システムソフトウェアと
オペレーティング・システム 85－15
（２０００．８．４）

組み込み向け OS における デバイスドライバの自動生成について

村上 智一[1]　片山 徹郎[1]　最所 圭三[2]　福田 晃[1]

家電製品や計測機器などに組み込まれているコンピュータ、いわゆる、組み込みシステムが重要視され、様々なシステムが開発されている。それに伴い、製品に組み込むソフトウェアの開発は短期間で行なう事が要求され、開発者にとって大きな負担となっている。組み込みシステムに OS を用いた場合、ソフトウェアの再利用性などが高まり、ハードウェアの変化への対応を集中させることによりで開発工数を削減することができる。我々の研究室では、マイクロカーネルを採用した組み込み向け OS を開発中であり、本稿ではこの OS におけるデバイスドライバの自動生成について検討し、生成のためのモデルを提案する。例として電車模型制御システムを取り上げ、2 つのデバイスドライバの自動生成を行なった。その結果、デバイスドライバ開発者の労力を削減し、提案したモデルの有効性を確認した。

Automatic Generation of Device Drivers for an Embedded Operating System

Tomokazu Murakami[1]　Tetsuro Katayama[1]　Keizo Saisho[2]　Akira Fukuda[1]

Embedded systems such as computers used in electrical appliances and instruments are considered. Various embedded systems are developed. Developers of software embedded in products have a burden because it is required to develop the software in a short period. When we use an operating system in an embedded system, the software can be reused and the burden in developing can be reduced by adapting the operating system to various kinds of hardware. In the authors' laboratory, an embedded operating system adopting micro-kernel structure is developing. This paper describes automatic generation of device drivers for the embedded operating system and proposes a model for the generation. As an example, a train model control system is chosen and two device drivers for the system are generated automatically. As a result, the burden in developing device drivers can be reduced and the proposed model are available.

[1]奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Instituite of Science and Technology
[2]香川大学 工学部 信頼性情報システム工学科
Department of Reliability-based Information Systems Engineering, Faculty of Engineering, Kagawa University

1 はじめに

マイクロプロセッサ技術の発達により、家電製品やモバイル機器、および計測機器に組み込まれているコンピュータ、いわゆる組み込みシステムの応用分野は拡大してきている。また、マイクロプロセッサの性能向上や、制御対象となる機器の高度化や複合化にともなって、今まではオペレーティング・システム (以下 OS) が使われなかったようなシステムにおいても OS が使われるようになってきている。

組み込みシステムは、汎用のコンピュータシステムと比較して、以下のような特徴がある [1]。

- 人間との直接のコミュニケーションインターフェイスを持たないか、あるいは、持っていてもコンピュータを意識させないものである。
- コストや設置スペースの都合で、プロセッサの性能やメモリの制約を受ける。
- アプリケーションは組み込む際に固定され、出荷後に変更されることはほとんどない。
- ハードウェアを制御したり、さらにそれを通して外部の環境を操作することが多く、リアルタイム性が重視されることが多い。
- 容易にシステムダウンをしないなどの高信頼性が求められる。

家電製品やモバイル機器などは、新製品の開発が頻繁に行なわれ、競争の激化にともない新製品の開発期間を短くすることが要求されている。そのため、製品に組み込むソフトウェアの開発も短期間で行なうことが要求され、開発者にとって大きな負担となっている。

また、8 ビット、あるいは 16 ビットワンチップマイコンなどを使用した、制御向け小規模組み込みシステムは、制御対象ハードウェアの多様性やリソース、開発コストの制約などを受ける。開発の際には、リソースの制約のために、生産性、および再利用性の低い旧来の形態をとらざるを得ないことが多い [2]。

組み込みシステムに OS を用いた場合、ソフトウェアの再利用性などが高まり、ハードウェアの変化への対応を集中させることにより開発工数を削減することができる。特に、組み込みシステムのアプリケーション側の開発工期の短縮に大きく貢献する [3]。このため、我々の研究室では、マイクロカーネル構成を採用した組み込みシステム向け OS を開発中である [4]。

本稿では、この OS におけるデバイスドライバの自動生成について検討する。具体的には、デバイスドライバの自動生成のためのモデルを提案する。デバイスドライバはコーディングが複雑である。このため、デバイスドライバを自動生成できれば、組み込みシステムを開発する際の負担を軽減させる効果が、非常に高いと考えられる。

ここで、今回生成の対象とするデバイスは、PCI(Peripheral Component Interconnect) や ISA(Industry Standard Architecture) などのバスを使わずに機器を直接制御するような、操作や機構が単純なデバイスとする。例として電車模型「N ゲージ」[5] を制御するシステムを取り上げ、電車の位置検出用デバイスドライバとポイント状態検出用デバイスドライバの自動生成を試みる。

2 章ではデバイスドライバの自動生成のためのモデルを提案する。3 章では電車模型における適用事例を説明する。4 章では議論と評価を行なう。

2 デバイスドライバ自動生成システム

2.1 システムの入力情報

デバイスドライバの自動生成のモデルを提案するために、デバイスドライバを構成する際に必要な情報について調査を行なった。その結果、デバイスドライバを作成するために必要となる情報を、以下の項目に分類した。

- OS とのインターフェイス

  デバイスドライバ内部のデータや状態を、OS が扱うデータ形式に変換して渡す部分の手続きを記述する。また、OS からのデータを受けとり、デバイスドライバ内部のデータ形式に変換する部分の手続きを記述する。さらに、デバイスドライバを OS へ登録する部分をは

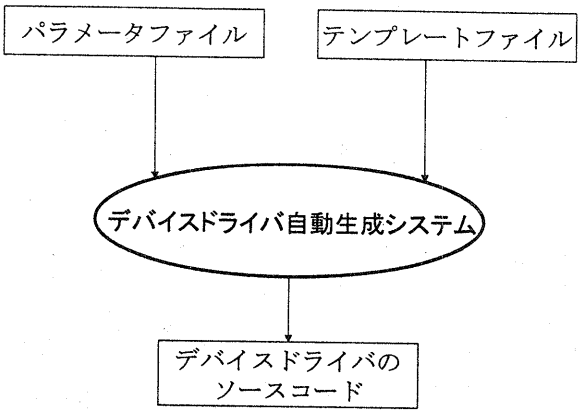

図 1: デバイスドライバ自動生成システムの概要

じめとする手続きである。OS の種類ごとに必要である。

- デバイスドライバの識別子

  OS に対してデバイスドライバは複数存在することが多いので、OS がそれらを区別するために記述する。デバイスドライバごとに必要である。コンパイル時にはデバイスドライバの関数名として、OS はメジャー番号として区別する。

- デバイス固有のパラメータ

  デバイスを操作するための I/O ポート番号や、デバイスのレスポンスなどのタイミングを記述する。デバイスの種類ごとに必要である。

- デバイスの操作手順

  デバイスで扱うデータを、デバイスドライバで扱いやすいように変換する手続きを記述する。また、デバイスドライバで扱うデータをデバイスで扱うデータに変換する手続きを記述する。さらに、デバイスをコントロールする手順を記述する。同系統の操作手順を持つデバイスごとに必要である。

以上の項目を基にして、自動生成システムのモデルを提案する。

2.2 システムのモデル

本稿で提案する自動生成システムのモデルの概要を図1に示す。システムの入力であるテンプレートファイルとパラメータファイルの構成は、下記の通りである。

- パラメータファイル
  - デバイスドライバの識別子
  - デバイス固有のパラメータ
- テンプレートファイル
  - OS とのインターフェイス
  - デバイスの操作手順

デバイスドライバ開発者は、まず、事前に用意したテンプレートファイルの中から OS やデバイスの操作手順に照らし併せて適切なものを選び出す。次に、デバイスドライバの識別子、デバイス固有のパラメータ、を書いたパラメータファイルを作成する。この 2 つのファイルを自動生成システムに与えることにより、デバイスドライバを作成する。

テンプレートファイルは、その役割から再利用性が高い方が望ましい。このため、デバイスドライバ自動生成システム開発者がテンプレートファイルを原則として記述し、一括して管理を行ない、

```
DEVICE_NAME location_dev
DEVICE_NUMBER 100
READ_PORT 0x8000
```

図 2: 電車の位置検出用パラメータファイル

デバイスドライバ自動生成システムの利用者、すなわちデバイスドライバ開発者に提供する。

このモデルでは、デバイスドライバ開発者がデバイスドライバを作成する際に必要なことは、適切なテンプレートファイルを選び出すことと、パラメータファイルを書くことだけである。その結果、デバイスドライバをスクラッチから書く場合に比べて、大幅に労力を削減できる。

2.3 パラメータファイル

パラメータファイルにおいて必要とされる項目には以下のようなものがある。

DEVICE_NAME デバイスドライバの名前 (デバイスドライバを構成する関数の名前)

DEVICE_NUMBER メジャー番号

READ_PORT 入力用 I/O アドレス番号

WRITE_PORT 出力用 I/O アドレス番号

DELAY_TIME ディレイ時間

パラメータファイルの例として 図 2を挙げる。これは、後述する電車模型制御システムの電車の位置検出用デバイスドライバのためのパラメータファイルである。

2.4 テンプレートファイル

テンプレートファイルの記述には、C 言語を拡張したものを用いる。これは、デバイスドライバ開発者が既に習得していると考えられるプログラミング言語であるとともに、テンプレートファイルからデバイスドライバを容易に生成するためである。また、既存のデバイスドライバからテンプレートファイルを作成することも容易となる。

```
void
%%%DEVICE_NAME%%%_main(T_RQIO *p_ReqIo)
{
  int minor;
  int access_addr;
  int value;

  minor = p_ReqIo->minor;

  if( minor < 0 || minor > MAX)
    return ERROR;

  access_addr =
    %%%READ_PORT%%% + (minor / 8);

  value = inb(access_addr);
  value &= 0xff;
  value >>= (minor % 8);

  if( (value & 0x1 ) == 1)
    value = ON;
  else
    value = OFF;

  *(p_ReqIo->recv_message) =
    value;
  *(p_ReqIo->status) = NOERR;

}
```

図 3: bit 情報を取り出すテンプレートファイルの一部

テンプレートファイルの例として図3を示す。これは、我々の研究室で開発している組み込み向け OS[4] を対象としたもので、デバイスから bit 情報を取り出すデバイスドライバのテンプレートファイルの一部である。この部分は実際にデバイスを操作する部分であり、初期化などの OS に登録する部分は除いている。デバイスからの bit 情報は図 4のように、8bit ごとにパックして格納されているものであり、このパックされているデータは I/O アドレス空間にマップされている。bit 情報はマイナー番号として識別される。

図 3のテンプレートファイルでは、以下の手順を記述している。

1. OS からマイナー番号を受け取る。

   このマイナー番号の受け取り方、関数の定義の仕方は OS によって変わるので、「OS との

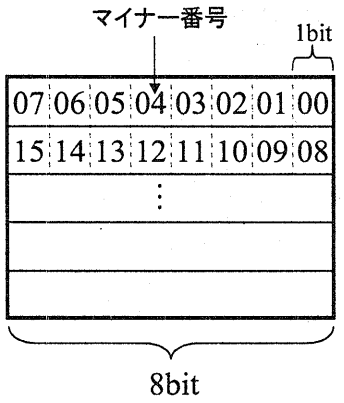

図 4: 8bit ごとにパックして格納されているデータ

インターフェイス」部分である。

2. マイナー番号の値を調べる。

   不正なマイナー番号が与えられた場合にエラーを返す。

3. マイナー番号を、そのマイナー番号が示すデータが存在する I/O アドレス番号に変換する。

   マイナー番号で表されるデータは 8bit ごとにパックされているので、マイナー番号を I/O アドレス番号に変換する。

4. I/O 命令を発行してデバイスから情報を得る。

   I/O 命令の発行の仕方は OS により異なるので、「OS とのインターフェイス」部分である。

5. 4. で得た情報の中から、マイナー番号で表されるデータの値について調べる。

6. 5. で調べた結果を OS に渡す。

   この渡し方は OS により異なるので、「OS とのインターフェイス」部分である。

このデバイスドライバの中で、「OS とのインターフェイス」部分以外の部分と、記述全体の流れとは、デバイスの操作を表しているので、「デバイスの操作手順」である。

以上の情報を基にして、デバイスドライバ自動生成システムはデバイスドライバを生成する。

3 電車模型における適用事例

3.1 電車模型制御システム

我々の研究室では、マイクロカーネルを採用した組み込み向け OS を開発している。この適用事例として、現在電車模型制御システムを用いている。これは、工作の必要が少ないことと、必要な部品を入手し易いことと、扱い易いことを考慮したためである。

この電車模型制御システムにおいて使用するデバイスは以下の通りである。

- ユーザ入力
- 電車の位置検出
- レールのポイント状態検出
- レールのポイント切替え
- 電車の速度制御

以上のデバイスは単純なものであり、デバイスコントローラが無く、入力と出力を同時に必要としないデバイスである。また、I/O 操作に関してディレイ時間を指定する必要がが無い。

この電車模型制御システムにデバイスドライバ自動生成システムを適用し、電車の位置検出用デバイスドライバと、レールのポイント状態検出用デバイスドライバを例として取り上げ、評価を行なう。

3.2 電車の位置検出、およびポイント状態検出における適用事例

デバイスドライバ自動生成システムの実例として、電車模型制御システムの電車の位置検出用のデバイスドライバの一部を挙げる。この部分は実際にデバイスを操作する部分であり、初期化などこのデバイスドライバを OS に登録する部分などを除いている。

電車模型「N ゲージ」ではレールは電車の通り道となるとともに、直流モーターへの電源供給にも使われている。レールは区間ごとに区切られており、その区間に電流が流れていると、その区間に電車が存在することとなる [4]。

```
void
location_dev_main(T_RQIO *p_ReqIo)
{
  int minor;
  int access_addr;
  int value;

  minor = p_ReqIo->minor;

  if( minor < 0 || minor > MAX)
    return ERROR;

  access_addr =
    0x8000 + (minor / 8);

  value = inb(access_addr);
  value &= 0xff;
  value >>= (minor % 8);

  if( (value & 0x1 ) == 1)
    value = ON;
  else
    value = OFF;

  *(p_ReqIo->recv_message) =
    value;
  *(p_ReqIo->status) = NOERR;

}
```

図 5: 電車の位置検出用デバイスドライバ

この区間 1 つに 1bit を割り当て、電車が存在するならば 1 を、存在しないならば 0 を返す。この情報を I/O アドレス空間 0x8000 から始まるマップ上に対応付けている。また、I/O アドレス 1 番地ごとに 1bit 割り当てるのは I/O アドレス空間資源の無駄であるので 8bit ごとにパックして割り当てている。

このデバイスドライバではまず、デバイスドライバの種類として 100 番のメジャー番号を割り当てている。次に区間の種類としてマイナー番号を割り当てている。

そこで、図 2 の電車の位置検出用パラメータファイルを記述するとともに、図 3のテンプレートファイルを自動生成システムの入力として用いると、図 5 の電車の位置検出用デバイスドライバが自動生成される。

さらに別の事例として、レールのポイント状態検出を行なうデバイスドライバの作成を検討する。

```
DEVICE_NAME point_dev
DEVICE_NUMBER 101
READ_PORT 0x9000
```

図 6: ポイント状態検出用パラメータファイル

ポイントの状態は定位であるか、反位であるかの 1bit で表される。ここで、定位というのは通常セットしてあるポイントの状態のことである。反位は通常セットしてある定位の状態から切替えたポイントの状態である。このポイント 1 つに 1bit を割り当て、定位であるならば 0 を、反位であるならば 1 を返すとする。この情報を I/O アドレス空間 0x9000 から始まるマップ上に対応付けるとする。また、I/O アドレス 1 番地ごとに 1bit 割り当てるのは I/O アドレス空間資源の無駄であるので図 4のように 8bit ごとにパックして割り当てる。

このデバイス用のデバイスドライバにおいて、電車の位置検出用のデバイスドライバと違うのは、デバイスドライバの識別子、デバイス固有のパラメータだけであり、OS とのインターフェイスとデバイスの操作手順は変わらない。このことは、電車の位置検出用のデバイスドライバに用いたテンプレートファイルが、ポイントの状態検出用のデバイスドライバにも用いることができることを意味する。

そこで、図 6 のようなポイント状態検出用パラメータファイルのみを新たに記述して、電車の位置検出用に用いたテンプレートファイルを自動生成システムの入力として用いると、ポイント状態検出用デバイスドライバが自動生成される。

現在のデバイスドライバ自動生成システムの実装においては、図 2のテンプレートファイルと出力される図 5デバイスドライバの例を見比べると明らかなように、テンプレートファイル中のキーワードをパラメータファイル中のパラメータで置換しているだけである。このように、パラメータファイルに記述するパラメータが、語句や数値だけで与えられるような単純なデバイスであれば、キーワードを置換するだけでデバイスドライバを生成できる。

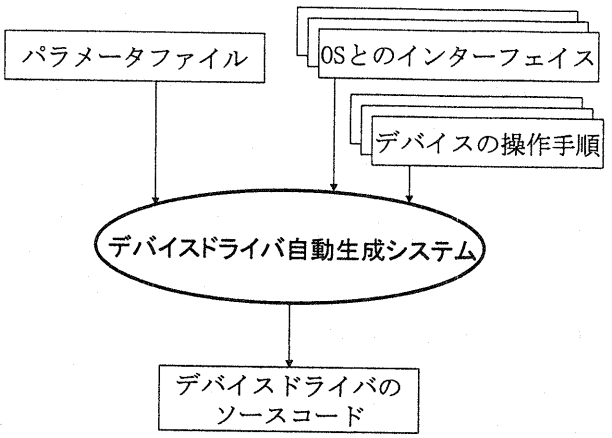

図 7: テンプレートファイルのモジュール化

4 議論および評価

4.1 モデルの有効性について

本稿ではデバイスドライバ自動生成のためのモデルを提案した。しかし、1つのテンプレートファイルを作成する労力は、1つのデバイスドライバを作成する労力と実際にはほとんど変わらない。

今回の適用例では、1つのテンプレートファイルで、電車の位置検出用デバイスドライバと、レールのポイント状態検出用デバイスドライバの2つのデバイスドライバを生成することができた。このことは、1つのデバイスドライバを作成する労力で、2つのデバイスドライバを生成できたことを意味する。よって、デバイスドライバを作成する労力が約半分に削減できた。さらに同系統の操作手順を持つ別のデバイスにおいて同じテンプレートファイルが適用できるならば、さらに労力が削減できることになる。

また、テンプレートファイルの作成者とデバイスドライバ開発者が別ならば、デバイスドライバ開発者の労力は適切なテンプレートファイルを選ぶことと、パラメータファイルを記述することのみである。このことは、デバイスドライバ開発者の労力を大幅に削減することになる。以上から、このモデルの有効性が確かめられた。

4.2 他の OS への適用

現在のデバイスドライバ自動生成システムは、我々の研究室で開発している組み込み向け OS を対象としている。そこで、この OS の代わりに ITRON [6] のような他の組み込み向け OS を用いる場合を考えてみる。デバイスドライバ自動生成のモデル上において他の OS への適用は、その OS とのインターフェイス部分さえ書き換えれば可能となる。すなわち、OS が異なれば OS とのインターフェイスが異なるために、新たにテンプレートファイルを作成しなければならない。その労力は新たにデバイスドライバを作成する場合とほとんど変わらない。しかし、他の OS 用のテンプレートファイルを蓄積していくことにより、その OS を用いた組み込みシステム開発の効率を高めることができる。

4.3 テンプレートファイルのモジュール化

デバイスドライバ開発者は、自動生成システム開発者が用意したテンプレートファイルの中から OS やデバイスの操作手順に照らし併せて適切なものを選び出す。次にデバイスドライバ開発者は、デバイスドライバの識別子、デバイス固有のパラメータ、を書いたパラメータファイルを作成する。この2つのファイルを自動生成システムに与えることによりデバイスドライバを作成する。

現在のデバイスドライバ自動生成システムでは、1つのデバイスドライバを生成するためには、テンプレートファイルが1つ必要である。そのためにデバイスの操作手順やOSとのインターフェイスが変わるごとにテンプレートファイルを用意する必要があり、テンプレートファイルの数が膨大なものになる。このため、デバイスドライバ開発者は、適切なテンプレートファイルを選ぶことが困難になる。また、自動生成システム開発者もテンプレートファイルの用意に膨大な労力が必要となる。

そこで、図7のようにこのテンプレートファイルをモジュールごとに分割し、適切なモジュールを組み合わせることによって、デバイスドライバを生成することを考えている。

例えば、テンプレートファイルを、OSとのインターフェイス部分と、デバイスの操作手順部分とに分割することにより、テンプレートファイルの見通しも良くなり、適切なモジュールを選びやすくなる。このようにすれば、テンプレートファイルの数が減少し、テンプレートファイルの再利用性も高まる。また、OSとのインターフェイス部分、デバイスの操作手順部分のそれぞれを、より細かく分割することによって再利用性がさらに高まると考えられる。

5 おわりに

本稿では、組み込みシステム開発の負担を軽減することを目的として、組み込み向けOSにおけるデバイスドライバの自動生成について検討し、生成のためのモデルを提案した。例として電車模型制御システムを取り上げ、電車の位置検出用デバイスドライバと、レールのポイント状態検出用デバイスドライバの、2つのデバイスドライバの自動生成を行なった。その結果、デバイスドライバ開発者の労力を削減し、デバイスドライバ自動生成のためのモデルの有効性を確認した。

今後の課題として以下の項目がある。

- テンプレートファイルのモジュール化

  現在のデバイスドライバ自動生成システムでは、テンプレートファイルの数が膨大なものになる。それを解消するためにテンプレートファイルをモジュール化し、再利用性の向上と見通しをよくする事を目指す。

- デバイスドライバ記述スクリプト

  テンプレートファイルが自動生成システム開発者によって用意されていない場合には、デバイスドライバ開発者がテンプレートファイルを作成する必要がある。このテンプレートファイルを作成する労力は、デバイスドライバを作成する労力と変わらない。そのために、デバイスドライバ記述スクリプトを作成し、テンプレートファイルの作成を支援することで、デバイスドライバ開発者の負担を軽減させる。

- 適応範囲の拡大

  今回は、デバイスドライバの自動生成システムを、パラメータファイルに記述するパラメータが語句や数値だけで与えられるような単純なデバイスのみに適用した。今後はより複雑なデバイスへの適用も考えている。

参考文献

[1] 中本幸一, 高田広章, 田丸喜一郎: "組込みシステム技術の現状と動向," 情報処理学会誌, Vol.38, No.10, pp.871-878, 1997.

[2] 西山直希, 片山徹郎, 最所圭三, 福田晃: "組み込みシステムにおける機器制御ライブラリの生成支援," 情報処理学会研究報告, 98-OS-79, pp.69-76, 1998.

[3] 福田晃, 最所圭三, 片山徹郎, 中西恒夫 : "組込システム向け実行環境の自動生成 δ プロジェクトの構想," 電子情報通信学会技術研究報告, CPSY99-125, pp.17-22, 2000.

[4] 森若和雄, 久住憲嗣, 中西恒夫 , 片山徹郎, 最所圭三, 福田晃: "電車模型制御用ソフトウエアシステムの設計," 情報処理学会研究報告, 2000-OS-84, pp.55-61, 2000.

[5] 株式会社トミー: "トミックス総合カタログ 1998-1999," 1999.

[6] トロン協会 ITRON 部会: "ITRON Project Home Page," http://www.itron.gr.jp/ .