Low power field programmable gate array implementation of fast digital signal processing algorithms: characterisation and manipulation of data locality — Source link

S. McKeown, Roger Woods

Institutions: Queen's University Belfast

Published on: 10 Mar 2011 - Iet Computers and Digital Techniques (Institution of Engineering and Technology)

Topics: Field-programmable gate array, Fast Fourier transform, Discrete Hartley transform, Discrete cosine transform and Dynamic demand

Published in IET Computers & Digital Techniques
Received on 18th May 2010
Revised on 7th September 2010
doi: 10.1049/iet-cdt.2010.0052

ISSN 1751-8601

Low power field programmable gate array implementation of fast digital signal processing algorithms: characterisation and manipulation of data locality

S. McKeown R. Woods
School of Electronics, Electrical Engineering and Computer Science, ECIT Institute, Queen's University Belfast, Queen's Island, Queen's Road, Belfast BT3 9DT, Northern Ireland
E-mail: r.woods@qub.ac.uk

Abstract: Dynamic power consumption is very dependent on interconnect, so clever mapping of digital signal processing algorithms to parallelised realisations with data locality is vital. This is a particular problem for fast algorithm implementations where typically, designers will have sacrificed circuit structure for efficiency in software implementation. This study outlines an approach for reducing the dynamic power consumption of a class of fast algorithms by minimising the index space separation; this allows the generation of field programmable gate array (FPGA) implementations with reduced power consumption. It is shown how a 50% reduction in relative index space separation results in a measured power gain of 36 and 37% over a Cooley–Tukey Fast Fourier Transform (FFT)-based solution for both actual power measurements for a Xilinx Virtex-II FPGA implementation and circuit measurements for a Xilinx Virtex-5 implementation. The authors show the generality of the approach by applying it to a number of other fast algorithms namely the discrete cosine, the discrete Hartley and the Walsh–Hadamard transforms.

1 Introduction

Digital signal processing (DSP) algorithms such as transforms, convolution and filtering are characterised by intensive computation and communication and contribute greatly to the system power consumption which is critical for their wider application. Power consumption comprises both a 'static' component, made up from transistor imperfections such as gate leakage and subthreshold currents, and a 'dynamic' part, generated from charging switched capacitance and short circuit currents which flow when the circuit is operating. Low power techniques aim to reduce one or more of these factors [1]. Static power is mostly influenced by technology choice for example, triple layer oxide and use of optimisations such as clock gating [2], however, in field programmable gate array (FPGA) implementations, designers have little control over these aspects and can typically act only to reduce the dynamic aspect by reducing the switched capacitance. In many applications such as mobile communications, the aim is to reduce energy; in applications such as radar and sonar where FPGAs are more commonly used, the system is always turned on so the focus has been to reduce dynamic power consumption.

The data dependency of successive operations impacts the distance over which data must be passed in order to complete the computation, thereby influencing power consumption. However, fast algorithms for example, Cooley–Tukey FFT [3], sacrifice data ordering looking to leverage symmetry and periodicity in the matrix to reduce the number of computations; this needs to be re-examined from a power perspective, as different mappings of the same algorithm can result in architectures with widely varying communications and performance characteristics [4]. It is well known that increasing this data locality has a direct relationship to the power consumed [5–7]. Indeed, previous work has involved investigating locality properties for FPGA-based systolic array implementations [8] and an FFT-based digital receiver solution based on a Xilinx Virtex-II FPGA implementation [9]; however, the real challenge is how to influence data locality at the algorithmic level in a systematic fashion, particularly for fast transforms that may not necessarily exhibit this locality in the first instance.

In this paper, a technique that allows the generation of power efficient, parallelised FPGA implementations based on manipulating and characterising data locality for a range of fast transforms, is presented. We show that by using the 'index space separation' as a measure of locality, realisations can be derived that are guaranteed to have a lower dynamic power consumption. The FFT is considered in detail but the approach is also applicable to other common transforms namely, the discrete cosine transform (DCT), the discrete Hartley transform (DHT) and the

Walsh–Hadamard transform [10]. Measured power results are given for a Xilinx Virtex-II device (taken from a purposely built Xilinx XUP (Xilinx University Program) board) and validated against a more recent FPGA realisation, namely Virtex-5, by comparing capacitance values.

Section 2 describes data locality in DSP algorithms, with Section 3 describing the fast algorithm domain and outlining the Cooley–Tukey FFT. Section 4 gives a definition of 'index space separation' and then goes on to present the proposed data locality methodology using the FFT as a design example with measured results. Section 5 indicates how it can be applied to other fast algorithms and is followed by conclusions.

2 Data locality in DSP algorithms

In massively parallel architectures, mappings with strong data locality result in 'systolic array' type architectures [4, 8] with all the advantages they bring, but only certain algorithm classes readily map to this type of structure. Kung [4] classified DSP algorithms as 'locally recursive' for example, matrix multiplication and 'globally recursive'; in locally recursive algorithms, data dependency is limited to adjacent elements and so the index space separation within each recursion has a predefined limit, resulting in an architecture with only local communication; in globally recursive algorithms however, inherently complex communication networks are required as the relative index space separation is larger.

This index space separation is given as a measure of total distance values between indices. Index space is defined as a lattice of points in an n-dimensional discrete space [11]. Defining the hierarchical space where each position in the lattice vector space is a Cartesian coordinate allows the index space to be defined in Euclidean geometry, and the index space separation as the Euclidean distance between indices. Index space separation, η, between two indices $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ is thus defined as

$$\eta = \sqrt{(a_1 + b_1)^2 + (a_2 + b_2)^2 + \cdots + (a_2 + b_2)^2} \tag{1}$$

and summarised as

$$\eta = \sqrt{\sum_{n=1}^{N} (a_n + b_n)^2} \tag{2}$$

In this work, the index space separation of each mapping is defined as the Euclidean distance in a single dimension in the spatial domain as the distance in the second dimension is always unitary. Using index space separation as a measure of 'data dependency' between consecutive operations, the relative distances over which data must be passed can be determined thereby giving a direct indication of the dynamic power requirements. In processor style implementations, this is not particularly important as all computation data are passed over a bus with a constant capacitance; however in FPGA solutions, these relate to separate, individual interconnections, created as a result of the place and route process in FPGA design, thereby producing varying levels of power consumption.

Fast algorithms exploit recursive patterns in computations by involving a structured matrix to compress the algorithms complexity, but require inherently globally interconnect to implement them [4]. This is achieved by folding the computation onto itself, thereby allowing common factors of the coefficients to be applied to multiple portions simultaneously. Data locality, however, is no longer dependent on the locality characteristics of the original structured matrix operation, but on the globally recursive definition of the compressed algorithm and the coefficient factorisation. The original fast algorithms were primarily designed for sequential software implementation, resulting in certain limitations on the range of feasible derivations. It is contended here that this approach largely used for processor implementation, is no longer valid for power efficient, parallelised FPGA realisations; the novel contribution of this work lies in re-examining fundamental assumptions underlying the standard fast algorithm derivations, leading to the exploration of the full range of mapping possibilities and to the creation of parallelised, power efficient implementations with strong data locality.

3 Fast algorithms

Many fundamental signal processing algorithms, such as the Fourier transform, can be expressed in terms of linear algebra on a dense structured matrix such as the matrix vector operation

$$\boldsymbol{y} = \boldsymbol{Ax} \tag{3}$$

Values in the structured matrix $\boldsymbol{A}$, are defined with reference to their relative positions in the form shown

$$\begin{bmatrix} y_{(0)} \\ y_{(1)} \\ \vdots \\ y_{(N-1)} \end{bmatrix} = \begin{bmatrix} a_{(0,0)} & \cdots & a_{(N-1,0)} \\ a_{(0,1)} & \cdots & a_{(N-1,1)} \\ \vdots & & \vdots \\ a_{(0,N-1)} & \cdots & a_{(N-1,N-1)} \end{bmatrix} \cdot \begin{bmatrix} x_{(0)} \\ x_{(1)} \\ \vdots \\ x_{(N-1)} \end{bmatrix} \tag{4}$$

The special structure of the matrix allows complexity to be reduced from $O(N^2)$ to $O(N \log N)$ by employing a fast 'divide and conquer' factorisation approach thereby avoiding operating on the N^2 entries and instead, focusing on a generator function encapsulating a compressed version of the matrix. The structured matrix representation is essentially a polynomial multiplication in the coefficient form of complexity $O(N^2)$. Fast algorithms utilise an alternate point-value representation in which a polynomial of degree $N - 1$ can be represented by N point-value pairs

$$\{\{x_{0:}y_0\}\{x_{1:}y_1\}_{:}\cdots\{x_{n-1:}y_{n-1}\}\} \tag{5}$$

where $y_k = A(x_k)$ and all x_k are distinct. For two polynomials of degree n which have a common x_k expression, the multiplication of the pair can now be represented as

$$\{\{x_{0:}y_0y'_0\}\{x_{1:}y_1y'_1\}_{:}\cdots\{x_{n-1:}y_{n-1}y'_{n-1}\}\} \tag{6}$$

This operation has a complexity of $O(N)$ but $2N$ points are required to represent both polynomials [12].

The reduction in complexity comes from the fact that in the operation $y_k = \boldsymbol{A}(x_k)$, any set, x_k, can be chosen so long as they are distinct [12]. In the case of fast transform algorithms, symmetry and periodicity in the transform kernel $\boldsymbol{A}$, enables the operation to be applied using a limited set of transform point-values [10]. This limited set represents common factors within the coefficients of the original transform matrix which arise because of the definition of the kernel.

In this way, problems are recursively decomposed $\log_r N$ times into smaller sub-problems of N/r (where r is the decomposition radix) until a simple base scenario is reached. The exact breakdown of each problem into r sub-problems depends on the symmetry and periodicity of the particular transform kernel being implemented. Solutions to the sub-problems are then combined to give a solution to the original problem with a complexity of only $O(N \log N)$ [3].

3.1 FFT algorithm

Consider the discrete Fourier transform (DFT)

$$\boldsymbol{X}_k = \sum_{n=0}^{N-1} \boldsymbol{x}(n)\boldsymbol{W}_N^{nk}, \quad k = 0, 1, \ldots, N-1 \tag{7}$$

where W_N is the twiddle factor, $W_N = \mathrm{e}^{-\mathrm{j}(2\pi/N)}$. The transformed samples, separated by angle θ, are periodic and mirrored to the left and right of the imaginary (Im), and above and below the real (Re) axis (Fig. 1). This symmetry and periodicity in the coefficients of the transform kernel (W_N) gives rise to a family of fast algorithms. The radix-2 form of the Cooley–Tukey FFT algorithm [3] recursively decomposes the algorithm until only two-point DFTs are required, with results then combined to compute the N-point transform. It is computed using the butterfly unit and perfect shuffle network shown in Fig. 2. In an application specific integrated circuit (ASIC) or FPGA implementation with separate processing elements, the globally recursive nature translates to irregular routing in the layout stage. The radix-2

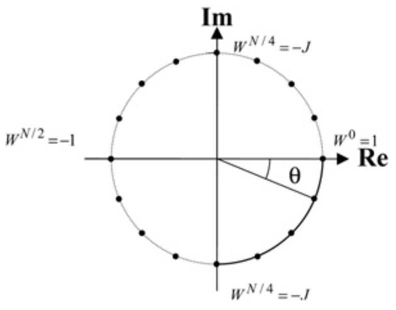


Fig. 1 *Fourier transform kernel where $N = 16$*

algorithmic expression of (7) is thus given as

$$\boldsymbol{X}_k = \sum_{n=0}^{N/2-1} \boldsymbol{x}(n)\boldsymbol{W}_N^{nk} + \boldsymbol{W}_N^{Nk/2} \sum_{n=0}^{N/2-1} \boldsymbol{x}(n+N/2)\boldsymbol{W}_N^{nk} \tag{8}$$

However, this is only one of a number of ways to implement a fast version and the methodology proposed here acts to gives more flexibility in the ordering of sub-problems [13].

4 Index space separation as a means of power efficiency

As described earlier, such matrix multiplications are directly implementable with only local communications in systolic array type architectures, generally described as

$$\boldsymbol{F}(k) = \sum_{n=0}^{N-1} \boldsymbol{f}(n) \cdot \boldsymbol{a}(k, n), \quad k = 0, 1, \ldots, N-1 \tag{9}$$

The computational complexity can be reduced from $O(N^2)$ to $O(N \log N)$ by repeatedly decomposing a series N into shorter series $N = N_1, N_2, \ldots, N_M$ and replacing indices k and n with the equivalent composite $k_1, k_2, \ldots, k_M$ and $n_1, n_2, \ldots, n_M$. In the Cooley–Tukey mapping, by considering the block length N as the composite N_1N_2 [3], the input and output indices n and k can be represented as

$$n = n_2N_1 + n_1, \quad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \tag{10}$$

$$k = k_1N_2 + k_2, \quad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \tag{11}$$

Although complexity is reduced, communication is global. The factorisation process involves partitioning the original index space onto equivalent subspaces which can be considered as multi-dimensional arrays with various implementations corresponding to different methods of unwrapping the multi-dimensional into one dimensional (1D) arrays, through index mapping of $k_1, k_2, \ldots, k_M$ and $n_1, n_2, \ldots, n_M$ [14]. For the Fourier transform, the mostly widely known mappings are decimation in time (DIT) and decimation in frequency (DIF) Cooley–Tukey [3] and Gentleman–Sande [15] algorithms. The DIT algorithms is represented by a top down recursive tree traversal of Fig. 3, and the DIF algorithms, a bottom up traversal from the leaf nodes.

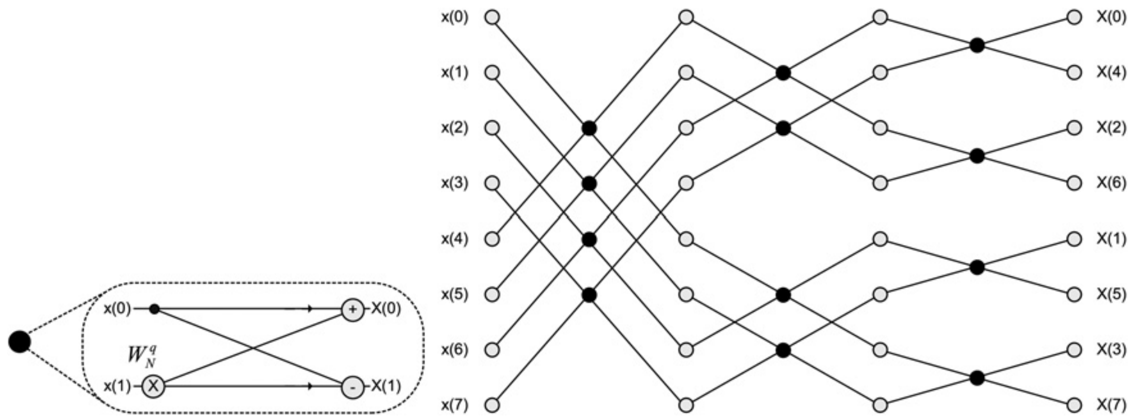


Fig. 2 *Eight-point radix-2 FFT structure*

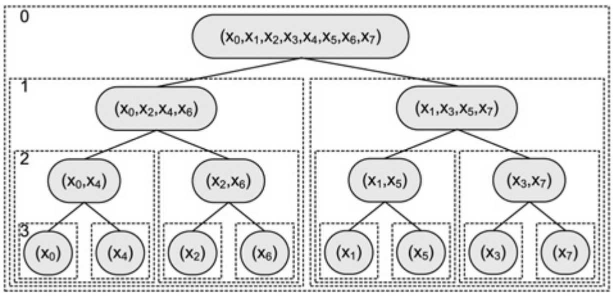


Fig. 3 *Tree structure of FFT recursive calls*

The Cooley–Tukey (Fig. 2) and Gentleman–Sande (Fig. 4*a*) algorithms represent a traditional in place recursive traversal where in this radix-2 decomposition, all even vectors move to the left and odd vectors to the right. Each recursive call makes two more calls, until vectors of length 1 are reached at 3rd level node. The level of interconnection is identical in both these implementations.

For a conventional realisation, a strict unwrapping of the multi-dimension array indices $k_1, k_2, \ldots, k_M$ and $n_1, n_2, \ldots, n_M$ for in-place computation is followed, resulting in a long interconnections in the resulting signal flow graph (SFG) such as that shown in Fig. 4*a*. Here the 1D unwrapped index varies as $X_m(n_1 \cdots n_{M-m}, k_m \cdots k_1)$ for $m = 0 \cdots M$, where the right most index varies the fastest. This is illustrated in the notation above each stage, showing how the operation initially described only in terms of the time domain index n, has each term systematically substituted by frequency domain index k terms as the graph is traversed from left to right until at the output, it is described solely in the frequency domain [14]. The restriction of in-place computation while critical to efficient

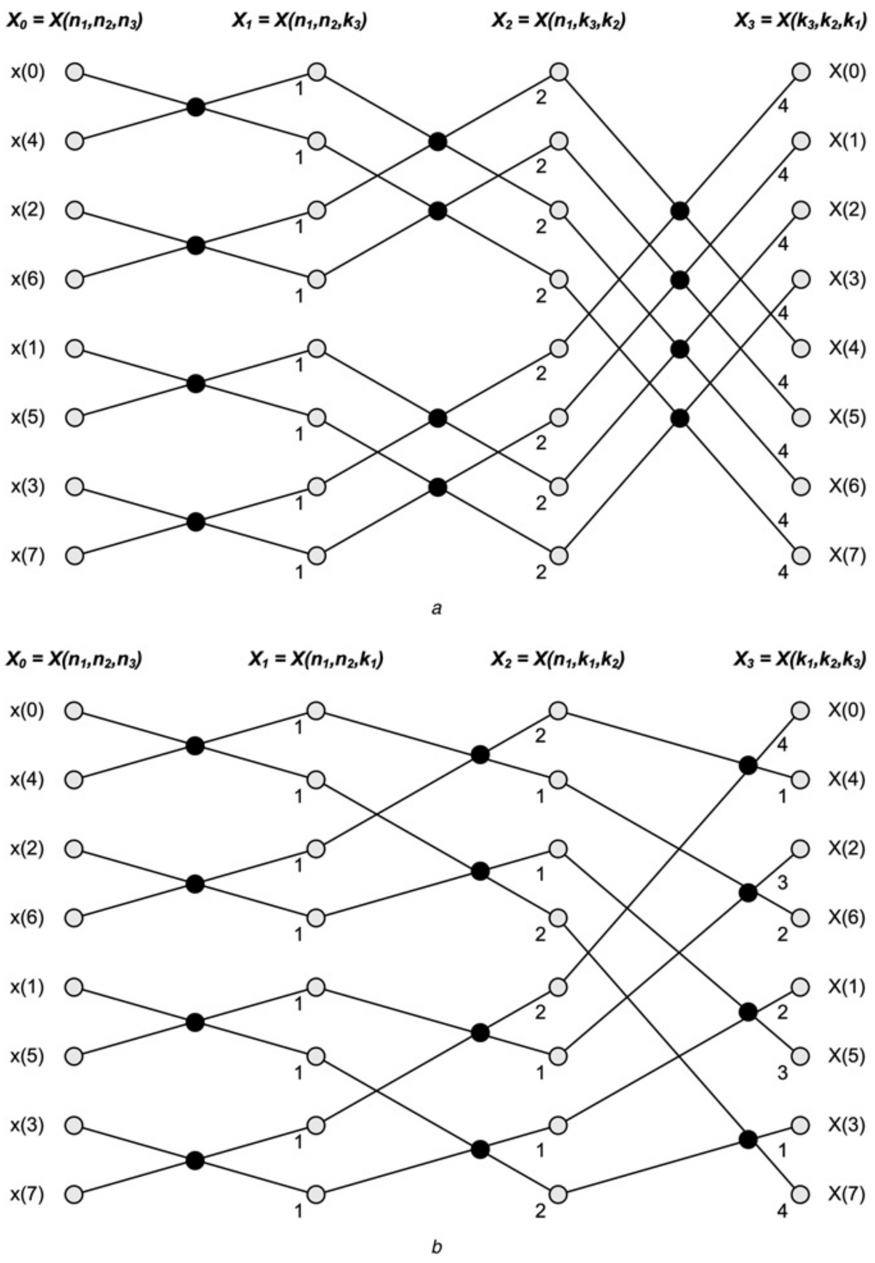


Fig. 4 *Various FFT flow graphs*

a Eight-point radix-2 in-place Gentleman–Sande
b Eight-point radix-2 modified flow graph

storage of intermediate results in software, has little real benefit in parallelised hardware implementation.

Revisiting this assumption, an alternative mapping scheme is possible which removes the in-place restriction meaning that index k_m is no longer required to take the position of n_{M-m+1}. By varying the placement and weighting of index k_m as a function of m during unwrapping, it is possible to limit the space separation between subsequent indices to achieve increased data locality. In this case, the 1D unwrapped index varies as $X_m(n_1 \cdots n_{M-m}, k_1 \cdots k_m)$ and has both the input and output in reverse digit order [14], as illustrated by a comparison of the in-place radix-2 SFG of Fig. 4*a* with the mapping of the proposed modified SFG of Fig. 4*b*. Index space separation distance is shown as numbers at each node where it represents the vertical distance in the graph.

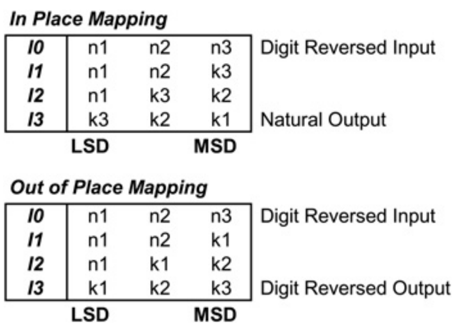
The index schedule, where $n_1, n_2, n_3 = 0, 1$ and $k_1, k_2, k_3 = 0, 1$ for iterations I_1, I_2, I_3 and I_4, for the in-place mapping schedule (equation (12)) and the out-of-place mapping schedule (equation (13)) is shown in Fig. 5.

$$\begin{aligned} x(n) &= I_1(n_1 + 2n_2 + 4n_3) \\ X(k) &= I_4(4k_3 + 2k_2 + 1k_1) \end{aligned} \tag{12}$$

$$\begin{aligned} x(n) &= I_1(n_1 + 2n_2 + 4n_3) \\ X(k) &= I_4(1k_1 + 2k_2 + 4k_3) \end{aligned} \tag{13}$$

For the in-place mapping schedule, the weighting of each index remains constant as time domain components n are substituted with frequency domain components k. In the modified schedule however, by changing the sequence in which frequency domain components are substituted and by varying the weighting of the components between iterations to compensate, the index space separation between each stage is now reduced; this now means that we are generating intermediate results in closer proximity to where they are required at the next stage. In the schedule given in (13) and Fig. 5, the most heavily weighted component remains constant on the rightmost side of the equation as the indices are substituted, meaning that the Hamming distance between consecutive stages is reduced. It is this proximity of intermediate results from increased data locality that leads to shorter interconnect and consequently lower power requirements.

Thus for the FFT design, the index space separation is reduced by 28.6% from 56 to 40 for the eight-point version. This can be generalised as a reduction from $O(N(N-1))$ to $O(N(N + \log_2 N - 1)/2)$ in radix-2 design for all N, converging to a 50% reduction for larger N as shown in Table 1 for the in-place and modified versions given in Figs. 4*a* and *b*.



In Place Mapping

I0	n1	n2	n3	Digit Reversed Input
I1	n1	n2	k3	
I2	n1	k3	k2	
I3	k3	k2	k1	Natural Output
	LSD		MSD	

Out of Place Mapping

I0	n1	n2	n3	Digit Reversed Input
I1	n1	n2	k1	
I2	n1	k1	k2	
I3	k1	k2	k3	Digit Reversed Output
	LSD		MSD	

Fig. 5 *Index schedule*

Table 1 Radix 2 index space separation

N	In-place (Fig. 4*a*)	Modified (Fig. 4*b*)	Reduction (%)
2	2	2	0.0
4	12	10	16.7
8	56	40	28.6
16	240	152	36.7
32	992	576	41.9
64	4032	2208	45.2
128	16 256	8576	47.2
256	65 280	33 664	48.4
512	261 632	133 120	49.1
1024	1 047 552	528 896	49.5
2048	4 192 256	2 107 392	49.7
4096	16 773 120	8 411 136	49.9

4.1 FFT index mapping schedules

The 50% reduction in index space separation achieved by the proposed mapping scheme is optimal in terms of data locality of the range of mappings. There are eight principle index-mapping schedules for the DIT and also their DIF equivalents [14] as shown in Table 2. The resulting eight-point radix-2 SFG are shown in Fig. 6. The subscripts r and l of the schedules denote whether the right most or left most index respectively varies the fastest.

Transforms *T*1 and *T*2 are the commonly used standard Cooley–Tukey types with in-place computation and input/output in reverse digit order; transforms *T*3 and *T*5 are those proposed here which were previously considered less attractive for implementation; transforms *T*4 and *T*6 also require out-of-place computation and have both natural ordered inputs and outputs; this can be of some advantage in software realisation but they have the widest index space separation which has implications for power consumption in hardware implementations; transforms *T*7 and *T*8, have a constant geometry factorisation across the transform stages making them suitable for a reduced area column based processor. These have either input or output in reverse digit order as with *T*1 and *T*2, and are also out-of-place, but the index space separation is high even when compared to the original Cooley–Tukey.

A 64-point radix-4 FFT version of *T*3 has been implemented for a high throughput digital radar receiver application. The details of the setup and measurements are

Table 2 FFT index mapping schedules for an eight-point radix-2 SFG

Transform	Schedule	Index space
T1	$X_m(n_1, \ldots, n_{M-m}, k_m, \ldots, k_1)_r$	56
T2	$X_m(n_1, \ldots, n_{M-m}, k_m, \ldots, k_1)_l$	56
T3	$X_m(n_1, \ldots, n_{M-m}, k_1, \ldots, k_m)_r$	40
T4	$X_m(n_1, \ldots, n_{M-m}, k_1, \ldots, k_m)_l$	76
T5	$X_m(k_1, \ldots, k_m, n_1, \ldots, n_{M-m})_r$	40
T6	$X_m(k_1, \ldots, k_m, n_1, \ldots, n_{M-m})_l$	76
T7	$X_m(k_m, \ldots, k_1, n_1, \ldots, n_{M-m})_r$	60
T8	$X_m(k_m, \ldots, k_1, n_1, \ldots, n_{M-m})_l$	60

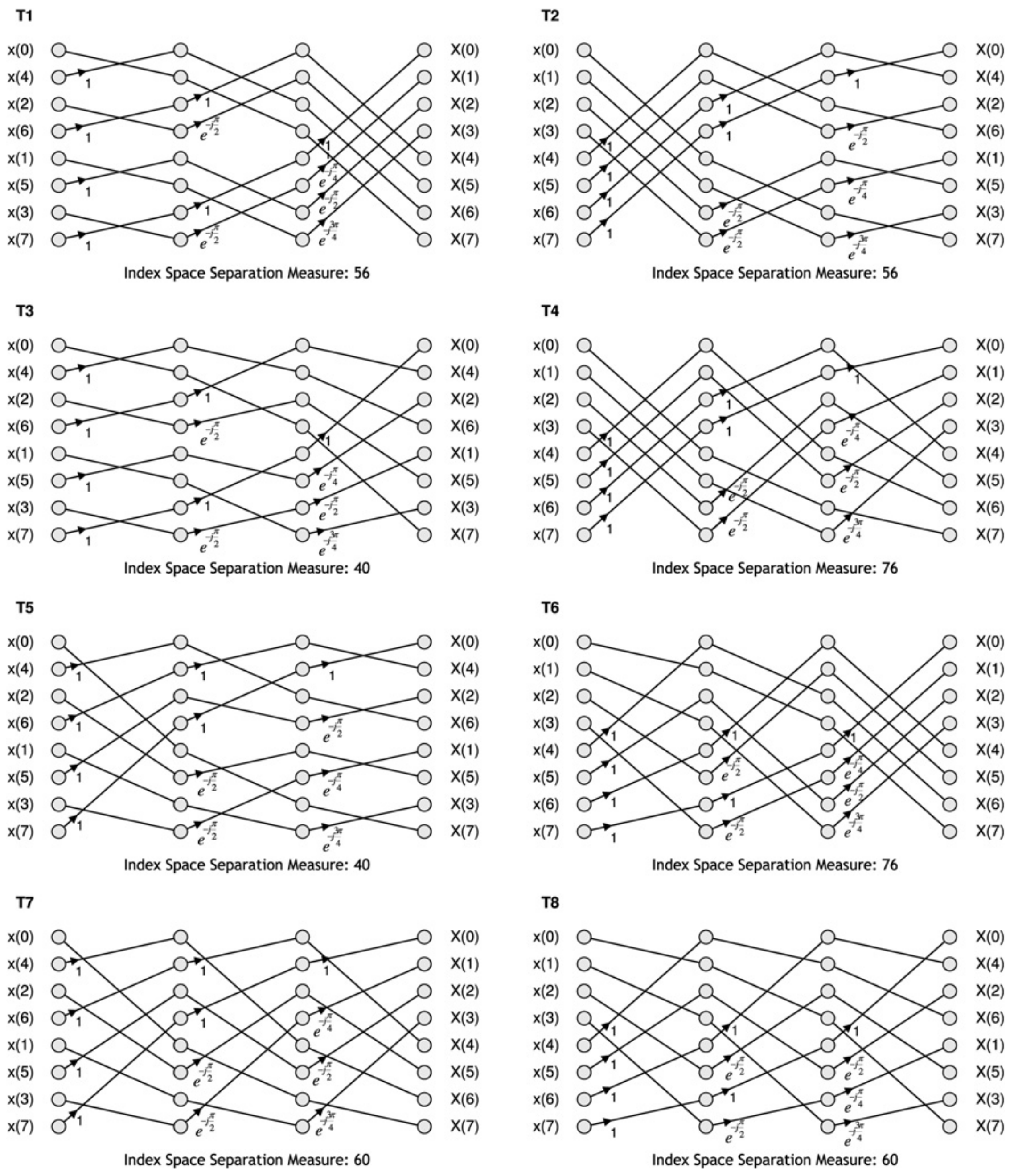

T1
Index Space Separation Measure: 56
T2
Index Space Separation Measure: 56
T3
Index Space Separation Measure: 40
T4
Index Space Separation Measure: 76
T5
Index Space Separation Measure: 40
T6
Index Space Separation Measure: 76
T7
Index Space Separation Measure: 60
T8
Index Space Separation Measure: 60

Fig. 6 *Eight-point radix-2 FFT mappings*

described in detail in [9] with the intention here to use the results to demonstrate the usefulness of the technique. The approach (QFFT) was coded in VHDL, synthesised using the Xilinx-II FPGA technology and then implemented on the Xilinx XUP Virtex-II Pro board which has been specifically created to allow power measurements to be taken. This hardware setup uses a custom test harness in a controlled environment and can be used to obtain real power results for this design and the two commercial, Cooley–Tukey type FFT derivations from Xilinx and Amphion. The power measurements for all three implementations were taken using 'real' data captured from a digital receiver. Shorter interconnect was achieved with average net capacitance (and standard deviation) reduced from 1.4 pF (3.3) and 1.9 pF (8.5) in the Xilinx and Amphion designs respectively (both Cooley–Tukey implementations), to 1.0 pF (2.6) in the QFFT (Fig. 7*a*). Power savings of between 36 and 37% that is, from 1616 to 1029 mW, were achieved over the Xilinx and Amphion designs with even higher savings against the other core [9]. Area savings were also made with the design as it used only 35% of the slices and 50% of the DSP48E blocks when compared to the Xilinx design and 20% of the slices when viewed against the Amphion design which does not use any DSP48E blocks. These area gains should be treated lightly as it is not clear what additional functionality either of the two comparative designs had; however, based on the post place and route analysis, there is clear indication that the QFFT design results in much smaller interconnect; more detail on this work is given in [9].

The availability of the Virtex-II XUP board allowed actual measurements to be made. However as this was an older technology, we implemented the same designs using the

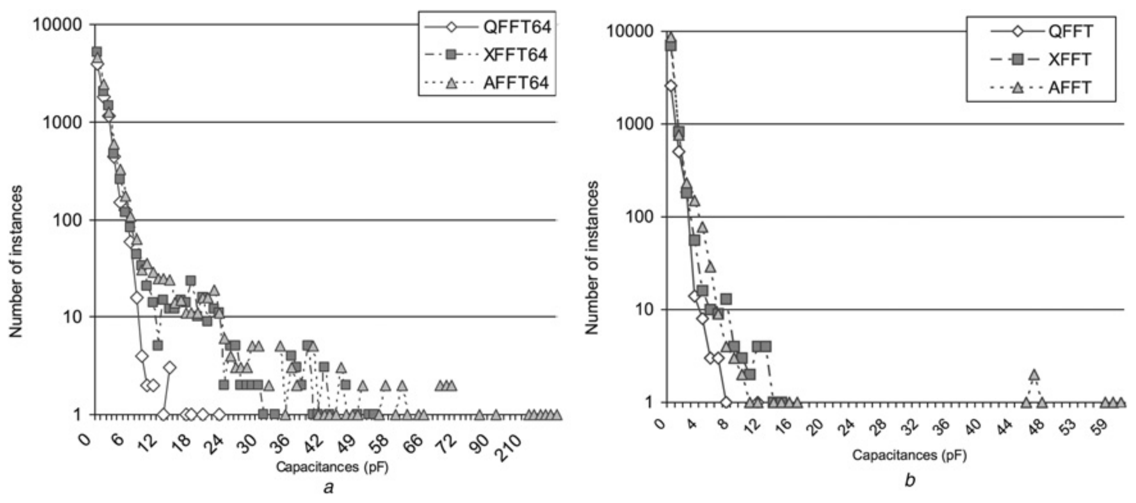


Fig. 7 *Interconnect capacitance*

a Virtex-II interconnect capacitance
b Virtex-5 interconnect capacitance

newer Xilinx Virtex-5 technology and showed that the similar improvements were achieved in terms of power consumption and interconnect (actual power measurements were not possible as a suitable FPGA board was not available). Here interconnect capacitance (and standard deviation) is reduced from an average of 1.6 pF (3.8) and 2.1 pF (5.4) in the Xilinx and Amphion designs, respectively, to 1.4 pF (2.7) in the QFFT (Fig. 7*b*). This equated to a 45% reduction in total driven interconnect capacitance of the QFFT over the nearest Cooley–Tukey implementation.

5 Application to other fast algorithms

The properties of symmetry and periodicity shared by fast transform algorithms mean that it is possible to create efficient derivations of a range of common algorithms in a similar manner. Given the detailed analysis just outlined for the FFT, the same power reductions should be possible by achieving a reduction in index space separation. The technique is now applied to the Walsh-Hadamard, DCT and DST transforms.

The in-place fast Walsh algorithm (14) has a structure similar to that of the FFT as shown in Fig. 8*a* but with coefficients limited to values of +1 and −1 [16]. The Walsh algorithm is defined as

$$X_m = \sum_{n=0}^{M-1} x(n)\text{wal}(m, n), \quad m, n = 0, 1, \ldots, M-1 \quad (14)$$

for an $M =$ eight-length real array, where wal is recursively defined as $\text{wal}(m, n) = \text{wal}([m/2], 2n).\text{wal}(m - 2[m/2], n)$; for the initial value, $\text{wal}_0 = 1$ with coefficient values of +1 and −1 [16]. When remapped the eight-point radix-2 example (Fig. 8*b*) achieves a 28.6% reduction that is, from 56 to 40 in index space separation and approaches a 50% reduction for larger transforms.

The DST and DCT algorithms differ in the application of the cosine and sine functions in the transform kernel as shown in (15) and (16), respectively. There are several versions of these algorithms, with synonymous implementations existing between the wide range of FFT derivations and DCT and DST equivalents [17]. Common derivations are given in Fig. 9*a* for the DCT and in Fig. 9*b* for the DST [18].

For the remapped eight-point radix-2 decompositions, the DCT (Fig. 10*a*) achieves a 25.4% reduction from a total index space separation (63 to 47) and in the DST (Fig. 10*b*), a 23.9% reduction from 67 to 51. Reductions approach 50% for larger *N* for both transforms. Therefore

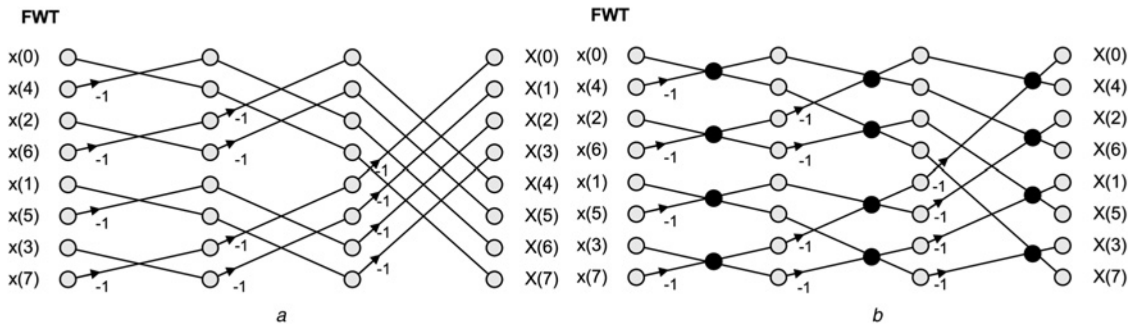


Fig. 8 *Eight-point radix-2 fast Walsh transforms*

a In-place
b Modified

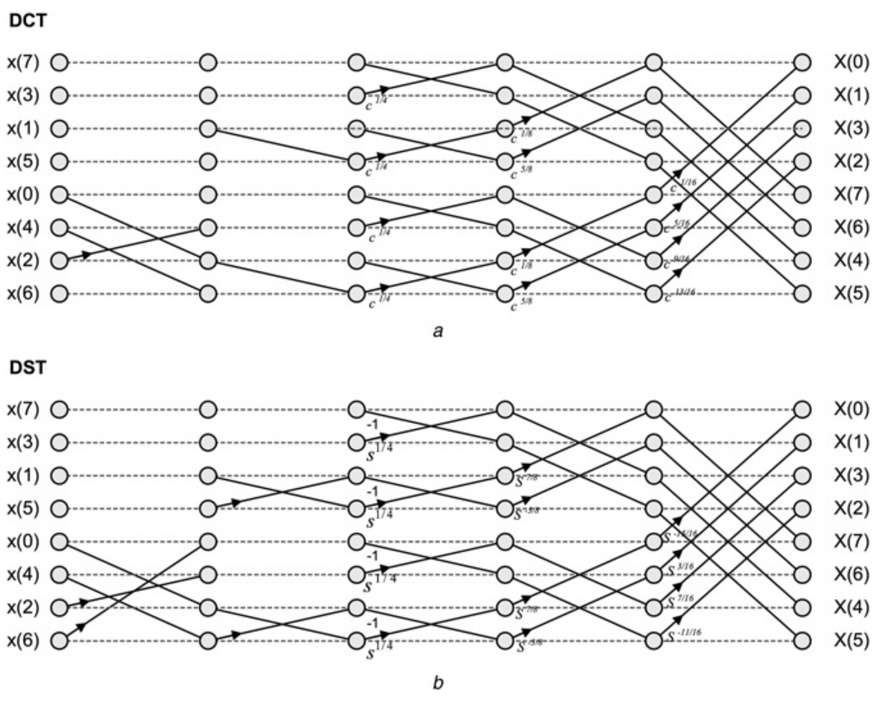


Fig. 9 *Eight-point radix-2 in-place fast sine and cosine transforms*

a Cosine transform
b Sine transform

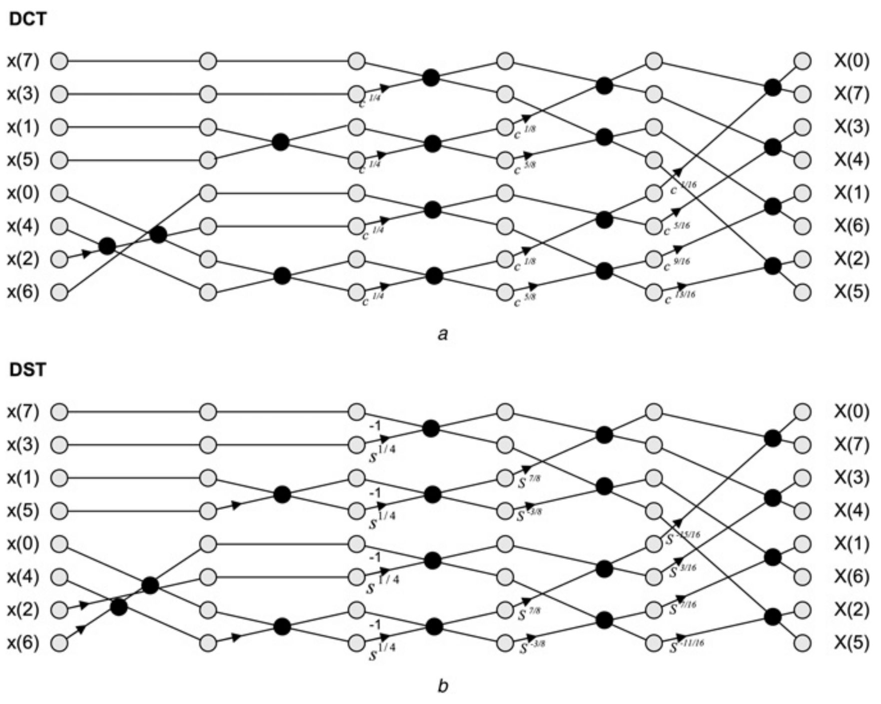


Fig. 10 *Eight-point radix-2 modified fast sine and cosine transforms*

a Cosine transform
b Sine transform

similar power benefits are expected as to those observed in the FFT example

$$X_k = \sum_{n=0}^{N-1} x(n)\cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right], \quad k = 0, 1, \ldots, N-1 \tag{15}$$

$$X_k = \sum_{n=0}^{N-1} x(n)\sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k+1\right], \quad k = 0, 1, \ldots, N-1 \tag{16}$$

6 Conclusions

A technique for improving the data locality of common DSP transforms has been presented which gives the system designer the ability to optimise for power consumption at the earliest stages of the design flow. By adopting an algorithmic mapping strategy that maximises data locality in fast algorithms, more power efficient architectures for FPGA implementation are obtained. The mapping process produces designs with short local interconnect that are particularly advantageous in FPGA implementations where interconnect is a dominant factor in power consumption. The technique is not, however, FPGA platform specific and is therefore suitable for any massively parallel architecture, for instance systolic arrays [4, 8].

Measured capacitance and power results demonstrate the effectiveness of the approach. The technique, applicable across a range of fast DSP applications, has demonstrated a 50% reduction in FFT index space separation resulting in a measured power gain of 36–37% over a Cooley–Tukey FFT-based solution. Similar reductions are achieved in the DCT, DST and Walsh–Hadamard transforms. Fundamental algorithmic similarities necessary to this approach that exist across the extensive class of fast divide and conquer DSP algorithms mean its application is wide ranging.

7 References

1 Tuan, T., Rahman, A., Das, S., Trimberger, S., Kao, S.: ‘A 90 nm low-power fpga for battery-powered applications’, *IEEE Trans. CAD*, 2007, **26**, (2), pp. 296–300
2 Woods, R., McAllister, J., Turner, R., Yi, Y., Lightbody, G.: ‘FPGA-based implementation of signal processing systems’ (Wiley, New Jersey, 2008)
3 Cooley, J.W., Tukey, J.W.: ‘An algorithm for the machine calculation of complex fourier series’, *Math. Comput.*, 1965, **19**, pp. 297–301
4 Kung, S.Y.: ‘VLSI array processors’ (Prentice-Hall, New Jersey, 1988)
5 Wilton, S.J.E., Luk, W., Ang, S.S.: ‘The impact of pipelining on energy per operation in field-programmable gate arrays’. Proc. Int. Conf. on Field Programmable Logic, August 2004, pp. 719–728
6 Lee, T., Cong, J.: ‘The new line in ic design’, *IEEE Spectr.*, 1997, **34**, (3), pp. 52–58
7 Rabaey, J.M., Chandrakasan, A., Nikolic, B.: ‘Digital integrated circuits: a design perspective’ (Prentice-Hall, New Jersey, 2003, 2nd edn.)
8 Choi, S., Prasanna, V.K.: ‘Time and energy efficient matrix factorization using fpgas’. Int. Conf. on Field Programmable Logic and Applications, Lisbon, Portugal, 1–3 September 2003, pp. 507–519
9 McKeown, S., Woods, R., McAllister, J.: ‘Algorithmic factorisation for low power fpga implementation through increased data locality’. Proc. IEEE Int. Symp. VLSIDAT, April 2008, pp. 271–274
10 Guoan, B., Zeng, Y.: ‘Transforms and fast algorithms for signal analysis and representations’ (Birkhauser, Boston, 2004)
11 Parashar, M., Browne, J.C.: ‘Systems engineering for high performance computing software: the hdda/dagh infrastructure for implementation of parallel structured adaptive mesh refinement’, IMA Volume 117: Structured Adaptive Mesh Refinement (SAMR) Grid Methods, 2000, vol. 117, pp. 1–18
12 Morris, J.: ‘Data structures and algorithms’, March 2009, available at http://oopweb.com/Algorithms/Documents/PLDS210/VolumeFrames.html
13 Cormen, T.H., Leiserson, C.E., Rivest, R.L.: ‘Introduction to algorithms’ (The MIT Press, Cambridge, MA, 2001, 2nd edn.)
14 Thong, T.: ‘Algebraic formulation of the fast fourier transform’, *IEEE Circuits Syst. Mag.*, 1981, **3**, pp. 9–19
15 Gentleman, W.M., Sande, G.: ‘Fast fourier transforms-for fun and profit’. Proc. AFIPS Fall Joint Computation Conf., 1966, vol. 29, pp. 563–578
16 Shanks, J.L.: ‘Computation of the fast walsh-fourier transform’, *IEEE Trans. Comput.*, 1969, **C-18**, pp. 457–459
17 Olshevsky, V. (Ed.): ‘Fast algorithms for structured matrices: theory and applications’ (American Mathematical Society, Providence, 2003)
18 Cvetkovic, Z., Popovic, M.V.: ‘New fast recursive algorithms for the computation of discrete sine and cosine transforms’, *IEEE Trans. Signal Process.*, 1992, **40**, pp. 2083–2086