SystemC言語、採用の検討から実装までの事例
―無線通信機器への応用―

サンデン株式会社
技術本部　技術開発センター
岩崎　渉

１．はじめに
１.１．要約
本稿では無線通信機器（３方式無線通信モデムmoderno）の開発において、SystemC言語のための開発ツールとしてキャッツ株式会社のSystemC Debuggerとキャッツ株式会社で代理店販売している株式会社礎デザインオートメーションのDesignPrototyperを採用するまでの経緯と、これらのツールを活用することでハードウェアのCベース設計に取り組んだ事例を紹介します。

１.２．会社概要
サンデン株式会社は“冷やす・暖める”をコア技術とし、自動車機器システム事業、流通システム事業、住環境システム事業を軸に世界４極体制でグローバルに事業を展開しています。

１.３．無線通信モデム開発の経緯
当社は、無線通信モデムを利用した自動販売機オンライン情報収集システムを開発しています。その中で、無線通信モデムの相互運用性・双方向性・信頼性を向上させることが課題でした。

そこで、当社の呼びかけの元、KDDI株式会社、サンデンシステムエンジニアリング株式会社、株式会社高崎共同計算センター、国立大学法人群馬大学、群馬県立群馬産業技術センターとのコンソーシアムが結成され、これらの問題を解決する無線通信モデムの開発プロジェクトが発足しました。

本発足テーマは、経済産業省の平成16年度地域新生コンソーシアム研究開発事業に「共通インターフェースによる相互運用可能な無線通信モデムの開発」として採用されています。

２．３方式無線通信モデム
２.１．現在のモデムの課題
現在の無線通信モデムは、特定の通信方式専用に開発されています。したがって、モデムを接続する上位装置は特定の通信方式にしか対応することができません。しかし、実際には上位装置の設置場所や機能に対応した最適な通信方式を選択しなければ、通信品質を確保することは困難です。

このため、通信方式の選択が不可能であることは、無線通信ネットワークサービスの普及を阻害する大きな要因となっています（図１）。
さらに、上位装置を特定の通信方式の規格に合わせて設計してしまうと、新たな通信方式による通信モジュールが提供される毎に上位装置の設計変更の必要が生じてしまう問題があります（図２）。

２.２．３方式無線通信モデムの特徴
開発している３方式無線通信モデムは、国内を代表する通信事業会社（CDMA、PHS、DoPa）の３種類の通信方式に対応し、無線通信サービス事業を展開する上で必要な通信方式のすべてをカバーすることが可能です。上位装置と通信モジュールの間に共通インターフェースを置き、共通インターフェースを変更できる構造とすることで、低コストで複数の通信方式に対応することを実現します（図３）。

３方式無線通信モデムを実現する共通インターフェースは、上位装置と通信モジュール間のデータのフィルタリングや、変換が主要な機能です。通信方式、上位装置が異なればフィルタリング、変換する内容も変わってきます。このため新たな通信方式や上位装置による違いに柔軟に対応できなくてはなりません。

そこで、本プロジェクトはFPGA（Field

図1　課題（1）複数通信方式への対応

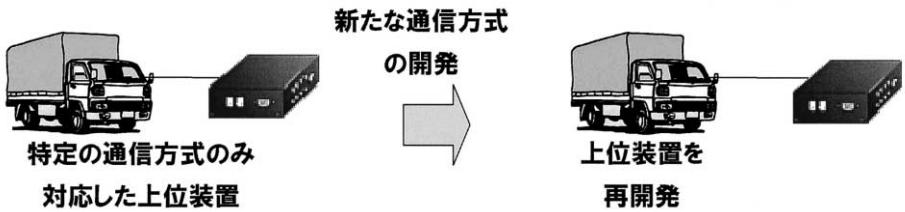

図2　課題（2）新通信方式への対応

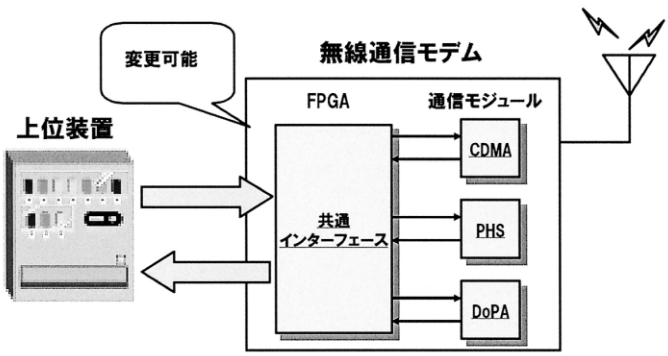

図3　共通インターフェース

Programmable Gate Array）を採用することにしました。FPGAは開発者の手元で内部の回路を自由に設計・変更できるデバイスであり、内部回路の変更により上記の問題を解決することを意図しています。

3．開発環境の検討

3.1．SystemC言語

FPGAはHDL（Hardware Description Language）を使用しての設計が一般的ですが、新たな通信方式の開発に伴い頻繁に共通インターフェースが変更されることが予測され、より上流のSystemC言語による設計に取り組むことにしました。

SystemC言語は、C++のクラスライブラリで、ハードウェア記述の機能が拡張されています。SystemC言語を導入する上で、次のことに期待しました。

- 上流の設計から下流の設計までSystemC言語で設計したい
- ソフトウェア設計者がハードウェアを設計できるような環境を構築したい
- C,C++で記述された過去のソフトウェアの資産を部分的にハードウェアに変更したい

3.2．開発ツール

図4はSystemC言語を利用したFPGAの開発フローの概略図です。アルゴリズム検討からコーディングまでがSystemC言語を使用する工程で、当社ではキャッツ株式会社のSystemC

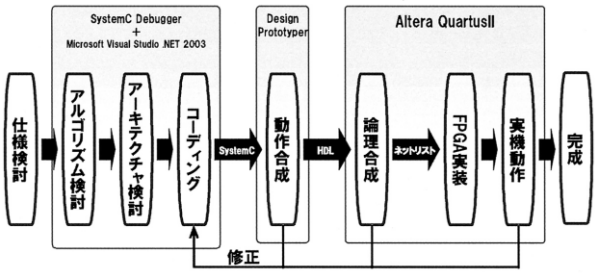

図４　FPGA開発フローの概略図

DebuggerとMicrosoft Visual Studio .NET2003の環境で開発しました。

SystemC Debuggerは、Microsoft Visual Studio .NET2003のアドインとして提供されています。フリーの開発環境も存在しますが、Microsoft Visual Studio .NET2003の環境で構築したかった点と、フリーの環境に比べて機能的に優れている（SystemCウォッチ、波形観測）点でキャッツ株式会社のツールを採用しました。

動作合成の工程では、キャッツ株式会社で代理店販売している株式会社礎デザインオートメーションの動作合成ツール、DesignPrototyperを使用しました。動作合成ツールを検討したときにDesignPrototyperの他にも候補がありましたが、唯一の国内ベンダーで、サポートが期待できる点と、価格の面で、DesignPrototyperを選択しました。

論理合成から先の工程ではAltera社のQuartusIIを使用しました。QuartusIIはFPGAの統合開発環境です。

使用する開発ツールをまとめると、次のようになります。

・SystemCコーディング、シミュレーション、デバッグツール
①SystemC Debugger（キャッツ株式会社)、
② Microsoft Visual Studio .NET 2003（Microsoft）
・動作合成ツール
①DesignPrototyper（株式会社礎デザインオートメーション）
・論理合成、HDLシミュレータ、配置・配線ツール
①QuartusII（Altera）

３.３．コンサルティング

ツールの導入後、キャッツ株式会社に依頼し、DesignPrototyperの開発元である株式会社礎デザインオートメーションの有料コンサルティングを受けました。アーキテクチャ検討からFPGA実装まで、丁寧にご指導頂きました。

４．開発

４.１．アルゴリズム検討

アルゴリズム検討をする段階ではSystemC言語の環境は構築できていなかったので、C#で検討しました。SystemCでもC#と同様の検討が可能だと思われます。

４.２．アーキテクチャ検討

まず、ソフト・マクロCPUの使用を検討しました。ソフト・マクロCPUをFPGAに実装することで、マイコンのようにFPGAを使うことができます。次に、ソフト・マクロCPUを使用しない方法を検討しました。この場合、ロジック回路だけで無線通信モデムに必要な機能を実装することになります。

本プロジェクトでは、ソフト・マクロCPUを使用する方法と、使用しない方法の両方で試作することにしました。私が担当するのは、主にソフト・マクロCPUを使用しない方法です。ソフト・マクロCPUを使用する方法より安価（回路規模を小さく）で、高性能な回路を作ることが目標です。ソフト・マクロCPUを使用する方法は、SystemC言語を利用しなかったため本稿では省略します。

次に、データの流れについて検討しました。基本的に、データは上位装置から通信モジュール（“上り”と定義する）に送信される方向と通

信モジュールから上位装置（“下り”と定義する）に送信される方向の２方向です。それぞれの方向のデータ処理が並列で動作する構成を目指しました。

これらの動作検討は、パソコンを使って次のようなクロス開発環境で実行しました。共通インターフェースは、SystemC Debuggerでエミュレートして、上位装置と通信モジュールは実機を使用しました。上位装置と通信モジュールに実機を使用する環境を構築することによって、テスト環境を整える工数を削減できます。

クロス開発環境で検討した結果、共通インターフェースは上りのデータ処理をするModule1、下りのデータ処理をするModule2、受信・送信のUART、FIFO1,2の構成となりました（図５）。

４.３. コーディング

コーディングはキャッツ株式会社のSystemC Debuggerを使用しました。図７はSystemC Debuggerの画面です。SystemCウォッチで変数の値、プロセスの状態を知ることができます。図８はポートの波形の画面です。

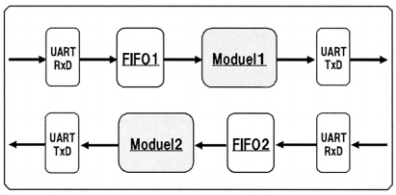

図５　共通インターフェースの構造

通常、SystemC言語で波形を観測したい場合は、観測したいポートのトレースコードをソース上に記述する必要があるのですが、SystemC Debuggerは波形を観測したいポートをGUI上で簡単に選択できます（図６）。波形を観測したいポートを選択して、トレースファイル作成のボタンを押すと、自動的にコードを作成してくれます。手作業での入力によるミスと、入力に費やす時間を大幅に削減してくれます。

コーディングは、次の動作合成のステップで利用する動作合成ツールの制約を意識して行う必要があります。主な制約を次に挙げます。

- BCA（Bus Cycle Accurate）レベルで記述しなくてはならない
- C言語の標準ライブラリ関数は使用できない
- メモリは動的に確保できない

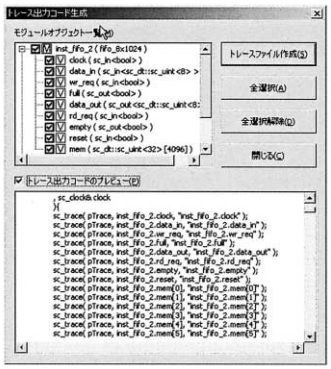

図６　トレースファイルの作成

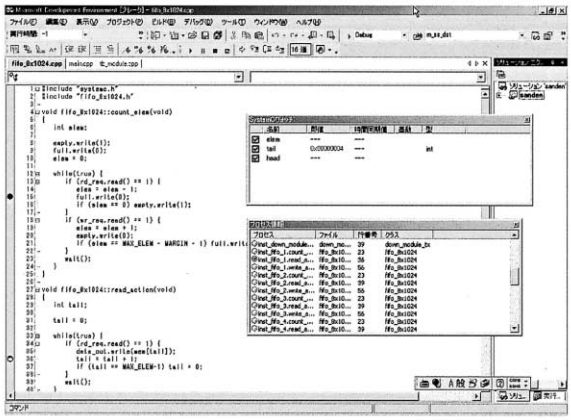

図７　SystemC Debuggerの画面

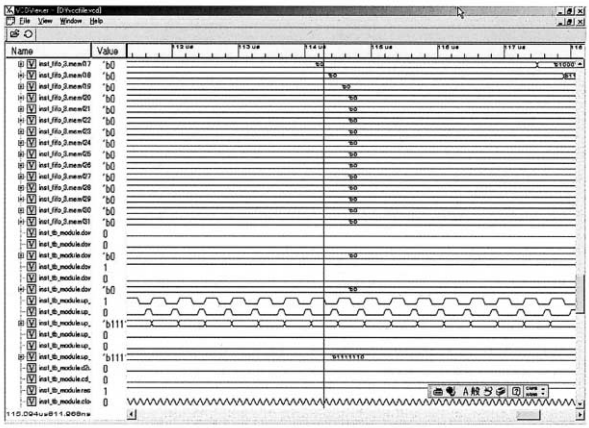

図８　波形観測の画面

・関数の再起呼び出しは使用できない
・固定小数点型、浮動小数点型は使用できない

上記の制約の中で苦労したのがBCAレベルで記述することでした。BCAレベルの記述は、モジュール間のインターフェース（データの送受信）をクロック精度で記述する必要があるため、クロックを考慮してコーディングする必要があり、タイミングを合わせるのに苦労しました。

自分で作成したモジュール間ではタイミングの問題が起こることは少ないのですが、IP（Intellectual Property）として提供されているモジュールとの間では、タイミングが合わずに頻繁にデッドロック状態に陥ってしまいました。IPの仕様を把握することと、波形を観測することで、問題は回避することができると思います。

４.４．動作合成

コーディングのあと、動作合成をします。動作合成ツール、DesignPrototyperにSystemC言語を入力して動作合成することでHDLが出力されます。

図９はDesignPrototyperの実行画面です。動作合成をする段階で、変数を共有する、回路を共有化する、配列変数をメモリにマッピングするなどの、コードの最適化をしました。

変数や回路を共有化すると、並列性が失われて処理速度が遅くなってしまいますが、作成される回路規模が小さくなりFPGAのコストを抑えることができます。

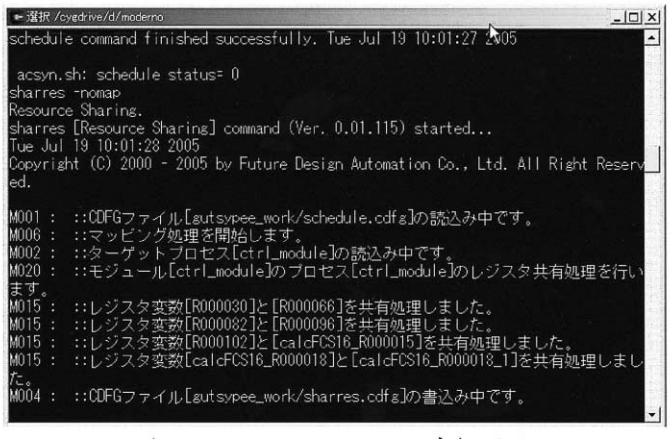

図９　DesignPrototyperの実行画面

配列変数をメモリ・ブロックにマッピングすることも回路規模の縮小化につながります。当初は、配列変数も他の回路と同様に、ロジック・エレメントにマッピングしていたのですが、その方法は、合成後の回路規模が大きくなりすぎてしまい、論理合成ができない問題が発生しました。

DesignPrototyperのサポートに相談したところ、配列変数は、ロジック・エレメントにマッピングするのではなく、メモリ・ブロックにマッピングすることを勧められました。配列変数をメモリ・ブロックにマッピングすることで回路規模を縮小することができ、問題を解決することができました。HDLを使いこなすハードウェア技術者には非常に初歩的なことだと思うのですが、本プロジェクトが発足した当初は、わからないことばかりで毎日のようにサポートに助けてもらいました。

コードの最適化のあと、DesignPrototyperによるSystemC言語からHDLへの動作合成をしたところ、SystemC言語で500行程度のモジュールの場合、数十秒の処理時間で合成できました（Windows2000、Intel Mobile Celeron CPU 2.2GHz、768MB　RAM）。

5．実装

5.1．論理合成

DesignPrototyperで動作合成したHDLファイルは、加工することなく論理合成ツールで合成できます。ただし、モジュールをインスタンス化するファイルは手作業で作成しなければなりません。モジュールをインスタンス化したり、ポートと信号線をつないだりしなくてはならず、時間もかかり、人為的ミスも誘発されやすい部分です。論理合成ツールはALTERA社のQuartusIIを使用しました。

5.2．FPGA実装

論理合成した結果をFPGAに実装しました。FPGAへはJTAGを使用して書き込みます。

5.3．実機動作

実機動作は、ネットワーク網に接続するところまで確認しました。CDMAの通信モジュールを使用して、100kbps以上の通信速度を記録することができたので、要求される仕様を満たすことができることが確かめられました。

6．まとめ

キャッツ株式会社のツール、SystemC Debuggerとキャッツ株式会社で代理店販売をしている礎デザインオートメーションのツール、DesignPrototyperを使用して3方式無線通信モデムmoderno の共通インターフェースのSystemCによるハードウェア設計に取り組み、実機で動作の確認ができました。

したがって、SystemC言語で比較的上流から下流までの設計ができることと、ソフトウェア技術者がSystemC言語を使ってハードウェアの設計ができることが確かめられました。当社としては、今後の開発にも活用したいと考えています。

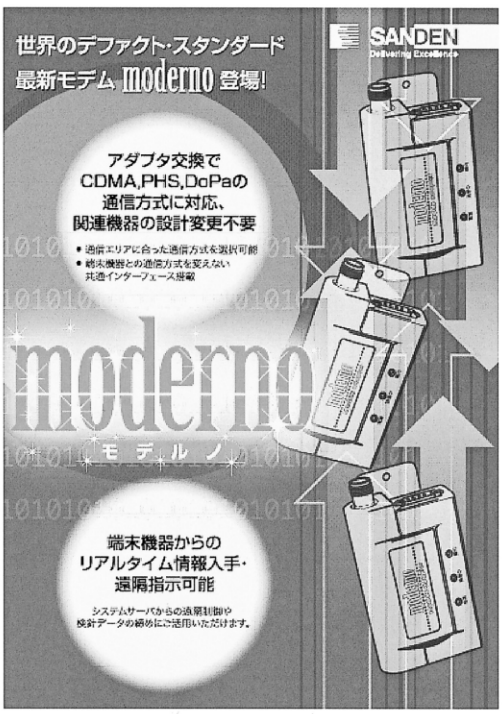

図10　moderno