組み込みシステムの開発環境整備に関する研究（第２報）

平川　寛之・清水　章良・宮本　博永

Research on improvement of developing environment for Embeded Systems(2nd report)

Hiroyuki HIRAKAWA, Akio SHIMIZU and Hironaga MIYAMOTO

要　約

近年の組み込みシステムは，高性能化，高機能化の一途にあり，その開発サイクルも短くなる一方である．これらに対応するため，組み込みシステムにおいても OS（オペレーティングシステム）を導入し，既存のソフトウエア部品の再利用を図るなど，様々な取り組みが行われているが，システムの複雑化やハードウエア・ソフトウエア設計者間の情報共有不足などによるトラブルも多く発生している．

このような状況を改善するため，ハードウエア・ソフトウエアを同一レベルで扱うことで設計者間の情報の共有を促し，効率的な設計が可能な環境を提案する．また，オープンソースの組み込み OS として実績のある Toppers/JSP[1)] を対象にデバイスドライバ等の自動合成について検討を行ったので報告する．

1. 緒　　言

携帯電話や情報家電等に代表される組み込みシステムは，近年，高性能化，高機能化が進められており，その開発サイクルも短くなる一方である．これらに対応するため，組み込みシステムにおいても OS（オペレーティングシステム）を導入するなど，様々な取り組みが行われている．

開発対象となる組み込みシステムは，複雑化する一方であり，各機能モジュールや，ハードウエア・ソフトウエア間のインターフェースの仕様の不備などがバグの発生原因となることも多い．一般に，ハードウエアとソフトウエアは，別々の行程で開発されることが多く，設計情報も一元的に管理されていないこともあるため，両者の認識の違いによって不具合が発生することも多い．

このような現状を改善するため，近年，広く用いられるようになった，FPGA（書き換え可能な論理素子）を用いたシステムを対象に，ハードウエア・ソフトウエアを同一環境下で開発が可能なシステムの構築を行った．

また，組み込み OS として広く用いられている Toppers/JSP を対象に，デバイスドライバの自動生成についても，検討をおこなった．

2. 開発した環境について

近年，ハードウエアは一見してプログラミング言語のように見える，ハードウエア記述言語を用いて開発される例が多くなっている．この点を利用して，開発環境を構築することとした．

ハードウエアとソフトウエアを同一環境で開発する方法として，C 言語で書かれたソースプログラム（ソフトウエア）にハードウエア記述言語の一つである VHDL 言語で記述したハードウエアを埋め込む手法を提案する．

本環境下では，ハードウエアは C 言語の関数として扱うことが可能となる．

```
/* ハードウエア関数 */
hdl_inline char test_func()
{
    architecture RTL of test_func is
    signal temp : std_logic_vector(31 downto 0);
    begin

        process (clk)
        begin
            if (rising_edge(clk) and clk = '1') then
            .....
}

/* C言語関数 */
int main()
{
    printf("%c¥n", test_func());
}
```

図１　C 言語，VHDL の混在ソース

具体例を図 1 に示す．

2−1　ハードウエア関数の定義

VHDL 言語で記述したハードウエア（ハードウエア関数）を C 言語の関数として扱うためには，相互のインターフェースを明示的に定義する必要がある．今回は，新たに hdl_inline プレフィックスを定義し，関数宣言の頭に，このプレフィックスがある場合は，ハードウエア関数として処理することとした（図 2）．

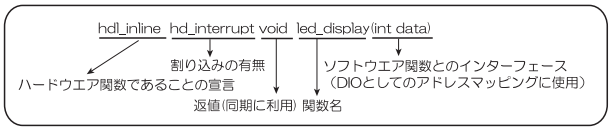

図 2　ハードウエア関数の定義

2−2　データの受け渡し方法

ハードウエア関数とソフトウエア関数間でデータの受け渡しを行う方法として，

・ポーリング（ハードウエア関数定義の引数部分）

・割り込み

以上の，2 つの方法を提供する．データ形式については，通常の C 言語の変数定義によるもの（例えば int, char など）とし,ハードウエアーソフトウエア間で，データ幅に齟齬がある場合は，LSB 側に一致させるものとする．void を指定した場合は，上記の受け渡し方法は使用しない．

ハードウエア関数と CPU との接続については，今回，WishBone[2]バスを用いた共有バス方式により行うこととした（図 3）．

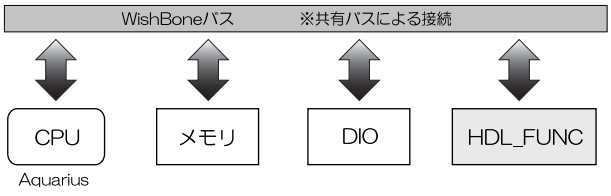

図 3　WishBone バスによる接続

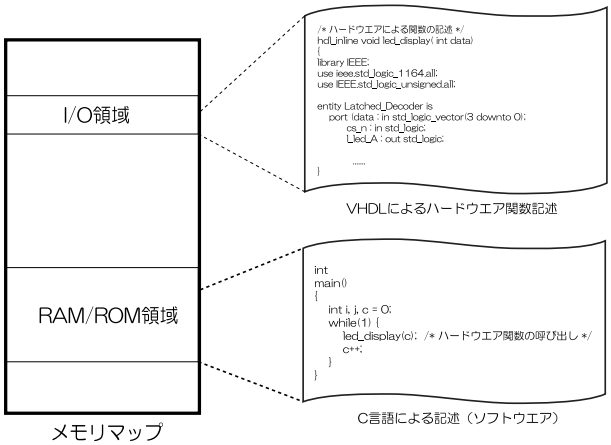

図 4　アドレスへのマッピング

また，アドレスのマッピングについては，DIO として扱うよう I/O 領域にマッピングを行うこととした．（図 4）

2−3　関数間の同期について

C 言語等で記述されたソフトウエアは，通常，時間に従って順番に処理される（図 5）（マルチ CPU などのシステムはこの限りでない）が，ハードウエアは空間的な広がりを持つため，必ずしも時間に沿って順番に処理されるわけではない．

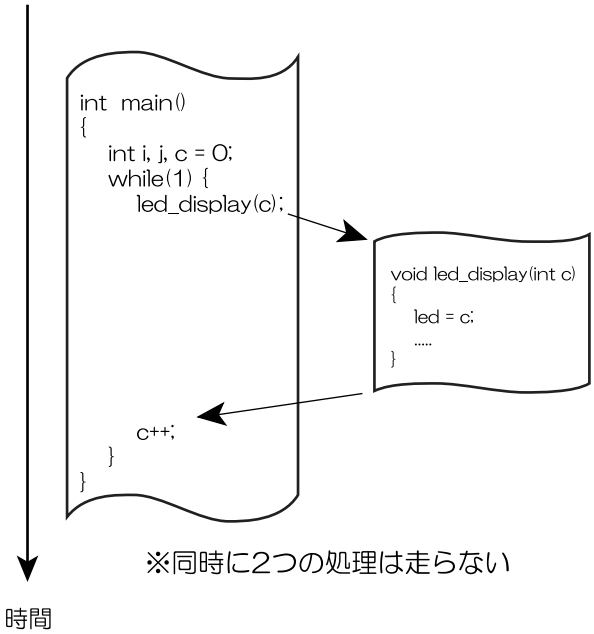

図 5　ソフトウエア関数の実行と時間との関係

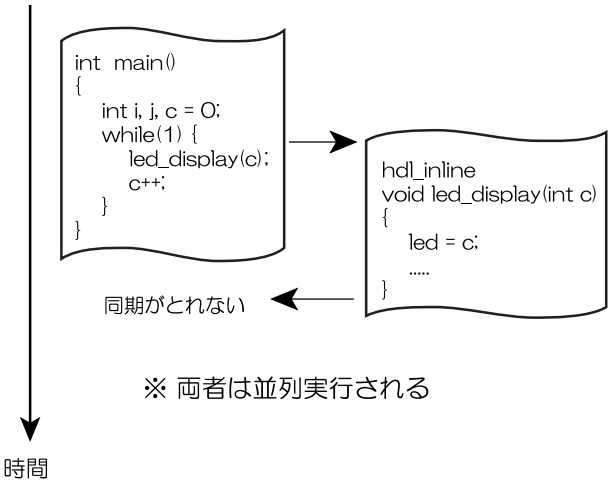

図 6　ハードウエア関数の実行と時間との関係

従って，ハードウエアと，ソフトウエアの処理がそれぞれの処理の間で，何らかの方法で同期を取る必要がある．

今回は，ファイルアクセスで用いられるブロッキング／非ブロッキング呼び出しを用いて，両者の同期を取る

こととした．この手法では，ハードウエア関数が終了しているかどうかは，ハードウエア関数の返値を用いて判定する（図 6）．

2−4　デバイスドライバについて

ハードウエア関数を C 言語関数から呼び出すには，デバイスドライバの仲介が必要となる．通常，デバイスドライバは，手作業により記述を行うが，VHDL によりハードウエアの構造が提供されているため，この情報を用いて自動生成を試みた．具体的には，図 7 のドライバを生成する．

```
/*
 *
 * HDL FUNCTION HANDLER
 *
 */

__sfr __at (__HDL_AUTO_ASSIGN_DIO0_ADDR__) __HDL_AUTO_ASSIGN_DIO0__

void
__hd_handler(int __v0__)
{
    __HDL_AUTO_ASSIGN_DIO0__ = __v0__;
}
```

図 7　デバイスドライバの例

なお，本ドライバは，Toppers/JSP への対応を想定して生成している．

2−5　付随するハードウエアの生成

WishBone バスにハードウエア関数を接続するには，

・アドレスのマッピング
・アドレスデコーダの生成（図 8）
・割り込みの割り当て
・割り込み優先順位の決定
・割り込み調停回路の生成

が必要となる．

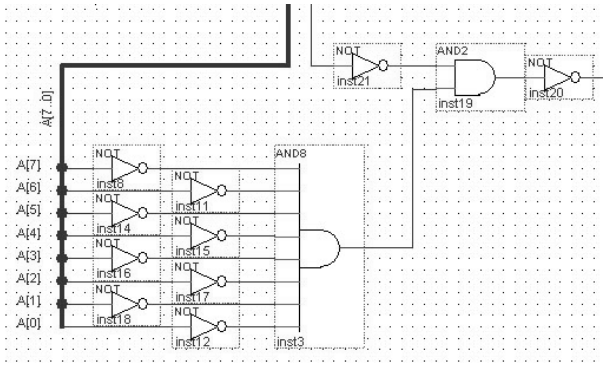

図 8　アドレスデコーダの例

現状では，これらの回路の自動合成は出来ておらず，手動で作成している．

2−6　システム構成

VHDL 言語と C 言語は，本来，全く異なる目的で設計された言語であり，これまで，ツール群も別個に開発が行われてきたため，両者を同時に処理できるシステムは，当然のことながら存在しない．それぞれのツールは，ユーザからの様々な要求に応えるべく，逐次，改良が加えられてきており，完成度も非常に高い．従って，既存のツール群を有効活用して，これまでに述べた要求を満たすべく，システムの設計を行うのが最も効率的と判断し，開発を行うこととした．

VHDL 言語によって論理合成，配置配線を行うツールとして，米 Altera 社の QuartusII[3]を用いることとした．また，C 言語のコンパイラとして，GNU[4]から配布されている，gcc を用いることとした．両者，いずれも無料で使うことが出来る．

本システムでは，VHDL/C 混在ソース（ファイル）をトランスレータを介して分離し，デバイスドライバ等の自動合成を行う．トランスレータは，混在ソース内の各関数について構文解析を行い，必要に応じて QuartusII と gcc の入力形式に変換を行う．トランスレータは，テキスト処理言語として広く普及している perl 言語を用いることとした．

システム全体の構成を図 9 に示す．

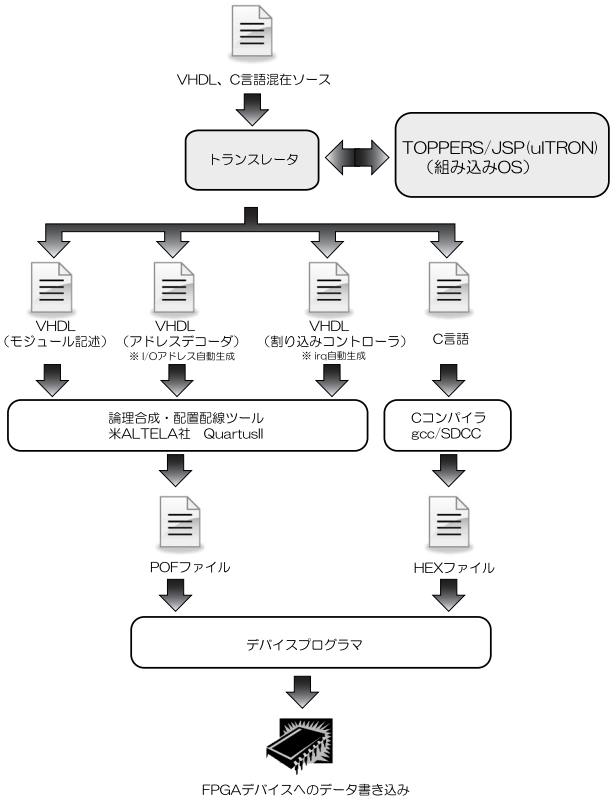

図 9　全体構成

3. 実機による検証

今回，FPGA 上に実装するソフトコア CPU として，opencores[2] で配布されている Aquarius を用いることとした．この CPU は，ルネサステクノロジ社製 RISC CPU である，SH2 と互換性がある．FPGA については，Altera 社製の CycloneIII（EP3C40Q240C8N）が実装された，HuMANDATA 社製のブレッドボード（ACM-018C）を用いることとした．

CycloneIII には，約 50KB の内蔵メモリが搭載されているが，Toppers/JSP を動作させるには不足すると思われたので外部に SRAM を増設している．また，デバック等の作業でシリアルインターフェースがあると便利なため，RS-232C レベル変換 IC を付加することとした．制作した基板の写真を図 10 に示す．

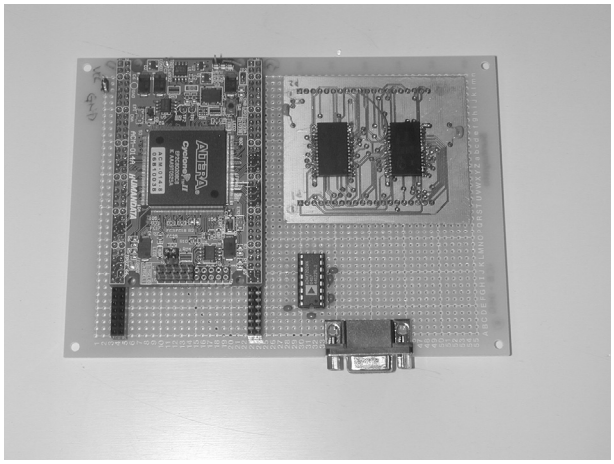

図 10　試作基板

図 9 のシステムを用いて，混在ソースを処理し，本基板を用いて動作確認を行った．手動で補正が必要な部分も，一部に存在するが，システムの動作は確認できた．

4. 考　　察

ハードウエア・ソフトウエア混在開発環境を用いることで，効率的なシステム開発が可能な環境を提示した．

現状で，アドレスデコーダ等の自動合成は出来ていないため，今後これらの開発を行ってゆく必要があると思われる．

また，本来別々の目的で開発されたツール群を用いてシステムを構成しているため，使い勝手に難がある．今後，この点をどのように改良するか検討の余地がある．

5. 結　　言

本研究は H19，H20 年度の 2 カ年をかけて，試験研究重点化事業として実施したものである．今後は，組み込み技術研究会などを通じて，現場での使用に耐える環境の構築を行ってゆく予定である．

参考文献

1）TOPPERS プロジェクト http://www.Toppers.jp

2）OPENCORES プロジェクト http://www.opencores.org

3）アルテラ社 http://www.altera.co.jp

4）GNU プロジェクト http://www.gnu.org