第1章

LSIの開発にC/C++を使う理由と設計スタイル

ソフトウェアを出発点にしたIPコア開発のポイント

森岡澄夫

本特集では，システムLSI中に組み込まれるデータ処理回路[注1]（例えば画像処理やセキュリティ処理など）を対象とし，「ソフトウェアのハードウェア化」について解説する．IP（Intellectual Property）コアの設計が主な対象となるが，その範ちゅうに留まらず，システムLSI設計についても扱う．この章では，ソフトウェアのハードウェア化というコンセプトがLSI設計において重要になってきた背景について整理する．（筆者）

C/C＋＋言語などで書かれたソフトウェア処理をハードウェア化する技術に，かつてないほど強い関心が集まっています．製品に搭載するLSIの処理が高度化，専門化してくるとともに，EDA（Electronic Design Automation）ツールも進歩して機が熟してきたからです．

しかし，「ソフトウェアのハードウェア化」は，ソフトウェアのコードを動作合成ツールにかけるだけで済むものではありません．ソフトウェアとハードウェアの間には，設計言語といった表面的な違いを越える本質的な組み方の違いがあるからです．また，LSI作成の目的が製品によって異なる中で，何でもかんでも処理を専用ハードウェア化しようとするのは褒められた方法ではありません．適切な「ソフトウェアのハードウェア化」のためにまず必要な知識は，EDAツールの細かい利用法やコードの記述ノウハウよりも，人（回路技術者）がどのような考え方を持ってソフトウェア処理のハードウェア化を組み立てているのか，ということです．それが本特集の主テーマです．

1．ソフトをハード化する必要性の高まり

●「ソフトのハード化」は古くから実践されてきた

C/C＋＋言語，あるいはより上位のモデリング言語などで書かれたアルゴリズムのハードウェア化は，最近になって突然話題になった技術テーマではありません．

実際にはかなり古い話であり，筆者が自ら直接に経験した限りでも，1990年までには既にディジタル回路設計の現場で実践されていました．

例えば数値演算や誤り訂正などは高速性を求めて古くからハードウェア化の対象になってきました．そのような回路を作る技術者たちは，ソフトウェアからのハードウェア化をごく自然なこととして行っていました．なにしろ，当時の十分整っているとは言い難い回路設計環境下[注2]でデータ処理回路を作ろうとすれば，まずソフトウェアを組んでみてアルゴリズムを検証し，次にそれをハードウェア化するというように段階を踏むことが，今以上に必要でした（図1）．そうしなければ，設計した回路が動作しなかったときに，アルゴリズムの理解を間違えたのか，回路化の時点でタイミング設計や論理設計をミスしたのか，原因の特定が

注1：いろいろなアルゴリズムを使ってデータを加工・演算する回路のことを指す．これに対し，例えばUSB，PCIバス，オンチップ・バス（AMBAなど），メモリ・インターフェースなどシーケンス制御が中心の回路も多くあるが，このタイプの回路をソフトウェアから作る必然性は高いとはいえない．ただし，データ処理回路であっても，演算とデータ入出力処理（プロトコル制御など）が密着している場合は多く，データ処理とシーケンス制御を別々の言語・環境で作るようなことは非現実的である．

注2：HDLや論理合成の能力は今よりずっと貧弱であった．さらに以前には手書き回路図で設計をしていた時代があった．当時は，回路図エントリしてシミュレーションができるだけで感動していたものである．

Keyword　システムLSI，IPコア，C/C++，EDAツール，動作合成，LSI設計方法論，逐次実行，並列動作，オンチップ・バス

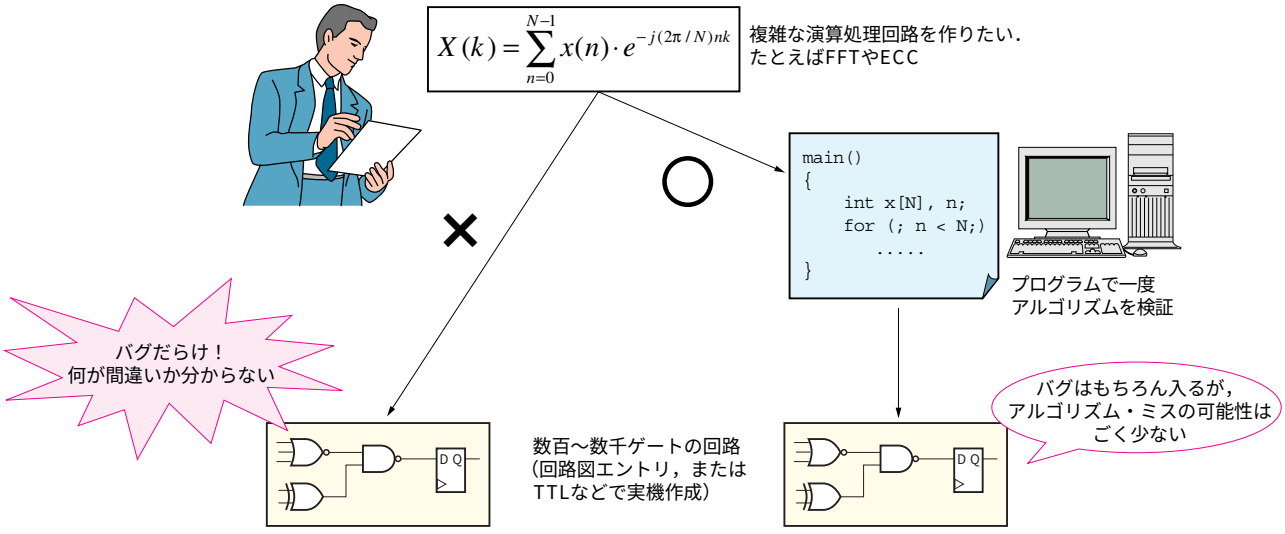

図1　「ソフトウェアのハードウェア化」のアイデアが生まれた当時の状況

ソフトウェアでアルゴリズムを検証せずに，ゲート・レベルで直接ハードウェアを設計してしまうと，動作しなかったときに，アルゴリズムの理解を間違えたのか，タイミング設計や論理設計などのケアレス・ミスなのか，原因がまったく特定できない．

できなかったからです．ただし，データ処理回路を作る人たちはディジタル回路技術者の中でも限られていたので，誰もがこのような手法を使っていたわけではありません[注3]．

また，ソフトウェアとハードウェアの組み方の違いという点についても，昔からさまざまな知識が蓄積されています[注4]．それは今でも（むしろ今だからこそ）高い価値があります．作る回路や使うツール，細かい設計技法は時代とともに変わっても，設計バランスのとり方やリスク回避の仕方など，設計の根本にある考え方はほとんど変化しないものです．そのような視点から，参考文献(1)などもひもといてみてください．

● 今までは人手によるソフトのハード化が主流だった

時代が進んでHDL（Hardware Description Language）設計への移行が済んでも，「ソフトウェアのハードウェア化」に関してはあまり革新的変化がなく，設計現場ではおおむね次のいずれかの形態がとられていました[(2),(3)]．

- ハードウェア用のアルゴリズムを一度ソフトウェアに実装し，デバッグを行う．そのソフトウェアを書くのは回路技術者自身であり，ハードウェア構造もある程度反映した形で作られる[注5,(2),(4)]．ただし，この方法では，アルゴリズムは検証できるが，複数回路ブロックの並列動作やブロック間のデータ・フロー制御までは検証できない．次に，そのソフトウェアをベースにしてRTL（Register Transfer Level）コードを起こし，並列動作時の様子を検証する．
- ハードウェア仕様書の一部として，普通のソフトウェア・コードが回路技術者に渡される[注6]．回路技術者はそれを読んで骨子をつかみ，処理アルゴリズムをハードウェア用に考え直したうえで，RTLコードを起こして検証する．

いずれの場合も，ソフトウェアを作った後でまた別途RTLコードを作らなければならず，二度手間である点は同じです．

また，いくらソフトウェアの段階で検証をしても，人手でRTLコードを作成すれば，また別のバグを作りこんでしまいます．特に上記の並列動作やデータ・フロー制御の部分には，容易には発見できないバグが入りやすく，検証項目のうちかなりの重みがあります．そこで，「動作合成でソフトウェアを自動的にRTLコードに落とし込めないものか」という願望は非常に根強くありました．

注3：ソフトウェアを利用しない技術者でも，別の方法（状態遷移図やタイミング・チャートなど）で回路の動きを吟味し，それから回路図を引いていたので，段階を踏むという点では同じ．
注4：古くはソフトウェアとハードウェアが今ほど分業化されておらず，ソフトウェアとハードウェアの両方の設計経験とセンスを兼ね備えた技術者が少なくなかった．Cベース設計の導入には，そのような両方の素質を持つ人材が大いに役立つであろう．
注5：RTLより高い抽象度での動作確認のためには，必ずしもC/C++などのソフトウェアを使う必要はなく，HDLのビヘイビア記述を使う手もあった．しかし，筆者は実際にそうやられているのを見たことがないし，自らやったこともない．ほとんどの場合，検証環境の作成がやりやすいなどの理由でC言語が使われていた．
注6：このソフトウェア・コードは，作成したRTLのテスト・データ生成にも使えるので，無駄になるわけではない．