論文

制御プログラムからのリアルタイムプロセッサ生成方式

武田有志*1)　坂巻佳壽美*1)　乾　剛*1)　村越英樹*2)

A Real-time Processor Generation Method for Control Programs

Yuji TAKEDA, Kazumi SAKAMAKI, Takeshi INUI and Hideki MURAKOSHI

Abstract　We propose a real-time processor generation method for embedded control systems. Recently, it is a problem has developed wherein controllers are developed for a shorter period of time and by fewer hardware quantities in embedded systems, and the responsibility of the controllers is required in the control systems. In the conventional methodologies, the controllers are realized from general CPU, real-time OS, and control programs from the bottom-up. However, their methodologies cannot acquire the controllers having responsibility based on the specifications such as the designs in UML cannot be acquired without changing control programs. Then, we solved these problems by generating the architectures of the processor core from control programs by making use of the reconfigureable device, FPGA. We show that the generation algorithm can generate applicable instruction set of the processor core.

Keywords　Real-time processor, Embedded control system, UML, FPGA

１．はじめに

近年の組込みシステムは，適用分野の拡大に伴い，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。組込みシステムは大きく分けて制御，通信，信号処理の 3 つの分野からなるが，中でも生産ラインの監視装置やロボットの位置制御装置など，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求められる。現在のコントローラの多くは，汎用 CPU，RTOS（Real-time OS），そして制御プログラムの 3 つで構成されている。

コントローラ開発において，従来はボトムアップでの実現がほとんどであり，制御プログラムは既存の CPU と OS，すなわち，固定されたアーキテクチャに強く依存した記述がなされている。しかし，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。そのため，開発の短期間化への鍵を握るプログラムの再利用性は著しく低下するとともに，テスト期間に対しても大きく影響を及ぼしている。一方，UML（Unified Modeling Language）[1) 2)] によるモデリング技術の標準化が進むにつれ，組込みシステム開発は，従来のボトムアップからトップダウン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工程での手戻りを抑制することができる。UML2.0 は，シーケンス図の拡張，タイミング図の追加，さらに，各ダイアグラムの階層化がなされており，割込み処理の表現力を除いて，組込み分野に十分適用できるようになっている。しかし，リアルタイム性が保証されているか否かは実際にシミュレーション等を行った後で確認するに留まっている。

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応答性能を向上させる研究がいくつかなされている[3) 4)]。これらの研究の多くは，基本的な命令セットは決まっており，最適化の範囲はプログラムの流れに即した複数命令の同時実行である。また，ハードウェアの並列性を十分に活かせないだけでなく，組込みで要求されるハードウェア量の削減については考慮されていない。

そこで本論文では，従来の CPU，OS を決定した上での制御プログラム開発ではなく，FPGA（Field Programmable Gate Array）の柔軟性を利用して，プログラムに応じて最適な CPU の命令セットと OS を含めたアーキテクチャを生成するリアルタイムプロセッサ生成システム coregen を提案する。coregen では，プロセッサの命令セットを生成する際に，プログラム記述をできる限りハードウェアに展開してから行う。そのため，ハードウェアの並列性を活かすことが可能である。

２．リアルタイムプロセッサ生成システム

リアルタイムプロセッサ生成システム coregen は，制御プ

*1) 情報科学グループ　*2) 都立科学技術大学

ログラムを入力として，ハードリアルタイム性を持ち，かつ，ハードウェア量の少ないプロセッサを生成するためのシステムである。ここで指すハードリアルタイム性とは，制御プログラムに与えられる全ての時間制約のうち，実行前に静的に求められるものを対象にしている。これにより，従来の時間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。coregen では，再構成可能なデバイスである FPGA を使うことを前提としており，coregen で生成されたプロセッサアーキテクチャの変化は FPGA に吸収させている。

図 1 は，本システムによる FPGA までの実装過程を示している。中央にある coregen は，制御プログラムを入力することによってプロセッサの基本アーキテクチャを決定し，命令セットを決定した上で実装環境である FPGA とメモリに設定可能なコードを出力する。入力となる制御プログラムは，タスクとタスク制約の 2 つで構成される。タスクの記述には生産ラインの分野で広く用いられている LD（Ladder Diagram）もしくは組込み分野で広く用いられている C 言語プログラムであり，これらは時間的な制約を考慮しない純粋なアルゴリズムである。また，タスク制約は UML の状態図とシーケンス図である。状態図にはシステムの状態を記述し，シーケンス図には各状態に応じたタスク間の通信および時間制約を記述する。一方，出力となるコードは，ハードウェアコードとソフトウェアコードの 2 つで構成される。ハードウェアコードはリアルタイムプロセッサを VHDL（Very High Speed IC Hardware Description Language）で表したものであり，各社ベンダによる合成ツールを用いて FPGA に書き込むことができる。プロセッサは，入力プログラムの並列度に応じて複数のプロセッサコアで構成される。また，ソフトウェアコードは生成されたリアルタイムプロセッサ専用の命令コードおよびデータを表し，FLASH メモリに書き込むことができる。

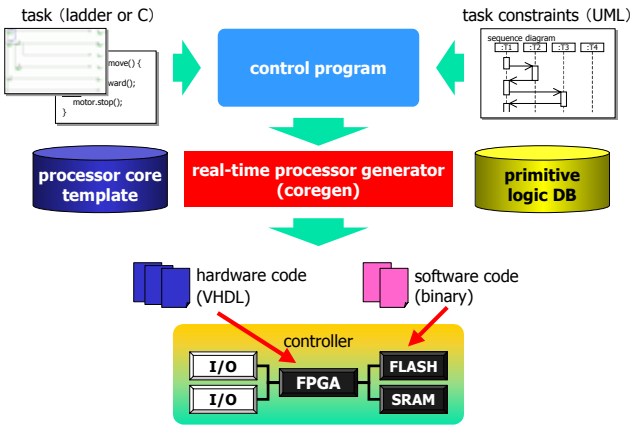

図 1　FPGA への実装過程

coregen は，次の手順でプロセッサを生成する。

(1) シーケンス図を解析し，必要なプロセッサ数と割当てるべきタスク，そして，並列実行の際に必要となるセマフォ，メッセージキューを抽出する。

(2) プロセッサコアテンプレートを参照し，セマフォ，メッセージキュー，そして，入出力および外部メモリとのデータバスと制御信号を決定する。この時点でプロセッサ全体の基本的なアーキテクチャが決定する。

(3) プリミティブロジック DB を参照して，各プロセッサに割当てられたタスクをハードウェアに展開し，それぞれの応答時間を見積もる。

(4) 時間制約に余裕がある場合に，類似する処理を抽出していき，該当する処理を 1 つの命令としてまとめ上げ，ハードウェア量を削減する。

(5) 得られた命令群をコード化し，命令デコーダを生成する。そして，命令デコーダおよび命令コードに対する処理を (2) におけるアーキテクチャに組込む。

ここで，(3) においてタスクをハードウェアに展開することは，ハードウェアの並列性を活かした十分な応答性能を引き出だせることを意味する。

3．プロセッサの基本アーキテクチャ

図 2 は，プロセッサコアテンプレートに保持しているプロセッサの基本アーキテクチャである。基本アーキテクチャは，レジスタファイル，命令デコーダ，ALU，プログラムカウンタで構成されており，制御目的に特化した次に示す特徴を持つ。

a) 汎用プロセッサに比べてデータ転送量が少ないことから，外部メモリにあるデータは ALU を通さずに一度レジスタファイルに転送される。

b) 応答性を向上させるために，入出力状態は direct in，direct out を通じて，直接，レジスタファイルが参照される。

c) 1 つの命令は，IF（Instruction Fetch），ID（Instruction Decode），EX（Execution）の 3 つのステージからなるパイプラインで実行される。

図中，レジスタファイル，命令デコーダ，ALU の 3 つユニットは coregen によって生成される部分であり，制御プログラムによって異なるハードウェアになる。また，パイプライン実行は，メモリアクセスにおけるレイテンシを隠蔽するとともに，FPGA に与えられるクロック周波数および 1 メモリアクセスに要するクロックサイクル数から時間制約を満足する最大の命令発行数を事前に見積もることができる。

本アーキテクチャは，周辺入出力の状態が直接参照できる環境を想定している。しかし，PLC のようにスキャン実行を行う制御プログラムの場合には一定周期ごとに一括して入出力を行う必要がある。そこで，制御プログラムに LD が用いられる場合には，プログラムの開始に同期して direct in の状態を取り込み，また，direct out の外側にもう一段のレ

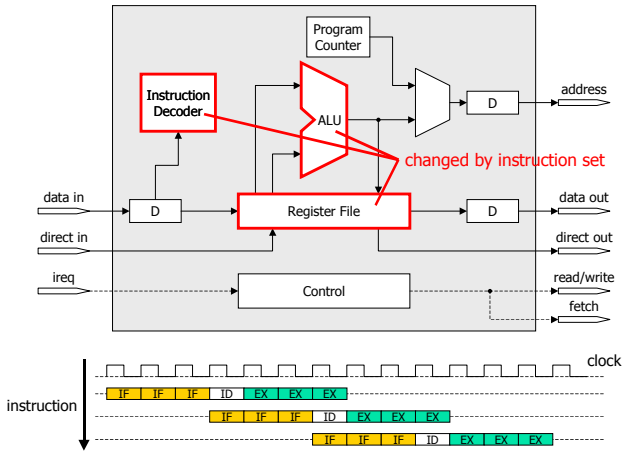

図２　プロセッサの基本アーキテクチャ

ジスタを挿入し，プログラムの終了に同期して出力状態を反映させることで対応する。

制御プログラムにおいて，タスクが並列性を有する場合には，プロセッサはその分，生成される。プロセッサ間の同期や状態を管理するための TCB（Task Control Block）が形成され，TCB 同士とプロセッサを組み合わせることで最終的なコントローラが生成される。ここで，マルチプロセッサでは，メモリアクセスにおけるボトルネックが生じる。この課題を解消するために，coregen ではタスクをコードブロックごとに分割し，時間制約の緩いブロックを後で実行することで，必要以上のプロセッサが生成されるのを防いでいる。

４．命令セットの生成

各プロセッサで実行すべきタスクが決まると，そのタスクはハードウェアに展開される。これによって，最小の遅延時間を求めることができる。しかし，すべてをハードウェアに展開することは，ハードウェア量が膨大になり，実装コストにも影響が出る。そこで，与えられた時間制約に対して十分に間に合う場合には，類似する処理を命令コードとしてまとめ上げた命令セットを生成することでハードウェア量の増加を抑制する。命令セットの生成には，ハードウェア処理における信号線の分割が課題となる。一方，FPGA の内部は，複数の回路素子によって構成されており，さらにそれぞれの回路素子は演算を行う LUT（LookUp Table），レジスタ，選択回路の 3 つで構成されている。したがって，FPGA において効果的にハードウェア量を削減するためには，3 つの要素に着目して分割方針を定めれば良い。

図 3 は，処理の分割によるハードウェアの削減対象を示している。ここで，左側は分割前，右側は分割後に必要となるハードウェアを表している。また，四角は演算処理を，矢印はデータが流れる信号線を表し，縦の線分は処理を分割する信号線を指している。縦の線分には，それぞれ①，②が振られており，①は②に比べてハードウェア量が少なく実現できることを意味している。本図から，ハートウェア量の削減には，それぞれの要素に対して以下を行えば良いということが導き出せる。

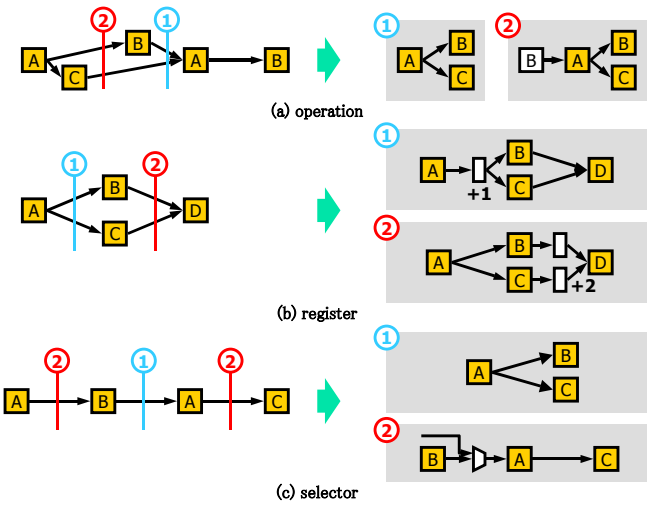

図３　ハードウェア量の削減対象

(a)　演算処理（LUT）の削減：ある信号線に対して，出力元と入力先の演算処理を見たとき，他にも同じ関係があるときは分割しない。例として，図中（a）において A→B が 2 つ存在しているが，②は A→B で分割しており，①よりも演算処理が 1 つ多い。

(b)　レジスタ数の低減：多くの演算処理の出力を参照する演算処理の手前では分割しない。例として，図中（b）において，②は B，C 両方を参照する D の手前で分割しており，①よりもレジスタが 1 つ多い。

(c)　セレクタ数の低減：ある演算処理の出力が多く参照される場合，その演算処理の後ろでは分割しない。例として，図中（c）において A は B および C から参照されているが，②は A→B，A→C で分割しており，①よりも選択回路が 1 つ多い。

これらすべての条件を満たすことは，NP（Non- deterministic Polynomial）完全問題であることから何らかの最適化アルゴリズムが不可欠である。これに対する一つの候補は，GA（Genetic Algorithm）が挙げられる。しかし，GA は最適解に至るまでに実時間内で求められるものの，良質な解を得るための焼きなまし速度に反比例して非常に多くの時間を要するため，制御プログラムの変更の度に命令セットを求める本システムには適用できない。

そこで，バネ力学の原理を応用した命令セット生成アルゴリズムを提案する。本アルゴリズムは，演算処理を物体，信号線をバネとして捉え，それぞれの条件をバネ定数として表現する。また，物体とバネの長さを時間として表現する。すると，演算処理 i，j 間の力は $F_{ij} = -k_{ij}\,x_{ij}$ と表すことができる。ここで，バネ定数 k_{ij} を $k_{ij} = k_{ija} \cdot k_{ijb} \cdot k_{ijc}$ と置くと前述の条件は次のように表すことができる。

(a) 出力元と入力先が同一の演算処理であった場合は $k_{ija} = k_{ps}$（$1 \leqq k_{ps}$）とする。逆に，入出力ともに同じ演算処理が接続されている場合は処理が分散されるように $k_{ija} = k_{pd}$（$0 < k_{pd} \leqq 1$）とする。それ以外は $k_{ija} = 1$ とする。

(b) 演算処理の出力をすぐに処理させるために，バネの基本的な性質である力をそのまま利用する。ただし，別の演算処理が完了していないと処理できない場合は $k_{ijb} = k_r$（$0 < k_r \leqq 1$）とする。それ以外は $k_{ijb} = 1$ とする。

(c) 演算処理の出力が複数の異なる演算処理に入力されている場合は $k_{ijc} = k_s$（$1 \leqq k_s$）とする。それ以外は $k_{ijc} = 1$ とする。

これらすべてを各演算処理に対して求め，力が均衡する位置に演算処理を時間的に移動させる。ただし，時間的な順序を強制するために演算処理の移動範囲は制限される。力の平衡点に達した後は，命令発行数で区切ることでそれぞれがプロセッサにおける命令として生成される。

５．実装と考察

図 4 は，４．で示した命令セットの生成アルゴリズムを Java で実現したツールを示しており，２．で述べたプロセッサの生成手順（3）～（4）に相当する処理を行う。また，このツールは各演算処理の実行順序や命令セットの生成過程を視覚的に表示させることができる。アルゴリズムの評価方法としては，LD で記述されたトグルスイッチを IL（Instruction List）に変換して入力し，応答時間を 60 単位時間，各演算処理の遅延を 10 単位時間とし，いくつかの命令発行数で分割した場合の演算処理，レジスタ，選択回路における，それぞれの数の比較で行う。

表 1 は，$k_{ps} = 2$，$k_{pd} = 1/2$，$k_r = 1/8$，$k_s = 2$ とし，命令発行数を変化させたときの結果を表している。ただし，レジスタと選択回路は，命令として扱う前からの純粋な増加分が示されている。この結果から，命令発行数が 1 のときに比べ，演算処理数を 1～2 個削減できたことが分かる。また，命令発行数が多くなるにつれ，レジスタ数が増加しているが，分割点における最大のレジスタ数は 3 個であり，命令間のレジスタ割付けを適宜変えることで，これ以上増加させることはない。選択回路においては，命令発行数が 2 のときに他と比べて 1 個増加しているが，各命令での処理時間が制約として与えられており，演算処理が時間的に移動できなかったからである。

以上のことから，演算処理，レジスタ，選択回路の数を抑え，命令発行数によって適切な命令セットを生成できた。また，生成されるまでに要する時間も小規模であれば Pentium 4 3.2GHz であれば 1 秒以内で収束することが可能であり，バネ力学の適用により，GA やニューラルネットワークに比べて，局所解に陥りにくいという特徴がある。しかし，k_{ij} に乗ずる係数は経験的に求めたものであり，異なるプログラムでは良い結果が得られるとは限らない。したがって，これらの値はユーザが視覚的にリアルタイムで変更できるような仕組みを取り入れる必要がある。

図４　開発したハードウェア分割ツール

表１　分割後のハードウェア量

命令発行数	演算処理数	レジスタ増加数	選択回路増加数
1	5	0	0
2	3	3	1
3	3	4	0
4	4	6	0
5	3	8	0

６．まとめ

本論文では，プログラムに応じて最適な CPU の命令セットと OS を含めたアーキテクチャを生成するリアルタイムプロセッサ生成システム coregen を述べ，その中核となるプロセッサの基本アーキテクチャおよび命令セット生成アルゴリズムを提案した。この結果，時間制約から求まる命令発行数によって，演算処理，レジスタ，選択回路の数の削減に着目した命令セットが生成されることを確認した。これによって，従来のソフトウェアによるプロセッサよりも応答性能に対して柔軟に対応することができ，制御プログラムの再利用性の向上が計れる。

謝辞　本研究を推進するに当たり，ビン整列機を提供して頂いた（株）ケーテー製作所の方々に厚く御礼を申し上げる。

参考文献

1) Unified Modeling Language: Object Management Group, 2004. http://www.uml.org/

2) B. P. Douglass 著，渡辺博之訳：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) K. Tanaka and T. Matsumoto: Proc. Advances in Infrastructure for Electronic Business, Science, and Education on the Internet(2001).

4) S. Kobayashi, K. Mita, Y. Takeuchi and M. Imai: IEICE Trans., Vol.E85-A, No.12(2002).

（原稿受付　平成 17 年 8 月 3 日）