Using Divide-And-Conquer on Custom Lengthed Fourier Transforms

Thomas DETTBARN, Frank MAYER
Fraunhofer Institute for Integrated Circuits IIS, Erlangen, Germany

Abstract– **Fourier Transforms (FTs) are the key to all modern, OFDM-based (Orthogonal Frequency Division Multiplexing) transmission schemes. This paper demonstrates the principles and the efficient implementation of arbitrary length FTs, using divide and conquer techniques for decomposition into "atomic" transforms. Using "Digital Radio Mondiale" as an example, we will present generalized FT processing hardware, its complexity, processing time, precision and design re-use.**

I. Introduction

Fourier Transform A *Fourier Transform* FTn is an isomorphism mapping a signal with n samples from time to the frequency domain. Basically this is nothing more than a multiplication using the *Vandermonde*-matrix V_n :

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} \quad (1)$$

Where $\omega_n^1 = \omega_n = \cos(2\pi/n) - \sin(2\pi/n)\, j,\ j^2 = -1$ are called *Twiddle factors* [1]. The time efficiency is $\theta(n^2)$.

FT2^d, a special case: Instead of evaluating each output separately, a shortcut is to re-use previous results. The circuit in Fig. 1 is equivalent to $y = V_4 x$:

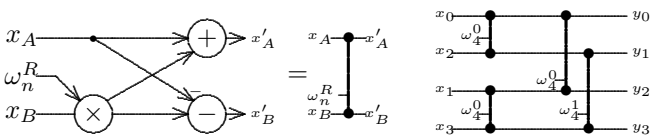

Fig. 1. Left: A DFT2 circuit. Right: 3 DFT2 form one FFT4

The left circuit in Fig. 1 implements a *Discrete Fourier Transform* with 2 in- and 2 outputs (DFT2) so-called *Butterfly*. For a FT2^{d+1} two DFT2^ds can be connected:

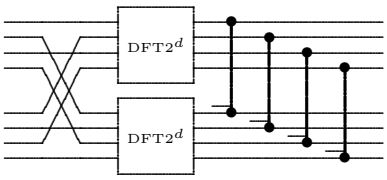

Fig. 2. Two connected FFT2^ds form an FFT2^{d+1}

This way it takes no more than $n \log_2 n$ multiplications to calculate the FTn, plus an additional permutation of the inputs. This subset of FT is called *Fast Fourier Transform*, or FFTn, which is why most implementations are FFT2^ds, restricting the input length to 2^d.

II. Divide and Conquer

The problem is this: What if $n \neq 2^d$? One solution is to increase the size of the input vector to the next power of 2, with either Zero-Padding or *Hawkins-Interpolation*[2]. However, these two approaches create new elements in the output vector which could result in slowing down later calculations, and are more vulnerable to rounding-errors. Hence it is much smarter to decompose n into its d prime-factors instead

$$n \quad = \quad p_1 \cdot p_2 \cdot \ \ldots \ \cdot p_\sigma \cdot \ \ldots \ \cdot p_d \quad (2)$$

and implement every stage as a special DFTp_σ, which takes np_σ multiplications. Consequently, it is faster than the matrix-multiplication, and more robust than approximation. Again, each stage re-uses the output of the previous one as the input. The connecting pattern is obvious when visualized as in Fig. 3:

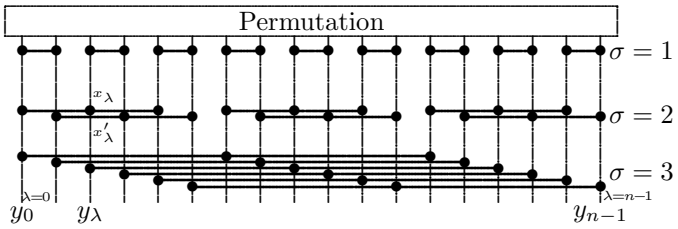

Fig. 3. Connecting the DFTp_σ

Even though it is possible to implement the DFTp_σ as butterflies, it is better to implement them as a multiplication with V_{p_σ}, which is in stage σ for "Column" λ

$$x'_\lambda = \sum_{k=0}^{p_\sigma - 1} x_{\alpha(\lambda,\sigma,k)} \cdot \omega_n^{\beta(\lambda,\sigma,k)} \quad (3)$$

III. Address Generating Units (AGUs)

The remaining problem of calculating the FT is to define the two address generating functions $\alpha(\lambda, \sigma, k)$ and $\beta(\lambda, \sigma, k)$. Comparison between the Vandermonde-matrix and the connect pattern reveales that those functions are

$$\alpha(\lambda,\sigma,k) \quad = \quad Lp_\sigma \cdot \left\lfloor \frac{\lambda}{Lp_\sigma} \right\rfloor + kL + (\lambda \bmod L) \quad (4)$$

$$\beta(\lambda,\sigma,k) \quad = \quad (kR(\lambda \bmod Lp_\sigma)) \bmod n \quad (5)$$

$$L = \prod_{i=1}^{\sigma-1} p_i \qquad R = \prod_{i=\sigma+1}^{d} p_i \quad (6)$$

Where L is the product of the already processed prime-factors, and R is the product of the remaining ones. α and β are valid for calculating a FTn of any size. However, because of the modulo-operations it is inefficient to implement α and β directly in hardware. A good solution is to calculate α and β offline and make them available as a lookup table or as special counters. Again, the easiest stages are those where $p_\sigma = 2$, hereby all the AGUs are regular Up-By-One-Counters.

IV. Implementation

To demonstrate the Divide and Conquer technique we use Digital Radio Mondiale (DRM) [3] as an example. Because

of the different length of the OFDM symbols, FT288, FT256, FT176 and FT112 are required; in addition several fractional length Inverse FTs (IFTs) are used.

In our implementation input samples are stored and processed in a central memory, which provides multiple input- and output-paths for data-access. We also used a strict separation of the data-path (Butterfly) and addressing logic (AGU), as shown in Fig. 4.

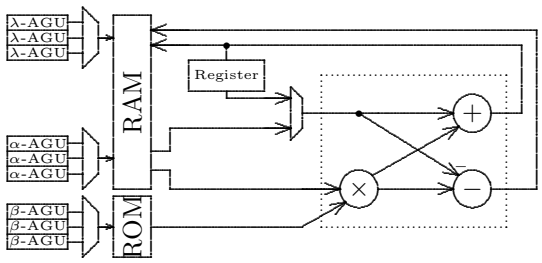

Fig. 4. A circuit for Custom Lengthed FTs

The butterfly together with a register and a multiplexer form the multiply-accumulate. Two α-AGUs and two read ports plus one β-AGU and a third read port provide the inputs, while two λ-AGUs and two write ports store the results. The AGUs are implemented as special counters.

Permutation of the input vector Figure 2 shows that a permutation of the input may be required. Due to using separate memories for in- and output data in our system, this permutation can be done on the fly, while copying input data to output memory. The algorithmn to get the correct permutation is a recursive one and is pre-calculated:

1. Start with an ordered set $0, \ldots, n-1$, and at stage $\sigma = d$
2. In stage σ create p_σ subsets, and put every p_σth element in one of them.
3. If the subsets contain only one element, finish. If not, decrease σ and repeat step 2 with each subset.

This will produce a tree-like structure, with the bottom leaves as the correct permutation. For a FT18 the tree would look like in Fig. 5:

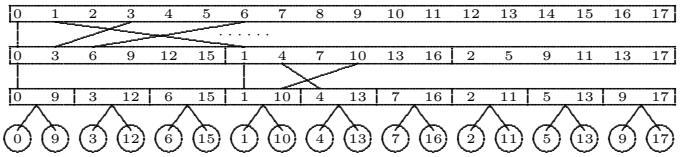

Fig. 5. The permutating tree for FT18

V. Results

Speed Table I shows the number of multiplications:

Table I: Number of multiplication

Algorithm	$\theta(n)$	FT288	FT256	FT176	FT112
Zeropadding	$\lceil n \rceil \log_2 \lceil n \rceil$	4608	2048	2048	896
Hawkins-Int.	$\lceil n \rceil \log_2 \lceil n \rceil$	4608	2048	2048	896
Vandermonde	n^2	82944	65536	30976	12544
Divide & Conquer	$\sum np_\sigma$	3168	2048	2640	1232

Regarding the number of multiplications it is beneficial for divide and conquer that the input length is not too close to the next power of 2 and the p_σ are small. If memory access is considered, a multiply-accumulate stage requires 2R/1W, whereas a butterfly-stage requires 3R/2W.

Table II: Number of memory-accesses

Algorithm	$\theta(n)$	FT288	FT256	FT176	FT112
Zeropadding	$\lceil n \rceil \log_2 \lceil n \rceil$	23040	10240	10240	4480
Hawkins-Int.	$\lceil n \rceil \log_2 \lceil n \rceil$	23040	10240	10240	4480
Vandermonde	n^2	248832	196608	92928	37632
Divide & Conquer	$\sum np_\sigma$	12384	10240	9328	4592

Thus our implementation is superior in the number of memory-accesses in most cases.

Precision With a real-life DRM-Signal as input, the four algorithms differed from GNU Octave's build-in FFT function by the values shown in table III.

Table III: Maximum absolute/relative differences

Algorithm $f(x)$		FT288	FT256	FT176	FT112
Zeropadding	float	66k/12	0/0	98k/82	79k/11
Hawkins-Int.	float	145k/161	0/0	179k/19	148k/41
Vandermonde	fixed	6.62/0.02	16.22/0	6.49/0.03	4.40/0
Divide&Conquer	fixed	8.46/0	11.55/0	7.65/0.03	6.37/0

Abs.: $\max_{k=1\ldots n}\{\Delta_k\}$, rel.: $\max_{k=1\ldots n}\{\Delta_k / FFT(x)_k\}$, with $\Delta_k = \|\mathrm{FFT}(x)_k - f(x)_k\|$, $x \in \mathbb{C}^n$ is the inputvector, $f(x)_k$ the k-th element in the outputvector

Both Zeropadding and Hawkins-Interpolation used floating-point. The fixed-point algorithms had Twiddle factors with 15 bits each, the input were 16-Bit IQ-Samples.

Silicon Compared to the butterfly, the extra hardware is a multiplexer, a register, and a ADDSUB-circuit. The latter is for calculating IFTs. The AGUs needed nine new counters, none of them having more than 12 bits. Because the internal values can gain an extra amount of $\log_2 n$ bits, our FFT288 produces 25 bit values (16 bits input $+\log_2 288$) resulting in a total accumulator size of 40 bits.

Memory Consumption Even though it is possible to implement the DFTp_σ-stages with no more than $\max_\sigma(p_\sigma)$ words memory usage, it was decided to use a temporary buffer. This buffer is four times the size of the input vector (2304 Bytes) and holds intermediate results with full precision. The Twiddle factors are stored in a ROM-table with a total of 3328 Bytes.

VI. Conclusion

This paper demonstrates the principles of implementing arbitrary FTn, applicable for both hard- and software systems. The proposed hardware has no length restrictions and only minimal overhead compared to the well-known FFT circuit. The same structure may be configured to implement different length FTn and IFTn; this provided significant savings in the design time for the DRM base band IP [4] that was used as an example in this paper.

References

[1] T. Cormen, C. Leierson, R. Rivest, C. Stein, "Introduction to algorithms, second edition", ISBN 0-262-03293-7, 2001, pp. 836

[2] "Nuclear Science Symposium and Medical Imaging Conference, 1994", ISBN 0-7803-2544-3, 1995, pp. 1433-1437

[3] EBU, "Digital Radio Mondiale (DRM), System Specification (V2.1.1)", ETSI Standard ES 201980, 2004

[4] F. Mayer, M. Schlicht, A. Heuberger, S. Melzer, "Chipset Development for Digital Radio Mondiale (DRM)", Proceedings of ICCE05, 10.-12.01.2005, Las Vegas, USA