デバイスドライバの
生産性・移植性の向上に向けて
ーデバイスドライバ設計ガイドラインからー

三菱電機マイコン機器ソフトウエア株式会社
技術企画センター
／デバイスドライバ設計ガイドラインWG幹事
宿口　雅弘
ms89019@mms.co.jp
http://www.mms.co.jp

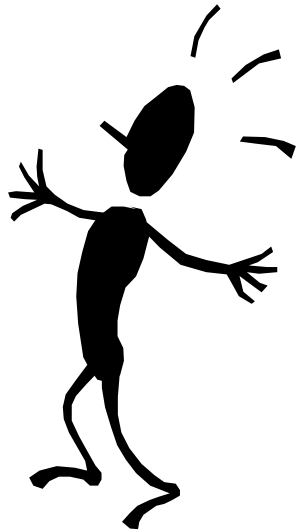
http://www.itron.gr.jp/GUIDE/device-j.html

- デバイスドライバとは？

周辺デバイス (ハードウェア)を
ドライブ(制御)するプログラム
(比較的小規模のプログラム)

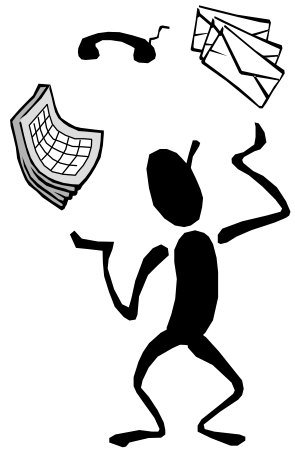
のハズだが
その定義範囲が広すぎる

- 組込みシステムの周辺デバイスに着目
  - 共通にもつデバイスはCPUだけ
  - システム毎に制御したデバイスが異なる
  - 周辺デバイスを複数のタスクで共有したい例は少ない
  - 制御にタイムクリティカルなものが多い
    - アセンブリ言語を必要とする
    - 高度なコーディングを必要とする
  - 周辺デバイスを制御するのが主目的



- 組込みシステムのデバイスドライバって？
  (汎用システムと比較して)
  - OSが採用されはじめのたのは近年
    - OS組込み型デバイスドライバの概念が無い
  - ハードウェア（デバイス）の制御が主
    - 組込みプログラム　デバイスドライバ であろう
  - システム自体が少量多品種
    - あるデバイス制御専用システムの場合もある

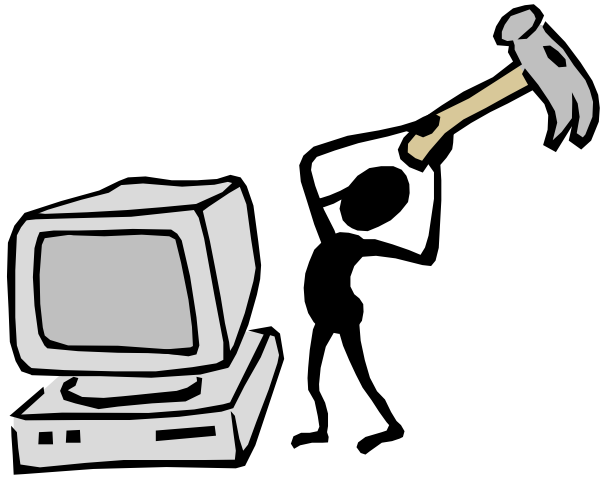
    → 移植性の要求は低かった。



- 汎用システムと組込みシステムの構成
  - 汎用システムのハードウェア構成

    CPU＋コンソール＋ハードディスク＋ネットワーク
    - 構成は概ね固定されている
      - » WindowsOSはPC専用OS
      - » LinuxもPC専用UNIX（だった）
    - OSはハードウェアの違いを隠蔽する役割をもつ
    - 不特定多数のプロセスがデバイスを共有する

<u>OSのデバイスドライバサポートは必須</u>



- 汎用システムと組込みシステムの構成
  - 組込みシステムのハードウェア構成

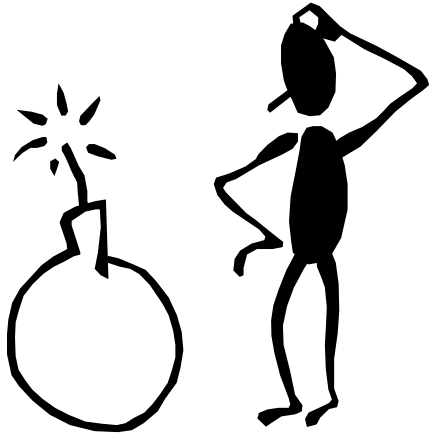
    CPU＋。。。etc. etc

    ！決まった形が無い

    (製品分野毎にはある程度決まっているが)
    - タスク制御(ＣPUの効率的活用)がＯＳ主目的
    - ハードウェアの隠蔽は必須でない

ＯＳがデバイスドライバをサポートする必要無し

しかし近年の特徴として。。。

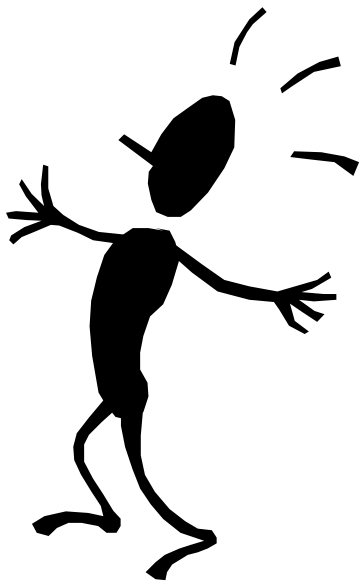


- 近年の組込みシステムの状況
  - 組込みシステムの機能の複合化
    - 特に通信機能(ネットワーク)が不可欠
  - 汎用機の機能が取込まれている
    - ネットワーク機能はPCの世界で発達

    → PCベースの考え方
  - 組込みシステムの複雑化
  - 製品の短命化
  - 開発期間の短縮
  - 開発コストの圧縮
  - 品質の向上

- 近年の組込みシステムの現況
  - 開発期間の短縮
  - 開発コストの圧縮
    - 出来る限りモノを作らない方が良い
  - 品質の向上
    - 設計、評価時間を充分にとる

    →現実問題　難しい



- 近年の組込みシステムの状況

結論:

・使えるプログラムは流用する。

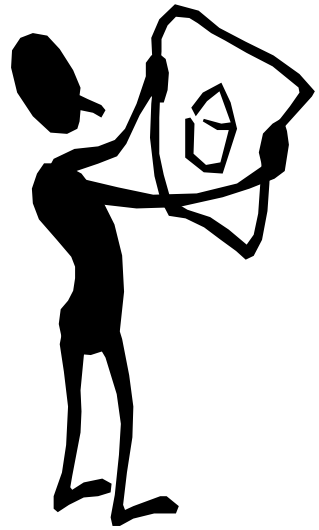
例: プロトコルスタック

・移植性の高いプログラムを作成する。

周辺デバイスが変るケースは少ない。

同じデバイスならば制御プログラムは流用できる。

→ 移植性重視の設計・製作が必須である。

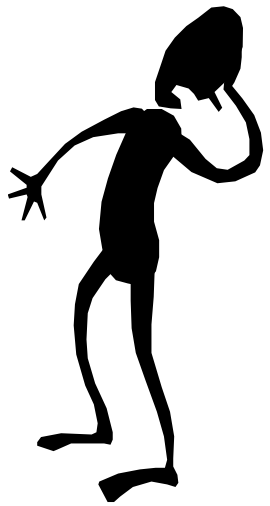


- デバイスドライバの製作が難しい理由
  - デバイス制御処理
  - タイムクリティカル処理
  - 割込み処理
  - システムの主役でない
    アプリケーションに依存
  - OS組込み型の場合
  - OS組込み型でない場合

- デバイスドライバの製作が難しい理由
  - デバイス制御処理
    - デバイスの仕様の理解が必須
    - ハードウェアの知識が必須

！デバイスはCPUと非同期に動作している
→タイミングの問題等一筋縄ではいかない

→学習する以外に道は無い



- デバイスドライバの製作が難しい理由
  - タイムクリティカル処理
    - システムとしてリアルタイム処理が要求される。
    - デバイス制御に時間制約がある。

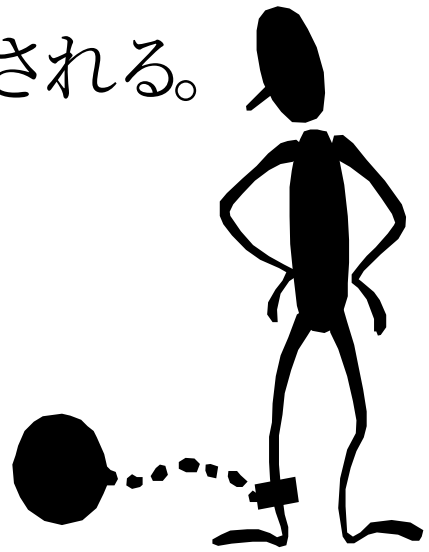
      例 SIOのデータ受信

      一定時間内にデータを受信(プログラムでCPUのメモリに取込む) しなければデータが失われる。

→ 高度なコード最適化技術が要求される。

→ CPUの動作の理解が必須。

→ コンパイラの特性の理解が必須。



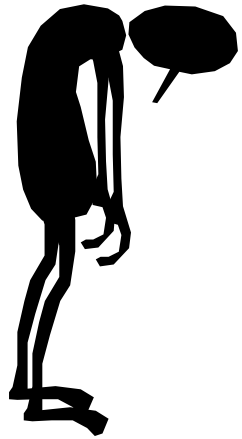
- デバイスドライバの製作が難しい理由
  - 割込み処理
    - 割込みはCPU外のイベントを知る効率的な手段。
    - デバイスからの応答を知るために有効。
    - 割込みでない場合はポーリング処理。
      →システム負荷が上昇する場合あり。
    - しかし、様々な問題が。。。
      - 割込み処理自体がタイムクリティカルな処理である。
        → 割込みオーバヘッドがどれくらいか？
      - 多重割込みへの対応が必要な場合がある。
      - 非割込み部との排他処理が必須である。

- デバイスドライバの製作が難しい理由
  - システムの主役でない
    ！アプリケーション機能に依存

  → 他のプログラムへの影響を考慮する。

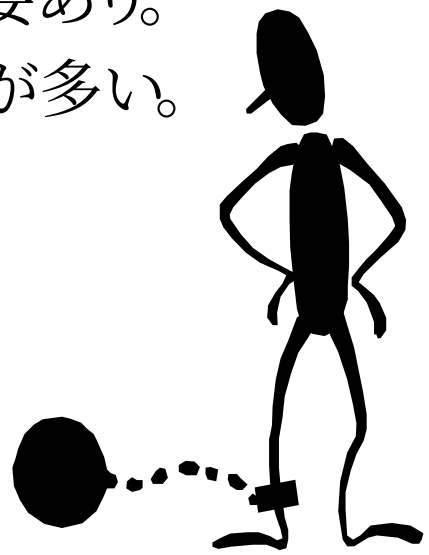
  → 製品機能を意識すると汎用的に書けない。

  ※ 矛盾、ジレンマ



- デバイスドライバの製作が難しい理由
  - OS組込み型の場合
    - → デバイス制御のシステムコールを持つＯＳ。
    - → 他のシステムコールと同様に動作する。
    - → 多くの場合サブルーチンとして提供される。

    - OSの動きを詳細に知る必要あり。
    - OSの決めたルールを詳細に知る必要あり。
    - OSの一部であるがゆえの注意事項が多い。



- デバイスドライバの製作が難しい理由
  - OS組込み型でない場合

    → デバイス制御のシステムコールを持たないOS。

    μITRONなど　(以前はあったが使われていなかった)

    → OS制御下のタスクとして動作する。

    • 全くデバイスドライバの決まり毎が無い。

    → 手も足も出ない！

    → そこで

    「デバイスドライバ設計ガイドライン」

- デバイスドライバ設計ガイドラインの動機
  - ソフトウェア開発者
    - デバイスドライバ→規模が小さいが開発困難
    - 再利用性・流通性を高めたい
  - デバイス提供者
    - デバイスの付加価値を高めたい
    - デバイスとともにドライバを提供したい
  - 技術者教育面
    - デバイスドライバ設計の教科書が無い
    - 用語を統一したい　→　技術者間の意志の疎通
  - ITRONプロジェクト
    - デバイスドライバを揃えたい

- 生産性向上の考え方
  - ガイドラインにより生産性が向上するか？
    - ガイドライン→パターンを多く見せる
      →型（アーキテクチャ）を多く見せることができる
      →ドライバの勘所が解る
      →生産性が向上する
  - ガイドラインにより移植性が向上するか？
    - 共通事項（標準的規格）の提示
    - パターンに従った設計

    →移植性の向上（→生産性の向上）

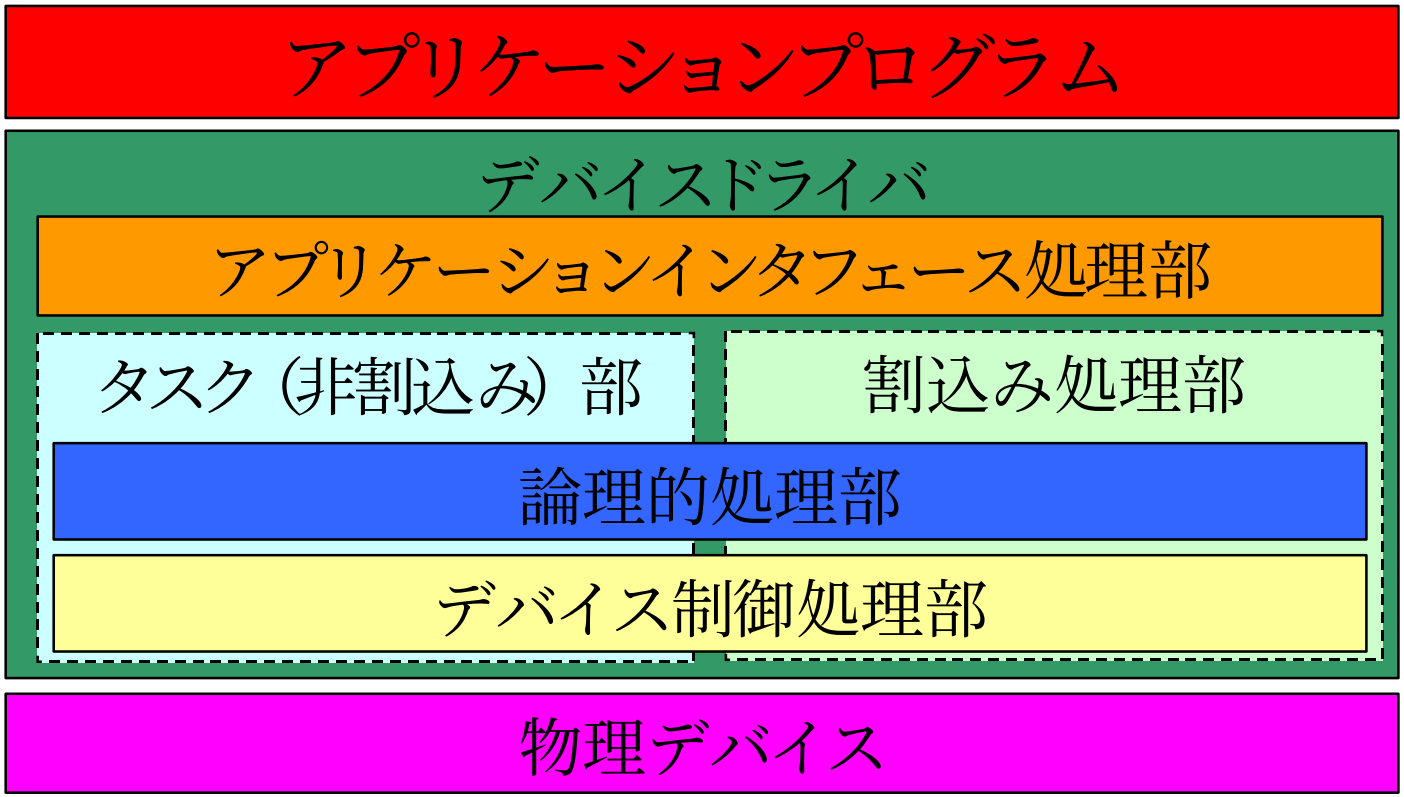
    - パターンの理解による移植対象の早期理解

- 移植性を損ねる要因は？
  - CPUに依存するもの
    - 割込みアーキテクチャ
    - ポートアクセス方式
    - エンディアン問題
  - プラットホームに依存するもの
    - プログラムの動作
      - マルチタスク or非マルチタスク
    - 割込み禁止方式

どのように対応すべきか？

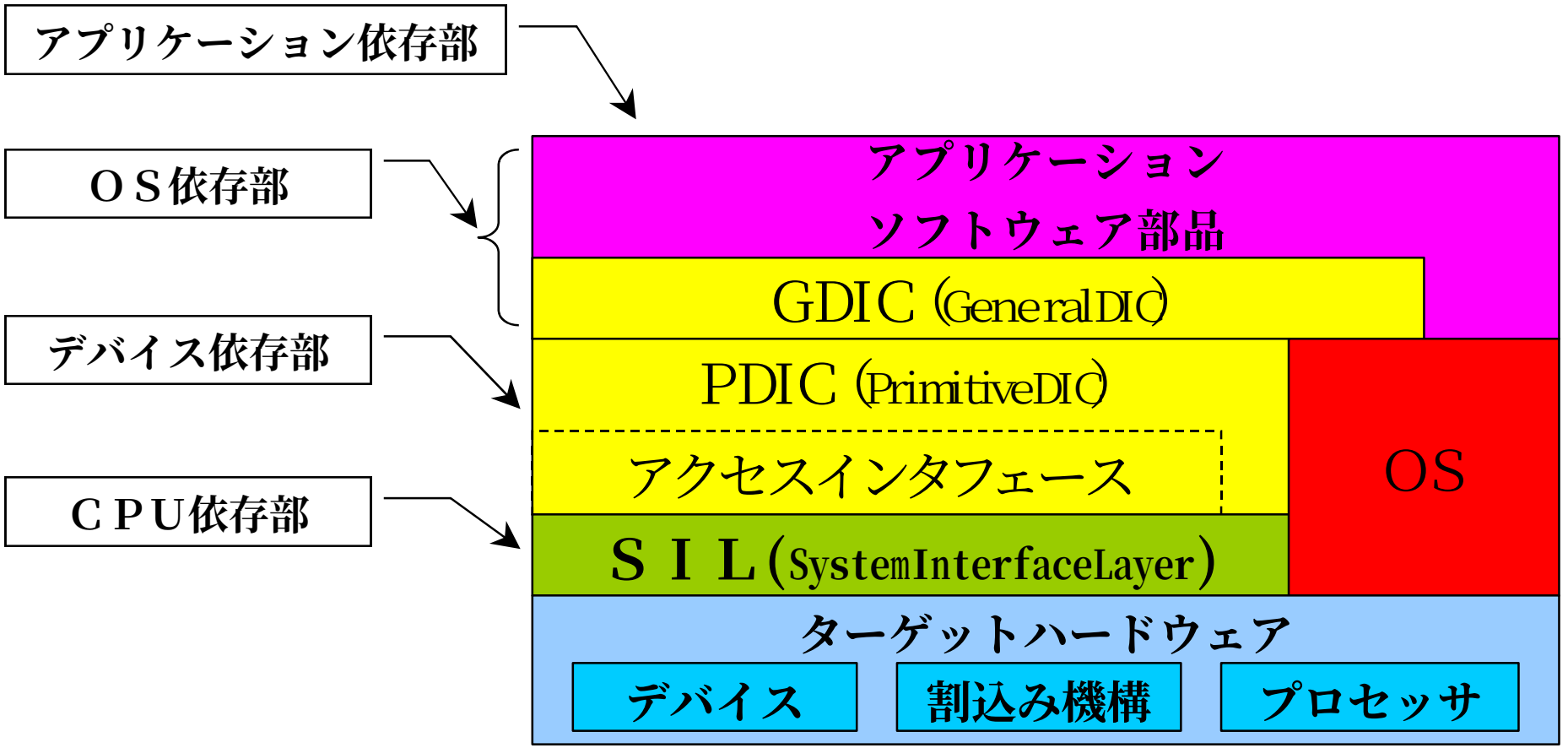
- 階層化構造の導入
  - デバイスドライバの構造を見る

  ↓　デバイスドライバの内部を構造的に分析

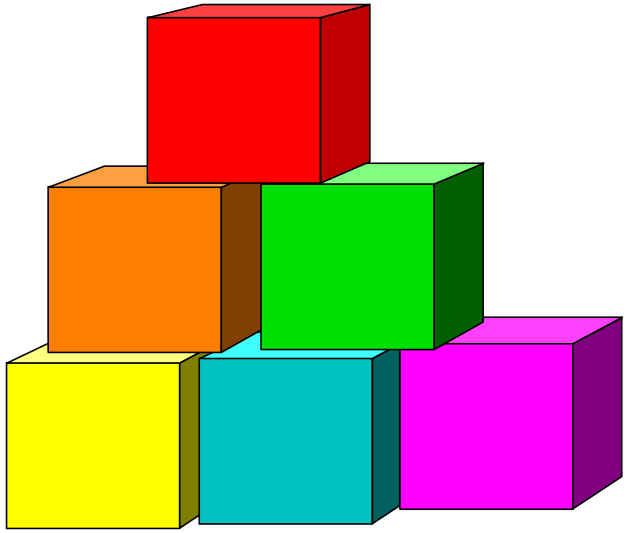


- 階層化構造の導入
  - デバイスドライバの構造を見る
    - アプリケーションインタフェース部
      - アプリケーションプログラムとインタフェースをとる。
    - タスク（非割込み）処理部
      - タスク処理を実行する。
    - 割込み処理部
      - 割込み処理を実行する。
    - 論理的処理部
      - デバイス制御以外の論理的処理を実行する。
    - デバイス処理部
      - デバイス制御処理を実行する。
    - CPU依存部
    - OS依存部

- デバイスドライバ設計ガイドラインのアプローチ
  - DIC(Device Interface Component)アーキテクチャ



- 階層化構造導入のメリット
  - 各依存性を分離できる
    - デバイス依存性
    - CPU依存性
    - OS依存性
    - アプリケーション依存性



- 階層化構造導入のメリット
  - デバイス依存性の分離
    - 何を制御するのか明確にする／できる。
    - デバイスの制御に専念できる。
    - 単純な制御文に集約できる。

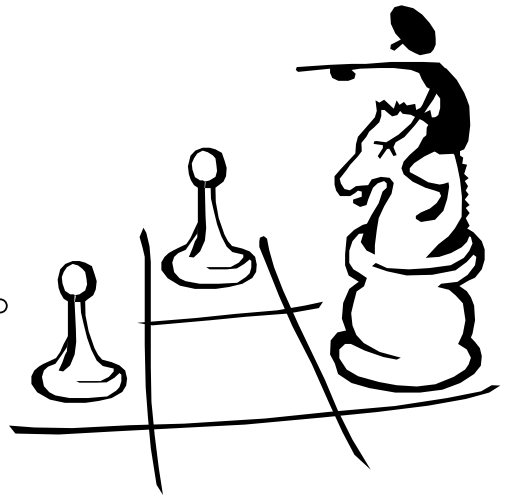
→

- デバイスの制御は基本的には制御レジスタ（ポート）へのアクセスのみ。
- ※周辺ソフトウェアとの相互関係が処理を難しくしている。

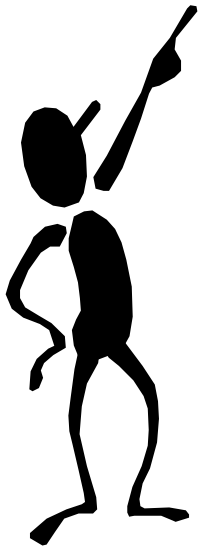
- 階層化構造導入のメリット
  - CPU依存性の分離
    - 割込み処理
      - 割込みアーキテクチャの吸収
      - デバイスドライバの割込みハンドラはデバイスの制御に専念すべき

      → CPUの割込み処理、割込みコントローラの処理はそれぞれのルーチンにて実施すべき。
    - 割込み禁止／解除処理
      - 処理関数の導入
    - エンディアン変換
      - エンディアン変換関数の導入

        →wordアクセス時に制約を持たせる。



- 階層化構造導入のメリット
  - **ＲＴＯＳ依存性の分離**
    - OS非使用レイヤ
      - PDIC
      - OS 存性の排除
      - OS の機能を使わない
        待ちに入れない (非同期処理)
      - OS 存性の排除
    - OS 使用レイヤ
      - GDIC
      - OS の機能を使う
        待ちに入れる(同期処理)



- 階層化構造導入のメリット
  - アプリケーション依存性の分離
    - アプリケーション特化機能を明確にする。
    - 再利用性が高いデバイス部を切り出す。

★デバイスドライバはCPUからみた機能分割された後の概念

例：DARTS(Design Approach for RealTimeSystem)法

I/0タスク構造化基準(I/0デバイスに着目)

以下それぞれを１つのタスクとする

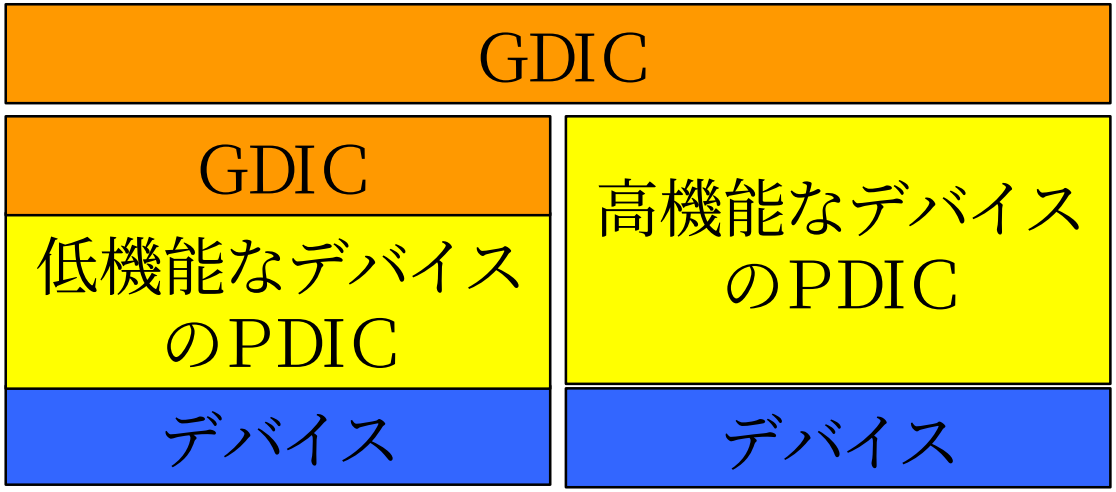
- **非周期的なI/0デバイスのドライバ**
- **周期的なI/0デバイスのドライバ**
- **複数からイベントを受信するI/0デバイスドライバ**

- デバイスドライバ設計ガイドラインのアプローチ
  - DIC(Device Interface Component)アーキテクチャ

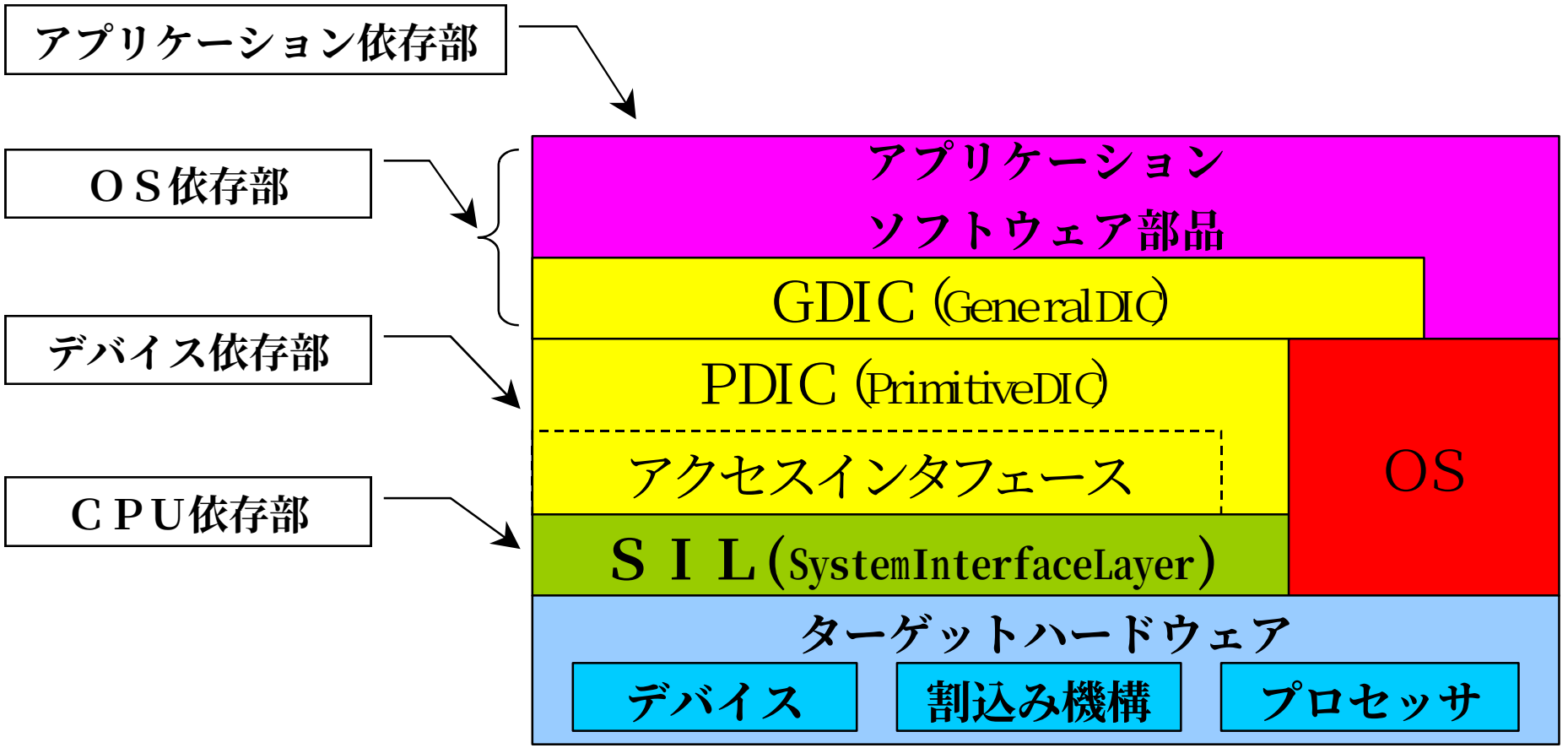
  階層構造の導入
    - P(Primitve) DIC

    →デバイスに対する最低限のインタフェースのみ提供
    - G(General) DIC

    →PDICに機能を追加する



- デバイスドライバ設計ガイドラインのアプローチ
  - DIC(Device Interface Component)アーキテクチャ

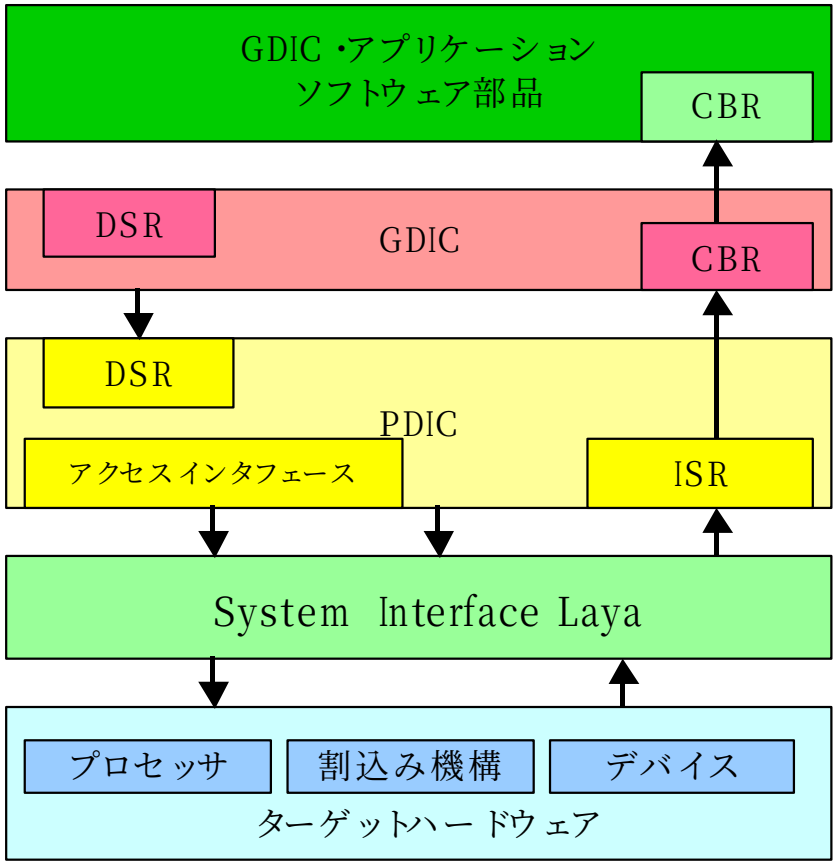


• デバイスドライバ設計ガイドラインのアプローチ
– DIC(Device Interface Component)アーキテクチャ

DSR：
(Device Service Routine)
PDICと上位層とのインタフェースを司る

アクセスインタフェース：
メモリ空間かIO空間かなどのデバイスへのアクセス方法を抽象化する

システムインタフェースレイヤ(SIL)：
プロセッサ(I/O命令、円ディアン、割込み禁止／許可命令)と割込みコントローラを抽象化する

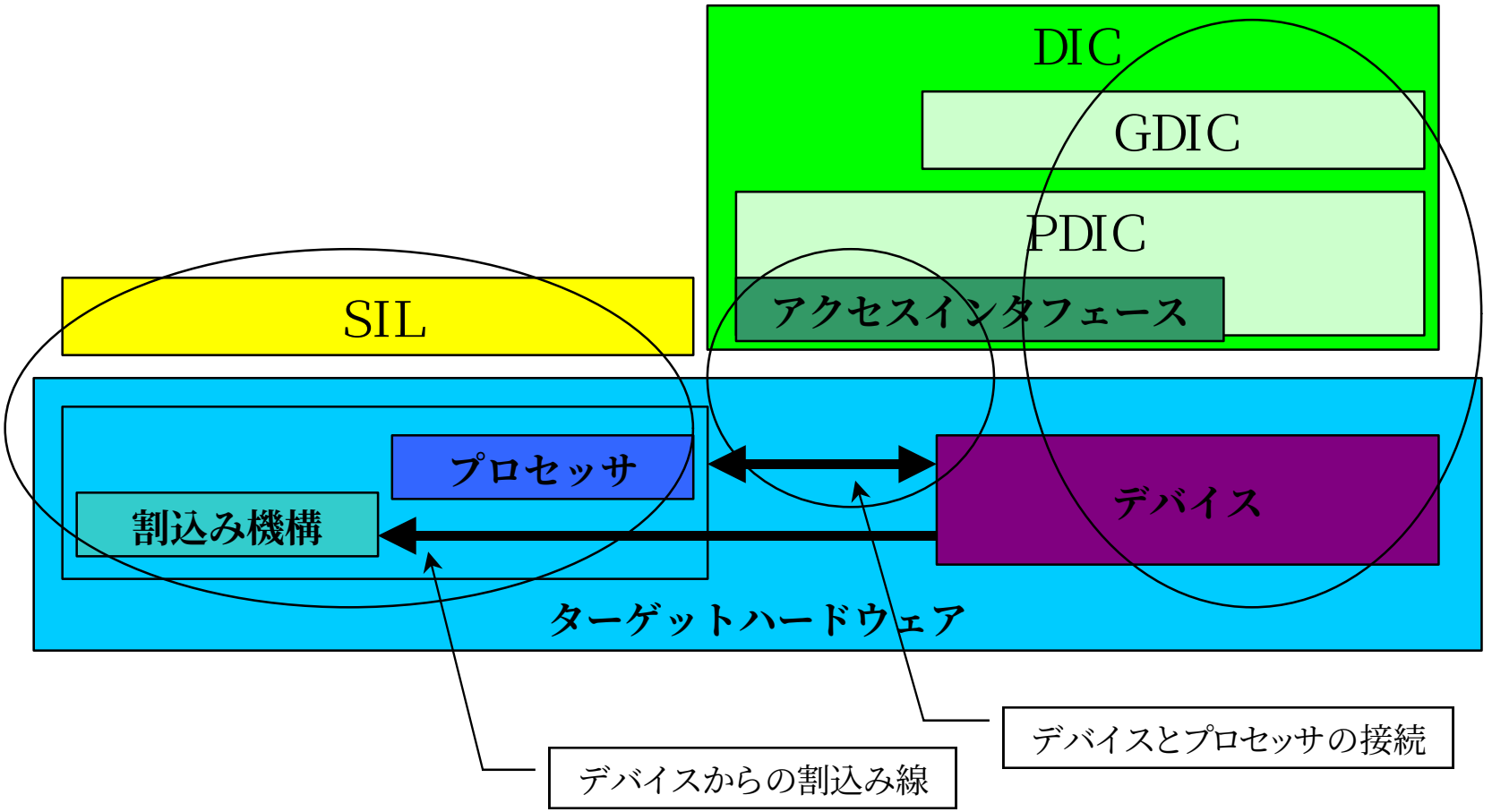


CBR:
(Call Back Routine)
割込み機構から処理要求完了通知または事象通知で呼ばれるルーチン

ISR:
Intrrupts Service Rutine：
デバイスの割込み処理の本体

- デバイスドライバ設計ガイドラインのアプローチ
  - DIC(Device Interface Component)アーキテクチャ

    各レイヤのターゲットハードウェアへの対応



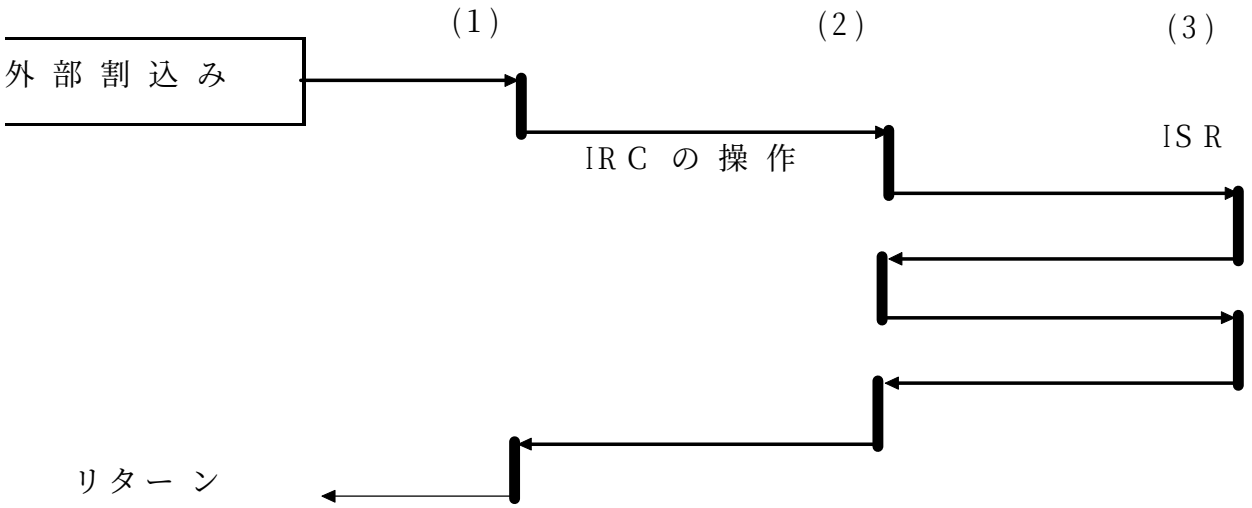
- デバイスドライバ設計ガイドラインのアプローチ
  - μITRON4.0の割込みシステムモデル

※デバイスWGの議論の成果

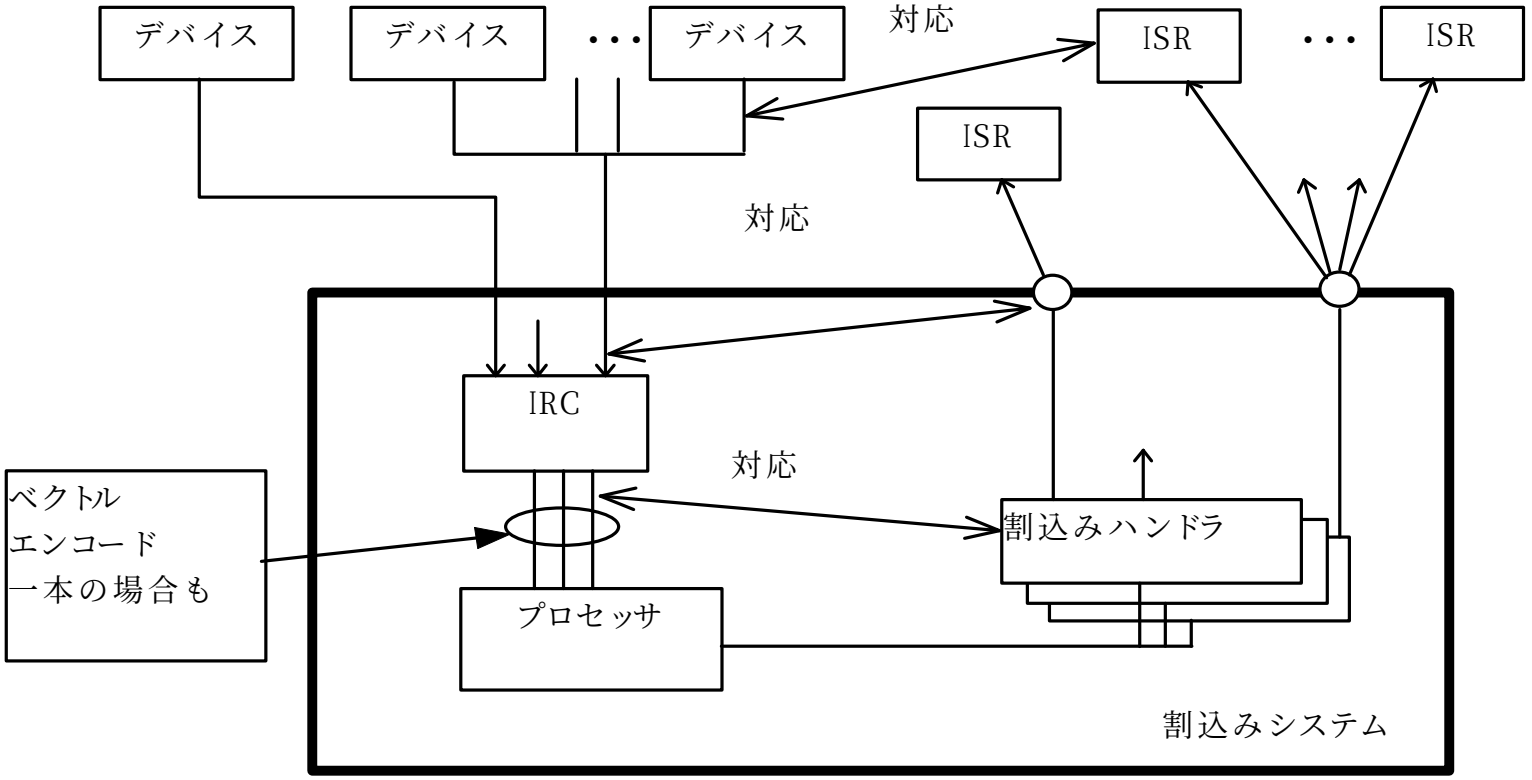
(1) CPUの割込みアーキテクチャに依存する処理

(2) 割込みコントローラなどのCPU以外のハードウェアに依存する処理

(3) デバイスに依存する処理



- デバイスドライバ設計ガイドラインのアプローチ
  - μITRON4.0の割込みシステムモデル
    参照した割込みハードウェア構成



•システムインターフェースレイヤー

– ターゲットハードウェアを抽象化し、
各PDIC間で共通に利用できるレイヤー

- デバイスとのデータ入出力
（アドレス空間、データサイズ、エンディアン等）
- 微小時間の遅延
- 割込みロック状態の制御
- 割込みサービスルーチンの管理

•デバイスとのデータ入出力API

– I／O空間とメモリ空間の違いの吸収

- ８ビットのデータを読む場合

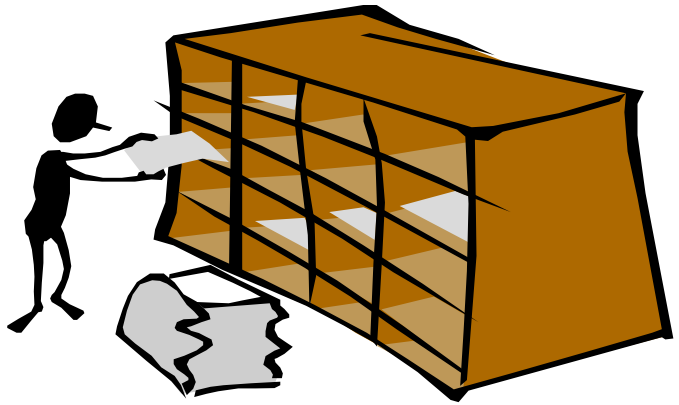
  I／O空間から　　VB data = sil_reb_iop(VP iop);

  メモリ空間から　　VB data = sil_reb_mem(VP mem);

- ８ビットのデータを書く場合

  I／O空間へ　　void sil_wrb_iop(VP iop, VB data);

  メモリ空間へ　　void sil_wrb_mem(VP mem, VB data);



•デバイスとのデータ入出力API

– データサイズの違いの吸収

- メモリ空間上の指定したアドレスからデータを読み取る。

8 ビットサイズ	VB data = sil_reb_mem(VP mem);
16ビットサイズ	VH data = sil_reh_mem(VP mem);
32ビットサイズ	VW data = sil_rew_mem(VP mem);

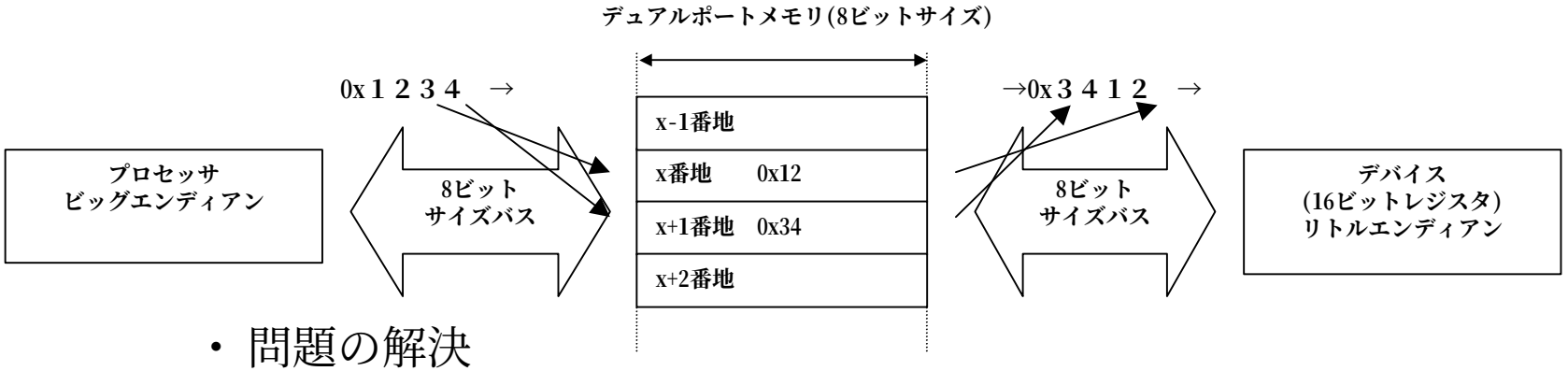
- メモリ空間上の指定したアドレスへデータを書き込む。

8 ビットサイズ	void sil_wrb_mem(VP mem,VB data);
16ビットサイズ	void sil_wrh_mem(VP mem,VH data);
32ビットサイズ	void sil_wrw_mem(VP mem,VW data);

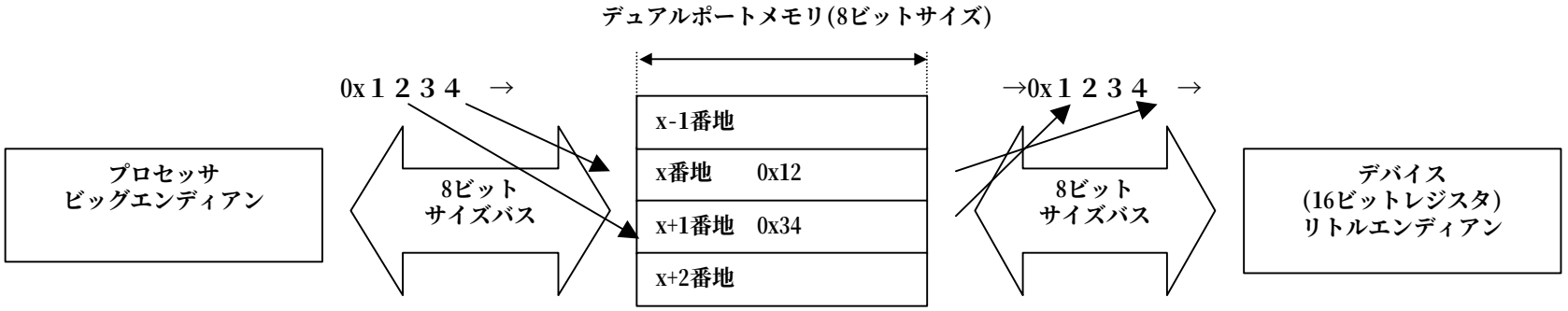
•デバイスとのデータ入出力API

– エンディアンの違いの吸収

・ エンディアンの違いによる問題



・ 問題の解決



•デバイスとのデータ入出力API

– メモリ空間デバイスへのデータ入出力

• メモリ空間上の指定したアドレスから16ビットのデータを読み取る。

自然なエンディアン	VH data = sil_reh_mem(VP mem);
リトルエンディアン	VH data = sil_reh_lem(VP mem);
ビッグエンディアン	VH data = sil_reh_bem(VP mem);

• メモリ空間上の指定したアドレスへ16 ビットのデータを出力する。

自然なエンディアン	void sil_wrh_mem(VP mem, VH data);
リトルエンディアン	void sil_wrh_lem(VP mem, VH data);
ビッグエンディアン	void sil_wrh_bem(VP mem, VH data);

自然なエンディアン→エンディアンの違いを考慮する必要の無い状況

•エンディアンの判別

– プロセッサーのエンディアン判別用定数
  - 定数 SIL_ENDIAN をプロセッサーのエンディアンに合わせて定義
  - デバイスのエンディアンが設定できる場合はDIC内でのデバイス初期化時にこの定数を用いてエンディアン変換なしで入出力できるように初期化することができる。

```
#define SIL_ENDIAN_LITTLE        0  リトルエンディアン
#define SIL_ENDIAN_BIG           1  ビッグエンディアン
```

ビッグエンディアンを定義する場合

```
#define SIL_ENDIAN    SIL_ENDIAN_BIG
```

•微小時間の遅延API

– デバイスによっては一定の微小時間（OSのタイマ割込み間隔よりもはるかに短い時間）デバイスを操作できない場合がある。
→CPUの動作クロックが様々なので、毎移植毎に調整が必要となる。

– 微少時間のウェイト機能を用意する。

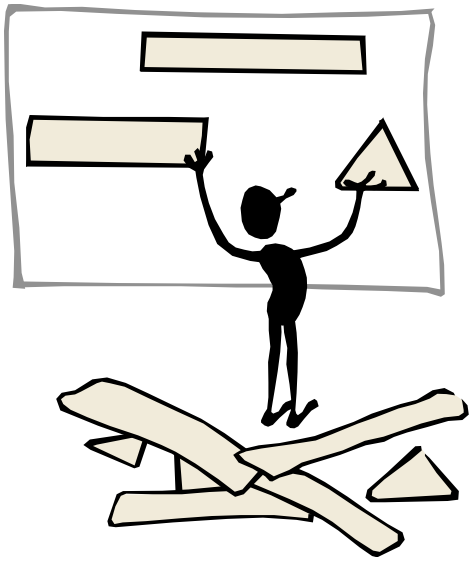
微小時間のウェイト
void sil_dly_nse(UINT dlytim)
UINT dlytim ウェイトする時間[nsec]

・タイマーまたはビジーループで実装する
・RTOS上ではこのAPI内で自ら待ち状態に入らない

•割り込みロック状態の制御API

- CPUロック状態
  - ITRONの管理下にある割込みのみ禁止
- 割り込みロック状態
  - NMIを除く全割り込みの禁止する。
    DIC内で割込みロック制御を使用する場合、DICの呼び出し前後で状態を変えないための前処理が必要。

void SIL_PRE_LOC()	割り込みロック前処理
void SIL_LOC_INT()	割り込みロック
void SIL_UNL_INT()	割り込みロック解除

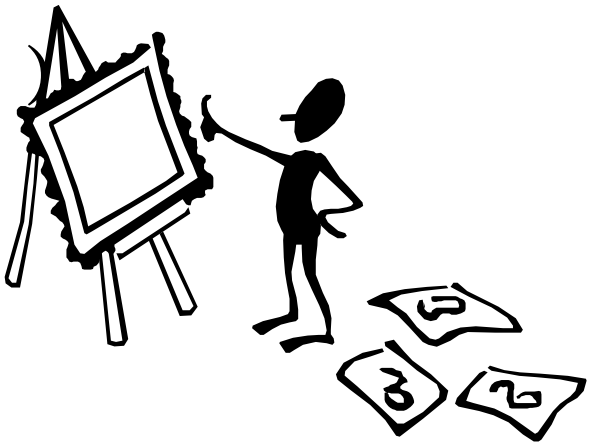


- 実装例

  以下のサンプルプログラムを作成

  - ＳＩＯ
    - » μPD72001用
  - フラッシュメモリ
    - » 26F016SA用

  共にWGのWebにて公開予定。
  （遅れています。済みません。）

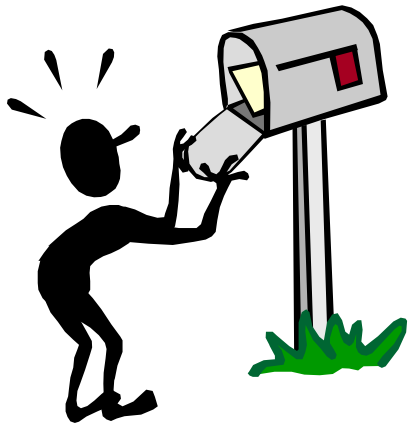


協力：

豊橋技術科学大学
組込みリアルタイムシステム研究室

- 今後の予定
  - サンプルプログラムの追加
  - GDICアーキテクチャの深堀
    - DIC（タスク）のデザインパターン？
    - DICの構成方法
    - etc. etc.

引続き、ご意見募集しています。



御静聴ありがとうございました

- デバイスドライバ設計ガイドライン第１版
  は以下のURLで間もなく公開です。
  (公開が遅れています。済みません)
  http://www.itron.gr.jp/GUIDE/device-j.html

- WGへのご意見は以下までお願いします。
  device-wg@itron.gr.jp