論文

アプリケーションベースの組み込み制御システム

武田有志＊1)　坂巻佳壽美＊1)　森久直＊1)　村越英樹＊2)
馬場敬信＊3)　横田隆史＊3)　大津金光＊3)

Application Based Embedded Control Systems

Yuji TAKEDA, Kazumi SAKAMAKI, Hisanao MORI, Hideki MURAKOSHI
Takanobu BABA, Takashi YOKOTA and Kanemitsu OOTSU

Abstract　We propose a new development methodology for embedded control systems called “ Application Base’’. Recently, it has become problematic that the controllers which are located in the core of systems are being developed for a shorter period of time and by fewer hardware quantities in embedded control systems. Using the conventional methodologies, the controllers are realized from general CPU, real-time OS, and application programs from the bottom-up. Therefore, it is difficult to acquire a certain level of responsibility based on the specifications generated by a top-down structure such as the designs in UML. In light of this, we solve the problems by generating the architecture of the processor satisfying the required responsibility from application programs by making use of reconfigureable devices such as, FPGA, PLD and so on which are connected to memory devices. This methodology can improve the reusability of software, and can reduce the developmental period. We propose this concept and its effects on Application Base.

Keywords　Embedded control system, UML, Processor core, Real-time OS, FPGA

１．はじめに

近年，組み込みシステムの制御装置は，適用分野の拡大と多様化に伴い，要求される機能を如何に少ないハードウェア資源で短期間に実現できるかが課題となっている。中でも生産ラインやロボットなどの制御システムの中核を担うコントローラにおいては，制御で最も重要な要素であるリアルタイム性が強く要求される。

現状の制御システムでの多くのコントローラは，汎用のCPU，リアルタイム OS（RTOS），そしてアプリケーションプログラムの３つで構成されている。UML（Unified Modeling Language）[1) 2) 3)] の登場によって，組み込みシステムの開発手法は，これまでの実装環境に強く依存した開発から，実装環境に依存しない開発へと徐々に移行しつつある。この移行によって，要求仕様の変更による下位設計での手戻りの負担を抑制し，開発期間の短縮が期待できる。また，UML2.0では，シーケンス図の拡張やタイミング図の追加がなされており，割込み処理の表現力を除いて組み込み分野に十分適用できるように改善されてきている。しかし，実装段階では，既存の CPU と OS，すなわち，既存のアーキテクチャに強く依存したプログラムを記述しており，十分な応答性能が得られない場合には，プログラムの変更が余儀なくされる。その結果，開発期間の短期化への鍵を握るプログラムの再利用性が著しく低下するとともにテスト期間にも大きく影響を及ぼす。このことは，文献 4）のバグ作り込み工程比率に現れており，全設計期間のうち，実装段階であるコーディング時のバグ発生の比率を上昇させていることから理解できる。

この問題を解決するために，プログラムを変更することなく CPU の命令セットを変更することで処理能力を向上させる研究がいくつかなされている[5) 6) 7)]。しかし，これらの研究の多くは，CPU の基本的な内部構造が固定されており，命令セットの最適化によるハードウェア量の削減に留まっている。そのため，ハードウェアの並列性を十分に活かせないだけでなく，プログラムを構成するタスクの個々の応答性については考慮されていない。一方，命令セットだけでなく CISC，RISC，パイプライン段数といった CPU の構造自体を自由に定義できる方式[8)] も存在するが，組み込み分野では各システムに特化して用いられることが多く，それぞれの用

＊1) 情報科学グループ　＊2) 都立科学技術大学
＊3) 宇都宮大学

途に対する最適な CPU を手作業で設計するには限界がある。また，近年のハードウェア設計では，プログラムの変更を極力抑え，同一のプログラムから複数のアーキテクチャを得ることができるという点で，C 言語に置き換わりつつある。しかし，RTOS のような複数タスクの並列実行に対しての最適化や，ハードウェア量を抑えるためのソフトウェア的な処理とのバランスなどについて，考慮するに至っていない。

そこで本論文では，従来の既存の CPU，OS を基本としたシステム開発ではなく，アプリケーションプログラムに応じて最適なハードウェア構成を生成する，すなわち，CPU，OS を含めたアーキテクチャを生成することで，プログラムの再利用性と応答性の高い組み込み制御システム構築手法を提案する。そして，本手法によるアーキテクチャの生成についての結果と考察を述べる。

2．システム構築手法

本論文で述べるアプリケーションベースの構築手法とは，アプリケーションプログラムを変更することなく，リアルタイム性を持った適切な制御アーキテクチャを生成する技術である。本技術は，再構成可能なデバイスである FPGA（Field Programmable Gate Array）などが中心となって構成される組み込み制御システムを前提にしている。また，適切な制御アーキテクチャとは，アプリケーションプログラムに与えられる時間的制約，すなわち，リアルタイム性を限られたハードウェア資源で達成することである。図 1 は，本手法による効果を示している。アプリケーションベースの組み込み制御システム構築手法では，従来の時間的制約に対する検証時間を削減し，実装までの手戻りを大幅に少なくすることができる。

上述のアーキテクチャを生成するためには，1）特定アーキテクチャに依存しないプログラミング方法，2）時間的制約を満たすアーキテクチャの探索，3）ハードウェア量を抑えるためのコード化およびプロセッサ化，の 3 つの課題を解決する必要がある。そこで，これらの問題に対し，1）実装環境に強く依存せざるを得ないデバイスドライバ以外につ

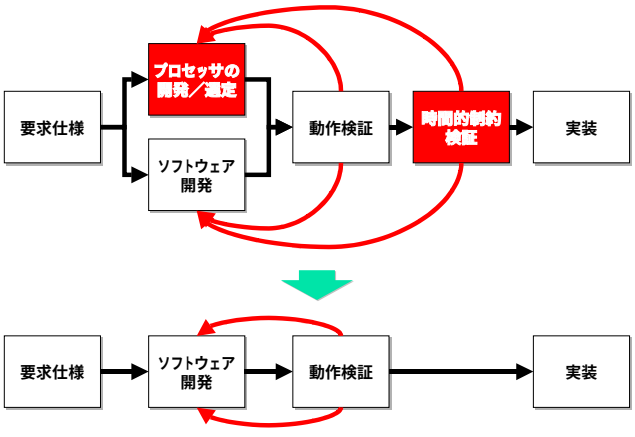

図 1　提案手法による設計期間への効果

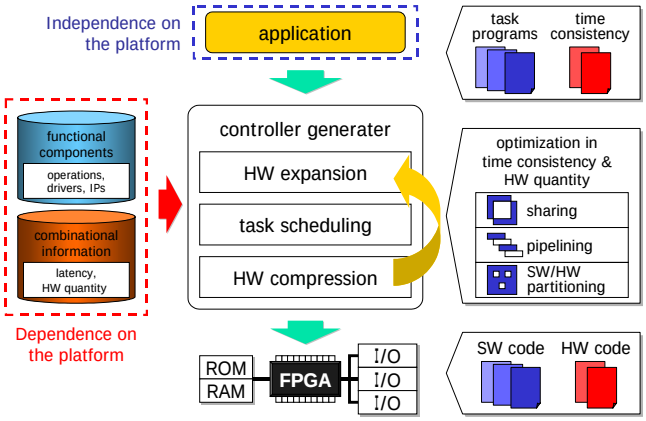

図 2　FPGA への実装過程

いては，時間的制約に関する記述を UML のタイミング図を用い別途定義する，2）アプリケーションを構成するタスクプログラムを繰返し回数が不定であるループや再帰呼出しは除き，すべてハードウェアに展開することで遅延時間を最小にする，3）タスクスケジューリングを行いながら，徐々にコード化してプロセッサコアを生成することで解決する。

図 2 は，本手法による FPGA までの実装過程を示している。中央にあるコントローラ生成ツールは，タスクと時間的制約を入力することによってアーキテクチャを生成し，実装環境である FPGA とメモリおよび周辺 I/O デバイスに設定可能なコードを出力する。入力となるのは，RTOS で扱われるタスクを単位とした C 言語プログラムであり，そのプログラムには，演算コードだけでなく RTOS の基本要素であるセマフォ，メッセージキューの関数呼出しを記述する。そして，出力となるのはハードウェア記述言語 VHDL（Very High Speed IC Hardware Description Language）のプログラムコードとソフトウェアのメモリイメージである。

コントローラ生成ツールの左側に示す機能部品，構成情報は実装環境に依存する情報である。機能部品には，論理演算器や加減算器といった基本演算回路だけでなく，その他の IP としてデバイスドライバが含まれる。また，構成情報には機能部品の組合せに対する遅延時間とハードウェア量が登録されている。これらの情報は，3．で述べるアーキテクチャの生成アルゴリズムで使用される。

3．アーキテクチャの生成アルゴリズム

図 3 はアーキテクチャの生成過程を示している。図中，上はタスクがハードウェアに展開された状態を表しており，下はタスクの時間的制約を元にタスクの実行状態を UML のタイミング図として表している。なお，本図で示しているタイミング図は，文献 2）の表記に従っている。アーキテクチャの生成は次の順序で行われ，それぞれ図 3 の（1）～（4）に対応している。

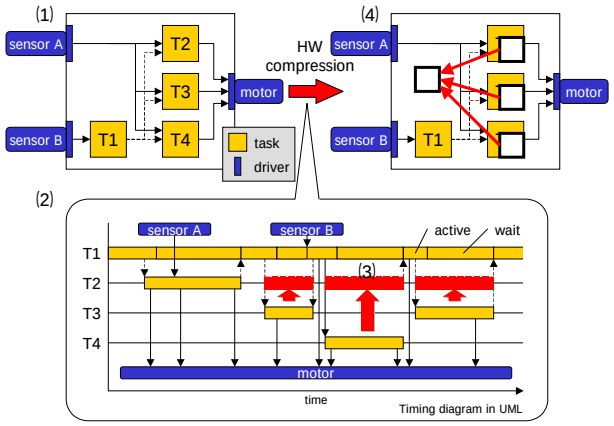

図 3 アーキテクチャの生成過程

（1） タスクを遅延時間が最小となるようにハードウェアに展開する。この時点で仮のアーキテクチャが完成し，応答時間が見積もれる状態になる。

（2） 与えられるタイミング図の上に現状のアーキテクチャの下での実行時間をマッピングする。

（3） 同時刻に実行されず，かつ，同一の処理を多く含むタスク群を集め，1 プロセスとして集約する。

（4） それぞれのプロセスごとに同一の処理を抽出し，抽出した処理を 1 つのハードウェアで共有させ，それをコード化してプロセッサ化する。

上述の（3）で同時刻に実行されないタスクをまとめているのは，コンテキストスイッチによるペナルティを抑制するためである。一方，（4）のコード化は，メモリアクセスなどのペナルティによって応答時間が変動する。そのため，本アルゴリズムでは，時間的制約に応じて（2）～（4）を繰返し実行する。最終的に生成されるアーキテクチャは，実装環境のメモリ帯域に応じた複数のプロセッサコアと，各処理の実行順序に制限を加えるための TCB（Task Control Block）で構成される。

4．アプリケーションへの適用結果と考察

ハードウェア量削減に対する効果を確認するために，RTOS の応用事例としてよくみられる，簡易アプリケーションの 1 つである自律歩行ロボットを取り挙げる（図 4）。本ロボットは，2 体で相撲を取ることを目的としており，複数の並列のタスクで動作し，リアルタイム性を要求する。前方センサは前方の相手を検出でき，下腹部センサは土俵内もしくは土俵際であるかを検出でき，そして，後部モータは自身を前後左右に移動できる。本体上部には Xilinx 社製 PLD（Programmable Logic Device）の Cool Runner（XCR3256XL-7-TQ144）が搭載されており，前述のセンサの状況に応じた行動をプログラミングすることが可能になっている。

本ロボットを動作させるプログラムは，表 1 に示すように 4 つのタスクで構成されている。T1 は，タスクの決定において，下腹部センサから土俵内であることを検出した場合には T2，T3，T4 を決められた順序で選択し，それ以外は他のタスクを止め T4 を選択することで，土俵外に出ることを抑制する。タスク以外のプログラムは，センサからの信号をスキャンするためのドライバ，モータを駆動させるためのドライバである。ドライバは，あらかじめ VHDL で記述し，生成されるアーキテクチャと単純に結合できるように，入出力ポートに対してシステムクロックに同期したレジスタが挿入されている。

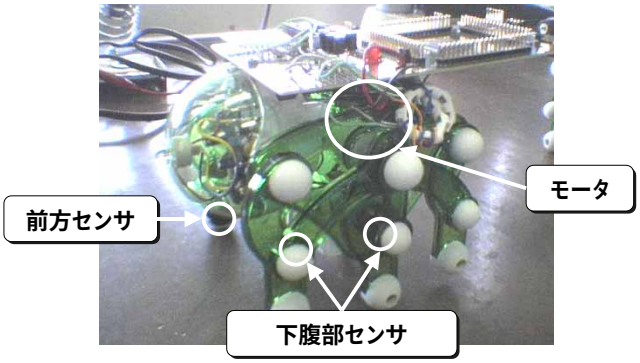

図 4　自律歩行ロボットの概観

表 1 自律歩行ロボットのタスク構成

タスク	処理内容
T1	次に行動すべきタスクを選定する。
T2	前方に障害物が現れるまで回転する。
T3	一定時間ある方向へ回転する。
T4	一定時間ある方向へ移動する。

3．で述べたアルゴリズムの適用前と適用後を比較した結果，適用前のハードウェアのみの展開では，PLD に含まれるマクロセル 256 個のうち 180%以上のセルを必要とした。しかし，適用後には，応答性を損なうことなく 80%程度にまで削減できたことを確認した。削減されたハードウェアは，主に T3，T4 における一定時間行動するためのタイマであった。ハードウェアが半分以上削減されたのは，他の演算処理に比べ，タイマを構成するカウンタに非常に多くのハードウェアを必要としていたからである。

本アルゴリズムの欠点として，次の 2 点を挙げることができる。

（a） プログラムが変更されるたびにアーキテクチャを生成し直していることから，再試行を重ねることで非常に長い時間を要する。

（b） 同一の処理が現れない場合には，汎用の CPU と同様に細粒度の命令セットを持ったプロセッサができてしまい，ハードウェア削減の効果が期待できない。

よって，これらの欠点を解決するためには，同一の処理だけでなく，良く似た処理を高速に探すための何らかのパターン抽出手法を取入れる必要がある。例えば，X = A * B + C，Y = D * E の 2 つの処理を考えた場合，乗算器と加算器を 1 つにし，$f(X_1, X_2, X_3) = X_1 * X_2 + X_3$ の処理を 1 命令として抽出

する。これにより，双方ともに X = f(A, B, C), Y = f(D, E, 0) で演算できることから, ハードウェア量を抑えることが可能になる。

このパターン抽出を行う手法の一つに, 自己組織化が挙げられる[9)]。自己組織化の特徴は，教師なし競合強化学習および近傍学習によって, ある分布に従う入力データの特徴を抽出し, その分布を近似した特徴マップを生成できることにある。自己組織化は，高速な学習ができるとともに，傾向分析や最適化問題など, あらゆる分野で有効であることが示されている。これをハードウェアに適用すると, 次のようになる。

（1） 特徴マップを構成する細胞に演算素子の入出力ポートを割り当てる。

（2） 展開されたハードウェアにおける２つの演算素子間の入出力関係をベクトル化し，それらを自己組織化で学習させる。

以上の操作を行うことで, 良く似た処理に対して非常に多くの細胞が集まり, 最も高い密度の細胞群とその近傍の細胞に割り当てられた演算素子をまとめ挙げることで, ハードウェアの削減が可能になる。

５．まとめ

本論文では，従来の設計手順とは逆のアプローチである，アプリケーションプログラムからコントローラのアーキテクチャを生成するための組み込み制御システム構築手法を提案した。本手法によって，プログラムにおける時間的制約を切り分けることでプログラムの再利用性を向上させ, システム開発期間の短縮することができる。さらに，本手法は，プログラムを他の実装環境にも移植することが容易なことから, 生産ラインなどの機器故障時のリプレースにも貢献できる。

今後の課題は, アーキテクチャ生成アルゴリズムの改良と，それを含めたアプリケーションプログラムからアーキテクチャを自動的に生成するためのツール開発である。また，本手法の有効性を示すためには，実用性があり，かつ，リアルタイム性が要求されるアプリケーションに対して本手法を適用することが必須である。そこで，現在，図５に示す生産ラインの集積機への適用を試みている。集積機は，ベルトコンベア上を流れてくるビンを一定量ごとに押出して整列するための比較的単純な装置である。しかし, 動力にエアーを用いているため, 個々のバルブにおけるエアーの抜けを並列に監視する必要があること, 既にプログラミングされているシーケンサとの信号の比較が可能であることから本装置を採用し，研究を継続している。この装置への適用によって，本手法の有効性を示すとともに，実用化する予定である。

謝辞　本研究を推進するに当たり, 生産ラインの提供に快くご協力していただいた（株）ケーテー製作所の方々に厚く御礼を申し上げる。

SFC ブロック数	12
入力接点数	30
出力コイル数	25
内部コイル数	93
監視タイマ数（常時）	23（11）

図５ 実装対象の生産ラインとそのプログラム規模

参考文献

1) UML (Unified Modeling Language) Infrastructure Final Adopted Specification, Object Management Group（2003） http://www.uml.org/technology/documents/

2) B. P. Douglass 著，渡辺博之訳：リアルタイム UML 第２版―オブジェクト指向による組込みシステム開発入門，翔泳社（2001）.

3) 渡辺博之：組み込み UML－eUML によるオブジェクト指向組み込みシステム開発，翔泳社（2002）.

4) 坂村健 監修，μITRON3.0 標準ガイドブック，社団法人トロン協会，pp.36-37（2001）.

5) J. Sato et.al.: PEAS-I: A Hardware/ Software Codesign System for ASIP Development, IEEE Trans. Fundamentals, Vol.E77-A, No.3（1994）.

6) N. Tagawa et al.: A Hardware/Software Cosynthesis System for Digital Signal Processor Cores, IEEE Trans. Fundamentals, Vol.E82-A（1999）.

7) K. Tanaka et al.: Casablanca: A Real-Time RISC Core for Embedded Systems, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet（2001）.

8) S. Kobayashi et al.: A Compiler Generation Method for HW/SW Codesign Based on Configurable Processors, Vol.E85-A, No.12（2002）.

9) T. Kohonen: The Self-Organizing Map, Proc. of the IEEE, Vol.78, No.9, pp.1464-1480（1990）.

（原稿受付　平成 16 年 8 月 6 日）