組込み機器のスクリプト制御技術

田村 修†

†有限会社リカージョン 〒600-8148 京都市下京区飴屋町 252 井上ビル 4F
E-mail: †tamura@recursion.jp

1. はじめに

組込み機器の開発においては, 多様な CPU のアーキテクチャを理解し，専用の開発環境に習熟する必要がある．案件ごとに異なる CPU が採用される，あるいは異種 CPU が混在するシステムでは開発負荷が大きくなる問題がある．さらに，1 台限り，あるいは多品種少量の生産においては，開発環境の維持も含めた長期的な保守が難しくなることが懸念されている．すでに 10 年以上前に製作された FA 装置の更新に際し，互換 CPU を入手できない，当時と同一のバイナリを生成する環境を再構築できない，といった障害が発生している．クロス開発においては，メーカ提供の統合環境の更新, OS のバージョンアップ，それに伴う PC 仕様の進化などにより，開発当時の環境を再現することが，今後ますます困難になると予想される．

これとは別に，シーケンサや実験装置など，運用現場で動作シーケンスのカスタマイズが必要とされる製品がある．また，従来はハードウェアで制御されていた機構が，小規模マイコンに代替されていくという予測もある [1]．開発途上地域や特殊環境に投入される工作機械などでは，マイコン制御部分の調整も含めた現地での応急処置によって，製品寿命が延びる可能性がある．これらの装置では，カスタマイズに備えて処理の大枠をスクリプト制御化する需要が見込まれる．

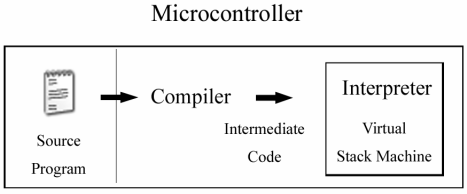

図 1 組込みコンパイル
Fig.1 Embedded Compile

提案する開発方式では,
・長期的な保守性の向上
・運用時のカスタマイズ
を目標とし，ひとつのアプローチとして
(1) CPU アーキテクチャの仮想化
(2) 基本的な開発環境の機器内蔵
を試みた．本稿では，個々の CPU アーキテクチャをスタックマシンに仮想化するとともに，軽量化した言語処理系をマイクロコントローラ内に実装することにより，長期的な保守を容易にするとともに，柔軟なシステム運用が可能となることを示す．

2. 基本構成

2.1 アーキテクチャの仮想化

異なるアーキテクチャの CPU には, それぞれの開発環境を用いてのコード生成が必要となる．システムの CPU を別品種に入れ替える際にはソースコードを再コンパイルするが，組込み機器においては周辺回路の見直しから CPU 依存記述の変換まで, 多くの作業が必要とされる．組込み開発での移植性や再利用性を向上させる試みとしては，コンポーネントベースの多層アーキテクチャ化による方式が提案されている [2]．これは，実装アプリケーションの視点からタスクを分類整理し，ハードウェア非依存の部分を再利用していくものであり，CPU ごとのハードウェア依存ドライバが整備された同一の開発環境下であれば，実用性が高いと考えられる．ここでは CPU メーカを越えてのハードウェア依存を隠蔽するとともに機種依存の開発環境を統一するため，あらかじめアーキテクチャをソフトウェア的に均質化する方法を採った（図１）．

CPU は，中間コードインタープリタを実装することにより，仮想スタックマシンとして動作する．バイトコードを順に読み出し，対応するスタック操作を行うものであるが, これにより CPU 固有のアーキ

テクチャが隠蔽される．そして，このスタックマシン用の中間コードを生成するコンパイラにより，すべての CPU へのプログラミング環境が共通化される．CPU の交換に際しては，それぞれの CPU へのインタープリタ部分の移植のみが作業となる．

しかしながら，アーキテクチャの仮想化には，ハードウェア依存を払拭し難い部分があることや，性能低下の面で課題がある．

割り込みは，テーブル登録された関数をスレッドループの狭間で呼び出すが，CPU 割り込みに対応したテーブル構造になるとともに，遅延が生じる．また，周辺機能へは特殊機能レジスタの直接操作でアクセスするため，今後は汎用的な組込み関数に入れ替えるなどして依存色を薄める必要がある．再利用性向上のために割り込み機能を抽象化する研究は，組込み OS ベースで行われたものがある [3]．

インタープリタ動作により実行効率は 1/2〜1/10 程度に低下する．性能限界での処理を行う場合や低消費電力化が必須の応用では，インタープリタ部分にネイティブコードでの拡張が必要となる．とはいえ昨今のローエンドマイコンでは数〜数十 MIPS の性能を持つものが普及し始めており，多くの応用においてはこの性能低下を許容し得る状況にあるとみられる．

仮想マシンによるアーキテクチャの隠蔽で可搬性を高める例は Java に見られる．組込み機器においては，仮想化によってマイクロコントローラの置き換えが容易になるとともに，言語処理系が統一される利点がある．

2.2 言語処理系の内蔵

言語処理系自体を組込み機器に内蔵させる利点として以下が考えられる．

・開発環境の散逸の防止
・運用現場でのコード変更
・機器内部構成の隠蔽

ソースプログラムや言語処理系が内蔵されていれば，テキストエディタと通信プログラムを用意するのみで，コードの補修，入れ替えが可能となる．また，コンパイルに際しては，システムの構成に応じた設定やライブラリのリンクが必要となる場合があるため，処理系が内蔵されれば外部クロス環境との同期を取るといった手続きが不要となる．

ソースプログラムは，開発時にホストから転送し内部で中間コード化するか，予め機器に内蔵しておき実行前にコンパイルする．生成した中間コードは，プログラム可能なフラッシュメモリや EEPROM に格納する．

プログラミング言語としては，コンパイラがスタックマシン向けにコード生成できるのであればどのようなものでもよい．スタックマシンを想定したコード生成の後に機種依存コードを生成するコンパイラであれば，転用は容易である．

PC 性能の向上に伴い，現在の組込み機器の多くはクロス開発で実装される．充実した開発環境のすべてを機器側に移すことは現実的でないが，機器単独で，補修や機能拡張など最低限の保守を行える程度の処理系を内蔵させることは有用と考えられる．

なお，マイクロコントローラ内に言語環境を組込んだ前例としては，8051 系 CPU に搭載された BASIC-52 がある．CPU が限定されること，実装が大きく外付けメモリを要することなど難点も多いが，手軽な開発スタイルとして長期にわたり支持されている．また，機器側に言語処理系を内蔵させる例としては，PostScript プリンタがある．印刷系特有の事情を反映したものであるが，高級言語でのインターフェースによって自由度の高いラスタライズ処理が実現されている．

3. コンパイラの軽量化

組込み機器開発用の言語処理系としては標準化が進み普及した C や Java が望ましい．Tiny CC，Wava のように言語処理系を軽量化する試みもあり，搭載メモリや CPU の処理能力が十分であれば検討に値する．ここでは搭載メモリが数 K バイト程度のローエンドマイコンを実装対象としたことや，標準仕様の定着した言語に極端なサブセットや方言は好ましくないとの考えにより，独自の言語を設計した．

ローエンドマイコンに要する処理としては，C 言語レベルで数十〜数百行程度のものが多いため，この規模の記述に耐え得る範囲で言語仕様を縮小した．言語処理系としての実装コストが小さく拡張性が高い FORTH を範にしたが，フロー記述の見難さを改善しポインタ的操作を禁止するなど大幅に改変・簡略化し，PostScript 的な構文で C 言語風な記述をする後置記法型プログラミング言語 ForCy を開発した（図２）．コンパイラは，仮想スタックマシンへの中間コードを生成するが，ソースコード中の単語の ID 化が主な処理であり，単語と生成される中間コードとが同順でほぼ一対一に対応する．{ } で囲まれた

ブロックの長さをカウントし，スキップさせるコードに対応させることが，唯一の構文解析処理となっている．

今回実装したコンパイラは，仮想スタックマシンへコード生成する処理系としては最も単純なものである．スタックマシンベースであるため，目的に応じて他の言語に進化あるいは置換させることは難しくない．

仮想スタックマシンの動作は，インタープリタにより実現される．中間コードに対応する処理を順次呼び出しデータスタックとリターンスタックを操作するのみで，実行時の構文解析は行わないため，機能の追加や削除が容易な構造となっている．演算はすべて 16bit で処理される．

なお，コンパイラは自己記述によって自身も中間コード化されインタープリタ上で動作する．各 CPU への移植は，インタープリタ部分の 1K バイト程度をアセンブラもしくは C コンパイラで記述するのみとなる．また，ボトルネックとなる処理部分はアセンブラ記述してインタープリタ部分に追加し，コンパイラにシステム定義ワードを登録して高速化することが可能である．コンパイラ，インタープリタを合わせた言語処理系全体を 3K バイト程度で実装できる．

```
        "¥n¥tHello, World.¥n" %s    //  文字列表示
:hello

        ;x ;y                       //  九九表

        1 =y
        {
                y 9 <= while        //  1〜9まで繰返し
                1 =x
                {
                        x 9 <= while    //  1〜9まで繰返し
                        x y * %3d       //  x * y  を表示
                        ++ =x
                } do .
                '¥r' %c '¥n' %c     //  改行
                ++ =y
        } do .
:multiply

        1 > { dup -- self * } if    //  階乗
:factorial
```

図 2　プログラム例

Fig.2　Examples of the ForCy Program

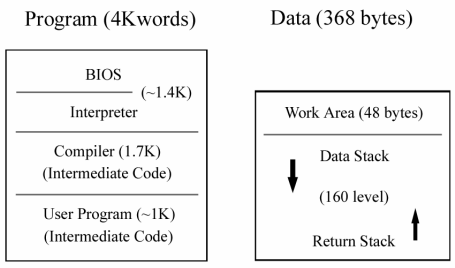

図 3　PIC16F88 のメモリマップ

Fig.3　Memory Usage of PIC16F88

Communication	2,400 bps xon/xoff
Compile	25 words/sec
Execution	12,000 empty loops/sec
User Program	~ 300 lines
	(PIC16F88 - 20MHz)

図 4　プロセッサ処理性能

Fig.4　Performance of the Processor

4. 実装

4.1 ローエンドマイコンへの組込み

コンパイラの組込みが，搭載メモリが少ないマイクロコントローラにおいても実用的に機能することを示すため，マイクロチップ社の PIC 16F88 (ROM 4K 語，RAM 368 バイト)に実装した．RS-232C 回線より送り込まれるプログラムテキストを，逐次コンパイルしてプログラム書き込み可能なフラッシュメモリ領域に書き込んだ後，インタープリタ実行する．PIC16 アーキテクチャでは，寸断された RAM 領域の連結，フラッシュ領域でのアクセス遅延，間接アドレッシングなどの手間により，スタックマシン動作のオーバヘッドが著しく大きい．20MHz クロックでは，円周率 100 桁の計算に 8 秒を要する性能となった．

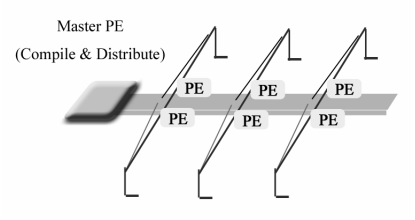

図 5　分散プロセッサへの組込み
Fig.5　A built-in Compile for Multiprocessor

4.2 分散マルチプロセッサへの応用

システム側に言語処理系を移し，柔軟な開発を可能とする例として，分散マルチプロセッサ環境の動作設定に応用した. ロボット制御などでの CPU の分散化は，負荷分散によるリアルタイム性や拡張性の向上，配線引き回しの削減など多くの利点がある [4] が，複数プロセッサへの動作設定が煩雑になる．ここでは最も簡単な実例として，欧米で学習教材として普及しているムカデ型ロボット Stiquito[5]に実装した. これは単一 CPU で電流発熱型形状記憶合金の 6 本の足を動かすものであるが, 各足にプロセッサ 1 個を割り当て，さらにコンパイル配信用プロセッサを持った 7 プロセッサ搭載型に改造した. ホスト PC 上で作成したプログラムを一括して通信プロセッサに送り込み，機器内部でそれぞれのプロセッサ向けにコンパイル，配信する．配信プロセッサにはルネサス R8C/Tiny, 足制御プロセッサにはマイクロチップ PIC12F683 を用いた.

5. まとめ

本方式は, 組込み開発現場の負荷を軽減すること, なかでも長期的な保守を確実に行うことに主眼をおいて考案した．そして，コンパイラを組込みスクリプト制御する技術が広範囲のマイクロコントローラに適用可能であり，特別な環境をセットアップすることなく迅速な開発を行えることを，ローエンドマイコンへの実装とロボット組込みにより確認した. 発展途上で改善すべき点は多いが，有効性が期待される利用状況としてはつぎが考えられる.

(1) 長期的な維持管理が必要とされる FA 機器でのマイコン制御
(2) 運用現場での動作シーケンスのカスタマイズを要する実験装置，シーケンサ
(3) 多種多様のプロセッサを統合管理するスタンドアローンシステム（ロボットなど）
(4) マイコン応用の学習教材（テキストエディタとターミナルのみでプログラム可）

アーキテクチャの仮想化とともに言語処理系などのシステム構築技術を可能な範囲で機器側に移す発想は，長期的なシステム補修を確実に行うための技術として応用が期待できる．今後の課題としては，より現実的なシステムでの実証を行うこと，必要とされる言語処理系や OS 機能の要件を絞り込み開発の枠組みを整備していくことが挙げられる.

なお，本開発は 2005 年度 IPA 未踏ソフトウェア創造事業で支援されている.

文　献

[1] 青山 幹雄, “ 組込みソフトウェアの転機, ” 情報処理, 39 巻，7 号，pp.689－692，July 1998.
[2] 安部田 章, “多層化による組込みソフトウェアの移植性の向上，”情報処理学会ソフトウェア工学研究会資料, no.140-16，pp.117－122，Mar 2003.
[3] 砂川 克志，岡本 康明, 最所 圭三, 福田 晃, “組込み機器向け OS の移植性を考慮した割込み機能の抽象化，” 情報処理学会システムソフトウェアとオペレーティング・システム研究会資料, no.82-5, pp.33－40, Aug 1999.
[4] 松井 俊浩, 比留川 博久, 石川 裕, 山崎 信行, 加賀 美聡, 堀 俊夫, 金広 文男, 斎藤 元, 稲邑 哲也, “ヒューマノイド・ロボットのための実時間分散情報処理”, 信学技法，CPSY2003-45，Mar 2004.
[5] Conrad, J.M.，Stiquito Controlled!，IEEE CS Press，New Jersey，2005.

（誤記修正済：
円周率 100 桁の計算に 16 秒→8 秒
PIC16F683→PIC12F683
）