

# IEEE1888 通信ボード (学習用キット)

## ユーザーズ・マニュアル

2011年11月版  
落合秀也 (東京大学)

### 1. はじめに

このマニュアルは、IEEE1888 通信ボード(学習用キット)の使い方、プログラミング方法、ハードウェアの構成方法を記載しています。このマニュアルに従って作業を進めていくことにより、初めての方でも一通りの知識を身に着けることができますが、組込みマイコンやデジタル回路についての知識があれば、さらに多様な応用に結び付けていくことができるようになります。なお、IEEE1888 通信ボードは **Arduino** をベースとしています。

### 2. キットの内容

IEEE1888 通信ボード、学習用シールド<sup>1</sup>、台座、ACアダプタ、USBケーブル、LANケーブル、DVD-R、初期設定の情報、本マニュアル、が同梱されていることを確認してください。

DVD-Rには、

- Arduino IDE (フォルダ名: arduino-0023)
- IEEE1888 通信ライブラリ(フォルダ名: FIAPUploadAgent)
- EthernetDHCP/DNS ライブラリ (フォルダ名: EthernetDHCP, EthernetDNS)
- Time ライブラリ (フォルダ名: Time)
- IEEE1888 SDK (一般的な IEEE1888 のソフトウェア開発キット)

が、同梱されています。

---

<sup>1</sup> Arduino(本 IEEE1888 通信ボードの原型となった組込みハードウェア)の世界では、ボード本体に接続する基板のことを、“シールド”と呼びます。

### 3. 組み立ててみよう

まず、IEEE1888 通信ボードに台座を取り付けます(図 1)。その後、学習用シールドをコネクタに接続してください(図 2)。これで組み立ては完了です。

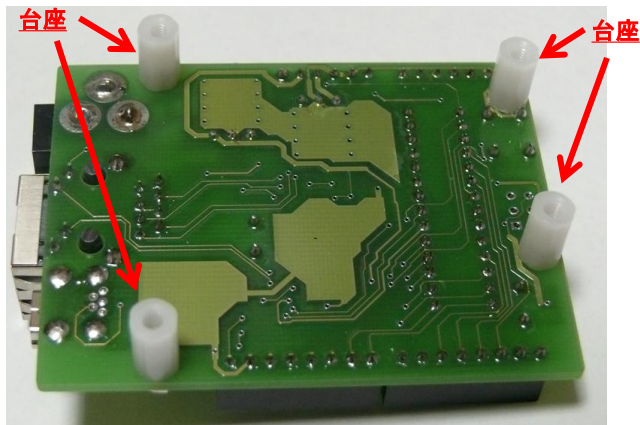


図 1: 台座の取り付け

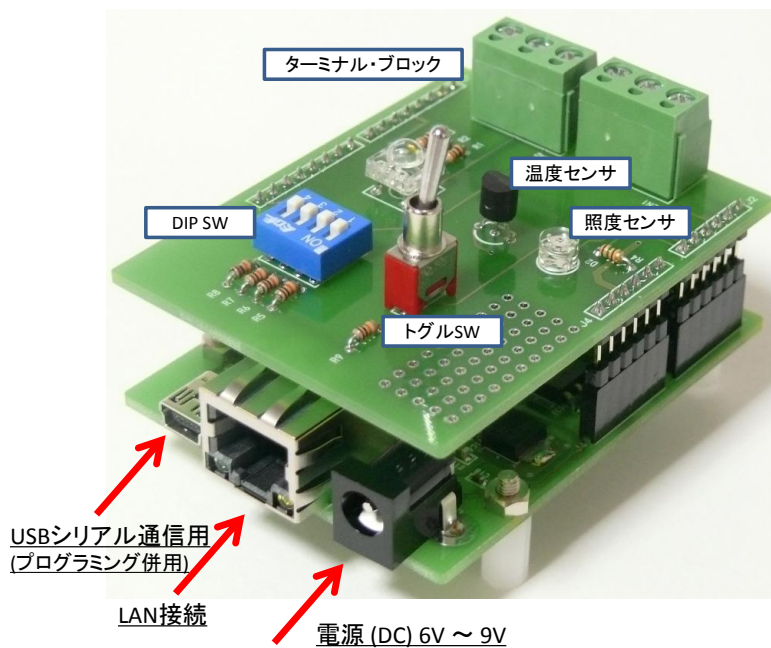


図 2: 学習用シールドを搭載した様子(完成図)

## 4. 初期設定で動作を確認してみよう

DHCP(動的ホスト設定プロトコル)が有効なネットワークに接続し、電源を入れれば、初期設定情報に基づいて立ち上がります。LED インジケータが、図 3 のように変化すれば成功です(LED インジケータの意味は表 1 の通りです)。

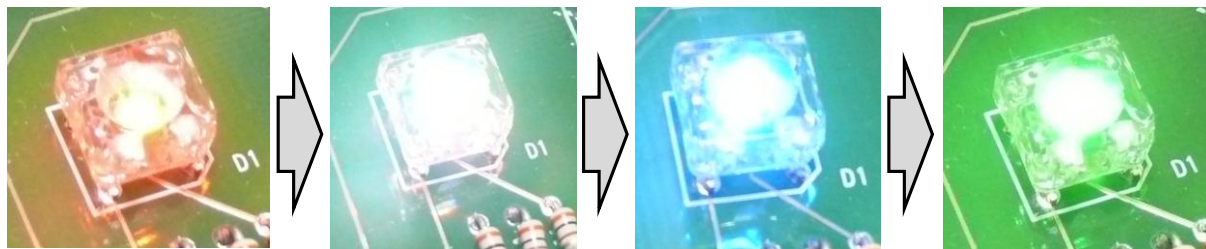


図 3: 電源の立ち上げから成功までの LED 表示 (赤点滅 → 白 → 青 → 緑)

表 1: LED インジケータによるステータス表示の内容

LED の色	意味
赤点滅	DHCP による IP アドレス情報の取得中
黄点滅	DNS によるサーバの IP アドレス情報の解決中
白色	NTP による時刻情報の取得中
青色	サーバへのアクセスおよびアップロード処理中
緑色	アップロード成功
消灯	次の測定までの待機中
紫色	サーバへの接続失敗 (TCP 接続失敗)
黄色	HTTP 通信エラー
水色	IEEE1888 通信エラー
赤色	その他のエラー

成功すると、fiap-sandbox サーバに計測データがアップロードされます。アップロードされたデータは、次の URL,

## http://fiap-sandbox.gutp.ic.i.u-tokyo.ac.jp/

から閲覧できます(図 4)。同封されている「初期設定の情報」を参照し、該当するポイント ID を探してください。ポイント ID をクリックすると、アップロードされたデータのトレンドをグラフ(あるいは一覧)表示します(図 5)。

<a href="http://gutp.jp/Arduino/Sample05/DIPSW">http://gutp.jp/Arduino/Sample05/DIPSW</a>	2011-11-15T14:03:41.000+09:00	15
<a href="http://gutp.jp/Arduino/Sample05/Illuminance">http://gutp.jp/Arduino/Sample05/Illuminance</a>	2011-11-15T14:03:41.000+09:00	306
<a href="http://gutp.jp/Arduino/Sample05/TGLSW">http://gutp.jp/Arduino/Sample05/TGLSW</a>	2011-11-15T14:03:41.000+09:00	ON
<a href="http://gutp.jp/Arduino/Sample05/Temperature">http://gutp.jp/Arduino/Sample05/Temperature</a>	2011-11-15T14:03:41.000+09:00	23.4

図 4: 該当するポイント ID を探し、タイムスタンプが最新であるかどうかを確認する。

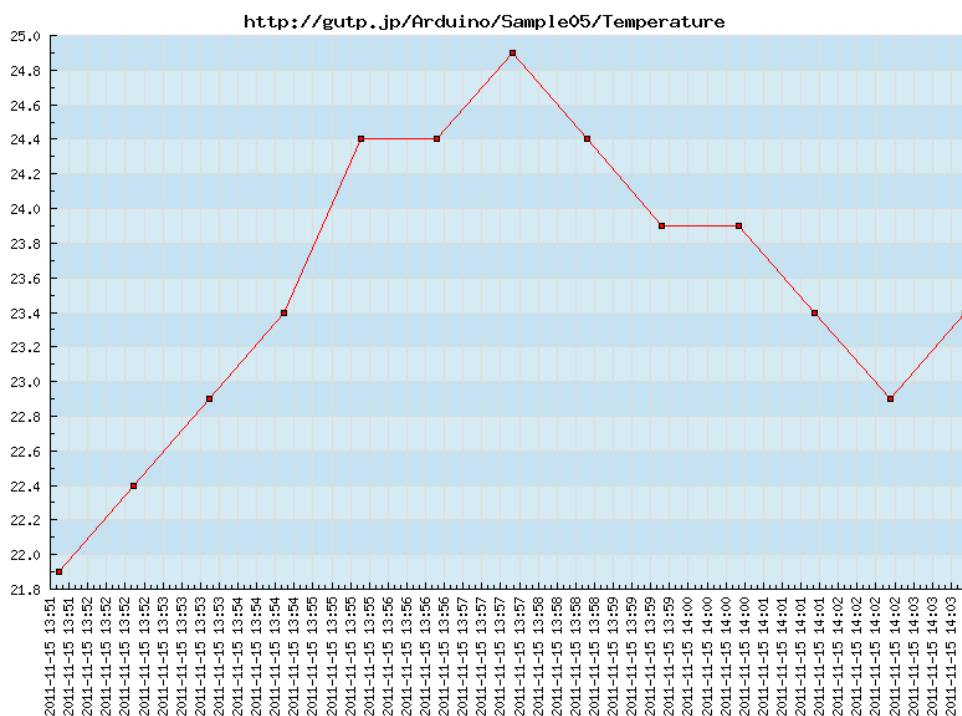


図 5: ポイント ID をクリックすると、直近のトレンドが、一覧表示される。

(注) DHCP が有効でないネットワークでは、アドレスが取得できないため、赤点滅までしか行きません。

(注) DHCP が有効であっても、NTP が無いネットワークの場合は、時計を合わせることができないため、正常に動作しません。

(注) DHCP が有効であっても、外部ネットワークとの接続に HTTP PROXY サーバを介す必要がある場合は正常に動作しません。

## 5. プログラミング環境の整備

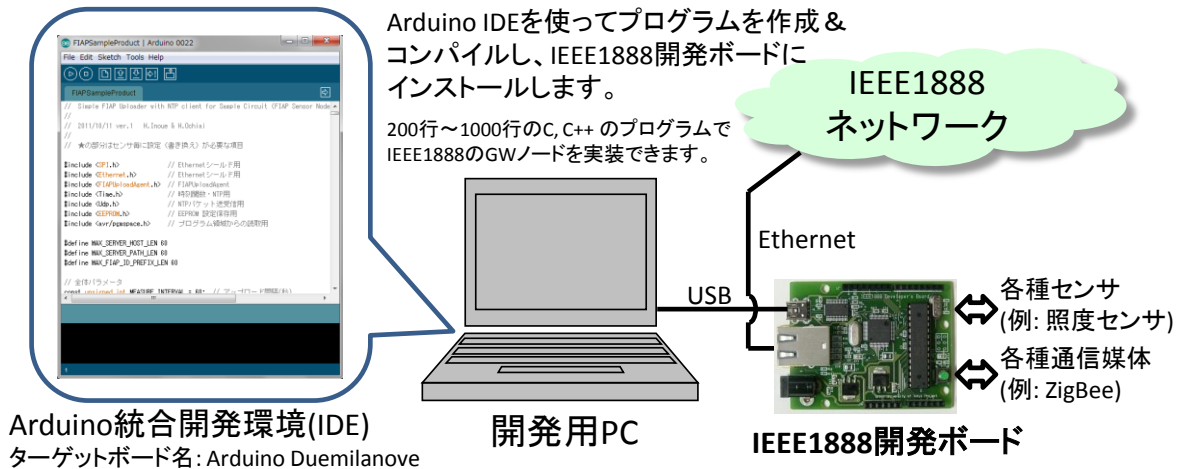


図 6: プログラミングのための環境設定(全体像)

プログラミング環境の設定は、次の 4 つのステップで構成されます。

- Arduino 統合開発環境(IDE)のインストール
- ライブラリの登録
- FTDI USB-Serial インタフェース・ドライバのインストール (オプション)
- ターゲットボード, シリアルインタフェースの設定

### 5. 1. Arduino 統合開発環境(IDE)のインストール

DVD-R に含まれている Arduino IDE (arduino-0023)を、適当なディレクトリ(例えば C:¥Program Files¥arduino-0023)にコピーしてください。Arduino IDE は、Arduino の公式ページ(<http://arduino.cc/en/Main/Software>)からも、ダウンロードすることができます。2011 年 11 月現在、arduino-0023.zip というファイルがダウンロードできるので、上記のように適当なディレクトリに展開してください。

※ arduino.exe へのショートカットを、デスクトップに作成しておくと、便利です。

## 5. 2. 必要なライブラリの登録

このボードの IEEE1888 通信対応化は、ソフトウェアによって実現されています。そのソフトウェアは FIAPUploadAgent という名前にライブラリ化されており、その設定を事前に施しておく必要があります。

ライブラリの設定は、Windows の場合は、ドキュメントフォルダの Arduino フォルダに対して行います。ここでは、FIAPUploadAgent ライブラリの他に Time ライブラリ、EthernetDHCP、EthernetDNS ライブラリの設定方法についても記載します。IEEE1888 通信は、FIAPUploadAgent ライブラリがあれば実現できますが、時刻情報の管理に Time ライブラリがあると便利です。またネットワーク通信に DHCP や DNS 機能があると、利便性がさらに向上します。

### ▲FIAPUploadAgent ライブラリ

東京大学 落合秀也, 広島市立大学 井上博之先生により開発された Arduino 向けの IEEE1888 通信ライブラリ(WRITE クライアント専用)です。同梱の DVD-R に FIAPUploadAgent というフォルダ名で含まれています。

### ▲Time ライブラリ

Arduino 上で、時計を Unixtime で管理するためのライブラリです。このライブラリは、同梱の DVD-R にも Time というフォルダ名で含まれていますが、オリジナルは <http://www.arduino.cc/playground/Code/Time> から入手できます。

### ▲EthernetDHCP, EthernetDNS ライブラリ

Arduino+Ethernet で、DHCP や DNS 機能を持たせるためのライブラリです。このライブラリは同梱の DVD-R にも EthernetDHCP や EthernetDNS というフォルダ名で含まれていますが、オリジナルは <http://gkaindl.com/software/arduino-ethernet/> から入手できます。

※ ※ ※

ライブラリは、フォルダとなって提供されているので、入手後、それらをユーザ・ドキュメントの Arduino フォルダの libraries フォルダ(なければ作成のこと)に配置してください(図 7)。その後、Arduino を再起動すると、これらのライブラリは自動的に読み込まれます。図 8 のように、メニューで、Sketch→ Import Library ... と進み、ライブラリが登録されていることを確認して、登録完了です。

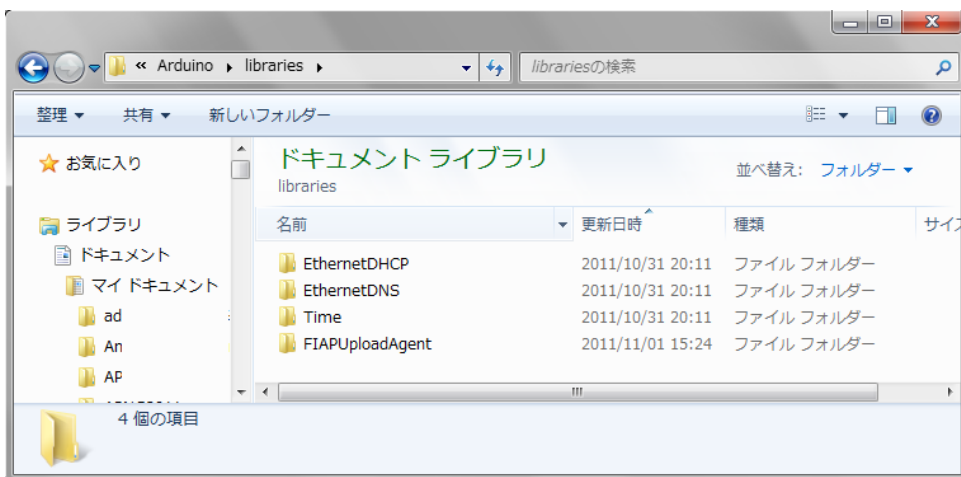


図 7: Arduino の libraries フォルダへライブラリを設定

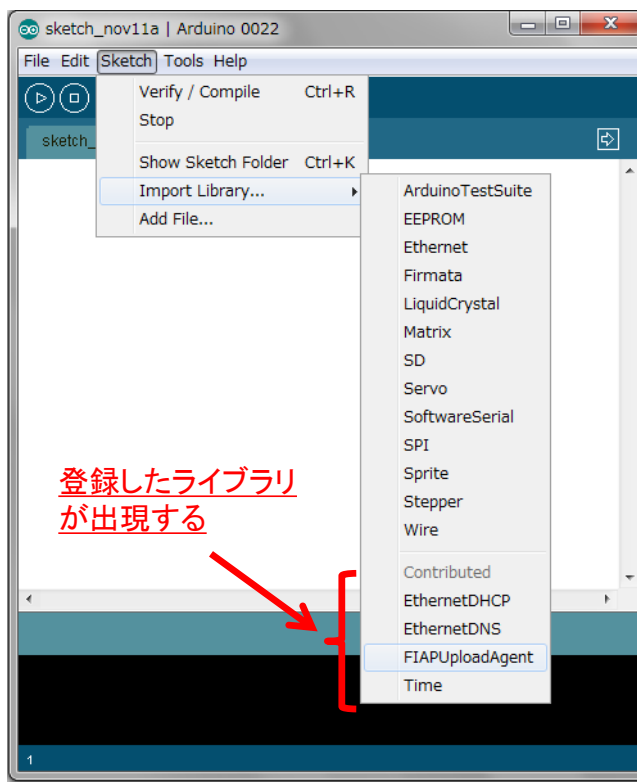


図 8 : ライブラリ登録の確認

### 5. 3. FTDI USB-Serial インタフェース・ドライバのインストール (オプション)

IEEE1888 通信ボードを開発用 PC に USB で接続すると、ドライバのインストールが行われます。成功すると、図 9 のようにポップアップが出現しますが、もし、そのようにならない場合は、下記 URL にアクセスし、FTDI ドライバのダウンロードおよびインストールを行ってください(OS によって、インストールすべき内容は異なりますので具体的な指示は、この Web サイトに従ってください)。

<http://www.ftdichip.com/FTDrivers.htm>

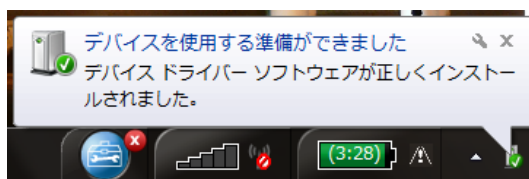


図 9 : ドライバのインストールに成功すること

### 5. 4. ターゲットボードの選択とシリアルポートの選択

#### 5. 4. 1. ターゲットボードの選択

Arduino IDE のメニューで、Tools → Board と進むと、図 10 のように、様々なターゲットボードの種類が出てきます。ここでは、ターゲットボード名に **Arduino Duemilanove or Nano w/ ATmega328** を指定してください。

#### 5. 4. 2. シリアルポートの選択

ボードをパソコンに正しく接続されると USB シリアル通信インタフェースに対し、COM ポートが割り当てられます。Arduino IDE のメニューで、Tools → Serial Port と進む(図 11)と、COM が出現しますので、その COM を選択しておいてください。図 11 の場合は、COM 9 になっています。

※ ※ ※

以上で、プログラミング環境の整備は完了です。

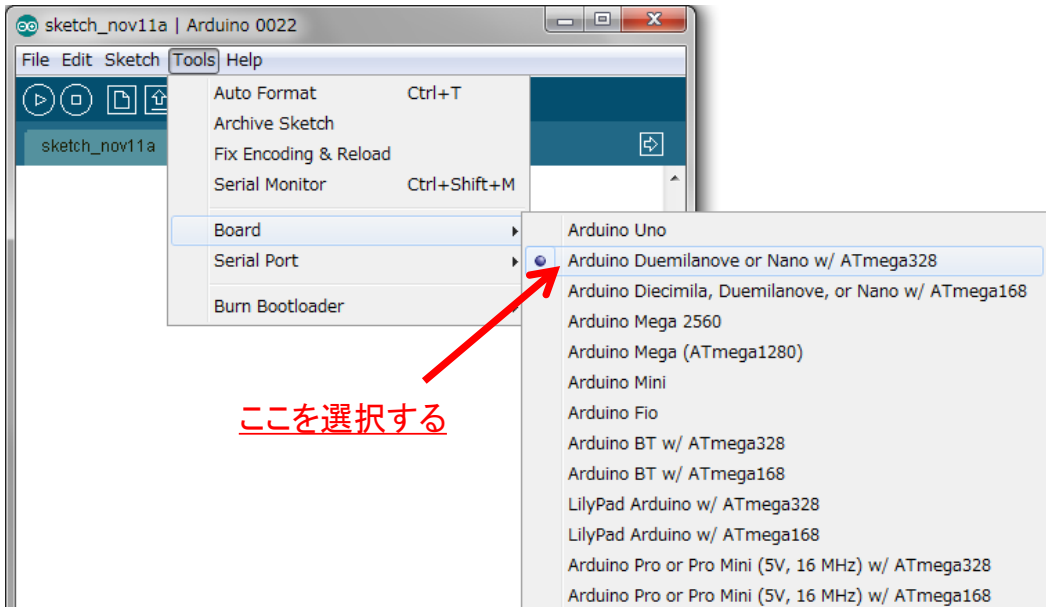


図 10: ターゲットボードの選択

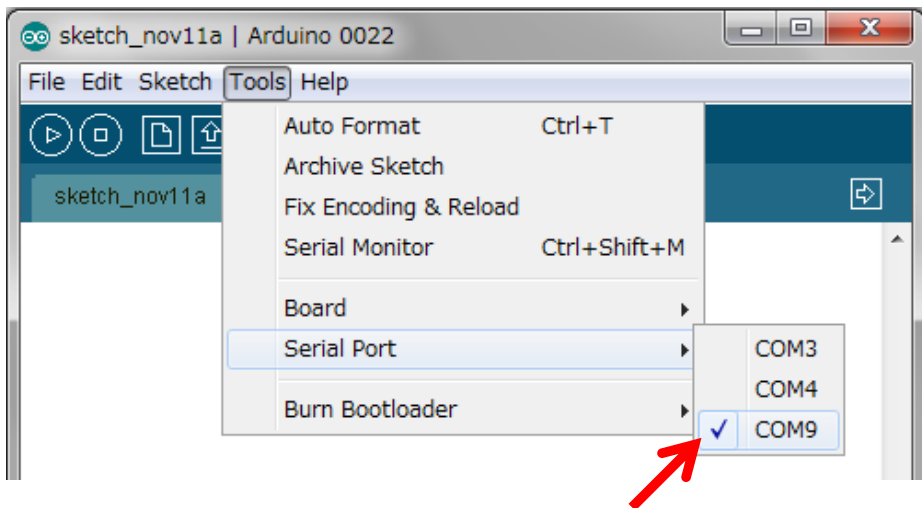


図 11: 該当する COM を選択する

## 6. サンプルプログラムを書きこんでみる

### 6. 1. 付属のサンプルプログラムについて

サンプルプログラムは、ライブラリを設定したときに、Arduino IDE に同時に読み込まれています。IEEE1888 通信関係のサンプルプログラムは、図 12 のように、メニューから File --> Examples と進むと、FIAPUploadAgent という項目が現れ、そこに FIAPsensorsAPIHowto, FIAPsensorsSimple, FIAPsensorsDHCP, FIAPsensorsCLI が出現します。

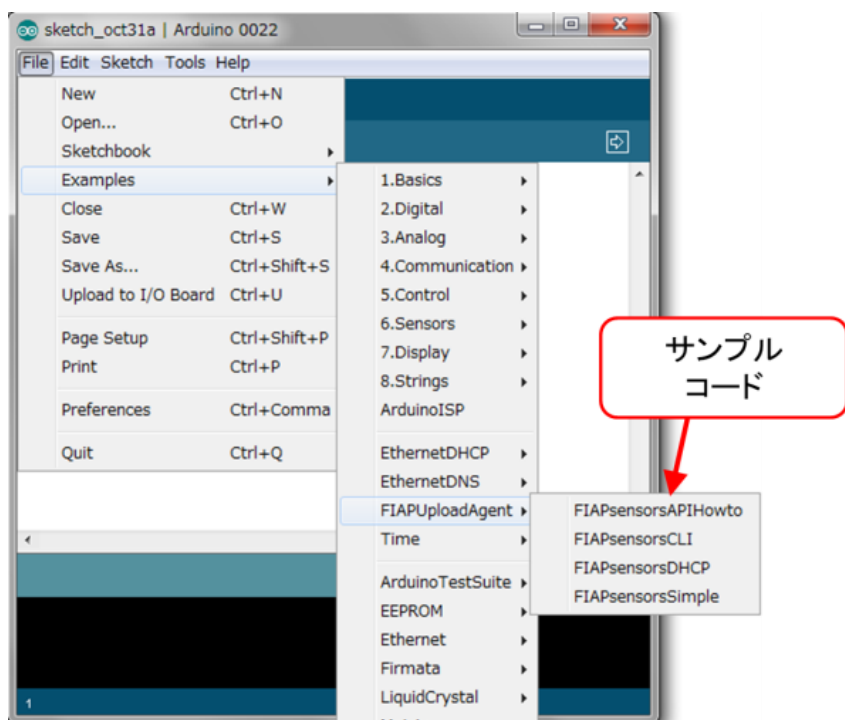


図 12: サンプルコードの読み込み方法

これらは FIAPUploadAgent を利用するサンプルプログラムです。それぞれの特徴を下記に記します。

- **FIAPsensorsAPIHowto**: ライブラリの使い方を解説した最も簡易なプログラムになります。学習用センサシールドの利用は想定していません。

- **FIAPsensorsDHCP** : 起動時に動的に IP アドレスを取得し、学習用センサシールドから計測したデータ（温度、照度、DIPSW の状態、トグル SW の状態）を fiap-sandbox サーバ(後述)へ送信します。
- **FIAPsensorsSimple** : DHCP 版を簡易にしたもので、IP アドレスはプログラムに直書きとなっています。
- **FIAPsensorsCLI** : コマンドラインインタフェース(CLI)を備え、プログラミング環境が無い人でも、各種通信設定(IP アドレスやポイント ID の設定など)を行えるようになっています。

以下では、FIAPsensorsSimple を例に上げ、プログラムを書込む方法を記載します。このプログラムは学習用ですので、サーバの情報や、ローカル・ネットワークの情報などをすべて直書きしています(実際的な運用を考える場合は、FIAPsensorsCLI や DHCP の方を参照してください)。

まず、図 12 を参考に、FIAPsensorsSimple のサンプルコードを読み込んでください。その後、図 13 の部分を、接続するネットワークに応じて設定してください。

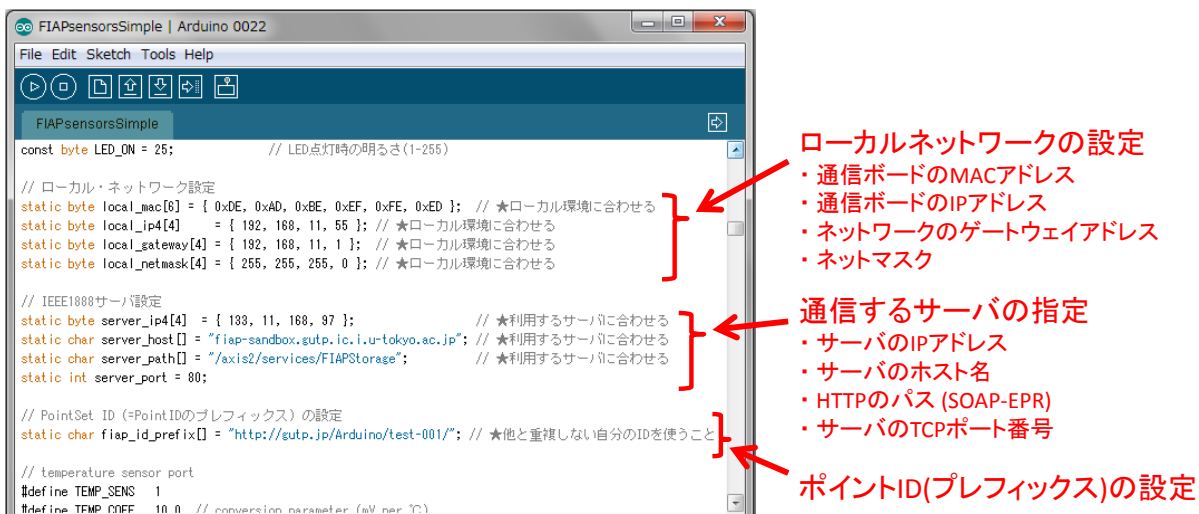


図 13: サンプルコードを読み込み、コードを改変 (対象とするネットワークに合わせる)

自由に利用可能なサーバとして、fiap-sandbox サーバが常時運用されています。このサーバの IEEE1888 通信アドレスは、

<http://fiap-sandbox.gutp.ic.i.u-tokyo.ac.jp/axis2/services/FIAPStorage>

であり、データの閲覧は下記 URL から可能となっています。

<http://fiap-sandbox.gutp.ic.i.u-tokyo.ac.jp/>

## 6. 2. ビルドとインストール

パソコンと IEEE1888 通信ボードを接続し、図 14 のように Arduino IDE の アップロードボタンをクリックします。ソフトウェアのビルドが実行され、成功すると、生成されたバイナリイメージが Arduino 本体に書き込まれます。図 15 のように、Done uploading と表示されたら、ビルドおよびインストールが完了です。

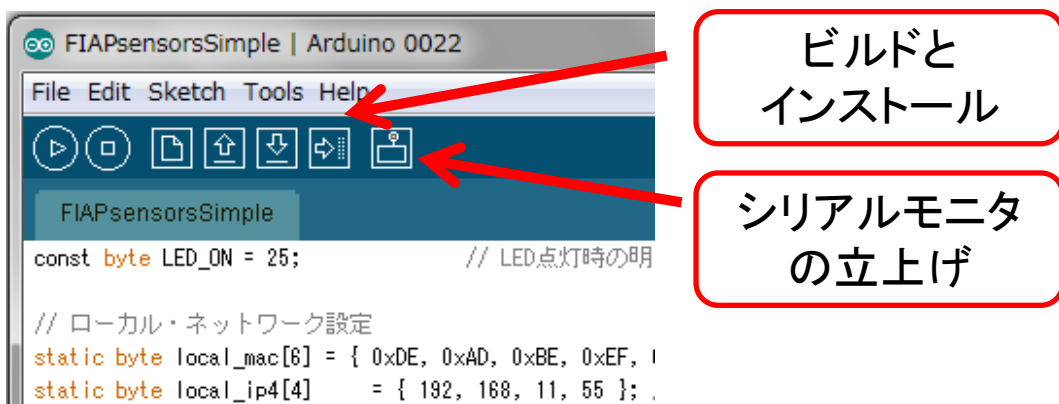


図 14: ソフトウェアの書込みと動作確認方法

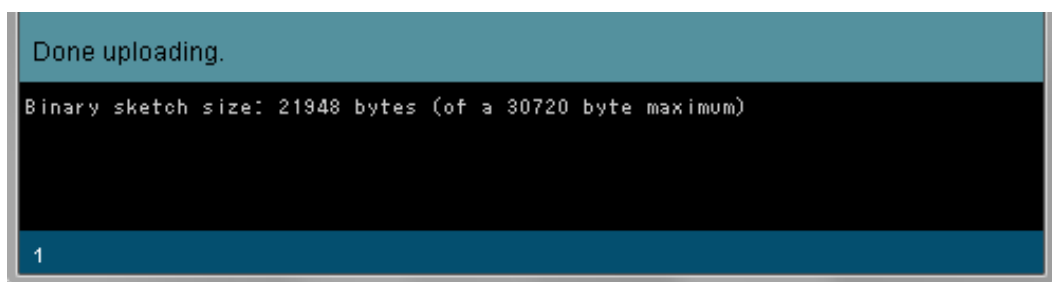


図 15: 「Done uploading」 -- 書込みが成功したことを示すサイン

ファームウェア書込みが完了すると IEEE1888 通信ボードは自動的に再起動します。この際に想定されたネットワークに接続されていれば、センサからの観測データを、サーバにアップロードします。

このサンプルコードは、随所でステータス情報を USB シリアルポートに出力するようになっています。図 14 に示す方法で Arduino IDE のシリアルモニタを起動して、シリアル通信の内容を確認できます(図 16)。

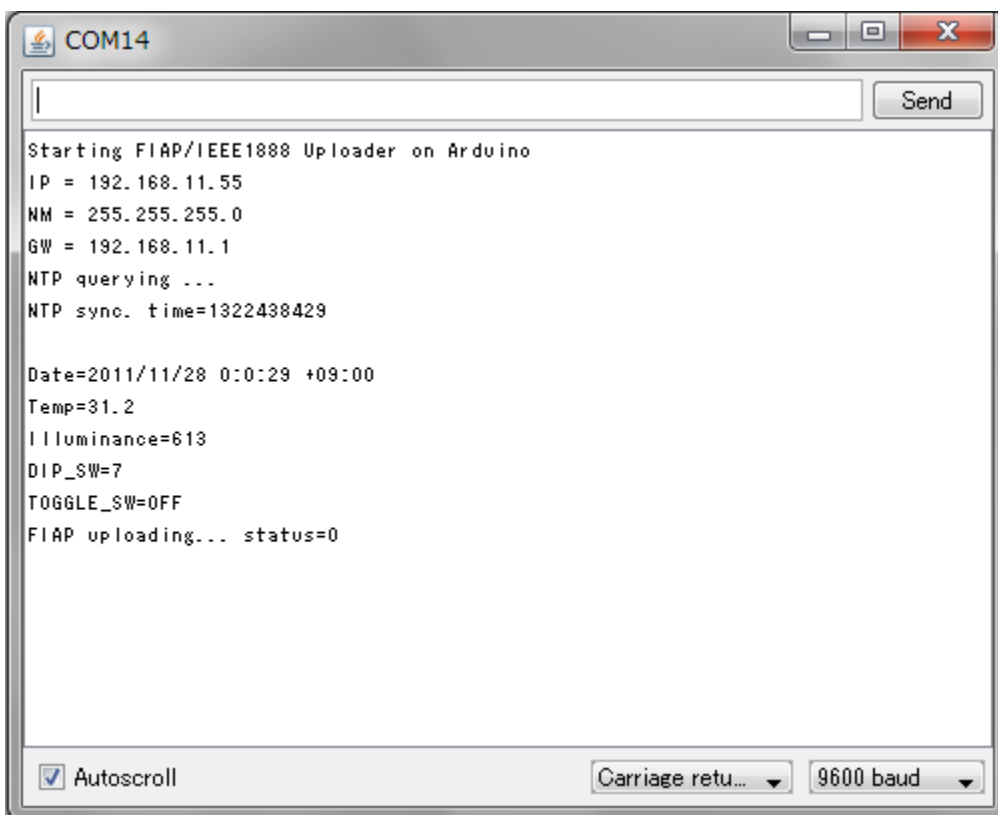


図 16: シリアルモニタの画面

## 7. IEEE1888 通信機能のプログラミング

ここでは IEEE1888 通信プログラミングの方法を述べます。Arduino IDE をオープンし、次のように記述し、必要なファイルをインクルードしてください。

```
#include <SPI.h>           // Ethernet シールド用
#include <Ethernet.h>      // Ethernet シールド用
#include <FIAPUploadAgent.h> // FIAPUploadAgent
#include <Time.h>          // 時刻関数・NTP 用
#include <Udp.h>           // NTP パケット送受信
```

IEEE1888 通信に限らず、ネットワーク接続を行うためには、SPI.h と Ethernet.h のファイルが必要になります。FIAPUploadAgent.h が IEEE1888(WRITE クライアント)の通信ライブラリを参照します。Time.h をインクルードすることによって、通信ボード上で、Unix timestamp での時刻管理が可能になります。Udp.h は、このボードを、ネットワーク上の NTP(時刻)サーバと UDP 通信させ、時刻同期を行うため、読み込んでおきます。

IEEE1888 の通信を行うには、まず、FIAPUploadAgent のインスタンスを作成する必要があります。その作成方法は、

```
FIAPUploadAgent  FIAP;
```

です。これは、どの関数にも属さないグローバル領域に書いておきます。

次に、setup 関数内で、

```
FIAP.begin( ... );
```

を実行しライブラリを初期化します。

loop 関数内で、

```
int ret=FIAP.post( ... );
```

を実行させることでサーバにデータを送信します。

ここで、setup 関数や loop 関数は、Arduino が標準で装備している関数で、setup 関数は起動時に一度だけ呼ばれ、loop 関数はその後、何度も呼ばれます。

FIAPUploadAgent が提供するメソッドの種類と、その引数や戻り値の意味定義は、表 1 および表 2 を、ご参照ください。

**表 1: FIAPUploadAgent の API**

メソッド名	引数	型	意味
begin	server_ip4	const uint8_t*	サーバの IPv4 アドレスを指定 {133, 11, 168, 97} などへのポインタ
	server_host	const char*	サーバのホスト名を指定(HTTPヘッダの Host パラメータに利用) “fiap-sandbox.gutp.ic.i.u-tokyo.ac.jp” などへのポインタ
	server_path	const char*	IEEE1888 サービスを提供している HTTP サーバのパス “/axis2/services/FIAPStorage” などへのポインタ
	server_port	uint16_t	サーバの TCP ポート番号 通常は 80 (HTTP) を指定する
	fiap_id_prefix	const char*	ポイント ID の共通部分をプレフィックスとして指定する “http://gutp.jp/Arduino/demo/” などへのポインタ
	戻り値	void	戻り値は無し
post	v	struct fiap_element*	サーバに送信するセンサデータ (fiap_element 構造体) の配列へのポインタ (表 2 参照)
	esize	uint8_t	fiap_element 構造体の個数
	戻り値	int	サーバとの通信状態を応答 成功: FIAP_UPLOAD_OK TCP 接続失敗: FIAP_UPLOAD_CONNFALL DNS 解決失敗: FIAP_UPLOAD_DNSERR (このエラー応答はない) HTTP サーバエラー: FIAP_UPLOAD_HTTPERR IEEE1888 エラー応答: FIAP_UPLOAD_FIAPERR

**表 2: fiap\_element 構造体**

変数	型	意味
cid	const char*	ポイント ID のポストフィックス (センサごとに個別に指定される) “Temperature” や “Illuminane” 等へのポインタ ポイント ID は、表 1 の fiap_id_prefix と組み合わせて生成される
value	char*	このポイントに結び付けられて送信される値(文字列)へのポインタ
year	uint16_t	送信する値の時刻 (年の部分)
month	uint8_t	送信する値の時刻 (月の部分) 1 - 12
day	uint8_t	送信する値の時刻 (日の部分) 1 - 31
hour	uint8_t	送信する値の時刻 (時の部分) 0 - 23
minute	uint8_t	送信する値の時刻 (分の部分) 0 - 59
second	uint8_t	送信する値の時刻 (秒の部分) 0 - 59
timezone	char*	送信する値の時刻 (タイムゾーン表記) “+00:00” などへのポインタ

以下、FIAPsensorsAPIHowto のプログラムを記載します(このプログラムは NTP サーバによる時刻同期機能は持たないため、Udp.h はインクルードしていません)。

----- FIAPsensorsAPIHowto.pde (ここから) -----

```
// Sample Code (Usage) of FIAPUploadAgent
//
// 2011/09/18 ver.1 H.Inoue & H.Ochiai
//
// ★の部分はセンサ毎に設定（書き換え）が必要な項目

#include <SPI.h>           // Ethernet シールド用
#include <Ethernet.h>     // Ethernet シールド用
#include <FIAPUploadAgent.h> // FIAPUploadAgent
#include <Time.h>         // 時刻関数

// 全体パラメータ
const unsigned int MEASURE_INTERVAL = 60; // アップロード間隔(秒)

// ローカル・ネットワーク設定
static byte local_mac[6] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // ★ローカル環境に合わせる
static byte local_ip4[4] = { 192, 168, 11, 55 }; // ★ローカル環境に合わせる
static byte local_gateway[4] = { 192, 168, 11, 1 }; // ★ローカル環境に合わせる
static byte local_netmask[4] = { 255, 255, 255, 0 }; // ★ローカル環境に合わせる

// IEEE1888 サーバ設定
static byte server_ip4[4] = { 133, 11, 168, 97 }; // ★利用するサーバに合わせる
static char server_host[] = "fiap-sandbox.qutp.ic.i.u-tokyo.ac.jp"; // ★利用するサーバに合わせる
static char server_path[] = "/axis2/services/FIAPStorage"; // ★利用するサーバに合わせる
static int server_port = 80;

// PointSet ID (=PointID のプレフィックス) の設定
static char fiap_id_prefix[] = "http://gutp.jp/Arduino/demo/"; // ★他と重複しない自分の ID を使うこと

// 値保持フィールド
char str_P1[] = "ON";
char str_P2[] = "OFF";
char timezone[] = "+00:00"; // UTC(GMT)

// 各ポイントの設定
struct fiap_element element[]=
{
  {"P1", str_P1, 0,0,0,0,0,0,timezone},
  {"P2", str_P2, 0,0,0,0,0,0,timezone},
};

// FIAPUploadAgent のインスタンス
FIAPUploadAgent FIAP;

// 初期設定ルーチン
void setup()
{
  Serial.begin(9600);

  // ネットワーク初期化
  Serial.print("IP = "); Serial.println(ip_to_str(local_ip4));
  Serial.print("NM = "); Serial.println(ip_to_str(local_netmask));
```

```

Serial.print("GW = "); Serial.println(ip to str(local_gateway));
Ethernet.begin(local_mac, local_ip4, local_gateway, local_netmask);

// FIAP ライブラリの初期化
FIAP.begin(server_ip4, server_host, server_path, server_port, fiap_id_prefix);

// 2011年11月10日00時00分00秒(UTC)を時刻(time)ライブラリに設定(あくまでサンプル)
setTime(1320818400);
}

// メインループ
void loop()
{
  int i;
  time_t t;

  // 各データに現在時刻をセット
  t = now();
  for(i = 0; i < 2; i++) {
    element[i].year = year(t);
    element[i].month = month(t);
    element[i].day = day(t);
    element[i].hour = hour(t);
    element[i].minute = minute(t);
    element[i].second = second(t);
  }

  // データ設定(文字列として格納)
  sprintf(str_P1, "ON");
  sprintf(str_P2, "OFF");

  // IEEE1888 アップロードを実行
  Serial.print("FIAP uploading... ");
  int ret = FIAP.post(element, 2);
  Serial.print("status="); Serial.println(ret);

  // 測定間隔だけ待つ
  delay(MEASURE_INTERVAL * 1000UL);
}

// IP アドレス(IPv4)をドットで区切られた文字列に変換する
char *ip_to_str(byte *ip)
{
  static char str[16];
  sprintf(str, "%d.%d.%d.%d¥0", ip[0], ip[1], ip[2], ip[3]);
  return(str);
}

// end of code
----- FIAPsensorsAPIHowto.pde (ここまで) -----

```

## 8. 周辺装置とのインタフェース(ハードとソフト)

### 8. 1. ハードウェア

IEEE1888 通信ボードのピンコネクタは、図 17 に示す通り、Arduino Duemilanove や Arduino Uno と互換になっています。ただし、pin10~13 は、TCP/IP コントローラで利用されているため、インターネット接続を行う場合は、これらのピンの利用には注意が必要です。また、pin0~1 は、USB シリアルインタフェースで利用しているため、USB シリアルポートを利用する場合は、これらのピンも注意して利用する必要があります。

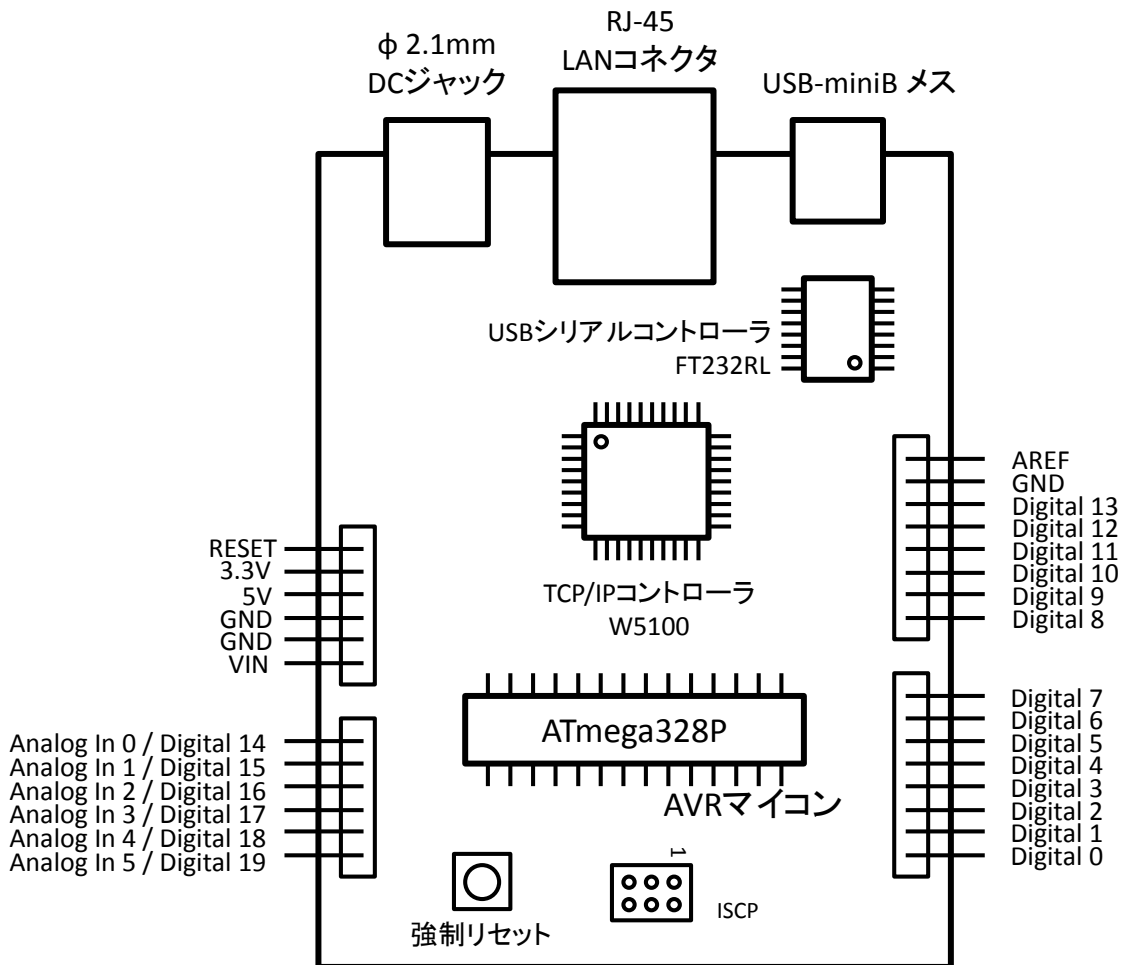


図 17: IEEE1888 通信ボードのピン配置図

## 8. 2. ソフトウェア

図 17 の各ポートへのソフトウェアでのアクセス方法(API: Application Programming Interface) は、各種用意されています(<http://arduino.cc/en/Reference/HomePage> を参照). 代表的なものとしては、

```
pinMode(...); // デジタルポートの入出力モードを指定
digitalRead(...); // 指定したデジタルポートの HIGH/LOW 状態を取得
digitalWrite(...); // 指定したデジタルポートに状態(HIGH or LOW)を設定する
analogRead(...); // 指定したアナログポートの電圧を AD 変換する(10 ビット)
analogWrite(...); // 指定したデジタルポートから PWM 波形を出力する (8 ビット PWM)
Serial.XXX(...); // USB シリアルとの通信 (ATmega328P のシリアル通信モジュール利用)
SoftwareSerial.XXX(...); // 汎用デジタルポートをシリアル通信ポートとして利用する
```

などがあります。以下、これらの利用方法を記載します。

### 8. 2. 1. デジタル値の読み込み

Digital 3 をデジタル入力ポートとして使用する場合は、

```
pinMode(3, INPUT);
```

と、`setup` 関数内などで宣言しておけば、

```
digitalRead(3); // HIGH or LOW を返す
```

と記述することで、そのポートの値(HIGH or LOW)を読むことができます。

### 8. 2. 2. デジタル値の設定

Digital 4 をデジタル出力ポートとして使用する場合は、

```
pinMode(4, OUTPUT);
```

と、まず宣言しておきます(`setup` 関数内などで).

その後、LOW 出力をする場合は、

```
digitalWrite(4, LOW);
```

とし、HIGH 出力する場合は、

```
digitalWrite(4, HIGH);
```

のように、記述します。

### 8. 2. 3. アナログ値の読み込み

Analog In 2 をアナログ入力ポートとして使用する場合は、

```
analogRead(2); // 0~1023 を返す
```

のように記述します。AREF(=通常 5V)の電圧に対する電圧比を、0~1023 で返します。1024 が AREF の電圧に対応します。

### 8. 2. 4. アナログ値(PWM)の出力

アナログ出力は、正式には PWM 比(0~255)でのデジタル出力となります。PWM 出力ができるのは、Digital 3, 5, 6, 9, 10, 11 です。

```
analogWrite(3, 128);
```

とすると、Duty 比 50%の PWM 信号を出力します。

### 8. 2. 5. USB シリアルとの通信

IEEE1888 通信ボードの Digital 0, Digital 1 は、それぞれ TX, RX として機能し、これらは 1kΩ の抵抗を介して、FT232RL(USB シリアル通信チップ)に接続されています。従って、Arduino に標準装備されている Serial インスタンスを使って、USB シリアルとの通信ができます。

まず、通信速度の設定のため、

```
Serial.begin(9600);
```

を、setup 関数内などで宣言します。

#### ※ USB シリアルへの書出し方法

USB シリアルに書き出す(つまり、接続されているコンピュータに送り出す)には、

```
Serial.println( "Hello World!!" );
```

と記述します。

(\*) println メソッドは最終文字の後、自動的に改行文字を出力します。print メソッドを使えば、そのような改行文字は出力されません。

#### ※ USB シリアルからの読み込み方法

USB シリアルから送られてきた文字を読み出す(つまり、接続されているコンピュータから送られてきた文字を読み取る)には、

```

while(Serial.available()){
    char c = Serial.read();
    ... 処理 ...
}

```

とします。

## 8. 2. 6. ソフトウェア・シリアル通信の設定

汎用の入出力ポートを使ってシリアル通信を行うために、**SoftwareSerial** ライブラリが標準で装備されています。**SoftwareSerial** は、通常の **Serial** と比べると、通信速度の上限が **9600bps** であったり、**Serial.available()**に相当するメソッドが使えなかったり、などの制限はありますが、簡単なシリアル通信には十分に耐えるものとなっています。以下が、それらの使い方です。

----- ソフトウェア・シリアル通信例 (ここから) -----

```

#include <SoftwareSerial.h>

// ピン番号の設定 (ハードウェア依存)
#define XBEE_TX 8
#define XBEE_RX 9

// XBee ZiqBee モジュールとのシリアル通信設定
// 汎用ピン をシリアル通信に使用する
SoftwareSerial xbeeSerial=SoftwareSerial(XBEE_RX, XBEE_TX);

// 初期化
void setup(){
    Serial.begin(9600); // USB シリアルとの通信速度を 9600bps に設定
    pinMode(XBEE_RX, INPUT); // RX を入力モードに設定
    pinMode(XBEE_TX, OUTPUT); // TX を出力モードに設定
    xbeeSerial.begin(9600); // XBee との通信を 9600bps で行う
}

// メインプログラム
void loop(){
    int n;
    char c;
    char str_buffer[100]; // 1 行の最大文字数を 99 文字と想定する
    while((c=xbeeSerial.read())!=0x0d){ // 改行文字を受信するまで str_buffer に読み込む
        str_buffer[n++]=c;
    }
    if(n>0){
        str_buffer[n]='\0'; // 最後を NULL 文字で埋める
    }
    Serial.println(str_buffer);
}

```

----- ソフトウェア・シリアル通信例 (ここまで) -----

## 9. シールドの回路設計

Arduino の世界では、ボード本体に接続する基板（ハードウェア・アプリケーション）のことをシールドと呼びます。ここでは、簡単に、その回路設計の例を示します。

### 9. 1. 学習用キットのセンサシールド

このキットに付属してくる学習用キットのセンサシールドの回路構成は、図 18 のようになっています。

「第6章 サンプルプログラムについて」で紹介したサンプルコードである、FIAPsensorsSimple, FIAPsensorsDHCP, FIAPsensorsCLI は、この回路構成を前提に組まれていました。

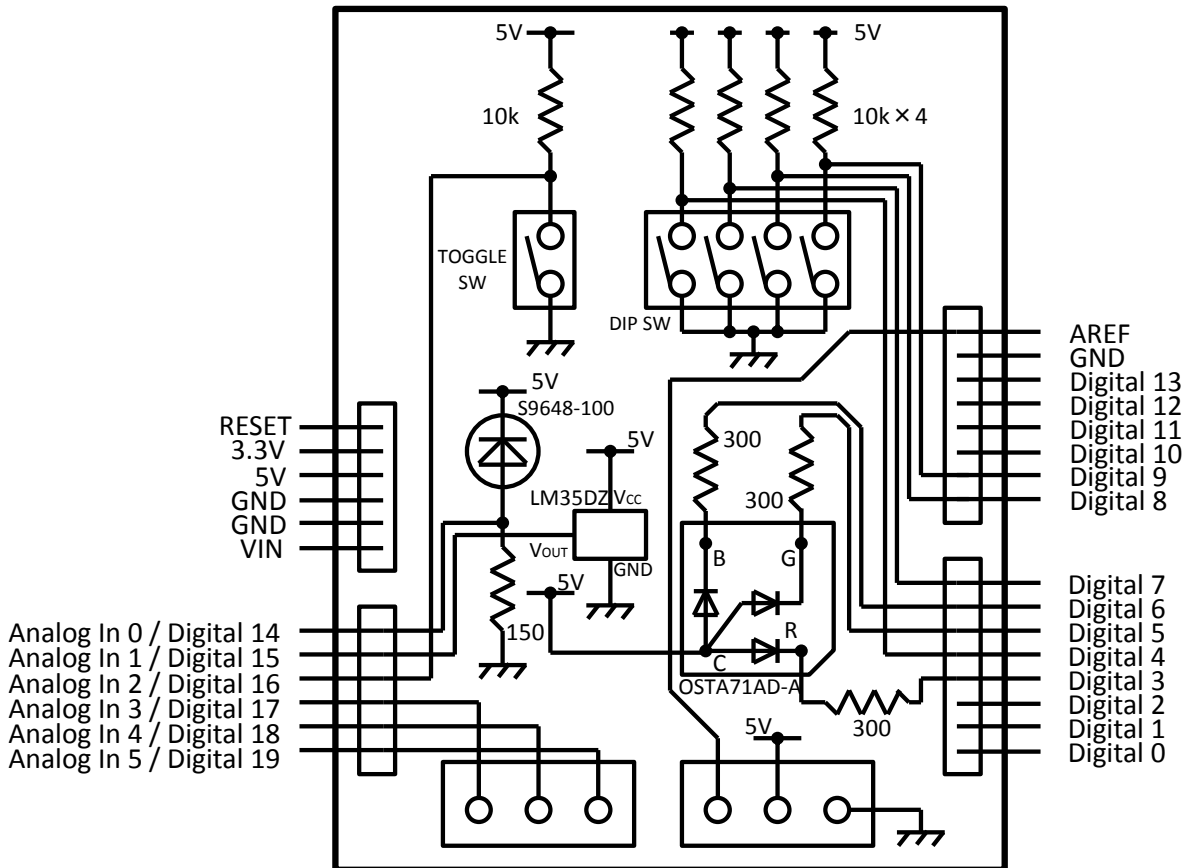


図 18: 学習用キットのセンサシールドの回路図

照度センサ，温度センサの出力信号は，それぞれアナログ入力ポートに接続しています．また，フルカラーLEDは，PWM出力のポートに接続し，ソフトウェアにより様々な色や明るさを表現できるようにしています．トグルスイッチや，DIPスイッチの信号線は，適当なデジタルポートに接続しています(pin 0, 1, 10~13 はすでに共同利用されているため避けています)．

## 9. 2. ZigBee(XBee)モジュールの接続例

IEEE1888 通信ボードに ZigBee モジュールを接続することで，IEEE1888 と ZigBee の変換 GW を実現することができます．その時に用いられる回路構成の例を図 19 に示します．

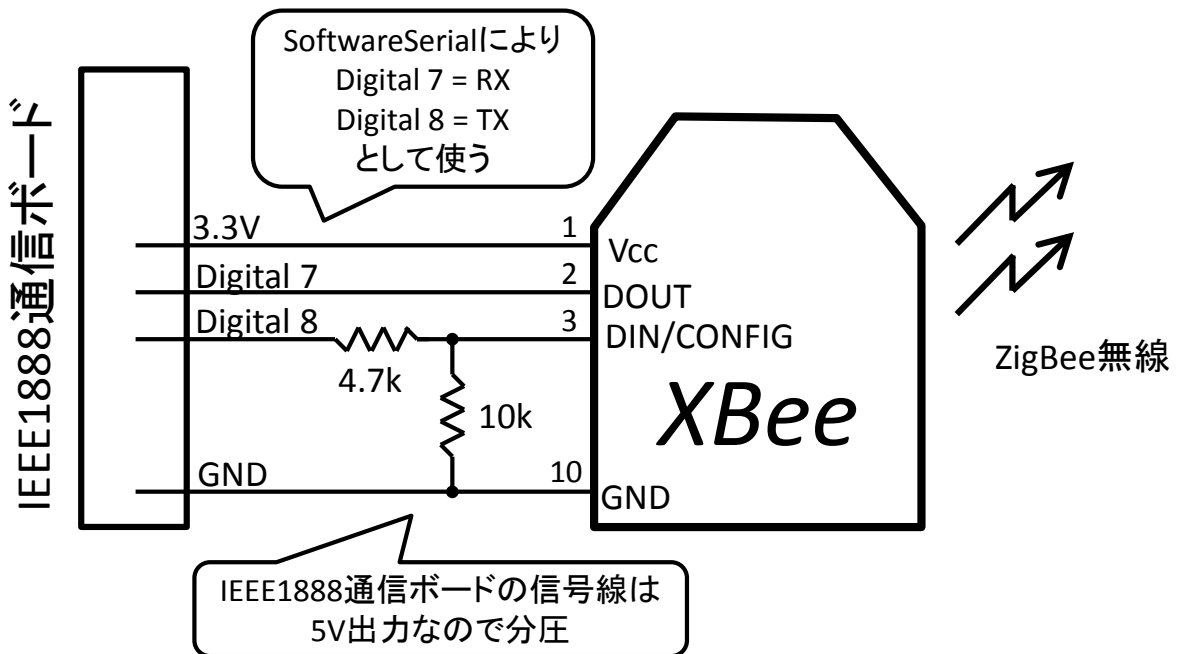


図 19: ZigBee モジュール(XBee)の接続例

### 9. 3. RS232C ドライバの接続例

IEEE1888 通信ボードに RS232C ドライバを接続することで、IEEE1888 と RS232C の変換 GW を実現することができます。その時に用いられる回路構成の例を図 20 に示します。

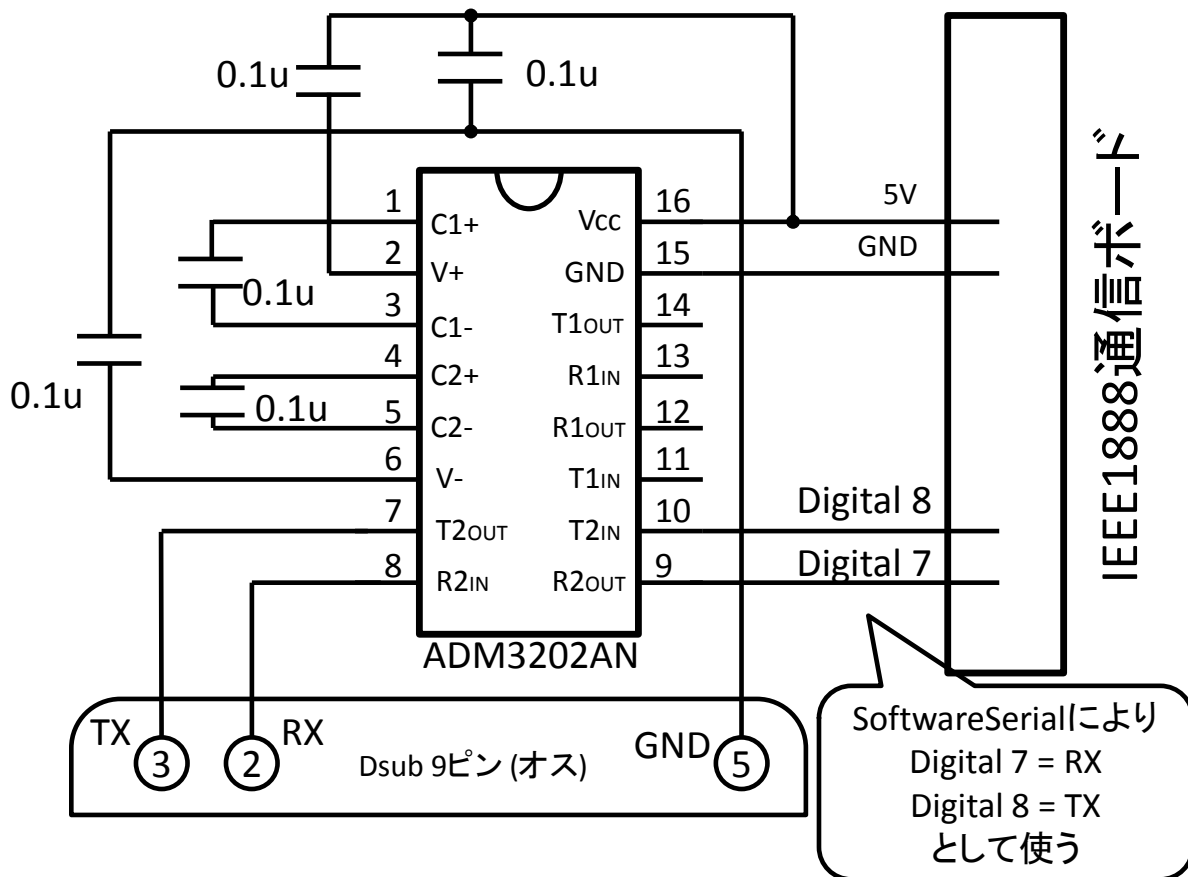


図 20: RS232C ドライバの接続例

## 9. 4. パルス出力端子の接続例

IEEE1888 通信ボードに、電力メータなどからのパルス出力信号を取り込むための、回路構成例を図 21 に示します。この回路例は、オープンコレクタ出力(フォトカプラなど)でパルス出力される機器へのインタフェース回路となっています(パルス出力の形態に応じて、この回路は異なってきますので、利用においては注意が必要です)。

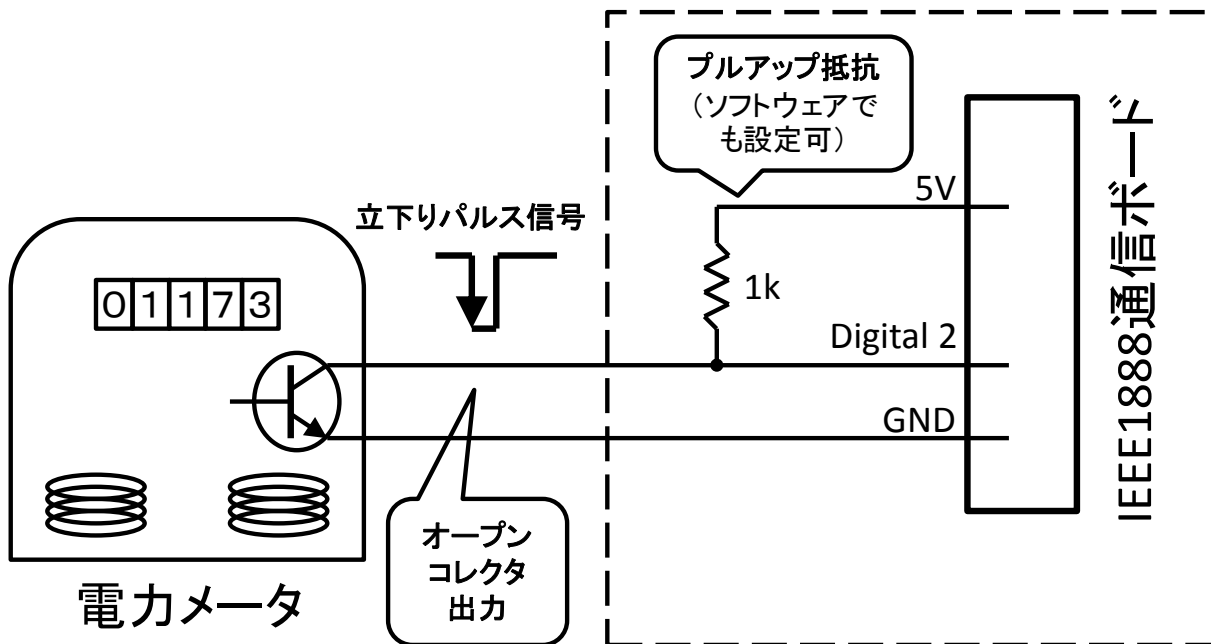


図 21: パルス出力信号とのインタフェース回路例

ここで接続している Digital 2 ポートは、割り込み生成が可能なポートとなっています。次のプログラムによって立下りパルスソフトウェアで扱えるようになります(もちろん、ピンの状態をポーリングすることで立下りを検出する方法もあります)。

```
void setup(){
  Serial.begin(9600);
  // 0 = digital2; 1 = digital3; パルス検出→ pulse_detect 関数を呼ぶ
  attachInterrupt(0, pulse_detect, FALLING);
}
void pulse_detect(){
  Serial.println( "Pulse Detected!!" );
}
```

## 9. 5. シールドの実装方法について

Arduino シールドの設計は自由自在です。Arduino のピン配置の関係で、通常のユニバーサル基板は搭載しにくいですが、Arduino 向けのユニバーサル基板(例えば、サンハヤト製の UB-ARD01)を購入すれば、様々なシールドを実装することができます。「しっかりした基板にしたい」というご要望があれば、本マニュアル末尾の連絡先までメールをお送りください。IEEE1888 の普及にとって重要な案件と判断されれば、積極的に基板化を進めてまいります(その際にプロトタイプが既にあるとスムーズです)。ユニバーサル基板に XBee モジュールや RS232C ドライバを搭載した様子を、図 22、図 23 に示します。

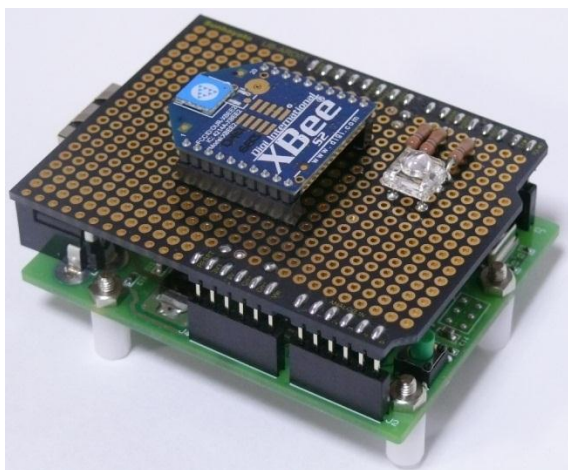


図 22: XBee モジュールを搭載した様子

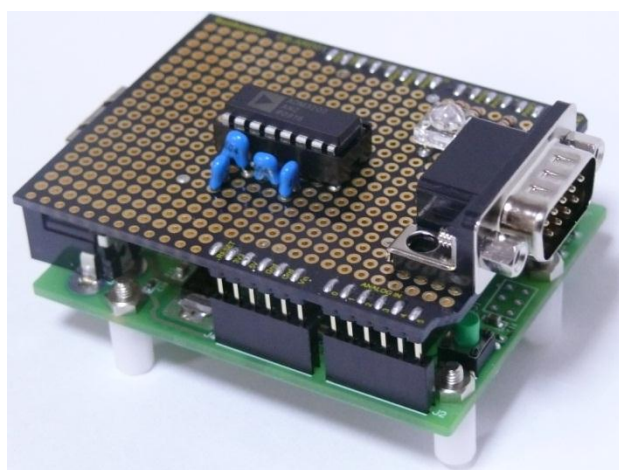


図 23: RS232C ドライバを搭載した様子

## 10. その他

IEEE1888 通信ボードは，Arduino の回路をベースとしています．この Arduino は，“Creative Commons Attribution Share-Alike license” の下で利用可能であり，商用目的でも自由に利用できるものとなっています．IEEE1888 通信ボードのパターン設計および，本マニュアルの作成は，東京大学の落合秀也が行いました．ボード製造はフタバ企画が行っています．本マニュアルは自由に配布して構いません．

なお，この通信ボードを使用したことにより発生した事故や損害は，東京大学，東大グリーン ICT プロジェクト，落合秀也，フタバ企画は責任を負いかねますので，予めご了承ください．

お問合せ先： 東京大学 落合秀也 (ochiai@vdec.u-tokyo.ac.jp)