第4章

ソフトのハード化でボトルネックを解消する

-- Nios II C-to-Hardware アクセラレーション・コンパイラを使いこなす

猪狩貴寛

ここでは，米国 Altera 社の「Nios II C-to-Hardware Acceleration Compiler」を取り上げる．これは，ANSI C ソフトウェア・コードから，ハード・ワイヤード論理のアクセラレータ回路を生成する合成ツールである．C 言語のソフトウェア・コードのうち，ユーザが指定したファンクション（サブルーチン）を RTL（Register Transfer Level）の回路（VHDL または Verilog HDL）に変換する．C ソース・コードを書き換えなくても使えるが，より高い性能を得るためにはガイドラインで示された推奨コーディング・スタイルに従う必要がある．そこで本稿では，C ソース・コードの記述の違いによるアクセラレーション効果の差について，例を挙げながら説明する．（編集部）

今日の製品開発においては，早期の市場投入を実現するために，開発期間の短縮が求められます．しかも競争力を高めるためには，性能の向上は欠かせません．このため，システム設計においては，ソフトウェアとハードウェアの切り分けが悩みどころになっています．一般には，ボトルネックとなる処理をハードウェア化し，残りをソフトウェア処理にします．

本稿では，ソフト・マクロの CPU を実装する FPGA の設計を想定し，ソフトウェア処理部のボトルネックの解消方法の一例を紹介します．使用する CPU コアは，米国 Altera 社の「Nios II」です．また，Nios II の開発ツールとして提供されている「Nios II C-to-Hardware（C2H）Acceleration Compiler」を活用します．

Nios II C2H Acceleration Compiler（以降「C2H」と呼ぶ）は，C 関数をハードウェア・アクセラレータに変換する合成ツールです．ソフトウェア処理をハードウェア化するので，動作周波数を上げずに性能を向上することが期待できます．Nios II の開発環境（Nios II IDE，Quartus II，MegaCore）があれば使用できます．

C 関数のハードウェア化のための操作は，基本的にはマウスで右クリックを行うだけです．C 関数をハードウェアに変換し，Avalon（Altera 社の独自オンチップ・バス）に統合します．合成したハードウェアにアクセスする関数（ラッパ部）の生成やソフトウェアのビルドを行います．ただし，C2H によるアクセラレーション効果は，C ソース・コードの記述のしかたによって異なります．

そこで本稿では，C ソース・コードの記述のしかたと C2H によるアクセラレーション効果について，実際のコードを用いて説明します．C ソース・コードとしては，sobel フィルタと呼ばれる画像の水平微分（エッジ検出）を行うプログラムを用います．Nios II の開発環境は ver.7.0 を使用します．

1．ボトルネックの抽出を行う

ソフトウェア処理におけるボトルネックの抽出は，ソフトウェア開発ツールの持つ機能で行えます．例えば GNU コンパイラ（gcc）を使用する場合は，オプション -pg を付けてコンパイルし，実行するだけです．生成されたモニタリング・ファイル（gmon.out）には，関数などの実行時間が示されているので，これで確認します．

Nios II のソフトウェア開発用に用意されているコンパイ

KeyWord FPGA，Nios II，C2H，ANSI C，ソフト・マクロ，CPU コア，Sobel フィルタ，エッジ検出

ラは，GNUコンパイラをベースとしています．従って，Nios II IDE上から，オプションの指定を行えばよいことになります．具体的には，Nios II IDE上からSystem Library（**図1**）を開き，「Link with profiling library」をチェックします．これにより，gccの-pgオプションが有効になります．

この後，Build（makeコマンド相当）を行い，実行すると**図2**のようにgmon.outが得られます．gmon.outのうち，time欄は，実行時の要した時間を割合（%）で示したものです．関数などに分けて示されます．

図2の例では，`special_filter`と`alt_irq_register`の二つの関数で全体の処理時間のほとんどを要していることが分かります．今回は`special_filter`をハードウェア化対象関数として扱います．

2. C2Hの制約とチューニングのポイント

C2HはANSI Cをサポートしています．ただし，ハードウェア化ツールのため，`printf`をはじめとする標準関数はサポートしていません．本稿執筆時の制約事項は以下の通りです．

- `printf`，`memcpy`などの標準関数
- ハードウェア化に適さない関数の再起呼び出し
- 浮動小数点演算（将来サポート予定，ペリフェラルとして実装することで可能）
- 関数からのサブ関数呼び出し

C2Hを効果的に活用するためには，以下で説明する各項目に対応する必要があります．実際の設計（チューニング）では，要求性能を満たす段階まで対応すればよく，すべての項目を実行する必要はないでしょう．

● ソース・コードの分離

デバッグ効率を上げるために，ソース・コードの分離を行います．

コンパイラは，ソース・ファイル単位でコンパイルを行います．そこで，ハードウェア化対象関数を別ファイルに分割しておきます．こうすることで，期待通りに動作しない場合に，問題がハードウェア側にあるのか，ソフトウェア側にあるのかの切り分けをしやすくなります．

また，C2Hにより合成されたハードウェアの信頼性を上げるため，ハードウェア化対象関数では，ソフトウェアの段階でワーニングをなくしておきます．

今回の例では，ハードウェア化対象関数を`target`

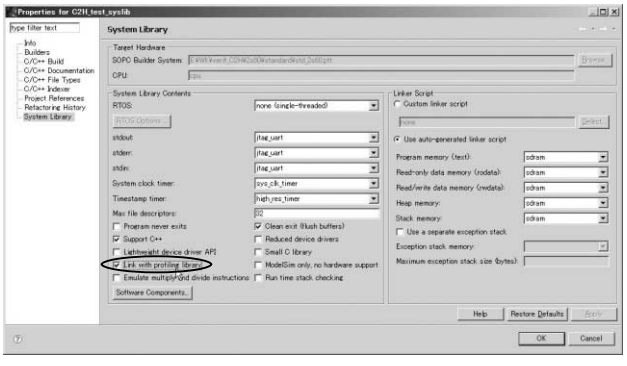

図1　System Library画面
「Link with profiling library」をチェックするとgccの-pgオプションが有効になる．

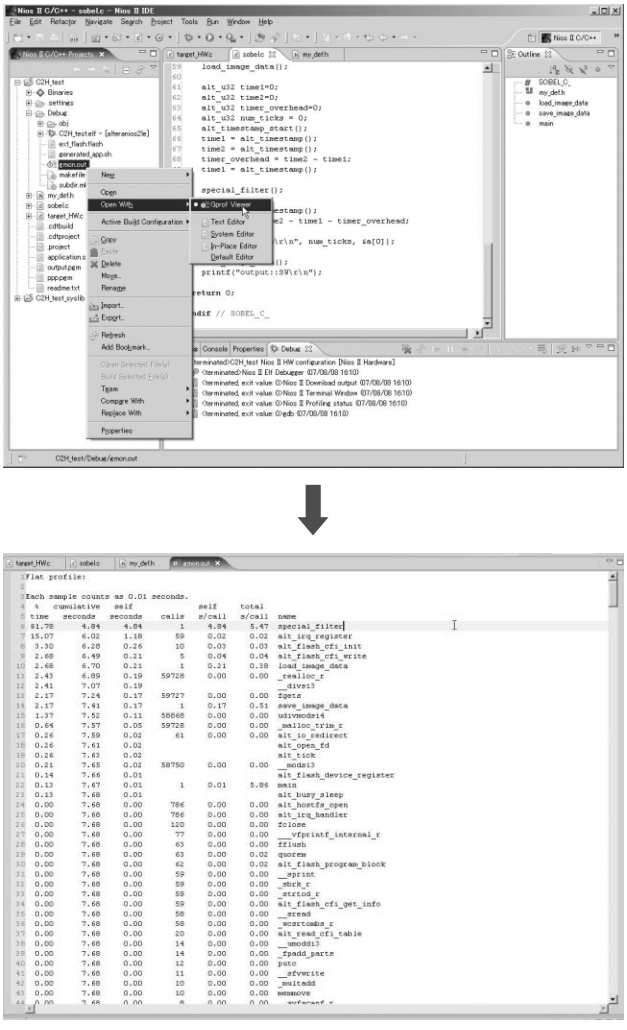

図2　プロファイラ（gmon.out）
モニタリング・ファイルgmon.outは，main関数からreturnされるときに生成される．