IMPLEMENTATION OF FIXED DSP FUNCTIONS USING THE REDUCED COEFFICIENT MULTIPLIER

R. H. Turner, T Courtney and R Woods
School of Electrical and Electronic Engineering, The Queen's University of Belfast,
Ashby Building, Stranmillis Road, Belfast, BT9 5AH, Northern Ireland

ABSTRACT

Distributed Arithmetic (DA) has been successfully applied to the design of area efficient multipliers on FPGAs for DSP applications. Whilst DA is efficient in applications where coefficients are fixed, there is little option for applications with a limited range of coefficient values. This paper describes a technique for developing area-efficient multipliers for a range of DSP applications that fall into this category. This is accomplished by employing multiplexers at no extra cost to increase the functionality of existing fixed coefficient multipliers. The technique has been applied to a DCT FPGA implementation where an area decrease of up to 50% and speed increase of 33% was achieved over the conventional route.

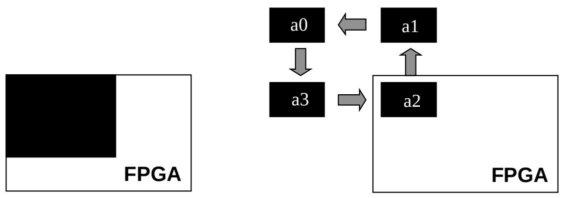
1. INTRODUCTION

FPGAs are an attractive platform for DSP implementation as they provide concurrency in the form of parallelism and pipelining and programmability. Research has centred around the efficient use of the on-board Look Up Tables (LUTs) common on many FPGAs such as the Xilinx Virtex FPGA series. High performance DA-based DSP implementations have been demonstrated for the FFT, digital filters [1], and the DCT [2].

DA exploits the feature of some DSP functions where fixed multiplication occurs by performing pre-computation of the possible results and storing them in the LUTs of the FPGA. It is highly attractive in applications where the coefficient or transform values do not change. For applications where the coefficient values are updated, fully programmable multipliers are typically required which are relatively expensive. To date, no method has been presented for low area structures that only need to multiply a limited range of coefficients. An efficient mechanism for achieving this objective, is presented here.

2. RECONFIGURATION MUX PRESPECTIVE

The use of fully programmable multipliers in DSP applications is expensive in FPGA area. Alternatively, some applications only require a fixed number of computations such as the multipliers used in computing the DCT where only a limited number of coefficients are required. Other examples include poly-phase filters used in up- or down-sampling and other fixed transforms such as the FFT. One solution is to derive highly efficient, fixed coefficient implementation for each case and swap these fixed circuits into the circuit as required. This is demonstrated in fig. 1 which shows how four separate processors computing *a1*, *a2*, *a3* and *a4* can be swapped in as required. The attraction of this approach is that the fixed processors will be smaller and faster than the fully programmable versions but the problem is the time required to swap in these separate functions, known as the reconfiguration time which can be prohibitive. This process only becomes efficient if the reconfiguration is not frequent which is not typical in DSP. Reference [3] suggests that hundreds if not thousands of samples need to be processed between reconfigurations to make this process worthwhile.



(a) Fully Programmable (b) Re-configurable processors

Fig. 1. General purpose versus fixed processors.

The design of re-configurable systems has been explored by Shirazi et al. [4] who presents a method for identifying parts of the circuit that can be reconfigured. This is achieved by transforming the circuit in such a manner that parts of the circuit, A and B, can be placed between a MUX and DEMUX as shown in fig. 2. Circuits A and B can be reconfigured as only one is required at any time. The MUX and DEMUX are conceptual and represent reconfiguration, hence the term reconfigurations MUXes.

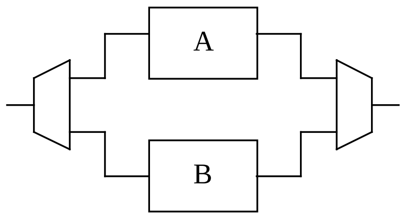


Fig. 2. Reconfiguration MUX and DEMUX.

In our approach, the reconfiguration MUXes have been implemented in the FPGA in a highly efficient manner by exploiting the redundant hardware on the FPGA that has occurred as a result of the normal mapping process. The availability of these muxes allows the efficient implementation of multipliers.

3. MULTIPLIER IMPLEMENTATION ON LUT-BASED FPGAS

Consider the implementation of a bit addition in the Virtex FPGA ½ slice shown in fig. 3 (The term "cell" is used for this figure in the paper). The implementation of an adder that adds *a*, *b*, *ci* and produces *so* and *co* is shown in fig. 4. The most efficient implementation uses the dedicated fast carry logic, labeled *CY XOR* and *CY* which means that two inputs of the LUTs are used to implement the $a \oplus b$ function.

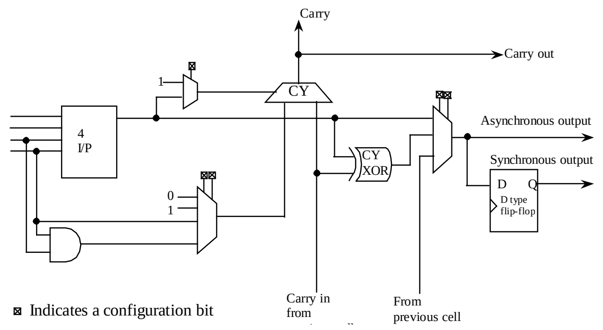


Fig. 3. One half of the Virtex slice (simplified diagram).

This can be extended to a single bit multiplication and addition as shown in fig. 5 which needs three inputs of the LUT. In cases where the coefficients are fixed, the circuit reduces to the form of fig. 4 where the wiring of the inputs defines the functionality. Connection of bit *a* in figure 4 implies *s*=1 whereas connection to 0 for one bit of the exor gate implies *s*=0.

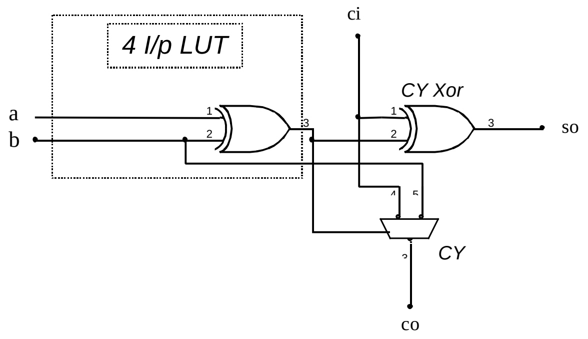


Fig. 4. Adder implementation in the Xilinx Virtex FPGA.

The implementation means that two available LUT inputs can be used to implement the some functionality, in our case, the MUX of fig. 2. This not only provides a mechanism for implementing the reconfiguration MUX but also increases the functionality of the cell as shown in fig. 6. Various functions can be implemented with this same logic without having to increase the area of the design or reconfigure the cell. The values *A1*, *B1*, *B2* and *S* are the four inputs to the LUTs. This form represents the general case for circuits of the type shown in figure 5. In most cases, the circuit area used for multiplication by one coefficient can be used to perform multiplication by a number of coefficients.

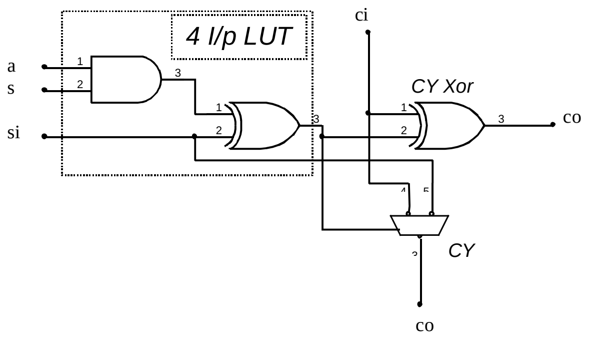


Fig. 5. Implementation of a single bit multiplication and addition

This basic concept of sharing terms has been described by Potkonjak et al. [5] and applied to filter design. However, their approach is to produce constant coefficient (multiplier-less) blocks for a fixed set of coefficients. Our approach is different in that it allows the coefficients to change during operation. This allows the cells to have more than one mode of operation and improves the FPGA utilization by using hardware that is already available on the FPGA as a consequence of the mapping process. Thus, there are a limited number of functions that can be implemented on the FPGA when the dedicated logic is fully utilized, as described in table 1.

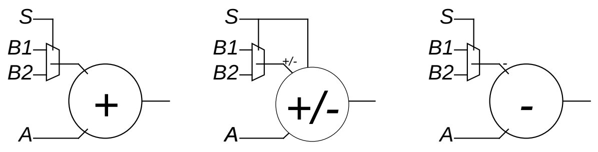


Fig. 6. Examples implementations using MUX-based technique.

A simple example of this technique is shown in fig. 7. This shows how multiplication by either 15 or 45 can be carried out using 2 cells by selecting different shifted partial products. The multiplexer in the second cell can also be used to add a shifted value of product ($P*2^3$), thereby providing multiplication by 23 (fig. 8). This technique provides a mechanism for implementing efficient multipliers that operate on a limited range of input values.

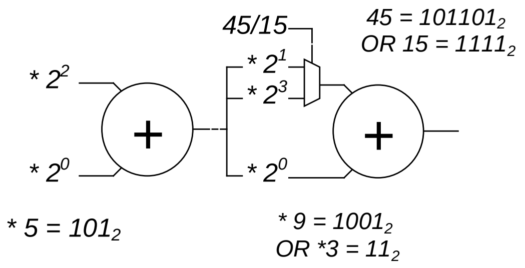


Fig. 7. Multiplication by either 45 or 15

S=0
S=1
	A+B1
	A+B2
		A-B1
		A-B2
			A+B1
			A-B2
				A-B1
				A+B2

S=0
S=1
	B1
	A+B2
		-B1
		A-B2
			B1
			A-B2
				B1
				A+B2

Table 1. Eight function combinations possible with a 4 input LUT with 2 available inputs.

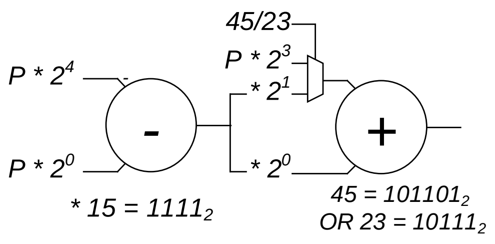


Fig. 8. Multiplication by either 45 or 23

4. PARAMETERISABLE DCT DESIGN

The technique has been applied to the Virtex FPGA implementation of an existing DCT IP core [6]. The 2D DCT is an important transform in many image based applications such as JPEG and MPEG video standards and is given as:

$$DCT_{2d}(k,l) = \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n,m).c(n,k).c(m,k) \qquad (1)$$

where *x(n,m)* is the input data and the values *c(n,k)= cos(2n+1)πk/2N* and *c(m,k)=cos(2m+1)πk/2N, (k, l= 0,..,N-1)* are the coefficients. The values *α(k)* and *α(l)* are scaling variables. The 2D DCT is separable and can decomposed into two 1D DCTs. Whilst efficient 1D DCT implementations based on matrix-vector representations are possible, Hunter [6] uses an alternative strategy based on a recursive implementation of the DCT that allows a parameterisable core to be developed. The 1D DCT for a sequence of input values, *x(n),* is shown in equation (2)

$$DCT(k) = \alpha(k)\sum_{n=0}^{N-1} x(n)\cos\frac{(2n+1)\pi k}{2N} \qquad (2)$$

where 0≤k≤N-1 and $\alpha(k)=1/\sqrt{N}$, except for $\alpha(0)$ ($\sqrt{2}/\sqrt{N}$). The computation with a sampled datastream $x(n)=[x(t), x(t+1),...x(t+N-1)]$ and k mapped between 0 and N-1 inclusive, is given as follows :

$$y_t(k,n)=C(k)\sqrt{\frac{2}{N}}\sum_{n=t}^{t+N-1} x(n)\cos\left[\left(n-t+\frac{1}{2}\right)\frac{k\pi}{N}\right] \qquad (3)$$

Using the z transform, equation (3) can be manipulated into a finite difference equation:

$$\begin{aligned} y[n] \;=\; & (-1)^k\alpha(k)\sqrt{\frac{2}{N}}\cos\left(\frac{\pi k}{2N}\right)[x[n]-x[n-1]] \\ + \; & 2\cos\left(\frac{\pi k}{2N}\right)y[n-1]-y[n-2] \end{aligned} \qquad (4)$$

The equation above is equivalent to a 2nd order direct form realisation of an IIR filter with a time varying amplitude. This schematic for this representation is given in Fig. 9. The original DCT implementation is based on the specification in [6] and processes monochrome images. The M_1 and M_2 coefficients are 12 and 10 bits and given as $-2\cos\left(\frac{\pi k}{N}\right)$ and $\alpha(k)\sqrt{\frac{2}{N}}\cos\left(\frac{\pi k}{2N}\right)$ respectively.

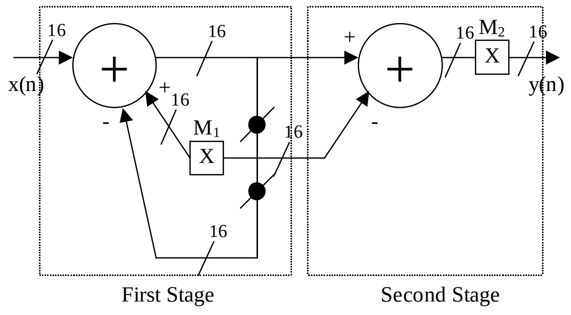


Fig. 9. Canonical DCT Core Block

Four configurations were investigated. The first represents a modular design approach. In the implementation of figure 9, eight samples are required before any data is produced. Thus, the previous 7 output samples are ignored and means that the second stage is only used once every 8 cycles in the implementation. Thus, it is possible to time-share the second stage as shown in figure 11.

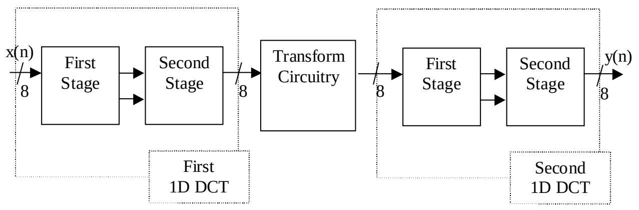


Fig. 10. First configuration

Another two implementations to allow the 8 point DCT to be split into two 4 point DCTs based on the Hou algorithm [7], were also explored (fig. 12). In this figure, two samples (one odd, one even) are supplied in parallel and two outputs are generated, the first and second 4 outputs respectively. This means that the multipliers are only required to cover 4 rather than 8 coefficients. The disadvantage of this approach is the extra circuit required.

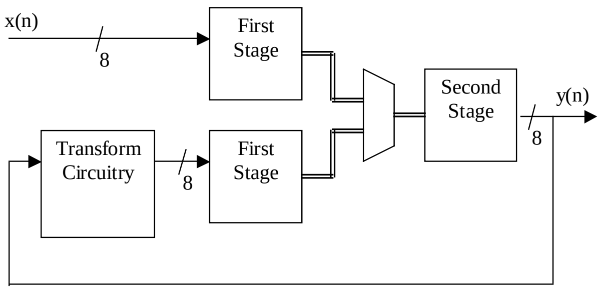


Fig. 11. Second configuration

The technique was applied to a number of 8 point 2D DCT implementations and compared to same implementation achieved using the standard Xilinx tools. Table 2 gives the performance comparison (No of slices are given in brackets for the last two designs). Our approach (*KCM*) consistently gives an area decrease of either 33% or 50% and speed increase of 33%.

Design		Area (LUTs)	Speed (ns)
1. 8 point DCT (fig 10) with 8 dual port memories	Xil	734	67.3
	KCM	485	51.5
2. 8 point DCT (fig 11) with 8 dual port memories (second stage, time-shared)	Xil	670	64.6
	KCM	435	47.0
3. 8 point HOU DCT using two 4 point DCT with 16 dual port memories	Xil	1701 (1052)	70.2
	KCM	852 (531)	51.0
4. 8 point HOU DCT using two 4 point DCT with 16 dual port memories (second stage, time-shared)	Xil	1452 (952)	76.1
	KCM	758 (505)	54.4

Table 2. Performance of various DCT circuits implemented on a Xilinx XVC50 Virtex FPGA

5. CONCLUSIONS

A technique for reducing the size of fixed point multipliers for DSP applications has been given. The technique uses spare capacity in the LUT to implement a MUX and thereby increase the functionality of the slice in the Virtex. The application of the approach to the DCT has been demonstrated in this paper but can also be applied to some types of filtering e.g. poly-phase filtering and other transforms such as the FFT, DHT and Wavelet transforms. Similar hardware reductions have been achieved with the poly-phase filter [8]. Currently, software is being developed to automatically produce efficient multipliers for a required set of coefficients. The impact of using only 4 as opposed to 8 coefficients in designs 3 and 4 gave a 12-16% relative gain in area.

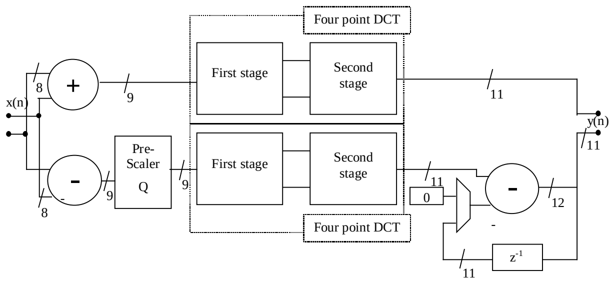


Fig. 12. Hou's 8 point DCT implementation

6. ACKNOWLEDGEMENTS

The authors gratefully acknowledge financial support from the DHFETE, IRTU, Nortel Networks and the Engineering and Physical Sciences Research Council.

7. REFERENCES

[1] G. R. Goslin, "Using Xilinx FPGAs to design custom Digital Signal Processing Devices", Proc. of the DSPX 1995, pp565-604, Jan 1995.
[2] R. Woods, D. Trainor, and J. P. Heron, "Real-time Image Processing using the Xilinx XC6200", IEEE Design and Test of Computers, pp30-38, Jan-Mar 1998.
[3] T. Courtney, R. H. Turner, and R. Woods, "An Investigation of Reconfigurable Multipliers for use in Adaptive Signal Processing", IEEE Symp. on FPGAs for Custom Computing Machines (FCCM), Napa, USA, pp341-343, May 2000.
[4] N. Shirazi, W. Luk, P. Cheung, "Automating Production of Run-Time Reconfigurable Designs", Proc. IEEE Symp. on FCCM, USA, pp. 147-156, April 1998.
[5] M. Potkonjak, M. B. Srivastava and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination", IEEE Trans. On CAD of Integrated Circuits and Systems", Vol. 15., No. 2, pp. 151-165, 1996.
[6] J. Hunter and J. V. McCanny, "Discrete Cosine Transform Generator for VLSI Synthesis", IEEE ICASSP, Vol. 5, pp. 2997–3000, 1998.
[7] H.S.Hou, "The fast Hartley transform algorithm", IEEE Transactions on Computers, Vol. C-36, No.2, pp.147-156, Feb. 1987.
[8] C. N. Ang, R. H. Turner, T. Courtney and R. Woods, "Virtex FPGA implementation of a polyphase filter for sample rate conversion", 34th Asilomar Conf. on Signals, Systems and Computers, Asilomar, USA, to be published, IEEE Computer Society, Oct. 2000.