(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2010-16832
(P2010-16832A)
(43)公開日　平成22年1月21日(2010.1.21)

(51)Int.Cl.		F I		テーマコード（参考）
H04J 11/00	(2006.01)	H04J 11/00	Z	5K022

審査請求 未請求　請求項の数 10　ＯＬ　外国語出願　（全 73 頁）

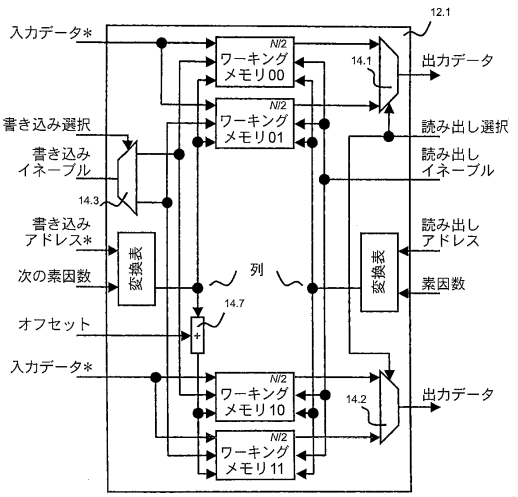
(21)出願番号　特願2009-160545 (P2009-160545)
(22)出願日　平成21年7月7日 (2009.7.7)
(31)優先権主張番号　08159859.1
(32)優先日　平成20年7月7日 (2008.7.7)
(33)優先権主張国　欧州特許庁 (EP)

(71)出願人　503163527
ミツビシ・エレクトリック・アールアンドディー・センター・ヨーロッパ・ビーヴィ
ＭＩＴＳＵＢＩＳＨＩ　ＥＬＥＣＴＲＩＣ　Ｒ&Ｄ　ＣＥＮＴＲＥ　ＥＵＲＯＰＥ　Ｂ．Ｖ．
オランダ国、１１１９　エヌエス・スヒプホール・レーイク、カプロニラーン　４６
Ｃａｐｒｏｎｉｌａａｎ　４６，　１１１９　ＮＳ　Ｓｃｈｉｐｈｏｌ　Ｒｉｊｋ，　Ｔｈｅ　Ｎｅｔｈｅｒｌａｎｄｓ
(74)代理人　100110423
弁理士　曾我　道治
(74)代理人　100084010
弁理士　古川　秀利

最終頁に続く

(54)【発明の名称】ルリタニアマッピングを用いるＰＦＡアルゴリズムに従って種々のサイズのＤＦＴを計算する装置及び方法

(57)【要約】　（修正有）
【課題】計算を加速するために、２つの計算コアを並列に使用してＤＦＴを計算することを提供する。
【解決手段】データは、ＰＦＡルリタニアマッピング行列の偶数行及び奇数行に従って、２つのコアのいずれかに送られる。コアごとに別個の記憶手段が使用され、基数計算ステップと基数計算ステップとの間に２つの別個の記憶手段間でデータを交換する。
【選択図】図１４Ａ



【特許請求の範囲】
【請求項１】
ルリタニアマップを用いるＰＦＡアルゴリズムに従って種々のサイズのＤＦＴを計算する装置であって、
－　フロー上の種々のサイズを有するすべてのバタフライを計算する２つのバタフライ計算ユニットと、
－　前記ＤＦＴのサイズの半分に等しいサイズを有するデータを記憶する２つの別個の記憶手段と、
－　前記ルリタニアマップの偶数行及び奇数行に従って、前記２つの別個の記憶手段内に入力データを送る手段と、
－　ルリタニアマップが異なることに起因して、基数が変化するごとに、データを再分配する手段と、
－　前記２つの別個の記憶手段から出力データを取り出す手段と、
を備えることを特徴とする、ルリタニアマップを用いるＰＦＡアルゴリズムに従って種々のサイズのＤＦＴを計算する装置。
【請求項２】
前記記憶手段のそれぞれは、基数サイズが変化するときに入力演算及び出力演算のために交互に使用される２つの異なるメモリをさらに備えることを特徴とする、請求項１に記載の装置。
【請求項３】
トランスペアレントなアクセスのために前記２つの記憶手段をカプセル化する仮想化手段をさらに備えることを特徴とする、請求項１又は２に記載の装置。
【請求項４】
前記ルリタニアマップの変更に基づくアドレス変換手段をさらに備えることを特徴とする、請求項３に記載の装置。
【請求項５】
データの再分配は、各バタフライユニットからの前記出力データに対して同様の再配列を適用すると共に、該２つの再配列されたストリームにおいて対交換を行うことによって達成されることを特徴とする、請求項１に記載の装置。
【請求項６】
ルリタニアマップを用いるＰＦＡアルゴリズムに従って種々のサイズのＤＦＴを計算する方法であって、
－　２つのバタフライ計算ユニット上で、フロー上の種々のサイズを有するすべてのバタフライを計算するステップと、
－　前記ＤＦＴのサイズの半分に等しいサイズを有するデータを、２つの別個の記憶手段内に記憶するステップと、
－　前記ルリタニアマップの偶数行及び奇数行に従って、前記２つの別個の記憶手段内に入力データを送るステップと、
－　ルリタニアマップが異なることに起因して、基数が変化するごとに、データを再分配するステップと、
－　前記２つの別個の記憶手段から出力データを取り出すステップと、
を含むことを特徴とする、ルリタニアマップを用いるＰＦＡアルゴリズムに従って種々のサイズのＤＦＴを計算する方法。
【請求項７】
前記データを記憶するステップは、基数サイズが変化するときに入力演算及び出力演算のために交互に使用される２つの異なるメモリ内の各記憶手段内に交互にデータを記憶するステップをさらに含むことを特徴とする、請求項６に記載の方法。
【請求項８】
トランスペアレントなアクセスのために前記２つの記憶手段をカプセル化する仮想化ステップをさらに含むことを特徴とする、請求項６又は７に記載の方法。

【請求項９】
前記ルリタニアマップの変更に基づくアドレス変換ステップをさらに含むことを特徴とする、請求項８に記載の方法。
【請求項１０】
データの再分配は、各バタフライユニットからの前記出力データに対して同様の再配列を適用すると共に、該２つの再配列されたストリームにおいて対交換を行うステップによって達成されることを特徴とする、請求項６に記載の方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、包括的には、離散フーリエ変換（ＤＦＴ）に関する。より具体的には、本発明は、ＤＦＴ拡散直交周波数分割多重（ＤＦＴ拡散ＯＦＤＭ）変調技法のディジタル回路上でのハードウェア実施（実装）に関する。
【背景技術】
【０００２】
ＤＦＴ拡散ＯＦＤＭは、次世代（beyond third generation）（Ｂ３Ｇ）セルラネットワーク向けに第３世代パートナーシッププログラム（３ＧＰＰ）によって発行された長期的発展型（ＬＴＥ）標準規格において、（端末から基地局への）アップリンク送信のためのものとして指定されている。ＤＦＴ拡散ＯＦＤＭでは、変調シンボルは、従来のＯＦＤＭ変調に先立ってＤＦＴによって拡散され、ピーク対平均比（ＰＡＰＲ）を低くすることによる利益を得る。ＬＥＴの命名規則に従ってプリコーディング変換とも呼ばれるこの拡散の目的は、端末の電力消費を削減することである。３ＧＰＰ－ＬＴＥ標準規格では、ＤＦＴ拡散ＯＦＤＭは、周波数分割多元接続（ＦＤＭＡ）技法と組み合わせられて、いわゆる単一搬送波周波数分割多元接続（ＳＣ－ＦＤＭＡ）技法を形成する。ＳＣ－ＦＤＭＡでは、各ユーザには、送信時間間隔（ＴＴＩ）にわたって、２０ＭＨｚ帯域幅の１つのサブバンドから１１０個のサブバンドまで、１２個の連続した副搬送波で構成される所与の個数の連続したサブバンドが割り当てられる。ＤＦＴのハードウェア複雑度を低減するために、ＤＦＴサイズのサブセットのみが指定され、より具体的には、１２の倍数の４３個の異なるサイズＮ（１２から１９４４まで）が指定される。ここで、$N = 2^{p} \times 3^{r} \times 5^{s}$であり、ｐ、ｑ、及びｓは整数である。
【０００３】
ＤＦＴは、ジョセフ・フーリエによるフーリエ級数の導入にちなんで命名されたフーリエ変換の１つの特定の形態である。その定義は、John G. Proakis及びDimitris G. Manolakis著「Digital Signal Processing, principles, algorithms, and applications」（New-York: Macmillan Publishing Company, 1992, ch. 9）に見つけることができる。ＤＦＴは、有限時間の関数（Ｎ個の値）に適用され、その結果は、Ｎ個の等間隔の離散周波数の集合におけるフーリエ変換の値を求めることによって計算される。Ｎ個の複素数のシーケンス｛x_k，０≦ｋ≦Ｎ－１｝は、公式：
【０００４】
【数１】

$$X_n = \sum_{k=0}^{N-1} x_k e^{-2i\pi \frac{p}{N}}, \quad 0 \le n \le N-1$$

【０００５】
に従うＤＦＴによってＮ個の複素数のシーケンス｛X_n，０≦ｎ≦Ｎ－１｝に変換される。ここで、ｅは自然対数の底であり、ｉは複素数の虚数単位（$i^2 = -1$）である。ＤＦＴの特性の多くは、通例、ひねり因子（twiddle factor）と呼ばれる係数
【０００６】

【数２】

$$w_N^p = e^{-2i\pi\frac{p}{N}}$$

【０００７】

が、１の原始累乗根であるということに依存する。その基本形態では、Ｎ点ＤＦＴアルゴリズムの計算は、Ｏ（Ｎ^2）の複素乗算を必要とする。ハードウェアでは、ＤＦＴは通常、演算回数がより少ない高速フーリエ変換（ＦＦＴ）と通例呼ばれる専用の変換を使用して計算される。ＦＦＴアルゴリズムの大部分は、クーリー－テューキーアルゴリズムから導出された方法、素因数アルゴリズム（ＰＦＡ）から導出された方法、多項式因数分解法、及び畳み込み法の４つに分類することができる。これらの方法は互いに排他的なものではない。すなわち、これらの方法のうちの１つを別の方法の中で使用して、全体的な複雑度を削減することができる。

【０００８】

クーリー－テューキーアルゴリズムは、ＤＦＴ変換を計算するための基準アルゴリズムである。その記載を、James W. Cooley及びJohn W. Tukey著「An algorithm for the machine calculation of complex Fourier series」（Math. Comput., vol. 19, 1965, pp. 297-301）に見つけることができる。その最もポピュラーな形態では、クーリー－テューキーアルゴリズムは、サイズＮのＤＦＴをサイズＮ／２の２つの変換に再帰的に分割することに本質がある（分割統治手法）。クーリー－テューキーアルゴリズムを導出する単純な手法は、偶数番号の周波数サンプル及び奇数番号の周波数サンプルの計算を分けることである。同じ原理が各ステップで適用されるので、結果のＦＦＴアルゴリズムは、周波数間引き変換（ＤＩＦ、サンデ－テューキーアルゴリズムとも呼ばれる）と呼ばれる。本質的に２の累乗のサイズ（Ｎ＝２^n）に制限されるこの形態は、基数２のＤＩＦクーリー－テューキーＦＦＴアルゴリズムとして知られている。また、再帰的変換を時間サンプルに適用して、時間間引き（ＤＩＴ）バージョンを得ることもできる。また、ＤＦＴは、以下ように、行列・ベクトル乗算として表すこともできる。

【０００９】

【数３】

$$Z = W_N Y$$

$$Y = [x_0\, x_1 \ldots x_{N-1}]^T, Z = [X_0\, X_1 \ldots X_{N-1}]^T$$

$$W_N = \left[w_N^{ij}\right]_{i=0,\ldots,N-1, j=0,\ldots,N-1}$$

【００１０】

ここで、ｉは行のインデックスを示し、ｊは列のインデックスを示す。行列W_Nは、Ｎ次のＤＦＴ行列と呼ばれる。そうすると、ＦＦＴアルゴリズムは、W_Nを疎行列のシーケンスの積に因数分解することによって定義される。基本的な基数２のＤＩＦクーリー－テューキーＦＦＴアルゴリズムは、ジェントルマン－サンデアルゴリズムを使用して行列因数分解を通して得ることができる。ＤＦＴの計算では、クーリー－テューキーアルゴリズムを、$\log_2$（Ｎ）個のステージを表すグラフとして表すことが一般的である。ここで、各ステージは、Ｎ／２個のバタフライと呼ばれる基本的な２点ＤＦＴで構成される。各ＤＩＦバタフライは、加算及び減算に還元された後、ひねり因子による複素乗算を通して回転される。したがって、２の累乗のサイズＮの場合、クーリー－テューキーＤＦＴの複雑度は、（Ｎ／２）×$\log_2$（Ｎ）個の複素乗算、及びＮ×$\log_2$（Ｎ）個の複素加算に還元され、基本的なＤＦＴのＯ（Ｎ^2）の複雑度よりもはるかに低くなる。

【００１１】

クーリー－テューキーアルゴリズムの１つの利点は、バタフライのシーケンスであるコアアルゴリズムの計算を、インプレース（in-place）で実行することができる、すなわち、記憶装置を追加することなく実行することができるということである。確かに、バタフライの出力が計算されたとき、入力対を保存する必要はない。したがって、バタフライの

結果は、２つの入力と同じロケーションに記憶することができる。コンピュータサイエンスでは、インプレースアルゴリズムは、少量且つ一定量の追加の記憶空間を使用してデータ構造を変換するアルゴリズムである。この定義によれば、クーリー－テューキーは、インプレースであるとみなすことができる。この利点は、クーリー－テューキーが自己整列型（self-sorting）でない、すなわち、出力サンプルが線形順序で生成されないことによって相殺される。ベクトルを操作する関数は、線形に順序付けられた入力から線形に順序付けられた出力を生成する場合に自己整列型であると言われる。クーリー－テューキーは、部分ＤＦＴの奇数番号のサンプル及び偶数番号のサンプルの再帰的計算に起因する間引きのために自己整列型ではない。したがって、複雑度及び／又はレイテンシが増加することを犠牲にして、変換を達成するためのスクランブル解除演算を実行することが必要となる。基数２のＦＦＴの場合、この演算は、出力サンプルのアドレスのビット反転の形を取る。ビット反転は、数字$b_2b_1b_0$（たとえば、Ｎ＝８個の入力の３つの数字）を有する２進数で記述されたインデックスｎのデータが、反転された数字$b_0b_1b_2$を有するインデックスに変換される置換である。実際には、クーリー－テューキーアルゴリズムは、２の累乗のサイズに制約されない。クーリー－テューキーアルゴリズムは、任意のサイズの形$N=I^n$に拡張することができる。ここで、Ｉは整数である。一般的な例は、３の累乗のサイズに適用可能な基数３のクーリー－テューキーＦＦＴ、及び４の累乗のサイズに適用可能な基数４のクーリー－テューキーＦＦＴである。これらの場合にも、スクランブル解除ステップは、反転演算によって行うことができるが、基数３及び基数４で表されるアドレスに適用される。実際、クーリー－テューキーアルゴリズムは、サイズＮから任意の整数の積$N=n_1\times n_2\times n_3\times\cdots\times n_L$への任意の因数分解に適用することができる。これによって、いわゆる混合基数ＦＦＴアルゴリズムがもたらされる。この場合、スクランブル解除ステージは、より複雑な数字反転の形を取り、それらのアルゴリズムの実際の実施は制限される。

【００１２】

１９８４年に、ジョンソン及びビュルスは、「An in-place in-order radix-2 FFT」（Proc. IEEE ICASSP, 1984, pp. 28A.2.1-4）において、自己整列型で且つインプレースのＦＦＴアルゴリズムを得るように基数２のＤＩＴクーリー－テューキーアルゴリズムを書き換えることができることを示した。クライブ・テンパートンは、「Self-sorting in place fast Fourier transforms」（SIAM J. Sci. Stat. Comput., vol. 12, July 1991, pp. 808-823）において、インプレースで且つ自己整列型のクーリー－テューキーＦＦＴを作成するのに使用される原理を、他の基底、たとえば基数３のアルゴリズム及び基数５のアルゴリズムに一般化することができることを示した。グラフでは、この変換は、インターリーブされた出力を有する複数のタイプ１バタフライ（従来のバタフライ）をグループ化するタイプ２バタフライで構成された連続した複数のステージで構成される。これはさらに、Ｎの因数分解が回文（palindrome）を形成していれば、混合基数ＦＦＴに拡張することができる。したがって、このアルゴリズムは、$N=144=3^2\times4^2$の場合、因数が３、４、４、３の順序又は４、３、３、４の順序で使用されれば機能する。多くの変換長において、自己整列型インプレースクーリー－テューキーアルゴリズムに必要とされる対称形式で因数を配列することは不可能である。これらのサイズに対する代替手段は、回文として記述できない因数を因数分解の中央に配置することである。たとえば、$N=6000=3\times4^2\times5^3$の場合、因数を、次のように、すなわちＮ＝６０００＝４×５×１５×５×４のように配列することができる。

【００１３】

クーリー－テューキーアルゴリズムに加えて、この文書で説明する発明は、グッドによって、「The interaction algorithm and practical Fourier analysis」（J. Roy. Statist. Soc. Ser. B., vol. 20, 1958, pp. 361-372）において最初に紹介されたＰＦＡアルゴリズムを使用する。素因数ＦＦＴアルゴリズムの掘り下げた説明は、ビュルスによって、「Index mappings for multidimensional formulation of the DFT and convolution」（IEEE Trans. Acoust. Speech Signal Process., vol. 25, 1977, pp. 239-242）、及

び、「An in-place, in-order prime factor FFT algorithm」（IEEE Trans. Acoust. Speech Signal Process., vol. 29, 1981, pp. 806-817）に与えられている。ＰＦＡは、互いに素な因数で構成された合成サイズにおけるクーリー－テューキーアルゴリズムの一変形とみなすことができる。このクーリー－テューキーアルゴリズムは、サイズ$N_1 \times N_2$のＤＦＴをサイズN_1及びN_2のより小さな多くのＤＦＴに再帰的に分割することに本質がある。これらの演算は、入力サンプル及び出力サンプルの適切な再インデックスを通して形式化することができる。これは、インデックスｋ及びｎを

【００１４】

【数４】

$$k = N_1 k_2 + k_1 \quad \text{ここで } 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1$$
$$n = N_2 n_1 + n_2 \quad \text{ここで } 0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1$$

【００１５】

と書き換えることによって達成される。第１の演算は、N_1行N_2列で構成される２次元アレイとして入力（ｋ）を再インデックスすることとして解釈することができる。ここで、データのオリジナルセットは、このアレイの列に線形に記憶される。第２の演算は、N_1行N_2列で構成される２次元アレイとして出力（ｎ）を再インデックスする。ここで、データの期待されるセットは、行に線形に記憶される。この再インデックスが、ＤＦＴの公式のｎｋに代入されると、残る項は

【００１６】

【数５】

$$X_{N_2 n_1 + n_2} = \sum_{k_1=0}^{N_1-1} \left[e^{-\frac{2j\pi}{N} k_1 n_2} \right] \left[\left(\sum_{k_2=0}^{N_2-1} x_{N_1 k_2 + k_1} e^{-\frac{2j\pi}{N_2} k_2 n_2} \right) e^{-\frac{2j\pi}{N_1} k_1 n_1} \right]$$

【００１７】

を与える。このクーリー－テューキーアルゴリズムは、合成サイズ$N = N_1 N_2$のＤＦＴを

・アレイの列をシーケンシャルに満たして、入力を$N_1 \times N_2$アレイとして再インデックスすること。

・サイズN_2のN_1個のＤＦＴを、アレイのN_2個の列に対して実行すること。

・ひねり因子を乗算すること。

・サイズN_1のN_2個のＤＦＴを、アレイのN_1個の行に対して実行すること。

・アレイの行をシーケンシャルに読み出して、出力を再インデックスすること。

として、再帰的に表現し直していると見ることができる。ＰＦＡは、整数（ｋ）及び（ｎ）と、ＤＦＴアルゴリズムの２次元解釈の対応する整数対（k_1, k_2）及び（n_1, n_2）との間の特定のマッピングに依拠する。実際のところ、I. J. Good著「The relationship between two Fast Fourier Transforms」（IEEE Trans. Comp., vol. 20, 1971, pp. 310-317）に記載されているような、中国の剰余定理マップ（ＣＲＴ）に基づくマッピング又はルリタニアマップ（Ruritanian map）（グッドのマップとも呼ばれる）に基づくマッピングのいずれかの少なくとも２つのマッピングを使用することができる。入力及び出力に異なるマッピングを使用することもできる。ＰＦＡ変換は、ここでは入力及び出力の双方がルリタニアマップである場合について紹介される。このルリタニアマップは

【００１８】

【数６】

$$n_1 = (pn) \bmod N_1,\ n_2 = (qn) \bmod N_2,$$
$$k_1 = (pk) \bmod N_1,\ k_2 = (qk) \bmod N_2,$$

【００１９】

によって定義される。ここで、整数ｐ及びｑは、N_1とN_2が互いに素である場合に

【００２０】

【数７】

$$pN_2 = rN_1 + 1, \quad \text{ここで } 0 < p < N_1, 0 < r < N_2$$

$$qN_1 = sN_2 + 1, \quad \text{ここで } 0 < q < N_2, 0 < s < N_1$$

【００２１】

となるような整数ｐ、ｑ、ｒ、ｓを見つけることができるとするＣＲＴ定理に従って規定される。逆マップは

【００２２】

【数８】

$$n = (N_2 n_1 + N_1 n_2) \bmod N$$

$$k = (N_2 k_1 + N_1 k_2) \bmod N$$

【００２３】

によって与えられる。Ｎ＝４０（N_1＝８及びN_2＝５）の一例は、以下の表に示される。

【００２４】

【表１】

	0	1	2	3	4
0	0	8	16	24	32
1	5	13	21	29	37
2	10	18	26	34	2
3	15	23	31	39	7
4	20	28	36	4	12
5	25	33	1	9	17
6	30	38	6	14	22
7	35	3	11	19	27

【００２５】

ルリタニア式の解を見つける必要はなく、容易にマッピングを構築できることが見て取れる。第１列のエントリーは、０からＮ／N_1（＝N_2）ずつ増加する。一方、第１行のエントリーは、０から（Ｎ／N_2＝N_1）ずつ増加する。次に、残りの列（又は行）は、第１列（又は行）と同じ増分を使用しつつ、結果のモジュロＮを取ることによって埋めることができる。このマッピングをＤＦＴのオリジナルの式に適用することによって、

【００２６】

【数９】

$$X(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left(\sum_{k_1=0}^{N_1-1} x(k_2, k_2) e^{-\frac{2j\pi}{N_1} N_2 k_1 n_1} \right) e^{-\frac{2j\pi}{N_2} N_1 k_2 n_2}$$

【００２７】

が与えられる。N_1又はN_2を指数部に乗算していることが見て取れるが、これを別にすると、この式はN_1×N_2の２次元ＤＦＴの形を取る。クーリーテューキーアルゴリズムと同様に、長さN_1のN_2個のＤＦＴを一方の次元で実行した後、他方の次元で長さN_2のN_1個のＤＦＴを実行することによって、ＤＦＴを計算することができる。この変換は、他方の次元から開始して計算することもできる。しかしながら、これら２つのステージ間にひねり因子は存在せず、これによって、演算回数がクーリーテューキーＦＦＴの場合よりも少なくなることを強調しなければならない。指数部にN_1及びN_2を乗算することによって、入力サンプルに対して適用される回転方法が単純に変更される。ランクｎを有する周波数サンプルは、０から開始して増分ｎで単位円の周囲を巡回する１の累乗根によって加重された時間サンプルの線形結合として計算される。因数N_a（ａ＝１又は２）を適用す

ると、増分は単純にｎからｎ×N_aへ変更される。フーリエ変換の特性に起因して、出力サンプルは、標準的なＤＦＴの出力サンプルに対して循環シフトを単純に適用することによって得ることもできる。すなわち、長さN_aの変換に回転ｒを適用することは、結果がオリジナルの順序０，１，…，N_a－１で現れる代わりに、順序０，ｒ，２ｒ，…，（N_a－１）で現れることを意味する。ここで、インデックスはモジュロN_aで解釈される。入力及び出力の双方について同じマッピングを使用することによって、ＰＦＡをインプレース且つ自己整列型にすることができる。確かに、マッピングが同じであるので、出力サンプルは入力サンプルと同じ順序で現れる。１次元アレイからルリタニア２次元表現を形成する再インデックス演算が、暗黙的なアドレス変換を通して行われる場合、出力サンプルはワーキングアレイにおいて線形順序で現れる。すなわち、このアルゴリズムは自己整列型である。クーリー－テューキーアルゴリズムと同様に、ＰＦＡ法を再帰的に適用することによって、サイズＮを互いに素な因数$N = n_1 \times n_2 \times n_3 \times \cdots \times n_L$の積に分解することができる。この場合、ＰＦＡアルゴリズムを行列の形で表すことがより適切である。ＰＦＡアルゴリズムの原理を再帰的に適用することによって、行列W_Nの以下の因数分解が導かれる。

【００２８】

【数１０】

$$W_N = R^{-1}\left(W_{n_L}^{[r_L]} \times \ldots \times W_{n_2}^{[r_2]} \times W_{n_1}^{[r_1]}\right)R, \text{ ここで } r_i = \frac{N}{n_i} \bmod n_i \quad 1 \le i \le L$$

【００２９】

ここで、Ｒは、ルリタニアマップに従って整数ｎを対応するＬ次元空間（n_1，n_2，…，n_L）にマッピングする置換行列であり、$W_N^{[r]}$は、行列W_Nのすべての要素がｒ乗されたものである。上述したように、実際のところ、置換行列との積を実際に計算する必要はない。必要とされるマッピングは、インデクシングロジックを介して暗黙的に実施することができる。

【００３０】

ＰＦＡの１つの問題は、サイズを互いに素な因数に分解すると、単純な基数変換による直接計算に適合しない大きな因数を含む場合があるということである。したがって、他のＤＦＴ変換のうちの１つを使用してこれらの変換を実施することが必要となる。一般的な合成サイズの場合には、クーリー－テューキーアルゴリズムを使用して、より小さなＤＦＴを計算することができる。ジョンソン－ビュルス及び「Implementation of self-sorting in-place prime factor FFT algorithm」（J. Comput. Phys., vol. 58, 1985, pp. 283-299）におけるテンパートンの研究のおかげで、自己整列型で且つインプレースの形式でクーリー－テューキー変換を計算することが可能である。この拡張を、ＰＦＡアルゴリズムにおける合成因数（composite factor）の計算に適用することによって、一般化されたジョンソン－ビュルス（自己完結型インプレースクーリー－テューキー）変換の厳密な適用よりも低い計算複雑度で、自己整列型で且つインプレースのＤＦＴアルゴリズムを定義することが可能になる。このアルゴリズムは、テンパートンによる「A generalized prime factor FFT algorithm for any $2^p3^q5^r$」（SIAM J. Sci. Stat. Comput., vol. 13, May 1992, pp. 676-686）において、一般化されたＰＦＡ（ＧＰＦＡ）アルゴリズムとして、$N = 2^p \times 3^q \times 5^r$（ｐ、ｑ、ｒは整数）の形のサイズの場合について記載されている。このアルゴリズムは、サイズが互いに素な整数の累乗の積である任意の分解に拡張することができる。これは、ＰＦＡアルゴリズムの回転因子を取り扱うようにクーリー－テューキーアルゴリズムを変更することによって可能になる。テンパートンは、以下の変更、すなわち

・回転ｒ（モジュロｐ）を各基数ｐのモジュールに適用すること。

・すべてのひねり因子をｒ乗すること。

を適用することによって、

行列

【００３１】
【数１１】

$$W_{N=p^m}^{[r]}$$

【００３２】
を有する基数ｐのＤＦＴを、
行列
【００３３】
【数１２】

$$W_{N=p^m}$$

【００３４】
を有するＤＦＴのように計算することができることを示した。たとえば、
【００３５】
【数１３】

$$W_{N=p^m}^{[r]}$$

【００３６】
の場合、Ｎ＝ｐｍは、各基数５のモジュールをｒ’＝４（９モジュロ５）だけ回転した後、すべてのひねり因子を９乗することによって実施することができる。これ以降、この変換が適用される変換の名称に「回転（rotated）」と付加することによって、この変化を識別する。
【００３７】
任意のＮ＝$2^p\times 3^q\times 5^r$に対して、一般化された自己整列型インプレースの素因数ＦＦＴアルゴリズムは、次のように構成される。
・ルリタニアマップを使用して、サイズ（2^p，3^q，5^r）の３次元アレイとして入力サンプルを再インデックスする。
・一般化された回転ジョンソン－ビュルス法を使用して、長さ2^p，3^q，5^rの１次元変換を計算する。したがって、より小さなＤＦＴは、インプレース自己整列型形式で実施される。
・ルリタニアマップを使用して、サイズ2^p，3^q，5^rの１次元変換として出力サンプルを再インデックスする。
【００３８】
ＰＦＡに適したＮの値の場合、このアルゴリズムはＰＦＡに還元され、同じ演算回数を有する。Ｎ＝2^p、3^q、又は5^rの場合、このアルゴリズムはクーリー－テューキーアルゴリズムの自己整列型インプレースの導出物に還元され、同じ演算回数を有する。Ｎが因数の混合を含むがＰＦＡに適していない場合、このアルゴリズムは基本的なクーリー－テューキーアルゴリズムよりも少ない演算回数を有する。
【００３９】
この文書に提示される発明は、ディジタル集積回路上でのＧＰＦＡアルゴリズムのハードウェア実施を扱う。ＤＦＴの広範囲の適用に関連して、多くの解決法がＦＦＴアルゴリズムの実施に利用可能である。これらの解決法の大部分は、プロセッサ上で実施する場合のクーリー－テューキーアルゴリズム及びその導出物に焦点を当てている。しかしながら、速度が極めて重要となる状況では、多くの場合、ＦＦＴアルゴリズムは集積回路上で実施される。ＤＦＴのクーリー－テューキー因数分解は、すべてのクーリー－テューキーＦＦＴ装置に何らかの形で適用されるが、これらのアルゴリズムを実施するための解決法には大きな多様性がある。集積回路によって、本来的に機能を並列化することが可能になる。したがって、所与のサイズのＦＦＴのすべてのバタフライを、そのグラフ表現と同様に実施することが可能である。これによって、削減されたクロックサイクル数で結果を計算することが可能になるが、複雑度が高くなるという犠牲が伴う。実際、複雑度とレイテン

シとの間のトレードオフは、ＦＦＴの種々の実施態様間の相違を生み出すものとなる。加えて、ハードウェア装置の仕様に応じて、解決法は、メモリ使用量、及びバタフライを実施するための乗算器の個数と加算器の個数と比の点からも異なる。

【００４０】

基数２のクーリー－テューキーＦＦＴは、その人気及びかなり低い複雑度のために、ＤＦＴをハードウェアで実施するための圧倒的に最も広く使用されている技法のうちの１つである。多くの状況では、ＤＦＴのサイズはあまり重要ではない。したがって、余分な複雑度及び／又はレイテンシを犠牲にするものの、ゼロパディングによって２の累乗のＦＦＴを使用することができる。ＯＦＤＭ送信のような他の状況では、システムの仕様は、多くの場合、この制約条件を考慮して規定される。基数２のクーリー－テューキーＦＦＴアルゴリズムをハードウェア実施に適合させるものは、そのグラフ表現に示すようなモジュール性である。これは実際には、Ｎ／２個の同様のバタフライのlog_2（Ｎ）個の連続したステージで構成される。上記で紹介したように、最も直接的な解決法は、グラフ表現と同様にすべてのバタフライを実施することである。これによって、低いレイテンシでＤＦＴを計算することが可能になる。また、実際にはあまり一般的ではないが、データが並列に提示される場合には、連続したＤＦＴをパイプライン化することも可能になる。この解決法の明らかな欠点は、各ステップの出力を記憶する必要があるので、計算資源（乗算器及び加算器）並びにメモリの双方の点で複雑度が高いことである。

【００４１】

排他的でない２つの異なる手法を使用して、全体的な複雑度を削減することができる。第１の手法は、同じハードウェア資源を再利用して、一時に１つのステージを計算することである（時に列ＦＦＴ（column FFT）と呼ばれる解決法）。結果は、プロセス要素の同じセットにフィードバックされ、次のステージが計算される。これによって、メモリを節約することが可能になるが、レイテンシが長くなると共にルーティングアーキテクチャがより複雑になる犠牲が伴う。他方の手法は、各ステージ内において、異なるバタフライの計算をパイプライン化することによって、処理要素の個数を最終的には１つに削減することである（時にパイプライン化ＦＦＴ（pipelined FFT）と呼ばれる解決法）。この解決法は、データがＦＦＴ装置にシリアルな方法で適用される状況に特に適している。この状況は、実際には非常に一般的な状況である。

【００４２】

基数２のＦＦＴアルゴリズムの最も直接的なパイプライン化された実施態様は、L.R. Rabiner及びB. Gold著「Theory and Application of Digital Signal Processing」（Prentice-Hall, Inc., 1975）に記載されている基数２マルチパス遅延コミュテータ（Ｒ２ＭＤＣ）である。入力シーケンスは、順方向に流れる２つの並列データストリームに分割され、ステップごとに一意のバタフライに入力されるデータ要素間の正確な「距離」が、適切な遅延によってスケジューリングされる。バタフライ及び乗算器の双方の利用率は５０％である。この実施態様は、log_2（Ｎ）－２個の乗算器、log_2（Ｎ）個の基数２のバタフライ、及び３／２Ｎ－２個のレジスタを必要とする。各ステージ内において、単一のバタフライ及びＮ個の複素要素の単一のアレイによってＦＦＴを実施することが実際に可能である。E. H. Wold及びA. M. Despain著「Pipeline and parallel-pipeline FFT processors for VLSI implementation」（IEEE Trans. Comput., vol. C-33(5), May 1984, pp. 414-426）に記載されているように、基数２の単一パス遅延フィードバック（Ｒ２ＳＤＦ）は、バタフライの出力をフィードバックシフトレジスタに記憶することによって、レジスタをより効率的に使用する。単一のデータストリームは、すべてのステージの乗算器を通過する。このＲ２ＳＤＦは、Ｒ２ＭＤＣ手法と同じ個数のバタフライユニット及び乗算器を有するが、メモリ所要量は大きく削減されている（Ｎ－１個のレジスタ）。この手法は、圧倒的に最も一般的なものであり、複雑度をさらに削減する基数４の導入によって多くの変形（Ｒ４ＭＤＣ、Ｒ４ＳＤＦ、Ｒ４ＳＤＣ、$R2^2SDC$）を有する。

【００４３】

基数２又は基数４以外の他のステージ、たとえば基数３又は基数５を単純に追加するこ

とによって、パイプライン化クーリー－テューキーＦＦＴアルゴリズムを、混合基数ＦＦＴを計算するように変更することができる。また、いくつかのステージをスキップすると共に異なるステージ間のルーティングを適切にアレンジすることによって、ＤＦＴのサイズを動的に変更することも可能である。しかしながら、基数２のバタフライは実施がかなり単純であるのに対して、他の基数のバタフライの複雑度は大幅に高くなる。したがって、基数２又は基数４以外のステージの個数に依存して、ＦＦＴ装置の複雑度は扱いにくいものとなる。実際には、これらの解決法の主な制限は、それらの解決法が自己整列型でない基本的なクーリー－テューキーアルゴリズムを実施するということである。したがって、余分なメモリ及び／又はレイテンシのいずれかを消費するスクランブル解除ステージを実施することが必要となる。このスクランブル解除ステージも、混合基数ＦＦＴの場合には、かなり複雑になる可能性がある。

【００４４】

より高次の基数を使用することの別の制限は、基本的なＤＦＴの実施及びひねり因子による回転の双方について、回転を実施する必要があるということである。上述したように、これらの場合には、入力データ及び出力データの適切なインデクシングを使用することによってひねり因子による乗算を除去するＰＦＡアルゴリズムに依拠することがより優れている。ＰＦＡアルゴリズムの別の利点は、自己整列型で且つインプレースとなるようにＰＦＡアルゴリズムを実施でき、したがって、クロックサイクル及びメモリの双方を節約することができるということである。問題は、ＰＦＡアルゴリズムが、互いに素な因数に因数分解できるサイズでないと適用できないということである。これは、１つの所与の長さのＤＦＴの計算を必要とするアプリケーションにとっては問題ではない。

【００４５】

Ｌ．Zou及びX．Huangによる２００５年８月の欧州特許出願第０５３００６５１．６号「3780-point Discrete Fourier Transformation processor」に記載されているように、３７８０点ＤＦＴの場合の解決法は、互いに素な因数に対応する異なるＤＦＴモジュールをパイプライン化することであり、ここでは１４０点ＤＦＴ及び２７点ＤＦＴをパイプライン化する。１４０点ＤＦＴは、４点ＤＦＴモジュール、５点ＤＦＴモジュール、及び７点ＤＦＴモジュールに分解された、ネストウィノグラード変換（nested Winograd transform）を使用して計算される。一方、２７点ＤＦＴは、９点ＤＦＴモジュール及び３点ＤＦＴモジュールに分解されたクーリー－テューキーアルゴリズムを使用して計算される。ＰＦＡアルゴリズムに依拠することの利点は、ひねり因子による乗算を回避することである。しかしながら、異なるステージをパイプライン化すると、いくつかのバッファが各ステージ間で必要とされるので、ＰＦＡアルゴリズムのインプレース性を活用することができない。加えて、クーリー－テューキーアルゴリズムとパイプライン化手法とを組み合わせることによって、スクランブル解除ステージを実施することが必要となる。

【００４６】

特に、サイズを動的に変更することを取り扱う場合には、ＰＦＡアルゴリズムにおいてパイプライン化手法を使用することの利点から利益を得ることはかなり難しいように見える。

【００４７】

関心のある解決法は、ＦＦＴの列手法を使用することである。この手法は、単一の記憶要素をループすることによって、処理要素の同じセットを使用して異なるステージを処理することに本質がある。この解決法は、R．S．Buchert、S．M．Sharier、及びP．Beckerによる２００６年４月の米国特許第７，０２８，０６４号「Optimized discrete Fourier transform method and apparatus using prime factor algorithm」で実施されている。ここでは、「ＰＦＡ回路」と呼ばれる単一の処理要素が、互いに素な因数へのサイズ分解に関連付けられる部分ＤＦＴを実行するのに使用される。この装置は、プログラマブル乗算・加算（ＭＡＣ）ユニットの一種を使用して部分ＤＦＴを実施する。実際には、この装置では、ひねり因子の本来的な対称性を活用するために、互いに結合されたこのような２つのＭＡＣユニットが組み込まれ、これによって全体的な複雑度が削減される。この解決

法は、サイズを比較的小さな素因数に分解することができる限り（これは、ＰＦＡアルゴリズムの本来的な制限である）非常に魅力的である。確かに、分解が$N_i = 2^m$という形の因数を含む場合、2^m点ＤＦＴはＭＡＣユニットを使用して計算され、これは計算効率がよくない。加えて、ＰＦＡ装置は自己整列型形式で実施されないため、出力置換ステージが必要となる。

【先行技術文献】

【特許文献】

【００４８】

【特許文献１】米国特許第７，０２８，０６４号

【発明の概要】

【発明が解決しようとする課題】

【００４９】

いくつかのＤＦＴをシリアルに計算する目的でＤＦＴコアを実施するときの主な問題のうちの１つは、ＤＦＴ計算のレイテンシを制限することである。ＤＦＴ計算モジュールは、可能な限り高速であるべきである。目標は、データストリームのクロックにおけるフロー上の一連のＤＦＴ計算を、２つのＤＦＴ計算の間の遅延を低減して、又はさらに有利には一切遅延を発生させることなく達成することである。

【課題を解決するための手段】

【００５０】

本発明は、計算を加速するために、２つの計算コアを並列に使用してＤＦＴを計算することを提案する。データは、ＰＦＡルリタニアマッピング行列の偶数行及び奇数行に従って、２つのコアのいずれかに送られる。コアごとに別個の記憶手段が使用され、基数計算ステップと基数計算ステップとの間に２つの別個の記憶手段間でデータを交換する手段が提供される。

【００５１】

本発明は、種々のサイズのＤＦＴを計算する装置に関する。この装置は、フロー上の種々のサイズを有するすべてのバタフライを計算する２つのバタフライ計算ユニットと、ＤＦＴのサイズの半分に等しいサイズを有するデータを記憶する２つの別個の記憶手段と、ルリタニアマップの偶数行及び奇数行に従って、２つの別個の記憶手段内に入力データを送る手段と、ルリタニアマップが異なることに起因して、基数が変化するごとに、データを再分配する手段と、２つの別個の記憶手段から出力データを取り出す手段とを備える。

【００５２】

本発明の特定の実施の形態によれば、記憶手段のそれぞれは、基数サイズが変化するときに入力演算及び出力演算のために交互に使用される２つの異なるメモリをさらに備える。

【００５３】

本発明の特定の実施の形態によれば、上記装置は、トランスペアレントなアクセスのために２つの記憶手段をカプセル化する仮想化手段をさらに備える。

【００５４】

本発明の特定の実施の形態によれば、上記装置は、ルリタニアマップの変更に基づくアドレス変換手段をさらに備える。

【００５５】

本発明の特定の実施の形態によれば、データの再分配は、各バタフライユニットからの出力データに対して同様の再配列を適用すると共に、当該２つの再配列されたストリームにおいて対交換を行うことによって達成される。

【００５６】

また、本発明は、種々のサイズのＤＦＴを計算する方法にも関する。この方法は、２つのバタフライ計算ユニット上で、フロー上の種々のサイズを有するすべてのバタフライを計算するステップと、ＤＦＴのサイズの半分に等しいサイズを有するデータを、２つの別個の記憶手段内に記憶するステップと、ルリタニアマップの偶数行及び奇数行に従って、

２つの別個の記憶手段内に入力データを送るステップと、ルリタニアマップが異なることに起因して、基数が変化するごとに、データを再分配するステップと、２つの別個の記憶手段から出力データを取り出すステップとを含む。

【００５７】

本発明の特定の実施の形態によれば、データを記憶するステップは、基数サイズが変化するときに入力演算及び出力演算のために交互に使用される２つの異なるメモリ内の各記憶手段内に交互にデータを記憶するステップをさらに含む。

【００５８】

本発明の特定の実施の形態によれば、上記方法は、トランスペアレントなアクセスのために２つの記憶手段をカプセル化する仮想化ステップをさらに含む。

【００５９】

本発明の特定の実施の形態によれば、上記方法は、ルリタニアマップの変更に基づくアドレス変換ステップをさらに含む。

【００６０】

本発明の特定の実施の形態によれば、データの再分配は、各バタフライユニットからの出力データに対して同様の再配列を適用すると共に、当該２つの再配列されたストリームにおいて対交換を行うステップによって達成される。

【００６１】

本発明の特徴は、一例の実施形態の以下の説明を読むことによってより明らかになる。当該説明は、添付図面に関して作成されている。

【図面の簡単な説明】

【００６２】

【図１】本発明の一例示の実施形態における、ＤＦＴコアアーキテクチャを示す。

【図２Ａ】本発明の一例示の実施形態において使用される、行列形式での基数５のバタフライの計算の原理を示す。

【図２Ｂ】本発明の一例示の実施形態において使用される、行列形式での基数５のバタフライの計算の原理を示す。

【図３】本発明の一例示の実施形態における、マルチ基数バタフライモジュールのアーキテクチャを示す。

【図４】本発明の一例示の実施形態において使用される、加算・累算（ＭＡＣ）ユニットのアーキテクチャを示す。

【図５Ａ】本発明の一例示の実施形態における、ＭＡＣユニットの制御の一変形を示す。

【図５Ｂ】本発明の一例示の実施形態における、ＭＡＣユニットの制御の一変形を示す。

【図５Ｃ】本発明の一例示の実施形態における、ＭＡＣユニットの制御の一変形を示す。

【図６】本発明の一実施形態における、マルチ基数バタフライモジュールの実施態様の一変形を示す。

【図７】本発明の一例示の実施形態における、２つのＭＡＣユニットを連結したものを示す。

【図８】本発明の一例示の実施形態における、ＤＦＴエンジンモジュールのアーキテクチャの一例を示す。

【図９Ａ】本発明の一例示の実施形態における、アドレス生成器で使用される累算器の一例を示す。

【図９Ｂ】本発明の一例示の実施形態における、アドレス生成器で使用される累算器の一例を示す。

【図１０】本発明の一例示の実施形態における、データマネージャの第１のバージョンの状態マシンを示す。

【図１１Ａ】本発明の代替的な一実施形態による、ＤＦＴのシーケンスを最適化するためのデータマネージャモジュールの一変形を示す。

【図１１Ｂ】本発明の代替的な一実施形態による、ＤＦＴのシーケンスを最適化するためのデータマネージャモジュールの一変形を示す。

【図１１Ｃ】本発明の代替的な一実施形態による、ＤＦＴのシーケンスを最適化するためのデータマネージャモジュールの一変形を示す。
【図１２Ａ】本発明の代替的な一実施形態による、レイテンシを削減するためのデータマネージャモジュールの一変形を示す。
【図１２Ｂ】本発明の代替的な一実施形態による、レイテンシを削減するためのデータマネージャモジュールの一変形を示す。
【図１３】本発明の最後の実施形態における、データマネージャモジュールのロードステップを詳細に示す。
【図１４Ａ】本発明の最後の実施形態における、データマネージャモジュールの計算ステップを詳細に示す。
【図１４Ｂ】本発明の最後の実施形態における、データマネージャモジュールの計算ステップを詳細に示す。
【図１５】本発明の最後の実施形態における、データマネージャモジュールの計算ステップ中にデータを再編成するために必要な演算を示す。
【図１６】本発明の最後の実施形態における、データマネージャモジュールのダンプステップを詳細に示す。
【図１７】基数３のバタフライの計算を示す。
【発明を実施するための形態】
【００６３】
本発明においてＤＦＴコアとして説明される装置は、ディジタル集積回路上において、サイズ$N = 2^p \times 3^q \times 5^r$の複素数値のセットに対するＤＦＴの計算を必要とする、任意のシステムで使用することができる。ここで、ｐ、ｑ、ｒは、変換ごとに変化する可能性がある整数である。この装置はＧＰＦＡアルゴリズムのハードウェア実施を実現する。その文脈において、この装置を互いに素な数の積に因数分解することができる任意の合成サイズに容易に拡張することができる。ここで、これらの数のうちのいくつか又はすべては、所与の基数の累乗として表すことができる。ＤＦＴコアは、ユーザクロックとして示される同一のクロック周波数で機能するデータ元とデータ宛先との間で動作するものと仮定される。ＤＦＴコアの以降の説明は、図１に示される３つの主要独立部分から成る。これらの主要独立部分は、データマネージャモジュール１．１、ＤＦＴエンジンモジュール１．２、及びマルチ基数バタフライモジュール１．３である。ＤＦＴコアのこれら３つの部分は、協働するように設計されていることに加えて独立している。すなわち、それらを異なるフレームワークで実施することができる。データマネージャ１．１は、ユーザシステムとのインターフェースである。これは、データマネージャ１．１が、データ元からデータ値を受信し、結果値をデータ宛先へ送信することを意味する。ＤＦＴエンジンモジュール１．２は、基本ＤＦＴステージをシーケンスし、データマネージャモジュール１．１がマルチ基数バタフライモジュール１．３にデータ値を適切に供給して結果値を保存できるようにする。ＤＦＴエンジンモジュール１．２は、マルチ基数バタフライモジュール１．３の制御も行う。
【００６４】
単一のマルチ基数バタフライモジュールを使用して種々のサイズのすべてのバタフライを計算することは、少ない資源を使用して種々のサイズのＤＦＴを計算すると共にインプレースであることを可能にする柔軟な設計の問題に対処するための重要なポイントである。ＧＰＦＡのソフトウェア実施では、基数計算の異なるステップに異なる関数を使用するのに対して、本発明は、ＤＦＴを計算するいくつかの異なる基数モジュールに依拠せず、単一の再コンフィギュレーション可能なユニットに依拠する。
【００６５】
第１のステップにおいて、マルチ基数バタフライモジュール１．３は、ｒ次の複素行列と複素データ値の列ベクトルとの積を計算する。ここで、ｒは現在の基数である。第２のステップにおいて、マルチ基数バタフライモジュール１．３は、第１のステップの計算結果から得られる列ベクトルのすべての要素に対して回転を適用する。複素行列は、ｒ次の

ＤＦＴ行列に対応する。追加の回転は、ひねり因子の回転を適用するために使用される。サイズｒのＤＦＴの複素行列公式は、以下の式に示される。

【００６６】

【数１４】

$$B_r = (b_{k,l})_{\substack{0\le k<r \\ 0\le l<r}} = e^{\frac{2i\Pi kl}{r}}$$

【００６７】

ここで、ｉは複素数の虚数単位（ｉ^2＝－１）である。

【００６８】

実際には、マルチ基数バタフライモジュール１．３は、カルテシアン表現に従って実数値にのみ作用する。２つの複素数の乗算は、２次の行列と列ベクトルとの積として見ることができる。マルチ基数バタフライモジュール１．３は、２次の実行列とｒ個の列ベクトルとのｒ個の積を作成する前に、２ｒ次の実行列と１つの列ベクトルとの積を実行する。図２Ａは、ｒ＝５のバタフライ行列によって計算される実数積を示している。以下の式は、$b_{k,l}$及び$t_{k,l}$の値を示している。

【００６９】

【数１５】

$$B_r = (b_{k,l})_{\substack{0\le k<r \\ 0\le l<r}} = \begin{cases} \cos\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & k \text{ と } l \text{ が同じパリティの場合} \\ \sin\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & k \text{ が偶数で } l \text{ が奇数の場合} \\ -\sin\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & k \text{ が奇数で } l \text{ が偶数の場合} \end{cases}$$

$$T_r = (t_{k,l})_{\substack{0\le k<r \\ 0\le l<r}} = \begin{cases} \cos(i\alpha\lfloor k/2 \rfloor) & k = l \text{ の場合} \\ \sin(i\alpha\lfloor k/2 \rfloor) & k = l+1 \text{ の場合} \\ -\sin(i\alpha\lfloor k/2 \rfloor) & l = k+1 \text{ の場合} \\ 0 & \text{その他の場合} \end{cases}$$

【００７０】

第１の最適化は、行列Ｔ及びＢの特性を利用することによって、資源、すなわちエネルギーを消費する乗算の個数を削減することに本質がある。係数$b_{2k,0}$、$b_{2k+1,1}$、$b_{0,2l}$、及び$b_{1,2l+1}$は１に等しく、係数$b_{2k,1}$、$b_{2k+1,0}$、$b_{1,2l}$、及び$b_{0,2l+1}$はヌル（null）である。同様に、係数$t_{0,0}$値及び$t_{1,1}$値は１に等しく、係数$t_{1,0}$値及び$t_{0,1}$値はヌルである。これらの係数に乗算は必要とされない。図２Ｂは、簡単化した後のバタフライ行列によって計算される積を示している。

【００７１】

図１７は、基数３のバタフライの説明図である。この図は、３つのエントリーａ、ｂ、及びｃを示している。また、この図は、ひねり因子及び回転を求めるために適用される角度も示している。この図で分かるように、第１の値ａは角度０による影響を常に受ける。角度０は、この第１の値ａが各出力の計算において値１と常に乗算されることを意味する。また、第１の出力Ａの計算では、すべての値が角度０による影響を受けること、及び、０乗されている回転項W_N^0が常に１に等しく、これは第１の出力の計算に乗算が必要とされないことを意味することにも留意することができる。この注意点は、すべての基数について正しいことが証明される。

【００７２】

実施のための残る解決法は、基本計算ユニットによって行列の各行を独立且つ並列に取り扱うことである。通常、１つの基本ユニットは、複素数の各実部に１つずつの合計２つ

のＭＡＣユニットを備える。以下では、実行列について考える。上記で解説したように加算器のみを必要とする最初の２つの行を除いて、ＭＡＣユニットは、行列Ｂ又はＴの１つの行の各要素をデータ値の列ベクトルの各要素と乗算し、中間結果を累積して最終的な列ベクトルを得るために使用される。基数５をサポートしなければならないので、８個のＭＡＣユニットが実施される。ただし、より大きい基数が必要とされる場合には、より多くのユニットを実施することができる。さらに、この実施では、それよりも小さい基数のバタフライを計算するときには、ＭＡＣユニットを非アクティブ化することができる。したがって、エネルギーが節約される。図３は、マルチ基数バタフライモジュール１．３の第１の実施形態における加算器及びＭＡＣユニットのレイアウトを表している。ＭＡＣユニットの一例は図４に示されている。入力Ｘ及びＹは、乗算器４．６の前の２つのバッファ４．１及び４．２に接続されている。乗算器４．６の出力は、バッファ４．３に接続されている。バッファ４．３は、加算器４．７に接続されている。この加算器は、マルチプレクサ４．５の出力も取り込み、自身の出力をバッファ４．４に入れる。バッファ４．４は、マルチプレクサ４．５の一方の入力に戻って接続され、累算が実現される。累算器は、マルチプレクサの入力Ｃを使用して、Ｃにおいてヌル入力を選択することによってリセットすることができる。バッファ４．４の出力が、ＭＡＣユニットの出力となる。図５Ａは、中間結果Ｘ、Ｘ’、及びＸ”を提供するデータベクトルＤ、Ｄ’、及びＤ”に対する、ＭＡＣユニット３．２の内部における３つの連続した基数３のバタフライ処理のタイミングシーケンスを示している。行列Ｂの第４行の要素及び列ベクトルの要素は、１つずつ乗算されて累算器に加えられる。累算器は、新たな各バタフライ計算の前に事前にリセットされてヌル値にされている。p_i値は、d_i値と$b_{3,i}$値との積の結果である。また、s_i値は、$j \leqq i$であるp_j値の累算の結果である。中間結果x_3が、最後の値s_iである。

【００７３】

マルチ基数バタフライモジュール１．３の最適化バージョンは、ＭＡＣユニットの２つのステージ（バタフライ及び回転）を単一のステージに融合する。基数行列Ｂの１つの行の最初の２つの要素は、偶数行では常に（１，０）であり、奇数行では常に（０，１）である。そのため、偶数行に取り付けられたＭＡＣユニットには、列ベクトルの第１の値をプリロードすることができ、奇数行に取り付けられたＭＡＣユニットには、第２の値をプリロードすることができる。図５Ｂに示すように、各バタフライ計算間においてＭＡＣユニットの処理時間を解放することができる。図５Ｃに示すように、これらの時間は回転処理を挿入するために使用される。２つ一組の数値を再挿入するために、２つのステージ間のシリアル化が依然として必要であり、これは２つのＭＡＣユニットを連結することを必要とする。図５では、行列Ｂの第３行に取り付けられたＭＡＣユニットによって提供される中間結果x_2が必要であり、この中間結果x_2は中間結果x_3で共に挿入される。y_i値は、x_i値と$t_{3,i}$値との積の結果である。また、y_iは、$j \leqq i$であるx_jの累算の結果である。最終的な結果r_3は、y_i値の合計である。

【００７４】

ＭＡＣユニットの減算を行う能力に依存する第２の最適化は、係数値$b_{k,l}$及び$t_{k,l}$（異なるパリティのｋ及びｌ）が負の極性をもっているときに、加算を減算に置き換えることである。したがって、値の符号を反転する必要はもはやなくなり、演算のタイプを示すフラグのみが必要となる。この最適化は、基数バタフライに対してπの回転を行わなければならない場合である対向角を考慮するとき、ＤＦＴが逆ＤＦＴ（ＩＤＦＴ）としてコンフィギュレーションされるとき、又は、角度が区間［π；２π］に属するべきである（ｓｉｎ（－α）＝－ｓｉｎ（α））ときに非常に役立つ。

【００７５】

マルチ基数バタフライモジュール１．３のこの実施形態は、データ値の種々の表現に適合しており、固定小数点又は浮動小数点を使用することができる。固定小数点を使用する場合、ＭＡＣユニットの内部の余分なビットを、累算器を飽和させないようにするために使用することができる。この実施形態は、入力データ値をより多くのビットに拡張すること、及びデータを表すのに使用されるビットの数を制御するスケーリングモジュールが存

在することを必要とする。上位ビットが除去されるとき、スケーリングは飽和の制御によって有利に達成される。下位ビットを除去するとき、スケーリングは最も近い値に丸め込むことで有利に達成される。スケーリングを行うときの問題は、動的性と精度との間に最適なトレードオフを適用することである。有利には、このスケーリングモジュールは動的であり、計算されているＤＦＴのサイズに適合することができる。たとえば、これはマルチ基数バタフライの出力における結果をサイズ変更できる２つのスケーリングモジュールを使用して実現することができる。第１のスケーリングモジュールは、ＭＡＣユニットの内部の結果を再挿入して回転を適用するために、ＭＡＣユニットの出力をサイズ変更する。余分なビットが使用されるので飽和管理は必要とされない。第２のスケーリングモジュールは、誤った値を出力しないように飽和を管理しなければならず、好ましくは、出力値のダイナミクス範囲と精度との間を調整するためにサイズ変更ウィンドウを動的に制御する能力を有する。

【００７６】

最後に、マルチ基数バタフライモジュール１．３は、その最後の実施形態において、ひねり因子がヌルの場合に出力利得を調整する能力を伴う。この能力は、最後のステージ中に回転が必要とされないＤＦＴでは非常に役立つものとなる。したがって、ＤＦＴの全体的な利得を、計算のまさに最後で調整することができる。この新しい機能は、もはや行列Ｔを使用するのではなく対角行列Ｇを使用すると共に、加算器（たとえば３．１）の入力を使用されていないＭＡＣユニットへ逸らすことによって行われる。ここで、対角行列Ｇは係数$g_{k;l}$が所望の利得調整に等しい行列であり、加算器（たとえば３．１）の入力を使用されていないＭＡＣユニットへ逸らすことは、マルチ基数バタフライモジュール１．３のサイズが最大基数ｒに合わせられており、最後のステージがｒよりも小さい基数ｒ’を使用する場合に行われる。図６は、マルチ基数バタフライモジュール１．３の実施態様の一例をその最適化バージョンで提示している。入力データd_iは、複素データベクトルの実部及び虚部を交互にして挿入される。係数$b_{k,l}$、$t_{k,l}$、及び$g_{k,l}$も挿入される。マルチプレクサ６．３ａ、６．３ｂ、及び６．４は、乗算が必要とされるので、利得を調整するために、Ｂ行列の最初の２つの行の計算を加算器から最後のＭＡＣユニットへ逸らすのに使用される。この実施態様では、実部r_{2i}及び虚部r_{2i+1}は、加算器モジュール６．１及びＭＡＣユニット（たとえば６．２）の対の出力においてシーケンシャルに現れるが、同時に現れるようにすることもできる。加算器の対は、行列Ｂの最初の２つの行の実部及び虚部を個別に加算し、それら２つの結果をシリアル化してスケーリングし、ＭＡＣユニットの対のレイテンシと適合するようにそれら２つの結果を遅延させる。ＭＡＣユニットの対は残りの計算を行う。シリアラル化モジュール６．５は、加算器及びＭＡＣユニットからの結果をシリアル化し、結果r_iの複素数値の実部及び虚部を交互に出力する。ただし、別の実施態様では、１／２の低い周波数で複素数値の実部及び虚部を同時に出力することもできる。図７は、２つ一組のＭＡＣユニットの内部のスケーリングモジュールの実施態様の一例を示している。マルチプレクサ７．４、７．５、７．６、及び７．７は、バタフライ計算のフェーズ及び回転計算のフェーズを交互に行うために使用される。モジュール７．１は、マルチプレクサ７．２及び７．３を介してＭＡＣユニットの累算器に入力データを挿入するために、入力データを遅延させる。マルチプレクサ７．２及び７．３は、累算器をヌル値でリセットすることもできる。結果がＭＡＣユニットの出力に最初に現れる時、それらの結果はバッファリングされて、モジュール７．８内にスケーリングされる。回転計算用の回転のウィンドウが現れると、２つの値はシーケンシャルに出力され、ひねり因子又は利得係数のいずれかと共にＭＡＣ内部に再入力される。この実施態様では、結果がＭＡＣユニットの出力に２度目に現れるとき、それらの結果は、単一のスケーリングモジュール７．１１内にシーケンシャルに入力される前に、ブロッカ７．９及び７．１０の内部にバッファリングされる。資源を使用していたであろう第２のスケーリングモジュールは省かれる。スケーリングモジュール７．１１は、１／２の低い周波数で動作することもできる。

【００７７】

このマルチ基数バタフライを、実際の所望の計算に従う種々のアーキテクチャ設計に使用することができることに留意すべきである。このマルチ基数バタフライは、説明したＤＦＴ計算設計に特に適しているが、このＤＦＴ計算設計に限定されるものではない。

【００７８】

ＤＦＴエンジンモジュール１．２は、ＤＦＴコアモジュールの第２の主要要素であり、別の節で説明されるデータマネージャモジュール１．１とマルチ基数バタフライモジュール１．３との間に配置される。その役割は、他の２つの主要要素の設定、実行、及び調整を行うことである。このモジュールは、ＤＦＴをそのサイズに従って基本バタフライのシーケンスで表現する表現手段を備える。また、このモジュールは、ネストされたループのシーケンスの形でＤＦＴをスケジューリングすると共に、データマネージャにおける関連のある入力データアドレス及び出力データアドレスをバタフライごとに生成するスケジューリング手段も備える。さらに、このモジュールは、ＤＦＴのサイズに従ってネストされたループのシーケンスをカスタマイズするカスタマイズ手段も備える。

【００７９】

ＤＦＴコアに挿入されたＤＦＴエンジンモジュールの詳細な図は、図８に示されている。このＤＦＴエンジンモジュールは、６つの下位要素から構成される。基数選択モジュール８．１は、ＤＦＴステージをシーケンスする。グローバル設定モジュール８．２は、ＤＦＴエンジンの他のすべての下位要素を設定する。アドレス生成モジュール８．３は、下位要素の基数行列モジュール８．４及びひねり行列モジュール８．５を用いてデータマネージャモジュール１．１及びマルチ基数バタフライモジュール１．３を駆動することによって、ＤＦＴステージを構成する基本バタフライを実行する。これら２つの最後の下位要素は、インデックスをマルチ基数バタフライによって使用可能な係数に変換する。マルチ基数バタフライモジュール１．３の実施態様に応じて、スケーリングテーブルモジュール８．６はオプションとすることができる。固定小数点表現では、スケーリングテーブルモジュール８．６は、マルチ基数バタフライの内部の中間結果のスケーリングを設定する。本明細書で説明するＤＦＴエンジンでは、データマネージャモジュール１．１は、１つの読み出しポート及び１つの書き込みポートを有する単一のメモリであると見なすことができる。

【００８０】

ＤＦＴエンジンは、データマネージャモジュールが入力データのセット全体を収容すると始動する。データマネージャはバッファとして動作するので、ＤＦＴエンジンは自身の独立したクロックで動作する。基数選択モジュール８．１は、ＤＦＴのサイズ（入力としてセットアップすることもできるし、格納期間中にカウンタを使用して計算することもできる）を知ると、ＤＦＴステージへの分解に対応する基数２、３、５（及び、いくつかの実施形態では最終的に４）のシーケンスを準備し、第１の基数をグローバル設定モジュール８．２へ送信する。グローバル設定モジュール８．２は、他のすべてのモジュールを設定する。アドレス生成モジュール８．２は、一旦設定されると始動し、他のすべてのモジュールを自動的に駆動する。アドレス生成モジュール８．２は、アドレスを使用することによって、データマネージャモジュール１．１がマルチ基数バタフライにデータを適切な順序で供給すると共に中間結果を正しい位置に保存することを可能にする。基数行列モジュール８．４及びひねり行列モジュール８．５は、インデックスを使用してバタフライの計算に必要とされる係数を生成する。アドレス生成モジュール８．３がその最後のバタフライに達すると、基数選択モジュール８．４は、最後の基数となるまで（アドレス生成モジュール８．３を再び始動させて）次の基数を選択する。基数選択モジュール８．４が終了すると、データマネージャモジュール１．１は、ＤＦＴの最終結果を正しい順序で収容することになる。

【００８１】

ＧＰＦＡアルゴリズムは、サイズＮのＤＦＴを計算するとき、ルリタニアマップに従って、ＤＦＴ

【００８２】

【数１６】

$$W_{N_2}^{[N_3N_5]},\ W_{N_3}^{[N_2N_5]},\ W_{N_5}^{[N_2N_3]}$$

【００８３】

を連続して処理する。ここで、$N=N_2\times N_3\times N_5=2^p\times 3^q\times 5^r$である。再インデックス行列Ｒ及び$R^{-1}$は、各機能の内部に取り込まれている。このシーケンスは、第１のループを規定し、サイズ$N_2\times(N_3N_5)$、$N_3\times(N_2N_5)$、及び$N_5\times(N_2N_3)$の２次元ＤＦＴの計算をイニシエートする。第２のループで、サイズ$N_2\times(N_3N_5)$、$N_3\times(N_2N_5)$、及び$N_5\times(N_2N_3)$の２次元ＤＦＴは、ｐ個の基数２のステージ、ｑ個の基数３のステージ、及びｒ個の基数５のステージにそれぞれ分解される。各基数は、ＧＰＦＡアルゴリズムに従って、タイプ１又はタイプ２のいずれかとすることができる。

【００８４】

基数選択モジュール８．１は、これら２つのループを担当する。ここで説明する実施形態では、各基数のサイズ及びタイプに対応する複数のカウンタが、基数の個数を保持する。これらのカウンタは、基数ステージへのＤＦＴ分解によって初期化される（基数２　タイプ１、基数２　タイプ２、基数５　タイプ１、基数５　タイプ２、基数３　タイプ１、及び基数３　タイプ２）。代替的な一実施形態では、タイプ１及び２の基数４のステージが分解に入る。これらの分解は、計算することもできるし、表に記憶することもできる（取り扱われるＤＦＴの個数が、記憶されている分解の個数に制限される）。新しい基数ステージを開始すると、対応するカウンタがデクリメントされる。ＧＦＰＡアルゴリズムに従って基数のサイズ及びタイプをシーケンスするために、カウンタ間に優先順位が設定される。たとえば、基数２のステージに関連付けられるカウンタは、基数５のステージに関連付けられるカウンタよりも高い優先順位を有し、基数５のステージに関連付けられるカウンタは、基数３のステージに関連付けられるカウンタよりも高い優先順位を有する。代替的な一実施形態では、たとえば、基数４及び基数２のシーケンスは、回文を形成しなければならない。

【００８５】

最後に、基数選択モジュール８．１は、グローバル設定モジュール８．２を駆動する。基数選択モジュール８．１は、選択された基数をサイズ及びタイプと共に出力し、

【００８６】

【数１７】

$$W_{N_i}^{[N_o]}$$

【００８７】

の計算内部の進行度を出力する。この進行度は、同じ素因数のステージ内部のステージ番号（stage number）によって定義される。同じ素因数のステージの個数は、素因数の累乗に対応する。基数のサイズと素因数との間の区別は、基数２及び基数４をシーケンスするのに必要である。代替的な一実施形態では、基数４を選択すると、ステージ番号が２だけインクリメントされる。

【００８８】

グローバル設定モジュール８．２は、ＤＦＴエンジンモジュール１．２の他のサブモジュールへの有用なグローバルパラメータを生成するために使用される。

【００８９】

アドレス生成器モジュール８．３は、６つの瓦状のループ（imbricate loop）から成る。グローバル設定モジュール８．２は、ループの最大繰り返し数及びアドレス計算のインクリメント値を生成する。以下の表は、基数２、３、５の外側から内側へのループをリストしている。N_o及びN_iは、サイズ$N_o\times N_i=N$となるようになっている。ここで、ＮはＤＦＴのサイズであり、N_iは現在の基数の累乗である。これら２つの値は、計算することもできるし、記憶することもできる。値radix、stage、及びpowerは、基数選択モジュール８．１から得られたものである。

【００９０】
【表２】

説明	名称	タイプ１の基数	タイプ２の基数
分割数	NB_DIVISION	$radix^{(stage-1)}$	$radix^{2(stage-1)-power}$
次の分割へのジャンプ	JTN_DIVISION	$N_o \times radix^{power-(stage-1)}$	$N_o \times radix^{power-(stage-1)}$
サブ分割数	NB_SUBDIVISION	1	$Radix^{power-stage}$
次のサブ分割へのジャンプ	JTN_SUBDIVISION	-	$N_o \times radix^{stage}$
並列方式数	NB_TRANSVERSE	N_o	N_o
次の並列方式へのジャンプ	JTN_TRANSVERSE	N_i	N_i
インターリーブされたバタフライ数	NB_BUTTERFLY	$Radix^{power-stage}$	$Radix^{power-stage}$
次のインターリーブされたバタフライへのジャンプ	JTN_BUTTERFLY	N_o	N_o
基本バタフライ数	NB_SUBB	1	Radix
次の基本バタフライへのジャンプ	JTN_SUBB_IN JTN_SUBB_OUT	- -	$N_o \times radix^{(stage-1)}$ $N_o \times radix^{power-stage}$
入力／出力数	RADIX	Radix	Radix
次の入力／出力へのジャンプ	JTN_INPUT JTN_OUTPUT	$N_o \times radix^{power-stage}$ $N_o \times radix^{power-stage}$	$N_o \times radix^{power-stage}$ $N_o \times radix^{(stage-1)}$

【００９１】
すべてのひねり因子は単一の角度の倍数であるので、グローバル設定モジュール８．２は、ひねり因子を生成するための開始角も計算する。この開始角は、ＤＦＴサイズ及びステージ番号に依存する。開始角は２つの因子の積である。第１の因子はステージ番号及び基数サイズにのみ依存し、その値は２π／$prime\_factor^{stage-1}$（１≦stage≦power）である。ここで説明する実施形態では、角度を記憶するためにインデックス付きの表が使用される（ひねり行列モジュール８．５を参照）。この表は、２iπ／$prime\_factor^{max\_power}$（０≦i≦$prime\_factor^{max\_power}-1$）の値を含む。ここで、max_powerは、取り扱われるＤＦＴの中の累乗の最大値である。それ故、インデックス$prime_factor^{max_power}$は角度２πに対応する。演算時には、スケーリングが適用されて、インデックス$prime_factor^{power}$が角度２πに確実に対応するようにされる。したがって、第１の因子の値は、$prime\_factor^{stage-1+max\_power-power}$（１≦stage≦power）になる。第２の因子は、商$N_o = N／prime_factor^{power}$である。$prime_factor^{power}$は角度２πに対応するので、値（$N_o \bmod prime\_factor^{power}$）のみが使用される。これによって、角度を区間［０，２π］に維持することが可能になる。

【００９２】
ＧＰＦＡアルゴリズムでは、マルチ基数バタフライの説明で導入されたバタフライ行列Ｂに対して回転を適用しなければならない。回転を適用することによって、行列は顕著に変更される。回転バタフライ行列は、列の転置若しくは置換又はその双方によって基準の

行列から導き出すことができる。有利には、これらの転置及び置換は、参照表のアドレス指定を適合させることによって得られる。必要とされる演算は、値（（N_o modulo N_i）modulo Radix）に依存する。ここで、N_iは現在の基数の累乗であり、積$N_o \times N_i$はＤＦＴのサイズである。基数２について可能な値は１つあり、基数３について可能な値は２つあり、基数４について可能な値は２つあり、基数５について可能な値は４つある。各可能な値は異なる行列に対応する。以下の表は、可能な値と演算との間の関係を示している。グローバル設定モジュール８．２は、転置行列の場合にマルチ基数バタフライのサイン値を反転するためのフラグ、及び基数行列モジュール８．４用の列を置換するためのフラグを生成する。

【００９３】

【表３】

$((N_o \bmod N_i) \bmod Radix)$	0	1	2	3	4
基数２		オリジナル			
基数３		オリジナル	転置		
基数４		オリジナル		転置	
基数５		オリジナル	置換	置換＋転置	転置

【００９４】

基数行列モジュール８．４は、マルチ基数バタフライモジュール１．３に対して、マルチ基数バタフライの説明で導入されたバタフライ行列Ｂの係数を提供する。ここで説明する実施形態では、マルチ基数バタフライは、係数が列ごとに生成される必要がある。したがって、基数行列モジュールは入力として、現在の基数、出力する列番号（column number）、及び列を置換するためのフラグを取り込む。係数$b_{k,l}$は、計算することもできるし、表に記憶することもできる。表に記憶する場合、列全体が記憶され、インデックスは現在の基数及び要求される列インデックスによって形成される。マルチ基数バタフライは実数に作用するので、基数行列モジュール８．４は、複素係数の実部及び虚部を交互にして実数の列を形成しなければならない。基数行列モジュール８．４は、複素行列の列を生成するように要求されると、最初に列のすべての複素係数について符号を有するサイン値の対（コサイン，サイン）を生成し、次に反対の符号を有する対（サイン，コサイン）を生成する。この符号は、サイン値に対して加算又は減算のいずれを行わなければならないのかをマルチ基数バタフライモジュール１．３に伝える。

【００９５】

ひねり行列モジュール８．５は、マルチ基数バタフライモジュ１．３に対して、マルチ基数バタフライの説明で導入された行列Ｔの係数を提供する。ひねり行列モジュール８．５は入力として、現在の基数及びバタフライの第２の要素の回転の角度を取り込む。出力は、係数の列及び符号の列である。バタフライの第２の要素を回転するために求められたひねり因子αが与えられると、ひねり行列モジュール８．５は、バタフライのすべての要素を回転するのに必要なひねり因子０、α、２α、…のシーケンスを生成する。ここで説明する実施形態では、マルチ基数バタフライは列ごとに係数を必要とし、実数を扱う。ひねり行列モジュール８．５は、ひねり因子を生成するように要求されると、最初に列のすべての複素係数について符号を有するサイン値の対（コサイン，サイン）を生成し、次に反対の符号を有する対（サイン，コサイン）を生成する。この符号は、サイン値に対して加算又は減算のいずれを行わなければならないのかをマルチ基数バタフライモジュール１．３に伝える。

【００９６】

コサイン値及びサイン値は、計算することもできるし、表に記憶することもできる。表に記憶する場合、すべての値を記憶するためには、素因数ごとに単一の表で十分である。各表のサイズは、取り扱われるＤＦＴの中での素因数に付随する最大累乗に依存する。角

度２ｉπ／prime_factor^{max_power}（０≦ｉ≦prime_factor^{max_power}－１）のコサイン値及びサイン値を含む表の粒度は、２π／prime_factor^{max_power}になる。表のサイズを削減する第１の最適化は、サイン関数の特性を使用することである。角度ｉ及びprime_factor^{max_power}－ｉ（０≦ｉ≦prime_factor^{max_power}）はモジュロπで等しく、等しいコサイン値及び反転したサイン値を有する。πよりも小さい角度のみが必要であり、表のサイズは、素因数２、３、５についてそれぞれ

【００９７】

【数１８】

$$2^{max\_power(2)}/2,\ \lfloor 3^{max\_power(3)}/2 \rfloor,\ \lfloor 5^{max\_power(5)}/2 \rfloor$$

【００９８】

となる。要求される角度αが

【００９９】

【数１９】

$$\lfloor prime\_factor^{max\_power}/2 \rfloor$$

【０１００】

よりも大きい場合、新しく考えられる角度は

【０１０１】

【数２０】

$$\lfloor prime\_factor^{max\_power}/2 \rfloor - \alpha$$

【０１０２】

であり、フェッチされたサインの符号は反転される。

【０１０３】

アドレス生成モジュール８．３は、３つの主要部分に分解することができる。第１の部分はループの管理に専用化され、第２の部分はデータマネージャモジュール１．１のアドレスを生成するのに使用される累算器に専用化され、第３の部分はコマンドに専用化されている。

【０１０４】

ＧＰＦＡアルゴリズムでは、バタフライの入力及び出力を求めるために、ループが使用される。ここで説明する実施形態では、アドレス生成モジュール８．３は、６つのカスケードされたループを使用する。これらのループは、グローバル設定モジュール８．２によって初期設定される。第１のループTRANSVERSE（最も外側のループ）は、２次元ＤＦＴを多数の１次元ＤＦＴに分解する。第２のループDIVISIONは、クーリーテューキーアルゴリズムの分割統治手法に対応する分割の回数をカウントする。タイプ２の基数の場合、第３のループSUBDIVISIONが使用される。第４のループBUTTERFLYは、インターリーブされたバタフライをシーケンスする。タイプ２の基数の場合、第５のループSUB-BUTTERFLYが、タイプ１の基本バタフライをシーケンスする。最後に、第６のループI/O（最も内側のループ）は、バタフライの入力及び出力をシーケンスする。ここで説明する実施形態では、ループは繰り返し回数を記憶するカウンタを使用して実施される。図９にモジュール９Ａ．１によって表された最大繰り返し回数でループが設定されると、最も内側のループ９Ａ．２のカウンタがカウントを開始する。このカウンタは、最後の繰り返しに達するか又は終了すると、外側のループ９Ａ．３のカウンタをインクリメントすると共に自身をリセットする。９Ａ．４のような他のすべてのカウンタも、自身の内側のループ９Ａ．２及び９Ａ．３が最後の繰り返しに達するという条件の下で、自身の外側のループのカウンタに対して同じことを行う。アドレス生成モジュール８．５は、すべてのカウンタが最後の繰り返しに達したときに終了する。

【０１０５】

データマネージャモジュール１．１への宛先のアドレスは、各ループの繰り返し回数の

線形結合である。この線形結合の係数は、グローバル設定モジュール８．２によって最初に初期化されたインクリメント値又はＪＴＮ値である。ここで説明する実施形態では、複雑度を削減するために乗算は行われず、データアドレスの計算は各ループの中間累算器を使用する。ループのカウンタがリセットされるとき、そのループに取り付けられた累算器は、カウンタがリセットされていない最も外側のループに取り付けられた累算器で自身の値を更新する。ループのカウンタがインクリメントされるとき、ループに付随したＪＴＮ値が対応する累算器に加算される。図９Ｂは、最も内側のループI/0に取り付けられた累算器９Ｂ．７を示している。内側のカウンタ９Ａ．２～９Ａ．６の終了条件に応じて、マルチプレクサ９Ｂ．２～９Ｂ．６は、最も外側の９Ｂ．１２を優先して内側の累算器９Ｂ．８～９Ｂ．１２の値のうちの１つで累算器を再初期化するか、又はマルチプレクサ９Ｂ．１の値を使用する。このマルチプレクサは、ＪＴＮ値の加算の許可／不許可を行い、第１の内側ループの終了条件によって駆動される。最も内側のループの場合、このマルチプレクサ９Ｂ．１は、JTN_INPUT値を使用するためにバイパスされる。同じ方式は、マルチプレクサの個数が少ない内側の累算器にも使用される。

【０１０６】

モジュロＮの最も内側の累算器９Ｂ．１３は、データマネージャモジュール１．１への宛先の読み出しアドレスである。書き込みアドレスは、２つの方法で生成することができる。第１の方法は、２つの最も内側のループに２つの追加の累算器を使用する。これら２つの累算器は、読み出しアドレスと同じであるが、ＪＴＮ値の順序が並べ替えられている読み出しアドレスである。第２の方法は、行及び列の個数が基数のサイズに等しい２次元アレイを使用する。生成された読み出しアドレスは、行ごとにアレイに書き込まれる。一旦満杯になると、アレイを列ごとに読み出すことによって、書き込まれたアドレスが得られる。カウンタCNT_IOは、列番号を待っている基数行列モジュール８．４を直接駆動するために使用することができる。計算は基数計算で開始するので、このアドレス信号に対する遅延は必要とされない。ひねり因子生成に適用される角度は、グローバル設定モジュール８．２によって生成された基準角とCNT_BUTTERFLYとの積として与えられる。その後、結果の角度は０と２πとの間に適合するように補正され、ひねり行列モジュール８．５へ送信される。各コマンドは、マルチ基数バタフライへの宛先のデータ及び係数が一致して生じるように送信されなければならない。

【０１０７】

この第１の実施形態では、サイズ$N = N_2 \times N_3 \times N_5 = 2^p \times 3^q \times 5^r$のすべてのＤＦＴが利用可能である。基数４は、基数２のステージの個数及び計算時間を同時に削減するために使用することができる。それでもなお、いくつかのＤＦＴは、ジョンソン及びビュルスによって示されたように、それらのＤＦＴが回文として分解されることによって計算が不可能になる。$N = N_{128} \cdot N_3$がＤＦＴのサイズであると仮定する。N_{128}＝（４×２×４×４）は、正しく分解することができない。N_{128}＝（４×８×４）を作成するには、基数８が必要になる。

【０１０８】

マルチ基数バタフライモジュール１．３のアーキテクチャを考えると、基数８のバタフライの計算は、１６個のＭＡＣユニットを必要とする。ＤＦＴ複雑度を低く維持するために、基数８のバタフライは、基数２のバタフライの第１のステージ及び基数４のバタフライの第２のステージをシーケンスすることによって行うことができる。したがって、追加のＭＡＣユニットは必要ない。ジェントルマン－サンデアルゴリズムの第１の部分を使用すると、基数８の行列を積：

【０１０９】

【数２１】

$$W_8 = \left(P_4^2\left(I_2 \times W_4\right)\right)\left(D_4^2\left(W_2 \times I_4\right)\right)$$

【０１１０】

に容易に分解することができる。

【０１１１】
この基数８は、ＤＦＴのサイズに応じて、１、３、５、又は７だけ引き上げる必要がある。ＤＦＴの回転に使用される補助定理によって、
【０１１２】
【数２２】

$$W_8^{[r'=\mathrm{mod}(r,8)]} = \left(P_4^2\left(I_2 \times W_4^{[r''=\mathrm{mod}(r',4)]}\right)\right)\left(D_4^{2[r']}\left(W_2 \times I_4\right)\right)$$

【０１１３】
が与えられる。ここにはひねり因子の適用がないので、この基数８モジュールは依然として不完全である。
【０１１４】
【数２３】

$$M_8^{[kr]} = \begin{pmatrix} 1 & 0 \\ 0 & \omega_8^{kr} \end{pmatrix}, \qquad T_4^{[kr]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega_4^{kr} & 0 & 0 \\ 0 & 0 & \omega_4^{2kr} & 0 \\ 0 & 0 & 0 & \omega_4^{3kr} \end{pmatrix}$$

【０１１５】
であると仮定する。ひねり因子を有する完全な基数８のモジュールは
【０１１６】
【数２４】

$$D_8^{1[kr]} W_8^{[r']} = \left(T_4^{[kr]} \times I_2\right)\left(I_4 \times M_8^{[kr]}\right)\left(P_4^2\left(I_2 \times W_4^{[r'']}\right)\right)\left(D_4^{2[r']}\left(W_2 \times I_4\right)\right)$$

$$D_8^{1[kr]} W_8^{[r']} = \left(T_4^{[kr]} \times I_2\right) P_4^2 \left(M_8^{[kr]} \times I_4\right)\left(I_2 \times W_4^{[r'']}\right)\left(D_4^{2[r']}\left(W_2 \times I_4\right)\right)$$

$$D_8^{1[kr]} W_8^{[r']} = \left(P_4^2\left(I_2 \times T_4^{[kr]}\right)\right)\left(I_2 \times W_4^{[r'']}\right)\left(\left(M_8^{[kr]} \times I_4\right) D_4^{2[r']}\right)\left(W_2 \times I_4\right)$$

【０１１７】
となる。最後に、次数Ｎ＝ｐｑの対角行列Ｄ'$^{p}_{q,t}$を
【０１１８】
【数２５】

$$D'^{p}_{q,t}(j,k) = \omega_N^{st} \qquad j = k = sq + m \text{ の場合}$$

$$= 0 \qquad \text{それ以外の場合}$$

【０１１９】
によって定義する。前の式は
【０１２０】
【数２６】

$$D_8^{1[kr]} W_8^{[r']} = P_4^2\left(I_2 \times \left(T_4^{[kr]} W_4^{[r'']}\right)\right) D'^{2}_{4,kr}\, D_4^{2[r']}\left(W_2 \times I_4\right)$$

【０１２１】
となる。
【０１２２】
この式の主な関心は、基数２のバタフライの結果及び基数４のバタフライの結果の双方の第１の要素に適用されるひねり因子が常に１であるということである。その結果、基数４の乗算のいくつかが基数２の計算内で行われる。したがって、第１の要素の計算は乗算器を必要とせず、加算器のみを必要とする。これによって、２つ一組の加算器をマルチ基数バタフライモジュール１．３の内部に維持することが可能になる。最後に、第１のステージは、特定のひねり因子を有する通常の基数２のステージであり、第２のステージは、

インターリーブされた出力を有する通常の基数４のステージである。主要な変更は必要とされない。

【０１２３】

回文を形成するために、基数選択モジュール８．１は、最初にタイプ１の基数４のステージ、オプションのタイプ１の基数２又は基数８のステージ、及び最後にタイプ２の基数４のステージをシーケンスするように更新される。基数８のステージは、基数の２のステージ及び基数４のステージで構成され、それらのステージを標準のステージと区別するフラグが作成される。さらに、累乗値が素因数2^pに付随するので、基数４についてステージカウンタを２だけ増加しなければならない。グローバル設定モジュール８．２においては、基数が４である場合、アドレス生成モジュール８．３のループの設定は、すべての値stage-1をstage-2に置き換えることによって得られる。基数８のバタフライの２つの部分は、タイプ２の基数バタフライと非常に類似しているので、ループ設定は容易に実施される。以下の表は、基数８に付随したループの設定を要約したものである。

【０１２４】

【表４】

説明	名称	基数８の第１の部分	基数８の第２の部分
分割数	NB_DIVISION	$2^{(stage-1)}$	$2^{(stage-3)}$
次の分割へのジャンプ	JTN_DIVISION	$N_o \times 2^{power-(stage-1)}$	$N_o \times 2^{power-(stage-2)}$
サブ分割数	NB_SUBDIVISION	1	1
次のサブ分割へのジャンプ	JTN_SUBDIVISION	-	-
並列方式数	NB_TRANSVERSE	N_o	N_o
次の並列方式へのジャンプ	JTN_TRANSVERSE	N_i	N_i
インターリーブされたバタフライ数	NB_BUTTERFLY	$2^{power-stage-2}$	$2^{power-stage}$
次のインターリーブされたバタフライへのジャンプ	JTN_BUTTERFLY	N_o	N_o
基本バタフライ数	NB_SUBB	4	2
次の基本バタフライへのジャンプ	JTN_SUBB_IN JTN_SUBB_OUT	$N_o \times 2^{(stage-1)}$ $N_o \times 2^{(stage-1)}$	$N_o \times 2^{(stage-1)}$ $N_o \times 2^{power-stage}$
入力／出力数	RADIX	2	4
次の入力／出力へのジャンプ	JTN_INPUT JTN_OUTPUT	$N_o \times 2^{power-stage}$ $N_o \times 2^{power-stage}$	$N_o \times 2^{power-stage}$ $N_o \times 2^{(stage-2)}$

【０１２５】

基数行列モジュール８．４では、係数を記憶している場合には、基数４の行列の係数がこの表に挿入される。ひねり行列モジュール８．５では、生成されたひねり因子に角度オフセットを加算することが、基数８の内部の基数２のバタフライの計算に課せられる。このオフセットは、基数８の内部の基数２のランクに依存し、その値は

【０１２６】

【数２７】

$$e^{i\frac{\pi}{4}rank\cdot(N_o \bmod 8)}$$

【０１２７】

である。

【０１２８】

データマネージャモジュール１．１は、データのロード及びダンプを含めて、データにアクセスするためのすべての演算をサポートする。また、データマネージャモジュール１．１は、ロード、ＤＦＴ計算、及びダンプをチェックする状態マシンも伴っている。図１０に示すこの状態マシンは、モジュールの機能を示している。図１１ａに示すそのまさに基本の機能では、データマネージャモジュール１．１は、サイズN_{max}（N_{max}は取り扱われるＤＦＴの最大サイズであると仮定する）の１つのダブルポートメモリ（１つのポートは読み出し用であり、もう１つのポートは書き込み用）と、データ及びアドレスをルーティングするマルチプレクサとから成る。ロード演算は、入力サンプルをメモリの書き込みポートにリンクし、たとえばカウンタによってメモリを線形にアドレス指定することによって行われる。計算中、データはメモリの読み出しポートから取り出され、結果は書き込みポートを通して書き込まれる。この時、ＤＦＴエンジンからのアドレスが使用される。マルチ基数バタフライモジュール１．３のレイテンシは、特にバタフライの入力及び出力が異なるタイプ２の基数の期間中、読み出し／書き込みのオーバーラップに対する保護の働きをする。完了すると、最終結果は読み出しポートから線形にダンプされる。ロード及びダンプは線形演算であるので、新しいロードはダンプの開始直後に開始することができる。図１１ｂは、この処理のタイミングシーケンスを示している。

【０１２９】

レイテンシを削減し、さらには、一定フローの入力サンプルの計算を可能にするために、いくつかの独立した最適化を適用することができる。

【０１３０】

第１の最適化は、ユーザクロックから独立したＤＦＴコア駆動クロックを増加させることに本質がある。この第１の最適化は、レイテンシを大幅に削減することができる。

【０１３１】

第２の最適化は、ロード、計算、及びダンプの演算をパイプライン化するものであり、図１１Ａに示されている。このパイプラインは、第１のメモリと同一の第２のメモリを使用することによって作成される。一方のメモリのサンプルに対して計算を行っている間、他方のメモリはロード及びダンプに利用可能になる。２つの異なるメモリは、入力／出力演算のためのメモリ及びワーキングメモリとして交互に使用される。新たなＤＦＴのロードが現在のＤＦＴの計算よりも多くの時間を要するという条件下では、この最適化によってレイテンシが削減され、連続したＤＦＴをシーケンスすることが可能になる。先に述べた最適化と組み合わせることによって、より幅広いパネルのＤＦＴをシーケンスすることが可能になる。図１１Ｂ及び図１１Ｃは、第１の最適化を伴わない処理及び第１の最適化を伴った処理を示している。

【０１３２】

第３の最適化は、ＤＦＴ内部クロックを増加させることなく計算時間を削減するものである。これは、クロック周波数がターゲット装置の限界に達しているときに非常に役立つ。この解決法は、並列性を活用するものであり、実際のマルチ基数バタフライユニットから独立している。この解決法は、任意のバタフライ計算ユニットに使用することができる。また、この解決法は、たとえばマルチコアプロセッサ上でソフトウェアにおいて実施することもできる。計算中、サンプルの前半がマルチ基数バタフライモジュール１．３の内部で扱われる。一方、サンプルの後半はインスタンス化された新しいマルチ基数バタフライモジュール内で扱われる。この切り取りは、ルリタニアマップの偶数番号の行及び基数番号の行に従って行われる。ＤＦＴエンジンは変更されないが、この場合、現在の基数に付随したルリタニアマップの偶数行上にある（figuring）点に関連するアドレスのみを出

力する。偶数行をアドレス指定するとき、次の行（奇数）のアドレスが導き出され、双方の行からの点を２つの独立した計算ユニットの内部で同時に扱うことができる。この切り取りの利点は、行に対して行われる厳密な同一の演算のために、行を容易に並列計算することができるということである。いくつかの解決法が存在するが、すべての解決法は、予測された基数に付随したルリタニアマップの偶数行及び奇数行上にある点が２つの別々のメモリ内に常に置かれるように点を分散させなければならない。これは、ルリタニアマップが異なるので、基数が変化するごとに、点を再分散することを暗に意味する。したがって、４つの分散が予測される。第１の分散はロードの場合であり、第２の分散は或る基数から同じ基数に変化する場合であり、第３の分散は或る基数から別の基数に変化する場合であり、最後の分散はダンプの場合である。

【０１３３】

以下で詳細に説明する解決法は、サイズN_{max}／２の２つのメモリへのデータの分散及び記憶の問題を解決する。ここで、N_{max}は、取り扱われるＤＦＴの最大サイズである。基数を変更すると読み出しアドレス及び書き込みアドレスが異なるものになるので、さらに２つの同一のメモリが、読み出し／書き込みのオーバーラップを防止するために使用される。さらに、特定の基数シーケンス（４，３，５）が、実施の可能な最適化及び容易化のために使用される。図１２Ａ及び図１２Ｂは、全体的な解決法をその処理の時間シーケンスと共に示している。既存のアーキテクチャを最大限維持するために、抽象レイヤが４つのメモリをカプセル化する。これにより、ＤＦＴエンジンは、あたかもサイズN_{max}の単一のメモリが存在するかのようにメモリにアクセスする。このレイヤによって、データが存在すべきアドレスに対応する仮想アドレス及びデータが実際に記憶されるアドレスに対応する物理アドレスが導入される。このレイヤは、ルリタニアマップの圧縮に基づく自己構築型アドレス変換表と、データを再編成するためのロジックとから成る。

【０１３４】

圧縮によって、素因数に付随した２次元ルリタニアマップが、２行の新しいルリタニアマップに変換される。第１行は、オリジナルのルリタニアマップの偶数行からのすべての点を含む。第２行は、奇数行からのすべての点を含む。使用されるアルゴリズムは、次のとおりである。

ＮがＤＦＴのサイズであると仮定する。点は０からＮ－１まで１つずつ新しいアレイに挿入される。点は、挿入されるときに、列０から行に挿入される。点は、ルリタニアマップの偶数行に属する場合には第１行に挿入され、そうでない場合には第２行に挿入される。完了すると、循環回転が第２行に適用されて点N_iが点０の下に配置される。奇数の行数を含むルリタニアマップに対しては例外が設けられ、この例外は、N_o個の点が挿入されるごとに第２行にパディング点（padding point）を追加することに本質がある。しかしながら、基数３及び基数５に付随した圧縮されたルリタニアマップは、常に、それらの第１行には偶数点を含み、それらの第２行には奇数点を含む。

【０１３５】

この圧縮されたルリタニアマップは、２つのメモリの内部のデータの位置を表す。ＮがＤＦＴサイズであると仮定すると、抽象レイヤで使用されるアドレス変換表は、圧縮されたルリタニアマップの内部のデータ０からデータＮ－１の座標である行及び列を含む。常に同じである基数３及び基数５に付随した変換表を除いて、異なるルリタニアマップと同数の変換表が存在する（前のパラグラフ参照）。このことが、ＤＦＴを処理する前に基数４の変換表を構築しなければならない理由である。ＮがＤＦＴサイズであり、R_rがサイズN_i行×N_o列のルリタニアマップであり、CT_Tが変換表であると仮定する。ここで、N_iはｒの累乗であり、N_o＝Ｎ／N_iである。完全な変換表を再生成するために、ルリタニアマップにおいて１番目の点が配置される行R_1、及び圧縮されたルリタニアマップにおいてN_i番目の点が配置される最終列FCN_iのような事前に計算された値が必要である。累算器Ｒｘは、ルリタニアマップの内部でｘ番目の点が配置される行を含む。カウンタJumpが、パディング要素を加えるのに使用される。

【０１３６】

【数２８】

```
Point = 0;
Rx = 0;
Index0 = 0;
Index1 = FCNi;
Jump = 0;
While Point < N do
        If (Rx is even) then
                CT(Point) = (0,Index0);
                Index0 = Index0 + 1;
        Else
                CT(Point) = (1,Index1);
                Index1 = Index1 + 1;
        End if;
        Jump = Jump + 1;
        If (Jump = No) then
                Jump = 0;
                Index1 = Index1 + 1;
        End if;
        Rx = (Rx + R1) mod Ni;
        Point = Point +1;
End while;
```

【０１３７】

データロード中、データは、図１３に示すように、その後の基数４のバタフライ用にデータに容易にアクセスすることができるように、２つのメモリの内部に分散される。取り扱われるＤＦＴサイズの分解は、常に基数４のバタフライを暗に意味するので、基数４のバタフライが最初に計算され、データロードの最適化が可能になる（同じ引数が、常に基数３の後に来るダンプに使用される）。ロード及びアドレス変換表構築の双方は直線的（linearly）に行われるので、基数４アドレス変換表構築の開始直後にロードを開始することができる。

【０１３８】

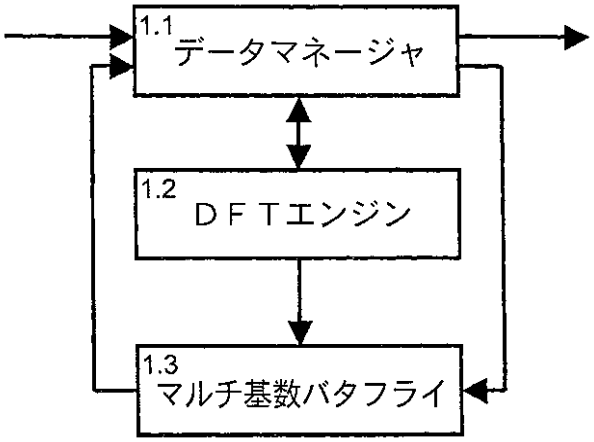
バタフライの計算中、ＤＦＴエンジンからの読み出しアドレスを変換し、その結果を４つのメモリの読み出しアドレスポートに適用することによって、入力データが２つのメモリからフェッチされる。図１４Ａに示すように、マルチプレクサ１４．１及び１４．２が、データバス出力において使用され、データを含むメモリが選択される。マルチ基数バタフライモジュールの結果を、ＤＦＴエンジンからの書き込みアドレスで記憶するには、次の基数が現在の基数と異なる（基数２は基数４とみなされる）ときは読み出し／書き込みのオーバーラップを防止し、基数がシーケンスの最後のものであるときはロードデータ及びダンプデータが決して同じメモリ内にないように、メモリ対を変更することがまず必要になる。マルチプレクサ１４．３によって、これを行うことが可能になる。その場合、より多くの演算が場合に応じて適用される。第１の演算はデータに作用する。双方のマルチ基数バタフライからのデータは、図１４．４に従って８つの要素から成るセットによって独立に再編成される。シーケンス（ｄ（０），ｄ（１），ｄ（２），ｄ（３），ｄ（４），ｄ（５），ｄ（６），ｄ（７））は、（ｄ（０），ｄ（４），ｄ（１），ｄ（５），ｄ（２），ｄ（６），ｄ（３），ｄ（７））となる。これは、双方の信号を一致して保持す

るためにアドレスに適用されるレイテンシを暗に意味する。この演算は、現在の基数がタイプ２であるとき又は基数が４であるときに行わなければならず、基数８の計算に属する。第１のメモリに達するデータ又は第２のメモリに達するデータを交互にすることが必要である。第２の演算もデータに作用する。この時、双方のマルチ基数バタフライからのデータは、２つの要素から成るセットによってそれらのデータ間でスクランブルされる。シーケンス（ｄ（０），ｄ（１））が第１のバタフライから来たものであり、シーケンス（ｅ（０），ｅ（１））がそれと同時に第２のバタフライから来たものであると仮定すると、このスクランブルによって、図１４Ｂ（要素１４．５）によって示されるように、４つ１組がスクランブルされて、シーケンス（ｄ（０），ｅ（０））が第１のメモリへ、シーケンス（ｄ（１），ｅ（１））が第２のメモリへ、同時に出力される。これも、アドレスに渡されるレイテンシを暗に意味する。この演算は、図１４．６に示すＤＦＴエンジンからの２つの書き込みアドレスから成るセットに作用する第３の演算を伴っている。シーケンス（ａ（０），ａ（１））は、（ａ（０），ａ（０）＋N_i）となる。ここで、N_iは、サイズＮのＤＦＴの分解における現在の基数の累乗である。これら最後の２つの演算は、素因数４から別の素因数に変化するときに必要であり、２つのメモリの内部でデータを再分散する。最後の演算は変換されたアドレスに作用するが、第２のメモリに付随したアドレスにのみ作用する。N_o（＝Ｎ／N_i）番目の点が次の素因数に付随した圧縮されたルリタニアマップの内部に配置される列番号に対応するオフセットが、加算器１４．７によってアドレスに加算される。これは、０番目の点と同時に来る点がN_o番目の点であるので必要であり、素因数５がＮの分解に属するときに適用されなければならない。最後に、図１５に示される表は、これらの演算を適用しなければならない場合を要約している。

【０１３９】

結果のダンプの前に、基数シーケンス（４，５，３）のために、また３の累乗に付随した変換表の特定の形のために、偶数番目の点は第１のメモリの内部に線形に記憶され、奇数番目の点はN_3番目の点からではあるが第２のメモリの内部に線形に記憶される。１番目の点の位置は、第２のメモリのアドレスを補正するために知られていなければならない。次に、点は、第１のメモリ又は第２のメモリのいずれかから交互に出力される。図１６は、データダンプのブロック図を表している。加算器１６．１は、第２のメモリのアドレスを補正するために使用される。マルチプレクサモジュール１６．２は、双方のメモリの出力を交互にするために使用される。

【図１】



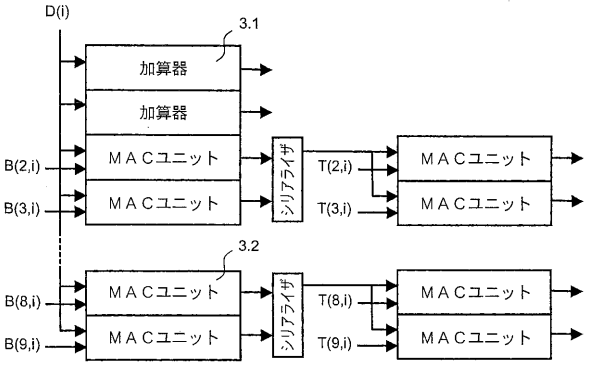
【図２Ａ】

$$\begin{pmatrix} t_{0,1} & t_{0,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t_{1,0} & t_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{2,2} & t_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{3,2} & t_{3,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{4,4} & t_{4,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{5,4} & t_{5,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{6,6} & t_{6,7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{7,6} & t_{7,7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{8,8} & t_{8,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{9,8} & t_{9,9} \end{pmatrix} \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} & b_{0,6} & b_{0,7} & b_{0,8} & b_{0,9} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} & b_{1,8} & b_{1,9} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} & b_{2,8} & b_{2,9} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} & b_{3,8} & b_{3,9} \\ b_{4,0} & b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} & b_{4,8} & b_{4,9} \\ b_{5,0} & b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} & b_{5,8} & b_{5,9} \\ b_{6,0} & b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} & b_{6,8} & b_{6,9} \\ b_{7,0} & b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} & b_{7,8} & b_{7,9} \\ b_{8,0} & b_{8,1} & b_{8,2} & b_{8,3} & b_{8,4} & b_{8,5} & b_{8,6} & b_{8,7} & b_{8,8} & b_{8,9} \\ b_{9,0} & b_{9,1} & b_{9,2} & b_{9,3} & b_{9,4} & b_{9,5} & b_{9,6} & b_{9,7} & b_{9,8} & b_{9,9} \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \end{pmatrix}$$

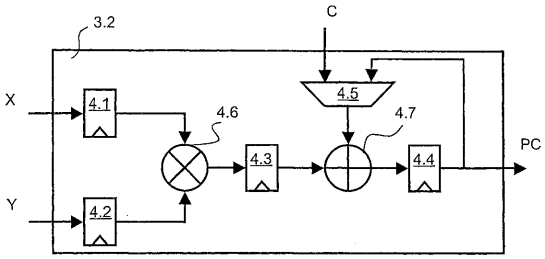
【図２Ｂ】

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{2,2} & -t_{3,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{3,2} & t_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{4,4} & -t_{5,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{5,4} & t_{4,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{6,6} & -t_{7,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{7,6} & t_{6,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{8,8} & -t_{9,8} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{9,8} & t_{8,8} \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & b_{2,2} & -b_{3,2} & b_{2,4} & -b_{3,4} & b_{2,6} & -b_{3,6} & b_{2,8} & -b_{3,8} \\ 0 & 1 & b_{3,2} & b_{2,2} & b_{3,4} & b_{2,4} & b_{3,6} & b_{2,6} & b_{3,8} & b_{2,8} \\ 1 & 0 & b_{4,2} & -b_{5,2} & b_{4,4} & -b_{5,4} & b_{4,6} & -b_{5,6} & b_{4,8} & -b_{5,8} \\ 0 & 1 & b_{5,2} & b_{4,2} & b_{5,4} & b_{4,4} & b_{5,6} & b_{4,6} & b_{5,8} & b_{4,8} \\ 1 & 0 & b_{6,2} & -b_{7,2} & b_{6,4} & -b_{7,4} & b_{6,6} & -b_{7,6} & b_{6,8} & -b_{7,8} \\ 0 & 1 & b_{7,2} & b_{6,2} & b_{7,4} & b_{6,4} & b_{7,6} & b_{6,6} & b_{7,8} & b_{6,8} \\ 1 & 0 & b_{8,2} & -b_{9,2} & b_{8,4} & -b_{9,4} & b_{8,6} & -b_{9,6} & b_{8,8} & -b_{9,8} \\ 0 & 1 & b_{9,2} & b_{8,2} & b_{9,4} & b_{8,4} & b_{9,6} & b_{8,6} & b_{9,8} & b_{8,8} \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \end{pmatrix}$$

【図３】



【図４】



【図５Ａ】

ステージ	クロック	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C			0						0						0					
2	4.1		d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5	
	4.2		0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3			p_0	p_1	p_2	p_3	p_4	p_5	p'_0	p'_1	p'_2	p'_3	p'_4	p'_5	p''_0	p''_1	p''_2	p''_3	p''_4	p''_5
	4.7			0	s_0	s_1	s_2	s_3	s_4	0	s'_0	s'_1	s'_2	s'_3	s'_4	0	s''_0	s''_1	s''_2	s''_3	s''_4
4	4.4				s_0	s_1	s_2	s_3	s_4	x_3	s'_0	s'_1	s'_2	s'_3	s'_4	x'_3	s''_0	s''_1	s''_2	s''_3	s''_4

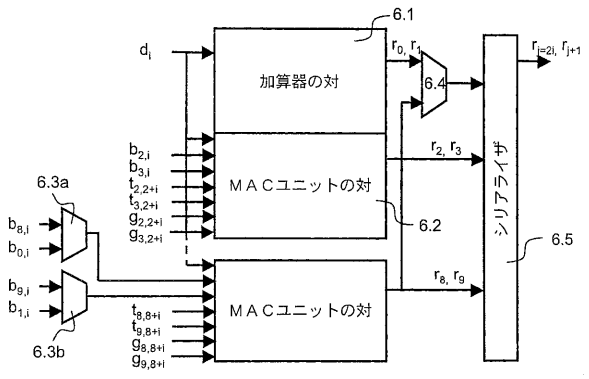
【図５Ｂ】

ステージ	クロック	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C					d_1						d'_1						d''_1			
2	4.1				d_2	d_3	d_4	d_5			d'_2	d'_3	d'_4	d'_5			d''_2	d''_3	d''_4	d''_5	
	4.2				$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3					p_2	p_3	p_4	p_5			p'_2	p'_3	p'_4	p'_5			p''_2	p''_3	p''_4	p''_5
	4.7					d_1	s_2	s_3	s_4			d'_1	s'_2	s'_3	s'_4	x'_3		d''_1	s''_2	s''_3	s''_4
4	4.4						s_2	s_3	s_4	x_3			s'_2	s'_3	s'_4	x'_3		s''_2	s''_3	s''_4	

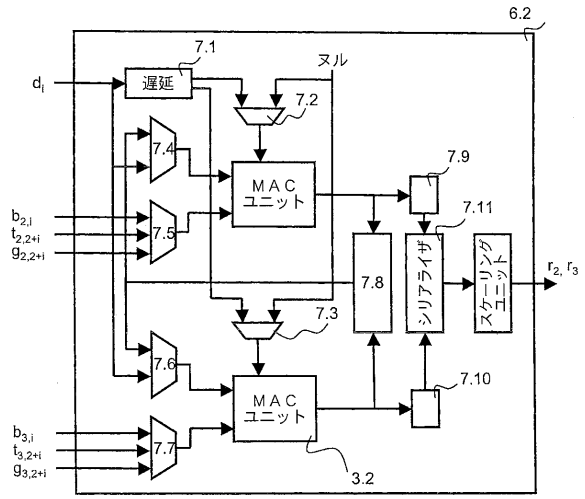
【図５Ｃ】

ステージ	クロック	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C					d_1						d'_1				0		d''_1			
2	4.1				d_2	d_3	d_4	d_5			d'_2	d'_3	d'_4	d'_5	x_2	x_3	d''_2	d''_3	d''_4	d''_5	
	4.2				$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	$t_{3,2}$	$t_{2,2}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3					p_2	p_3	p_4	p_5			p'_2	p'_3	p'_4	p'_5	y_0	y_1	p''_2	p''_3	p''_4	p''_5
	4.7					d_1	s_2	s_3	s_4			d'_1	s'_2	s'_3	s'_4	0	y_0	d''_1	s''_2	s''_3	s''_4
4	4.4						s_2	s_3	s_4	x_3			s'_2	s'_3	s'_4	x'_3	y_0	r_3	s''_2	s''_3	s''_4

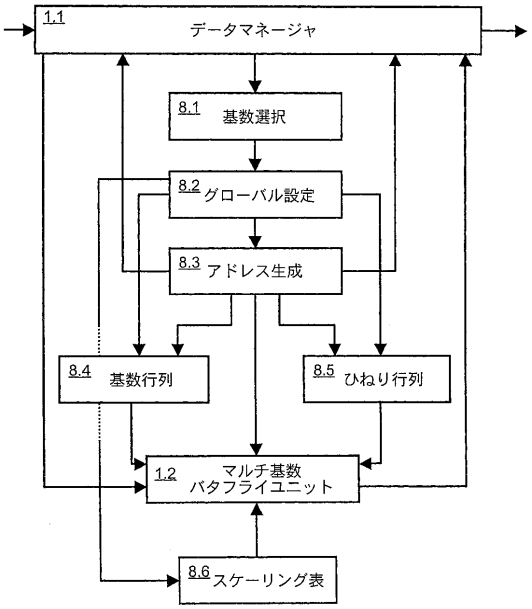
【図６】



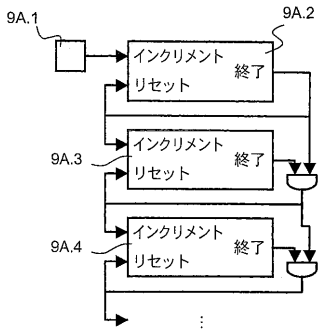
【図７】



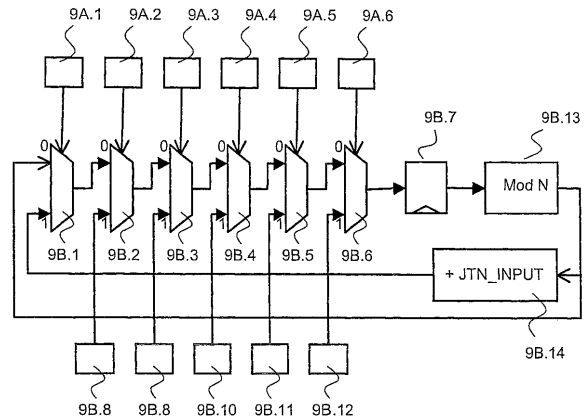
【図８】



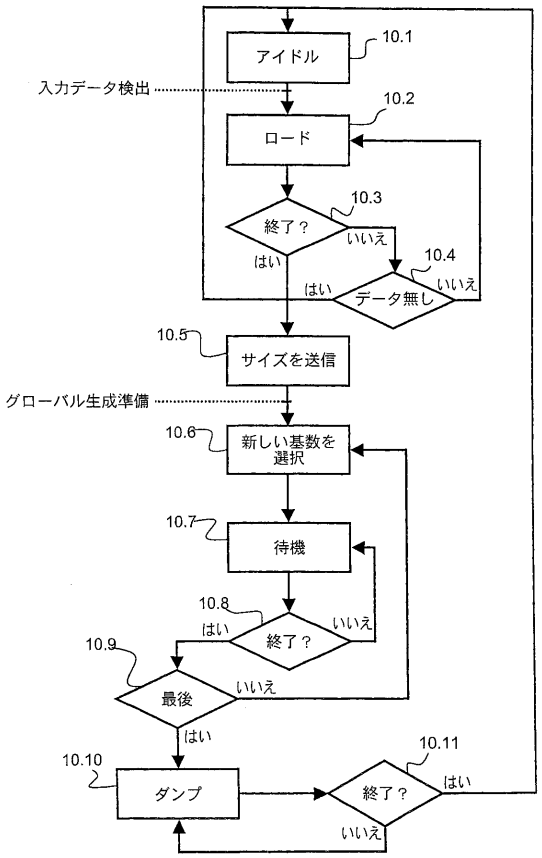
【図９Ａ】



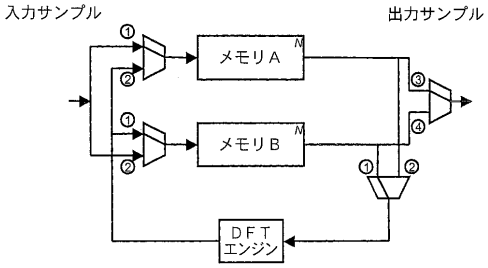
【図９Ｂ】



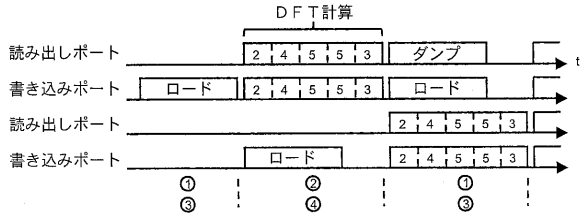
【図１０】



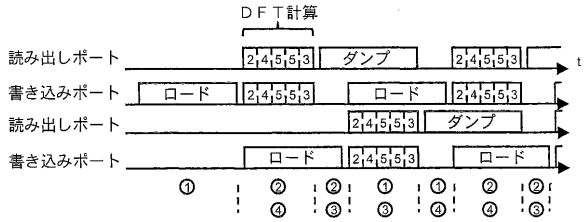
【図１１Ａ】



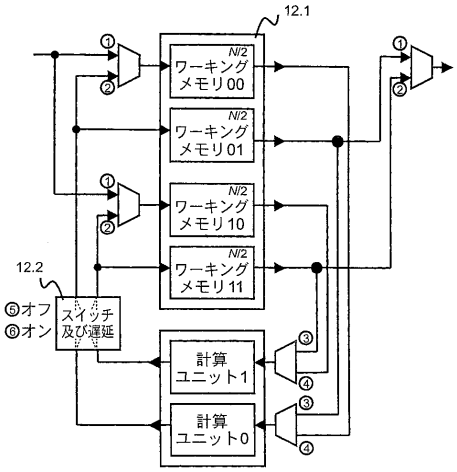
【図１１Ｂ】



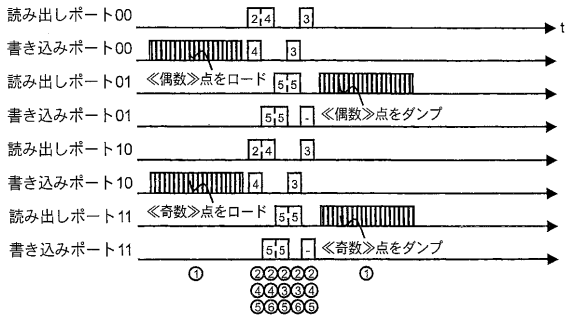
【図１１Ｃ】



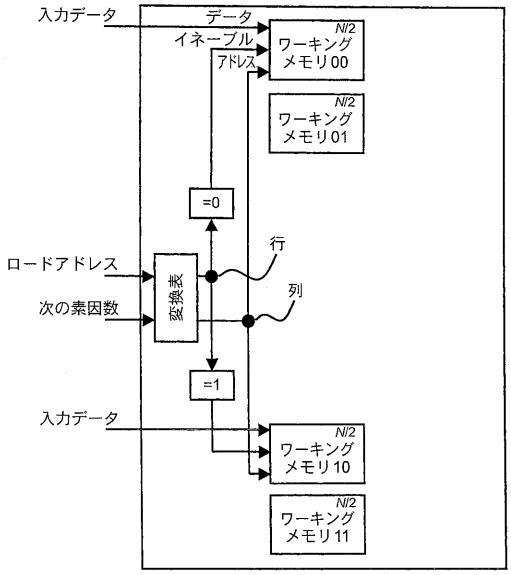
【図１２Ａ】



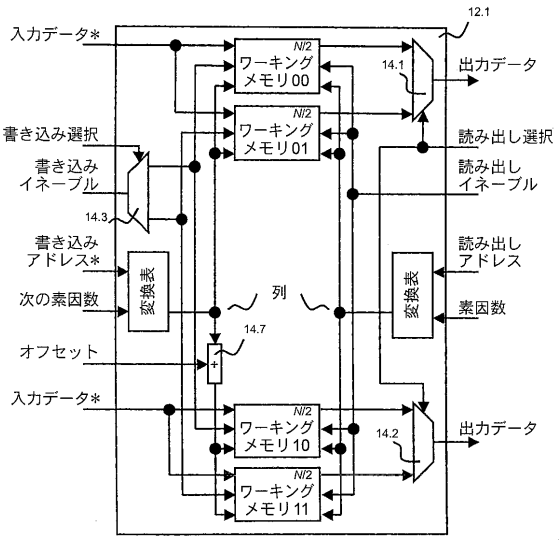
【図１２Ｂ】



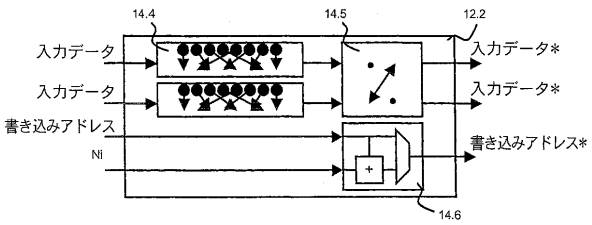
【図１３】



【図１４Ａ】



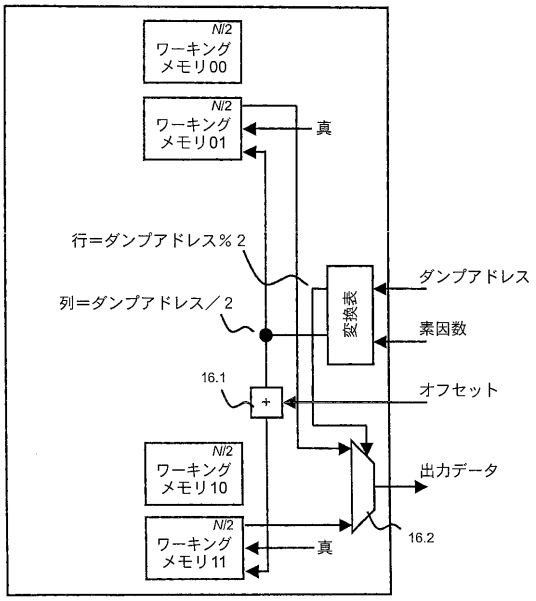
【図１４Ｂ】



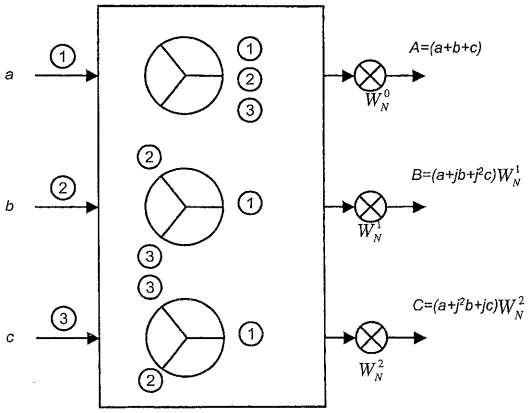
【図１５】

基数＼次の基数		4/2	5	3
4/2	タイプ１	演算なし	４つ１組をクロス／アドレスを変更／オフセットを加算	４つ１組をクロス／アドレスを変更
	タイプ２又は基数８	演算なし	データを再配列／４つ１組をクロス／アドレスを変更／オフセットを加算	データを再配列／４つ１組をクロス／アドレスを変更
5			演算なし	オフセットを加算
3				演算なし

【図１６】



【図１７】



フロントページの続き

(74)代理人 100094695
弁理士 鈴木 憲七
(74)代理人 100111648
弁理士 梶並 順
(74)代理人 100147500
弁理士 田口 雅啓
(72)発明者 グザヴィエ・ヌリッソン
フランス国、３５７０８ レンヌ・セデックス 7、セーエス １０８０６、アレ・ドゥ・ボーリュー 1、ケアオヴ・ミツビシ・エレクトリック・アールアンドディー・センター・ヨーロッパ
(72)発明者 アルノー・ブティエ
フランス国、３５７０８ レンヌ・セデックス 7、セーエス １０８０６、アレ・ドゥ・ボーリュー 1、ケアオヴ・ミツビシ・エレクトリック・アールアンドディー・センター・ヨーロッパ

Ｆターム(参考) 5K022 DD01 DD13 DD19 DD23

【外国語明細書】

Title of the invention

Device and Method to compute different sizes of DFT
according to the PFA algorithm with the Ruritanian mapping.

The invention generally relates to discrete Fourier transforms (DFT). More specifically, the invention relates to the hardware implementation on digital circuits of the DFT-spread orthogonal frequency division multiplexing (DFT-spread OFDM) modulation technique. DFT-spread OFDM is specified for uplink transmissions (from the terminal to the base station) in the Long-Term Evolution (LTE) standard issued by the third generation partnership program (3GPP) for beyond third generation (B3G) cellular networks. In DFT-spread OFDM, modulation symbols are spread by a DFT prior classical OFDM modulation to benefit from a lower peak-to-average ratio (PAPR). The purpose of this spreading also called pre-coding transform according to LTE naming conventions is to reduce the power consumption of the terminals. In the 3GPP-LTE standard, DFT-spread OFDM is combined with the frequency division multiple access (FDMA) technique to form the so-called single carrier frequency division multiple access (SC-FDMA) technique. In SC-FDMA, each user is allocated over a time transmission interval (TTI) with a given number of contiguous sub-bands made of 12 consecutive sub-carriers, from 1 to 110 sub-bands in a 20 MHz bandwidth. To reduce the hardware complexity of the DFT, only a subset of DFT sizes are

specified, more specifically, 43 different sizes N multiple of 12 (from 12 to 1944) where $N=2^p\times3^r\times5^s$, p, q and s are integers.

The DFT is one specific form of the Fourier transforms named after the introduction of the Fourier series by Joseph Fourier. A definition could be found in "*Digital Signal Processing, principles, algorithms, and applications*" by John G. Proakis and Dimitris G. Manolakis, New-York: Macmillan Publishing Company, 1992, ch. 9. The DFT applies to functions with finite duration (N values) and the result is computed by evaluating the Fourier transform at a set of N equally spaced discrete frequencies. The sequence of N complex numbers $\{x_k, 0 \le k \le N-1\}$ is transformed into the sequence of N complex numbers $\{X_n, 0 \le n \le N-1\}$ by the DFT according to the formula:

$$X_n = \sum_{k=0}^{N-1} x_k e^{-2i\pi\frac{p}{N}}, \quad 0 \le n \le N-1$$

where e is the base of the natural logarithm and i the imaginary unit of complex numbers ($i^2 = -1$). Many of the properties of the DFT depend on the fact that the coefficients $w_N^p = e^{-2i\pi\frac{p}{N}}$ usually called *twiddle factors* are primitive roots of unity. In its basic form, the computation of a N-point DFT algorithm requires $O(N^2)$ complex multiplications. In hardware, DFT are commonly computed using dedicated transforms usually called fast Fourier transforms (FFT) with a lower operations count. The great majority of FFT algorithms can be divided into four classes: the methods derived from the Cooley-Tukey algorithm, the methods derived from the Prime Factor Algorithm (PFA), the polynomial factorization methods and the convolution methods. These methods are not mutually exclusive, i.e. one of these methods can be used within another one to reduce the overall complexity.

The Cooley-Tukey algorithm is the reference algorithm for computing DFT transforms. A description could be found in "An algorithm for the machine calculation of complex Fourier series" by James W. Cooley and John W. Tukey, *Math. Comput.*, vol. 19, 1965, pp. 297-301. In its most popular form, the Cooley-Tukey algorithm consists in recursively dividing a DFT of size N into two transforms of size $N/2$ (divide and conquer approach). A simple approach to derive the Cooley-Tukey algorithm is to split the computation for the even- and odd-numbered frequency samples. As the same principle is applied at each step, the resulting FFT algorithm is called a *decimation-in-frequency* transform (DIF, also called the Sande-Tukey algorithm). This form which is intrinsically limited to power-of-2 sizes ($N=2^n$) is

known as the radix-2 DIF Cooley-Tukey FFT algorithm. A decimation-in-time (DIT) version can also be obtained applying a recursive transform on the time samples. The DFT can also be expressed as a matrix-vector multiplication as follows:

$$Z = W_N Y;$$

$$Y = [x_0\, x_1 \ldots x_{N-1}]^T, Z = [X_0\, X_1 \ldots X_{N-1}]^T;$$

$$W_N = \left[w_N^{ij}\right]_{i=0,\ldots,N-1, j=0,\ldots,N-1};$$

where i denotes the row index and j the column index. The matrix W_N is called the DFT matrix of order N. FFT algorithms are then defined by factorizing W_N into the product of a sequence of sparse matrices. The basic radix-2 DIF Cooley-Tukey FFT algorithm can be obtained through matrix factorization using the Gentleman-Sande algorithm. It is common to represent the Cooley-Tukey algorithm as a graph representing the $\log_2(N)$ stages in the computation of the DFT where each stage is made of $N/2$ basic 2-point DFT called butterflies. Each DIF butterfly reduces to an addition and a subtraction with the latter being rotated through a complex multiplication with a twiddle factor. Thus, for power-of-2 sizes N, the complexity of the Cooley-Tukey DFT reduces to $(N/2)\times\log_2(N)$ complex multiplications and $N\times\log_2(N)$ complex additions, much lower than the $O(N^2)$ complexity of the basic DFT.

One advantage of the Cooley-Tukey algorithm is that the computation of the core algorithm, the sequence of butterflies, can be performed *in-place*, i.e. without additional storage. Indeed, when the output of a butterfly has been computed, there is no need to save the input pair. The result of the butterfly can therefore be stored in the same location as the two inputs. In computer science, an in-place algorithm is an algorithm which transforms a data structure using a small, constant amount of extra storage space. According to this definition, the Cooley-Tukey can be described as being in-place. This advantage is counterbalanced by the fact that the Cooley-Tukey is not *self-sorting*, i.e. the output samples are not produced in a linear order. A function manipulating vectors is said self-sorting if it produces linearly ordered outputs from linearly ordered inputs. The Cooley-Tukey is not self-sorting due to the decimation resulting from the recursive computation of the odd- and even-numbered samples of the sub-DFT. It is therefore required to perform a de-scrambling operation to achieve the transform at the expense of an increased complexity and/or latency. In the case of radix-2 FFT, this operation takes the form of a *bit reversal* of the output samples addresses. Bit reversal is the permutation where the data at an index n, written in

binary with digits $b_2b_1b_0$ (e.g. 3 digits for N=8 inputs), is transferred to the index with reversed digits $b_0b_1b_2$. Actually, the Cooley-Tukey algorithm does not restrict to power-of-2 sizes. It can be extended to any size of the form $N=I^n$, where I is an integer. Typical examples are the radix-3 and radix-4 Cooley-Tukey FFT applicable respectively on power-of-3 and power-of-4 sizes. In these cases, the unscrambling step can also be achieved by means of a reverse operation but applied on the addresses expressed in base 3 and base 4. In fact, the Cooley-Tukey algorithm can be applied to any factorization of the size N into the product of any integers $N=n_1\times n_2\times n_3\times \ldots\times n_L$, leading to the so-called *mixed-radix FFT* algorithms. In this case, the unscrambling stage takes a more complicated digit reversal form, limiting their actual implementation.

In 1984, Johnson and Burrus showed that the radix-2 DIT Cooley-Tukey algorithm can be rewritten to obtain a *self-sorting* and *in-place* FFT algorithm in "An in-place in-order radix-2 FFT," *Proc. IEEE ICASSP*, 1984, pp. 28A.2.1-4. Clive Temperton in "Self-sorting in place fast Fourier transforms," *SIAM J. Sci. Stat. Comput.*, vol. 12, July 1991, pp. 808-823, showed that the principle used to make the Cooley-Tukey FFT both in-place and self-sorting can be generalized to other bases, e.g. to radix-3 and radix-5 algorithms. Graphically, the transform is made of a successive number of stages made of type 2 butterflies grouping a number of type 1 butterflies (the traditional ones) with interleaved outputs. It can be further extended to the mixed-radix FFT as long as the factorization of N forms a *palindrome*. Thus, the algorithm will work for $N=144=3^2\times4^2$ if the factors are used in the order 3, 4, 4, 3 or 4, 3, 3, 4. For many transform lengths, it is not possible to arrange the factors in the symmetrical manner required for the self-sorting in-place Cooley-Tukey algorithm. An alternative to these sizes is to put the factors that cannot be written as a palindrome in the middle of the factorization. For example, for $N=6000=3\times4^2\times5^3$, the factors can be arranged as follows $N=6000=4\times5\times15\times5\times4$.

In addition to the Cooley-Tukey algorithm, the invention described in this document uses the PFA algorithm that was originally introduced by Good in "The interaction algorithm and practical Fourier analysis," *J. Roy. Statist. Soc. Ser. B.*, vol. 20, 1958, pp. 361-372. An in-depth description of prime factor FFT algorithms has been given by Burrus in "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 25, 1977, pp. 239-242, and in "An in-place, in-order prime factor FFT algorithm," *IEEE Trans.*

Acoust. Speech Signal Process., vol. 29, 1981, pp. 806-817. The PFA can be viewed as a variation of the Cooley-Tukey algorithms for composite sizes made of mutually prime factors. The Cooley-Tukey algorithm consists in recursively dividing a DFT of size $N_1 \times N_2$ into many smaller DFT of sizes N_1 and N_2. These operations can be formalized through an appropriate re-indexing of the input and output samples. This is achieved by re-writing the indices k and n as:

$$k = N_1 k_2 + k_1 \text{ where } 0 \le k_1 \le N_1 - 1 \text{ and } 0 \le k_2 \le N_2 - 1,$$

$$n = N_2 n_1 + n_2 \text{ where } 0 \le n_1 \le N_1 - 1 \text{ and } 0 \le n_2 \le N_2 - 1.$$

The first operation can be interpreted as re-indexing the input (k) as a two-dimensional array made of N_1 rows and N_2 columns where the original set of data is linearly stored in the columns of the array. The second operation re-indexes the output (n) as a two-dimensional array made of N_1 rows and N_2 columns where the expected set of data is linearly stored in the rows. When this re-indexing is substituted into the DFT formula for nk the remaining term gives:

$$X_{N_2 n_1 + n_2} = \sum_{k_1=0}^{N_1-1} \left[e^{-\frac{2j\pi}{N} k_1 n_2} \right] \left[\left(\sum_{k_2=0}^{N_2-1} x_{N_1 k_2 + k_1} e^{-\frac{2j\pi}{N_2} k_2 n_2} \right) e^{-\frac{2j\pi}{N_1} k_1 n_1} \right]$$

It appears that the Cooley-Tukey algorithm recursively re-expresses a DFT of a composite size $N = N_1 N_2$ as:

- Re-index the input as a $N_1 \times N_2$ array sequentially filling the columns.
- Perform N_1 DFTs of size N_2 on the N_2 columns of the array.
- Multiply by the twiddle factors.
- Perform N_2 DFTs of size N_1 on the N_1 rows of the array.
- Re-index the output sequentially reading the rows of the array.

The PFA relies on a specific mapping between the integers (k) and (n) and the corresponding integer pairs (k_1, k_2) and (n_1, n_2) of the bi-dimensional interpretation of the DFT algorithm. As a matter of fact, at least two such mapping can be used, either the one based on the Chinese Remainder Theorem map (CRT) or the Ruritanian map (also called Good's map) as described in "The relationship between two Fast Fourier Transforms" by I. J. Good, *IEEE Trans. Comp.*, vol. 20, 1971, pp. 310-317. A different mapping can be used for the input and the output. The PFA transform is introduced here in the case of the Ruritanian map for both the input and the output which is defined by:

$n_1 = (pn) \bmod N_1,\ \ n_2 = (qn) \bmod N_2,$

$k_1 = (pk) \bmod N_1,\ \ k_2 = (qk) \bmod N_2,$

where the integers p and q are defined according to the CRT theorem which stands that if N_1 or N_2 are mutually prime, then it is possible to find integers p, q, r, s such that:

$pN_2 = rN_1 + 1$, where $0 < p < N_1, 0 < r < N_2$,

$qN_1 = sN_2 + 1$, where $0 < q < N_2, 0 < s < N_1$.

The inverse map is given by:

$n = (N_2 n_1 + N_1 n_2) \bmod N,$

$k = (N_2 k_1 + N_1 k_2) \bmod N.$

An example for N=40 (N_1=8 and N_2=5) is shown in the table below.

	0	1	2	3	4
0	0	8	16	24	32
1	5	13	21	29	37
2	10	18	26	34	2
3	15	23	31	39	7
4	20	28	36	4	12
5	25	33	1	9	17
6	30	38	6	14	22
7	35	3	11	19	27

It appears that the mapping can be easily built without having to find a solution to the Ruritanian equation. The entries in the first column increase from 0 in steps of N/N_1 (=N_2) while those in the first row increase from 0 in steps of (N/N_2= N_1). The remaining columns (or rows) can then be filled in by using the same increment as in the first column (or row) and taking the results modulo N. Applying this mapping into the DFT original formula gives:

$$X(n_1, n_2) = \sum_{k_2=0}^{N_2-1} \left(\sum_{k_1=0}^{N_1-1} x(k_2, k_2) e^{-\frac{2j\pi}{N_1} N_2 k_1 n_1} \right) e^{-\frac{2j\pi}{N_2} N_1 k_2 n_2}$$

Apart from the appearance of N_1 or N_2 multiplying the exponents, the equation takes the form of a bi-dimensional DFT of dimension $N_1 \times N_2$. Just as for the Cooley-Tukey algorithm, the DFT can be computed by performing N_2 DFT of length N_1 in one dimension followed by N_1 DFT of length N_2 in the other. The transform can also be computed starting on the other dimension. However, it must be emphasized that there

are no twiddle factors between the two stages leading to a lower operation count than for the Cooley-Tukey FFT. The multiplication by N_1 and N_2 of the exponent simply modify the way the rotations are applied onto the input samples. The frequency sample with rank n is computed as the linear combination of the time sample weighed by the roots of unity that are applied circulating around the unit circle with a step n starting from 0. The application of the factor N_a (a = 1 or 2) simply modifies the step from n to $n \times N_a$. Due to the Fourier transform property, the output samples can also be obtained by simply applying a circular shift on the output samples of the standard DFT: applying a rotation r to a transform of length N_a means that instead of appearing in the original order 0, 1, ..., N_a-1, the same results appear in the order 0, r, $2r$, ..., (N_a -1), where the indices are to be interpreted modulo N_a. By using the same mapping for both the input and the output, the PFA can be made both in-place and self-sorting. Indeed, as the mapping is the same, the output samples appear in the same order as the input samples. If the re-indexing operation form the one-dimensional array to the Ruritanian bi-dimensional representation is done through implicit address translation, the output samples appear in linear order in the working array, i.e. the algorithm is self-sorting. Similarly to the Cooley-Tukey algorithm, the PFA method can be applied recursively to the decomposition of the size N into the product of mutually prime factors $N=n_1 \times n_2 \times n_3 \times \ldots \times n_L$. It is more appropriate in this case to express the PFA algorithm in matrix form. Applying recursively the principle of the PFA algorithm conducts to the following factorization of the matrix W_N:

$$W_N = R^{-1}\left(W_{n_L}^{[r_L]} \times \ldots \times W_{n_2}^{[r_2]} \times W_{n_1}^{[r_1]}\right)R, \text{ where } r_i = \frac{N}{n_i} \bmod n_i \quad 1 \le i \le L$$

where R is the permutation matrix which maps the integer n to the corresponding L-dimensional space (n_1, n_2, ..., n_L) according to the Ruritanian map and where $W_N^{[r]}$ is equal to the matrix W_N with all elements raised to the power r. As mentioned above, it is not necessary in practice to actually compute the matrix product with the permutation matrices. The required mapping can be implemented implicitly via the indexing logic.

One issue of the PFA is that the decomposition of the size into mutually prime factors may contain large factors that are not compatible with a direct computation as a simple radix transform. It is therefore required to implement these transforms using one of the other DFT transforms. In the general case of composite sizes, the smaller DFT can be computed using the Cooley-Tukey algorithm. Thanks to the work of Johnson-Burrus and Temperton in "Implementation of self-sorting in-place prime

factor FFT algorithm," *J. Comput. Phys.*, vol. 58, 1985, pp. 283-299, it is possible to compute the Cooley-Tukey transform in a self-sorting and in-place manner. The application of this extension to the PFA algorithm for the computation of the composite factors allows defining a self-sorting and in-place DFT algorithm with a lower computational complexity than the strict application of the generalized Johnson-Burrus (self-sorting in-place Cooley Tukey) transform. This algorithm has been described for sizes of the form $N=2^p\times3^q\times5^r$, p, q and r integers by Temperton in "A generalized prime factor FFT algorithm for any $2^p3^q5^r$," *SIAM J. Sci. Stat. Comput.*, vol. 13, May 1992, pp. 676-686, as the Generalized PFA (GPFA) algorithm. This algorithm can be extended to any decomposition of the size as the product of powers of mutually prime integers. This was made possible by modifying the Cooley-Tukey algorithm to handle the rotation factors of the PFA algorithm. Temperton showed that the radix-p DFT with matrix $W^{[r]}_{N=p^m}$ can be computed like the DFT with matrix $W_{N=p^m}$ by applying the following modifications:

- Apply the rotation r (modulo p) to each radix-p module.
- Raise all the twiddle factors to the power r.

For example, $W^{[r]}_{N=p^m}$ $N=pm$ can be implemented by rotating each radix-5 module by $r'=4$ (9 modulo 5), and by raising all the twiddle factors to the power 9. This transformation is identified in the sequel by appending "rotated" to the name of the transform it is applied to.

The generalized self-sorting in-place prime factor FFT algorithm for any $N=2^p\times3^q\times5^r$ is constructed as follows:

- Re-index the input samples as a three-dimensional array of size $\left(2^p,3^q,5^r\right)$ using the Ruritanian map.
- Compute the one-dimensional transforms of length 2^p, 3^q, 5^r using the generalized rotated Johnson-Burrus scheme. The smaller DFT are thus implemented in an in-place self-sorting manner.
- Re-index the output samples as a one-dimensional transform of size $2^p,3^q,5^r$ using the Ruritanian map.

For values of N suitable for the PFA, this algorithm reduces to the PFA and has the same operation count. For $N=2^p$, 3^q or 5^r, the algorithm reduces to the self-sorting in-place derivation of the Cooley-Tukey algorithm and has the same operation count.

If N contains a mixture of factors but is unsuitable for the PFA, the algorithm has a lower operation count than the basic Cooley-Tukey algorithm.

The invention presented in this document deals with the hardware implementation of the GPFA algorithm on digital integrated circuits. Regarding the wide range of applications of the DFT, many solutions are available for the implementation of the FFT algorithms. A great majority of these solutions focus on the Cooley-Tukey algorithm and its derivations for an implementation on processors. However, for the situations where the speed is crucial, FFT algorithms are often implemented on integrated circuits. Although the Cooley-Tukey factorization of the DFT applies in some manner to all the Cooley-Tukey FFT devices, there exists a great diversity of solutions for implementing these algorithms. Integrated circuits intrinsically allow the parallelization of functions. It is therefore possible to implement all the butterflies of a given size FFT similarly to its graphical representation. This allows computing the result in a reduced number of clock cycles, but at the expense of a high complexity. The tradeoff between complexity and latency is actually what makes the difference between the different implementations of the FFT. In addition, depending on the specificities of the hardware device, solutions also differ in terms of memory usage and ratio between the number of multipliers and adders for implementing the butterflies.

Due to its popularity and rather low complexity, the radix-2 Cooley-Tukey FFT is by far one of the most widely used technique for implementing DFT in hardware. In many situations, the size of the DFT is not crucial, thus allowing the use of power-of-2 FFT with zero-padding at the expense however of an extra complexity and/or latency. In other situations such as for OFDM transmissions, the specifications of the system are often defined taking into account this constraint. What makes the radix-2 Cooley-Tukey FFT algorithm compatible with a hardware implementation is its modularity as illustrated in its graphical representation. It is actually made of $\log_2(N)$ successive stages of $N/2$ similar butterflies. As introduced above, the most straightforward solution is to implement all the butterflies just as in the graphical representation. This allows computing the DFT with a low latency and to pipeline successive DFT if data are presented in parallel which is not very common in practice. The obvious drawback of this solution is a high complexity both in terms of computing resources (multipliers and adders) and memory as the output of each step needs to be stored.

Two different approaches not mutually exclusive can be used to reduce the overall complexity: the first one is to re-use the same hardware resources to compute one stage at a time (solutions sometimes called *column FFT*). The results are fed back to the same set of process elements to compute the next stage. This allows saving memory but at the expense of a longer latency and a more complex routing architecture. The other approach is to reduce within each stage the number of processing elements ultimately to one by pipelining the computation of the different butterflies (solutions sometimes called *pipelined FFT*). This solution is particularly well suited to the situations, very common in practice, where data are applied in a serial way to the FFT device.

The most straightforward pipelined implementation of radix-2 FFT algorithm is the Radix-2 Multi-path Delay Commutator (R2MDC) described in *Theory and Application of Digital Signal Processing* by L.R. Rabiner and B. Gold, Prentice-Hall, Inc., 1975. The input sequence is broken into two parallel data stream flowing forward, with a correct "distance" between data elements entering a unique butterfly for each step scheduled by proper delays. Both butterflies and multipliers are in 50% utilisation. The implementation requires $\log_2(N)$-2 multipliers, $\log_2(N)$ radix-2 butterflies and $3/2N$-2 registers. It is in fact possible to implement the FFT with a single butterfly within each stage and a single array of N complex elements. The Radix-2 Single-path Delay Feedback (R2SDF) uses the registers more efficiently by storing the butterfly output in feedback shift registers as described in "Pipeline and parallel-pipeline FFT processors for VLSI implementation" by E. H.Wold and A. M. Despain, *IEEE Trans. Comput.*, vol. C-33(5), May 1984, pp. 414–426. A single data stream goes through the multiplier at every stage. It has same number of butterfly units and multipliers as in R2MDC approach, but with much reduced memory requirement (N-1 registers). This approach is by far the most common with many variations with the introduction of radix-4 to further reduce the complexity (R4MDC, R4SDF, R4SDC, R2^2SDC).

By simply appending other stages other than radix-2 or radix-4, e.g. radix-3 or radix-5, the pipelined Cooley-Tukey FFT algorithm can be modified to compute mixed-radix FFT. It is also possible to dynamically change the size of the DFT by skipping some stages and arranging appropriately the routing between the different stages. However, while radix-2 butterflies are rather simple to implement, the complexity of other radices butterflies is significantly higher. Depending on the

number of stages other than radix-2 or 4, the complexity of the FFT device can therefore become cumbersome. The main limitation of these solutions is actually that they implement the basic Cooley-Tukey algorithm which is not self-sorting. It is therefore required to implement an unscrambling stage that consumes either extra memory and/or latency. This unscrambling stage can also become rather complex in the case of mixed-radix FFT.

Another limitation of the use of higher order radices is the necessity to implement rotations both for the implementation of the basic DFT and the rotation by the twiddle factors. As mentioned above, it is better in these cases to rely on the PFA algorithms that gets rid of the multiplication by the twiddle factors using an appropriate indexing of the input and output data. Another advantage of the PFA algorithm is that it can be implemented so as to be self-sorting and in-place thus saving both clock cycles and memory. The issue is that the PFA algorithm is only applicable for factorizations of the size into mutually prime factors. This is not an issue for applications that require the computation of one given length DFT.

As described in "3780-point Discrete Fourier Transformation processor" by L. Zou and X. Huang, *European patent application No. 05300651.6*, August 2005, in the case of a 3780-point DFT, a solution is to pipeline different DFT modules that corresponds to the mutually prime factors, here 140-point and 27-point DFT. The 140-point DFT is computed using the nested Winograd transform decomposed into 4-point, 5-point and 7-point DFT modules while the 27-point DFT is computed using the Cooley-Tuckey algorithm which is decomposed into 9-point and 3-point DFT modules. The advantage of relying on the PFA algorithm is to avoid the multiplication by the twiddle factors. However, the pipelining of the different stages does not allow taking advantage of the in-place nature of the PFA algorithm as several buffers are needed between each stage. In addition, the combination of the Cooley-Tukey algorithm and the pipelining approach makes it necessary to implement an unscrambling stage.

It seems rather difficult to benefit from the advantages of the PFA algorithm using the pipelining approach, especially for handling dynamically changing sizes. An interesting solution is to use the column approach of the FFT that consists in processing the different stages using the same set of processing elements by looping over a single storage element. This solution is implemented in "Optimized discrete Fourier transform method and apparatus using prime factor algorithm" by R. S.

Buchert, S. M. Shahrier and P. Becker, *United State Patent No. 7,028,064 B2*, April 2006, where a single processing element called "PFA circuit" is used to perform the sub-DFT associated to the decomposition of the size into mutually prime factors. This device implements the sub-DFT using a kind of programmable multiply and add (MAC) unit. In fact, the device embeds two such MAC units combined together to take advantage of the inherent symmetries of twiddle factors, thus reducing the overall complexity. This solution is very attractive as long as the size can be decomposed into relatively small prime factors, which is the intrinsic limitation of the PFA algorithm. Indeed, if the decomposition contains a factor of the form $N_i=2^m$, the 2^m-point DFT is computed using the MAC unit, which is not computationally efficient. In addition, the PFA device is not implemented in a self-sorting manner, thus requiring output permutation stages.

Patent Document:

US 2006/7,028,064 B2

One of the main concerns when implementing a DFT core in the purpose of serially computing several DFT is to limit the latency of the DFT computation. The DFT computation module should be as fast as possible. The goal is to achieve series of DFT computation on the flow at the clock of the data stream with a reduced or even advantageously no delay between two DFT computations.

To speed up the computation the invention proposes to use two computation cores in parallel to compute a DFT. Data are dispatched between the two cores according to the even and odd lines of the PFA Ruritanian mapping matrix. Separate storage means are used for each core and means are provided to exchange data between the two separate storage means between the radix computation steps.

The invention relates to a device to compute different sizes of DFT, comprising two butterfly computation unit to compute all the butterflies of different sizes on the flow ; two separate storage means to store data with size equal to half the size of the DFT; means to dispatch input data within the two separate storage means according to the even and odd numbered rows of the Ruritanian maps ; means to redistribute data each time that the radix changes because of a different Ruritanian map and means to retrieve output data from the two separate storage.

According to a particular embodiment of the invention, each storage means further comprises two different memories being used alternatively for input and output operations when changing of radix size.

According to a particular embodiment of the invention, it further comprises virtualization means to encapsulate the two storage for transparent access.

According to a particular embodiment of the invention, it further comprises address conversion means based on the modification of the Ruritanian map.

According to a particular embodiment of the invention, data redistribution is achieved by means of applying a similar reordering on the output data from each butterfly unit and performing pair-wise exchanges on the two reordered streams.

The invention also relates to a method to compute different sizes of DFT comprising a step to compute all the butterflies of different sizes on the flow on two butterfly computation unit ; a step to store data with size equal to half the size of the DFT into two separate storage means ; a step to dispatch input data within the two separate storage means according to the even and odd numbered rows of the Ruritanian maps ; a step to redistribute data each time that the radix changes because of a different Ruritanian map and a step to retrieve output data from the two separate storage.

According to a particular embodiment of the invention, the step to store data stores data alternatively in each storage means within two different memories being used alternatively for input and output operations when changing of radix size.

According to a particular embodiment of the invention, it further comprises a virtualization step to encapsulate the two storage for transparent access.

According to a particular embodiment of the invention, it further comprises address conversion steps based on the modification of the Ruritanian map.

According to a particular embodiment of the invention, data redistribution is achieved by steps of applying a similar reordering on the output data from each butterfly unit and performing pair-wise exchanges on the two reordered streams.

The characteristics of the invention will emerge more clearly from a reading of the following description of an example embodiment, the said description being produced with reference to the accompanying drawings, among which:

Fig. 1 illustrates the DFT core architecture in an exemplary embodiment of the invention.

Fig. 2 illustrates the principle of the computation of a radix-5 butterfly in a matrix form as used in an exemplary embodiment of the invention.

Fig. 3 illustrates the architecture of the Multi-radix Butterfly module in an exemplary embodiment of the invention.

Fig. 4 illustrates the architecture of a multiply and accumulate (MAC) unit as used in an exemplary embodiment of the invention.

Fig. 5 illustrates a variant of the control of the MAC unit in an exemplary embodiment of the invention.

Fig. 6 illustrates a variant of the implementation of the Multi-radix Butterfly module in an embodiment of the invention.

Fig. 7 illustrates the coupling of two MAC units in an exemplary embodiment of the invention.

Fig. 8 illustrates an example of architecture for the DFT engine module in an exemplary embodiment of the invention.

Fig. 9 illustrates an example of accumulator as used in the Address Generator in an exemplary embodiment of the invention.

Fig. 10 illustrates the state machine of a first version of the data manager in an exemplary embodiment of the invention.

Fig. 11 illustrates a variant of the Data Manager module for optimizing sequence of DFTs according to an alternative embodiment of the invention.

Fig. 12 illustrates a variant of the Data Manager module for reducing the latency according to an alternative embodiment of the invention.

Fig. 13 illustrates in details the loading step of the Data Manager module in the last embodiment of the invention.

Fig. 14 illustrates in details the calculation step of the Data Manager module in the last embodiment of the invention.

Fig. 15 illustrates the necessary operations for reorganizing data during the calculation step of the Data Manager module in the last embodiment of the invention.

Fig. 16 illustrates in details the dumping step of the Data Manager module in the last embodiment of the invention.

Fig. 17 illustrates the computation of a radix-3 butterfly.

The device described in this invention as the DFT core can be used by any system requiring the computation of DFT on digital integrated circuits on a set of complex values with size $N=2^p\times3^q\times5^r$, where p, q and r are integers potentially varying from one transform to the other. It realizes an hardware implementation of the GPFA algorithm. In that context, It could easily be extended to any composite sizes that can be factorized into the product of mutually prime numbers, where some or all of these numbers can expressed as the power of a given base. The DFT core is assumed to work between a data source and a data destination functioning at the same clock frequency, denoted as the user clock. The description hereafter of the DFT core

consists of three main independent parts presented in Fig. 1 which are a Data Manager module 1.1, a DFT engine module 1.2 and a Multi-radix Butterfly module 1.3. Besides being designed to work together, the three parts of the DFT core are independent, i.e. could be implemented in a different framework. Data Manager 1.1 is the interface with the user system which means that it receives data values from the data source and sends resulting values to the data destination. DFT engine module 1.2 sequences the elementary DFT stages, enables the Data Manager module 1.1 to feed appropriately the Multi-radix Butterfly module 1.3 with data values and save the resulting values. The DFT engine module 1.2 also controls the Multi-radix Butterfly module 1.3.

The use of a single Multi-radix Butterfly module to compute all the butterflies of different sizes is the key point to address the issue of a flexible design able to compute different sizes of DFT using few resources and being in place. Contrary to the software implementation of GPFA which use different functions for the different step of the radix computation, the invention does not rely on several different radix modules to compute the DFT, but on a single reconfigurable unit.

In a first step, the Multi-radix Butterfly module 1.3 computes the product of a complex matrix of order r where r is the current radix with a column vector of complex data values. In a second step, it applies a rotation on all the elements of the column vector resulting from the first step computation. The complex matrix corresponds to the DFT matrix of order r. The extra rotations are used to apply twiddles factors rotations. The complex matrix formula of a DFT of size r is shown the equation below:

$$B_r = (b_{k,l})_{\substack{0 \le k < r \\ 0 \le l < r}} = e^{\frac{2i\Pi kl}{r}}$$

where i the imaginary unit of complex numbers ($i^2 = -1$).

Actually, the Multi-radix Butterfly module 1.3 works only on real values according to the Cartesian representation. The multiplication of two complex numbers is seen as the product of an order 2 matrix with a column vector. The Multi-radix Butterfly module 1.3 performs the product of a real matrix of order $2r$ with a column vector before making r products of a real matrix of order 2 with r column vectors. Fig. 2A shows the real product calculated by the butterfly matrix with r=5. The equations below show the values of $b_{k,l}$ and $t_{k,l}$:

$$B_r = (b_{k,l})_{\substack{0\le k<r \\ 0\le l<r}} = \begin{cases} \cos\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & \text{if } k \text{ and } l \text{ are of same parity} \\ \sin\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & \text{if } k \text{ is even and } l \text{ is odd} \\ -\sin\left(\frac{2i\Pi\lfloor k/2 \rfloor \lfloor l/2 \rfloor}{r}\right) & \text{if } k \text{ is odd and } l \text{ is even} \end{cases}$$

$$T_r = (t_{k,l})_{\substack{0\le k<r \\ 0\le l<r}} = \begin{cases} \cos(i\alpha\lfloor k/2 \rfloor) & if\ k = l \\ \sin(i\alpha\lfloor k/2 \rfloor) & if\ k = l+1 \\ -\sin(i\alpha\lfloor k/2 \rfloor) & if\ l = k+1 \\ 0 & else \end{cases}$$

A first optimization consists in reducing the number of multiplications which consumes resources and thus energy by using properties of the matrices T and B. Coefficients $b_{2k,0}$, $b_{2k+1,1}$, $b_{0,2l}$ and $b_{1,2l+1}$ equal 1 whereas coefficients $b_{2k,1}$, $b_{2k+1,0}$, $b_{1,2l}$ and $b_{0,2l+1}$ are null. In the same way, coefficients $t_{0,0}$ and $t_{1,1}$ values equal 1 whereas coefficients $t_{1,0}$ and $t_{0,1}$ values are null. No multiplication is needed for these coefficients. Fig. 2B shows the product calculated by the butterfly matrix after simplification.

Fig. 17 is an illustration of the radix-3 butterfly. This figure shows the three entries *a, b* and *c*. It also shows the angles applied to determine the twiddle factors and the rotations. As can be seen on this figure, the first value *a* is always affected with an angle of 0 meaning that it is always multiplied by the value 1 in the computation of each output. It can also be noted that for the computation of the first output *A*, all values are affected with an angle of 0 and that the rotation term W_N^0 being elevated to the power 0 is always equal to 1, meaning that no multiplication are needed for the computation of the first output. This remark is verified for all radices.

The retained solution for implementation is the treatment of each row of the matrix independently and in parallel by elementary computation units. One elementary unit comprises typically two MAC units, one for each real part of the complex. In the following we consider the real matrix. Excepting the first two rows that require only adders as explained above, MAC units are used to multiply each element of one row of the matrix B or T with each element of the column vector of data values and cumulate intermediate results to obtain the final column vector. Since radix-5 must be supported, 8 MAC units are implemented but more units could be

implemented if higher radix were required. Moreover, this implementation enables to inactivate MAC units when a butterfly of lower radix is computed thus saving energy. Fig. 3 represents the layout of the adders and the MAC units in a first embodiment of the Multi-radix Butterfly module 1.3. An example of MAC unit is presented in Fig. 4. The inputs X and Y are connected to two buffers 4.1 and 4.2 before the multiplier 4.6. The output of the multiplier 4.6 is connected to a buffer 4.3 that is connected to the adder 4.7. This adder also takes the output of the multiplexer 4.5 and puts its output into the buffer 4.4 that is connected back to one input of the multiplexer 4.5 to realize the accumulation. The accumulator can be reset using the input C of the multiplexer by selecting a null input on C. The output of the buffer 4.4 is the output of the MAC unit. Fig. 5A shows the timing sequence for the processing of three contiguous radix-3 butterflies inside MAC unit 3.2 on data vector D, D' and D'' providing intermediate results X, X' and X''. Elements of the fourth row of matrix B and elements of the column vector are multiplied one by one and add to an accumulator previously reset to null value before each new butterfly computation. The p_i values result from the product of d_i values with $b_{3,i}$ values while s_i values result from the accumulation of p_j values with $j \leq i$. The intermediate result x_3 is the last value s_i.

An optimized version of the Multi-radix Butterfly module 1.3 fuses the two stages of MAC units (butterfly and rotation) into a single one. Since the two first elements of one row of the radix matrices B are always (1,0) for even row and (0,1) for odd one, MAC unit attached to even row can be preloaded with the first value of column vector whereas MAC unit attached to odd row can be preloaded with the second one. It enables to free processing times in MAC units between each butterfly computation as shown in Fig. 5B. These times are used to insert rotation processing as shown in Fig. 5C. Serialization between the two stages is still necessary and requires the coupling of two MAC units in order to reinsert couples of values. In Fig. 5, the intermediate result x_2 provided by the MAC unit attached to the third row of matrix B is necessary and is inserted with the intermediate result x_3, the y_i values result from the product of x_i values with $t_{3,i}$ values while y_i values result from the accumulation of x_j values with $j \leq i$. The final result r_3 is the sum of y_i values.

A second optimization that depends on the MAC units ability to make subtraction is to replace the addition by a subtraction when coefficients values $b_{k,l}$ and $t_{k,l}$ (k and l of different parity) come with a negative polarity. Thus, the need to inverse the sign of values is no more relevant and only a flag indicating the type of operation

is required. This optimization is very useful when it comes to consider opposed angle which is the case when a rotation of π must be done on a radix butterfly, when the DFT is configured as an Inverse DFT (IDFT) or when angles should belong to the interval $[\pi;2\pi]$ ($\sin(-\alpha) = -\sin(\alpha)$).

This embodiment of the Multi-radix Butterfly module 1.3 is compatible with different representation of data values and fixed or floating point can be used. In the case of fixed point usage, extra bits inside MAC Units can be used in order not to saturate accumulator. It requires the extension of input data values to more bits and the presence of a scaling module to control the number of bits used to represent data. . When most significant bits are removed, the scaling is advantageously achieved with control of the saturation. When removing least significant bits, the scaling is advantageously achieved with rounding to the closest value. On issue when performing scaling is to apply the optimal tradeoff between dynamic and precision. Advantageously, this scaling module is dynamic which means that it could be adapted to the size of the DFT being calculated. For example, it could be realized using two scaling modules able to resize the results at multi-radix butterfly outputs. The first one resizes MAC Units outputs in order to reinsert the results inside the MAC units for rotation application. Since extra bits are used, saturation management is not required. The second scaling module must manage saturation in order not to output erroneous values and could preferably have the ability to dynamically control the resizing window in order to adjust between dynamics range and precision of output values.

Finally, the Multi-radix Butterfly module 1.3 comes in its last embodiment with the ability to adjust output gain in the case of null twiddle factors. It becomes very useful in DFT where no rotation is needed during the last stage. Thus, the overall gain of the DFT can be adjusted at the very end of the computation. This new functionality is made by no more using matrix T but a diagonal matrix G whose coefficients $g_{k;l}$ are equal to the desired gain adjustment and by deviating inputs of adders (3.1 for example) to unused MAC units which is the case when Multi-radix Butterfly module 1.3 has been sized for a maximal radix r and last stage uses a radix r' inferior to r. Fig. 6 presents an example of implementation of the Multi-radix Butterfly module 1.3 in its optimized version. Input data d_i is inserted alternating real and imaginary part of the complex data vector. Coefficients $b_{k,l}$, $t_{k,l}$ and $g_{k,l}$ are inserted also. Multiplexers 6.3a, 6.3b and 6.4 are used to deviate the computation of the first two rows of the B matrix from the adders to the last MAC units in order to adjust the gain since multiplications

are required. In this implementation, real part r_{2i} and imaginary part r_{2i+1} appear sequentially at outputs of the Pair of Adders module 6.1 and MAC units (6.2 for example) but could appear at the same time. The pair of adders makes the computation of the two first rows of the matrix B by adding the real part and the imaginary part individually, serializing and scaling the two results, and delaying them to adjust with the pairs of MAC units' latency. The pairs of MAC units make the rest of the computation. The Serialization module 6.5 serializes the results from adders and MAC units and outputs results r_i alternating real and imaginary part of the complex values but could output at the same time in another implementation real and imaginary part of complex values at a two times lower frequency. Fig. 7 shows an example of implementation of the scaling module inside a couple of MAC Units. Multiplexers 7.4, 7.5, 7.6 and 7.7 are used to alternate phases of butterfly computation and phases of rotation computation. The module 7.1 delays input data in order to insert it into accumulators of MAC units via multiplexers 7.2 and 7.3. The latter can also reset the accumulators with null value. When results appear at the outputs of MAC units for the first time, they are buffered and scaled into module 7.8. When a window for rotation for rotation computation appears, the two values output sequentially and re-enter inside MAC Units with either twiddle factors or gain coefficient. In this implementation, when results appear at the outputs of MAC units for the second time, they are buffered inside blockers 7.9 and 7.10 before sequentially entering inside the single scaling module 7.11 saving a second scaling module that would have used resources. The scaling module 7.11 can work at a two times lower frequency.

It is to be noted that this Multi-radix butterfly can be used in different architecture designs according to the actual desired computation. While being particularly suited to the described DFT computation design it is not limited to this one.

The DFT Engine module 1.2 is the second main element of the DFT core module and takes place between the Data Manager module 1.1 described in another section and the Multi-Radix Butterfly module 1.3. Its role is to configure, run and coordinate the two other main elements. This module comprises expression means to express the DFT according to its size in a sequence of elementary butterflies. It also comprises scheduling means to schedule the DFT in the form of a sequence of nested loops and to generate for each butterfly the relevant input and output data addresses in

the data manager. It also comprises customization means to customize the sequence of nested loops according to the size of the DFT.

A detailed view of the DFT Engine module inserted in the DFT core is depicted on Fig. 8. It consists in 6 sub-elements. The Radix Selection module 8.1 sequences the DFT stages. The Global Settings module 8.2 configures all the other sub-elements of the DFT engine. The Address Generator module 8.3 runs the elementary butterflies that compose a DFT stage by driving the Data Manager module 1.1 and the Multi-Radix Butterfly module 1.3 through the sub-elements Radix Matrix module 8.4 and Twiddle Matrix module 8.5. These two last sub-elements convert indexes into coefficients useable by the Multi-Radix Butterfly. Depending on the implementation of the Multi-Radix Butterfly module 1.3, the Scaling Table module 8.6 may become optional. In fixed point representation, it configures the scaling of intermediate results inside the Multi-Radix Butterfly. In the DFT Engine described herein, the Data Manager module 1.1 can be seen as a single memory with one reading port and one writing port.

The DFT Engine starts when the Data Manager module contains the entire set of input data. It runs with its own independent clock since the Data Manager acts as a buffer. Knowing the size of the DFT (that can be set up as an input or computed using a counter during storage), the Radix Selection module 8.1 prepares the sequence of radices 2, 3, 5 (and eventually 4 in some embodiments) corresponding to decomposition into the DFT stages, and send the first radix to the Global Settings module 8.2 which configures all the other modules. Once configured, the Address Generator module 8.2 starts and drives all the others modules automatically. Using addresses, it enables the Data Manager module 1.1 to feed the Multi-Radix Butterfly with data in the appropriate order and to save intermediate results in the right places. Using indexes, the Radix Matrix module 8.4 and the Twiddle Matrix module 8.5 produce the coefficients required for the calculation of the butterflies. When the Address Generator module 8.3 reaches its last butterfly, the Radix Selection module 8.4 selects the next radix (starting again the Address Generator module 8.3) until the last one. Once the Radix Selection module 8.4 ends, the Data Manager module 1.1 contains the final result of the DFT in the right order.

When computing a DFT of size N where $N=N_2\times N_3\times N_5=2^p\times 3^q\times 5^r$, the GPFA algorithm processes successively the DFT $W_{N_2}^{[N_3N_5]}$, $W_{N_3}^{[N_2N_5]}$ and $W_{N_5}^{[N_2N_3]}$ following the Ruritanian map. The re-indexing matrices R and R^{-1} are absorbed inside each function.

This sequence defines a first loop and initiates the calculation of the two-dimensional DFTs of size $N_2 \times (N_3 N_5)$, $N_3 \times (N_2 N_5)$ and $N_5 \times (N_2 N_3)$ decomposed relatively in a second loop in p radix-2 stages, q radix-3 stages and r radix-5 stages. Each radix can be either of type 1 or type 2 according to GPFA algorithm.

The Radix Selection module 8.1 is in charge of these two loops. In the described embodiment, counters maintain the number of radix to do for each radix size and type and are initialized with the DFT decomposition into radix stages (radix-2 type-1, radix-2 type-2, radix-5 type-1, radix-5 type-2, radix-3 type-1 and radix-3 type-2). In an alternative embodiment, radix-4 stages of type 1 and 2 enter in the decomposition. These decompositions can either be calculated or stored in a table (limiting the number of managed DFTs to the number of stored decompositions). When starting a new radix stage, the corresponding counter is decremented. Priorities are set among the counters to sequence the radix size and type according to the GPFA Algorithm. For example, counters associated to radix-2 stages have priority on the ones associated to radix-5 stages which have priority on the ones associated to radix-3 stages. In an alternative embodiment, for example, the sequence of radix-4 and radix-2 must form a palindrome.

At last, the Radix Selection module 8.1 drives the Global Settings module 8.2. It outputs the selected radix with size and type, and the progression inside the calculation of $W_{N_i}^{[N_o]}$. This progression is defined by the stage number inside the stages of same prime factor. The number of stages of same prime factor corresponds to the powers of prime factors. The distinction between radix size and prime factor is necessary to sequence radix-2 and radix-4. In an alternative embodiment, the selection of a radix-4 increments the stage number by 2.

The Global Settings module 8.2 is used to produce useful global parameters to other sub-modules of the DFT Engine module 1.2.

The Address Generator module 8.3 consists of six imbricate loops. The Global Settings module 8.2 generates maximum iterations of the loops and increments for address calculation. The table below lists the loops from outer to inner for radices 2, 3 and 5. N_o and N_i are such that the size $N_o \times N_i = N$ where N is the size of the DFT and N_i a power of the current radix. These two values can either be calculated or stored. The values *radix*, *stage* and *power* come from the Radix Selection module 8.1.

Description	Name	Radix of type 1	Radix of type 2

Number of divisions	NB_DIVISION	$radix^{(stage-1)}$	$radix^{2(stage-1)-power}$
Jump to next division	JTN_DIVISION	$N_o \times radix^{power-(stage-1)}$	$N_o \times radix^{power-(stage-1)}$
Number of subdivisions	NB_SUBDIVISION	1	$Radix^{power-stage}$
Jump to next subdivision	JTN_SUBDIVISION	-	$N_o \times radix^{stage}$
Number of parallel schemes	NB_TRANSVERSE	N_o	N_o
Jump to next parallel scheme	JTN_TRANSVERSE	N_i	N_i
Number of interleaved butterflies	NB_BUTTERFLY	$Radix^{power-stage}$	$Radix^{power-stage}$
Jump to next interleaved butterfly	JTN_BUTTERFLY	N_o	N_o
Number of elementary butterfly	NB_SUBB	1	Radix
Jump to next elementary butterfly	JTN_SUBB_IN JTN_SUBB_OUT	- -	$N_o \times radix^{(stage-1)}$ $N_o \times radix^{power-stage}$
Number of inputs/outputs	RADIX	Radix	Radix
Jump to next input/output	JTN_INPUT JTN_OUTPUT	$N_o \times radix^{power-stage}$ $N_o \times radix^{power-stage}$	$N_o \times radix^{power-stage}$ $N_o \times radix^{(stage-1)}$

The Global Settings module 8.2 also calculates the start angle for the twiddle factors generation since all the twiddle factors are multiple of a single angle. The latter depends on DFT size and stage number. It is the product of two factors. The first one depends only on the stage number and the radix size. Its value is $2\pi/prime\_factor^{stage-1}$ $(1 \leq stage \leq power)$. In the described embodiment, an indexed table is used in order to store the angles (see Twiddle Matrix module 8.5). The table contains the $2i\pi/prime\_factor^{max\_power}$ $(0 \leq i \leq prime\_factor^{max\_power}-1)$ values where max_power is the maximum of powers among the managed DFTs. Then, the index $prime\_factor^{max\_power}$ corresponds to angle 2π. In operation, a scaling is applied to make sure that the index $prime\_factor^{power}$ corresponds to the angle 2π . Therefore, the value of the first factor becomes $prime\_factor^{stage-1+max\_power-power}$ $(1 \leq stage \leq power)$. The second factor is the quotient $N_o=N/prime\_factor^{power}$. Since $prime\_factor^{power}$ corresponds to the angle 2π, only the value $(N_o \ mod \ prime\_factor^{power})$ can be used. This allows maintaining the angle in the interval $[0, 2\pi]$.

In the GPFA algorithm, rotations must be applied on the butterfly matrix *B* introduced in Multi-Radix Butterfly description. Applying rotation modifies the matrix in a notable way. The rotated butterfly matrix can be deduced from the reference one by a transposition or by a permutation of the columns or both. Advantageously these transposition and permutation are obtained by adapting the addressing of the reference table. The required operations depend on the value *((No modulo Ni) modulo Radix)* where N_i is a power of current radix and the product $N_o \times N_i$ the size of the DFT. There is one possible value for radix-2, two possible values for radix-3, two possible values for radix-4 and four possible values for radix-5. Each possible value corresponds to a different matrix. The following table shows the relation between possible values and operations. The Global Settings module 8.2 generates the flags for inverting sinus values for the Multi-Radix Butterfly in the case of transposed matrices and the flags for permuting columns for the Radix Matrix module 8.4.

((No mod Ni) mod Radix)	0	1	2	3	4
Radix-2		Orig.			
Radix-3		Orig.	Trans.		
Radix-4		Orig.		Trans.	
Radix-5		Orig.	Perm.	Perm.+Trans.	Trans.

Orig.: Original; Trans.: Transposed; Perm.: Permuted

The Radix Matrix module 8.4 provides the Multi-Radix Butterfly module 1.3 with the coefficients of the butterfly matrix *B* introduced in the Multi-Radix Butterfly description. In the described embodiment, the Multi-Radix Butterfly requires coefficients being produced column by column. Thus, the Radix Matrix module takes as input the current radix, the column number to output, and the flag for permuting columns. The coefficients $b_{k,l}$ can either be calculated or stored in a table. In that case, entire columns are stored and indexes are formed with the current radix and the requested column index. Since the Multi-Radix Butterfly works on real numbers, the Radix Matrix module 8.4 must then form columns of real numbers, alternating real parts and imaginary parts of the complex coefficients. When the Radix Matrix module 8.4 is requested to produce a column of the complex matrix, it first generates the pairs (cosine, sinus) with the sign of the sinus values for the all complex coefficients of the column and then the pairs (sinus, cosine) with opposite signs. The sign tells the Multi-

Radix Butterfly module 1.3 whether addition or subtraction must be done on the sinus value.

The Twiddle Matrix module 8.5 provides the Multi-Radix Butterfly module 1.3 with the coefficients of the matrix T introduced in the Multi-Radix Butterfly description. It takes as input the current radix and the angle of rotation of the second element of the butterfly. The outputs are a column of coefficients and a column of signs. Given the twiddle factor α determined to rotate the second element of the butterfly, the Twiddle Matrix module 8.5 generates the sequence of twiddle factors 0, α, 2α, ... necessary to rotate all the elements of the butterfly. In the described embodiment, the Multi-Radix Butterfly requires coefficients column by column and works with real numbers. When the Twiddle Matrix module 8.5 is requested to produce twiddle factors, it first generates the pairs (cosine, sinus) with the sign of the sinus value for all complex coefficients of the column and then the pairs (sinus, cosine) with opposite signs. The sign tells the Multi-Radix Butterfly module 1.3 whether addition or subtraction must be done on the sinus value.

Cosines and sinuses values can be calculated or stored in a table. In that case, a single table for each prime factor is sufficient to store all values. The size of each table depends on the maximum power attached to the prime factor among the managed DFTs. The granularity of the table becomes $2\pi/prime\_factor^{max\_power}$ which contains the cosines and sinuses values of the angles $2i\pi/prime\_factor^{max\_power}$ $(0 \leq i \leq prime\_factor^{max\_power}-1)$. A first optimization to reduce the sizes of tables is to use a property of sinus function. Angles i and $prime\_factor^{max\_power}-i$ $(0 \leq i < prime\_factor^{max\_power})$ are equal modulo π and have equal cosines values and opposed sinus values. Only angles inferior to π are necessary and the sizes of the tables become relatively $2^{max\_power(2)}/2$, $\lfloor 3^{max\_power(3)}/2 \rfloor$ and $\lfloor 5^{max\_power(5)}/2 \rfloor$ for the prime factors 2, 3 and 5. If the asked angle α is superior to $\lfloor prime\_factor^{max\_power}/2 \rfloor$, the new considered angle is $\lfloor prime\_factor^{max\_power}/2 \rfloor - \alpha$ and the sign of fetched sinus is opposed.

The Address Generator module 8.3 can be decomposed into 3 main parts, a first one is dedicated to the management of the loops, a second one to the accumulators used to generate addresses for the Data Manager module 1.1 and a third one to commands.

In the GPFA Algorithm, loops are used to determine the inputs and outputs of the butterflies. In the described embodiment, the Address Generator module 8.3 uses 6

cascaded loops which are first configured by the Global Settings module 8.2. The first loop TRANSVERSE (outermost loop) decomposes the two–dimensional DFT in many one-dimensional DFTs. The second loop DIVISION counts the number of divisions which correspond to the divide and conquer approach of the Cooley-Tukey algorithm. In the case of a radix of type 2, a third loop SUBDIVISION is used. The fourth loop BUTTERFLY sequences the interleaved butterflies. In the case of a radix of type 2, a fifth loop SUB-BUTTERFLY sequences the elementary butterflies which are of type 1. Finally, the sixth loop I/O (innermost loop) sequences the inputs and outputs of the butterfly. In the described embodiment, loops are implemented using counters to store the iteration number. Once loops are configured with the maximum number of iterations represented in Fig. 9A by the module 9A.1, the counter of the innermost loop 9A.2 starts counting. When reaching its last iteration or end, it increments the counter of the outer loop 9A.3 and resets itself. For all other counters such as 9A.4, they do the same with the counter of their outer loop but at the condition that the counters of their inner loop 9A.2 and 9A.3 reach their last iterations. The Address Generator module 8.5 ends when all counters reach their last iteration.

The addresses at destination to the Data Manager module 1.1 are linear combinations of the number of iteration of each loop. The coefficients of the linear combination are the increments or JTN values first initialized by the Global Settings module 8.2. In the described embodiment, no multiplication is made in order to reduce complexity and the calculation of the data addresses use intermediate accumulators for each loop. When resetting a counter of a loop, the accumulator attached to it updates its value with the one attached to the outermost loop whose counter does not reset. When incrementing a counter of a loop, the JTN value attached to the loop is added to the correspondent accumulator. The Fig. 9B illustrates the accumulator 9B.7 attached to innermost loop I/O. Depending on the end condition of the inner counters 9A.2 to 9A.6, the multiplexers 9B.2 to 9B.6 either reinitialize the accumulator with one of the values of inner accumulators 9B.8 to 9B.12 with priority to the outermost 9B.12, or use the value of the multiplexer 9B.1. This multiplexer allows or not the addition of the JTN value and is driven by the end condition of the first inner loop. In case of the innermost loop, this multiplexer 9B.1 is bypassed to use the JTN_INPUT value. The same scheme is used for inner accumulator with a decreasing number of multiplexers.

The innermost accumulator modulo N 9B.13 is the read address at destination to the Data Manager module 1.1. The write address can be generated in two ways. The

first uses two extra accumulators for the 2-innermost loops. The latter are the same as read address but where JTN values are permuted. The second uses a two dimensional array whose number of row and column is equal to the size of the radix. The generated read addresses are written row by row in the array. Once full, the written addresses are obtained by reading the array column by column. The counter CNT_IO can be used directly to drive the Radix Matrix module 8.4 which waits for column number. Because calculation starts with radix calculation, no delay on this address signal is needed. The angles to be applied for twiddle factors generation are given as the product of the reference angle generated by the Global Settings module 8.2 by CNT_BUTTERFLY. The resulting angle is then corrected to fit between 0 and 2π and sent to the Twiddle Matrix module 8.5. Each command must be sent so that data and coefficients at destination to the Multi-Radix Butterfly occur in phase.

In this first embodiment, all the DFTs of size $N=N_2\times N_3\times N_5=2^p\times 3^q\times 5^r$ are achievable. A radix-4 can be used to reduce the number of radix-2 stages and the calculation time at the same time. Nevertheless, some DFT become impossible to calculate because of their decomposition as a palindrome as showed by Johnson and Burrus. Suppose $N=N_{128}.N_3$ is the size of the DFT. $N_{128}=(4\times 2\times 4\times 4)$ cannot be decomposed correctly. A Radix-8 is required to make $N_{128}=(4\times 8\times 4)$.

Considering the architecture of the Multi-Radix Butterfly module 1.3, the calculation of a radix-8 butterfly would require 16 MAC units. To keep low the DFT complexity, the radix-8 butterfly can be done by sequencing a first stage of radix-2 butterflies and a second stage of radix-4 butterflies. Thus, no extra MAC unit is needed. Using the first part of the demonstration of the GentleMan-Sande Algorithm, the radix-8 matrix can be easily decomposed into the product:

$$W_8=\left(P_4^2\left(I_2\times W_4\right)\right).\left(D_4^2\left(W_2\times I_4\right)\right)$$

This radix-8 will need to be raised by 1, 3, 5 or 7 depending of the DFT size. The lemma used for rotating a DFT gives:

$$W_8^{[r'=\mathrm{mod}(r,8)]}=\left(P_4^2\left(I_2\times W_4^{[r''=\mathrm{mod}(r',4)]}\right)\right)\left(D_4^{2[r']}\left(W_2\times I_4\right)\right)$$

This radix-8 module is still incomplete since the application of twiddle factors is not there. Suppose $M_8^{[kr]} = \begin{pmatrix} 1 & 0 \\ 0 & \omega_8^{kr} \end{pmatrix}$ and $T_4^{[kr]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega_4^{kr} & 0 & 0 \\ 0 & 0 & \omega_4^{2kr} & 0 \\ 0 & 0 & 0 & \omega_4^{3kr} \end{pmatrix}$.

A complete radix-8 module with twiddle factors becomes:

$$D_8^{1[kr]}.W_8^{[r']} = \left(T_4^{[kr]} \times I_2\right).\left(I_4 \times M_8^{[kr]}\right).\left(P_4^2\left(I_2 \times W_4^{[r'']}\right)\right)\left(D_4^{2[r']}\left(W_2 \times I_4\right)\right)$$
$$D_8^{1[kr]}.W_8^{[r']} = \left(T_4^{[kr]} \times I_2\right).P_4^2.\left(M_8^{[kr]} \times I_4\right).\left(I_2 \times W_4^{[r'']}\right).\left(D_4^{2[r']}\left(W_2 \times I_4\right)\right)$$
$$D_8^{1[kr]}.W_8^{[r']} = \left(P_4^2\left(I_2 \times T_4^{[kr]}\right)\right)\left(I_2 \times W_4^{[r'']}\right).\left(\left(M_8^{[kr]} \times I_4\right)D_4^{2[r']}\right)\left(W_2 \times I_4\right)$$

Finally, define a diagonal matrix $D'^{p}_{q,t}$ of order $N=pq$ by:

$$D'^{p}_{q,t}\ (\mathrm{j,k}) = \omega_N^{st} \qquad \text{if } j = k = sq + m,$$
$$= 0 \qquad \text{otherwise.}$$

The previous formula becomes:

$$D_8^{1[kr]}.W_8^{[r']} = P_4^2\left(I_2 \times \left(T_4^{[kr]}W_4^{[r'']}\right)\right)D'^{2}_{4,kr}\,D_4^{2[r']}\left(W_2 \times I_4\right)$$

The main interest of this formula is that the twiddle factor applied to the first element of both radices 2 and 4 butterflies results is always 1. It results in making some of the multiplication of the radix 4 within the radix 2 computation. Thus, the calculation of the first element does not require multipliers but only adders. This enables keeping the couple of adders inside the Multi-radix Butterfly module 1.3. Finally, the first stage is a typical radix-2 stage with specific twiddle factors and the second stage a typical radix-4 stage with interleaved outputs. No major modifications are needed.

In order to form palindromes, the Radix Selection module 8.1 is updated to sequence first radix-4 stages of type 1, an optional radix-2 type 1 or radix-8 stage and finally radix-4 stages of type 2. Radix-8 stage is composed of a radix-2 stage and a radix-4 stage and flags are created to differentiate them from regular ones. Moreover, since the power value is attached to the prime factor 2^p, the stage counter must be

increased by 2 for a radix-4. In the Global Settings module 8.2, if radix is 4, settings for loops of Address Generator module 8.3 are obtained by replacing all values *stage-1* by *stage-2*. Then, since the two parts of a radix-8 butterfly are very similar to a radix butterfly of type 2, loops settings are easily implemented. The table below summarizes the settings for loops attached to radix-8.

Description	Name	Radix-8 1st part	Radix-8 2nd part
Number of divisions	NB_DIVISION	$2^{(stage-1)}$	$2^{(stage-3)}$
Jump to next division	JTN_DIVISION	$N_o \times 2^{power-(stage-1)}$	$N_o \times 2^{power-(stage-2)}$
Number of subdivisions	NB_SUBDIVISION	1	1
Jump to next subdivision	JTN_SUBDIVISION	-	-
Number of parallel schemes	NB_TRANSVERSE	N_o	N_o
Jump to next parallel scheme	JTN_TRANSVERSE	N_i	N_i
Number of interleaved butterflies	NB_BUTTERFLY	$2^{power-stage-2}$	$2^{power-stage}$
Jump to next interleaved butterfly	JTN_BUTTERFLY	N_o	N_o
Number of elementary butterfly	NB_SUBB	4	2
Jump to next elementary butterfly	JTN_SUBB_IN JTN_SUBB_OUT	$N_o \times 2^{(stage-1)}$ $N_o \times 2^{(stage-1)}$	$N_o \times 2^{(stage-1)}$ $N_o \times 2^{power-stage}$
Number of inputs/outputs	RADIX	2	4
Jump to next input/output	JTN_INPUT JTN_OUTPUT	$N_o \times 2^{power-stage}$ $N_o \times 2^{power-stage}$	$N_o \times 2^{power-stage}$ $N_o \times 2^{(stage-2)}$

In the Radix Matrix module 8.4, coefficients of the radix-4 matrix are inserted into the table if the storage of coefficients is used. In the Twiddle Matrix module 8.5, the calculation of radix-2 butterflies inside radix-8 imposes to add an angle offset to the generated twiddle factors. This offset depends on the rank of radix-2 inside the radix-8 and its value is $e^{i\frac{\pi}{4} \cdot rank \cdot (N_o \bmod 8)}$.

The Data Manager module 1.1 shall support all the operations for accessing data including loading and dumping data. It shall also come with a state machine that checks loading, DFT computation and dumping. The state machine depicted on Fig. 10 shows the functioning of the module. In its very basic function shown in Fig. 11a, the Data Manager module 1.1 consists of one double port memory (one reading and one for writing) of size N_{max} (supposing N_{max} is the maximum size of managed DFT) and multiplexers to root data and addresses. Loading operation is done by linking input samples to the write port of the memory and by addressing linearly the memory thanks to a counter for example. During computation, data is retrieved from the reading port of the memory and results are written trough the writing port. Addresses from the DFT engine are used at this time. The latency of the Multi-radix Butterfly module 1.3 protects from read/write overlaps, especially during radix of type 2 where inputs and outputs of butterflies are different. When completed, final results are dumped linearly from the reading port. Since loading and dumping are linear operations, a new loading can start just after the beginning of a dumping. Fig. 11b shows the timing sequence of the processing.

Several independent optimizations can be applied in order to reduce latency and even enable the computation of a constant flow of input samples.

The first optimization consists in increasing the DFT core driving clock which is independent from the user clock. It can severely reduce latency.

The second optimization pipelines the operation of loading, computing and dumping and is presented in Fig. 11A. This pipeline is made by using a second memory identical to the first one. While computing on samples of one memory, the other memory becomes available for loading and dumping. The two different memories are used alternatively for input/output operations and as working memory. This optimization reduces latency and enables the sequencing of successive DFTs provided that the loading of a new DFT takes more time than the computing of the current DFT. Combined with the previous optimization, it enables the sequencing of a wider panel of DFTs. Fig. 11B and Fig. 11C show the processing without and with first optimization.

The third optimization reduces the computation time without increasing the DFT internal clock which can be very useful when clock frequency reaches the limit of the target device. This solution exploits parallelism and is independent of the actual multi-radix butterfly unit, it could be used with any butterfly computation unit. This

solution could also be implemented in software, for example on a multi core processor. During computation, a first half of samples is treated inside the Multi-radix Butterfly module 1.3 while the second half is treated in a new instantiated Multi-radix Butterfly module. The cut-out is made according to the even- and odd-numbered rows of the Ruritanian maps. The DFT engine remains unchanged but now outputs only addresses relatives to the points figuring on even rows of the Ruritanian map attached to the current radix. When addressing an even row, addresses for the next one (odd) will be deduced and points from both rows can be treated simultaneously inside two independent computation units. The advantage of this cut-out is that rows can be easily computed in parallel because of the strict identical operations made on rows. Several solutions exist but all must distribute points so that points figuring on even and odd rows of the Ruritanian map attached to the foreseen radix are always placed in two separate memories. This implies to redistribute points each time that the radix changes because of a different Ruritanian map. Thus, four distributions are foreseen: a first for loading, a second for passing from a radix to the same, a third for passing from a radix to another one and a last for dumping.

The solution described thereafter in details resolves the issue of data distribution and storage into two memories of size $N_{max}/2$, where N_{max} is the maximum size of managed DFTs. Because read and write addresses will differ when changing radix, two more identical memories will be used to prevent read/write overlaps. Moreover, the particular radix sequence (4, 3, 5) will be used because of possible optimizations and eases of implementation. Fig. 12A and 12B show the overall solution with its time sequence of the processing. In order to keep at most the existing architecture, an abstraction layer encapsulates the four memories so that DFT engine access to memories as if there were a single one of size N_{max}. This layer introduces virtual addresses corresponding to addresses where data should be and physical addresses corresponding to addresses where data are really stored. This layer consists of selves-building address conversation tables based on the compression of the Ruritanian maps and logics for reorganizing data.

The compression transforms the two-dimensional Ruritanian map attached to a prime factor into a new Ruritanian map of two rows with the first row containing all the points from the even rows of original Ruritanian map and the second row all the points from the odd ones. The used algorithm is the following: Suppose N is the size of the DFT. Points are inserted one by one from 0 to N-1 into the new array. When

inserting, points are inserted on rows from column 0. Point is inserted on first row if it belongs to an even row of the Ruritanian map and on second row otherwise. When completed, a circular rotation is applied on second row to place the point N_i under the point 0. An exception is made for Ruritanian map containing an odd number of rows and consists in adding a padding point on second row each time N_o points have been inserted. However, it can be demonstrated that compressed Ruritanian maps attached to radix-3 and radix-5 contain always even points on their first row and odd points on their second one.

This compressed Ruritanian map represents the position of data inside the two memories. Suppose N is the DFT size, the address conversion table used in abstraction layer contains the coordinates, row and column, of data 0 to data N-1 inside the compressed Ruritanian map. There are as many conversion tables as different Ruritanian maps excepting conversion tables attached to radix-3 and radix-5 which are always the same (see previous paragraph). This is why conversion tables for radix-4 must be built before processing a DFT. Suppose N is the DFT size, R_r the Ruritanian map of size N_i rows × N_o columns where N_i is a power of r and $N_o=N/N_i$ and CT_r the conversion table. For regenerating a complete conversion table, pre-computed values are necessary such as the row R_1 in which point number 1 is placed in Ruritanian map and the final column FCN_i in which point number N_i is placed in compressed Ruritanian map. The accumulator Rx contains the row in which point number x is placed inside the Ruritanian map. The counter Jump is used to add padding element.

```
Point = 0;
Rx = 0;
Index0 = 0;
Index1 = FCNi;
Jump = 0;
While Point < N do
    If (Rx is even) then
        CT(Point) = (0,Index0);
        Index0 = Index0 + 1;
    Else
        CT(Point) = (1,Index1);
        Index1 = Index1 + 1;
    End if;
```

Jump = Jump + 1;
If (Jump = N_o) then
Jump = 0;
Index1 = Index1 + 1;
End if;
Rx = (Rx + R1) mod N_i;
Point = Point +1;
End while;

During data loading, data is distributed inside the two memories so that data can be easily accessed for future radix-4 butterflies as shown in Fig. 13. Radix-4 butterflies are computed at first because the decompositions of the managed DFT sizes imply always radix-4 butterflies, enabling optimization on data loading (Same argument is used for dumping which will come always after a radix-3). Since both loading and address conversion table building are made linearly, loading can start just after the beginning of radix-4 address conversion table building.

During butterflies computation, input data is fetched from the two memories just by converting the read address from the DFT engine and applying the result to the read address ports of the four memories. As shown on Fig. 14A, multiplexers 14.1 and 14.2 are used at data bus outputs to select the memories that contain data. The storage of results from Multi-radix Butterfly module with write address from DFT engine requires first to change pair of memories when next radix is different of the current one (radix-2 is considered as a radix-4) preventing read/write overlaps and when radix is the last of the sequence so that loading and dumping data are never in the same memory. Multiplexer 14.3 enables to do this. Then more operations shall be applied depending on cases. The first operation works on data. Data from both multi-radix butterflies is reorganized independently by set of 8 elements according the Fig. 14.4. The sequence (d(0), d(1), d(2), d(3), d(4), d(5), d(6), d(7)) becomes (d(0), d(4), d(1), d(5), d(2), d(6), d(3), d(7)). It implies latency which shall be applied to addresses to maintain both signals in phase. This operation must be made when current radix is of type 2 or when radix is 4 and belongs to radix-8 computation. It is necessary to alternate data that will come either to the first memory or the second one. The second operation works still on data. This time data from both Multi-radix Butterflies is scrambled between them by set of 2 elements. Supposing the sequence (d(0),d(1)) comes from the first butterfly and (e(0),e(1)) comes from the second one

simultaneously, this scrambling scrambles the quadruplet and outputs simultaneously the sequence (d(0),e(0)) to the first memory and (d(1),e(1)) to the second one as shown by Fig. 14B (Element 14.5). It implies latency too which shall be passed to address. This operation comes with a third operation that works on set of two write addresses from DFT engine shown in Fig. 14.6. The sequence (a(0),a(1)) becomes (a(0),a(0)+N_i) where N_i the power of current radix in the decomposition of the size N of DFT. These two last operations redistribute the data inside the two memories and are necessary when passing from prime factor 4 to another one. Finally, the last operation works on converted addresses but only those attached to the second memory. An offset corresponding to the column number in which the point number N_o (=N/N_i) is placed inside the Compressed Ruritanian map attached to the next prime factor is added on addresses by the adder 14.7. This is required because the point coming simultaneously with point number 0 is point number N_o and must be applied when prime factor 5 belongs to decomposition of N. Finally, the table presented in Fig. 15 resumes the cases in which the operations must be applied.

Before results dumping, because of the radix sequence (4, 5, 3) and because of the particular form of conversion table attached to power of 3, even points are stored linearly inside first memory whereas odd points are stored linearly inside the second one but from point number N_3. The position of point number 1 must be known in order to correct addresses for second memory. Then, points are output alternatively from either first memory or second one. Fig. 16 represents the block diagram for data dumping. Adder 16.1 is used to correct addresses for the second memory. Multiplexer module 16.2 is used to alternate outputs of both memories.

1/ Device to compute different sizes of DFT according to the PFA algorithm with the Ruritanian mapping, characterized in that it comprises :

- two butterfly computation unit to compute all the butterflies of different sizes on the flow ;

- two separate storage means to store data with size equal to half the size of the DFT;

- means to dispatch input data within the two separate storage means according to the even and odd numbered rows of the Ruritanian maps ;

- means to redistribute data each time that the radix changes because of a different Ruritanian map;

- means to retrieve output data from the two separate storage.

2/ Device according to Claim 1, characterized in that each storage means further comprises two different memories being used alternatively for input and output operations when changing of radix size.

3/ Device according to one of claims 1 to 2, characterized in that it further comprises virtualization means to encapsulate the two storage for transparent access.

4/ Device according to claim 3, characterized in that it further comprises address conversion means based on the modification of the Ruritanian map.

5/ Device according to claim 1, characterized in that data redistribution is achieved by means of applying a similar reordering on the output data from each butterfly unit and performing pair-wise exchanges on the two reordered streams.

6/ Method to compute different sizes of DFT according to the PFA algorithm with the Ruritanian mapping, characterized in that it comprises :

- a step to compute all the butterflies of different sizes on the flow on two butterfly computation units ;

- a step to store data with size equal to half the size of the DFT into two separate storage means ;

- a step to dispatch input data within the two separate storage means according to the even and odd numbered rows of the Ruritanian maps ;

- a step to redistribute data each time that the radix changes because of a different Ruritanian map;

- a step to retrieve output data from the two separate storage.

7/ Method according to Claim 6, characterized in that the step to store data stores data alternatively in each storage means within two different memories being used alternatively for input and output operations when changing of radix size.

8/ Method according to one of claims 6 to 7, characterized in that it further comprises a virtualization step to encapsulate the two storage for transparent access.

9/ Method according to claim 8, characterized in that it further comprises address conversion steps based on the modification of the Ruritanian map.

10/ Method according to claim 6, characterized in that data redistribution is achieved by steps of applying a similar reordering on the output data from each butterfly unit and performing pair-wise exchanges on the two reordered streams.

ABSTRACT

To speed up the computation the invention proposes to use two computation cores in parallel to compute a DFT. Data are dispatched between the two cores according to the even and odd lines of the PFA Ruritanian mapping matrix. Separate storage means are used for each core and means are provided to exchange data between the two separate storage means between the radix computation steps.

Representative Drawing

Fig. 14A

$$\begin{pmatrix} t_{0,1} & t_{0,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ t_{1,0} & t_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{2,2} & t_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{3,2} & t_{3,3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{4,4} & t_{4,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{5,4} & t_{5,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{6,6} & t_{6,7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{7,6} & t_{7,7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{8,8} & t_{8,9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{9,8} & t_{9,9} \end{pmatrix} \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} & b_{0,6} & b_{0,7} & b_{0,8} & b_{0,9} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} & b_{1,8} & b_{1,9} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} & b_{2,8} & b_{2,9} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} & b_{3,8} & b_{3,9} \\ b_{4,0} & b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} & b_{4,8} & b_{4,9} \\ b_{5,0} & b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} & b_{5,8} & b_{5,9} \\ b_{6,0} & b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} & b_{6,8} & b_{6,9} \\ b_{7,0} & b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} & b_{7,8} & b_{7,9} \\ b_{8,0} & b_{8,1} & b_{8,2} & b_{8,3} & b_{8,4} & b_{8,5} & b_{8,6} & b_{8,7} & b_{8,8} & b_{8,9} \\ b_{9,0} & b_{9,1} & b_{9,2} & b_{9,3} & b_{9,4} & b_{9,5} & b_{9,6} & b_{9,7} & b_{9,8} & b_{9,9} \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \end{pmatrix}$$

Fig. 2A

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{2,2} & -t_{3,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_{3,2} & t_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{4,4} & -t_{5,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_{5,4} & t_{4,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{6,6} & -t_{7,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & t_{7,6} & t_{6,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{8,8} & -t_{9,8} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_{9,8} & t_{8,8} \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & b_{2,2} & -b_{3,2} & b_{2,4} & -b_{3,4} & b_{2,6} & -b_{3,6} & b_{2,8} & -b_{3,8} \\ 0 & 1 & b_{3,2} & b_{2,2} & b_{3,4} & b_{2,4} & b_{3,6} & b_{2,6} & b_{3,8} & b_{2,8} \\ 1 & 0 & b_{4,2} & -b_{5,2} & b_{4,4} & -b_{5,4} & b_{4,6} & -b_{5,6} & b_{4,6} & -b_{5,8} \\ 0 & 1 & b_{5,2} & b_{4,2} & b_{5,4} & b_{4,4} & b_{5,6} & b_{4,6} & b_{5,8} & b_{4,8} \\ 1 & 0 & b_{6,2} & -b_{7,2} & b_{6,4} & -b_{7,4} & b_{6,6} & -b_{7,6} & b_{6,8} & -b_{7,8} \\ 0 & 1 & b_{7,2} & b_{6,2} & b_{7,4} & b_{6,4} & b_{7,6} & b_{6,6} & b_{7,8} & b_{6,8} \\ 1 & 0 & b_{8,2} & -b_{9,2} & b_{8,4} & -b_{9,4} & b_{8,6} & -b_{9,6} & b_{8,8} & -b_{9,8} \\ 0 & 1 & b_{9,2} & b_{8,2} & b_{9,4} & b_{8,4} & b_{9,6} & b_{8,6} & b_{9,8} & b_{8,8} \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \\ d_8 \\ d_9 \end{pmatrix}$$

Fig. 2B

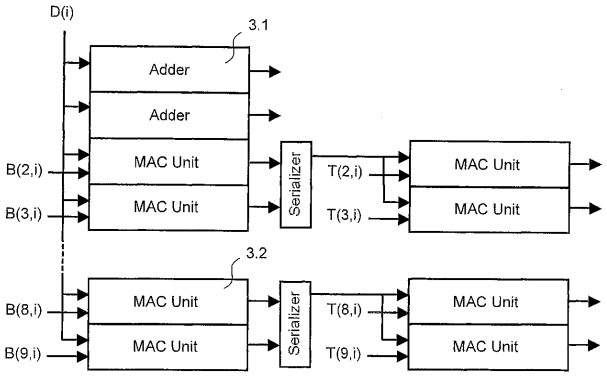


Fig. 3

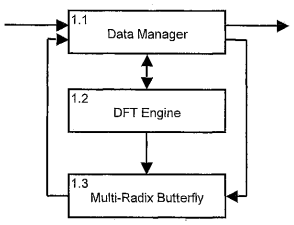


Fig. 1

Stage	Clock	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C			0						0						0					
2	4.1		d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5	
	4.2		0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	0	1	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3			p_0	p_1	p_2	p_3	p_4	p_5	p'_0	p'_1	p'_2	p'_3	p'_4	p'_5	p''_0	p''_1	p''_2	p''_3	p''_4	p''_5
	4.7			0	s_0	s_1	s_2	s_3	s_4	0	s'_0	s'_1	s'_2	s'_3	s'_4	0	s''_0	s''_1	s''_2	s''_3	s''_4
4	4.4				s_0	s_1	s_2	s_3	s_4	x_3	s'_0	s'_1	s'_2	s'_3	s'_4	x'_3	s''_0	s''_1	s''_2	s''_3	s''_4

Fig. 5A

Stage	Clock	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C					d_1						d'_1						d''_1			
2	4.1				d_2	d_3	d_4	d_5			d'_2	d'_3	d'_4	d'_5			d''_2	d''_3	d''_4	d''_5	
	4.2				$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3					p_2	p_3	p_4	p_5			p'_2	p'_3	p'_4	p'_5			p''_2	p''_3	p''_4	p''_5
	4.7					d_1	s_2	s_3	s_4			d'_1	s'_2	s'_3	s'_4			d''_1	s''_2	s''_3	s''_4
4	4.4						s_2	s_3	s_4	x_3			s'_2	s'_3	s'_4	x'_3			s''_2	s''_3	s''_4

Fig. 5B

Stage	Clock	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	D	d_0	d_1	d_2	d_3	d_4	d_5	d'_0	d'_1	d'_2	d'_3	d'_4	d'_5	d''_0	d''_1	d''_2	d''_3	d''_4	d''_5		
	C					d_1						d'_1				0		d''_1			
2	4.1				d_2	d_3	d_4	d_5			d'_2	d'_3	d'_4	d'_5	x_2	x_3	d''_2	d''_3	d''_4	d''_5	
	4.2				$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$			$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	$t_{3,2}$	$t_{2,2}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$	
3	4.3					p_2	p_3	p_4	p_5			p'_2	p'_3	p'_4	p'_5	y_0	y_1	p''_2	p''_3	p''_4	p''_5
	4.7					d_1	s_2	s_3	s_4			d'_1	s'_2	s'_3	s'_4	0	y_0	d''_1	s''_2	s''_3	s''_4
4	4.4						s_2	s_3	s_4	x_3			s'_2	s'_3	s'_4	x'_3	y_0	r_3	s''_2	s''_3	s''_4

Fig. 5C

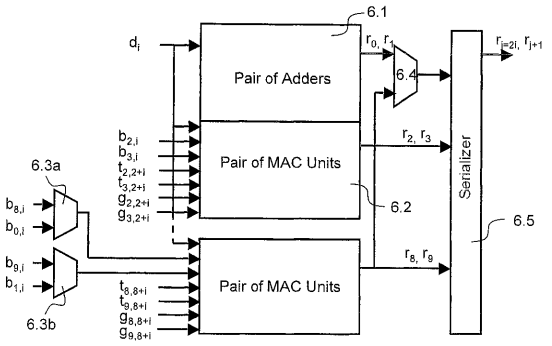


Fig. 6

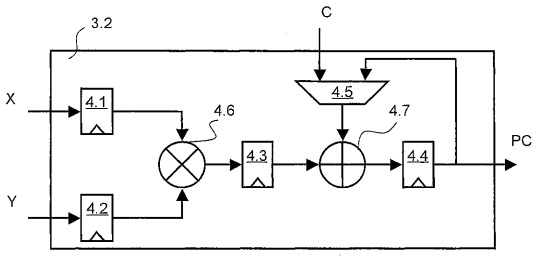


Fig. 4

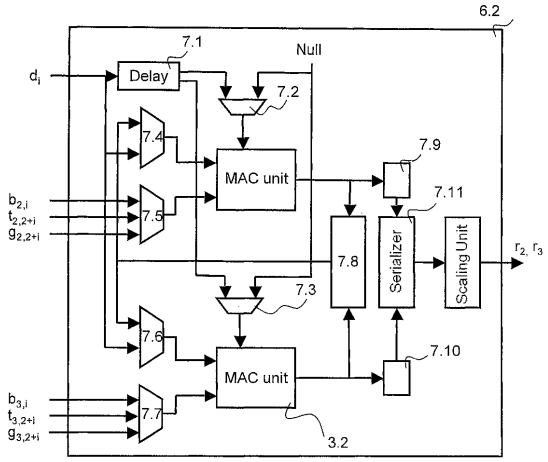


Fig. 7

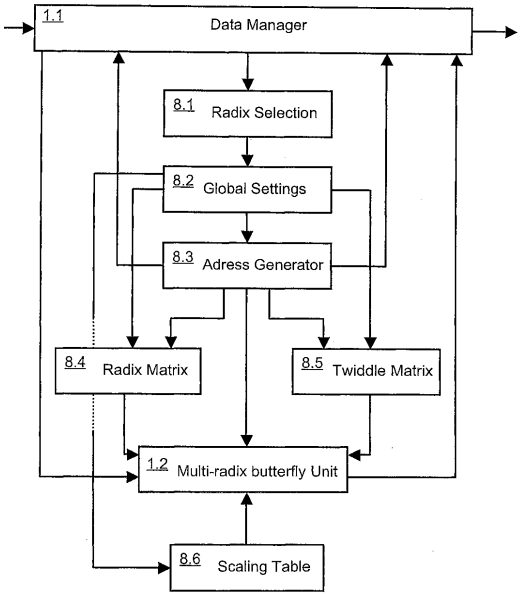


Fig. 8

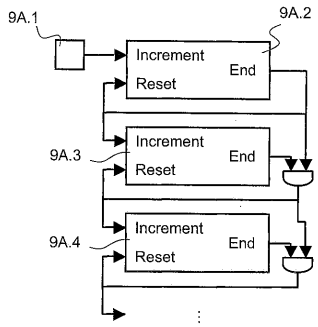


Fig. 9A

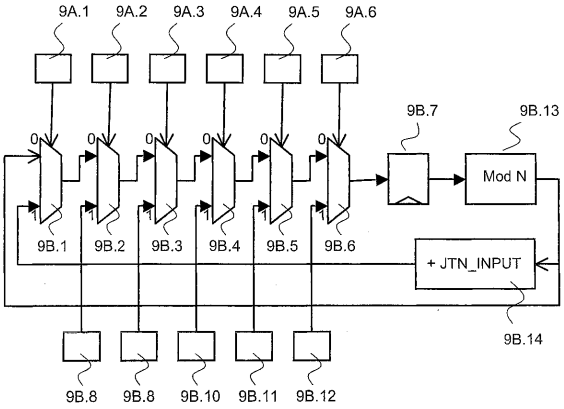


Fig. 9B

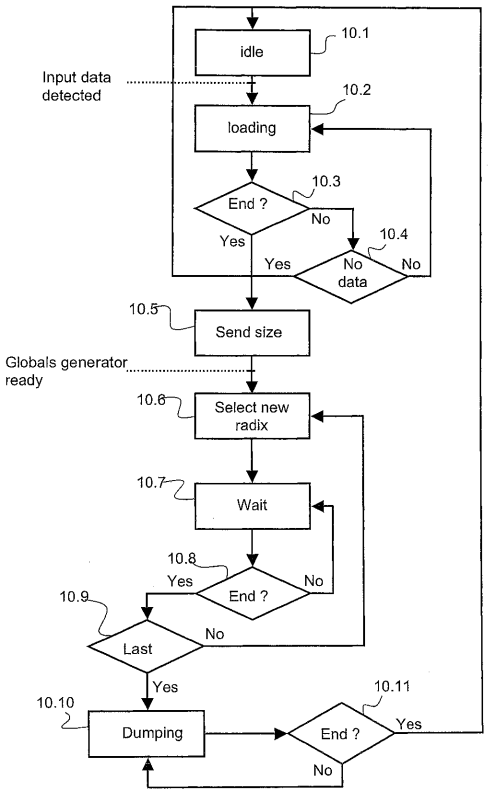


Fig. 10

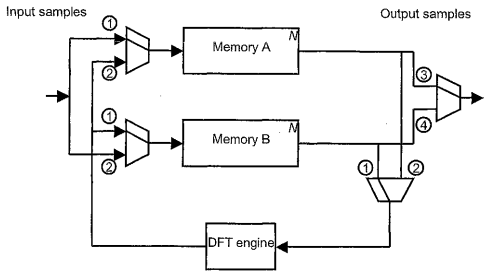


Fig. 11A

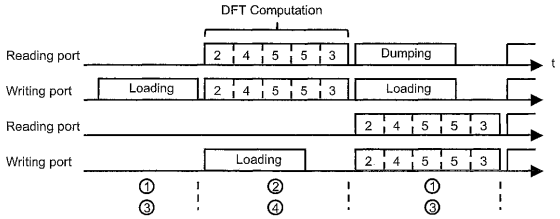


Fig. 11B

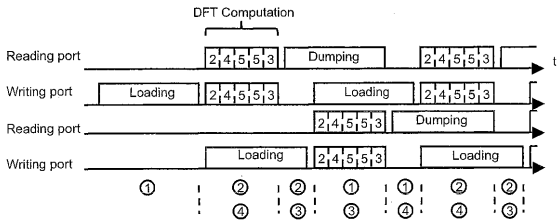


Fig. 11C

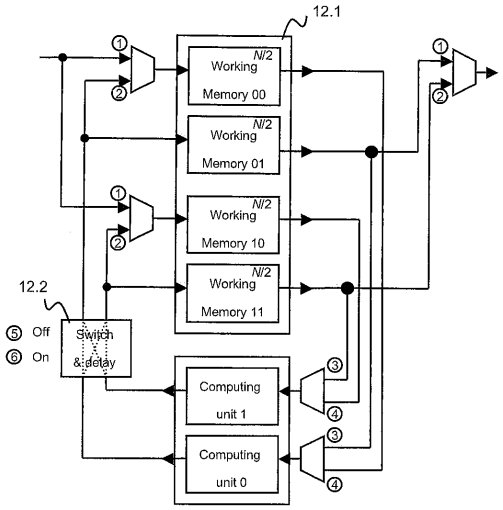


Fig. 12A

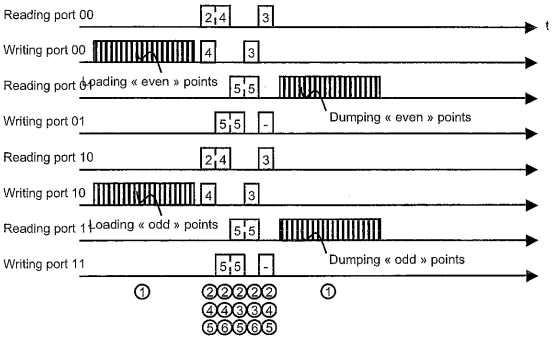


Fig. 12B

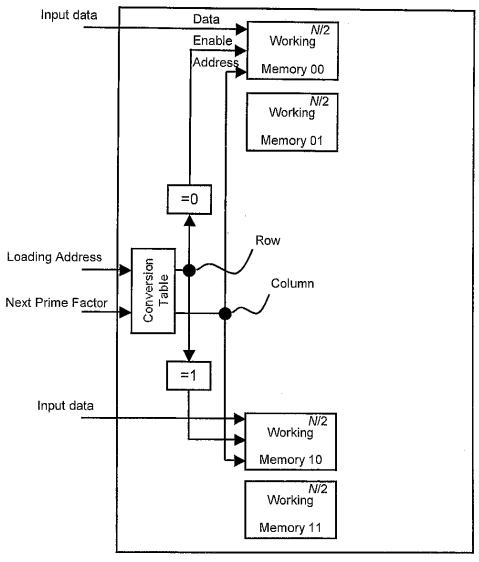


Fig. 13

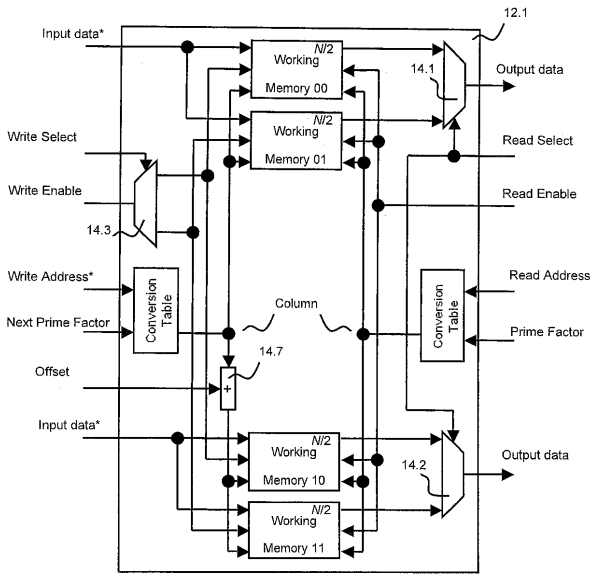


Fig. 14A

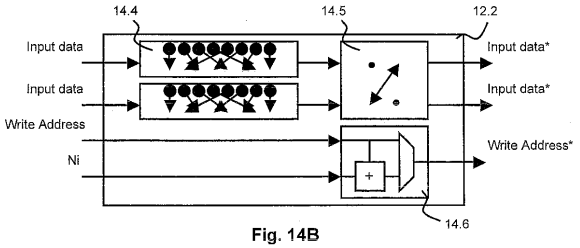


Fig. 14B

radix \ next radix		4/2	5	3
4/2	type 1	no operation	cross quadruplets / modify addresses / add offset	cross quadruplets / modify addresses
	type 2 or radix-8	no operation	reorder data / cross quadruplets / modify addresses / add offset	reorder data / cross quadruplets / modify addresses
5			no operation	add offset
3				no operation

Fig. 15

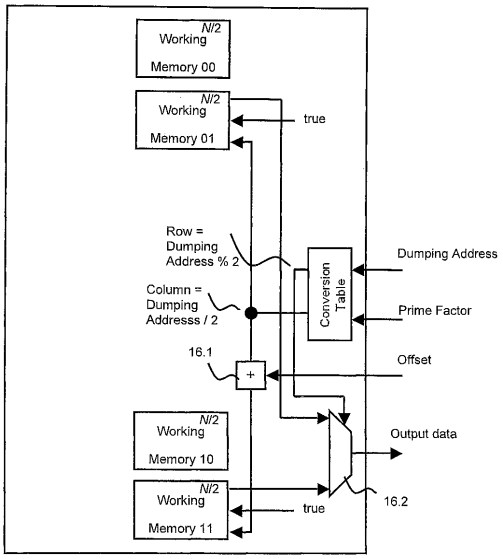


Fig. 16

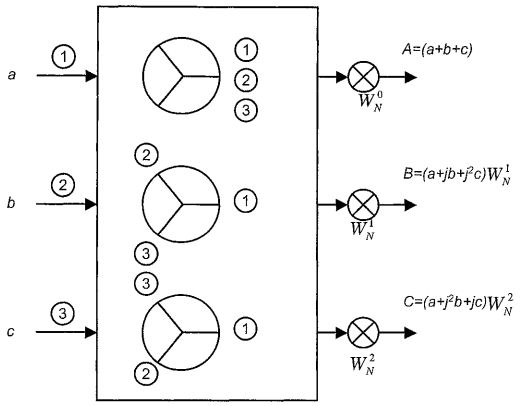


Fig. 17