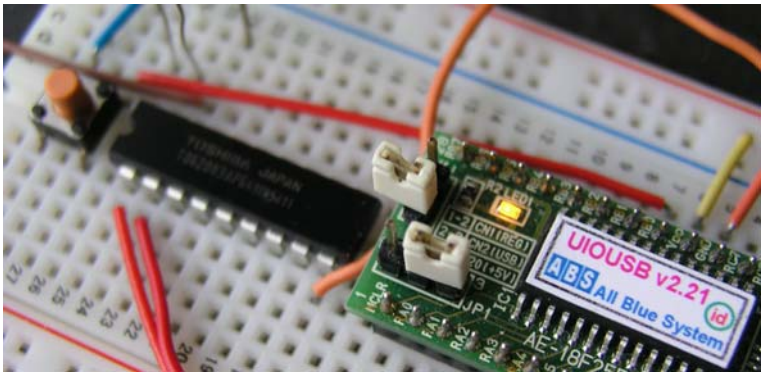
ホームセンサーキット応用ガイド

ABS-9000 UIOUSB

HomeSensorKit Application Guide Rev A.1.4

2013/2/15



オールブルーシステム (All Blue System)

ウェブページ: www.allbluesystem.com

コンタクト:contact@allbluesystem.com

1 このマニュアルについて 5
2 始めに 5
2.1 UIOUSB デバイスとサーバーのセットアップ 5
2.2 UIOUSB デバイスの回路を作成するときの注意 5
2.3 スクリプト編集用エディタを用意する 7
2.4 スクリプトファイルの実行について 8
2.5 UIOUSBのリスタートについて 9
3 [HS001] LED の点滅 10
3.1 アプリケーション説明 10
3.2 回路図 10
3.3 配線図 10
3.4 スクリプトファイル設定と説明 11
3.5 動作確認 (スクリプトファイルの手動実行方法) 13
4 [HS002] ボタンを押したときにLED点滅＆ブザーを鳴らす 17
4.1 アプリケーション説明 17
4.2 回路図 17
4.3 配線図 18
4.4 スクリプトファイル設定と説明 18
4.5 動作確認 22
4.6 応用(1)最初にボタンを押した時にランプとブザーをつけて、2回目に押した時に消す 23
4.7 応用(2)ボタンを押した時にランプとブザーをつけて、しばらく経つと自動的に消す 25
5 [HS003] リードスイッチでドアの開閉を監視する 27
5.1 アプリケーション説明 27
5.2 回路図 27
5.3 配線図 28
5.4 スクリプトファイル設定と説明 29
5.5 動作確認 34
5.6 改良(1)リードスイッチが連続して反応した時に対応 34
5.7 改良(2)ランプ点滅中に監視を中止したら、すぐにランプを消灯する 37
5.8 完成(3) 稀にしか発生しない不具合に対応する 39
6 [HS004] 温度と明るさを測定する 46
6.1 アプリケーション説明 46
6.2 回路図 46
6.3 配線図 46
6.4 スクリプトファイル設定と説明 47

6.5 動作確認 49
6.6 改良(1)A/Dリファレンス電圧を測定する 50
7 [HS005] １日の温度と明るさの変化を EXCEL でグラフにする 52
7.1 アプリケーション説明 52
7.2 回路図 52
7.3 配線図 53
7.4 スクリプトファイル設定と説明 53
7.5 動作確認 63
7.6 応用(1) 毎時 0 分の正確な時間に測定する 65
7.6.1 Windows タスクスケジューラについて 70
7.6.2 ScriptExecCmd.exe プログラムの設定 70
7.6.3 タスクスケジューラへのJOB登録 71
7.6.4 動作確認 72
8 [HS006] 温度と明るさ、ドアの開閉を監視する 73
8.1 アプリケーション説明 73
8.2 機能と動作フロー 73
8.3 回路図 73
8.4 配線図 74
8.5 スクリプトファイル設定と説明 74
8.6 メール設定 85
8.7 動作確認(タクトスイッチで監視操作) 86
8.8 動作確認(外出先から携帯で監視操作) 86
9 [HS007] メール受信時にR/Cサーボで旗を立てる 88
9.1 アプリケーション説明 89
9.2 回路図 89
9.3 配線図 90
9.4 スクリプトファイル設定と説明 90
9.5 メール設定 100
9.6 動作確認 100
10 パーツの説明 101
10.1 ブレッドボード 101
10.2 ジャンパー線 101
10.3 タクトスイッチ 101
10.4 リードセンサー 102
10.5 温度センサー 102
10.6 LED 102
10.7 ブザー 102

10.8 トランジスタアレイIC 103
10.9 光センサー(CDS) 103
10.10 抵抗 103
10.11 コンデンサ 104
11 更新履歴 104

1 このマニュアルについて

このたびは ホームセンサーキットをご利用いただき、誠にありがとうございます。

このマニュアルでは、サンプルアプリケーションの作成を通して、お客様自身のセンサーシステム構築時に役に立つ情報を提供することを目的としています。

サンプルアプリケーションの作成例では UIOUSB と DeviceServer を使用して、デバイスのI/O 操作やデータ収集、システム全体をコントロールする仕組みについてより理解を深めていただけます。ここで紹介した回路やスクリプトを土台にして、ニーズに合ったより使い易い機能をもったアプリケーションを構築してください。

この応用ガイドで作成するアプリケーションは、ホームセンサーキットに付属のパーツと基本的な電子工作用の工具(ニッパーやピンセット)などがあれば作成できます。テスター（マルチメータ）などの測定器があるとポートの値やセンサー出力電圧などが、直接確認できますので準備されることをお勧めします。

このマニュアルで紹介したアプリケーションの他にも、応用事例(アプリケーションノート)をオールブルーシステムのホームページで公開していますので参考にしてください(http://www.allbluesystem.com)

このマニュアルに対する意見や要望などがございましたら遠慮なく contact@allbluesystem.com までメールでお寄せください。応用事例として載せてほしいサンプルの要望も大歓迎です。

2 始めに

2.1 UIOUSB デバイスとサーバーのセットアップ

最初に、UIOUSB デバイスとサーバーソフトウエア(DeviceServer) が PC にセットアップされていることを確認してください。まだ、ホームセンサーキットの UIOUSB デバイスのセットアップとサーバーソフトウエアのインストールが完了していない方は、キットに付属のメディア中にある “セットアップガイド” マニュアルを参照して、先にセットアップを行ってください。

2.2 UIOUSB デバイスの回路を作成するときの注意

UIOUSB の配線について

UIOUSB デバイスを使用してアプリケーションの回路を作成するときには、必ずサーバーを停止した後、UIOUSB デバイスを PC から切り離した状態で作業してください。UIOUSB を PC に接続したまま回路を操作すると、配線中に誤ってショートした場合に **PC に回復不能なダメージを与える場合があります**。必ず PC から切り離した状態で配線してください。

UIOUSB デバイスの切り離しと接続は、サーバーソフトウエアが停止した状態で行います。サーバーの停止は スタートメニューから “All Blue System” -> “サーバー設定”を選択してサーバー設定プログラムを起動して、“サーバー停止”ボタンを押してください。しばらくすると DeviceServer が停止します。全てのサービスモジュールが停止するまでに数秒から１分程度の時間がかかります。

サーバーを起動する場合には、同様にサーバー設定プログラムを起動して、“サーバー起動”ボタンを押してください。しばらくすると DeviceServer が起動します。全てのサービスモジュールが起動するまでに数秒から１分程度の時間がかかります。

サーバーの起動と停止が確実に完了するまでは部分的にサービスモジュールが動作している状態ですので、全てのサービスモジュールが開始または終了するまで待ってください。サービスモジュールの動作状態は、ログを表示しておくと確認できます。詳しくは “セットアップガイド” の ”最初のアプリケーション”の章に詳しく手順が書いてありますので参照してください。

サーバーが完全に停止したら、UIOUSB デバイスを PC から切り離すことが出来ます。

次回 UIOUSB を PC に接続したときに、UIOUSB のコンフィギュレーションを事前にEEPROM に保存していた場合には、**保存していたコンフィギュレーションをロードして起動します。このときに、UIOUSB に配線されている回路がそのコンフィギュレーションでは不適切な場合には、最悪 UIOUSB が壊れてしまう場合があります。**例えばポートからサーボ信号を出力していたりシグナル出力可能な状態で、出力ポートを間違って “GND” や ”Vcc”、”常時閉” のスイッチに直接接続した場合などが考えられます。

それらを回避するために念のため、回路変更前には現在の UIOUSB のコンフィギュレーションをクリアしておくことをお勧めします。UIOUSB のコマンド “clear_eeprom” をWindows のハイパーターミナルやターミナルエミュレータプログラムから実行することでクリアできます。サーバーが停止した状態でないとハイパーターミナルやターミナルエミュレータプログラムは UIOUSB の COM ポートに接続できませんので注意してください。ハイパーターミナルの操作方法は、“セットアップガイド” のインストール手順で記載されていますので参考にしてください。サーバー実行中に UIOUSB のコンフィギュレーションをクリアすることもできます。”UIOUSB_CLEAR” スクリプトを実行してください。このスクリプトファイルはインストール時に設定されていて、”clear_eeprom” コマンドを実行します。

アプリケーション回路作成は以下の手順で行います。

① “サーバー設定プログラム”で サーバーを停止する
② UIOUSB の現在のコンフィギュレーションをクリアする。または、作成する回路に適したコンフィギュレーションを設定してあからじめ EEPROM に保存しておく。
③ UIOUSB デバイスを PC から切り離す(USB ケーブルを抜く)
④ 回路を配線する。電源とグランド間のショートに特に気をつけてください。
⑤ UIOUSB デバイスを PC に接続する。Windows の仮想COM ポートとして自動認識されることを確認してください
⑥ “サーバー設定プログラム”で サーバーを起動する
⑦ スクリプトを作成・修正してアプリケーションを作成する

アプリケーションの動作テスト中に、スクリプトに問題が見つかった場合には、⑦を繰り返してアプリケーションを完成させていきます。スクリプトを修正するときに回路の変更を伴わない場合には、サーバーの再起動は必要ありません。スクリプトはエディタで修正すると直後に有効になります。

UIOUSB のコンフィギュレーションもサーバー起動中に変更できます。ただしこの場合は新しいUIOUSB のコンフィギュレーションが現在の回路にマッチしていることを十分確認してください。

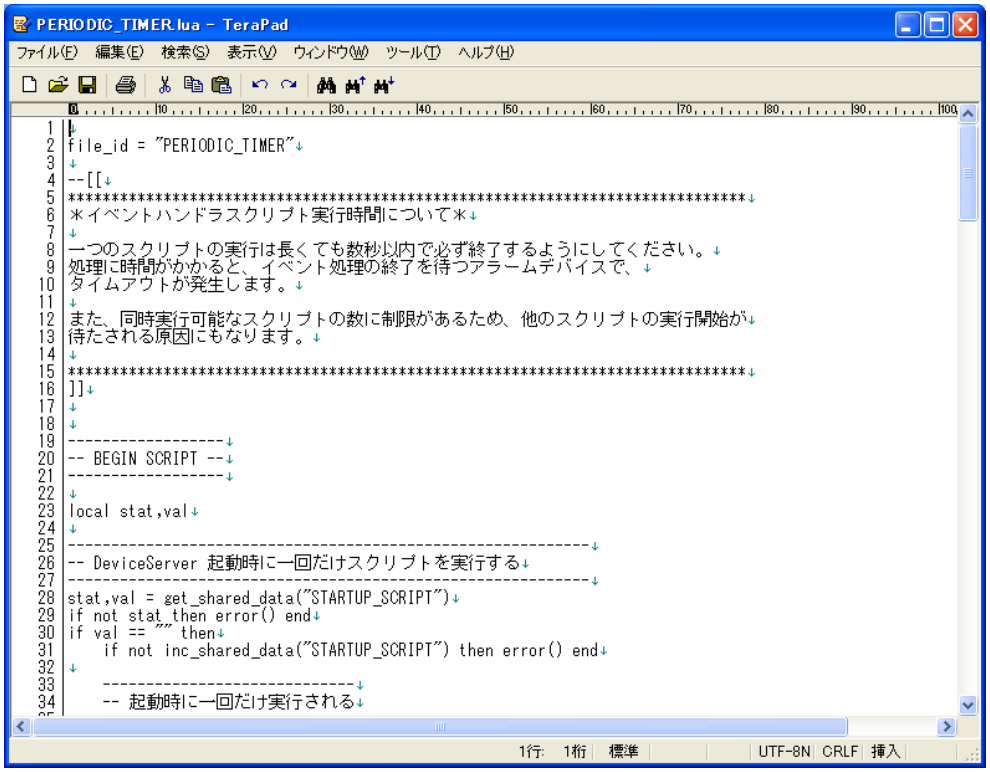
⚠Windows から UIOUSB が認識されない、Windows で認識と切断を繰り返えして音が鳴る
⑤ で Windows に UIOUSB デバイスが認識されないときには、回路が適切に配線されていない可能性があります。時に電源回りを十分に確認してから PC に接続してください。

2.3 スクリプト編集用エディタを用意する

サーバー側で動作するイベントハンドラやスクリプトはテキストエディタで簡単に作成できます。

⚠スクリプト中で日本語を使用する
日本語を使用する場合には必ず UTF-8N(BOMなし) のエンコード形式で保存してください。 UTF-8(BOMあり)や Windows 標準の Shift_JIS 形式、その他のエンコード形式では動作しませんので注意してください。

このため、UTF-8 文字コードで編集ができるエディタソフトを準備してください。
Windows付属のワードパッドやメモ帳ではこの形式で保存できませんので、別途 UTF-8N 形式で保存可能なエディタソフトを使用してください。エディタソフトウエアはフリーソフトの TeraPad (下記 URL 参照)をお勧めします。
http://www5f.biglobe.ne.jp/~t-susumu/



TeraPad で DeviceServer のスクリプトファイルを編集している画面です。この様に日本語をスクリプト中に記述する場合には、画面右下に表示されているコードが “UTF-8N” になっている必要があります。もし “SJIS” になっている場合には、ファイルメニューから”文字コード指定保存”を選択して、”UTF-8N” を選択して一度保存してから、再びオープンして編集作業を行ってください。スクリプトファイル中に日本語が含まれていない場合には “SJIS” のままで構いません。

アプリケーション動作中にスクリプトをエディタで編集した場合でも、サーバーはすぐに新しいスクリプトに従って動作しますので、サーバーを再起動させる必要はありません。

2.4 スクリプトファイルの実行について

ユーザーが作成したスクリプトファイルはイベントハンドラスクリプトとユーザースクリプトに区別されます。

イベントハンドラスクリプトは、UIOUSB デバイスから自動サンプリングデータを受信した場合や、I/O 値が変化した場合等に自動で実行されます。その他にも DeviceServer のイベント発生時(メール受信、サーバー起動 その他...)に予め決められたファイル名のスクリプトファイルがそれぞれ自動で実行されます。これらのイベントハンドラについての詳しい説明は、”DeviceServer ユーザーマニュアル”中の “イベント”の章を参照して下さい。

イベントハンドラスクリプトは、必ず ”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダ直下に配置します。サブフォルダを作成してその中にイベントハンドラスクリプトを置くことはできません。サブフォルダを使用したい場合には、イベントハンドラ中からサブフォルダ中に配置したスクリプトファイル(ユーザースクリプト)をコールするようにします。全てのイベントハンドラスクリプトファイルは、DeviceServer インストール時にデフォルトのファイルが作成されています。

ユーザースクリプトを実行するためには、上記のイベントハンドラ中からコールする方法と、外部から手動で実行する方法があります。手動でスクリプトを実行する方法についてはこのマニュアル中の “HS001 LED の点滅” サンプルの ”動作確認（スクリプトファイルの手動実行方法)” の項で説明しています。

この応用ガイド中で説明している UIOUSB のコンフィギュレーションを行うスクリプト(***_CONF.lua)などは、一回だけ手動で実行する必要があります。

2.5 UIOUSBのリスタートについて

UIOUSB をサーバーに接続しているときに、UIOUSB デバイスに接続した回路の動作不良や、USB バスに他のデバイスを抜き差ししたために発生したUSB電源の変動などによって UIOUSB デバイスがリセットされる場合があります。このような原因で、UIOUSB デバイスの仮想シリアルポート(COMxx) が通信不能になった場合には、“サーバー設定プログラム” を使用してDeviceServer を再起動させることで回復させることができます。

DeviceServer を再起動しても UIOUSB デバイスに接続できない場合には、一旦 UIOUSB デバイスを PC から取り外して再度接続・認識させてください。

DeviceServer 全体を再起動させなくても、UIOUSB に関連する部分だけを自動でリスタートさせることもできます。”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにある PERIODIC_TIMER.lua スクリプトファイルを編集して、下記のコードを追加します。

ファイル名: C:¥Program Files¥AllBlueSystem¥Scripts¥PERIODIC_TIMER.lua

```
file_id = "PERIODIC_TIMER"
--[[
*********************************************************************************
＊イベントハンドラスクリプト実行時間について＊
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つアラームデバイスで、
タイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
*********************************************************************************
]]
if not uio_command("version") then
    if not service_module_restart("UIOUSB") then error() end
end
```

上記のスクリプトは、定期的に（１分に一回）UIOUSB デバイスで “version” コマンドを実行して、正常な応答が返ってこなかった場合には DeviceServer の UIOUSB サービスモジュールをリスタートさせます。

3 [HS001] LED の点滅

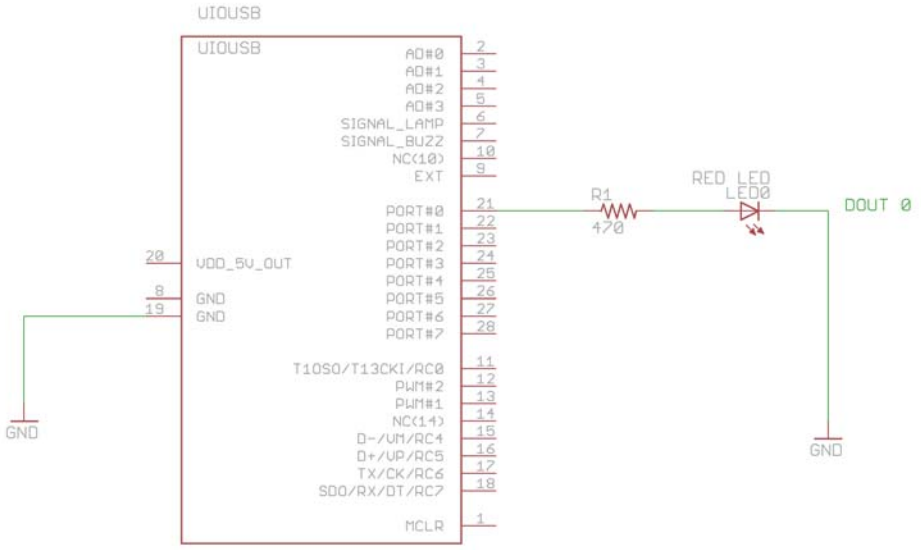
3.1 アプリケーション説明

I/O ポートに接続したLED を ５回点滅させます。

セットアップガイドの “最初のアプリケーション” の章で説明した回路と同一の回路を使用します。

セットアップガイドでは、Flash アプリケーションの GUI を使用してマウスで LED の点灯と消灯を操作しましたが、今回は自分で作成したスクリプトから I/O ポートを操作することで、自動で点滅動作を行います。

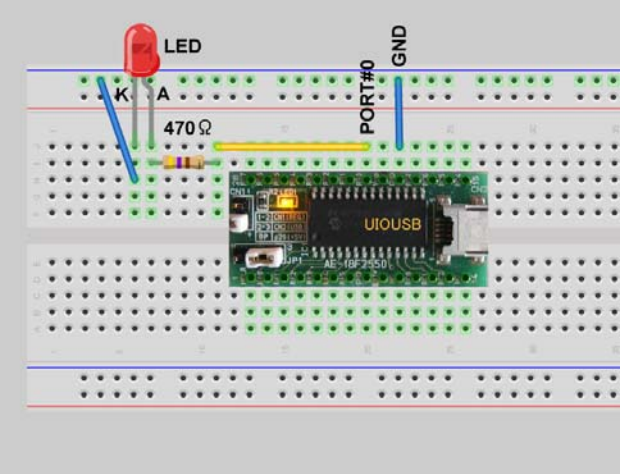
3.2 回路図



3.3 配線図

キットに付属の ブレッドボードで配線をする場合の例です。

ジャンパー線や部品の配置などは実際にパーツを配置した状態で調整してください。部品のリード線が長すぎる場合には、適当な長さになるようにニッパーで切断してください。配線をするときには、上記の回路図を見ながらピン番号を間違えないように注意しながら作業してください。



3.4 スクリプトファイル設定と説明

I/O ポートに接続したLED の点滅動作を行うスクリプトを作成します。

ファイル名:HS001_ONOFF.lua

キットに付属のメディア中の"スクリプトファイル¥HS001"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS001_ONOFF"

--[[
********************************************************************************
サンプルアプリケーション: LED の点滅
UIOUSB のポートビット#0 に 1(High),0(Low) を繰り返し出力する
********************************************************************************
]]
log_msg("start..",file_id)
-------------------------------------
-- UIOUSB 全ポートを出力に設定する。
-------------------------------------
if not uio_command("dcfg 0x0") then error() end
---------------------------------------
-- 5 回 High,Low 出力を繰り返す
---------------------------------------
local cnt
for cnt = 1,5,1 do
    if not uio_command("do0 1") then error() end;
    wait_time(500);
```

```
    if not uio_command("do0 0") then error() end;
    wait_time(500);
end
```

--[[と]] で囲まれた部分はコメント行です。また行中の -- より右側の部分もコメントです。

log_msg() は、ログにメッセージを出力するための関数です。メニューから “All Blue System” -> “ログコンソール” を選択してログ出力を画面にも表示している場合には、この関数で指定したメッセージがリアルタイムに出力されます。ログコンソールを起動していなかった場合でも、ログサーバーには全てのログが記録されています。ログサーバーで保存されているログを後で確認する場合には、ログコンソールを起動して”ログファイル切り替え” ボタンを押します。この操作でメモリ中に溜まっているログがファイルに出力されます。その後、”ログフォルダを開く” ボタンを押して、確認したいログファイルをエディタで開いて過去のログを表示できます。

セミコロン”;” は文の終端を表しています。スクリプト中の文の終端は Lua の構文から自動的に判断されますので、セミコロンを記述しなくてもエラーにはなりません。

local cnt は、スクリプト中で Lua ローカル変数 “cnt” を宣言しています。宣言をしないで変数を使用するともできます。ローカル宣言しておくと、使用した変数がスコープ（スクリプト全体、if, while 文などのブロック）から抜ける時に自動的に削除されますので、意図しない変数を不注意で使用するなどの不具合防止に役立ちます。

```
for cnt = 1,5,1 do
.....
```

end は、”for” と “end” の間を５回繰り返す指定です。

if not uio_command("do0 1") then error() end; は、UIOUSB デバイスに “do0 1” コマンドを送信しています。これによって LED が点灯状態になります。uio_command() の文を実行した結果によって、True またはFalse の論理値が返されるので、それをif文でチェックしてエラーが発生したときに、スクリプト動作を中止するようにしています。

error() は、スクリプト実行中にエラーを検出した場合などに、スクリプトの実行を中止するときにコールする関数です。error() 関数を実行するとサーバーのログ中にエラー発生が記録されます。もしエラー状態にしないでスクリプトを途中で終了したい場合には、error() の代わりに do return end; の様に記述します。

uio_command(“dcfg 0x0”) は UIOUSB デバイスにコンフィギュレーション設定コマンドを送信しています。ここではポートの全ビットを出力モードに設定しています。このスクリプト中では ”save” コマンドを使用していないので、ここで設定した出力モードは UIOUSB の電源が OFF になると(USB ケーブルを切り離すと)初期設定、または最後に “save” コマンドで保存したコンフィギュレーションに戻ります。

uio_command("do0 0") で I/O ポートのビット#0 にLow が出力されて LED が消灯します。

`wait_time(500)` は指定した 500ミリ秒間ウェイトします。

上記のライブラリ関数の仕様については、”DeviceServer ユーザーマニュアル”を参照してください。

`uio_command()` 関数のパラメータに指定した UIOUSB のコマンドの機能やパラメータの詳細については、”UIOUSB ユーザーマニュアル” のコマンドリファレンスに、詳しい機能が記載されていますので参照してください。

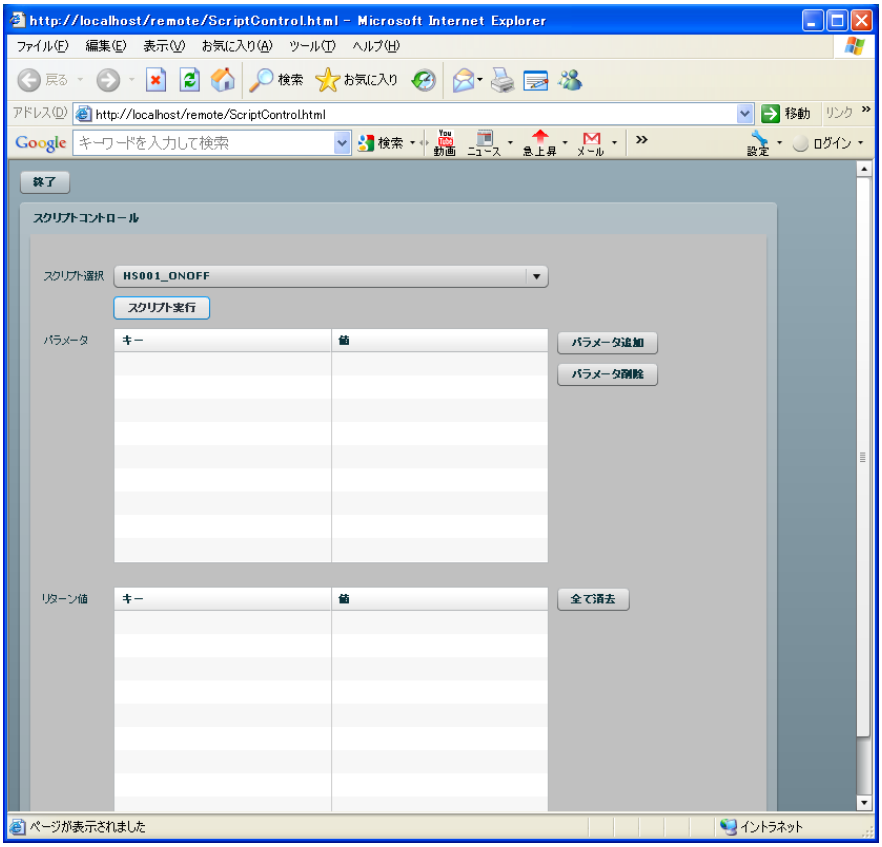
3.5 動作確認 (スクリプトファイルの手動実行方法)

作成した HS001_ONOFF スクリプトを実行させると、LED が点滅します。

このスクリプトを手動で実行するためには幾つかの方法があります。ここでは、それらの手動でスクリプトを実行する方法について簡単に説明します。

1. **ScriptControl Flashアプリ**

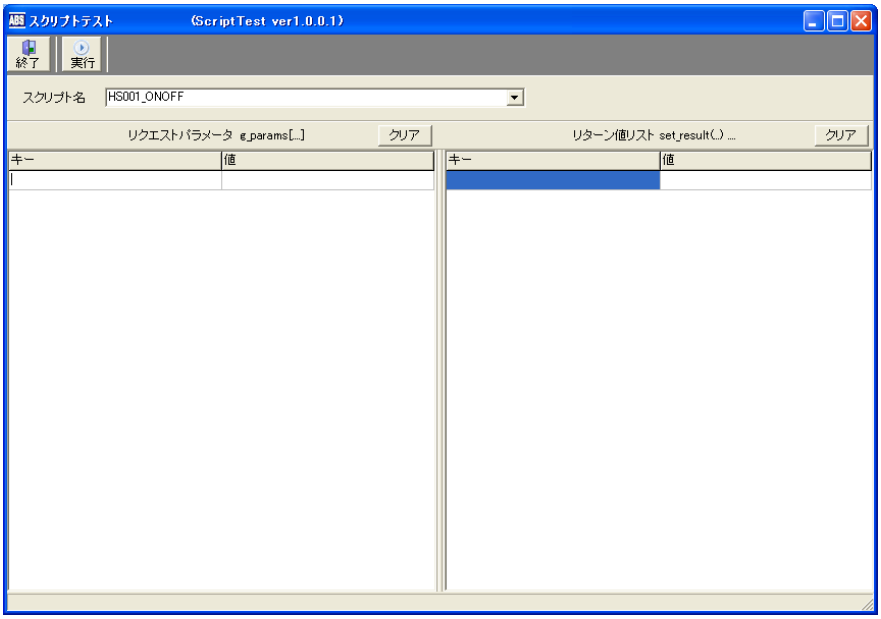
   Webブラウザ(インターネットエキスプローラなど)を起動して、http://localhost/remote/ScriptControl.html にアクセスするか、プログラムメニューから “ALL BLUE SYSTEM” -> “WebProxy-スクリプト操作” を選択します。ログイン画面でログインした後、下記の画面が表示されますので、実行したいスクリプトを選択して実行します。ログイン時に使用するユーザー名とパスワードは、セットアップガイドで作成した作業ユーザーアカウントのものを使用してください。



2. **“スクリプトテスト” DeviceServer クライアントプログラム**

   プログラムメニューから “ALL BLUE SYSTEM” -> “クライアント起動” を選択してログインの後、”スクリプト” ツールボタンを押して “スクリプトテスト” プログラムを起動します。ログインに使用するユーザー名とパスワードは、セットアップガイドで作成した作業ユーザーアカウントを使用してください。

スクリプトテストプログラムを実行すると下記の様な画面が表示されますので、スクリプト名のプルダウンメニューで実行したいスクリプトを選択します。実行ボタンを押すとスクリプトが実行されます。



3. ScriptExecCmd.exe コマンドプログラム

サーバーをインストールしたPC の “C:¥Program Files¥AllBlueSystem¥Demo” フォルダ中に “ScriptExecCmd.exe” プログラムが格納されています。このプログラムをWindows のコマンドプロンプトから “scriptexeccmd <ScriptName> <hostname> <user> <password>” の様なパラメータを指定して実行します。下記はユーザー名が “user” で パスワードが “xxxxxxx” の場合の実行例です。プログラム実行時に指定するパラメータを “ScriptExecCmd.ini” ファイルに記述して実行することもできます。

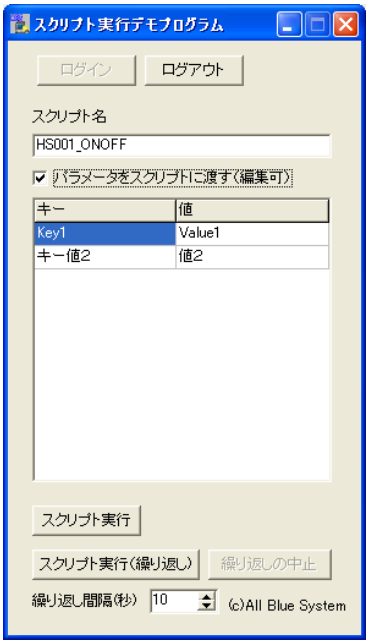
ユーザー名とパスワードは、セットアップガイドで作成した作業ユーザーアカウントを使用してください。

```
scriptexeccmd.exe HS001_ONOFF localhost user xxxxxxx
```

4. ScriptExecDemo GUI プログラム

サーバーをインストールしたPC の “C:¥Program Files¥AllBlueSystem¥Demo” フォルダ中に “ScriptExecDemo.exe” プログラムが格納されています。このプログラムを起動して、”ログイン” ボタンでログインした後、スクリプト名を指定して実行します。

ユーザー名とパスワードは、セットアップガイドで作成した作業ユーザーアカウントを使用してください。



5. WebAPI 経由でスクリプトを実行

Web ページ中の JavaScript や、Web サーバー側の perl などの CGI プログラムから http://<hostname>:<port>/script または http://<hostname>:<port>/json/script にアクセスして、その URL パラメータにスクリプト名やセッションを指定することで、DeviceServer のスクリプトを実行できます。あからじめ DeviceServer 側でセッション情報を作成しておくことで、ログイン操作を省略してリモート PC からスクリプトを実行できます。これらの、DeviceServer を HTTP 経由でアクセスするための、WebAPI 機能については “DeviceServerユーザーマニュアル” を参照してください。

6. DeviceServer API用ライブラリ (XASDLCMD.dll) を使用する

Win32 API を使用したWindows プログラムや EXCEL の VBA などから、API用ライブラリ(XASDLCMD.DLL) 経由でログイン操作やスクリプトを実行できます。

API用ライブラリ (XASDLCMD.dll) は、DeviceServer インストール時にインストールフォルダに格納されています。これらの詳しい使用方法については “DeviceServerユーザーマニュアル” を参照してください

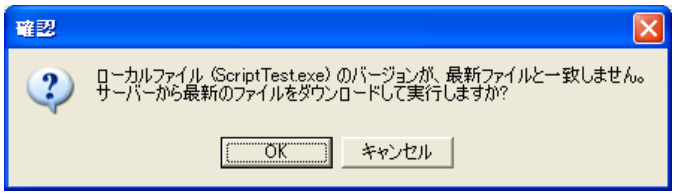
今回は ２番目の方法の “スクリプトテスト” DeviceServer クライアントプログラム を使用してスクリプトを実行します。プログラムメニューから “ALL BLUE SYSTEM” -> “クライアント起動” を選択してください。

セットアップガイドで作成した作業ユーザーアカウントのユーザー名とパスワードを入力します。

デスクトッププログラムが起動してツールボタンが表示されます。ここで “スクリプト” ボタンを押します。

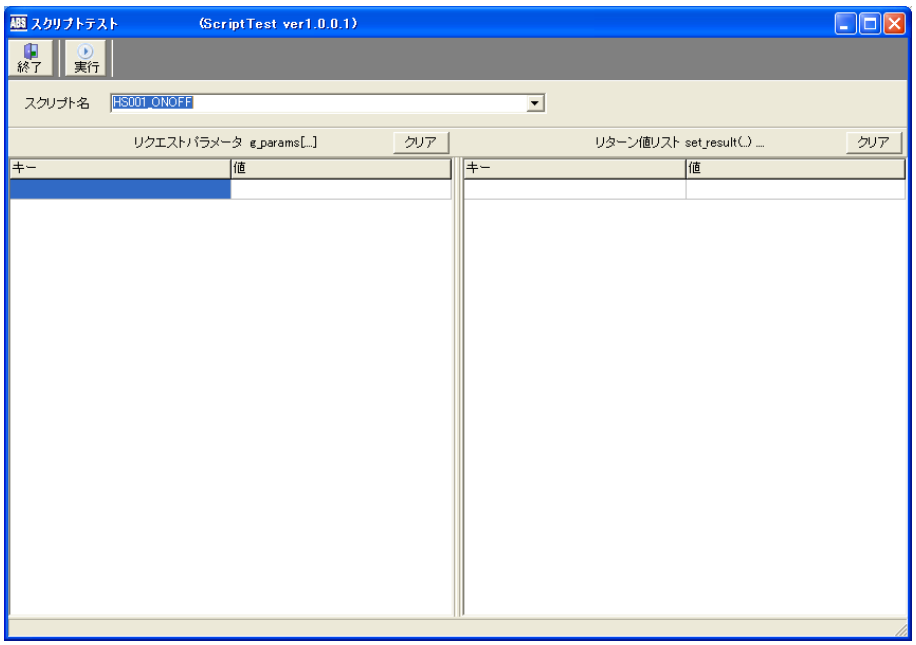
”スクリプト” ツールボタンが表示されない場合には、セットアップガイドで作業ユーザーアカウントを作成した時に、スクリプトテストプログラムを実行するための、アプリケーション許可フラグをチェックしていないのが原因です。このときはセットアップガイドに従って、管理者アカウントでユーザー管理プログラムを起動した後、検索ボタンを押して、作業ユーザーを表示した後、ダブルクリックで作業ユーザーを選択してユーザーアカウント修正画面に入ります。アプリケーション許可タブ中の “ScriptTest” にチェックを付けてユーザー情報を修正してください。

“スクリプト” ボタンを押すと下記の様な画面が表示される場合があります。

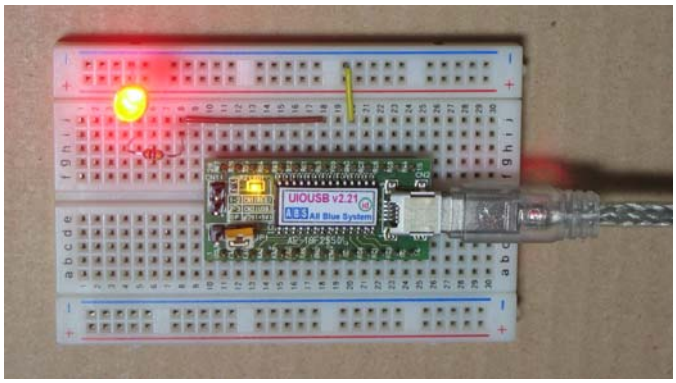


初めて、スクリプトテスト” DeviceServer クライアントプログラムを起動した場合には、この画面が表示されますので “OK” ボタンを押してください。

DeviceServer のクライアントプログラムは、プログラムのバージョンが DeviceServer の動作している PC 上に保管された最新のクライアントプログラムと一致しているかどうかを常にチェックしています。もし一致しない場合や、初めてクライアントプログラムを起動した場合には、サーバーPC からクライアントプログラムをダウンロードした後、プログラムを起動します。（今回はサーバーPC とクライアントPC が同一 PC になっています）



全てのスクリプト名がプルダウンメニューに表示されますので、”HS001_ONOFF” を選択して”実行”ボタンを押してください。UIOUSB に接続したLED が５回点滅するのを確認できると思います。



4 [HS002] ボタンを押したときにLED点滅＆ブザーを鳴らす

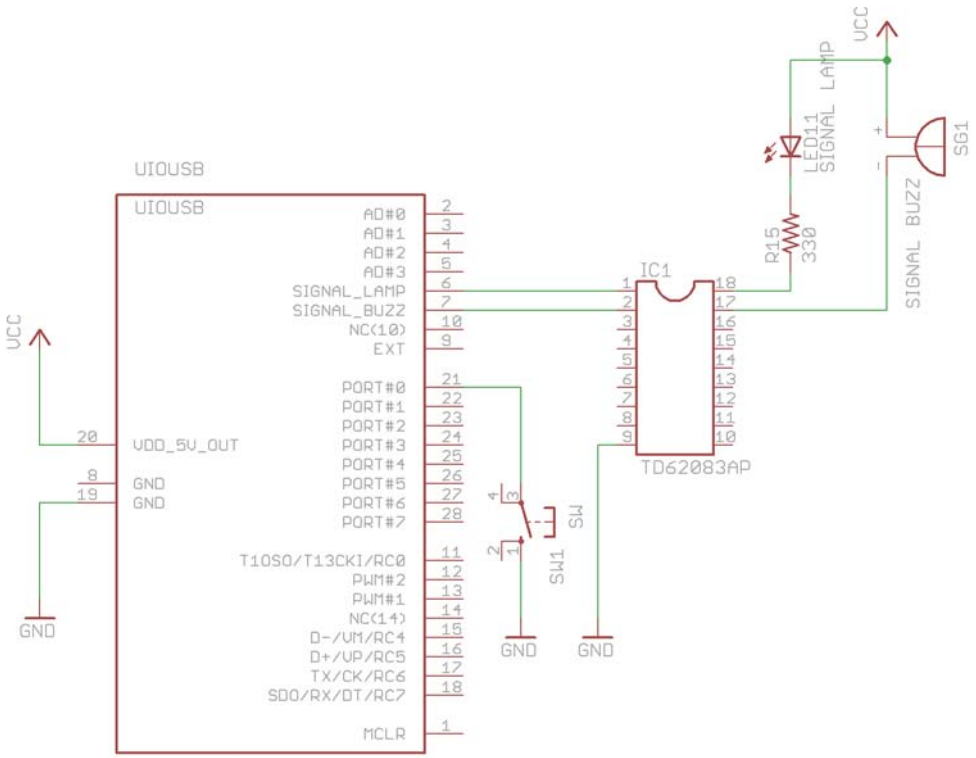
4.1 アプリケーション説明

タクトスイッチを押した時に、LEDが点滅すると同時にブザーを鳴らします。

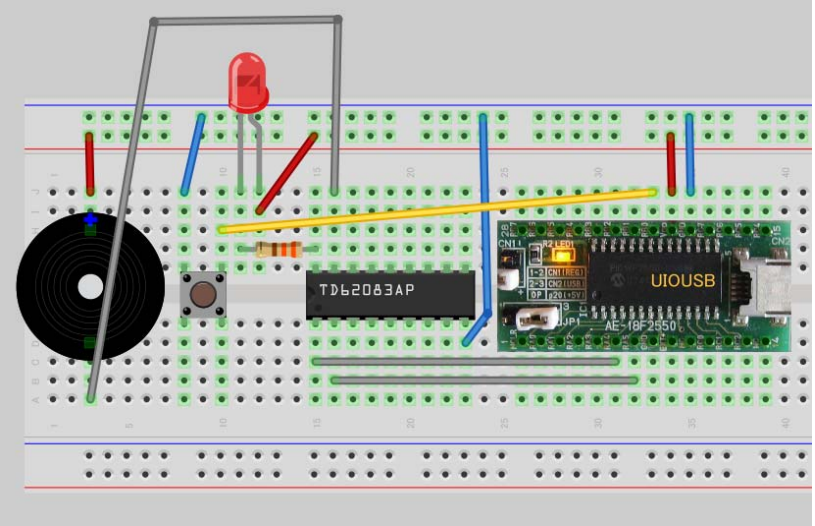
LED とブザーは通常のI/O ポート出力ではなく、UIOUSB のシグナル出力に接続します。これによって点滅やブザー音の間欠動作についてプログラムすることなく簡単に実現できます。

LED とブザーは、今回はドライバIC　経由で駆動しています。

4.2 回路図



4.3 配線図



4.4 スクリプトファイル設定と説明

最初に、UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS002_CONF.lua

キットに付属のメディア中の"スクリプトファイル¥HS002"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS002_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: ボタンを押した時にLED点滅とブザーを鳴らす
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFF") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x01") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`uio_command("dcfg 0xFF")` は、全てのポートを入力に設定します。

uio_command("pullup 1") は入力ポートのプルアップ設定を行います。

uio_command("change_detect 0x01") は、ビット#0 に接続したタクトスイッチの変化を検出します。検出すると、イベントハンドラスクリプト(UIOUSB_EVENT_DATA) が自動的に実行されます。

次に、タクトスイッチを押した時に実行されるスクリプト(UIOUSB イベントハンドラ)を作成します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の"スクリプトファイル¥HS002"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"
--[[
********************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値                          値                                          例
--------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                         2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA_1 は常にイベントプリフィックス文字列を表す          "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                             "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る
```

```
                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  -- ビット#0 が変化している時を選択
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) then
    -- ビット#0の 値が 0 の時を検出（ボタンプッシュ時）
    if (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then
      log_msg("ボタン押した",file_id);
      if not uio_command("signal_write 0x18") then error() end;
    end;

    -- ビット#0の 値が 1 の時を検出（ボタンリリース時）
    if (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 1) then
      log_msg("ボタン離した",file_id);
      if not uio_command("signal_write 0x00") then error() end;
    end;
  end;
end;
```

`file_id = "UIOUSB_EVENT_DATA"` は、log_msg() コマンドでログにメッセージを出力するときのモジュール名を統一させるための変数です。この変数を使用しないで log_msg() の第二パラメータで文字列を指定しても構いません。

`log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id)` はこのイベントハンドラスクリプトがコールされたときの全てのイベントパラメータをログに出力しています。
`g_params["EVENT_DATA_WHOLE"]`は、カンマ区切りの全てのイベントデータが文字列で格納されています。

UIOUSB イベントデータ(CHANGE_DETECT) の内容は下記の様になっています(UIOUSB ユーザーマニュアルより抜粋)

```
$$$,CHANGE_DETECT,[diff_bits],[port_val]
```

[diff_bits] には、変化したビットを 1、変化していないビットを 0にした値が、16進数で 00 から FF までの値で、先頭の"0x"部分を取り除いて大文字で表記したものが入ります。

[port_val] には 現在のポート値が16進数で 00 から FF までの値で、先頭の"0x"部分を取り除いて大文字で表記したものが入ります。

上記のカンマ区切りの最初のデータ "$$$" は、イベントハンドラでは `g_params["EVENT_DATA_1"]` に格納されいま

す。後は順に “CHANGE_DETECT” が g_params[“EVENT_DATA_2”] の様に格納されています。

if g_params[“EVENT_DATA_2”] == “CHANGE_DETECT” then では、このイベントハンドラスクリプトが実行されたときに、UIOUSB のどのイベントが発生していたかを調べています。イベントハンドラでは、イベントごとにあからじめ決められたイベントパラメータが g_params[] 配列(文字列をキーとした連想配列)に格納されています。
g_params[“EVENT_DATA_2”] は、UIOUSB イベントの２つめのイベントデータを指定していて、イベントの種類を表します。ポートの値が変化した時にはイベント種類は “CHANGE_DETECT” になります。その他にもUIOUSB には A/D 変換値が決められた範囲よりも変化した時のイベント “ADVAL_UPDATE” などがあります。イベントハンドラスクリプトに渡されるパラメータの詳細は、”UIOUSBユーザーマニュアル”のイベントリファレンスの章を参照してください。

if (bit_and(tonumber(g_params[“EVENT_DATA_3”],16),0x01) > 0) then は、ポートのビット#0 が変化しているかどうかをチェックしています。まず、g_params[“EVENT_DATA_3”] には、CHANGE_DETECT イベント発生時に変化しているポートのビット位置が16 進数文字列で格納されています。例えば g_params[“EVENT_DATA_3”] の値が “FF” の場合には全てのポートのビットが変化していることを示します。tonumber(g_params[“EVENT_DATA_3”],16) で、この16進数の文字列を数値に変換します。bit_and(tonumber(g_params[“EVENT_DATA_3”],16),0x01) で算術 AND 演算を行って、ビット位置#0 が変化しているかどうかを、0 またはそれ以外の値になることで判断しています。

if (bit_and(tonumber(g_params[“EVENT_DATA_4”],16),0x01) == 0) は、ポートのビット#0 の値がイベント発生時に Low になっていたかどうかを調べています。Low の場合にはタクトスイッチが押し込まれた状態を意味します。このスクリプトは、タクトスイッチを押したとき１回目が実行されて、その後タクトスイッチを離したときに２回目が実行される点に注意してください。g_params[“EVENT_DATA_4”] には、CHANGE_DETECT イベント発生時に、変化しているときのポート値が16 進数の文字列で格納されています。

uio_command(“signal_write 0x18”) は、UIOUSB のシグナル出力でランプ点滅と間欠ブザーを出力する設定です。

signal_write コマンドで指定するパラメータ(シグナルステータス) は下記の様になっています。
(UIOUSB ユーザーマニュアルより抜粋)

UIOUSB には ランプの点滅やブザー出力でアラーム出力の機能があります。PORTA の RA4 と RA5 は常に出力ポートとしてこれらの出力を行っています。
UIOUSB コマンドでランプやブザーのシグナルステータスを変更することで、シグナル出力ポートから “点滅” やブザー用の２種類の出力パターンが連続的に出力されます。

シグナルステータスは 8it幅のバイトデータで、各ビットは下記の意味があります。

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
BEEP2			LED 点滅	BEEP1			LED 点灯

上記のビット位置が “1” の時に、該当シグナル状態が ON であることを示します。”0” の時に、該当シグナル状態が

OFF であることを示します。(空白の部分のステータスビットは使用していませんので値を設定しても無視されます)

シグナルステータスの値は、”signal_write”, “signal_bit”, “signal_read” コマンドで変更または参照できます。

現在のシグナルステータスの値によって、UIOUSB は PORTA に下記の出力を行います。

PORTA	RA5	シグナル ブザー出力
	RA4	シグナル LED 出力

シグナル LED は PORTA の RA4 に以下の値を出力することを意味します。

点灯 ON(High) または OFF(Low)

点滅 (High-Low 繰り返し)

また、PORTA の RA5 (シグナル ブザー) には、以下の値を出力します。

OFF(Low)

BEEP1(High-Low の 早い繰り返し) ブザー音 : PiPiPiPi....

BEEP2(High-Low の ゆっくりとした繰り返し) ブザー音 : Pi-,Pi-,Pi-....

0x18 をシグナルステータスに設定すると BEEP1 と LED 点滅を指定したことになります。

if (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 1) then は、ビット#0 の値がイベント発生時に High になっていたかどうかを調べています。High の場合にはタクトスイッチが離された状態を意味します。(入力ポートがプルアップされているため)。

uio_command("signal_write 0x00") は、UIOUSB のシグナル出力でランプ点滅と間欠ブザー停止する設定です。

4.5 動作確認

最初に、UIOUSB デバイスの設定を行います。HS002_CONFIGスクリプトファイルを手動で実行してください。

スクリプトの実行方法は、最初のアプリケーション “HS001 LEDの点滅”の動作確認で説明した方法で行います。

HS002_CONFIG スクリプト中に UIOUSB デバイス内部の EEPROM に設定内容が保存するコマンド "save" が入っていますので、電源をOFF にした場合でもこのコンフィギュレーション設定のスクリプトを繰り返し実行する必要はありません。

設定が終了したら、タクトスイッチを押してみてください。タクトスイッチを押している間だけ、LED の点滅とブザーが鳴ると思います。

チャタリングの防止

タクトスイッチを操作すると、非常に短い時間に ON と OFF を繰り返すチャタリングが発生しています。

UIOUSB ではそれらをフィルタ処理してからイベントを発生させています。このためポート値が変化したときのイベントハンドラではこれらのチャタリングを考慮する必要はありません。

詳しくは、“UIOUSBユーザーマニュアル”の イベントリファレンスの章を参照してください。

4.6 応用(1)最初にボタンを押した時にランプとブザーをつけて、２回目に押した時に消す

同じ回路のままで、タクトスイッチを押した時にLED の点滅とブザーを鳴らして、スイッチを離したときもその状態を維持し続けるように改造します。そして２回目にタクトスイッチを押した時にLED とブザーを停止させます。

回路の配線と UIOUSB のコンフィギュレーションは同一ですので変更する必要はありません。

次に、スイッチを押した時に実行されるスクリプト(UIOUSB イベントハンドラ)を作成します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS002¥応用1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
********************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値                          値                                          例
--------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                         2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA_1 は常にイベントプリフィックス文字列を表す          "$$$"
                    "$$$" 文字列
```

```
                    EVENT_DATA_2 はイベント名が入る                                    "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  -- ビット#0 が変化していて、かつビット#0 の値が 0 の時を選択
  --（ボタンを押した時）
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) and
              (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then
    local stat,val = inc_shared_data("BUTTON_COUNT")
    if not stat then error() end
    if (tonumber(val) == 1) then
      -- ボタンを1回目に押した
      if not uio_command("signal_write 0x18") then error() end;
    else

      -- ボタンを2回目に押した
      if not set_shared_data("BUTTON_COUNT","") then error() end
      if not uio_command("signal_write 0x00") then error() end;
    end
  end;
end;
```

`if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) and bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then` は、スイッチが押された時(スイッチを離した時は無視する）ための判断文です。

`local stat,val = inc_shared_data("BUTTON_COUNT")` は、サーバーのグローバル共有変数機能を使用してボタンの押された回数をカウントしています。"BUTTON_COUNT"（別の名前でも構いません）のキー名を持った、グローバル共有変数を見つけて、その値(文字列）を数値として見た場合にそれに１を足した値を設定して同時に val 変数にその値を取得します。例えば既存の "BUTTON_COUNT" 共有変数の内容が "0"（文字列）の場合には、val には "1"（文字列)が返ります。既存の "BUTTON_COUNT" グローバル共有変数が見つからなかった場合には常に "1" が返ります。サーバーには全てのイベントハンドラやスクリプト間で共有するグローバル共有変数と、ログインセッションごとに独立したセッション共有変数、サーバーが再起動した時には自動的に削除されるグローバル共有変数に対して、永続化（データベース）機能をつけたパーマネント共有変数があります。詳しい説明は "DeviceServer ユーザーマニュアル"を参照してください。

`if (tonumber(val) == 1) then` は最初にボタンを押した場合を判断しています。`inc_shared_data("BUTTON_COUNT")` 関数はスレッドセーフになっています。これによってイベントハンドラや他のスクリプト中で同時に同じグローバル共有変数に対して実行した場合でも、インクリメント操作と値の取得が確実に行われることを保障しています。この `inc_shared_data()` で取得した値を判断することでスイッチが何回目に押されたかを判断しています。

`if not set_shared_data("BUTTON_COUNT","") then error() end` は、グローバル共有変数を設定する関数です。パラメータに空文字列 “” を渡すと、指定したグローバル共有変数が削除されます。

4.7 応用(2)ボタンを押した時にランプとブザーをつけて、しばらく経つと自動的に消す

次の応用は、タクトスイッチを押した時にLED の点滅とブザーを鳴らして、スイッチを離したときもその状態を維持し続けるようにしますが、その後タクトスイッチを操作しなくても 10秒後に自動的にLED とブザーを停止させます。

回路の配線は同一ですので変更する必要はありません。

最初に、UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS002_CONF.lua

キットに付属のメディア中の”スクリプトファイル¥HS002¥応用2”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS002_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: ボタンを押した時にLED点滅とブザーを鳴らす
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFF") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x01") then error() end
if not uio_command("signal_timer 10") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`if not uio_command("signal_timer 10") then error() end` は、シグナル出力(LED やブザーの出力)を行った時に自動的にリセット(“signal_write 0”コマンドを実行するのと同じ) されるまでの時間(秒)を設定しています。デフォルトでは”signal_timer” には 0 が設定されていて無効になっていますので、1(秒) 以上の値を指定することで有効になります。

次に、スイッチを押した時に実行されるスクリプト(UIOUSB イベントハンドラ)を作成します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS002¥応用2”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
********************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値                          値                                          例
--------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                         2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA_1 は常にイベントプリフィックス文字列を表す           "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                                 "SAMPLING"
```

```
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る


                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);


if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  -- ビット#0 が変化している時を選択
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) then
    -- ビット#0の 値が 0 の時を検出（ボタンプッシュ時）
      if (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then
        log_msg("ボタン押した",file_id);
        if not uio_command("signal_write 0x18") then error() end;
      end;
  end;
end;
```

5 [HS003] リードスイッチでドアの開閉を監視する

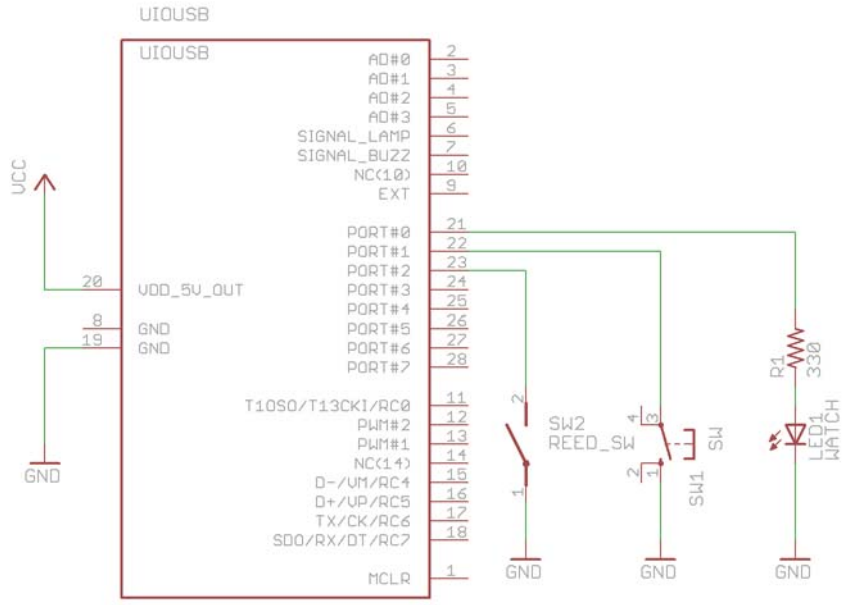
5.1 アプリケーション説明

このアプリケーションはドアや窓などの開閉を監視するためのリードスイッチを使用して監視を行います。

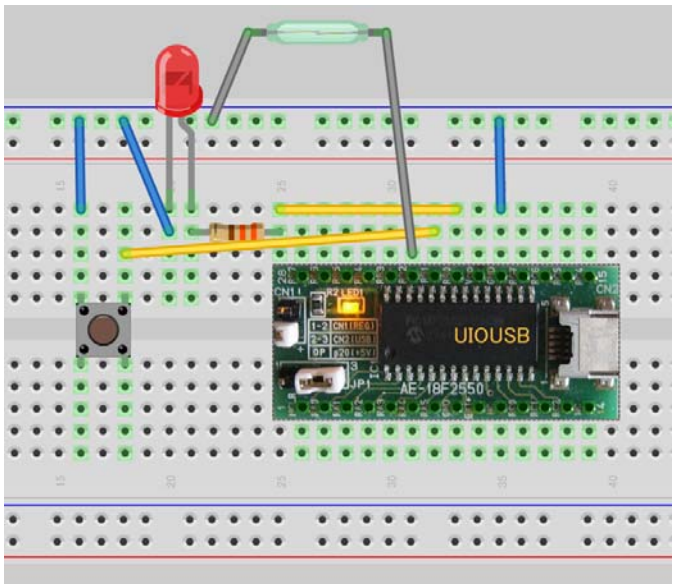
タクトスイッチで監視の開始とストップを指示します。監視中は LED が点灯します。

もし、監視中にドアの開閉があった場合には LED が点滅して知らせます。

5.2 回路図



5.3 配線図



リードスイッチの末端はよじってブレッドボードに挿しやすくしてください。もしケーブルが入りづらい場合には、抵抗のリードの切れ端などをケーブルの末端に半田付けしてください。センサーのケーブルは電線で延長できますので設置場所に応じて継ぎ足してください。延長する電線があまり長くなるとノイズの影響で誤動作することがありますので数メートル以内にしてください。

リードスイッチはドアや窓枠などにネジや両面テープで設置します。リードスイッチ本体は動かない窓枠側に固定して、磁石側を窓板やドア側に設置するほうが使いやすいと思います。

5.4 スクリプトファイル設定と説明

UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS003_CONF.lua

キットに付属のメディア中の"スクリプトファイル¥HS003"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS003_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: リードスイッチでドアの開閉を監視する
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFE") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x06") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`if not uio_command("dcfg 0xFE") then error() end` は、ポートビット#0 を出力に設定して、それ以外のビットを全て入力に設定しています。

`if not uio_command("pullup 1") then error() end` は、入力ポートのプルアップを有効にしています。

`if not uio_command("change_detect 0x06") then error() end` は、入力ポートのビット#1(リードスイッチ)またはビット#2(タクトスイッチ) が変化した場合に UIOUSB イベントを発生させるように指定しています。イベントが発生するとUIOUSB_EVENT_DATA イベントハンドラスクリプトが実行されます。

次に、タクトスイッチを押した時やリードスイッチが動作したときに実行されるスクリプト(UIOUSB イベントハンドラ)を作成します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS003”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
*******************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値                          値                                          例
-------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                        2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA__1 は常にイベントプリフィックス文字列を表す          "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                              "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

local stat,val,flag,taskid,handle;
if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
```

```
  ------------------------------------------------------------
  -- ビット#1 が変化していて、かつビット#1 の値が 0 の時を選択
  -- 監視開始/終了ボタンを押した時
  ------------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x02) > 0 ) and
    (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x02) == 0) then
    stat,val = inc_shared_data("ALARM_ONLINE_FLAG")
    if not stat then error() end
    if (tonumber(val) == 1) then
      -- ボタンを1回目に押した
      if not uio_command("do0 1") then error() end;
      log_msg("監視開始",file_id);
    else
      -- ボタンを2回目に押した
      if not set_shared_data("ALARM_ONLINE_FLAG","") then error() end
      if not uio_command("do0 0") then error() end;
      log_msg("監視終了",file_id);
    end
  end;
  ------------------------------------
  -- ビット#2 が変化している時を選択
  -- リードスイッチが変化している時
  ------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x04) > 0 ) then
    ------------------------------------
    -- 監視中の場合のみアラートを検出する
    ------------------------------------
    stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
    if not stat then error() end
    if (flag ~= "") then
      log_msg("アラート検出",file_id);
      ------------------------------------------------
      -- 別スレッドでランプの点滅タスクを起動する
      ------------------------------------------------
      stat,taskid = script_fork_exec("HS003_ALARM","","")
      if not stat then error() end
    end;
  end;
end;
```

`stat,val = inc_shared_data("ALARM_ONLINE_FLAG")` は、グローバル共有変数を利用してボタンを押した回数を判断しています。最初のイベントでこの文を実行すると val には “1” が得られ、次に実行すると “2” が得られます。

`if not uio_command("do0 1") then error() end;` は、監視開始と同時にポートビット#0 に接続した LED を点灯させています。

`if not set_shared_data("ALARM_ONLINE_FLAG","") then error() end` は、ボタンが２回目に押された時に実行されて、ボタンを押した回数を保存したグローバル共有変数を削除しています。これによって再びボタンが押された場合に１回目として判断されるようになります。

`if not uio_command("do0 0") then error() end;` は監視終了と同時にポートビット#0 に接続した LED を消灯させています。

```
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then error() end
if (flag ~= "") then
```

は、ボタンの押した回数を保存しているグローバル共有変数が存在するかどうかを判断しています。ボタンを１回目に押した後は flag に “1” が取得できるので if 文の内部が実行されます。

`stat,taskid = script_fork_exec("HS003_ALARM","","")` は、LED を点滅させるためのスクリプトを実行しています。この UIOUSB イベントハンドラ中で HS003_ALARM スクリプトで記述しているのと同などの処理を行うと、LED 点滅処理が終了するまでイベントハンドラを抜け出なくなってしまいます。これを防ぐために、イベントハンドラでは時間がかかる処理を直接実行しないで、script_fork_exec() を使用して別スレッドで実行するようにします。

イベントハンドラは別スレッドで並行して実行されます

DeviceServer で発生するイベントによって実行されるイベントハンドラスクリプトは全て別スレッドで実行されています。そのためサーバーで UIOUSB イベントハンドラ実行中に、同一イベントが発生すると別スレッドで UIOUSB イベントハンドラが並行して実行されます。

別スレッドでスクリプトを実行する

スクリプトから別のスクリプトを実行するときにscript_exec() を使用すると同一スレッドでスクリプトを実行して、コールされたスクリプトの処理が終了するまでコールした側の実行が待たされます。

script_fork_exec() を使用すると別スレッドでスクリプトが実行されて、コールされたスクリプトの終了を待たずに並行してスクリプトが実行されます。

次に、アラーム状態を知らせるための LED 点滅を行うスクリプト(HS003_ALARM)を作成します。

ファイル名:HS003_ALARM.lua

キットに付属のメディア中の"スクリプトファイル¥HS003"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS003_ALARM"
--[[
********************************************************************************
UIOUSB のポートビット#0 に 1(High),0(Low) を繰り返し出力する
********************************************************************************
]]
log_msg("start..",file_id)
local cnt,stat,flag;
----------------------------------------
-- 50 回 LED 点滅を繰り返す
----------------------------------------
for cnt = 1,50,1 do
  ------------------------------------------------
  -- 100ms ごとに LED の点灯と消灯を繰り返す
  ------------------------------------------------
  if not uio_command("do0 1") then error() end;
  wait_time(100);
  if not uio_command("do0 0") then error() end;
  wait_time(100);
end


------------------------------------------------------
-- 現在監視中かどうかを判断して、このタスクを抜けるときに
-- LED を点灯または消灯のどちらの状態にするかを決める
------------------------------------------------------
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then error() end
if (flag ~= "") then
  if not uio_command("do0 1") then error() end;
else
  if not uio_command("do0 0") then error() end;
end;


log_msg("end.",file_id)
```

for cnt = 1,50,1 do

end

は、50 回繰り返し実行するための文です。

wait_time(100); で 100ms ウェイトを入れています。

stat,flag = get_shared_data("ALARM_ONLINE_FLAG") はLED 点滅を行った後に、まだ監視中かどうかを調べるためにグローバル共有変数を調べています。これは LED 点滅中に監視を中止することがあるためです。

5.5 動作確認

最初に、UIOUSB デバイスの設定を行います。HS003_CONFIGスクリプトを手動で実行してください。スクリプトの実行方法は、最初のアプリケーション “HS001 LEDの点滅”の章を参照してください。

タクトスイッチを押すと LED が点灯して監視状態に入ります。その後、リードスイッチが動作すると LED が点滅してセンサー動作を知らせます。再び、タクトスイッチを押すと LED が消灯してその後はリードスイッチが動作してもLED は点滅しません。

5.6 改良(1)リードスイッチが連続して反応した時に対応

監視中にリードスイッチを短い時間に連続して感知させると、どの様になるかを試してみてください。
LED の点滅が不規則になり表示がおかしくなると思います。また、短い時間間隔でリードスイッチを反応させ続けるとサーバーでエラーが発生する場合があります。

この原因は、LED 点滅動作をさせるスクリプト(HS003_ALARM)が並行していくつも同時に実行しているためです。
サーバーで並行して同時実行可能なスクリプトの数は 15 までで、これ以上のスクリプトを別スレッドで起動した場合には、先に実行中のスクリプトが終了してから実行されます。

これらの問題が発生するのを防ぐためにグローバル共有変数を使用して、既に HS003_ALARM が実行中の場合には平行して同一のスクリプトを実行しないようにします。

最初に、UIOUSB イベントハンドラスクリプトファイルを修正します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS003¥改良1”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"


local stat,val,flag,taskid,handle;
```

```
if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  ------------------------------------------------------------
  -- ビット#1 が変化していて、かつビット#1 の値が 0 の時を選択
  -- 監視開始/終了ボタンを押した時
  ------------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x02) > 0 ) and
     (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x02) == 0) then
    stat,val = inc_shared_data("ALARM_ONLINE_FLAG")
    if not stat then error() end
    if (tonumber(val) == 1) then
      -- ボタンを1回目に押した
      if not uio_command("do0 1") then error() end;
      log_msg("監視開始",file_id);
    else
      -- ボタンを2回目に押した
      if not set_shared_data("ALARM_ONLINE_FLAG","") then error() end
      if not uio_command("do0 0") then error() end;
      log_msg("監視終了",file_id);
    end
  end;

  ------------------------------------
  -- ビット#2 が変化している時を選択
  -- リードスイッチが変化している時
  ------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x04) > 0 ) then
    ---------------------------------------
    -- 監視中の場合のみアラートを検出する
    ---------------------------------------
    stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
    if not stat then error() end
    if (flag ~= "") then
      log_msg("アラート検出",file_id);
      ------------------------------------------------------------
      -- ランプが既に点滅中の場合には重複してタスクを起動しない
      ------------------------------------------------------------
      local stat2,val2 = inc_shared_data("LAMP_BLINKING")
      if not stat2 then error() end
```

```
      if (tonumber(val2) == 1) then
        ------------------------------------------------
        -- 別スレッドでランプの点滅タスクを起動する
        ------------------------------------------------
        stat,taskid = script_fork_exec("HS003_ALARM","","")
        if not stat then error() end
      end;
    end;
  end;
end;
```

```
local stat2,val2 = inc_shared_data("LAMP_BLINKING")
if not stat2 then error() end
if (tonumber(val2) == 1) then
```

部分で、val2 に取得した"LAMP_BLINKING" グローバル共有変数が "1" になるかどうかで、スクリプトが既に起動中かどうかを判断しています。"LAMP_BLINKING" グローバル共有変数は、LED 点滅スクリプト(HS003_ALARM) 中で点滅動作の後で消去しています。

次に、アラーム状態を知らせるための LED 点滅を行うスクリプト(HS003_ALARM)を修正します。

ファイル名:HS003_ALARM.lua

キットに付属のメディア中の"スクリプトファイル¥HS003¥改良1"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS003_ALARM"
--[[
********************************************************************************
UIOUSB のポートビット#0 に 1(High),0(Low) を繰り返し出力する
********************************************************************************
]]
log_msg("start..",file_id)
local cnt,stat,handle,flag;
---------------------------------------
-- 50 回 LED 点滅を繰り返す
---------------------------------------
for cnt = 1,50,1 do
  ------------------------------------------------
  -- 100ms ごとに LED の点灯と消灯を繰り返す
  ------------------------------------------------
```

```
  if not uio_command("do0 1") then error() end;
  wait_time(100);
  if not uio_command("do0 0") then error() end;
  wait_time(100);
end


------------------------------------------------------
-- LED 点滅が終了したので、これ以降リードスイッチの検出で
-- このタスクが起動されても構わない
------------------------------------------------------
if not set_shared_data("LAMP_BLINKING","") then error() end


------------------------------------------------------
-- 現在監視中かどうかを判断して、このタスクを抜けるときに
-- LED を点灯または消灯のどちらの状態にするかを決める
------------------------------------------------------
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then error() end
if (flag ~= "") then
  if not uio_command("do0 1") then error() end;
else
  if not uio_command("do0 0") then error() end;
end;


log_msg("end.",file_id)
```

`if not set_shared_data("LAMP_BLINKING","") then error() end` の部分で、スクリプトを同時実行するのを防いでいるグローバル共有変数を削除しています。

動作確認をしてください。監視中にリードスイッチを連続して感知させても、アラーム中で LED が点滅中の場合にはスクリプトが重複して起動されないようになっているのが分かると思います。

5.7 改良(2)ランプ点滅中に監視を中止したら、すぐにランプを消灯する

次の改良点は、アラーム中で LEDが点滅しているときに、監視を中止するためにタクトスイッチを押しても、LED の点滅途中の場合にはタスクを中断できない部分を直します。

アラーム状態を知らせるための LED 点滅を行うスクリプト(HS003_ALARM)を修正します。

ファイル名:HS003_ALARM.lua

キットに付属のメディア中の”スクリプトファイル¥HS003¥改良2”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS003_ALARM"
--[[
*******************************************************************************
UIOUSB のポートビット#0 に 1(High),0(Low) を繰り返し出力する
*******************************************************************************
]]
log_msg("start..",file_id)
local cnt,stat,handle,flag;
---------------------------------------
-- 50 回 LED 点滅を繰り返す
---------------------------------------
for cnt = 1,50,1 do
  ------------------------------------------------
  -- LED 点滅前に監視が中止されているのが判明したら
  -- タスクを終了する
  ------------------------------------------------
  stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
  if not stat then error() end
  if (flag == "") then
    if not set_shared_data("LAMP_BLINKING","") then error() end
    do return end;
  end;

  ------------------------------------------------
  -- 100ms ごとに LED の点灯と消灯を繰り返す
  ------------------------------------------------
  if not uio_command("do0 1") then error() end;
  wait_time(100);
  if not uio_command("do0 0") then error() end;
  wait_time(100);
end

------------------------------------------------------
-- LED 点滅が終了したので、これ以降リードスイッチの検出で
-- このタスクが起動されても構わない
------------------------------------------------------
```

```
if not set_shared_data("LAMP_BLINKING","") then error() end


-------------------------------------------------------
-- 現在監視中かどうかを判断して、このタスクを抜けるときに
-- LED を点灯または消灯のどちらの状態にするかを決める
-- 監視フラグ(ALARM_ONLINE_FLAG) の値を読み込んでから
-- LED に出力するまでの間に監視フラグが変更されると
-- 監視状態と LED の関係が一致しなくなるので注意が必要
-- 排他制御が望ましい
-------------------------------------------------------
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then error() end
if (flag ~= "") then
  if not uio_command("do0 1") then error() end;
else
  if not uio_command("do0 0") then error() end;
end;


log_msg("end.",file_id)
```

`stat,flag = get_shared_data("ALARM_ONLINE_FLAG")` は、for ブロックの中で常に "ALARM_ONLINE_FLAG"グローバル共有変数を調べて、監視中止になっているかどうかを判断しています。

```
if (flag == "") then
if not set_shared_data("LAMP_BLINKING","") then error() end
  do return end;
end;
```

部分で、flag 変数が空文字列 "" の場合には UIOUSB イベントハンドラスクリプトで、監視中止の処理を行って、"ALARM_ONLINE_FLAG"グローバル共有変数が削除されたことになります。その場合には、LED を点滅中を示すグローバル共有変数 "LAMP_BLINKING" を消去した後に、このスクリプト実行を終了するために return 文を実行します。`do return end` となっているのは、Lua 言語の仕様で return 文がブロックの最後にしか記述できないため、スクリプト修正時にもエラーにならないようにこのように記述しています。

動作を確認してみます。今度は LED が点滅中でもタクトスイッチを押して監視を中止すると同時に、LED がすぐに消灯するが確認できると思います。

5.8 完成(3) 稀にしか発生しない不具合に対応する

最後の改良点は、ごく稀にしか発生しない不具合を修正します。

前述の HS003_ALARMスクリプト後半の下記の部分を見てください、

```
..
..
.. スクリプトの前の部分を省略 ..
-------------------------------------------------------
-- 現在監視中かどうかを判断して、このタスクを抜けるときに
-- LED を点灯または消灯のどちらの状態にするかを決める
-- 監視フラグ(ALARM_ONLINE_FLAG) の値を読み込んでから
-- LED に出力するまでの間に監視フラグが変更されると
-- 監視状態と LED の関係が一致しなくなるので注意が必要
-- 排他制御が望ましい
-------------------------------------------------------
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then error() end
-- この間にタクトスイッチが操作されると問題が発生する
if (flag ~= "") then
  if not uio_command("do0 1") then error() end;
else
  if not uio_command("do0 0") then error() end;
end;
```

`stat,flag = get_shared_data("ALARM_ONLINE_FLAG")` 文を実行した後で、`if (flag ~= "") then` 文で判断を行なうまでの間に、タクトスイッチが操作された場合に不具合が発生します。この２つの文の実行間隔は非常に短時間なので通常は無視しても構いませんが、もしこの間にスイッチが操作されると実際の監視状態と LED の点灯状態が一時的に一致しなくなります。たとえば、この部分に wait_time(5000) などの記述を入れてこの間にスイッチを操作することで簡単に不具合を再現できます。

このような不具合は発生頻度が極端に低いので、非常に見つけにくいので厄介です。以降で、これらの部分を修正していきます。

最初に、UIOUSB イベントハンドラスクリプトファイルを修正します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS003¥完成3”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"


local stat,val,flag,taskid,handle;
```

```
if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  ------------------------------------------------------------
  -- ビット#1 が変化していて、かつビット#1 の値が 0 の時を選択
  -- 監視開始/終了ボタンを押した時
  ------------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x02) > 0 ) and
    (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x02) == 0) then
    ------------------------------------------------------------------------
    -- ランプの点滅タスク終了時の LED の状態を ALARM_ONLINE_FLAG フラグ
    -- の値を読み込んで判断しているので、その間フラグは操作できないように
    -- 排他制御する
    ------------------------------------------------------------------------
    stat,handle = critical_section_enter("AlarmFlagKey",10000);
    if not stat then error() end

    stat,val = inc_shared_data("ALARM_ONLINE_FLAG")
    if not stat then
      if not critical_section_leave(handle) then error() end;
      error();
    end;

    if (tonumber(val) == 1) then
      -- ボタンを1回目に押した
      if not uio_command("do0 1") then
        if not critical_section_leave(handle) then error() end;
        error();
      end;
      log_msg("監視開始",file_id);
    else
      -- ボタンを2回目に押した
      if not set_shared_data("ALARM_ONLINE_FLAG","") then
        if not critical_section_leave(handle) then error() end;
        error();
      end;

      if not uio_command("do0 0") then
        if not critical_section_leave(handle) then error() end;
        error();
      end;
```

```
      log_msg("監視終了",file_id);
    end
    ------------------------------------------------------
    -- 排他制御を終了
    ------------------------------------------------------
    if not critical_section_leave(handle) then error() end;
  end;

  ------------------------------------
  -- ビット#2 が変化している時を選択
  -- リードスイッチが変化している時
  ------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x04) > 0 ) then
    ---------------------------------------
    -- 監視中の場合のみアラートを検出する
    ---------------------------------------
    stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
    if not stat then error() end
    if (flag ~= "") then
      log_msg("アラート検出",file_id);
      ------------------------------------------------------------
      -- ランプが既に点滅中の場合には重複してタスクを起動しない
      ------------------------------------------------------------
      local stat2,val2 = inc_shared_data("LAMP_BLINKING")
      if not stat2 then error() end
      if (tonumber(val2) == 1) then
        ------------------------------------------------
        -- 別スレッドでランプの点滅タスクを起動する
        ------------------------------------------------
        stat,taskid = script_fork_exec("HS003_ALARM","","")
        if not stat then error() end
      end;
    end;
  end;
end;
```

`stat,handle = critical_section_enter("AlarmFlagKey",10000);` は、排他制御をするために “AlarmFlagKey” という名前でクリティカルセクションを開始します。パラメータで指定する名前は別の名前を指定しても構いません。10000 は、クリティカルセクションに 10秒(10000ms)以内に入れなかった場合にエラーを検出するためのタイムアウ

ト時間です。この関数の実行が成功した場合には、これ以降 critical_section_leave() 関数をコールするまで、同じ名前のパラメータ“AlarmFlagKey” を指定してクリティカルセクション開始を待つスクリプトが、並行して実行されることはありません。

```
-----------------------------------------------------
-- 排他制御を終了
-----------------------------------------------------
if not critical_section_leave(handle) then error() end;
```

の部分で、排他制御を終了してクリティカルセクションを抜けます。

critical_section_enter()をコールしてから、critical_section_leave() を実行するまでの間に、何らかのエラー条件を検出してスクリプトの動作を中止する場合にも、必ずcritical_section_leave() を実行してから error() をコールしてください。現在のバージョンの Lua 言語には “try-finally” 文が用意されていませんので、ユーザー側でクリティカルセクション内から抜け出る全ての箇所に、critical_section_leave()を配置してください。

自動的にクリティカルセクションのミューテックスハンドルが開放されます

スクリプト中で critical_section_enter()をコールしてから、スクリプトを終了（エラーによって強制的に終了する場合を含む）するまでの間に critical_section_leave()をコールしなかった場合には、Windows のリソースを無駄に消費するのを防ぐために、DeviceServer は強制的にcritical_section_leave()をコールして、ログに警告メッセージを出力します。

次に、アラーム状態を知らせるための LED 点滅を行うスクリプト(HS003_ALARM)を修正します。

ファイル名:HS003_ALARM.lua

キットに付属のメディア中の”スクリプトファイル¥HS003¥完成3”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS003_ALARM"
--[[
*********************************************************************************
UIOUSB のポートビット#0 に 1(High),0(Low) を繰り返し出力する
*********************************************************************************
]]
log_msg("start..",file_id)
local cnt,stat,handle,flag;
-------------------------------------
-- 50 回 LED 点滅を繰り返す
-------------------------------------
```

```
for cnt = 1,50,1 do
  ------------------------------------------------
  -- LED 点滅前に監視が中止されているのが判明したら
  -- タスクを終了する
  ------------------------------------------------
  stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
  if not stat then error() end
  if (flag == "") then
    if not set_shared_data("LAMP_BLINKING","") then error() end
    do return end;
  end;
  ------------------------------------------------
  -- 100ms ごとに LED の点灯と消灯を繰り返す
  ------------------------------------------------
  if not uio_command("do0 1") then error() end;
  wait_time(100);
  if not uio_command("do0 0") then error() end;
  wait_time(100);
end

------------------------------------------------------------
-- LED 点滅が終了したので、これ以降リードスイッチの検出で
-- このタスクが起動されても構わない
------------------------------------------------------------
if not set_shared_data("LAMP_BLINKING","") then error() end

------------------------------------------------------------
-- 監視の開始・中止ボタンのイベント処理が同時に進行しないように
-- 排他制御を行う
------------------------------------------------------------
stat,handle = critical_section_enter("AlarmFlagKey",10000);
if not stat then error() end
------------------------------------------------------------
-- 現在監視中かどうかを判断して、このタスクを抜けるときに
-- LED を点灯または消灯のどちらの状態にするかを決める
-- 監視フラグ(ALARM_ONLINE_FLAG) の値を読み込んでから
-- LED に出力するまでの間に監視フラグが変更されると
-- 監視状態と LED の関係が一致しなくなるので注意が必要
-- 排他制御が望ましい
```

```
------------------------------------------------------
stat,flag = get_shared_data("ALARM_ONLINE_FLAG")
if not stat then
  if not critical_section_leave(handle) then error() end;
  error();
end

-- 不具合試験用
-- wait_time(5000);

if (flag ~= "") then
  if not uio_command("do0 1") then
    if not critical_section_leave(handle) then error() end;
    error();
  end;
else
  if not uio_command("do0 0") then
    if not critical_section_leave(handle) then error() end;
    error();
  end;
end;

------------------------------------------------------
-- 監視の開始・中止ボタンイベント処理の排他制御を終了
------------------------------------------------------
if not critical_section_leave(handle) then error() end;

log_msg("end.",file_id)
```

`stat,handle = critical_section_enter("AlarmFlagKey",10000);` の部分で、UIOUSB_EVENT_DATA イベントハンドラ中で記述したクリティカルセクションと同じ名前のクリティカルセクションを開始します。

```
------------------------------------------------------
-- 監視の開始・中止ボタンイベント処理の排他制御を終了
------------------------------------------------------
```

`if not critical_section_leave(handle) then error() end;`

の部分で、クリティカルセクションを終了しています。

これでLED 点滅終了時にタクトスイッチが操作されても、問題が発生しないようになりました。

`-- wait_time(5000);` 部分のコメントを外すと、ウェイト中にボタンが操作されても問題が発生しなくなっているのを簡単にテストできます。

6 [HS004] 温度と明るさを測定する

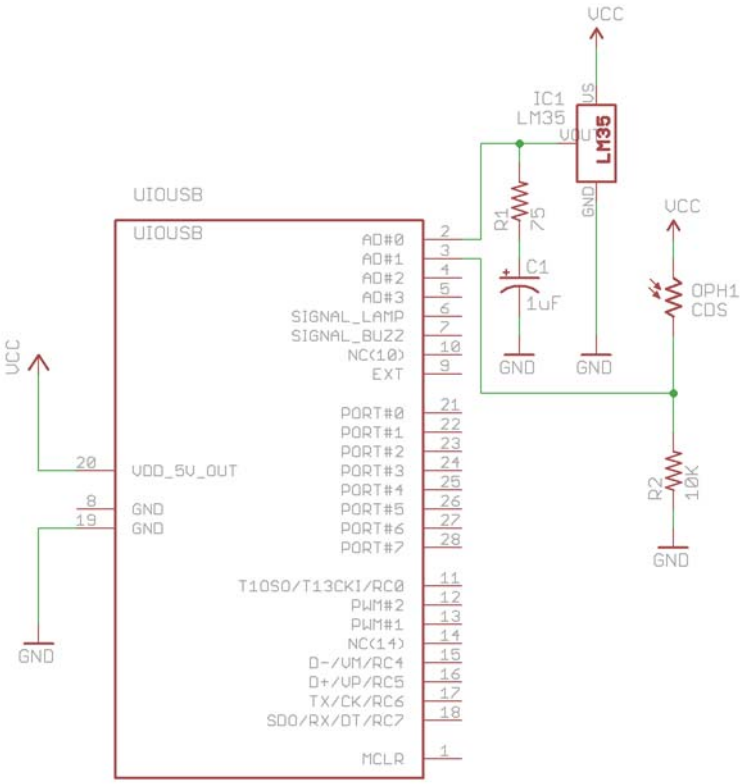
6.1 アプリケーション説明

UIOUSB の A/D 変換機能を使用して温度と明るさを測定します。

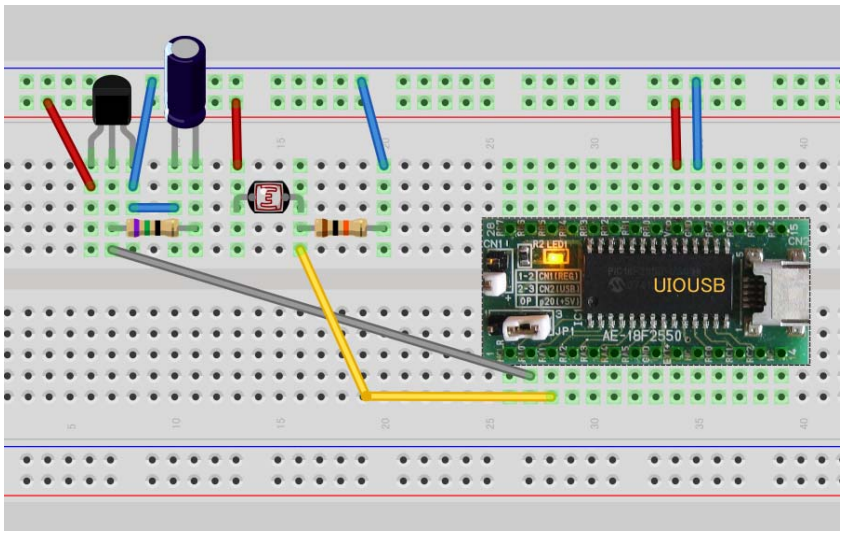
温度は、温度センサーIC LM35 を使用して摂氏温度を電圧値として取得します。明るさは相対的な明るさを簡単に検出できる CDS（フォトレジスタ）を使用して、電圧値として取得します。

測定は１分ごとに UIOUSB デバイスの自動サンプリング機能によってA/D 変換が行なわれて、測定結果がログに出力されます。

6.2 回路図



6.3 配線図



6.4 スクリプトファイル設定と説明

UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS004_CONF.lua

キットに付属のメディア中の”スクリプトファイル¥HS004”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS004_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: 温度と明るさを測定する
********************************************************************************
]]
log_msg("start..",file_id)
---------------------------------------
-- UIOUSB コンフィギュレーション設定
---------------------------------------
if not uio_command("dcfg 0xFF") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x00") then error() end
if not uio_command("sampling_rate 600") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`if not uio_command("sampling_rate 600") then error() end` は、現在の I/O ポートと A/D 変換入力値を自動的に繰り返し取得してサンプリングイベントを発生させるための設定です。”sampling_rate” UIOUSB コマンドで指定

している 600 は 60秒を意味します。”sampling_rate” コマンドの繰り返し間隔のパラメータの単位は x100ms 単位になっています。

UIOUSB のA/D 変換機能は常に有効になっていますので、UIOUSB のコンフィギュレーションで特に設定する項目はありません。

次に、”sampling_rate” で指定した間隔で発生するサンプリングイベントを処理するための UIOUSB イベントハンドラスクリプトファイルを作成します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS004”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
********************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値                          値                                          例
--------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                         2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA_1 は常にイベントプリフィックス文字列を表す          "$$$"
                    "$$$" 文字列
```

```
                    EVENT_DATA_2 はイベント名が入る                                    "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

if g_params["EVENT_DATA_2"] == "SAMPLING" then
  local temperature = tonumber(g_params["EVENT_DATA_5"]) / 1024 * 5.0 * 100;
  local cds_val = tonumber(g_params["EVENT_DATA_6"]);
  log_msg(string.format("温度 %3.1f℃  明るさ %d",temperature,cds_val),file_id);
end;
```

`if g_params["EVENT_DATA_2"] == "SAMPLING" then` は、このイベントハンドラが “sampling_rate” コマンドで設定した自動サンプリングイベントによって実行された場合のみ if 文の中を実行するようにしています。

UIOUSB イベントデータ(SAMPLING) の内容は下記の様になっています(UIOUSB ユーザーマニュアルより抜粋)

```
$$$,SAMPLING,[port_val],[counter_val],[ad0],[ad1],[ad2],[ad3]
```

[port_val] には 現在のポート値が16進数表記で設定されます。
[counter_val] 現在のカウンタ値が10進数で入ります。
[adc0].. [ad3] には、A/D 入力変換値が 10 進数表記で設定されます。

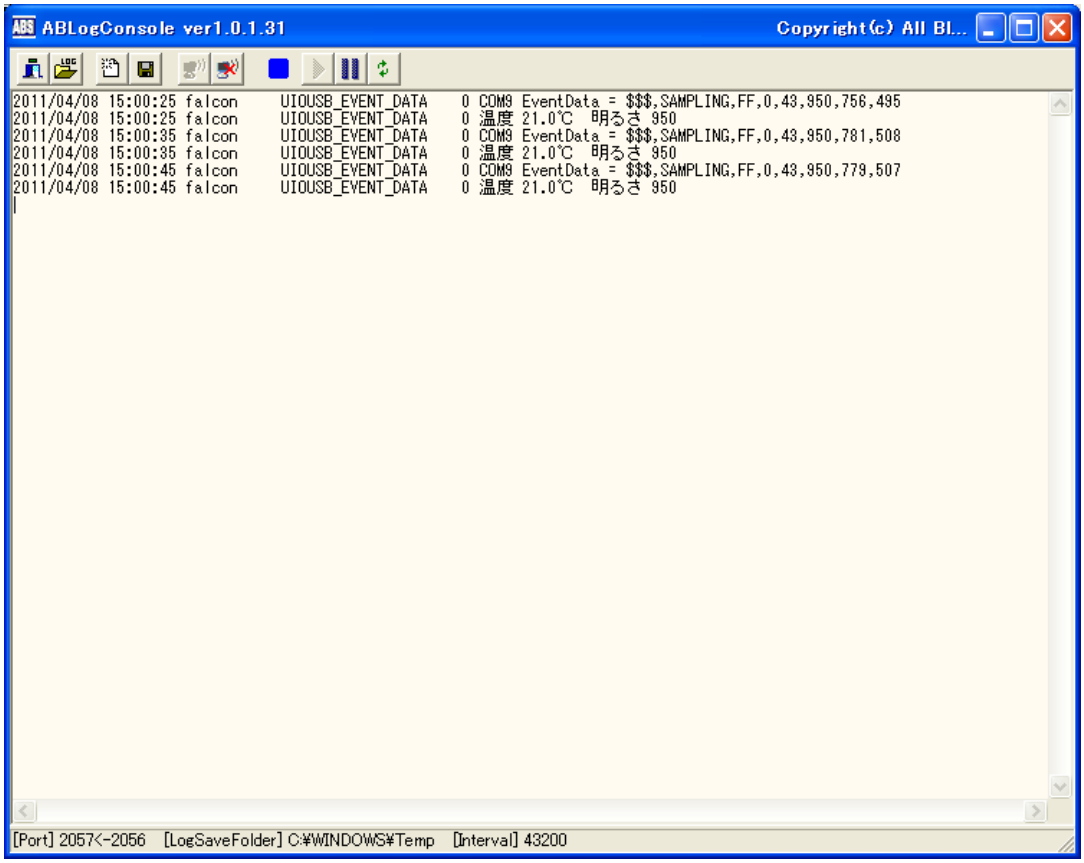
今回の回路で、LM35 温度センサーが接続されている A/D#0 の値は、カンマ区切りで出力されたイベントデータの５項目目なので `g_params["EVENT_DATA_5"]`に 温度センサーデータが入っています。
また同様に CDS の値は A/D#1 に接続されているので、`g_params["EVENT_DATA_6"]`にセンサーデータが入っています。

`local temperature = tonumber(g_params["EVENT_DATA_5"]) / 1024 * 5.0 * 100;` は、温度センサーのA/D 変換値を摂氏温度に変換しています。A/D 変換の精度は 10bit(0 から 1023までの 1024ステップ) なので1024 で割り算をした後、A/D 変換時のリファレンス電圧 Vcc(5V) を掛けてLM35センサーの電圧を得ます。LM35センサーの電圧と摂氏温度の関係は、１度℃ あたりセンサー電圧10mV なので、100 倍することで摂氏温度が求められます。

`tonumber()` は、文字列を数値に変換するための Lua 関数です。

6.5 動作確認

計測結果はログに出力されますので、ログコンソールプログラムをプログラムメニューから起動します。
下記の用に定期的に計測されてメッセージが表示されると思います。



（上記のログは、サンプリングレートを10秒ごとに変更したときのものです “sampling_rate 100”）

6.6 改良(1)A/Dリファレンス電圧を測定する

UIOUSBV のA/D 変換時に使用しているリファレンス用の電圧は UIOUSB の電源Vcc を共有しています。そのため、UIOUSB の電源は USB バスから取得していますので、USB の状態によって標準の 5V から電圧が多少変化しています。このため、リファレンス電圧が 5V の前提で計算した温度が、正確な温度を示していない場合があります。

UIOUSB はリファレンス電圧を外部から指定する機能はありませんので、常に UIOUSB の電源をA/D 変換のリファレンス電圧に使用します。このため正確な A/D 変換の電圧値を得るためには、あからじめUIOUSB 電源(Vcc) を測定しておきます。デジタルマルチメータが必要ですが、便利ですので安価なもので構いませんので入手されることをお勧めします。デジタルマルチメータで UIOUSB の GND と Vcc 間(ピン19と ピン20 間)を測定してください。5.0V 前後の値になりますので、この値をスクリプトに記述して前述の温度計算に利用します。

サンプリングイベントを処理するための UIOUSB イベントハンドラスクリプトファイルを修正します。

ファイル名: UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS004¥改良1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"
```

```
--[[
*******************************************************************************
* イベントハンドラスクリプト実行時間について                                  *
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値                          値                                          例
-------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名        "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                        2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA__1 は常にイベントプリフィックス文字列を表す          "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                              "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

-------------------------------------------------------------
-- UIOUSB の VDD を実際にマルチメータで測定した値を代入する
-------------------------------------------------------------
local vref = 4.95;
if g_params["EVENT_DATA_2"] == "SAMPLING" then
  local temperature = tonumber(g_params["EVENT_DATA_5"]) / 1024 * vref * 100;
  local cds_val = tonumber(g_params["EVENT_DATA_6"]);
  log_msg(string.format("温度 %3.1f℃  明るさ %d",temperature,cds_val),file_id);
```

```
end;
```

`local vref = 4.95;` は、デジタルマルチメータで測定した電圧値（単位 V）が入っています。

`local temperature = tonumber(g_params["EVENT_DATA_5"]) / 1024 * vref * 100;` 温度計算をするときに、`verf` 変数に代入したリファレンス電圧を使用して計算します。

7 [HS005] １日の温度と明るさの変化を EXCEL でグラフにする

7.1 アプリケーション説明

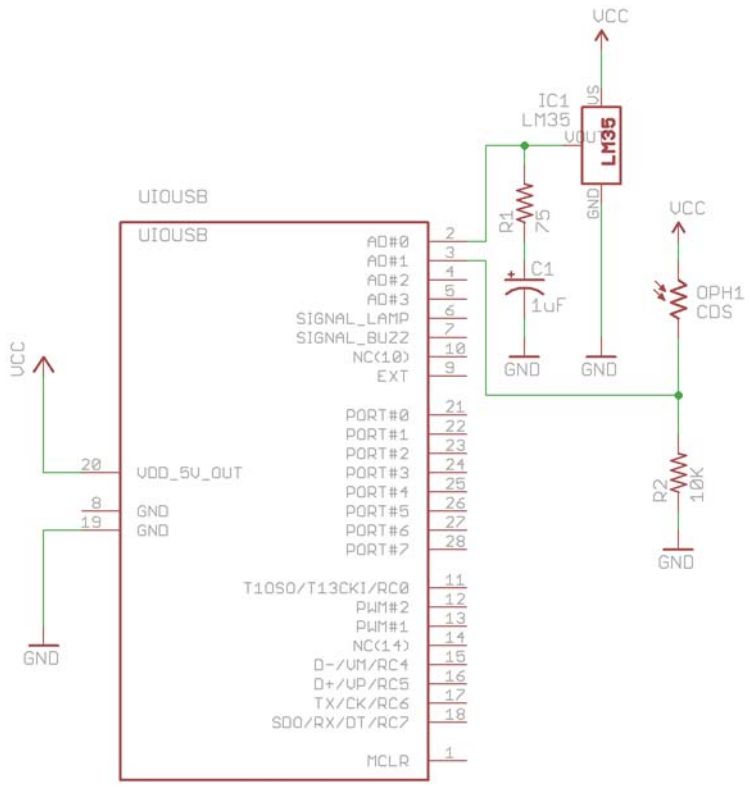
UIOUSB の A/D 変換機能を使用して温度と明るさを定期的に測定してEXCEL 上でグラフを作成します。

温度は、温度センサーIC LM35 を使用して摂氏温度を電圧値で取得します。明るさは相対的な明るさを検出できる CDS（フォトレジスタ）を使用します。測定は１分ごとに UIOUSB デバイスの自動サンプリングによって A/D 変換を行って、サーバーのデータベースに一時的に保管します。

集計を行うときには、集計用スクリプトを実行してサーバーのデータベースを使用して１時間ごとの集計を行って、集計結果を CSV ファイルに出力します。集計結果のCSV ファイルを後で EXCEL で読み込んでグラフを表示します。

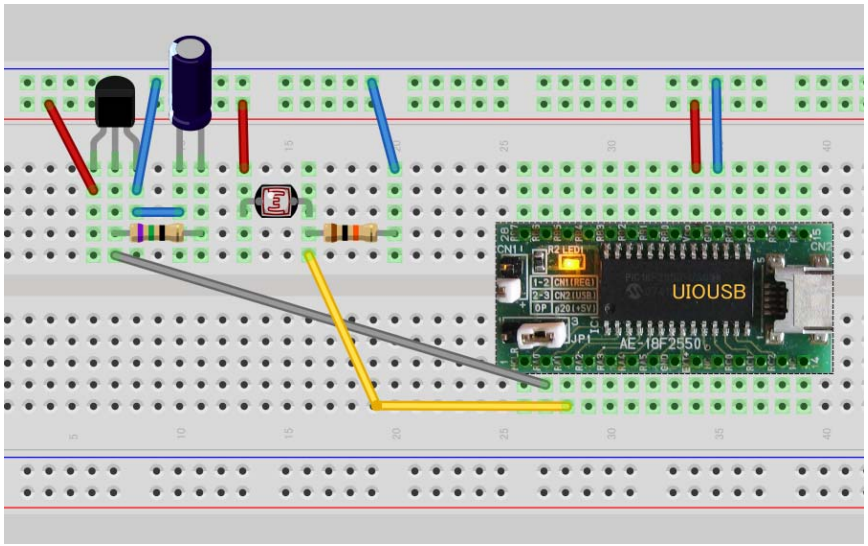
7.2 回路図

(HS004 と同一の回路です)



7.3 配線図

(HS004 と同一の配線です)



7.4 スクリプトファイル設定と説明

UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS005_CONF.lua

キットに付属のメディア中の"スクリプトファイル¥HS005"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS005_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: 温度と明るさを測定する
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFF") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x00") then error() end
if not uio_command("sampling_rate 600") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

UIOUSB デバイスで１分ごとの自動サンプリングによって発生したイベントを処理するための、イベントハンドラスクリプトを作成します。

ファイル名:UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS005”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
********************************************************************************
* イベントハンドラスクリプト実行時間について                                   *
********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
--------------------------------------------------------------------------------
キー値                          値                                              例
--------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名    "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                     2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA__1 は常にイベントプリフィックス文字列を表す      "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                             "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
```

```
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

if g_params["EVENT_DATA_2"] == "SAMPLING" then
  local key_list = list_to_csv("TemperatureLevel","CDSLevel")
  local val_list = list_to_csv(g_params["EVENT_DATA_5"],g_params["EVENT_DATA_6"])
  if not script_fork_exec("HS005_STORE",key_list,val_list) then error() end;
end;
```

`local key_list = list_to_csv("TemperatureLevel","CDSLevel")` は、HS005_STORE スクリプトをコールする時にパラメータで指定するキー名リストを作成しています。パラメータは、キー名リストとそれに対応する値リストをカンマ区切り形式の文字列で script_fork_exec() 関数に渡します。`list_to_csv()` はパラメータで指定された文字列をカンマ区切り表現で連結するための関数です。

`local val_list = list_to_csv(g_params["EVENT_DATA_5"],g_params["EVENT_DATA_6"])` は、パラメータキー名リストに対応する値リストを作成しています。

スクリプトパラメータの指定について

`script_fork_exec()` ライブラリ関数でスクリプトにパラメータを渡す場合には、第２パラメータにキー名リスト、第３パラメータに値リストを CSV 形式で指定します。例えば キーと値がそれぞれ “Key1”、“値1” の１ペアのパラメータをスクリプト“ABC” を実行するときに渡す場合には、

`script_fork_exec(“ABC”,”Key1”,”値1”)` になります。

キーと値がそれぞれ “Key1”、“値1”と“Key2”、“値2”の２ペアをパラメータで渡す場合には、

`script_fork_exec(“ABC”,”Key1,Key2”,”値1,値2”)` の様になります。

“ABC” スクリプト中で、渡されたパラメータを取得する場合には、`g_params[]` 配列にキー名を指定してアクセスします。例えば `g_params[“Key1”]` には “値1” が入っていて、`g_params[“Key2”]` には “値2” が格納されています。

スクリプト実行用ライブラリ関数の詳細については “DeviceServerユーザーマニュアル”を参照してください。

`if not script_fork_exec("HS005_STORE",key_list,val_list) then error() end;` でHS005_STORE スクリプトを別スレッドでコールしています。この時 “温度” と “明るさ” のデータをパラメータに渡しています。別スレッドで実行するのは、データベース格納など比較的時間がかかる処理を行う時にイベントハンドラスクリプトの終了が遅くなるのを防ぐためです。今回のような短い処理ではイベントハンドラ中でデータベースへの登録を行っても問題ありませんが、機能を拡張してスクリプト中に多くの処理を追加したときにも問題が発生しないようにこのような形で構築しています。

UIOUSB イベントハンドラスクリプトからコールされて、データベースに A/D 値を保存するためのスクリプトを作成します。

ファイル名: HS005_STORE.lua

キットに付属のメディア中の"スクリプトファイル¥HS005"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS005_STORE"

--[[
HS005_STORE スクリプト起動時に渡されるパラメータ
-------------------------------------------------------------------------------
キー値              値                                  値の例
-------------------------------------------------------------------------------
TemperatureLevel    温度センサーの値を A/D 変換した値   35
CDSLevel            フォトレジスタで検出した A/D 変換値 621
-------------------------------------------------------------------------------
温度センサーAD値    統計用データベースキー名            "HS005_TEMP"
フォトレジスタAD値  統計用データベースキー名            "HS005_CDS"

統計用データベースに登録する値
          フォトレジスタAD値          数値に変換した値
          温度センサーAD値            摂氏温度
]]

local temperature,cds_val;
local adc_vref = 4.95;
-------------------------------------------------------------
-- 温度計算
-------------------------------------------------------------
if g_params["CDSLevel"] and g_params["TemperatureLevel"] then
  temperature = (100 * adc_vref * tonumber(g_params["TemperatureLevel"])) / 1024;
  cds_val = tonumber(g_params["CDSLevel"]);
else
  log_msg("*ERROR* parameter error",file_id);
  error();
end;

-------------------------------------------------------------
-- データベースに登録
-------------------------------------------------------------
```

```
log_msg(string.format("温度 %3.1f℃ 明るさ %d",temperature,cds_val),file_id);
if not add_stat_data("HS005_CDS",cds_val) then error() end;
if not add_stat_data("HS005_TEMP",temperature) then error() end;
```

`if g_params["CDSLevel"] and g_params["TemperatureLevel"] then` は、このスクリプトをコールするときに“CDSLevel”と“TemperatureLevel”の２つのパラメータが指定されているかどうかをチェックしています。

`temperature = (100 * adc_vref * tonumber(g_params["TemperatureLevel"])) / 1024;` は、パラメータで渡された温度センサーの A/D 変換値を摂氏温度に変換しています。

`cds_val = tonumber(g_params["CDSLevel"]);` は、パラメータで渡されたCDS の A/D 変換値を数値に変換しています。

`if not add_stat_data("HS005_CDS",cds_val) then error() end;` は、cds_val で表された数値をデータベース（統計用）にキー名“HS005_CDS”で登録しています。登録時の日付時刻情報もデータベースに格納されますので、後から集計時に利用します。`if not add_stat_data("HS005_TEMP",temperature) then error() end;` も同様に統計用データベースにキー名“HS005_TEMP”で温度データを格納しています。

集計用データベースについて

集計用データベースは `add_stat_data()` で統計データを登録して、後から`summary_stat_data()` でデータを集計するように設計されています。各々のライブラリ関数ではキー名を指定することで集計時に使用する統計データを区別します。集計時には統計データを登録した時のタイムスタンプを元に、日付時刻の範囲や集計間隔時間などを指定できます。

集計用ライブラリ関数の詳細については“DeviceServerユーザーマニュアル”を参照してください。

１時間単位の集計を行って、CSV ファイルを出力するスクリプトを作成します。

ファイル名: HS005_SUMMARY.lua

キットに付属のメディア中の”スクリプトファイル¥HS005”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS005_SUMMARY"
--[[
集計計算を行って、CSV ファイルに書き出す
HS005_SUMMARY スクリプト起動時に渡されるパラメータ
------------------------------------------------------------------------
キー値                          値                          値の例
```

```
--------------------------------------------------------------------------------
TargetDate                      集計対象日付                    "2011/01/01"
                                パラメータ省略時はスクリプトを起動した当日になる
--------------------------------------------------------------------------------
]]
log_msg("start..",file_id)
------------------------------
-- 集計対象日付を取得する
------------------------------
local target_datetime;
if g_params["TargetDate"] then
  target_datetime = g_params["TargetDate"] .. " 0:0:0";
else
  local now = os.date "*t";
  target_datetime = string.format("%4.4d/%2.2d/%2.2d 0:0:0",now["year"],now["month"],now["day"]);
end;

------------------------------------------------------------
-- 集計結果を格納するファイル名
-- "c:/" は Windows の C: ドライブのトップディレクトリを示す
------------------------------------------------------------
local filename = "c:/summary.csv";

--------------------------------------------------------------------------
-- 集計パラメータ
-- interval: 指定した秒間隔で集計を行う。
-- count:    指定した回数、interval で指定した集計を実行する
--------------------------------------------------------------------------

----------------------------------
-- 一日の１時間ごとの集計を指定する
----------------------------------
local interval = 3600;
local count = 24;

------------------------------
-- 明るさのデータを集計する
------------------------------
local cds_stat,cds_datetime,cds_sample,cds_total,cds_mean,cds_max,cds_min =
```

```
  summary_stat_data("HS005_CDS",target_datetime,interval,count);
if not cds_stat then error() end;


-------------------------------
-- 温度データを集計する
-------------------------------
local temp_stat,temp_datetime,temp_sample,temp_total,temp_mean,temp_max,temp_min =
  summary_stat_data("HS005_TEMP",target_datetime,interval,count);
if not temp_stat then error() end;


-------------------------------
-- ファイルをオープンする
-------------------------------
local file = io.open(filename,"w+");
if (file == nil) then error() end;


-----------------------------------------
-- CSVファイルのヘッダレコードを書き出す
-----------------------------------------
file:write("¥"DateTime¥",¥"Temperature¥",¥"Count¥",¥"CDS¥",¥"Count¥"¥n");
local key,val
for key,val in ipairs(cds_datetime) do
  -----------------------------------------
  -- 集計結果をログに出力する
  -----------------------------------------
  log_msg(string.format("日付時間 = %s¥t平均温度(データ数) = %g(%d)¥t平均明るさ(データ数) = %g(%d)",
    val,temp_mean[key],temp_sample[key],cds_mean[key],cds_sample[key]),file_id)
  -----------------------------------------
  -- 集計結果をファイルに出力する
  -----------------------------------------
  file:write(string.format("¥"%s¥",¥"%g¥",¥"%d¥",¥"%g¥",¥"%d¥"¥n"
    ,val,temp_mean[key],temp_sample[key],cds_mean[key],cds_sample[key]));
end


file:close();
```

`target_datetime = g_params["TargetDate"] .. " 0:0:0";` は、このスクリプトを実行する時に “TargetDate” スクリプトパラメータで集計対象日付が指定されていたときに実行されます。この日付と一日の始めの時刻”0:0:0“ から

集計用ライブラリ関数 summary_stat_data() の集計開始日時データパラメータを作成しています。

```
local now = os.date "*t";
target_datetime = string.format("%4.4d/%2.2d/%2.2d 0:0:0",now["year"],now["month"],now["day"]);
```

は、このスクリプトを実行する時に “TargetDate” スクリプトパラメータが指定されていなかったときに、スクリプトを実行した日を集計対象日時にしています。local now = os.date "*t"; はLua の標準ライブラリ関数で現在の日時を now 変数(テーブル値) に取得しています。now["year"] はそのnow 変数(テーブル)から年を取り出しています。

local filename = "c:/summary.csv"; は、集計結果を出力する CSV ファイル名です。フォルダの区切りがWindows でフォルダの区切りを表す “¥” ではなくて “/” になりますので注意してください。この場合だと C: ドライブのトップディレクトリにある summary.csv ファイルを表します。

```
local cds_stat,cds_datetime,cds_sample,cds_total,cds_mean,cds_max,cds_min =
summary_stat_data("HS005_CDS",target_datetime,interval,count);
```

は、キー名”HS005_CDS” で統計用データベースに格納された ”明るさ” の統計データ値を、指定した日時から、指定した繰り返し間隔と繰り返し開始回数分、集計する様にライブラリ関数をコールしています。ここでは一日分のデータを１時間ごとに集計しますので、指定した日付の “0:0:0” から 1時間間隔(3600秒) で24回(24時間) 集計するようにパラメータを指定しています。集計結果は集計時間ごとに、開始日時、データ個数、合計値、平均値などが計算されてテーブル値に格納されます。

```
local temp_stat,temp_datetime,temp_sample,temp_total,temp_mean,temp_max,temp_min =
summary_stat_data("HS005_TEMP",target_datetime,interval,count);
```

は同様に温度データの集計を行っています。

local file = io.open(filename,"w+"); は、Lua の io ライブラリを使用してファイルに出力しています。”w+” は既存のファイルを上書きモードでオープンする指定です。

for key,val in ipairs(cds_datetime) do は、集計結果で得られたデータの個数分繰り返す指定です。for 文でブロック内をループする時に、key 変数には 1 から 24 までの数値が順番に指定されます。

file:write("¥"DateTime¥",¥"Temperature¥",¥"Count¥",¥"CDS¥",¥"Count¥"¥n"); は、CSV ファイルの１レコード目にタイトルを書き出しています。

```
file:write(string.format("¥"%s¥",¥"%g¥",¥"%d¥",¥"%g¥",¥"%d¥"¥n"
,val,temp_mean[key],temp_sample[key],cds_mean[key],cds_sample[key]));
```

で、集計結果を CSV レコードに書きだしています。

file:close(); は CSV ファイルをクローズします。

サーバーで２時間ごとに定期的に実行するための記述を、PERIODIC_TIMER スクリプトに作成します。

ファイル名: PERIODIC_TIMER.lua

キットに付属のメディア中の”スクリプトファイル¥HS005”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "PERIODIC_TIMER"

--[[
********************************************************************************
＊イベントハンドラスクリプト実行時間について＊
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つアラームデバイスで、
タイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
********************************************************************************
]]

------------------------------------------------------------
-- 2時間に一回 HS005_PURGE スクリプトを実行する
------------------------------------------------------------
stat,val = inc_shared_data("TIME_2H")
if not stat then error() end
if (tonumber(val) == 1) then
  if not script_fork_exec("HS005_PURGE","","") then error() end
end

if (tonumber(val) > 120) then
  stat = set_shared_data("TIME_2H","")
  if not stat then error() end
end
```

PERIODIC_TIMER スクリプトは DeviceServer で自動的に１分に１回コールされています。

`stat,val = inc_shared_data("TIME_2H")` は、２時間(120分)をカウントするための `"TIME_2H"` グローバル共有変数をインクリメントしています。

```
if (tonumber(val) == 1) then
if not script_fork_exec("HS005_PURGE","","") then error() end
end
```

部分は、カウンタ値が “1” になったときに “HS005_PURGE” スクリプトを実行しています。

```
if (tonumber(val) > 120) then
stat = set_shared_data("TIME_2H","")
if not stat then error() end
end
```

部分は、２時間ごとにカウンタをリセットしています。

PERIODIC_TIMER スクリプトから２時間ごとにコールされて、古いサンプリングデータをデータベースから自動的に削除するためのスクリプトを作成します。

ファイル名: HS005_PURGE.lua

キットに付属のメディア中の”スクリプトファイル¥HS005”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS005_PURGE"

--------------------------------
-- 30 日以前のデータを削除する
--------------------------------
log_msg("start..",file_id)

local now = os.date "*t"
local stat,yyyy,mm,dd = inc_day(-30,now["year"],now["month"],now["day"])
local timestamp = string.format("%4.4d/%2.2d/%2.2d 23:59:59",yyyy,mm,dd)
if not clear_stat_data("HS005_",timestamp,"") then error() end
```

`local now = os.date "*t"` で、このスクリプトを実行した日時を取得しています。

`local stat,yyyy,mm,dd = inc_day(-30,now["year"],now["month"],now["day"])` は現在日から 30 日前の日付を計算するために、`inc_day()`ライブラリ関数を実行しています。

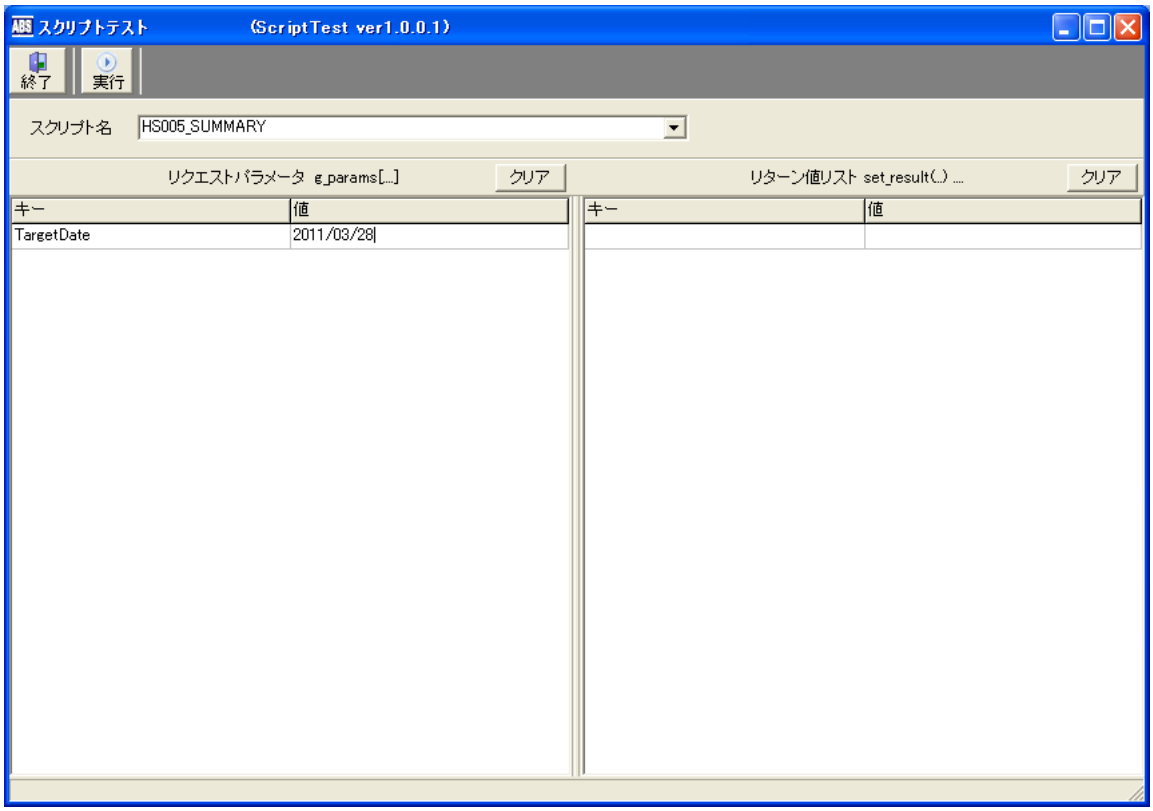
`if not clear_stat_data("HS005_",timestamp,"") then error() end` は統計データベースに格納されたデータの中で、キー名が “HS005_” で始まってかつ30日以前の古いデータを全て削除しています。これによって集計対象としな

い過去の 温度と明るさの統計データを削除しています。

7.5 動作確認

スクリプトの設定が終了したら、自動的に１分ごとにサーバーのデータベースに測定値が記録されていきます。

任意のタイミングで “HS005_SUMMARY” スクリプトを手動で実行してみてください。スクリプトパラメータに “TargetDate” のキー名で “YYYY/MM/DD” 形式で日付を与えると、指定した日付の集計が実行されて CSV ファイルが作成されます。



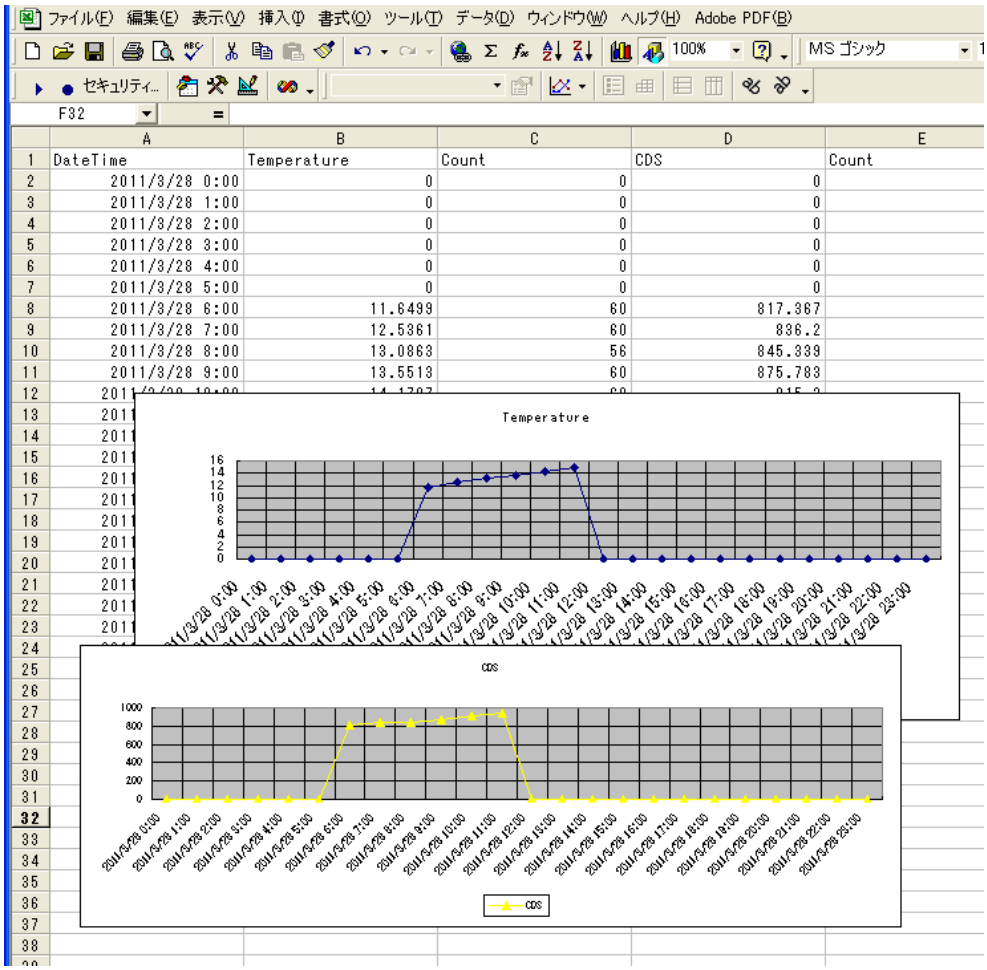
“HS005_SUMMARY” スクリプトをクライアントプログラムから実行した例です。

集計結果は、C: ドライブの中に summary.csv ファイルに作成されています。

```
"DateTime","Temperature","Count","CDS","Count"
"2011/03/28 00:00:00","0","0","0","0"
"2011/03/28 01:00:00","0","0","0","0"
"2011/03/28 02:00:00","0","0","0","0"
"2011/03/28 03:00:00","0","0","0","0"
"2011/03/28 04:00:00","0","0","0","0"
"2011/03/28 05:00:00","0","0","0","0"
"2011/03/28 06:00:00","11.6499","60","817.367","60"
"2011/03/28 07:00:00","12.5361","60","836.2","60"
"2011/03/28 08:00:00","13.0863","56","845.339","56"
```

```
"2011/03/28 09:00:00","13.5513","60","875.783","60"
"2011/03/28 10:00:00","14.1797","60","915.3","60"
"2011/03/28 11:00:00","14.8975","33","948.667","33"
"2011/03/28 12:00:00","0","0","0","0"
"2011/03/28 13:00:00","0","0","0","0"
"2011/03/28 14:00:00","0","0","0","0"
"2011/03/28 15:00:00","0","0","0","0"
"2011/03/28 16:00:00","0","0","0","0"
"2011/03/28 17:00:00","0","0","0","0"
"2011/03/28 18:00:00","0","0","0","0"
"2011/03/28 19:00:00","0","0","0","0"
"2011/03/28 20:00:00","0","0","0","0"
"2011/03/28 21:00:00","0","0","0","0"
"2011/03/28 22:00:00","0","0","0","0"
"2011/03/28 23:00:00","0","0","0","0"
```

このファイルを EXCEL で読み込んでグラフを作成します。



7.6 応用(1) 毎時 0 分の正確な時間に測定する

HS005 で作成したアプリケーションを、決められた時刻に定期的にデータを収集するアプリケーションに変更します。前回の例では集計間隔(１時間)に対して、１分単位の細かいサンプリングデータの平均(約60データ分)を求めていましたが、今回は決められた時間ごとに正確にサンプリングを行います。

UIOUSB のサンプリング間隔は UIOUSBデバイス内蔵のクロックに依存していて、長期間の繰り返し間隔はあまり正確にはなりません。またサンプリング時刻も PC 上の時計とは一致できないので、定刻のxx時xx分にサンプリングが必要な今回の様な場合には、PC 側でスケジュールしたタイミングで、A/D サンプリングを実行することで実現します。

正確な時刻と繰り返し間隔でサンプリングを行うために、Windows のタスクスケジューラ機能を使用します。タスクスケジューラから定期的に、コマンドプロンプトで動作可能な ScirptExecCmd.exe プログラム(サーバーソフトに同梱されています)を実行して、DeviceServer に設置したスクリプト実行します。そのスクリプト中で、A/D 変換を行ってデータベースに格納するようにします。

UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS0051_CONF.lua

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS005_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: 温度と明るさを測定する
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFF") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x00") then error() end
if not uio_command("sampling_rate 0") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`if not uio_command("sampling_rate 0") then error() end` でサンプリング間隔を0 に設定して UIOUSB 側での自

動サンプリングを停止しています。

定期的にScriptExecCmd.exe プログラム から起動されるスクリプトを作成します。

スクリプトからA/D 変換を行ってデータベースに値を保存します。

ファイル名: HS0051_ACQUISITION.lua

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS0051_ACQUISITION"
--[[
   温度センサーAD値              統計用データベースキー名                  "HS0051_TEMP"
   フォトレジスタAD値            統計用データベースキー名                  "HS0051_CDS"
   統計用データベースに登録する値
          フォトレジスタAD値              数値に変換した値
          温度センサーAD値                摂氏温度
]]
local temperature,cds_val;
local adc_vref = 4.95;
-----------------------------
-- A/D 変換値読み込み
-----------------------------
local stat,ad = uio_ad()
if not stat then error() end


-------------------------------------------------------------
-- 温度計算
-------------------------------------------------------------
temperature = (100 * adc_vref * ad[1]) / 1024;
cds_val = ad[2];


-------------------------------------------------------------
-- データベースに登録
-------------------------------------------------------------
log_msg(string.format("温度 %3.1f℃ 明るさ %d",temperature,cds_val),file_id);
if not add_stat_data("HS0051_CDS",cds_val) then error() end;
if not add_stat_data("HS0051_TEMP",temperature) then error() end;
```

`local stat,ad = uio_ad()` は、温度センサーと CDS が接続されたチャンネルを含む、全ての A/D チャンネルの変換を行っています。

次に、１時間単位の集計を行って、CSV ファイルを出力するスクリプトを作成します。ただし今回は集計間隔には１個のサンプリングデータしか含まれませんので平均値は個々のサンプリングデータと同一値になります。

ファイル名: HS0051_SUMMARY.lua

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS0051_SUMMARY"
--[[
集計計算を行って、CSV ファイルに書き出す

HS0051_SUMMARY スクリプト起動時に渡されるパラメータ
--------------------------------------------------------------------------------
キー値              値                                  値の例
--------------------------------------------------------------------------------
TargetDate          集計対象日付                        "2011/01/01"
                    パラメータ省略時はスクリプトを起動した当日になる
--------------------------------------------------------------------------------
]]
log_msg("start..",file_id)
------------------------------
-- 集計対象日付を取得する
------------------------------
local target_datetime;
if g_params["TargetDate"] then
  target_datetime = g_params["TargetDate"] .. " 0:0:0";
else
  local now = os.date "*t";
  target_datetime = string.format("%4.4d/%2.2d/%2.2d 0:0:0",now["year"],now["month"],now["day"]);
end;

------------------------------------------------------------
-- 集計結果を格納するファイル名
-- "c:/" は Windows の C: ドライブのトップディレクトリを示す
------------------------------------------------------------
local filename = "c:/summary.csv";
```

```
--------------------------------------------------------------------------
-- 集計パラメータ
-- interval: 指定した秒間隔で集計を行う。
-- count:    指定した回数、interval で指定した集計を実行する
--------------------------------------------------------------------------

-----------------------------------
-- 一日の１時間ごとの集計を指定する
-----------------------------------
local interval = 3600;
local count = 24;

-------------------------------
-- 明るさのデータを集計する
-------------------------------
local cds_stat,cds_datetime,cds_sample,cds_total,cds_mean,cds_max,cds_min =
  summary_stat_data("HS0051_CDS",target_datetime,interval,count);
if not cds_stat then error() end;

-------------------------------
-- 温度データを集計する
-------------------------------
local temp_stat,temp_datetime,temp_sample,temp_total,temp_mean,temp_max,temp_min =
  summary_stat_data("HS0051_TEMP",target_datetime,interval,count);
if not temp_stat then error() end;

-------------------------------
-- ファイルをオープンする
-------------------------------
local file = io.open(filename,"w+");
if (file == nil) then error() end;

-----------------------------------------
-- CSVファイルのヘッダレコードを書き出す
-----------------------------------------
file:write("¥"DateTime¥",¥"Temperature¥",¥"Count¥",¥"CDS¥",¥"Count¥"¥n");

local key,val
```

```
for key,val in ipairs(cds_datetime) do
  ----------------------------------------
  -- 集計結果をログに出力する
  ----------------------------------------
  log_msg(string.format("日付時間 = %s¥t平均温度(データ数) = %g(%d)¥t平均明るさ(データ数) = %g(%d)",
    val,temp_mean[key],temp_sample[key],cds_mean[key],cds_sample[key]),file_id)
  ----------------------------------------
  -- 集計結果をファイルに出力する
  ----------------------------------------
  file:write(string.format("¥"%s¥",¥"%g¥",¥"%d¥",¥"%g¥",¥"%d¥"¥n"
    ,val,temp_mean[key],temp_sample[key],cds_mean[key],cds_sample[key]));
end

file:close();
```

サーバーで２時間ごとに定期的に実行するための記述を、PERIODIC_TIMER スクリプトに作成します。

ファイル名: PERIODIC_TIMER.lua

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "PERIODIC_TIMER"


------------------------------------------------------------
-- 2時間に一回 HS0051_PURGE スクリプトを実行する
------------------------------------------------------------
stat,val = inc_shared_data("TIME_2H")
if not stat then error() end
if (tonumber(val) == 1) then
  if not script_fork_exec("HS0051_PURGE","","") then error() end
end

if (tonumber(val) > 120) then
  stat = set_shared_data("TIME_2H","")
  if not stat then error() end
end
```

PERIODIC_TIMER スクリプトから２時間ごとにコールされて、古いサンプリングデータをデータベースから自動的に

削除するためのスクリプトを作成します。

ファイル名: HS0051_PURGE.lua

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS0051_PURGE"
--------------------------------
-- 30 日以前のデータを削除する
--------------------------------
log_msg("start..",file_id)

local now = os.date "*t"
local stat,yyyy,mm,dd = inc_day(-30,now["year"],now["month"],now["day"])
local timestamp = string.format("%4.4d/%2.2d/%2.2d 23:59:59",yyyy,mm,dd)
if not clear_stat_data("HS0051_",timestamp,"") then error() end
```

7.6.1 Windows タスクスケジューラについて

定期的に決められた時刻でスクリプトを実行するために、WindowsXP もしくは Windows2003 Server のタスクスケジューラとスケジュール用コマンド“schtasks” を使用します。また、コマンドプロンプトからスクリプトを実行するためのプログラムは、DeviceServer インストール時に保管されている ScriptExecCmd.exe を使用します。

HS0051_ACQUISITION スクリプトを実行する、タスクスケジューラのジョブ登録は、DeviceServer の動作している PC で行います。

7.6.2 ScriptExecCmd.exe プログラムの設定

”C:¥Program Files¥AllBlueSystem¥Demo¥ScriptExecCmd.exe プログラムを "C:¥Tools¥Bin" にコピーしてください。また、"C:¥Tools¥Bin"フォルダに ScriptExecCmd.ini ファイルを作成して、スクリプト名とユーザー名、パスワードをあからじめ指定しておいてください。

”C:¥Tools¥Bin” フォルダは、フォルダ名に空白文字や日本語を含まないようにするためのフォルダ名になっています。また、ScriptCmdExec.exeプログラムに渡すパラメータのスクリプト名やログインユーザー名などの情報を、プログラムと同一フォルダにある ScriptExecCmd.ini 設定ファイルで指定します。他のScriptExecCmd.exe を使用するプログラムの動作に影響しないように、今回は別フォルダにプログラムファイルごとコピーしています。このフォルダ以外の場所に ScriptExecCmd.exe と ScriptExecCmd.ini ファイルを配置する場合は、schtasks コマンドを実行する時のパス名指定部分を適宜変更してください。

ScriptExecCmd.iniファイルは下記の内容で作成します。ScriptExecCmd.exe を実行したときに、iniファイルが見つからない場合は、自動的にiniファイルがデフォルト値で作成されます。

ファイル名: ScriptExecCmd.ini

キットに付属のメディア中の”スクリプトファイル¥HS005¥応用1”フォルダに格納されていますので、

”C:¥Tools¥Bin” フォルダにコピーして使用してください。

```
[ScriptExecCmd]
ScriptName=HS0051_ACQUISITION
HostName=localhost
UserName=user
Password=xxxxxx
KeyList=
ValList=
```

`Password=xxxxxx` 部分には、DeviceServer のユーザーアカウント(user) のパスワードを入れてください。

`ScriptName=HS0051_ACQUISITION` で実行するスクリプトを指定しています。

`HostName=localhost` は、スクリプトを実行するサーバーのホスト名で “localhost”を指定します。

7.6.3 タスクスケジューラへのJOB登録

DeviceServer の動作する PC で、コマンドプロンプトを起動して、下記のコマンドを実行してください。必ずシステム管理者権限を持った Windows アカウントでログインしてから実行してください。

下記のコマンドは､Windows のシステム管理者特権のユーザー名が “Administrator” でパスワードが “******” の場合の例になります。2 行で表示されていますが、実際には１行でコマンドプロンプトから入力してください。

```
schtasks /create /tn HS0051_TASK /sc hourly /mo 1 /st 00:00:00 /ru Administrator /rp ******* /tr
    C:¥tools¥bin¥scriptexeccmd.exe
```

このコマンドで1 時間ごとに C:¥tools¥bin¥scriptexeccmd.exe プログラムがタスクスケジューラから呼び出されて、、HS0051_ACQUISITION スクリプトが実行されます。

Windows タスクスケジューラと “schtasksコマンド” についての詳しい説明はマイクロソフト社のドキュメントを参照してください。

タスクスケジューラに対する、他のコマンド実行例は下記のようになります。

```
** Windows スケジューラ登録コマンド例 **
schtasks /create /tn HS0051_TASK /sc hourly /mo 1 /st 00:00:00 /ru Administrator /rp ******* /tr
```

```
C:¥tools¥bin¥scriptexeccmd.exe


** Windows スケジューラ確認コマンド例 **
schtasks /query


** Windows スケジューラ削除コマンド例 **
schtasks /delete /tn HS0051_TASK
```

7.6.4 動作確認

集計用スクリプトを実行すると、前回と同様に CSV ファイルが作成されます。

```
"DateTime","Temperature","Count","CDS","Count"
"2011/03/28 00:00:00","0","0","0","0"
"2011/03/28 01:00:00","0","0","0","0"
"2011/03/28 02:00:00","0","0","0","0"
"2011/03/28 03:00:00","0","0","0","0"
"2011/03/28 04:00:00","0","0","0","0"
"2011/03/28 05:00:00","0","0","0","0"
"2011/03/28 06:00:00","11.1182","1","759","1"
"2011/03/28 07:00:00","12.085","1","852","1"
"2011/03/28 08:00:00","13.0518","1","835","1"
"2011/03/28 09:00:00","13.5352","1","858","1"
"2011/03/28 10:00:00","14.0186","1","894","1"
"2011/03/28 11:00:00","14.502","1","941","1"
"2011/03/28 12:00:00","0","0","0","0"
"2011/03/28 13:00:00","0","0","0","0"
"2011/03/28 14:00:00","0","0","0","0"
"2011/03/28 15:00:00","0","0","0","0"
"2011/03/28 16:00:00","0","0","0","0"
"2011/03/28 17:00:00","0","0","0","0"
"2011/03/28 18:00:00","0","0","0","0"
"2011/03/28 19:00:00","0","0","0","0"
"2011/03/28 20:00:00","0","0","0","0"
"2011/03/28 21:00:00","0","0","0","0"
"2011/03/28 22:00:00","0","0","0","0"
"2011/03/28 23:00:00","0","0","0","0"
```

前回の集計結果とは異なって、Count 部分は常に 1 になっています。これは１時間に１回だけ正確な時刻にデータ

を登録しているためです。

8 [HS006] 温度と明るさ、ドアの開閉を監視する

8.1 アプリケーション説明

外出先からメールで、温度や明るさ、リードスイッチの監視の開始や中止を操作します。

監視中は LED が点灯します。監視中にセンサーがあからじめ決められた範囲を越えて変化した場合には、センサー値に変化があった事を知らせるアラートメールが届きます。

タクトスイッチを押すことでも、監視の開始と中止を設定できます。この場合はタクトスイッチを押してから監視を開始するまでに 60 秒の猶予時間があります。また、タクトスイッチで監視を開始した場合には、監視しているセンサーの変化を検出してからアラートメール送信まで、同様に60 秒間猶予時間があります。その間にタクトスイッチを押して、監視を中止した場合にはアラートメール送信は行いません。

8.2 機能と動作フロー

このアプリケーションでは下記の機能を実現します。

(1) A/D 変換値が一定以上変化した場合にメールを送信

(2) Digital I/O(リードSW) が変化した場合にメールを送信

(3) I/O(タクトSW) が変化した場合に監視状態を ON,OFF 切り替え

(3) DeviceServer の共有変数機能を使用して、複数のスクリプト間でのフラグ(監視フラグ値)の管理

(4) メールからスクリプトを起動させるためのメールコマンド機能

これらの機能を使用して、下記の動作フローでアプリケーションが実行されます。

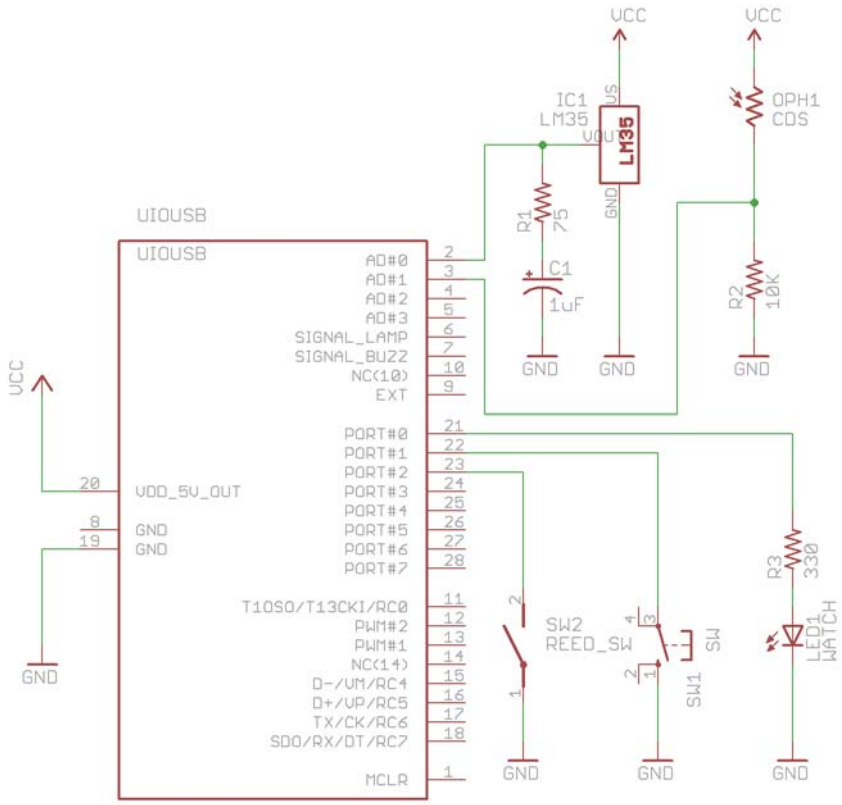
(A) 最初に監視開始のためにタクトスイッチを押します。監視状態になるまでの猶予時間(１分)ウェイトされる。

(B) 監視中に、明るさや温度、リードスイッチが変化した場合にはメールを送信します。

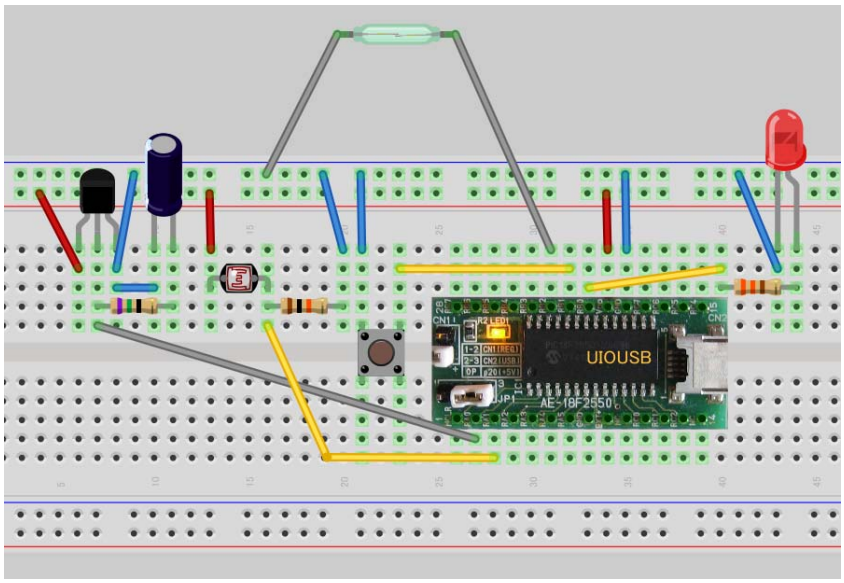
(C) 監視停止にするためには再度タクトスイッチを押します。

(D) 監視開始と停止用のスクリプトは電子メールからスクリプトを指定して実行させることもできます。

8.3 回路図



8.4 配線図



8.5 スクリプトファイル設定と説明

最初にUIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS006_CONFIG.lua

キットに付属のメディア中の”スクリプトファイル¥HS006”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS006_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: 温度と明るさリードスイッチを監視する
********************************************************************************
]]
log_msg("start..",file_id)
----------------------------------------
-- UIOUSB コンフィギュレーション設定
----------------------------------------
if not uio_command("dcfg 0xFE") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x06") then error() end
if not uio_command("sampling_rate 0") then error() end
if not uio_command("ad_margin0 2") then error() end
if not uio_command("ad_margin1 50") then error() end
if not uio_command("save",1000) then error() end
log_msg("end.",file_id)
```

`if not uio_command("dcfg 0xFE") then error() end` は I/O ポートのビット#0 をLED 点灯用に出力モードに設定して、他のビットを全て入力モードに設定してタクトスイッチとリードスイッチの入力を行います。

`if not uio_command("change_detect 0x06") then error() end` は、てタクトスイッチとリードスイッチの入力を監視します。

`if not uio_command("ad_margin0 2") then error() end` は、A/D #0に接続した温度センサーの A/D 変換値が 2以上変化した時に自動的にイベントを発生させています。A/D 変化値 2 は約１℃の変化に相当します。ここで設定した範囲を超えて A/D 値に変化があると、UIOUSB は “ADVAL_UPDATE” イベントを送信します。ここでは温度に変化があったことだけを検出していて、監視を開始したときと比べて温度が一定以上変化したかどうかはイベントハンドラ中で調べています。

`if not uio_command("ad_margin1 50") then error() end` は、A/D #0に接続した光センサーの A/D 変換値が 50以上変化した時に自動的に“ADVAL_UPDATE”イベントを発生させています。ここでは光センサーに変化があったことだけを検出していて、監視を開始したときと比べて明るさが一定以上変化したかどうかはイベントハンドラ中で調べています。

次に、スイッチ入力や A/D 変換値が変化した時に実行される UIOUSB イベントハンドラを作成します。

ファイル名:UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の"スクリプトファイル¥HS006"フォルダに格納されていますので、

"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"

--[[
*******************************************************************************
* イベントハンドラスクリプト実行時間について                                  *
*******************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*******************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
-------------------------------------------------------------------------------
キー値                          値                                              例
-------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名    "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                     2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA__1 は常にイベントプリフィックス文字列を表す      "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                             "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);
```

```
if g_params["EVENT_DATA_2"] == "ADVAL_UPDATE" then
  ------------------------------------------------------
  -- 温度または明るさが変化している時を選択
  ------------------------------------------------------
  local adc_vref = 4.95;
  local stat,val;
  local diff_temp,diff_cds;

  stat,val = get_shared_data("REF_TEMP")
  if not stat then error() end
  if (val ~= "") then
    local val_diff = math.abs(tonumber(g_params["EVENT_DATA_4"]) - tonumber(val));
    diff_temp = (100 * adc_vref * val_diff) / 1024;
    ------------------------------------------------------------
    -- 温度が 2℃以上変化したときにアラーム送信する
    ------------------------------------------------------------
    if diff_temp > 2 then
      local stat2,taskid = script_fork_exec("HSO06_ALARM","Type","温度")
      if not stat2 then error() end
    end;
  end;

  stat,val = get_shared_data("REF_CDS")
  if not stat then error() end
  if (val ~= "") then
    diff_cds = math.abs(tonumber(g_params["EVENT_DATA_5"]) - tonumber(val));
    ------------------------------------------------------------
    -- 明るさが 100 以上変化したときにアラーム送信する
    ------------------------------------------------------------
    if diff_cds > 100 then
      local stat2,taskid = script_fork_exec("HSO06_ALARM","Type","明るさ")
      if not stat2 then error() end
    end;
  end;
end;

if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  ------------------------------------------------------
  -- リードスイッチ が変化している時を選択
```

```
  ------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x04) > 0 ) then
    ---------------------------------------
    -- 現在監視中かどうかをチェックする
    ---------------------------------------
    local stat,flag = get_shared_data("WATCH_START")
    if not stat then error() end
    if (flag ~= "") then
      ------------------------------------------------
      -- アラーム送信する
      ------------------------------------------------
      local stat2,taskid = script_fork_exec("HS006_ALARM","Type","リードスイッチ")
      if not stat2 then error() end
    end;
  end;


  ------------------------------------------------------------------------------
  -- 監視開始/中止ボタンが押された時を検出
  ------------------------------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x02) > 0 ) and
    (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x02) == 0) then


    local stat,val = inc_shared_data("BUTTON_COUNT")
    if not stat then error() end
    if (tonumber(val) == 1) then
      -- ボタンを1回目に押した
      local stat2,taskid = script_exec("HS006_WATCH_START","DelayTime","60")
      if not stat2 then error() end
    else
      -- ボタンを2回目に押した
      if not set_shared_data("BUTTON_COUNT","") then error() end
      local stat2,taskid = script_exec("HS006_WATCH_STOP","","")
      if not stat2 then error() end
    end
  end;
end;
```

`if g_params["EVENT_DATA_2"] == "ADVAL_UPDATE" then` は、イベントが “ad_margin” コマンドで設定したA/D 変換値の範囲を超えて変化したために発生した “ADVAL_UPDATE” であるかどうかを調べています。

UIOUSB イベントデータ(ADVAL_UPDATE) の内容は下記の様になっています(UIOUSB ユーザーマニュアルより抜粋)

```
$$$,ADVAL_UPDATE,[ad_update_bits],[ad0],[ad1],[ad2],[ad3],[diff0],[diff1],[diff2],[diff3]
```

[ad_update_bits] には、変化したビットを 1、変化していないビットを 0にした値が、16進数で 00 から 0F までの値で、先頭の"0x"部分を取り除いて大文字で表記したものが入ります。
例えば、[ad_update_bits] が 03 の場合には ad0, ad1 の両方が、前回 ad0, ad1 のそれぞれに対するA/D 変化イベントを検出してから ad_margin0,ad_margin1 値で設定した値以上に変化したことを示します。

[adc0].. [ad3] には、最新の A/D 入力変換値が 10 進数表記で設定されます。

[diff0].. [diff3]には、各A/D チャンネルごとの変化(差分)値が入ります。値がマイナスの場合は前回と比較して値が減少している事を示します。

```
stat,val = get_shared_data("REF_TEMP")
if not stat then error() end
if (val ~= "") then
```

は、監視開始時に実行した HS006_WATCH_STARTスクリプトで設定されたグローバル共有変数 "REF_TEMP"が存在するかどうかを調べています。もし存在しない場合にはまだ監視開始状態になっていないので処理を行いません。もし存在する場合には、グローバル共有変数の値は監視を開始したときの温度が格納されていますので、現在の温度との比較に使用します。

```
local stat2,taskid = script_fork_exec("HS006_ALARM","Type","温度")
```

はアラームメールを送信するためのスクリプトを呼び出しています。スクリプトパラメータに異常を検出したセンサータイプを渡すことで、アラームメール中にその内容を含められます。

```
local stat,flag = get_shared_data("WATCH_START")
if not stat then error() end
if (flag ~= "") then
```

は、監視開始時に実行した HS006_WATCH_STARTスクリプトで設定されたグローバル共有変数 "WATCH_START"が存在するかどうかを調べています。もし存在しない場合にはまだ監視開始状態になっていないので処理を行いません。

次に、アラームメールを送信するためのスクリプトを作成します。

ファイル名: HS006_ALARM.lua

キットに付属のメディア中の"スクリプトファイル¥HS006"フォルダに格納されていますので、
"C:¥Program Files¥AllBlueSystem¥Scripts" フォルダにコピーして使用してください。

```
file_id = "HS006_ALARM"

--[[
**********************************************************************
監視中にセンサーが反応したのでアラームメールを送信する
**********************************************************************
]]
log_msg("start..",file_id)
-------------------------------------------------------
-- アラームメールの宛て先を設定する
-------------------------------------------------------
local mail_addr = "監視警報メール宛先 <your_mail_address@your.mail.domain>";

-------------------------------------------------------
-- 前回アラーム出力を行ったときからの経過時間を計測する
-------------------------------------------------------
local t = os.time();

local stat,prev_t = get_shared_data("PREV_ALARM_TIME")
if not stat then error() end
if (prev_t ~= "") then
  -------------------------------------------------------
  -- 前回のアラーム出力から 60 秒以内の場合には送信しない
  -------------------------------------------------------
  local diff_t = os.difftime(t,tonumber(prev_t));
  if diff_t < 60 then
    log_msg("連続送信キャンセル:" .. g_params["Type"],file_id)
    do return end;
  end;
end;
-------------------------------------------------------
-- 次回アラーム出力時の経過時間計測用に現在時刻を保存する
-------------------------------------------------------
if not set_shared_data("PREV_ALARM_TIME",tostring(t)) then error() end

-------------------------------------------------------------
-- 監視開始時に指定された猶予時間がある場合には、アラーム送信
-- 実行までにもその猶予時間を適用する
```

```
------------------------------------------------------------
local stat,delay = get_shared_data("WATCH_DELAY")
if not stat then error() end
if (delay ~= "") then
  wait_time(tonumber(delay) * 1000);
end;

------------------------------------------------------
-- 猶予時間ウェイト中に監視を中止されたかどうかを調べる
------------------------------------------------------
stat,flag = get_shared_data("WATCH_ABORT")
if not stat then error() end
if (flag ~= "") then
  log_msg("監視停止ボタンによってキャンセルされました:" .. g_params["Type"],file_id)
  do return end;
end;

------------------------------
-- アラームメール送信
------------------------------
local body = {};
table.insert(body,"センサー値の変化を検出しました:" .. g_params["Type"])
if not mail_send(mail_addr,"","** アラームメール送信 **",unpack(body)) then error() end

------------------------------
-- アラームログ出力
------------------------------
for key,val in ipairs(body) do
  log_msg((val),file_id)
end
```

`local mail_addr = "監視警報メール宛先 <your_mail_address@your.mail.domain>";` メール送信先のアドレスを変数に保存しています。この部分を送信したい先のメールアドレスに変更してください。

`local t = os.time();` は現在の時刻を取得する Lua のライブラリ関数です。

`local stat,prev_t = get_shared_data("PREV_ALARM_TIME")` は、前回メール送信を行った時刻が格納されたグローバル共有変数の値を取得しています。ここで取得した値と現在時刻を比較することで連続して短時間にメール送信さ

れるのを防止しています。

`local diff_t = os.difftime(t,tonumber(prev_t));` は、前回メール送信を行ってから現在までの経過秒数を計算しています。

```
local stat,delay = get_shared_data("WATCH_DELAY")
if not stat then error() end
if (delay ~= "") then
  wait_time(tonumber(delay) * 1000);
end;
```

監視を開始するスクリプト HS006_WATCH_START で設定されたグローバル共有変数 “WATCH_DELAY” を読み込んでいます。この値は、監視開始をタクトスイッチで行った時だけ猶予時間(秒) が設定されます。グローバル共有変数 “WATCH_DELAY”が設定されていた場合には、アラームメール送信を行う前にこの猶予時間分だけウェイトします。その後ウェイト中に監視が中止されていないことを確認した後に、実際のメール送信を実行するようにします。

```
local body = {};
table.insert(body,"センサー値の変化を検出しました:" .. g_params["Type"])
if not mail_send(mail_addr,"","** アラームメール送信 **",unpack(body)) then error() end
```

は、アラートメールの送信を行っています。メールの宛先、本文、表題を指定してメール送信用ライブラリ関数を実行します。

次に、監視を開始するためのスクリプトを作成します。

ファイル名: HS006_WATCH_START.lua

キットに付属のメディア中の”スクリプトファイル¥HS006”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS006_WATCH_START"
--[[
*******************************************************************************
監視を開始する
*******************************************************************************

HS006_WATCH_START スクリプト起動時に渡されるパラメータ
-------------------------------------------------------------------------------
  キー値          値                                  値の例
-------------------------------------------------------------------------------
  DelayTime       開始するまでの猶予時間(秒)          "60"
                  アラートメール送信までの猶予時間にも同じ値を使用する。
```

```
                    パラメータ省略時はすぐに監視を開始する
--------------------------------------------------------------------------------
]]
log_msg("start..",file_id)

local cnt,stat,handle,flag;

--------------------------------------------------------------------------------
-- このスクリプトを起動してから猶予時間で指定された間ウェイトする。
-- ウェイト中に監視を中止された場合を検出するために WATCH_ABORTフラグを使用している
--------------------------------------------------------------------------------
if not set_shared_data("WATCH_ABORT","") then error() end

------------------------------------------
-- パラメータで指定された猶予時間ウェイトする
------------------------------------------
if g_params["DelayTime"] then
  wait_time(tonumber(g_params["DelayTime"]) * 1000);
  if not set_shared_data("WATCH_DELAY",g_params["DelayTime"]) then error() end
else
  if not set_shared_data("WATCH_DELAY","") then error() end
end;

--------------------------------------------------
-- 猶予時間ウェイト中に監視を中止されたかどうかを調べる
--------------------------------------------------
stat,flag = get_shared_data("WATCH_ABORT")
if not stat then error() end
if (flag ~= "") then
  do return end;
end;

------------------------------------------
-- 監視開始時のセンサー値を比較用に保存する
------------------------------------------
local stat,ad,io;
stat,ad = uio_ad()
if not stat then error() end
```

```
stat,io = uio_di()
if not stat then error() end

if not set_shared_data("REF_TEMP",tostring(ad[1])) then error() end
if not set_shared_data("REF_CDS",tostring(ad[2])) then error() end

---------------------------------------
-- 監視開始フラグの設定
---------------------------------------
if not set_shared_data("WATCH_START","1") then error() end

---------------------------------------
-- 監視開始ランプを点灯
---------------------------------------
if not uio_command("do0 1") then error() end;

log_msg("end.",file_id)
```

`wait_time(tonumber(g_params["DelayTime"]) * 1000);` はスクリプトパラメータに “DelayTime” が指定されていたときにウェイトしています。またグローバル共有変数 “WATCH_DELAY” に “DelayTime” パラメータで与えられた内容を保存しておいて、アラームメール送信前にも同様にウェイトするために使用します。

`stat,flag = get_shared_data("WATCH_ABORT")` は監視を中止するスクリプト”HS006_WATCH_STOP” で設定されるグローバル共有変数をチェックして監視が中止されていたかどうかを判断しています。

```
stat,ad = uio_ad()
stat,io = uio_di()
if not set_shared_data("REF_TEMP",tostring(ad[1])) then error() end
if not set_shared_data("REF_CDS",tostring(ad[2])) then error() end
```

は、監視を開始したときのセンサー値をグローバル共有変数に保存しています。この値とセンサー値が変化した時のイベントハンドラ中で取得した値とを比べてアラームメールを送信するかどうかを判断します。

次に、監視を終了するためのスクリプトを作成します。

ファイル名: HS006_WATCH_STOP.lua

キットに付属のメディア中の”スクリプトファイル¥HS006”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

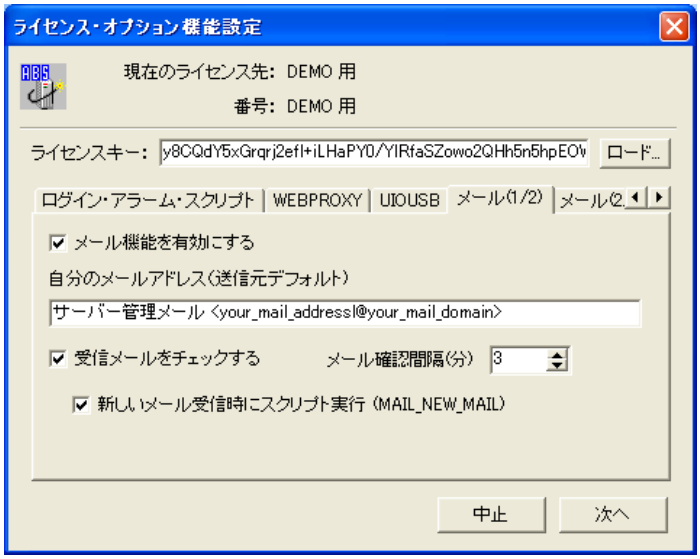
```
file_id = "HS006_WATCH_STOP"
```

```
--[[
********************************************************************************
監視を終了する
********************************************************************************
]]
log_msg("start..",file_id)
if not set_shared_data("WATCH_ABORT","1") then error() end
if not set_shared_data("WATCH_START","") then error() end
if not uio_command("do0 0") then error() end;
if not set_shared_data("REF_TEMP","") then error() end
if not set_shared_data("REF_CDS","") then error() end
if not set_shared_data("PREV_ALARM_TIME","") then error() end
log_msg("end.",file_id)
```

8.6 メール設定

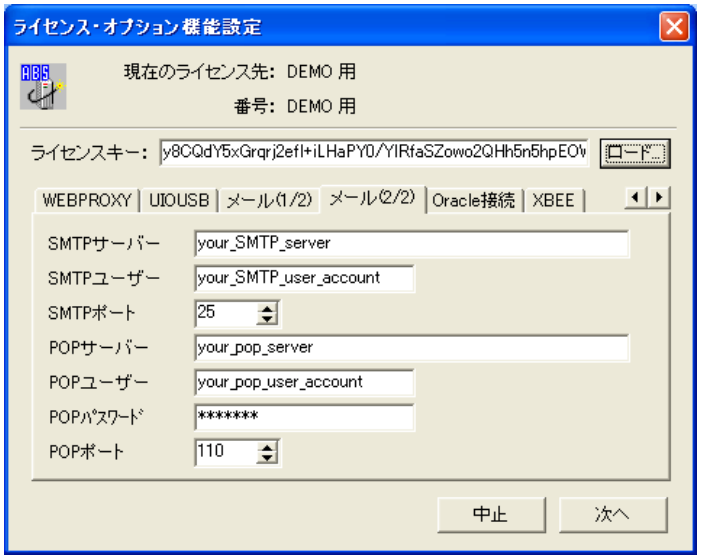
アラームメール送信時に使用する SMTP メールアカウントと、携帯電話などからインターネットメール経由でログインとスクリプト実行を行うための POP メールアカウントを DeviceServer に設定します。

プログラムメニューから “All Blue System” -> “サーバー設定プログラム” を選択してサーバー設定画面を表示します。設定項目タブの ”メール(1/2)” と”メール(2/2)”を選択して、契約されているインターネットプロバイダのメールアカウント設定を行ってください。



“メール機能を有効にする” にチェックをすることで DeviceServer でメールの受信と送信機能が使用できるようになります。”自分のメールアドレス” は、メール送信ライブラリ関数を使用するときの送信元アドレスに使用されます。

その他の設定は、メール経由でスクリプト実行を行うために必要となりますので、全てチェックを付けてください。



全ての設定項目の入力が終わりましたら、”次へ”ボタンを押すことで、新しい設定内容で DeviceServer が再起動します。

8.7 動作確認(タクトスイッチで監視操作)

最初にタクトスイッチを押して監視開始を行います。

ボタンを押すと監視が開始されるまでに 60 秒の猶予時間がありますので、その間にセンサーの検知範囲から外に出てください。監視が開始されると LED が点灯します。

監視中に温度センサーや明るさのセンサー値がスクリプト中で指定した範囲を超えて変化した場合や、リードスイッチが変化した場合には下記のメールが届いて、アラーム内容が報告されてきます。

メール受信	
メール件名	** アラームメール送信 **
メール本文	センサー値の変化を検出しました:温度

監視を中止する場合には、タクトスイッチを押してください。タクトスイッチを操作する間にセンサーが反応する可能性がありますが、アラームメールが送信されるまでには、同様に 60 秒の猶予時間があります。その間に監視を中止すればアラームメールが送信されません。

8.8 動作確認(外出先から携帯で監視操作)

次に、外出先などから携帯電話のインターネットメールを経由して監視の開始と中止を行います。

インターネットメール経由で DeviceServer のスクリプトを実行するために、最初にログインメールを送信して認証の後、スクリプト実行メールを送信します。スクリプト実行メールには、ログイン認証メールの返信で受信したセッショントークンと、実行するスクリプト名とパラメータリストを記述します。

- (1) ログイン認証を行うために、ログインメールを送信する

メール送信	
メール宛先	your_mail_address@your_mail_domain
メール件名	$LOGIN$
メール本文	user_name user_password

メール宛先は、DeviceServer で設定したPOPサーバーのメールアドレスを指定します。

メール件名には “$LOGIN$” を記述してください。（ASCII大文字）

user_name, user_password には、DeviceServerのユーザーに登録されたものを使用します。

メールを送信すると、DeviceServer は定期的に POP サーバーをチェックしていて、メール件名が “$LOGIN$” のメールをメールボックスに見つけた場合にはそのメールを取得した後、ログイン操作を行って結果をメール送信元に返信します。

メール経由でログインするDeviceServer のユーザーアカウント情報の “メールコマンド応答先” 項目を設定することで、メール経由でのログイン返信先を固定することもできます。この場合にはログインメールの処理結果はメール送信元ではなく、“メールコマンド応答先” 項目で設定した宛先に返信することでセキュリティを高められます。詳しくは、”DeviceServerユーザーマニュアル” を参照してください。

- (2) ログイン認証成功、セッショントークンメール受信する

ログインメールを送信してしばらく経つと、下記の様なログイン認証結果のメールを受信します。

メール受信	
メール件名	$REPLY$ -LOGIN SUCCESS-
メール本文	ログイン成功 ST02296B192B391F

２行目の文字列がセッショントークン文字列です。この文字列をクリップボードにコピーしておいて、次のスクリプト実行メールを送信するときに利用してください。

ログインに失敗した場合はエラー内容が記載されてメールが送られてきます。

- (3) 監視開始スクリプトを実行するためにスクリプト実行メールを送信する

ログイン認証に成功したら、下記のスクリプト実行メールを送信します。

メール送信	
メール宛先	your_mail_address@your_mail_domain
メール件名	$SCRIPT$
メール本文	ST02296B192B391F

	HS006_WATCH_START

メール宛先は、DeviceServer で設定したPOPサーバーのメールアドレスを指定します。

メール件名には “$SCRIPT$” を記述してください。（ASCII大文字）

メール本文の１行目にはログイン認証結果メールで取得したセッショントークン文字列を記入します。

ログイン認証を行ってから、スクリプト実行メールの送信までの時間がかかりすぎると、DeviceServer 側で自動的にセッションが削除されて、スクリプト実行が失敗してしまいますので注意してください。

デフォルトでは 10 分に設定されています。サーバー設定プログラムを使用して “自動ログアウト時間” 設定項目でこの制限値を変更できます。

ログインを省略してスクリプト実行メールだけを使用する

DeviceServer であからじめ自動ログアウトしない秘密のセッショントークンを create_session() ライブラリ関数を使用して作成しておくことで、ログイン認証を省略していきなりスクリプト実行メールを送信できるように設定することもできます。

詳しくは、”DeviceServerユーザーマニュアル” の “WebAPI 機能” 章の “セッショントークン作成方法” を参照してください。

- Step4 スクリプト実行成功メール受信

  スクリプト実行に成功すると、下記のようなメールが送信されてきますので監視状態に入ったことがこれで確認できます。

メール受信	
メール件名	$REPLY$ -SCRIPT SUCCESS-
メール本文	スクリプト実行成功

スクリプト実行に失敗した場合はエラー内容が記載されてメールが送られてきます。

監視中に温度センサーや明るさのセンサー値がスクリプト中で指定した範囲を超えて変化した場合や、リードスイッチが変化した場合には下記のメールが届いて、アラーム内容が報告されてきます。

メール受信	
メール件名	** アラームメール送信 **
メール本文	センサー値の変化を検出しました:温度

監視を中止する場合にも同様の手順で、”HS006_WATCH_STOP” スクリプトを実行してください。

9 [HS007] メール受信時にR/Cサーボで旗を立てる

このアプリケーションを構築する時には、ホームセンサーキットに同梱されたパーツの他に、下記の部品が必要になります。必要に応じて入手してくださいます様お願いします。

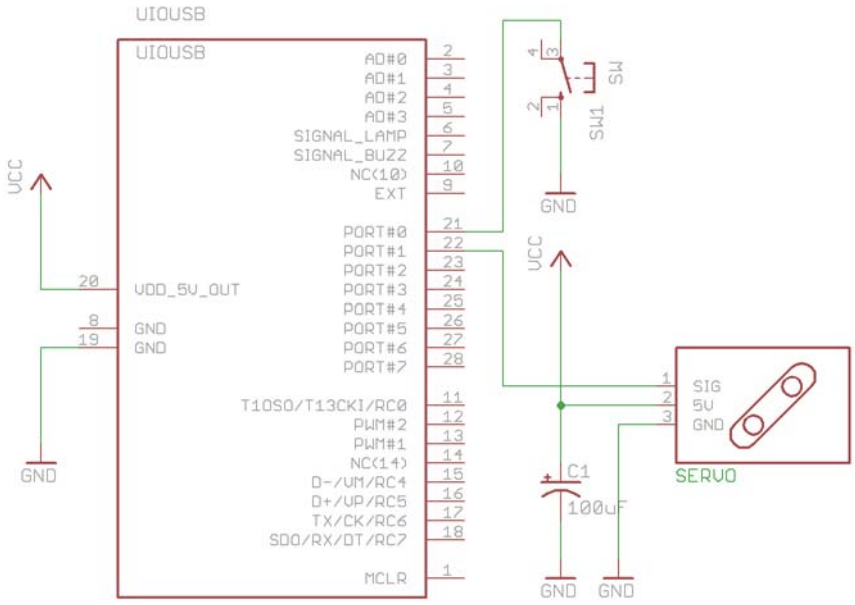
ラジコン用小型サーボ x 1 	小型のラジコン用サーボ(Futaba, JR 方式の信号に対応) Futaba, JR 方式のPWM 信号に対応します。信号線と GND、電源ラインが製品ごとに異なっていますので、事前にサーボのマニュアルを確認してください。 トルクのある強力なサーボを使用すると、消費電流が大きすぎてUSB からの電源供給では不足します。電源不要な USB ハブ経由で UIOUSB を使用している場合にも電源供給が不足する場合があります。電源供給が不足すると、UIOUSB はリセットされてWindows からは、デバイスの切断と認識が繰り返すような現象が発生します。 このような場合には、UIOUSB の電源を外部から供給する必要があります。 UIOUSB の電源ジャンパー JP1 を外部電源モードに設定して、EXT(9)ピンに外部電源(6..9V程度)を接続するように回路を変更してください。

9.1 アプリケーション説明

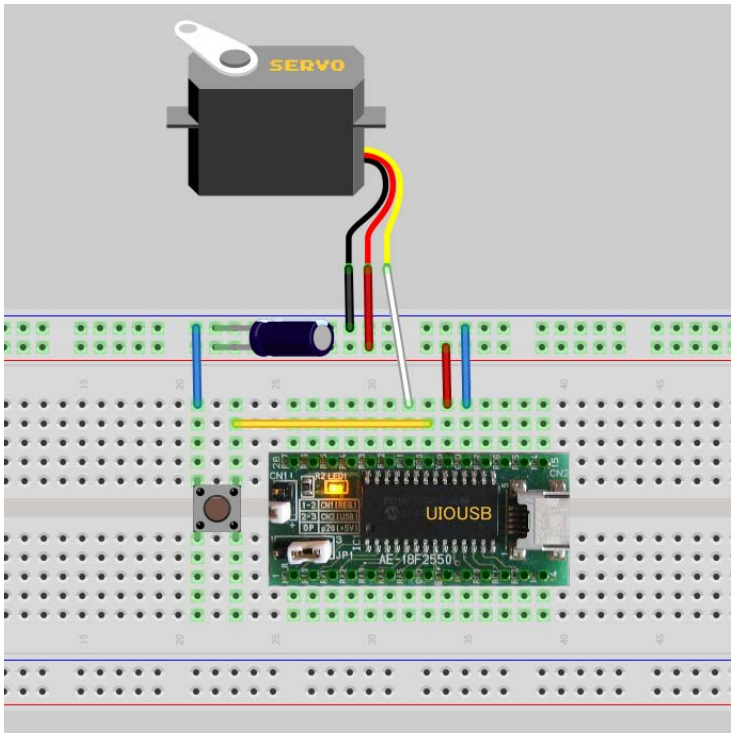
メールが届いた時に、旗を立てるアプリケーションを作成します。
旗はサーボを使用してリアルに仕上げることで、パソコンの画面を確認しなくてもメール到着をすぐに知らせてくれます。

POP メールサーバーのメールボックスを定期的にサーバーから監視して、メールが見つかった場合に UIOUSB に接続したサーボモータを動作させて旗を立てます。メールクライアントプログラムを使用して、メールをPOP サーバーのメールボックスから取り出した場合には、サーボモータを動作させて旗を降ろします。
メールボックスは数分ごとに確認して旗を動作させますが、タクトスイッチを押すことですぐに旗を降ろすこともできます。

9.2 回路図



9.3 配線図



9.4 スクリプトファイル設定と説明

UIOUSB デバイスの初期設定を行うスクリプトファイルを作成します。

ファイル名:HS003_CONF.lua

キットに付属のメディア中の"スクリプトファイル¥HS007"フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS007_CONFIG"
--[[
********************************************************************************
サンプルアプリケーション: メール受信時にサーボで旗を立てる
********************************************************************************
]]
log_msg("start..",file_id)
---------------------------------------
-- UIOUSB コンフィギュレーション設定
---------------------------------------
if not uio_command("dcfg 0x01") then error() end
if not uio_command("pullup 1") then error() end
if not uio_command("change_detect 0x01") then error() end
if not uio_command("save",1000) then error() end


log_msg("end.",file_id)
```

`if not uio_command("dcfg 0x01") then error() end` ポートのビット#0 をタクトスイッチ用に入力モードにして、その他のビットは全て出力モードに設定しています。サーボ出力を行うビット(今回はビット#1)は出力モードに設定してください。

`if not uio_command("pullup 1") then error() end` タクトスイッチのプルアップを有効にして、スイッチを押した時に Low 、離したときに High になるようにします。

`if not uio_command("change_detect 0x01") then error() end` タクトスイッチに接続したビット#0 のみを、ポート値の変化検出対象としています。

次に、タクトスイッチ入力を行った時に実行される UIOUSB イベントハンドラを作成します。

ファイル名:UIOUSB_EVENT_DATA.lua

キットに付属のメディア中の”スクリプトファイル¥HS007”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "UIOUSB_EVENT_DATA"


--[[
********************************************************************************
```

```
* イベントハンドラスクリプト実行時間について                                    *
*********************************************************************************
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つサーバー側でタイムアウトが発生します。
また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
頻繁には発生しないイベントで、処理時間がかかるスクリプトを実行したい場合は
スクリプトを別に作成して、このイベントハンドラ中から script_fork_exec() を使用して
別スレッドで実行することを検討してください。
*********************************************************************************
UIOUSB_EVENT_DATA スクリプト起動時に渡される追加パラメータ
---------------------------------------------------------------------------------
キー値                          値                                              例
---------------------------------------------------------------------------------
COMPort                         イベントを送信した UIOUSB デバイスのポート名    "COM3"
EVENT_DATA_WHOLE                カンマで区切られたUIOUSB イベントデータ全体が入る
"$$$,ADVAL_UPDATE,01,805,767,512,257"

EVENT_DATA_COUNT                UIOUSB EVENT データカラム数                     2
EVENT_DATA_<Column#> UIOUSB イベントデータ値(ASCII 文字列)
                    EVENT_DATA_1 は常にイベントプリフィックス文字列を表す       "$$$"
                    "$$$" 文字列
                    EVENT_DATA_2 はイベント名が入る                             "SAMPLING"
                    EVENT_DATA_3以降のデータはイベントごとに決められた、オプション文字列が入る

                    <Column#> には 最大、EVENT_DATA_COUNT まで 1から順番にインクリメントされた値が入る。
]]
log_msg(g_params["COMPort"] ..  " EventData = " .. g_params["EVENT_DATA_WHOLE"],file_id);

if g_params["EVENT_DATA_2"] == "CHANGE_DETECT" then
  ---------------------------------------------------------------------------------
  -- サーボ旗を DOWN の状態にする
  ---------------------------------------------------------------------------------
  if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) and
    (bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then
    local stat,taskid = script_fork_exec("HSO07_FLAG_MOVE","Flag","DOWN")
    if not stat then error() end
  end;
end;
```

```
```

`if (bit_and(tonumber(g_params["EVENT_DATA_3"],16),0x01) > 0 ) and`
`(bit_and(tonumber(g_params["EVENT_DATA_4"],16),0x01) == 0) then`
は、タクトスイッチが操作されていて、かつスイッチが押し込まれた状態(Low) を検出しています。スイッチを離した状態(High)の時のイベントは無視しています。

`local stat,taskid = script_fork_exec("HS007_FLAG_MOVE","Flag","DOWN")` は、サーボを操作するスクリプト“HS007_FLAG_MOVE” をパラメータとしてキー “Flag” と“DOWN” を指定してコールしています。これによって旗を降ろした状態にします。

`script_fork_exec()` は別スレッドでスクリプトをコールします。このためコールしたスクリプト動作の終了を待たずに、イベントハンドラは並行して実行されます。別スレッドで実行するのは、“HS007_FLAG_MOVE” スクリプト内部で時間がかかるウェイト動作や、クリティカルセクション操作を行っているためです。イベントハンドラがこれらの時間がかかる “HS007_FLAG_MOVE” スクリプトの終了を待つのは不適切なためです。

サーボを操作して、旗の上げ下ろしをするスクリプトファイルを作成します。

ファイル名: HS007_FLAG_MOVE.lua

キットに付属のメディア中の”スクリプトファイル¥HS007”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "HS007_FLAG_MOVE"

--[[
********************************************************************************
サーボ信号を操作する
********************************************************************************
HS007_FLAG_MOVE スクリプト起動時に渡されるパラメータ
--------------------------------------------------------------------------------
    キー値          値                          値の例
--------------------------------------------------------------------------------
    Flag            "UP" または "DOWN"          "UP"
--------------------------------------------------------------------------------
]]
local stat,handle,flag;

-------------------------------------
-- スクリプトパラメータ Flag チェック
```

```
-------------------------------------
if g_params["Flag"] then
  if (g_params["Flag"] ~= "UP") and (g_params["Flag"] ~= "DOWN") then
    log_msg("パラメータ Flag は UP または DOWN を指定してください",file_id);
    do return end;
  end;
else
  log_msg("パラメータ Flag が指定されていません",file_id);
  do return end;
end;

-------------------------------------------------------------
-- 現在の旗の位置が Flag で指定したものと一致する場合には
-- 無駄なサーボ動作を省略する
-------------------------------------------------------------
stat,flag = get_shared_data("SERVO_FLAG_POS")
if not stat then error() end
if (flag == g_params["Flag"]) then
   do return end;
end;

-----------------------------------------------------
-- サーボ操作を複数同時に行わないように
-- 排他制御を行う
-----------------------------------------------------
stat,handle = critical_section_enter("BiffServoOperation",10000);
if not stat then error() end

-------------------------------------------------------------
-- 現在の旗の状態を共有データに書き込んでおく
-------------------------------------------------------------
if not set_shared_data("SERVO_FLAG_POS",g_params["Flag"]) then
  if not critical_section_leave(handle) then error() end;
  error();
end;

--------------------------
-- サーボ信号を有効にする
--------------------------
```

```
if not uio_command("servo1 1",1000) then
  if not critical_section_leave(handle) then error() end;
  error();
end;


-------------------------------------------------
-- サーボ位置 旗をUP  状態にする。2.0ms パルス幅
--                DOWN状態にする。0.9ms パルス幅
-------------------------------------------------
log_msg("move position:" .. g_params["Flag"],file_id)
if g_params["Flag"] == "UP" then
  if not uio_command("pos1 200",1000) then
    if not critical_section_leave(handle) then error() end;
    error();
  end;
else
  if not uio_command("pos1 90",1000) then
    if not critical_section_leave(handle) then error() end;
    error();
 end;
end;


-------------------------------------------------------------------------------
-- サーボ位置が安定するまで 1秒ウェイトした後サーボ信号を止める。
-- サーボ位置を微調整するごとにサーボが細かく動作するのを止めています。
-- 旗は軽いので常にサーボトルクがかかっていなくても問題ありません。
--
-- 使用するサーボによっては、サーボ信号が停止した時に最後のポシションを
-- 維持しつづけない場合があります。その時は以下をコメントアウトしてください。
-------------------------------------------------------------------------------
wait_time(1000);
if not uio_command("servo1 0") then
  if not critical_section_leave(handle) then error() end;
  error();
end;


-----------------------------------------------------------
--  サーボ操作の排他制御を終了
-----------------------------------------------------------
```

```
if not critical_section_leave(handle) then error() end;
```

`if (g_params["Flag"] ~= "UP") and (g_params["Flag"] ~= "DOWN") then` スクリプトパラメータ “Flag” は必須パラメータとしています。このパラメータが指定されているかどうかをチェックしています。”UP” で旗を揚げた状態にして、”DOWN” で降ろした状態にします。

`stat,flag = get_shared_data("SERVO_FLAG_POS")` 前回このスクリプトを実行して旗を操作したときに、旗の状態が"SERVO_FLAG_POS"グローバル共有変数に設定されています。そのときに設定したグローバル共有変数の内容を調べて、もし今回パラメータで指定された状態と一致する場合には、なにもしないでスクリプトを抜けます。

`stat,handle = critical_section_enter("BiffServoOperation",10000);` サーボ操作を複数同時に行わないように排他制御をするために、クリティカルセクションを作成しています。
定期的にメールボックスを確認して旗を動かすためにサーボが動作すると同時にタクトスイッチが押されて、旗を下げるためのサーボ動作が重なった場合などを考慮しています。

`if not set_shared_data("SERVO_FLAG_POS",g_params["Flag"]) then` 次回このスクリプトが実行されたときに、旗の現在の状態をグローバル共有変数に記録しておくことで無駄なサーボ動作を防止します。

`if not uio_command("servo1 1",1000) then` ポートビット#1 からサーボ信号を出力します。サーボ位置（パルス幅）は最後に “pos” コマンドで設定した値で出力します。

```
if not uio_command("pos1 200",1000) then
if not uio_command("pos1 90",1000) then
```

はサーボ位置を変更するために UIOUSB の “pos” コマンドを実行しています。サーボの種類やサーボの取り付け位置によってパルス幅(“200” や”90” で指定している数値）を変更して、最適な位置に調整してください。

```
wait_time(1000);
if not uio_command("servo1 0") then
```

は、サーボの位置が安定するまで１秒ウェイトした後、サーボ信号を停止しています。サーボ信号を出力したままだとサーボ位置の微調整の度にサーボ動作音が発生して耳障りになるのを防いでいます。旗は軽いので、サーボ信号が無くなってサーボにトルクがかからなくなっても旗が動くことはないと思います。

`if not critical_section_leave(handle) then error() end;` はサーボの排他制御を終了するためにクリティカルセクションから抜けます。

DeviceServer で１分ごとに定期的に実行されるスクリプトファイルを作成します。

ファイル名: PERIODIC_TIMER.lua

キットに付属のメディア中の”スクリプトファイル¥HS007”フォルダに格納されていますので、

”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "PERIODIC_TIMER"
--[[
*******************************************************************************
＊イベントハンドラスクリプト実行時間について＊

一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
処理に時間がかかると、イベント処理の終了を待つアラームデバイスで、
タイムアウトが発生します。

また、同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。

*******************************************************************************
]]

-------------------------------------------------------------
-- 5 分に一回 メールボックス中のメール数を確認する
-------------------------------------------------------------
local stat,val = inc_shared_data("TIME_5M")
if not stat then error() end
if (tonumber(val) == 1) then
  -------------------------------------------------------------
  -- MAILBOX のメール数が 0 の場合には サーボ旗を DOWN して、
  -- 1 以上の場合には、サーボ旗を UP にする
  -------------------------------------------------------------
  local mailbox_stat,mailcount = get_shared_data("$MAILBOX_COUNT")
  if not mailbox_stat then error() end
  if (mailcount == "") or (mailcount == "0") then
    if not script_fork_exec("HS007_FLAG_MOVE","Flag","DOWN") then error() end;
  else
    if not script_fork_exec("HS007_FLAG_MOVE","Flag","UP") then error() end;
  end;
end

if (tonumber(val) > 5) then
  stat = set_shared_data("TIME_5M","")
```

```
  if not stat then error() end
end
```

`local stat,val = inc_shared_data("TIME_5M")` は、５分をカウントするためのグローバル共有変数に１をインクリメントしています。

`local mailbox_stat,mailcount = get_shared_data("$MAILBOX_COUNT")` DeviceServer が POP メールサーバーに定期的にメールボックスに最後にメールを確認したときのメール数が常にグローバル共有変数 “$MAILBOX_COUNT” に格納されています。このグローバル共有変数の内容(数値を文字列形式にしたもの)を取得して未読メール数を判断します。

```
if (mailcount == "") or (mailcount == "0") then
  if not script_fork_exec("HS007_FLAG_MOVE","Flag","DOWN") then error() end;
else
  if not script_fork_exec("HS007_FLAG_MOVE","Flag","UP") then error() end;
end;
```

で、メールボックスの中に入っているメール数が１以上の場合には、スクリプト “HS007_FLAG_MOVE” をパラメータキー “Flag” パラメータ値 “UP” でコールして旗を揚げた状態にします。メール数が 0 の場合にはパラメータ値を “DOWN” にして旗を降ろした状態にします。

最後に、新規メール到着時に実行されるスクリプトファイルを作成します。
このスクリプトを設定しなくても、メールボックス中のメール数に応じて旗は上下します。ただし、定期的(デフォルト設定は 3 分)にPOP サーバーのメールボックス確認したときに、グローバル共有変数 “$MAILBOX_COUNT”に値を設定した後、５分ごとにこのグローバル共有変数を確認するためどうしてもタイムラグが大きくなります。メール到着時にすぐに旗を揚げるために、POP サーバーを確認してそのときに新規メールを見つけた場合に実行される MAIL_NEW_MAIL イベントハンドラにも旗の操作を追加することで、メール到着時の旗を揚げる操作を早められます。

ファイル名: MAIL_NEW_MAIL.lua

キットに付属のメディア中の”スクリプトファイル¥HS007”フォルダに格納されていますので、
”C:¥Program Files¥AllBlueSystem¥Scripts” フォルダにコピーして使用してください。

```
file_id = "MAIL_NEW_MAIL"
--[[
*****************************************************************************
＊スクリプト実行時間について＊
```

```
一つのスクリプトの実行は長くても数秒以内で必ず終了するようにしてください。
同時実行可能なスクリプトの数に制限があるため、他のスクリプトの実行開始が
待たされる原因にもなります。
*******************************************************************************
MAIL_NEW_MAIL スクリプト起動時に渡される追加パラメータ

-------------------------------------------------------------------------
  キー値          値                          値の例
-------------------------------------------------------------------------
  $RECIPIENTS$    メールの宛先アドレス        xxxさん <xxxx@your.corp.jp>
  $FROM$          メールの送信元アドレス      abc@xxxx.yyyy.zzzzz
  $SUBJECT$       メールの件名                お知らせメールです
  $ID$            メールのID                  id123.456.789.abc
  $PRIORITY$      メールのプライオリティ      3
  $MAILBOX$       メールボックス中のメール数  10
  $TIMESTAMP$     メール検索を行った日時      2000/01/01 01:05:31

＊補足＊
$TIMESTAMP$ の値はメールを検索したときに、新規メールが複数あった場合に、
それぞれのメールごとに呼び出された、MAIL_NEW_MAIL スクリプト実行時に同一
の値を示します。

$MAILBOX$ の値は新規メールの数ではなく、POP サーバーで保持されたメールの
数を示します。メールクライアントで読み込み時に、POP サーバーのメールを削除
している場合は、未読メールの数になどしくなります。

＊制限事項＊
メールヘッダのデコード時に、スペース文字が余分に入ったり抜けたりする場合があり、
これらは使用上の制限事項になります。(日本語 <-> アルファベット の区切り部分に多い)
文字列のマッチをスクリプト中で操作する場合はこれらを考慮して、実際のメールを受信
してマッチパターンの調整が必要になります。
送信元で記入したヘッダと、完全には一致しない場合がありますので注意してください。
メールの転送経路によってもヘッダ内容が変化する場合もあります。
]]

log_msg("start..",file_id)

for key,val in pairs(g_params) do
  log_msg(string.format("g_params[%s] = %s",key,val),file_id)
```

```
end


------------------------------------------------
-- 新着メールが到着したので、サーボ旗を UP にする
------------------------------------------------
if not script_fork_exec("HS007_FLAG_MOVE","Flag","UP") then error() end;
```

```
for key,val in pairs(g_params) do
 log_msg(string.format("g_params[%s] = %s",key,val),file_id)
end
```

は、新規メール到着時にヘッダ情報をログに出力するための記述です。

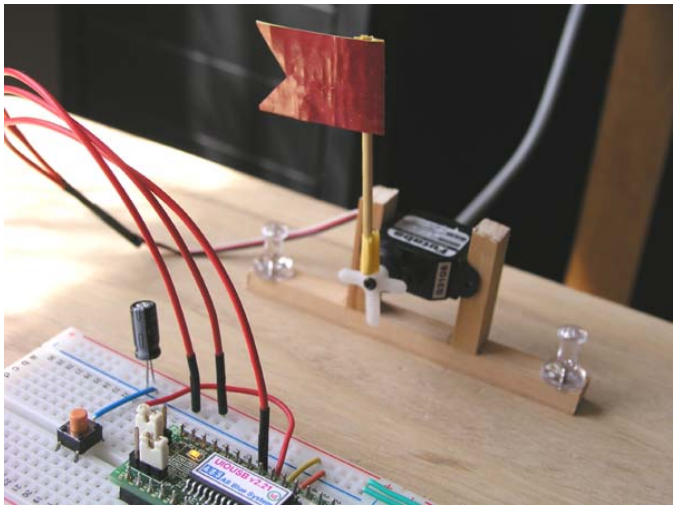
`if not script_fork_exec("HS007_FLAG_MOVE","Flag","UP") then error() end;` 新規メールが到着したときには、サーボを動作させて旗を揚げた状態にします。

9.5 メール設定

前の章“HS006”のアプリケーション中のアラートメール送信設定と同様の手順で、メール送信(SMTP)とメール受信(POP)の設定を行ってください。

9.6 動作確認

サーボのホーンの位置を調整して旗をつけます（下記写真を参考にしてください）。



POP メールサーバーにメールが入ってくると旗が揚がります。

メールクライアントプログラムを実行して、POP メールサーバー上のメールを全て取得すると、しばらくして自動的に旗が下がります。タクトスイッチを押すと、旗をすぐに降ろせます。

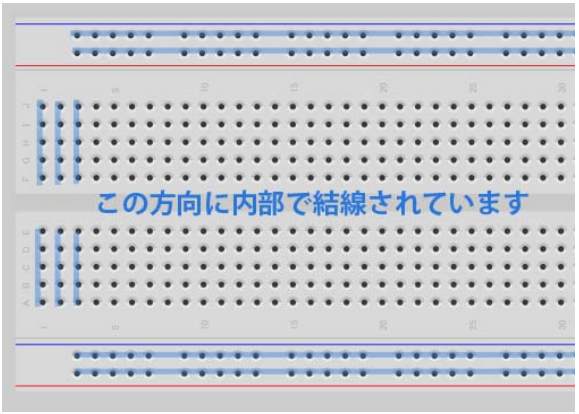
旗を操作するためにメール数を確認する頻度は、サーバー設定プログラムの“メール確認間隔”と、PERIODIC_TIMER

スクリプト中に記述したタイマー間隔を変更することで調整できます。

10　パーツの説明

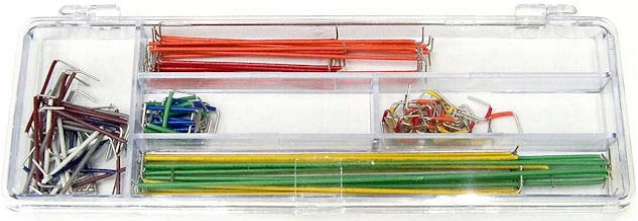
この応用ガイドで使用している各パーツの簡単な説明です。

10.1　ブレッドボード



電子部品やケーブルを接続するための基板です。内部で横・縦方向にあからじめ結線されていますので、これとジャンパー線や部品のリード線を利用して配線を行います。キットに付属している実際のパーツは、上図とは大きさや色分けが違う場合がありますが、内部の配線などは同じですので同様に使用できます。

10.2　ジャンパー線



ブレッドボード上のパーツ間を配線するために使用します。曲げて使うこともできますので、長さを調整して使用してください。

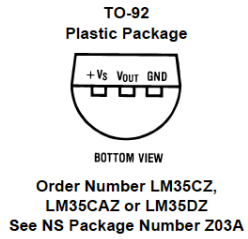
10.3　タクトスイッチ

押すと ON になって 放すと OFF になるスイッチです。方向によってスイッチ内部の結線が異なっていますので注意してください。写真手前２ピン間がスイッチ操作時に ON -> OFF に切り替わります。
手前２ピンと奥２ピン間は互いに結線されています。

10.4 リードセンサー

磁石とリードスイッチのペアです。ドアなどの開閉部分にとりつけて使用します。リードスイッチに磁石が近づいているときに ON になるように配置してください。

10.5 温度センサー



0 から 100 ℃程度まで測定可能な温度センサー(LM35DZ)です。ピンはラベル面に向かって、左から電源、計測値出力、GND です。出力値(V)は 10mv ＊ 温度(℃)で計算できます。

10.6 LED

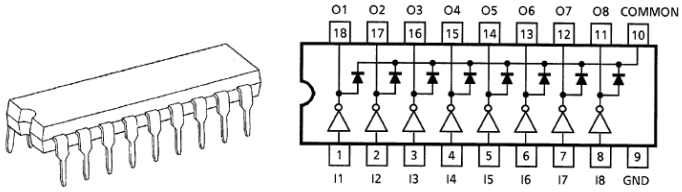
緑色、赤色、黄色の LED です。このガイドでは１つのLED のみを使用しますが、他のアプリケーションを作成する時などに利用可能な複数種類のLED が同梱されていますので活用してください。

長いリードがアノード(A) のプラス側です。短い側がカソード(K)です。ブレッドボードに差し込み易い様にリードを短く切った場合には、方向を間違えないように注意してください。

10.7 ブザー

電子ブザーです。プラス側には＋マークがありますので間違えないようにしてください。

10.8 トランジスタアレイIC



UIOUSB のポート出力で取り出し可能な 各I/O ピンの電流は 25mA 、全ポート合わせて 200mA までです。トランジスタアレイIC はそれ以上の電流が必要な場合に使用します。

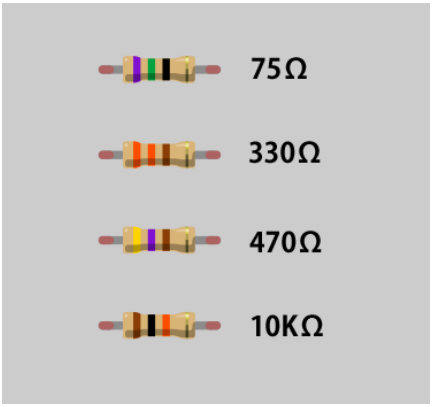
IC の各ピンごとに約 300mA 程度までの電流を扱えます。GNDピンの配線を忘れないようにしてください。

この応用ガイドのサンプルアプリケーションでは、この IC を使用しなくても PIC18F2550 から十分な電流が取れますので使用しなくても動作しますが、他のアプリケーションを試作する時に便利なように同梱しています。小型のリレーを駆動するときや高輝度LEDを使用するときなどに活用してください。

10.9 光センサー(CDS)

光の強弱で抵抗値が変化します。ピンの極性はありません。

10.10 抵抗



抵抗です。抵抗値ごとに上記のカラーコードが付いています。ピンの極性はありません。

10.11 コンデンサ

電解コンデンサです。容量はパッケージに印刷されています。

パッケージ横に縦帯でマークがある方がマイナス側になります。極性に気をつけてください。

11 更新履歴

2014/2/15 Rev A.1.4 改訂版

2012/11/14 Rev A.1.3 改訂版

2012/10/12 Rev A.1.2 改訂版

2011/12/24 Rev A.1.1 改訂版

2011/03/03 Rev A.1.0 初版作成