第4章 ITRONのデバイス制御フレームワーク

ITRON仕様におけるデバイス・ドライバ構想

組み込み向けOSであるITRONでは，メモリ容量の制限やリアルタイム性への要求から，デバイス・アクセス部をアプリケーション・プログラム内に記述することが多かった．しかし現在，トロン協会 デバイス・ドライバ検討会では，ITRONにおいてもデバイス・ドライバの枠組みを導入し，移植性や再利用性を高めている．本稿では，ITRON仕様におけるデバイス・ドライバ構想について，実際のプログラムを示しながら解説する．（編集部）

金田一 勉，宮下 光明

1. ITRON 仕様策定の背景

ITRONは，1984年に組み込み機器向けのリアルタイムOS（RTOS）規格として策定され，その仕様がオープンになっています．当時の組み込み機器はCPUの性能が低かったり，搭載されているメモリ容量が小さいといった制限がありました．そのようなハードウェアにおいては，リアルタイム処理の制約に対応し，しかもマルチタスク管理を行うOSを搭載するのは困難であると思われていました．

ITRON仕様では，そのような環境でもリアルタイムOSとして満足できる機能を提供できるように考慮されています．ただし，組み込み機器に採用されるCPUのアーキテクチャはさまざまです．それらのアーキテクチャへの適合方法は仕様では特に触れられておらず，個々のハードウェアに対する合わせ込みはリアルタイムOSの実装者に委ねられています．

ITRON仕様は，時間の経過に伴って機能を増やしながら仕様のバージョン・アップが行われています．これは，技術の進歩によりCPU性能が向上し，複雑な機能を仕様として規定できる余裕が出てきたためです．

ITRON仕様の骨子は，リアルタイム性とマルチタスク機能の提供です．例えば，スケジューラの動作を規定することでリアルタイム性を確保するようにしています．また，複数のタスクを管理し，タスク間の事象発生通知および情報通知の機能を規定することで，マルチタスクで動作する環境を規定しています．

最近の組み込みシステムは大規模化する傾向にあり，システムの開発期間の短縮も求められています．それとは別に，システムの品質向上も求められています．組み込みシステムを開発する際には，「急ぎながら慎重に開発を進めていく」ことになります．

このような背景から，最近ではプロジェクト管理やソース・コード検証といったことが開発プロセスの中で行われるようになってきています．ソフトウェア開発の立場からは，ソフトウェアの再利用が有効な手段の一つとして考えられています．一度開発を行い，動作検証済みのプログラムを部品として再利用することにより，開発や，動作検証の工数を短縮できます．

組み込みシステムでは，必ずハードウェアを制御するプログラムを開発します．前述したように，組み込み機器で採用されるCPUのアーキテクチャは多様になっています．ハードウェアを制御するソフトウェアを部品として再利用できるようにすることは，有益だと思います．ハードウェアに対してアクセスするソフトウェアは，汎用OSでは「デバイス・ドライバ」として共通化され，再利用が進められています．当初のITRONではデバイス・ドライバは規格化されていなかったのですが，上記のような流れから，ITRONでもデバイス・ドライバを導入する必要性が高まっています．

そのために導入された概念がDIC（Device Interface Component）です．これは，デバイスにアクセスするプログラムを，「OSに依存する部分」，「ハードウェアに依存する部分」，「アーキテクチャに依存する部分」に分け，異なるハードウェア環境で利用する場合に，実装のための修正が少なくて済むように規定されたものです．

KEYWORD ── ITRON，デバイス・ドライバ，デバイス・ドライバ設計ガイドライン，DICアーキテクチャ，GDIC，PDIC，SIL，ThreadX-μITRON

トロン協会では，デバイス・ドライバ検討会を開き，1999年に「デバイス・ドライバ設計ガイドラインWG中間報告―DICアーキテクチャの提案―」を発表しました．それを実際のデバイスに当てはめ，具体的な機能を検討し，「組込みシステムにおけるPDIC機能ガイドライン―DICアーキテクチャの導入―」を2004年に発表しています．

2. ITRON仕様のデバイス制御プログラム

ITRON仕様を利用するシステムの場合，タスクからデバイスのレジスタに対して制御を行い，割り込みサービス・ルーチンで割り込み処理を行い，事象をタスクに通知するという流れが一般的です(**図1**)．

ITRON仕様では，デバイスを制御するプログラムを特別に管理する機能はなく，汎用OSのデバイス・ドライバのようなものは規定されていません．DICは，ITRON仕様でも汎用OSでも利用できるように，OSが持っている機能に依存する部分を分離するような構造になっています．

そして，組み込みシステムでよく利用されるデバイス(例えばシリアルやEthernet)を選択し，利用する方法を規定しています．

DICはデバイスを制御するプログラムの基本構造です．デバイスを利用する上位層の処理，例えばファイル・システムやTCP/IPなどのプロトコルのプログラムは，アプリケーション側で用意する必要があります．

3. DICの構造

DICは，次の三つから構成されています(**図2**)．

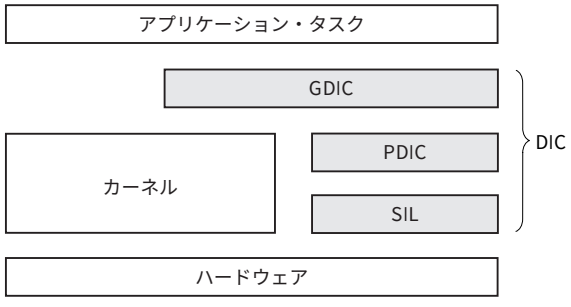

図2　DICの位置付け

DICは，GDIC，PDIC，SILで構成される．

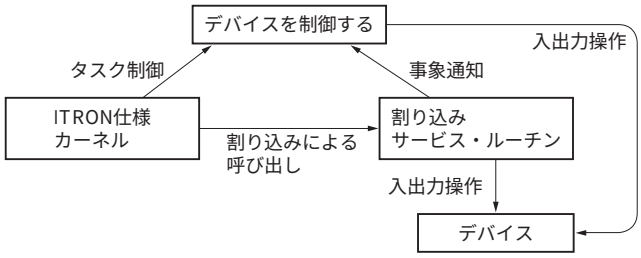

図1　ITRON仕様カーネルを利用する場合の環境

ITRON仕様カーネルの制御を受けるタスクでデバイスへの操作を行い，割り込みサービス・ルーチンで割り込み処理を行う．事象(割り込み発生，もしくは入出力の完了通知など)を，OSの機能を利用してタスクに通知する．

- GDIC(General DIC)
- PDIC(Primitive DIC)
- SIL　(System Interface Layer)

GDICは，タスクからの要求を受け付ける部分で，動作するOS環境に依存する処理になります．

PDICは，制御対象とするデバイスを制御する処理を行います．ガイドラインでは，デバイスを機能別に分類し，その種別ごとにインターフェースを規定しています．

SILは，ハードウェアのアーキテクチャなどに依存する部分です．ハードウェアの違いを吸収することで，PDICを各種ハードウェアで流用できるようにしています．

● OS依存の処理を行うGDIC

GDICは，OSに依存する処理を行う部分です．一つのデバイスを複数タスクで利用する場合(複数タスクでシリアルにログ・メッセージを出力するなど)の排他制御(**図3**)，および入出力要求の完了を待つ場合の事象待ち処理(**図4**)などの機能を持ちます．

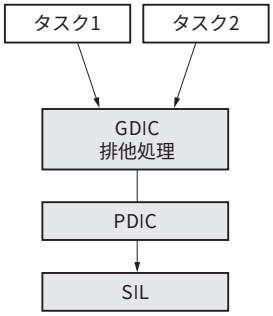

図3　GDICによる排他処理

GDICは，セマフォなどを使って排他処理を行う．

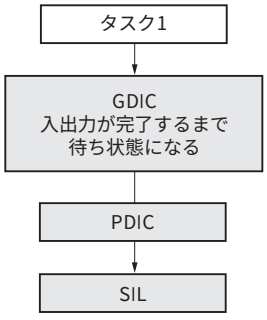

図4　GDICによる入出力完了待ち

GDICは要求処理が完了するまで待ち状態となり，割り込み発生時などにその待ち状態を解除する．