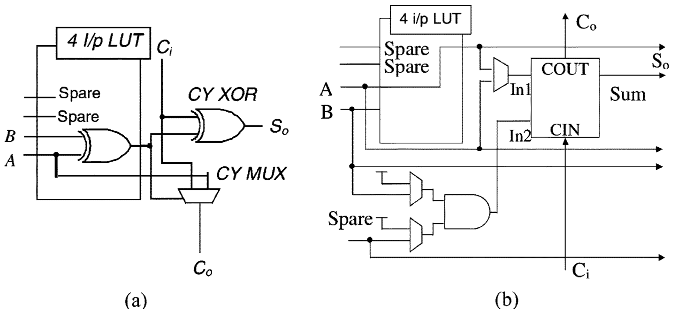


Fig. 1. Adder implementation in LUT-based FPGA technologies. (a) Xilinx Virtex FPGA slice. (b) Lattice XPGA cell.

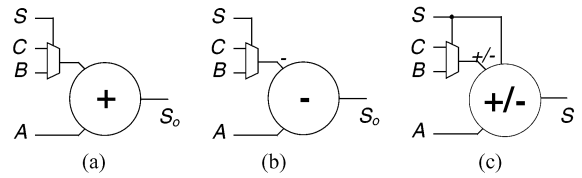


Fig. 2. Possible implementations using the *rc_*mux design technique. (a) Cell 1. (b) Cell 2. (c) Cell 3.

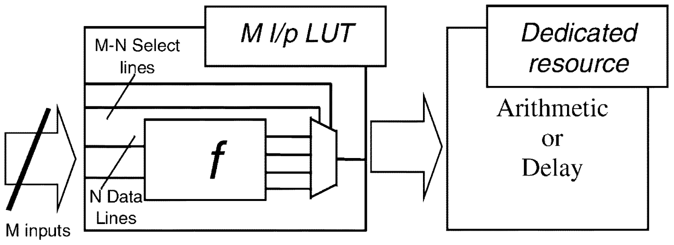


Fig. 3. Abstract FPGA cell view with *N* input functions mapped to an *M* input LUT and spare *M-N* inputs used for function selection.

A. Redundancy Creating by Mapping Process

Many FPGAs such as the Altera Stratix [6], the Xilinx Virtex [7] and the Lattice XPGAs [10] are composed of dedicated blocks of logic such as the fast carry logic, connected to LUTs as shown in Fig. 1 for the Virtex and XPGA technologies. Each circuit adds bits A, B, Ci and produces bits So and Co using the fast carry logic and, in the case of the Xilinx Virtex in Fig. 1(a), the 4-input LUT is used to implement the remaining XOR gate. This process occurs in synthesis tools which opt (quite correctly) to utilize the fast carry logic rather than implement the adder using the slower LUT hardware. These unused LUT inputs can be used to implement the *rc_mux* as shown in Fig. 2, thereby increasing cell functionality. For example, cell 3 can be used to implement either $A + B$ or $A - C$ using the select signal S.

This concept has been generalized in Fig. 3 for an M-input LUT cell and a dedicated resource. Whilst the resource is an adder in this example, Courtney [11] has demonstrated how the same approach can be used for cyclic redundancy checker (CRC) circuits but using a flip-flop as the dedicated resource. The key aim is to develop a technique which most efficiently uses these cells to create the required multipliers.

Manuscript received February 28, 2003; revised August 15, 2003. This work was supported in part a student grant from the Department of Education for Northern Ireland.

The authors are with the Institute of Electronics, Communications and Information Technology, Queen's University of Belfast, Belfast, Northern Ireland (e-mail: rht@qub.ac.uk; r.woods@qub.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2004.833399

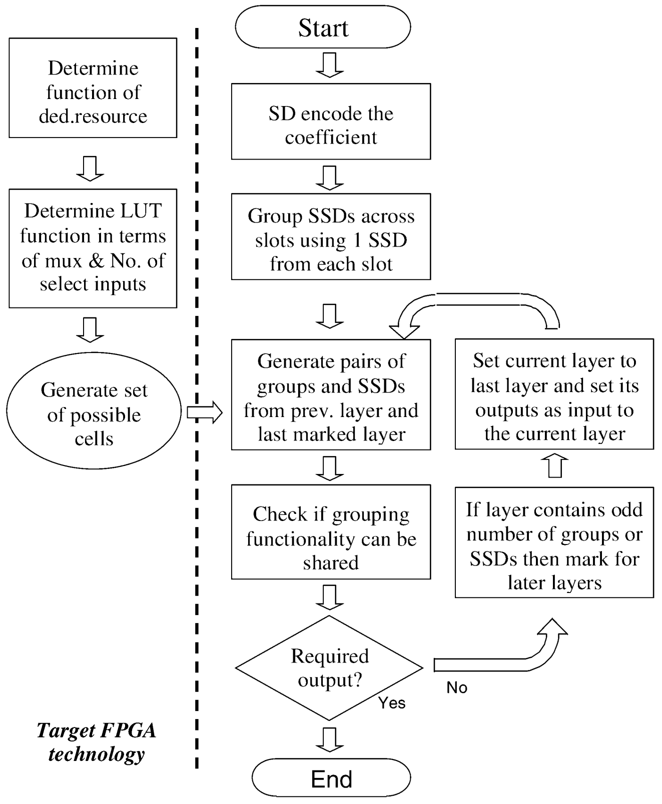

Determine function of ded.resource
Determine LUT function in terms of mux & No. of select inputs
Generate set of possible cells
Target FPGA technology
Start
SD encode the coefficient
Group SSDs across slots using 1 SSD from each slot
Generate pairs of groups and SSDs from prev. layer and last marked layer
Check if grouping functionality can be shared
Required output?
Yes
No
End
Set current layer to last layer and set its outputs as input to the current layer
If layer contains odd number of groups or SSDs then mark for later layers

Fig. 4. RCM general design methodology.

TABLE I
VIRTEX FUNCTIONALITY FOR 3 INPUTS (A, B, & C) AND CONTROL (S)

Symbol	Operation	
	#1	**#2**
1	A+B	A+C
2	A-B	A-C
3	A+B	A-C
4	B	A+C
5	-B	A-C
6	B	A-C
7	-B	A+C

III. RCM DESIGN METHOLOLOGY

The RCM design methodology (Fig. 4) involves two major steps. The first step is to generate a full set of possible cell configurations for the specific FPGA technology. This will depend on the number of LUT inputs and the dedicated hardware resource. The second step is the detection of terms that can be shared between coefficients. As signed digit (SD) encoding was found to give the consistently best results, the coefficients are SD encoded and the resulting shifted signed digits (SSDs) then grouped to develop the tree structure for the final RCM circuit. The process is finished when the required outputs are produced. A prototype C++ program was developed that was able to go through all the possibilities listed in Tables I and II and find all computable combinations thereby allowing the best implementation to be detected. This is described in more detail later.

A. Generation of FPGA Cell Definitions

This concept can be expanded beyond the list of cells given in Fig. 2, by considering more possible combinational inputs, M and select lines, N, as given in Table I. The additional AND gate in the Xilinx Virtex (Fig. 5) can be used to give direct control over the A input allowing additional operations such as 0 and 2 A to be implemented and giving the additional functions in Table II.

B. Development of RCM Circuits

The main objective of the mapping technique is to generate the RCM structure with a minimal number of cells. This RCM structure (Fig. 6) is formed by combining a number of data inputs, X_n in this case two, in a tree structure and producing a number of outputs, Y_n, in this case one.

The design process is illustrated using a single-input, single-output (SISO) example which multiplies an input X by four coefficients $a(0), a(1), a(2)$, and $a(3)$ which is given in Table III. This process is done in sequential fashion, so only one of the coefficients is required at any one time and is termed a *slot* where $a(0)$ is in *slot* 0, etc. The first step is to convert the coefficients into a SD representation (Table IV). In this representation, a $+$ indicates an addition, a $-$ means a subtraction and a subscript has been added to signify that each of these operate on the input data, X. This represents a different mapping than straightforward string encoding [4] and is part of the design technique.

The next phase involves grouping these SSDs to generate the tree structure which is influenced by the cell functionality. Groupings are made such that there is only one SSD from each slot which equates to the concept of selecting only one of the mux inputs to the circuits at any one time. The SSDs operate on the same input data, X, as indicated by the subscript in Table V. The focus is to combine columns together based on the cell computations. For example, columns 2^3, 2^1, and 2^0 can be mapped into cell 1 (in Fig. 7) which performs $A+B$ and $A-C$ where $A=2^3(+_X), B=2^1(+_X)$, and $C=2^0(+_X)$. The number within the cells in Fig. 7, refers to the cell functionality labeling given in Tables I and II. The cell "0" covers SSDs in columns 2^5 and 2^7. This gives the inputs to Layer 1 in Table V where the "X_n" indicates a "don't care" term and was generated because the cell output was 0. This is used to reduce the number of states that the cell has to cover in the next layer. The "2" cell with functionality of 12 in Table II, can cover terms in columns 2^5 and 2^9. Inputs to layer 2 are given in the third part of Table V. Finally the "3"cell can cover terms in columns 2^0 and 2^5 giving the multiplier structure shown in Fig. 7.

A 10-bit general-purpose multiplier with this specification would have required nine cells, but only four are needed here which is the same for the worse case KCM for $a(2)$. Table VI compares a "general purpose multiplier," a "four coefficient multiplier" implemented using a VHDL CASE statement to allow for optimization and the RCM. The circuits were synthesised using Synplify Pro [12] for a Virtex XCV300-4 PQ240. The reduction from design 1 to 2 shows the power of the synthesis tools but the RCM results are better because Synplify is unable to perform the optimization shown here.

A C++ software program was developed to perform the process just described which takes as it's inputs, the SD encoded coefficients and the library of the cells given in Tables I and II and generates the RCM in a number of formats including structural VHDL. The software avoids an exhaustive search because of knowledge about the RCM. First, the search is limited to a tree network which does not adversely affect the quality of the resulting circuit. Second, the problem is trimmed at each level because two groupings containing the same SSD cannot be combined, thereby avoiding an exponential growth in the number of groups to check. It only took a few seconds on a Pentium III 300 MHz machine, to derive the multiplier structure described earlier.

TABLE II
VIRTEX 1/2 SLICE FUNCTIONALITY FOR 2 INPUTS (A & B) AND CONTROL (S1&S2)

	Operation					Operation					Operation			
	#1	#2	#3	#4		#1	#2	#3	#4		#1	#2	#3	#4
8	A	0	2A	A-B	27	B	2A	-B	A	46	B	A	-A	A-B
9	-A	A	-B	A-B	28	0	A	B	2A	47	B	A+B	-A	0
10	0	2A	-A	A-B	29	0	A	-B	2A	48	0	A	-B	A-B
11	B	2A	0	A-B	30	A	2A	-B	A-B	49	-A	A	-B	A+B
12	A	0	A+B	A-B	31	B	2A	-B	0	50	B	A	-B	0
13	-A	0	-B	A-B	32	0	A	B	A+B	51	B	A+B	-A	A-B
14	0	2A	-B	A-B	33	0	A	-B	A+B	52	-A	A	-B	0
15	B	2A	-A	A	34	A	A+B	-A	A-B	53	-A	2A	-B	A+B
16	2A	A	A+B	0	35	B	2A	-B	A-B	54	B	A	-B	A-B
17	0	A	B	A-B	36	0	2A	B	A+B	55	B	A+B	-B	A
18	0	A+B	-A	A-B	37	0	2A	-A	A+B	56	B	2A	-A	A+B
19	B	2A	-A	0	38	A	A+B	-B	A-B	57	A	2A	-A	A+B
20	2A	A	A+B	A-B	39	B	A+B	0	A-B	58	B	0	-A	A-B
21	0	A	-A	2A	40	A	2A	B	A+B	59	A	2A	B	A-B
22	0	A+B	-B	A-B	41	0	2A	-B	A+B	60	B	2A	-B	A+B
23	B	2A	-A	A-B	42	B	A	-A	0	61	A	2A	-B	A+B
24	2A	0	A+B	A-B	43	B	A+B	-A	A	62	B	0	-B	A-B
25	0	A	-A	A+B	44	0	A	-A	A-B	63	A	A+B	B	A-B
26	A	2A	-A	A-B	45	-A	A	-B	2A					

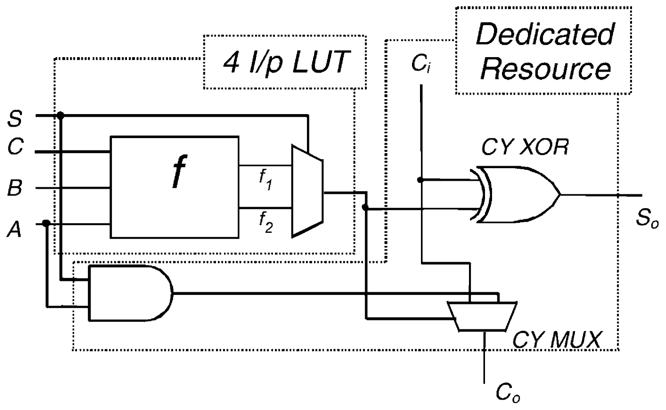


Fig. 5. Additional Virtex feature for LUT setup with $M = 4, N = 3$.

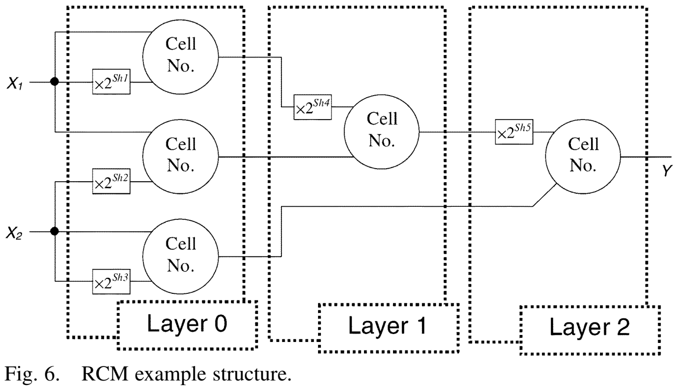


Fig. 6. RCM example structure.

IV. PRACTICAL APPLICATION OF RCM TECHNIQUE

A. Application to a SISO Application

An FPGA-based 2-D DCT implementation (Fig. 8) was explored based on an existing intellectual property (IP) core [13] where the DCT coefficients are given by

$$a(k) = 2\mathrm{Cos}\left(\frac{\pi k}{N}\right) \text{ and}$$

$$b(k) = \sqrt{\frac{2}{N}}a(k)(-1)^k \mathrm{Cos}\left(\frac{\pi k}{2N}\right)$$

TABLE III
FOUR COEFFICIENTS FOR A 4 POINT DCT

		2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
a(0)	-502	1	0	0	0	0	0	1	0	1	0
a(1)	473	0	1	1	1	0	1	1	0	0	1
a(2)	-426	1	0	0	1	0	1	0	1	1	0
a(3)	362	0	1	0	1	1	0	1	0	1	0

TABLE IV
SD RECODING OF COEFFICIENTS

		2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
a(0)	-502	-x						+x		+x	
a(1)	473	+x				-x		-x			+x
a(2)	-426	-x		+x		-x		-x		-x	
a(3)	362	+x		-x		-x		+x		+x	

TABLE V
INPUTS TO VARIOUS LAYERS FOR 4 CELL EXAMPLE

	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	#
Inputs to layer 0	-x						+x		+x		3
	+x				-x		-x			+x	4
	-x		+x		-x		-x		-x		5
	+x		-x		-x		+x		+x		5
Inputs to layer 1	-x				X_0					$+_1$	3
	+x				$-_0$					$-_1$	3
	-x				$-_0$					$-_1$	3
	+x				$-_0$					$+_1$	3
Inputs to layer 2					$-_2$					$+_1$	2
					$-_2$					$-_1$	2
					$-_2$					$-_1$	2
					$-_2$					$+_1$	2

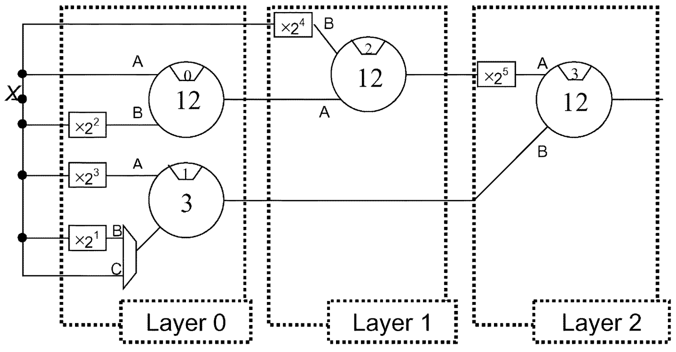


Fig. 7. RCM implementation using only four cells.

TABLE VI
COMPARISON OF A 12 × 16 BIT MULTIPLIER

No.	Description	Area (LUTs)	Speed (MHz)
1	General purpose multiplier	333	29.1
2	Four coefficient multiplier	193	27.1
3	RCM	86	43.4

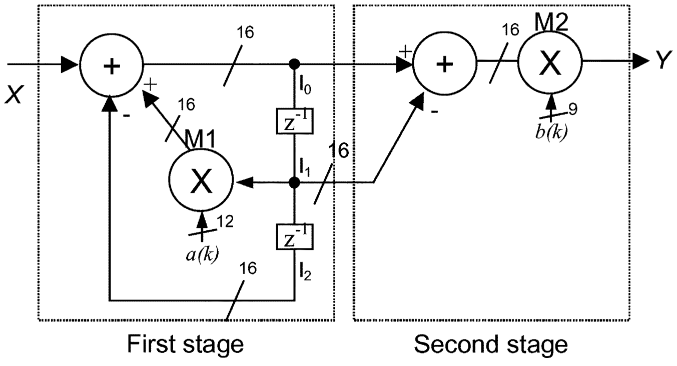


Fig. 8. DCT Core Block.

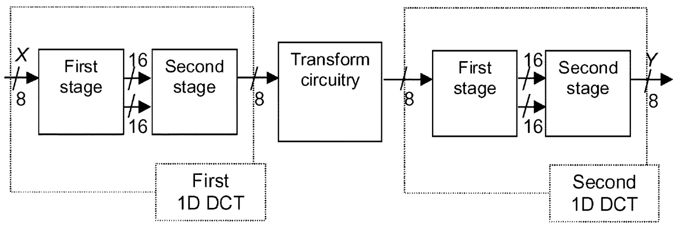


Fig. 9. First configuration.

Two configurations were investigated. The first (Fig. 9) represents a modular design approach. As eight samples are required before any data is produced, the previous seven output samples are ignored and this means that the second stage is only used once every eight cycles in the implementation. Thus, it is possible to time-share the second stage which represents the second design. Table VII gives the performance comparison between versions using the RCM and Xilinx's general-purpose multiplier [14].

B. Application to a SIMO Block

In a SIMO system, an additional optimization is possible by sharing common subexpressions used by different outputs. This has been termed *multiplier-less multiplier* blocks [2], [15] and the best reductions [1]–[3], [5], [15], [16] were found where common subexpressions were shared between a group of KCMs, so this approach is adopted here.

TABLE VII
DCT PERFORMANCE ON A XILINX VIRTEX XVC50 FPGA

Description	General-purpose		RCM	
	Area (LUTs)	Speed (MHz)	Area (LUTs)	Speed (MHz)
DCT (Fig. 9)	734	14.8	485	19.4
DCT second stage, time-shared	670	15.5	435	21.3

TABLE VIII
COEFFICIENTS REQUIRED AT THE DIFFERENT OUTPUTS

	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
a(0)	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
a(1)	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	0
b(0)	1	1	1	1	1	0	0	1	1	0	0	0	1	0	1	0
b(1)	1	1	1	1	1	0	0	0	0	1	0	1	1	1	0	0
c(0)	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	1
c(1)	0	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0

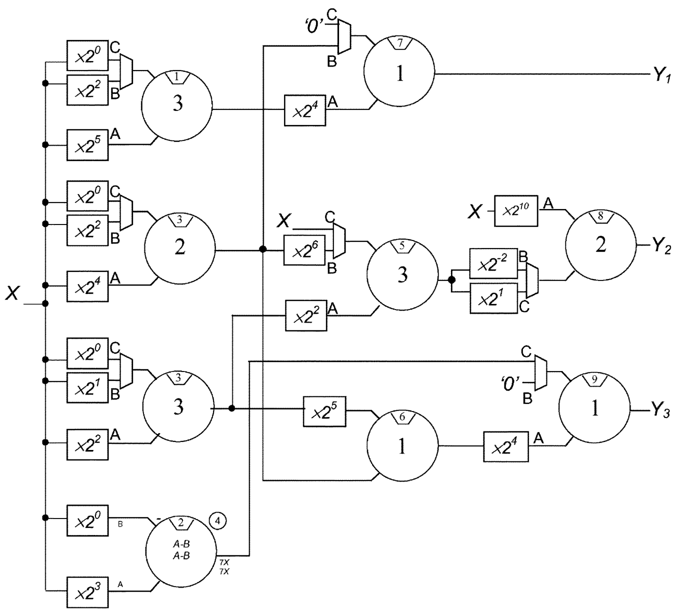


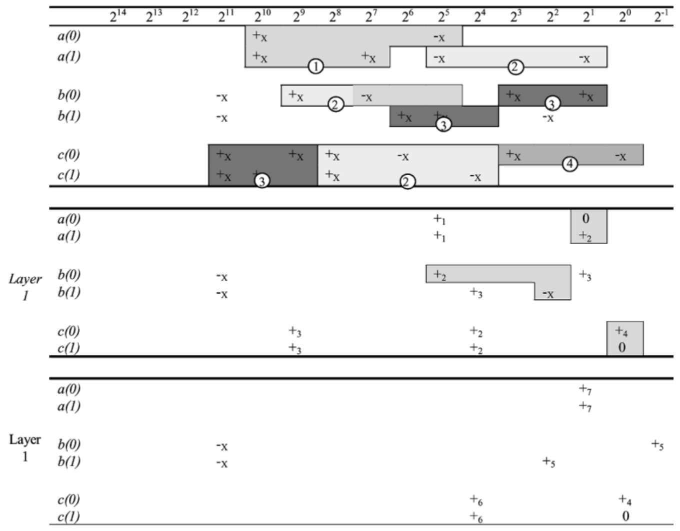
Fig. 10. Example of a multiplier block used in the filter.

The outputs Y_n are generated as shown by (1) where X is the input, vector Y_n is the output, S selects the slot and $a(S), b(S)$, and $c(S)$ are the coefficients. This forms part of the poly-phase filter example. Table VIII gives three sets of two coefficients that are used and the circuit is shown in Fig. 10. The design goal here is not only to share subexpressions within taps (as in the previous section) but to share terms between taps

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} a(S) \\ b(S) \\ c(S) \end{bmatrix} X \tag{1}$$

As before, the coefficients are SD encoded (Table IX) and the mapping process carried out starting with grouping of SSD's in the coefficients $c(0)$ and $c(1)$ as shown. The sub-expression sharing leads to four groupings two of which, 2 and 3, are identified in other coefficients. This leaves a pattern in $a(0)$ and $a(1)$ which is covered by grouping 1 and a pattern in $c(0)$ which is covered by grouping 4. The uncovered

TABLE IX
SD Encoded Coefficients and Groupings



		2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}
	a(0)					+x					-x						
	a(1)					+x			+x		-x				-x		
	b(0)				-x		+x		-x				+x		+x		
	b(1)				-x					+x	+x			-x			
	c(0)				+x		+x	+x		-x			+x			-x	
	c(1)				+x	+		+x				-x					
Layer 1	*a(0)*										$+_1$				0		
	a(1)										$+_1$				$+_2$		
	b(0)				-x						$+_2$				$+_3$		
	b(1)				-x							$+_3$		-x			
	c(0)						$+_3$					$+_2$				$+_4$	
	c(1)						$+_3$					$+_2$				0	
Layer 1	*a(0)*														$+_7$		
	a(1)														$+_7$		
	b(0)				-x												$+_5$
	b(1)				-x									$+_5$			
	c(0)											$+_6$				$+_4$	
	c(1)											$+_6$				0	

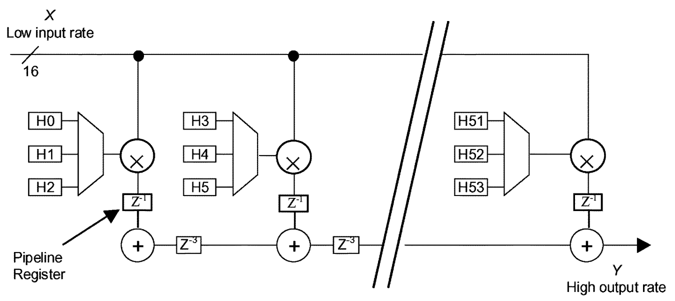


Fig. 11. Transformed poly-phase design.

TABLE X
Performance of four Design Approaches

	Clock Rate (MHz)	Throughput (MSample/s)	Area (LUTs/Slices)	TR/Area (Samples/s/LUT)
General-purpose	26.2	26.2	(5599/4123)	4500
RCM	28·0	28.0	(1661/1583)	16857
Hybrid ROM/adder	37·3	28.0*	(2199/1528)	12733
CoreGen (Bit serial DA)	44·5	2.6**	(416/235)	6250

* Produces a result every 4 cycles but operate 3 block in parallel
** Produces a result after 17 clock cycles

bits in $b(0)$ and $b(1)$ are considered in a later layer. These groupings form the first layer of cells in Fig. 10. The layer 1 groupings are made in Table IX. The shaded areas indicate that some groupings can be made across SSDs that do not have same input source. This is because cell 5 in Fig. 10, has a mux input which can accommodate this. Columns 2^9 and 2^4 in $c(0)$ and $c(1)$ are grouped together in cell 6 and columns 2^5 and 2^1 in $a(0)$ and $a(1)$ are grouped together in cell 7. The term in 2^0 in $c(0)$ and $c(1)$ is delayed to the second layer as shown.

The circuit in Fig. 10 is one component of a poly-phase filter (Fig. 11) that was designed using this technique. A full poly-phase filter implementation [17] was implemented that had an interpolation of 1 : 3 and a filter length of 55. This design was used as not all the filter taps are used in the computation of each output sample, therefore a number of taps time-share a resource.

Four implementations were investigated: a general-purpose design synthesised from VHDL using Synplify [12]; a design using hybrid ROM/adder KCM design blocks based on Kean [18] that the authors believe is very efficient; a bit serial DA implementation from Xilinx's CoreGen software and; the RCM-based design. The middle two designs were chosen as they were ROM-based. Table X gives performance figures for each of the designs and shows that the RCM implementation results in improved area and speed. As multipliers dominate filter area, the gain has been substantial.

V. Conclusion

A design methodology for developing RCMs that operate on a limited number of coefficient inputs, has been presented. Spare LUT capacity which naturally results from using the dedicated fast adder circuitry, is used to implement reconfigurable multiplexers. Area savings and speed improvement are achieved for a full 2-D DCT and a polyphase filter and the technique can be applied to a range of other transformations such as the fast Fourier transform (FFT) and Wavelet transforms.

The technique has been demonstrated for a Xilinx Virtex FPGA but it is possible to develop a generic library for a range of FPGA technologies as many FPGA technologies have the LUT/dedicated resource combination. Exploitation of detailed features of specific FPGA technologies, allows much greater flexibility in the library.

With the current Xilinx Virtex II FPGA architecture, it could be argued that the on-board multipliers could be used to implement these functions. However, this technique can be still be applied to the LUT and adder resource and leaving the on-board multiplier for other parts of the complex system, for in many cases, functions such as the DCT and poly-phase filter form only part of a complex system.

References

[1] A. Yurdakul and G. Dundar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *J.VLSI Signal Processing*, no. 22, pp. 163–172, 1999.
[2] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 569–577, Sept. 1995.
[3] G. R. Goslin, "Using Xilinx FPGAs to design custom digital signal processing devices," in *Proc. DSPX.*, Jan. 1995, pp. 565–604.
[4] A. R. Omondi, *Computer Arithmetic Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1994.
[5] B. Feher, "Efficient synthesis of distributed vector multipliers," in *Proc. 19th Symp. Euromicro Microprocessing Microprogramming*, 1993, pp. 345–350.
[6] Stratix Device Handbook. Altera Corp., San Jose, CA, USA. [Online]. Available: http://www.altera.com
[7] Virtex-E 1.8 V Field Programmable Gate Array (2002, Nov.). [Online]. Available: www.xilinx.com
[8] AN279 Digital Downconverter (DDC) Reference Design (2003, Apr.). [Online]. Available: http://www.altera.com/products/ip/dsp
[9] N. Shirazi, W. Luk, and P. Cheung, "Automating production of run-time reconfigurable designs," in *Proc. IEEE Symp. FCCM*, Apr. 1998, pp. 147–156.
[10] IspXPGA Data Sheet [Online]. Available: http://www.latticesemi.com/products/fpga/xpga/index.cfm
[11] T. Courtney, R. Turner, and R. Woods, "Mapping multi-polynomial parallel CRC circuits to virtex FPGA using embedded MUXes," in *ProcIEEE Symp. FCCM*, Apr. 2002, pp. 318–319.
[12] Synplify Pro: The Industry #1 FPGA Solution (2003). [Online]. Available: www.synpilicity.com
[13] J. Hunter and J. V. McCanny, "Discrete Cosine transform generator for VLSI synthesis," in *IEEE ICASSP*, vol. 5, 1998, pp. 2997–3000.
[14] Variable Parallel Virtex Multiplier (1999, Oct.). [Online]. Available: http://www.xilinx.com/ipcenter/catalog/logicore/docs/mult_vgen_v1_0.pdf
[15] D. Li, "Minmum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters," *IEEE Trans. on Circuits and Systems II*, vol. 42, pp. 451–460, July 1995.
[16] Q. Znao and Y. Tadokoro, "A simple design of FIR filters with power-of-two coefficients," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 5, pp. 566–570, May 1988.
[17] R. H. Turner, R. Woods, and T. Courtney, "Multiplier-less realization of a poly-phase filter using LUT-based FPGAs," in *Proc. on Field Programmable Logic*, Sept. 2002.
[18] T. Kean, B. New, and B. Slous, "A fast constant coefficient multiplier for the XC6200," in *Field Programmable Logic and Applications*, Darmstadt, 1996, pp. 230–241. Springer LNCS 1142.