太陽光発電と制御技術

スマートコミュニティ技術講座III

(株) 協栄エレクトロニクス

目次

1. 太陽光発電の概要
   I. 太陽光発電の必要性
   II. 太陽光発電の課題
   III. 関係する技術
2. 太陽光発電とマイコン(組み込みシステム)
   I. 制御の必要性
   II. マイコン/組み込みシステムとは
   III. マイコンにおけるプログラミング
3. マイコン基礎実習I
4. マルチタスク・OSについて
5. マイコン基礎実習II

太陽光発電の概要

太陽光発電の必要性

- 再生可能エネルギーの一形態
  - 太陽光を用いるため排出物がない
  - 二酸化炭素を排出しない
- エネルギー源は太陽
  - どこでも発電が可能
  - 既存インフラから離れていても日光さえ当たれば良い
- 家庭から売電が可能
  - 家電を動かし、余った分は電力会社へ

太陽光発電の必要性

- 東日本大震災
  - 福島第一原子力発電所事故
  - 日本各地の原子力発電所の稼働停止

電力不足

- ピーク時(日中)の代替電源
- 電力買い取り制度

太陽光発電の課題

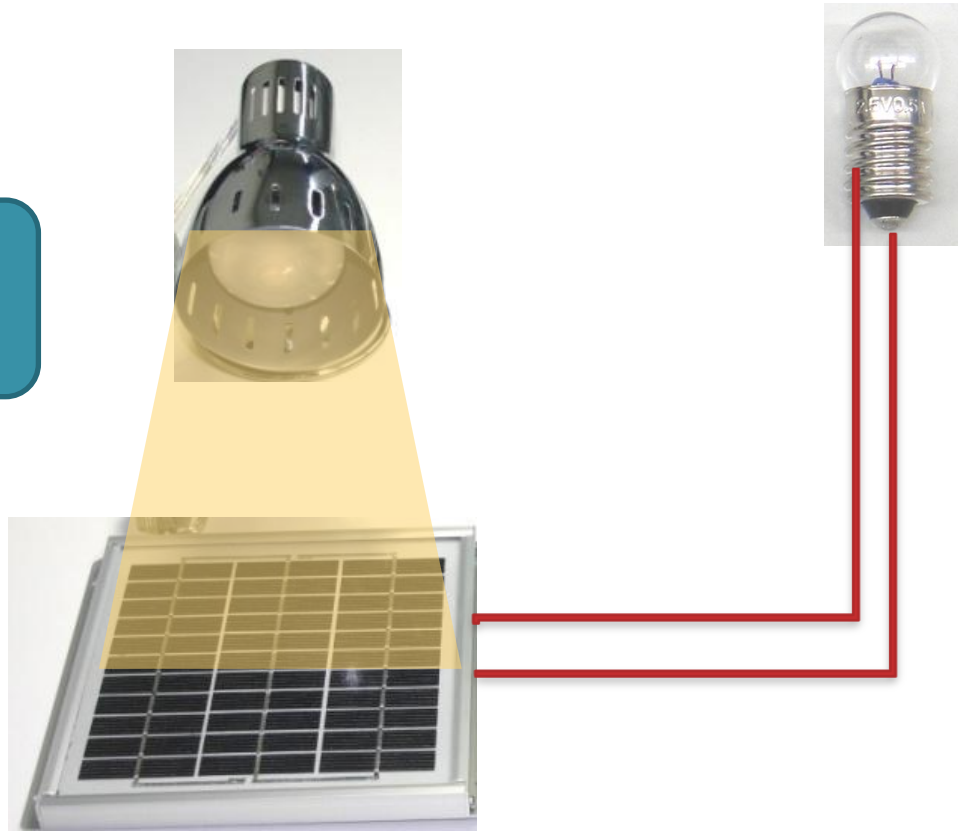
- 太陽電池は直流電源
  - 家電は交流のため**インバータ**等で交流に
- 電流・電圧の関係が変則的
  - **最大電力点追従制御**が必要
- 直射日光が当たらないと発電できない
  - 曇り・雨・夜間は発電できない
  - 補完する装置(**二次電池**等)が必要

- 温度が上がりすぎてもいけない
- 熱放射(太陽光)以外の光では発電しない
  - 蛍光灯やLED電灯の光では発電しない

太陽電池の課題

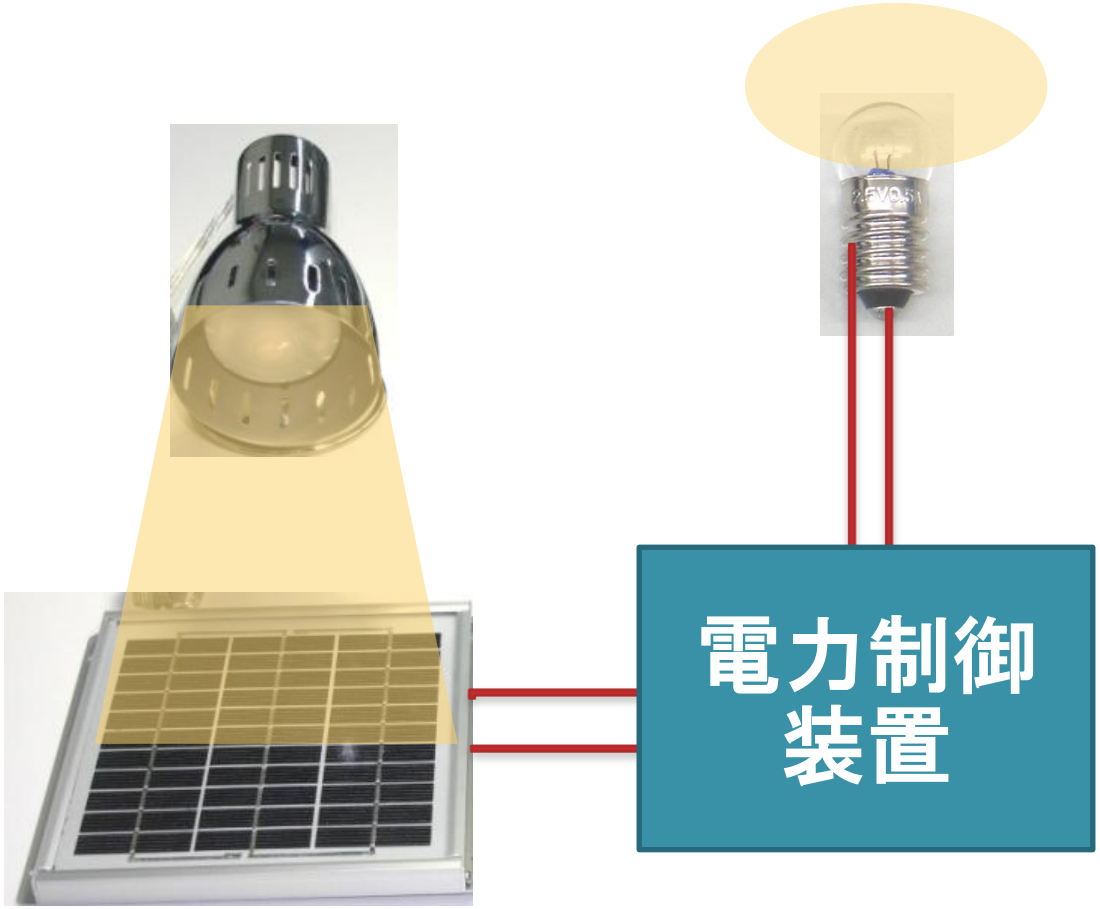
- 具体例として…
- 太陽電池と豆電球を繋いでも光らない!

白熱灯の
光では



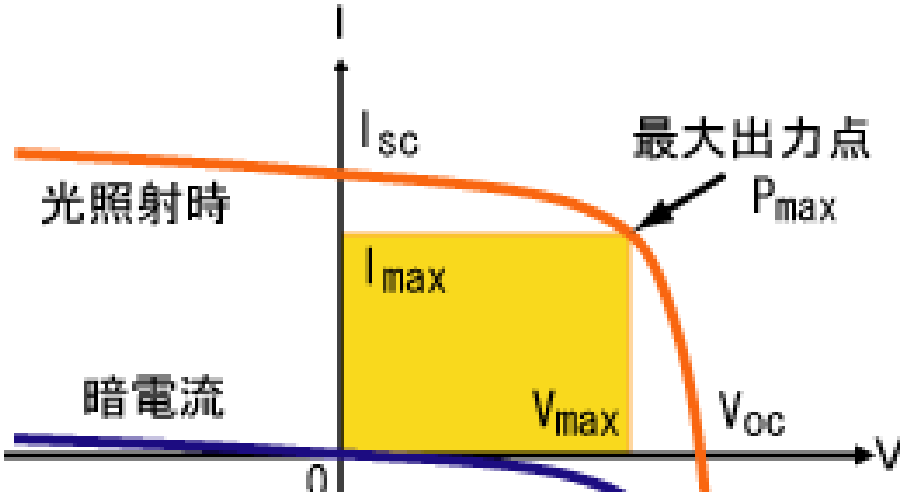
太陽電池の課題

- 電力制御装置を使うと光る!



太陽電池の課題

- なぜ光らないか?
  - 太陽電池の電圧-電力特性



- 電力が最大になるように制御しないと最大電力は使えない

関係する技術

- **最大電力点追従制御**
  - MPPT(Maximum Power Point Tracking)**制御**
  - **太陽電池が最大の電力を生じる点を追いかける制御**
  - **電圧を絞ると電流が増加する**

- **電圧を絞るために、PWM制御**
  (Pulse Width Moderation)

- **具体的には2・3日目に…**

関係する技術

- PWM制御
  - パルス幅を変えることで、作りたい平均電圧を作る
- インバータ（今回は除外）
  - 正負両方のPWM制御回路を備えることで、直流から交流を作る
  - 位相の同期が最大の課題

太陽光発電と
組み込みシステム

太陽光発電と組み込み

- 太陽光発電
  - スケーラブルな直流発電
  - さまざまな制御が必要
- 電力を制御する仕組みが必要
- ハードウェアのみでの制御では不完全
- 計測・演算・制御の流れ
- マイコン制御・組み込み機器

なぜ組み込み?

- トランスやダイオードのようなハードウェアだけでは制御できない

演算能力が必要

- PCでもいいのでは?
  - 不可能ではないが、分散化のため、いかなる機器にもPCを入れなくてはいけない
- 機器が「スマート」に動く
- マイコン・組み込み機器

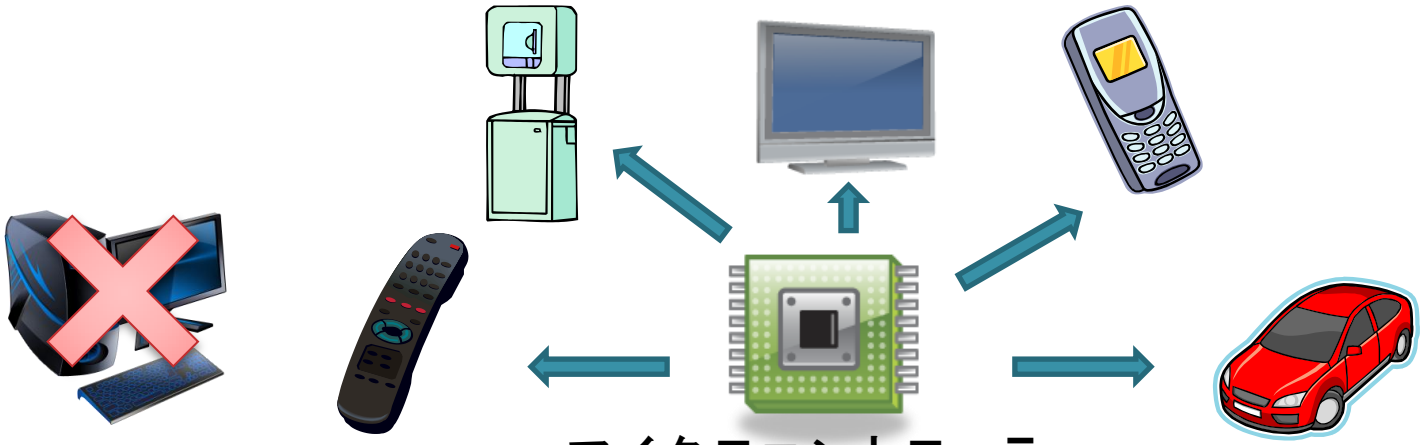
組み込みシステムとは?

組み込みシステムの要素

- ハードウェア
  - そのシステムの主たる機構(eg. 炊飯器⇒炊飯機構)
  - 最低限の制御用ハード
- OS
  - タスク処理を行うソフトウェア(軽量)
- ソフトウェア
  - 実際のシステムで行う処理

なぜ組み込みシステムなのか?

- リアルタイム性
  - 応答時間が一定の範囲内にあること
  - 目的の時間内に処理を完了させること
- 目的に特化
  - 汎用的な目的にはそもそも使わない
  - 汎用なシステムでは重すぎる



組み込みシステムとPCの違い

- 組み込みシステムは以下のような制約
  - 自律して動作
  - リソースは必要最低限
  - 特定の処理専用

- PCとの決定的な違い
  - PCは汎用的・高額・高機能
  - 単純な目的の機器にはオーバースペック

組み込みシステムとPCの違い

	組み込みシステム	PC
CPU性能	数十～数百MHz	数GHz
搭載メモリ	数k～数十MBytes	数GBytes
電源	省電力・バッテリ等	安定した電源が必要
UI	シンプル・限定的	キーボード、マウス等
起動・終了	高速起動、即時停止	少々時間がかかる
冷却	低発熱、ファンレス	低発熱なものもあるが…
信頼性	長時間の連続動作	高信頼なものもあるが…
できること	基本的に単機能	さまざま(ソフト次第)
価格	できるだけ安価	十数万円～

組み込みシステムとPCの違い

	組み込みシステム	PC
リアルタイム性	必要なケースが多い	さほど重要ではない
マルチタスク	リソースの有効活用のため厳密なタスク管理	豊富なリソースを用いたタイムシェアリング
安定性	高い安定性が求められる	連続動作には不向き
開発手法	PCを用いたクロス開発	オウン開発
開発環境	CPUや基板メーカ製(有料) 一部GNU開発環境	様々なメーカから発売 選択肢多数
デバッグ環境	ICEやJTAGを用いる 追加機材が必要	統合開発環境に組み込まれていることが多い
言語	C言語とその派生言語	様々 仕様で指定・慣れた環境

組み込みシステムの適用分野

- 産業用機械
  - 生産工場ライン
  - 組み立てロボット
- 単一用途の機器・家電
  - デジタルカメラ
  - 炊飯器
- 停止すると困るもの
  - ネットワーク機器
- その他
  - 自動車
  - 人工衛星・宇宙機

組み込みシステムの適用分野

- PC向き
  - 複数用途
    - 事務用
    - 個人用
  - 停止してもさほど害がないもの
    - 装置のモニタ
    - ミッションクリティカルでない制御
  - 物を動かさないシステム
    - 制御を伴わない計算処理

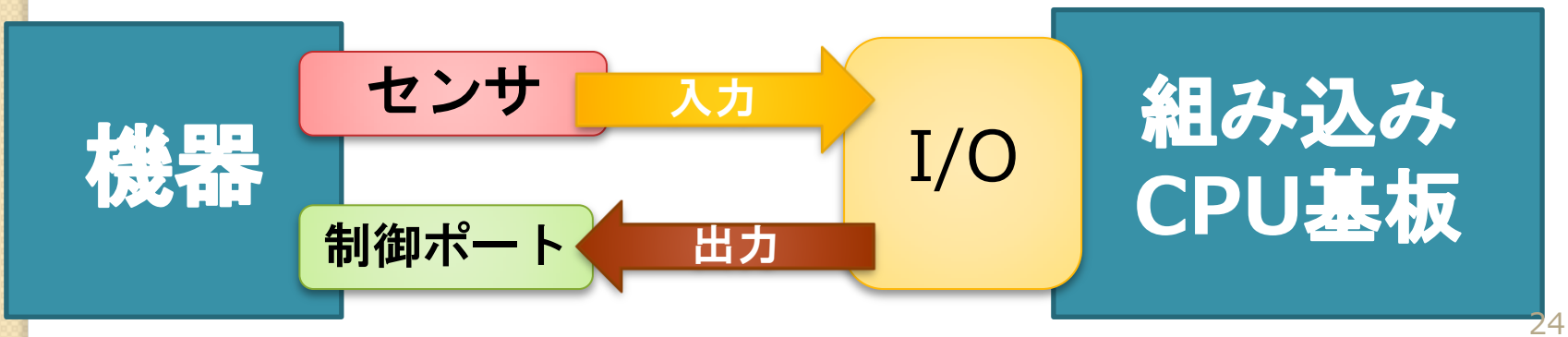
組み込みシステムの適用分野

- 組み込みシステムとして使われるPC
  - GUI処理
  - ネットワーク
  - 高速処理
- 例
  - スーパーやコンビニのレジ
  - マルチメディアステーション(KIOSK端末)
  - レントゲン端末
  - デジタルサイネージ

- PCの信頼性向上
- それでも、大型機械の制御は組み込みOS

I/Oとは

- 組み込みシステムはI/Oと密接な関係
  - PCでは基本的なI/Oはシステム側で用意
    - モニタ・スピーカ(出力)、キーボード・マウス(入力)、LAN(入出力)
  - 組み込みでは基本的なI/Oも自ら用意
  - I/Oを介して機器を制御



I/Oとは

- PCでのプログラミング
  - 例: いわゆる「hello, world.」

```
void main()
{
  printf(“hello, world.¥n”);
}
```

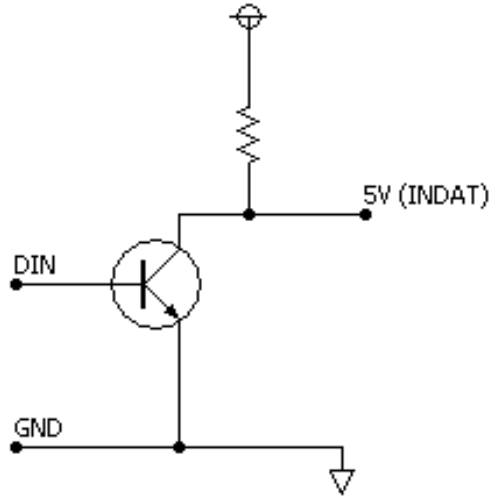
  - 標準出力(モニタ)に「hello, world.」と出力される
- 組み込みでのプログラミング
  - 何かの処理をさせるにも、どこかに出力が必要
  - 標準出力は用意されていない(ことが多い)
  - いきなりI/Oの制御が必要となる

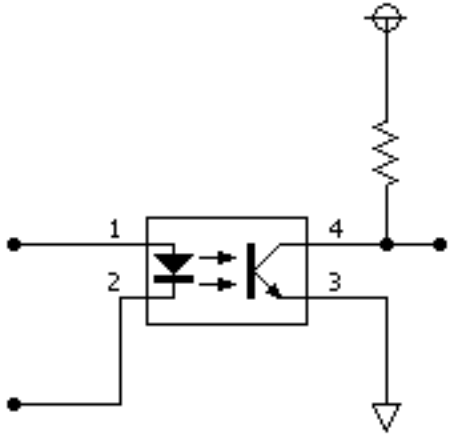
I/Oとは

- 入出力(Input/Output)を行う装置
- よくあるI/O
  - DI (デジタル入力)
    - スイッチ、接点、リレー
  - DO (デジタル出力)
    - リレー、ランプ、LED
  - AD (アナログ入力)
    - 音声入力、電圧測定
  - DA (アナログ出力)
    - 音声出力、波形発振
  - 通信ポート
    - 入力・出力を備える、シリアルポート、パラレルポート、LAN

I/Oの例

- DI/DO
  - Digital Input / Digital Ou
  - **接続方法に2種類**
    - **トランジスタ**
    - **絶縁タイプ**
  - On/Offの2値
    - **実体は電圧の高/低**



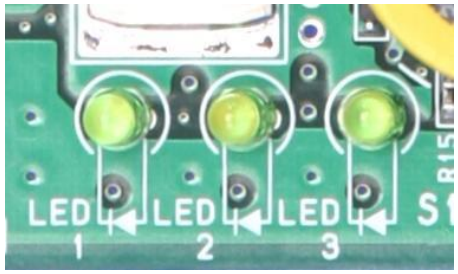


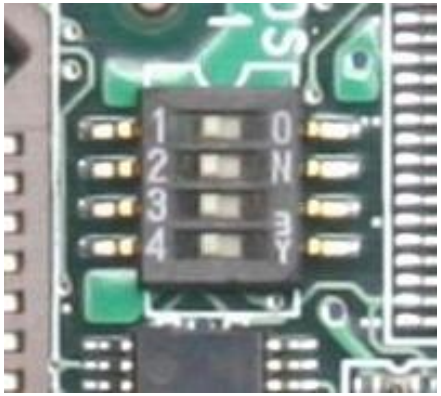
I/Oの例

- AD/DA
  - Analog to Digital (INPUT) / Digital to Analog (OUTPUT)
  - **アナログ電圧を入出力**
  - **入出力手順**
    - **アナログをデジタル値に変換 → デジタル値で入力**
    - **デジタル値で出力→デジタル値をアナログに変換**
  - **変換方法**
    - **逐次比較型**
    - **ハシゴ型**
    - **重み抵抗型**

I/Oの例

- **その他のI/O (CPUボード上)**
  - LED
    - **メインボード上に5つ**
  - RS-232C
    - **シリアル通信ポート**
  - **スイッチ**
    - DIPスイッチ
    - **押しボタンスイッチ**





学習キットでのI/O制御方法

- I/O制御
  - 該当アドレスに対し値の入出力
  - CPUの種類ごとに方法が異なる

  - Intelでは、IN命令とOUT命令
  - SHでは、MOV命令

  - 指定アドレスに対し代入 → Output
  - 指定アドレスの値を読む → Input

学習キットでのI/O制御方法

ビット	アドレス	接続先/名称	アクセス	内容
PA13	0xFFFFFC84	LED4	R/W	1: 点灯 / 0: 消灯
PA12		LED5	R/W	1: 点灯 / 0: 消灯
PA11		DIPスイッチBIT4	RO	1: OFF / 0: ON
PA10		DIPスイッチBIT3	RO	1: OFF / 0: ON
PA9		DIPスイッチBIT2	RO	1: OFF / 0: ON
PA8		DIPスイッチBIT1	RO	1: OFF / 0: ON
PA6		LED3	R/W	1: 点灯 / 0: 消灯
PA5		LED2	R/W	1: 点灯 / 0: 消灯
PA4		LED1	R/W	1: 点灯 / 0: 消灯
PA1		押しボタンスイッチ2	RO	1: OFF / 0: ON
PA0		押しボタンスイッチ1	RO	1: OFF / 0: ON
PB10	0xFFFFFX8C	D-sub 9pin/DTR1	R/W	RS232C CH1 DTR
PB9		D-sub 9pin/DSR1	RO	RS232C CH1 DSR
PB3		10pinヘッダ/DTR2	R/W	RS232C CH2 DTR
PB2		10pinヘッダ/RTS2	R/W	RS232C CH2 RTS
PB1		10pinヘッダ/DSR2	RO	RS232C CH2 DSR
PB0		10pinヘッダ/CTS2	RO	RS232C CH2 CTS

学習キットでのI/O制御方法

0xFFFFFC84

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		L4	L5	D4	D3	D2	D1		L3	L2	L1			P2	P1

P*n*: **押しボタン**SW *n*

L*n*: LED *n*

D*n*: DIP SW *n*bit

0xFFFFFC8C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					1	1						2	2	2	2
					DTR	DSR						DTR	RTS	DSR	CTS

1: RS-232C 1ch

2: RS-232C 2ch

学習キットでのI/O制御方法

- たとえば…
  - LED1を点灯させたい場合…
    - アドレス0xFFFFFC84の4ビットを立てる
      「0000 0000 0001 0000」 → 0x0010

      ```
      *((unsigned short*)0xFFFFFC84) = 0x0010;
      ```

  - DIPスイッチ1ビット目の情報を得る場合…
    - アドレス0xFFFFFC84の8ビット目を得る
      「0000 000● 0000 0000」
      ●部が 1 → On　0 → Off

オプションボードのI/O

- オプションボード上にあるI/O
  - DI/O: 入力8bit、出力8bit
  - ADコンバータ: 入力4ch
  - DAコンバータ: 出力4ch
  - モータードライバ (ステッピングモーター・DCモーター・ロボットアーム)
  - デバイス
    - LED (7SEG・マトリクス)
    - 圧電サウンダ
    - LCD

オプションボードのI/O

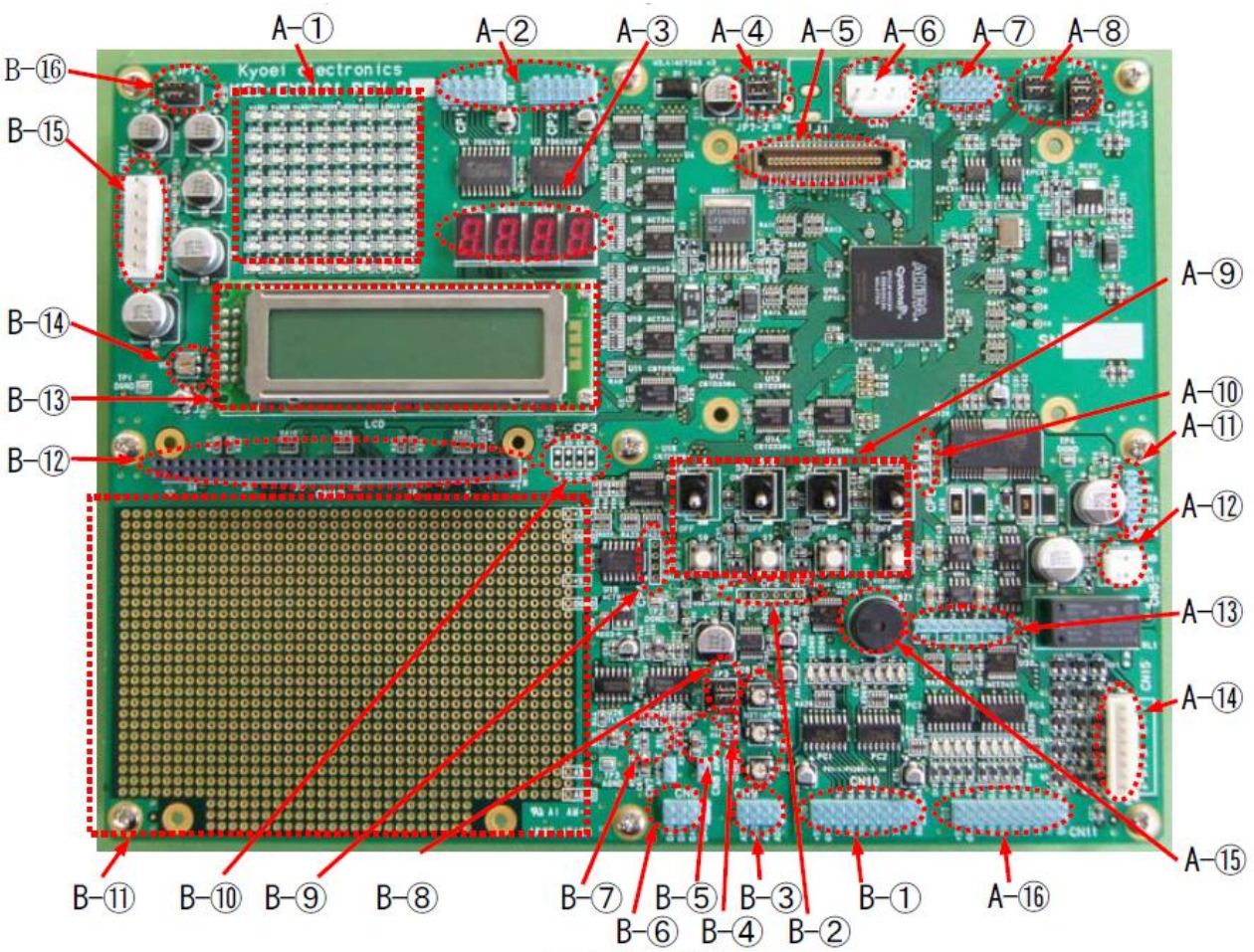
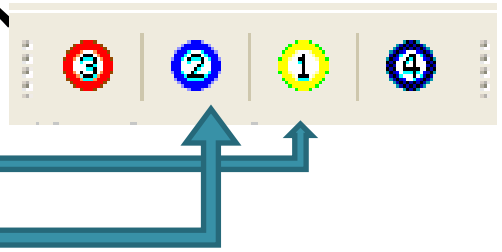


図 2-2　外観図

マイコンにおけるプログラミング

コンパイル方法(1)

- Eclipseを使う場合
  - デスクトップ上の「起動 eclipse.bat」を実行
  - ワークスペースを「C:¥KED-SH101¥cygwin¥home¥user¥jsp」に設定
  - プロジェクトを作成・開く
  - ファイルを編集
  - make depend
  - make (all)
  - 該当ディレクトリに生成されるxxxx.srec(OSなしの場合はxxxx.S)ファイルをボードに転送



コンパイル方法(2)

- Eclipseを使わない場合
  - デスクトップ上の「起動 cygwin.bat」を実行
  - カレントディレクトリを「/home/user/jsp」に移動
  - プロジェクトごとのディレクトリへ移動
  - ファイルを編集 (cygwin上でも、Windows上でも可)
  - $ make depend
  - $ make
  - 該当ディレクトリに生成されるxxxx.srec(OSなしの場合はxxxx.S)ファイルをボードに転送

ボードへの転送手順

- シリアルケーブルでPCとボードを繋ぐ
- ターミナルソフト(TeraTerm等)を起動
- 通信ポートにボードがつながったポート(COM4等)を選択
- 標準では9600bps
- ボードの両方のボタン(リセット/モード)を押し、リセットボタンから先に離す
- 生成されたモトローラファイル(拡張子: S or SREC)を転送
- 転送終了まで待つ
- 転送終了後、リセットボタンを押す

マイコン基礎実習I

これより、実機を使った実習に移ります。

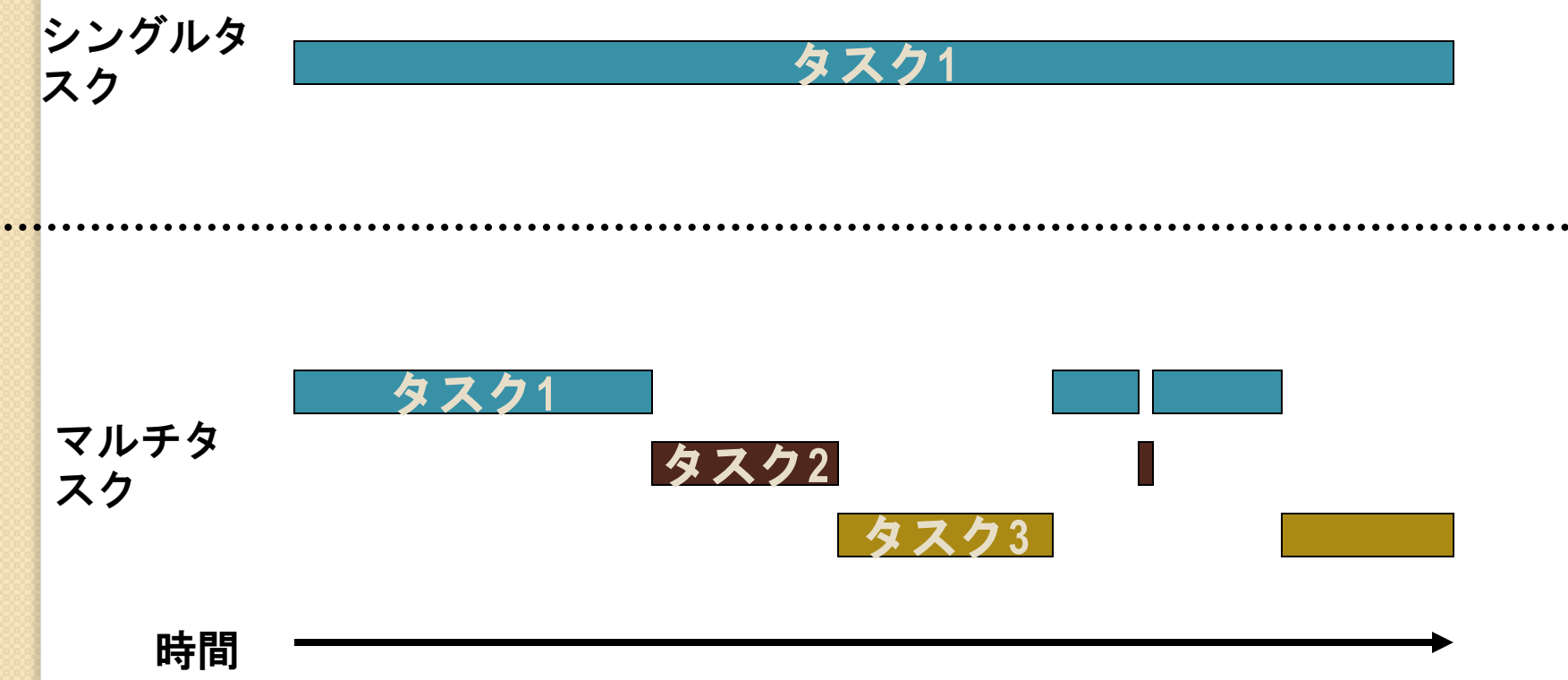
マルチタスク

シングルタスクとマルチタスク

- シングルタスク
  - 1度に1つのアプリケーションしか実行できない
  - PC用OSでの例: MS-DOS、MacOS 6以前
  - ここまでの実習で作成したものもシングルタスク
- マルチタスク
  - 1つのCPUで複数のプログラムを(ほぼ)同時に動かせる
    - μITRONではイベントドリブンにより実現
  - PC用OSでの例: Windows、Linux、MacOS 7以降

- マルチタスクにする理由
  - (がんばれば)シングルタスクでも同様な処理は可能
  - シングルタスクでは複雑になりやすい
  - シンプルなタスクを複数用意しマルチタスクに

シングルタスクとマルチタスク



シングルタスクとマルチタスク

- マルチタスクの種類
  - イベントドリブン
    - 何かをきっかけにタスク切り替え
    - 前のタスクの終了、外部からの入力、タイマ
    - リソースはそんなにいらない
    - 精密なタスク設計が必要
  - タイムスライス
    - 時間を小さく刻んで順次タスク切り替え
    - 切り替える必要がなくても切り替え
    - タスク設計が不十分でも変な動作はしにくい
    - リソースを多く消費する

シングルタスクとマルチタスク

- マルチタスクの種類
  - イベントドリブン
    - 何らかのイベントでタスクを切り替える
    - Windows 3.1以前
    - μITRON
  - タイムスライス
    - CPU時間を切り分けてタスクに分配する
    - Windows95以降

タスクとは?

- プログラムの一連の実行単位
  - Windowsではスレッド
  - ある程度独立した実行体系
  - 厳密な範囲の定義はない

- μITRONでのタスクの実体は
  - ユーザが作成した関数のこと
  - 1関数+呼び出し関数群

タスクとは?

- 携帯電話のタスクを想像してみる
  - 通話タスク
  - パケット通信タスク
  - メロディ再生タスク
  - 時計表示タスク
  - アラームタスク
  - 電波の送受信タスク
  - ボタンの監視タスク

  これらタスクの組み合わせで「携帯電話」として成り立つ

- アプリの「マルチタスク」とは別
  - ただの表示切り替えのこともある
    (後ろで動いていない)

タスク管理

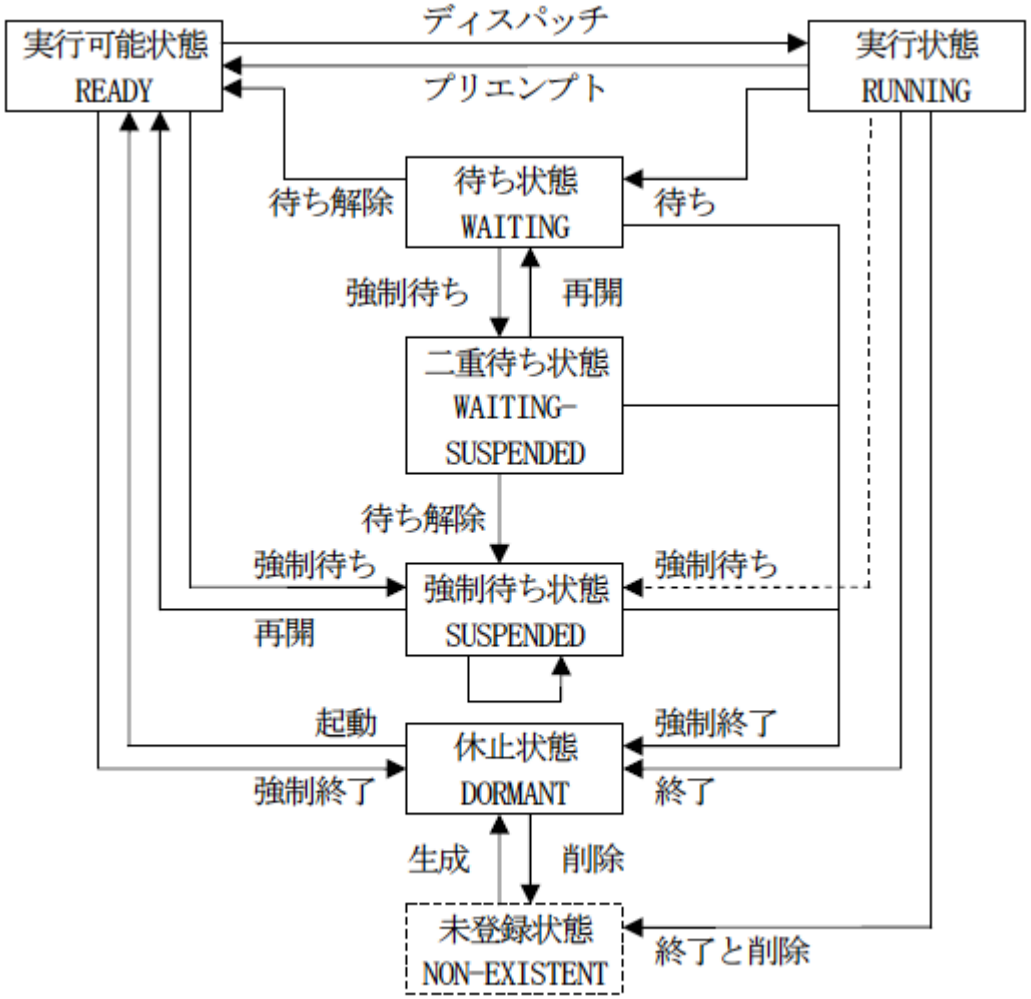
- マルチタスクではタスク管理が重要
  - シングルタスクの場合
    - ソースコードの順で実行される
  - マルチタスクの場合
    - 優先度/優先順位
    - 割り込み(イベント)
    - タスク間通信

    等によりタスクの実行を管理する必要がある

タスク管理

タスクの状態	説明
*実行状態 (RUNNING)	タスクが実行されている状態
*実行待ち状態 (READY)	タスクの準備はできているが、優先度の高いタスクが実行中
*待ち状態 (WAIT)	条件が整うまで、タスクの実行を待っている状態
強制待ち状態 (SUSPENDED)	他のタスクによって実行を中断された状態
二重待ち状態 (WAITING-SUSPENDED)	待ち状態中に他のタスクから中断された状態
*休止状態 (DORMANT)	タスクが起動指定にない状態
未登録状態 (NON-EXISTENT)	まだタスクが生成されていない仮想的な状態

タスク管理



タスク状態の遷移図

μITRON 4.0 **仕様書より**
http://www.ertl.jp/ITRON/SPEC/FILE/mitron-402j.pdf

タスク管理

- 割り込み
  - 割り込み処理とは?

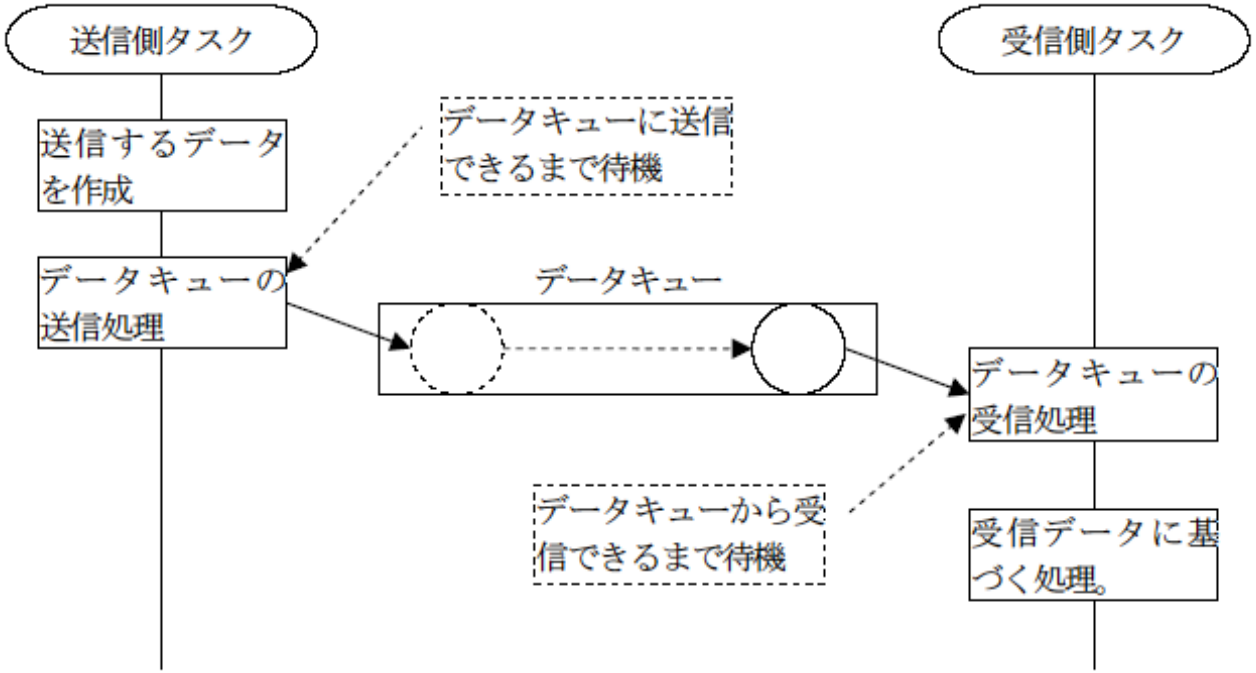
    ある連続処理中に、非同期で別の処理を割り込ませること
    - ユーザ入力/機器からのデータ入力 (外部割り込み)
    - CPU内部(内部割り込み)
  - 割り込みハンドラ
    - 割り込み発生時に実行される処理

タスク管理

- タスク間通信
  - タスク間で同期を取るために必要
  - 同期オブジェクト
    - セマフォ
    - イベントフラグ
    - データキュー
    - メールボックス

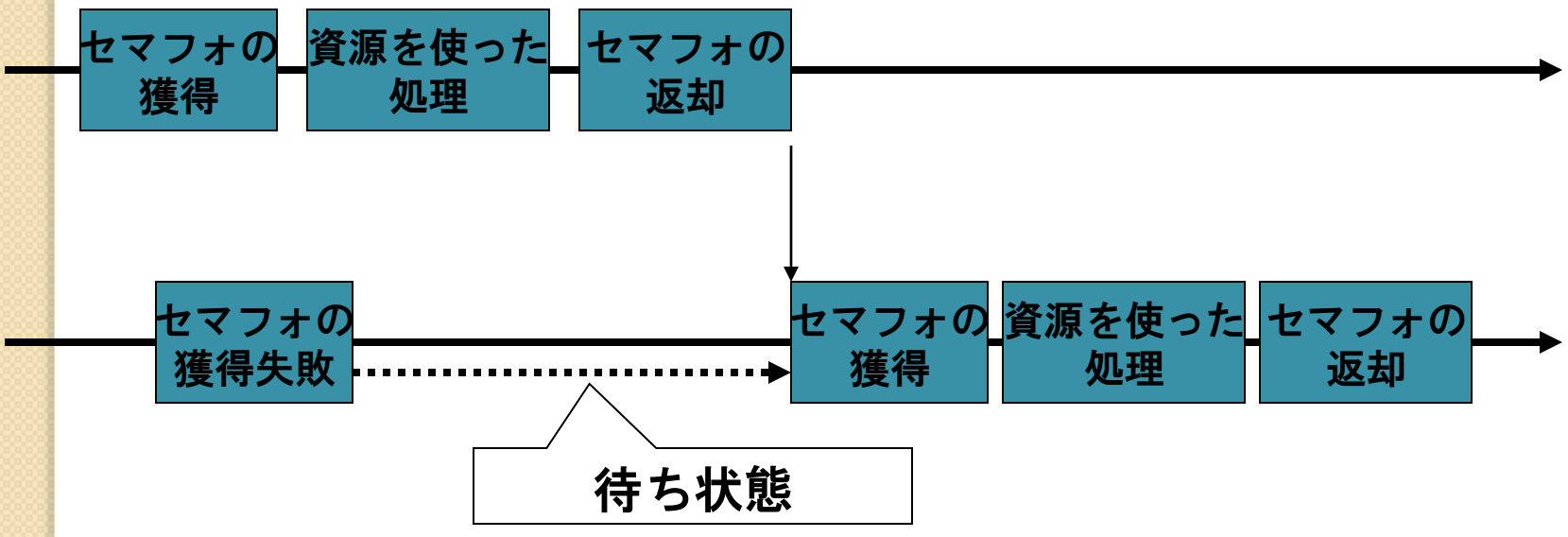
タスク管理

- データキューについて
  - タスクとタスクの橋渡し
  - 受信側はデータが受信されるまで待ち状態
  - 送信側はデータが送信されるまで待ち状態



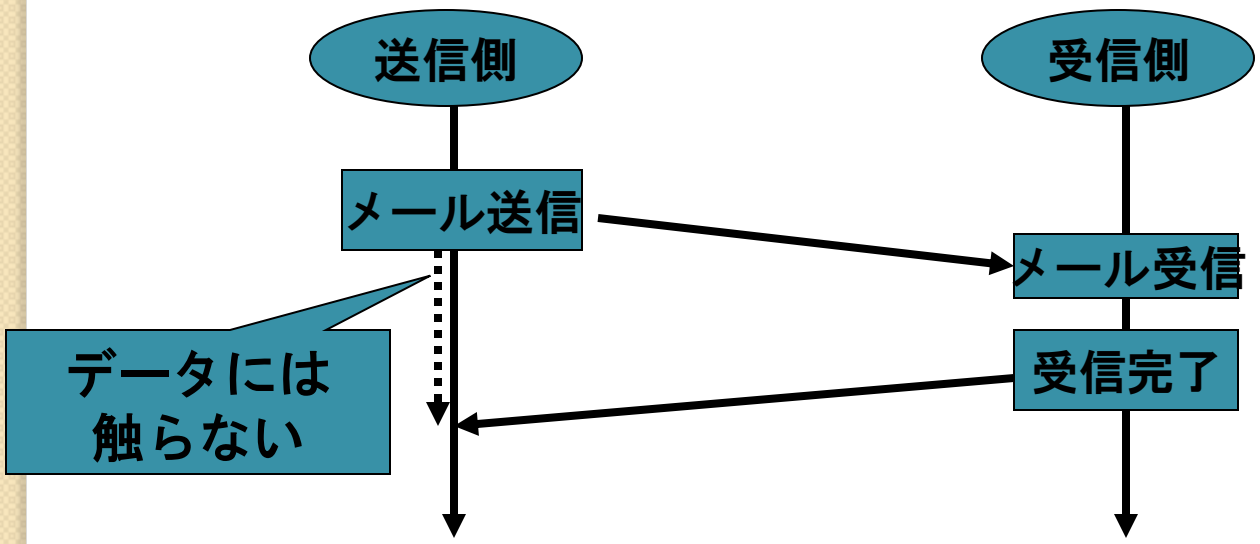
タスク管理

- セマフォについて
  - 資源(I/Oなど)の排他制御に用いる
  - セマフォを獲得できない場合、待ち状態
  - 1つの資源 --- バイナリセマフォ
  - 複数の資源 --- カウンタセマフォ (eg. 駐車場の発券器)



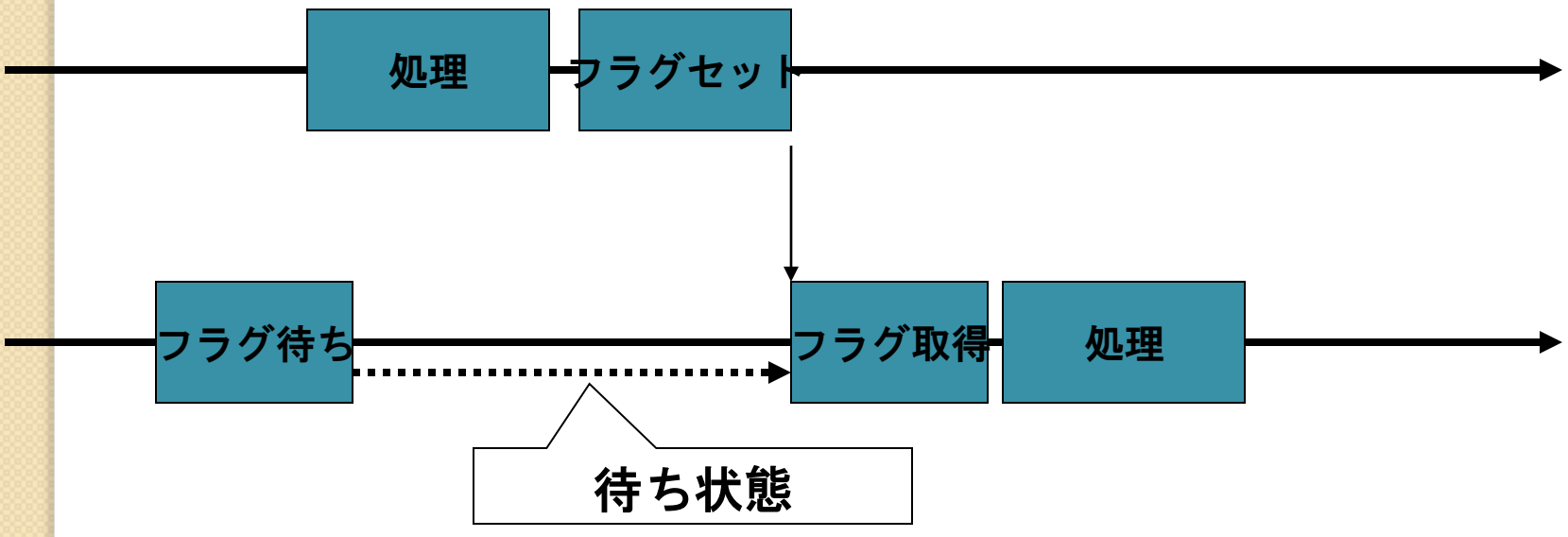
タスク管理

- メールボックスについて
  - ブロックデータの通信に利用
  - サイズは可変長
  - データではなく、データ領域の先頭アドレスを送信
  - 受信確認まで、送信元はアクセス禁止



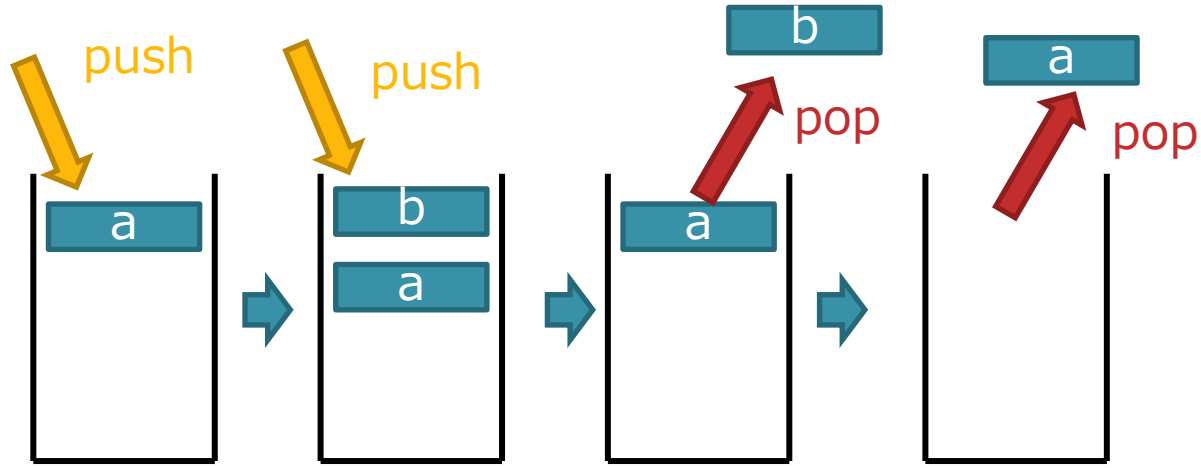
タスク管理

- イベントフラグについて
  - 他タスクにイベントを通知
  - 処理の完了の通知などに利用
  - データの受渡しも可能 (16bit)
  - 受信側はフラグがセットされるまで待ち状態



タスク管理

- スタック
  - 後入れ先出し(LIFO)の構造でデータを保持するもの

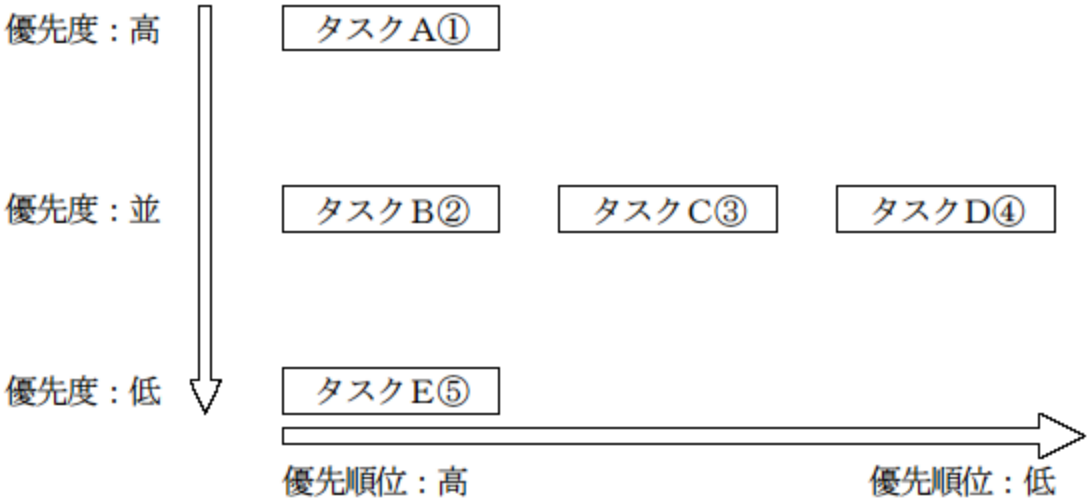


この仕組みを用い、関数呼び出し時にプログラムカウンタ(PC)を保持する目的に使う

  - スタックサイズをオーバーしないような設計が必要

μITRONでのタスク管理

- 優先度/優先順位
  - 複数のタスクが動作している場合
    - 優先度の高いタスクから
    - 同一優先度の場合は優先順位の高いタスクから実行される



μITRON 4.0 仕様書より
http://www.ertl.jp/ITRON/SPEC/FILE/mitron-402j.pdf

μITRONでのタスク管理

- コンフィギュレーションファイルについて
  - タスクや資源の生成情報を記述
  - 静的にタスクを生成する

```
CRE_TASK(   タスクID,
                {タスク属性，拡張情報，起動アドレス，
                 起動時優先度，スタックサイズ，スタック位置});
```

- タスクID　　：タスクの識別用ID
- タスク属性　　：TA_HLNGやTA_ACT (後述)
- 拡張情報　　：タスク起動時に渡されるデータ
- 起動時アドレス　　：関数名
- 起動時優先度　　：1~16、小さいほど高い
- スタックサイズ　　：スタックの大きさ
- スタック位置　　：TOPPERSでは「NULL」固定

μITRONでのタスク管理

- CRE_TSKの「タスク属性」
  - 言語に関するもの
    - **TA_HLNG**
      - タスクが高級言語(C言語)で記述されていることを示す
    - TA_ASM
      - タスクがアセンブリ言語で記述されていることを示す
  - タスクの状態に関するもの
    - **TA_ACT**
      - OSの起動時にタスクを「**実行状態 / 実行可能状態**」にする
- 複数(TA_HLNGとTA_ACT 等)指定する場合は、「|」で区切る

例

```
CRE_TSK(TASK1,{TA_HLNG|TA_ACT,0,test1,5,8192,NULL});
```

この場合、関数test1()をTASK1タスクとし、優先度5、スタックサイズ8192bytes、高級言語使用、OS起動時に実行(可能)状態で生成

OS μITRONについて

μITRONとは?

- **組み込みシステム用リアルタイムOSの仕様**
- TRONプロジェクトの一部

- **本キットでは…**
  - μITRON 4.0に準拠したTOPPERS/JSPカーネルを使用
    - TOPPERS/JSPの特徴
      - マルチタスク
      - μITRON APIに準じたタスク/リソース管理
      - リアルタイムカーネル
      - 実行形式プログラムが小さい
      - オープンソース

他のOSとの比較

	μITRON	RT-Linux	Vx-Works	Symbian
対象	家電組み込み	制御組み込み	制御組み込み	携帯電話
オープンソース	○	○	×	×
対象CPU	多数	多数 (x86得意)	多数	携帯電話用
リアルタイム性	○	△	○	○

TRONプロジェクトとは?

- 東京大学　坂村教授が提唱
- 1980年に基本構想が提唱され、1984年より産学協同プロジェクトとしてスタート
  - 現在、成果物であるTRON仕様のOSは数多くの組み込み機器用OSとして採用される。
- TRONプロジェクトの目的
  - “どこでもコンピュータ”
  - “ユビキタスコンピューティング”

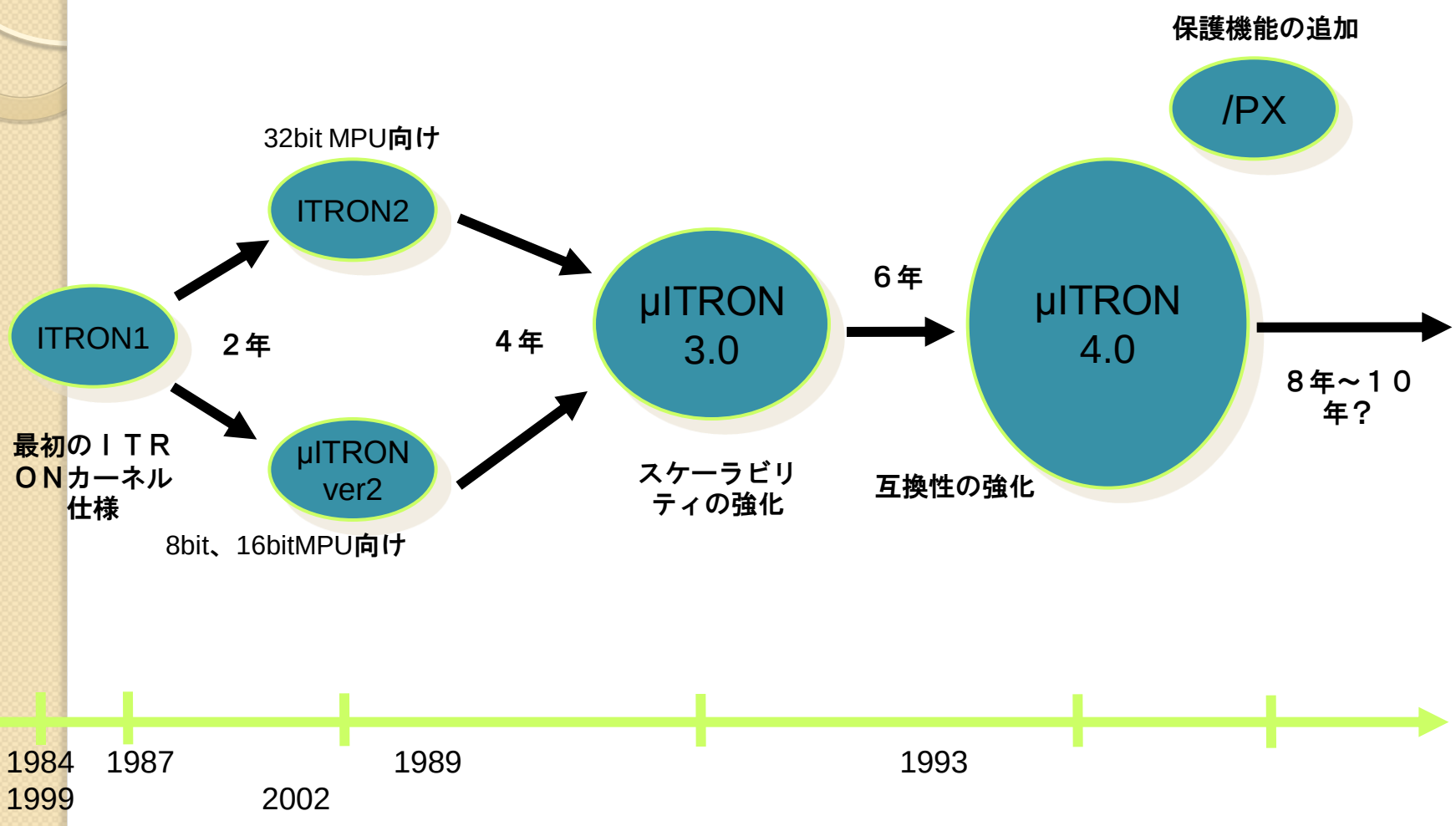
これまでの成果物

- 「ITRON」（Industrial TRON ）
  - 組み込みシステム用リアルタイムOS仕様
- 「BTRON」（Business TRON ）
  - PC・ワークステーション用OS仕様
- 「CTRON」（Central TRON）
  - 比較的大型のコンピュータで動作するリアルタイムOS仕様（主に通信系）
- その他
  - JTRONなど

ITRONの特徴

- OSの小形軽量化が可能な仕様である
- 仕様が無料で公開されていて、自由に利用可
- 多種多様なマイコンで使うことができる
- 多くのメーカーが採用している

ITRON仕様の歴史


保護機能の追加
/PX
32bit MPU向け
ITRON2
ITRON1
２年
４年
μITRON
3.0
６年
μITRON
4.0
８年～１０
年？
最初のＩＴＲ
ＯＮカーネル
仕様
μITRON
ver2
8bit、16bitMPU向け
スケーラビリ
ティの強化
互換性の強化
1984
1987
1989
1993
1999
2002

ITRON OSの現状

- TOPPERS/JSP（名古屋大学　高田教授）
  - これまでの“弱い標準化”を継承
  - 貧弱なハードウェアに適合するための標準化手法
- T-Engine（東京大学　坂村教授）
  - “強い標準化”
  - ハードウェアやカーネルを規定

マイコン基礎実習II

これより、実機を使った実習に移ります。