システムレベル設計環境：SystemBuilder

本田　晋也[†a)]　　冨山　宏之[††]　　高田　広章[††]

SystemBuilder: A System Level Design Enviroment

Shinya HONDA[†a)], Hiroyuki TOMIYAMA[††], and Hiroaki TAKADA[††]

あらまし　近年の組込みシステム開発においては，設計生産性の向上による開発期間の短縮化が不可欠である．設計生産性を向上させる方法としては，設計対象をできるだけ高い抽象度で設計する方法があるが，ハードウェアとソフトウェア間を接続するインタフェースであるデバイスドライバやバスインタフェースの設計を自動化するツールはこれまで実用化されていない．そこでは我々は，高い抽象度でのインタフェース記述が可能なシステムレベル設計環境（SystemBuilder）を開発した．SystemBuilder はソフトウェアとハードウェアの区別なく記述されたデザイン記述とその分割方法を入力とし，ソフトウェアとハードウェア及びその間のインタフェースを自動合成する．本論文では開発したシステムの概要と実装技術について述べ，その有用性を JPEG デコーダの設計事例を通じて評価した．

キーワード　ソフトウェア/ハードウェア・コデザイン，組込みシステム，動作合成，リアルタイム OS

1. まえがき

近年，組込みシステム開発においては，製品の企画から市場への投入までの時間（Time-to-Market）の短縮化が重要となっている．Time-to-Market の短縮化には，設計生産性の向上による開発期間の短縮化が不可欠である．設計生産性を向上させる方法としては，設計対象をできるだけ高い抽象度で設計する方法があり，ソフトウェア開発においてはオブジェクト指向設計が一般的に用いられ，ハードウェア設計においては C 言語若しくはシステムレベル言語と動作合成技術を用いた動作レベルの設計が適用されつつある [1], [2].

しかしながら，ハードウェアとソフトウェア間を接続するインタフェースであるデバイスドライバやバスインタフェースの設計に関しては，設計抽象度を引き上げる手法はこれまで開発されていない．特にデバイスドライバのうちハードウェアを直接操作する部分の設計については，設計者はデバイスレジスタの特定のビットへのアクセスやプロセッサの割込みといった低いレベルの設計を直接行わなければならならず，多くの設計工数がさかれているのが現状である [3].

また，LSI 技術やマイクロプロセッサの発達により，従来はソフトウェアにより実現していた機能をハードウェアとして実現することや，その逆が可能となってきている．このような状況においては，設計制約を満たすよう設計対象の機能を適切にソフトウェアとハードウェアに分割することが重要となる．最適な分割方法の決定には，様々な分割での性能評価が必要であるが，正確な性能見積りは困難であるため，正確な評価を行うためには分割方法に従い実際に実装する必要がある．従来の設計手法では，設計の初期段階から設計対象をソフトウェアとして実装する部分とハードウェアとして実装する部分とを明確に区別し，それぞれを独立して設計する．そのため，分割方法の変更は設計上の大きな手戻りを意味し，多くの開発工数が必要となる．

そこで，まず設計対象をハードウェアとソフトウェアの明確な区別をせず C 言語等で一体に記述し，次にこの記述の分割方法を決定した後，その分割方法に従い実装記述を自動合成する手法が考えられる．一体記述からのハードウェア合成に関しては動作合成技術に

† 豊橋技術科学大学情報工学系，豊橋市
Department of Information and Computer Sciences, Toyohashi Univ. of Technology, Toyohashi-shi, 441–8580 Japan
†† 名古屋大学大学院情報科学研究科情報システム学専攻，名古屋市
Department of Information Engineering, Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan
a) E-mail: honda@ertl.jp

より実現可能であるが，加えてインタフェースの生成が自動化されていなければ，一体記述から実装記述を自動合成することは不可能である．

そこで我々は，インタフェースの設計抽象度を引き上げたシステム設計が可能なシステム設計環境（SystemBuilder）を開発した．設計者は，設計対象の機能と通信を明確に区別して C 言語若しくは SpecC 言語 [4] で記述する．機能間の通信に関しては，通信プリミティブと呼ばれる 3 種類の通信チャネルを用意しており，設計者はこの通信チャネルに定められたアクセス関数を用いて機能間の通信を記述する．

この記述を入力とし，ソフトウェアとハードウェアへの分割方法を指定すると，SystemBuilder は指定された分割方法に従って，ソフトウェアとハードウェア及びその間インタフェースを自動合成する．なお，ハードウェア合成に関しては外部の動作合成ツールを呼び出す．SystemBuilder は，このほかにも，機能レベルのシミュレーション，コシミュレーション，FPGA への実装機能をもつ．なお，合成するソフトウェアやデバイスドライバは ITRON 仕様の RTOS の使用を前提とする．

以下，本論文では，まず **2.** で本システムの概要と特徴及び関連研究について述べ，続く **3.** で設計対象の記述方法について述べる．**4.**〜**7.** では，本システムの実装技術について述べる．**8.** では JPEG デコーダの設計事例により本システムの有効性を評価する．

2. システムの概要及び特徴

2.1 概　　要

SystemBuilder は組込みシステム用の設計環境であり，連携して動作する複数のソフトウェアとライブラリの集合である．SystemBuilder を用いた設計フローを図 1 に示す．四角が設計者による入力，楕円が SystemBuilder による処理である．

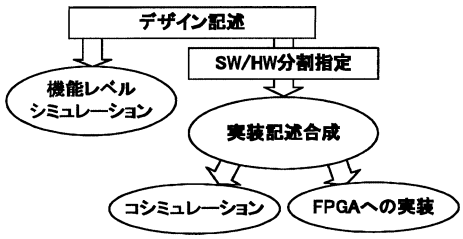

図 1　設計フロー
Fig. 1　Design flow.

設計者は設計対象を SpecC 言語または C 言語を用いて記述する（この記述をデザイン記述と呼ぶ）．設計対象の機能はまとまった単位で機能単位と呼ぶオブジェクトとして記述する．機能単位間の通信は通信オブジェクトとして定義し，機能単位は定められたアクセス関数を用いて通信オブジェクトにアクセスする．

次に記述したデザイン記述が機能的に正しいかをシミュレーション環境を用いて確認する（機能レベルシミュレーション）．

機能検証の後，デザイン記述のソフトウェアとハードウェアへの分割方法を指定して実際にソフトウェア・ハードウェアとして実装可能な実装記述を合成する．

実装記述の合成後，ソフトウェアとハードウェアをそれぞれ RTOS シミュレータと HDL シミュレータ上で協調させて実行するコシミュレーションを行うことや，ソフトコアのプロセッサ用いて FPGA へ実装することが可能である．

2.2 構　　成

SystemBuilder は設計環境全体を示す言葉であり，実際には複数のソフトウェアにより構成されている．SystemBuilder の構成を図 2 に示す．図中の楕円はソフトウェアであり，長方形は入出力ファイルである．

（1） 新規開発ソフトウェア

SystemBuilder の構成ソフトウェアのうち，以下に示すものは新規に開発を行った．

（1-1）Sc2C

SpecC 言語により記述されたデザインを入力とし，入力された SpecC 記述の構成を解析して構成記述ファイル（SDF）に出力し，記述自体を C 言語記述に変換する．入力言語を SpecC 言語とした場合，SpecC 言語を直接扱えるコンパイラや動作合成ツールはないため，本ソフトウェアにより SpecC 言語入力をいったん C 言語記述に変換する（SpecC 記述変換）．SpecC Reference Compiler [5] をベースに開発し，記述言語は C++言語で，新規開発したソースコードの行数は 16249 行，変換に用いるテンプレートや，変換後の記述で用いるライブラリの行数は 1121 行である．

（1-2）SysGen

構成記述ファイル（SDF）を入力とし，機能レベルシミュレーションと実装記述用の ITRON コンフィギュレーションファイル，トップモジュール HDL ファイル，SDF からソフトウェア・ハードウェア間のインタフェースの情報のみを取り出し，後述するインタフェース生成ソフトウェアである BusConnecter の入

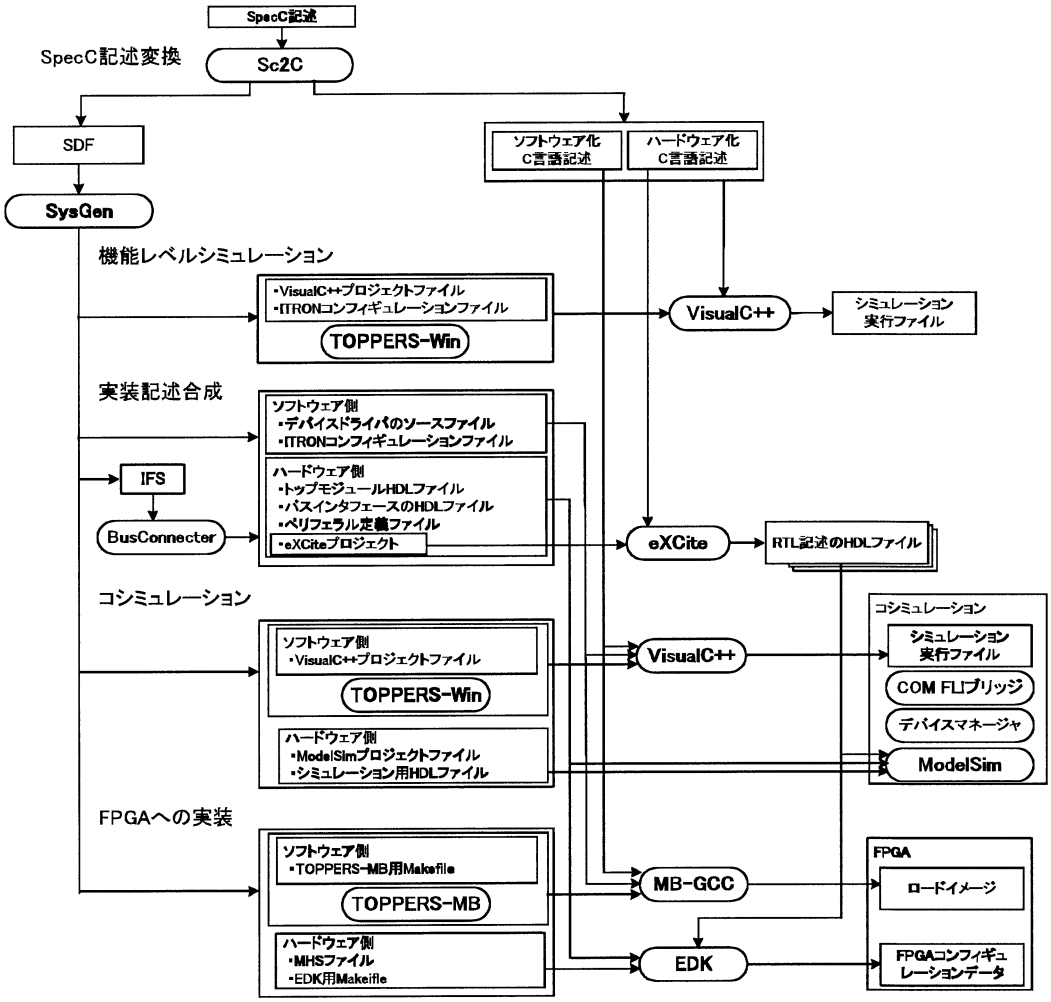

図 2　SystemBuilder の構成
Fig. 2　Organization of SystemBuilder.

力となるインタフェース構成ファイル（IFS），及び他のソフトウェアのプロジェクトファイルの生成を行う．記述言語は C++言語で，ソースコードの行数は 9066 行，変換に用いるテンプレートや，変換後の記述で用いるライブラリの行数は 5198 行である．

（1-3）BusConnecter

インタフェース構成ファイル（IFS）を入力とし，ソフトウェアとハードウェア間のインタフェースとして，デバイスドライバとバスインタフェースの HDL ファイル，ペリフェラル定義ファイルを生成する．記述言語は C++言語で，ソースコードの行数は 11348 行，変換に用いるテンプレートや，変換後の記述で用いるライブラリの行数は 6327 行である．詳細については **5. 4** で述べる．

（1-4）COM FLI ブリッジ

コシミュレーションで用いるシミュレーションバックプレーンであるデバイスマネージャとハードウェアシミュレータを接続するソフトウェア．記述言語は C 言語で，ソースコードの行数は 1553 行である．コシミュレーションの詳細については **6.** で述べる．

（2） 既存のソフトウェアの変更

以下のソフトウェアは性能のチューニングや安定化のため手を加えた．

（2-1）TOPPERS-Win

ITRON 仕様準拠の TOPPERS/JSP カーネル [6] の Windows 上のシミュレーション環境．VisualC++

を用いてユーザコードとリンクすることにより，シミュレーション用の実行ファイルが生成される．

（2-2）デバイスマネージャ

TOPPERS-Win に付随するシミュレーションバックプレーン．TOPPERS-Win と VisualBasic や C 言語で記述された外部アプリケーション（ハードウェアモデル）を Windows のプロセス間通信機能である COM によって接続する．

（2-3）TOPPERS-MB

FPGA 用のソフトコアプロセッサである Microblaze 用 TOPPERS/JSP カーネル．MB-GCC を使いコンパイル及びユーザコードとのリンクすることによりロードイメージが生成される．

（3） 既存のソフトウェアを利用

以下のソフトウェアは，既存のものを変更を行わずに用いている．

（3-1）eXCite [7]

動作合成ツール．C 言語を入力として単一の FSM で構成された RTL 記述の HDL ファイルを生成する．

（3-2）EDK [8]

MHS と呼ばれるプロセッサシステムの構成を記述したファイルから，FPGA 上のシステムを生成する．

（3-3）VisualC++

Windows 用の C/C++コンパイラ．

（3-4）MB-GCC

Microblaze 用の C/C++コンパイラ．

（3-5）ModelSim

HDL シミュレータ．

2.3 特　　徴

本システムの特徴をまとめると以下のようになる．

（1） インタフェースの設計抽象度の引上げ

前述のように本システムでは，機能間の通信は通信オブジェクトとして定義し，アクセス関数を介してアクセスする．実装記述合成では，指定された分割方法によりソフトウェア・ハードウェア間の境界となる通信オブジェクトからソフトウェア・ハードウェア間のインタフェースを自動合成する．そのため設計者は通信オブジェクトの実装形態（ソフトウェア間の通信，ハードウェア間の通信，ソフトウェア・ハードウェア間の通信）を考慮することなく設計可能である．

（2） 分割指定後の実装記述合成の自動化

これまで，SpecC 言語または C 言語によるデザイン記述を分割し，それぞれソフトウェアとハードウェアとして実装するためには，数多くのツールを組み合わせ，それらのツール間を人手で連携させなくてはならなかった．本システムでは，インタフェース合成を核として既存のツールを組み合わせて連携させることにより，分割指定後の実装記述の自動合成を実現した．そのため，一度設計対象を記述した後，様々な分割方法で実装した場合の評価が容易に可能となり，設計対象のソフトウェアとハードウェアの最適な分割方法の探索が短時間で実現できる．

（3） RTOS を考慮したソフトウェアモデル

近年の複雑な組込みシステムのソフトウェア開発においては，RTOS の使用は不可欠である．そのため，本システムで合成するソフトウェアは RTOS 上で動作するマルチタスクモデルとしている．

（4） コシミュレーション

一般に HDL 記述のハードウェアのシミュレーションにはテストベンチを用意する必要がある．本システムでは，実装記述合成によって合成されたソフトウェアとハードウェアを協調検証することが可能なコシミュレーション機能をもつため，新たにテストベンチを用意する必要がない．

（5） FPGA への実装機能

実装記述合成後のハードウェア記述は RTL 記述であるため，シミュレーションが低速である．そのため，FPGA への実装機能は，高速シミュレーション機能という面もある．

（6） SpecC 言語入力のサポート

デザイン記述において C 言語による入力と比較して言語レベルで並列性や通信を記述することが可能である．

2.4 関 連 研 究

C/C++言語をベースとしてソフトウェアもハードウェアも記述するための言語として提案されているシステムレベル言語 [4], [9] では，通信の設計抽象度を引き上げる試みとして，通信の記述のためにチャネルと呼ばれるオブジェクトを用意している．これらのシステムレベル言語を用いたツールが開発されているが [5], [10]，これらはシミュレーションを目的としたものであり，実装可能な記述を合成することはできない．

インタフェースを自動合成するシステムとして CoWare [11] がある．本ツールと異なり CoWare は動作合成技術を利用しておらず，機能の記述に関してはその機能をソフトウェアとして実装するかハードウェアとして実装するかを明確に区別してそれぞれ C 言語，HDL で記述しなければならない．

CoDeveloper [12] は，本システムと同様に C 言語によるソフトウェア・ハードウェア記述を行い，この記述に対して分割を指定した後の FPGA への実装を自動化している．しかしながら，RTOS を考慮したソフトウェアモデルが扱えないことや，コシミュレーション機能がなく合成したハードウェアの性能を容易に検証することができないといった問題がある．

3. デザイン記述

本システムの入力となるデザイン記述は SpecC 言語または C 言語により記述する．

3.1 C 言語記述

C 言語記述では，設計対象をまとまった機能ごとに機能単位（FU）として記述する．機能単位間の通信は 3 種類の通信プリミティブ（CP）として定義し，機能単位からはアクセス関数を用いて通信するよう記述する．機能単位はソフトウェアとしてはタスクに，ハードウェアとしては一つの FSM 回路に相当する．図 3 に 4 個の機能単位と 5 個の通信プリミティブで構成されたデザイン記述の例を示す．C 言語記述では，言語自体にこのような情報を記述できないため，System DeFinition（SDF）と呼ばれるファイルに記述する．SDF については **3.1.3** で述べる．

3.1.1 機 能 単 位

機能単位はそれぞれ固有の名前を付け，同じ名前をもつ関数を本体としてこの関数から実行される．

関数内部の記述に関してはソフトウェア化する場合には特に制約はない．ハードウェア化する機能単位の記述制約は，内部で呼び出す動作合成ツールである eXCite の制約に準ずる．具体的には，無限ループ内に処理を記述しなければならいことや，再帰関数が記述できないといった制約がある．なお，ハードウェア化を前提に書かれた記述はソフトウェア化が可能であるが，その逆は不可能である．

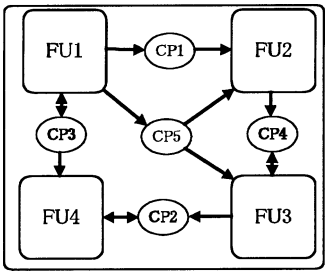

図 3 機能単位と通信プリミティブによる記述例
Fig. 3 An example of system description with function unit and communication primitive.

3.1.2 通信プリミティブ

機能単位間の通信は 3 種類の通信プリミティブを用いて定義する．システム中の通信プリミティブにはそれぞれ固有の名前を付ける必要がある．それぞれの通信プリミティブにはアクセス関数が用意されており，機能単位はこのアクセス関数を呼び出すことにより通信プリミティブを介して通信を行う．

それぞれの通信プリミティブは機能単位に接続されており，実装時には接続されている機能単位の実装先（ソフトウェアかハードウェア）により，ソフトウェア，ハードウェア，インタフェースのいずれかとして実装される．実装先により通信プリミティブに接続可能な機能単位の数には制限がある．例えばノンブロック通信プリミティブの場合，ソフトウェアとして実装された場合は接続数には制限はないが，ハードウェアとして実装された場合の書込み可能な機能単位の数は 2 個の制限がある．更にソフトウェアとハードウェア間のインタフェースとして実装された場合のハードウェア側に関してはリード可能な機能単位の数には制限はないが，書込み可能な機能単位の数は 1 個の制限がある．デザイン記述がこの制限を満たしているかは実装記述合成時にチェックされる．

以下にそれぞれの通信プリミティブの解説とアクセス関数について示す．なお，XXX は通信プリミティブに付けた名前を大文字化したものである．

（1） ノンブロック通信プリミティブ

ソフトウェアとしては共有変数，ハードウェアとしてはレジスタに相当する．アクセスはノンブロックで行われる．

```
アクセス関数
    XXX_READ(int* pdata)
    XXX_WRITE(int data)
接続可能な機能単位数
    ソフトウェア化：制限なし
    ハードウェア化：リード：制限なし，ライト：2 個
    インタフェース化
        ソフトウェア側：制限なし
        ハードウェア側：リード：制限なし
                        ライト：1 個
SDF ファイル
    NBCPRIM XXX, SIZE = 8|16|32
```

（2） ブロック通信プリミティブ

ソフトウェアとしては OS の通信機能，ハードウェ

アとしてはFIFOに相当する．アクセスは単方向にブロッキングで行われ，内部にバッファを任意個もつ．XXX_PREAD，XXX_PWRITEはインタフェース化の場合，ソフトウェア側からのアクセスのみで使用可能であり，ノンブロックでアクセスする．

```
アクセス関数
    XXX_(P)READ(int* pdata)
    XXX_(P)WRITE(int data)
接続可能な機能単位数
    ソフトウェア化：制限なし
    ハードウェア化：リード：1 個，ライト：1 個
    インタフェース化
        ソフトウェア側：制限なし
        ハードウェア側：リード若しくはライト：1 個
SDF フォーマット
    BCPRIM XXX, SIZE = 8|16|32, DEPTH = xx
```

（3） メモリプリミティブ

ソフトウェアとしてはグローバル配列，ハードウェアとしてデュアルポートメモリに相当する．アクセスはノンブロックで行われる．

```
アクセス関数
    XXX_READ(offset, int* pdata);
    XXX_WRITE(offset, data);
接続可能な機能単位数
    ソフトウェア化：制限なし
    ハードウェア化：2 個
    インタフェース化
        ソフトウェア側：制限なし
        ハードウェア側：1 個
SDF フォーマット
    MEMPRIM xxx, SIZE = 8|16|32, DEPTH = xx
```

3.1.3 SDF

C言語記述の場合，設計対象を構成する機能単位や通信プリミティブの数や名前及びそれらの接続関係はSDFに記述する．図3に相当するSDF記述を図4に示す．機能単位の宣言（**BEGIN_FU**）は，**FILE**で機能単位を記述したファイル及び実行に必要なファイルを指定する．**USE_CP**では，通信プリミティブとの接続関係を定義する．まず接続する通信プリミティブの名前を書き，その通信プリミティブに対して読込みを行うか書込みを行うかそれとも読み書きするかによって，括弧内にIN，OUT，INOUTを指定する．

3.1.4 SpecC言語記述

SpecC言語は，ANSI-Cをベースとして，ハードウェア記述のための並列性，同期・通信，タイミングなどの概念と構文が追加された言語である．SpecC言語記述の場合，機能単位がビヘイビアに，通信プリミティブがチャネルに相当する．

```
#デザイン名
SYS_NAME = test

#ソフトウェア化，ハードウェア化指定
SW = FU1, FU4
HW = FU2, FU3

#通信プリミティブの宣言
BCPRIM    CP1, SIZE = 32, DEPTH = 0
BCPRIM    CP2, SIZE = 32, DEPTH = 1
NBCPRIM   CP3, SIZE = 32
MEMPRIM   CP4, SIZE = 32, DEPTH = 1024
NBCPRIM   CP5, SIZE = 16

#機能単位の宣言
BEGIN_FU
    NAME = FU1
    FILE    = "fu1.c"
    USE_CP= CP1(OUT), CP3(INOUT), CP5(OUT)
END
BEGIN_FU
    NAME = FU2
    FILE    = "fu2.c"
    USE_CP= CP1(IN), CP4(OUT), CP5(IN)
END
...
```

図4 SDFの例
Fig. 4 An example of SDF.

本システムではSpecC言語の構文を制限付きでサポートしている．制限にはSpecC言語の言語仕様で明確化していない部分を明確化したものを含む．具体的には，piped変数/構文やタイミング構文は未サポートや，par構文でのビヘイビアの多重起動の禁止などである．更にハードウェア化する対象のビヘイビアに関しては**3.1.1**で述べたC言語記述と同様に内部で呼び出す動作合成ツールであるeXCiteの制約に準ずる．また，チャネルにはついては，カスタムなチャネルはサポートせず，通信プリミティブと同等の機能をもつ3種類のチャネルのみサポートしている．

4. 機能レベルシミュレーション

機能レベルシミュレーションでは，デザイン記述が機能的に正しいかを検証する．機能レベルシミュレーションは，デザイン記述全体をRTOS上で動作するソフトウェアに変換し，PC上で動作するRTOSシミュレータを用いて実行する．具体的には，機能単位（ビヘイビア）にRTOSのタスクを割り当て，通信プリ

ミティブ（チャネル）は共有変数やRTOSの通信オブジェクトに変換する．なお，機能単位自体の記述の変更は行わない．SpecC記述のシミュレーションについては文献[13]を参照のこと．

RTOSシミュレータとしては，ITRON仕様準拠のTOPPERS/JSPカーネルのWindowsシミュレーション環境（TOPPERS-Win）を用いている．このシミュレーション環境では，ITRONのタスクをWindowsのスレッドにマッピングするため，Windowsのネイティブデバッガ（VisualC++）がもつスレッドデバッグ機能を用いることが可能であり，効率的なデバッグ作業が行える．

なお，変換後の記述をコンパイルしてシミュレーションライブラリとリンクして実行するためのVisualC++用プロジェクトファイルは自動的に生成される．

5. 実装記述合成

実装記述合成は，SDFを入力とし，SDFに指定された分割方法に従ってソフトウェア，ハードウェア，インタフェースの実装記述を合成する．

5.1 分割指定

設計対象のソフトウェアとハードウェアへの分割指定は機能単位の粒度で指定可能である．C言語記述の場合，分割指定は図5に示すようにSDFの**SW**,**HW**にそれぞれソフトウェア化する機能単位とハードウェア化する機能単位の名前を指定する．SpecC言語記述の場合は，ハードウェア化するビヘイビアのIDを指定する．IDは記述を解析することにより，ビヘイビアのインスタンスごとに自動的に割り付けられる．

通信プリミティブに関しては，機能単位に指定された分割情報と接続関係より，ソフトウェアとして実装するか，ハードウェアとして実装するか，ソフトウェアとハードウェア間のインタフェースとして実装するかを自動的に判定する．例えば，図5の(a)の場合，CP3はソフトウェアとして，CP4はハードウェアとして実装され，残りCP1，CP5，CP2はインタフェースとして実装されることになる．また，同時にそれぞれの通信プリミティブがアクセス数制約を満たしているかチェックする．

5.2 ソフトウェア

ソフトウェア化する機能単位と通信プリミティブをITRON上のソフトウェアとして動作するよう変換する．変換方法は基本的に機能レベルシミュレーションでの変換と同じである．また，後述するコシミュレーションのため，TOPPERS-Win用のプロジェクトファイルも同時に生成する．

5.3 ハードウェア

ハードウェア化指定された機能単位ごとに動作合成ツールを呼出しハードウェアを合成する．動作合成ツールとしては，eXCiteを用いる．通信に関しては，通信プリミティブとの接続情報から動作合成ツールのサポートする通信インタフェースを割り当てる．ハードウェア部の合成前と合成後の関係を図6に示す．

SysGenは，ハードウェア間の通信を実現するため，通信プリミティブに応じたハードウェアとeXCiteの出力した機能単位ごとのハードウェアモジュールを接続するためのトップモジュールのHDLファイルを生成する．これはeXCiteが通信のためのポートは生成するが（図6中の破線の丸），合成した機能単位間を実際に接続する通信プリミティブ自体の合成は行わないためである．例えばメモリプリミティブならRAMへの接続ポートは合成するが，デュアルポートメモリ自体は生成しないため，SysGenにより生成する．ソ

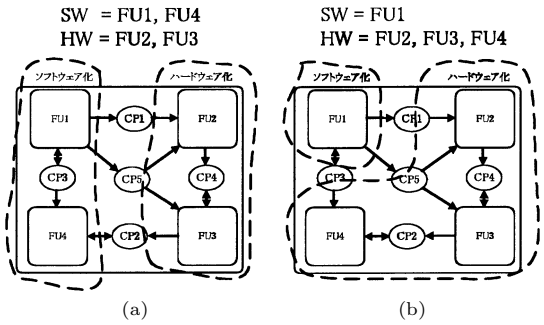

図5 ソフトウェア/ハードウェア化指定
Fig. 5 Specification of software hardware partioning.

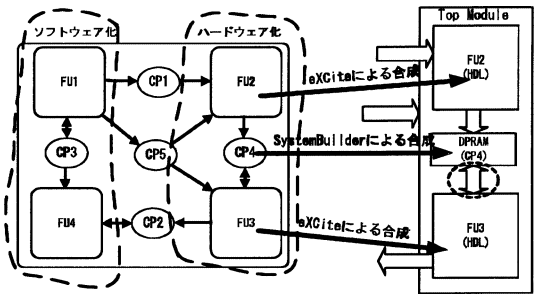

図6 ハードウェア合成
Fig. 6 Hardware synthesis.

フトウェアとハードウェア間のインタフェースとなる通信プリミティブへの接続に関しては，eXCite が生成する通信ポートを直接外部に出す．

5.4 インタフェース

SysGen は，指定された分割方法により，ソフトウェアとハードウェア間のインタフェースとなる通信プリミティブの情報を集める（図 6 の例では CP1，CP2，CP5）．そしてこれらの情報をもとに，インタフェース構成ファイル（IFS）を生成する．IFS は BusConnecter により処理され，インタフェースの実装として，デバイスドライバ，バスインタフェースの HDL ファイルを生成する

5.4.1 デバイスドライバ

各通信プリミティブに応じた ITRON 用のデバイスドライバを合成する．デバイスドライバには，アクセス関数，割込みハンドラ，割込みハンドラとの同期のための同期オブジェクトが含まれる．

手作業による設計では，設計者はハードウェア側のレジスタ構成，アドレス値，割込み番号等を考慮してデバイスドライバを設計する必要がある．一方，本システムではハードウェア構成と整合したソフトウェアを自動合成するため，設計者はソフトウェア・ハードウェア間の通信の実現方法の詳細を考慮する必要がない．

なお，二つのブロック通信プリミティブをインタフェースとした場合の生成されるデバイスドライバのコード量は 262 行である．

5.4.2 バスインタフェース

インタフェースのハードウェア側として，バスに接続するためのグルーロジックと，各通信プリミティブに応じた回路（バッファやメモリ）を生成する．各通信プリミティブごとに生成する回路はハードウェア合成で生成する回路と同等の機能をもつが，片方のインタフェースが eXCite が合成する通信ポートではなく，グルーロジックと接続するようになっている．

グルーロジックとしては，インタフェース化する通信プリミティブの数と種類に応じてアドレスデコーダや割込み管理回路を合成する．グルーロジックは各種のバスに接続可能にするため，VBUS と呼ばれる基本的なバスへのインタフェースをもつ．そのため，実際のバスに接続するためには，VBUS へのインタフェースの先にプロトコル変換回路を接続する．

また，ハードウェアの合成で生成したハードウェアと接続するためのモジュール（図 7 中の CORE）も生成する．図 7 に合成後のハードウェアの構成図を示す．

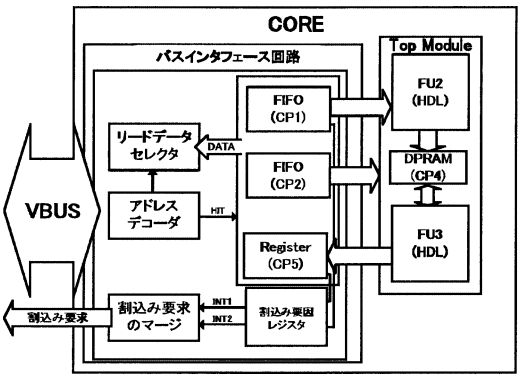

図 7 ハードウェア構成図
Fig. 7 Hardware composition overview.

なお，二つのブロック通信プリミティブをインタフェースとした場合の生成される HDL ファイル（ライブラリファイルは除く）のコード量は 502 行である．

6. コシミュレーション

合成されたソフトウェアとハードウェアをそれぞれ ITRON シミュレータ（TOPPERS-Win）と HDL シミュレータ（ModelSim）で動作させ，それらを協調させることによりコシミュレーションを行う．コシミュレーションの詳細については文献 [14] を参照のこと．

図 8 に示すように，ITRON シミュレータと HDL シミュレータはデバイスマネージャと呼ばれるプログラムにより接続する．デバイスマネージャは ITRON シミュレータと Visual Basic や C 言語で記述された外部アプリケーション（ハードウェアモデル）を Windows のプロセス間通信機能である COM によって接続する．この機能を用いると，ITRON シミュレータ上で，ITRON デバイスドライバ設計仕様で定められたデバイスアクセスインタフェースに従ったアクセスを行うと，デバイスマネージャに接続している外部アプリケーションへのアクセスに変換される．また，外部アプリケーションから ITRON シミュレータに対して割込みを発生させることも可能である．

デバイスマネージャと HDL シミュレータの接続は，ModelSim の Foreign Language Interface（FLI）を用いて，VBUS のプロコルと COM 接続を変換する C 言語記述のモジュール（図 8 中の COMFLI Bridge）により実現している．

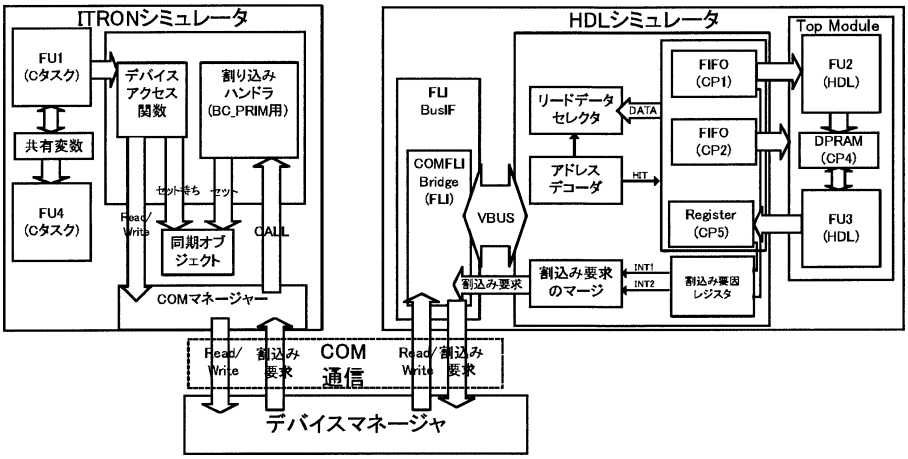

図 8　コシミュレーション
Fig. 8　Co-simulation overview.

COMFLI Bridge は，ITRON シミュレータからデバイスマネージャを経由してリード–ライト要求を受けると，VBUS のバスシーケンスを発生させる．リードの場合は，VBUS からデータを読み込み，デバイスマネージャ経由で値を返す．また，ハードウェア側から割込み要求信号がアサートされると，ITRON シミュレータに割込み要求を出す．

7. FPGA への実装

SysGen は，合成した実装記述を FPGA で動作させるために必要なファイルを生成する．具体的には，ソフトウェア側をコンパイルするためのプロジェクトディレクトリを作成し，その中に TOPPERS-MB 用の Makefile を生成する．ハードウェア側も同様にプロジェクトディレクトリを作成し，EDK 用の MHS ファイルと Makefile を生成する．生成した Makefile を実行することにより，ロードイメージと FPGA のコンフィギュレーションデータが生成される．

FPGA 上のハードウェアシステムの構築には，Xilinx 社の Embedded Development Kit（EDK）を用いる．EDK は，システムで使用するペリフェラルの種類や数またパラメータを記述した Microprocessor Hardware Specification（MHS）と呼ばれるファイルを読込む．そして，MHS に記述された情報からペリフェラルをインスタンス化しそれらをバスで接続する HDL ファイルを生成する．次に，ペリフェラルを構成する HDL ファイルやネットリストとともに論理合成及び配置配線を行い，FPGA のコンフィギュレーションデータを生成する．なお，プロセッサには，FPGA 上で動作するソフトコアの Microblaze を用いる．MHS の一部を図 9 に示す．図 9 では，プロセッサ（microblaze）とタイマ（opb_timer）の宣言とそのパラメータを指定している．

```
BEGIN microblaze
    PARAMETER INSTANCE = mb_pr0
    PARAMETER HW_VER = 2.10.a
    PARAMETER C_USE_DIV = 1
    PORT CLK = clk_54mhz
    PORT INTERRUPT = mb_intr_pr0
    BUS_INTERFACE DLMB = d_lmb_pr0
    BUS_INTERFACE ILMB = i_lmb_pr0
    BUS_INTERFACE DOPB = myopb_bus_pr0
    BUS_INTERFACE IOPB = myopb_bus_pr0
END

BEGIN opb_timer
    PARAMETER INSTANCE = my_timer
    PARAMETER HW_VER = 1.00.b
    PARAMETER C_BASEADDR = 0xffff8000
    PARAMETER C_HIGHADDR = 0xffff80ff
    BUS_INTERFACE SOPB = myopb_bus_pr0
    PORT Interrupt = mytimer_intr
END
...
```

図 9　MHS の例
Fig. 9　An example of MHS.

また，合成したハードウェアをペリフェラルとして EDK で扱うためには，EDK で定められたペリフェラル定義ファイル用意する必要がある．この定義ファイルは実装記述合成時に BusConnecter がバスインタ

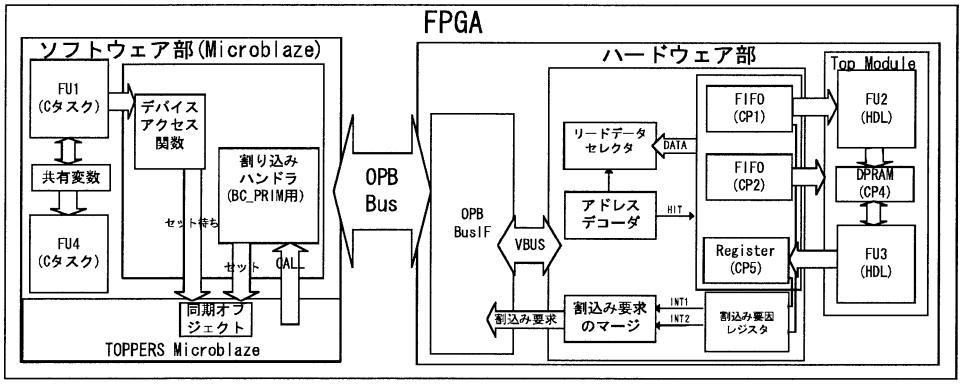

図 10　FPGA 上への実装
Fig. 10　Implementation on the FPGA.

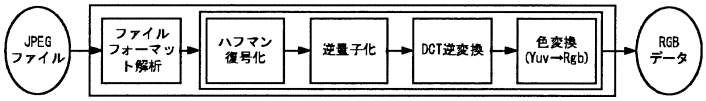

図 11　JPEG デコーダの概要図
Fig. 11　Overview of JPEG decoder.

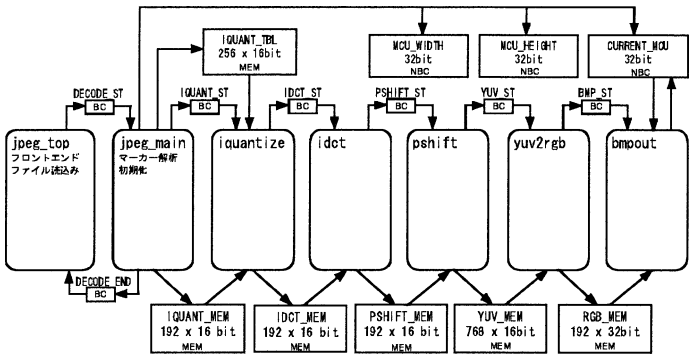

図 12　機能単位と通信プリミティブによる JPEG デコーダの記述
Fig. 12　JPEG decoder with function unit and communication primitive.

フェースとともに生成する．

FPGA に実装した場合のモデルを図 10 に示す．Microblaze システムの標準バスは OnChip Peripheral Bus（OPB）であるため，ハードウェア側には VBUS を OPB に接続するためのプロトコル変換回路を VBUS インタフェースの先に接続する．

8. JPEG デコーダへの適用

本章では，SystemBuilder を用いることによって分割方法を変更した実装が容易に可能であるかを JPEG デコーダの設計を通じて評価する．

8.1 JPEG デコーダの仕様

設計対象とする JPEG デコーダの概要図を図 11 に示す．入力は JFIF 形式のファイルであり，ファイルフォーマットを解析した後，デコード処理を行い RGB データを出力する．なお，デコード処理は JPEG ベースラインに準拠しているが，Minimum Coded Unit（MCU）のサンプリング比は輝度 Y4，色差 Cr，Cb1 で固定，カラー画像のみの制限を課す．

8.2 C 言語 + SDF 記述

入力となる JPEG デコーダの C 言語と SDF によるデザイン記述を図 12 に示す．機能単位は 7 個あり，そのうちの iquantize，idct，pshift，yuv2rgb の 4 個

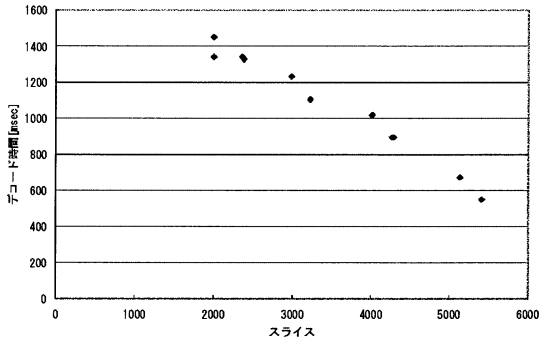

図 13　JPEG デコーダのデコード時間とスライス数
Fig. 13　Decode time and slice of JPEG decoder.

はソフトウェアとハードウェアのどちらにも実装可能である．他の 3 個の機能単位 jpeg_top，jpeg_main，bmpout はソフトウェアとしてのみ実装可能である．

機能単位間でのデコードデータ受渡しは，メモリプリミティブ（図中_MEM の名前が付いている四角）を介して行い，ブロック通信プリミティブ（図中_ST の名前が付いている四角）により同期を行う．なお，それぞれのメモリプリミティブはダブルバッファとなるよう容量を確保している．

8.3　分割方法変更の評価

ソフトウェアとしてもハードウェアとしても実装可能な 4 個の機能単位の実装先の組合せを変更して，SystemBuilder を用いて FPGA に実装し，QVGA（320 × 240）の JPEG 画像のデコードに要した時間と FPGA の使用スライス数（面積）をプロットしたものを図 13 に示す．なお，実装は Xilinx 社の VirtexII 上に行い，動作周波数はプロセッサ，ハードウェアともに 50 MHz である．

JPEG デコーダ記述後，プロセッサ PentiumM1.7，メモリ 1 GByte の WindowsPC を用いて図 13 の結果を得るのに要した時間は 6 時間程度である．処理時間の多くは eXCite による動作合成と EDK による論理合成/配置配線の時間であり，SysGen と BusConnecter による処理時間は 8.5 秒であった．また，設計者が行う作業は，SDF の **SW**，**HW** での指定の変更だけであった．よって，短時間で多くの分割方法による実装及び性能評価が容易に行えることが確認できた．

9. むすび

本論文では組込みシステム開発の効率化を目的としたシステムレベル設計環境 SystemBuilder について，その特徴及び実装技術について述べた．

本システムはインタフェース合成を核として動作合成ツール等を連携させることにより，C 言語若しくは SpecC 言語により記述したデザイン記述に対して分割方法を指定すると，ソフトウェアとハードウェア及びその間インタフェースを自動合成する．また，コシミュレーション環境や FPGA への実装機能ももつ．

また，JPEG デコーダを用いた評価の結果，様々な分割方法で実装した場合の性能を短時間で評価可能であることを確認した．

今後の課題としては，他のハードウェアプラットホーム（バスやプロセッサ）への適用がある．また，デザイン記述を単にソフトウェアとハードウェアに分割するのではなく，ソフトウェアとして実装する部分を複数のプロセッサに割当可能なよう，すなわちマルチプロセッサシステムへの拡張を行う予定である．

謝辞　本システムに開発の一部は，2004 年度 IPA 未踏ソフトウェア創造事業の援助を受けた．プロジェクトマネージャーの中島達夫先生に感謝致します．

文　献

[1] K. Wakabayashi, “C-based behavioral synthesis and verification analysis on industrial design examples,” Proc. Asia and South Pacific Design Automation Conference, pp.344–348, 2004.
[2] C. Sullivan, A. Wilson, and S. Chappell, “Using C based synthesis to bridge the productivity gap,” Proc. Asia and South Pacific Design Automation Conference, pp.349–354, 2004.
[3] 高田広章，“組込みシステム開発技術の現状と展望，” 情処学論，vol.42, no.4, pp.930–938, 2001.
[4] R. Dömer, A. Gerstlauer, and D. Gajski, SpecC Language Reference Manual Version 2.0.
[5] SpecC Reference Compiler, http://www.cecs.uci.edu/~specc/reference/
[6] TOPPERS Project, http://www.toppers.jp/
[7] Y Explorations 社，“eXcite,” http://www.yxi.com/
[8] Xilinx 社，“Embedded Development Kit (EDK),” http://www.xilinx.com/edk/
[9] SystemC 2.0.1 Language Reference Manual Revision 1.0.
[10] InterDesign Technologies 社，“VisualSpec,” http://www.interdesigntech.co.jp/
[11] D. Verkest, K. Van Rommpaey, I. Bolsens, and H. de Man, “CoWare—A design environment for heterogeneous hardware/software systems,” Proc. Design Automations for Embedded Systems, vol.1, no.4, pp.357–386, 1996.
[12] Impulse Accelerated Technologies 社, “CoDeveloper,” http://www.impulsec.com/
[13] 本田晋也，高田広章，中島　浩，“SpecC によるソフトウェ

ア記述の実装記述への変換," 情処学論, vol.44, no.SIG11 (ACS 3), pp.236–246, 2003.
[14] S. Honda, T. Wakabayashi, H. Tomiyama, and H. Takada, "RTOS-centric hardware/software cosimulator for embedded system design," Proc. International Conference on Hardware/Software Codesign and System Synthesis, pp.158–163, 2004.

（平成 16 年 5 月 21 日受付，9 月 16 日再受付）

本田　晋也

2002 豊橋技術科学大学大学院情報工学専攻修士課程了．現在，同大学院電子・情報工学専攻に在学．リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事，修士（工学）．2002 年度情報処理学会論文賞受賞．

冨山　宏之　（正員）

平 11 九州大学大学院システム情報科学研究科博士後期課程了．同年より米国カリフォルニア大学アーバイン校客員研究員．平 13（財）九州システム情報技術研究所研究員．平 15 名古屋大学講師．現在同大学助教授．EDA，コンパイラなどの研究に従事．情報処理学会，ACM，IEEE 各会員．博士（工学）．

高田　広章　（正員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 東京大学大学院理学系研究科情報科学専攻修士課程了．同学科の助手，豊橋技術科学大学情報工学系助教授などを経て，2003 より現職．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術などの研究に従事．ITRON 仕様の標準化活動に，中心的メンバとして参加．博士（理学）．IEEE，ACM，日本ソフトウェア科学会各会員．