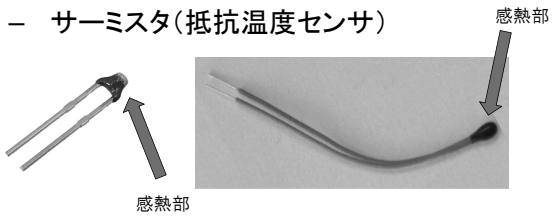
制御技術とコンピュータ そして組み込み

Ei2　ハードウエア技術

- 再び水温制御　温度センサ‥‥検出部

設定温度以上未満でスイッチング（ON－OFF）

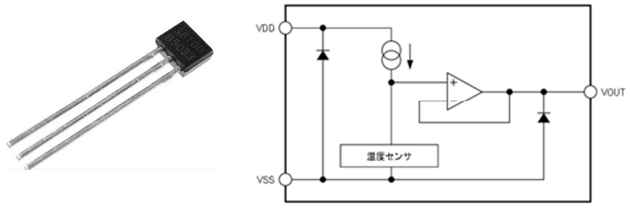
- 温感素子
  - サーミスタ（抵抗温度センサ）



- 再び水温制御　温度センサ‥‥検出部

設定温度以上未満でスイッチング（ON－OFF）

- 温感素子
  - 半導体温度センサ（IC型・ダイオード）

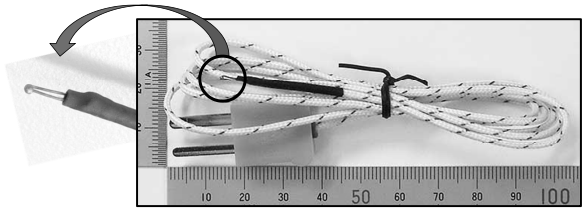


- 再び水温制御　温度センサ‥‥検出部

設定温度以上未満でスイッチング（ON－OFF）

- 温感素子
  - 熱電対（センサ）温度計

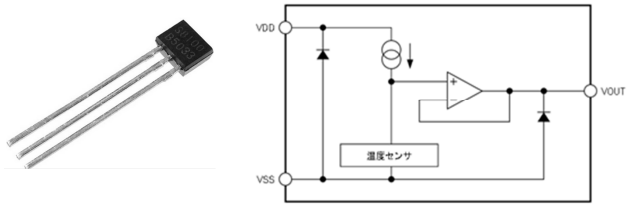
2種金属の接合点において起電力が生じる現象（ゼーベック効果）を利用



- 再び水温制御　温度センサ‥‥検出部

設定温度以上未満でスイッチング（ON－OFF）

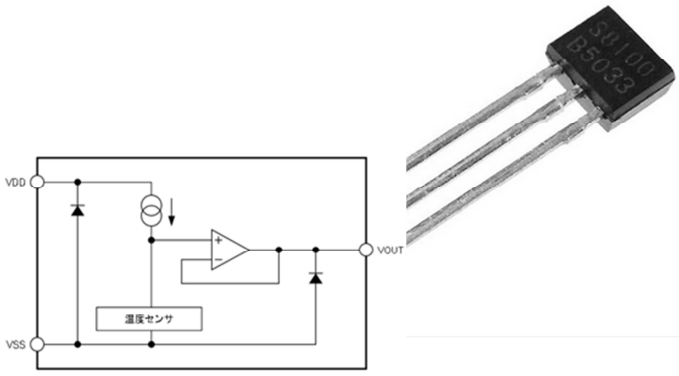
- 温感素子
  - 半導体温度センサ（IC型・ダイオード）



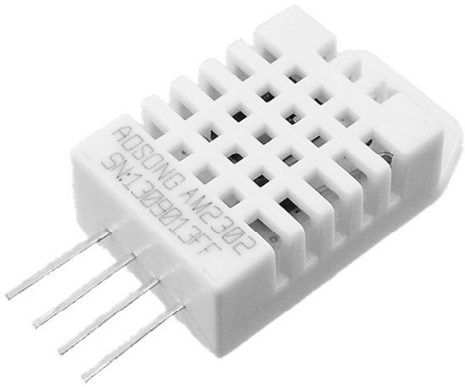
- 温度センサ　サーミスタ素子(PCB実装型)

- 温度センサ　サーミスタ（自動車用）

- 温度センサ　IC温度センサ　(PCB実装型)



- 温湿度センサ　センサーモジュール



- 温湿度センサ　センサーモジュール

図1 高分子素子型湿度センサの種類

図2 抵抗変化型湿度センサの基本構造

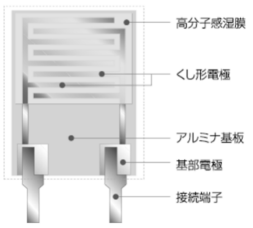
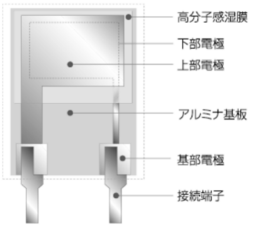
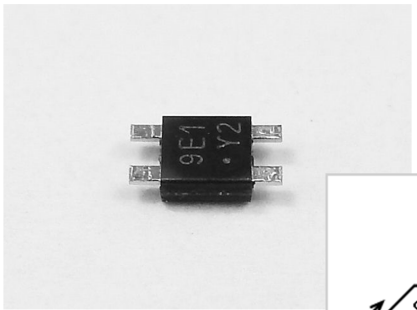
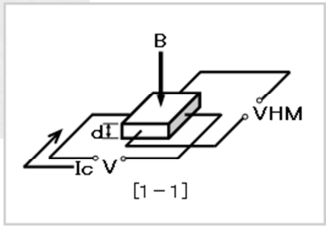


図3 静電容量変化型湿度センサの基本構造

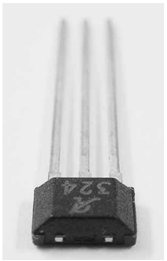


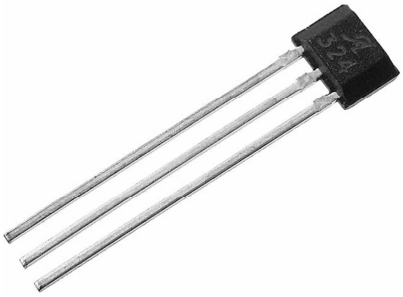
- 磁気センサ　ホール素子



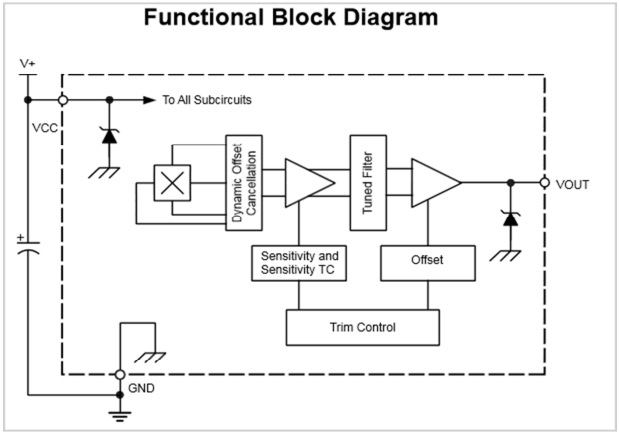


- 磁気センサ　　ホール素子

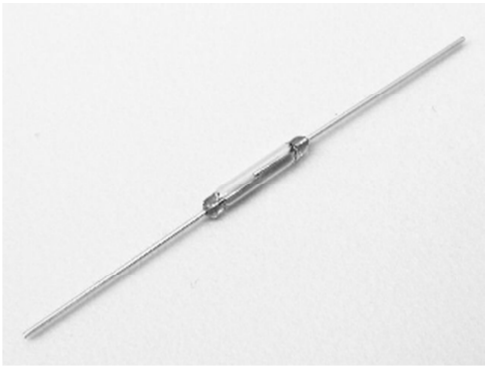




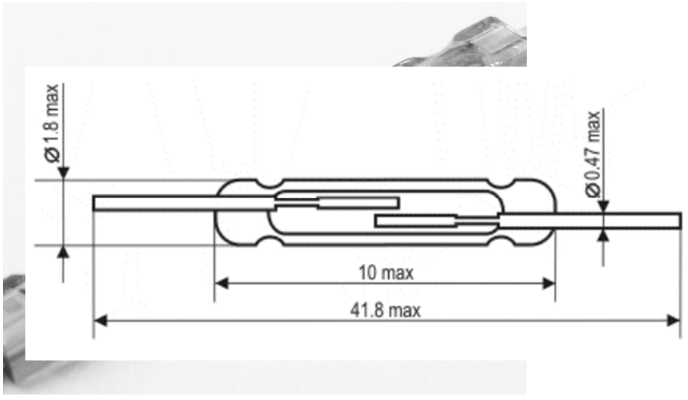
- 磁気センサ　　ホール素子



- 磁気センサ　リードスイッチ



- 磁気センサ　リードスイッチ

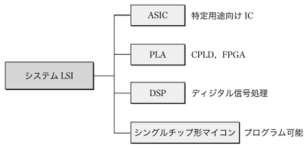


組込みハードウェアの LSI 化

組込みハードウェアを小型化するためには，回路を LSI 化することが有効である。近年では，半導体技術の進歩によって大規模な回路であっても，比較的容易に LSI 化することが可能になった。ここでは，組込みハードウェアに使われる LSI について学習する。

システム LSI

ハードウェアとして必要な主要機能を 1 個の LSI チップに内蔵することを，SoCといい，このような LSI をシステム LSIという。
図 6-18 に，システム LSI のおもな種類を示す。システム LSI は，ASICや PLAを用いて実現することができる。また，音声や画像処理を専用に行う LSI である DSPやシングルチップ形マイコンもシステムLSIの1種である。



Complex Programmable Logic Device
Field Programmable Gate Array

図 6-18　システム LSI のおもな種類

- センサのまとめ

何で何を測定するの？

・ジャイロセンサー　　・星測位システム関連（GPSモジュール）

・加速度センサー　　・温度センサー　　・湿度センサー

・光センサー　CdS(硫化ｶﾄﾞﾐｳﾑｾﾙ)　　フォトダイオード
焦電ｾﾝｻｰ(人感ｾﾝｻｰ)　　UVｾﾝｻ(紫外線ｾﾝｻ)
ﾌｫﾄﾘﾌﾚｸﾀ(反射型光ｾﾝｻ)　　ﾌｫﾄｲﾝﾀﾗﾌﾟﾀ(通過型光ｾﾝｻ)

・放射線センサ　（シンチレーター）GM管　（ ガイガー管 ）

・音センサ　音声帯域（マイク）　超音波センサ　　・ｐＨ(酸/ｱﾙｶﾘ(塩基)ｾﾝｻｰ)

・アルコール・ガス センサ

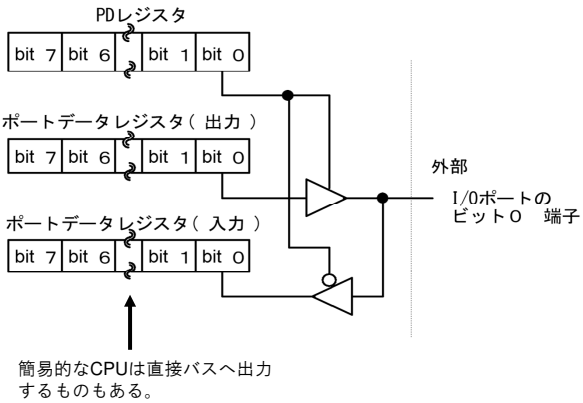
・距離センサー　　・動体ｾﾝｻｰ(ﾄﾞｯﾌﾟﾗｰﾚｰﾀﾞｰｾﾝｻｰ)

・角度センサー　　・磁気センサー/ホール素子　リードスイッチ

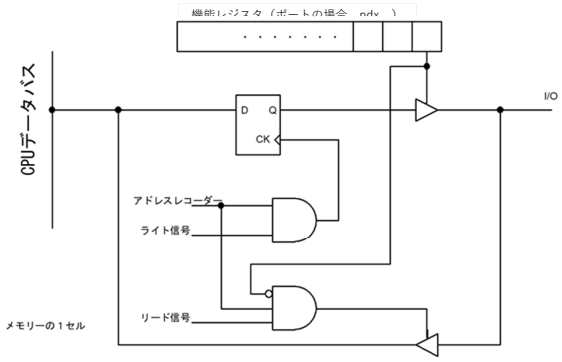
・圧力･加圧･感圧･大気圧ｾﾝｻｰ　荷重変換器(ﾛｰﾄﾞｾﾙ Loadcell)

・ジョイスティック　　・チルトスイッチ　　・電流センサ

I/Oポートの設定



I/Oポートの仕組み もうちょっと詳しく　R8Cの実際



I/O関係だけ抜粋

マイコンの初期化　I/Oの準備

```
// ポートの入出力設定
  p0   = 0x00;
  prc2  = 1;          // PD0のプロテクト解除
  pd0   = 0x70;       // Sin p0_7
  pur0 |= 0x04;       // P1_3～P1_0のプルアップON
  p1   = 0x00;
  pd1   = 0x10;
```

p1 や　pd1は変数のようにアクセスできますが変数とはちょっと違います。
変数の実アドレスはリンカーが決定。
p1　pd1はLSI設計時にアドレスが決められます。

アドレス情報をコンパイラに伝える
＃ｐraguma　ADDRESS

```
#pragma  ADDRESS  p0_addr  00E0H   // Port P0 Register
#pragma  ADDRESS  pd0_addr  00E2H  //P0 Direction  Register
```

• この後

```
#define   p0     p0_addr .byte
#define   pd0    pd0_addr .byte
```

というようにdefaineしています。

＃ｐraguma
機種依存のいろいろな設定を行う命令なので、
この命令の働きはコンパイラによって異なります

#define宣言で　割り当ててみる？

```
#define   p0_addr  00E0H
```

・・・・ではだめ！！　理由を考えよう

```
unsigned char  p0_addr ;      //変数として宣言
```

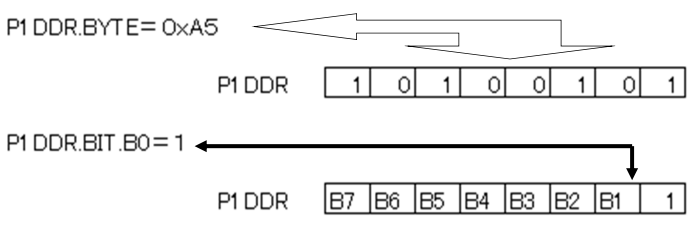
変数宣言ではアドレスがどこに振られるかわかりません。
（ちなみにビルドでエラーは出ません）

⑵ビットフィールドを読み取る場合

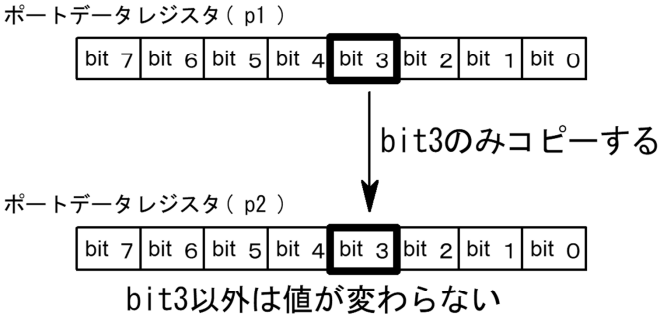
```
Union  pd1 {                        /*  union  pd1    */
       unsigned char BYTE;          /*  Byte Access   */
       struct {                     /*  Bit  Access   */
           unsigned char B7:1;      /*  Bit 7         */
           unsigned char B6:1;      /*  Bit 6         */
           unsigned char B5:1;      /*  Bit 5         */
           unsigned char B4:1;      /*  Bit 4         */
           unsigned char B3:1;      /*  Bit 3         */
           unsigned char B2:1;      /*  Bit 2         */
           unsigned char B1:1;      /*  Bit 1         */
           unsigned char B0:1;      /*  Bit 0         */
           }   BIT;
};
```

８ビットアクセス　pd1.BYTE
１ビットアクセス　pd1.BIT.B0

８ビットアクセス　P1DDR.BYTE
１ビットアクセス　P1DDR.BIT.B0
UNION　メモリーを共有する（共用体）



p1の３bitをp2の３ビットへ転送する



ビットフィールド構造体を利用すると

p1の３bitをp2の３ビットへ転送する例、同じ実行結果になるプログラムです

```
315 ## # C_SRC :          p2_3 = p1_3;

 316  0001E  7EBF0B07              btst 3,_p1_addr
 317  00022  7E2F2307FA            bmnz    3,_p2_addr
```

```
315    ;## # C_SRC :      p2=( p2 & 0xf7 ) | ( p1 & 0x08);

 316  0001E  33E400          S   mov.b    _p2_addr,A0
 317  00021  7724F700            and.w    #00f7H,A0
 318  00025  0BE100          S   mov.b    _p1_addr,R0L
 319  00028  B3              Z   mov.b    #00H,R0H
 320  00029  77200800            and.w    #0008H,R0
 321  0002D  9904                or.w R0,A0
 322  0002F  724FE400            mov.b    A0,_p2_addr
```

コード効率が良くなる＝実行速度が速い

ビットフィールドの長所と短所

- 長所
  - プログラムがわかりやすい（マスク処理省略）
  - コード効率が良い（実行が速い）
- 短所
  - 機種依存性があり、プログラムの移植性が悪い
  - 実変数名が長くなる
  - ヘッダファイル（構造体宣言）が必要

機能モジュールが膨大になり大変な量となる

以前はプログラマーが必要な機能だけ宣言していたが、利便性の向上のためメーカーが準備するようになった。
“ヘッダーファイルのないCPUは売れなくなった”