

# マイクロエレクトロニクス マイコンで利用されるシリアル通信技術

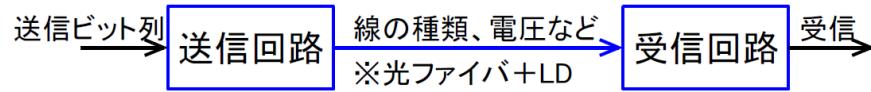
出展：東北学院大学熊谷正朗  
氏

## 通信のハードウェア

### ○ 信号の伝送経路

#### ◇線と送受信回路

- ・ただの線/インピーダンスの決まった線
  - ・信号レベル (=0/1の表記)
- 例) ロジック信号(5V, 3.3V等)そのまま、  
高めの電圧に(RS232C=±数V等)、  
低めの差動電圧に(RS485, CAN, LVDS等)

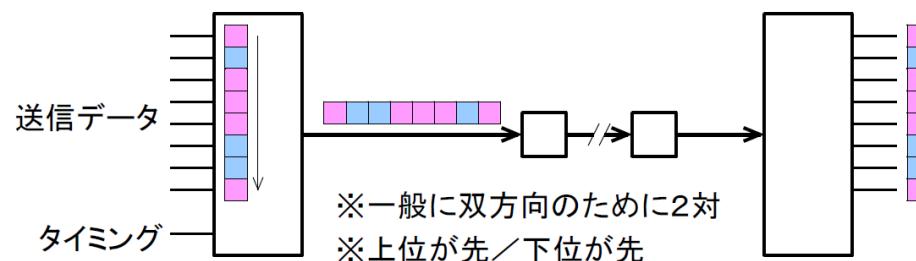


## 通信のハードウェア

### ○ バイトと信号の変換

#### ◇UART, シリアライザ/デシリアルライザ

- ・信号の直列化 / ビット列から復元
- ・一般に専用回路を使うがソフトで実装も。

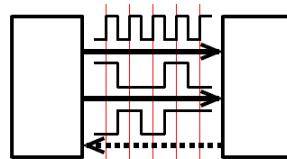


## 通信のハードウェア

### ○ 信号のタイミング

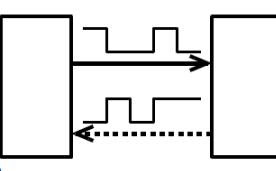
#### ◇データ線 + クロック線

- ・信号のタイミングを専用線(CLK)で規定。
- ・通信速度は自由。



#### ◇データの線のみ

- ・送受で速度・タイミングをあわせる。
- ・データにクロックを埋める。

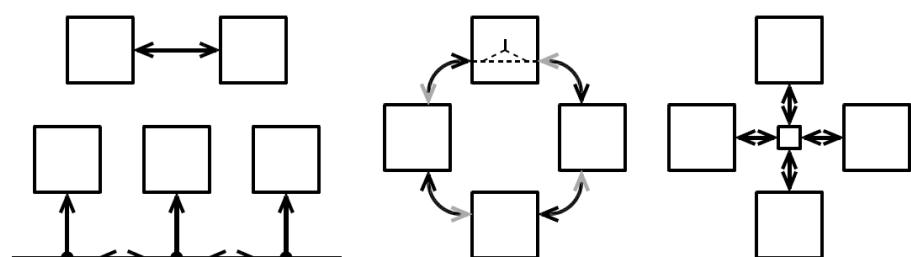


## 通信のハードウェア

### ○ トポロジー

#### ◇接続の形態

- ・1対1 / バス / リング  
(/スター/メッシュ)

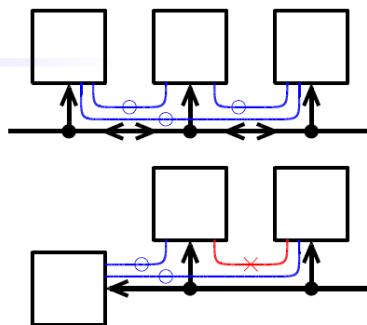


## 通信のハードウェア

### ○ トポロジー

#### ◇バス接続

- ・通信用の信号を3個以上で**共用**する。
- ・多数を**省配線**で接続可能。
- ・全部対等 / 1個が全体を制御。
- ・通信の**衝突**に対する対策が必要(検出・再送)
- ・I2C, CAN、昔のネット(10BASE2など)



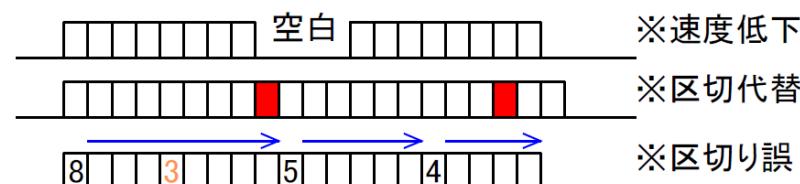
## 通信のソフトウェア

### ○ データとバイト列の変換

#### ◇データの区切りの明確化

- ・主に受け側での復元処理のため。
- ・1: 明確な特定の信号状態、バイト値  
例) 信号オフ、0は区切り、改行

2: バイト列に長さ情報を付加→終点

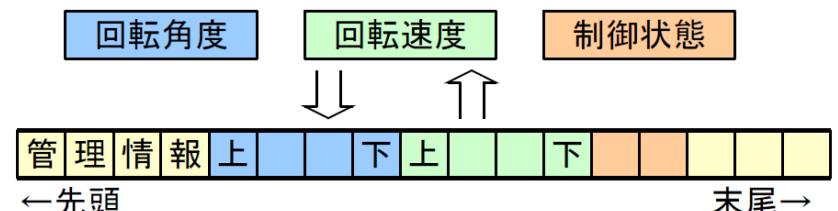


## 通信のソフトウェア

### ○ データとバイト列の変換

#### ◇データのパックと送信順序

- ・多くの通信は本質的には1ビット単位、大抵はバイト単位にまとめて取り扱い。
- ・情報をバイト列に並べる、取り出す。
- ・上位と下位の位置に注意(エンディアン)

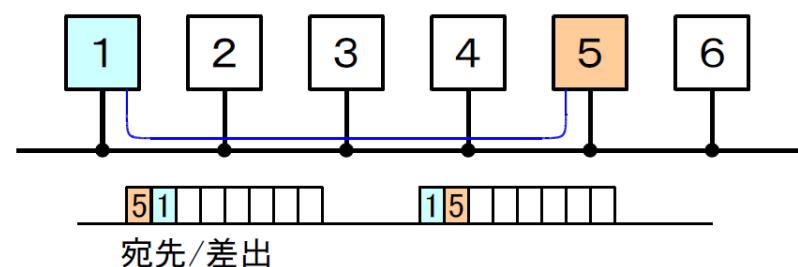


## 通信のソフトウェア

### ○ データとバイト列の変換

#### ◇宛先(アドレス)の付加

- ・1対多、多対多の通信では**宛先**が必須。
- ・返答をもらうために自分のアドレスも含む。

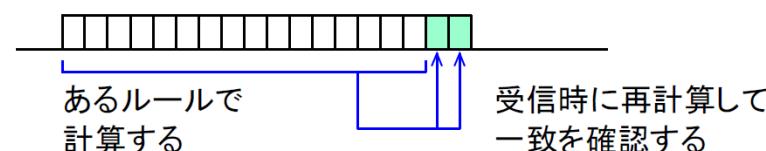


## 通信のソフトウェア

### ○ データとバイト列の変換

#### ◇バイト列の正しさの検出

- ・ローレベルな通信では通信エラーは存在するとして、対策する必要がある。
- ・エラー発生の検出→破棄、or→復元
- ・パリティ / チェックサム / CRC



## 通信のソフトウェア

### ○ 送受データの取り決め (プロトコル)

#### ◇送る内容の明確化

- ・複数種類のデータのやりとりをしたい。  
→「なにを送るか」を送信に含める  
※データ長などとあわせ、ヘッダと呼ばれる

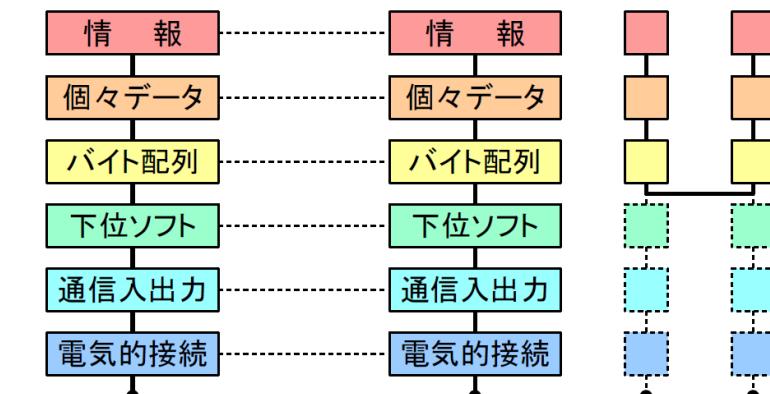
#### ◇要求と応答

- ・情報の要求→応答のルール化
- ・情報受信の返答(ACK=アクノリッジ)

## コンピュータ間通信

### ○ 通信の実際

#### ◇ハードとソフトの多層構造 / 下位の隠蔽



## 実際に用いられる通信

### ○ 概 要

#### ◇デバイス(センサ、AD等)～マイコン間

- ・SPI型 (Microwire, 3線式、4線式、並種多数)
- ・I2C

#### ◇マイコン間

- ・調歩同期 (シリアルポート、UART, RS232)
- ・I2C (ただし、対等ではない)
- ・CAN
- ・(Ethernet, USB)

## 実際に用いられる通信

### ○ 概 要

#### ◇マイコン～パソコン(OS付き高性能マイコン)

- ・調歩同期 (シリアルポート、UART, RS232)
- ・USB (ネイティブ、シリアルポート変換)
- ・Ethernet (いわゆるネットワーク接続)

#### ◇無線系

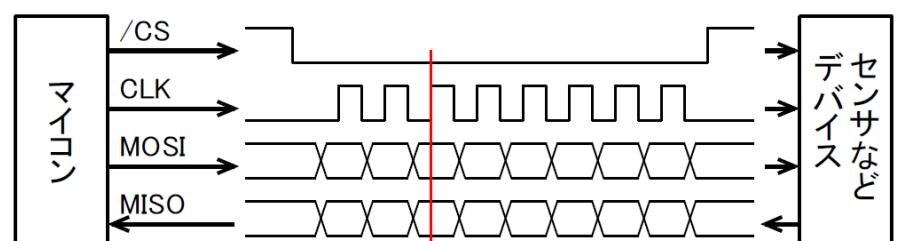
- ・Bluetooth, Zigbee, XBee, 特定小電力(2.4G)
- ・無線LAN
- ・携帯電話ネットワーク

## デバイス～マイコン間の通信

### ○ SPI型

#### ◇特徴

- ・1対1(1対多)
- ・クロックあり、信号線の方向は専用。
- ・センサやAD変換器の接続に多い。

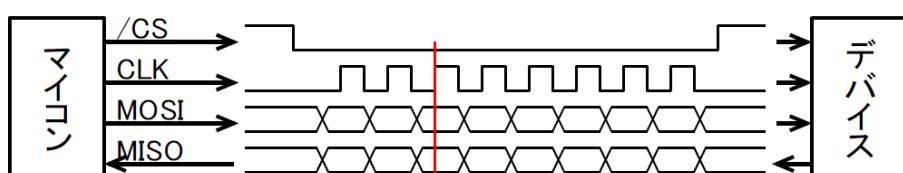


## デバイス～マイコン間の通信

### ○ SPI型

#### ◇使い方 (専用IFは不要、ソフトでOK)

- ・CSを下げる→MOSIに出力ビットを用意  
→CLKを△▽→次のビット
- ・CLK▽→MISOで読み→CLK△
- ・CSを個別につなぐと複数デバイス可。



## デバイス～マイコン間の通信

### ○ I<sup>2</sup>C

#### ◇特徴

- ・1対多 (多対多) バス配線型 (=双方向)
- ・デバイスにアドレスがある。
- ・クロックあり、データは上り下り共通。
- ・低～中速のセンサを複数つかうケース。

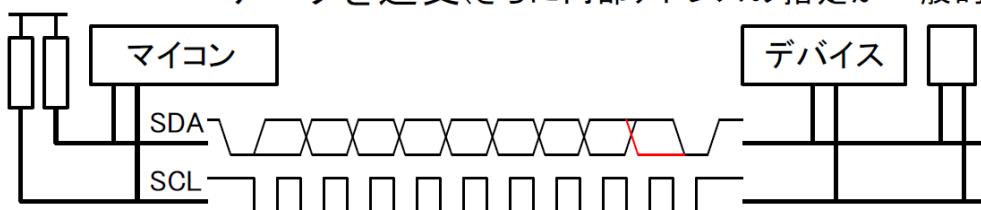


## デバイス～マイコン間の通信

### ○ I<sup>2</sup>C

#### ◇使い方

- ・専用IFを持つマイコンを採用する。
- ・マイコンのIFを適切に設定する。
- ・デバイスの説明書に従い、アドレスを指定、データを送受(さらに内部アドレスの指定が一般的)



## デバイス～マイコン間の通信

### ○ I<sup>2</sup>C

#### ◇補足メモ

- ・SPI型に比べて、マイコン設定のため複雑。
- ・デバイスの内部アドレスの指定→デバイスからの応答受信などの手続きが面倒(信号線共有のため)
- ・トラブル発生時に、IFの設定が悪いのかデバイスの使い方が悪いかの切分け面倒。→確実に動く機能をデバイスで見つける

## マイコン間の通信

### ○ 調歩同期式通信

#### ◇特徴

- ・1本の線で、送信側から受信側に送る。
- ・クロックなし(速度は送受側の双方で一致させる)

#### ◇バリエーション

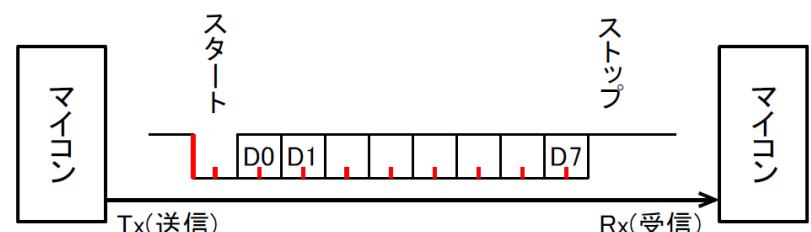
- ・いわゆるシリアルポート、RS-232C(D,E)
- 1対1、信号電圧±数V、もしくは0-5, 0-3.3
- ・RS422 1対1、高速、差動信号 **UAR**
- ・RS485 1対多、多対多に対応した**422T**

## マイコン間の通信

### ○ 調歩同期式通信

#### ◇通信の原理

- ・1ビットあたりの速度を送受で合わせる。
- ・スタートビットを検出して、データの取り込みタイミングを決定する。

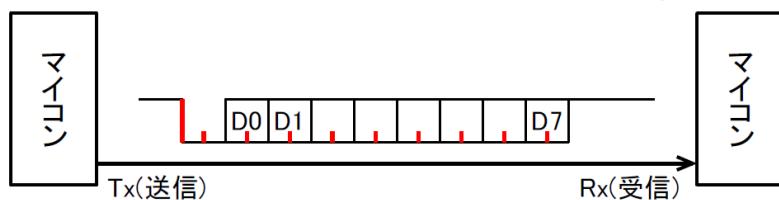


## マイコン間の通信

### ○ 調歩同期式通信

#### ◇特徴

- ・ほとんど全てのコンピュータが持つ機能  
(マイコン、パソコン等)
- ・USBから変換する機器多数。
- ・手頃な速度 (無難には ~115.2kbps)

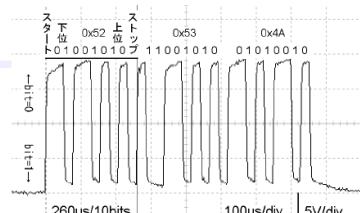


## マイコン間の通信

### ○ 調歩同期式通信

#### ◇補足メモ

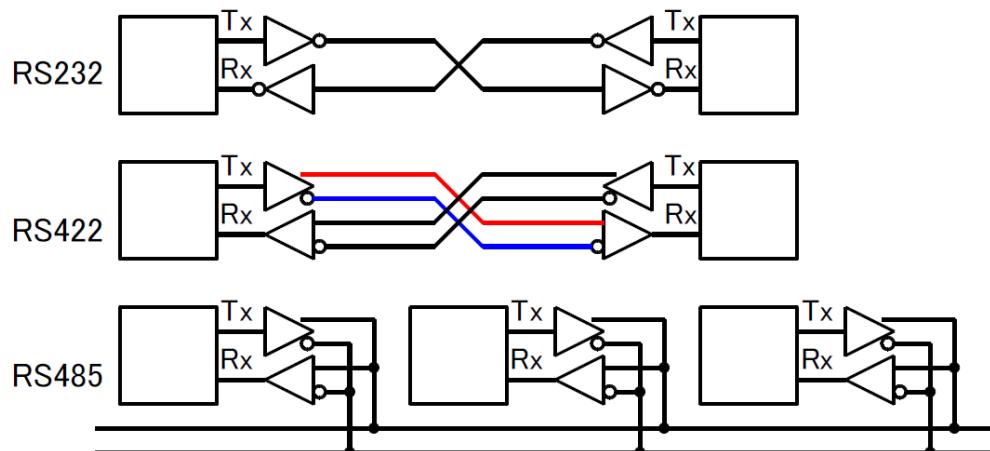
- ・送受信のクロック(タイミング設定)に誤差があると通信エラーになる、5%以内にする→ものによって高速時に誤差が出やすい。  
※メインのクロックを1/nしてつくるため
- ・RS422,485は終端抵抗が必要。
- ・速度設定はオシロスコープでわかる。



## マイコン間の通信

### ○ 調歩同期式通信

#### ◇バリエーション

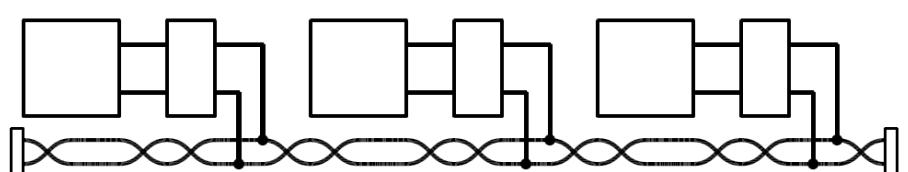


## マイコン間の通信

### ○ CAN (Controller Area Network)

#### ◇特徴

- ・多対多のリアルタイム通信を前提。  
衝突回避、送信の優先度
- ・**クロック無し、バス接続。**
- ・差動信号(平衡信号)による耐ノイズ性。



## マイコン間の通信

### ○ CAN (Controller Area Network)

#### ◇補足メモ ※ほぼ未経験

- ・多対多で伝統的なRS485に比較して、通信の方式まで定めている
- リアルタイム制御を志向している
- ※優先度、データ長の短縮
- 速度控えめ(485:10Mbps越え/CAN:1Mbps程度)
- という違いが見られる。

## マイコン～パソコン間の通信

### ○ USB

#### ◇特徴

- ・(比較的)高速に、多数のデバイスを接続。
- ・ホスト(制御する親)とデバイス(接続する機器)
- ・デバイス機能を内蔵したマイコンは多数。
- ・ホストには複雑な処理が求められるため、小型のマイコンでは無理。
- 最近はホスト機能を持つ小型マイコンも。  
例) PIC24の一部、RX621など

## マイコン～パソコン間の通信

### ○ USB

#### ◇使い方

- ・ハードは対応マイコンに線をつなぐ程度。
- ・動作が複雑なため、**ソフトの難易度が高い**。
- ・USB対応の機器をつくるには
  - ・フルに書く+デバドラをつくる
  - ・何かを偽装する+汎用ドライバ
  - ・**USB-シリアル変換IC**をつかう
- ※FTDI社 FT2232,4232 など

## マイコン～パソコン間の通信

### ○ イーサネット Ethernet

#### ◇特徴・使い方

- ・いわゆる「ネットワーク」(の一つ)で汎用性高。
- ・高性能マイコンでは機能を内蔵しており、多少の外付けで、ハードはできる。
- ・「ネットワーク通信」(後述)をさせるには、その上の**プロトコルのコード群**が必要で、“すべて自作”は困難。
- ・基本的にOS上で使用。

# 無線系の通信

## ○ ZigBee

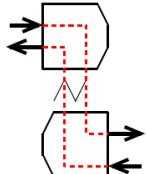
### ◇特徴・使い方

- ・センサネットワーク用に、低速ながら低消費電力を想定した規格。
- ・メッシュネットワークが可能で、直接通信ができなくとも、ネット経由で情報送受ができる。→広範囲の通信
- ・コーディネータ(ネットに1個必須)、ルータ(転送)、エンドデバイス、の3種

# 無線系の通信

## ○ XBee

### ◇使い方



- ・3.3V動作、シリアルポートも。
- ・最低限、電源とシリアルをつなぐと動作。
- ・パソコン上のX-CTUソフトで設定。
- ・送受アドレスを設定すると単に通信経路を無線化できる。
- ・シリーズ2はコーディネータが必須。(ファームの書き換えで設定)

# 無線系の通信

## ○ XBee

### ◇特徴

- ・Digi社の無線モジュール基板
- シリーズ1：独自通信規格
- シリーズ2：ZigBee仕様
- ・少しの設定で  
モジュールでシリアル通信を無線化  
モジュールの汎用入出力を遠隔操作  
などが可能。



Digi社サイトより

# 無線系の通信

## ○ Bluetooth

### ◇特徴・使い方

- ・省電力で短距離の周辺機器通信でよく使われている。
- ・ただし、制御系での使用例などはあまり見かけない。
- ・ホビー系の使用例：  
Wii用のコントローラ、ボードをPCで使用  
マイコンでUSBホスト+BTドングルで通信

Bluetooth Low Energyの実用化が進んでいます

2本線で複数のデバイスをつなげられて拡張性がよい

## 高機能シリアル通信I<sup>2</sup>C

# I<sup>2</sup>Cの実際 リアルタイム・クロックICの利 用実習

### 6-1 マイコンとディジタル・センサや通信モジュール・デバイス間のシリアル通信

PCなどのCOMポートとの間で行われるシリアル通信以外にも、図6-1に示すような各デバイスとの間で行われるSPI、I<sup>2</sup>Cのシリアル通信があります。これらのシリアル通信機能についてもArduino IDEの標準ライブラリが用意され、これらの通信仕様の基本的な知識を得るだけで、Arduinoとこれらの通信を行うデバイスとデータのやりとりを行ったり、デバイスの制御を行うことができるようになります。ライブラリは章末のAppendix2で各機能について説明しました。

本章では、I<sup>2</sup>Cの通信について確認し、具体的に複数のセンサ・デバイスを利用できるスケッチを作成します。

#### 6-1-1 I<sup>2</sup>C (Wire)

I<sup>2</sup>Cは、フィリップス社から提案されたクロック信号線とデータ信号線の2本の通信線を使用し、多数のデバイスが通信することのできる通信規格です。I<sup>2</sup>Cは、Inter-Integrated Circuitの頭文字IIC

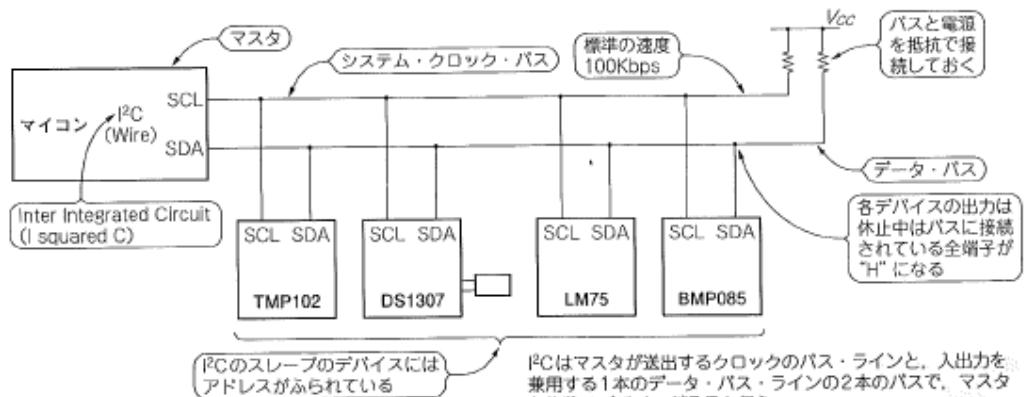


図6-1 Arduinoで利用できるI<sup>2</sup>C

I<sup>2</sup>Cはマスターが送出するクロックのバス・ラインと、入出力を兼用する1本のデータ・バス・ラインの2本のバスで、マスターと複数のデバイスが通信を行う。  
通信の制御はマスターが行い、デバイスのアドレスで通信相手を指定し、それぞれアドレスを指定した相手と交互に通信する。

出典：神崎康宏著

Arduinoで計る、計る、量る CQ  
出版社

がI<sub>2</sub>Cと表され、アイ・スケア・シーとも呼ばれます。また、Arduinoではこの通信方式をTWI(線式通信)と呼び、そのライブラリはWireとなっていますが、I<sub>2</sub>Cインターフェースの仕様と同じです。

I<sup>2</sup>Cの通信路は、図6-2に示すようにプルアップ抵抗により電源電圧に引き上げられています。I<sup>2</sup>Cに接続されているデバイスは、「マスタ」とマスタに制御されて通信を行う「スレーブ・デバイス」に分かれます。データの信号線は1本ですが、データの送受信のタイミングを決めるクロック・ラインが別に1本用意されています。このクロックはマスタから出力され、スレーブが出力することはありません。

◆スレーブにはアドレスが割り振られている

スレーブには7ビットのアドレスが割り振られます。I<sub>C</sub>に対応したデバイスには、このアドレスが割り振られています。このアドレスはデバイスのデータシートで確認することができます。

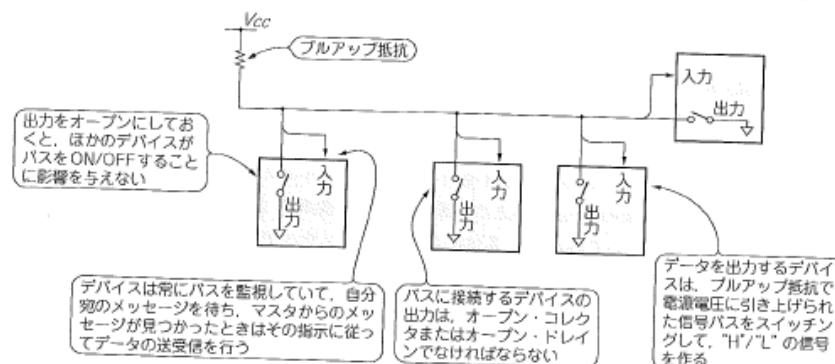


図6-2 I<sup>2</sup>CでバスはワイヤードOR

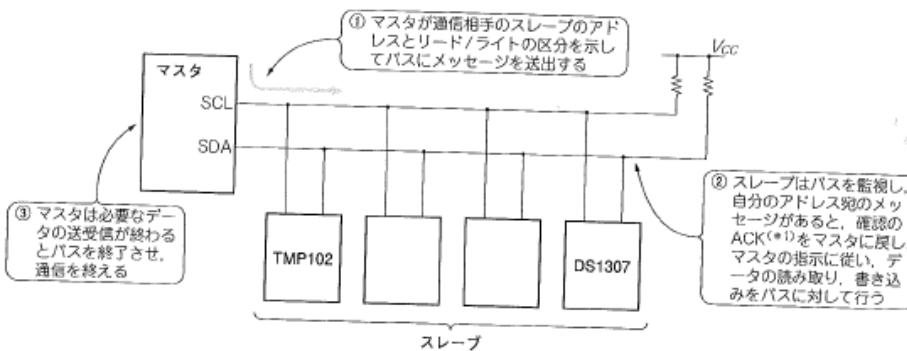


図6-3 マスタはスレーブのアドレスを指定してそれぞれ個別にスレートを送信する。

(※1) ACK (ACKnowledge) は、データ通信を行うときに送信したデータが正しく受信側で受け取ったことを送信側に返す返事処理のこと

スレーブ・デバイスはピンの数が多くありませんが、その中でピン設定によって複数のデバイス・アドレスを設定できるようになっているものもあります。

スレーブ・デバイスには固有のアドレスが振られているので、バスの上に多くのデバイスをつないで利用できます。接続するデバイスの数が増えてもArduinoに必要なピン数は増えません。第7章で説明するSPIでは、I<sup>2</sup>Cより高速にデータのやりとりができますが、デバイスごとにセレクト信号が必要になるので、ArduinoにはたくさんのSPIデバイスをつなぐことができません。

- ◆ 通信はマスターとスレーブ間で行われる

図6-3に示すように、マスターからスレーブに送信先スレーブのアドレスを送信し、該当のアドレスをもったスレーブ・デバイスにマスターからのデータの受信またはマスターへのデータの送信を要求します。スレーブは當時バス上のデータを監視していて、自分宛のメッセージがあればその内容の指示に従い、マスターからのデータを受信したり、マスターへデータを送信します。

受信する場合、マスターからデータが送信されるので、指定されたスレーブはマスターからのデータを受信します。必要なデータの送信が終了したら、通信の終了がマスターから送信されます。受信側スレーブがマスターからの終了メッセージを受信すると、マスターからの受信セッションを終了し待機の状態に戻ります。

### 6-1-2 2本の信号線の組み合わせで通信手順を制御

I<sup>2</sup>Cでは図6-4に示すように、SDAとSCLが共にHIGH("H")のときはバスは解放されています。SCLが"H"の状態でSDAが"H"からLOW("L")に変わると、スタート・コンディションとしてI<sup>2</sup>Cのセッションが開始します。スレーブは、自分のアドレスが送信されてくるか監視を開始します。SCLの信号の"L"に合わせてSDAの信号を読み取ります。

通信処理が終わると、マスターはSCLが“H”的ときにSDAを“L”から“H”にして、ストップ・コンディションとしてバスを解放します。

会員登録

PCの開始は、マスターによるスタート・コンディションに続けてマスターから図6-5に示すように通

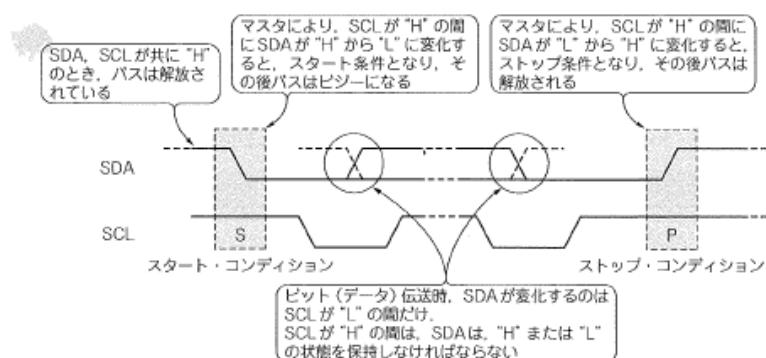


図6-4 スタート・コンディションとストップ・コンディション  
I<sup>2</sup>Cの通信は、スタート・コンディションで始まり、ストップ・コンディションで終わる。

信相手のスレーブの7ビットのアドレスと、このデータの方向を示すR/Wのビットを組み合わせた1バイトのデータがバスに送信されます。アセンブラー・プログラムを作る場合は、アドレスとR/Wビットを組み合わせた開始バイトの作成から始める必要がありますが、Wireのライブラリを使用するとスレーブのアドレスと読み書きのそれぞれの関数を用意することだけでスケッチが記述できます。

したがって、I<sup>2</sup>Cの基本的なやりとりさえ確認しておけば、細かなタイミングを考えることなくスケッチの作成であまり困ることはありません。

### 6-1-3 I<sup>2</sup>Cの送受信データの基本的なやりとり

I<sup>2</sup>Cの送受信データの基本的なやりとりは、図6-6に示すようになります。マスタからスレーブにデータを書き込むときは、図6-6(a)に示すようにスタート・コンディションに続いて開始バイト、データをマスタからスレーブに書き込みます。

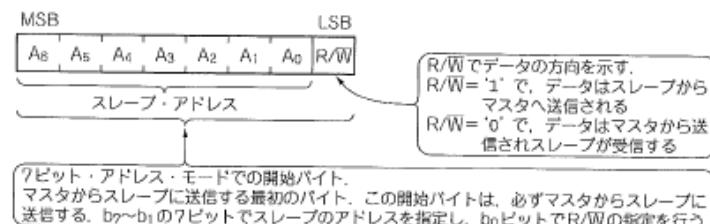


図6-5 I<sup>2</sup>Cで最初にマスタから送出される開始バイト

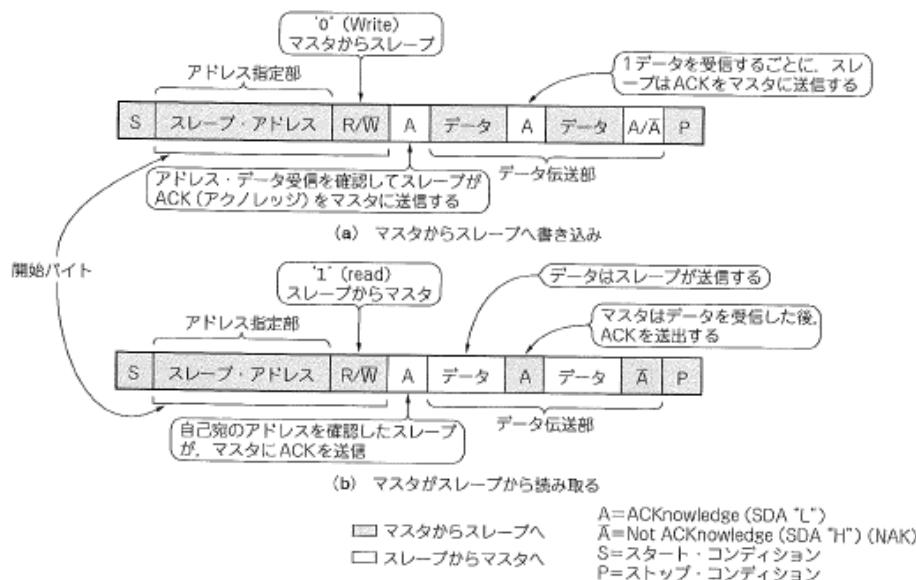


図6-6 I<sup>2</sup>Cの送受信データの基本的なやりとり

### ◆スレーブからのデータをマスタが受け取るとき

マスタがスレーブからデータを読み込む場合は、図6-6(b)に示すようにアドレス指定部をマスタがバスに送信すると、バス上のアドレス指定部を監視し自分宛のメッセージを見つけたスレーブがマスターへデータを送信します。マスターから正しくデータが受信できることを示すACKコードを受信するたびに次のデータを送信し、マスターからNAK(\*2)コードが送られてくるまで続けます。

この通信制御のため、マスターとスレーブのデバイス間で交換される制御データのピットごとの状態の確認などが必要になります。しかしArduinoのマイコン・ボードに採用されているマイコン・デバイスはこの厄介な処理を行う機能を内蔵していて、Arduinoのマイコン・ボードのアナログ・ポート4番と5番がI<sup>2</sup>C(Wire)の信号ピンとして用意されています。また、Wireライブラリが用意されていますから、わかりやすいスケッチの命令を使ってI<sup>2</sup>C(Wire)の処理が記述できます。

### 6-1-4 Wireライブラリを使用すると細かい手順は知らなくても済む

Appendix2に示すWireライブラリの命令を使用すると、I<sup>2</sup>Cのスケッチを容易に作成することができます。シンプルな例として、リアルタイム・クロックDS1307、温度センサTMP102からデータを読み取る方法を説明します。

## 6-2 I<sup>2</sup>Cインターフェースでやりとりするリアルタイム・クロック

最初に、ArduinoでI<sup>2</sup>Cインターフェースをもつリアルタイム・クロックを使用する事例を説明します。正確な時刻がわかると、センサから取得したデータを記録するとき、日時とともに保存デバイスに書き込むことができます。

### 6-2-1 現在時刻を知るには

PCを利用しているときは、PCに内蔵されたタイマで現在の時刻を知ることができます。Arduinoの本体はリアルタイム・クロックをもっていません。Arduinoで測定したデータを記録するには、何時測定したかも重要な記録事項です。そのため、測定時の時刻を知るための方法として図6-7に示すようないくつかの方法があります。

#### (1) PCと接続している場合

- ① 常時PCと接続していてPC側でデータも記録する場合は、PC側で時刻を付加して記録する。Arduinoは測定データを送信するだけで済む。
- ② 測定時に、PCからの時刻のデータを読み取り、測定データに時刻を付加しArduinoのシステム側に保存することもできるようになる。

#### (2) PCと接続していない場合

- ① リアルタイム・クロック・モジュール

PCと接続していない場合は、Arduinoのシステム側にリアルタイム・クロックのモジュールを用意し、測定したデータに時刻データを付加し保存することができる

(\*2) NAK(Negative Acknowledge)は、通信内容などにエラーがあり、要求が受けつけられないことを示す。

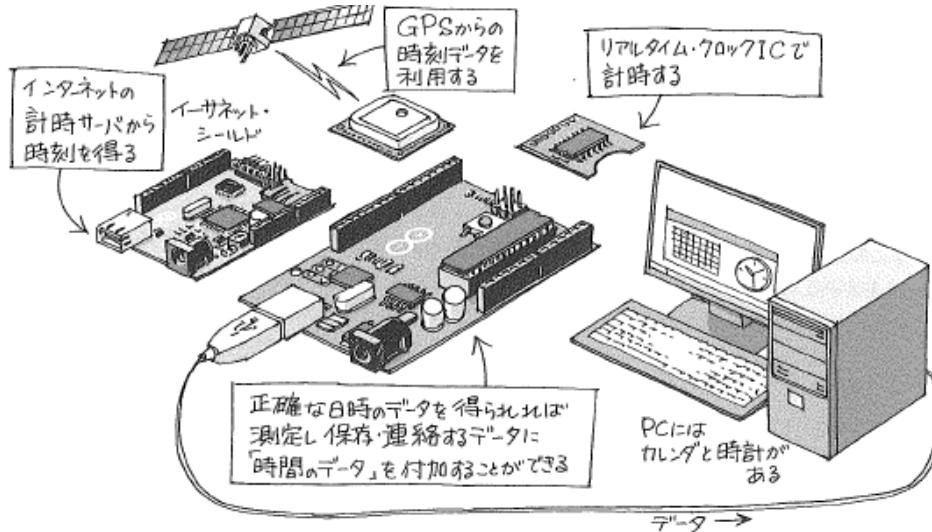


図6-7 PCで日付、時刻を得る方法

#### ② GPS衛星から時刻を得る

GPS受信モジュールを利用し正確な時刻を得る。位置の検出を行うのでなければ衛星の補足の数が少なくて済み、室内でも時刻データ得ることができる場合が多い。

これらの方が考えられます。ここでは、次に示すリアルタイム・クロック・モジュールRTC1307を使用することにします。

このモジュールはSparkfun製のモジュールで、DS1307というICを搭載しI<sup>2</sup>Cのインターフェースをもっています。バックアップ電源としてCR1225電池を搭載しているので、長期(9年以上)にわたって、Arduino側の電源と無関係に時刻を刻むことができます。この他にもI<sup>2</sup>CのインターフェースをもったArduino用のセンサも多くあるので、アナログ入力A<sub>5</sub>、A<sub>4</sub>の2本のピンで二つ以上のデバイスを接続できます。I/Oピンのあまり多くないマイコンでは助かります。

### 6-2-2 RTCモジュール(DS1307)

RTCモジュールは、図6-8(a)に示すように8ピンのDS1307と水晶振動子で構成されています。このICは5Vの動作電圧で、主な仕様は表6-1に示すようになります。電源の5VとGNDの電源端子、SDA、SCLのI<sup>2</sup>Cの端子と1Hz～32.768kHzのクロックを出力することもできるSQW端子が用意されています。

このRTCモジュール基板の端子の穴は2.54mmピッチのピン・ヘッダが利用できるので、このモジュールにピン・ヘッダをはんだ付けし、ブレッドボードやユニバーサル基板で容易に利用できます。

動作電源が5Vなので5V電源のArduinoと直接接続することができます。このリアルタイム・クロック・モジュールは共立エレショップまたはスイッチサイエンスなどから入手することができます。

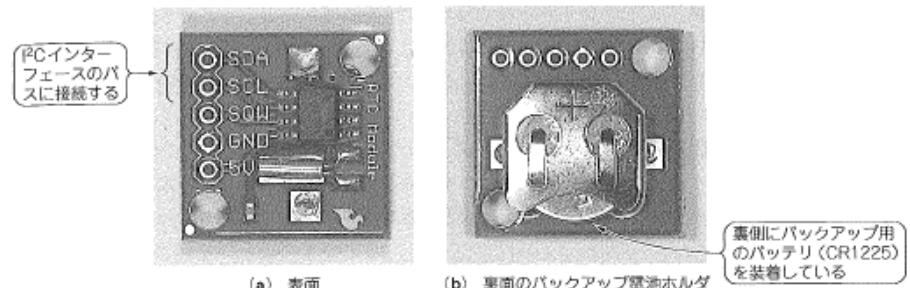


図6-8 DS1307と水晶振動子が実装されたRTCモジュール

表6-1 RTC1307の主な動作仕様

	min	max	
動作電源電圧( $V_{cc}$ )	4.5V	5.5V	動作時
Logic 1 (HIGH)	2.2V	$V_{cc} - 0.3V$	
Logic 0 (LOW)	-0.5V	+0.8V	
バッテリ電圧	2.0V	3.5V	待機時
SCL クロック		100kHz	I <sup>2</sup> C クロック

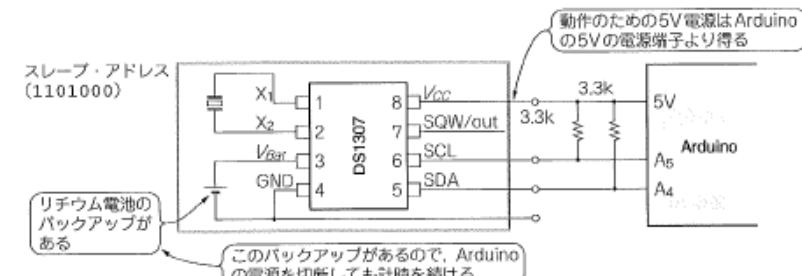


図6-9 ArduinoにDS1307を接続する

基板の背面には、図6-8(b)に示すようにCR1225のリチウム電池によるバックアップ電源も用意されています。

#### ◆ DS1307の設置方法

DS1307の動作電源電圧は5Vです。そのため標準の5V動作のArduinoとDS1307との接続は図6-9に示すようになります。Arduinoのアナログ入力ポート5番(SCL)、4番(SDA)をDS1307のSCL端子、SDA端子に接続します。アナログ入力ポートですが、Wireライブラリを利用するときは、この二つのポートがデジタル信号でデータをやりとりするI<sup>2</sup>Cのバスのポートとして割り当てられています。

モジュールにピン・ヘッダをはんだ付けして、ブレッドボードにセットしてテストすることもできるようになります。使用するときは、板配置でなくArduino用のユニバーサル基板を使用し、DS1307のリアルタイム・クロック・モジュールを取り付けるための5ピンのピン・ソケット、I<sup>2</sup>Cバスのプルアップ抵抗3.3kΩ 2本を基板に取り付け、はんだ付けします。

このDS1307のリアルタイム・クロック・モジュールは電源電圧が5Vですから、信号線をつなぐときArduinoとの間にレベル・コンバータを挿入することなく接続することができます。3.3V電源のI<sup>2</sup>Cのセンサ類を追加する場合、例えば電圧レベル・コンバータのPCA9306を追加します（後述）。

### 6-2-3 DS1307の使い方

ArduinoとI<sup>2</sup>Cのスレーブ・デバイスとのデータの読み書きの方法について、DS1307を例に確認します。図6-10にスレーブからのデータの読み込みの基本的な手順を、図6-11にスレーブへデータを

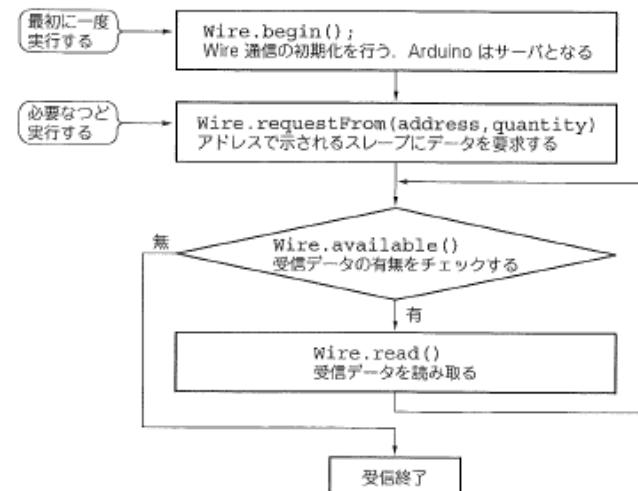


図6-10 スレーブからデータを読み込む

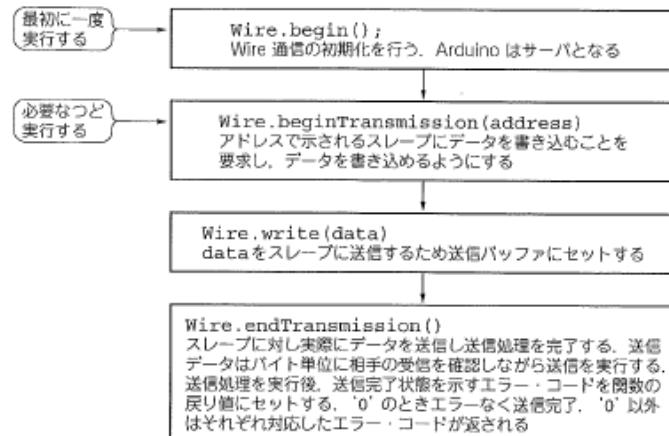


図6-11 スレーブへデータを出力する

出力する基本的な手順を示します。Wireのライブラリを使用する場合は、最初にWire.begin()でWire通信の初期化を行う必要があります。この処理はsetup()関数の中で行います。

#### ◆ I<sup>2</sup>Cスレーブ・アドレス

DS1307のI<sup>2</sup>Cのスレーブ・デバイスとしてのアドレスは0x68(1101000)と定められています。このアドレスを指定して、データの書き込みのときは、

`Wire.beginTransmission(address)`

と送信相手のスレーブにマスタからの送信準備を促し、`Wire.write(data)`で送信データdataをスレーブに対して送信します。

### 6-2-4 DS1307の内部メモリ

DS1307は内部に64バイトのメモリ・レジスタをもつていて、表6-2に示すように00～07までのアドレスの8バイトに、秒、分、時、曜日、日付、月、年のデータとSQWのパルスの制御を行う制御データが割り当てられています。

#### ◆ データの読み書き

DS1307のメモリ・レジスタの読み書きは、図6-12に示す手順によって行います。マスタからこのDS1307のメモリ・レジスタの読み書きの対象となるのは、DS1307内の「メモリ・レジスタのアドレス」を示すワード・アドレスが指示するメモリ・レジスタです。

#### ◆ メモリ・レジスタの設定

メモリ・レジスタの設定は、マスタからのI<sup>2</sup>Cの書き込みサイクルの最初に書き込まれたデータがワード・レジスタの値となります。具体的にスケッチで記述すると、次のようにになります。

DS1307に対して、送信セッションを開始し、読み書きをしたいメモリ・レジスタのアドレスをワード・レジスタに書き込みます。

`Wire.beginTransmission(デバイス・アドレス);`

`Wire.write(ワード・レジスタ);`

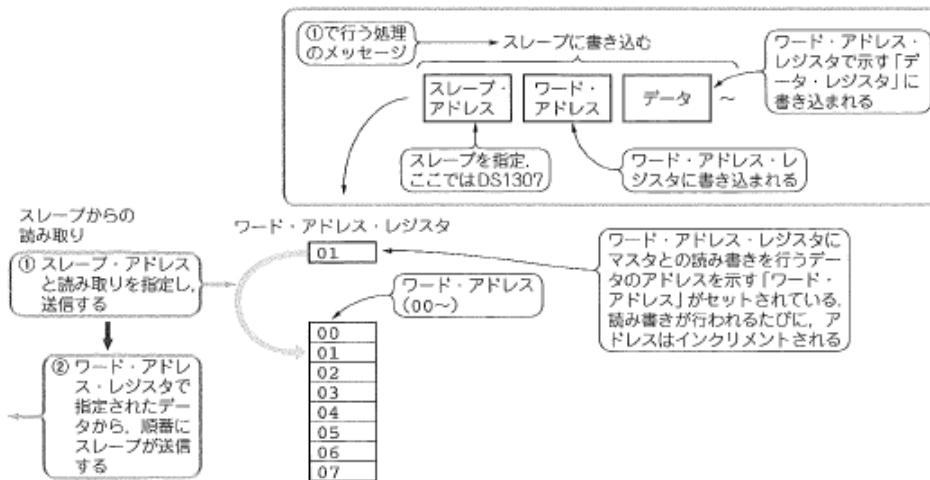
`Wire.endTransmission();`

マスタからの送信セッションが確立し、最初にDS1307に書き込まれたデータはメモリ・レジスタのワード・アドレスと解釈され、ポインタ・アドレスとして設定されます。続いて書き込み処理を行うと、ワード・アドレスが示すメモリ・レジスタにデータが書き込まれます。データを書き込むと

表6-2 DS1307のレジスタ

アドレス(値)	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
00 (00～59)	CH		秒(10位)			秒(1位)		
01 (00～59)	0		分(10位)			分(1位)		
02 (01～12/00～24)	0	12/24	時P/A <sup>(注)</sup>	時間(10)			時間(1位)	
03 (1～7)	0	0	0	0	0			曜日
04 (1～28, 29, 30, 31)	0	0		日付(10位)			日付(1位)	
05 (01～12)	0	0	0	月(10)			月(1位)	
06 (00～99)			年(10位)				年(1位)	
07 制御データ	OUT	0	0	SQWE	0	0	RS <sub>1</sub>	RS <sub>0</sub>

\* 24時間表示：時間(10位)/12時間表示：PM/AM



もにワード・アドレスはインクリメントされます。そのため、連続してデータを書き込むとメモリ・レジスタに順番にデータを書き込むことができます。

#### ◆ メモリ・レジスタの読み取り

DS1307からデータを読み出すときも、同様にワード・アドレスが示すメモリ・レジスタの値が読み取られ、そのたびにワード・アドレスがインクリメントされます。特定の読み出すレジスタを指定する場合は、最初にワード・アドレスを書き込んでワード・アドレスの値を指定した値にしてから、その後に読み込み処を行います。

#### 6-2-5 DS1307のメモリ・レジスタの内容を確認する

DS1307を利用したリアルタイム・クロック・モジュールのArduinoへの接続を図6-9のように終えました。まず、DS1307のメモリ・レジスタの内容を読み取って確認してみます。

まず、ワード・アドレスを00に設定して、クロック、カレンダ・データのほかに出力クロックの制御データも含めて8バイトのデータを読み出します。そして、読み取ったデータをシリアル通信でPCに送信し、Arduino IDEのシリアル・モニタで読み取ったデータの確認を行います。

そのためにスケッチを次のように作りました。

#### ◆ Wireライブラリのインクルード、変数定義と初期化部

図6-13(a)にライブラリの読み込み、スレーブ・アドレスの設定、初期化処理の部分を示します。Sketch>Import Libraryで表示したリストからWireを選択し#include <Wire.h>の文を取り込みます。その後、DS1307のI<sup>2</sup>Cのアドレスの値を定数DS1307\_ADDRESS=0x68と定義します。sec(秒)からyear(年)までと制御バイトの値を格納する変数を定義し、I<sup>2</sup>Cのマスターとして初期化およびシリアル通信の初期化を行っています。

```

ardens060010 | Arduino 1.0
File Edit Sketch Tools Help
ardens060010
#include <Wire.h>
int DS1307_ADDRESS=0x68; int val=0;
byte sec,minute,hour,day,week,month,year,ctrlb;
void setup(){
  Wire.begin(); // I2C (wire) の初期化
  Serial.begin(9600);
}
void loop(){
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire.write(val);
  Wire.endTransmission();
  Wire.requestFrom(DS1307_ADDRESS,8);
  sec=Wire.read();
  minute=Wire.read();
  hour=Wire.read();
  week=Wire.read();
  day=Wire.read();
  month=Wire.read();
  year=Wire.read();
  ctrlb=Wire.read();
  Serial.print("sec=");
  Serial.println(sec,HEX);
  Serial.print("minute=");
  Serial.println(minute,HEX);
  Serial.print("hour=");
  Serial.println(hour,HEX);
  Serial.print("week=");
  Serial.println(week,HEX);
  Serial.print("day=");
  Serial.println(day,HEX);
  Serial.print("month=");
  Serial.println(month,HEX);
  Serial.print("year=");
  Serial.println(year,HEX);
  Serial.print("ctrlb=");
  Serial.println(ctrlb,HEX);
  delay(2000);
}

Done uploading
Binary sketch size: 4090 bytes (of a 32256 byte maximum)

```

図6-13 DS1307のメモリ・レジスタの内容を確認するスケッチ

```

#include <Wire.h>
CONST int DS1307_ADDRESS=0x68; int val=0;
byte sec,minute,hour,day,week,month,year,ctrlb;
void setup(){
    Wire.begin();
    Serial.begin(9600);
}

```

#### ◆データの読み込み

図6-13(b)にDS1307からデータを読み込む部分のスケッチを示します。Wire.beginTransmission(DS1307\_ADDRESS)からWire.endTransmission();までの3行でワード・アドレスを00にセットし、次のデータの読み込み処理でメモリ・レジスタのアドレス00から読み込めるようになります。

次のWire.requestFrom(DS1307\_ADDRESS, 8);でDS1307に8バイトのデータをマスター(Arduino)へ送信することを要求します。sec=Wire.read();から各変数にDS1307から読み取った値をセットします。8バイトのデータを読み取り終えるとスレーブとのセッションを終了します。

読み込んだデータは、図6-13(c)に示すようにシリアル通信で見出しをつけて、シリアル・モニタに表示します。Serial.println(sec,HEX);のHEXは16進数表示を指定します。最後のdelay()関数は、読み取り間隔を調整するために用意しています。

スケッチをアップロードして実行結果をシリアル・モニタに表示したようすを図6-14に示します。最初のバイトは0x80となっています。b<sub>7</sub>のCH(clock halt)ビットが'1'でクロックが停止しています。そのため、0年1月1日0時0分0秒の初期値のまま時間が経過しても変化しません。

クロックの停止を解除するために、setup()処理の中で次に示すCHをクリアするスケッチ文を追加します。

1回目のWire.write(val);はワード・アドレスの設定で、次のWire.write(val);で

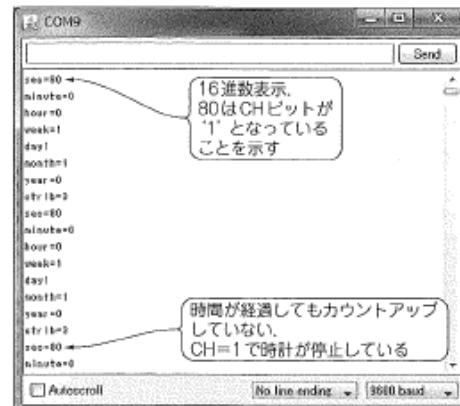


図6-14 実行結果一カウントしていなかった



図6-15 CH=0にすると計時を始める

アドレス0x00メモリ・レジスタに0x00を書き込み、CHをクリアします。初期化ルーチンなので、秒の値もゼロ・クリアしてあります。

```

Wire.beginTransmission(DS1307_ADDRESS);
Wire.write(val); // val=0
Wire.write(val);
Wire.endTransmission();

```

このスケッチをsetup()処理に追加し、Arduinoにアップロードして実行した結果を図6-15に示します。secの値がカウントアップしているのが確認できます。

#### 6-2-6 時刻と日付を設定するDS1307のメモリ・レジスタの内容を確認する

次に、時刻と日付をそれぞれ設定するスケッチを関数として定義します。時刻の設定は、日付の設定よりも多く利用する機会がありそうなので、図6-16に示すようにそれぞれ別の関数とします。

これらの関数は図6-17に示す手順で処理を行います。シリアル・ポートからのデータの有無を確



図6-16 リアルタイム・クロックのセットのための関数

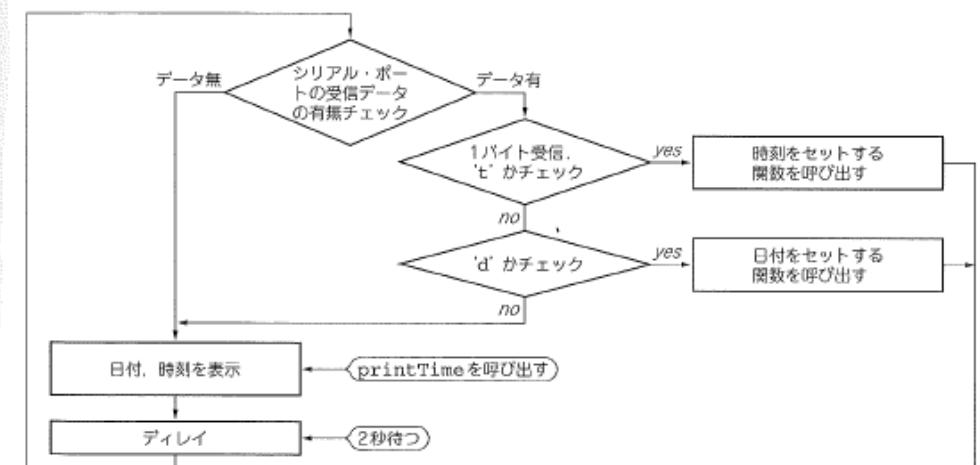


図6-17 リアルタイム・クロックの設定表示の処理フロー

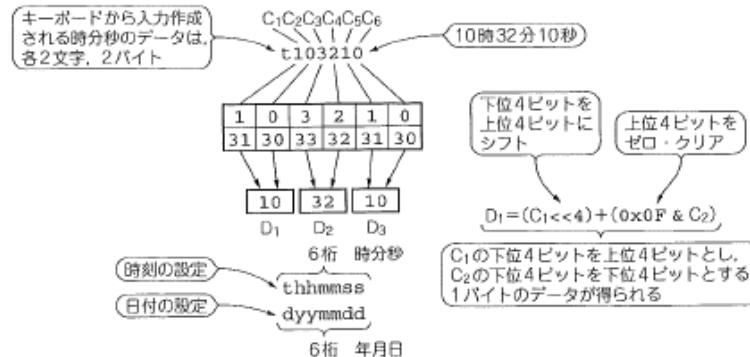


図6-18 時刻、日付の設定コマンド

認し、受信データがあればtかdの文字を確認し、これらの文字があればそれぞれの文字に対応した関数を呼び出します。受信文字がなければリアルタイム・クロックからデータを読み取って、表示する関数を呼び出します。その後、ディレイを置き最初に戻り、繰り返します。

#### ◆ 時刻、日付はPCよりシリアル・モニタ経由で送信

時刻、日付は図6-18に示すフォーマットでPCのシリアル・モニタから入力します。時間の場合はt、日付の場合はdの文字を先頭にコマンドとして付加します。シリアル・モニタからの受信データをチェックし、コマンドの文字を検出したら時刻セット関数、日付セット関数を呼び出します。コマンド以外の文字の場合は、何もしないで次に進みます。セットするデータのシリアル・ポートからの読み取りはそれぞれの関数の中で行います。

#### [例]

```
t153000 ..... 15:30:00をセットする  
d110528 ..... 2011年5月28日をセットする
```

#### ◆ 文字データを数値に変換する

コマンドのチェックは、

```
if(Serial.read() == "t")
```

でチェックすることができます。一方、シリアル・ポートから入力された時間の2バイトのデータ15は、「1」の文字コード0x31と「5」の文字コード0x35で構成されています。DS1307に設定する15時の設定値データは、1バイトの0x15となります。コマンドで指定されたそれぞれ2バイトの時分秒のデータは、1バイトのデータに変換する必要があります。そのため変換処理は次のようにします。

```
byte hour=(Serial.read() <<4);  
hour=hour + (Serial.read() & 0x0F);
```

同様に分、秒も次のように変換します。

```
byte minute=(Serial.read() <<4);  
minute=minute + (Serial.read() & 0x0F);  
byte sec =(Serial.read() <<4);  
sec=sec+ (Serial.read() & 0x0F);
```

以上でセットされた時、分、秒の値は、4ビットごとに0～9の値が割り当てられたBCD<sup>(\*)3</sup>コードでセットされた日時、時刻などのデータになります。そして、この値はDS1307の各メモリ・レジスタにセットする値として利用できます。

#### ◆ 時刻をセットする関数

次に、時刻をセットする関数 setTime()を示します。最初の6行のスケッチで時、分、秒のデータをシリアル・ポートから読み取ります。

```
void setTime(){  
    byte hour=(Serial.read() <<4);  
    hour=hour + (Serial.read() & 0x0F);  
    byte minute=(Serial.read() <<4);  
    minute=minute+(Serial.read() & 0x0F);  
    byte sec=(Serial.read() <<4);  
    sec=sec+(Serial.read() & 0x0F);
```

次の処理で、書き込むアドレスを00からはじめるように00を書き込みます。続いて設定データの値を書き込みます。

```
Wire.beginTransmission(DS1307_ADDRESS);  
Wire.write(val);  
Wire.write(sec);  
Wire.write(minute);  
Wire.write(hour);  
Wire.endTransmission();  
}
```

以上で、時刻の設定を終えます。

#### ◆ 日付の設定

日付の設定も同様に行っています。関数名はsetDay()です。day\_of\_weekは曜日を示すコードで、1が日曜日で月曜日から順番に割り振られ土曜日が7になります。シリアル・ポートから設定データを読み取った後は、曜日の設定アドレス03を書き込み、その後、設定データをWire.write関数でDS1307に書き出します。

```
void setDay(){  
    byte year=(Serial.read() <<4);  
    year=year+(Serial.read() & 0x0F);  
    byte month=(Serial.read() <<4);  
    month=month+(Serial.read() & 0x0F);  
    byte day=(Serial.read() <<4);  
    day=day+(Serial.read() & 0x0F);  
    byte day_of_week=(Serial.read() & 0x07);  
    Wire.beginTransmission(DS1307_ADDRESS);  
    Wire.write(0x03);  
    Wire.write(day_of_week);  
    Wire.write(day);  
    Wire.write(month);
```

(\*3) BCD : Binary coded decimal. 2進化10進表示。

```

Wire.write(year);
Wire.endTransmission();
}

```

#### ◆ 日付時刻の表示を行う関数

現在時点の日付、時刻を読み取りシリアル・ポートに表示する関数はprintTime()です。メモリ・レジスタの00のアドレスから表示データを読み取り、そのままHEX表示で表示しています。特に新しいことはありません。

```

void printTime(){
    Wire.beginTransmission(DS1307_ADDRESS);
    Wire.write(val);
    Wire.endTransmission();
    Wire.requestFrom(DS1307_ADDRESS, 7);
    byte r_sec=Wire.read();
    byte r_minute=Wire.read();
    byte r_hour=Wire.read();
    byte r_day_of_week=Wire.read();
    byte r_date=Wire.read();
    byte r_month=Wire.read();
    byte r_year=Wire.read();
    Serial.print(r_year,HEX);
    Serial.print("/");
    Serial.print(r_month,HEX);
    Serial.print("/");
    Serial.print(r_date,HEX);
    Serial.print(" ");
    Serial.print(r_hour,HEX);
    Serial.print(":");
    Serial.print(r_minute,HEX);
    Serial.print(":");
    Serial.print(r_sec,HEX);
    Serial.print(" smtwtfs= ");
    Serial.println(r_day_of_week,HEX);
}

```

この場合、HEXで表示していますから、BCDコードの設定データも同じ値に表示されます。時、分、秒の各値を通常の整数として扱う場合は、次のようにBCDコードからバイナリ・データに変換する必要があります。

bcd\_data .....メモリ・レジスタから読み取った値

b\_data .....変換されたバイナリ・データ

b\_data = (bcd\_data/16) \* 10 + bcd\_data&0x0F

(bcd\_data/16) \* 10で10位の桁の値が求まり、bcd\_data&16で1位の桁の値が求められます。

#### ◆ メインのスケッチの部分

シリアル・ポートからの受信データを確認し、“t”, “d”かをチェックしています。コマンドを増やす場合は、if文とコマンドの処理を行う関数を追加します。

```

#include <Wire.h>
int DS1307_ADDRESS=0x68;int val0=0;
byte command;

```

ここに、上記のsetTime(), setDay(), printTime()の三つの関数を記述します。

```

void setup(){
    Serial.begin(9600);
    Wire.begin();
    Serial.flush();
}
void loop(){
    if(Serial.available()){
        command=Serial.read();
        if (command==0x74){ // "t"=0x74
            setTime();
        }
        Serial.println(command);
        if (command==0x64){ // "d"=0x64
            setDay();
        }
    }
    printTime();
    delay(2000);
}

```

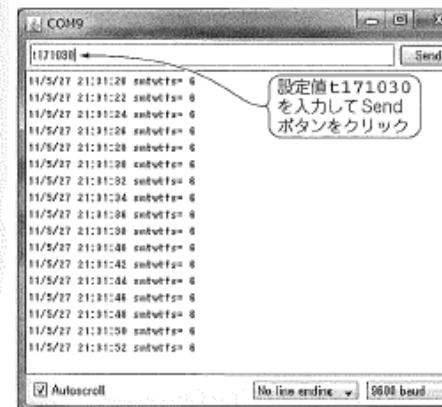


図6-19 送信部にコマンドを入力

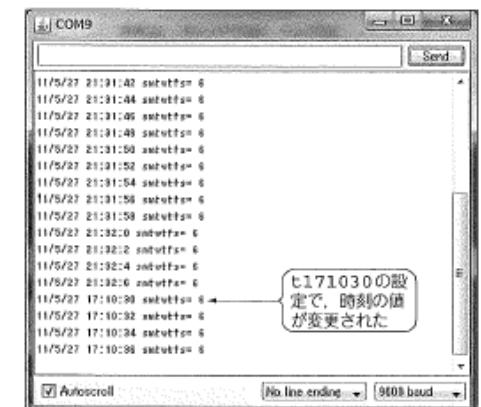


図6-20 コマンドが送信されリアルタイム・クロックの設定が変更された

## 加速度センサーとは？

by Tomoaki Tsuzuki

## 加速度センサーとは？

加速度センサーとは加速度の測定を目的とした慣性センサーです。振動センサーと異なり、加速度センサーは直流(DC)の加速度が検出可能である為、加速度センサーを使って重力を検出する事も可能です。加速度を測定し適切な信号処理を行う事によって、傾きや動き、振動や衝撃等様々な情報が得られます。加速度センサーには、様々な種類があり加速度の検出方式によって大別されます。本稿では MEMS (Micro Electro Mechanical System) 技術を応用した MEMS 加速度センサーの紹介をします。

## MEMS 加速度センサーの歴史

近年のマイクロマシン技術または MEMS 技術の発達により、半導体微細加工技術を応用した加速度センサーが大量かつ安定的に生産出来るようになりました。アナログ・デバイセズは1993年に初の MEMS 加速度センサーADXL50 の量産を開始しました。ADXL50 はパッケージに平行な 1 軸の直線加速度を検出できるアナログタイプの加速度センサーで、必要な信号処理回路の殆どをチップ内部に内蔵しています。ADXL50 は外付け回路を簡略化する事が出来、またそれまでの機械式加速度センサーに比べて小型で安価であったため、自動車のエアバッグシステムに衝撃を検出するセンサーとして採用され、使用されるようになりました。

ADXL50 は  $\pm 5.0 \text{ g}$  (\*1) の検出範囲を持つセンサーで、車が衝突する際の大きな加速度を検出する事が出来ます。しかし ADXL50 はエアバッグの展開という比較的大きな加速度を検出する用途には最適ですが、傾きの検出や人間の動きといった比較的小さな加速度 ( $1 \text{ g} \sim 10 \text{ g}$  程度) を検出する用途には向いていません。この問題を解決する為にアナログ・デバイセズは ADXL202 をリリースしました。ADXL202 は測定範囲が  $\pm 2 \text{ g}$  と小さく、傾きの検出や人間の動きの検出に最適です。また、ADXL50 が 1 軸であったのに対しパッケージに平行な X 軸と Y 軸の 2 軸の検出機構をワンパッケージに内蔵した画期的な加速度センサーです。

ADXL202 はその小型パッケージと使い勝手の良さから、ゲーム機器やプロジェクター等多くの民生機器に搭載されました。

Rev. o

アナログ・デバイセズ株式会社

本 社 / 〒105-0091 東京都港区海岸 1-16-1 ニュービア竹芝サウスタワー

電話 03 (5402) 8200

大阪営業所 / 〒532-0003 大阪府大阪市淀川区宮原 3-5-36 新大阪 MT ビル 2 号

電話 06 (6350) 6868

$F = k \times x$

(2)

ここで、 $k$  はばね係数  $x$  はバネの伸縮距離です。式 (1) と式 (2) の連立方程式を解くと加速度  $a$  は以下の式で表す事ができます。

$$a = \frac{k \times x}{m}$$
 (3)

式 (3) より、加速度は既知のばね係数と重さを持った锤の移動距離を計測する事で計算が可能である事が解ります。図 1 は加速度の検出原理を図示したものです。

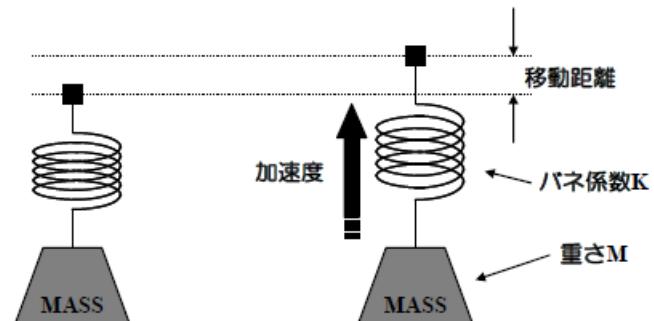
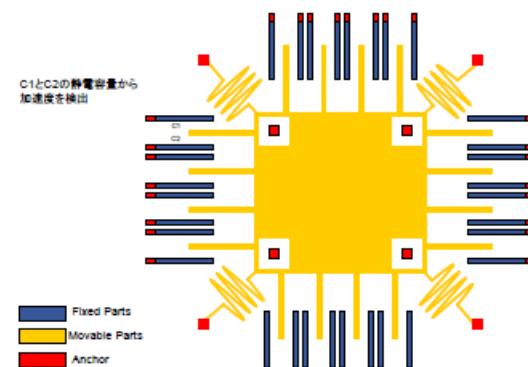


図 1. 加速度センサーの検出原理

アナログ・デバイセズの静電容量型加速度センサーは式 (3) を利用して、锤の移動距離を計測する事によってセンサーに加わっている加速度を出力するように設計されています。

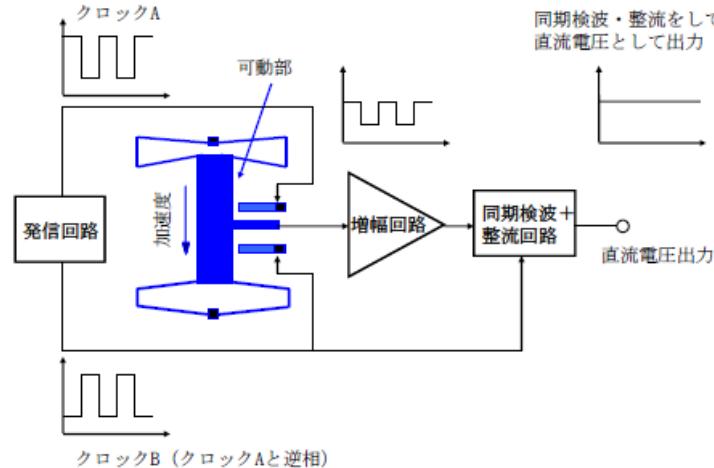
れています。図 2 は代表的な低 G 加速度センサーの椡出素子部の模式図です。



Rev. o | Page 4 of 8

検出素子部には加速度によって動く可動部（錐）とバネ、またその動き（移動距離）により静電容量変化を発生するための構造状電極が形成されており、可動マスに形成された可動電極1本当たりにつき2本の固定電極に挟まる形で電極の単位セルを形成しています。

図4に内部信号処理の流れを示します。単位セルの2本の固定電極にそれぞれ逆相のクロック信号を印加すること



で、加速度によって可動電極がどちらかの固定電極に近付いたときに、近付いた固定電極に印加されているクロックと同相の電荷変化が可動電極に発生します。この電荷変化を增幅し、同期検波および整流を行うことで可動部の移動距離、つまり加速度に比例した電圧出力を得る事ができます。

### 加速度センサーのアプリケーション

加速度センサーは大きく分けて重力・振動・動き・衝撃の3つの現象を測定する事が出来ます。それぞれの現象を旨く検出する事によって、加速度センサーの出力信号は実際のアプリケーションに役立てられています。例えば、携帯機器画面

の表示向きを使用環境に合わせて変更するアプリケーション（縦横検出）では、重力を計測して、加速度センサーの傾きを計算する事で実現する事が出来ます。図5に加速度センサーの検出対象と代表的なアプリケーションを図示します。

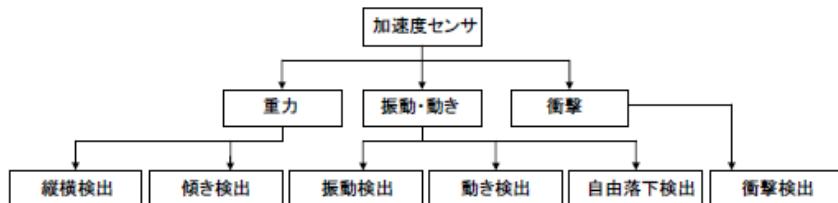


図6. 加速度センサーの検出対象とアプリケーション

### 加速度センサーの選定手順

最適な加速度センサーを選定する為には、まず実現したいアプリケーションを明確にする事が大切です。この時出来るだけ具体的にアプリケーションの仕様を量定化する事が大切です。

アプリケーションが明確になったら、各アプリケーションの使用条件でどういった加速度が入力されるかを測定します。これにはアナログ・デバイス社の加速度センサー評価ボードが最適です。例えば、加速度センサーを使ったユーザーインターフェースを検討している場合には、実際にデモボードに任意の動作を加えて、加速度センサーからの出力波形を測定します。また、測定する加速度の周波数や振幅がわからない場合には、高価な圧電型加速度センサーを利用して、入力加速度を測定する場合もあります。

測定対象が明確になれば、実際のアプリケーション開発を行います。例えば、ユーザーインターフェースの場合は、測定した波形の特徴から、任意の動作だけに反応するようなアルゴリズムを開発します。振動抑制のようなアプリケーションでは、シミュレーションでシステム全体の安定性を確認するといった事も行われます。

殆どの場合において、アプリケーション開発の後、もしくはアプリケーション開発と並行して試作を行う必要があります。試作に使用する加速度センサーは、入力加速度の測定に使用した加速度センサーが一般的です。ただ入力加速度の測定に高価な圧電型加速度センサーを使用した場合や、要求される特性によっては実現したアプリケーションの精度と入力加速度の特性から最適な加速度センサーを選定します。

加速度センサーを搭載して各アプリケーションを実現する事によって商品に新たな付加価値を付与しています。加速度センサーを搭載している代表的な民生機器には、ゲーム機、携帯電話、デジタルカメラ、プロジェクター、PC等があります。例えば携帯電話では、縦横検出により画面の表示向きの切り替えを行うといったアプリケーションや、ある特定の衝撃（例えばタップ）を検出して特定の機能を動作させるといった形で利用しています。デジタルカメラではカメラの傾きを画面に表示するといった利用方法や、前述の携帯電話と同様に衝撃を検出して様々な機能を動作させるといった使われ方をしています。プロジェクターでは加速度センサーの傾きを検出して、プロジェクターが傾いていても画面が台形にならないように補正をかけています。PCでは衝撃を検出したらHDDのヘッドを退避させてHDDドライブを保護するといった形で利用されています。

産業機器市場では、振動抑制や姿勢制御といった用途に加速度センサーが使われています。振動抑制の場合は、加速度センサーからの信号とアクチュエーターを使用して、外部で振動があった場合でも対象物が振動しないように制御するといった形で利用されています。

ここに挙げたアプリケーションは一例にすぎませんが、加速度センサーの信号は搭載される機器によって様々な形で利用されており、今後もその利用方法は益々拡大していくと思われます。