これからの組込みシステム開発

豊橋技術科学大学 情報工学系 組込みリアルタイムシステム研究室

高田　広章

ZIPC Watchers に ITRON について何か書くように依頼されたが、ITRON については以前（Vol.3）に書かせていただいている。そこで今回は、最近考えている今後の組込みシステム開発の目指すべき姿の一つ（システム LSI を用いた組込みシステム開発に着目したもの）について書かせていただくことになった。

1．ソフトウェア技術者とハードウェア技術者のギャップ

最近、学会や標準化活動などの場で、システム LSI 設計手法・ツールの研究者・開発者と議論する機会が多い。システム LSI の適用分野の多くは、組込みシステムに分類されるものであり、この分野の研究者・開発者の組込みシステムに対する関心は高い。システム LSI は、文字通りシステム全体を一つの LSI 上に集積するものであるため、LSI 設計技術者や EDA ツールの開発技術者（以下、ハードウェア技術者と総称させていただく）も、ソフトウェアまで含めたシステム全体を理解する必要があると考えられるためである。

ところが、議論してすぐに気付くのは、ソフトウェア技術者とハードウェア技術者の考え方に、大きなギャップがあることである。典型的には、SpecC や SystemC など、C または C++をベースとしたハードウェアを記述するための言語が提案されている。ハードウェア技術者は、これらの言語をシステム記述言語(ないしは、システムレベルの記述言語）と呼び、「システム」の「仕様」を記述するものと捉えているが、ソフトウェア技術者にとっては、C 言語レベルまで落ちてしまえば実装そのものであり、それを「仕様」といわれても困ってしまう。また、ハードウェア技術者のいう「システム」には、ソフトウェアが複雑化して困っている部分（例えば、GUI）は抜け落ちてしまっていることが多い。「システム」の捉え方の違いは、ハードウェア技術者とソフトウェア技術者に、(例えば) 携帯電話の「システム構成図」を書いてもらえばよくわかる。

逆にソフトウェア技術者には、ハードウェア技術者が抱えている問題は理解できていない（私もソフトウェア側であり、言葉では聞いていても、本当の問題はよく理解できていないと思うので、これ以上は書かない)。結局、ソフトウェア技術者もハードウェア技術者も、相手の苦労(言い換えると、仕事) を理解しておらず、それぞれが抱える問題だけを主張してい

る場面が多い。

2．ソフトウェア/ハードウェア コデザイン

そのような中で、ハードウェアとソフトウェアのコデザイン（協調設計）技術の研究・開発が行われてきたわけだが、それらがハードウェア技術者に主導されてきたこともあり、ソフトウェア技術者にメリットがあるものとはなっていない。

しかし、コデザイン技術を、ハードウェアとソフトウェアを協調して設計することで、システムの性能・品質や設計効率を向上させるための技術と広く捉えれば、ソフトウェア技術者にもメリットがあるはずである。例えば、

(a) デバイスの機能がCベースの言語で書かれていれば、わかりにくいデバイスのマニュアルを読むよりはデバイスの理解が容易になり、デバイスドライバの開発は容易になるだろう。

(b) 逆に、ソフトウェア技術者側でハードウェアの機能を記述し、その実現作業をハードウェア技術者に任せられれば、考えていたものと違ったハードウェアができてくるという心配はないはずである。また、その記述が実行可能であれば、ハードウェアができてくる前にソフトウェアのテストを進めることができる。

(c) 同じ記述から、ソフトウェアにもハードウェアにも落とすことができれば、ある機能モジュールをソフトウェアで実現するかハードウェアで実現するかの決定を遅らせることが可能になり、ソフトウェアとハードウェアの役割分担を最適化しやすくなるだろう。

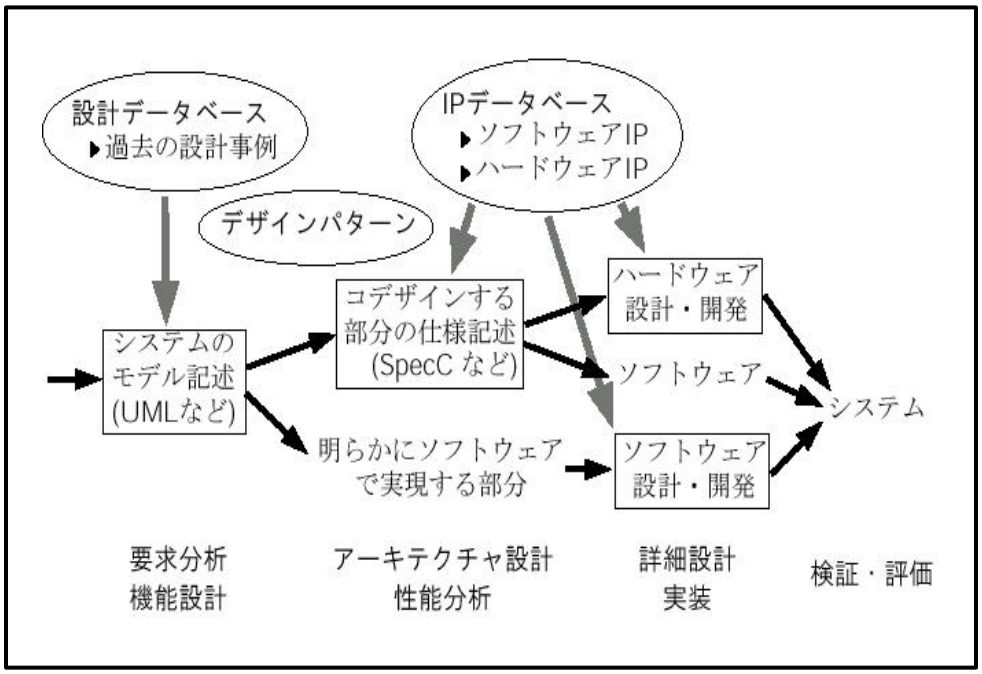
といったことが考えられる。

ちなみに、この節のタイトルを「ソフトウェア/ハードウェア コデザイン」と通常の用語とハードウェアとソフトウェアを入れ換えたのは、ソフトウェア技術者のためのコデザイン技術を強調したかったためである。

3．組込みシステム開発の目指すべき姿（ の1つ ）

このようなことを考えながら描いたのが、次ページの図に示す組込みシステム開発の流れである[1]。

この図のポイントはいくつかあるが、コデザインの観点からは次の点が重要である。システムの開発期間短縮のためには、ソフトウェアとハードウェアの並行開発が有効であるが、システムのモデル記述を行った段階で明らかにソフトウェアで実現する部分とどちらで実現する可能性もある部分（これを図ではコデザインする部分と呼んでいる）に分離し、そこから並行開発を行うのが現実的である。コデザインする部分については、ソフトウェアとハードウェアのいずれにも落とせる記述を用いて設計を進め、ソフトウェアとハードウェアの役割分担の最適化を図る（前節の(c)のメリット）。



図．組込みシステム開発の目指すべき姿

本稿ではここまで、コデザインの観点に重点を置いて説明してきたが、図に示す開発の流れの中で、上流の設計工程としてシステムのモデル記述が必要であること、過去の設計資産（設計事例や IP）の再利用やデザインパターンの活用が重要であることも強調しておきたい。なお、この図に関する詳しい説明については、文献[1]を参照していただけると幸いである。

ZIPC は、上流に ROSE などのオブジェクト指向設計ツールとの連携を、下流に SpecC をベースとした設計ツールである Visual Spec との連携をサポートしており、この図の中では、システムのモデル記述の最後の段階およびコデザインする部分の仕様記述のためのツールとして有効と考えられる。

４．課題は多い

図に示したシステム開発の流れを実現するために解決すべき課題は多い[1]。

例えば、SpecC などのいわゆるシステム記述言語が、その狙い通り、ソフトウェアとハードウェアの分離前の記述に十分であるかは、十分に評価されていない。これらの言語は、EDA ツールの研究者・技術者から出てきたものであるから、そこからハードウェアを生成する方法については検討がなされていると思われる。

ソフトウェアについても、OS（ないしは、カーネル）を使わない単純なプログラムについては大丈夫であろう。

しかし、OS を使ったソフトウェアのことは、ほとんど考慮されていないと言える。具体的には、SpecC の記述の中で OS のシステムコールを呼び出すと、ハードウェアには落とせなくなるのは明らかである。逆に、SpecC の持つ並列性や同期の記法だけでは、OS を使った並行動作するソフトウェアの記述には十分でない（例えば、排他制御のための記法は用意されていない）。

SpecC の言語仕様の改良については、STOC(SpecC Technology OpenConsortium)で議論が始まっており、そのような活動の中で徐々に課題が解決されていくことを期待したい。

筆者の研究室でも、上の(a)の考え方を一歩進めて、デバイスとデバイスドライバの機能を一体で記述し、そこからデバイスとデバイスドライバを生成する手法について研究をはじめたところである。この一体の記述に SpecC を使おうと考えたことから、STOC の活動にも参加させていただいている。数多くの課題の解決に、いくらかでも貢献できればと考えている。

５．参考文献

[1] 高田広章，組込みシステム開発技術の現状と展望，情報処理学会論文誌，Vol.42，No.4（掲載予定）。