μITRON4.0仕様研究会
デバイスドライバ設計ガイドラインWG
中間報告書
－ ＤＩＣアーキテクチャの提案 －

1999年11月

μITRON4.0仕様研究会
デバイスドライバ設計ガイドラインWG

まえがき

このドキュメントは、μITRON4.0 仕様研究会 デバイスドライバ設計ガイドライン WG における検討成果の中間報告書である。デバイスドライバ設計ガイドライン WG は、μITRON 仕様カーネル上のデバイスを扱うソフトウェアモジュールの設計ガイドラインを策定することを目的に、1998 年 4 月より検討作業を行っている。

この中間報告は、検討開始から約 1 年半を経過した 1999 年 10 月時点での途中結果をまとめたものである。組込みシステムにおいては、デバイスを扱うソフトウェアモジュールの設計手法はノウハウに属する部分が多く、それをガイドラインとしてまとめるのは困難な作業である。そのため、約 1 年半の検討作業にもかかわらず、この中間報告にはまだ数多くの未検討課題が残っている。未検討課題の中には、ほとんど議論していない項目もあれば、議論したが結論が得られていない項目もある。またこの間、検討のための時間の多くを、割込み処理をポータブルに記述するための枠組みである「割込みシステム」の検討に費やした。その検討成果はすでに、1999 年 6 月に公開されたμITRON4.0 仕様に反映されている。

未検討課題が数多く残っているにもかかわらず、検討の途中結果を中間報告という形で公表することにしたのは、現在の検討の方向性に対して多くの技術者からご意見をいただき、今後の検討の参考にするとともに、このテーマに興味のある技術者に検討作業への参加を呼び掛けるためである。この中間報告に対してご意見をお持ちの方、ガイドライン策定に参加されたい方は、デバイスドライバ設計ガイドライン WG（device-wg@itron.gr.jp）まで連絡いただければ幸いである。

この中間報告は、デバイスドライバ設計ガイドライン WG のメンバが分担して執筆した。執筆を分担したメンバは次の通りである（あいうえお順）。

児玉 剛（アルパイン情報システム(株)）
宿口 雅弘（三菱電機マイコン機器ソフトウエア(株)）
高田 広章（豊橋技術科学大学）
舘 義宏（セイコーエプソン(株)）
綱川 寧孝（ソニー(株)）
横澤 彰（(株)東芝）

1 はじめに

1.1 ガイドライン策定の動機と目的

デバイスドライバ設計ガイドラインを策定する背景には、以下に挙げるようないくつかの動機がある。

・ ソフトウェア開発者からの動機

デバイスドライバは、ソフトウェアのサイズが小さいわりには、設計に高度なノウハウが必要で、デバッグも難しいと言われている。ソフトウェア開発者の立場からは、一度開発したデバイスドライバの再利用性と流通性を高め、開発工数を削減したい。そのため、デバイスドライバの再利用性と流通性を高めるためのガイドラインが求められる。また、ガイドラインに従ってデバイスドライバを開発することで、質の高いデバイスドライバが容易に開発できれば、メリットは極めて大きい。

・ デバイス提供者からの動機

デバイスを提供する半導体メーカ（または IP ベンダ）には、デバイスの付加価値を高める方策の一つとして、デバイスと一緒にデバイスドライバを提供することで、デバイスをできる限り容易に使えるものにしたいという動機がある。そのためには、提供するデバイスドライバが広い適用性を持つこと、具体的には、用いるプロセッサやカーネルに依存しないことが望ましい。すなわち、プロセッサやカーネルに依存せずにデバイスドライバを記述するガイドラインがあれば、メリットが大きい。

・ 技術者教育面からの動機

技術者の教育面からは、ガイドラインが、デバイスドライバ作成の教科書として使えるとメリットが大きい。また、ガイドラインによって用語の意味が標準化されれば、教育面のみならず、技術者間の意思疎通を容易にする観点からもメリットがある。

・ ITRON プロジェクトからの動機

μITRON 仕様カーネルの問題点として、デバイスドライバが揃っていない点がしばしば指摘される。その原因の一つとして、μITRON 仕様にはデバイスを扱うための API が含まれておらず、デバイスドライバの形態が標準化されていないために、デバイスドライバのソフトウェアが「μITRON 仕様カーネル用」という形で流通しにくいことが挙げられる。設計ガイドラインを提示することによってデバイスドライバの形態が標準化され、この問題が解決することが期待できる。

デバイスドライバをデバイスに付属させて提供することは、組込みシステム開発におけるハードウェアとソフトウェアの境界の流動化に伴って、重要性を増す傾向にある。システム開発の効率化のためには、あるコンポーネント（IP）がハードウェアで実装されているかソフトウェアで実装されているかに関わらず、ソフトウェアから見た場合のコンポーネントのインタフェースを標準化しておくべきである。すなわち、ハードウェアとして実装されたコンポーネントには、最低限のインタフェースソフトウェアが付属しているのが望ましいことになる。

以上で述べた動機より、このガイドライン策定の目的として、次の 4 つを挙げることができる。

(a) μITRON 仕様カーネル用のデバイスを扱うソフトウェアモジュールの標準的なモ

デルを提示し、その流通を促すこと。このガイドラインに従って構築されたデバイスを扱うソフトウェアモジュールを、Device Interface Component（DIC と略す）と呼ぶ。

(b)デバイスを扱う最低限のインタフェースソフトウェア（これを Primitive DIC と呼び、PDIC と略す）を、プロセッサやカーネルに依存せずに記述する枠組みを与えること。PDIC の適用性をさらに広げるためには、PDIC はカーネルを用いない場合にも適用できることが望まれる。

(c)ガイドラインにしたがって DIC を設計することで、熟練していない技術者にも安定した品質の DIC が開発できるようにすること。

(d)デバイスを扱うソフトウェアに関連する用語の意味を標準化すること。

このガイドラインで新たに DIC という用語を導入した理由は、技術者によってデバイスドライバという用語で想起するイメージの違いが大きく、誤解を防ぐ意味でも、また用語を標準化するためにも、新しい用語を導入するのが望ましいと考えたためである。具体的には、デバイスドライバという用語はデバイスを抽象化するための OS 内のモジュール（図 1 左）という意味に使われる場合が多く、デバイスを扱う API を持たないμITRON 仕様カーネルのモデルでは何を指すのかが明らかでない。

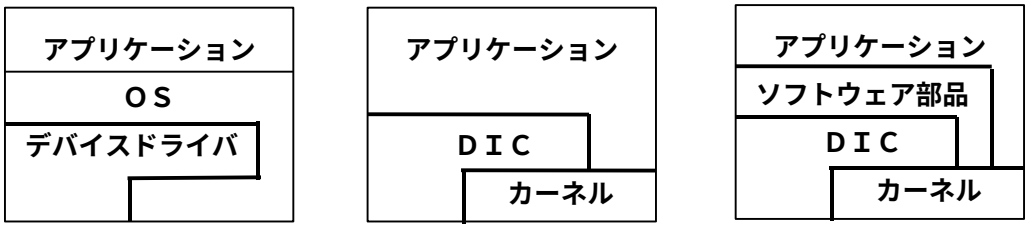


図 1．デバイスドライバと DIC

1.2 組込みシステムにおけるデバイスの多様性

このガイドラインで対象とする「デバイス」とは、プロセッサに接続され、周辺装置を接続するためのインタフェースおよびプロセッサの機能を拡張するためのユニットのことを言う。代表的なデバイスとしては、SIO（シリアル I/O）、PIO（パラレル I/O）、LANC（LAN コントローラ）、SCSI コントローラ、USB・IEEE1394・IrDA などの各種の通信インタフェース、グラフィックコントローラ、各種のセンサ（またはそのインタフェース）、フラッシュメモリなどが挙げられる。また、組込みシステムにおいては、システム独自のデバイスが用いられることも多い。

組込みシステムにおいては、このような極めて多様なデバイスをサポートしなければならず、デバイスドライバとアプリケーションの境界が必ずしも明確ではない。応用分野によっては、デバイスの操作がアプリケーションそのものと言える場合もある。多様なデバイスに効率よく対応できるデバイスドライバの API や作成方法を「標準化」することは容易ではない。そのためこのドキュメントでは、標準化された「仕様」ではなく、標準的な「ガイドライン」を提示する。

1.3 ガイドラインの内容と構成

以上より、このガイドラインにおいては、次の 3 つのモデルならびにガイドラインを提示する。

(1) μITRON 仕様カーネル上でデバイスを扱うソフトウェアモジュール（DIC）を構築する標準的なモデル

(2) プロセッサやカーネルによらないポータブルな PDIC の作成ガイドライン

(3) μITRON 仕様カーネル上に GDIC（General DIC; PDIC 以外の DIC を GDIC と呼ぶ）を構築する場合の設計ガイドライン

この中間報告の段階では、(3)のガイドラインの提示については、多くの部分が今後の課題として残っている。

本ガイドラインは、次のように構成されている。２章では、DIC の基本アーキテクチャについて解説し、DIC の標準的な構築モデルを提示する。３章では、PDIC をプロセッサやカーネルに依存せずに記述する際に前提とする割込みシステムを提示し、その実現方法について述べる。４章では、その割込みシステムを前提に、プロセッサやカーネルによらないポータブルな PDIC の作成ガイドラインを提示する。５章では、μITRON 仕様カーネル上に GDIC を構築する場合の設計ガイドラインについて述べる。６章では、このガイドラインに従った DIC の設計例を示す。

2 DICの基本アーキテクチャ

2.1 DICとは？

ソフトウェアからデバイスのアクセスを容易にするためのインタフェースソフトウェアで、このガイドラインに従って構築されたものを、DIC（Device Interface Component）と呼ぶ。

リアルタイムカーネルを用いる場合には、DIC はカーネル上に構築され、アプリケーションから直接呼び出されるのが基本であり（図 1 中）、デバイスの違いを隠蔽することが一義的な目的ではない。すなわち、OS ないしは上位のソフトウェア部品に対してデバイスの抽象化を提供することを一義的な目的とはしていない。デバイスの違いを隠蔽しないことは、逆に言うと、デバイスの特徴を引き出せることを意味する。

もちろん、アプリケーションからは上位のソフトウェア部品（例えば、ファイルシステムやプロトコルスタック）のみを呼び出す場合には、DIC はアプリケーションから直接呼び出されることはない（図1右）。

2.2 DICの階層モデル

一般に、デバイスを扱うソフトウェアモジュールはカーネルの機能を必要とするため、カーネルに依存せずに記述することはできない。一方で、デバイスを扱う最低限のインタフェースソフトウェアは、プロセッサやカーネルに依存せずに記述できることが望まれる。そこで、デバイスを扱うソフトウェアモジュールを、カーネルに依存せずに記述でき、デバイスに対する最低限のインタフェースのみを提供するモジュールと、それに対して機能を追加するモジュールに分割して構築する方針をとる。このガイドラインでは、前者を Primitive DIC（PDIC）と呼び、後者を General DIC（GDIC）と呼ぶ。

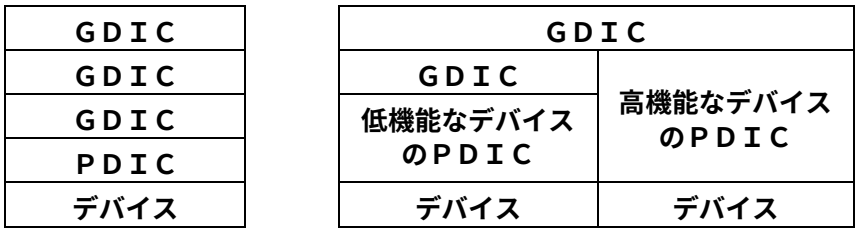


図 2．DIC の階層構造

PDIC は、デバイスのハードウェアが持つ機能をそのまま提供し、ソフトウェアによる機能追加をしないことを基本とする。具体的には、PDIC 以下の階層では、デバイスに関する以下のような事項のみを隠蔽するのを基本とする。

- デバイスのレジスタ構成やビット割付け
- デバイスへのコマンドのビットパターン
- プロセッサとデバイスの接続方法

PDIC は、デバイスを提供する半導体メーカ（または IP ベンダ）が提供するのが望ましいという考え方から、PDIC を通してデバイスのすべての機能を扱えるのが理想的である。また、PDIC をカーネルを用いない場合にも適用できるものとするために、PDIC ではカーネルの持つタスク管理機能を用いないこととする。したがって、PDIC の中で待ち状態になることはできない。それに対して、カーネルの機能であっても、カーネルがない場合も容易に実現できる割込みの禁止/許可などは用いる（用いずに記述する

ことは困難であろう）。PDIC を、プロセッサやカーネルに依存せずに記述するためのガイドラインについては、４章で述べる。

それに対して GDIC は、PDIC ないしは他の GDIC の上に構築され、下の階層の DIC に対して機能を追加するものである。すなわち DIC は、PDIC を最下位層とし、その上に GDIC を重ねた階層構造を取る（図 2 左）。GDIC がデバイスをアクセスする場合には PDIC を経由することとし、GDIC が直接デバイスをアクセスしてはならない。

GDIC は、プロセッサには依存しないのに対して、内部で待ち状態に入る場合があるなど、一般にはカーネルの機能に依存する。そのため、異なるリアルタイムカーネル上で利用する場合には、改造が必要になる。

PDIC は、デバイスのハードウェアが持つ機能をそのまま提供することを基本としているため、低機能なデバイスの PDIC と高機能なデバイスの PDIC では、提供するサービスのレベルが異なる。そのため、高機能なデバイスの PDIC の上に構築された GDIC を低機能なデバイスに適用する場合には、デバイスの機能差を埋めるための GDIC が必要になる場合が考えられる（図 2 右）。

一例として、i8251 のような単純な SIO デバイス用の DIC の場合、PDIC では「文字を受信していれば、その文字を取り出す」「送信バッファが空いていれば、文字を送信バッファに書き込む」「文字を受信したことを通知する」「送信バッファに空きができたことを通知する」といった基本的な機能のみを提供し、パケット単位のバッファリングや文字の送受信を待つ機能などは、その上に構築する GDIC に持たせる。ハードウェアでバッファリング機能を持つ高機能な SIO デバイスの場合には、PDIC レベルでバッファリング機能を提供することになる。さらにそれらの上に、HDLC などの通信プロトコルをサポートする DIC を載せることもできる。

ただし、以上で述べた DIC の階層構造は、あくまでもデバイスを扱うソフトウェアモジュールを設計する上でのモデルであって、実装においては複数の階層を併合して最適化してもよい。

2.3 DIC の API の枠組み

ITRON 仕様共通規定では、API を「サービスコール」「コールバック」「静的 API」「パラメータとリターンパラメータ」「データ型」「定数」「マクロ」「ヘッダファイル」の 8 つの要素で構成されるものとしており、DIC の API もこれに準拠する。8 つの構成要素の中で、ここでは、DIC の機能を呼び出すためのサービスコールと、DIC から事象を通知するためのコールバックについて扱う。DIC の階層モデルを考えあわせると、DIC と上下の階層のソフトウェア（またはハードウェア）との関係は図 3 のようになる。PDIC においては、デバイスに対するアクセスが下の階層へのサービスコールに、割込みが下の階層からのコールバックに対応する。

処理の流れに着目して、DIC のサービスを以下のパターンに分類する。DIC のドキュメントでは、どのサービスがどの分類に属するかを明記しなければならない。

(1)即時完了型サービス

サービスを要求すると、即時に（おおよそ、カーネルのタスク切替え時間の数倍以下の時間で）完了するようなサービス。ポーリングによる待ちが含まれる可能性はあるが、他のタスクへ切り替えた方が良いような長い待ちは含まない。

即時完了型のサービスの API は、即時完了型の処理要求を行うサービスコールで構成される。

(2)同期型サービス

サービスの完了までに時間がかかるか、他のサービス要求と同期する必要があるために、DIC 内でカーネルのタスク管理機能を用いて待ち合わせる必要があるよう

なサービス。PDIC はカーネルに依存しないことから、同期型サービスを提供できない。

同期型のサービスの API は、同期型の処理要求を行うサービスコールと、待ちあわせている状態のサービスの実行をキャンセルするためのサービスコールから構成される。処理要求のサービスコールは、必要ならタイムアウト指定ができるものとする。キャンセルのサービスコールのサポートは、オプションである。

(3)非同期型（ノンブロッキング型）サービス

サービスの完了までに時間がかかるか、他のサービス要求と同期する必要があるが、処理要求を行うサービスコールからは即時にリターンするようなサービス。

非同期型のサービスの API は、非同期型の処理要求を行うサービスコール、非同期型のサービスが完了した（または、キャンセルされた）ことを通知するコールバック、継続中のサービスの実行をキャンセルするためのサービスコールから構成される。非同期型の処理要求を行うサービスコールからは即時にリターンするが、要求した処理は継続している（ただし、デバイスの状態などによっては、要求した処理が即時に終わる場合もある）。処理が完了する（または処理がキャンセルされる）と、非同期型の処理要求の完了通知を行うコールバックにより、上位層のソフトウェアに通知される。キャンセルのサービスコールのサポートは、オプションである。なお、要求処理キャンセル時に完了通知を行うコールバックを呼び出すかどうかは、現時点では未検討である。

(4)事象通知

DIC から上位層のソフトウェアに対する非同期な事象の通知。事象通知の API は、事象通知を行うコールバックによって構成される。

以上を逆にみると、DIC の API の枠組みを次のように整理することもできる。

(a) DIC のサービスコール

(a-1) 処理要求

・即時完了型の処理要求

・同期型の処理要求

・非同期型の処理要求

(a-2) 処理要求のキャンセル

・同期型の処理要求のキャンセル

・非同期型の処理要求のキャンセル

(b) DIC からのコールバック

(b-1) 処理要求の完了通知

・非同期型の処理要求の完了通知

(b-2) 事象通知

DIC 内のサービスコールを実現するためのルーチンをデバイスサービスルーチン（DSR と略す）、コールバックを実現するためのルーチンをコールバックルーチン（CBR と略す）と呼ぶ。PDIC においては、割込みサービスルーチン（ISR）がコールバックルーチンに対応する（図 3）。

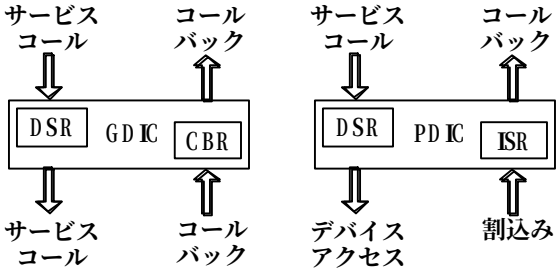


図 3．DIC と上下の階層とのインタフェース

ここで、DIC の DSR および CBR がリエントラントかどうか（同時に複数のタスクが呼び出してよいかどうか）は、DIC 毎にそのドキュメントに明記するものとし、このガイドラインでは規定しない。

2.4 DIC の API

デバイスを扱うための API として、デバイスをファイルとして抽象化する UNIX 流の方法が広く使われているが、組込みシステムにおいてサポートすべきデバイスの多様性を考えると、必ずしも最適とはいえない。特に、センサーやアクチュエータなどの制御系のデバイスの場合、デバイスに対して送受信するデータが存在しないために、UNIX 流の方法で抽象化するとすべての操作を ioctl で実現したということになりかねない。

また DIC がアプリケーションから直接呼び出される場合には、その API を少数のサービスコールに限定する必然性はない（それに対して、OS から呼び出されるデバイスドライバは、デバイスの種類によらずにサービスコールの種類を標準化する必要がある）。このことからこのガイドラインでは、DIC の API の標準化は行わない。ただし、今後の検討の中で、デバイスの種類を限定して、DIC の標準的な API を提示する可能性はある。例えば、ストレージデバイスの DIC の標準的な API は、標準化できる可能性がある。

また、初期化処理と終了時処理、オープンとクローズなどについては、デバイスの種類を限定せずに、API のガイドラインを示す予定である。

2.5 未検討課題

DIC の実行コンテキストについては、検討が十分ではない。μITRON4.0 仕様では、タスクコンテキストから呼び出せるサービスコールと、非タスクコンテキストから呼び出せるサービスコールを分離したことから、DIC の各サービスコールやコールバックがどのコンテキストで実行されるかを明確にする必要がある。

また、1 つの DIC が複数個の物理/論理デバイスをサポートしている場合に、扱うデバイスを指定する方法については今後の課題である。オープンやクローズの手順は、これに関連して検討する予定である。

3 割込みシステム

3.1 想定する仮想アーキテクチャ

PDIC を、プロセッサやカーネルに依存せずに記述できるようにするためには、それらを何かしら仮想化する必要がある。仮想化は実装時のオーバーヘッドと裏腹であるが、ここに提示するモデルは、多くの実装に大きなオーバーヘッドなしに適用可能であることを目指して作成したものである。もちろん、モデルの提示であって、現実の実装時のチューニングを禁止するものではない。

PDIC がデバイスのみに依存して記述できるようにするために PDIC の下位にシステムインタフェースレイヤーを設ける。この中には、「割込みシステム」、「アクセスインタフェース」、「システム提供関数群」が含まれる。この様子を図 3-1 に示す。

割込みシステムとは、デバイスからの割込み要求に対して、それに対応する割込みサービスルーチン (Interrupt Service Routine, 以下 ISR と略す) が起動されるまでに必要な機能をブラックボックス的に捉えた概念である。ISR は PDIC を構成する要素であり、この章では、そのポータビリティの前提となる割込みシステムについて説明する。アクセスインタフェースとシステム提供関数群については、4 章で説明する。

仮に、システムインタフェースレイヤーの機能がプロセッサ, 割込みコントローラ (Interrupt Request Controller, 以下 IRC と略す) およびカーネルによって実装されていれば、PDIC をそのまま(コンパイルのみで)動かすことが可能である。不足する部分があれば、PDIC の移植者がそのギャップを埋める必要がある。

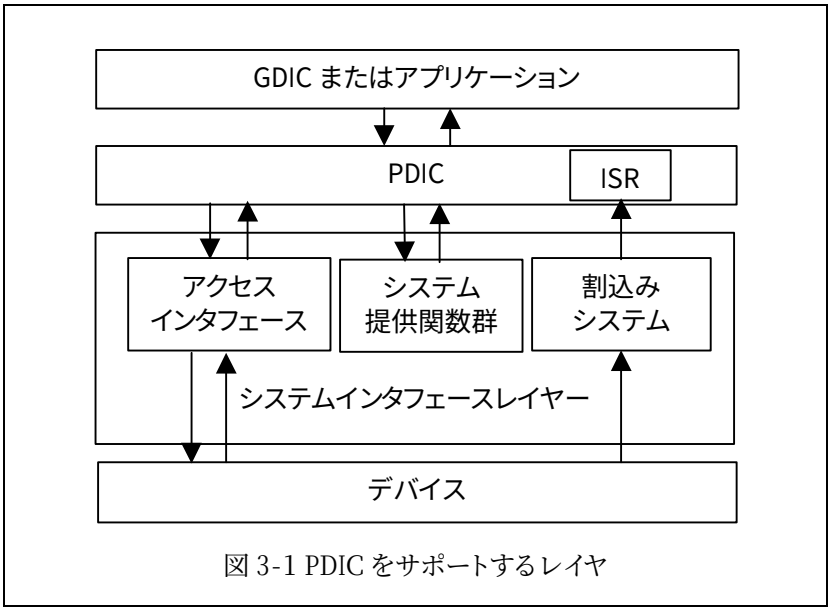


図 3-1 PDIC をサポートするレイヤ

3.2 割込みサービスルーチン(ISR)

デバイスからの割込み要求に対して、プロセッサ上で実行される処理(カーネル内の処理を含む)には、

(1) プロセッサの割込みアーキテクチャに依存する処理

(2) IRC や割込み信号ラインの接続等、プロセッサ以外のハードウェア(デバイス自身を除く)に依存する処理

(3) デバイス自身に関わる処理

が含まれると考える。この処理モデルを図 3-2 に示す。外部割込みに対して、まずプ

ロセッサが割込み処理を呼び出す処理(1)があり、次に、必要なら IRC の状態を調べて、割込みを要求したデバイスに対応する割込み処理を呼び出す処理(2)を経て、デバイス自身に関わる処理(3)が起動されるというフローである。プロセッサのアーキテクチャによっては、(1)の部分はハードウェアで処理され、ソフトウェアの処理が不要な場合もある。前節に述べた仮想アーキテクチャに従えば、(1)と(2)の部分は割込みシステムで処理される部分であり、(3)に対応するのが ISR となる。この部分を他と切り離して作成することをガイドラインとする。このようにして作られた ISR は、PDIC に求められるポータビリティを持つことになる。この章では、割込みシステムについて説明すると共に、これが完全にサポートされていないプラットフォームにこのモデルを当てはめる際に必要となる(2)に対応する部分についてそのテンプレートを示す。なお、ISR 自体の設計ガイドラインについては、4 章で説明する。

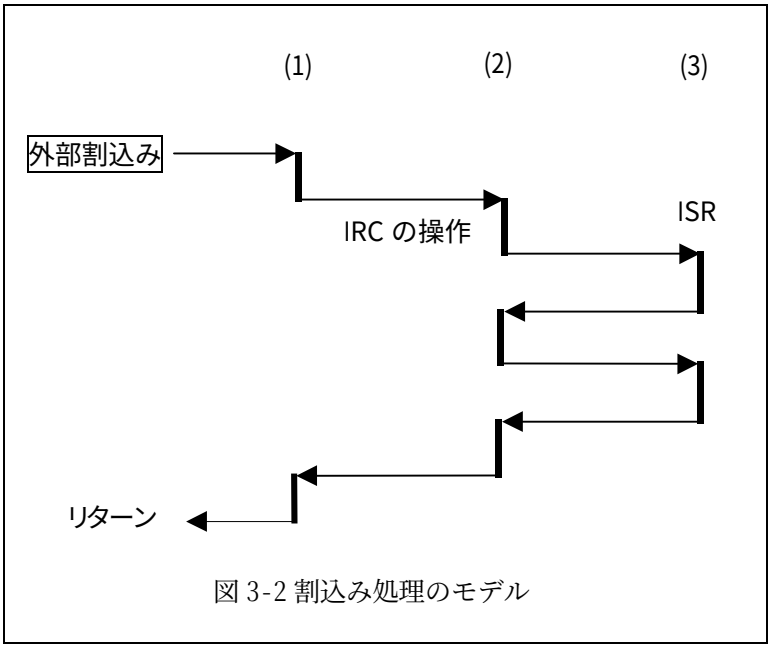


図 3-2 割込み処理のモデル

3.3 μITRON4.0 仕様との関係

μITRON4.0 仕様の割込み処理モデルは、このガイドラインと整合している。(μITRON4.0 仕様書の 3.3 節を参照)

μITRON4.0 仕様では、カーネルが 図 3-2 の (1) のみをサポートする場合と、(1) と (2)をサポートする場合の両方に必要な API を規定している。μITRON4.0 仕様では、(2)の部分を割込みハンドラ、(3) の部分を割込みサービスルーチンと呼んで区別している。(2) と (3) の区別をしない場合はまとめて割込みハンドラの扱いとなる。

3.4 割込みシステムの外部インターフェース

割込みシステムの外部インタフェースモデルを図 3-3 に示す。

要するに、割込み要求を入力すると、対応する ISR が起動されるというモデルである。ここではデバイスからの割込み出力を割込み要求ラインと呼び、割込みシステムへの入力を割込み入力ラインと呼ぶ。

3.4.1 割込み要求の発生

デバイスは、割込みシステムに対して割込み要求ラインで接続されており、割込みを要求する。一つのデバイスが複数の割込み要求ラインを持つことも許される。

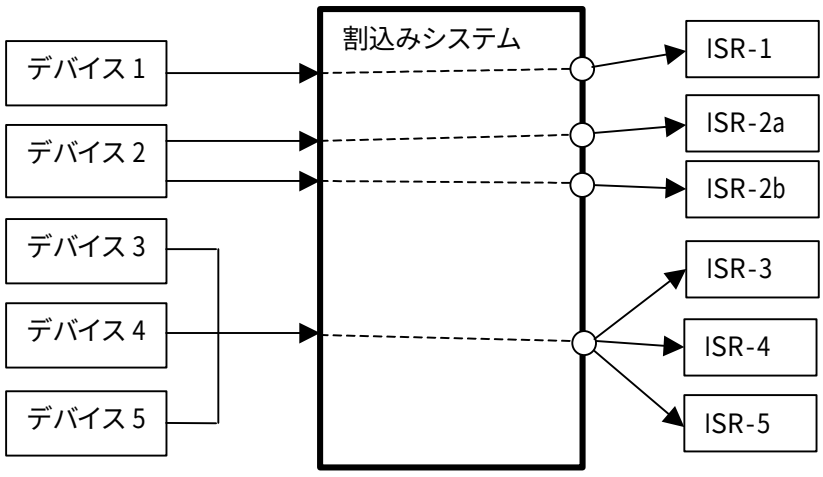


図 3-3 割込みシステムの外部インタフェースモデル

3.4.2 レベル入力とエッジトリガ入力

割込みシステムに対するデバイスからの割込み要求の入力には、レベルとエッジトリガの 2 種類がある。割込み要因が解除されるまで、デバイスがレベルで割込みを要求し続ける方式が確実であるが、ハードウェアの制約や、周期タイマ割込みを分周で作る場合など、レベルが使えない場合には、エッジトリガも使用される。エッジトリガの場合は、割込みシステムがその要求を一時記憶する。

3.4.3 割込み要求のワイヤード OR の扱い

割込み要求ラインがワイヤード OR される場合に対応するため、割込みシステムは、一つの割込み入力ラインに対して、複数の ISR を登録する機能を持つ。そのような割込み入力ラインに入力があると、割込みシステムは登録されている ISR をすべて順に呼び出す。この場合、ISR は、デバイスの割込み要求ラインに対応して一つずつ用意されると考える。

ただし、エッジトリガの割込み要求をいくつかワイヤード OR して割込みシステムに入力すると、ワイヤード OR されたデバイス同士で複数の割込み要求が縮退する可能性がある。プログラミングの工夫でこのような状態を検出して正しく処理する事は可能であるが、そこまで考慮すると処理が複雑になり、オーバーヘッドが増えたり処理時間が予測出来なくするなどの弊害があるため、このガイドラインでは推奨できない方法とする。

このモデルでは、割込みを要求していないデバイスに対応する ISR が起動される可能性があるため、ISR では自身の処理すべき割込みが発生していることを確認し、発生していなければ何もせずに戻ることが必要となる。

このようなモデルを採用した理由は以下の通りである。ISR のポータビリティを確保するために、基本方針として、ISR はデバイスのみを参照、操作することが許されるものとしている。逆に、ISR を呼び出す割込みシステムの側では、デバイスの状態を参照することはできないとしている。このため、入力がワイヤード OR されている場合、割込みシステムでは割込みを要求しているデバイスを特定することができない。このため、入力に対応する ISR を順に呼ばざるをえない。

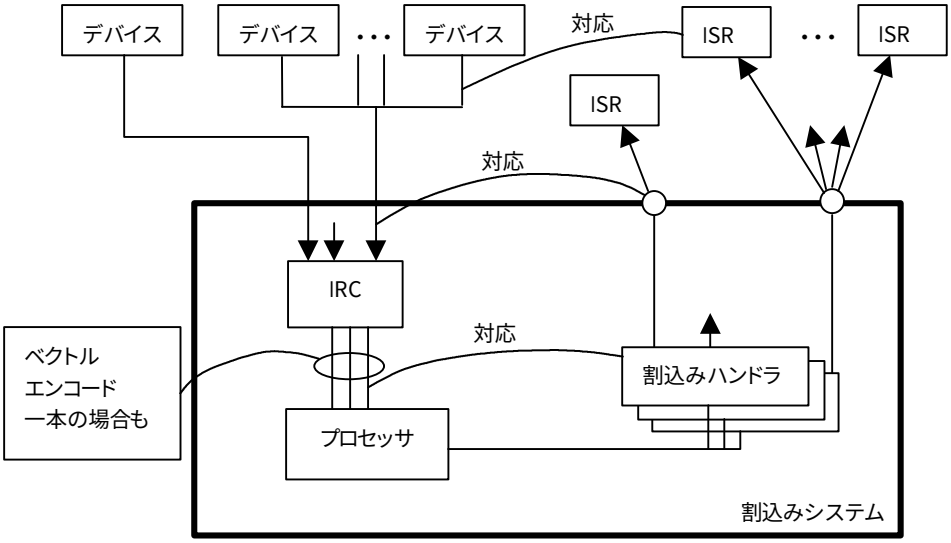


図 3-4 割込みシステムの内部構造

3.5 割込みシステムの内部構造

次に、割込みシステムの典型的な内部構造を説明する。図 3-4 に内部構造のモデルを示す。このモデルでは、デバイスからの割込み要求は、IRC に入力される。IRC は割込み要求をプロセッサに伝える。プロセッサは割込み要求を受けるとそれに対応する割込みハンドラを起動する。割込みハンドラは必要なら IRC の状態を調べ、割込み入力に対応する ISR を起動するなどの処理を行う。

μITRON4.0 仕様との対応について補足すると、割込みハンドラは、IRC からプロセッサへの割込み要求の種類に対応して DEF_INH という静的 API によって登録される。割込みベクトルを持つプロセッサでは、ベクトル毎に異なる割り込みハンドラが存在すると考える。割込みベクトルを持たないプロセッサでは、割込みハンドラが一つだけ存在すると考える。

IRC の入力は割込み番号で識別される。前述した割込みシステムへの入力のワイヤード OR は、一つの割込み番号に複数のデバイスの割込み要求ラインが対応するというモデルとなる。ISR はデバイスからの割込み要求ラインに対応して存在する。μITRON4.0 仕様では、割込み番号と ISR は ATT_ISR という静的 API によって対応づけられる。

PDIC の ISR を移植して利用するために必要な作業は、カーネルを使うかどうか、また、そのカーネルがここに述べたモデルをどの程度サポートしているかどうかに依存して増減する。

カーネルが ISR までサポートしていれば、ATT_ISR 相当の機能で ISR を登録するだけでよい。割込みハンドラまでのサポートの場合は、後述する「割込みサービスルーチンを起動するための処理」の部分を割込みハンドラとして作成する必要がある。カーネルを使用しない場合は、これに加えてやはり後述する「割込みハンドラの出入口処理」の部分も作成する必要がある。

これら仮想化に対応するソフトウェアを作成するためには、ハードウェアの実装方法のうち、

- **デバイスの割込み要求ラインと IRC の割込み入力ラインとの接続方法**
- **IRC の入力とプロセッサへの割込み要求の対応**

などに関する情報が必要となる。

3.6 割込みコントローラ（IRC）のモデル

割込みハンドラの処理のテンプレートを議論する際には IRC のモデルが必要である。ここでは、このガイドラインで想定するモデルについて説明する。いくつかの典型的な IRC を調査し、このガイドラインでは多くの IRC に適用可能なモデルとして図 3-5 を採用した。

この IRC モデルの機能は次のようにまとめられる。

(1)**優先レベル制御** 複数の割込み要求がある場合に、優先度の高いものを選択して割込み要求をプロセッサに伝える機能

(2)**エッジ/レベル変換** エッジトリガ入力をレベルに変換して記憶する機能、およびそれをクリアする機能

(3)**割込みマスク** 割込み入力別にマスクする機能

(4)**サービス中割込みの管理** 優先レベル制御と関連するが、プロセッサが処理中の割込みを管理し、それよりも優先度の高い割込みが発生したら、その割込み要求をプロセッサに伝える機能。この機能がある場合、割込み処理プログラムは、割込み処理の最後で EOI 命令などでサービス中フラグをクリアする必要がある。

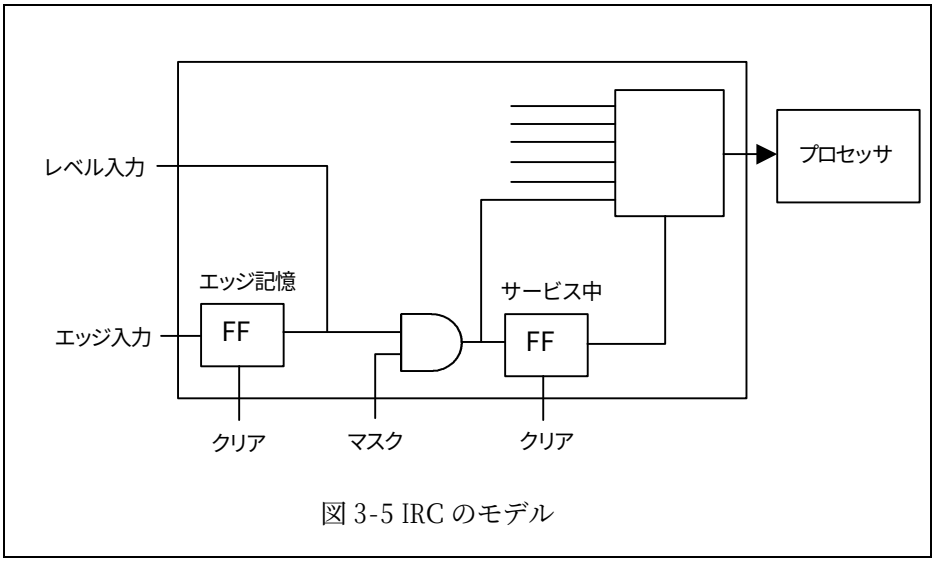


図 3-5 IRC のモデル

3.7 多重割込みの扱い

割込みシステムの移植性を考慮して、多重割込みについては次のように定めている。

割込み優先レベルの数は、1 以上いくつでもよい。ISR の起動時に他の割込みが許可されているかどうかは実装に依存する。これはμITRON4.0 仕様において、ISR の起動直後が、CPU ロック解除状態であると規定されているのと整合しており、ISR を記述する際には、多重割込みを前提とする必要がある（詳細は 4 章参照）。

3.8 ハードウェアによる割込みサポートのバリエーション

デバイスからの割込み要求がプロセッサに伝えられる方式は、プロセッサの割込みアーキテクチャ、使用する IRC、信号ラインの接続方式などに依存して多様である。プロセッサの割込みアーキテクチャでみれば、外部割込みと CPU 例外が一つのアドレスに分岐するだけという単純なものから、割込みベクトルを認識し、動作モードやスタックの切替え処理を経て対応する割込み処理プログラムを起動するという高機能なものまで存在する。このガイドラインで想定している割込みサポート機能のバリエーションは以下の通りである。

- プロセッサの割込みアーキテクチャのバリエーション
- 割込みベクトルのサポート　有／無
- モード切り替えのサポート　有／無
- スタック切り替え　有／無

- IRC のバリエーション
- 割込み優先レベルの有無とレベルの数
- IRC のカスケード接続　有／無
- 割込みベクトル供給　有／無
- 割込みサービス中フラグ　有／無
- プロセッサと IRC の機能の境界

- 接続のバリエーション
- デバイスからの割込み要求ラインのワイヤード OR　有／無
- デバイスからの割込みベクトル供給　有／無

3.9 割込みハンドラの出入口処理

割込みシステムのモデルでは、割込みハンドラを起動する前後の処理は、IRC を見る事なく、プロセッサの機能だけで実現できる範囲と考える。この部分をソフトウェアで実装する場合、μITRON4.0 仕様の用語では、割込みハンドラの出入口処理と呼ぶ。

このように定義しているので、割込みハンドラの出入口処理の内容は、プロセッサのアーキテクチャ、および、カーネルが存在する場合には、その実装方式に強く依存する。

ここでは、特定のプロセッサアーキテクチャを想定せず、一般的にこの部分に含まれる可能性のある処理を羅列する。これらの処理順序も実装に依存する。

＊入口処理

- 最低限のレジスタのセーブ
- CPU 例外と割込みの判別
- 割込み用スタックへの切替え
- 割込みの種類の判別
- 対応する割込みハンドラのアドレス取得
- レジスタセーブ
- 割込みハンドラが C 言語で書かれる場合のスタックフレーム形成
- 割込みハンドラの呼び出し

＊出口処理

- **スタックフレーム解放**
- **レジスタ復帰**
- **スタック切替え**
- **割込みからの復帰またはタスクディスパッチ**

3.10 割込みハンドラの処理のテンプレート

割込みサービスルーチン（ISR）を起動するための処理は、割込みハンドラとして実装される。割込みハンドラの記述方式は、μITRON4.0 仕様でも標準化されていないため、ポータブルな形で提供することはできない。しかし、テンプレートとして以下が参考となろう。このテンプレートは、前述したハードウェア構成のバリエーションのうち、ソフトウェアの処理が多い場合に対応するものである。実装に応じて不必要な処理を省くことが可能である。

```
INT_Handler()
{
        IRC の割込み要因の検査;
        if（エッジトリガ）
                エッジトリガのクリア;
        CPU ロック解除状態にする;

        for（アタッチされたすべての ISR に対して）{
                アタッチされた ISR の実行;
                /* デバイス内の割込み要因のクリアは ISR 内で行う */
        }
        CPU ロック解除状態を元に戻す; /* 割込み禁止状態になる */
        IRC のサービス中フラグをクリア;
                (EOI の発行など)
        割込みハンドラから戻れる状態にする;
        return;
}
```

割込みハンドラは、プロセッサの割込みベクトルに対応して登録される。そこでは、IRC の状態を調べ、どの IRC 入力の割込みを処理すべきか判定する。エッジトリガの場合は、エッジトリガのクリアをここで行う。

前述したように、ISR は CPU ロック解除状態で呼び出されるのがμITRON4.0 仕様およびこのガイドラインの規定なので、プロセッサの割込みアーキテクチャとカーネルの実装に応じてその状態にしたのち、該当の割込み入力ラインに対して生成／追加されている ISR を順に呼び出す。

すべての ISR が終了したら、IRC のサービス中フラグをクリアする。最後に、CPU ロック解除状態を元に戻し割込みハンドラから戻れる状態にし、割込みハンドラから戻る。

3.11 未検討課題

3.11.1 IRC 対応 DIC

DIC の定義から逸脱する面があるが、IRC を一つのデバイスとしてとらえ、それに対

する DIC を提供するというアプローチの可能性がある。その場合、DIC として提供すべき機能は次の通りである。

- 割込みサービスルーチンを起動する処理(割込みハンドラ)
- 割込みマスク／マスク解除
- 初期化

  エッジ／レベル設定
  ベクタ設定
  優先度設定
  etc.

前述した通り、割込みハンドラの記述方式はプロセッサおよびカーネルに依存するため、移植可能な方法で提供することはできない。しかし、3.9 に示したテンプレート中に、

- IRC の割込み要因の検査
- エッジトリガのクリア
- IRC のサービス中フラグのクリア（EOI の発行など)

と表記した部分は、IRC が決まれば記述できる。特定のプロセッサとカーネル用のサンプルコードが提供されるだけでも、他のプラットフォームへの移植の際に有用であろう。

3.11.2 IRC のカスケード接続

割込み入力の数が多くて一つの IRC では不足する場合、IRC をカスケード接続して使用することがある。IRC 用の DIC を、カスケード接続時にも使えるようにするためのガイドラインが作れれば有用と思われるが、IRC の種類や接続の方法に強く依存するため、未検討である。

4 PDIC 実装ガイドライン

4.1 はじめに

この章では PDIC を実装するためのガイドラインを示す。

DIC のうち少なくともデバイスを直接操作する部分は、そのデバイスを設計した側で提供したほうがよい。デバイスの仕様書を読みながら試行錯誤せずにすむような DIC をあらかじめ提供しておけば、デバイスを利用する側の実装作業を大きく削減できる。そこでデバイスを直接操作する部分を PDIC として他の DIC と区別し、デバイスを設計した側で容易に実装できるものとする。

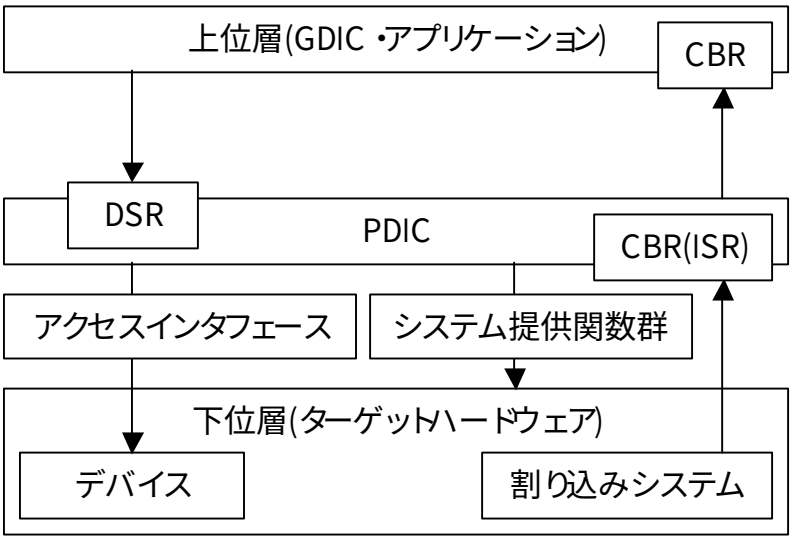
なお、この章では以下の言葉を次のような意味で用いる。

ターゲットハードウェア	PDIC が最終的に動作することになるハードウェア。
実装者	デバイスの仕様書に基づいて PDIC を作成するひと。
移植者	ターゲットハードウェアに PDIC を移植するひと。
利用者	ターゲットハードウェアに移植された PDIC を利用するひと。

4.2 PDIC の概要

4.2.1 PDIC のモデル

PDIC を取り巻く環境は次の図のようなモデルで表現される。



PDIC より上位には別の DIC(GDIC)またはアプリケーションが存在する。下位にはシステムインタフェースレイヤーやターゲットハードウェアが存在する。PDIC 自体は後述する DSR や CBR(ISR)などから構成されている。

4.2.2 PDIC と上位層のインタフェース

PDIC と上位層は PDIC で宣言するインタフェースを介してやりとりする。PDIC で宣言するインタフェースには次のようなものが含まれる。

（1） PDIC 自体を初期化するサービスコール
PDIC の実装によっては PDIC 自体を初期化するサービスコールが必要になる。

（2） デバイスの機能に対応したサービスコール
上位層はサービスコールを呼び出すことにより、デバイスの機能に対応する即時完了型か非同期型の処理を PDIC に要求する。

（3） 上位層へのコールバック
PDIC は CBR(ISR)などから上位層へコールバックすることにより、事象の発生を上位層に通知する。

これら PDIC で宣言するインタフェースにより、デバイスの操作方法を知らなくとも GDIC などの上位層を記述できる。

4.2.3 PDIC と下位層のインタフェース

PDIC と下位層としてシステムインタフェースレイヤーを呼び出す。システムインタフェースレイヤーには次のようなものが含まれる。

（1） アクセスインタフェース
アクセスインタフェースはプロセッサとデバイスの接続方法を抽象化する。PDIC はアクセスインタフェースを用いてデバイスレジスタや共有メモリにアクセスする。

（2） システム提供関数群
システム提供関数群はプロセッサの能力を抽象化する。PDIC はシステム提供関数を用いて、割り込みの禁止・割り込みの許可・時間待ちなどのプロセッサの能力に依存する機能を利用する。

（3） 割り込みシステム
割り込みシステムは割り込み処理を抽象化する。割り込みシステムは PDIC が用意した ISR をコールバックすることで、割り込みの発生を PDIC に通知する。

これらシステムインタフェースレイヤーを利用することにより、実装者は PDIC をターゲットハードウェアに依存せずに記述できる。

また、システムインタフェースレイヤーを利用して実装された PDIC であれば、PDIC 自体を変更しなくとも、移植者がターゲットハードウェアにあうシステムインタフェースレイヤーを作成することで PDIC を移植できる。つまり、仮にデバイスの知識がなくとも、ターゲットハードウェアさえ理解していれば PDIC を移植できることになる。

4.3 PDIC の実装

4.3.1 プログラミング言語

PDIC はプログラミング言語として ANSI C を用いる。ANSI C で記述すれば、最も多くのターゲットハードウェアで PDIC を利用できる。

4.3.2 ファイル構成

デバイス *device* の PDIC は次のようなファイルから構成される。

（1） 実装者が用意するファイル

以下のファイルは実装者が PDIC ごとに用意する。

(a) *device*.h(必須)

この PDIC が提供するマクロや関数のプロトタイプなど、上位層が必要とする情報を含むヘッダファイル。PDIC 内部および上位層にインクルードされる。

(b) *device*_sil.h(必須)

この PDIC が依存するシステムインタフェースレイヤーのためのヘッダファイル。この PDIC 内部にインクルードされる。必要に応じて移植者が書き換える。

(c) *device*_cfg.h

この PDIC が依存する定数やデータ型の情報を含むヘッダファイル。この PDIC 内部にインクルードされる。必要に応じて移植者が書き換える。*device*.cfg この PDIC をカーネルに組み込むためのコンフィギュレーションファイル。

(d) *device*.c, *device*_*.c

この PDIC を構成するソースファイル。

(e) *device*_*.h

あるソースファイル *device*_*.c で定義された名前のうち、PDIC 内部の他のソースファイルから必要となる名前を宣言するヘッダファイル。つまりスタティックでない関数やグローバル変数の宣言を含む。

（2） 別途に用意されるファイル

以下のファイルはターゲットハードウェアに対してただ一つ存在する。もっぱら移植者の責任で用意することになる。実装者が用意する必要はない。

（a） sil.h

システム提供関数群のプロトタイプやマクロの情報を含むヘッダファイル。PDIC から独立して存在する。

（b） pdic.h

標準 C ライブラリのうち PDIC から利用できる関数のプロトタイプやマクロの情報を含むヘッダファイル。PDIC から独立して存在する。(将来的に、デバイスドライバ設計ガイドラインの一部として提供することを検討中である。)

（c） itron.h

ITRON 仕様共通定義で規定されるデータ型、定数、マクロの定義などを含むヘッダファイル。PDIC から独立して存在する。特にデータ型を参照するため必要になる。

実装者は PDIC がこれらのファイル以外に依存しないように実装する。

移植者は、実装者が用意した *device*_sil.h などを書き換えることで PDIC をターゲットハードウェアに移植する。また移植者は *device*_sil.h や *device*_cfg.h や *device*.cfg をターゲットハードウェアで用意した別のヘッダファイルに依存するように変更して

よい。

4.3.3 PDIC 設計の原則

実装者は PDIC と上位層のインタフェースを次のように設計する。

（1） デバイスが定義したビットパターンなどに依存しないこと。
PDIC と上位層のインタフェースが、デバイスのレジスタ構成やビット割りつけ、デバイスへのコマンドのビットパターンに直接影響されてはならない。あくまでデバイスの機能に基づくインタフェースとする。

（2） ターゲットハードウェアに依存しないこと。
プロセッサとデバイスの接続方法、例えばデバイスレジスタが I/O 空間とメモリ空間のどちらに見えるか、あるいは何番地に見えるかといったことに関わらず利用できるように、インタフェースを宣言しなければならない。

4.3.4 サービスコールの設計

実装者は上位層が呼び出す PDIC のサービスコールを次のように設計する。

（1） もともと複数チャネルを持つデバイスならば、サービスコールのパラメータとしてチャネルを設ける。

（2） 必要に応じて PDIC 自体の初期化方法を設ける。
PDIC がスタックやデバイス以外にデータを置く場合などには PDIC 自体の初期化が必要になる。初期化のサービスコールは DSR としてだけではなく、静的 API として実装することもできる。

（3） 上位層に CBR が存在するならその CBR を登録する方法を設ける。
デバイスが割り込み要因となる場合などには上位層の CBR を PDIC から呼び出すことになる。実装者は上位層が用意すべき CBR のインタフェースとともに、その登録方法を宣言することが必要になる。
上位層の CBR の登録は静的 API として実装することもできる。上位層の CBR の呼び出しをソースコード中にあらかじめ埋め込むのでも構わない。
また上位層の CBR の登録は PDIC 自体の初期化とまとめたサービスコールとしてもよい。

（4） PDIC 初期化・上位層の CBR の登録以外では、デバイスの機能に対応したサービスコールを設ける。
実装者はデバイスの抽象化や高機能化を考えず、デバイスが持つ機能をそのまま提供する。ソフトウェアによる抽象化や高機能化は利用者に検討させればよい。
デバイスの機能を抽出するときには以下の点に注意する。

（a） 利用者はデバイスの全機能を扱えるか。
利用者がインタフェースの変更やインタフェースの新規追加を検討せずにすむように宣言する。

（b）デバイスを初期状態と同等に再設定できるか。
デバイスによっては利用者が再初期化などする場合がある。可能ならば初期状態と同等に再設定できるようにしておく。

（c） デバイスへの一連の処理か。
デバイスの仕様によりあらかじめ一連で処理するものとされた内容に対しては、できる限りまとめたサービスコールを設ける。

（d） 即時完了型処理要求・非同期型処理要求・非同期型処理要求のキャンセ

ルのどれに当てはまるか。

同期型処理要求や同期型処理要求のキャンセルに該当するようなサービスコールが PDIC に存在してはならない。

（ｅ） 非同期型処理要求には対応する処理要求完了通知(コールバック)が存在するか。

（５） 必要に応じてパラメータ・リターンパラメータとデバイスのビットパターンを一致させる。
デバイスの状態を提示するサービスコールなどでは、デバイスのビットパターンをそのままリターンパラメータにしてよい。また、デバイスのビットパターンをそのままパラメータにしてもよい。ただし、リターンパラメータと論理演算するための定数やパラメータとして指定する定数も PDIC で提供すること。

（６） サービスコールの仕様をドキュメント化する。
利用者があらためてデバイスの仕様書を読み直さなくてもよいように PDIC の仕様をドキュメント化する。

シリアル I/O の PDIC を実装する場合、サービスコールの構成として次のような例が考えられる。

サービスコールの機能	サービスコールの処理要求分類
PDIC 初期化	(初期化)
周辺デバイス初期化	即時完了型処理要求
データ送信	非同期型処理要求
データ受信	即時完了型処理要求
周辺デバイス送信状態提示	即時完了型処理要求
周辺デバイス受信状態提示	即時完了型処理要求
周辺デバイス送信割り込み状態提示	即時完了型処理要求
周辺デバイス受信割り込み状態提示	即時完了型処理要求
周辺デバイス受信エラー状態提示	即時完了型処理要求

4.3.5 コールバックの設計

デバイスが割り込み要因になる場合などに、実装者は上位層へのコールバックを次のように設計する。

（１） デバイスの機能に対応したコールバックを設ける。
原則として上位層へのコールバックは処理要求完了通知か事象通知のどちらかである。処理要求完了通知のコールバックには対応する非同期型処理要求のサービスコールが存在する。上位層へのコールバックの前処理・後処理によっては、ひとつの割り込み要因から複数回のコールバックが生じて構わない。

（２） 上位層の CBR を起動するとき CPU ロック解除状態であるようにする。
原則として CPU ロック状態で上位層の CBR を起動する。もしも CPU ロック状態である場合、利用者向け PDIC 仕様書に明記しておく。

（３） コールバックのパラメータおよびリターンパラメータは最小限にする。
コールバックの前処理で必然的に得られたものでない限り、コールバックのパラ

メータにしない。またコールバックの後処理で必要なものでない限り、コールバックのリターンパラメータにしない。ただし、CBR で必要な情報を取得する DSR が CBR から呼び出し不可能な場合、同等の情報をパラメータとして CBR へ渡せるようにする。

（４） 必要に応じてパラメータ・リターンパラメータとデバイスのビットパターンを一致させる。

（５） コールバックの仕様をドキュメント化する。
利用者があらためてデバイスの仕様書を読み直さなくてもよいように PDIC の仕様をドキュメント化する。

シリアル I/O の PDIC を実装する場合、上位層へのコールバックの構成として次のような例が考えられる。

コールバックの起動原因	コールバックの通知分類
データ送信完了通知	処理要求完了通知
データ受信通知	事象通知

4.3.6DSR の実装

PDIC の DSR は次の条件に注意して実装する。

（１） DSR と DSR の間の危険領域を排他制御する必要はない。
DSR の実行中に DSR が割り込んで実行されるときに、DSR が正しく動作しないとしても、実際にこのような状況が生じるかどうかは上位層による。このような状況を回避するために DSR 中で排他制御する必要はない。

（２） DSR と ISR の間の危険領域を排他制御する。
多くのターゲットハードウェア上では、DSR の実行中に ISR が割り込んで実行される。そこで DSR と ISR の間の危険領域は PDIC 中で排他制御する。
システムインタフェースレイヤーは割り込みを禁止するポータブルな手段として、loc_cpu と unl_cpu を提供している。これらはμITRON4.0 仕様の同名 API と同じ機能を持つので、必要ならば実装者はこの機能を用いる。

（３） カーネルやターゲットハードウェアに依存しない。
PDIC を移植するターゲットハードウェアにカーネルが存在するとは限らない。そこで PDIC はカーネルに依存しないように実装する。例えば PDIC はタスクを用いないようにする。カーネルの機能を用いる DIC は GDIC として PDIC の上位層に実装する。
また、割り込みの禁止やタイマの利用やデバイスへのアクセスといった処理については必ずシステムインタフェースレイヤーを呼ぶように実装する。

（４） DSR はデバイス以外の状態をできる限り DSR 呼び出し前の状態に戻してから復帰する。
例えば DSR 内部で CPU ロック状態にする場合、まずもとの状態を保存し、CPU ロック状態にし、復帰の前にもとの状態に戻す。上位層の CBR から呼び出せる DSR については利用者向け PDIC 仕様書に明記する。
上位層の CBR から呼び出される DSR についてはまだ検討が不十分である。当面は以下の点に注意する。

（ａ） DSR は原則として上位層の CBR から呼び出し不可能でよい。

上位層の CBR から呼び出せる DSR は、CBR から呼び出し可能であると仕様書に明記すること。

（b） DSR が呼び出されたとき、原則として CPU ロック解除状態であり、かつ割り込みはマスクされていない。

DSR によっては割り込みマスク状態または CPU ロック状態であると仮定してもよい。ただし、このように仮定した DSR はその仮定を仕様書に明記しておく。仕様書に何の記述もなければどのような状態からも呼び出し可能な DSR であると見なされる。

（c）上位層の CBR から呼び出すことになる DSR は、割り込みマスク状態から呼び出し可能にする。

（d） 上位層の CBR から呼び出すことになる DSR は、ISR に割り込まれる DSR の実行途中にも実行できるようにする。

4.3.7 ISR の実装

PDIC の CBR は 3 章で述べた割り込みシステムにより ISR として起動される。PDIC の ISR は次の条件に注意して実装する。

（1） 割り込みの途中で他の割り込みが入る可能性がある。
μITRON4.0 仕様の用語を使えば、ISR が起動した時は CPU ロック解除状態である。したがって割り込みシステムが多重割り込みをサポートしていれば、他の割り込みが入る可能性がある。

（2） 割り込みの途中で同じ割り込みが入ることはない。
割り込みシステムが多重割り込みをサポートしている場合でも、自分自身への割り込みや、より優先度の低い割り込みはマスクされているのが普通である。したがって同じ割り込みが入らないと見なしてよい。

（3） ISR と ISR の間の危険領域を排他制御する。
デバイスが複数の割り込み要因を持つ場合、多くのターゲットハードウェア上では、ISR の実行中に ISR が割り込んで実行される。そこで ISR と ISR の間の危険領域は PDIC 中で排他制御する。
システムインタフェースレイヤーは割り込みを禁止するポータブルな手段として、iloc_cpu と iunl_cpu を提供している。これらはμITRON4.0 仕様の同名 API と同じ機能を持つので、必要ならば実装者はこの機能を用いる。

（4） ISR の途中で割り込み優先順位を下げてはならない。
割り込み優先順位を下げても同じ ISR が起動しないような場合でも、その ISR に割り込まれている別の ISR が再起動する可能性がある。

（5） ISR が呼ばれてもデバイスの割り込み要因があるとは限らない。
割り込みシステムによってはひとつの割り込み信号線に複数の ISR を登録する場合がある。このような場合、ISR が呼ばれても割り込み要因が発生しているとは限らない。したがって ISR はまず割り込み要因の有無を調べ、要因がなければすぐに復帰するようにする。ISR 呼び出し時の要因の有無が判別できないようなデバイスは利用者向け PDIC 仕様書にその旨明記する。

（6） ISR はデバイスの割り込み要因を ISR 中で必ずクリアする。
デバイスの割り込み要因があった場合、割り込み要因をすべてクリアしたのちに ISR から復帰する。

（7） 実装者は ISR を実装する際に、上位層の CBR のインタフェースを決めてかま

わない。

また、ISR からは少なくとも次の関数を呼び出すことができる。
(1) システムインタフェースに属する各関数
(2) 上位層の CBR

4.4 アクセスインタフェース

4.4.1 概要

(1) アクセスインタフェースの目的

PDIC が DSR や ISR からデバイスを制御しようとする場合、なんらかの形でデバイスとの入出力が必要となる。しかしデバイスとの入出力の方法はターゲットハードウェアによって異なる。例えば、デバイスのレジスタが I/O 空間に接続されるか、メモリ空間に接続されるかの違いがある。あるいはメモリ空間に接続されていても、そのアドレスが異なることがある。

そこでターゲットハードウェアに依存する部分をアクセスインタフェースとして分離し、アクセスインタフェースを用いることで、デバイスとの入出力をターゲットハードウェアに依存しないように PDIC で記述できるようにする。

(2) アクセスインタフェースの分類

デバイスとの入出力は次の 2 つに大きく分類できる。

デバイスレジスタ	プロセッサとデバイスとの入出力のうち、数バイト(2^0 から 2^3 バイト)のレジスタを単位として実現されるもの。レジスタとレジスタの間に連続という概念はない。
共有メモリ	プロセッサとデバイスとの入出力のうち、連続したバイト領域で実現されるもの。領域内にどのような構造があるかはデバイス依存であり、レジスタのような単位を持たない。

それぞれの入出力に対応して、デバイスレジスタとのアクセスインタフェースと、共有メモリとのアクセスインタフェースの 2 種類のアクセスインタフェースを設ける。

(3) アクセスインタフェースの宣言

あるデバイス *device* のアクセスインタフェースは、PDIC の実装者がおのおのの PDIC に対して 1 つ用意するヘッダファイル *device*_sil.h で宣言する。
この *device*_sil.h は原則として該当する PDIC からのみインクルードされ、それ以外からは参照されない。これは、あるターゲットハードウェアで複数の PDIC を利用する場合に、PDIC ごとに異なる入出力機能を用いたアクセスインタフェースの定義を可能にするためである。

PDIC の移植者は実装者が用意したアクセスインタフェースを変更することで、その PDIC をターゲットハードウェアに移植する。実装者は、なるべく *device*_sil.h のみ書き換えるだけで PDIC が移植できるように PDIC を実装する。

4.4.2 アクセスインタフェースの仕様

(1) デバイスレジスタからデータを読む

[C 言語 API]

```
VB data = sil_reb_reg(ID regid);
VH data = sil_reh_reg(ID regid);
VW data = sil_rew_reg(ID regid);
```

[パラメータ]

ID	regid	データを読むデバイスレジスタの ID

[リターンパラメータ]

VB/VH/VW	data	読んだデータ

[機能]

指定したデバイスレジスタからデータを読む。デバイスレジスタのサイズにより API が異なる。

[補足説明]

API はデバイスとプロセッサとの間のバス幅で異なっているのではない。

(2) デバイスレジスタへデータを書く

[C 言語 API]

```
void sil_wrb_reg(ID regid, VB data);
void sil_wrh_reg(ID regid, VH data);
void sil_wrw_reg(ID regid, VW data);
```

[パラメータ]

ID	regid	データを書くデバイスレジスタの ID
VB/VH/VW	data	書くデータ

[機能]

指定したデバイスレジスタへデータを書く。デバイスレジスタのサイズにより API が異なる。

[補足説明]

API はデバイスとプロセッサとの間のバス幅で異なっているのではない。

(3) 共有メモリからデータを読む

[C 言語 API]

```
VB data = sil_reb_mem(VP mem);
VH data = sil_reh_mem(VP mem);
VW data = sil_rew_mem(VP mem);
```

(リトルエンディアン)

```
VH data = sil_reh_lem(VP mem);
VW data = sil_rew_lem(VP mem);
```

(ビッグエンディアン)

```
VH data = sil_reh_bem(VP mem);
VW data = sil_rew_bem(VP mem);
```

[パラメータ]

VP	mem	データを読む共有メモリ上のアドレス

[リターンパラメータ]

VB/VH/VW	data	読んだデータ

[機能]

共有メモリ上の指定したアドレスからデータを読む。デバイスレジスタの場合と同様に、アドレスから読むデータのサイズにより API が異なる。

また、共有メモリ上に置かれているデータの並び順(リトルエンディアン・ビッグエンディアン)により API が異なる。エンディアンを指定しない場合、プロセッサにとって自然な並び順となる。

(4) 共有メモリへデータを書く

[C 言語 API]

```
void sil_wrb_mem(VP mem, VB data);
void sil_wrh_mem(VP mem, VH data);
void sil_wrw_mem(VP mem, VW data);
```

(リトルエンディアン)

```
void sil_wrh_lem(VP mem, VH data);
void sil_wrw_lem(VP mem, VW data);
```

(ビッグエンディアン)

```
void sil_wrh_bem(VP mem, VH data);
void sil_wrw_bem(VP mem, VW data);
```

[パラメータ]

VP	mem	データを書く共有メモリ上のアドレス
VB/VH/VW	data	書くデータ

[機能]

共有メモリ上の指定したアドレスへデータを書く。デバイスレジスタの場合と同様に、アドレスへ書くデータのサイズにより API が異なる。

また、データを共有メモリ上に置くときの並べ順(リトルエンディアン・ビッグエンディアン)により API が異なる。エンディアンを指定しない場合、プロセッサにとって自然な並び順となる。

(5) 共有メモリからデータをブロック転送する

[C 言語 API]

```
void sil_rek_mem(VP mem, VP data, UINT len);
```

[パラメータ]

VP	mem	データの転送元となる共有メモリ上の領域の先頭アドレス
VP	data	データの転送先となる領域の先頭アドレス
UINT len		データを入れる領域の長さ

[機能]

共有メモリ上の指定した領域からデータを読む。データの並び順は転送元と転送先との間で保持される。

(6) 共有メモリへデータをブロック転送する

[C 言語 API]

```
void sil_wrk_mem(VP mem, VP data, UINT len);
```

[パラメータ]

VP_INT	mem	データの転送先となる共有メモリ上の領域の先頭アドレス
VP	data	データの転送元となる領域の先頭アドレス
UINT len		データを入れた領域の長さ

[機能]

共有メモリ上の指定した領域へデータを書く。データの並び順は転送元と転送先との間で保持される。

(7) そのほかのアクセスインタフェース

以上に示したもの以外にも、例えば次のようなアクセスインタフェースが考えられる。

(a) デバイスレジスタ中の特定ビットを操作するアクセスインタフェース
プロセッサによってはビット単位の入出力命令を備えている場合がある。その命令による高速化を期待したアクセスインタフェースがありうる。

(b) デバイスへの制御端子を設定するアクセスインタフェース
デバイスによってはデバイスレジスタ・共有メモリの操作以外に、デバイスへの制御端子をソフトウェアから設定する場合がある。例えば、デバイスのリセット端子をプロセッサの汎用ポート端子の 1 ビットで制御する場合などである。この操作を抽象化したアクセスインタフェースがありうる。

あるデバイスがあらかじめアクセスインタフェースとして定義されていない機能を必要とする場合、実装者は独自のアクセスインタフェースを定義した上で PDIC を実装する。

実装者は独自に定義したアクセスインタフェースを *device*_sil.h で宣言し、その仕様を移植者向け PDIC 仕様書に明記する。独自に定義したアクセスインタフェースの名前も sil_ function と sil_で始まるようにする。なお多くの場合、アクセスインタフェースはデータサイズごとのバリエーションを持つ。

移植者は PDIC 独自のアクセスインタフェースについてもターゲットハードウェア上に移植する。

4.4.3 デバイスレジスタと共有メモリの定義

アクセスインタフェースで用いるデバイスレジスタの ID や共有メモリ上のアドレスはヘッダファイル *device*_sil.h で定数(マクロなど)として宣言する。

デバイスレジスタおよび共有メモリを示す定数の名前は、次の規則に基づいて実装者が決定する。

（1） アクセスインタフェースのパラメータまたはリターンパラメータに指定する可能性のあるすべてのデバイスレジスタ、共有メモリを定義する。

（2） DEV_REGNAME, DEV_MEMADDRESS のように、"DEV_"で名前を始める。
デバイスの名称でなく"DEV_"で名前を始める理由は、似たデバイス間で PDIC を共有したいという要求に答えるためである。

（3） "DEV_"に続く部分はデバイスのハードウェア仕様書で定義された名前に合わせる。

共有メモリには連続したアドレスからなる領域が存在するが、デバイスレジスタにそのような保証はない。またデバイスレジスタ間の順番などの保証もない。

したがって実装者はデバイスレジスタの ID を計算するような PDIC を実装してはならないし、PDIC がそうならざるをえないようなかたちでデバイスレジスタを定義しないように注意する。

4.4.4 アクセスインタフェースの実装

（1） 実装者の作業

実装者は PDIC で必要なすべてのアクセスインタフェースをヘッダファイル *device*_sil.h にマクロとして定義する。実装者はこれらのマクロを呼び出すように PDIC を実装して移植者に提供する。
ここで実装者の用意するマクロは、実際には何もしなくて構わない。

デバイスレジスタについて、実装者は例えば sil_wrh_reg(DEV_REGNAME+2, 0x1234) のような、特定のアドレスへのマッピング方法を想定して PDIC を実装してはならない。
sil_wrh_reg (DEV_REG(2), 0x1234)などのように、アドレステーブルの値が展開されるように指定することは可能である。ただしこの場合も、特定のアドレスへのマッピングを想定しないことが条件となる。

（2） 移植者の作業

移植者は PDIC が利用するアクセスインタフェースをターゲットハードウェアで実現するための関数を用意し、ヘッダファイル *device*_sil.h に定義された内容を変更する。これにより PDIC はターゲットハードウェアに移植される。
もちろん、移植者はアクセスインタフェースをマクロだけで処理してもかまわない。

アクセスインタフェースのうち、共有メモリとのブロック転送は memcpy()で定義する場合もある。しかし共有メモリが通常のメモリと違ったアクセス方法を持つ場合は、read と write で違う関数を定義する必要がある。例えば、共有メモリへ 1 回書き込むごとに一定時間の待ちが必要な場合がある。

ターゲットハードウェア上で最適なアクセスインタフェースとなるかどうかは、実態がマクロであれ関数であれ、移植者が定義した内容による。移植者は自分の定義したアクセスインタフェースがターゲットハードウェア上でどのように実行されるのか理解している必要がある。

4.4.5 デバイスの初期設定

DSR や ISR がデバイスを制御するに当たって、あらかじめ詳細な情報が必要な場合がある。例として、調歩同期シリアルポートの初期設定などをあげることができる。

このような値が必要な PDIC を実装した場合、実装者は初期値をマクロとして用意しておき、移植者が必要に応じてその値を書き換える。

実装者は初期値を設定するマクロをヘッダファイル *device*_cfg.h に定義する。マクロの名前は CFG_VALUE のように"CFG_"で始まることを推奨する。

実装者はマクロの値がどのような意味を持ち、どのような範囲が許されるかなどの詳細な情報を利用者向け PDIC 仕様書に提示しなければならない。

また、同様の初期設定は静的 API で実現してもよい。このような静的 API も *device*_cfg.h に定義するが、定義するヘッダファイル以外については PDIC のサービスコールに準じる。

ここで初期設定の実現方法として静的 API を選択するのは、

・パラメータの数が少ない。
・同等のサービスコールが存在する。

場合だと考えられるが、この点についてはまだ十分検討されていない。

4.5 システム提供関数群

4.5.1 概要

システムインタフェースレイヤーのうちデバイスに依存せず、ターゲットハードウェアのプロセッサや割り込みシステムに依存する部分をシステム提供関数群と呼ぶ。

システム提供関数群はターゲットハードウェアにおける割り込みシステムやタイマの操作、プロセッサの性能を隠蔽する。PDIC はシステム提供関数群を呼び出すことで、これらの機能をターゲットハードウェアに依存しないように記述できる。

現在のシステム提供関数群には次の機能が含まれる。

(1) CPU ロック状態の制御
(2) 微小時間の遅延

これらの機能を利用する場合、実装者は sil.h をインクルードする。

移植者はターゲットハードウェアに応じてこれらの機能を用意しておかなければならない。ただし、CPU ロック状態の制御についてはμITRON4.0 仕様が同名・同機能の API を提供しているので、ターゲットハードウェアにμITRON4.0 が搭載されていれば、移植者は微小時間の遅延機能のみ準備すればよい。

4.5.2 システム提供関数群の仕様

(1) CPU ロック状態の制御

システム提供関数群は CPU ロック状態の制御として以下の API を備える。これらはμITRON4.0 のシステムコールと同名・同機能のインタフェースなので、詳細についてはμITRON4.0 仕様を参照すること。

loc_cpu, iloc_cpu	CPU ロック状態への移行
unl_cpu, iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照

(2) 微小時間の遅延

[C 言語 API]

```
void dly_nse(UINT dlytim);
```

[パラメータ]

UINT dlytim　遅延時間(1ns を単位とする相対時間)

[機能]

dlytim で指定される時間(ns)の間、プログラムの進行を一時的に停止する。

[補足説明]

デバイスの仕様に従って、一定時間デバイスを操作できないときに呼び出す。したがって停止した状態の解除はできない。

dly_nse から復帰するときは、少なくとも dlytim で指定される以上の時間(ns)が経過した後であることが保証されなければならない。これは dlytim に 0 が指定されたときも同じである。

4.6 PDIC が使う名前

4.6.1名 前空間の問題

PDIC を移植するターゲットハードウェアでは、カーネルを含めたすべてのプログラムをひとつのオブジェクトにリンクする場合がある。

このような場合に、ある PDIC と他の PDIC やソフトウェア部品とが同じ名前を使用していると名前の衝突が発生する。名前が衝突した場合、移植者はいずれかのソースコ

ードを書き換えなければならない。
したがって、PDIC はできるかぎり衝突を避けるように名前を用いる必要がある。

ここではあるデバイスの名前を *device* とした場合の、ファイル・データ型・関数(定数・変数)・マクロの命名のガイドラインを示す。

4.6.2 ファイルの命名

PDIC で使うファイル名については、「PDIC の実装」の「ファイル構成」を参照すること。

4.6.3 データ型の命名

PDIC と上位層のインタフェースで利用するデータ型の名称は T_DEVICE_YYY という形にし、*device*.h で宣言する。一般に、YYY には操作などの対象となるものの名前をあてる。

PDIC 内部でのみ利用するデータ型の名称は自由な形でよいが、*device*_cfg.h または *device*_*.h で宣言する。

4.6.4 関数・定数・変数の命名

グローバルな関数・定数・変数は、他のソフトウェアと名前が競合する可能性が高い。命名の問題以前に、実装者はファイル外部から参照する必要のない関数・定数・変数を static として宣言する。

関数名・定数名・変数名は周辺デバイスの名前を接頭語としてつけることにより、他のソフトウェア部品との名前の競合を避ける。詳しくは次のようにする。

(1) PDIC が上位層にインタフェースとして提供する関数の名称は *device*_xxx_yyy の形とする。
xxx で操作の方法、yyy で操作の対象をあらわす。例えば i8251 という周辺デバイスの PDIC であれば、i8251_xxx_yyy という名称にする。

(2) 上位層が PDIC へ渡すパラメータや、PDIC からのリターンパラメータとして用いられる定数の名称は TUU_DEVICE_XXXXX の形とする。
ここで XXXXX は任意の文字列である。
TUU は次のように用いる。

TA_DEVICE_XXXXX	*device* の属性値
TSZ_DEVICE_XXXXX	XXXXX のサイズ
TBIT_DEVICE_XXXXX	XXXXX のビット数
TMAX_DEVICE_XXXXX	XXXXX の最大値
TMIN_DEVICE_XXXXX	XXXXX の最小値

(3) PDIC 内部に閉じて使われる関数・定数・変数のうち、オブジェクトファイルのシンボル表に登録され外部から参照できるものの名称は *_device_* または *_DEVICE_* で始まる名称にする。

上位層とのインタフェースとして利用する名前は *device*.h で宣言する。一方、PDIC 内部でのやりとりに使われる名前は *device*_cfg.h や *device*_sil.h や *device*_*.h で宣言する。static な名前はヘッダファイルで宣言しなくてよい。

4.6.5 マクロの命名

マクロは、その役割に応じ関数・定数・変数とみなして命名する。

4.7 PDIC に関する文書

利用者があらためてデバイスの仕様書を読み直す必要があるようでは、または移植者が移植に試行錯誤するようでは、PDIC が利用者や移植者の工数をじゅうぶん削減できたとは言えない。そこで実装者は利用者向け PDIC 仕様書と移植者向け PDIC 仕様書の 2 つを提供することで、利用者や移植者の工数を削減する。

4.7.1 利用者向け PDIC 仕様書

利用者向け PDIC 仕様書には次の内容が含まれる。

（1） デバイスの概要
デバイスの分類やその特徴について簡単に述べる。

（2） PDIC のサービスコール・静的 API の仕様
API ごとにアクセスインタフェースの仕様などと同様の書式で、次の内容を説明する。
(a) API の名前と概要
(b) C 言語 API
(c) パラメータ
パラメータごとに、その型と名前と意味を説明する。
(d) リターンパラメータ
リターンパラメータごとに、その型と名前と意味を説明する。また、リターンパラメータの値ごとにその意味を説明する。なお、一般に PDIC は ITRON のエラーコードに相当するものを返さない。
(e) 機能
この API が処理要求のどの分類に属するか明記し、この API がどのように振る舞うか、処理要求の分類を踏まえて説明する。
（f） 補足説明
この API がリエントラントかどうか明記する。
また、この API が上位層の CBR から呼び出し可能かどうか、あるいはほかの条件があればここに明記する。

（3） PDIC から呼び出す上位層の CBR の仕様
CBR ごとに PDIC のサービスコール・静的 API の仕様と同様の内容を説明する。ただし CBR の場合、機能の説明に次の内容も含む。
(a) 通知の分類
(b) CBR の通知内容
(c) CBR 前後の処理内容の説明

（4） 定数・マクロの説明
初期化用マクロなど、パラメータやリターンパラメータとして説明されていないものの説明を別途におこなう。

4.7.2 移植者向け PDIC 仕様書

移植者向け PDIC 仕様書には次のような内容が含まれる。

（1） ファイル構成 PDIC を構成するファイルの一覧を用いて、移植時に変更が必要な箇所を指示する。

（2） 使用した ANSI C ライブラリ

（3） 関数デバイスの機能と割り込み要因の説明
(a) デバイスの機能と DSR の対応
(b) デバイスの割り込み要因と CBR(ISR)・上位層の CBR の対応
(c) デバイスの機能のうち、PDIC から扱えないものこの PDIC で独自に定義したアクセスインタフェースの仕様

（4） PDIC の動作を確認したターゲットハードウェアに関する情報
動作を確認したターゲットハードウェア上にどのようなシステムインタフェースレイヤーを作ったかについて述べる。
(a) プロセッサとデバイスの接続方法やアドレスプロセッサのエンディアンやデータアライメント
(b) コンパイラ
(c) カーネル

（5） 提供する PDIC が条件コンパイルできるなら、その指定方法

4.8 未検討課題

（1） 同じデバイスが複数ある場合
ターゲットハードウェア上に同じデバイスが複数ある場合、どのように扱うべきか。

（2） 複数の異なる PDIC を同時に利用する GDIC への対応
複数の PDIC をマージして最適化するような場合、PDIC の定義方法、特にシステムインタフェースレイヤーの定義が競合するのをどう回避するのか。

（3） 割り込みコンテキストから呼び出される DSR
割り込みコンテキストから呼び出される DSR と、そのほかの DSR を分けて宣言するべきか。

（4） 個別割り込みの停止
個別の割り込みを停止させるようなシステム提供関数は必要か?PDIC がデバイスからの割り込みを停止させればじゅうぶんではないか?
特定の ISR だけを停止し、割り込みが入ってもその ISR を呼び出さないようにした場合、あとで再開したときに割り込みが入るようにするべきか?特定な ISR とそのレベルまで停止する?部分的な割り込みを止める機能があるとよいか?

（5） DMA
PDIC でどのように DMA を扱うか。DMA コントローラの PDIC が存在するのか。それとも DMA はシステムインタフェースレイヤーに入るのか。

（6） エンディアンの考慮

（7） PDIC から利用できる標準 C ライブラリ関数

（8） より具体的な実装テクニック

5 GDIC の設計ガイドライン

5.1 はじめに

この章では、GDIC を設計・実装するためのガイドラインを示す。ただし、実証例が不足しているため本稿ではその基本構想を紹介するにとどめる。また、デバイスを初期化するためのコンポーネントについては、議論がなされていないため、今回は割愛する。

5.2 概要

GDIC は、PDIC に機能を付加し、アプリケーション・ソフトウェアやミドルウェアが使いやすい様にデバイスを抽象化したり高機能化したりするための層である。GDIC 層は、PDIC 層の上に、必要に応じて複数積み重ねる事が出来る。

GDIC の各層は、上位の層にインターフェイスを提供する GDIC DSR と下位の層にインターフェイスを提供する CBR の集合体である。

GDIC の実装は PDIC やカーネル等、システムを構成する様々な要因に依存するため、システム毎に開発ないしカスタマイズする事となる。特に PDIC 上に直接積まれる層については、デバイス提供者が雛形を提供し、システム開発者はその雛形をカスタマイズする開発方法を推奨する。この雛形には、再利用性・可読性の高さだけでなく、PDIC を使用するための約束事を満たすことが求められる。本章ではそのためのガイドラインを提供する予定である。

5.3 GDIC のモデル

GDIC は、以下の様な階層構造を採る。

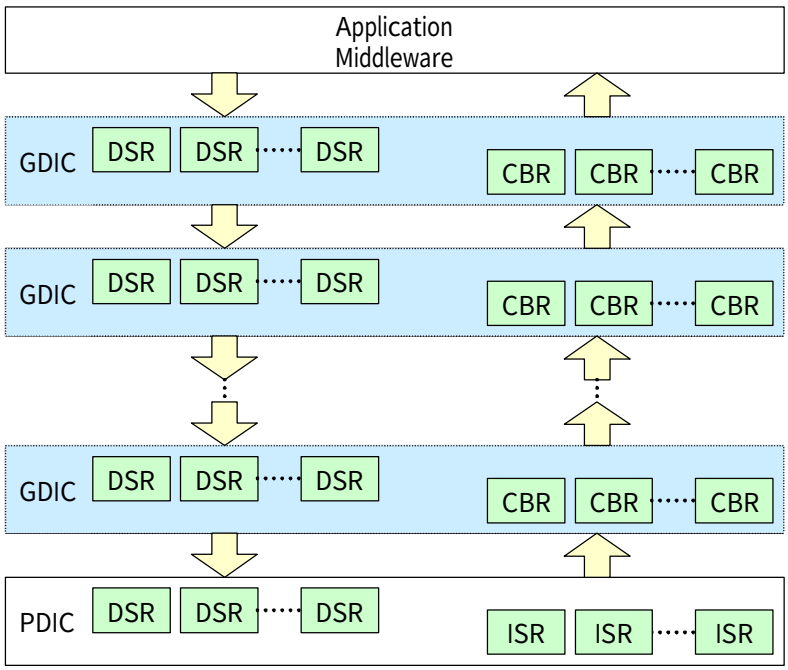


図 5-1 ＧＤＩＣの階層構造

階層の数は何段でも増やす事が出来るが、階層を飛び越えて他の層にアクセスする事

は認めない。トランスペアレント属性を持つデータは、各層がデータや構造体をそのまま次の層に渡す事により、その透過性を実現する。

5.4 GDIC の実装方法

5.4.1 はじめに

GDIC 実装のガイドラインを以下に示す予定。実証例が不足しているため今回はその概要のみを紹介する。また、初期化部分は割愛する。

5.4.2 GDIC モジュールの実装

GDIC をタスク型として実装するか共有ライブラリ型として実装するかは自由に選択できる。タスク型にすると、

- 複数の処理を並列に動作させる
- 個々の処理に異なる時間制約・優先度を与える
- 処理の状態を見通しよく管理する

といったことが容易に実現できる。一方、共有ライブラリ型には、

- 他の機能からの呼び出しの形で起動できる
- 別々の機能から呼び出された処理を並列に動作させる事が出来る
- カーネルのオーバヘッドが少ない

という利点がある。DIC 開発者はこれらを考慮に入れ、DIC 全体をシングルタスク型、マルチタスク型、共有ライブラリ型のいずれにするかを決定することとなる。GDIC 提供者は、この DIC 全体のアーキテクチャとその中での GDIC の位置を想定し、それに合わせた実装方法を選択する事が求められる。DIC に様々なバリエーションが考えられるのであれば、同一機能の GDIC をタスク型と共有ライブラリ型の両方とも揃えておく事も有効である。

5.4.3 主な付加機能の例

GDIC によって付加する機能の例として、ここでは PDIC がガイドライン上の制約によって実装できない機能を取り上げる。PDIC が実装できない機能には、大きく分けて以下の 4 種類がある。

- リソースの確保
- カーネルに依存する機能の利用
- 同期型サービスの提供
- デバイス名に依存しない API(特に関数名)の提供

以下に、これらの制約を補うために GDIC で付加する機能の例をそれぞれ示す。

付加機能の例1) リクエストのキューイング（リソースの確保）

リクエストキューに必要なメモリを確保し、複数のリクエストを蓄える機能を GDIC で付加する。リクエストの処理順序を優先度に応じて変更する機能も可能。

付加機能の例2) 排他制御（カーネルに依存する機能の利用）

カーネルの提供するセマフォを利用し、排他制御を実現する GDIC を付加する。

付加機能の例3) プリンタへの出力（同期型サービスの提供）

プリンタ・バッファ内のデータの処理が完了するまで待ちに入り、バッファが空になったら次のデータを送信するような機能を GDIC で付加する。用紙切れに対してエラーメッセージを出すとともに、用紙がセットされるまで待ちに入る機能を付

加することも可能。

付加機能の例4）汎用的な関数名への変更（デバイス名に依存しない API の提供）

SIO の様な枯れたデバイスに対し、汎用的な API を予め定義し、それに従って GDIC の DSR を用意する。例えば、PDIC がデバイス名 *device*_で始まる関数名であるのに対し、sio_で始まる関数とする。これにより、デバイスを同一機能の別のデバイスに置き換えても、この GDIC より上位の層のコード修正は不要とすることが出来る。

5.4.4 制限事項

CBR は内部で待ちに入ってはならない。また、内部で待ちに入るような他のコンポーネントを関数呼び出ししてもいけない (別タスクとして起動する事は問題ない)

5.4.5 推奨事項

GDIC、特に雛形として提供される GDIC の DSR には、再利用性・可読性を高めるために、μITRON4.0 仕様の『2.2 API の名称に関する原則』を遵守する事を推奨する。この原則の概要は以下の通り。詳細は、μITRON4.0 仕様書を参照の事。

- ソフトウェア部品識別名
  ソフトウェア部品の API の名称に用いるために、その種類を 2〜4 文字程度であらわすソフトウェア部品識別子を定める。以下、ソフトウェア部品識別名を *www* と記述する(大文字の場合は *WWW*)。
- サービスコール
  サービスコールの名称は、*xxx* で操作の方法、*yyy* で操作の対象をあらわし、*www_xxx_yyy* の形とする。

- コールバック
  パラメータの名称の原則に従う。
- 静的 API
  対応するサービスコールの名称を大文字で表記したものとする。
- パラメータとリターンパラメータ
  すべて小文字で 4〜7 文字程度とする。また、パラメータの意味に応じて、所定の prefix/suffix を付ける。また、名称が同じパラメータのデータ型は揃える。
- データ型
  大文字で 2〜10 文字程度とする。ポインタとパケット(構造体)にはそれぞれ、〜P、T_〜 の形式の名称をつける。ソフトウェア部品で用いる構造体の名称は、T_*WWW*_〜 の形式とする。
- 定数
  定数の名称はすべて大文字で記述する。同じ意味の定数が既に ITRON 仕様共通定数で規定されていれば、これを利用し、新たに定義する事は避ける。また、定数の意味に応じて、所定の prefix を付ける。ソフトウェア部品固有の定数であれば、prefix の後ろにソフトウェア部品識別名を付ける。例えば、オブジェクトの属性値をあらわす定数であれば TA_*WWW*_〜 など。
- マクロ
  マクロの名称はすべて大文字で記述し、定数と同様の原則に従って定める。
- ヘッダファイル

DIC API のプロトタイプ宣言やマクロなど、DIC 利用者が上位のソフトウェア内で参照する必要がある情報を含むヘッダファイルの名称を"*www*.h"、システムアクセスレイヤの関数のテンプレートやレジスタのアドレスを示すマクロなど、DIC を移植するためにユーザが書き換える必要のある項目を含むヘッダファイルの名称を"*www*_cfg.h"とする。

- ソフトウェア部品の内部識別子
  ソフトウェア部品の内部に閉じて使われるルーチンやメモリ領域の識別子の内、オブジェクトファイルのシンボル表に登録され外部から参照できるものの名称は、_*www*_～ もしくは_*WWW*_～ の形式とする。

5.5 GDIC のドキュメント

GDIC のドキュメントには、API や機能、必要リソースの説明などの基本的な項目のほかに、以下の要素が明記されることとする。

- API の種別
  API 毎に、"2.3 DIC の API の枠組み"のどのパターンに該当するのかを示す。
- リエントラントであるかどうか
  複数のタスクから同時に呼ばれる事が考慮されているかどうかを明示する。
- ポーティングの手引き
  他カーネルへの対応など典型的な移植例を想定し、その際に変更する部分を例示する。

5.6 記述例

ここでは特に、PDIC 層の上に直接重ねる GDIC 層のテンプレートを示す。ただし、エラー時の処理は割愛してある。

非同期処理型 API の例

```
GDIC_DSR1()                          /* アプリケーションから呼ばれて起動 */
{
        リクエストをキューイングする
        I/O 処理中かどうか確認するための PDIC DSR を呼ぶ
        if (I/O 処理中でない) {
                I/O 処理開始用 PDIC DSR を呼ぶ       /* キューの先頭のリクエストを実行 */
        }
}
```

```
GDIC_CBR1()                          /* PDIC ISR の完了要求を受けて起動 */
{
        処理結果を得る
        処理要求したタスクに完了通知
        if (キューにリクエストが残っている) {
                I/O 処理開始用 PDIC DSR を呼ぶ       /* キューの先頭のリクエストを実行 */
        } else {
                I/O 停止処理用の PDIC DSR を呼ぶ
        }
}
```

同期処理型 API の例

```
GDIC_DSR2()                                  /* アプリケーションから呼ばれて起動 */
{
        排他制御をする
        I/O 処理開始用 PDIC DSR を呼ぶ        /* 受け取ったリクエストを実行 */
        待ちに入る
        /* PDIC ISR の完了通知を受けて以下を実行 */
        処理結果を得る
        I/O 停止処理用の PDIC DSR を呼ぶ
        排他制御を解除する
        処理要求したタスクに完了通知
}
```

5.7 GDIC の課題

DIC 全体を通して言える事だが、特に GDIC は実装検討例が少なく、机上検討も不十分である。そのため、仕様の面でも導入効果の面でも未検討の部分が山積みされている。そこで、今後の検討にあたり、多くの方のご意見・ご協力をお願いしたい。以下に、主な課題を列挙する。

（1） 仕様未決定部分
- ◇ API の枠組み
- ◇ GDIC の実装方法
  －エラー時の処理

（2） 効果未確認部分
- ◇ 可読性
- ◇ 移植性
- ◇ 速度・応答性

6 ケーススタディ

これまで述べてきたDICアーキテクチャに則り、典型的なデバイスと思われる、ＳＩＯデバイスを例してケーススタディを行う。

6.1 デバイスの概要

代表例として、Intel系8251を取上げる。

8251 は RS232C などに使用される USART(Universal Synchronous Asynchronous Receiver-Transmitter)であり、以下の機能を有する。

(1) 通信データのシリアル－パラレル変換
(2) 送受信データの２重バッファリング
(3) 同期型プロトコル処理機能
(4) モデム制御

6.2 ハードウェア構成

(1) ハードウェア構成

図6.1に想定するハードウェア構成図を示す。

データ線は 8 ビット接続しており、コントロールレジスタとデータレジスタの切替えはアドレスの再開ビットで行う。また SIO デバイスの RxRDY(データ受信)、TxRDY(送信バッファ空)で割込みを CPU に通知する。

以下の図ではその他の制御線はＣｏｎｔｒｏｌとしてまとめている。

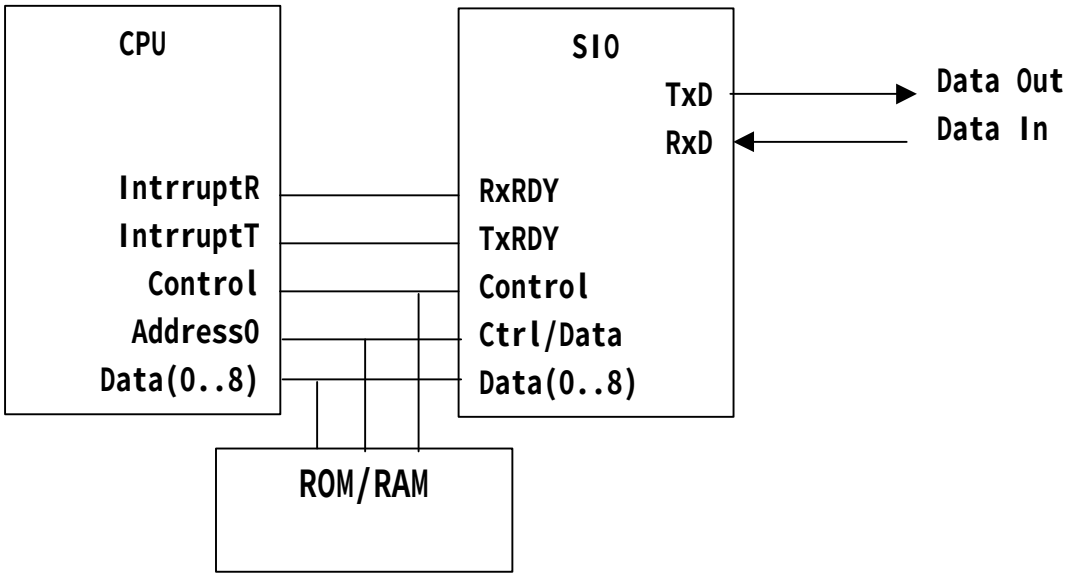
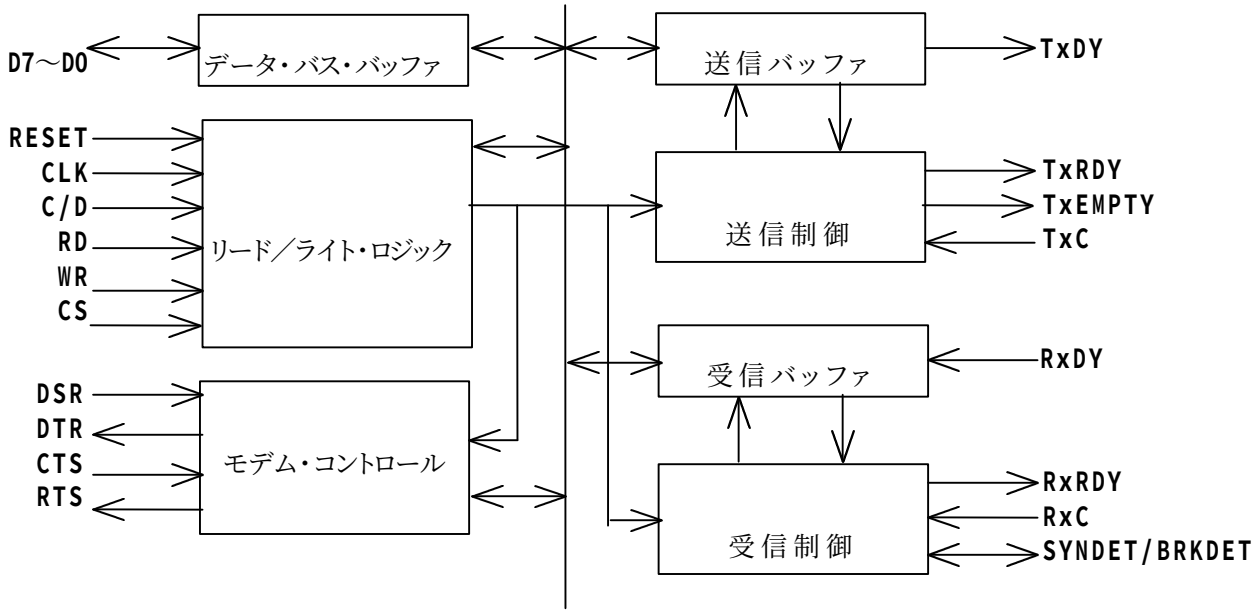


図 6.1 想定するハードウェア構成図

(2) デバイス内部構成



6.3 デバイスの説明

(1) 8251 の特徴

サポート通信方式　　非同期式および同期式

同期式の内容：

キャラクタ同期 BSC(2 バイトのキャラクタ同期)(同期キャラクタ指定可能)

非同期式の内容：

通信キャラクタ長 5,6,7 または 8 ビット
ストップビット長 1,1.5 または 2 ビット
パリティビット　なし、偶数、または奇数

(2) レジスタ構成

(a) 送受信データレジスタ

リード時：8251 よりデータを読み出す

ライト時：8251 へデータを書込む

(b) コマンドレジスタ

ライト時：8251 の動作を指示する。

コマンドレジスタの各 bit の意味は以下の通り

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

bit	記号	説明
D0	TxEN	Tx Enable：1＝送信可、0＝送信不可
D1	DTR	DTR：1＝DTR を L にする
D2	RxE	Rx Enable：1＝受信可、0＝受信不可
D3	SBRK	Send Break：1＝Break 送出
D4	ER	Error Reset：1＝エラーリセット
D5	RTS	RTS：1＝RTS を L にする
D6	IR	Internal Reset：1＝内部リセット
D7	EH	Enter Hunt Mode：1＝SYNC キャラクタのサーチ(同期モードのみ)

(c) モードレジスタ

ライト時：8251 の動作モードを設定する。

リセットコマンド発行直後のみコマンドレジスタがモードレジスタになる。

モードレジスタの各 bit の意味は以下の通り

<table>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>SYNC ｷｬﾗｸﾀ
1:ｼﾝｸﾞﾙ
0:ﾀﾞﾌﾞﾙ</td><td>SYNC 検出
1:外部
0:内部</td><td rowspan="2">ﾊﾟﾘﾃｨﾁｪｯｸ
1:even
0:odd</td><td rowspan="2">ﾊﾟﾘﾃｨ
1:有効
0:無効</td><td colspan="2" rowspan="2">ｷｬﾗｸﾀ長
0,0:5bit
0,1:6bit
1,0:7bit
1,1:8bit</td><td colspan="2">0,0:固定
(同期式)</td></tr>
<tr><td colspan="2">ｽﾄｯﾌﾟﾋﾞｯﾄ
0,0:無効
0,1:1bit
1,0:1.5bit
1,1:2bit
(非同期式)</td><td colspan="2">ﾎﾞｰﾚｰﾄ設定
0,1:1×
1,0:16×
1,1:64×
(非同期式)</td></tr>
</table>

(e) ステータスレジスタ

リード時：8251 の状態を参照する。

コマンドレジスタの各 bit の意味は以下の通り

D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYNDET	FE	OE	PE	TxE	RxRDY	TxRDY

bit	記号	説明
D0	TxRDY	1:送信バッファ空
D1	RxRDY	1:受信データあり
D2	TxE	1:送信レジスタ空
D3	PE	1:パリティエラー発生
D4	OE	1:オーバランエラー発生
D5	FE	1:フレーミングエラー発生
D6	SYNDET	1:SYNC キャラクタ検出
D7	DSR	DSR の状態

6.4 プリミティブ DIC DSR(Device Service Routine)の機能

8251 の機能をほぼ加工すること無く提供し、かつ、PDIC の上に GDIC を構築し易い構成とする。

6.4.1初期化機能

(1) 概要

デバイスを初期化し通信条件の設定を行う。

(2) C 言語 API

(a) コーリングシーケンス

ER ercd = i8251_ini_sio(ID chn_num,T_ISIO_8251 *pk_sio_8251)

(b) パラメータ

ID	chn_num	チャネル番号
T_ISIO_8251	*pk_sio_8251	デバイス情報構造体

T_ISIO_8251 の構造

（未定）

(c) リターンパラメータ

ER	ercd	エラーコード

(d) エラーコード

（未定）

(3) 機能

channle_number で示すチャネル番号の SIO デバイスを、pk_sio_8251 で指示される内容で初期化し、受信を有効にする。

6.4.2データ送信

(1) 概要

指定されたデータを送信する。

デバイスが送信中である場合(送信バッファに書込めない)は即時エラー終了する。送信完了まで PDIC 内で待つ選択もあるが、PDIC の設計方針(即時終了)に基づき即時エラー終了とする。なお、送信完了は割込みで通知されるものとする。

(2) C 言語 API

(a) コーリングシーケンス

```
ER ercd = i8251_snd_sio(ID chn_num,UB chr)
```

(b) パラメータ

ID	chn_num	チャネル番号
UB	chr	送信するデータ

(c) リターンパラメータ

ER	ercd	エラーコード

(d) エラーコード

デバイス未初期化

デバイスビジー

(3) 機能

chn_num でしめすチャネル番号の SIO デバイスに対し以下を実施する。

デバイスが未初期化状態の場合は、(デバイス未初期化)でエラー終了する。デバイスの送信バッファが空いている場合は、chr で指定されるデータをデバイスの送信バッファに書込み、送信を有効にする。デバイスの送信バッファに空きが無い場合は、(デバイスビジー)でエラー終了する。

6.4.3データ受信

(1) 概要

デバイスよりデータを受信する。

デバイスがデータを受信していない場合は即時エラー終了する。データが受信されるまで PDIC 内で待つ選択もあるが、PDIC の設計方針(即時終了)に基づき即時エラー終了とする。なお、受信完了は割込みにて通知されるものとする。

(2) C 言語 API

(a) コーリングシーケンス

```
INT chr = i8251_rcv_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

INT	chr	受信したデータ　または　エラーコード エラーコードは負数

(d) エラーコード

デバイス未初期化
データ未受信

(3) 機能

chn_num で示す SIO デバイスに対し以下を実施する。

デバイスが未初期化状態の場合は、(デバイス未初期化)でエラー終了する。

デバイスの受信バッファにデータが受信されている場合は、受信したデータを符号拡張せずデータをリターンパラメータとして終了する。デバイスの受信バッファにデータが受信されていない場合は、（データ未受信）でエラー終了する。

6.4.4デバイス送信状態提示

(1) 概要

デバイスがデータを送信している否かを返す。

(2) C 言語 API

(a) コーリングシーケンス

```
BOOL tstat = 8251_tst_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

BOOL	tstat	デバイスがデータを送信している場合	TRUE
		デバイスがデータを送信していない または	

デバイスが初期化されていない場合　　FALSE

(d) エラーコード

なし

(3) 機能

chn_num で示される SIO デバイスに対し以下を実施する。

デバイスがデータを送信している場合は TRUE を返却する。デバイスが未初期化状態またはデータを送信していな場合は FALSE を返却する。

6.4.5 デバイス受信状態提示

(1) 概要

デバイスがデータを受信している否かを返す。

(2) C 言語 API

(a) コーリングシーケンス

```
BOOL rstat = i8251_rst_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(b) パラメータ

BOOL	rstat	デバイスがデータを受信している場合　TRUE デバイスがデータを受信していない または デバイスが初期化されていない場合　FALSE

(d) エラーコード

なし

(3) 機能

デバイスがデータを受信している場合は TRUE を返却する。

デバイスが未初期化状態またはデータを受信していな場合は FALSE を返却する。

6.4.6 デバイス送信割込み状態提示

(1) 概要

デバイスで送信完了割込みが発生しているか否かを返す。

(2) C 言語 API

(a) コーリングシーケンス

BOOL itstat = i8251_itst_sio(ID chn_num)

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

BOOL	itstat	送信完了割込みが発生している場合	TRUE
		送信完了割込みが発生していない または デバイスが初期化されていない場合	FALSE

(d) エラーコード

なし

(3) 機能

デバイスで送信完了割込みが発生している場合は TRUE を返却する。

デバイスが未初期化状態または送信完了割込みが発生していない場合は FALSE を返却する。

6.4.7 デバイス受信割込み状態提示

(1) 概要

デバイスで受信完了割込みが発生しているか否かを返す。

(2) C 言語 API

(a) コーリングシーケンス

BOOL irstat = i8251_irst_sio(ID chn_num)

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

BOOL	irstat	受信完了割込みが発生している場合	TRUE
		受信完了割込みが発生していない または デバイスが初期化されていない場合	FALSE

(d) エラーコード

なし

(3) 機能

デバイスで受信完了割込みが発生している場合は TRUE を返却する。

デバイスが未初期化状態または受信完了割込みが発生していない場合は FALSE を返却する。

6.4.8デバイス受信エラー割込み状態提示

(1) 概要

デバイスで受信エラー割込みが発生しているか否かを返す。

(2) C 言語 API

(a) コーリングシーケンス

```
BOOL irestat = i8251_irest_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

BOOL	irestat	受信エラー割込みが発生している場合	TRUE
		受信エラー割込みが発生していない または デバイスが初期化されていない場合	FALSE

(d) エラーコード

なし

(3) 機能

デバイスで受信エラー割込みが発生している場合は TRUE を返却する。

デバイスが未初期化状態または受信エラー割込みが発生していない場合は FALSE を返却する。

6.5 プリミティブ DIC CBR(Call Back Routine)の機能

以下に PDIC の ISR から実行される各 CBR の処理内容を示す。

6.5.1送信完了割込みコールバック処理

(1) 概要

データ送信完了を通知する。

(2) C 言語 API

(a) コーリングシーケンス

VOID i8251_tcb_sio(ID chn_num)

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

なし

(d) エラーコード

なし

(3) 機能

ＩＳＲから起動され、chn_num で示すチャネル番号の SIO デバイスの送信完了を通知する。

6.5.2受信完了割込みコールバック処理

(1) 概要

データ受信完了を通知する。

(2) C 言語 API

(a) コーリングシーケンス

VOID ｉ8251_ｒcb_sio(ID chn_num,UB chr)

(ｂ)パラメータ

ID	chn_num	チャネル番号
UB	chr	受信したデータ

(ｂ)リターンパラメータ

なし

(ｄ)パラメータ

なし

(3) 機能

ＩＳＲから起動され、受信したデータを通知する。

6.5.3受信エラー割込みコールバック処理

(1) 概要

受信エラーおよびその内容を通知する。

(2) C 言語 API

(a) コーリングシーケンス

VOID i8251_recb_sio(ID chn_num,UB stat)

(b) パラメータ

ID	chn_num	チャネル番号
UB	stat	発生した受信エラー

(c) リターンパラメータ

なし

(d) エラーコード

なし

(3) 機能

ＩＳＲから起動され、受信エラー内容を通知する。

6.6 ジェネラル DIC(GDIC)の１例(ソフトバッファ機能有りの SIO ドライバ)

ＳＩＯのプリミティブ DIC の応用例としてソフトバッファ機能を追加した例を取上げる。データ受信の場合、常にデバイスの状況を監視すると CPU を占有する。更に、データの到来時期を予測できないため、デバイスを常に監視するのは適当でない。これを回避するため、受信したデータを一旦ソフトバッファに貯え、アプリケーションの必要に応じてデータを取出す。データ送信の場合、データ受信と同様にデータ送信完了を常に監視すると CPU を占有する。データ送信完了毎に次ぎに用意された送信したいデータを送信することで CPU の負荷を軽減できる。

6.7 ソフトウェア構成

ＧＤＩＣ内にＦＩＦＯ(Ｆｉｒｓｔ-Ｉｎ Ｆｉｒｓｔ-Ｏｕｔ)形式の送信および受信バッファをもち、アプリケーションには共有サブルーチン形式て提供される。このため複数のタスクが同時にコールする場合に備えてサブルーチン自身の排他制御をセマフォを用いて行う。また、送受信バッファおよびデバイスはＩＳＲ（ＣＢＲ）からもアクセスするので、ＤＳＲ－ＣＢＲ間の排他制御が必要であるが、これはＣＰＵロック機能を用いて行う。

6.8 GDIC の DSR 機能

6.8.1初期化機能

(1) 概要

自身が使用するセマフォやバッファなどの資源を用意する。

デバイスを初期化し通信条件の設定を行う。

(2) C 言語 API

(a) コーリングシーケンス

```
ER ercd = ini_sio(ID chn_num,T_ISIO *pk_sio)
```

(b) パラメータ

ID	chn_num	チャネル番号
T_SIO	*pk_sio	デバイス情報構造体

T_ISIO の構造

初期化する SIO の PDIC のデバイス情報構造体

(c) リターンパラメータ

ER	ercd	エラーコード

(d) エラーコード

（未定）

(3) 機能

自身が使用するセマフォや送受信バッファを用意する。
pk_sio で指示される内容で SIO デバイスを初期化し、受信を有効にする。

6.8.2データ列送信

(1) 概要

指定されたデータ列を送信する。

(2) C 言語 API

(a) コーリングシーケンス

```
ER ercd = snd_sio(ID chn_num,UB *str,UINT size)
```

(b) パラメータ

ID	chn_num	チャネル番号
UB	*chr	送信するデータ列へのポインタ
UINT	size	送信するデータ数

(c) リターンパラメータ

ER	ercd	エラーコード

(d) エラーコード

デバイス未初期化

(3) 機能

デバイスが未初期化状態の場合は、(デバイス未初期化)でエラー終了する。

送信機能使用セマフォを獲得する。

GDIC の送信バッファに送信したデータを転送する。GDIC の送信バッファがフルになった場合は、データ送信中ならば、GDIC の送信バッファが空くのを待つ。データ送信中で無い場合は、PDIC の 1 文字送信機能でデータを 1 つ送信して送信バッファが空くのを待つ。残りのデータ送信は割込みで実施する。

全てのデータを GDIC の送信バッファに転送した場合は送信機能使用セマフォを解放して終了する。

6.8.3データ列受信

(1) 概要

要求したデータ長のデータを受信する。

(2) C 言語 API

(a) コーリングシーケンス

```
ER ercd = rcv_sio(ID chn_num,UB *str,UINT *rsize,UINT size)
```

(b) パラメータ

ID	chn_num	チャネル番号
UB	*chr	受信データ格納域へのポインタ
UINT	*rsize	実際に受信したデータ数
UINT	size	受信したいデータ数

(c) リターンパラメータ

エラーコード

(d) エラーコード

デバイス未初期化

デバイスエラー発生

(3) 機能

デバイスが未初期化状態の場合は、(デバイス未初期化)でエラー終了する。

送信機能使用セマフォを獲得する。

GDIC の受信バッファに受信データがある場合は、ユーザのデータ格納域へ要求されたデータ数分データを転送する。要求されたデータ数に満たない場合は、要求されたデータ数分受信するか、エラーが発生するまで待つ。

待っている場合のデータ受信は割込みハンドラで実施する。

要求したデータ数を受信するか、受信エラーが発生した場合は、受信機能使用セマフォを解放して終了する。

6.9 SIO GDIC の CBR 機能

6.9.1データ送信割込みコールバック

(1) 概要

PDIC の送信 ISR から実行されるコールバック処理。

GDIC の送信バッファにデータが在る場合は次ぎのデータを送信する。

(2) C 言語 API

(a) コーリングシーケンス

```
VOID tcb_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

なし

(d) エラーコード

なし

(3) 機能

GDIC の送信バッファにデータがある場合は、次に用意されたデータを PDIC を経由して送信する。

送信バッファに充分な空きがあり、送信バッファ空きを待つタスクがある場合は、待っているタスクを起床する。

6.9.2データ受信割込みコールバック

(1) 概要

PDIC の受信 ISR から実行されるコールバック処理。

GDIC の受信バッファにデータを格納する。

(2) C 言語 API

(a) コーリングシーケンス

```
INT chr = rcb_sio(ID chn_num)
```

(b) パラメータ

ID	chn_num	チャネル番号

(c) リターンパラメータ

INT chr 受信したデータ

(d) エラーコード

なし

(3) 機能

PDIC からの受信データを以下のバッファに格納する。

待ちタスクが無い場合：GDIC の受信バッファ

待ちタスクが在る場合：待ちタスクの受信バッファ

待ちタスクがあり、要求されたデータ数分受信した場合が待ちタスクを起床する。

6.9.3データ受信エラー割込みコールバック

(1) 概要

PDIC の受信エラー割込み処理から実行されるコールバック処理。

(2) C 言語 API

(a) コーリングシーケンス

```
INT ercd = recb_sio(ID chn_num)
```

(b) パラメータ

ID chn_num チャネル番号

(c) リターンパラメータ

INT ercd 受信エラーコード

(d) エラーコード

なし

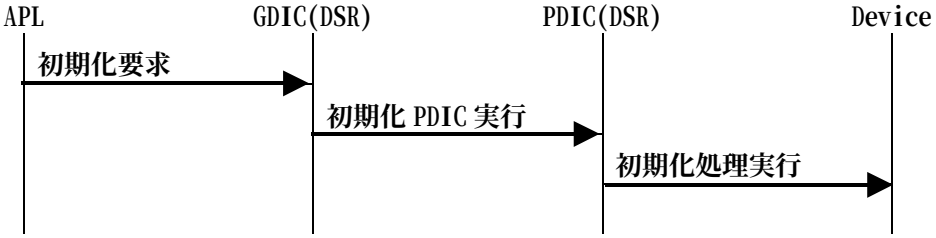
(3) 機能

待ちタスクがある場合は、受信エラー発生として待ちタスクを起床する。

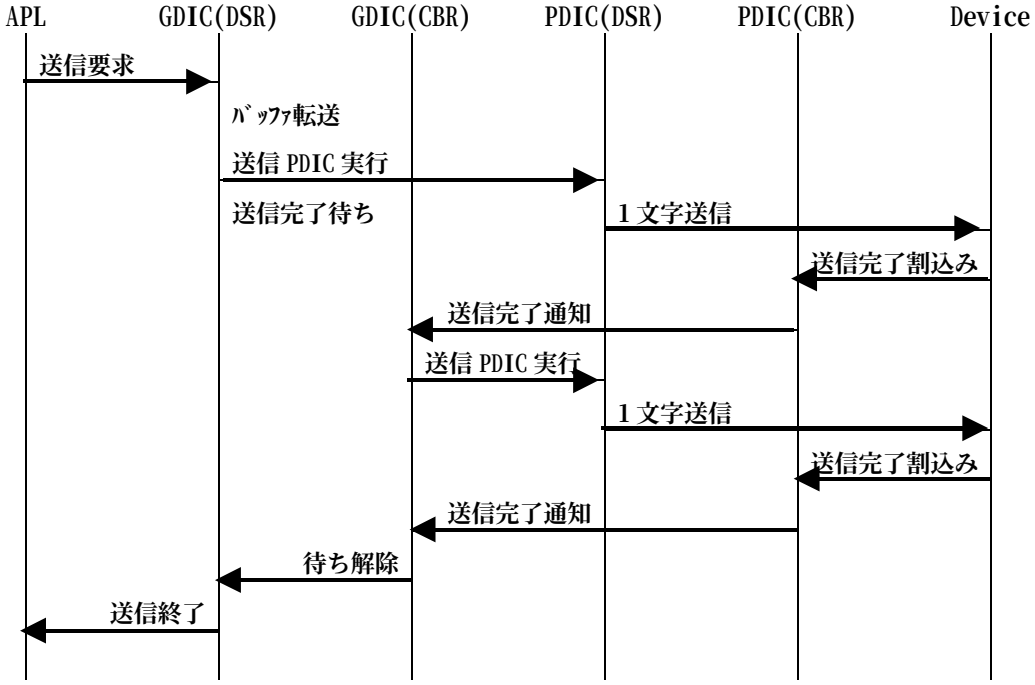
6.10 GDIC の処理の流れ

GDIC の各処理の流れを示す

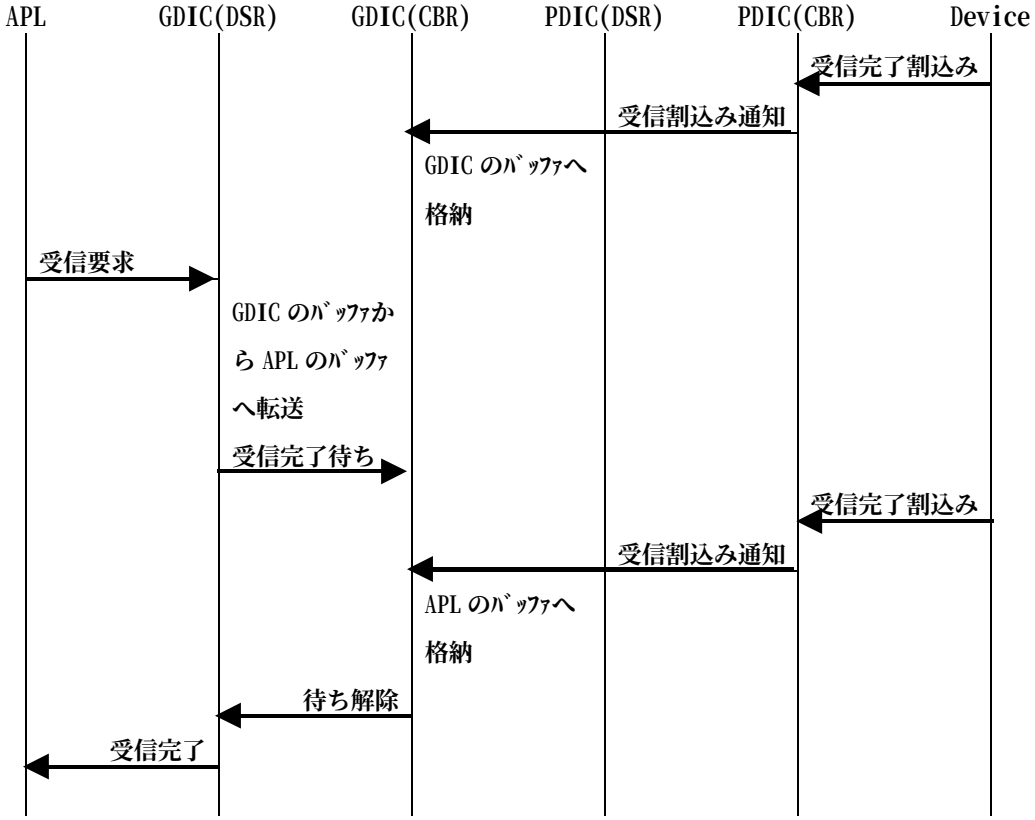
(1) 初期化



(2) データ列送信



(3) データ列受信


APL
GDIC(DSR)
GDIC(CBR)
PDIC(DSR)
PDIC(CBR)
Device
受信完了割込み
受信割込み通知
GDIC のﾊﾞｯﾌｧへ
格納
受信要求
GDIC のﾊﾞｯﾌｧか
ら APL のﾊﾞｯﾌｧ
へ転送
受信完了待ち
受信完了割込み
受信割込み通知
APL のﾊﾞｯﾌｧへ
格納
待ち解除
受信完了

7 今後の課題

本 WG ではこれまで、机上検討を中心にガイドライン作成作業を進めてきた。そのため、実装検討例がほとんどなく、仕様の面でも導入効果の面でも未検討の部分が山積みされている。主な課題を以下に列挙する。

◇ 初期化手順

デバイス毎の初期化や必要なリソースの確保を行う DIC コンポーネントについては、まだ全く検討されていない。また、DIC 自体の初期化やシステムへの登録手順も未定である。どの様な課題があるのかも含めて、実証検証で明らかにしていかなければならない。

◇ GDIC

GDIC は、DIC の顔となる API をアプリケーションやミドルウェアに提供する重要な部品である。また、メーカから提供される GDIC には、そのデバイスの使い方の手本を示すという重要な教育的役割がある。にもかかわらず、その詳細はまだほとんど決まっていないといっても良い。特に API や命名規約などは、早急につめる必要がある。層内で情報をどの様に保持し、共有するかも課題の一つである。また、移植性の評価を行い、有効性を示す事も重要。

◇ API から見たデバイスのカテゴライズ

UNIX ファイルシステムの API は、デバイスドライバの API としても良く使われる。しかし、必ずしも全てのデバイスに向いているわけではない。DIC の API の検討では、どの様なデバイスに UNIX 流が相応しく、どの様なデバイスには別の API が望ましいのかも重要な課題である。

◇ 複合デバイス

デバイスには、複数を組み合わせて一つの機能を実現する物がある。例えば、大量のデータを転送する必要がある多くのデバイスは DMAC と組み合わせて使われることが多い。このような場合の DIC のアーキテクチャも検討課題である。デバイスの PDIC から DMAC の PDIC を呼ぶ方法、GDIC から二つのデバイスの PDIC を順に呼ぶ方法等が考えられるが、どれが良いかの判断は実証検証を待つ必要がある。

◇ 複数の同一デバイス

同一システム上に同じデバイスが複数実装されたり、あるいは一つのチップに同一機能のデバイスが複数実装されている場合に、一つの DIC で全ての同一デバイスを制御できるようにコーディングする事は可能である。しかし、そのためのコーディングガイドラインは未検討。特に、複数に対応する事によるオーバヘッドが問題となる可能性がある。

◇ 複合 IRC

システムによっては、割込み信号の本数を確保するため、IRC を複数組み合わせて繋ぐ場合がある。このような場合、それぞれの IRC の PDIC ISR を組み合わせるだけで良いのか、PDIC ISR に何らかの変更が必要となるのかが現状では明らかになっていない。実例を集め、それに基づいて検討を進めなくてはならない。

◇ デザインパターン

デバイスを制御する手順には、他のデバイス制御手順と類似の処理フローを採るものが少なくない。この様な処理フローに対するコーディング例をデザインパターンとして用意しておけば、DIC 開発者の負担を軽くする事が出来ると考えられる。例えば、どのシステムコールを使うのが望ましいかなどのノウハウを得る事が出来る。デザインパターンの蓄積も、実証検証の際に併せて行っていく事が望ましい。ただし、デザインパターンはアプリケーション毎に蓄積するのが良いか、汎用のものとして蓄積できるのかは不明。

◇ コンテクスト

割込みに応じた処理を割込みコンテクストで実行するのが良いか、タスクコンテクストに情報を渡して、そちらで処理をするのが良いかについては、議論が収束していない。タスクコンテクストで使用される API と割込みコンテクストで使用される API を名前空間から分けるべきかどうかについても同様。

◇ DIC の流通 / データベース化

DIC を部品として蓄積し、流通を図るためには、DIC の書式以外にも様々な面での環境整備が必要となるが、現在はまったくの白紙状態である。

これらを踏まえ、今後は実証を中心とした検討に入る予定である。検証にあたっては、多くの方のご意見・ご協力をお願いしたい。

略語集

API	Application Program Interface	1.1節, 3.5節
CBR	CallBack Routine	2.3節
DIC	Device Interface Component	1.1節, 3.11節
GDIC	General DIC	3.1節 (図 3-1)
DMAC	Direct memory Access Controller	7章
DSR	Device Service Routine	2.3節
EOI	End Of Interrupt	3.6節
GDIC	General DIC	1.3節
IRC	Interrupt Request Controller	3.1節, 7章
ISR	Interrupt Service Routine	3.1節
PDIC	Primitive DIC	1.1節, 3.1節
SIO	Serial Input/Output	5.4節

用語集

アクセスインタフェース	3.1節
移植者	3.1節
エッジトリガ	3.4節
サービス中フラグ	3.6節
システムインタフェースレイヤー	3.1節
システム提供関数群	3.1節
レベル	3.4節
ワイヤードOR	3.4節
割込みシステム	3.1節
割込みハンドラ	3.3節
割込み入力ライン	3.4節
割込み要求ライン	3.4節

― **メモ** ―