BEST: International Journal of Humanities, Arts, Medicine and Sciences (BEST: IJHAMS)
Vol. 1, Issue 3, Dec 2013, 61-64

AN EFFICIENT FAST DHT ALGORITHM FOR VLSI IMPLEMENTATION USING SPLIT RADIX ALGORITHM

M.TAMILSELVI[1] & A.LAKSHMINARAYANAN[2]

[1]PG Scholar, Department of Electronics and Communication Engineering, Kongunadu College of Engineering and Technology, Trichy, Tamil Nadu, India

[2]Assistant Professor, Department of Electronics and Communication Engineering, Kongunadu College of Engineering and Technology, Trichy, Tamil Nadu, India

ABSTRACT

A new very large scale integration (VLSI) algorithm for a 2N-length discrete Hartley transform (DHT) that can be efficiently implemented on a highly modular and parallel VLSI architecture having a regular structure is presented. The DHT algorithm can be efficiently split on several parallel parts that can be executed concurrently. Moreover, the proposed algorithm is well suited for the subexpression sharing technique that can be used to significantly reduce the hardware complexity of the highly parallel VLSI implementation.

KEYWORDS: Discrete Hartley Transform (DHT), DHT Domain Processing, Fast Algorithms

INTRODUCTION

THE DISCRETE Fourier transform (DFT) is used in many digital signal processing applications as in signal and image compression techniques, filter banks [1], signal representation, or harmonic analysis. The discrete Hartley transform (DHT) [2], [3] can be used to efficiently replace the DFT when the input sequence is real. There are also several split-radix algorithms for computing DHT with a low arithmetic cost we have a highly parallel solution for the implementation of DHT based on a direct implementation of fast Hartley transform (FHT). It is worth to note that hardware implementations of FHT are rare. Multipliers in a VLSI structure consume a large portion of the chip area and introduce significant delays. This is the reason why memory-based solutions to implement multipliers have been more and more used in the literature.

To efficiently implement multipliers with lookup-table-based solutions, it is necessary that one operand to be a constant. When one of the operands is constant, it is possible to store all the partial results in a ROM, and the number of memory words is significantly reduced from 2^{2L} to 2^{L}.The normal arithmetic operations are performed in many applications. In VLSI architecture also additions, multiplication like arithmetic operation are performed, by using some transform function the speed of operation will increased. The additions take less amount of time to execution, but multiplication take more time because of complex operation. So in that the algorithm development is concentrated to Multipliers. The best known transforms are those named for Laplace, Fourier, Hilbert, Hankel, Mellin, and Abelall ofwhich continue to attract contributions to the mathematical literature.

PARALLEL PROCESS

Parallel version mainly used when combining a larger FFT with a 3 or 5 point FFT, since it is feasible to use 3 or 5 large FFTs in a single device.

SPLIT RADIX ALGORITHM

The split-radix FFTis a fast Fourier transform (FFT) algorithm for computing the discrete Fourier transform (DFT), split radix is a variant of the Cooley-Tukey FFT algorithm that uses a blend of radices 2 and 4: it recursively expresses a DFT of length N in terms of one smaller DFT of length N/2 and two smaller DFTs of length N/4 totally 3 expression. The split-radix FFT, along with its variations, long had the distinction of achieving the lowest published arithmetic operation count (total exact number of required real additions and multiplications) to compute a DFT of power-of-two sizes N. The split-radix algorithm can only be applied when N is a multiple of 4, but since it breaks a DFT into smaller DFTs it can be combined with any other FFT algorithm as desired.

$$X_k = \sum_{n=0}^{N-1} x_n \left(\cos\frac{2\pi}{N}nk + \sin\frac{2\pi}{N}nk\right)$$

These are computed form, the decoputed form are

$$X_{2k} = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} (x_n + x_{n+(N/2)}) \times (\cos\frac{2\pi}{N/2}nk + \sin\frac{2\pi}{N/2}nk)$$

$$X_{4k+1} = \sum_{n=0}^{\left(\frac{N}{4}\right)-1} \left[\left(A_n + A_{\left(\frac{N}{4}\right)-n}\right)\cos\frac{2\pi}{N}n - \left(B_{n-}B_{\left(\frac{N}{4}\right)-n}\right)\sin\frac{2\pi}{N}n\right] \times (\cos\frac{2\pi}{N/4}nk + \sin\frac{2\pi}{N/4}nk)$$

$$X_{4k+3} = \sum_{n=0}^{\left(\frac{N}{4}\right)-1} \left[\left(A_n - A_{\left(\frac{N}{4}\right)-n}\right)\cos\frac{2\pi}{N}3n + \left(B_n+B_{\left(\frac{N}{4}\right)-n}\right)\sin\frac{2\pi}{N}3n\right] \times (\cos\frac{2\pi}{N/4}nk + \sin\frac{2\pi}{N/4}nk)$$

The operation of split radix in DHT isThe split-radix algorithm3 is based on both even-term radix-2 decompositions and odd-term radix-4decompositions simultaneously.

$$N - \text{Point DHT} \rightarrow \frac{N}{2} - \text{Point DTH} + 2 \times \frac{N}{4} - \text{Point DHTs}$$

Split-radix refers to combinations of two (or more) FFT engines, Split-radix FFTs have a similar structure to 2D FFTs. Split-radix FFTs provide bin spacing that produce better results for many applications.

TYPES OF SPLIT-RADIX APPROACH

Two split-radix approaches employed by DE:

- **Serial (Traditional):** Lower performance, higher memory requirements by using serial versions of both FFTs.
- **Parallel:** Higher performance by placing larger radix FFTs in parallel and using a parallel version of the smaller radix FFT

Continuous data FFTs require enough memory to store two full copies of the data for each re-order stage

RELATIONSHIP WITH FOURIER TRANSFORMS

This transform differs from the classic Fourier transformF(ω)=ℱ{f(t)}(ω) in the choice of the kernel. In the Fourier transform,

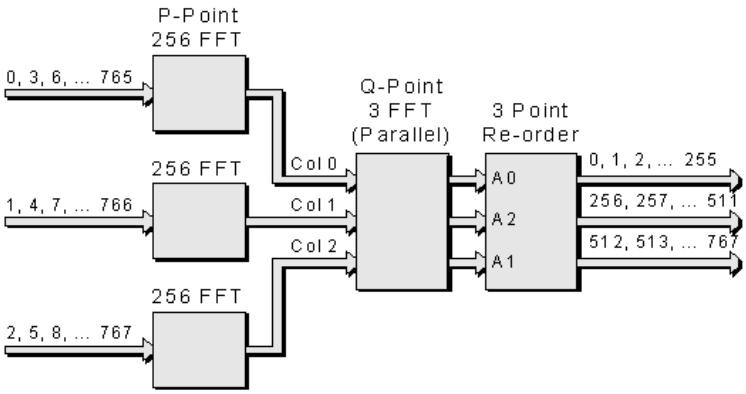

Figure 1: Parallel 768-Point Split-Radix FFT

That is, the real and imaginary parts of the Fourier transform are simply given by the even and odd parts of the Hartley transform, respectively. Conversely, for real-valued functions f(t), the Hartley transform is given from the Fourier transform's real and imaginary parts:

$$\{Hf\} = R\{\mathcal{F}f\} - I\{\mathcal{F}f\} = R\{\mathcal{F}f.(1+i)\}$$

Where R and Idenote the real and imaginary parts of the complex Fourier transform.

COMMON SUBEXPRESSION SHARING

Common subexpression sharing shares the subexpression among several multiplication-accumulation operations so that the total operation count is reduced. For example, Figure 2 shows the FIR filter coefficients represented by the canonical signed digit (CSD) form. The circled groups of digits have the same subexpression, so they can share the same computation unit. This approach is very effective for reducing the hardware cost of multiple constant multiplications, especially for the filter-like operation.

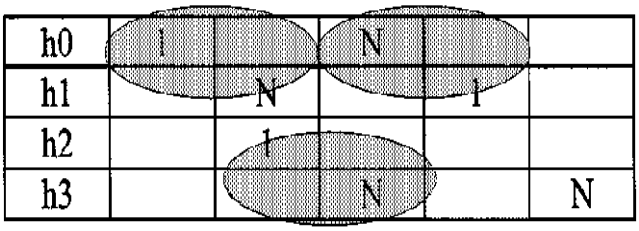

Figure 2: Coefficients and Common Subexpression

where N denotes -1

Thus, by sharing the common subexpression, the number of additionsis reduced (38% reduction). The common subexpression part is first calculated, and then we shift or negate the subexpressionfor other computations. The hardware to generate different constant multiplications is called adder network in the paper. By using subexpression sharing, much computation can be saved if we can maximally find these common subexpressions.

Considered one example of common subexpression sharing as a 3-tap FIR filter, the output Y(2) is given as follow.

$Y(2)=\sum_{i=0}^{2} A_i * X_{n-i}$

Consider the same example in Figure 3, where conventional horizontal subexpressions are given by

$x_2=x_1+x_1>>2$ and $x_3=x_1-x_1>>2$

From the remaining bits, two vertical subexpressions, 101 and n01, are obtained: $x_4=x_1+x_1[-2]$ and $x_5=-x_1+x_1[-2]$ where [-k] represents the delay operation.

COMBINED HORIZONTAL AND VERTICAL SUBEXPRESSION SHARING

The Combined Horizontal and Vertical common sub-expressions in 15-tap linear phase raised cosine filter coefficients are explained in the below diagram.

We consider the transposed direct form FIR filter structure for implementation. It can be noted that only twenty adders are required to implement the filter, two for horizontal common subexpressions,

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
h(0)					1		n			1		1
h(1)												
h(2)				1		n					1	
h(3)												
h(4)			1		n				1			n
h(5)												
h(6)		1		1					1			1
h(7)	1											

Figure 3: Combined Horizontal and Vertical

By combining the common subexpressions equations the output of the filter can be represented as:

$y=x_3>>5+x_2>>10+x_3[-2]>>4+x_1[-2]>>11+x_3[-4]>>3+x_4[-4]>>9+x_5[-4]>>12+x_2[-6]>>2+x_1[-7]>>1+x_2[-8]>>9-x_5[-8]>>12+x_3[-10]>>3+x_3[-12]>>4+x_1[-12]>>11+x_3[-14]>>5+x_2[-14]>>10$

The two for vertical common subexpressions, and sixteen for filter output. This method results in reduction rates of 16.6% and 6.6% when compared to vertical and horizontal common subexpression methods, respectively.

CONCLUSIONS & FUTURE WORK

A new highly parallel VLSI algorithm for the computation of a length-N = 2^nDHT having a modular and regular structure has been presented. Moreover, this algorithm can be implemented on a highly parallel architecture having a modular and regular structure with a low hardware complexity by extensively using a sub expression sharing technique and the sharing of multipliers having the same constant Moreover, this algorithm can be implemented on a highly parallel architecture having a modular and regular structure with a low hardware complexity by extensively using a sub expression sharing technique and the sharing of multipliers having the same constant.

REFERENCES

1. D. F. Chiper, “A Novel VLSI DHT Algorithm for a Highly Modular and Parallel Architecture,” IEEE Trans.CircuitsSyst.II, Exp.Briefs, vol.60, no.5, pp.282-286, May 2013.
2. B. Widrowet al., “Fundamental relations between the LMS algorithm and the DFT,” IEEE Trans. Circuits Syst., vol. CAS-34, pp. 814–820, July 1987.
3. Z. Wang, “A prime factor fast W transform algorithm,” IEEE Trans. Signal Processing, vol. 40, pp. 2361-2368, Sept. 1992.
4. R. N. Bracewell, “Discrete Hartley transform,” J. Opt. Soc. Amer., vol.73, pp. 1832–1835, Dec. 1983.