第3章

ストリーム・データ処理のハードウェア化

JPEGデコーダの設計事例

森岡澄夫

ここでは，JPEGデコーダの主要部を例題として，ソフトウェア処理を土台にハードウェアを作成していく過程を示す．JPEGを例題に選んだ理由は，音声・画像処理はもちろん，通常の回路設計でも課題となる事項が，ひと通り出てくるからである．本稿で述べる内容は，ざっと見るだけでは当たり前に思えるかもしれない．しかし慎重に見ると，単純で機械的な最適化は少なく，設計の意味内容を把握してうまい最適化のかけ方を考える「人知」が要ることが分かるはずである．現実の設計では，一見当たり前の工夫を抜かりなく適切にやるのが，できそうでいて案外できないことなのだ．（筆者）

1．ソフトとハードの違いが現れる回路

ここではJPEGデコーダの主要部の設計を例題として取り上げます．ただし，JPEG回路そのものの説明が目的というわけではありません．JPEGは一言でいえば「ハードらしいハード」です．つまり内部処理のさせ方や設計作業で考える事が，ほかの多くのハードウェアと共通しています．それがここで取り上げる理由です．特に，ソフトウェア記述の1行1行を1対1でハーウェアに焼き直してしまうと性能がまったく出ないので，ソフトウェア記述を意訳していく，という点で非常に良い典型例です．ソフトウェアとハードウェアの違いが実に顕著に現れます．

図1に，JPEG回路の全体構成や，本稿で説明する事項をまとめて示します．詳しい説明に入る前に，以下に概要を述べます．

● 回路をつなげて作るストリーム処理の典型例

JPEG処理の大きな特徴は，ストリーム・データを処理することです．すなわち，「処理するデータ全体を回路中に全部ため込み」→「処理をして」→「結果を全部まとめて出力する」というまとめ処理ではなく，データを連続的に受け取りながら処理してどんどん出力していく，という方式を取ります．

表1に示す通り，多くの種類の機能ブロック（IPコア）がそのようなストリーム処理方式を取ります．また，IPコアのみならず，システムLSIの全体設計では，ほとんどの場合にストリームの考え方が必要となります（IPコアやプロ

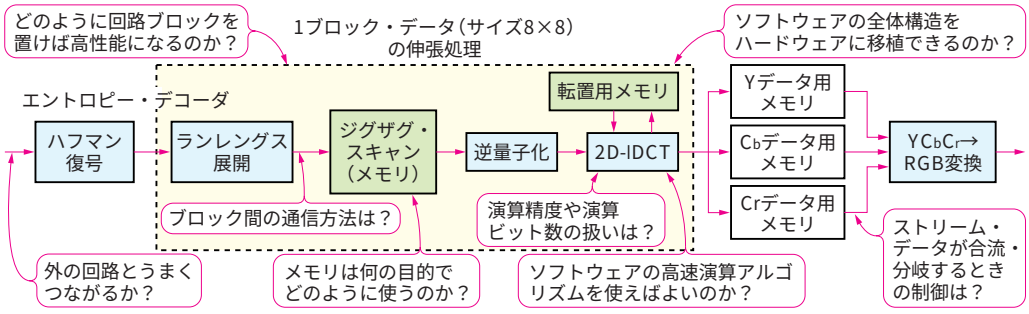

図1　JPEGデコーダには一般の回路設計で扱われる検討課題の大半が含まれる

破線で囲んだ黄色の部分が今回の説明で扱う範囲である．図で示したほかに，①この回路のテストをどのように行うのか，②ASIC（チップ）にしたら回路変更がきかないがどのようにリスクを回避するのか，といった重要な検討課題がある．

Keyword　ストリーム・データ処理，JPEGデコーダ，ジグザグ変換，量子化係数テーブル，DCT，IDCT，交信プロトコル，バースト転送，演算精度，小数演算，動作合成

セッサの間でどのようにストリーム・データをスムーズに流していくかを考える).

ストリーム処理方式をとる主な理由は次の通りです.

- パイプライン処理を可能にする. まとめ処理方式ではパイプライン処理を行いにくく, スループット(単位時間当たりのデータ処理量)やレイテンシ(一つのデータを入れてから出てくるまでの時間)が, 数倍～数十倍のオーダで悪くなる.
- 大量のデータを扱えるようにする. まとめ処理方式では, 回路中に一度にため込めるデータ量が, そのまま処理可能なデータの最大量となってしまう. しかし, 第2章でも述べたとおり, 回路では記憶素子(メモリやフリップフロップ)のコストが非常に高い.

パソコン用のソフトウェアにおいても, 例えば数十Gバイトもある動画ファイルを扱う際に, すべてのデータをメモリに読み込んでから処理するようなことはなく, 少しずつ読み込んで処理します. それと同じことです. ハードウェアの場合には, 一度に持てるデータ量がパソコンよりも数けたのオーダで小さくなるのが違いです.

● 全体構成でほぼ回路性能が決まってしまう

ストリーム・データをパイプライン的に流す場合, データ処理のレイテンシよりはスループットの方が, 優先度の高い性能指標となります. 分かりやすくいうと, ある1枚の絵を投入してから結果が出るまでの時間よりは, 1s(秒)間に何枚の絵を投入できるかの方が重要な指標です.

表1　JPEGデコーダと似た「ストリーム処理」の考え方で設計する回路

回　路	(補足)
大半の回路(IPコアないしシステム)のトップ階層	IPコア内部であれシステムLSI全体であれ, 多かれ少なかれ「データ・フローを考えて回路モジュールを結合する」作業は必須となる.
画像・音声処理のうち近傍データ処理	フィルタ, 相関計算, データ領域変換, 画像拡大・縮小など, 処理中の1点(画素)かその近傍だけに着目すれば計算ができるものは, ストリーム処理にする.
データ圧縮伸張	Loss-less, Lossyのいずれであっても, ストリーム処理をしないと大量データが扱えなくなる.
ブロック誤り訂正符号	データ・ブロックをストリーム的に流せるようにしないと大量データが扱えなくなる. 特にデコーダは, 内部をパイプライン化しないと非常に遅くなる.
共通鍵ブロック暗号, ハッシュ関数, セキュリティ・プロセッサ	共通鍵やハッシュでは大量のデータを扱うケースが多く, その入力はストリームであるとを仮定する. 暗号演算を組み合わせたセキュリティ・プロセッサも, 全体としてはストリーム処理を行うことになる.

そのような場合に最も重要なのは, 個々のモジュールの設計に入る前に全体的なデータの流れをよく考えることです. ソフトウェアと異なり, 各モジュールを個別に速くしてもスループットが上がるわけではありません. 全体設計の良し悪しでほとんどスループットが決定してしまいます.

例えば図1にはいくつかの処理ブロックがありますが, ストリームをパイプラインで扱う場合, 全体性能は最も性能の低いモジュールで抑えられてしまいます. 全体を考えずに一部の処理だけむやみに速くしても意味がありません.

このため, 各モジュールの配置・個数・目標性能や, モジュール間の交信プロトコルをトップダウンで決めたうえ, 設計する必要があります.

● メモリを使わずスムーズにデータを流す

全体構成検討において大きなウェイトを占めるのが, 回路の中へのメモリの置き方です. メモリを使う主目的はデータの流れ方の制御ですが(第2章を参照), まずい設計をすると多数の大容量メモリ・ブロックが必要になります. 繰り返しですが, 記憶素子のコストはパソコン用のソフトウェアよりけた違いに高くなります.

今回のJPEGに限れば, それほどの大容量メモリは必要ありません., メモリの配置に若干の無駄があったとしても許容範囲に収まる可能性はあります. しかし一般の画像処理では, メモリの配置はLSIどころか機器設計に大きく影響を及ぼします.

● ソフトとハードで「速い」アルゴリズムが異なる

JPEGには離散コサイン変換など数値演算処理があります. そのようなアルゴリズム絡みの回路設計で最も間違えやすいのが, 「速い(計算量の少ない)ソフトウェア・アルゴリズムを使えば, ハードウェアでも速くなるだろう」と考えてしまうことです.

しかし, 後で例を用いて説明しますが, ソフトウェアとハードウェアでは「速い」ことの意味合いが異なっており, 計算量とスループットは直接には関係しません. ハードウェアにしたときに速くなりやすいアルゴリズムは, データの流れを止めず, なおかつ演算の並列性や規則性も高い, というようなものです.

● 演算ビット数(精度)の決定もクリティカル

ハードウェアで数値演算を行う場合, とりうる値の範囲