システムソフトウェアと
オペレーティング・システム 79－10
（1998. 8. 6）

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

西山 直希　片山 徹郎　最所 圭三　福田 晃

奈良先端科学技術大学院大学
情報科学研究科
〒630-0101 奈良県生駒市高山町 8916-5
E-mail:{naoki-n, kat, sai, fukuda}@is.aist-nara.ac.jp

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。また併せて、組み込みシステムに特化した専用のソフトウェアも作成されている。本稿では、組み込みシステムの制御ソフトウェア開発の際にかかる負担を軽減するために、機器制御ライブラリの生成支援を目指す。ここでいう、機器制御ライブラリとは、機器を直接制御するコードであり、低レベルデバイスドライバと呼ぶ。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におけるアプリケーション開発を支援するため、低レベルデバイスドライバの生成システムを提案し、システムの入力形式、および生成自動化の可能性について考察する。

Support to Generate Libraries of Devices for Embedded Systems

Naoki Nishiyama　Tetsuro Katayama　Keizo Saisho　Akira Fukuda

Graduate School of Information Science,
Nara Instituite of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan

Embedded systems such as computers used in electrical appliances and instruments are considered. Various embedded systems are developed, and software dedicated to them is also. This paper shows support to generate libraries of devices to reduce the burden in development of software to control an embedded system. A library of devices is defined as a set of codes to directly control a device and is called as a low-level device drivers. A low-level device driver generation system is proposed in order to support development of an application program for a small scale embedded system with an 8-bit/16-bit microcomputer. Input forms of the low-level device driver generation system and the possibility of automatic generating the low-level device driver are examined.

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、組み込みシステムの応用分野は拡大しており、身の回りのほとんどの電子機器に組み込まれるようになってきている。また、制御対象となる機器の高性能化や複合化にともなって、組み込みシステムの大規模化・複雑化も著しい。さらに、機器のデジタル化も、組み込みシステムの重要性を増している [2]。

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。

- 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。

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く、開発環境がそれほど整備されていないことが多い。

我々の研究室では、オペレーティング・システム (以下 OS) の自動生成の可能性を追求している。OS の中で自動生成の効果が最も高いと考えられる、コーディングが複雑なデバイスドライバに着目して、デバイスドライバ生成システムを提案している [1][3][6]。

本研究では、小規模組み込みシステムの開発を支援する方法の一つとして、制御ソフトウェア開発の際の負担を軽減するために、機器を直接制御するライブラリ (低レベルデバイスドライバ) の自動生成を目標とする。ここで、小規模のシステムとは、主に、8ビット、あるいは16ビット CPU(含むワンチップ) を用いており、コードサイズは64K バイト未満で、既存のパッケージチップを使用したハードウェア設計を行なうようなシステムである。このようなシステムにおいて、低レベルデバイスドライバ生成支援を提案し、多様なハードウェアに対応できる支援システムの条件、またデバイスドライバ生成自動化の可能性について考察を行う。

2節では、低レベルデバイスドライバについて説明する。3節では、デバイスドライバの生成システムを提案し、A/D コンバータを例にとり、システムの入力について考察する。4節では、議論および考察を行なう。

2 低レベルデバイスドライバ

2.1 組み込みシステムと OS

ある組み込みシステムに特化した、そのシステム用 OS の研究・開発は、現在盛んに行なわれている [4][5]。しかしながら、組み込みシステムは OS の採用がまだまだ行われていない分野である。その理由として、以下が挙げられる。

- 低速なワンチップ 4 ビットや、8ビットマイコンを使用したようなシステムなどに、既存の組み込みシステム向け OS を組み込むには、メモリ、あるいはプロセッサの性能が不足である。
- システムに対して、非常に短期間での試作設計が求められてたり、また、要求される仕様が単純であるにも拘らず、ハードウェアが非標準であることが多い。すなわち、既存 OS の移植作業よりも、本体アプリケーション開発の工数に重点を置かざるをえない状況である。
- システムにリアルタイム性が重視されることが多いのだが、組み込みリアルタイム OS を取り扱える技術者が不足している。

OS 内のデバイスドライバの作成は開発工数の大きな部分を占めており、非常に時間と労力が費されている。OS を載せない場合であっても、デバ

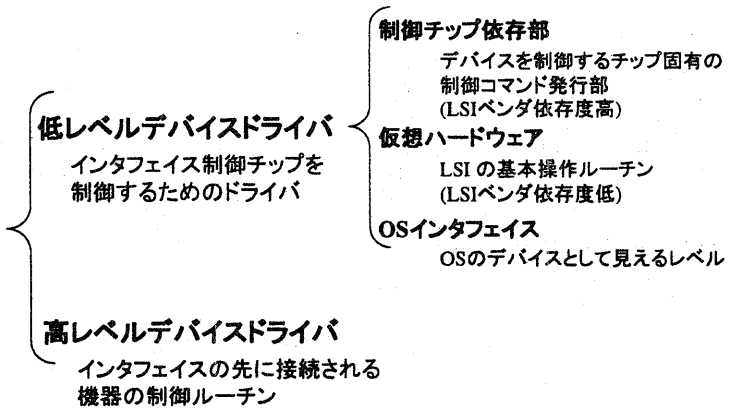

図 1: デバイスドライバの階層

イス制御のルーチンは作成しなければならないので、同じことが言える。すなわち、デバイスドライバ部は、システムの手足となる箇所であり、このデバイスドライバを自動生成することにより、システム開発における費用削減が見込まれる。さらに、デバイスドライバ部を部品化することができれば、再利用の可能性も見込まれる。

2.2 デバイスドライバの階層

我々は、デバイスドライバを、図 1 のように、区分する。

- 低レベルデバイスドライバ
  インタフェイス制御チップを制御するためのドライバ。以下のような階層に区分できる。
  - 制御チップ依存部 — デバイスを制御するチップ固有の制御コマンドなどを指す。デバイスとして、SIO(Serial I/O Interface) や、SCSI(Small Computer System Interface)、A/D コンバータ、モータコントローラなどがある。制御コマンドはチップの製造メーカによってまちまちで、ほとんど統一性がない。
  - 仮想ハードウェア — チップによって制御方法はまちまちであるが、同じ機能を提供するチップに実装されている基本命令 (プリミティブ) は、当然ながら類似している (例えば、SIO のチップなら 1 バイト受信、送信、受信チェックなど)。この層によってチップの製造メーカなどによる違いを吸収できるので、この層を通してデバイスの統一性が図れる可能性がある。
  - OS インタフェイス — OS のデバイスとして見えるレベルを指す。たとえば UNIX のような汎用 OS ではキャラクタデバイスを仮想的にファイルとして見せている。
- 高レベルデバイスドライバ
  SIO や SCSI インタフェイスの先に接続される機器 (モデム、プリンタ、スキャナなど) を制御するためのルーチンを指す。

本研究では、主に低レベルデバイスドライバを取り扱う。

従来の少規模組み込み機器向けアプリケーションの開発において、ハードウェアの仮想化を行わずにアプリケーションを作成した場合、制御チップ依存部のみを製作する。このため、作成したアプリケーションの性能は高いが、柔軟性、および、再利用性に著しく欠ける。

これに対して、ライブラリ形式での再利用を行なう開発は、プロセッサ (スピードや、バス幅、I/O

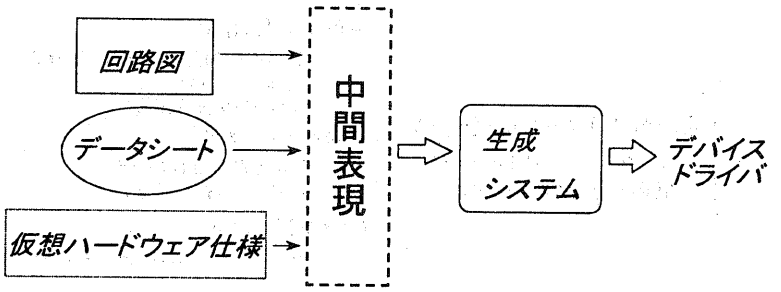

図 2: デバイスドライバ生成システムの概要

Mapping など) の違いを吸収することが困難である。汎用性を増したライブラリは、パラメータが膨大になる。また、プログラム中にシステムに関しての多くの分岐 (例えば #ifdef など) が存在することになり、プログラムのサイズも大幅に増える。

以上のことから、組み込み機器向けアプリケーションの開発の際には、ハードウェア構成の違いを、どこでどう吸収するか、が問題となる。次節では、ハードウェア構成を仮想化したデバイスドライバの生成について述べる。

3 デバイスドライバ生成システム

我々は、ハードウェア構成を仮想化したデバイスドライバの生成について検討する。理想的には、人間が開発する際に用いる資料から、デバイスドライバを生成するシステムを目指す。また、再利用性を考慮し、図 2 のようなデバイスドライバ生成システムを提案する。システムの入力として、回路図、仮想ハードウェア仕様、および、データシートの 3 つを考える。

- 回路図 — プロセッサから見た、アクセスに必要な情報を付加したブロックダイアグラム (例えば、ブロックダイアグラム記述言語) である。
- データシート — チップの仕様書である。
- 仮想ハードウェア仕様 — 仮想デバイスとして扱うために必要なプリミティブの集合と、それぞれのプリミティブに必要な情報を表すプロパティの集合とから成る。プリミティブとは、あるデバイスに対する基本操作のことであり、分割することなしに実行可能な操作 (すなわち、内部に長時間の待機などを含まない) を指す。プリミティブ本体の生成には、デバイスとの接続情報である、デバイスアドレス (信号線) や、データ幅、タイミングに関する情報、などを取得しなければならない。これらの接続情報は、回路図から取得する。プロパティは、プリミティブ内で必要となる情報を指す。また、デバイスのビット数などのプリミティブに隠蔽できない性質のもので、プリミティブを使ったデバイスドライバの記述に必要となる値も、プロパティに含む。プロパティの実際の値は、データシートから読み込むことにより、生成する。

回路図、データシート、および、仮想ハードウェア仕様から、ドライバ生成に必要な最低限情報のみを何らかの形で取り出す。このため、デバイスドライバの生成に必要な情報を一旦何らかの中間表現に直し、それからドライバを生成する。

例として、A/D コンバータを取り上げる。図 3 は、A/D コンバータのブロックダイアグラムである。A/D コンバータのアクセス手順は、以下に示す通りであり、Convert, isReady および Datain の 3 つがプリミティブにあたる。

1. A/D コンバータに変換開始指示を出す (Convert プリミティブ) — CONV を一定時間アクティブにする。

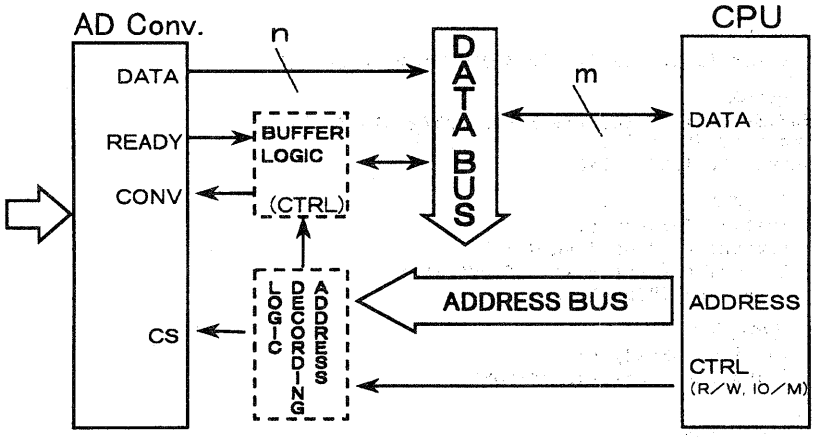

図 3: A/D コンバータ

(1) 特定のアドレスの特定のビットをアクティブにする

(2) 短い時間待つ (通常数百 *ns* 以上数百 *ms* 未満)

(3) (1) で反転したビットを元に戻す。

この Convert プリミティブでは、必要なプロパティは以下となる。

- 特定のアドレスとはどこか
- 特定のビット位置とはどこか
- アクティブは 0 と 1 のどちらか
- 短い時間とは具体的にどのくらいか

2. A/D コンバータの変換終了を確認する (isReady プリミティブ) — READY がアクティブであることを確認する

(1) 特定のアドレスのデータを読む

(2) データの特定のビットの状態で、終了を確認

この isReady プリミティブでは、必要なプロパティは以下となる。

- 特定のアドレスとはどこか
- 特定のビット位置とはどこか
- 終了を示すのは 0 と 1 のどちらか

3. A/D コンバータから変換結果を受け取る (Datain プリミティブ) — A/D コンバータにおける CS がアクティブになるアドレスからデータを読み込む

(1) 特定のアドレスのデータを読む

この Datain プリミティブでは、必要なプロパティは以下となる。

- A/D コンバータにおける CS がアクティブになるアドレスとはどこか
- 読んだデータの有効幅はどのくらいか

中間表現として、シンボル処理を用いた記述を用いる。これは、回路図における接続情報と各デバイスの機能記述とを基に、チップを動作させる条件などを走査することを考えているからである。現在は、図 4 のような、S 式での表現を試みている。但し、図 4は、図 3のブロックダイアグラムを記述したものであり、このダイアグラムが回路へのアクセスに必要な情報を持たないため、完全に記述できていない。

同じプリミティブ定義を持ち、かつ、同じ、プロパティの集合をもつデバイスを同クラスのデバイスと定義する。同クラスのデバイスであっても、プロパティの値は違っていて構わない。提案するシステムを用いてあるデバイスを作成し、次に、作成したデバイスと同クラスのデバイスを作成す

```
;;
;; コンバータ
(define_class ADConvSample (DATAWIDTH CONVPULSELEN)
     ;;
     ;; 信号線
     (define DATAOUT (create_lines DATAWIDTH "D"))
     (input "CONV" "CS")
     (output DATAOUT "READY")
     ;;
     ;; 変換要求
     (primitive Convert ()
        (:require ("CONV" *HIGH*))
        (:require (:interval CONVPULSELEN))
        (:require ("CONV" *LOW*)) ) )
     ;;
     ;; 変換終了?
     (primitive isReady ()
        (:provide (:results "READY"))
     )
     ;;
     ;; データ取得
     (primitive Datain ()
        (:require ("CS" *HIGH*))
        (:provide (:results DATAOUT))
     )
)
;;
;; バッファ
(define_class Buffer (DATAWIDTH)
     (define DATAIN  (create_lines DATAWIDTH "DI"))
     (define DATAOUT (create_lines DATAWIDTH "DO"))
     ;;
     (input DATAIN)
     (output DATAOUT)
     ;;
     (primitive Through (data)
        (:require ("EN" *HIGH*) (:args DATAIN data))
        (:provide (:results DATAOUT data))
     )
)
;;
;; CPU
(define_class CPUSample (DATAWIDTH ADDRESSWIDTH)
     (define DATA     (create_lines DATAWIDTH "D"))
     (define ADDRESS  (create_lines ADDRESSWIDTH "A"))
     ;;
     (input DATA)
     (output DATA ADDRESS "CS" "RW")
     ;;
     (define_code read (indata)
        (output "in @indata,reg0")
     )
     ;;
     (primitive Read (address)
        (:provide ("CS" HIGH) ("RW" HIGH) (:require address))
        (:require (:args INDATA) :codeout (read INDATA))
     )
)
```

図 4: S 式で記述した中間表現

る際には、まったく同じ仮想ハードウェア仕様を、入力として使用できる。このため、再利用が可能となり、生産性の向上が見込まれる。

4 議論および考察

この節では、ハードウェアの仮想化、システムの入力、プリミティブの生成、中間表現について、それぞれ議論および考察を行なう。

4.1 ハードウェアの仮想化について

デバイスドライバの生成に関して、I_2O (Intelligent I/O) SIG[7] では、OS と デバイスドライバとの間に標準のインターフェイス I_2O を定めている。I_2O の仕様では、デバイスドライバを、OS に依存した OSM (OS Specific Module)、ハードウェアに依存した HDM (Hardware Device Module)、OSM と HDM との間で情報を送受する Messenger の、3 つの階層に分ける。OSM と HDM 間の通信に使うパケットの形式を明確に定義することで、OS が異なっても一つのデバイスに対して、HDM 自体を書き変えることなく通信が可能となる。

I_2O の手法では、I/O デバイスの側を賢くすることで仮想化を動的に解決している。すなわち、以下のような特徴が挙げられる。

- 開発の際に考慮していなかったようなハードウェアの拡張に対しても、対応が可能である。
- 仮想化をハードウェア (HDM の側) にも分担させることで、OS 側のデバイスドライバ開発者の負担が軽減する。

これに対して、本手法の特徴として、以下のことが挙げられる。

- 仮想化をターゲットハードウェアや OS で行うのではなく、開発プロセスで行う。
- 低レベルデバイスドライバを自動生成することで実現したため、ハードウェアに関する知識が少なくても作成できる。また、デバイスやチップ独特の問題について考慮する必要がなくなる。

また、本手法の欠点としては、以下のことが挙げられる。

- 自動化の際に、生成に必要なデータを何らかの形で用意しなければならない。理想的には、チップのメーカが電子データシートとして供給してくれることが望ましい。
- データのないまったく新しいハードウェアを開発する場合には、手間がかかる。しかしながら、再利用が可能である。
- 開発の際に考慮していないハードウェアの拡張には対応できない。このため、ハードウェアの構成が変更されるたびに、システムごと入れ替える必要がある。しかしながら、小規模システムでは、このような場合自体が考えにくい。

4.2 システムの入力について

デバイスドライバ生成システムの入力として、回路図、データシート、および、仮想ハードウェア仕様を考えた。

回路図は、プロセッサからみたアクセスに必要な情報を付加したブロックダイアグラムである。仮想ハードウェア仕様は、仮想デバイスとして扱うのに必要なプリミティブの集合と、それぞれのプリミティブに必要な情報を表すプロパティの集合とから成る。データシートは、チップの仕様書であり、プロパティの実際の値を埋め込む。

但し、これらの入力だけでは決定できない事項がある。例えば、Wait などの表現や、プロセッサによる相違などが挙げられる。解決策として、Wait については、システムのスケジューリングポリシーに影響を与えるような長いものについては、プリミティブには含まずに、上位層に含むことにし、プリミティブ内では最小待ち時間のみを保証するように設定する方法が考えられる。また、プロセッサによる差異の吸収については、デバイスだけでなく、プロセッサのバス幅やクロックスピードなどもデータシートとして与え、アセンブリ言語レベルの差異の吸収は、生成コードを C 言語レベルにすることで吸収する方法が考えられる。

4.3 プリミティブの生成

プリミティブ本体の生成には、デバイスとの接続情報である、デバイスアドレス (信号線) や、データ幅、タイミングに関する情報、などを回路図から取得しなければならない。具体的には、信号線の接続表現を与えて、そこから接続情報を取得する。もしくは、プリミティブの実行の際に必要な条件を何らかの表現で記述しておき、それを満たすために必要な操作の集合を求める。データ幅、制御に関連する時間などはプロパティとして与え、

プリミティブ生成時にプロパティを参照して解決する。

しかしながら、回路図を公表してない、もしくは、特許などの関係で回路図を入手できない場合がある。この場合は、回路にアクセスする方法が明確に与えられなければならない。すなわち、プログラム中でのアドレスの求め方が分からなければ、生成することができない。

4.4 中間表現について

提案したデバイスドライバ生成システムでは、回路図、データシート、および、仮想ハードウェア仕様から、デバイスドライバの生成に必要な情報を一旦何らかの中間表現に直す。中間表現の作成支援としては、以下のことを考えている。

- ブロックダイアグラムやネットリストなどの回路の接続構成情報から、ドライバの生成に必要な情報 (使用するチップの種別や、バスなどとの接続形態など) を取り出すシステム
- チップのデータシートを見ながら、質問に答えてゆくことでデバイス依存情報 (擬似コード) を生成するシステム

また、中間表現としては、シンボル処理を用いた。これは、回路図における接続情報と各デバイスの機能記述とを基に、チップを動作させる条件などを走査することを考えているからである。現在は、S 式での表現を試みたが、完全に記述できていない。デバイスドライバ生成に必要十分なデバイス情報の定義や、デバイスアドレス、データ接続についても、同様のシンボル処理によって解決する方針である。

5 おわりに

本稿では、小規模組み込みシステムの開発を支援する方法の一つとして、制御ソフトウェア開発の際の負担を軽減することを目指し、機器を直接制御するライブラリ (低レベルデバイスドライバ) の生成システムを提案した。また、多様なハードウェアに対応できるシステムの条件や、デバイスドライバ生成自動化の可能性について、手順、および、入力について考察を行なった。例として、A/D コンバータを取り上げ、システムの入力形式について検討した。

今後の課題として、以下の項目がある。

- デバイス情報の表現
  提案したデバイスドライバ生成システムでは、回路図、データシート、および、仮想ハードウェア仕様から、デバイスドライバの生成に必要な情報を一旦何らかの中間表現に直す。中間表現として、シンボル処理を用いた記述を用いた。しかしながら、現在はまだ一部しか実現していないので、完全な記述を目指す。
- 処理系の作成
  今回は、デバイスドライバ生成自動化の可能性について、手順、および、入力形式について考察を行なった。作成した記述を処理できる処理系を作成して、実際の開発への適用・評価し、提案した方式の有効性を検証しなければならない。

参考文献

[1] 片山徹郎, 最所圭三, 福田晃: "デバイスドライバの自動生成に向けて –プリンタデバイスの生成に関する考察–," 情報処理学会研究報告, 97-OS-76, pp.43-48, 1997.

[2] 中本幸一, 高田広章, 田丸喜一郎: "組み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.

[3] 片山徹郎, 最所圭三, 福田晃: "デバイスドライバの仕様記述に適した言語の開発について," コンピュータシステムシンポジウム '97 論文集, pp.53-58 1997.

[4] 岩崎裕江, 長沼次郎, 遠藤真: "オンチップリアルタイム OS の構成法," 情報処理学会研究報告, 98-OS-77, pp.49-54, 1998.

[5] 藤井茂樹, 中谷信太郎, 松本正治, 平井誠, 清原督三: "メディア処理用組み込み OS," 情報処理学会研究報告, 98-OS-77, pp.61-66, 1998.

[6] T. Katayama, K. Saisho and A. Fukuda: "A Method for Automatic Generation of Device Drivers with a Formal Specification Languag," Proc. Int'l Workshop on Principles of Software Evolution (IWPSE98), pp.183-187 1998.

[7] I_2O Special Interesting Group(SIG): http://www.i2osig.org/