# 第2章

# C言語から直接生成したステート・マシン記述を用いたFFT回路の設計

## Design Wave設計コンテスト2007 Professional 部門第1位

石井康雄

**Design Wave 設計コンテスト2007 のProfessional部門第1位の設計を紹介する．乗算器の数を減らすために，仕様書で示されたRADIX-4方式ではなく，RADIX-4とRADIX-2を組み合わせたSplit-RADIX方式を採用した．また，加算器に対して規模の大きい乗算器の使用効率を高めるために，スケジューリングを行い，そのステート・マシンを生成するソフトウェアをC言語で開発した．（編集部）**

昨年は学生として参加したこのコンテストですが，今回は社会人として参加することになりました．目標は昨年争った学生達と沖縄で再会することです[注1]．

さて，今回の課題はFFTです．私は組み込み技術者というよりは計算機技術者なので，組み込み機器におけるFFTの利用よりはFFTE[(1)]などをはじめとした科学技術計算の話の方が身近に感じられます．そこで，低コストFPGA上への実装をプロトタイプとして最終的に高性能なFPGAやASIC上で実装し，科学技術計算に利用できる高性能FFTアクセラレータを作成するといったストーリを作れないものかと考えて設計を開始しました．

科学技術計算におけるアクセラレータというアプローチが近年注目を浴びています．GRAPEプロジェクト[(2)]のように完全な専用計算機もあれば，PlayStation 3に採用されているCELLプロセッサ[(3)]のSPE（Synergistic Processor Element）のように汎用的なSIMD（single instraction stream-multiple data stream）エンジンもあります．特に，GRAPE-7はFPGAで実装されていますから，FPGAによる科学技術計算は今日では選択肢の一つとして考えられていると言ってよいと思います[注2]．

数値的な目標はさておいて，当然，科学技術計算をさせるのだから，64ビットの浮動小数点演算などが必要になるだろう，などと考えながら，設計の方針を決めていきました．

## 1．設計の方針と目標

科学技術計算に使えるアクセラレータなどと大それた目標を立てたものの，実際に作るのはプロトタイプもどきです．低コストFPGAにFFT回路を実装して具合をみることにしました．

プロトタイプからの将来的な拡張もにらんで設計する必要があります．そこで今回は，設計の目標を以下のように定めました．

- 回路の小規模化
- 高性能化（高効率化）
- 可搬性の高さ

### ● 乗算器を削減して回路規模を抑える

回路の小規模化は，コスト削減につながります．回路が小規模であれば，ダイが小さくなるので，コスト削減には直接的な効果があります．また，一般に半導体製造においては，回路規模が下がるほど歩留まりが上がるので，回路規模は小さいに越したことはありません．回路の小規模化

注1：昨年に引き続き参加している学生がおり，非常に楽しいコンテストだった．
注2：2007年3月22日にMaxwell[(4)]というFPGAベースのスーパコンピュータが発表された．

KeyWord Split-RADIX，FFT，FPGA，バタフライ演算，複素数，三番地文，DAG，スクラッチパッド・メモリ

は必須の目標と考えました．

具体的には，乗算数の削減という最適化をしました．一般に，固定小数点や浮動小数点の演算器では，乗算器は加減算器と比較して多くの資源を必要とする傾向にあります[注3]．このため，アルゴリズムの変更により乗算の回数を減らすことで，必要とする乗算器を削減することは，回路規模や消費電力の削減に有効です．しかも，将来的にビット幅を増やす可能性が高い場合に，乗算器がビット幅の拡張により最も資源量が増えるものであるという事実も見逃せません．

### ● 低コストFPGA向けを想定して高い効率を目指す

また，最終的にアクセラレータとして動作してほしいので，ある程度の演算速度，演算効率を実現することを目標とします．専用アクセラレータは汎用計算機で出せない性能[注4]を出すことが最低条件なので，これも当然の目標です．

とはいえ，低コストFPGAで高性能・高効率を両方達成するのは困難ですから，高性能はひとまず据え置いて高効率を目指すことにします．加算器と乗算器のスループットが等しい汎用プロセッサの環境では，演算器の利用効率[注5]が62.5％を超えることがないので，63％以上の演算器の利用率を目標としました．

また，演算性能はFPGAの性能も考慮し，演算回路の内部クロック100MHzを目標としました．低コストFPGAでこの程度動くなら，高価格帯で150MHz～300MHz程度，ASIC[注6]にすれば300MHz～500MHz程度の動作周波数が得られるでしょう．また，演算器や内部メモリも大量に搭載できますから，専用計算機として実現する価値のあるものができそうです．

### ● 仕様変更を考慮して可搬性を高める

拡張性の高さも追及しなくてはなりません．今回の設計はプロトタイプという位置づけであれば，いったん実用化した後にさらなる拡張の要求が必ずあるわけです．この要求としては，例えばデータ幅の増大（固定小数点数から浮動小数点数の利用への変更），FFTに利用する点数の増大（64点から4096点など）などが挙げられます．このような要求に対して最小のコスト（VHDLファイルの一部の定数値の変更など）で対処できるようにデータ・パスを設計したいと考えました．

### ● コンテスト向けにおもしろく

これらの目標を達成するために，アルゴリズムとデータ・パスの実装方法を再検討して設計を行うことにしました．また，コンテストという特性もあるので，トレードオフで迷うことがあった場合には，おもしろい方を選ぶということも設計方針に付け加えています．

## 2．アルゴリズムの検討

ここでは，コンテストで採用したアルゴリズムと，その背景に関して説明します．

まず，最適化は大きく二つに分かれています．一方が複素数乗算のアルゴリズムの最適化で，もう一方がFFTのバタフライ演算のアルゴリズムの最適化です．それぞれに対して注目し，可能な限り乗算回数を減らすように最適化を行いました．

### ● 複素数計算のアルゴリズム

当たり前の話なのですが，FFT中に発生する乗算は，基本的にはすべて回転因子との複素数乗算です[注7]．これを踏まえて最適化を行っていきましょう．

一般に複素数（$x_r$, $x_i$）と回転因子（$w_r$, $w_i$）の乗算結果（$y_r$, $y_i$）は，以下のように計算されます．

$$y_r = x_r \cdot w_r - x_i \cdot w_i$$
$$y_i = x_r \cdot w_i + x_i \cdot w_r$$

これは皆さんご存知の数学の教科書に載っている方法です．この計算では，一つの複素数乗算に対して4回の実数乗算が必要です．しかし，この計算の形式を変更すると，乗算の回数を3回に減らすことができます[注8]．

$$tmp = (x_r + x_i) \cdot w_r$$
$$y_r = tmp - x_i \cdot (w_r + w_i)$$
$$y_i = tmp - x_r \cdot (w_r - w_i)$$

注3：$N \times N$乗算器の乗算器は$O(N^2)$以上の資源を必要とする．しかし，$N+N$の加算器は$O(N)$～$O(N \log N)$程度で実現されることが多い．
注4：ここでの性能とは消費電力あたりの演算性能などであってもよい．
注5：このレポート中では演算全体にかかった時間（サイクル数）で演算器が実際に演算した時間を除したもので定義する．
注6：ほんとうに作るとなれば，米国Altera社のHardCopyのようにFPGAからASICへの設計パスが存在するので，それを利用するのがよいだろう．
注7：RADIX-8の場合にはスカラ値との積も含まれる．
注8：正確には2回の加算と4回の乗算を3回の加算と3回の乗算に置き換えることになる．