## 4.1 通信実験 (1)

これまでの実験では Arduino と Processing をシリアルケーブルで接続してデータの送受信を行ってきた．本実験では Arduino と Processing 間で無線通信によるデータの送受信を行う方法を習得する．無線通信機器として XBee を用いる．

実験目標

- XBee を使って無線通信を行うための方法の習得
- XBee を用いた移動式センシング装置の作成

### 4.1.1 ZigBee プロトコルと XBee モジュール

ZigBee ではスター型，メッシュ型，ツリー型のネットワーク方式が可能である．図 4.1 にこれらのネットワーク方式を示す．ZigBee には 3 種類のデバイスタイプが存在し，それぞれ**コーディネータ**，**ルータ**，**エンドデバイス**と呼ばれる．各 ZigBee デバイスはデバイスタイプを設定する必要がある．コーディネータはネットワークを管理する親局の役割を果たし，1 つのネットワーク内には必ず 1 つのコーディネータが必要である．エンドデバイスは他デバイスの接続を受け入れる能力はないが，データを接続先のルータまたはコーディネータに固定的に送信する．エンドデバイスはスリープ機能を持ち，センサーなどを接続してデータを収集しながら間欠動作が可能である．ルータは中継機能を持ったデバイスで，いつデータの中継を依頼されるか分からないためスリープ機能を持たない．

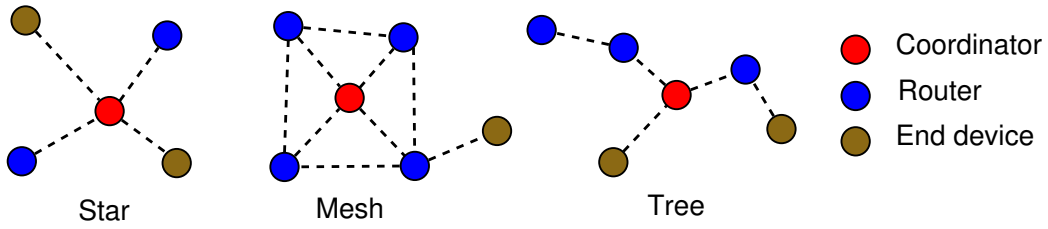


図 4.1: ZigBee で構成可能なネットワーク方式

ZigBee は 2 種類のネットワークアドレスを利用する．一つは **IEEE 64bit アドレス (固有アドレス)** と呼ばれる識別子で，世界中の ZigBee デバイスがそれぞれ固有の 64bit アドレスを持つ．XBee モジュールにはこのアドレスが裏面に 16 進数で上位 4 バイトと下位 4 バイトに分けて記載されている (例: 0013A200 4092FE4C)．もう一つは **16bit アドレス (ショートアドレス)** と呼ばれ，同一ネットワークに参加する際に動的に割り当てられる．同一ネットワークに属する ZigBee デバイスの 16bit アドレスは一意に割り当てられる．

ZigBee デバイスで構成するネットワークには同一の **PAN (Personal Area Network) ID** を割り当てる必要がある．PAN ID は 64 ビットの識別子で ID は任意に指定して良い．ZigBee デバイスは同一 PAN ID を持つデバイスとのみ通信が可能となり，逆に PAN ID が異なる ZigBee デバイス間では通信することができない．異なるネットワークには異なる PAN ID を割り当てることで複数のネットワークが混信することなく共存することが可能となる．

ZigBee デバイスで通信を可能にするためには，通信方式，デバイスタイプ，PAN ID 等の設定を行う必要がある．

**XBee モジュール**はアンテナ，高周波回路，信号処理回路など無線通信に必要なハードウェアが全部入った通信用のモジュールである．**XBee は ZigBee プロトコル**を使ってデータの送受信を行う．ZigBee は Bluetooth や Wi-Fi のような無線通信の規格を意味しており，ZigBee 物理層は IEEE 802.15.4 として規格化されている．本実験で用いる XBee モジュールはシリーズ 2 (XBee ZB と呼ばれることもある) を用いる．主なスペックは以下の通りである．

| | |
|---|---|
| 電源 | 2.1〜3.6 V DC |
| 室内通信距離 | 40 m |
| 見通し通信距離 | 120 m |
| 周波数帯域 | 2.4 GHz |
| 通信速度 | 250 kbps |
| デジタル I/O | 11 |
| アナログ I/O | 4 |
| PWM 出力 | なし |
| 通信時消費電流 | 40 mA |
| スリープ時消費電流 | 1 $\mu$A 以下 |

XBee には **Transparent Operation** と **API Operation** と呼ばれる 2 種類の運用方法が存在する．Transparent Operation は ZigBee デバイス間で無線化されたシリアル通信を行う運用方法で，非常に手軽に利用することができる．例えば，Transparent Operation では，可視化実験で実験した Arduino と Processing 間のシリアル通信をスケッチをほとんど変更することなく無線化することができる．一方，API Operation は API フレームデータと呼ばれる独自のフレームフォーマットを形成してデータの送受信を行う．API Operation は Transparent Operation と比較すると手軽に使用することはできないが，Transparent Operation よりも高度な機能を持つ (通信エラーの際の再送処理，通信先の指定，通信元の特定など)．本実験では手軽に利用可能な Transparent Operation を使用する．

本実験では 2 つの XBee モジュールを 1 セットとして配布する．今回の実験で配布する XBee モジュールには ZigBee 通信に必要な設定が既に書き込まれており，X-CTU を使ったデバイスの設定は行わない．各セットの XBee の設定は下記のようになっており (セット 12 の場合の例)，2 つの XBee 間で 1 対 1 通信 (ユニキャスト通信) のみ可能である．各 XBee モジュールセットには固有の PAN ID が割り当ててあるので，他の XBee モジュールセットと混信することなく通信が可能である．

| 名前 | デバイスタイプ | PAN ID | 宛先アドレス |
|---|---|---|---|
| 12C | Coordinator AT | 12ABCD | 相手 XBee の固有アドレス |
| 12R1 | Router AT | 12ABCD | 0 (=コーディネータが宛先) |

### 4.1.2 XBee 関連機器

本実験で使用する無線通信機器は，**XBee モジュール** (2 つ)，**XBee シールド** (1 つ)，**XBee USB インターフェース** (1 つ) である．図 4.2 にそれぞれの機器と装着例を示す．これらの機器の脱着は必要最小限に留め，出来る限り行わない事．脱着する必要がある場合はピンを曲げないように慎重に行うこと．

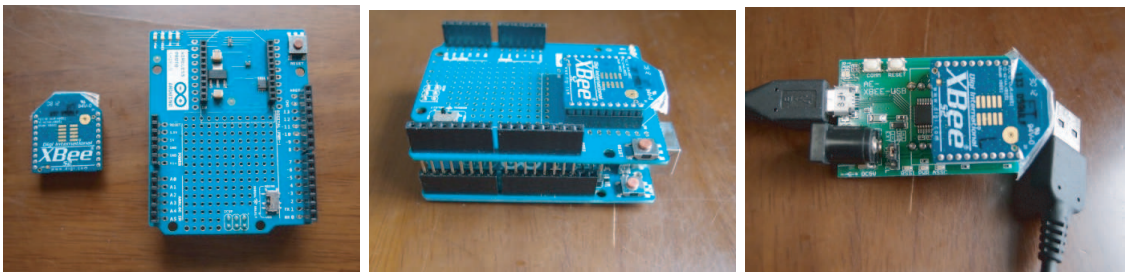

図 4.2: XBee モジュールと XBee シールド (左)，XBee シールド (XBee モジュール装着済み) を Arduino に装着 (中)，XBee モジュールを XBee USB インターフェースに装着 (右)

XBee シールドは XBee モジュールを Arduino に接続するために使用する．XBee シールドには SERIAL SELECT スイッチがあり，MICRO 側にすると XBee の無線通信が有効となり，USB 側にすると無効となる．Arduino にスケッチを書き込む際には SERIAL SELECT スイッチは USB 側にしておく必要がある．

XBee USB インターフェースは XBee モジュールと PC を接続するために使用する．XBee USB インターフェースが使用しているシリアルポート名は，Arduino を PC に接続して Arduino の [ツール] → [シリアルポート] メニューから確認できる．

### 4.1.3 復習：Arduino から Processing へのデータ送信 (シリアル通信による連携)

可視化実験 (1) では Arduino から Processing にシリアル通信でデータを送信する 2 つの方式 (通信方式 1 と通信方式 2) について説明し，それぞれの通信方式で実験を行った．本節ではそれぞれの通信方式について再度説明しておく (内容は 3.1.8 節とほぼ同じ)．

**通信方式 1** Arduino からは (Processing の都合は考えずに) 定期的にデータを送信する．Processing ではシリアルポートにデータが到着したことをトリガーとする割り込み処理によってデータを読み込む．

**通信方式 2(ハンドシェイク)** Processing は次のデータを受信する準備が整うと，データの送信要求を Arduino に送信する．Arduino は Processing から送られてくる送信要求を受け取ると次のデータを送信する．

通信方式 1 は等間隔にデータの送受信を行うための最も単純な方法である．しかし，一般に，機器 A から機器 B にデータを送信する場合，受信側 (B) が受信状態でない (あるいはデータの受信はできてもそれを処理する準備が出来ていない) 状況で送信側 (A) からデータを送信するとデータが失われる可能性がある．例えば，通信方式 1 では Processing の 1 フレームの間隔が Arduino のデータ送信間隔よりも長い場合，Processing は 1 フレームの間に複数回データを読み込むことになり，Arduino から送信されたデータの一部は利用されない[1]．このような問題が起きないようにデータの送受信を行うためには，お互いの状態を確認することが重要となる．通信方式 2 では受信側 (B) はデータの受信準備が整ったことを送信側 (A) に伝え，送信側 (A) はデータの送信要求を受けてからデータを受信側 (B) に送ることによりデータがロスすることを防いでいる．このような通信方式をまず握手してからデータを送信するということをイメージしてハンドシェイクと呼ぶ．

[1] 単に Processing で描画を行うという目的では，Processing は 1 フレームの間に複数回データを受け取っても，その中の任意の 1 つのデータに基づいて描画を行えば良いので特に問題はない．

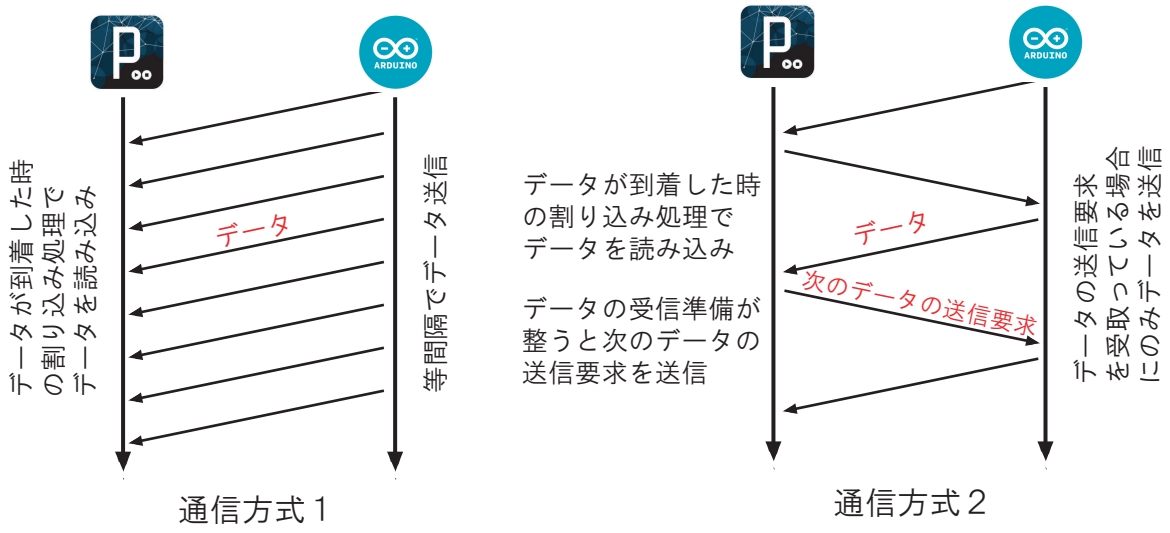


図 4.3: Arduino から Processing にデータを送信するための 2 つの通信方式

### 4.1.4 通信方式 1 で無線通信

Arduino から Processing にシリアル通信でデータ送信する簡単な例として次のタスク (タスク 2M) を考える．このタスクは可視化実験 (2) の演習 3.2.2 で出てきたものと同じである．

タスク 2M：2 つの電圧値情報の可視化 (複数)，大きな値 (0-1023)

- 図 4.4 に示す回路を Arduino とブレッドボードを用いて構成し，2 つの半固定抵抗を回して変化する 2 点 S0 と S1 の電圧値を Arduino の A0 ポートと A1 ポートで観測．
- Arduino で観測した点 S0 と S1 の電圧値 (を 0-1023 に AD 変換した値を) を Processing に送信し，S0 の電圧値 (に比例する値) を横幅，S1 の電圧値 (に比例する値) を縦幅，とする楕円として電圧値をリアルタイムに可視化．
- ウィンドウサイズは 500×500[a]とし，ウィンドウの中心を楕円の中心とする．楕円の横と縦の幅はそれぞれ電圧値が 0V の時に 0，電圧値が 5V の時に 500 となるように表示．

[a]可視化実験 (2) ではウィンドウサイズ 1024×1024 で表示していたが，大きすぎるので変更した．

**＜演習 4.1.1＞ シリアルケーブルによるシリアル通信 (通信方式 1)**

※ この演習は可視化実験 (2) の演習 3.2.2 で既に行っているが再度行う．

「タスク 2M」を通信方式 1(シリアルケーブルで通信) で実現せよ．

**基本手順**

1. 図 4.4 に示す回路を Arduino とブレッドボードを用いて構成．
2. 演習 3.2.2 で作成した Arduino スケッチ (serial_method1.ino) と Processing スケッチ (serial_method1_ellipse.pde) を用いる．
3. 半固定抵抗を回して正しく可視化できているか確認．

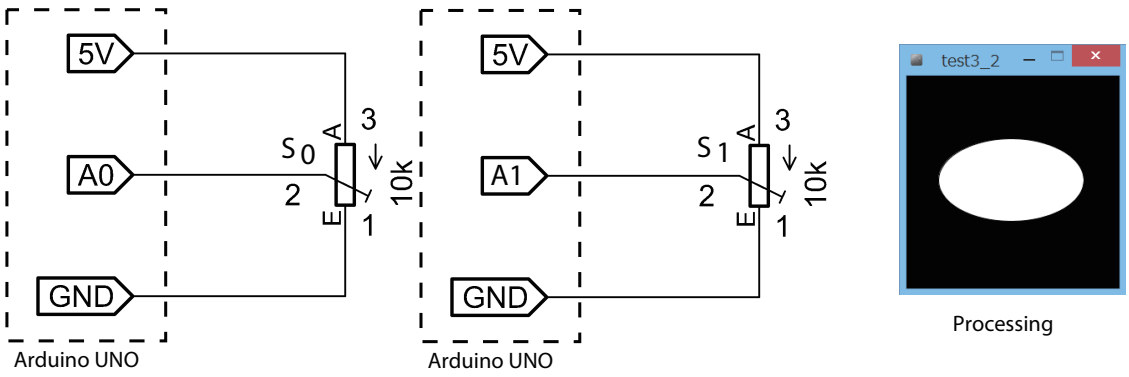


図 4.4: 2 つの半固定抵抗により変化する 2 点 S0 と S1 の電圧値を楕円の横幅と縦幅でリアルタイムに可視化.

**＜演習 4.1.2＞ XBee を使った無線通信の基礎 (通信方式 1)**

「タスク 2M」を通信方式 1(XBee を用いた無線通信) で実現する Arduino スケッチと Processing スケッチを作成して実行せよ．要するに演習 4.1.1 のシリアル通信を無線化する．

**基本手順**

1. XBee シールドを Arduino に装着する．XBee モジュール (Router) を XBee シールドに取り付ける．図 4.4 に示す回路を Arduino とブレッドボードを用いて構成 (作成済み).
2. XBee シールドの SERIAL SELECT を USB 側にする．Arduino スケッチ (serial_method1.ino) を書き込む.
3. この時点で Processing スケッチ (serial_method1_ellipse.pde) を実行して，シリアルケーブルによるシリアル通信が正しく動作していることを確認.
4. これから XBee 通信の設定を行う．XBee USB インターフェースに XBee モジュール (Coordinator) を取り付け，これを PC の USB ポート (どれでも良い) に接続.
5. XBee シールドの SERIAL SELECT を MICRO 側にする.
6. Arduino の [ツール] → [シリアルポート ] メニューから PC で利用可能なシリアルポート名を確認．するとシリアルケーブルが接続されているポート名/dev/ttyACM0(ACM1, ACM2, ... の場合もある ) と XBee USB インターフェースが接続されているポート名/dev/ttyUSB0(USB1, USB2, ... の場合もある ) がリストに表示されるはずである.
7. Processing スケッチのシリアルポート名を指定している部分を，先ほど調べた USB インターフェースが接続されているポート名に変更.
8. Processing スケッチを実行して無線通信がうまくいっていることを確認する．もし挙動がおかしい場合は，一度 Arduino への電源供給をストップして (シリアルケーブルを抜く ) から再度電源を供給してみよ．通信がうまくいっているか確認するために，Processing のデータ受信時に受信データの値をコンソールに表示すると良い.
9. 現状では Arduino への電源供給はシリアルケーブルで行われているが，シリアルケーブルを介して通信しているかもしれない．そこで，Arduino からシリアルケーブルをはずし，電池 (6 本組の電池ボックス) により Arduino に電源を供給せよ．電源の切り替えは，電池を Arduino に接続してからシリアルケーブルを抜くこと (実際には一度電源が切れても問題ない)．電源切り

替えの際，Processing は実行しっぱなしで良い．

10. XBee を使った無線通信でデータが正しく送信されていることを確認．
11. 時間が余っていれば以下も行う：無線通信が行われている状態で Arduino 持って移動し，どれくらい離れても通信が可能か確認せよ (移動中は誰かに Processing を見てもらう)．移動は部屋の中だけでなく部屋の外でも良い．一度通信が途絶えた後は Arduino を PC に近づけても通信が復活しない (不安定になる) ことが多いが，その場合は Arduino の電源供給を一度ストップしてから再度電源供給すると通信は復活する．

(注意) 電池の消耗を防ぐため，必要な時のみ電池で Arduino に電源供給すること．

#### 4.1.5 通信方式 2 で無線通信

XBee を用いた無線通信をしている時，通信している XBee 間の距離が離れるなどして通信が途切れる場合がある．通信方式 1 では通信が途切れた場合でも Arduino から Processing に一方的にデータを送信し続けることになるが，通信が途切れた状態でデータを送信し続けると，通信が再開した後のデータ通信が不安定になる場合がある．こうなった場合には Arduino の電源供給を一度ストップしてから電源を再度供給すると通信が正常に復活する (Arduino のリセットボタンを押してもダメ，Processing を再実行してもダメ)．この原因は今のところはっきりしていないが[2]，XBee の無線通信が途切れた状態で，一方的にデータを送信し続けると通信が不安定になるようである．

通信方式 2 を用いることでこの問題を解消することができる．Processing で Arduino から送られてくるデータを正常に受信した際に次のデータ送信要求を送ることにすれば，Arduino は Processing からのデータ送信要求を受け取ることで，前回送信したデータが正常に受信されたことを確認することができる．これにより，Arduino は前回のデータ送信が成功した場合にのみ次のデータを送ることになる．実際にはこの方法を少し修正して用いることが必要となるが (課題 4.1.2 で実施)，まずは”修正なしの”通信方式 2(XBee を用いた無線通信で通信) で「タスク 2M」を実行する演習を行う．

**＜演習 4.1.3＞ XBee を使った無線通信の基礎 (通信方式 2)**

XBee を用いた無線通信により「タスク 2M」を通信方式 2 で実現する Arduino スケッチ (serial_method2.ino) と Processing スケッチ (serial_method2_ellipse.pde) を作成して実行せよ．これらのスケッチを下記に示す．可視化実験 (1) でも通信方式 2(1 バイトのデータを 1 個のみ送信) を使った実験を行ったが，下記の Processing のスケッチはその時と若干仕様が異なっている．違いについてスケッチの後ろにコメントが書いてあるので確認すること．

**基本手順**

1. 演習 4.1.2 と同じ設定で回路を構成し，XBee シールド，XBee USB インターフェース，XBee モジュールの装着を行う (既に済んでいる)．
2. まずはシリアルケーブルを用いたシリアル通信により「タスク 2M」を実現する．XBee シールドの SERIAL SELECT を USB 側にして Arduino スケッチを書き込む．Processing スケッチを実行．通信を開始するためには最初に Processing からデータ送信要求を Arduino に送る必要がある．これは Processing スケッチを実行後，表示されるウィンドウをマウスでクリックすることで実行される (スケッチの該当部分を確認せよ)．

[2] 本実験で使用している Linux 環境では不安定になるが，Windows 環境では問題なかったりする．

3. 次に XBee を用いた無線通信により「 タスク 2M」を実現する．XBee シールドの SERIAL SELECT を MICRO 側にする．Processing スケッチのシリアルポート名を指定している部分に XBee USB インターフェースが接続されているポート名/dev/ttyUSB0(USB1, USB2，... の場合もある ) を指定して実行．
4. Arduino の電源供給を電池に切り替え，無線通信により「 タスク 2M」が実行できていることを確認．
5. 無線通信が行われている状態で Arduino 持って移動し，どれくらい離れても通信が可能か確認せよ (移動中は誰かに Processing を見てもらうと良い)．移動は部屋の中だけでなく，部屋の外でも良い．一度通信が途絶えた後は Arduino を PC に近づけても通信が復活しないはずである．そうなった場合は Processing のウィンドウをクリックして再度データ送信要求を送ることで通信が復活する (なぜそうなるか考えよ)．

Arduino スケッチ： serial_method2.ino

```
int sensorValue0, sensorValue1;
int inByte; // Processingからの送信要求を受け取る変数

void setup(){
  Serial.begin(9600);
}

void loop(){
  sensorValue0 = analogRead(0);
  sensorValue1 = analogRead(1);

  if (Serial.available() > 0) { // 送信要求を受け取った（受信バッファにデータあり）
    inByte = Serial.read(); // 受信済みの信号を読み込む（受信バッファが空になる）
    Serial.write(0x20);
    Serial.write(sensorValue0 / 0x20);
    Serial.write(sensorValue0 % 0x20);
    Serial.write(sensorValue1 / 0x20);
    Serial.write(sensorValue1 % 0x20);
  }
}
```

Processing スケッチ： serial_method2_ellipse.pde

```
import processing.serial.*;
Serial port;
int val0, val1;

void setup(){
  size(500, 500);
  port = new Serial(this, "/dev/ttyUSB0", 9600);
}

void draw(){
  background(0);
  int a = (int) map(val0, 0, 1024, 0, width);
  int b = (int) map(val1, 0, 1024, 0, height);
  ellipse(width / 2, height / 2, a, b);
}

void serialEvent(Serial p){
  if (p.available() >= 5) {
    if (p.read() == 0x20) {
      val0 = p.read() * 0x20 + p.read();
      val1 = p.read() * 0x20 + p.read();
      println(val0, val1);
      port.write(0xff); // 次のデータ送信要求 (任意の 1 バイト ) を送信
    }
  }
}

void mousePressed(){ // マウスボタンが押されたら呼び出される割り込み関数
  port.write(0xff); // 最初のデータ送信要求 (任意の 1 バイト ) を送信
}
```

(コメント ) 可視化実験 (1) でも通信方式 2 を用いたシリアル通信を行った (1 バイトのデータを 1 個送信)．その時はデータの送信要求は draw() 関数の最後に行っていたが (次の描画が可能になったという意図)，上記スケッチではデータ送信要求は serialEvent() 関数でデータ受信完了時 (次のデータ受信が可能になったので) に行っている．また，Processing からの最初のデータ送信要求をマウスボタンを押すことで行っている．

### 4.1.6 課題演習

**課題演習 1**

可視化実験 (1) の課題 3.1.4 では，図 4.5 に示す回路を Arduino とブレッドボードを用いて構成し，光センサ (照射光が強いほど電気抵抗が減少する素子，極性なし ) にあたる光の量によって変化する

点 S の電圧変化を時間-電圧グラフとして可視化する実験を行った．次の課題では同様の課題を XBee を使った無線通信で行う．これにより，ポータブル光量計測器を作成することができる．

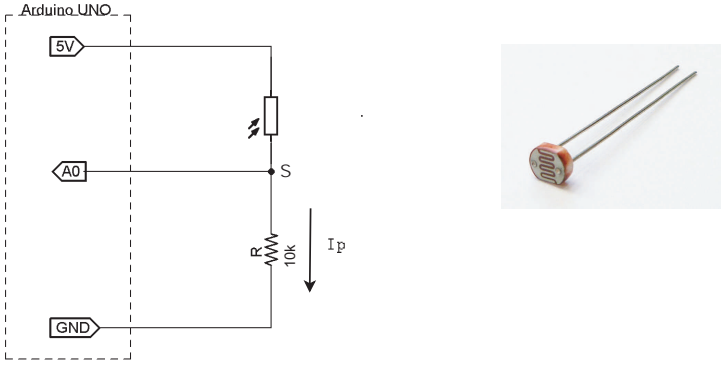


図 4.5: CdS 光センサ (右図) にあたる光の量で変化する電圧変化を Arduino で観測する回路図

**＜課題 4.1.1＞ 光センサ (移動式) による電圧変化の時間軸グラフによる可視化：基本**

ポータブル光量計測器で観測したデータを時間-電圧 (AD 変換した値) グラフとして可視化する Arduino スケッチ (serial_method2_light_basic.ino) と Processing スケッチ (serial_method2_light_basic.pde) を作成する．データの送受信は XBee を使った無線通信 (通信方式 2) を用いる．グラフ表示は以下の仕様に従うものとする (可視化実験 (2) の演習 3.2.3 で温度センサー情報をグラフ化した時と同じ)．

- Arduino から送信するデータは経過時間と点 S の電圧値を AD 変換した値 (0–1023 の整数)．
- Processing では横軸を経過時間，縦軸を点 S の電圧値を AD 変換した値とした時間-電圧グラフを描く．
- ウィンドウサイズは適当に決めて良いが，横軸の範囲は 20 秒，縦軸の範囲は 0–1024 とする (光量が増えるほどグラフ上部にプロット)．ただし，20 秒経過するごとにウィンドウをクリアして再度グラフの描画を繰り返す．
- Processing のコンソールにはデータを受信するたびに (経過時間 (ms)，電圧を AD 変換した値) を表示．

**基本手順**

1. 可視化実験 (2) の演習 3.2.3 で作成した Arduino スケッチ (serial_method1_thermo_basic.ino) は A0 ポートの電圧値を読み取って，50ms 間隔で時刻と電圧値 (AD 変換した値) を Processing に送信するスケッチである．このスケッチは通信方式 1 でデータ送信を行っているが，これを Arduino スケッチ (serial_method2.ino) を参考にして，通信方式 2 でデータ送信するスケッチに書き換える．作成したスケッチは (serial_method2_light_basic.ino) という名前で保存．※データの送信は等間隔で行う必要は無い．
2. 可視化実験 (2) の演習 3.2.3 で作成した Processing スケッチ (serial_method1_thermo_basic.pde) は受信データを時間-電圧 (AD 変換した値) グラフとして可視化するスケッチである．このスケッチは通信方式 1 でデータを受信しているが，Processing スケッチ (serial_method2_ellipse.pde)

を参考にして，通信方式 2 でデータ受信するスケッチに書き換える．作成したスケッチは (serial_method2_light_basic.pde) という名前で保存．※ 縦軸の描画範囲を調整する．

3. Arduino から Processing に XBee を使った無線通信でデータ送信し，時間-電圧グラフが描けていることを確認．Processing スケッチを実行後，表示されるウィンドウをマウスでクリックして最初のデータ送信要求を Arduino に送る必要があることに注意．
4. Arduino を PC から離して通信が一端途切れるとグラフの描画は停止する．この時，Processing のウィンドウ上でマウスをクリックしてデータ送信要求を送ると再び描画が再開するはずである．このことを確認せよ．※ 毎回 Arduino と PC の間の距離を離す手間を省くため，Processing を停止して通信を途絶させても良い．

**課題演習 2**

演習 4.1.1 では XBee を使ったシリアル無線通信 (通信方式 2) で Arduino から Processing にデータを送信した．しかし，通信を行っている XBee の距離が離れるなどの理由でいったん通信が途切れると，手動で Processing からデータ送信要求を送って通信を再開させる必要があった．自動で通信を再開させる単純な方法として次の 2 つの方法が考えられる．

1. Processing に一定期間以上 (例えば 1 秒以上)Arduino からデータが送られてこなければ，Processing からデータ送信要求を Arduino に送信．
2. Arduino に一定期間以上 (例えば 1 秒以上)Processing からのデータ送信要求が送られてこなければ，Arduino からデータを Processing に送信．

次の課題では 2 番目の方法を採用した方法を実装する．

**＜課題 4.1.2＞ XBee を使ったシリアル通信：途切れた通信からの復帰**

課題 4.1.1 で作成したスケッチを，通信が途切れても自動で復活するように改良せよ．作成した Arduino スケッチを (serial_method2_light_recovery.ino) という名前で保存せよ．Processing スケッチ (serial_method2_light_basic.pde) の変更はない．具体的には以下の仕様を満たすようなスケッチ (および回路) を作成する．

- Arduino から Processing に最後にデータ送信してから Processing からのデータ送信要求が 1000ms 以上ない場合，Arduino から Processing にデータ送信．
- Arduino と Processing の間でデータの送受信が正常に行われている場合は LED を点灯させ，そうでない (通信が途切れている) 場合は LED を消灯させる．LED は 330Ω の抵抗を介して Arduino の D13 と GND に接続する事．

スケッチが完成したら無線通信が正しく動作しているか確認．次いで Arduino を PC から離して通信を一端途切れさせ，再度近づけた時に通信が復活することを確認せよ．また，LED の点灯状況がデータ通信の成否と正しく連動しているか確認せよ．※ スケッチの作成段階では毎回 Arduino と PC の間の距離を離す手間を省くため，Processing を停止して通信を途絶させても良い．ただし，最終的な動作確認は Arduino を物理的に PC から離して実験すること．

**課題演習 3**

課題 4.1.2 で作成したスケッチを実行するとリアルタイムでデータを可視化するが，データを保存したい場合もある．次の課題では Arduino から Processing に送られてきたデータをファイルに保存して，そのファイルデータからグラフを描いてみよう．

**＜課題 4.1.3＞ 光センサによる電圧変化の時間軸グラフによる可視化：ファイル出力**

課題 4.1.2 では Processing で光センサの時間-電圧 (AD 変換後の値) グラフを表示しているが，これを修正して次の仕様を満たす Processing スケッチ (serial_method2_light_file.pde) を作成し，下記の手順でグラフを作成せよ．Arduino スケッチ (serial_method2_light_recovery.ino) の変更はない．

- Processing の基本仕様は課題 4.1.2 で作成した (serial_method2_light_basic.pde) と同じ．
- Processing で受信したデータ (経過時間 [ms]，センサー値) をカンマ区切りでファイルに出力．ファイルの出力例を以下に示す．

```
0,345
54,347
102,565
146,568
...
```

- ファイルに保存されたデータを計算表ソフト LibreOffice Calc を使ってグラフにする．
- ファイルで保存するデータは Arduino を移動させてデータを取得し，一時的にデータ通信が途切れている状況を含むようにすること．

**補足事項 1**

Processing でファイル出力を行うためのサンプルスケッチ (printWriter_sample.pde) を以下に示すので参考にせよ．このスケッチを実行すると file1 という名前のファイルにデータが書き込まれて作成される．ただし，データの書き込みを終了するためには Processing のウィンドウをアクティブにした状態でキー'q' を押して書き込みを終了する必要がある．ファイルは Processing スケッチがあるディレクトリ内に作成される．

Processing スケッチ： printWriter_sample.pde

```
int val1, val2;
PrintWriter output; // PrintWriterクラスのオブジェクトを宣言

void setup(){
  val1 = 0;
  val2 = 0;
  output = createWriter("file1"); // 出力先ファイル名を指定してインスタンス生成
}

void draw(){
  ++val1;
  val2 += 10000;
  println(val1, val2); // チェック用にコンソールに表示
  output.print(val1);  // データを指定ファイルに出力
  output.print(",");   // データを指定ファイルに出力
  output.println(val2);// データを指定ファイルに出力 (改行あり)
}

// キーが押されたら呼び出される割り込み関数
void keyPressed(){
  if (key == 'q') { // keyは最後に押されたキーの値を保持するシステム変数
    output.flush(); // 書き込みバッファの内容を全て書きだす
    output.close(); // ファイルを閉じる (閉じないと書き込まれない)
    exit(); // Processingを終了
  }
}
```

**補足事項 2**

本課題で得られるファイルデータから LibreOffice でグラフを描画するための手順を以下に記述する．

1. 計算表ソフト LibreOffice Calc を立ち上げる．データファイルをドラッグして立ち上がった画面に放り込む．
2. テキストのインポートというウィンドウが現れるので，区切りのオプションの設定で「区切る」，「コンマ」を選択して OK を押す．
3. 図 4.6 のようにデータが表に入力されているので，表示したいデータの列 (この場合は A,B) を選択してグラフボタン (図 4.6 参照) を押す．
4. 図 4.6 のようにグラフウィザードというウィンドウが現れるので，1. グラフの種類で「散布図/点のみ」を選択 (図 4.6 では「線のみ」を選択しているが，点のみの方が良い)．その他，4. グラフ要素も適宜設定する．完了ボタンを押すとグラフが表示される．

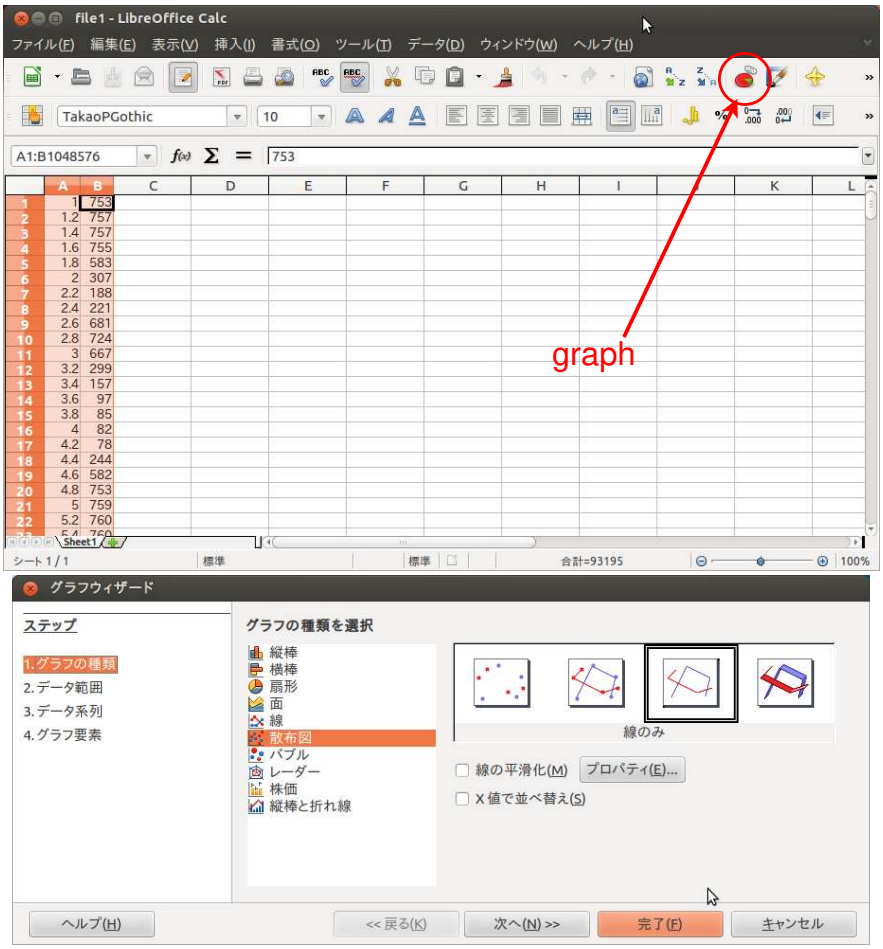


図 4.6: LibreOffice の設定例

**課題演習 4**

**＜課題 4.1.4＞ 光センサと温度センサによるセンシング情報の可視化：一定間隔での送信**

課題 4.1.2 では光センサによるセンシング情報を Processing で可視化した．本課題ではさらに温度センサによるセンシングを加え，二つの情報を同時に可視化する．また，現状では Arduino から Processing へのデータ送信間隔は特に決めていなかったが (データの送信要求があったら即時送信していた)，本課題では 2 つのセンサーで観測したデータを一定時間間隔で送信する．具体的には以下の仕様に従いデータを送信して Processing で可視化する．作成した Arduino スケッチを (serial_method2_light_interval.ino)，Processing スケッチを (serial_method2_light_interval.pde) という名前で保存せよ．

- 光センサの情報 (0–1023 の値) と温度センサの情報 (0–1023 の値) および時刻 (millis() 関数の値) を 50ms ごとに送信．
- Processing では，課題 4.1.2 で表示した，光センサーの時間-電圧 (AD 変換後の値) グラフに重ねて温度センサーの時間-電圧 (AD 変換後の値) グラフを表示する．ただし，温度センサーの電圧 (AD 変換後の値) を表示する縦軸のスケールは値の変化が分かるように調整せよ．また，2 つのグラフは異なる色で描く事．

### 4.1.7　発展課題

**＜発展課題 4.1.1＞ 光センサと温度センサによるセンシング情報の可視化：異なる間隔での送信**

課題 4.1.4 では光センサと温度センサーの情報を一定間隔で送信した．本課題ではそれぞれのセンサーの情報を異なる間隔で送信する．具体的には以下の仕様に従いデータを送信して Processing で可視化する．作成した Arduino スケッチを (serial_method2_light_interval2.ino)，Processing スケッチを (serial_method2_light_interval2.pde) という名前で保存せよ．

- 光センサの情報 (0–1023 の値) は 50ms ごとに送信する．
- 温度センサの情報 (0–1023 の値) は 200ms ごとに送信する．
- Processing での表示は課題 4.1.4 と同様.
- Processing では受信したデータが光センサのデータなのか温度センサのデータなのか判別する必要がある．そこで，光センサの情報と温度センサの情報を送信する際，それぞれ異なるヘッダーを送信して，これをもとに Processing ではどちらのセンサーの情報なのか判別する．

## 【 レポート 4.1　(2018 年 6 月 1 日出題，2018 年 6 月 8 日 (12:50) 締切)】

※ スケッチには詳細なコメントを記述すること．コメントの無いものは採点しない．

### レポート 4.1.1

課題 4.1.1 で作成した Arduino スケッチ (serial_method2_light_basic.ino) と Processing スケッチ (serial_method2_light_basic.pde) を報告せよ．また，本課題を行った結果 (グラフのスナップショットを含む) を報告せよ．

### レポート 4.1.2

課題 4.1.2 で作成した Arduino スケッチ (serial_method2_light_recovery.ino) を報告せよ．また，本課題を行った結果 (グラフのスナップショットを含む) を報告せよ．

### レポート 4.1.3

課題 4.1.3 で作成したグラフのスナップショットを報告せよ．ただし，Arduino を移動させてデータを取得し，一時的にデータ通信が途切れている状況を含むようにすること．

### レポート 4.1.4

課題 4.1.4 で作成した Arduino スケッチ (serial_method2_light_interval.ino) と Processing スケッチ (serial_method2_light_interval.pde) を報告せよ．また，本課題を行った結果 (グラフのスナップショットを含む) を報告せよ．

### 発展課題レポート 4.1.1

発展課題 4.1.1 で作成した Arduino スケッチ (serial_method2_light_interval2.ino) と Processing スケッチ (serial_method2_light_interval2.pde) を報告せよ．また，描画したグラフのスナップショットを報告せよ．

### 参考サイト

http://www.geocities.jp/zattouka/GarageHouse/micon/XBee/XBee1.htm

http://mag.switch-science.com/2012/08/01/startup_xbee_zb/

http://www.skyley.com/wiki/index.php?ZigBee%E5%85%A5%E9%96%80#va812dc3