

FPGA/CPU 混載システムのための C 言語による協調設計環境の実現

田中康一郎[†] 佐藤 寿倫^{††} 有田五次郎^{†††}

[†] 九州工業大学マイクロ化総合技術センター

〒 820-8502 福岡県飯塚市大字川津 680-4

^{††} 九州工業大学知能情報工学科

〒 820-8502 福岡県飯塚市大字川津 680-4

^{†††} 九州産業大学知能情報工学科

〒 813-8503 福岡県福岡市東区香椎松香台 2-3-1

E-mail: [†]tanaka@cms.kyutech.ac.jp, ^{††}tsato@ai.kyutech.ac.jp, ^{†††}arita@kyusan-u.ac.jp

あらまし 製品のライフサイクルの長期化を推進するために、近年の組込みシステムには高い柔軟性が要求されている。この解決策の一つとして組込み向けプロセッサを用いる方法が多用されているが、その構成が処理性能的に充分でない場合には、ハードウェアデバイスを追加する方法が考えられる。しかしこの方法ではプログラミングとハードウェアデザインの二つの異なる設計フローでアプリケーションを開発する必要があり、プロセッサベースで設計する方法などと比較して、TTM (Time to Market) が大幅に長期化することが予想される。本稿では、上記の問題を解決するために構築した、C 言語ベースのシステム設計環境について紹介する。本環境は商用の EDA ツール群と我々の開発したライブラリを統合化することで実現されており、プログラミング言語で協調設計を行える。この環境を利用して、構成の大きく異なる二つのハードウェアプラットフォームへ信号処理アプリケーションを実装した結果、従来方法と比較して短期間でシステムを設計できることが確認できた。

キーワード 協調設計, C 言語設計, SpecC 言語, FPGA, DSP

Development of a C-Based Co-Design Environment for an FPGA/CPU Platform

Koichiro TANAKA[†], Toshinori SATO^{††}, and Itsujiro ARITA^{†††}

[†] Center for Microelectronic Systems, Kyushu Institute of Technology

680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

^{††} Department of Artificial Intelligence, Kyushu Institute of Technology

680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

^{†††} Department of Intelligence Informatics, Kyushu Sangyo University

2-3-1 Shokodai, Higashi-ku, Fukuoka-shi, Fukuoka 813-8503, Japan

E-mail: [†]tanaka@cms.kyutech.ac.jp, ^{††}tsato@ai.kyutech.ac.jp, ^{†††}arita@kyusan-u.ac.jp

Abstract This paper presents a co-design environment for embedded systems, which consist of both embedded processors and programmable devices. To assure long life cycle of products, the latest embedded systems are required high flexibility. Embedded processors are utilized very well to solve the problems. When a processor-based system is not equipped with processing performance sufficient as hardware platform of target products, programmable devices such as FPGA are also integrated simultaneously. However, development of their system is very complex to describe both a programming language and a hardware description language as compared with processor-based systems. Our co-design environment, which consists of commercial-based EDA tools and our libraries, allows application designers to develop FPGA/CPU based systems in C-programming language. As a case study, an image processing application is implemented in two kinds of hardware platforms on the environment. As the result, we have confirmed that the environment enables us to develop systems in much shorter period than traditional ones do.

Key words Co-design, C-based design, SpecC, FPGA, DSP

1. はじめに

現在、組込みシステムに柔軟性の高さが求められている。柔軟性の高いシステムは製品出荷後の新機能の追加やバージョンアップ、バグフィックスなどを容易にし、その結果、製品の短命化を抑制するためである。この方法はデジタルTV、HDDレコーダ、ブロードバンドルータといったデジタル家電と総称される製品で採用されており、その製品の大半はプロセッサを基本としたソフトウェアシステムである。しかし今後さらに高い処理性能が要求されるようになると、全ての修正変更点をソフトウェア処理で対応することが難しくなることは明白である。この問題を解決できるものに、プログラマビリティを備えたハードウェアとの協調処理システムがある。

協調処理システムで問題となるのは、Time to Market (TTM) である。協調処理システムでは一つのアプリケーションに対して抽象度の異なるプログラミングとハードウェアデザインの二つが必要となり、ソフトウェアシステムと比較してTTMが長期化する傾向がある。そこでその解決策として、従来の設計フローへのシステムレベル設計の導入が検討される。システムレベル設計ではプログラミング言語に類似した言語でソフトウェアとハードウェアを同時に記述できるため、ソフトウェアシステムに近いTTMが実現できると考えられている。しかしそれらの既存設計環境では、目的とするアプリケーション以外に、そのアプリケーションを検証するためのテストベンチの必要性やシミュレーション終了後から実装までいくつかの工程が必要である。このようにプログラミング記述以外の作業に時間を費やされるため、全体としてTTMの短期化につながらない結果に陥っている。

本稿では上記の問題の解決を目的とした統合的なシステムレベル設計環境の検討と構築を行った。まず2章では関連研究について紹介する。次に3章ではシステムレベル設計環境に関する要求仕様と方針について述べ、4章で方針に沿って開発した設計環境の詳細について示す。さらに5章では本設計環境を用いた実装事例を示し、6章でまとめと今後の展開について言及する。

2. 関連研究

2.1 SpecC テクノロジー

SpecC言語は、ANSI-C言語に対してハードウェアの動作を記述できるよう拡張したシステムレベル設計言語の一つである[1]。拡張の基となる言語としてC言語が選択された理由は、C言語がソフトウェア開発の分野で非常に普及しており、既に膨大なライブラリがコード化されていることである。拡張された構文の特徴的なものとしては、ブーリアン型(bool)、ビット型(bit[msb:lsb])、タイム型(waitfor())、並行実行(par{})、パイプライン実行(pipe{})、同期操作(event, wait(), notify())などがある。

また、SpecC言語のもう一つの特長として、動作と通信を分離して記述できる点が挙げられる[2]。SpecCプログラムモデルは、ビヘイビア(動作)、チャネル、およびインタフェースの3種類で構成される。ビヘイビアは仕様にしたがった動作を記述する部分であり、階層化することも可能である。

SpecCテクノロジーを利用した研究では、SpecCの言語仕様に関する研究[3][4][5]と開発環境に関する研究に大別できる。本研究の関連する後者には、デバイスドライバとデバイスの一体記述を可能とした

開発環境[6][7]やUSBドライバを対象としたシステム設計[8]などがある。上記研究では、ハードウェアプラットフォーム(以下プラットフォーム)に依存するデバイスドライバをアプリケーションの設計対象とすることで設計容易化を狙っている。一方、我々はアプリケーション開発を円滑に進めるために、アプリケーション開発とデバイスドライバ開発を混同しないことでTTMに優れた設計環境を実現することを目標としている。

2.2 SpecC 言語ベース EDA ツール

現在、SpecC言語をベースとしたEDAツールは2つある。一つはカリフォルニア・バークレー・アーバイン校で開発されたSpecC Reference Compiler (SCRC)である[9]。SCRCでは、SpecC言語で記述されたデザインのシミュレーションを行うことができる。もう一つは、Interdesign Technologies社のVisualSpec[10]である。VisualSpecも基本的にはシミュレーションを行うツールではあるが、後述するeX-Citeと連携することでVHDLのRTL記述を出力することができる。

SCRCはコンパイラであるため、それを利用してシミュレーションを行う際、エディタを用いてSpecC言語を記述する必要がある。一方、VisualSpecはANSI-C言語から拡張された多くの部分をグラフィカルに入力することができる。これにより、C言語の知識のあるプログラマによるハードウェア設計を可能としている。しかしグラフィカル入力による作業は、非常に時間を要するだけでなくハードウェアの知識も必要となる。

2.3 C言語ベース高位合成ツール

C言語をベースとした高位合成ツールには、RMAC-V[11]、Dash Compiler[12]、SPARK[13]、およびeXCite[14]などが挙げられる。RMAC-Vは、FPGAとメモリから構成される汎用エンジンであるRM-Vを対象とした高位合成システムであり、ユーザはANSI-C言語のサブセットによって設計入力が行える。Dash Compilerは、C言語ライクなコードからソフトウェア、ハードウェア、およびプロセッサモデルのHandel-C Compiler[15]向けコードを生成する。そのコードから、FPGAのためのプロセッサ、プログラム、およびカスタムハードウェア向けコードを生成することで、FPGA上にハードウェア/ソフトウェア混在のシステムを実現する。SPARKはANSI-C言語を対象とした高位合成ツールであり、VHDLのRTL記述を出力することができる。eXCiteはY Explorations社が開発した高位合成ツールであり、SPARKと同様にANSI-C言語からVHDLのRTL記述を出力できる。

その他C言語をベースとした高位合成ツールとして、SystemC言語による設計ツールが考えられる。SystemC言語は時間の概念を持たないUTF(Un-timed Function)記述からRTL記述までを幅広い記述レベルを提供するためシステムレベル・シミュレーションとしては非常に有効である。しかしUTF記述からHDLのRTL記述への高位合成ツールが提供されていない[16]、[17]ため、現状では一般的なプログラムをハードウェアすることはできない。

3. 協調設計環境の構築方針

3.1 システム構成

協調設計環境を構築するにあたり、ハードウェア(HW)とソフトウェア(SW)が動作するプラットフォームのシステム構成を決定する必要がある。HW

とSWが互いにデータ交換するシステムを分類する場合、全てのシステムは、メモリを共有して利用する方法、ソフトウェアがハードウェアをメモリとみなしてアクセスする方法、その混在に大別できる。

それぞれの構成例を図1に示す。(a)は前者の方法であり、お互いがメモリにアクセスすることでデータの交換を行う。(b)は後者の方法であり、SWはHWをアクセラレータとして利用できる。(c)は(b)と同様に後者の方法であるが、HWがメモリを管理している点異なる。

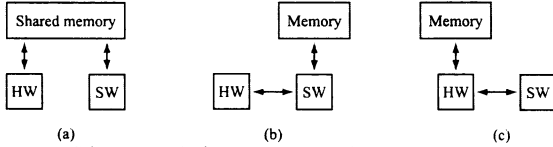


図1 ハードウェアとソフトウェアによるシステムの構成例

ここで、それぞれの特長について考察してみる。(a)は共有メモリのマルチプロセッサモデルと同等ともみなせる構成であり、HWとSWが共に同様のメモリアクセスを行えるため既存プログラムとの親和性が高い。(b)はHWをSWのアクセラレータとみなすことができ、HWとSWがメモリを介することなくデータ交換できることから、(a)と比較して頻繁にデータ交換する細粒度通信を含むアプリケーションに向いている。(c)は(b)と同様に細粒度な通信に効果的であるという特長を有するが、さらにメモリがHWと接続しているため、HWの並列性を効果的に利用できる。加えてSWとメモリ間にHWを挿入できるため、従来SWで行う必要があるSWのアドレス修飾やデータ加工をHWで肩代りすることができ、その結果SWの処理を軽減できる[18]。

このうち、我々が有効と考える構成は、(a)と(c)の組合せである。この2つを同時にサポートできれば、(b)もその中に包含することができ、図1中の全構成に対応することができる。ただし(a)と(c)は、上述のように通信の粒度が異なるため、細粒度向きの(c)に関してはHW/SW分割を自動化することで対応する[18]こととし、(a)に関してはプログラム自身によるHW/SW分割で対応することとした。以下本稿では、(a)を対象として言及する。

最後に、このシステムを検証するための制御機構について検討する。上記のようなシステムの多くは、スタンドアロン動作、または汎用PCへ組み込まれて利用されたと想定できる。さらに、前者の場合においても、システムに汎用PC向けのインタフェースが用意されていることが多く、それらを用いてシステム検証が行われている。そこで、本環境ではPCによる制御を前提とした。

3.2 要求仕様

以下に我々が考える理想的な設計フローを述べる。本環境では、製品開発におけるTTMを重要視している。つまり、迅速な設計および実装が行えることが要求される。システムを構築する場合、システムの一部であるモジュールの設計対象は、プラットフォームに依存するものと依存しないものに大別できる。これらが混在している場合、モジュール間の冗長な処理を省くことができ、性能的には効果的である。しかし、設計が複雑になるため設計期間が増加するだけでなく、再利用性も低くなる。そこで、本環境の設計フローでは、プラットフォームに依存する処理と依存しない処理を明確に区別し、依存しないモジュール

の設計者と依存しないモジュールの設計者が互いのモジュールに影響を及ぼさないこと、さらに、依存しないモジュールの設計者は、設計入力終了後、ツールの単純な操作だけで実装できることを目標とした。

3.3 方針

3.1節のシステム構成と3.2節からの要求仕様を反映させるために、本環境は次に述べる方針で実現することとした。プラットフォームに依存するモジュールの設計者と依存しないモジュールの設計者を区別して説明する。本稿では、前者をシステムデザイナー、後者をアプリケーションデザイナーと定義する。

本環境に用いるEDAツールには、2.2節で紹介したVisualSpecと2.3節で紹介したeXCiteを利用することとした。このVisualSpecではシステムデザイナーがプラットフォーム依存する領域を事前に設計することで、アプリケーションデザイナーに対して市販のCコンパイラと類似したエディタ主体の設計環境を提供できる。一方高位合成ツールに関しては、2.3節で紹介したRMAC-VとDash Compilerはターゲットとなるプラットフォームが明確であるため、本研究では対象としない。残りのSPARKとeXCiteは同等の機能を有すが、本研究ではVisualSpecとの親和性が高いeXCiteを利用することとした。

次に、3.1節において設計入力時にシステム構成が単一に固定したため、VisualSpec向けのテンプレートを用意することとした。テンプレートを用意することで、VisualSpecから生成されるコードのインタフェースを一意に決定でき、システムデザイナーが要求する各デバイスへのインタフェース名なども固定することが可能である。これにより、システムを新規に開発する際、アプリケーションデザイナーとシステムデザイナーが並行して設計作業を行うことができる。

さらに、VisualSpecから生成されるコードを、次処理であるコンパイラでできるようにトランスレータを用意することとした。2.2節で述べたように、VisualSpecではデザイナーにより入力されるコードはANSI-C言語であり、そのコードはANSI-C準拠のコンパイラでコンパイルすることができる。ただし、APIの利用やライブラリコードとの接続のために規則的なコードの変換をする必要があるため、それらに対応するためのトランスレータが必要となる。これにより、アプリケーションデザイナーは設計入力後にコードに手を加える必要がなくなる。

最後にライブラリについて述べる。ライブラリはハードウェアに密接に依存するため、プラットフォーム依存であるといえる。異なるハードウェアプラットフォームに対応できるライブラリを作成することはできない。したがって、本環境の仕様書に準拠してシステムデザイナーが開発することとした。

4. 協調設計環境の設計

4.1 VisualSpec テンプレート

図2に今回開発したVisualSpecテンプレートの構成を示す。VisualSpecはSpecC言語を基本に開発されたツールであるため、2.1節で紹介したように、主要な処理はビヘイビアで記述し、その間の通信プロトコルの定義はチャンネルで行う。以下、この構成の概要について述べる[19]。

このテンプレートではビヘイビアとして、ハードウェアビヘイビア(HW)、ソフトウェアビヘイビア(SW)、制御ビヘイビア(CONTROL)、メモリビヘイビア(MEMORY)を用意した。このうち、実際にアプリケーションデザイナーによって設計されるビヘイビアは、HW、SW、CONTROLである。

一方、チャンネルはおのおののビヘイビアを接続す

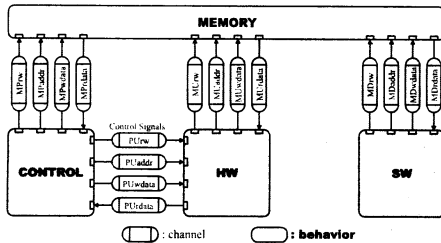


図2 汎用 VisualSpec テンプレート

る。MEMORYは共有メモリとして扱われるため、全てのビヘイビアと接続される。SWは割り込みを対象としていないため、CONTROLとSW間にチャンネルは用意していない。一方、HWの場合はシステムによってはHWを制御するために使用する可能性があるため、CONTROLとHW間にチャンネルを用意した。チャンネルの種類としては、表1に示す標準的なメモリアクセスで用いるものを用意した。データタイプがcharとlongしか使われていないが、これはVisualSpecの仕様によるものである。

表1 チャンネルのリスト

Channel	Data type	Function
rw	char	Read/Write
addr	long	Address
wdata	long	Data bus for writting
rdata	long	Data bus for reading

上記のテンプレートの構成は、HWとSW間のコードの移動の容易性についても配慮している。HWのコード記述例を図3に示す。双方のビヘイビアでは、ほぼ同等のSpecCコードを利用できる。変更を必要とする点は、インタフェース名とCONTROLへの対応である。前者はMUをMDに変更するなどの規則的な変換作業で対応できる。後者の用途はデバイスの起動であるため、移動は不要である。この結果、HWからSWへ、あるいはその逆を容易に行うことができる。

```
void main( void ) {
    // Interface for CONTROL behavior
    char PUrw;
    long PUaddr, PUwdata, PURdata;
    // Interface for MEMORY behavior
    char MUrw;
    long MUaddr, MUwdata, MURdata;

    // Variable declaration
    .
    .
    .

    // Booting
    PUrw = PUrw_r.blockRead();
    PUaddr = PUaddr_r.blockRead();
    PUwdata = PUwdata_r.blockRead();

    // for polling
    do {
        MUrw = READ;
        MUaddr = START_ADDR;
        MUrw_s.blockWrite( MUrw );
        MUaddr_s.blockWrite( MUaddr );
        MURdata = MURdata_r.blockRead();
        START = (unsigned long) MURdata;
    } while (START != HW_START);
    ... // User program
}
```

図3 HWのコード記述例

4.2 トランスレータ

トランスレータ[20]はVisualSpec上で設計されたSpecCコードをターゲット向けのCコードに変換するために利用する。このトランスレータでは、テクノロジファイルを用意することで、複数の環境に対応することができる。このテクノロジファイルには、インクルードファイル、グローバル変数、コンストラクタ（資源確保、初期化）、デストラクタ（資源

解放）を定義することができる。またトランスレータでは、テクノロジファイルで定義された情報以外にも、メモリアクセス関数の変更が自動的に行われる。具体的にはCONTROL向けのトランスレータではAPI用の関数に置き換えられる一方、SW向けのトランスレータでは関数からポインタによるメモリアクセスに置き換えられる。CONTROLとSWで変換方法に違いがある理由は、組込みプロセスでは関数によるメモリアクセスが性能低下の要因になることがあるためである。このトランスレータが用意されることで、アプリケーションデザイナーは手作業によるコードの修正から解放される。

5. 実装事例

本環境を利用してプラットフォームにアプリケーションを構築した。実装評価にあたり、本環境の適応性を評価するために、システムアーキテクチャが大きく異なる2つのプラットフォームを採用した。一つは本学で開発したFPGA/DSP搭載PCIカードであるRYUOH[21]、もうひとつはプロセッサを実装したFPGAであるXilinx社のVirtex-II Pro[22]である。以下にその概要を述べる。

5.1 RYUOH

RYUOHはFPGAとDSPを実装したPCIカードである。RYUOHでは主要なデバイスであるDSP、PCIコントローラ、メモリは全てFPGAと接続されている。そのため、FPGAにそれらを制御するためのロジックが必要となる。本環境におけるSW、HW、CONTROLは、それぞれDSP、FPGA、PC（PCIバス経由）となる。なおRYUOHにはDSP用のローカルメモリが用意されているが、今回は使用していない。

我々の開発したRYUOH用FPGAライブラリの構成を図4に示す。FPGAにはアプリケーションデザイナーが設計したUser Logicが実装されるが、それ以外にもDSP、PCIコントローラ、メモリを制御する回路とその間の通信を制御する調停回路（Arbiter）が実装される。この調停回路では各デバイスからのメモリに対する要求を調停する。この調停回路に対するマスタはDSP、PC、User Logicであるため、アプリケーション実行時に外部PCから自由にメモリをモニタすることができる。

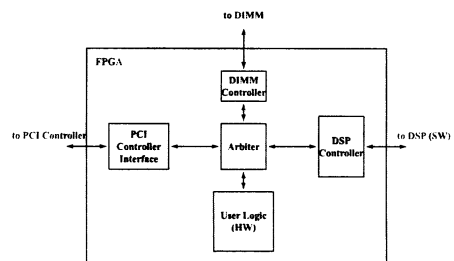


図4 RYUOH上のFPGAの内部構成

5.2 Virtex-II Pro

Virtex-II ProはXilinxが開発したエンベデッド・プロセッサを実装したFPGAである。今回のこのFPGAを図5に示す構成で用いた。このプロセッサは、Processor Local Bus (PLB)とOn-chip Processor Bus (OPB)で構成されるCoreConnect Busを利用することで、メモリ、ペリフェラル、ユーザロジックにアクセスできる。またそれらに容易にアクセスできるように多くのラップが用意されている。今回の構成で用いたラップは、プロセッサの命令コードを保持す

るための BRAM Interface Controller, 外部メモリとアクセスするための OPB Synchronous DRAM Controller, 外部 PC とアクセスするための OPB UART lite, ユーザロジックとアクセスするための OPB User Code Template Master Service Package, バス間を接続する PLB2OPB Bridge である。

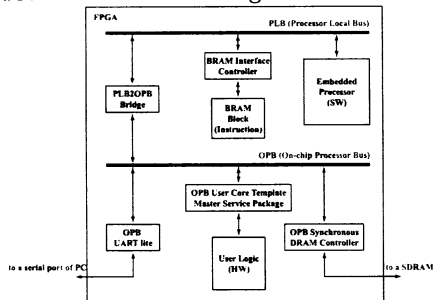


図5 Virtex-II Proの内部構成

本事例では Virtex-II Pro に対して次のように機能を割当てた。SW は Embedded Processor で実行する。ただし、今回は実用性を考慮して、命令コードは内部メモリである BRAM に割当て、データは外部の SDRAM に割当てた。次に HW は FPGA 内の User Logic で実行される。User Logic 用のラッパにはバスマスタ用とスレーブ用が用意されているが、今回は要求を満たすバスマスタ用を選択している。最後に CONTROL は、今回使用したプラットフォーム (MEMEC DS-BD-2VP7-FG456 REV3) [23] の仕様から、RS232C を経由した PC とした。ただし、RS232C 用のラッパである OPB UART lite はバスマスタとして動作することができない。つまり、RYUOH のように能動的に PC からメモリを参照することができない。そこで、今回は整備されたラッパで容易に利用できる Embedded Processor を利用して PC と SDRAM 間の通信を調停した。

5.3 設計フローの比較

図6に RYUOH における設計フローを示す。まず VisualSpec を用いて設計入力を行い、次に全体のシミュレーションをする。その後 FPGA, DSP, PC 用

のデータをおのおの生成する。CONTROL ではトランスレータである sc2win.pl と pci.tec というテクノロジーファイルを利用して、Microsoft 社の Visual C++ で利用できるコードに変換する。その後、RYUOH 用の API を用いて実行コードを生成する。HW では VisualSpec から eXCite 専用の C コードを生成する。eXCite ではそのコードを使って VHDL の RTL 記述を生成する。その VHDL から XST を用いてネットリストを生成した後、RYUOH 用の FPGA ライブラリと ISE を用いて実装データを作成する。SW ではトランスレータである sc2cpu.pl と c6x.tec というテクノロジーファイルを利用して、TI 社の Code Composer Studio (CCS) と称すコンパイラで利用できるコードに変換する。そのコードを利用して DSP 用の実行コードを生成する。このうちアプリケーションデザイナーがコードに手を加えるのは、図6中のグレーで示された VisualSpec への設計入力時である。他方システムデザイナーが開発するものは、図6中の黒で示されたトランスレータ向けのテクノロジーファイルとライブラリ群である。なお、これらはプラットフォームに変更がない限り、アプリケーション開発による修正は不要である。

次に、RYUOH における設計フローと図7に示す Virtex-II Pro における設計フローを比較する。図中の大きな違いとして、SW の実装時が挙げられる。RYUOH では DSP は FPGA の外部にあるため単体のバイナリコードが用意できれば良いが、今回の Virtex-II Pro 上に実装したモデルでは命令コードを FPGA 内部に保持する必要があるため、ISE で FPGA の実装データとマージする必要がある。それらの工程には、Xilinx 社の ISE Embedded Development Kit (EDK) を用いた。この EDK には、FPGA 内のプロセッサ用のコンパイラと共に XST や ISE を操作するツールが用意されており、EDK からそれらを操作することで実装データを生成できる。その他の工程に大きな違いはない。設計入力時に用いた VisualSpec のテンプレートはプラットフォームの構成が大きく違うにも関わらず、同一のものが利用できた。トランスレータに関しては、テクノロジーファイルを変更することで共通のものが利用できた。

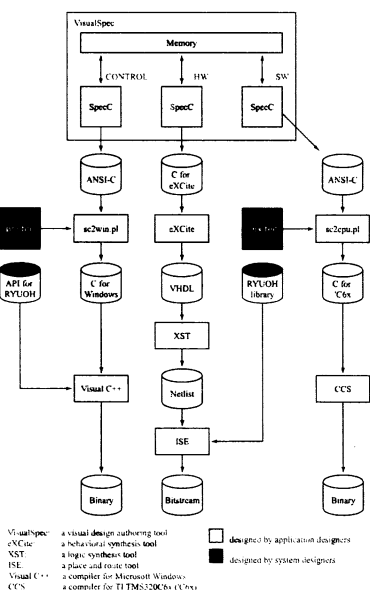


図6 RYUOH における設計フロー

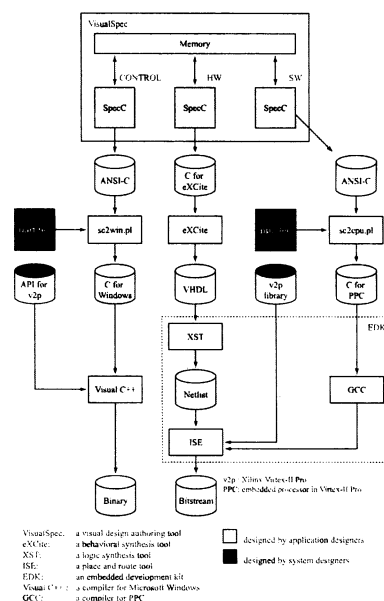


図7 Virtex-II Pro における設計フロー

5.4 設計環境の評価

上記設計環境を評価するために、画像処理アプリケーションを設計し実装した。このアプリケーションは、RGB フォーマットの画像を GBR フォーマットや BRG フォーマットに変換する。元画像は制御用の PC にファイルとして保存されており、データをメモリに転送後、処理デバイスとして FPGA、CPU、または両方のいずれかを選択する。両方を選択した場合、双方の処理デバイスは同時に処理を開始する。処理終了後は処理結果がメモリに書き込まれているため、制御用 PC はメモリから結果を取得し、それをファイルに保存する。

このアプリケーションを VisualSpec を用いて設計し、5.3 節の設計フローにしたがって実装を行った。その結果、設計した一つのアプリケーションがアプリケーションデザイナーによって途中でコードに修正を加えることもなく、二つの異なるプラットフォームで正常に動作することが確認できた。表 2 に実装結果を示す。

表 2 実装結果

	RYUOH	Virex-II Pro
FPGA	XCV1000E-6	XC2VP7-6
CPU	TMS320C6701	PowerPC 405
Code Size (Control)	48KB	52KB
Code Size (SW)	11964B	11136B
4 Input LUT	4,907	3,691
Flip Flops	2,276	2,681
Block RAMs	1	9
MULT18x18s	—	3
Maximum Frequency	35.272MHz	55.009MHz

6. ま と め

本稿では FPGA/CPU をそれぞれ一つずつ利用したプラットフォームに対する協調設計環境を紹介した。RYUOH と Virtex-II Pro といったプラットフォームの構成が異なる 2 つに対して実装した結果、一つの共有できる VisualSpec テンプレートを利用することで、同等の設計フローで設計できることが確認できた。また紙面の都合でその詳細は割愛するが、本環境を利用した設計セミナーの結果から、本環境は従来設計環境と比較して短期間にかつ容易に協調設計が行えることも確認できた。

今後の研究課題として次のことが挙げられる。まず 3.1 節で示した (c) への対応が挙げられる。我々は (a) と (c) における協調処理の粒度は異なっていると考え、(c) は (a) と比較してより細粒度な協調処理向きであると考え、現在 C プログラムからの自動生成を念頭においた方法について検討を行っている。次にハードウェアにおける実装面積制約問題が挙げられる。この問題を抑制するには、動的に構成を変更できるハードウェアの応用が効果的だと考えられる。現在そのような機構を備えたプラットフォームを開発 [24] しており、今後それらに対応できる設計環境の検討 [25] を進めていく予定である。

文 献

- [1] SpecC Technology Open Consortium: *SpecC Language Reference Manual Version 2.0* (2002).
- [2] 木下常雄, 富山宏之: SpecC 仕様記述言語と方法論, CQ 出版社 (2000).
- [3] Fujita, M. and Namakura, H.: The standard SpecC Language, *Proceedings of 15th International Symposium on System Synthesis*, pp. 81 – 86 (2001).
- [4] Tomiyama, H., Cao, Y. and Murakami, K.: Modeling Fixed-Priority Preemptive Multi-Task Systems

in SpecC, *Proceedings of IEEE The 10th Workshop on System And System Integration of Mixed Technologies*, pp. 93 – 100 (2001).

- [5] 松本吉弘: 拡張 SpecC によるソフトウェアアーキテクチャ記述, 組み込みシステム技術に関するサマワークシヨップ予稿集, pp. 71 – 82 (2001).
- [6] 本田晋也, 富山宏之, 高田広章: SpecC 記述のソフトウェア実装記述への変換とインタフェース生成, DA シンポジウム 2003 論文集, pp. 295 – 300 (2003).
- [7] 本田晋也, 高田広章: デバイスドライバとデバイスの一体設計手法への SpecC の適用性評価, 情報処理学会論文誌, Vol. 43, No. 5, pp. 1214 – 1224 (2002).
- [8] Katayama, T.: A Study of Co-design for Systems with Interfaces of USB devices in SpecC, *Proceedings of 7th World Multiconference on Systemics Cybernetics and Informatics*, Vol. 7, pp. 234 – 240 (2003).
- [9] Gajski, D. D., Zhu, J., Domer, R., Gerstlauer, A. and Zhao, S.: *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers (2000).
- [10] Interdesign Technologies: <http://www.interdesigntech.co.jp/>.
- [11] 室屋友和, 橋本匡史, 高林宏忠, 黒木修隆, 沼昌宏: 汎用エンジン RM-V を対象とする高位合成システム, 情報研報 (2000-SLDM-96), pp. 41 – 47 (2000).
- [12] Saul, J.: Hardware/Software Codesign for FPGA-Based Systems, *Proceedings of the 32nd Hawaii International Conference on System Sciences* (1999).
- [13] Gupta, S., Dutt, N., Gupta, R. and Nicolau, A.: SPARK: A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations, *International Conference on VLSI Design* (2003).
- [14] Y Explorations, Inc.: <http://www.yxi.com/>.
- [15] Page, I. and Newman, M.: Hardware Compilation Techniques for Real-Time Systems, 情報研報 (2002-SLDM-105), pp. 1 – 8 (2002).
- [16] Summit Design, Inc.: System Design トレーニング資料 (2004).
- [17] Synopsys, Inc.: *SystemC Compiler Behavioral User and Modeling Guide* (2003).
- [18] 橋本耕太郎, 林悠平, 田中康一郎, 佐藤寿倫, 有田五次郎: FPGA による DSP 向け多機能メモリの実現, 情報研報 (2003-SLDM-111), pp. 45 – 50 (2003).
- [19] Hayashi, Y., Tanaka, K., Sato, T. and Arita, I.: Design of Hardware/Software Co-Design Environment Using SpecC-Based Tools, *Proceedings of 18th International Technical Conference on Circuits/Systems, Computers and Communications*, pp. 1599 – 1602 (2003).
- [20] 田中康一郎, 岩谷祐一, 林悠平, 佐藤寿倫, 有田五次郎: SpecC による協調設計環境の構築, 情報処理学会 DA シンポジウム 2003, pp. 157 – 162 (2003).
- [21] Tanaka, K., Iwaya, Y., Hayashi, Y., Sato, T. and Arita, I.: Design and Implementation of FPGA/DSP Based PCI Card, *Proceedings of 15th International Conference on Systems Engineering (ICSEng)*, pp. 19 – 24 (2002).
- [22] Xilinx, Inc.: *Virtex-II Pro Platform FPGAs: Complete Data Sheet* (2004).
- [23] Memec, Inc.: *Virtex-II Pro (P4/P7) Development Board User's Guide* (2003).
- [24] 田中康一郎, 江島俊朗: 高度画像処理 LSI とその応用, 第 4 回計測自動制御学会制御部門大会予稿集, pp. 731 – 734 (2004).
- [25] 林悠平, 田中康一郎, 佐藤寿倫: SpecC 言語を用いた協調設計環境における動的部分再構成デバイスへの適用, 第 2 回リコンフィギャラブルシステム研究会, pp. 3 – 9 (2003).