.NET Gadgeteer IoT キット
学習テキスト
準備と基本操作

2015 年 2 月 12 日

株式会社デバイスドライバーズ

目次

1. はじめに

.NET Micro Framework (NETMF) および.NET Gadgeteer は、Microsoft Research 社が開発して来た組み込みシステム向けのオペレーティング・システム、開発環境とライブラリ、及びそれらを開発するための規格です。現在は Apache Version 2 ライセンスに基づいたオープンソースとなり、Microsoft 社のほか FEZ Spider 等のハードウェアを開発している GHI Electronics 社も参画して開発をしています。

この学習テキストでは.NET Gadgeteer が動作する FEZ Spider メインボード、及びそれと組み合わせて利用可能なモジュール群を使用して IoT (Internet of Things) を容易に体験するための準備と基本的な開発手順を示します。自己学習にも活用して下さい。

このテキストでは次の印と項目を使用しています。独習する際に参考にしてください。

●演習	自己学習を進める上で必須となる演習です。
○演習	オプションの演習項目です。条件に合致する人を対象にしています。
★注意★	演習を進める上での重要な注意点を示します。
■ヒント■	演習を進める上での有効なヒントと参考情報を示します。
□解説□	演習や手順に関する解説です。

■ヒント：インストール先のフォルダ名■

本文中の説明では、各種ソフトウェアを 64bit 版 Windows 環境にインストールして利用する場合を想定します。32bit 版環境にインストールして使用する場合には、インストール先のフォルダ名を C:¥Program Files (x86) から C:¥Program Files に読み替えてご利用ください。

1.1. 開発環境

オペレーティング・システム：Windows Vista 以降を搭載した PC

開発環境 (Visual Studio) : Visual Studio Express 2013 with Update 4 for Windows Desktop, Visual Studio Community 2013 Update 4, または Visual Studio Professional 2013 with Update 4 以上（上位版も可、他バージョンは利用不可）

NETMF SDK : Microsoft .NET Micro Framework SDK 4.3 (QFE2) (他バージョンは利用不可)

Gadgeteer Core：Microsoft Gadgeteer Core 2.43.1000 以降

GHI SDK：GHI Electronics NETMF SDK 2014.R5 以降

.NET Gadgeteer 修正版ライブラリ：Gadgeteer.Webserver43

（株式会社デバイスドライバーズにて作成・配布、本演習に添付）

必要な DISK 容量： 空きディスク領域 2GB

マシン仕様：Pentium 4 2GHz 以上、メモリ 2GB 以上を推奨。

■ヒント：複数種類の Visual Studio のインストール■

Visual Studio はインストールの順番に関係なく、Express 版でも別のバージョンでも同じ開発環境（マシン）にインストール可能です。各 SDK は、Visual Studio のプラグインとなりますので、後からインストールした Visual Studio で SDK を使用する場合には、SDK の再インストールが必要です。

1.2. ターゲット環境

本テキストで動作対象としているメインボードは、現在 FEZ Spider だけです。

メインボード GHI electronics FEZ Spider（TinyBooter4.3.4.0 以降 TinyCLR 4.3.6.0 以降）

モジュール：USB Client SP Module または USB Client DP Module

Multicolor LED Module, Ethernet J11 Module（オプションのプログラミング演習で使用）

★注意：ファームウェアのバージョン★

上記以外のバージョンでも動作する可能性はありますが､本テキスト執筆時点では動作確認していませんのでご注意下さい。

1.3. ライセンスと免責事項

① 本資料に基づいて演習、自己学習を実施した結果、付帯して配布するソフトウェアの運用において生じた、いかなる損害について一切責任は負いません。

② 本資料で提供するソフトウェアは、Ms-PL（http://opensource.org/licenses/ms-pl）で提供します。演習で使用するソフトウェアで別のライセンスが明示されている場合には、そのライセンスに従って使用してください。

1.4. 演習の概要

① 目的

本テキストでは.NET Gadgeteer を使用した組み込みアプリケーション開発方法の基礎を学びます。

② 必要知識

・ Visual Studio の基本的な使い方を知っていることが望ましい(必須ではない)

・ C#に関する知識があることが望ましい（必須ではない）

③ 開発言語

・ C#

④ 演習終了後に得られる知識

・ .NET Gadgeteer を利用した簡単な組み込みシステムの開発方法

・ .NET Gadgeteer の開発システムの使いこなしに必要なソフトウェアのインストールとファームウェアのメンテナンス手順

2. 開発環境の準備

.NET Gadgeteer / .NET Micro Framework の開発にはコンパイルとデプロイ、デバッグを行うツールである Visual Studio とそれをサポートする SDK が必要です。開発を始める前に Windows マシンに開発環境をインストールして準備します。

2.1. Visual Studio 2013 Update 4

○演習 1）

以下の手順で Visual Studio Express 2013 for Windows Desktop を入手し、インストールを行います。

すでにインストール済の場合、Professional 以上をインストール済の場合は必要ありません。

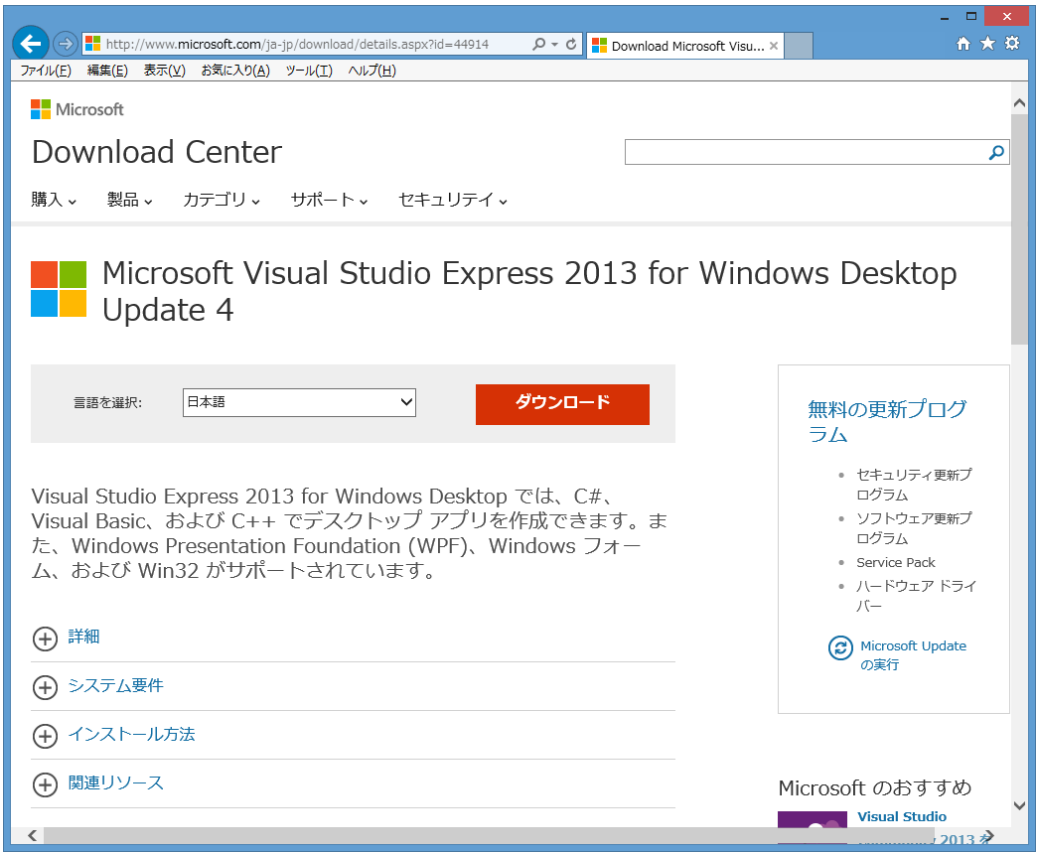
■ヒント：Visual Studio Express 2013 with Update 4 for Windows Desktop の入手先■

① http://www.visualstudio.com/downloads/download-visual-studio-vs

② http://www.microsoft.com/ja-jp/download/details.aspx?id=44914

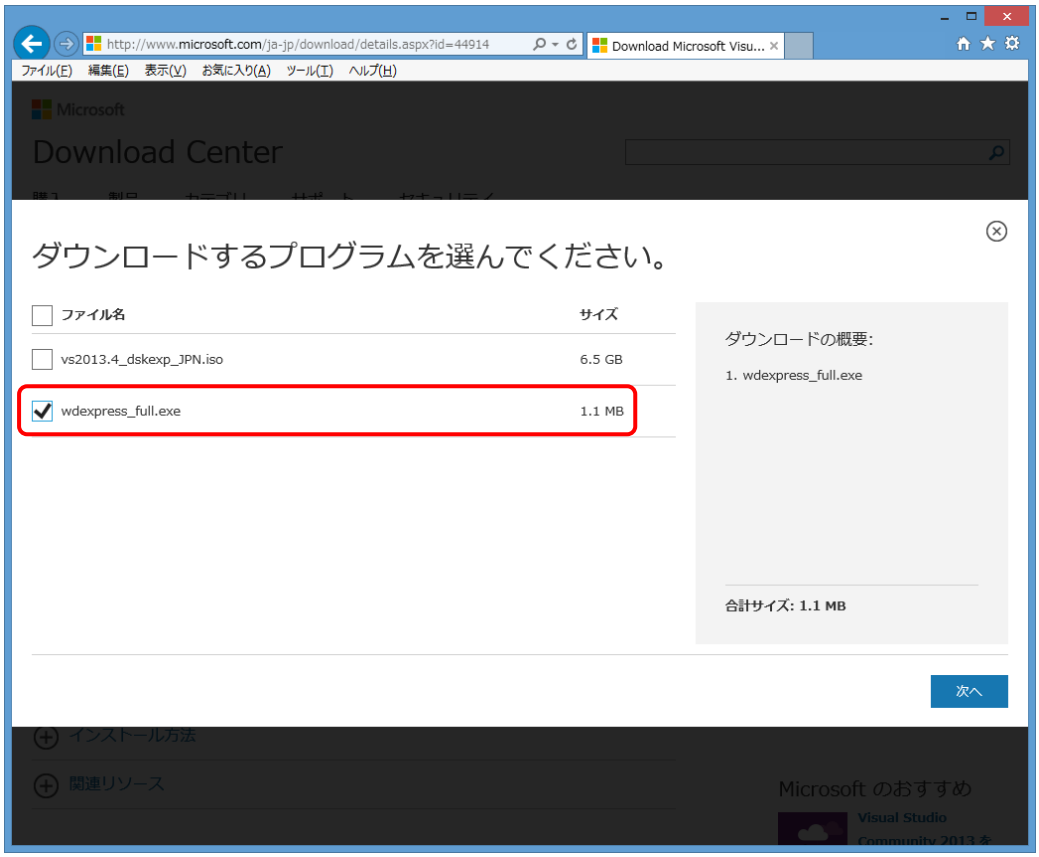
③ インターネットで「Visual Studio Express 2013 update 4」で検索する。

④ 利用権利を持っている人はMSDNのサブスクライバー・ダウンロードからも入手可能です。



http://www.microsoft.com/ja-jp/download/details.aspx?id=44914

から入手してインストールする場合には次の様に「ダウンロードするプログラムを選んでください」の画面で、wdexpress_full.exe を選択して実行すると、その場でインストールが始まります。



★注意：古いバージョン★

Visual Studio 2010 以前の古いバージョンは、現在のところ、このテキストで解説している.NET Micro Framework / .NET Gadgeteer バージョン 4.3 の開発には利用できません。

○演習 2）

前項のいずれかの方法で Visual Studio 2013 Update 4 を入手してインストールします。

□解説：Visual Studio 更新プログラム Update 4□

Visual Studio の更新プログラムは各バージョンの Visual Studio 公開後に発生した機能追加や、問題修正のために配布されています。Visual Studio 2013 Update 4 は、インストール時に最初から Update 4 の更新内容を含んでいるものです。

更新プログラムはバージョンが同じであれば Express 版を含む全ての Visual Studio に適合可能です。本テキスト作成時点での最新版の更新プログラムは「更新プログラム 4」です。Visual Studio 2013 をすでに利用していて、Update 4 にバージョンアップしていない場合は、コンパイラや開発ツール関連の機能拡張が行われているので、インストールしておくことをお勧めします。

2.2. SDK パッケージの入手

●演習３）

各サイトから SDK をダウンロードしてインストールします。

★注意：古いバージョンのアンインストール★

Microsoft .NET Micro Framework SDK 4.3 (RTM) 等、上記に指定したよりも古い各 SDK がインストールしてある場合には、SDK のインストールを始める前に、アンインストールしておく必要があります。**必ずコントロールパネルの「プログラムと機能」で確認して下さい。**

古い SDK をインストールしたままだと、新 SDK のインストール時にアンインストールや修復インストールを求められます。各略号は、

QFE=Quick Fix Engineering

RTM=Release To Market

の略で、QFE は RTM の不具合を修正したバージョンです。インストール済バージョンの確認とアンインストールは「プログラムと機能」を起動して行います。

□解説：各 SDK パッケージの入手先□

以下のサイトから各 SDK を入手します。

① Microsoft .NET Micro Framework SDK 4.3 (QFE2)
http://netmf.codeplex.com/releases/view/611040

② Microsoft .NET Gadgeteer Core バージョン 2.43.1000
http://gadgeteer.codeplex.com/releases

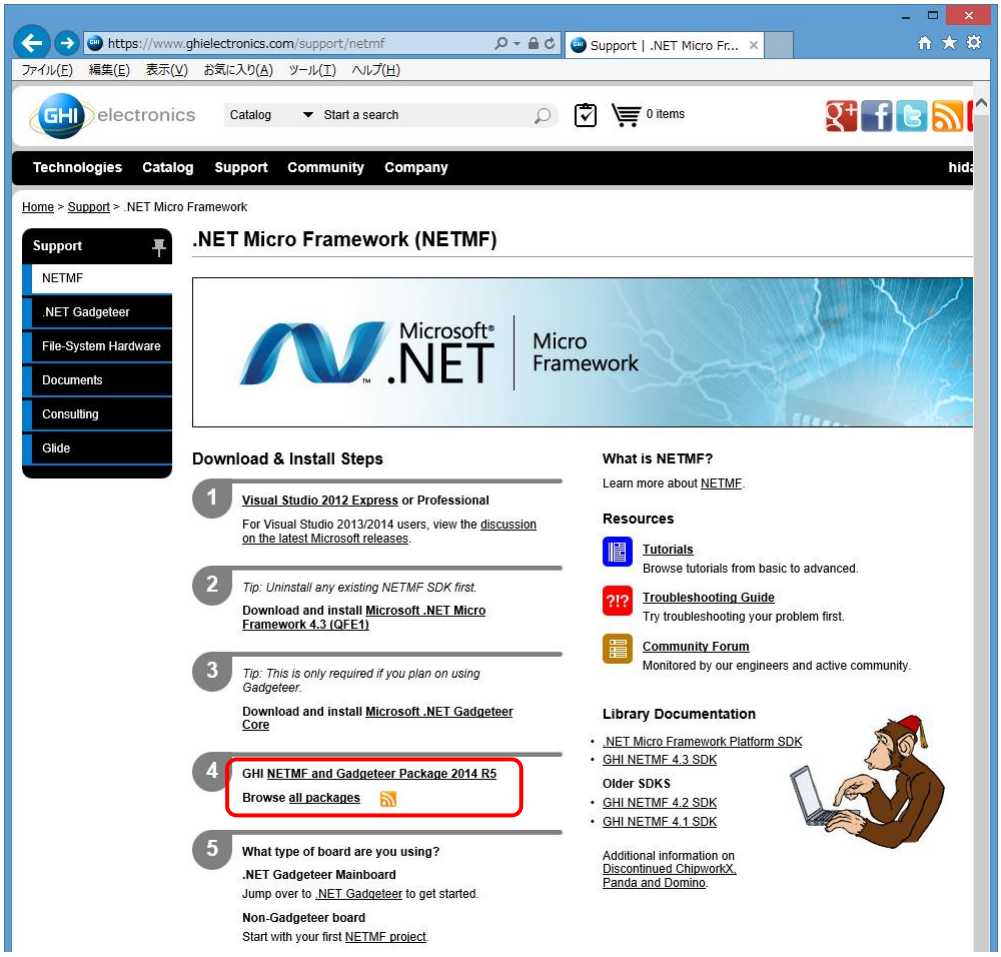
③ GHI NETMF SDK 2014 R5
https://www.ghielectronics.com/support/netmf/sdk/24/netmf-and-gadgeteer-package-2014-r5

■ヒント：各ソフトウェアの入手■

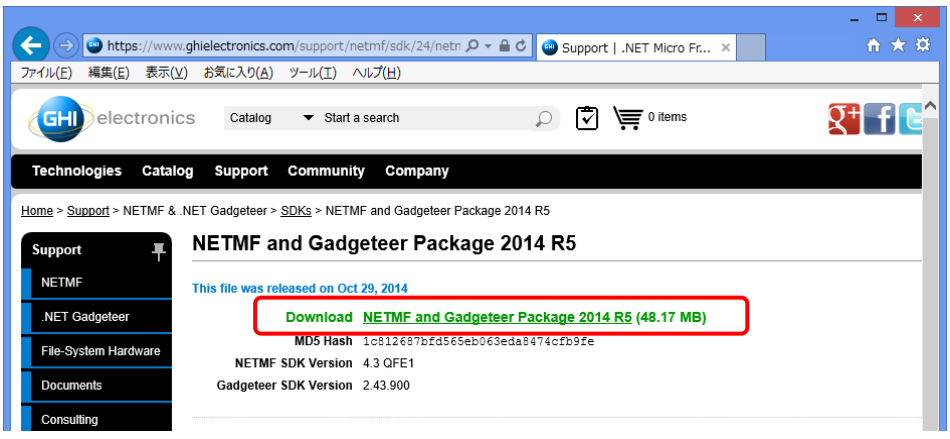
以下の GHI Electronics 社のホームページには入手先のリンクがインストール順に掲載されています。ただしここのリンクは現在のところ、英語版 Visual Studio 2012 と Microsoft .NET Micro Framework SDK 4.3 (QFE1)へのリンクとなっているので注意して下さい。

https://www.ghielectronics.com/support/netmf

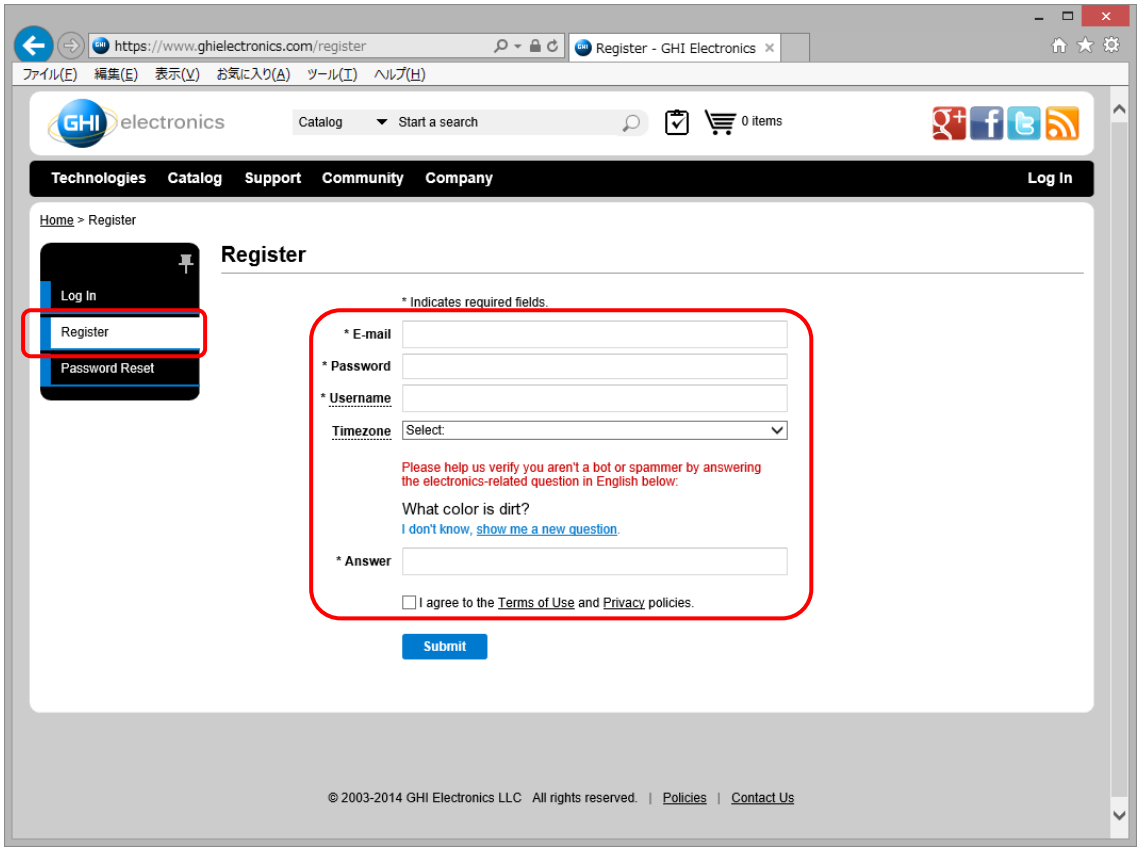
次ページに示す様に、.NET Micro Framework(NETMF)の画面の **Step 4** の NETMF and Gadgeteer Package 2014 R5 の部分をクリックするとダウンロード画面に移動します。



NETMF and Gadgeteer Package 2014 R5 のダウンロード画面では、**NETMF and Gadgeteer Package 2014 R5.exe (48.17 MB)** のファイル名をクリックします。



ファイル名をクリックするとログイン画面に移動するので、ユーザー未登録の場合には「Register」をクリックしてユーザー登録します。



英語の質問に答えてユーザー登録をした後は、そのままダウンロード画面に戻ってダウンロード可能です。

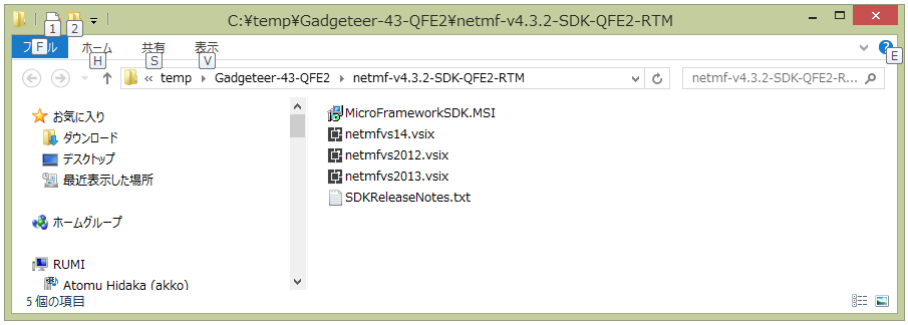
2.3. SDK パッケージのインストール

●演習４）

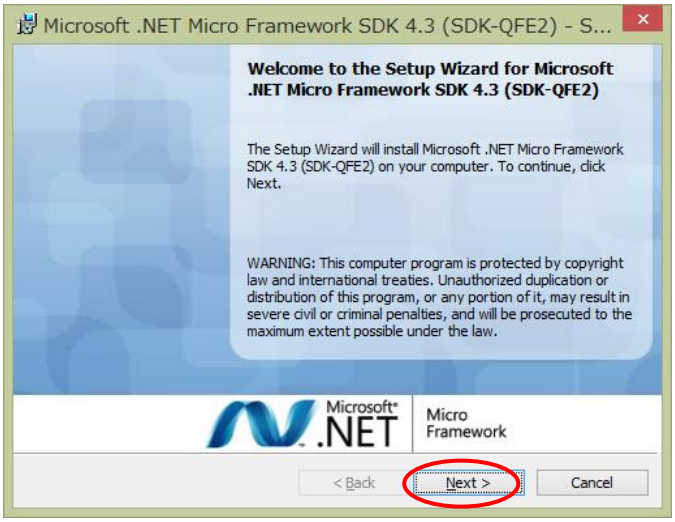
入手した SDK パッケージのインストールを順番に行います。

① Microsoft .NET Micro Framework SDK 4.3 (QFE2)

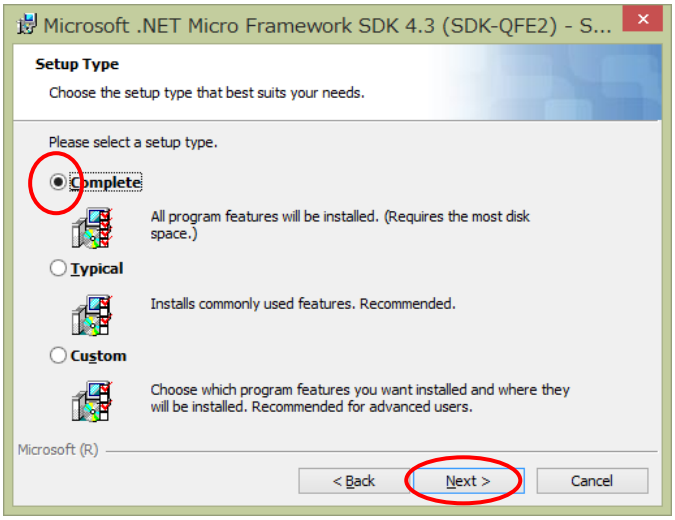
netmf-v4.3.2-SDK-QFE2-RTM.zip を適当なフォルダに展開すると次の様に、MicroFrameworkSDK.MSI ファイルと３種の vsix ファイルができます。vsix ファイルは各 Visual Studio 用のアドインです。

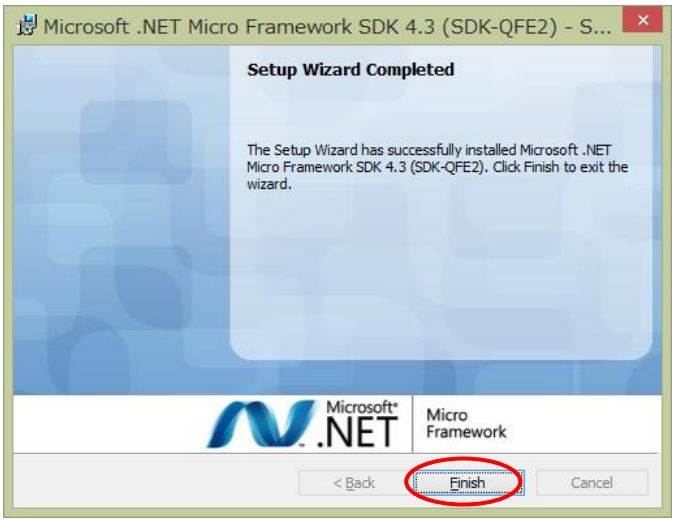


Visual Studio を終了しておいて、最初に MicroFrameworkSDK.MSI をダブルクリックして実行すると、Microsoft .NET Micro Framework SDK 4.3 (QFE2)のインストールが始まります。



ライセンス確認後、Setup Type の画面では「Complete」を選択します。

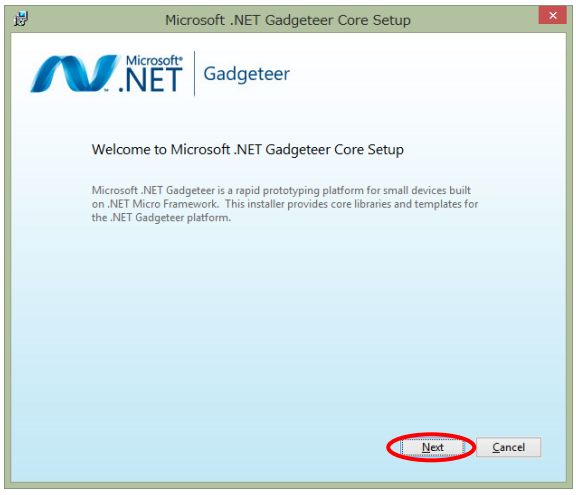




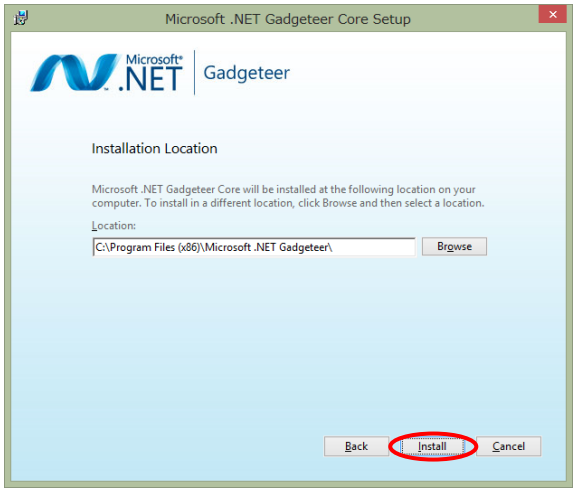
インストール完了後、Visual Studio を終了したまま、netmfvs2013.vsix ファイルをダブルクリックして、Visual Studio 2013 用アドインのインストールを行います。

② Microsoft Gadgeteer Core 2.43.1000

GadgeteerCore.msi.をダブルクリックして、NET Gadgeteer Core のインストールを行います。



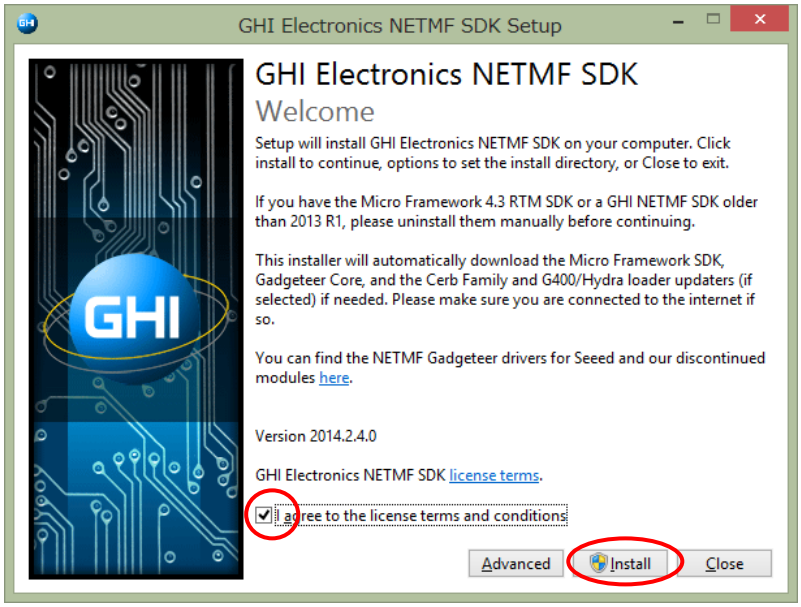
ライセンス確認後、インストール先ディレクトリはデフォルトのまま進みます。



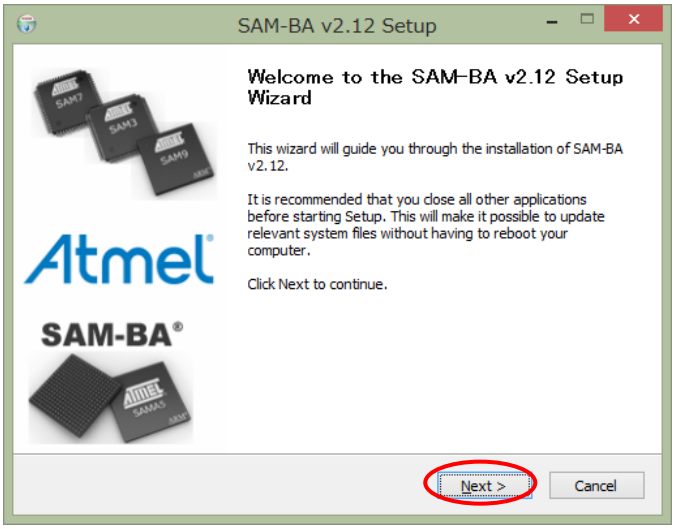
.NET Gadgeteer Core のインストールが完了しました。

③ GHI NETMF SDK 2014 R5

入手した NETMF and Gadgeteer Package 2015 R2.exe を起動して、表示される画面の利用条件にチェックを入れた後、install をクリックしてインストールを開始します。

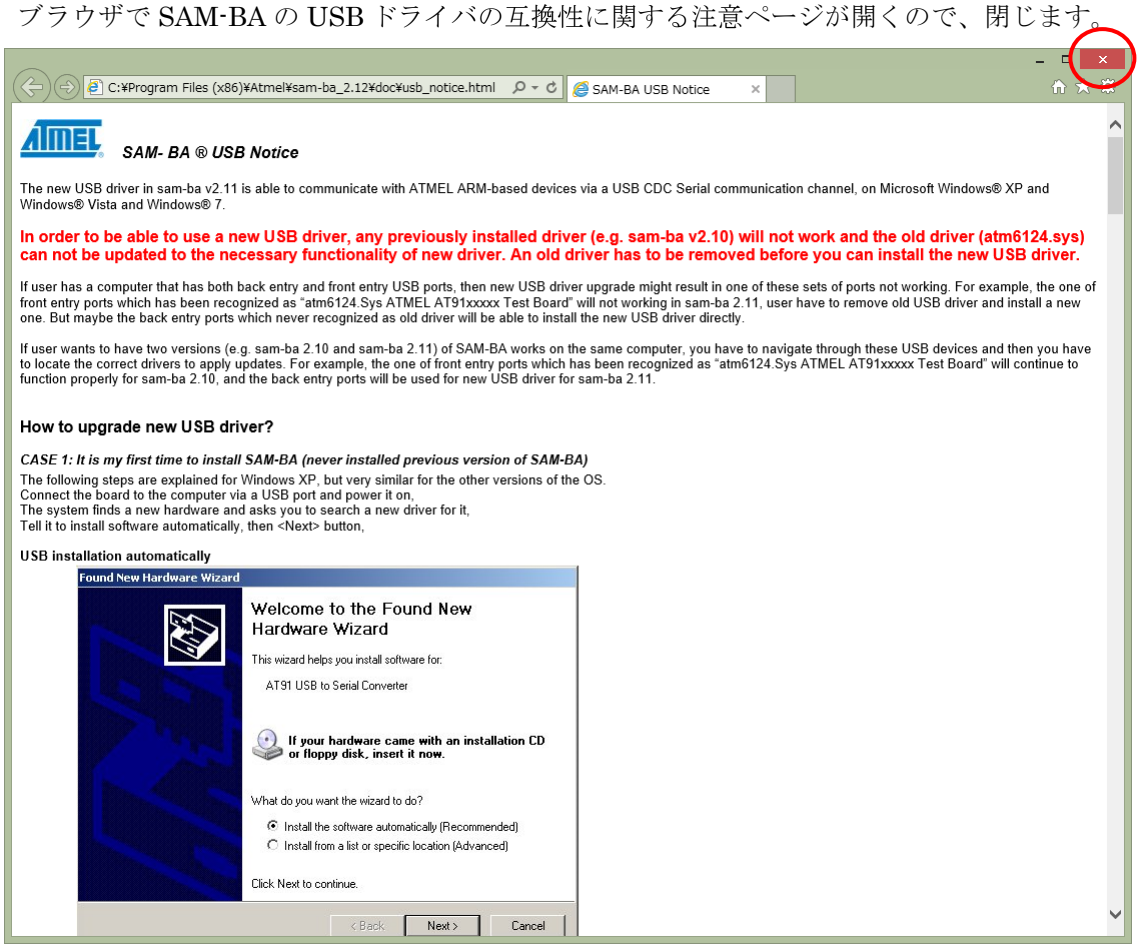


少しすると Atmel 社の Flash ROM メンテナンスツール(SAM-BA)のインストールダイアログが表示されるので、「Next」をクリックして進み、インストールを完了します。

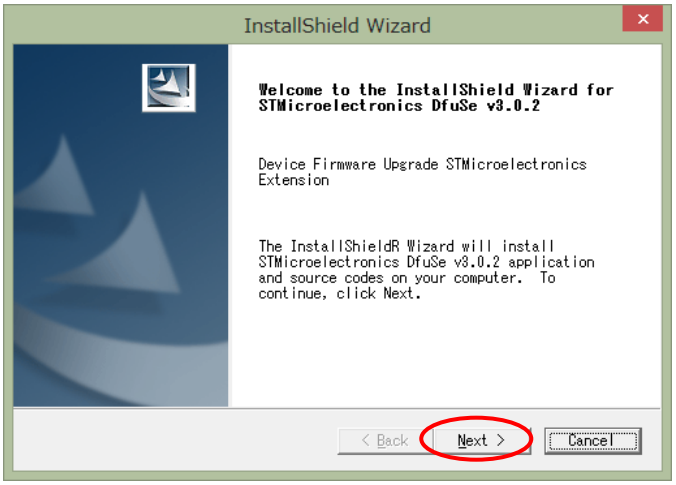


ライセンス確認、インストール先のディレクトリ確認、メニューフォルダー確認後、完了画面が出ます。

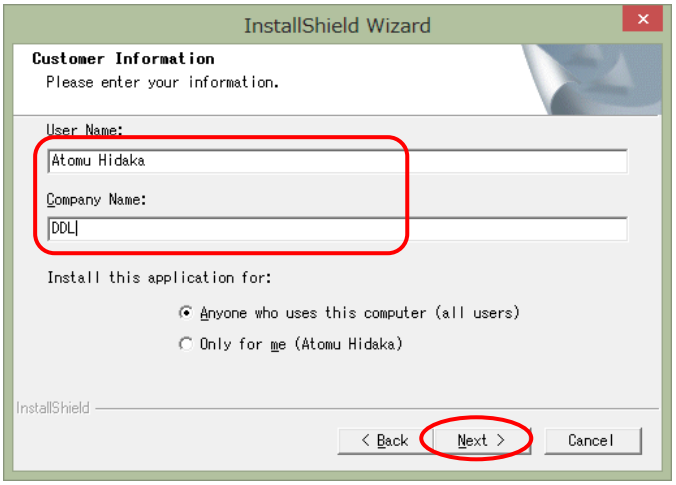
ブラウザで SAM-BA の USB ドライバの互換性に関する注意ページが開くので、閉じます。



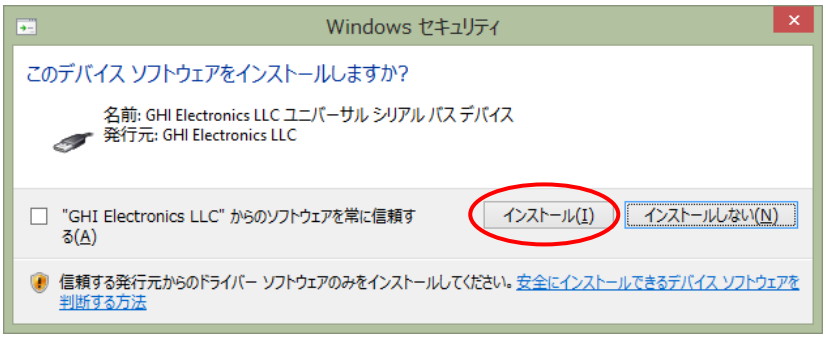
続いて STMicroelectronics 社の Flash ROM ツール(DfuSe)のインストールが自動的に始まります。



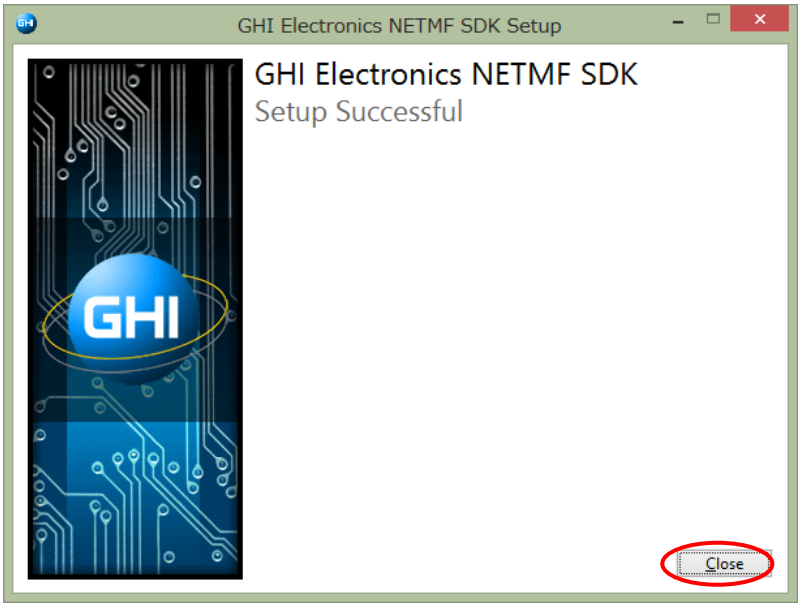
顧客情報入力のダイアログにはユーザー名と会社名を入力します。



インストール先のディレクトリ確認後インストールが進み、デバイスドライバのインストール確認ダイアログ画面が出るので、インストールをクリックします。



Close をクリックして GHI NETMF SDK のインストールを完了します。



3. ファームウェアの確認と更新

GHI electronics 社の FEZ シリーズを購入した場合、製品に最新のファームウェアがインストールされているとは限りません。.NET Micro Framework では、デバイスの Flash ROM に格納されているファームウェアを容易に更新できるのが特長のひとつでもあるので、常にファームウェアのバージョンを確認し、必要に応じて更新して行きます。

□解説：.NET Micro Framework のファームウェア構成□

.NET Micro Framework の基本ソフトウェアはブートローダである **TinyBooter** と、CLR インタプリタを含むオペレーティング・システムの **TinyCLR** から構成されます。これらは各ライブラリと密接に関係しているため、開発環境、使用するライブラリを含む開発対象アプリケーションとデバイスのファームウェアのバージョンを合わせる必要があります。

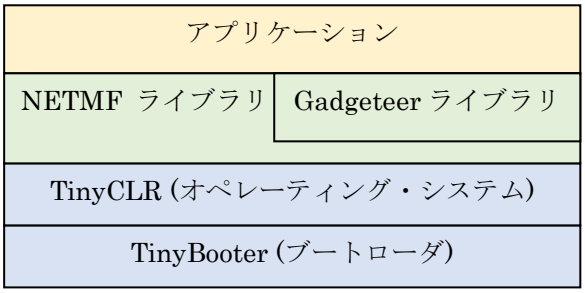


図　.NET Micro Framework / .NET Gadgeteer のソフトウェア構成

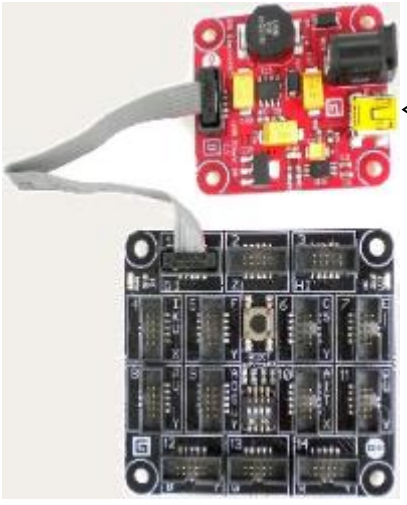
3.1. ファームウェアの確認

●演習５）

次の手順で FEZ Spider の現在のファームウェアのバージョンを確認します。

手順１：PC との接続

FEZ Spider メインボードの１番ソケット（ソケット D）に USB Client DP モジュールまたは、USB Client SP モジュールを接続後、USB ケーブルで PC の USB ソケットに接続します。

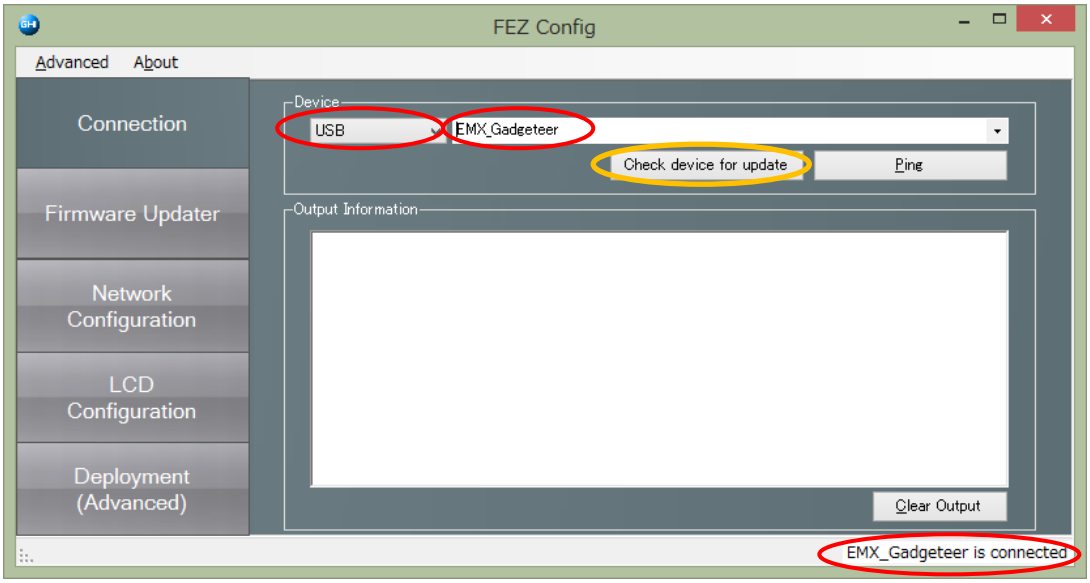


USB ケーブルで PC に接続します。

モジュールの消費電力が大きいとファームウェアの更新に問題が出る可能性があるので、他のモジュールは接続しないで下さい。

手順２：FEZ Spider の接続確認

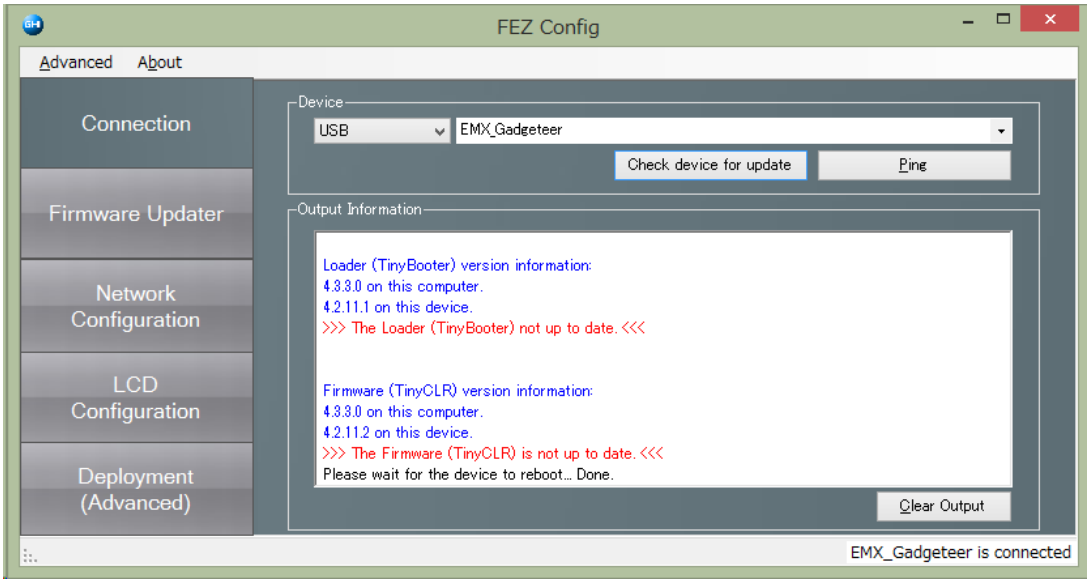
デスクトップ上に作成された FEZ Config のショートカットから FEZ Config を起動して接続を確認します。



接続先が USB EMX_Gadgeteer となっていて EMX_Gadgeteer is connected と表示されていれば OK です。問題がある場合にはケーブル接続とドライバのインストール状態を確認して下さい。

手順３：バージョンとアップデート確認

FEZ Config の Connection 画面で「Check device for update」をクリックすると画面の Output Information 領域にバージョン情報が表示されます。赤字で not up to date の項がある場合には、アップデートが必要ですので、内容に応じて次の各項の手順に進んで下さい。アップデートが不要の場合はアプリケーション開発演習に進んでください。



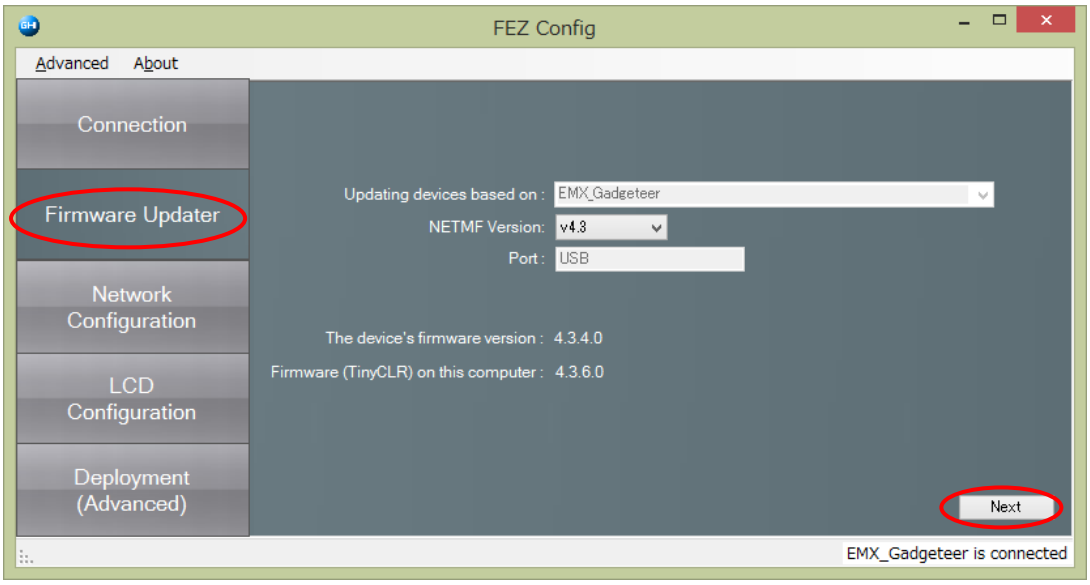
3.2.　TinyCLR だけの更新

>>> The Firmware (TinyCLR) is not up to date <<<

のメッセージが表示された場合、この手順ではオペレーティング・システム**(TinyCLR)** だけをアップデートします。ユーザーが転送したアプリケーション・プログラムは削除されるので、注意して下さい。

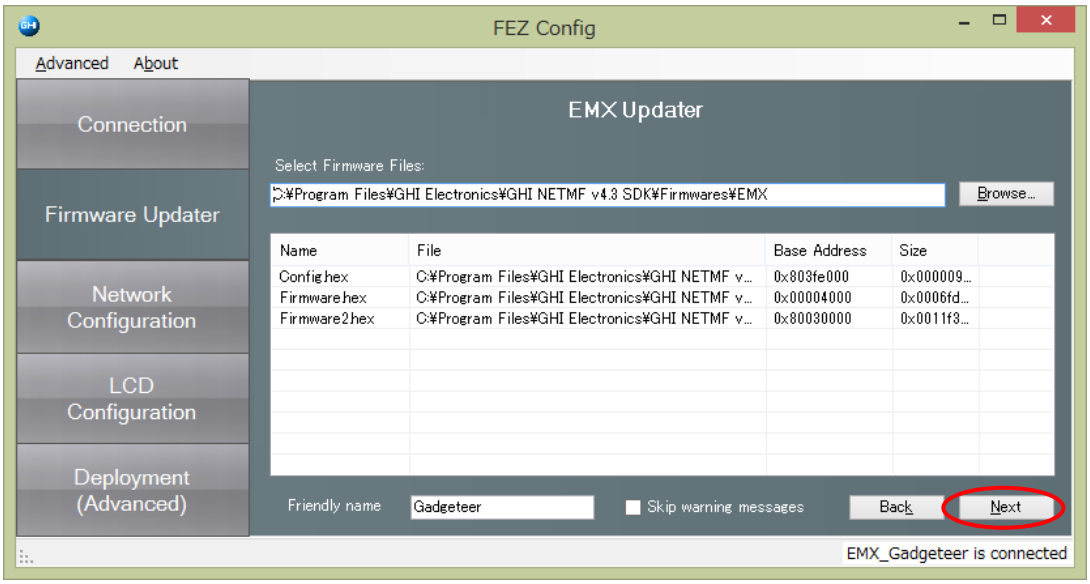
手順１：アップデートの起動

FEZ Config の右側タブ「Firmware Updater」をクリックして表示内容を確認し、「Next」をクリックします。

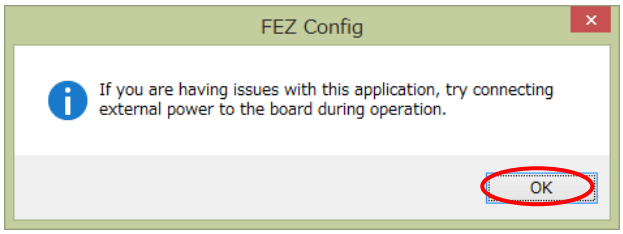


手順２：Update 実行

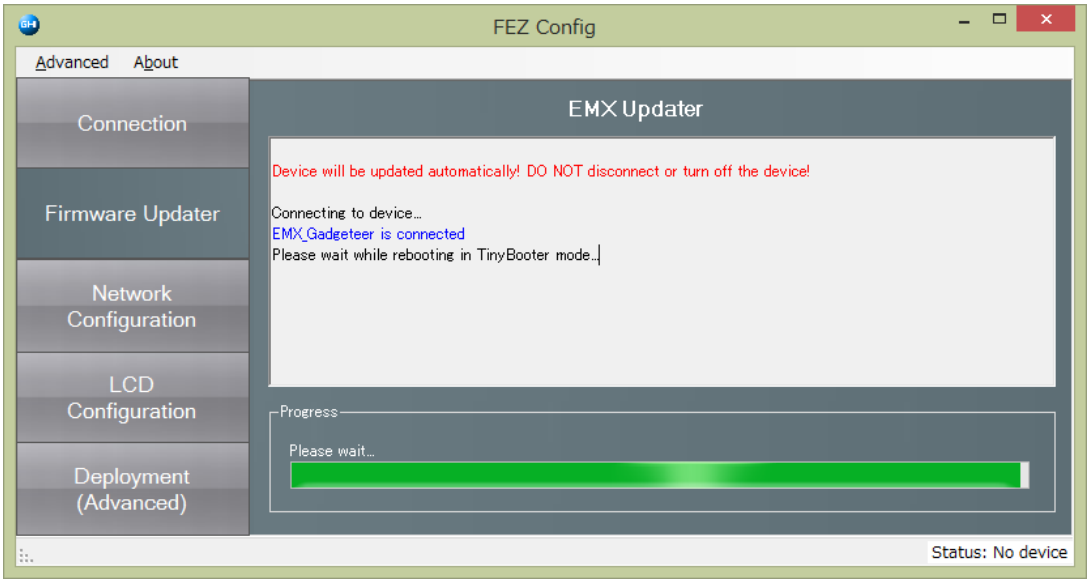
アップデートに使用するファームウェアのファイル名とアドレス情報が表示されます。画面の「Next」をクリックしてアップデートを開始します。



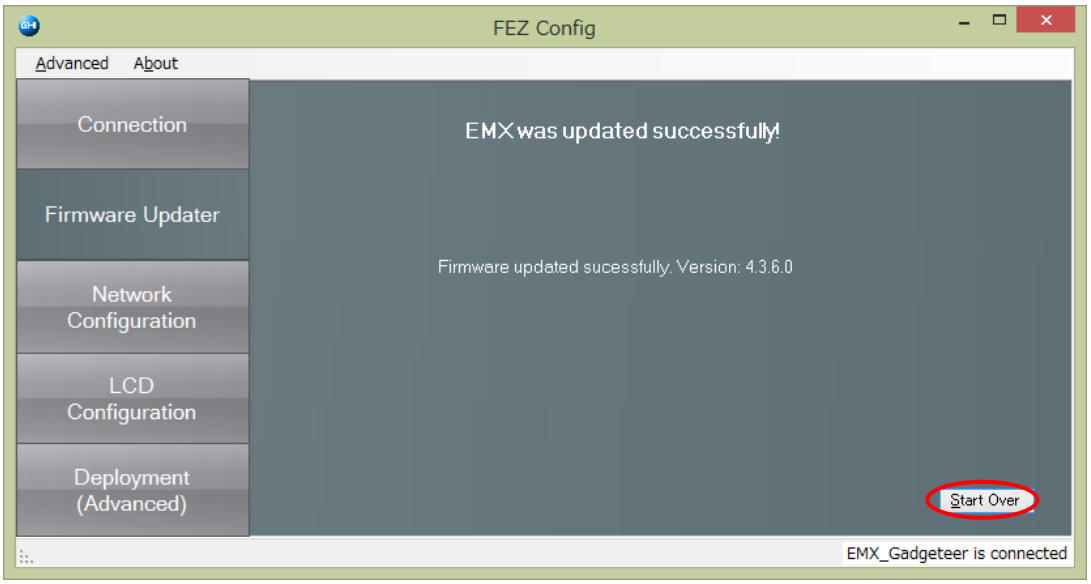
次に電源不足に関する警告メッセージが出ます。これは消費電力が大きいモジュールを接続したままアップデートを実行すると問題が発生する可能性があることを示すものです。メインボードに USB Client DP モジュールまたは SP モジュールを接続だけの状態では、問題ありませんので「OK」をクリックします。



以降、自動的にアップデートが進みます。



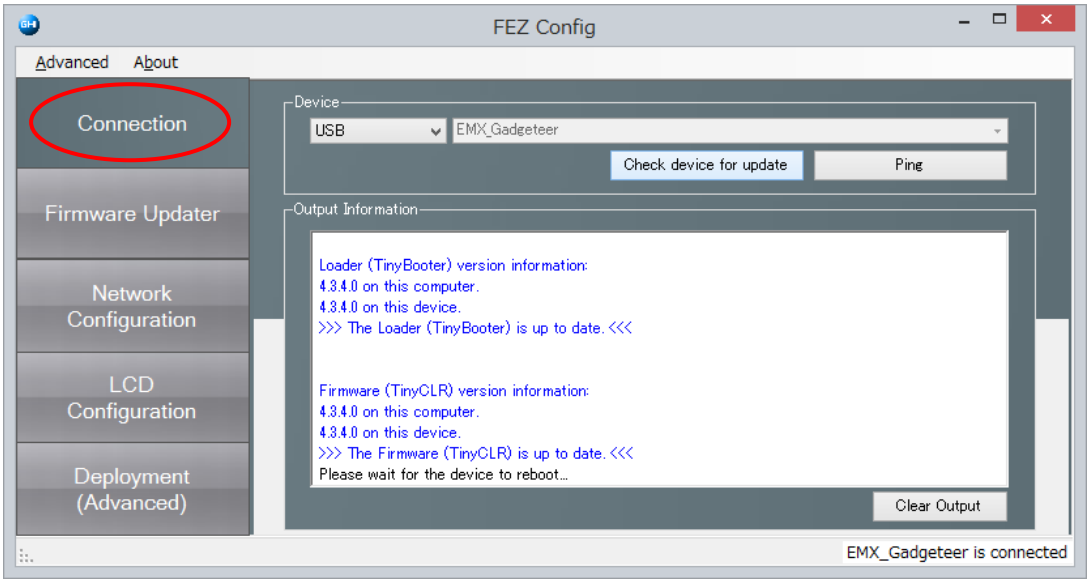
完了すると次の画面が表示されます。更新されたバージョン番号を確認して下さい。



手順３：再起動とバージョン確認

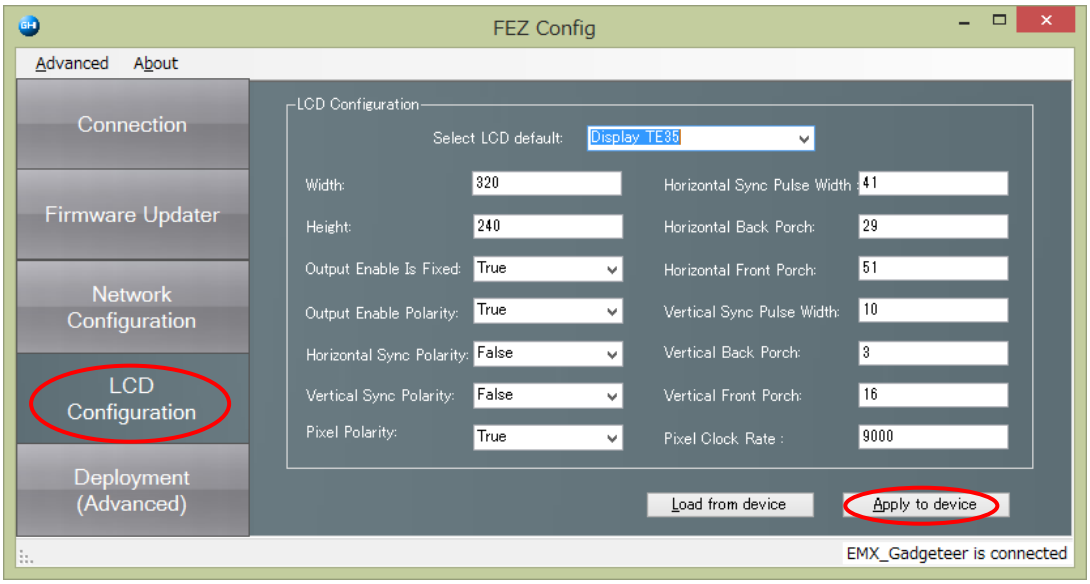
アップデート完了後必ず中央の「Reset」ボタンを押すか、電源を入れ直して、再起動しておきます。

左側「Connection」ボタンをクリックして元の画面に戻り、「Check device for update」をクリックして最新版に更新されたことを確認してください。



手順４： LCD 設定

TE35 等の **LCD** を接続して再起動し、画面表示が真っ白や乱れる場合には、LCD パラメーターの再設定を行います。右側「LCD Configuration」タブで設定画面を表示させて「Select LCD default:」で「Display TE35」等の使用する LCD 種類を設定し「Apply to device」で設定を完了します。



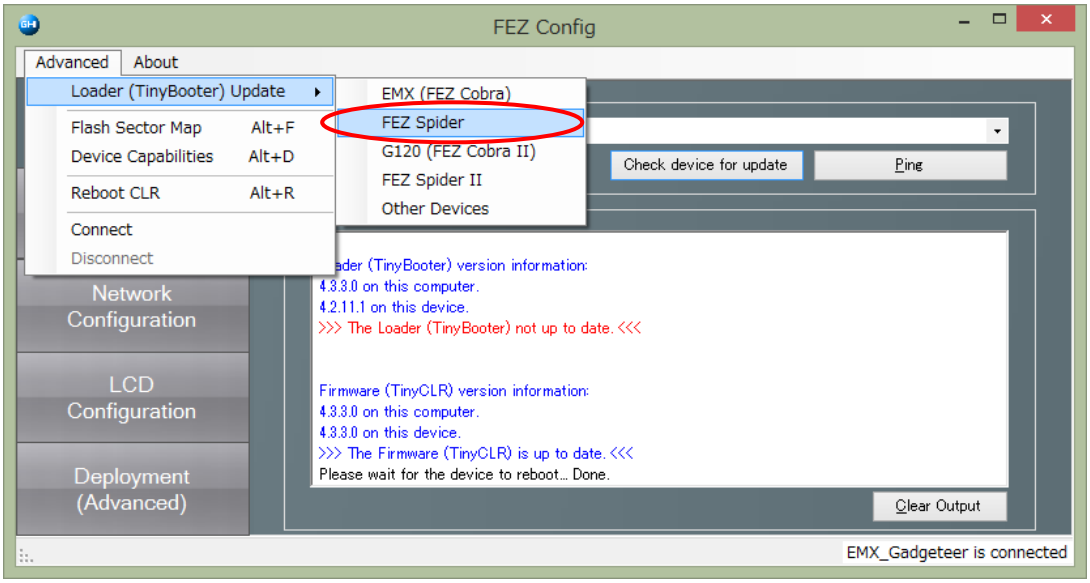
3.3. TinyBooter の更新

>>> The Loader (TinyBooter) is not up to date <<<

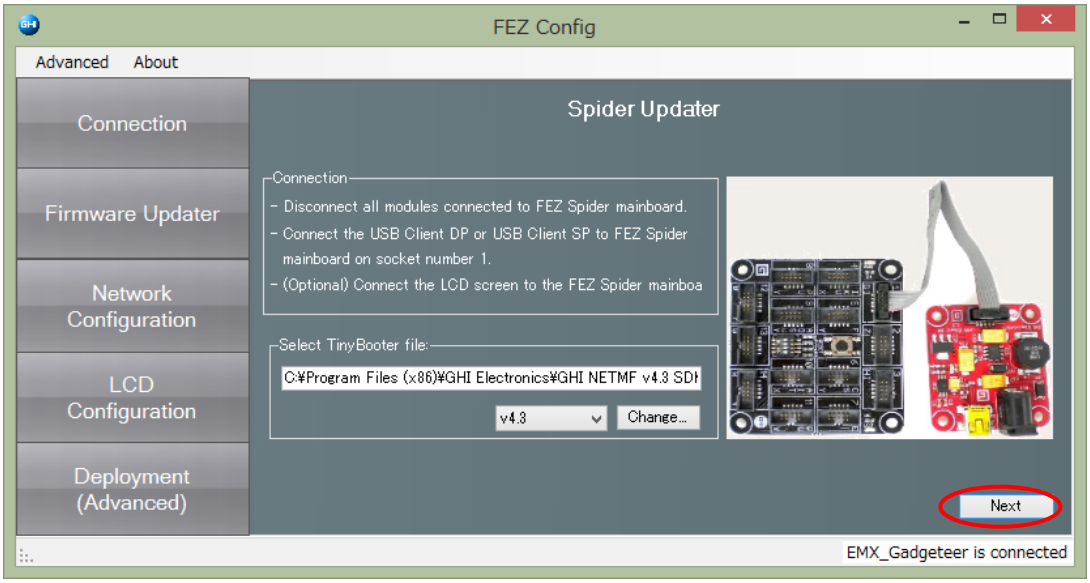
のメッセージが表示された場合、ブートローダ(**TinyBooter**) とオペレーティング・システム(**TinyCLR**) の両方をアップデートします。ブートローダだけをアップデートすることはできません。ブートローダの更新時は必ず **Advanced** メニューを使用して、ブートローダ(TinyBooter)と OS（TinyCLR）の両方をアップデートして下さい。

手順１：アップデートの起動

FEZ Config の上部メニューの Advanced から、Loader (TinyBooter) Update を選択して、サブメニューの FEZ Spider を選びます。

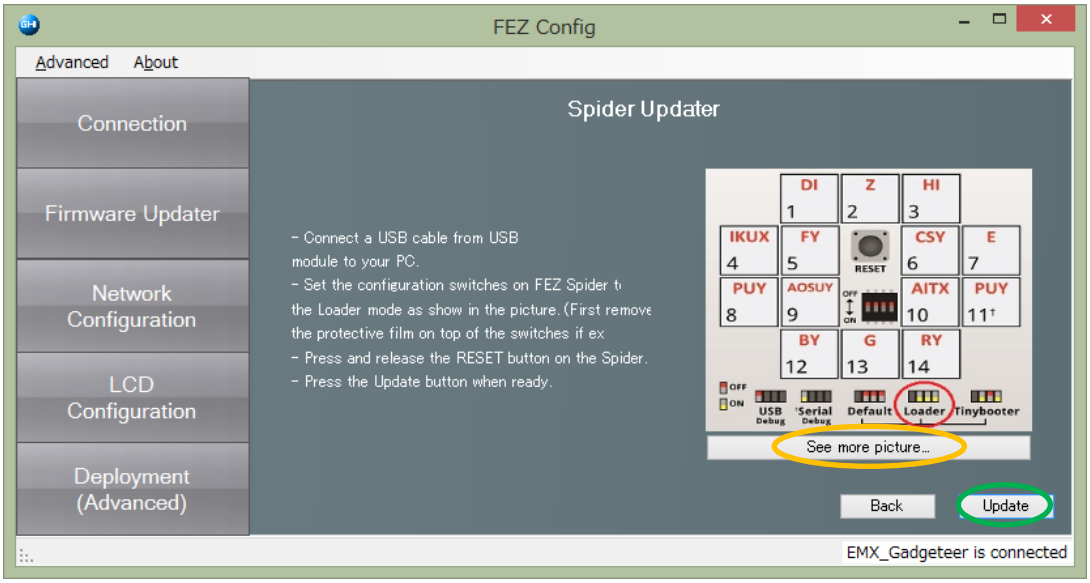


アップデート作業のガイド画面に変わるので「Next」をクリックします。



手順２：ブートローダモードでの再起動

「Next」をクリックして画面を次に切り替えてディップスイッチの設定を確認します。



FEZ Spider メインボード中央部のディッチをブートローダモードに(Loader)に切り替えます。

ブートローダモードはディップスイッチの右から３つ（1, 2, 3）を ON（下）に切り替えます。

ディップスイッチは小さいのでシャープペンやボールペンやの先などを使用して設定します。

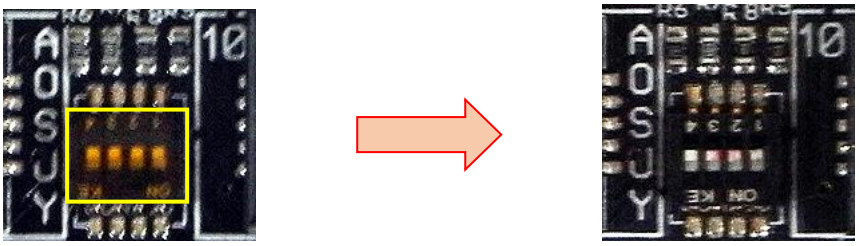
画面の「See more picture」でガイド画面を切り替えて確認できます。

切り替え後は必ず中央の「Reset」ボタンを押して、再起動しておきます。その後「Update」をクリックして下さい。

■ヒント：ディップスイッチ■

ディップスイッチの部品は上下反対についているので注意して下さい。

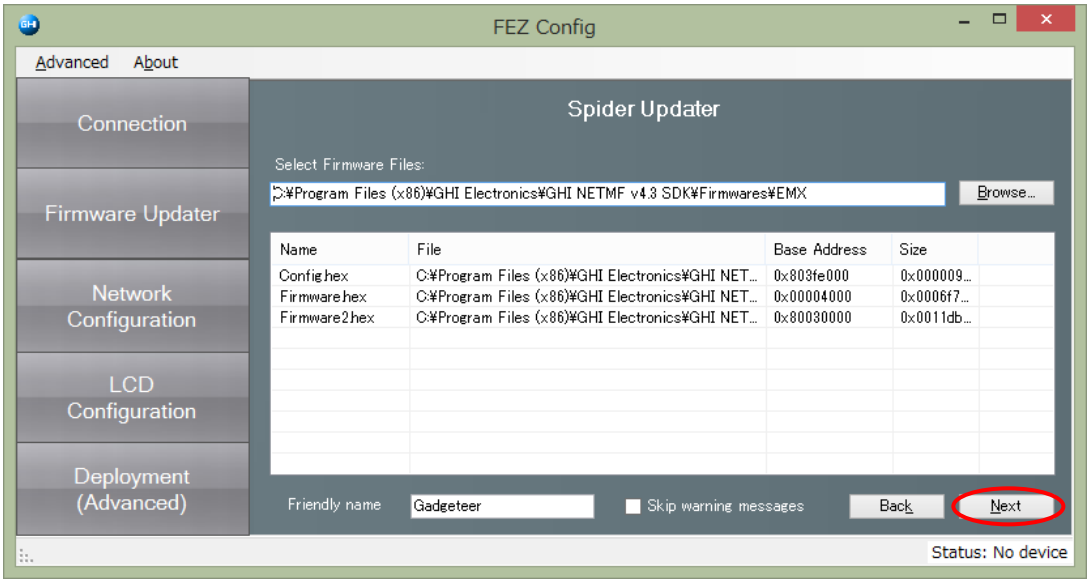
購入直後は、茶色の透明で薄いビニールでスイッチがカバーされている場合があります。そのような場合はカバーを剥がして下さい。



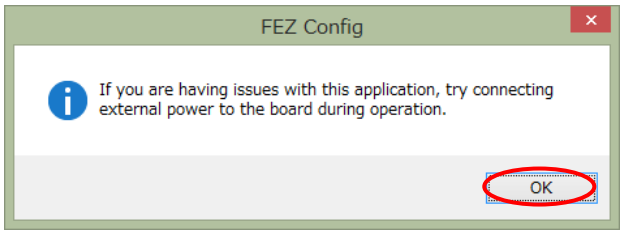
ディップスイッチは次の様に各スイッチのポジションによって役割が決まっています。

手順３：Update 実行

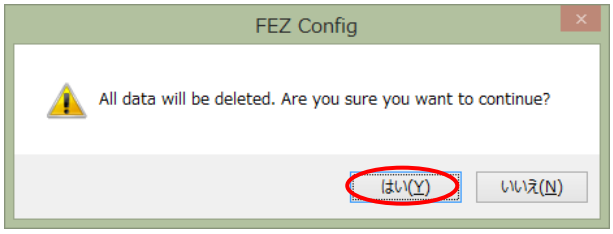
アップデートに使用するファームウェアのファイル名とアドレス情報が表示されます。画面の「Next」をクリックしてアップデートを開始します。



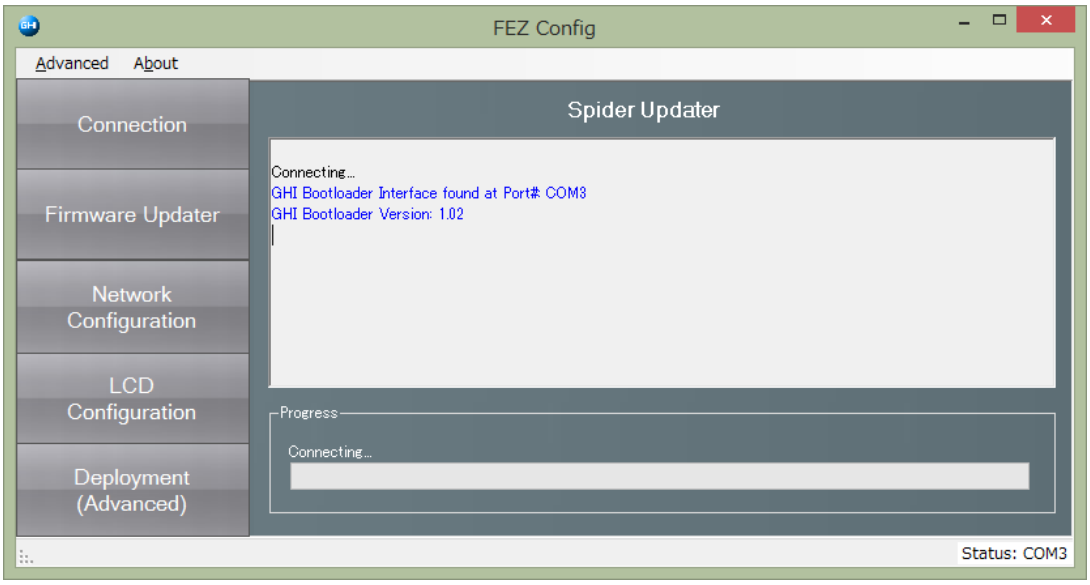
次に電源不足に関する警告メッセージが出ます。これは消費電力が大きいモジュールを接続したままアップデートを実行すると問題が発生する可能性があることを示すものです。メインボードに USB Client DP モジュールまたは SP モジュールを接続だけの状態では、問題ありませんので「OK」をクリックします。



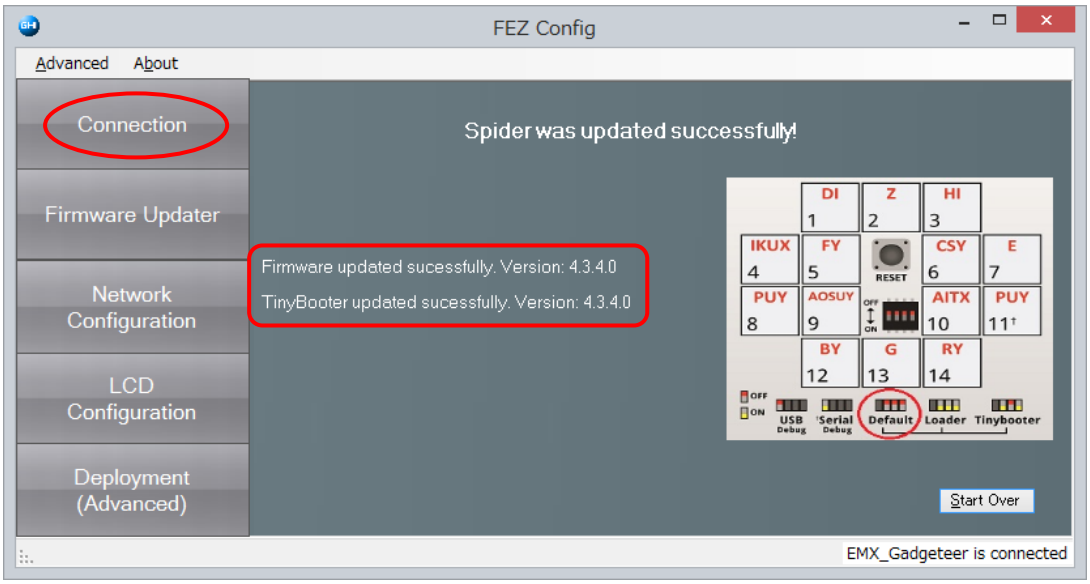
Flash ROM 内のデータとアプリケーションが全て削除されることに関する確認メッセージです。「はい」をクリックします。



以降、自動的にアップデートが進みます。



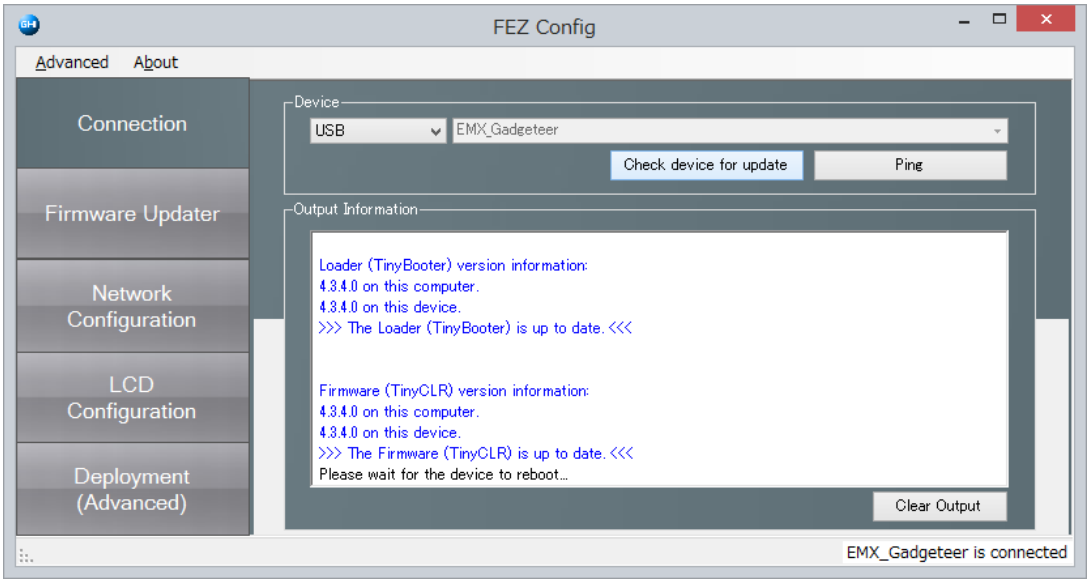
完了すると次の画面が表示されます。更新されたバージョン番号を確認して下さい。



手順４：再起動と確認

アップデート完了後必ずメインボードのディップスイッチの右から３つ(1, 2, 3)を OFF(上)に切り替えて Default モードに設定後、中央の「Reset」ボタンを押して、再起動しておきます。

左側「Connection」ボタンをクリックして元の画面に戻り、「Check device for update」をクリックして最新版に更新されたことを確認してください。



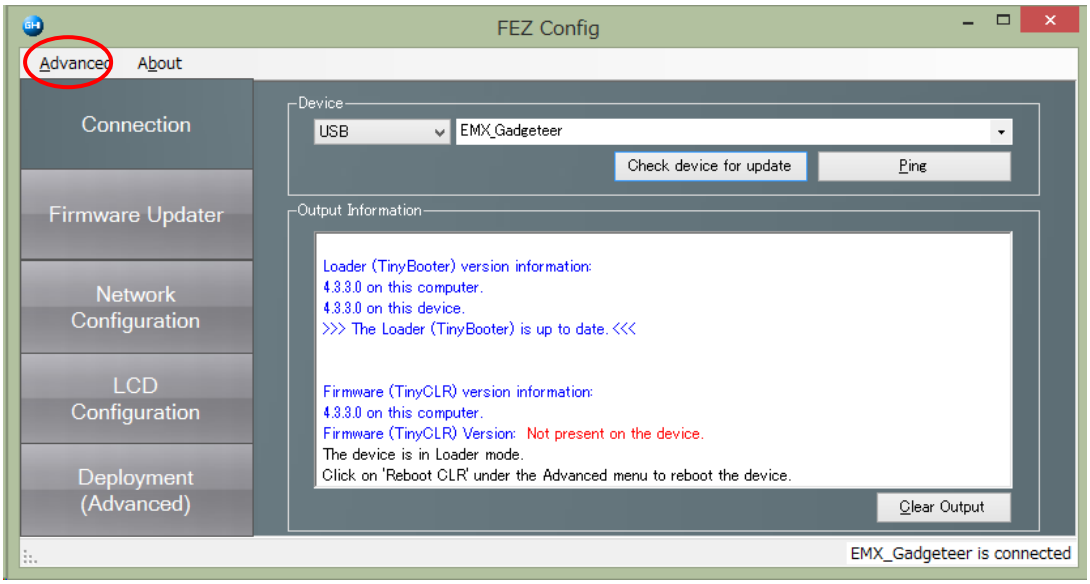
□解説：ファームウェアとアプリケーションのバージョン整合□

デバイスのファームウェア(TinyCLR)とアプリケーションのメイン・バージョンは一致している必要がありますが、ブートローダは互換性があります。従って最新のブートローダを搭載しておけば、一世代前の V4.2 のファームウェア(TinyCLR)を搭載することも可能です。

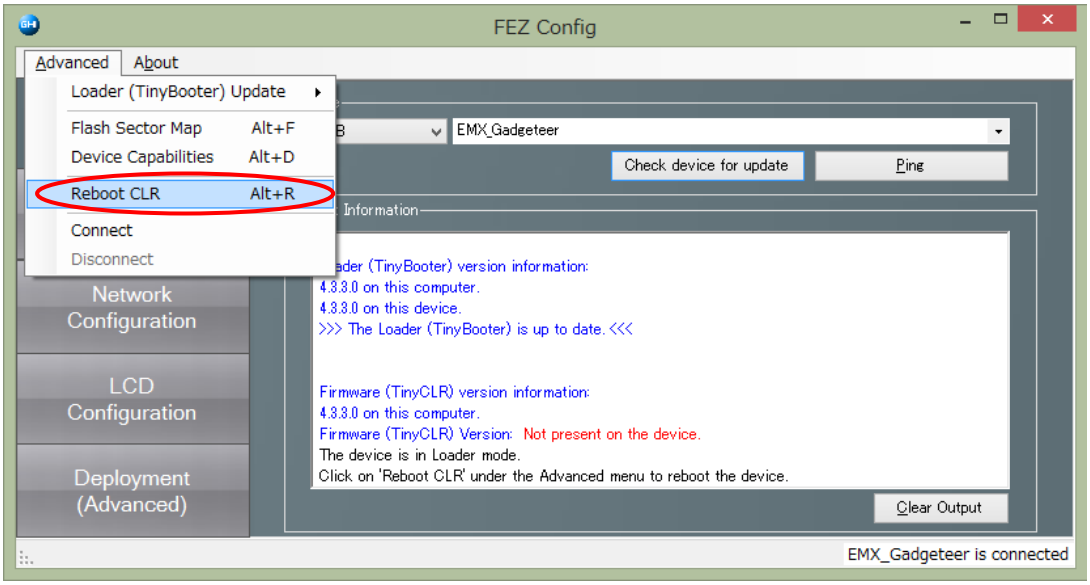
しかしその場合は開発環境（ライブラリ）の設定も V4.2 にしてアプリケーション開発する必要があります。

■ヒント：TinyCLR の再ロード■

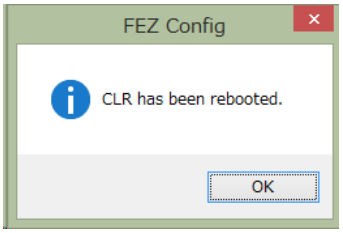
システムリセット直後等 TinyCLR の起動に時間がかかり、TinyCLR がまだロードされていないという趣旨で次のメッセージが表示される場合があります。その場合にはしばらく待って再実行するか、あるいは「Advanced」メニューから強制ロードして確認することも可能です。



「Reboot CLR」を選択すると、TinyCLR の再ロードを行います。



「CLR has been rebooted.」の表示で再ロード完了です。アップデートチェックが可能です。



4.　.NETMF アプリケーション開発の基本

4.1.　VC#利用前の準備

○演習６）

解説の手順に従って Visual Studio 2013 または Visual C# 2013 Express の利用前の設定をします。

□解説：VC# 2013 express 利用前の準備□

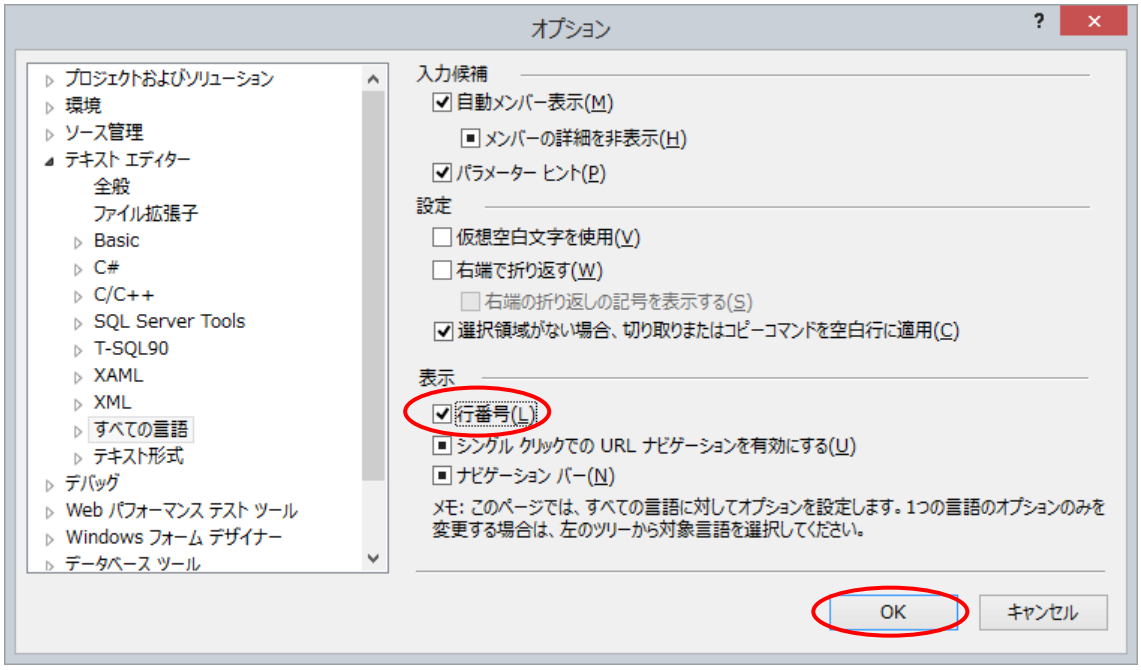
手順１：ベース・ディレクトリの作成

Visual Studio がデフォルトで作成するフォルダの場所は、フルパスの名前が長くなるために扱いがしにくく、また後からソースコード・ライブラリを追加する際にしばしばエラーとなります。このような問題に対応するため、今回の演習では全て C:¥NETMF というフォルダを作成して、そこで作業します。

最初の手順ではプロジェクト作成前に、エクスプローラーを使用して **C: ドライブ直下**に、Visual Studio ソースコード管理様の NETMF ディレクトリを作成します。

手順２：行番号の設定

次ページの様に「ツール」メニュー→「オプション」を開いて「テキスト エディター」の項、「すべての言語」の設定画面を開いて「行番号」の表示を有効にします。



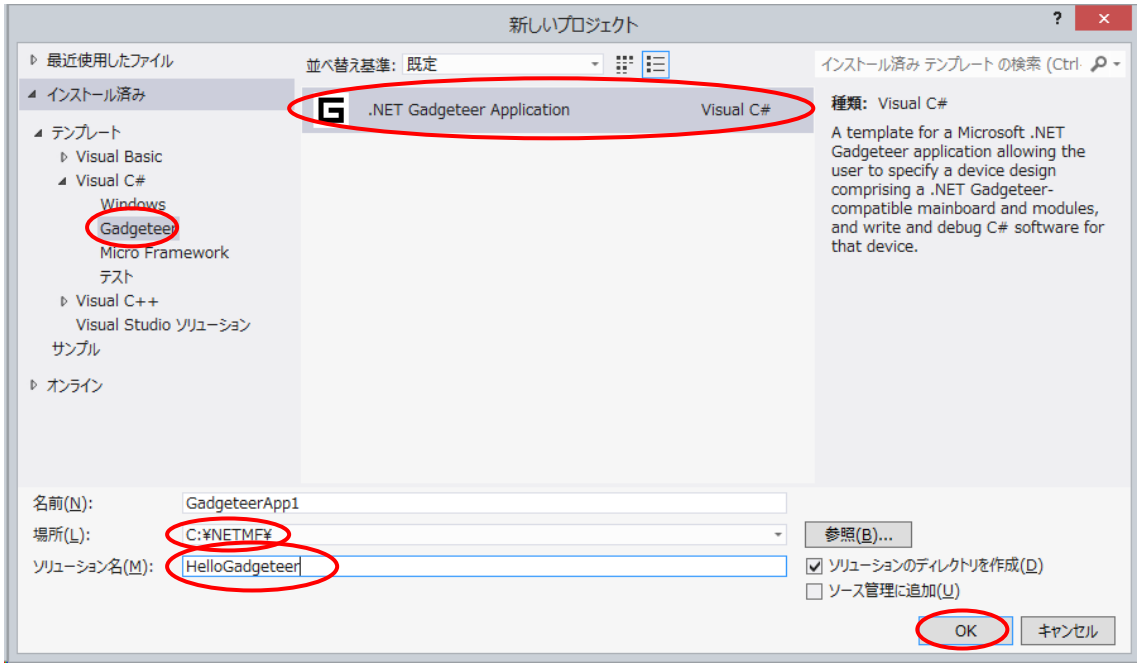
4.2.　単純なアプリケーションの開発

○演習７）

解説の手順に従って Visual Studio を使用して単純な.NET Micro Framework (以降 NETMF と表記)アプリケーション・プロジェクトを作成してビルドします。

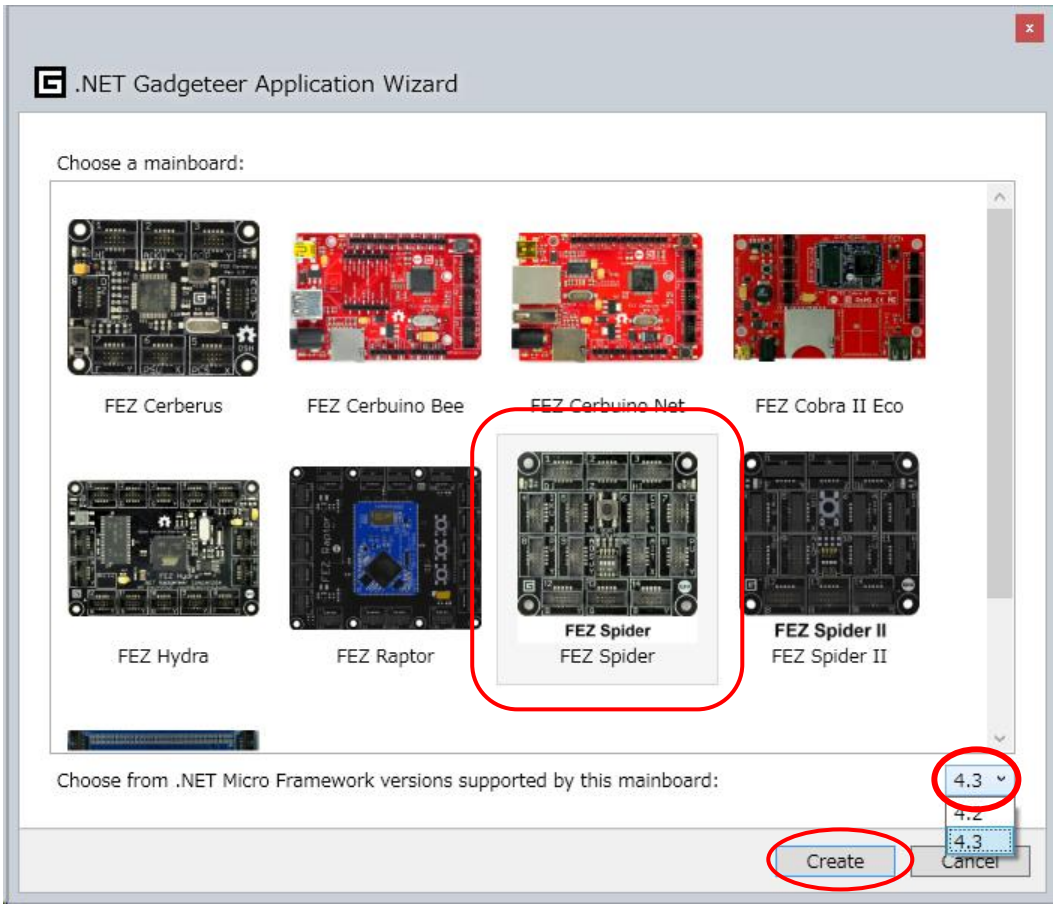
手順１：プロジェクトの作成とメインボード選択

Visual Studio を立ち上げて、新規に Visual C#コンソール・アプリケーションのソリューションを「HelloGadgeteer」という名前で作成します。作成するディレクトリは、C:¥NETMF 以下と指定します。



「OK」をクリックすると.NET Gadgeteer Application Wizard のメインボードの選択画面に移るので、「FEZ Spider」を選択します。

またこの画面では、開発で使用する.NET Micro Framework のバージョンを、右下のプルダウン・メニューで「4.3」に設定します。**デフォルトのバージョンが「4.2」となっているので、この「4.3」への変更は必須の作業ですから注意して下さい。**（次ページ参照）



手順２：モジュールの追加と配線

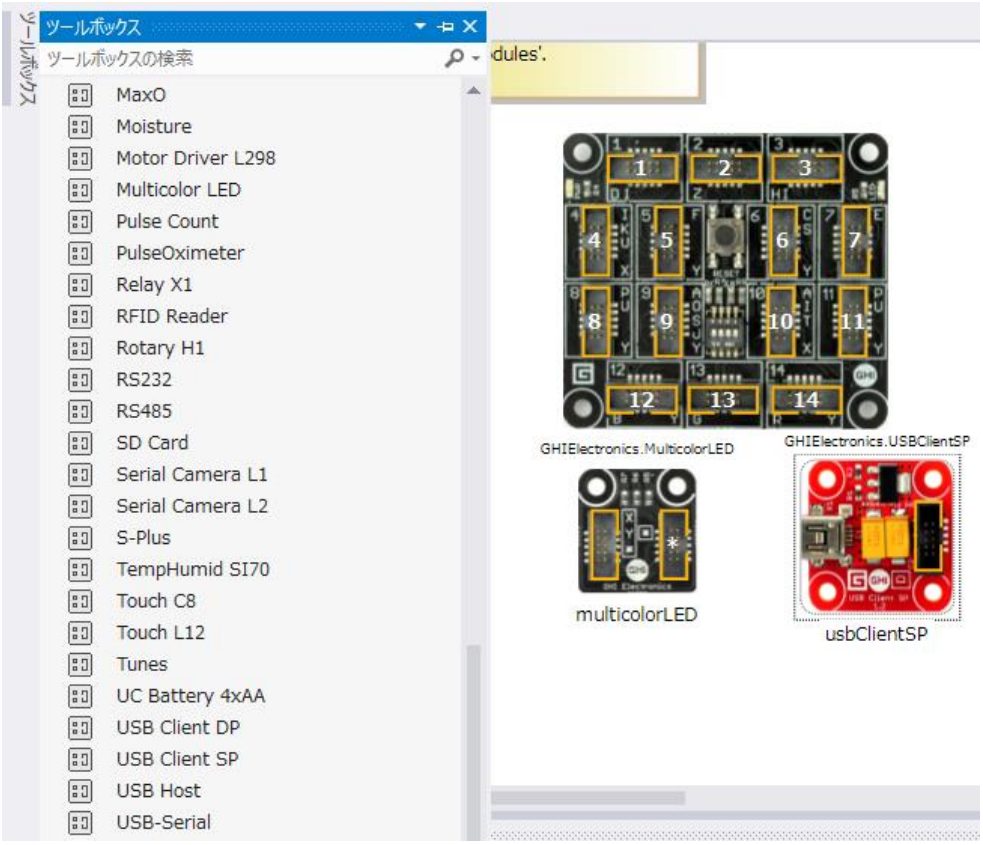
ここでは Gadgeteer のキャンバス上に部品を並べて配線する、モジュールの追加と配線作業を行います。この作業を完了することで、アプリケーションが使用するドライバの選択と初期化ルーチンの設定、SoC (System on a Chip)のピンアサインが行われます。

□解説：SoC のピンアサイン□

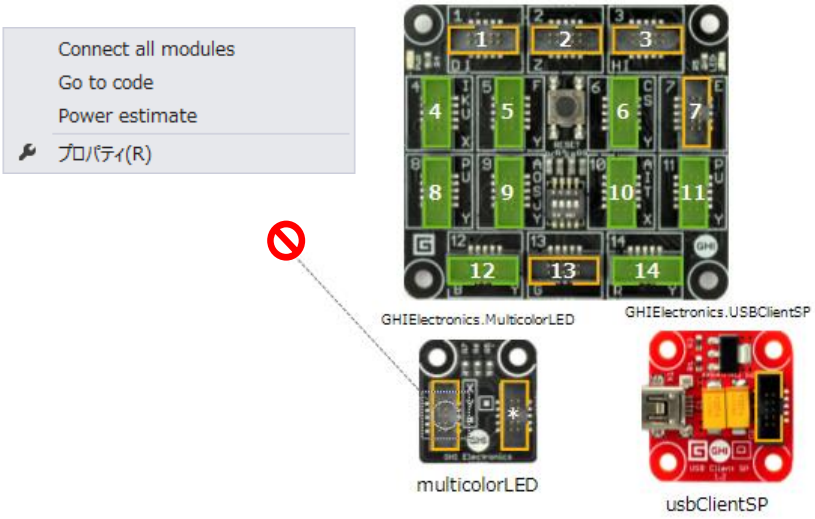
SoC は CPU チップに多数の周辺コントローラを搭載したチップです。SoC は今では組み込みシステムでは必須のチップですが、限られた数のピンに機能を様々なコントローラ機能を多重化して割り当てているため、使用する機能に応じて初期化時に面倒な設定が必要になります。.NET Gadgeteer ではこの画面の作業で自動的に行います。

次の操作は、慣れるとペイントツールの様に簡単に操作できるので、自由に試して下さい。

① ツールボックスを開いて、「Multicolor LED」と「USB Client SP」または「USB Client DP」をキャンバスにドラッグします。
配置したモジュールはマウスを使用してキャンバス内で自由に移動することが可能です。また「Delete」キーや右クリック・メニューで削除することも可能です。

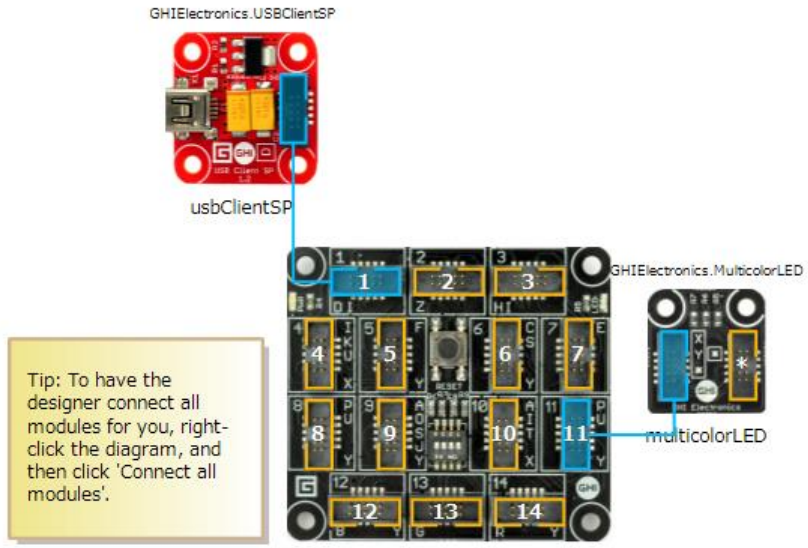


② 配線を行います。配線はモジュールのソケットをマウスでクリックして引き出すことで、接続可能なメインボードの場所が緑色に変わるので、そこに接続することが可能です。またキャンバスの何もない場所で右クリック・メニューから「Connect all modules」を選択すると自動的に未配線の配線を行います。結線した配線はマウス選択後、右クリックや「Delete」キーで削除可能です。どこに配線しても物理配線を一致させれば動作可能です。



③ 配線完了

以下に完了した配置と配線の図を示します。**配線完了後は必ず Visual Studio を操作して「すべてを保存」の操作を 2 回行って下さい。**



■ヒント：電源モジュールの互換性■

電源供給モジュールである「USB Client SP」と「USB Client DP」は互換性があるため、どちらを使用しても、あるいは両方とも画面上で接続しなくてもシステムは動作するのですが、作業した設定を記録するためにも、使用するモジュールに合わせて配置することをお勧めします。

「USB Client DP」は大容量の 12V(7V～30V) の外部電源でも動作可能なため、ディスプレイ等の消費電力が大きいモジュールを接続する場合や USB 供給電源が弱い場合に利用します。

手順３：コーディング

ここまでの手順で自動作成されたプロジェクトとソースコードのテンプレートを編集して FEZ Spider で動作するアプリケーション・プログラムを作成します。

C:¥NETMF¥ HelloGadgeteer 以下に「HelloGadgeteer.sln」というソリューションファイルと、C:¥NETMF¥HelloGadgeteer¥HelloGadgeteer 以下に「Program.cs」C#ソースコードが作成されています。これを少し追加修正して簡単なアプリケーションを作成します。

① スレッド起動の追加

ProgramStarted()のアプリケーションの主処理である BlinkEver() メソッドのスレッドを作成して呼び出す次の処理を ProgramStarted()内、
Debug.Print("Program Started");の前に追加します。

```
Thread startThread = new Thread(BlinkEver);
startThread.Start();
```

② BlinkEver() メソッドの追加

ProgramStarted() メソッドの直後に、アプリケーションでの主処理である LED 点滅を永久に実行する、以下の BlinkEver()メソッドを追加します。

```
void WaitEver()
{
    while (true)
    {
        multicolorLED.TurnGreen();
        Thread.Sleep(1 * 1000);
        multicolorLED.TurnOff();
        Thread.Sleep(1 * 1000);
        Debug.Print("LED Loop");
    }
}
```

追加部分全体を示します。

```
30              If you want to do something periodically, use a GT.Tim
31                  GT.Timer timer = new GT.Timer(1000); // every seco
32                  timer.Tick +=<tab><tab>
33                  timer.Start();
34              ******************************************************
35
36              Thread startThread = new Thread(BlinkEver);
37
38              startThread.Start();
39
40              // Use Debug.Print to show messages in Visual Studio's
41              Debug.Print("Program Started");
42          }
43
44          void BlinkEver()
45          {
46              while (true)
47              {
48                  multicolorLED.TurnGreen();
49                  Thread.Sleep(1 * 1000);
50                  multicolorLED.TurnOff();
51                  Thread.Sleep(1 * 1000);
52                  Debug.Print("LED Loop");
53              }
54          }
55      }
56  }
57
```

■ヒント：Thread.Sleep■

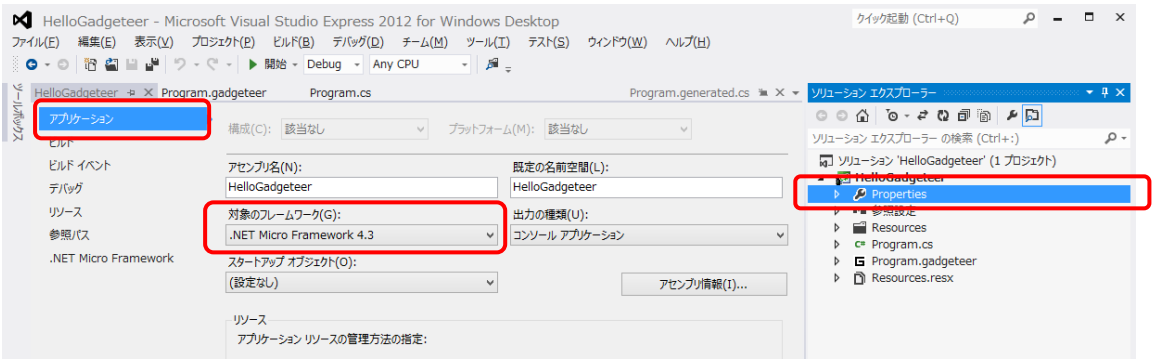
Thread.Sleep はパラメーターで指定した時間（ミリ秒）だけスリープするというメソッドです。このように多くのクラス、メソッドは.NET Framework の C#と互換性があります。

手順４：保存とビルド

追加のコーディングが終わったらば、全てのファイルを保存後、プロジェクトのプロパティを確認します。

ソリューション エクスプローラーで Properties を開くか、プロジェクト HelloGadgeteer を右クリックしてプロパティを開き、アプリケーション・タブで「対象のフレームワーク」が「.NET Micro Framework 4.3」となっていることを確認します。ここの表示バージョンが「4.2」の場合に、そのままでは実機で動作させることができませんので、プロジェクトを作り直してメインボードの選択時に「4.3」を選択してやり直します。

その後ビルドメニューからビルドを行います。ビルドにエラーが無い事を確認した後は、次の演習に進んで実機で動作させます。



4.3.　単純なアプリケーションの動作確認

○演習８）

次の手順に従ってメインボードとモジュールを接続し、実際に動作させて確認します。

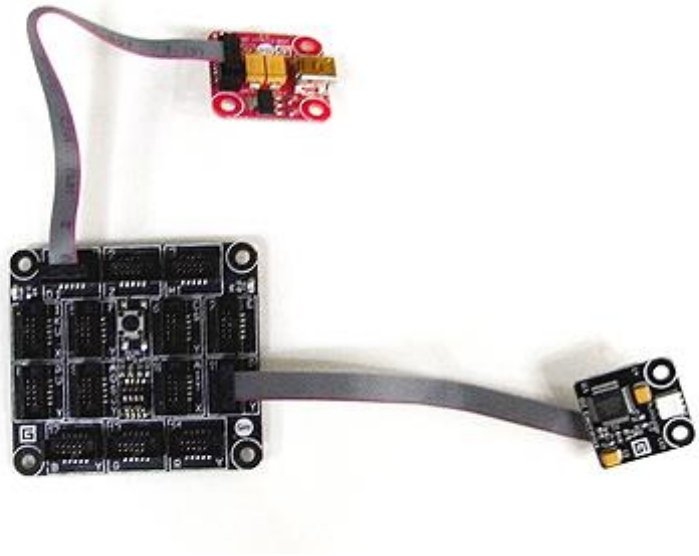
手順１：組立

Visual Studio の「Program.gadgeteer」タブを開いて、前項で配置・配線した画面に従ってモジュールとメインボードのソケットを配線します。

□解説：.NET Gadgeteer のソケット□

メインボードの各ソケットは 1～14 のソケット番号のほか、各ソケットが対応可能な機能（バスの仕様）に従って、A～Z のアルファベットが付けられています。メインボード上のソケットに何種類かのアルファベットがプリントされているソケットは複数機能の中から選択して利用できることを示します。

次のページに実際の接続写真を載せます。キャンバスの画像と比べてみて下さい。

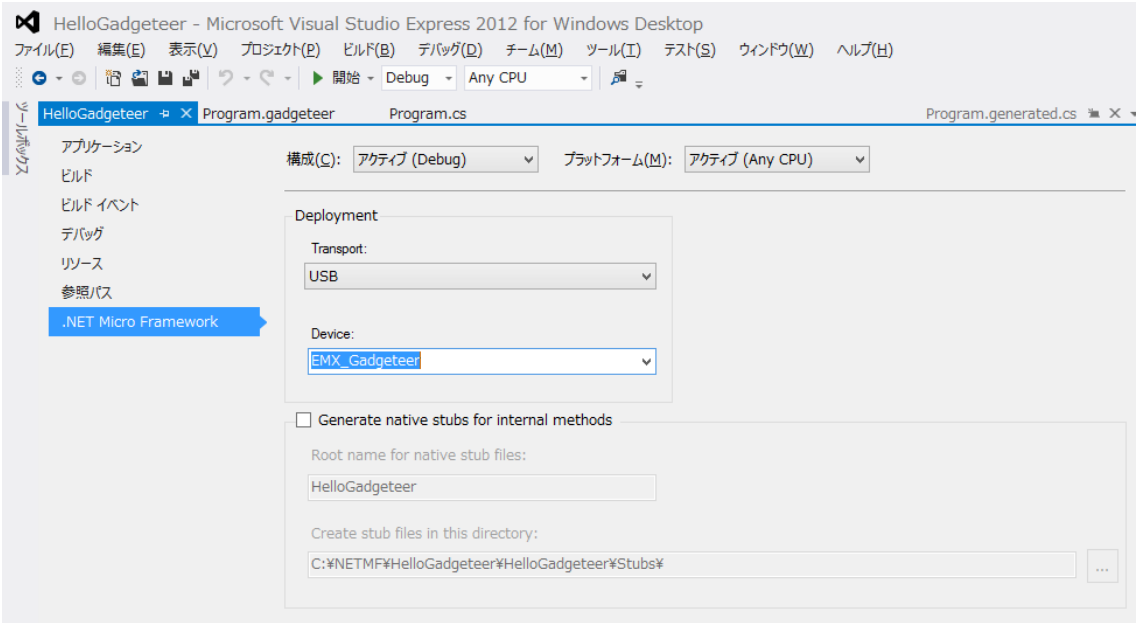


手順２：PC への接続

USB Mini-B ケーブルを使用してメインボードに接続した電源部分の USB Client DP または USB Client SP モジュールから PC の USB ソケットに接続します。

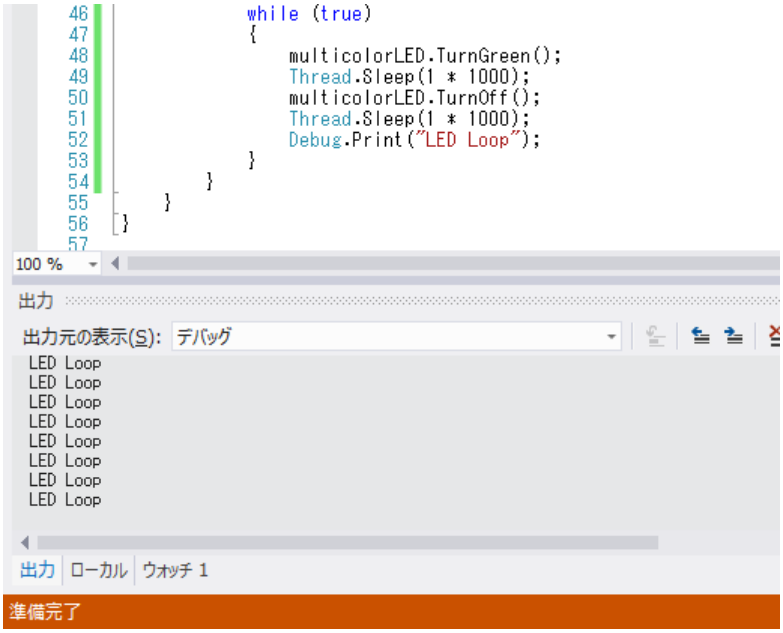
手順３：接続先確認

Visual Studio から HelloGadgeteer プロジェクトの Property を開いて、.NET Micro Framework タブを開き、Deployment の Target が USB、Device が EMX_Gadgeteer になっていることを確認します。USB に接続後２～３秒程度で自動認識されます。



手順４：実行

「F5」キーまたはデバッグメニューから「デバッグ開始」を選択して、プログラムをデバッグ実行します。LED の点滅と出力 Window へのメッセージ出力を確認します。



□解説：デバッグと Deployment□

.NET Gadgeteer のデバッグ操作は次の動作を行います。

① USB ケーブルを介して Visual Studio からメインボードにデバッグ手順で接続します。
② デバッグ手順を使用してビルドしたプログラム（MS-IL 形式の.NET のアセンブリ）を転送します。(Deployment)
③ メインボードでは受け取ったプログラムを不揮発性の Flash ROM 内の Deployment 領域に書き込みます。
④ メインボード上で動作している TinyCLR に対して、デバッグ手順で転送したプログラムの起動を指示します。
⑤ TinyCLR は Flash ROM 内の Deployment 領域からプログラムを読み出してインタプリタ実行します。

従ってデバッグ操作を起動することと、メインボードにアプリケーション・プログラムを転送することと、プログラムを実行することは全く同じ動作となります。

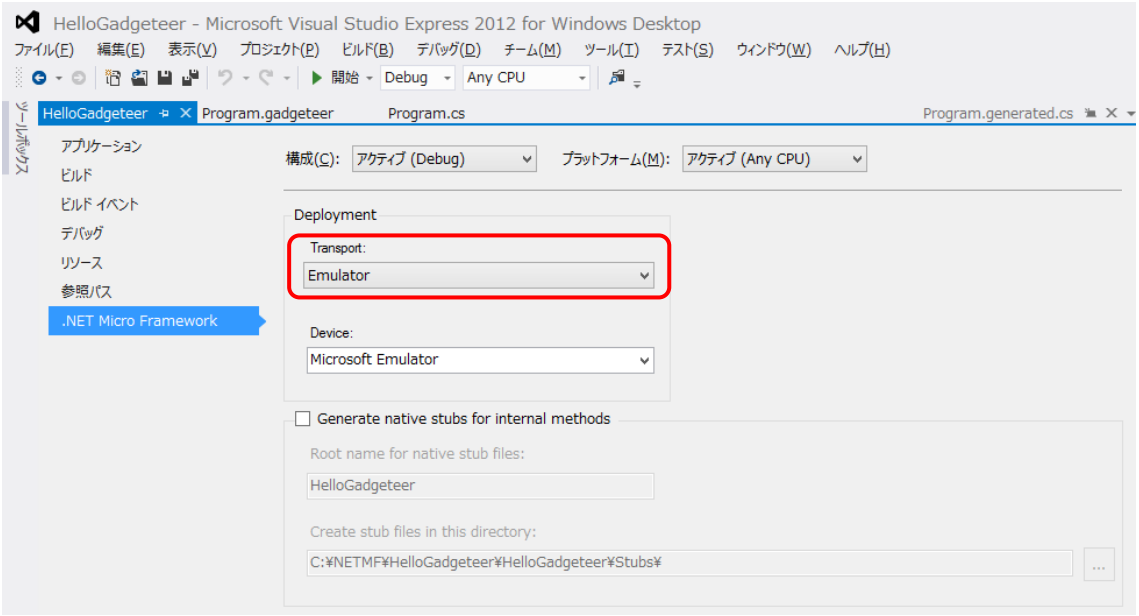
なおデバッグ停止後も転送したアプリケーション・プログラムはメインボードの不揮発性 Flash ROM 内に残り、また TinyCLR はインタプリタ実行を続けます。

また、Flash ROM 内の Deployment 領域に転送されたプログラムは電源を切ってもそのまま残るので、次回以降電源が入り TinyCLR が起動すれば Deployment 領域を自動認識して起動するようになっています。

□解説：接続 Target と Emulator□

手順３でデバイスとの接続を確認した、プロジェクトの Property 画面では.NET Gadgeteer を使用して作成したプログラムの転送先、デバッグ接続方法を設定・確認することができます。この画面の設定では実機が無い場合でも、メッセージ表示やグラフィック表示だけを行うプログラムを Visual Studio に組み込んだ Emulator を使用して動作させたり、デバッグしたりすることが可能です。ハードウェアのモジュールを使用するプログラムでは例外が発生します。

Emulator を使用して動作させる場合には、この「.NET Micro Framework」画面で次のように Target を「Emulator」に変更します。



設定すると自動的に Device が Microsoft Emulator」に変更されます。この状態で「F5」キーを押すか、デバッグメニューから「デバッグ開始」を選択することで、ビルドしたプログラムを実行します。

ハードウェアを制御しないプログラムであれば、デバッグの中断、ブレークポイント、変数内容表示等、Visual Studio のデバッグ機能が利用できることを確認します。

○演習９）

HelloGadgeteer プロジェクトで次のことを確認します。

① そのままエミュレータで実行して、例外発生の確認
② LED を制御しない様にソースコードを修正してデバッグメッセージ出力の確認

4.4. メインボードのネットワーク設定

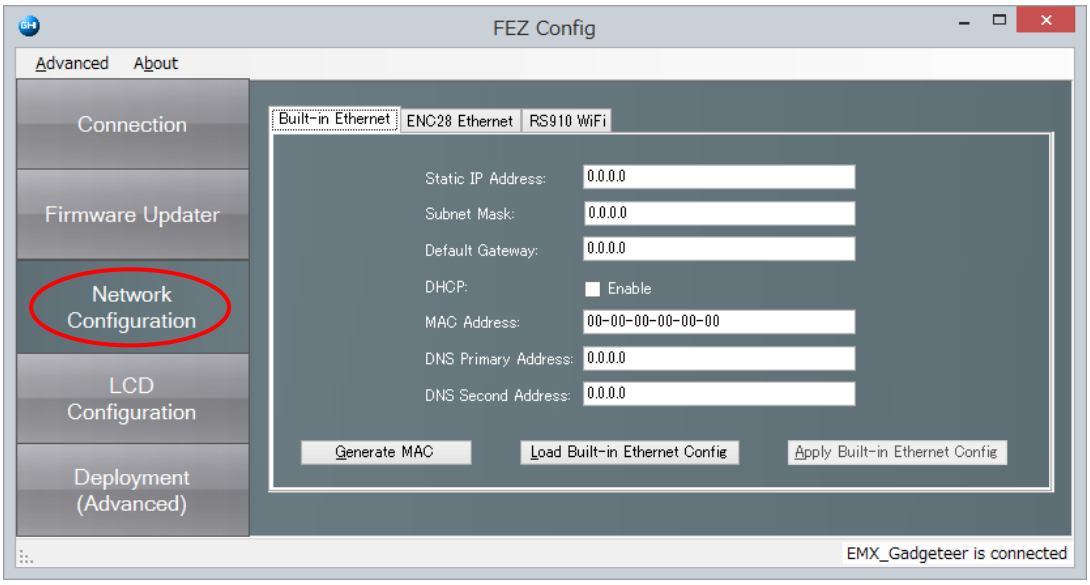
ネットワーク・アプリケーションを作成する前に、次の手順でメインボードの Flash ROM の領域のネットワーク情報を設定し保存します。

○演習１０）

手順１：FEZ Config の起動

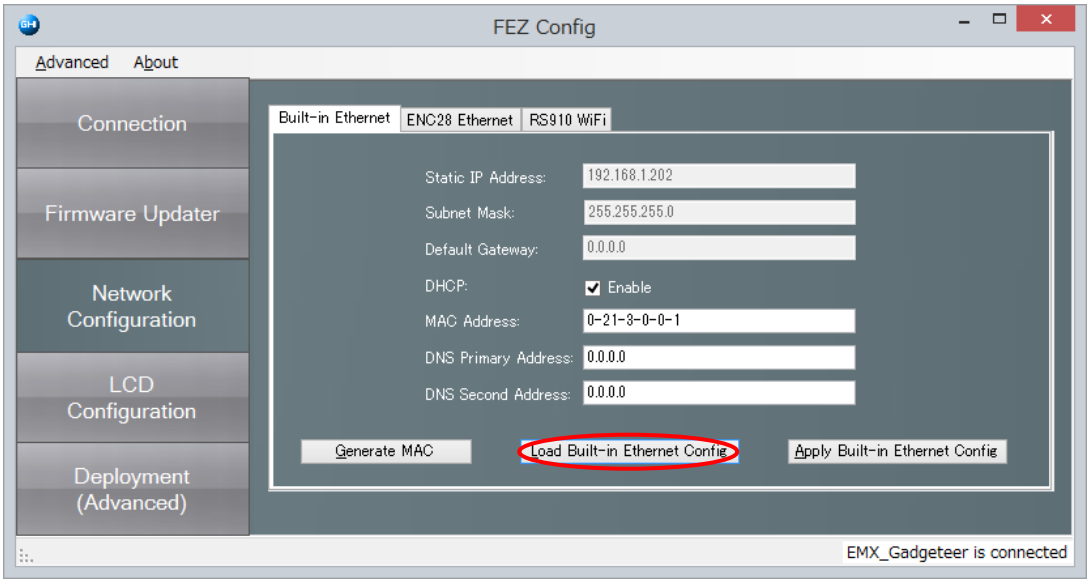
FEZ Spider メインボードに USB Client SP または Client DP モジュールを接続して PC の USB コネクタに接続します。LED 点滅などのプログラムが動作したままでも構いません。

FEZ Config を起動して「Network Configuration」タブをクリックします。



手順２：既存設定の読み込み

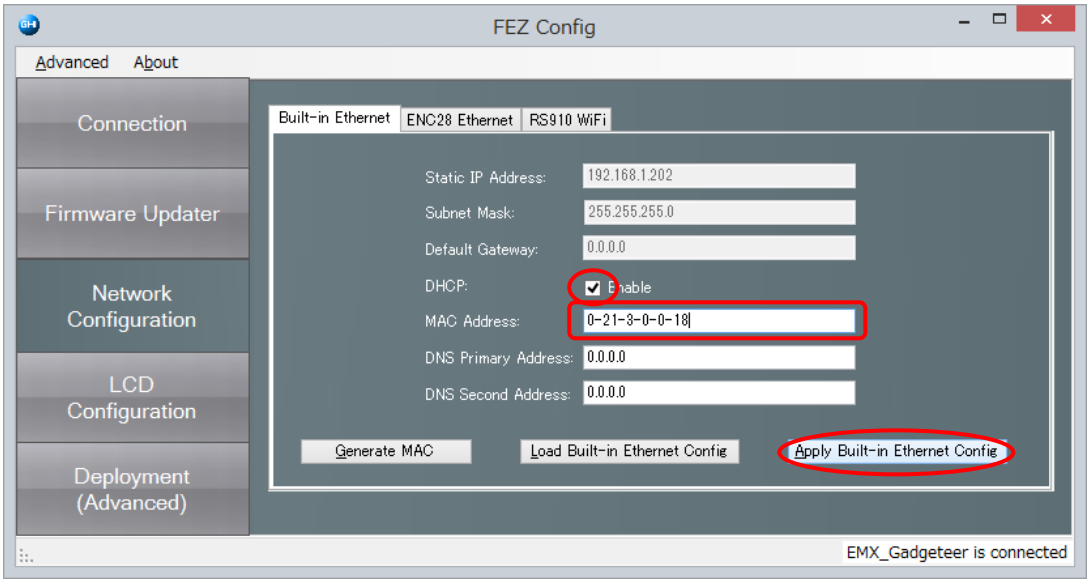
「Load Built-in Ethernet Config」をクリックして既存設定を読み込みます。



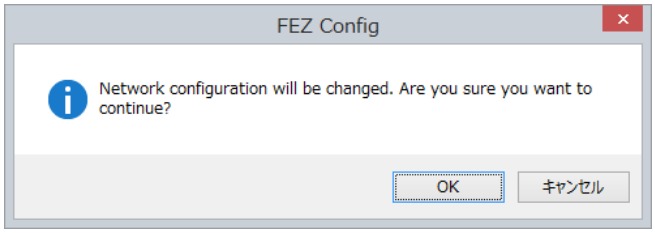
手順３：設定の変更（DHCP 使用）

この画面でネットワーク関連の設定を確認・変更できます。次の画面では DHCP を有効(Enable)にして、MAC Address の一番右の桁（一番低位の値）を 0x01 から 0x18 に変更しています。DHCP を使用する場合には、MAC Address 以外の項目は参照されません。

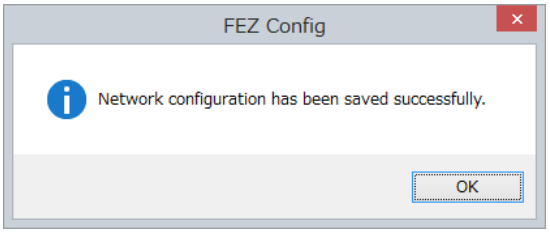
設定後「Apply Built-in Ethernet Config」をクリックして設定を保存します。



保存確認のダイアログが出るので「OK」をクリックします。



保存完了のダイアログが出るので「OK」をクリックします。



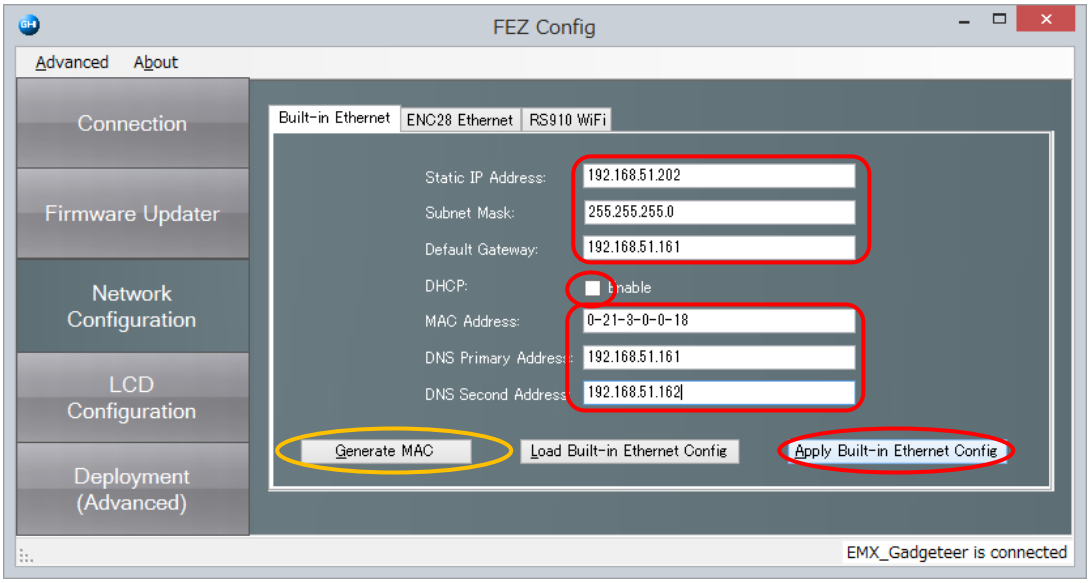
■ヒント：.NET Gadgeteer と.NET Micro Framework の MAC アドレス■

.NET Gadgeteer と.NET Micro Framework では、Ethernet の MAC アドレス（物理アドレス）をメインボード上の Flash ROM 内のコンフィグレーション領域にソフトウェア・データとして保存しています。そのため利用者がユニークな番号を設定・管理する必要があります。ローカルネットワーク上に接続する FEZ Spider 等の.NET Gadgeteer ボードが 1 台の場合には、デフォルトの「0-21-3-0-0-18」のまま変更しなくても動作しますが、2 台以上を同時に接続する場合には、この様に変更しておく必要があります。

またこの様に、自分で MAC アドレスの番号を入力・管理しなくても、右側の「Generate MAC」をクリックすることで、適当な MAC アドレスを自動生成する機能も用意しています。（下図）

手順４：設定の変更（スタティック IP 使用）

以下にスタティック IP アドレスを使用する場合の設定例を示します。ご使用になる環境に合わせた各アドレス値を設定して下さい。この場合も MAC アドレスを設定し直しています。



★注意：ネットワーク設定情報★

Flash ROM 領域に保存したネットワーク設定情報は、次の条件で容易に書き換えられます。

① TinyCLR をアップデートした場合、デフォルト状態に設定されます。

② アプリケーション・プログラムがネットワーク設定を変更した場合は、変更した内容に書き換えられ、そのまま残ります。

ネットワーク設定は、Flash ROM 領域の元の情報とは無関係に、アプリケーション・プログラムから強制的に変更して動作が可能です。例えば ROM 領域の DHCP が無効でも無視して DHCP を動作させたり、逆に常時スタティックな IP アドレスで動作させたりすることが可能です。ROM 領域の設定内容は容易に変更されることを念頭において、利用用途に合わせて最適な設定方法を選択して下さい。

4.5. ネットワーク・アプリケーションの開発

次の演習手順に従って.NET Gadgeteer V4.3 用のネットワーク・アプリケーション・プログラムを開発します。

V4.3 用のネットワーク・アプリケーション開発ではライブラリの一部に不具合があるため、注意が必要です。

★注意：.NET Gadgeteer V4.3 固有のネットワークの問題★

.NET Gadgeteer V4.3 では V4.2 からネットワーク関連のライブラリが変更されたため、いくつかの問題が発生しています。この章ではこれらの問題に応急的に対応する様に、プログラムと開発手順を追加しているのでご注意下さい。以降の SDK のリリース後、これらの問題が解決された場合にはご自身で判断して、対応策を取り除くことが可能です。

現在判明している問題点と対応策は次の 3 点です。

① DHCP でのネットワーク情報取得ができない場合がある。

対応策：

ProgramStarted()メソッドの最初に Thread.Sleep(1);を挿入して、ネットワークの動作開始を遅らせる。

② スタティック IP 利用時に、ネットワークが有効化できない。

対応策：

アプリケーション動作開始時、一旦 DHCP を有効化してネットワークを動作させ、その後スタティック IP に設定し直す。DHCP を有効化で ROM 内のネットワーク情報が書き換わるので、DHCP を有効化の前にスタティック IP 設定情報を一時的に保存しておく。

③ Builtin ネットワークデバイスのイベントが上がらない場合がある。

対応策：

イベントを使用せずにコーディングする。

④ Gadgeteer.Webserver ネームスペースのライブラリに例外が発生する。

対応策：

プログラムのバグが原因のため、Gadgeteer.Webserver ライブラリの DLL を入れ替える。

○演習１１）

手順１：Gadgeteer.Webserver ライブラリの DLL の入れ替え

Visual Studio を一旦終了後。管理者権限で、

C:¥Program Files (x86)¥Microsoft .NET Gadgeteer¥Core¥Assemblies¥.NET Micro Framework 4.3

以下に添付 CD の Gadgeteer.Webserver43 フォルダ内のファイルを全て上書きします。

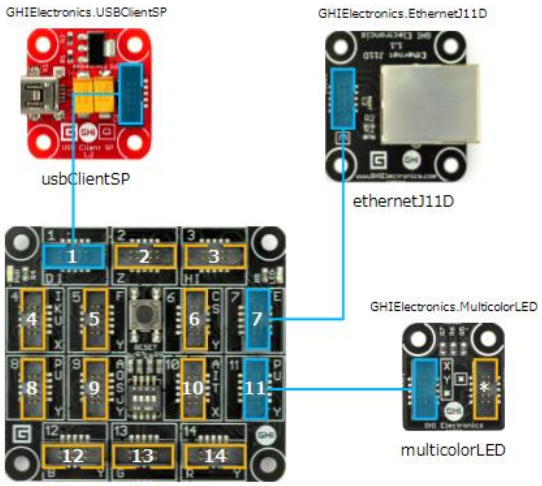
上書き更新するのは次の 11 個のファイルです。

Gadgeteer.WebServer.dll
Gadgeteer.WebServer.pdb
Gadgeteer.WebServer.xml
be¥Gadgeteer.WebServer.dll
be¥Gadgeteer.WebServer.pdb
be¥Gadgeteer.WebServer.pdbx
be¥Gadgeteer.WebServer.pe
le¥Gadgeteer.WebServer.dll
le¥Gadgeteer.WebServer.pdb
le¥Gadgeteer.WebServer.pdbx
le¥Gadgeteer.WebServer.pe

手順２：プロジェクトの作成とメインボード選択

演習７の開発手順と同様に、C:¥METMF 以下に「LEDServer」という名前の「FEZ Spider」のプロジェクトを作成します。**メインボード選択画面でバージョン番号を「4.3」に設定することに注意して下さい。**「FEZ Spider」メインボードに次のモジュールを組み込み、配線して保存します。組み込み・配線したキャンバスの例を以下に示します。

① USB Client SP または Client DP モジュール
② Smart Multicolor LED
③ Ethernet J11D



手順３：ソースコードの修正

ソースコードを次の様に追加修正します。このプログラムでは Flash ROM 領域内の DHCP のチェックに基づいて、DHCP の使用・未使用を決定する様に動作します。

① 10 行目として、次の using 文を using Microsoft.SPOT.Touch;の後に追加します。

```
using Microsoft.SPOT.Net.NetworkInformation;
```

□解説：NetworkInformation ネームスペース□

NetworkInformation ネームスペースは IP アドレスの様なネットワークの低レベル処理を扱うために頻繁に呼び出すクラスを含むため、using で指定しておくとプログラムがすっきりして見通しが良くなります。

② 変数宣言を Debug.Print("Program Started");の前に追加します。

```
string ipAddress = "";
string saveIpAddress = "";
string saveSubnetMask = "";
string saveGatewayAddress = "";
string[] saveDnsAddresses = null;
GHI.Networking.EthernetBuiltIn ei = ethernetJ11D.NetworkInterface;
NetworkInterface en = ethernetJ11D.NetworkSettings;
```

□解説：saveIpAddress 等の変数□

saveIpAddress 等の変数は本来必要ないものです。今回はスタティック IP アドレス使用時にネットワークを有効化できないという問題に対処するため、Flash ROM 領域で設定してある情報を一時保管するために使用します。

③ Sleep と Open 処理を Debug.Print("Program Started");の後に追加します。

```
Thread.Sleep(1);

if (!ei.Opened)
{
    Debug.Print("now open J11D");
    ei.Open();
}
```

□解説：Thread.Sleep(1)□

Thread.Sleep(1);は、本来必要ない処理です。DHCP を安定動作させるために付加しています。

④ 続いて DHCP の設定とスタティック IP の保存処理を追加します。

```
if (ei.IsDhcpEnabled)
{
    Debug.Print("now enable DHCP");
    ei.EnableDhcp();
}
else
{
    // Save Flash ROM settings
    char[] caIpAddress = en.IPAddress.ToCharArray();
    saveIpAddress = new string(caIpAddress);

    char[] caSubnetMask = en.SubnetMask.ToCharArray();
    saveSubnetMask = new string(caSubnetMask);

    char[] caGatewayAddress = en.GatewayAddress.ToCharArray();
    saveGatewayAddress = new string(caGatewayAddress);

    int numOfDns = en.DnsAddresses.Length;
    saveDnsAddresses = new string[numOfDns];
    if (numOfDns > 0)
    {
        char[] dnsAddress0 = en.DnsAddresses[0].ToCharArray();
        saveDnsAddresses[0] = new string(dnsAddress0);
    }
    if (numOfDns > 1)
    {
        char[] dnsAddress1 = en.DnsAddresses[1].ToCharArray();
        saveDnsAddresses[1] = new string(dnsAddress1);
    }
}
PrintNetworkInfo();
```

□解説：else 内の処理□

else 内の処理は本来必要ないものです。今回はスタティック IP アドレス使用時にネットワークを有効化できないという問題に対処するため、Flash ROM 領域で設定してある情報を一時保管するために記述しています。

⑤ インターフェース処理を追加します。

```
            foreach (var ni in NetworkInterface.GetAllNetworkInterfaces())
            {
                if (ni.NetworkInterfaceType == NetworkInterfaceType.Ethernet)
                {
                    if (ni.IsDhcpEnabled)
                    {
                        ipAddress = ni.IPAddress;
                        if (ipAddress != null && ipAddress != "0.0.0.0")
                        {
                            break;
                        }
                        else
                        {
                            ni.RenewDhcpLease();
                            Thread.Sleep(500);
                            ipAddress = ni.IPAddress;
                        }
                    }
                    else
                    {
                        ni.EnableDhcp();
                        Thread.Sleep(500);
                        ni.EnableStaticIP(saveIpAddress,                saveSubnetMask,
saveGatewayAddress);
                        ni.EnableStaticDns(saveDnsAddresses);
                        ipAddress = ni.IPAddress;
                    }
                    break;
                }
            }
```

□解説：インターフェース処理□

.NET Gadgeteer 4.3 では本来、インターフェース処理はイベントハンドラで記述することになっているのですが、動作しないため V4.2 以前の様にインターフェース内を検索する方法で記述しています。ここの記述を変更することで Flash ROM の設定内容とは関係なく、意図的に DHCP やスタティック IP の動作をさせることが可能です。

⑥ デバッグ用メッセージとイベントハンドラの登録処理をメソッドの最後に追加します。

```
            PrintNetworkInfo();
            Debug.Print("http://"              +              ipAddress              +
"/LED?status=on|off|red|green|blue|white");

            try
            {
                WebServer.StartLocalServer(ipAddress, 80);
            }
            catch (Exception e)
            {
                Debug.Print("Network error: " + e.Message);
            }
            var ledEvent = WebServer.SetupWebEvent("LED");
            ledEvent.WebEventReceived += ledEvent_WebEventReceived;
```

⑦ デバッグメッセージ出力用 PrintNetworkInfo()メソッドを追加します。

```
        void PrintNetworkInfo()
        {
            NetworkInterface en = ethernetJ11D.NetworkSettings;
            Debug.Print("---------------------------------------------");
            Debug.Print("IP Address:   " + en.IPAddress);
            Debug.Print("DHCP Enabled: " + en.IsDhcpEnabled);
            Debug.Print("Subnet Mask:  " + en.SubnetMask);
            Debug.Print("Gateway:      " + en.GatewayAddress);
            Debug.Print("---------------------------------------------");
        }
```

□解説：PrintNetworkInfo()メソッド□

このメソッドが無くてもプログラムは動作します。これは元々、デバッグ用のメッセージ出力処理で、デバッグ終了後削除する予定でした。しかし V4.3 の DHCP, IP アドレスの処理とインターフェースの起動では、様々な問題が発生しているため、トラブル発生時の情報出力のためにそのまま残しています。

⑧ イベントハンドラ ledEvent_WebEventReceived()をクラスの最後に追加します。

```
        void ledEvent_WebEventReceived(string path, WebServer.HttpMethod method,
Responder responder)
        {
            string status = responder.GetParameterValueFromURL("status");
            switch(status)
            {
                case "red":
                multicolorLED.TurnRed();
                    break;
                case "green":
                multicolorLED.TurnGreen();
                    break;
                case "blue":
                multicolorLED.TurnBlue();
                    break;
                case "white":
                case "on":
                    multicolorLED.TurnWhite();
                    break;
                default:
                    multicolorLED.TurnOff();
                    break;
            }
        }
```

□解説：ledEvent_WebEventReceived()メソッド□

イベントハンドラで登録した、クライアントからのアクセスがある度に呼び出されるメソッドです。今回の LEDServer の主処理を行います。

手順４：ビルドとデバッグ・

ソースコードを追加修正した後は、実際にモジュールを配線して、ビルドとデバッグを行います。デバッグ開始前に必ず、PC と Ethernet J11 アダプタの RJ45 ソケットには、デバッグの PC 同じネットワークに接続している LAN ケーブルを接続する必要があります。

LEDServer フォルダ内に動作確認済のソリューションを置いています。参考にして下さい。

□解説：動作とデバッグ手順□

開発したネットワーク・アプリケーションは、Web サーバー機能を持っています。つまり http のリクエストに応じて、メインボードに接続した Multicolor LED を制御するというものです。

動作・デバッグ手順は、同じネットワークに接続しているマシンのブラウザから次の様に、URL を入力するだけです。IP アドレスはデバッグ起動時に Visual Studio の「出力」Windows に表示されるので、同じものを入力します。

動作事例）

この場合では、IP アドレスが「192.168.51.202」の場合の入力例です。英字の大文字小文字は区別するので、注意して下さい。

緑色点灯：http://192.168.51.202/LED?status=green
白色点灯：http://192.168.51.202/LED?status=on
消灯：http://192.168.51.202/LED?status=off

□解説：REST インターフェース□

このように http の GET や POST 等のメソッドを使用して、URL で一意に定まる文字列を送ったり、その結果としてのデータを取得したりする手順を REST と呼びます。この場合はブラウザの窓に入力した文字列が「GET メソッド」で送ったデータになります。一般的に REST インターフェースは、この様にブラウザがあれば容易にデータの送受信ができるため、広く利用されています。

■ヒント：プログラムの改造■

Multicolor LED は任意の色合いで点灯させたり、点灯方法を制御したりすることが可能です。次の参考文献をヒントにして、ブラウザから制御して様々な光らせ方をする様に、プログラムを改造することが可能です。

https://www.ghielectronics.com/docs/98/smart-multicolor-led-module

4.6. 参考文献

■ヒント：マニュアルと参考資料■

.NET Micro Framework / .NET Gadgeteer のアプリケーション・プログラムを開発する際に参考となる API の解説やサンプルプログラムの入手先を次に示します。

●MSDN：.NET Micro Framework V4.3 API オンラインリファレンス

http://msdn.microsoft.com/en-us/library/jj610646.aspx

○MSDN：.NET Gadgeteer V4.3 API オンラインリファレンス

現在公開されていません。公開までお待ちください。

●GHI Electronics 社.NET Gadgeteer メインボードとモジュール等の資料

https://www.ghielectronics.com/docs

●GHI Electronics 社開発者向けコミュニティ

https://www.ghielectronics.com/community

●Socket 仕様、デザイン使用等.NET Gadgeteer 関連ドキュメント

http://gadgeteer.codeplex.com/documentation

5. その他

本テキストについて、誤りや不明な点をみつけた場合には、以下宛にメールでご連絡をお願いします。

メール連絡先）
株式会社デバイスドライバーズ E-Kit 事業部
e-kit@devdrv.co.jp

以上