第2章

演算回路設計のセンスをつかもう

――演算をハードウェア化する際にどんなことを考えるか

森岡澄夫

仕様で求められる演算処理の内容を理解できるようになったら，今度は実際に回路に落とし込む作業に入ります．本稿では，回路化にあたって使われる一般的な考え方や設計のコツを説明します．ほとんどの実用回路は，それらのいずれか，もしくは複数を組み合わせることで作られています．（著者）

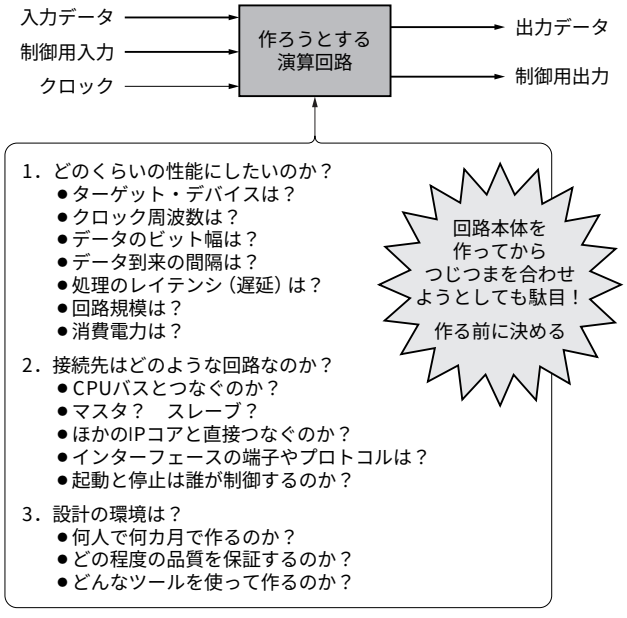

図1　演算回路の本体を作り出す前に大ざっぱでも決めるべきこと

演算回路本体をさっさと作って安心したくなるものだが，その気持ちをグッと抑えて，まずは目標性能や接続先との交信方法について決めることが大事．もちろん，システム開発初期でこれらを確定的には決められなかったり，開発中に変更が入ったりすることも多いが，何も決めないよりは「後から回路を全面作り直し」のリスクをはるかに低減できる．

① まずはシステム全体のことを考える

意外に思われるかもしれませんが，演算回路だからといって数値計算本体から設計に入っていくわけではありません．どちらかといえばそれは後回しです．

● 回路内部よりも先に目標性能やI/F仕様を考える

回り道のようでも，システム全体の中で回路がどう使われるかを考え，回路の目標性能や入出力インターフェースをはっきりさせるところから入ります[15),16)]（**図1**，**図2**）．これは，ソフトウェアを複数人で分担して作成するような場合に，例えば関数の中身をいきなり作り始めたりせず，

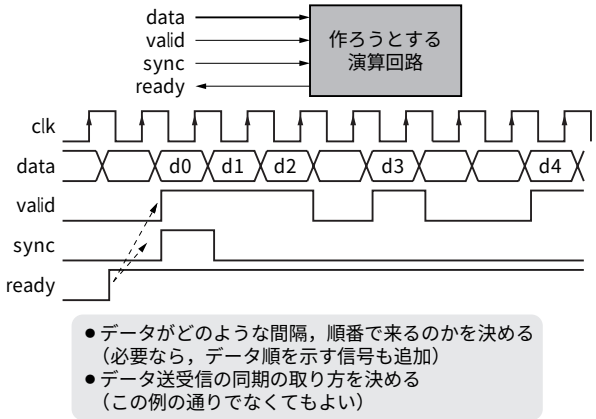

図2　データ転送プロトコル（タイミング・チャート）を決めた例

プロトコル設計というと難しそうだが，要はデータをやりとりする端子とタイミング・チャートを決めるということ．必ずしもこの図通りとは限らない．

KeyWord　64点高速フーリエ変換，FFT，テイラー展開，オンザフライ計算，離散コサイン変換，パイプライン，インターリーブ，抽象度

入出力仕様をまず決めるのと全く同じことです．

すなわち，

- どのくらいの性能の回路にしたいのか．加えて，使えるクロックの周波数や本数はどの程度で，どれくらいの回路規模なら許されるのか，どのようなデバイスを使うのか（FPGAかASICか．ASICなら何nmの製造プロセスか）
- 回路に対するデータの入出力（タイミング・チャートや，入出力のビット幅など）をどのようにするか
- どのくらいの設計期間なのか

などについて，システム設計者や隣接モジュールの担当者とよく打ち合わせます．これらの事項が最初から明確に分からない場合や，設計途中で変更が入るような場合もありますが，たとえ大ざっぱでも決めることが重要です．このことが演算回路本体をどのような方向性で作るかに，非常に大きく影響するからです．

また，何も考えずに演算回路本体だけ先に作ると，周辺回路とうまくつながらない，性能を全く発揮できないといった深刻な問題が後で発生しやすくなります．例えば，1クロックに1個，必ず入力データが来るという想定で必要性能が得られるよう演算アルゴリズムなどを選定したのに，実システムでデータ・レートが大幅に低いことが後で発覚したりすると，アルゴリズム変更が必要になって設計が完全にやり直しになってしまうかもしれません．

● 設計コンテストの着眼点

ところで，「Design Wave設計コンテスト2007」の課題は64点高速フーリエ変換（FFT）回路ですが[3]，この場合も単に汎用FFT回路の最適化ばかりに目を向けては面白くありません．FFTがどのような用途で使われており，その用途ではどれくらいの演算精度や回路性能が必要なのかを調べてみましょう．何か面白い特定用途向けに，ユニークな計算アルゴリズムや回路構成を考えてみると，際だった特色や高い実用性を示すことができるに違いありません．必ずしも型通りにバタフライ演算を行わなくても，本稿で説明するような色々なアイディアを適用する余地が出てくるからです．

② 数式を“そのまま”実装しなくてもよい

目標性能やインターフェースのタイミングが見えてきたら，仕様の計算内容（数式）をどのようなアルゴリズムを使って作るかを考えます．このときの基本的な発想の仕方は，誤解を恐れずに言えば，

- 数式をそのまま正直に計算しなくてもよい
- 数学的に複雑な計算を組むのは，なるべく避ける（バグのもと）
- 精度保証が必要だったり例外処理の多い計算方法は，なるべく避ける（テストが大変）

など，「設計の労力や回路コストをいかに少なく済ませるか」ということです．

例えば，FFT回路の数式，

$$X(k)=\sum_{n=0}^{N-1}x(n)\cdot e^{-j(2\pi/N)nk} \quad \cdots\cdots(1)$$

をもし素直に計算するならば，まず$(2\pi/N)nk$を計算し，次に$e^{-j(2\pi/N)nk}$を計算し，それを繰り返すという手順になりそうです．また，2次元離散コサイン変換（DCT）の計算式，

$$F(u,v)=\left(\frac{2}{N}\right)^{\frac{1}{2}}\left(\frac{2}{M}\right)^{\frac{1}{2}}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\Lambda(i)\Lambda(j)\cdot\cos\frac{\pi u(2i+1)}{2N}\cdot\cos\frac{\pi v(2j+1)}{2M}\cdot f(i,j) \quad \cdots\cdots(2)$$

ただし，$\Lambda(\xi)$は$\xi=0$のとき$1/\sqrt{2}$，それ以外のとき1

であれば，cos関数を例えばテイラー展開などを使って計算していくのかもしれませんし，開平計算も要りそうです．

しかし通常は，以下に述べるような方法のもとで，もっと簡単かつ確実に回路が作れないか，もっと回路性能が得られないかという点について検討を行います[注1]．

● オンザフライ計算と事前計算を使い分ける

回路実装に限らずソフトウェア実装でも頻繁に使われる方法ですが，難しい演算についてはオンザフライ計算（回路動作時にリアルタイムで計算すること）を行わず，設計時に事前計算した結果をテーブルで持っておくという手法があります（**図3**）．例えば離散コサイン変換のcos関数の値など，テーブルで持つことにするだけで設計はグッと楽になります．別途テーブル値を作成・検証する手間はかかりますが，込み入った回路を作るよりは楽です．

テーブルは，HDLのcase文で書いて，論理合成で組み

注1：一般的な傾向として，算術演算関数を数学の教科書にある定義・定理通りに計算すると，回路が著しく複雑になる，性能が低下する，精度保証が難しくなる，といった問題が起こりがち．数学の教科書ではそういった工学上のコスト概念をあまり扱わないのが，一つの理由．