第6章　ソフトウェア記述のままでは性能が上がらない！

Cプログラムの高速ハードウェア化テクニック

横山 雅一 Masakazu Yokoyama

画像処理アプリケーションを例に，Cプログラムから高速なハードウェアを合成するためのテクニックを紹介します．通常のアプローチではFPGAで構築することが難しいラベリング処理を扱います．SDSoCであれば比較的簡単にハードウェア化ができることを示し，その利点と問題点を考えていきます．

1．サンプルはラベリング処理

● 画像処理はSDSoC向き

ラベリング処理とは画像を2値化した上で連結している部分を抜き出す処理です．本誌のロゴにラベリング処理を施した例を**図1**に示します．通常，結果は数字になりますが，図では分かりやすいように彩色してあります．文字「F」と「P」はそれぞれ分離・独立した領域で，それ以外の「G」と「A」と「マガジン」の文字は連結しています．

Cの関数としてラベリングを実装してみた例（抜粋）を**リスト1**に示します．

● Cプログラムのハードウェア化では修正しなければならない点が多い

このプログラムは，Cのプログラムとしては普通に使うコーディング技法を幾つも使っています．ハードウェア化に当たっては，修正しなければならない点が幾つかあります．

- 入力にstdinを使用している
- 引き数がポインタである
- 2次元配列を使用している
- whileでループを使用している
- FPGAとしては巨大な16ビットのテーブルを必要とする
- FPGAとしては巨大なVRAMバッファを必要とする

これらを1つ1つ検証してSDSoCとして動作する関数に書き換えます．

2．Cプログラムの修正ポイント

● 修正点1…標準入出力（stdin/stdout）

UNIXでは汎用性を高めるために，単機能のプログラムを複数作り，シェル・プログラムで連結する方法が採られることがあります．画像処理に標準入出力を用いる方法が適切かどうかは議論のあるところではありますが，インターフェースという意味で汎用性が高まります．

元のプログラムは，標準入力から画像データを取得していました．高位合成では直接標準入出力を使うことはできないため，インターフェースを改める必要があります．

ここでは入力にuint8_tの配列を使用します．これがFIFO（First-in First-out）のインターフェースであることを，pragmaを使い定義します（**リスト2**）．

FIFOを使う利点は，メモリの使用効率と処理の並列化にあります．シリアル・アクセスがあらかじめ分かっているにもかかわらず，大きな配列をそのまま引き数に渡そうとすると，メモリ・ブロック（BRAM：Block RAM）に乗り切らなくてコンパイルできません（**図2**）．また，仮に載るサイズに収めたとしても，引き数で渡す配列をメモリ・ブロックにコピーするようなハードウェアだと，配列のコピーが完了するまで処理が始まらず，ハードウェア化の恩恵の1つである並列化をうまく活用できません．

オリジナル（カラー）　グレー・スケール処理　2値化処理　ラベリング処理

図1　ラベリング処理の例
文字「F」と「P」はそれぞれ独立している．「G」と「A」と「マガジン」の文字は境界が狭いので連結している