UARDECS アプリケーションガイド

（UARDECS　Ver0.8 以降用）

1.0 版

Written by Hideto Kurosaki

2017. 1. 6.

このアプリケーションガイドはUECS研究会ニュースレター2016年3月号に配信されたものを加筆修正したものです。

第1章 UARDECS の活用

1 はじめに

Arduino は世界中で使用されているマイコンボードで PC と USB 接続してプログラムを書き換えることで様々な用途に活用できるものです。日本でも概ね 2000～6000 円の価格帯で販売されており、通販などで容易に入手できます。Arduino の開発環境は無料で公開されており、C 言語に似たスケッチと呼ばれる言語で記述でき、使い方に関しては無数の文献が web 上で公開されています。今回は、これを利用して UECS 対応ノードを自作する方法を紹介します。

2 Arduino と Raspberry Pi

両方とも大変普及したマイコンボードですが、Arduino は Raspberry Pi より単純な構造をしています。一方、Raspberry Pi はパソコンに準じる機能を持ち、ソフトウェアを重視した開発に向いています。スペックでは Raspberry Pi の方が Arduino より優れていますが、OS を動作させるオーバーヘッドが生じるため、センサなどの入力に高速に応答するには Arduino が向きます。Arduino は CPU 内部のフラッシュメモリに書き込まれたプログラムを実行しますので OS の書かれた SD カードからプログラムを読み込んで動作する Raspberry Pi より高速に起動し、プログラム記録メディアの寿命を気にする必要がないという利点があります。また、Arduino は回路が公開され、独自に設計した互換機を合法的に製造できます。一方、Raspberry Pi は大容量なメモリと SD カードを利用して複雑なプログラムを実行でき、ネットワーク関係のソフトウェアも充実しているので相互補完的に利用するのが良いと思われます。Raspberry Pi 用の UECS 対応開発環境には UECS-Pi があります。

3 UARDECS とは

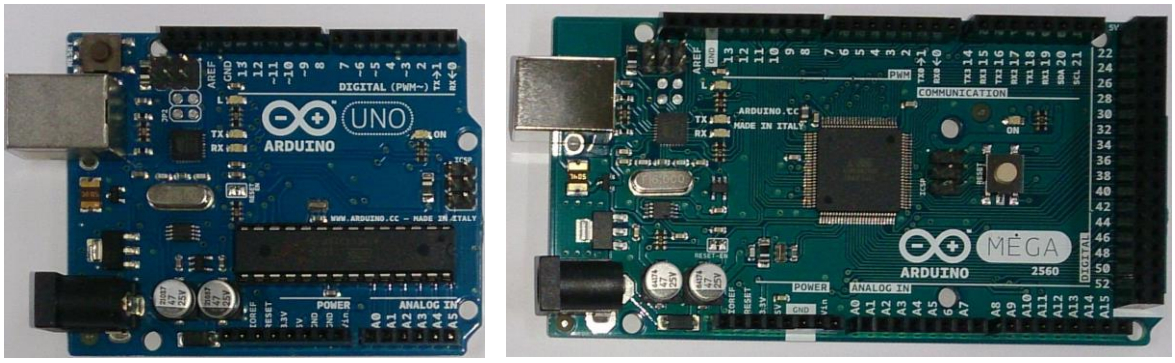
UARDECS は Arduino 用に UECS 対応ノードの開発を支援するためのミドルウェアです。これは、安場氏が開発したものを筆者(黒崎)が改修したもので、以下のサイトから入手できます。本稿執筆時点で最新版は 0.8 です。

UECS ノード開発用ミドルウェア(http://uecs.org/arduino/uardecs.html)

このミドルウェアは、UECS の通信実用規約"1.00-E10" に対応した通信文の送受信管理とマイコンボードに搭載した http サーバを利用した設定画面の生成、設定値の不揮発性メモリへの自動読み書き等を行うことができます。インストールの方法はマニュアルがあるので割愛させて頂きますが、本稿執筆時点で Arduino IDE 1.8.0(配布元: http://www.arduino.org/)がリリースされており、これに対応しています。UARDECS は発展途上のソフトウェアであり、今後も改修を行っていく予定です。

4 機種の選定

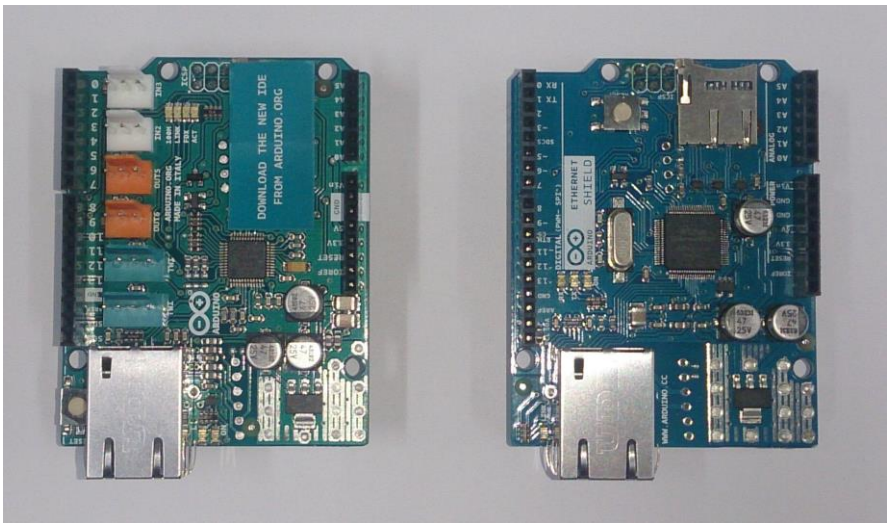
この章の末尾に付録として、Arduino の主要機種のスペックを掲載しておきます。Arduino には数多くの機種が存在しますが、大きく分けて３種類に分類されます。まず、最も標準的な Atmega328 を搭載した UNO とその性能向上型の MEGA、次に USB 通信機能を CPU に統合し、ATmega32u4 を搭載したグループ。そして、ARM 系の CPU を搭載し 3.3V で動作する新機種です。このうち、ARM 系機種は高性能なのですが不揮発性メモリの EEPROM を搭載していません。これは、電源を切った時に設定情報などを保持できないという大きな欠点になります。また UARDECS で使用しているライブラリの幾つかは動作しませんので、これらの機種は対象外とします。次に、Leonard をはじめとする ATmega32u4 を搭載したグループですが、どの機種も UNO よりユーザーが使用できるフラッシュメモリの総量が少ないのです。一見僅かな差ですが、UARDECS は最低限のプログラムでも 28000byte 程度を必要としますので、ATmega32u4 搭載機では UARDECS のコンパイルはできてもフラッシュメモリ不足でプログラムを転送できないという状況に陥ります。これは、ネットワーク通信機能を持つ Leonard Eth や Yun を使えないという点で少々痛いのですが、筆者の PC では Leonard と相性が悪いのか、プログラムの書き込み中にフリーズし、ブートローダーを壊す事故が発生しましたので信頼性の面からもこの機種は敬遠することにしました。実は、これまで UECS の開発用に最も扱いやすいのは Arduino Ethernet だったのですが、非常に悔しいことにこの機種は生産終了で入手できなくなってしまいました。Arduino Nano は小型機で外形が他の Arduino と異なるため、イーサネットシールドをそのままでは搭載できません(配線し直せば可能)。従って、**現時点で使えるのは UNO と MEGA のみに絞られます**。



Arduino UNO(左)と Arduino MEGA 2560(右)

5　ネットワーク通信機能の付加

Arduino UNO と MEGA は単体で LAN ポートがありません。そのため、別途 LAN ポートは追加しなくてはいけません。純正品ですと Ethernet Shield 2 というものが利用でき、およそ 3000 円ぐらいです。旧機種では Ethernet Shield というものがありましたが、これは生産終了になっており性能も劣るので、いまさら入手するメリットはありません。



Ethernet Shield 2(左)、Ethernet Shield(右、既に生産終了)

6　その他、必要なもの

(1) AC アダプタ

Arduino は PC と USB 接続している間は PC からの電力供給で動作しますが、PC から外して動作させるためには別途電源が必要です。最も簡単なのは AD-L50P100 や AD-B50P200 などを購入して PC の時と同じように USB 端子に接続することです。スマートフォン用の AC アダプタなども USB 端子が付いていますので代用できます。もう一つの方法は、Arduino に付いている DC ジャックに AC アダプタを接続することで、外形 5.5mm 内径 2.1mm のプラグが適合します。DC ジャックを使用する場合、5V より少し高い電圧が必要で通常は 9V を給電します。GF12-US0913 などが使えると思われます。ただし、DC ジャックを使用する場合はレギュレータの発熱が発生します。12V の AC アダプタも使用できますが、さらに発熱が多くなります。必要な電力量は LAN 接続時で最低 200mA ですので、少なくともその 2 倍は見積もったほうが良いでしょう。他に様々なデバイスを接続すれば消費電力はその分だけ増加します。

(2)ブレッドボードとジャンパー線

例えば、BB-801 や EIC-102BJ など1つぐらいはあったほうが良いです。さらに、ジャンパーケーブル(オスーオス型、オスーメス型)を揃えておくと試作がやりやすくなります。

(3)抵抗と LED

抵抗は最低限、1kΩと 10kΩがあるとプルアップやプルダウンに多用しますし、このぐらいの抵抗をつないでおけば間違って信号線を短絡させても部品を壊さずに済みます。何をどのぐらい揃えれば良いか分からない場合、まとめて買っても大した価格ではありません。LED は赤色の発光効率が良く 2V 以上の電圧で点灯できますので 1kΩの抵抗をつないで Arduino の動作確認に使用します。

(4)工具類

ちょっとした工作が必要になる場合は、小型のバネ付きニッパー、ワイヤーストリッパー、ハンダ付けが必要な場合は、はんだゴテ(白光 No.984-01)、糸ハンダ(白光 FS402-01)、コテ台(コテ台 No.603)などを用意します。

(5)開発用 PC とソフトウェア、LAN ケーブル、HUB

開発用 PC(WindowsPC を想定しています)には Arduino IDE をインストールします。さらに、ノードの動作テスト用に UECS 用パケット送受信ツールをダウンロードして準備しておきます。複数のノードを連動させる実験をする場合、HUB と LAN ケーブルを忘れないようにしましょう。

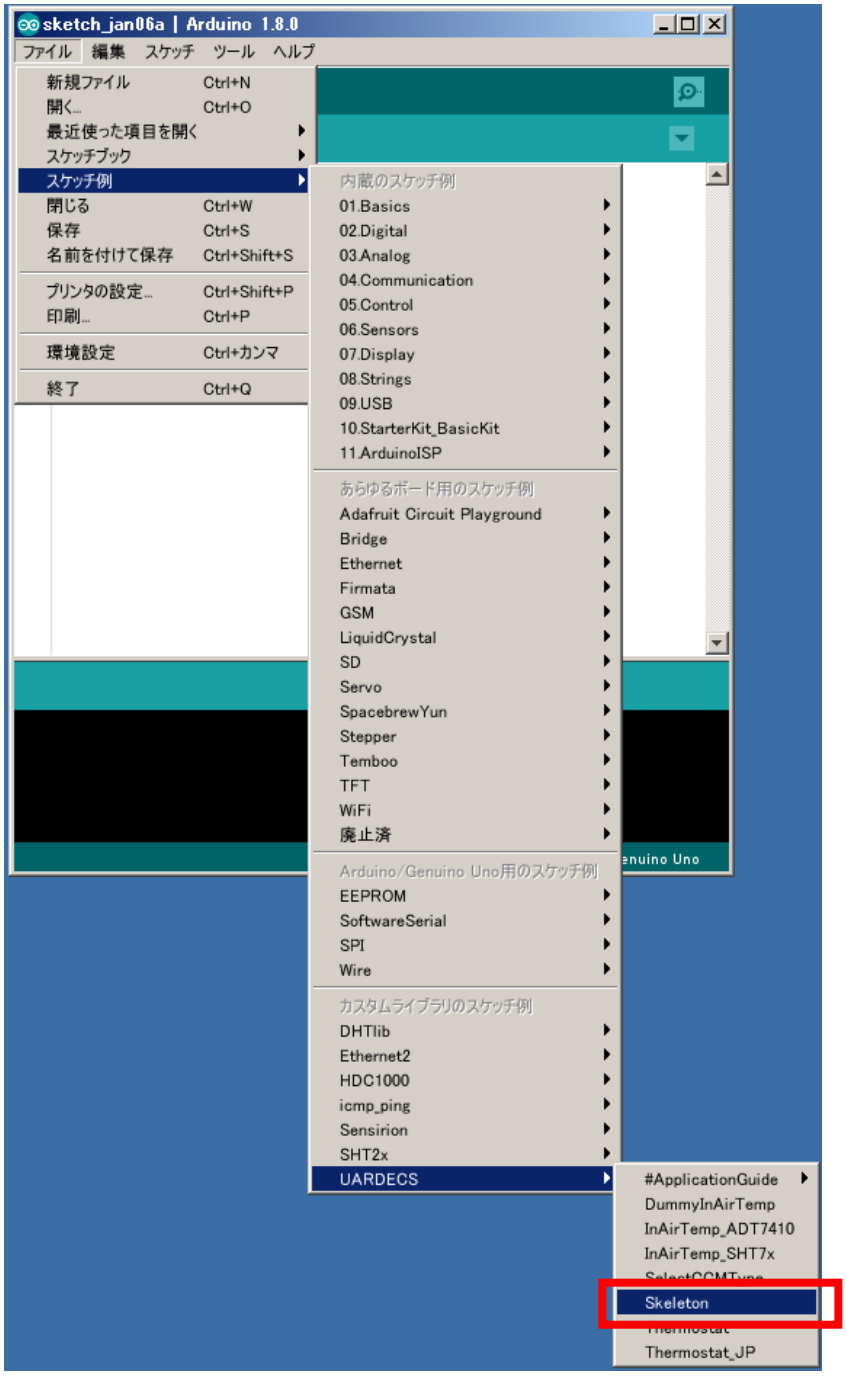
7 UARDECS のサンプルプログラム

UARDECS Ver0.8 には以下のサンプルが付属します。

#ApplicationGuide	このアプリケーションガイドに記載されたプログラムのソースコード
Skeleton	何もしないプログラム。コンパイルが通る最低限の宣言が記述されている。
DummyInAirTemp	ユーザーが Web インターフェースに入力した値を InAirTemp (気温)と見なして送信させる。
Thermostat	InAirTemp (気温)を受信して、ユーザーが Web インターフェースに入力した値と比較し 1 か 0 を出力する。
Thermostat_JP	Thermostat の表示を日本語化したサンプル。
InAirTemp_ADT7410	温度センサ ADT7410 の動作テスト、実物のセンサが必要。
SelectCCMType	ユーザーが CCM の種類を選択して変更するテスト。
InAirTemp_SHT7x	温湿度センサ SHT75/SHT71/SHT15/SHT11 の動作テスト、実物のセンサと制御用のライブラリが別途必要。

8　何もしないプログラム

下準備として、Ethernet Shield2 を装着した Arduino UNO を PC に USB 接続して下さい。Arduino IDE を起動し、ポートの設定画面で COM ポートの設定を確認したらサンプルプログラム Skeleton を読み込んでみます。マニュアル通りに UARDECS がインストールされていれば以下の場所から読み込めます。



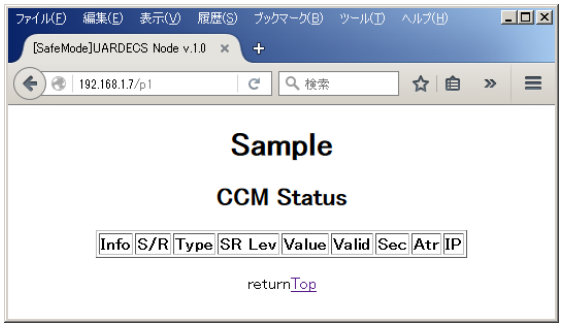
Skeleton の読み込み

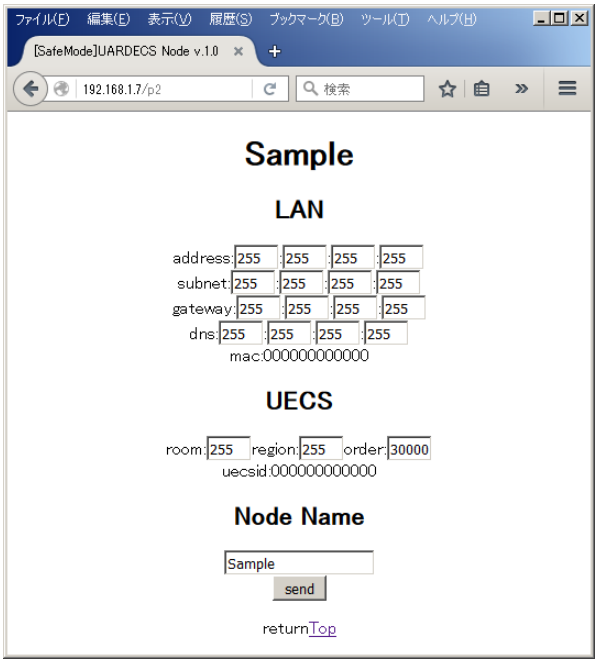
以下にSkeletonのソースコードを示します(簡略化のためコメント行を消しました)。読み込んだらIDE 左上の→ボタンを押して Arduino に書き込んでみてください。このプログラムは、単にコンパイルを通るだけで、CCM が1つも作成されていません。そのため、まだ UECS ノードとしては機能しません。ソースコード内に書かれているのは、UARDECS の起動に最低限必要な設定項目で、セーフモードジャンパーの設定(IP アドレスが分からなくなった場合に強制的に IP アドレスを設定するスイッチ)、ノードの機種名、製造元、UECSID、ノードの初期名、MAC アドレスなどです。(注意: MAC アドレスはノード毎に違う値が必要です、Ethernet Shield2 の裏に書いてある値を入力して下さい)

```
//Sample skeleton
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>

const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;

const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 0;
UECSCCM U_ccmList[U_MAX_CCM];

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//本当はここで CCM を初期化する

}
void OnWebFormRecieved(){}
void UserEverySecond(){}
```

```
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
UECSsetup();
}
```

Skeleton の状態でも Web サーバは機能しています。LAN ケーブルで Arduino を PC に接続し、ブラウザでアクセスすると以下のような画面を見ることができます。(工場出荷時の Arduino では IP アドレスが 192.168.1.7 になります。詳細は UARDECS マニュアル5章を参照)





Skeleton を実行中の Arduino にアクセスした場合の応答

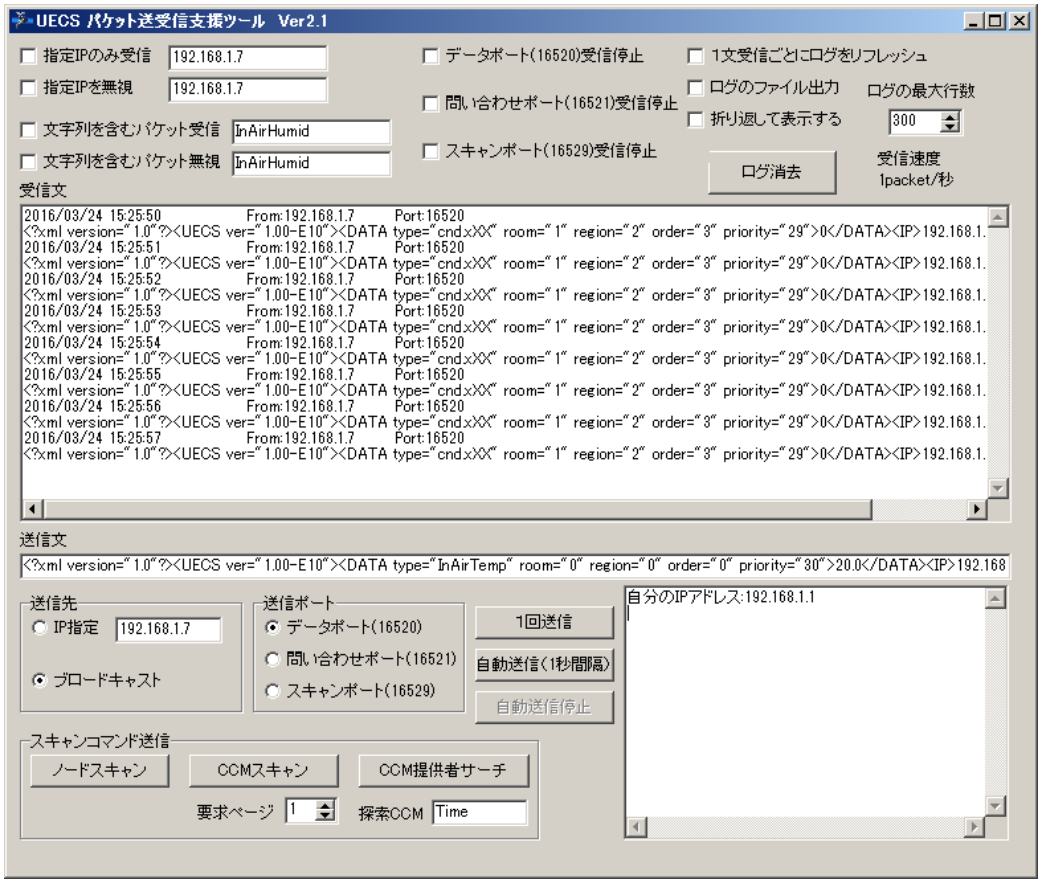
9　CCM を作成して自動送信させる

Skeleton に CCM を1つ作成するには以下の場所を変更します。この例では” cnd.xXX”という送信 CCM を作成しました。青字が変更箇所、赤字が新しく追加された箇所です。1つの CCM に3種類の文字列(CCM の説明,type,単位)が必要です。この文字列は必ず関数の外で宣言してグローバル変数にします。次に UserInit()の中で UECSsetCCM 関数を実行して CCM を宣言します。このサンプルプログラムは#ApplicationGuide の中の Sample1SendCCM に収録されています。

```
//Sample1SendCCM
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>

const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;

const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 1;//CCM の総数を 1 に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM 定義用の素材、被らないように適当な変数名で 3 つ宣言(必ず PROGMEM を付ける)
const char ccmInfoTest[] PROGMEM= "Test";//CCM の説明(Web でのみ表示)
const char ccmTypeTest[] PROGMEM= "cnd.xXX";//CCM の Type 文字列
const char ccmUnitTest[] PROGMEM= "";//CCM の単位(この場合単位無し)

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;
```

```
//UECSsetCCM(送受信の区分[true で送信],通し番号[0 から始まる],CCM 説明,Type,単位,priority[通常は 29],少数桁数[0 で整数],送信頻度設定[A_1S_0 で 1 秒間隔])
UECSsetCCM(true,0 ,ccmInfoTest,ccmTypeTest,ccmUnitTest,29,0,A_1S_0);
}
void OnWebFormRecieved(){}
void UserEverySecond(){}
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
UECSsetup();
}
```

PC に LAN ケーブルで繋いだ Arduino にこのプログラムを書き込んでから UECS 用パケット送受信ツールで見てみると、1 秒間隔で CCM が送信されているのが分かります。



CCM 作成後の応答を UECS 用パケット送受信ツールで見た時の画面

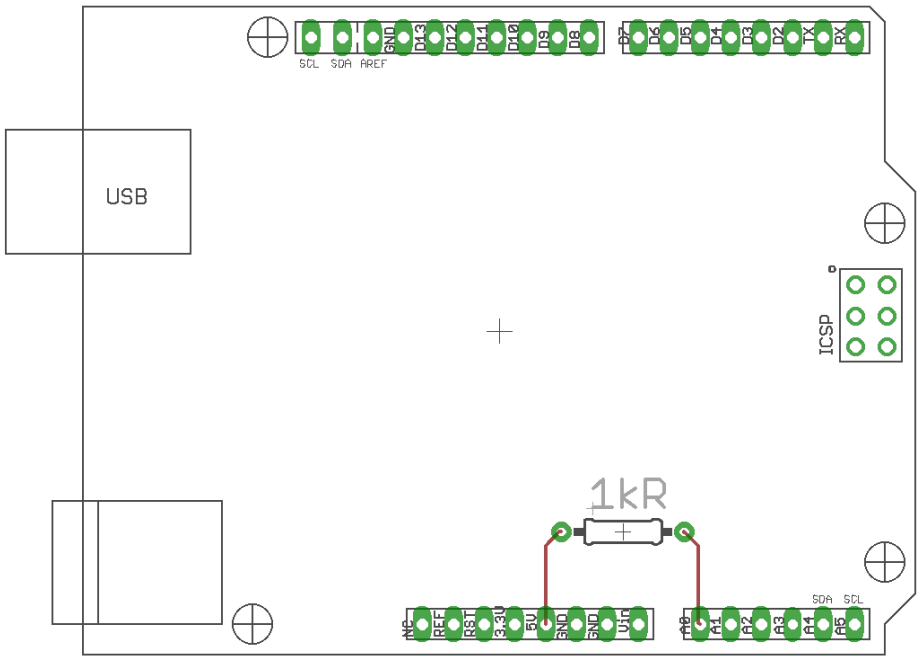
10 アナログ入力値の送信

先ほどのプログラムでは、作成した送信 CCM が自動送信される様子を示しましたが、そのままでは値を設定していないので常に 0 が出力されるだけです。そこで、Arduino のアナログ入力値を CCM として出力してみます。変更点は以下の色付きの箇所です。U_ccmList[0]の[0]の部分は UECSsetCCM 関数の 2 番目の引数で指定した通し番号です。このサンプルプログラムは #ApplicationGuide の中の Sample2SendAnalogVal に収録されています。

```
//Sample2SendAnalogVal
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>

const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;

const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 1;//CCM の総数を 1 に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM 定義用の素材、被らないように適当な変数名で 3 つ宣言(必ず PROGMEM を付ける)
const char ccmInfoTest[] PROGMEM= "Test";//CCM の説明(Web でのみ表示)
const char ccmTypeTest[] PROGMEM= "cnd.xXX";//CCM の Type 文字列
const char ccmUnitTest[] PROGMEM= "";//CCM の単位(この場合単位無し)

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;
```

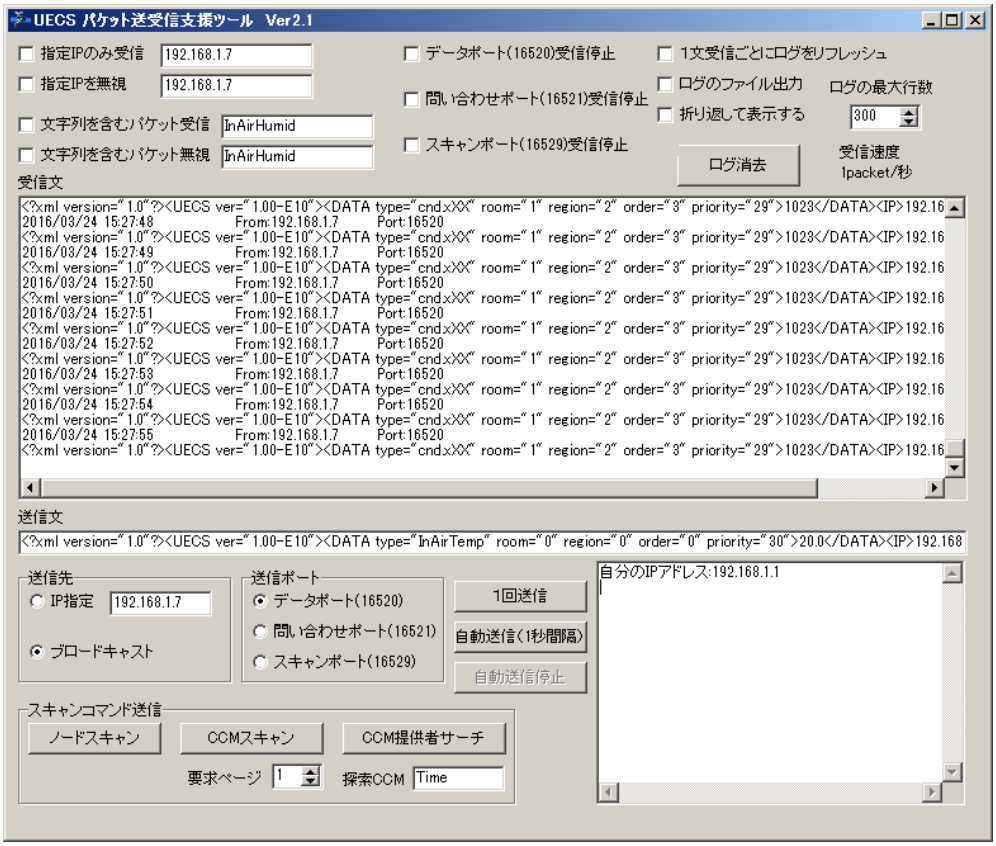
```
//UECSsetCCM(送受信の区分[true で送信],通し番号[0 から始まる],CCM 説明,Type,単
位,priority[通常は 29],少数桁数[0 で整数],送信頻度設定[A_1S_0 で 1 秒間隔])
UECSsetCCM(true,0 ,ccmInfoTest,ccmTypeTest,ccmUnitTest,29,0,A_1S_0);
}
void OnWebFormRecieved(){}

void UserEverySecond()
{
U_ccmList[0].value=analogRead(0);//CCM に AD コンバータの値を書き込む
}
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
UECSsetup();
}
```

プログラムでは Arduino の A0 端子の電圧を読み取るように設定しました。次に、UNO の A0 と 5V を 1kΩの抵抗でつないでみます。実は抵抗無しで直結しても大丈夫ですが、安全のためです。



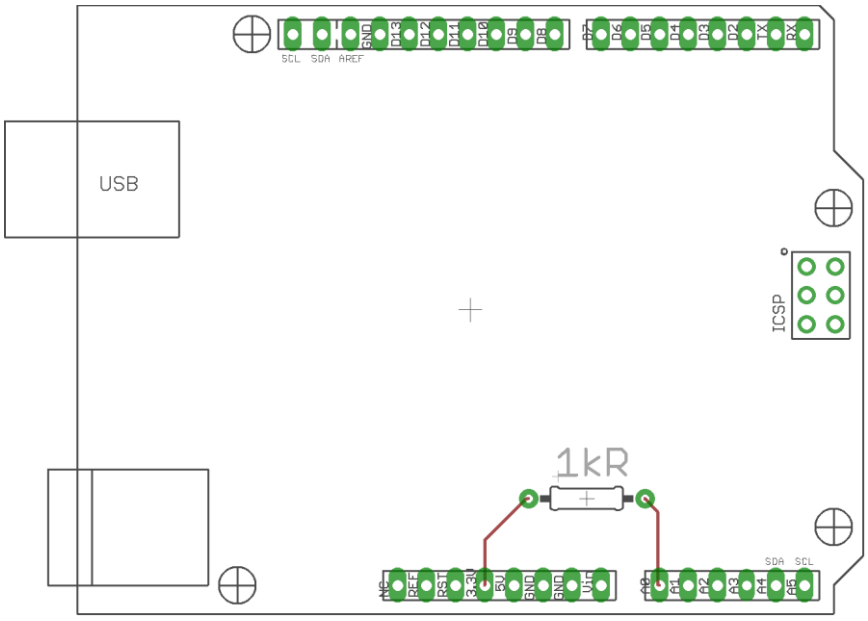
結線箇所(5V)

※図は UNO 本体ですが、Ethernet Shield 2 を載せていると見なして下さい

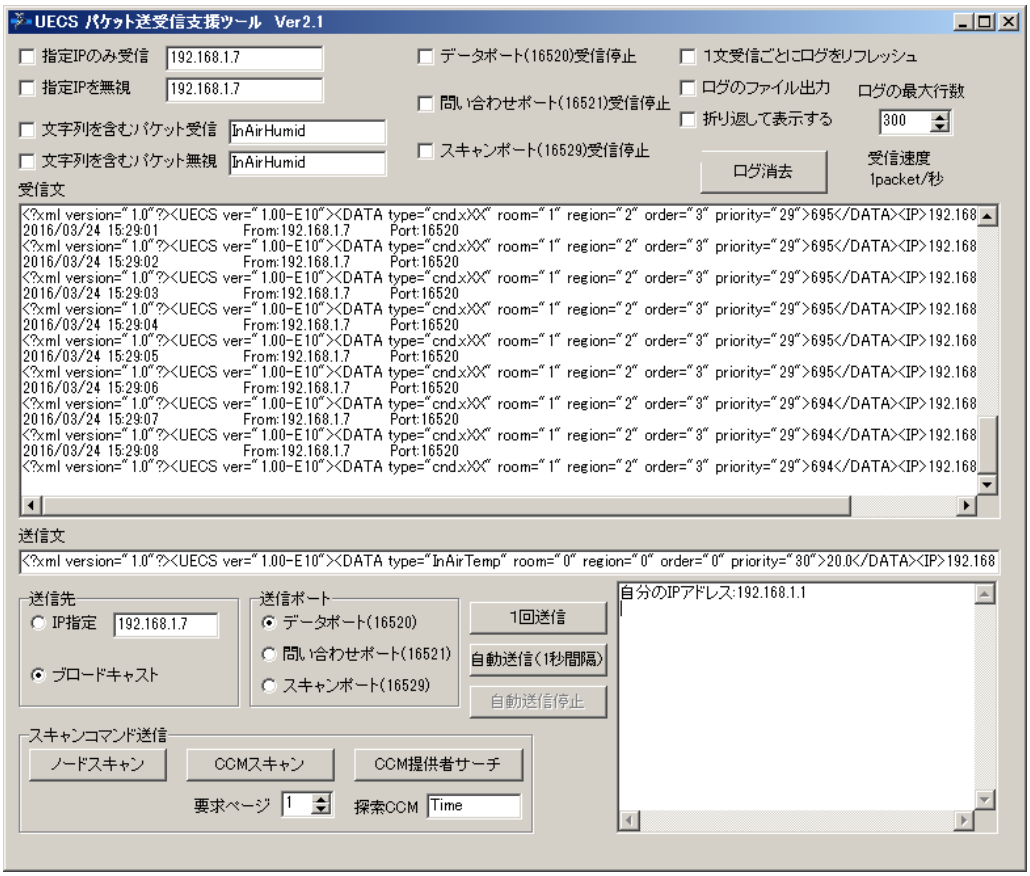


5V 入力時の応答

A0 端子に 5V を入力した結果、CCM の値は 1023 に変化しました。Arduino UNO と MEGA の AD コンバータは標準設定で 0～5V の電圧を 0～1023 で示すので、これで正常です。次に、A0 を 3.3V の端子に接続してみます。



結線箇所(3.3V)



3.3V 入力時の応答

A0 端子に 3.3V を入力した結果 CCM の値は大体 694～695 の間を行ったり来たりしているのが分かります(もしかすると個体差があるかもしれません)。ここから実際の電圧を計算する場合、694.5×5÷1023≒3.4V となり、大体正しい電圧を示していることが分かります。これで電圧を CCM として出力するノードが作れます。

注意点ですが、A0 端子に何も接続しない場合、空気中の静電気を拾ってしまい、出てくる値はメチャメチャになります。これはピンをデジタル入力として使う場合も同じで、何も接続されていないピンの値は不定になります。

11 セーフモードについて

サンプルプログラムではD3ピンをGNDに繋がない限り常にセーフモードで起動するようになっています。セーフモードではノードの設定値が初期化され、IP アドレスも自由に設定できません。ノードの IP アドレスなどを自由に設定したい場合、D3 ピンを 1kΩの抵抗を介して GND に接続するか、

```
const byte U_InitPin_Sense=HIGH;
```

と記述された行を

```
const byte U_InitPin_Sense=LOW;
```

に変更してセーフモードを抜けるようにして下さい。

付録　Arduino の機種

Arduino には大変多くの機種があり、一見するとどれがどう違うのか分からなくなりますが、大雑把に特徴をまとめておきます。ただし、ここに表記されているのは出回っている一部のみであり、互換機を含めればさらに膨大な種類があります。

機種名（一部）	駆動電圧 (V)	搭載ポート	全入出力ピン (アナログ入力/出力)	メイン CPU	サブデバイス
Arduino UNO R3	5	USB	20(6)	ATmega328	ATmega16u2 (USB-シリアル変換)
Arduino Mega 2560 R3	5	USB	70(16)	ATmega2560	ATmega16u2 (USB-シリアル変換)
Arduino Ethernet R3 ※1	5	LAN	20(6)	ATmega328	W5100 (LAN 制御用)
Arduino Nano Ver3.x	5	MiniUSB	22(8)	ATmega328	FT232RL (USB-シリアル変換)
Arduino Leonardo	5	MicroUSB	20(12)※2	ATmega32u4	
Arduino Leonardo Eth	5	MicroUSB LAN	20(12)※2	ATmega32u4	W5500 (LAN 制御用)
Arduino Micro	5	MicroUSB	20(14)	ATmega32u4	
Arduino Yun	5	MicroUSB USB LAN Wifi	20(12)※2	ATmega32u4	Atheros AR9331 (LAN/Wifi 制御用)
Arduino Due	3.3	MicroUSBx2	70(12/2)※3	SAM3X8E	ATmega16u2 (USB-シリアル変換)
Arduino M0	3.3	MicroUSB	20(6/1)※3	SAMD21G18A	
Arduino M0 Pro	3.3	MicroUSBx2	20(6/1)※3	SAMD21G18A	Atmel a06-0736 (オンボードデバッガ)

※1 既に生産終了

※2 ICSP 端子も使用する場合、総ピン数+3

※3 アナログ入力端子はデジタル入出力には使えない、デジタル入力端子に内蔵プルアップ機能が無い

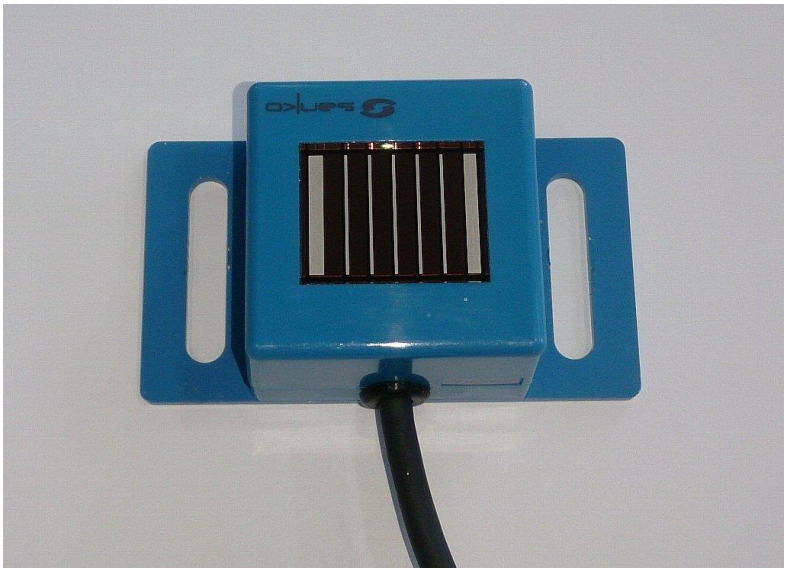
機種名（一部）	フラッシュメモリ（byte）	SRAM（kbyte）	EEPROM（kbyte）	特徴
Arduino UNO R3	32256	2	1	最も基本的な機種。 単に Arduino と言ったらこれを示す。
Arduino Mega 2560 R3	253952	8	4	UNO のピン数とメモリを大幅に増量したもの。処理速度は変わらない。一部のピンの機能は UNO と異なるので注意が必要。
Arduino Ethernet R3	32256	2	1	UNO から USB を外し Ethernet Shield の機能を付加したもの。
Arduino Nano Ver3.x	30720	2	1	UNO の小型版。形状が全く異なる。UNO よりピン数が増えている。
Arduino Leonardo	28672	2.5	1	CPU に USB-シリアル変換機能を統合して回路を単純化したものだが、内部配線が UNO とはかなり異なる。UNO よりプログラム用メモリが少ない。PC に接続し擬似的なキーボードやマウスのように動作させる事ができる。
Arduino Leonardo Eth	28672	2.5	1	Leonardo に Ethernet Shield 2 の機能を付加したもの。
Arduino Micro	28672	2.5	1	Leonardo の小型版だが Nano ともピン配置が異なる。
Arduino Yun	28672	2.5	1	Leonardo に Wifi 制御用の CPU を別途搭載したもの。
Arduino Due	524288	96	無し	外観は Mega に似ているが搭載 CPU が異なるため、全く別物である。駆動電圧の違いに注意。EEPROM を搭載しない。
Arduino M0	262144	32	無し	高性能な次世代機。CPU が違うため UNO とは全く別物。EEPROM を搭載しない。
Arduino M0 Pro	262144	32	無し	M0 にデバッガを搭載した開発用。EEPROM を搭載しない。

（西日本農業研究センター　黒崎秀仁）

第2章 日射量を計測する

1 はじめに

温室向けの日射量の測定は遮光カーテンの制御などに必要ですが、昔からある日射センサは精度が高いものの1個10万円ぐらいする上に、出力がミリボルト単位と微小で、計測にも高価なデータロガーが必要でなかなか農業用には向かないものでした。しかし、最近1万5000円ぐらいで買える安価な日射センサが登場しました。PV アレイ日射計です。



PV アレイ日射計(三弘)

PV アレイ日射計の出力は 1.0kW/m^2 あたり 1.0V と分かりやすい値で、日射センサの中では出力が大きく、Arduino のアナログ入力端子に接続して測定できるだけの十分な電圧があります。

2 ソフトウェアと結線

前章をご覧になった方はすぐに分かると思いますが、基本部分は特集1で示したサンプルプログラムを少し書き換えるだけですぐに作れてしまいます。今回は CCM を 2 つ作成し、片方は"cnd.mOC"としました。".mOC"の部分は屋外気象観測装置を示す種別名で、この部分の付け方は通信実用規約 1.00-E10 の 31 ページを参照して下さい。もう片方は"WRadiation"ですが、これは屋外日射を示します。こちらは予約 CCM に含まれている(通信規約 41 ページ)ので種別名を省略しています。

次に、アナログ入力値を日射量への換算する方法ですが、1.0V が入力された時の AD 変換値は 1023÷5 = 204.6 となりますので、AD 変換値 ×(1／204.6) = 日射量(kW/m^2) となります。ただし、整数値で演算すると小数点以下が切り捨てられてしまうので、サンプルではあえて分かりやすいように一度少数が扱える変数を介して計算させました。次に、小数の表現ですが、Arduino の仕

様では float や double 型の変数は浮動小数点数を扱えますが、UARDECS で CCM に出力するときは一度整数に直す必要があります。少々ややこしいですが、CCM の値の小数点の位置は UECSsetCCM()関数で宣言した時の 7 番目の引数の値で決まります。例えば、以下のように少数桁数に 3 を指定した場合、

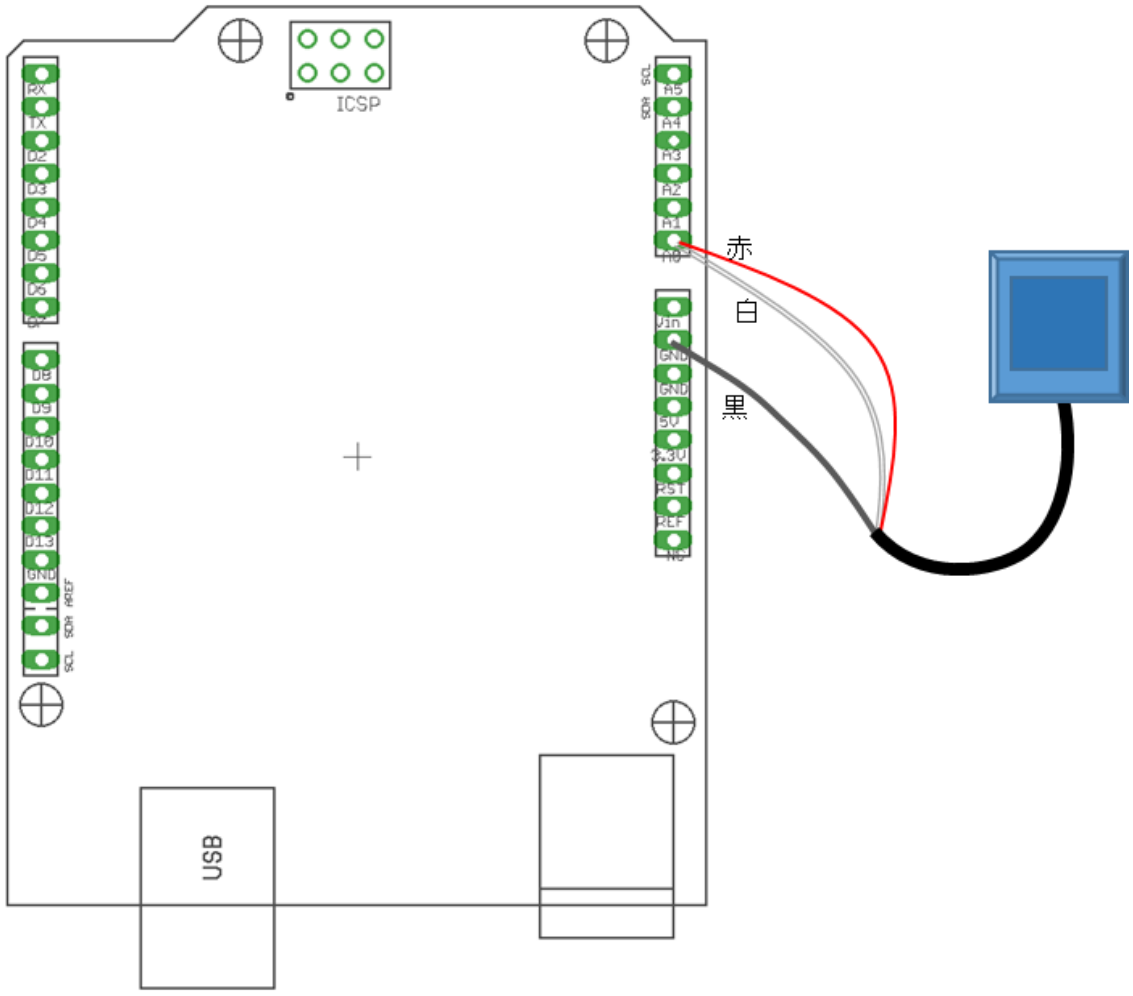
```
UECSsetCCM(true,1,ccmInfoRad,ccmTypeRad,ccmUnitRad,29,3,A_1S_0);
```

U_ccmList[1].value の値が 1234 の時、実際に CCM として出力される値は 1.234 になります。少数桁数が 3 の時に double 型の変数から U_ccmList[1].value に値を代入する場合、桁を合わせるために 1000 倍してから代入しなければなりません(少数桁数が 1 の時は 10 倍、2 の時は 100 倍して下さい)。これらの処理を施したのが以下のプログラムです。このサンプルプログラムは #ApplicationGuide の中の Sample3SendRadiation に収録されています。

```
//Sample3SendRadiation
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>
const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;

const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 2;//CCM の総数を 2 に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM 定義用の素材、被らないように適当な変数名で 3 つ宣言(必ず PROGMEM を付ける)
const char ccmInfoTest[] PROGMEM= "NodeCondition";//CCM の説明(Web でのみ表示)
const char ccmTypeTest[] PROGMEM= "cnd.mOC";//CCM の Type 文字列
const char ccmUnitTest[] PROGMEM= "";//CCM の単位(この場合単位無し)

const char ccmInfoRad[] PROGMEM= "Radiation";//CCM の説明(Web でのみ表示)
const char ccmTypeRad[] PROGMEM= "WRadiation";//CCM の Type 文字列(屋外日射)
const char ccmUnitRad[] PROGMEM= "kW m-2";//CCM の単位
```

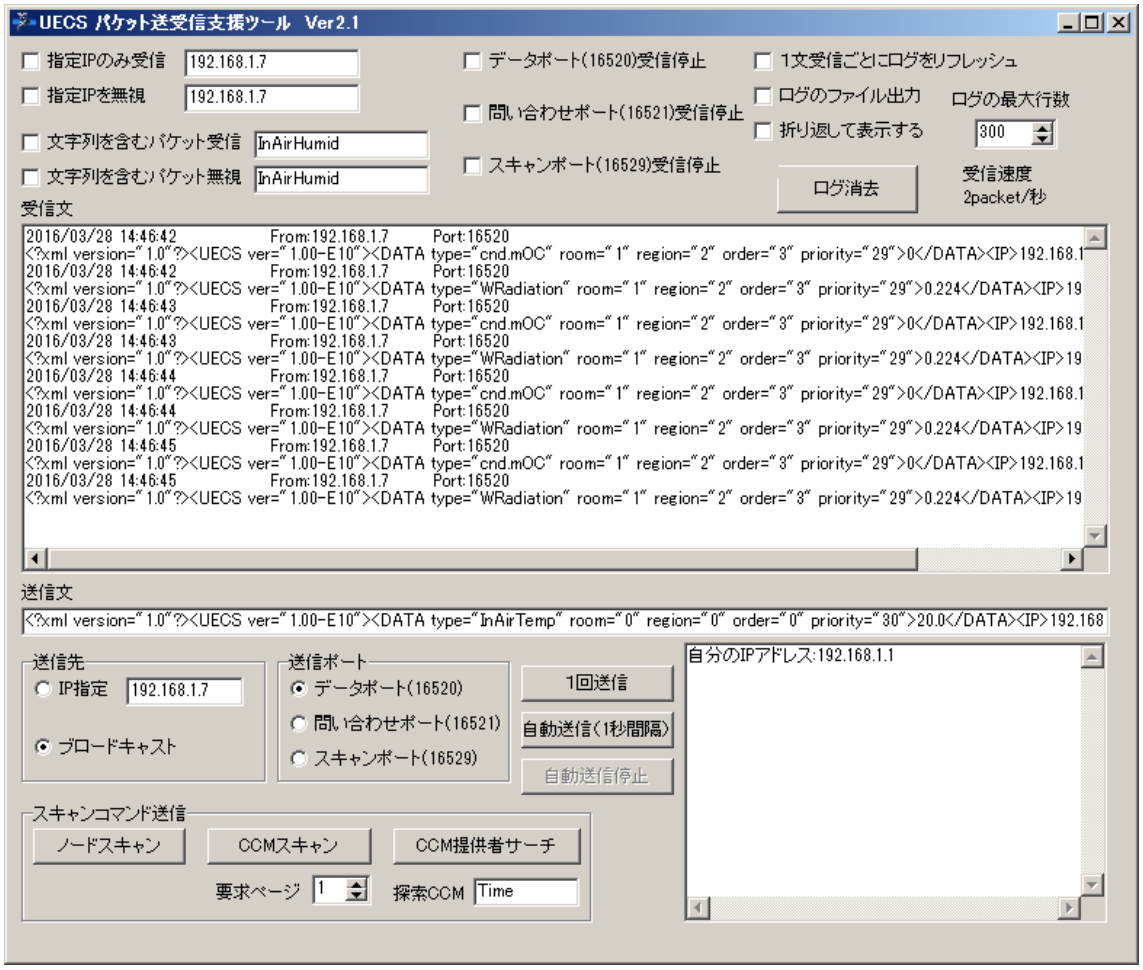
```
void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//UECSsetCCM(送受信の区分[true で送信],通し番号[0 から始まる],CCM 説明,Type,単
位,priority[通常は 29],少数桁数,送信頻度設定[A_1S_0 で 1 秒間隔])
UECSsetCCM(true,0,ccmInfoTest,ccmTypeTest,ccmUnitTest,29,0,A_1S_0);
UECSsetCCM(true,1,ccmInfoRad,ccmTypeRad,ccmUnitRad,29,3,A_1S_0);
}
void OnWebFormRecieved(){}

void UserEverySecond()
{
double radiation= (double)analogRead(0)*1.0/204.6;
U_ccmList[1].value= radiation*1000; //少数桁が 3 なので 1000 倍
}
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
UECSsetup();
}
```

PV アレイ日射計を使う場合、ケーブルの赤、白はまとめて A0 に、黒は GND に接続します。これは 1 個だけ接続する場合の配線で、複数並列して使う方法もありますが、使い方はセンサのマニュアルに書いてあります。ケーブルが太くて直接 Arduino への接続が難しい場合はジャンパーケーブルをハンダ付けしてしまいましょう。



Arduino への接続(赤白を A0 に、黒を GND に)
※図は UNO 本体ですが、Ethernet Shield 2 を載せていると見なして下さい

3　出力値を見てみる



出力値の例

WRadiation の部分に日射量が出力されます。日射計にはかなり強い光を当てないと大きな値は出力されません。一番日射の強い時間帯でも 1.2 kW/m^2 行くかどうかといったレベルです。一応、これでも日射計としては成立するのですが、残念なところは 0～5V の範囲を 1023 段階で測れる AD コンバータで 0～1V ぐらいを行ったり来たりする値を測っているので本来の 1/5 の範囲しか活用できていないことになります。そのため、分解能が低くなってしまい、出力値だけは小数点以下 3 桁もあるのに、実際には 200 段階ぐらいの分解能しかありません。次はこの解決策を解説します。

4 AD コンバータの計測レンジを変更する

実は、Arduino にはソフトウェア上で AD コンバータの計測レンジを変更できる機能があります。analogReference()関数による基準電圧の変更です。設定できる値は機種により異なり、UNO では 5V、1.1V の 2 種類、MEGA では 5V、2.56V、1.1V の 3 種類から選ぶことができます。機種により記号定数が違うので注意して下さい。

UNO の場合、

```
analogReference(DEFAULT);        //UNO 基準電圧 5V
analogReference(INTERNAL);       //UNO 基準電圧 1.1V
```

MEGA の場合

```
analogReference(DEFAULT);        //MEGA 基準電圧 5V
analogReference(INTERNAL1V1);    //MEGA 基準電圧 1.1V
analogReference(INTERNAL2V56);   //MEGA 基準電圧 2.56V
```

例えば、1.1V を指定すると 0～1.1V の範囲を 1023 段階で表すようになります。当然ながら、日射量への換算式も変わってきます。基準電圧を 1.1V に設定した場合の 1.0V が入力された時の AD 変換値は 1023÷1.1 = 930 となりますので、AD 変換値 × (1／930) = 日射量(kW/m²)となります。これらの処理を入れたものが以下のプログラムです。このサンプルプログラムは#ApplicationGuide の中の Sample4ArefChange に収録されています。

```
//Sample4ArefChange
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>
const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;
const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};
const int U_MAX_CCM = 2;//CCM の総数を 2 に
UECSCCM U_ccmList[U_MAX_CCM];
```

```
//CCM 定義用の素材、被らないように適当な変数名で 3 つ宣言(必ず PROGMEM を付ける)
const char ccmInfoTest[] PROGMEM= "NodeCondition";//CCM の説明(Web でのみ表示)
const char ccmTypeTest[] PROGMEM= "cnd.mOC";//CCM の Type 文字列
const char ccmUnitTest[] PROGMEM= "";//CCM の単位(この場合単位無し)

const char ccmInfoRad[] PROGMEM= "Radiation";//CCM の説明(Web でのみ表示)
const char ccmTypeRad[] PROGMEM= "WRadiation";//CCM の Type 文字列(屋外日射)
const char ccmUnitRad[] PROGMEM= "kW m-2";//CCM の単位

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//UECSsetCCM(送受信の区分[true で送信],通し番号[0 から始まる],CCM 説明,Type,単
位,priority[通常は 29],少数桁数,送信頻度設定[A_1S_0 で 1 秒間隔])
UECSsetCCM(true,0,ccmInfoTest,ccmTypeTest,ccmUnitTest,29,0,A_1S_0);
UECSsetCCM(true,1,ccmInfoRad,ccmTypeRad,ccmUnitRad,29,3,A_1S_0);
}
void OnWebFormRecieved(){}

void UserEverySecond()
{
double radiation= (double)analogRead(0)*1.0/930.0;
U_ccmList[1].value= radiation*1000; //少数桁が 3 なので 1000 倍
}
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
analogReference(INTERNAL);//UNO 基準電圧 1.1V
UECSsetup();
}
```

5 AD コンバータの計測レンジの自動変更

日射量はほとんどの時間帯で 1.1kW/m^2 以下に収まるのですが、厄介なことに日射量の多いごく僅かな時間帯だけ 1.2 kW/m^{2}ぐらいになり、計測レンジを 1.1V に固定すると微妙に上限を超えてしまいます。上限を超えるとanalogRead(0)の戻り値が 1023 になりますので、一度測ってみて超えていそうだったら計測レンジを 5V に戻して再計測することで解決できます。ただし、こちらで実験したところ、基準電圧を変更してから少しの間は正常な測定ができません。基準電圧の変更に 10 ミリ秒を要し、そこから安定した計測ができるまで数回 analogRead()を実行して読み飛ばす必要があります。これらの処理を実装するとこうなります。cnd.mOC には計測レンジの状態を出力するようにしました。このサンプルプログラムは#ApplicationGuide の中の Sample5AutoRange に収録されています。

```
//Sample5AutoRange
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>
const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;

const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;

const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 2;//CCM の総数を 2 に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM 定義用の素材、被らないように適当な変数名で 3 つ宣言(必ず PROGMEM を付ける)
const char ccmInfoTest[] PROGMEM= "NodeCondition";//CCM の説明(Web でのみ表示)
const char ccmTypeTest[] PROGMEM= "cnd.mOC";//CCM の Type 文字列
const char ccmUnitTest[] PROGMEM= "";//CCM の単位(この場合単位無し)

const char ccmInfoRad[] PROGMEM= "Radiation";//CCM の説明(Web でのみ表示)
const char ccmTypeRad[] PROGMEM= "WRadiation";//CCM の Type 文字列(屋外日射)
const char ccmUnitRad[] PROGMEM= "kW m-2";//CCM の単位

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
```

```
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//UECSsetCCM(送受信の区分[true で送信],通し番号[0 から始まる],CCM 説明,Type,単
位,priority[通常は 29],少数桁数,送信頻度設定[A_1S_0 で 1 秒間隔])
UECSsetCCM(true,0,ccmInfoTest,ccmTypeTest,ccmUnitTest,29,0,A_1S_0);
UECSsetCCM(true,1,ccmInfoRad,ccmTypeRad,ccmUnitRad,29,3,A_1S_0);
}

void OnWebFormRecieved(){}

void UserEverySecond()
{
static bool lowRange=false;
double radiation=analogRead(0);
if(radiation==1023 && lowRange)
  {
//1.1V モードでレンジ外になった場合、5V モードに変更
    lowRange=false;
    analogReference(DEFAULT);//UNO 基準電圧 5V
    analogRead(0);//読み飛ばす
    return;
  }

if(radiation<200 && !lowRange)
  {
//5V モードで 1V 未満になった場合、1.1V モードに変更
    lowRange=true;
    analogReference(INTERNAL);//UNO 基準電圧 1.1V
    analogRead(0);//読み飛ばす
    return;
  }

if(lowRange)
  {
  //1.1V モード換算式
  radiation= radiation*1.0/930.0;
  }
 else
  {
  //5V モード換算式
  radiation= radiation*1.0/204.6;
  }
  U_ccmList[1].value= radiation*1000; //少数桁が 3 なので 1000 倍
  U_ccmList[0].value= lowRange;//現在の計測レンジを出力
}
```

```
void UserEveryMinute(){}
void UserEveryLoop(){}

void loop()
{
UECSloop();
}

void setup()
{
analogReference(DEFAULT);//UNO 基準電圧 5V
UECSsetup();
}
```

実際には5Vモードの作動頻度はごく僅かだと思われます。この方式の難点は、レンジを切り替える時に1秒間計測ができずに過去の値が出力されてしまうところで、急激に値が変化し続ける時には追従できません。また、5Vの電源電圧が正確に供給されていないと5Vモードで出力値が狂う可能性があります。もし、MEGAを使う場合、妥協して最初から基準電圧を2.56Vモードで固定してしまえば、まあまあの精度で計測でき、通常の日射量の範囲であればレンジを超えることは無いと思われます。なお、実際の温室で使用する場合は十分な動作検証を行い、自己責任の上で使用して下さい。

（西日本農業研究センター　黒崎秀仁）

第3章 LED の点灯・消灯

1 受信 CCM の作成

これまでは CCM の送信ノードについて解説してきましたが、次は特定の CCM を受信することで何らかの動作を起こすノードの作り方を解説します。そのためには受信 CCM の扱いについて説明する必要があります。

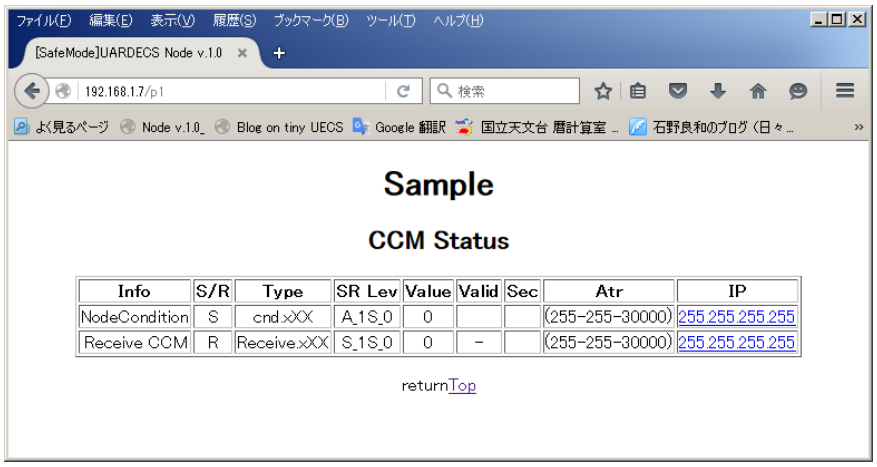
受信 CCM を作る場合も送信 CCM と同じように3種類の文字列(CCM の説明,type,単位)を作成します。次に UECSsetCCM 関数を記述するとき、最初の引数を false にすると受信 CCM になります。例えば以下のように記述します。

```
UECSsetCCM(false,1,ccmInfoRcv,ccmTypeRcv,ccmUnitRcv,29,0,A_1S_0);
```

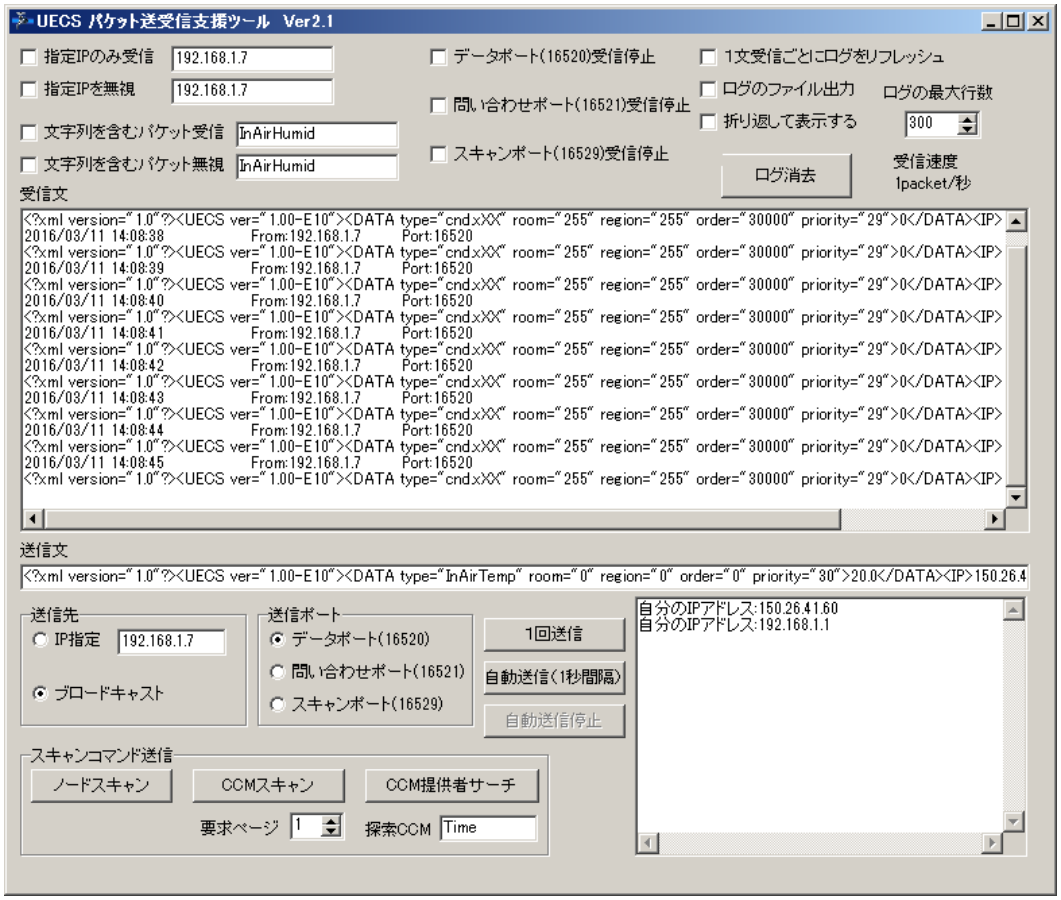
今度も Skeleton から改造してみます。以下のプログラムでは送信 CCM を1個、受信 CCM を1個作りました。改造箇所は色付きの部分です。このサンプルプログラムは#ApplicationGuide の中の Sample6ReceiveCCM に収録されています。

```
//Sample6ReceiveCCM
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>

const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;
const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;
const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};
```

```
const int U_MAX_CCM = 2;//CCMの総数を2に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM定義用の素材1、被らないように適当な変数名で3つ宣言
const char ccmInfoCnd[] PROGMEM= "NodeCondition";//CCMの説明(Webでのみ表示)
const char ccmTypeCnd[] PROGMEM= "cnd.xXX";//CCMのType文字列
const char ccmUnitCnd[] PROGMEM= "";//CCMの単位(この場合単位無し)
//CCM定義用の素材2、被らないように適当な変数名で3つ宣言
const char ccmInfoRcv[] PROGMEM= "Receive CCM";//CCMの説明(Webでのみ表示)
const char ccmTypeRcv[] PROGMEM= "Receive.xXX";//CCMのType文字列
const char ccmUnitRcv[] PROGMEM= "";//CCMの単位(この場合単位無し)

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//UECSsetCCM(送受信の区分,通し番号,CCM説明,Type,単位,priority,少数桁数,送信頻度
設定[A_1S_0で1秒間隔])
UECSsetCCM(true,0,ccmInfoCnd,ccmTypeCnd,ccmUnitCnd,29,0,A_1S_0);//送信CCM
UECSsetCCM(false,1,ccmInfoRcv,ccmTypeRcv,ccmUnitRcv,29,0,S_1S_0);//受信CCM
}
void OnWebFormRecieved(){}
void UserEverySecond(){}
void UserEveryMinute(){}
void UserEveryLoop(){}
void loop()
{
UECSloop();
}
void setup()
{
UECSsetup();
}
```



Web インターフェースの状態



CCM の送信状態

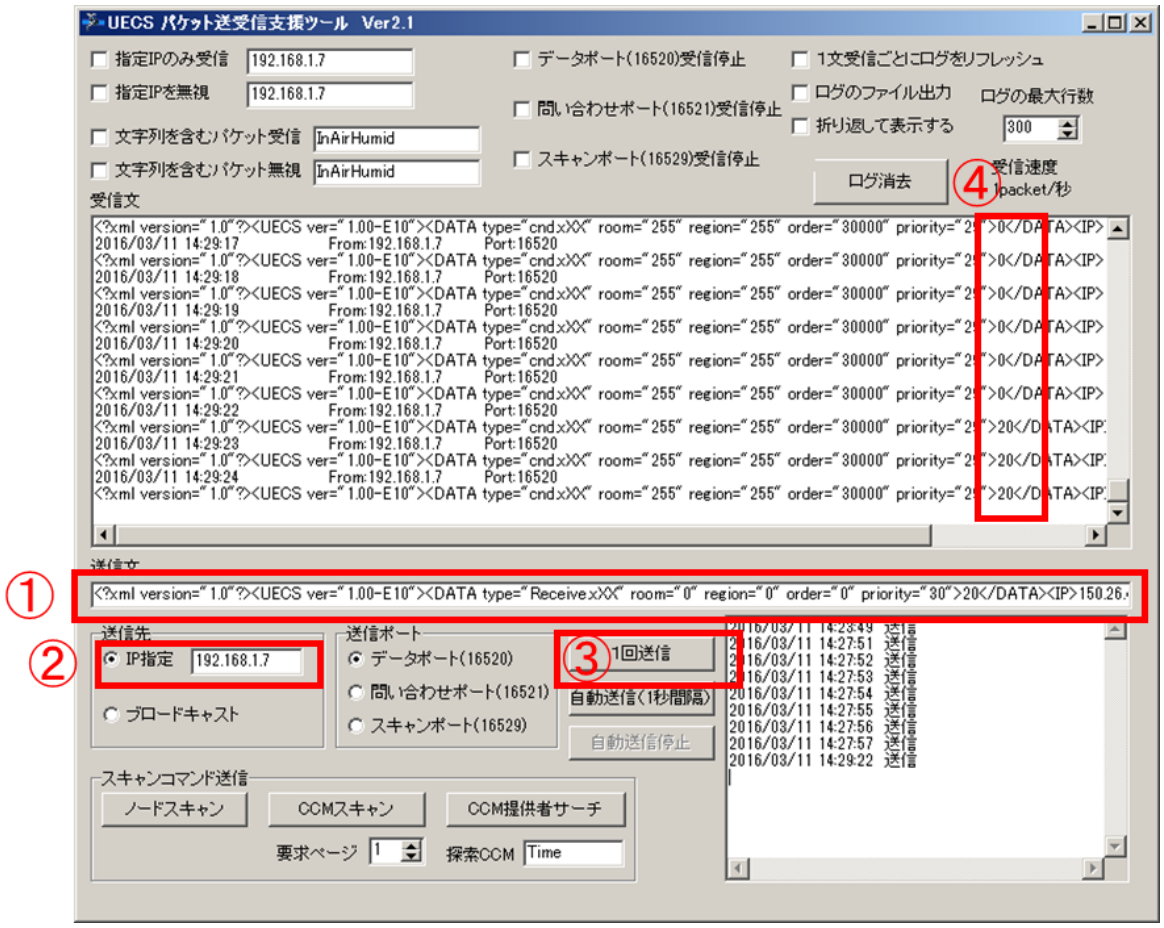
UNO に上記プログラムを転送してブラウザでアクセスすると、”cnd.xXX”と”Recieve.xXX”の 2 種類の CCM が確認できます。UDP パケットの方は”cnd.xXX”が1秒間隔で送信されているのが分かります。しかし、まだこの段階では CCM を受信させても何も起こりません。

2 受信 CCM の値へのアクセス

次の改造ですが、受信 CCM の値をそのまま送信 CCM に代入してみます。ただし、受信 CCM は常に有効な値が受信できているかどうか確認してから処理する必要があります。値の有効/無効は U_ccmList[通し番号].validity という変数を参照することで判定できます。このサンプルプログラムは #ApplicationGuide の中の Sample7EchoCCM に収録されています。

```
//Sample7EchoCCM
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>
const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;
const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";
UECSOriginalAttribute U_orgAttribute;
const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};

const int U_MAX_CCM = 2;//CCM の総数を 2 に
UECSCCM U_ccmList[U_MAX_CCM];
//CCM 定義用の素材 1、被らないように適当な変数名で 3 つ宣言
const char ccmInfoCnd[] PROGMEM= "NodeCondition";//CCM の説明(Web でのみ表示)
const char ccmTypeCnd[] PROGMEM= "cnd.xXX";//CCM の Type 文字列
const char ccmUnitCnd[] PROGMEM= "";//CCM の単位(この場合単位無し)
//CCM 定義用の素材 2、被らないように適当な変数名で 3 つ宣言
const char ccmInfoRcv[] PROGMEM= "Receive CCM";//CCM の説明(Web でのみ表示)
const char ccmTypeRcv[] PROGMEM= "Receive.xXX";//CCM の Type 文字列
const char ccmUnitRcv[] PROGMEM= "";//CCM の単位(この場合単位無し)

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
```

```
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;

//UECSsetCCM(送受信の区分,通し番号,CCM 説明,Type,単位,priority,少数桁数,送信頻度
設定[A_1S_0 で 1 秒間隔])
//送信 CCM
UECSsetCCM(true,0,ccmInfoCnd,ccmTypeCnd,ccmUnitCnd,29,0,A_1S_0);
//受信 CCM
UECSsetCCM(false,1,ccmInfoRcv,ccmTypeRcv,ccmUnitRcv,29,0,S_1S_0);
}
void OnWebFormRecieved(){}
void UserEverySecond()
{
//受信 CCM の値を送信 CCM に代入
if(U_ccmList[1].validity)
  {
  U_ccmList[0].value=U_ccmList[1].value;//値が有効な場合
  }
else
  {
  U_ccmList[0].value=0;//値が無効の場合 0 にする
  }
}
void UserEveryMinute(){}
void UserEveryLoop(){}
void loop()
{
UECSloop();
}
void setup()
{
UECSsetup();
}
```



CCM の送信テスト

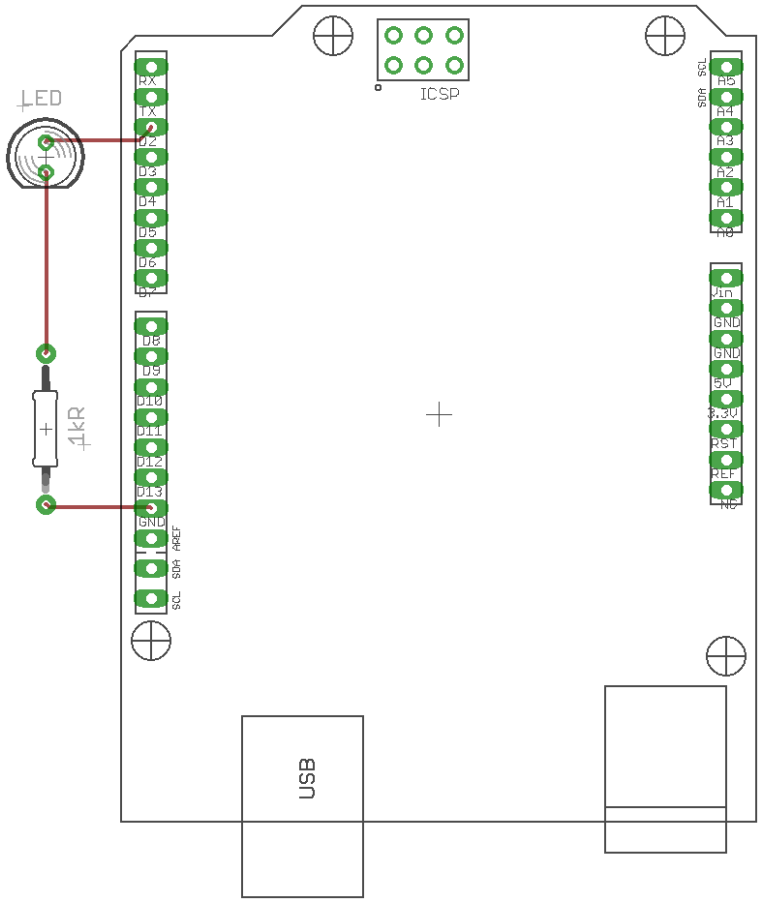
CCM の送信テストには PC と Arduino を LAN ケーブルで直結し、UECS 用パケット送受信ツールを使用します。①の欄の type=のところを“ Receive.xXX”に書き換え、値の部分を適当な数値に変更、②の部分で IP 指定を選び送信先 Arduino の IP アドレスを入力、③の1回送信を押します。正常に受信されれば、④の“cnd.xXX”の値が①で入力した値に 3 秒間だけ変化し、その後また 0 に戻ります。これで、送った CCM に対して Arduino が応答していることが分かりました。

1 回の送信に対して 3 秒間だけ値が帰ってくる理由は、UECSsetCCM()関数の最後の引数を S_1S_0 にしたのが原因です。UECS の通信実用規約 1.00-E10 では送受信頻度ごとに、CCM の有効期限が定められており、一般に送信間隔の 3 倍が有効期限です。したがって、1 秒間隔で送信される S_1S_0 の有効期限は受信してから 3 秒間です。これを 1 分間隔で送信される S_1M_0 に書き換えた場合、値は 3 分間有効と見なされます。

※A_1S_0 と S_1S_0 の違いは、A_??_??の CCM は常時一定時間おきに送信されるのに対して S_??_??の CCM は必要なときだけ一定時間おきに送信され、不要なときは停止することを示しています。

3 LED の点灯

次に 0 以外の値を受信した時、LED を点灯するように改造してみます。LED は D2 ピンに接続することにしました。どのピンが利用できるかは UARDECS マニュアルの3章を見て下さい。

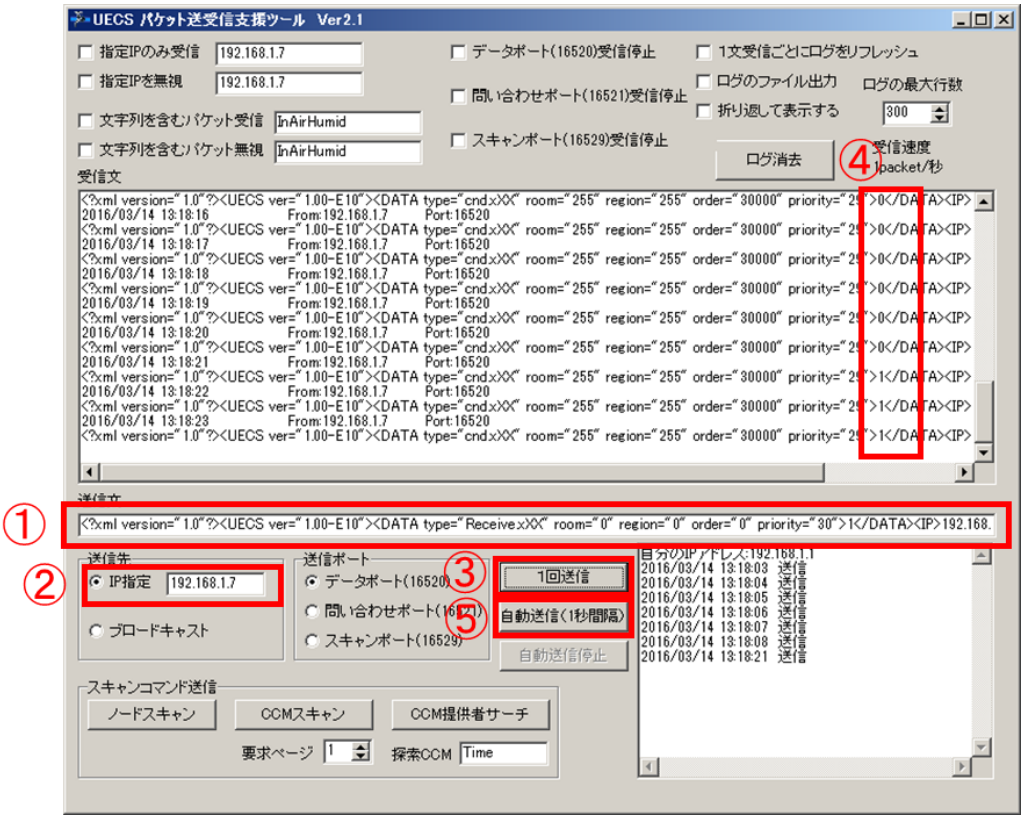


LED の結線（LED は足の長いほうを D2 側、短い方を GND 側に接続）
※図は UNO 本体ですが、Ethernet Shield 2 を載せていると見なして下さい

次にサンプルプログラムを示します。修正点として、setup()関数内でピンの初期化を行っています。先ほどの”cnd.xXX”の使い方は、微妙に UECS の通信実用規約 1.00-E10 から外れていたので、”cnd.xXX”には LED が点灯している間 1 を出力し、それ以外は 0 になるようにしました。”cnd”が付いた CCM にはノードの動作状態を示す値を出力することになっており、整数値で 0～4095 の範囲はユーザーが自由に意味を定義して使うことができます。このサンプルプログラムは #ApplicationGuide の中の Sample8LEDTest に収録されています。

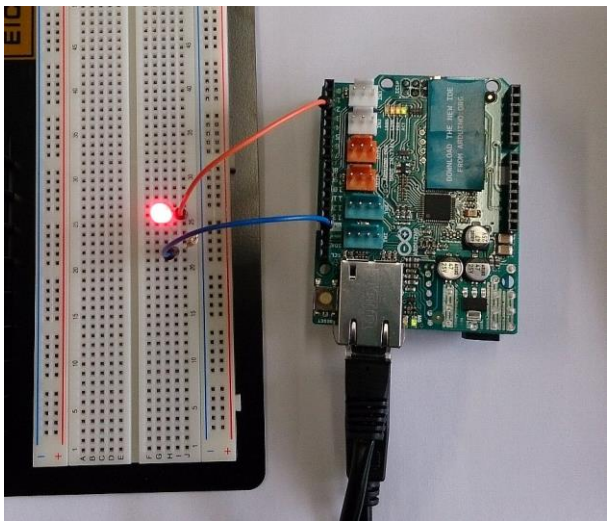
```
//Sample8LEDTest
#include <SPI.h>
#include <Ethernet2.h> //Arduino IDE Ver1.7.2 以降で W5500 搭載機種
//#include <Ethernet.h> //Ver1.7.2 以降で W5100 搭載機種
#include <avr/pgmspace.h>
#include <EEPROM.h>
#include <Uardecs.h>

const byte U_InitPin = 3;//このピンは変更可能です
const byte U_InitPin_Sense=HIGH;
const char U_name[] PROGMEM= "UARDECS Node v.1.0";
const char U_vender[] PROGMEM= "XXXXXX Co.";
const char U_uecsid[] PROGMEM= "000000000000";
const char U_footnote[] PROGMEM= "Test node";
char U_nodename[20] = "Sample";

UECSOriginalAttribute U_orgAttribute;
const int U_HtmlLine = 0;
struct UECSUserHtml U_html[U_HtmlLine]={};
const int U_MAX_CCM = 2;//CCM の総数を 2 に
UECSCCM U_ccmList[U_MAX_CCM];

//CCM 定義用の素材 1、被らないように適当な変数名で 3 つ宣言
const char ccmInfoCnd[] PROGMEM= "NodeCondition";//CCM の説明(Web でのみ表示)
const char ccmTypeCnd[] PROGMEM= "cnd.xXX";//CCM の Type 文字列
const char ccmUnitCnd[] PROGMEM= "";//CCM の単位(この場合単位無し)

//CCM 定義用の素材 2、被らないように適当な変数名で 3 つ宣言
const char ccmInfoRcv[] PROGMEM= "Receive CCM";//CCM の説明(Web でのみ表示)
const char ccmTypeRcv[] PROGMEM= "Receive.xXX";//CCM の Type 文字列
const char ccmUnitRcv[] PROGMEM= "";//CCM の単位(この場合単位無し)

void UserInit(){
U_orgAttribute.mac[0] = 0x00;
U_orgAttribute.mac[1] = 0x00;
U_orgAttribute.mac[2] = 0x00;
```

```
U_orgAttribute.mac[3] = 0x00;
U_orgAttribute.mac[4] = 0x00;
U_orgAttribute.mac[5] = 0x00;
//UECSsetCCM(送受信の区分,通し番号,CCM 説明,Type,単位,priority,少数桁数,送信頻度
設定[A_1S_0 で 1 秒間隔])
//送信 CCM
UECSsetCCM(true,0,ccmInfoCnd,ccmTypeCnd,ccmUnitCnd,29,0,A_1S_0);
//受信 CCM
UECSsetCCM(false,1,ccmInfoRcv,ccmTypeRcv,ccmUnitRcv,29,0,S_1S_0);
}
void OnWebFormRecieved(){}
void UserEverySecond()
{
//受信 CCM が有効かつ 0 以外
if(U_ccmList[1].validity && U_ccmList[1].value)
  {
  U_ccmList[0].value=1;
  digitalWrite(2,HIGH);
  }
else
  {
  U_ccmList[0].value=0;
  digitalWrite(2,LOW);
  }
}
void UserEveryMinute(){}
void UserEveryLoop(){}
void loop()
{
UECSloop();
}
void setup()
{
pinMode(2,OUTPUT);
UECSsetup();
}
```



CCM の送信テスト2

先ほどの実験同様UECS用パケット送受信ツールを使用し、PCとArduinoを繋いでテストします。①の欄の type=のところを“ Receive.xXX”に書き換え、値を 0 以外に変更、②の部分で IP 指定を選び Arduino の IP アドレスを入力、③の1回送信を押します。正常に受信されれば、④の“cnd.xXX”の値が3秒間だけ1に変化し、その間、下図のようにLEDが点灯します。また、⑤の自動送信ボタンを押せばずっとLEDを点灯させ続けることができます。



LED の点灯

前述したプログラムでLEDを点灯させ続けるにはCCMを延々と送信し続ける必要がありましたが、UserEverySecond()関数を次のように書き換えれば、“Receive.xXX”に0以外の値を一度送信すればずっと点灯を続け、消したいときは0を送信して消灯できます。ただし、CCMは通信環境が悪いとパケットロスが発生し、届かないことがあります。この例はパケットロスが起きると送信されたCCMと動作が一致しなくなるので、パケットロスがあっても問題にならないような場面に使用することをお勧めします。このサンプルプログラムは#ApplicationGuideの中のSample8LEDTestLatchに収録されています。

```
//ラッチ式点灯・消灯
void UserEverySecond()
{
if(U_ccmList[1].validity && U_ccmList[1].value)//点灯
  {
  U_ccmList[0].value=1;
  digitalWrite(2,HIGH);
  }
if(U_ccmList[1].validity && U_ccmList[1].value==0)//消灯
  {
  U_ccmList[0].value=0;
  digitalWrite(2,LOW);
  }
}
```

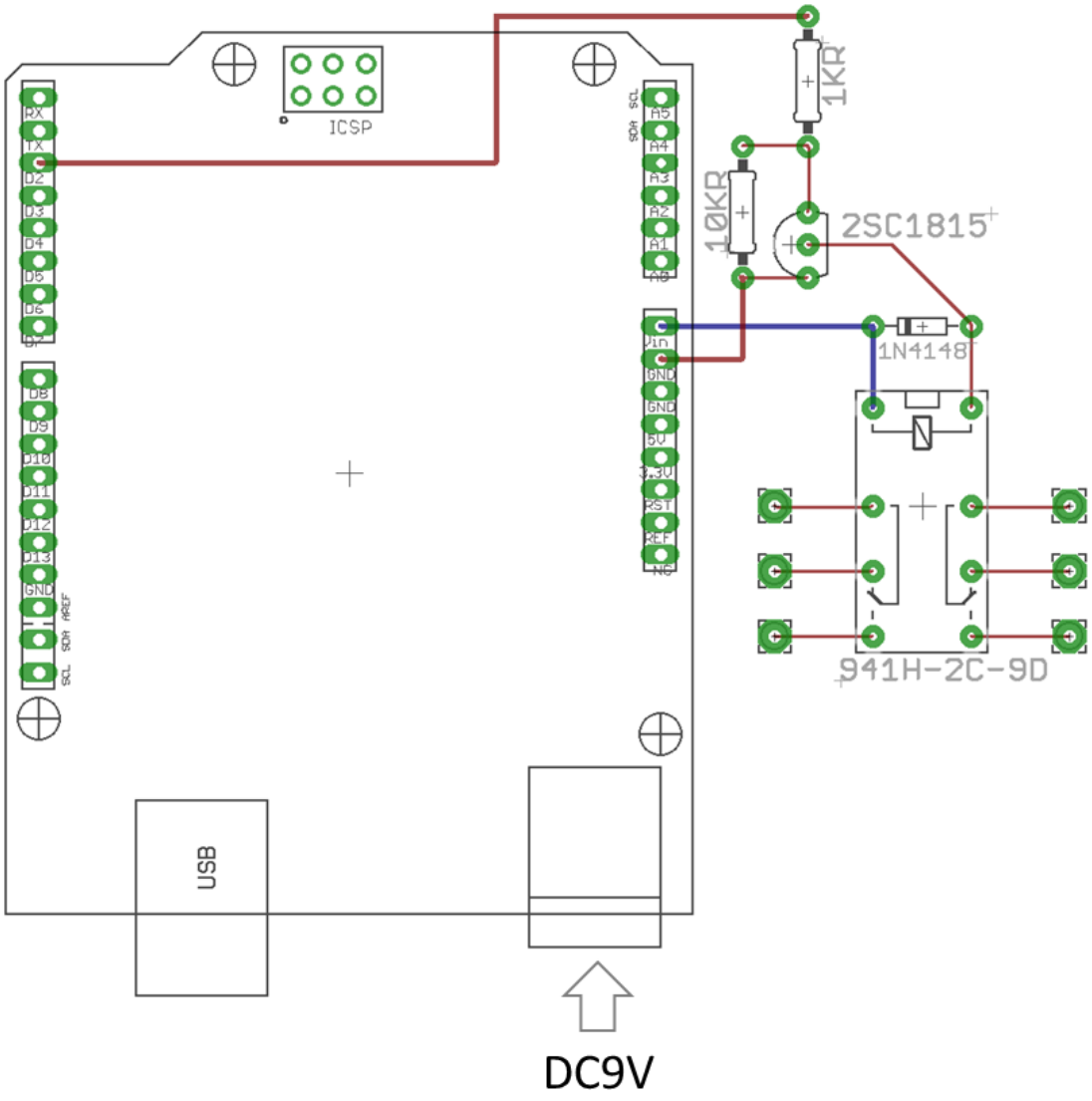
一方、点灯のCCMを送った後、タイムアウトするまでの時間を3秒より長く設定したい場合、以下のように書き換えて、自前で点灯時間を管理できます。このサンプルプログラムは #ApplicationGuide の中の Sample8LEDTestTimeout10s に収録されています。

```
//タイムアウト時間を 10 秒にする
void UserEverySecond()
{
if(U_ccmList[1].validity && U_ccmList[1].value)
  {
  U_ccmList[0].value=1;
  digitalWrite(2,HIGH);
  }
else if(U_ccmList[1].flagStimeRfirst && U_ccmList[1].recmillis>10000)//10sec
  {
  U_ccmList[0].value=0;
  digitalWrite(2,LOW);
  }
}
```

U_ccmList[1].recmillis には、最後に CCM を受信してからの経過時間がミリ秒単位で記録されており、細かく CCM の受信タイミングを知ることができます。この値は最大 24 時間分までカウントアップされます(誤差あり)。ただし、U_ccmList[1].flagStimeRfirst が true でない場合、U_ccmList[1].recmillis の値の正確さは保証されませんので、必ず確認してから参照します。

4 LED を別のデバイスに置き換える

これまでの説明で、CCM に応答する最低限の UECS ノードのプロトタイプができたことになります。LED の点灯をコントロールできるということは LED の代わりに様々なものを接続してスイッチの ON/OFF を制御できるということです。

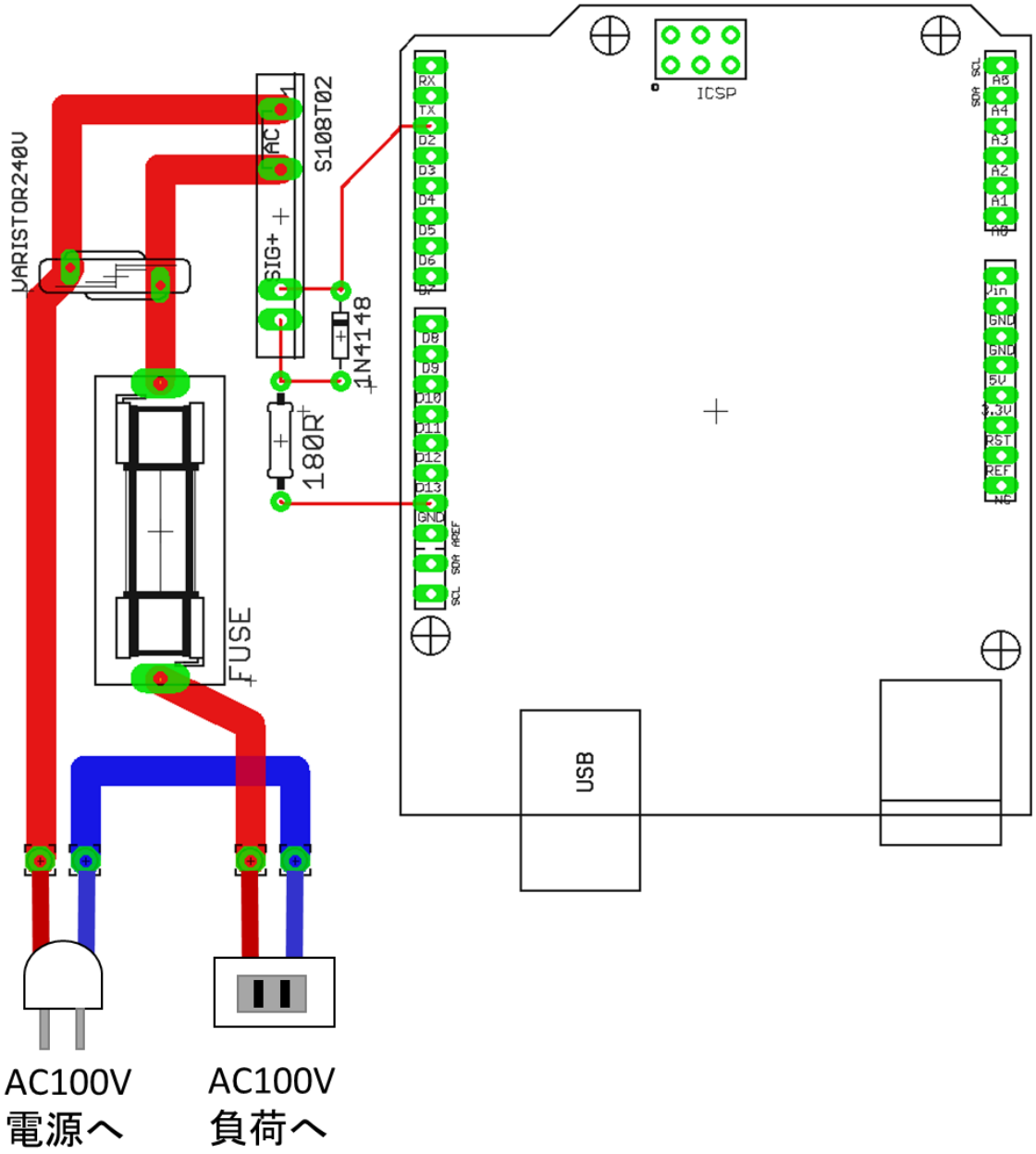


9V リレーの動作回路例

ここにLEDの代わりにリレーを接続した例を示します。ここでは941H-2C-9Dを使用しました。このリレーは9Vで動作するのでDCジャックに9VのACアダプタを接続する必要があります。リレーの電源はDCジャックの＋側に直結されたVin端子から取っています。9Vの電圧は直接Arduinoでは扱えませんので、間にトランジスタ2SC1815を入れています。様々な文献で言われていますが、リレーは作動するときにサージ対策が必要なので、ダイオード1N4148を入れました。このダイオードは安易に応答の遅い整流用を使わない方が良いです。合わないものを使うとサージを吸収できずにトランジスタを壊したり、リレーを動かす度にリセットがかかる回路ができあがります。

作動電圧5Vの941H-2C-5Dであれば9VのACアダプタは不要です。この場合、駆動には5V、約30mAが必要で、リレーの電源は5V端子から取りましょう。Arduino UNOのピンの電流出力はカタログスペックでは最大40mAとされており、一見すると直接つないで動かせそうですが、実際には出力電流が増えると出力電圧が低下します。こちらで実験したところ、UNOのデジタル出力の電圧は何もデバイスをつながないときは4.8V程度ですが、15mAで4.4V、30mAで4V、40mAで3.6Vでした。そのため、30mA出力すると941H-2C-5Dを駆動する電圧が確保できなくなる可能性があります。たとえ机上で動いても、部品の個体差で動かなくなったり、温度変化で動かなくなったりしそうなので、ギリギリの設計はやめてトランジスタなどを入れるのをお勧めします。

リレーは機械式の接点を動かすという一見すると分かりやすそうなものですが、駆動に必要な電流量が微妙に多かったり、大電流を流すと接点にスパークが生じたり、接点にチャタリングが発生したりと扱いにくい場面があります。一方、ソリッドステート・リレー(SSR)と呼ばれる半導体式のリレーが存在し、より簡単に大電流を制御できます。これを使えばArduinoの出力からAC100Vすら制御可能です。ゼロクロス機能を持つSSRはAC専用で、自動的に交流の電圧が0Vになった瞬間を捉えて電源をON/OFFするので、電気的なノイズが発生せず周辺の回路に対する影響も抑えられます。例えばSSRキット35AやS108T02(SSRキット8A)などです。ただし、SSRは通電中に発熱しますので、カタログスペックは大電流に対応していても放熱しないと十分な性能を発揮できないことがあります。また、AC100Vの取り扱いについては当然リスクが伴いますので、ここで取り扱う回路はデバイスのデータシート等を参照して注意点を確認した後、然るべき知識を持った方のみが自己責任の上で挑戦して下さい。



SSR を用いた回路の例

（注意：AC100V の取り扱いにはリスクが伴いますので然るべき知識を持った方のみが自己責任の上で挑戦して下さい）

（西日本農業研究センター　黒崎秀仁）