KCB-5 SDK Manual

Ver.1.0.0

2014/10

はじめに

このたびは KCB-5 開発キット（KCB-5 SDK）をお買い求めいただき、ありがとうございます。本製品をご使用の前に、本マニュアルを熟読なさいますようお願いいたします。

本マニュアルは KCB-5 を使ったプログラム開発手順ならびに開発ライブラリなどについて説明しています。KCB-5 ソフトウェア開発キットではプログラミング開発の参考のためソースコードをすべて公開いたしますが、著作権などの法的権利については近藤科学株式会社が有します。またマニュアルに記載の会社名、商品名およびロゴマークはそれぞれの会社の商標または登録商標ですので、無断で使用することはできません。

プログラム開発キットの性質上、本マニュアルおよび付属品に掲載された一切の情報の流用による結果について責任は負いかねます。また内容は予告無く内容が変更される場合があります。ご理解の上ご使用なさいますようお願いいたします。

諸注意

- 本製品をぬらしたり、湿度が高い場所で使用しないでください。
- 本製品は基板上の端子がむき出しであるので、ショートの危険性があります。ショートを起こしたり端子の誤接続をすると発熱/発火の恐れがありますのでご注意下さい。金属に接した状態での使用はおやめください。
- 本製品の日本国内以外での使用についてはサポートいたしかねます。
- ケーブルを差し込むときは、向きを間違えないようにしてください。
- ケーブル類はプラグ部分をつかんで挿抜してください。
- 異常（発熱・破損・異臭など）を感じたらすぐにバッテリーあるいは電源を切るようにします。問題が起こったら直ちに使用をやめ、弊社サービス部へご相談ください。

免責事項

●統合開発環境 Eclipse、Flash Loader Demonstrator について

本 CD-ROM に収録されている統合開発環境 Eclipse、Flash Loader Demonstrator は、サポート対象製品ではありません。サポートは一切行われませんので、あらかじめご了承ください。統合開発環境 Eclipse に関する一切の権利は、Eclipse Public License に基づきます。

●すべての収録ファイルについて

本 CD-ROM に収録されているすべての収録ファイル対して，その使用にあたって生じたトラブル等は、近藤科学(株)は一切の責任を負いません。インターネット等の公共ネットワーク、構内ネットワーク等へのアップロードなどは、近藤科学(株)の許可無く行うことはできません。

アフターサービスについて

本製品ならびに付属品についてのお問い合わせは弊社サポート窓口までご連絡下さい。

〒116-0014　東京都荒川区東日暮里 4-17-7
近藤科学株式会社　サービス部　TEL　03-3807-7648　（サービス直通）
土日祝祭日を除く 9:00 ～ 12:00　13:00 ～ 17:00

製品についての告知及びアップデータ等は弊社ウェブサイト http://www.kondo-robot.com に掲載されます。E-mail でのお問い合わせにつきましては、support@kondo-robot.com にて承りますが、回答までお時間を頂く場合がございます。

※C 言語や Eclipse についてのお問い合わせはお答えできない場合がありますので、あらかじめご了承ください。

目次

1. KCB-5 について

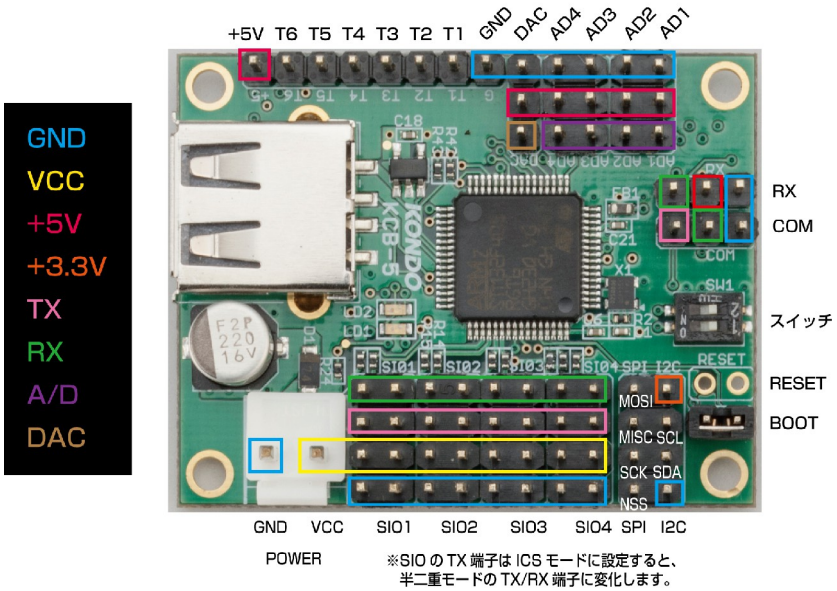
1.1. KCB-5 の概要

KCB-5 はシリアルサーボモーターの駆動を可能としたロボットコントローラーです。詳細は下記表をご覧ください。

スペック	
電源電圧	6～12V ※サーボモータを接続する場合はサーボモータの電源電圧にあわせます。
大きさ	35 x 45 mm
CPU	STM32F405R（Cortex-M4 STMicroelectronics 社製） ●168MHz 32bit ARM マイコン　●ROM：1MByte　●RAM：192+4kByte ●浮動小数点回路（FPU）内蔵　●アセンブラ言語、C 言語で開発可能
A/D 入力	12bit アナログ入力：4 電源電圧取得：1 ※電源電圧取得は、AD0 に接続されており、16.17V で 4095 を取得することができます。
DAC	12bit アナログ出力：1
タイマ	PWM 出力：6 ※Timer5/T1,T2　Timer3/T3～T6 　サンプルでは Timer4 を用い時間指定の割り込みを入れることができます。
UART	COM：1 ICS サーボモータ用ポート：4　（通常の UART としても使うことができます） 受信（RX）ポート：1
I2C	1 系統
SPI	1 系統 ※サンプルプログラムは用意しておりません。
USB	USB2.0 full-speed　　device/host/OTG ※device のプログラムを組むことはできますが、コネクタは USB-TypeA コネクタが実装されています。 ※USB のピン配置やポートの接続図は別途ご連絡ください。 ※接続された USB 機器の破損等の保障は致しません。 ※サンプルプログラムは用意しておりません。

1.2. ピン配置図

ピン配置図



コネクタのピン配置図です。図の Vcc ライン（SIO 端子のすべての 2 番ピン）には、電源である POWER IN 端子から入力された電圧がそのまま出力されます。Vdd 端子（AD 端子のすべての 2 番ピンおよび T 端子の+5V）にはレギュレータから出力される 5V 電圧が出力されます。

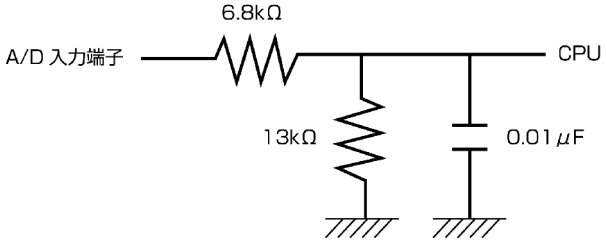
電源端子（POWER IN） VH コネクタ 2PIN(日本圧着端子製造株式会社)

電源には直流 6～12V のバッテリーまたは安定化電源を使用してください。HV 仕様のサーボに対しては、弊社製 ROBO パワーセル HV シリーズバッテリーおよび HV 電源スイッチハーネス の組み合わせが使用できます。使用するすべての機器を接続した後に電源を入れるようにしてください。また、供給された電源は直接 SIO 端子の電源（2 番ピン）に接続されています。接続するサーボモータの電源に合わせ電圧を設定してください。

アナログ入力ポート（AD1～AD4）

AD 入力ポートにはポテンショメーターや加速度センサーなどのアナログ計測装置が最大 4 個接続できます。2 番 Vdd 端子には 5V 電源が供給されます。信号電圧が 0～5V のアナログ計測装置などをご利用ください。AD 端子はマイコンのレジスター設定で出力端子として使うこともできます。なお、このポートへのシリアルデバイスの接続は行わないで下さい。A/D 入力は内部の CPU の関係上、下記の図のように分圧して CPU に入力されます。

AD 内部回路図



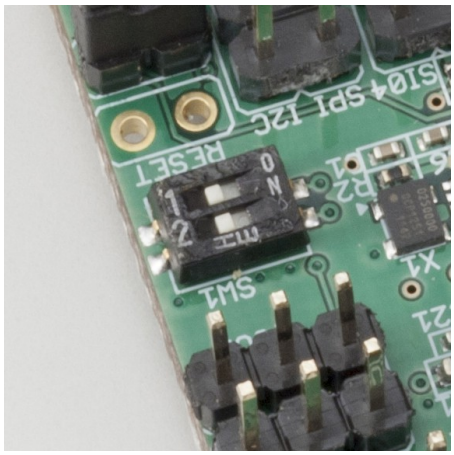
DAC 出力ポート（DAC）

DAC 出力ポートからは、アナログ電圧を出力することが可能です。アンプを介してスピーカーを接続し、アナログ波形を出力することで、音声などを再生することが可能です。 GPIO の設定を行うことで、入出力端子として使うことも可能です。 なお、このポートへのシリアルデバイスの接続は行わないで下さい。

ディップスイッチ（SW1）

2CH 分のディップスイッチを実装されています。常に入力としてお使いください。

スイッチ	ポート	ON	OFF
SW1-1	PC14	L	H
SW1-2	PC15		



LED

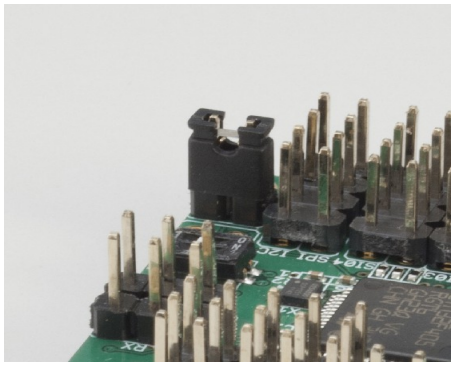
赤（LED1 RED）と緑（LED2 GREEN）の 2 つの LED が装備されています。

LED	色	ポート	消灯	点灯
LED1	赤	PB13	H	L
LED2	緑	PB14		

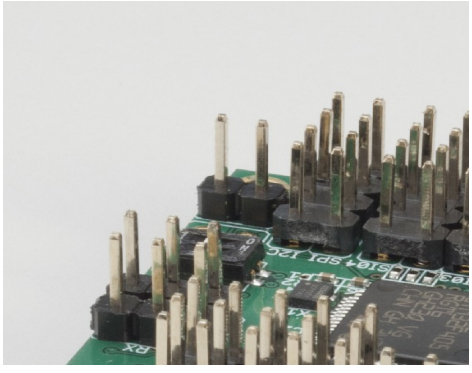
ブート端子（BOOT）

ジャンパなどでショートさせたまま電源を入れると、マイコンが書き込みモードになり、プログラム書き込みが可能となります。プログラム実行時は外してください。ジャンパブロックを挿抜する際は電源を切ってください。

ショート：プログラム書き込み



外す：プログラム実行



リセット端子（RESET）

リセット端子をジャンパなどでショートさせるとリセットがかかります。 端子は未実装ですので、必要に応じて増設が可能です。

I2C 端子（I2C）

I2C 通信に対応した液晶、各種センサ類、EEPROM などと接続可能な通信端子です。

1 番端子は 3.3V で、2 番端子は SCL、3 番端子は SDA、そして KCB-5 に対して外側の 4 番線がグランド線となります。GPIO の設定を行うことで、入出力端子として使うことも可能です。

SPI 端子（SPI）

SPI 通信に対応した液晶、各種センサ類などと接続可能な通信端子です。

1 番端子は MOSI で、2 番端子は MISO、3 番端子は SCK、そして KCB-5 に対して外側の 4 番線が NSS（チップセレクト）となります。電源 3.3V とグランドは、隣の I2C ピンのものを共用で使用します。

GPIO の設定を行うことで、入出力端子として使うことも可能です。

タイマ端子（T1～T6）

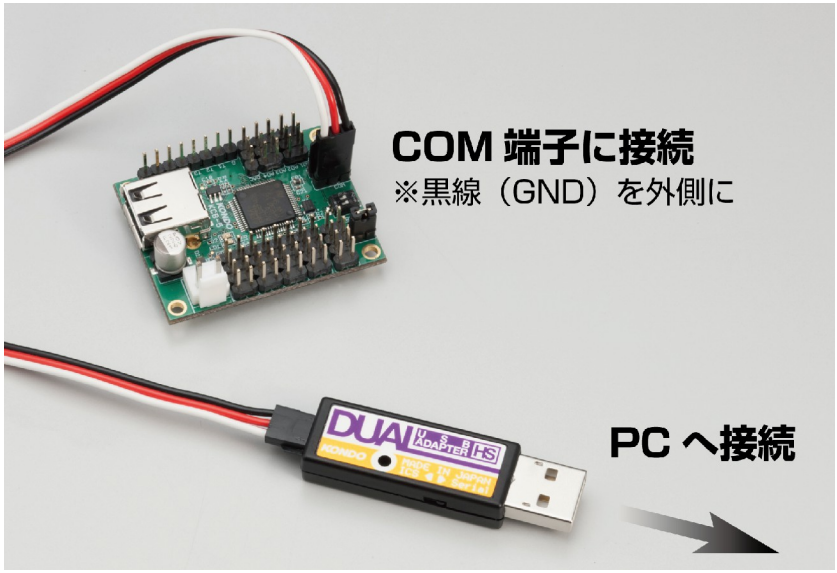
PWM の入力と出力が可能な信号端子です。タイマ 5 は 2 個（T1、T2）、タイマ 3 は 4 個（T3～T6）の端子にグループが分かれています。5V は+5 端子、グランド線は G 端子となります。

GPIO の設定を行うことで、入出力端子として使うことも可能です。

この信号端子は 3.3V でご使用ください。

COM 端子（HP15 COM）

パーソナルコンピュータ等との通信用に UART を利用したシリアル通信端子です。3 線式を採用していますので、コンピューターと接続する場合には、弊社製 Dual USB アダプター HS をご利用ください。



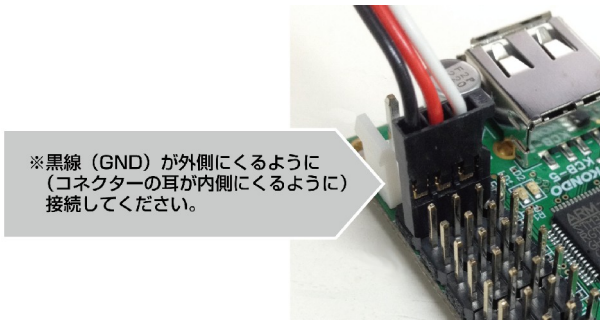
なお KRS シリーズなどの ICS シリアルモーターは接続できませんのでご注意ください。

1、2 番端子はいずれも信号線（1 番は送信、2 番は受信用端子）で、KCB-5 に対して外側の 3 番線がグランド線となります。

SIO 端子（シリアル IO 端子：SIO1、SIO2、SIO3、SIO4）

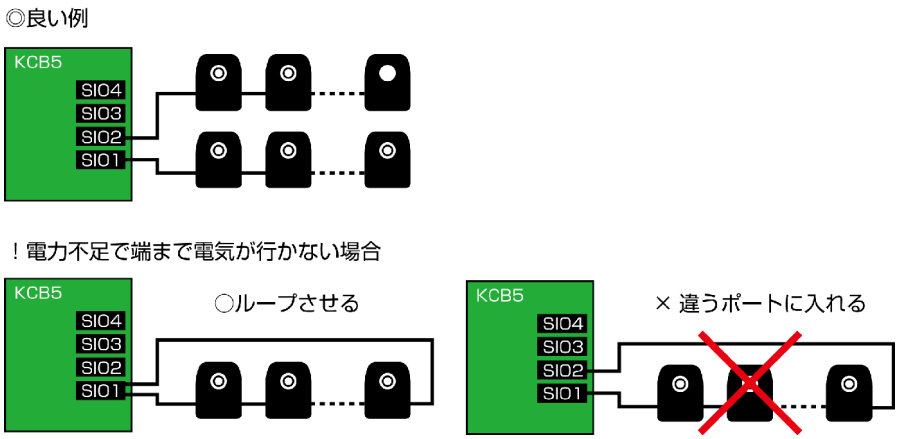
シリアルモーターを接続します。全ての SIO は 2 個の端子を持ち、それぞれ信号線を共有する 3 線式のシリアル通信を採用しています。3 線式では信号の送受信を信号線のみで行い、中央のラインはモーターへの電源供給ラインに利用していますので、通常のシリアル通信には使用できません。

中央のラインには電源端子に直接接続されていますので、サーボモータを接続する場合はサーボモータの定格電圧に合わせ電源を供給してください。



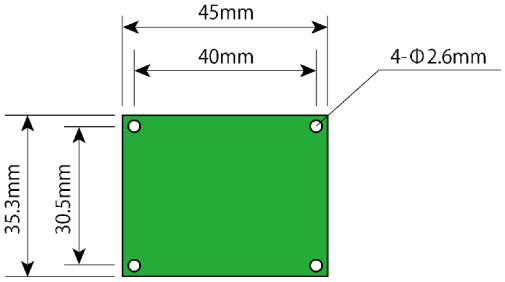
多数のモーターを数珠つなぎにしたときにモーターに負荷がかかり電圧が低下する場合があります。そのときには、下図のように数珠つなぎにした接続線を同じ SIO 端子へ差し込み、ループ接続にして下さい。ループ開始と終了のコネクターを違うコネクターに差し込まないでください。サーボモータは通信や電源の関係上1系統につき10個までとしてください。GPIO の設定を行うことで、入出力端子として使うことも可能です。この通信端子は 5V でご使用ください。

SIO（サーボモータ）接続図



1.3. 寸法図

外形図



2. 開発環境の整備

2.1. CDROM 内容の確認

KCB-5 SDK 付属 CDROM は以下のような内容となっています。

1. WorkSpace
   - サンプルプログラム（Samples）フォルダ
2. 開発環境
   - Setup_V1_0_0.exe： 統合開発環境ソフトウェア Eclipse
3. SerialTerm
   - SerialTerminal.exe： シリアル通信専用ターミナルソフトウェア
4. ICS3.5 シリアルマネージャー/ICS3.0 シリアルマネージャー

   弊社製シリアルモータを多数使用する場合には、モーターに固有の識別番号（以下モーター ID）が必要です。CDROM に付属の弊社製シリアルマネージャーソフトをご利用下さい。
5. マニュアル
   - このマニュアル
   - ICS3.0/ICS3.5 コマンドリファレンスマニュアル（シリアルサーボモーターを駆動するためのコマンド仕様書）

2.2. その他開発に必要なもの

●Dual USB アダプター HS、または SERIAL USB アダプター HS

作成したプログラムを KCB-5 に転送したり PC とシリアル通信をするには KCB-5 の COM 端子を使います。弊社製 DualUSB アダプター HS を USB ポートに接続しますと、USB ポートを COM ポートとして使用できるようになります。

※USB アダプターに関するソフト、マニュアルは以下のアドレスよりダウンロードしてご利用ください。

URL：http://kondo-robot.com/faq/dual-usb-adapter-hs-manual

※KHR などの組み立てでインストールされている場合は、再インストールは必要ありません。

※Dual USB アダプター HS はシリアルモードにします。

●フラッシュロム書き込みソフトウェア Flash Loader Demonstrator

Eclipse にて作成した実行プログラムをマイコンに転送するためのソフトウェアです。

Flash Loader Demonstrator は以下のアドレスよりダウンロードしてご利用ください。

URL：http://www.st-japan.co.jp/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257525

●電源

電源は 6～12V の直流電源またはバッテリーをご用意ください。KCB-5 に電源を接続する場合は、電源線と GND 線を間違えないように接続してください。必要な電流値（アンペア）は使用するモーターの種類や数などによって変わります。詳しくは使用するモーターの仕様書などをご覧になってください。

2.3. 開発環境インストール手順

Eclipse インストール

付属の CDROM の開発環境フォルダ内にある"Setup_V1_0_0.exe"を起動し、インストーラーの指示に従ってインストールしてください。

Flash Loader Demonstrator インストール

Flash Loader Demonstrator は以下のアドレスよりソフトをダウンロードして下さい。STSW-MCU005 の欄「Download」ボタンを押すと zip ファイルにてダウンロードされますので、解凍してください。解凍後、インストーラーの指示に従ってインストールしてください。

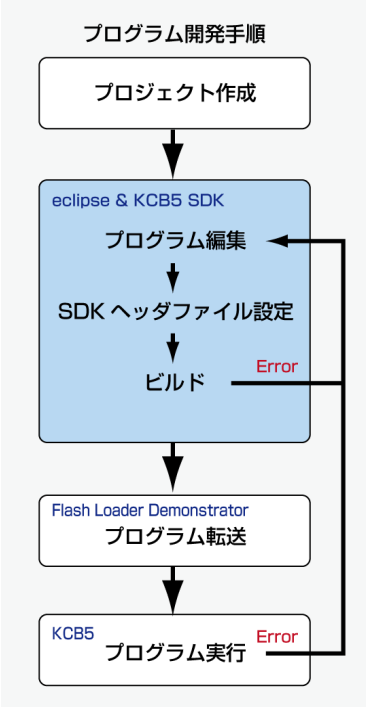
URL : http://www.st-japan.co.jp/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257525

3. プログラム開発

開発環境のインストールや設定など準備が終われば、次はプログラムの開発に挑戦です。プログラム開発から実行までは右図の手順で行います。

プログラム作成の前に、以下の項目について確認をしてください。

- Dual USB アダプター HS のドライバーをインストールし、アダプターの使用する COM ポート番号をデバイスマネージャで調べ、控えておきます。
  Dual USB アダプター HS はシリアルモードにします。
- 6～12V のバッテリーまたは安定化電源を用意し、KCB-5 と接続できるコネクタ（VH コネクタ 2PIN）を準備します。
- Eclipse のインストールをします。
- Flash Loader Demonstrator ソフトウェアのインストールをします。
- SerialTerminal または他のシリアル通信用のソフトウェアの準備をします。



以上が準備できましたら、始めに KCB-5 マイコンボードからコンピューターに「Hello World!」というメッセージを送るプログラミングにチャレンジしてみましょう！

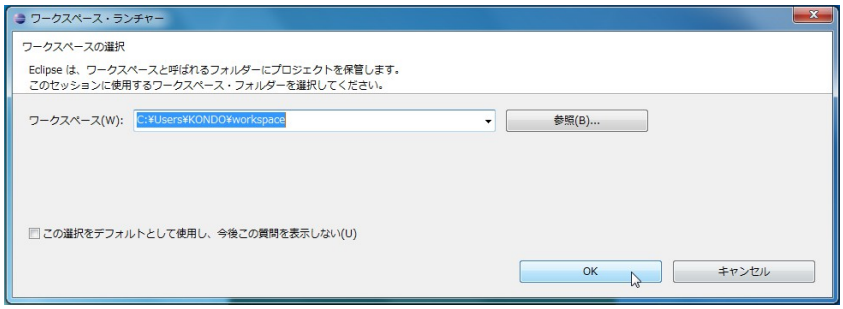
3.1. ワークスペース作成

Eclipse ではプログラムのソースコードやスタートアップルーチンファイルなどをまとめたものを「プロジェクト」と呼びプロジェクトをまとめて管理するディレクトリを「ワークスペース」と呼びます。Eclipse では、プログラム開発にはまずワークスペースを次の手順で作成します。

1.Eclipse を起動する。

- Eclipse の起動は、スタートメニューの「すべてのプログラム」から Eclipse→Eclipse の順で起動します。
- Windows Vista/7 ではユーザーアカウント制御ダイアログが表示されますが「許可」をクリックし起動して下さい。

2. ワークスペース・ランチャー

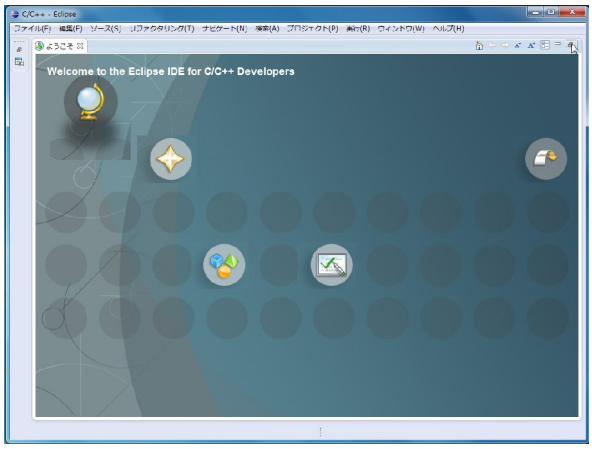


- Eclipse が起動すると、自動的に「ワークスペース・ランチャー」というタイトルのダイアログが表示され、任意のワークスペースを選択、または新規作成ができます。

▼自動的に表示されるフォルダ名のままでよければ、このまま「OK」ボタンを押して下さい。

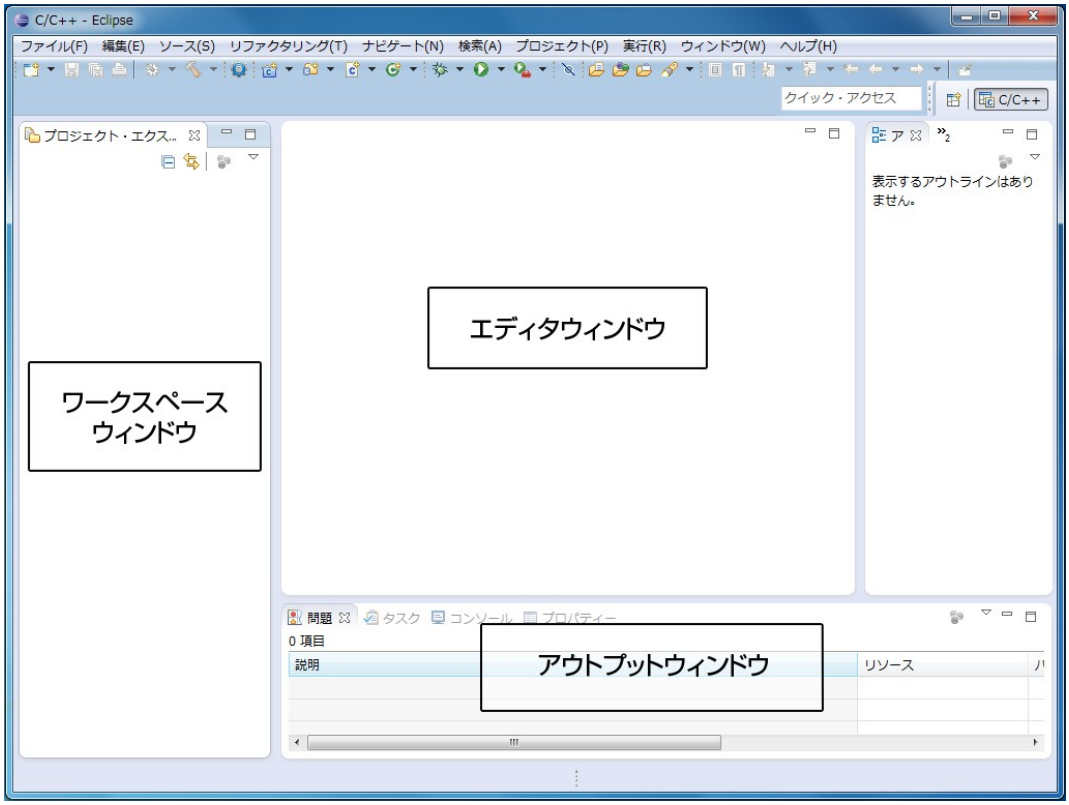
- 以前作成したワークスペースを読み込む場合には、パスの横のプルダウンメニューか、あるいは「参照...」ボタンを押し、任意のフォルダを選択してください。
- ワークスペースを保存するフォルダ名には日本語、あるいは空白を使ってはいけません。

3.ようこそ



- 新規にワークスペースを作成すると、「ようこそ」ページが表示されます。このページを閉じるか縮小化すると作業スペースが表示されます。

4.Eclipse プロジェクト画面構成

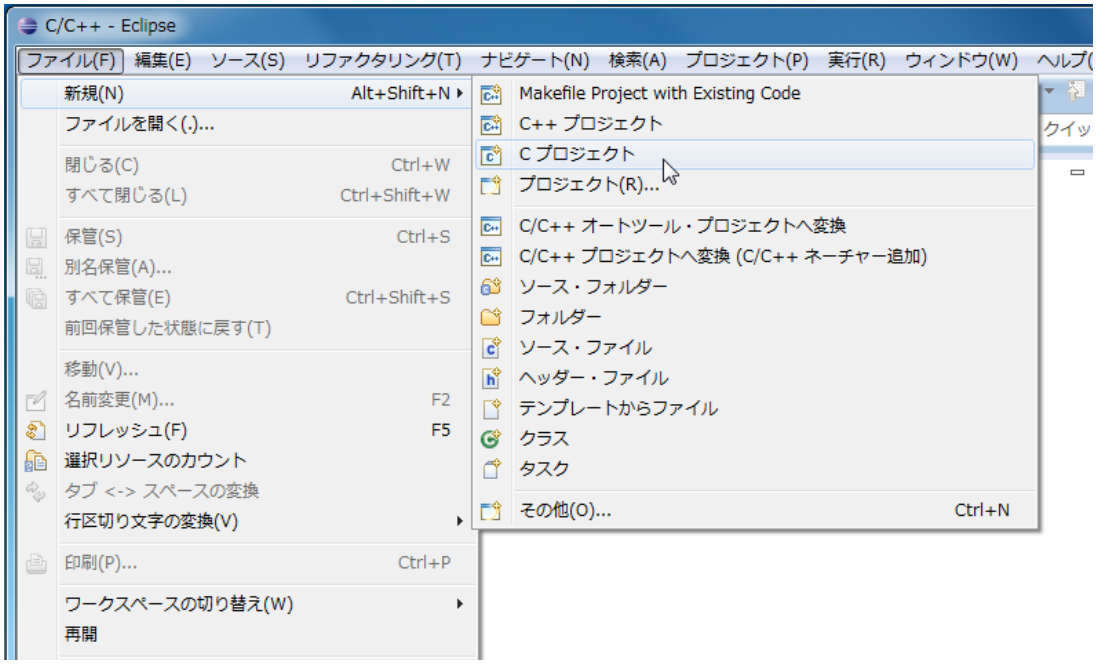


Eclipse のプロジェクト画面について説明します。

- Eclipse のプログラム編集画面は基本的に 3 つのフローティングパネルに分かれており、左上のパネルを「ワークペースウィンドウ」と呼び、プロジェクトを構成するファイルやディレクトリ構造がわかるようになっています。
- プログラムは右上の「エディタウィンドウ」で編集を行います。
- プログラムのコンパイル結果などは下の「アウトプットウィンドウ」に表示されます。

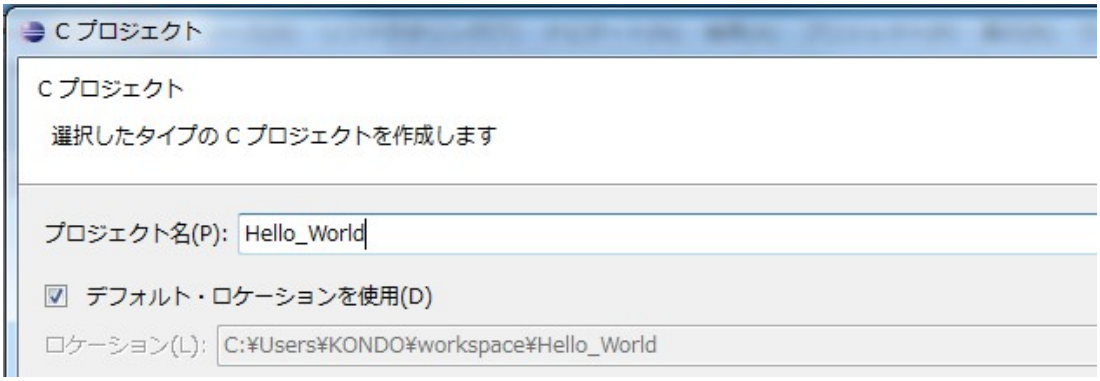
3.2. プロジェクトの作成

1. プロジェクトを作成します。

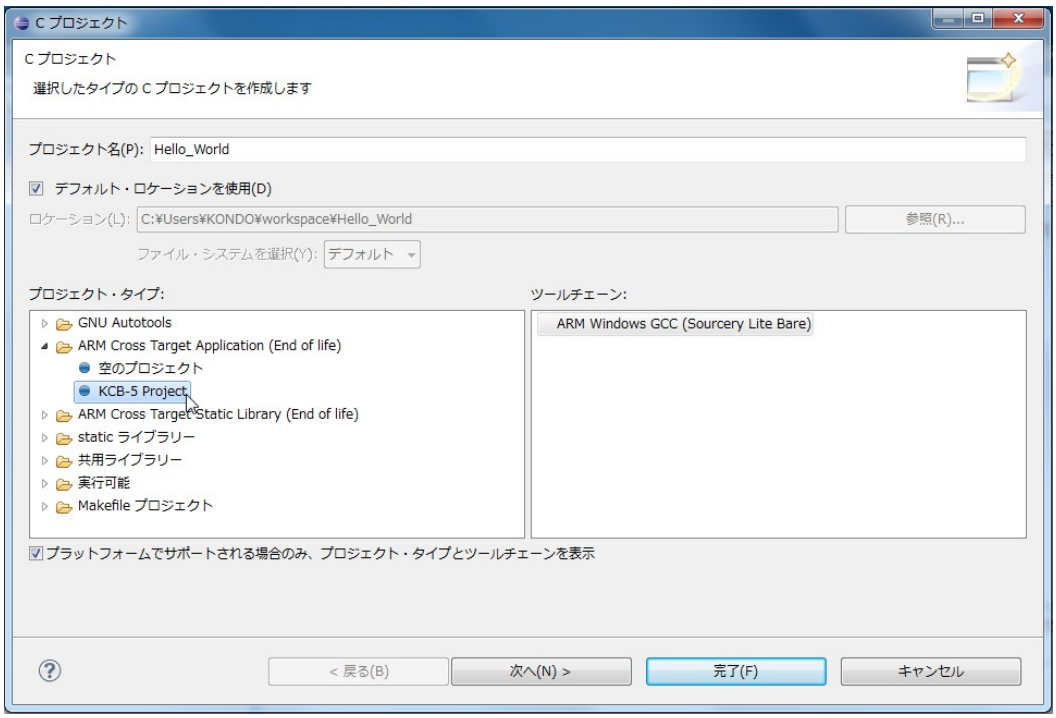


●「ファイル」 → 「新規」 → 「C プロジェクト」

2. C プロジェクト



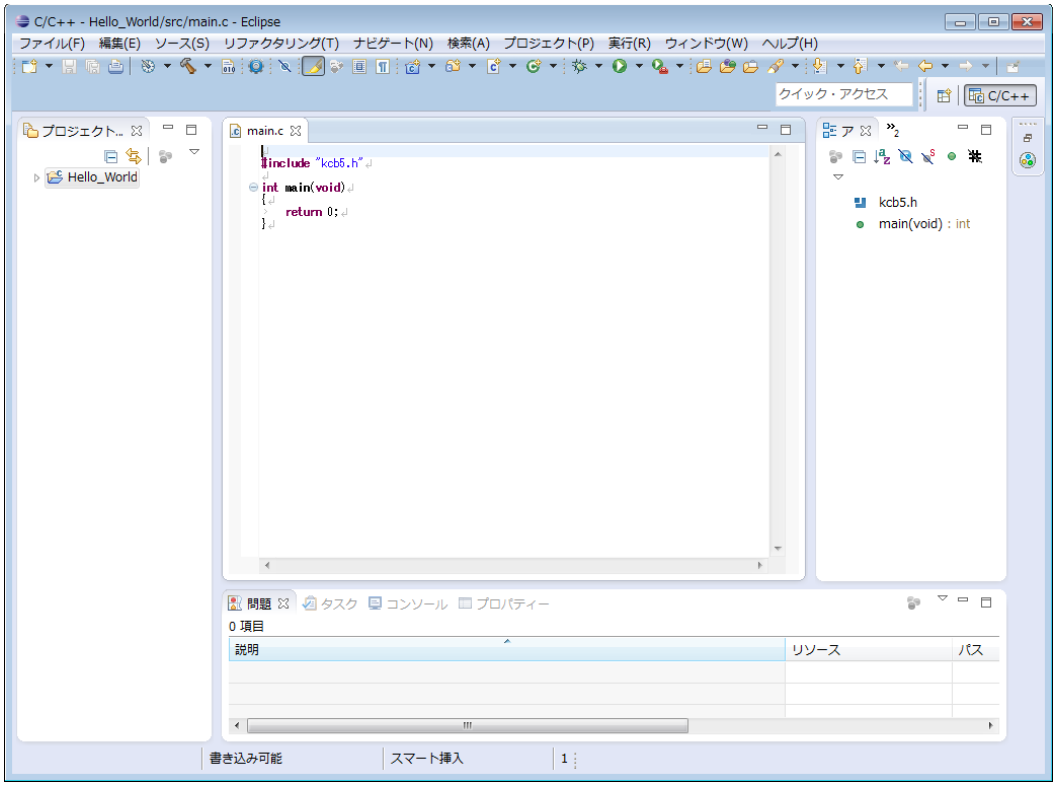
● プロジェクト名: 任意のプロジェクト名をつけます。ここでは例として「Hello_World」としました。



● プロジェクト・タイプ: "ARM Cross Target Application"の"KCB-5 Project"

● 編集が終わったら「完了」ボタンを押します。

3. プロジェクトが作成完了



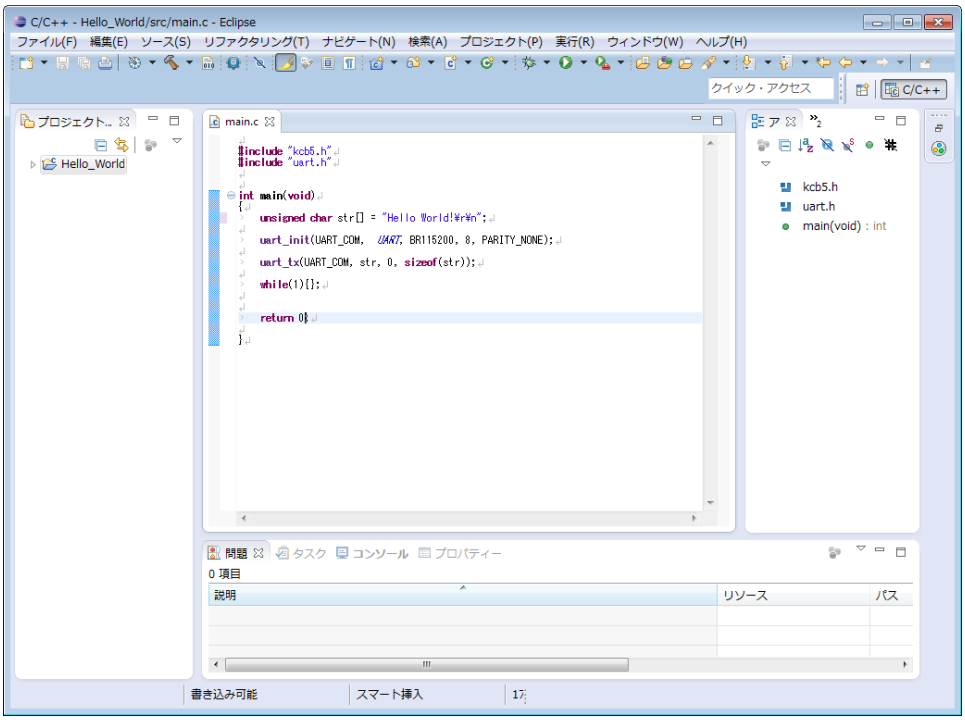
● プロジェクトスペースに KCB-5 を動作させるために必要なプログラムが作成されます。

3.3. プログラム編集

コンピューターにメッセージを送信するためには次の手続きが必要となります。

1. KCB-5 を使用する上で必要になるヘッダファイル kcb5.h、そしてシリアル通信関数が定義されているヘッダファイル uart.h を組み込みます(インクルードする)。
2. 配列に送信予定の文字列を定義する。（今回は「Hello World!」です）
3. COM ポートとコンピューター間のデータ送信条件を設定する。
4. 2.で定義した文字列を送信する。
5. ～4. の手続きに沿ってプログラムを作成・編集します。

はじめにプロジェクト・エクスプローラにある src に格納されている main.c のアイコンをダブルクリックすると、エディタウィンドウにファイル内容が表示されます。main.c を次のように書き換えてください。追加コード部分だけ青い文字で記述しています。
すべて半角英数字および半角記号を使い、大文字と小文字を間違えないようにしてください。エクリプスが自動生成するコメント部分は省略しています。



```
#include "kcb5.h"
#include "uart.h"

int main(void)
{
    unsigned char str[] = "Hello World!¥r¥n";

    uart_init(UART_COM,  UART, BR115200, 8, PARITY_NONE);

    uart_tx(UART_COM, str, 0, sizeof(str));

    while(1){};


    return 0;

}
```

1. `#include "kcb5.h"`
   KCB-5 の初期関数に関する C 言語関数が定義されています。

2. `#include "uart.h"`
   COM ポート関連の C 言語関数が定義されています。

3. `unsigned char str[] = "Hello World!¥r¥n";`
   配列に文字列を格納し、文字数分の配列を確保します。KCB-5 では￥n だけでは改行しか行なわれません。
   必ず ”￥r”（復帰コード)も入れてください。

4. `uart_init(UART_COM,  UART, BR115200, 8, PARITY_NONE);`
   COM ポートとコンピューター間で非同期通信接続する際の接続環境設定を行い、通信を開始します。
   ここで設定した条件は次の通りです。
   ●出力ポートは COM 端子（ uart_com ）
   ● ボーレート: 115200 bps
   ● データ長: 8bit
   ● パリティ: なし

5. `uart_tx(UART_COM, str, 0, sizeof(str));`

   `uart_tx` という関数でコンピューターに「Hello World!」という文字情報を送信します。

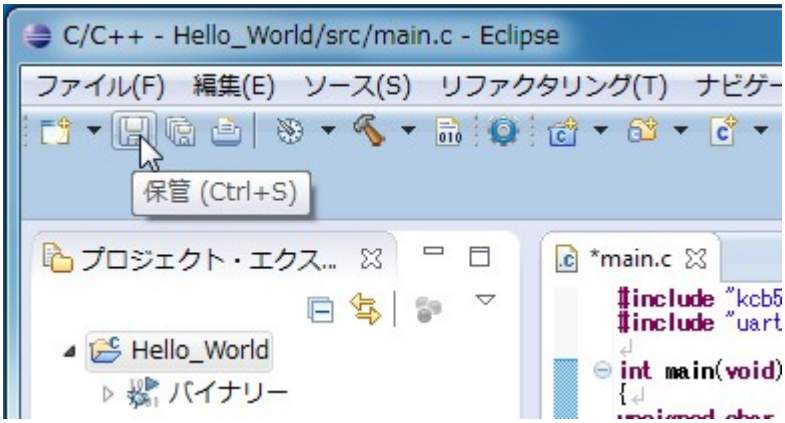
6. `while(1){};`

   無限ループにして、これ以上先に進まないようにブロックします。最後のセミコロンを忘れないでください。

7. return 0;

   int 型の main 関数ですので、戻り値を書いておきます。実際は上記無限ループがあるため、ここのプログラムは実行されません。

8. 編集が完了したら保管ボタンを押して保管します。



※編集後は必ず保管をする必要があります。保管されたプログラムに関してビルド（後述）を行いますので、保管される前のプログラムはビルドに反映されません。main.c タブにアスタリスク（＊）マークがついている状態は保管がされていない状態です。

※ビルド時に自動保管するには

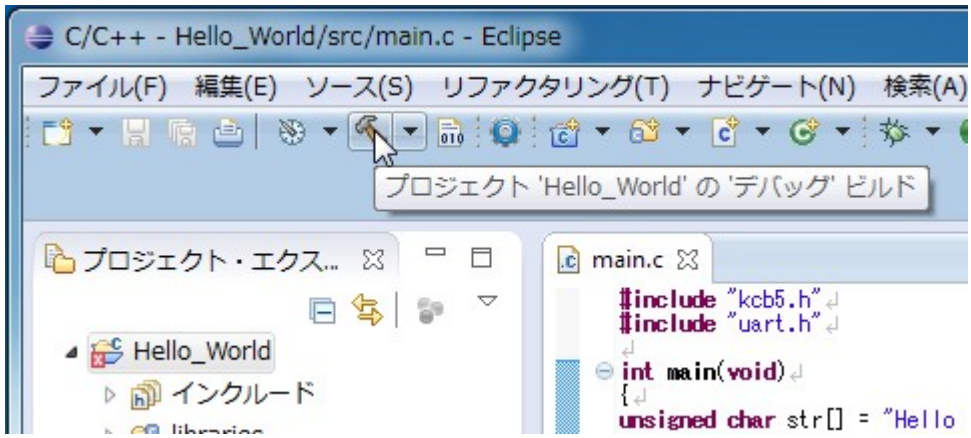
ウィンドウメニュー ＞ 設定 ＞ 一般 ＞ ワークスペース で

ビルド前に自動的に保管 にチェックを入れてください。

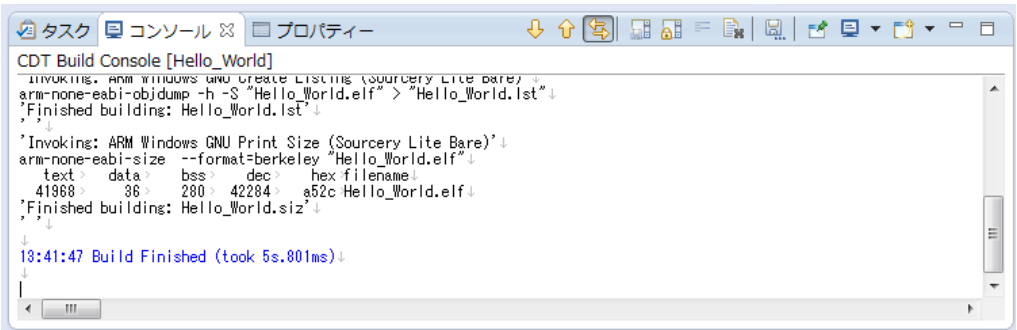
ビルド

Eclipse では、1)プログラムソースコードをコンパイル、2)必要なライブラリとリンク、3)プログラムをマイコンが使用できる実行形式に変換という手順を行いますが、これらをまとめて「ビルド」と呼んでいます。プログラムが完成したら次の手順でビルドを行います。

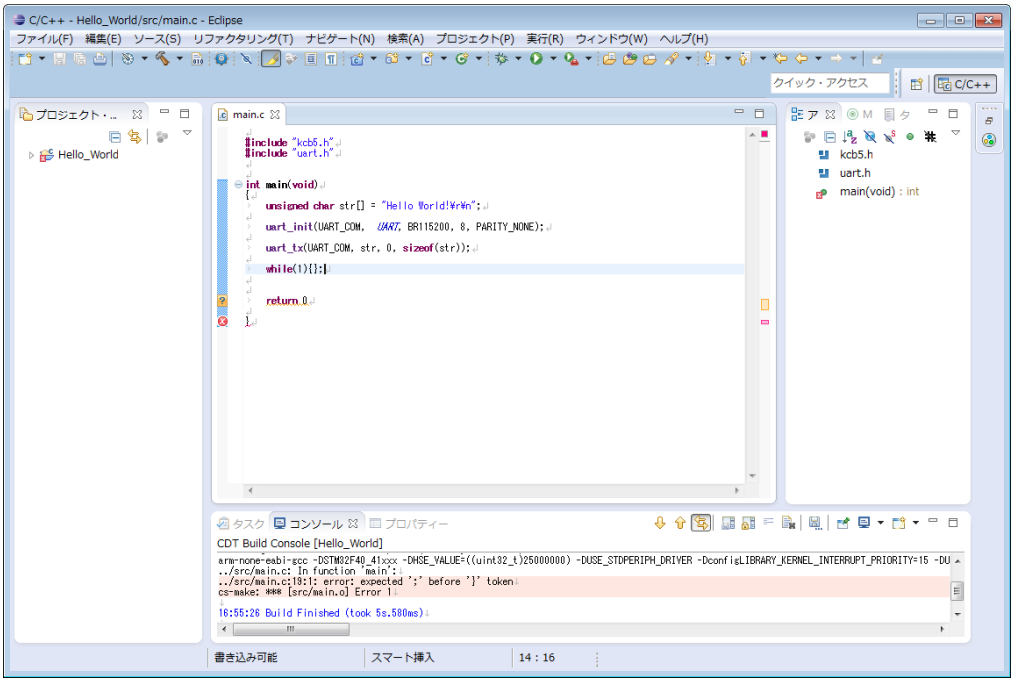
1. プロジェクトメニューから「プロジェクトのビルド」を選択する、またはツールバーのビルドボタン(金槌アイコン)を押すなどして、ビルドを実行します。



2. ソースコードにエラーが無く、ビルドが成功した場合には、アウトプットウィンドウのコンソールタブに「Build Finished 」というような表示が出ます。



3. ソースコードに間違いがあったり、include ディレクトリの指定を忘れていると、図のようにアウトプットウィンドウに赤いハイライトや黄色いハイライトと一緒にエラーメッセージ、またはワーニングが表示されます。
赤いハイライト、または黄色いハイライト(↓次のエラー、↑前のエラー ボタン)またはエラーメッセージをダブルクリックすると、エディタウィンドウのソースコードの該当エラー部分にカーソルが移動しますので、エラーメッセージを参考にしてプログラムの修正を行ってください。下図は、ソースコード return0 の後にセミコロン「;」を書き忘れたことによるエラー表示の例です。



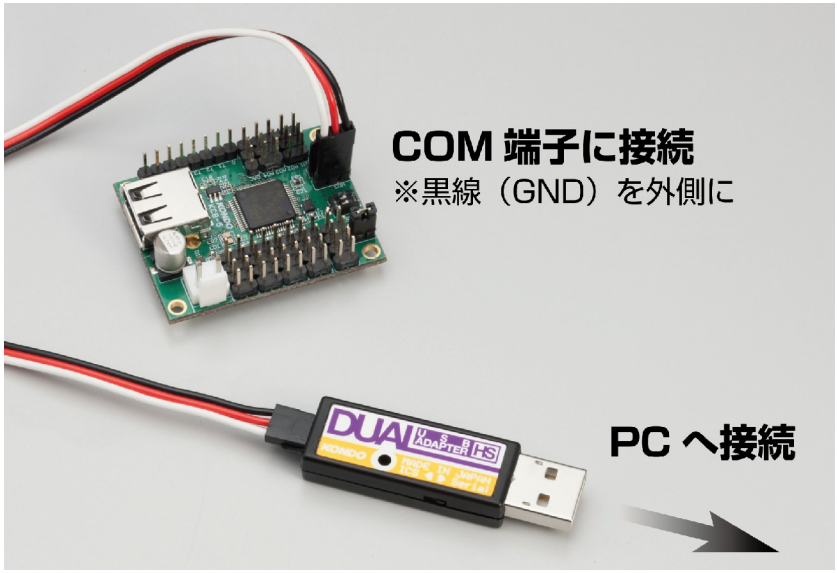
4. ビルドが成功すると、「C:¥Users¥(ユーザー名)¥workspace¥Hello_World¥Debug」フォルダに「Hello_World.hex」という STM32F405 マイコン用のの実行形式ファイルが作成されます。

3.4. プログラム転送

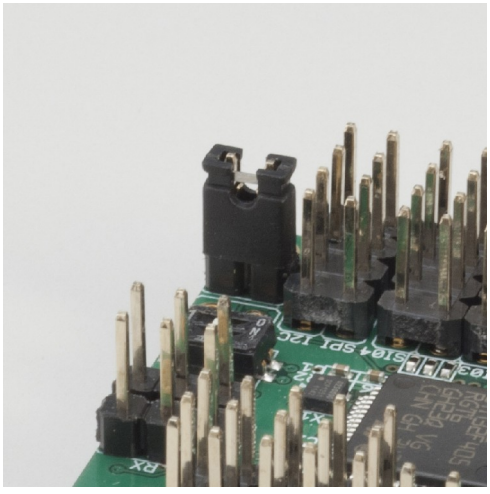
Flash Loader Demonstrator 使用手順

ビルドで作成した HEX ファイルを Flash Loader Demonstrator でマイコンに書込みます。

1. コンピューターの USB ポートに Dual USB アダプター HS（シリアルモード）を接続し、3 線ケーブルで KCB-5 の COM 端子と接続します。このとき端子番号を間違えないでください。黒いグランド線（3 番）が外側になります。

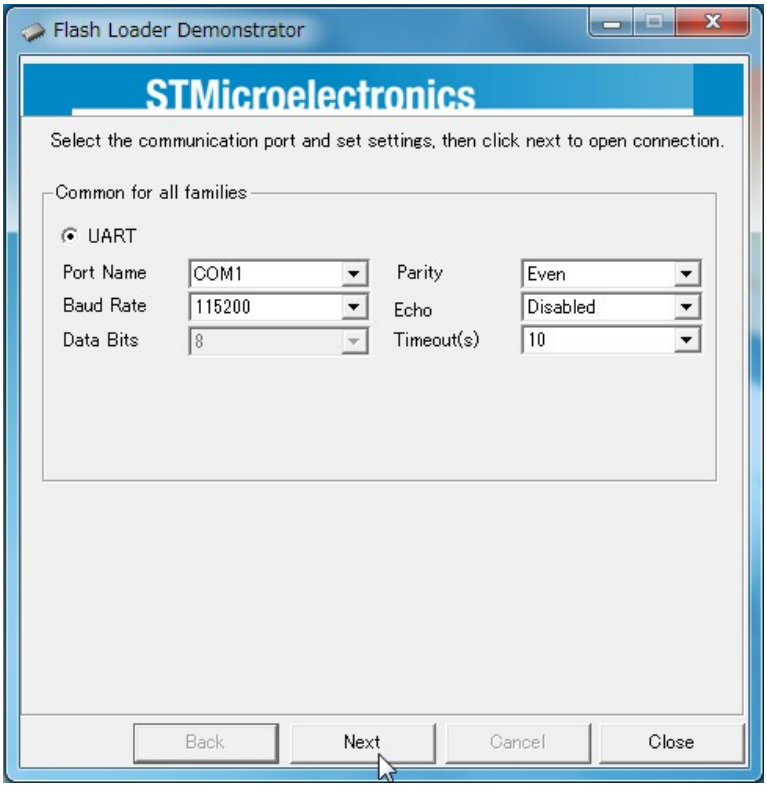


2. KCB-5 の BOOT 端子にジャンパピンを取付けた状態で 6V～12V 直流電源を接続し電源を入れ、書き込みモードにします。バッテリーの場合は KCB-5 対応のものを使用してください。使用するケーブルをコネクターにすべて接続してから通電してください。



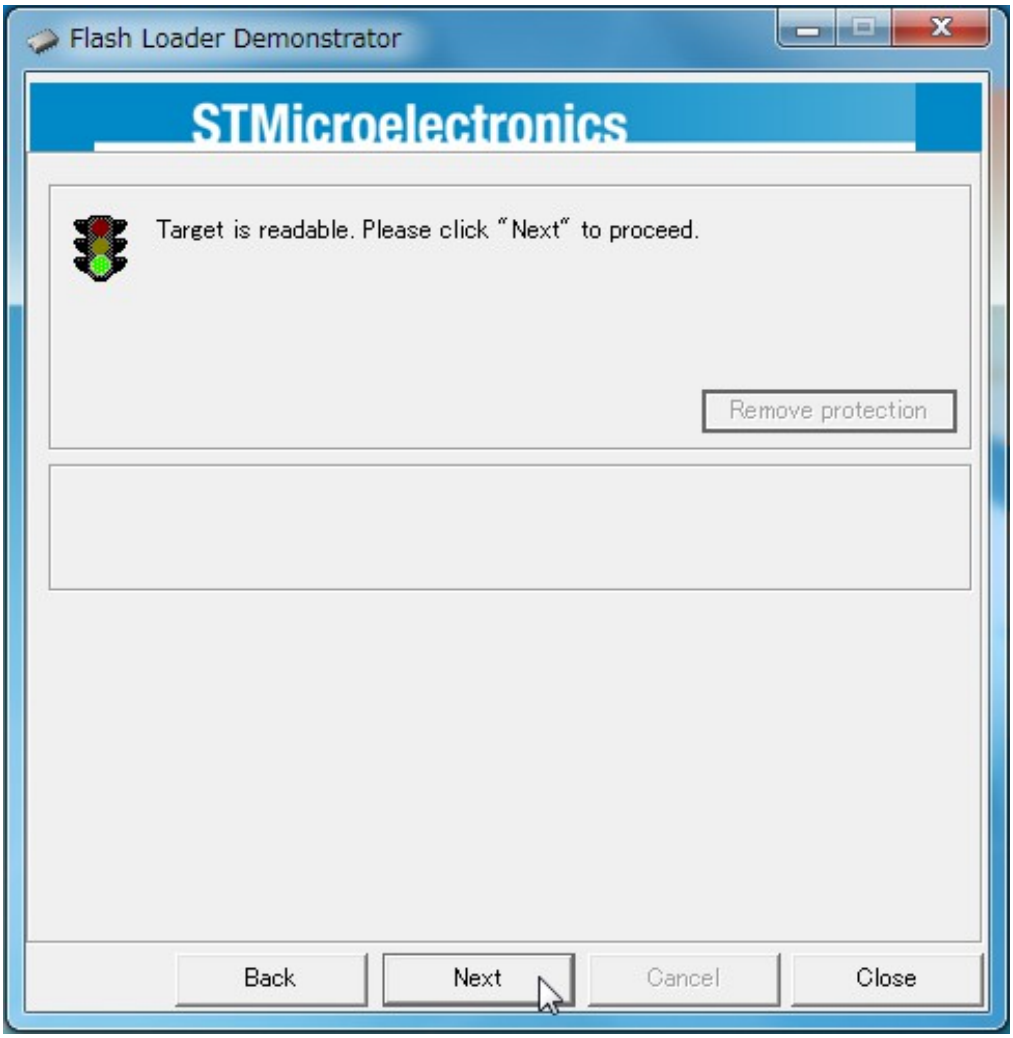
3. Flash loader Demonstrator（STMicroelectronics flash loader.exe をダブルクリックして）起動します。

4. 通信設定を行います。



- Port Name： Dual USB アダプター HS で接続した COM ポート番号を選択します。COM ポート番号を調べるには、Dual USB アダプター HS 付属のマニュアルを参考にしてください。
- Baud Rate： 115200
- Parity： Even
- Echo: Disable
- Timeout(s): 10
- 完了したら、Next ボタンを押します。

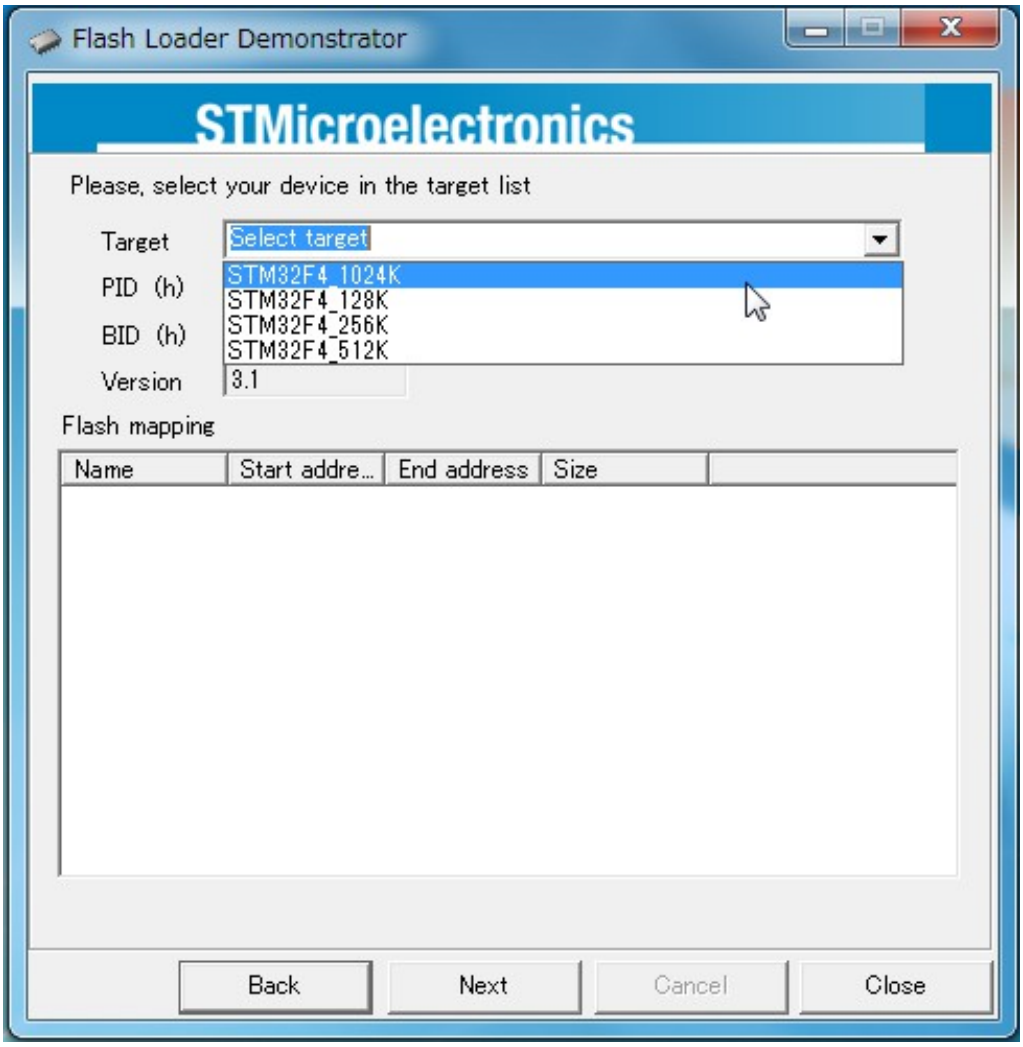
5. デバイス認識画面が表示されます。



● 通信が正常であれば、特に何もすることはありませんので、Next ボタンを押します。

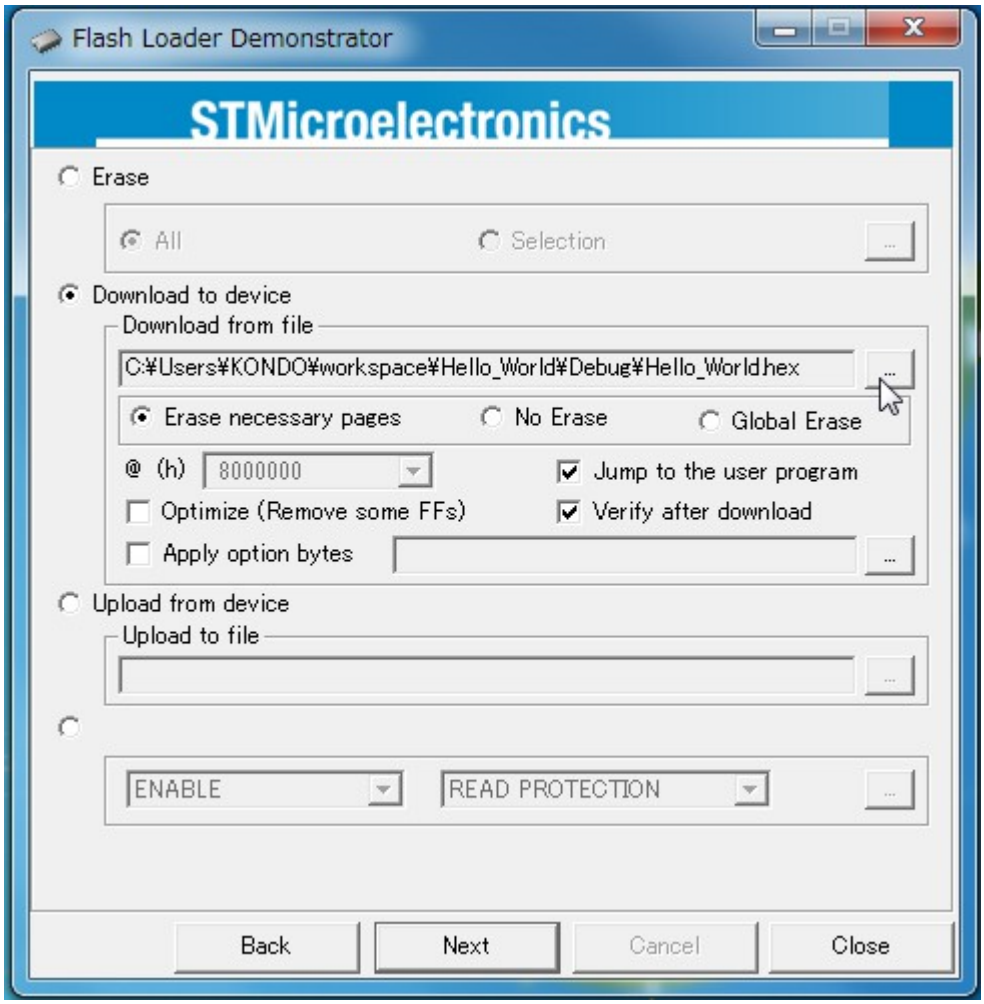
※うまくいかない場合は 27 ページ(書き込みが失敗した時の対処法)を参照してください。

6. デバイスの種類を指定します。



● Select target： STM32F4_1024K を選択します。

7.　書き込むプログラムを選びます。



● Download to device を選択：　プログラムをマイコンに書き込みます。

● Erase necessary pages を選択：　プログラムの書込みに必要な領域だけ消去します。

● Jump to the user program にチェック:　プログラムの書込み完了後に、プログラムを実行します。

● Verify after download にチェック:　プログラムの書込み完了後に、プログラムが正常に書込まれたか、照合（ベリファイ）します。

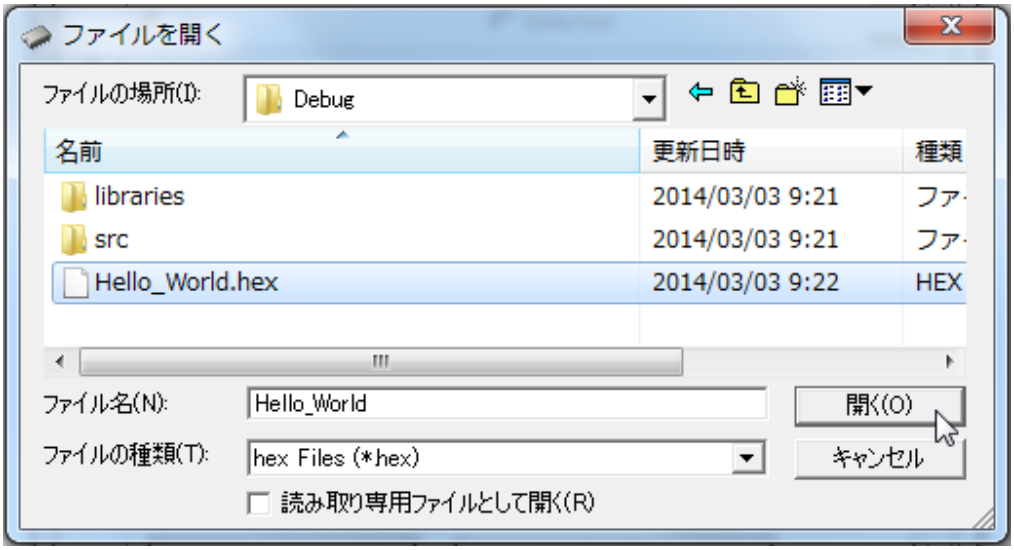
● 上図のカーソルがある「...」ボタンを押して、HEX ファイルを選択します。（次ページ参照）
（書込むプログラムが選択されていれば、再度選択する必要はありません）

● 設定がすべて完了したら、Next ボタンを押します。

※その他の選択肢説明

○ Erase：　プログラムの消去を行います。

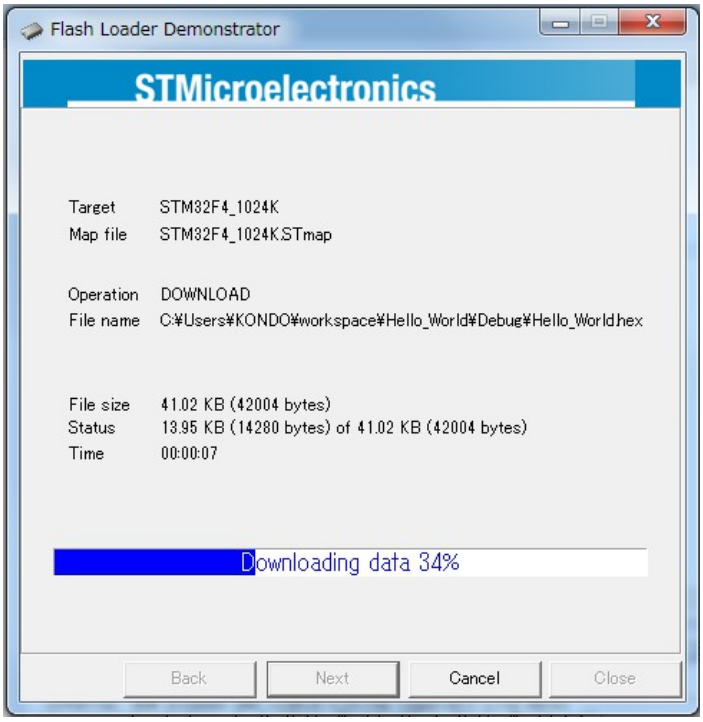
○ Upload from device：　現在マイコンに書込まれているプログラムを読出し、HEX ファイル形式で保存します。



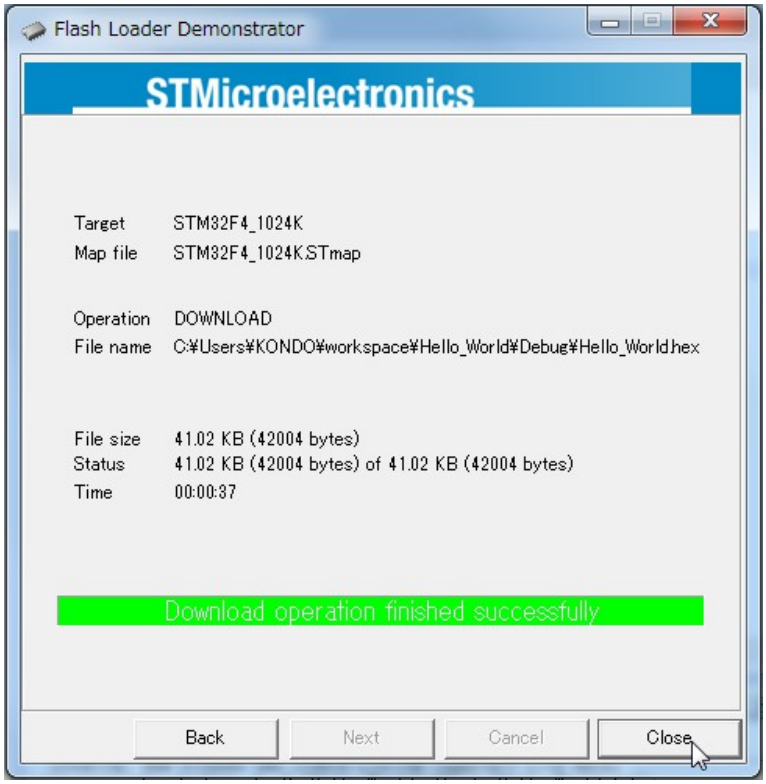
- ファイルの種類(T): hex Files (*.hex)
- ファイル名(N): 書込むプログラムを選択します。

  workspase¥Hello_World¥Debug¥Hello_World.hex

8. 書き込みが始まります。



● プログラムの消去 → プログラムの書き込み → プログラムの照合(ベリファイ)の順に自動で行われます。



● 書き込みが正常終了すると、「Download operation finished successfully」と表示されます。終わったら「Close」ボタンで Flash Loader Demonstrator プログラムを終了させます。

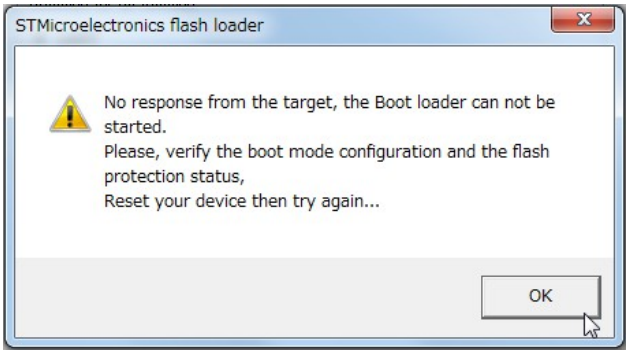
※備考：Jump to the user program にチェックが入っていない場合は、ひとつ前の画面に戻ります。

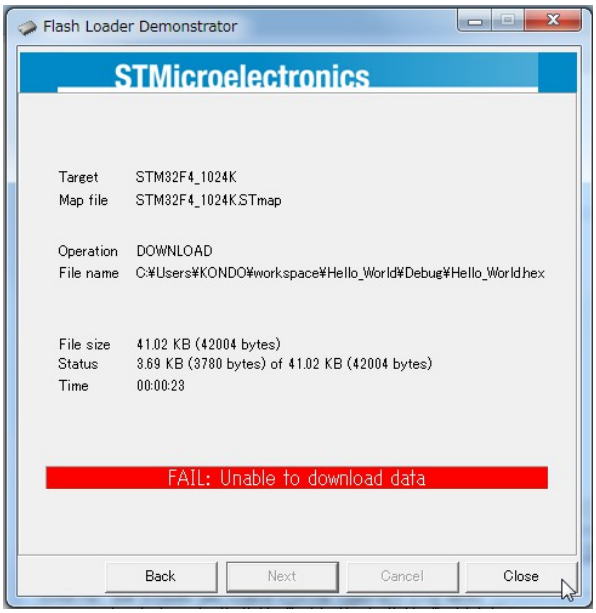
※失敗した時は次ページの項目を確認してください。

● これでプログラムの書き込みが完了です。次にプログラムを実行します。

書き込みが失敗した時の対処法

書き込みに失敗したり通信がうまくいかない場合は、以下の項目を確認してください。



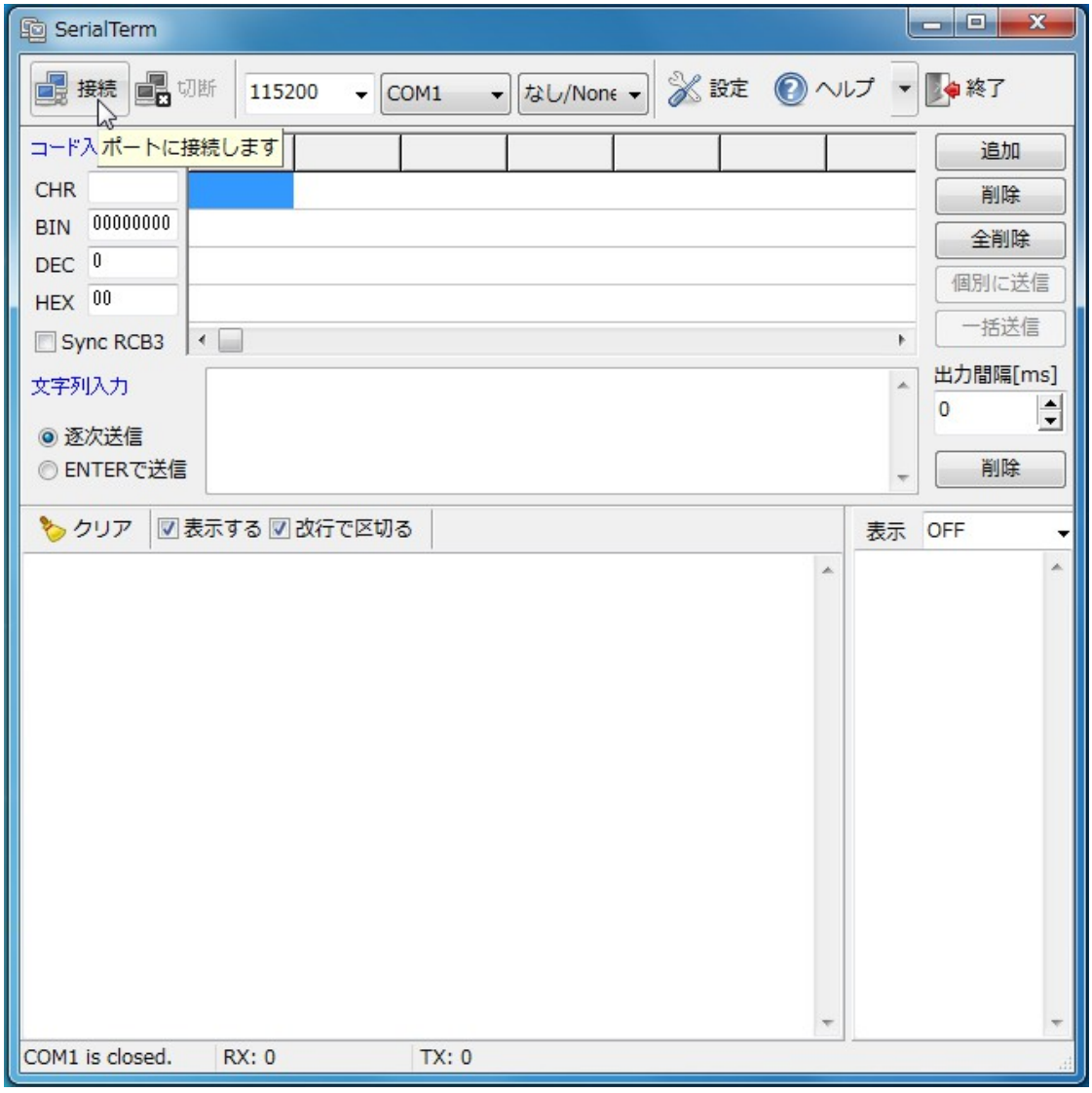


- Dual USB アダプター HS の接続状態を確認し、Dual USB アダプター HS が使用している COM ポート番号を正しく選択します。
- Dual USB アダプター HS がシリアルモードになっているか確認します。
- COM ポートが他のプログラムで使用されていないか確認します。
- 電源が正しく供給されている状態か、確認をします。KCB-5 に電源が供給されていない場合は、電源を供給します。
- リセット状態か、確認をします。リセット状態では動作しないため、リセット状態を解除します。
- 書き込みモードになっているか確認します。WRITE 端子をショートさせ書き込みモードにし、電源を入れます。電源を入れた後で WRITE 端子をショートさせた場合は、WRITE 端子をショートさせたまま、一度リセットするか、電源を入れ直します。

3.5. 実行

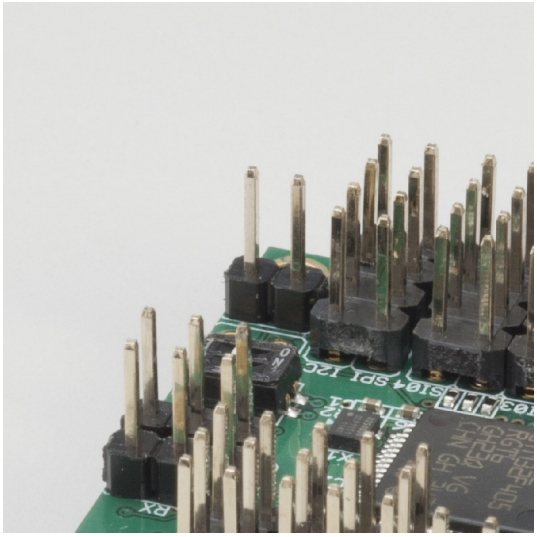
ついにプログラムの実行です。

1. Dual USB アダプター HS（シリアルモード）をコンピューターと KCB-5 に接続します。
2. シリアル通信ソフトウェア（SerialTerminal.exe など）起動し、プログラムと同じ通信条件を設定の上、受信状態にしておきます。

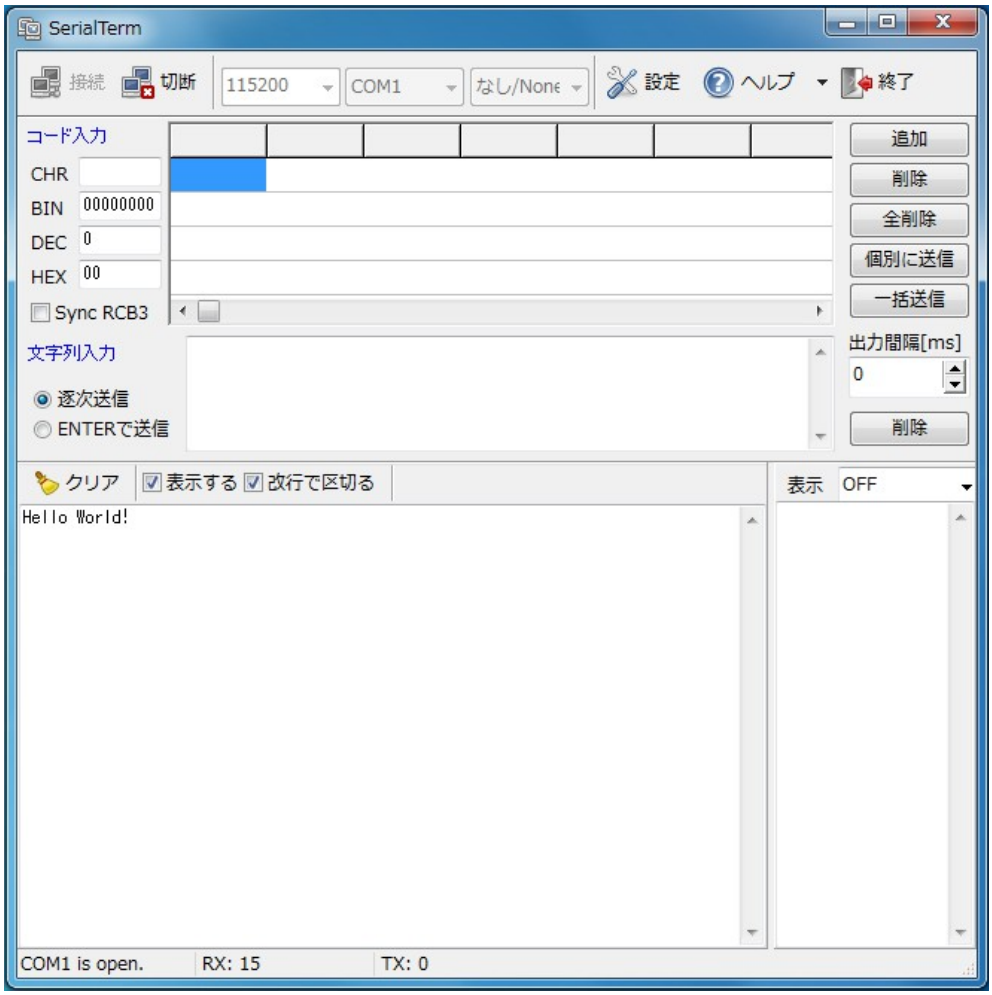


- ボーレート（Baud rate）： 115200
- COM ポート番号： デバイスマネージャで「ポート（COM と LPT）」の「DUAL USB ADAPTER HS」から COM ポート番号を確認します
- パリティ（Parity）： なし/None
- 設定後に「接続」ボタンを押します。

3. KCB-5 の BOOT 端子のジャンパーを外してから、KCB-5 を一度電源を入れ直します。



4. 電源を入れると自動的にプログラムが実行されます。シリアル通信ソフトウェアの画面に「Hello World!」と表示されたら成功です！

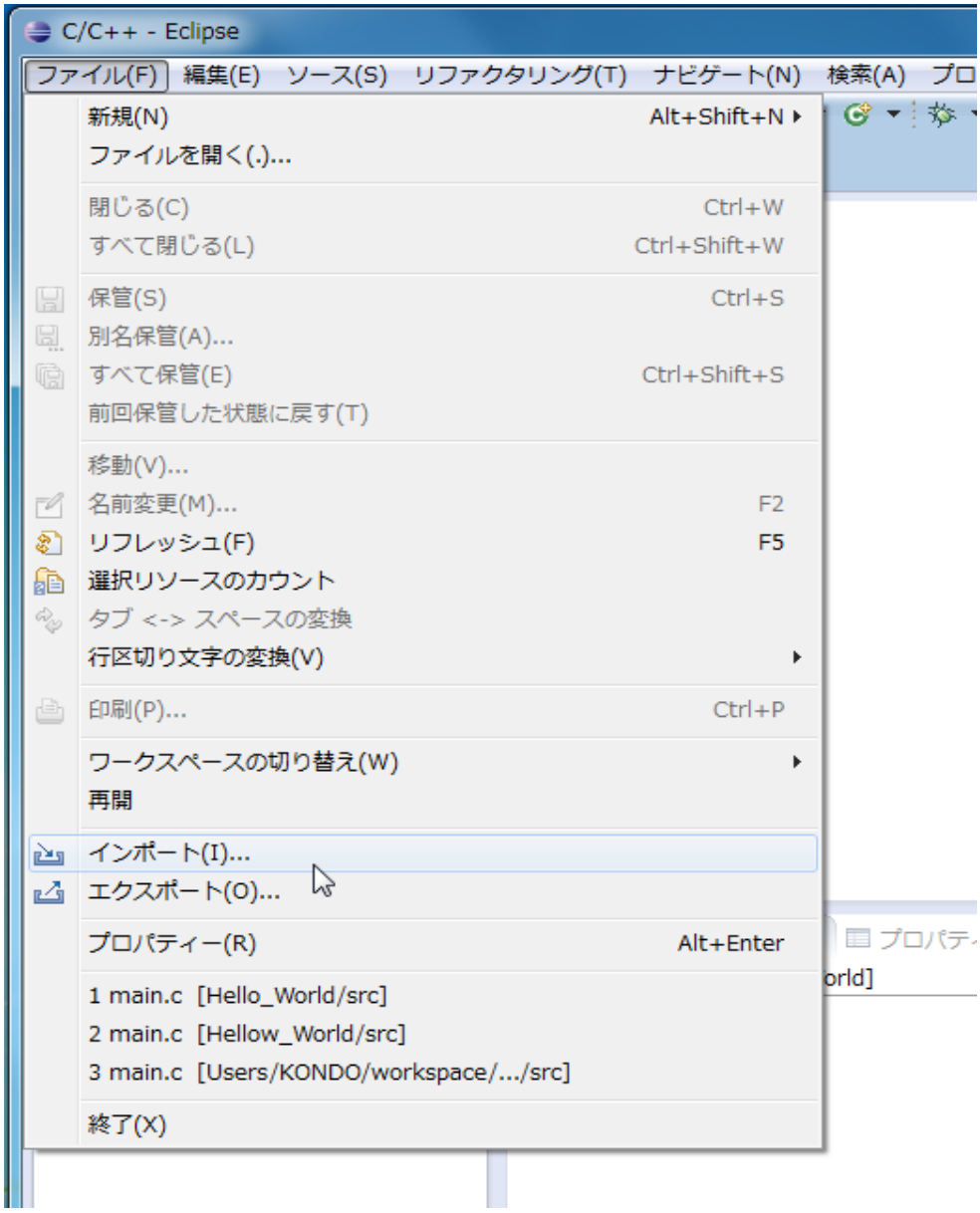


3.6. プロジェクトのインポート

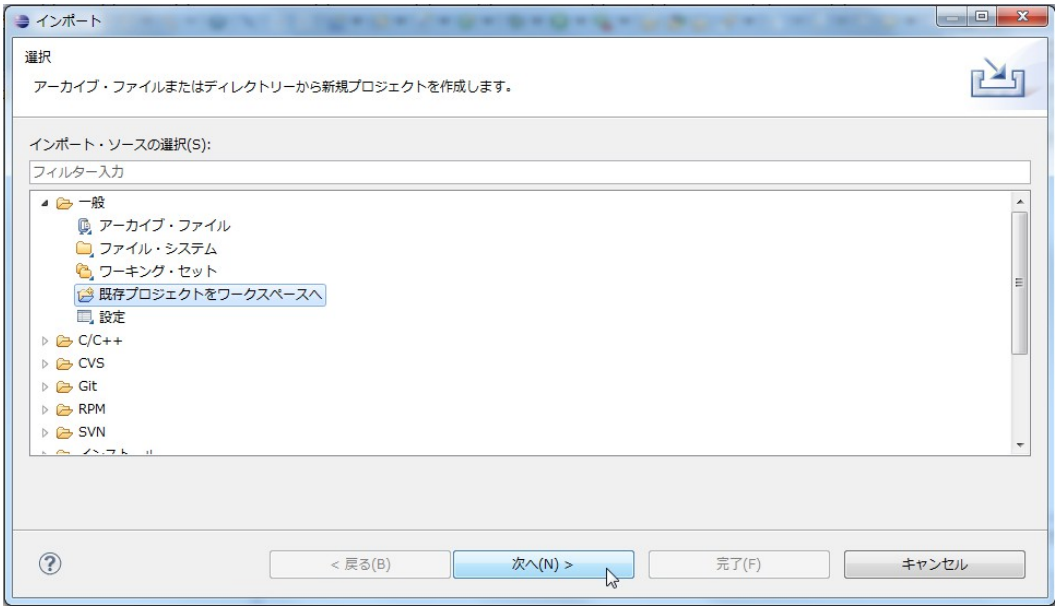
この章ではプロジェクトをインポートする方法を説明します。

※インポートするプロジェクトは、zip にて圧縮された状態である必要があります。
zip を解凍せずにご利用ください。

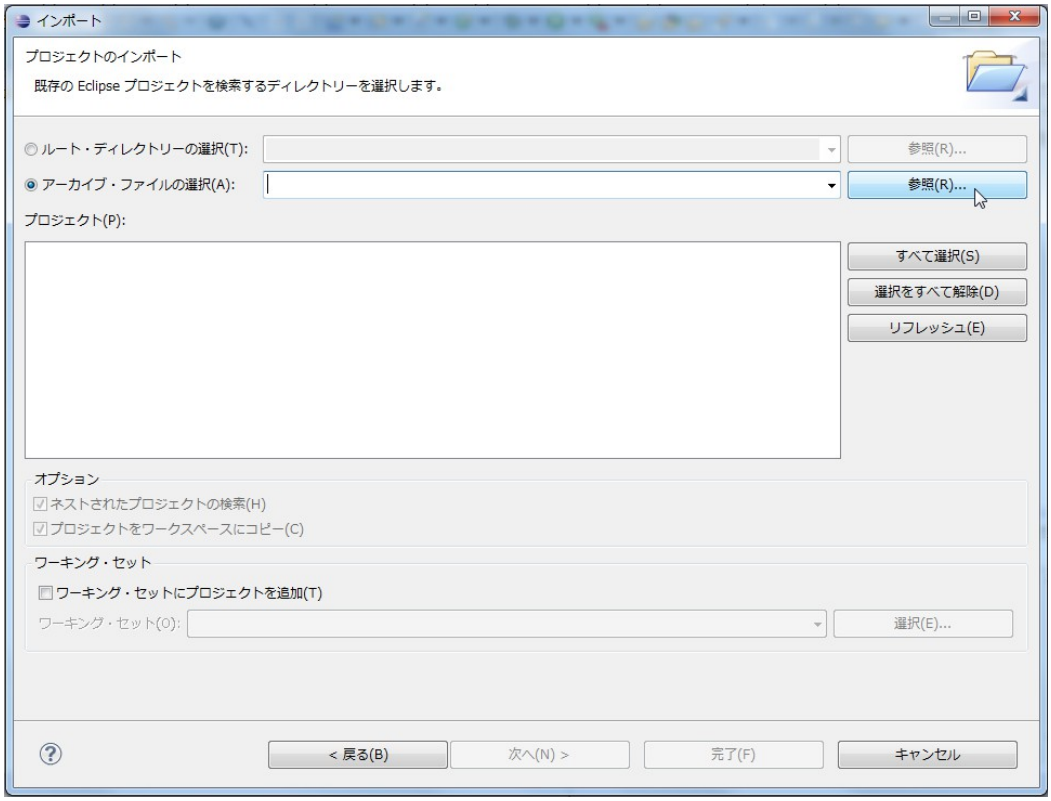
1. ファイルメニューから「インポート」を選択します。



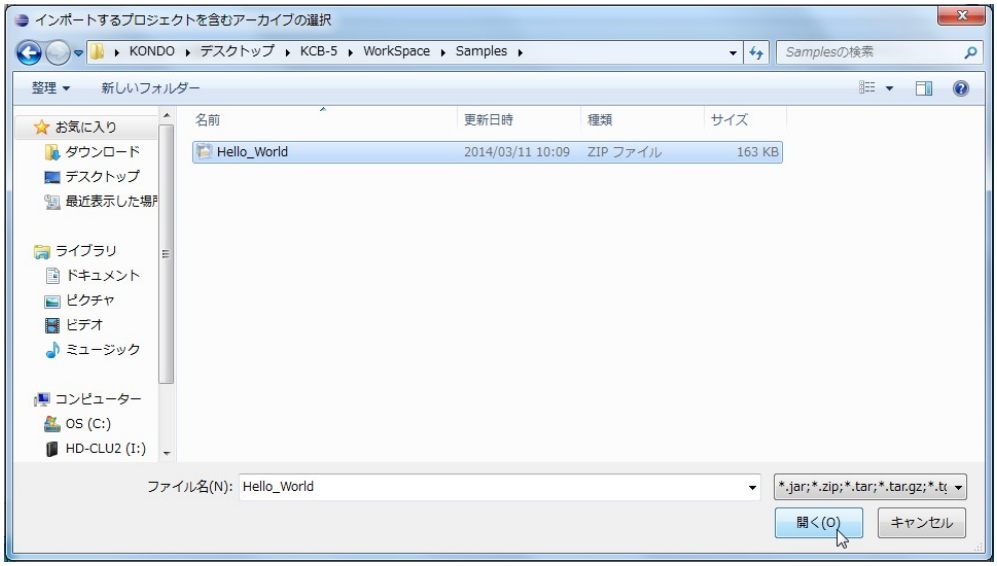
2. 開かれたインポートダイアログで「一般」を展開し「既存プロジェクトをワークスペースへ」を選択し、「次へ」ボタンを押します。



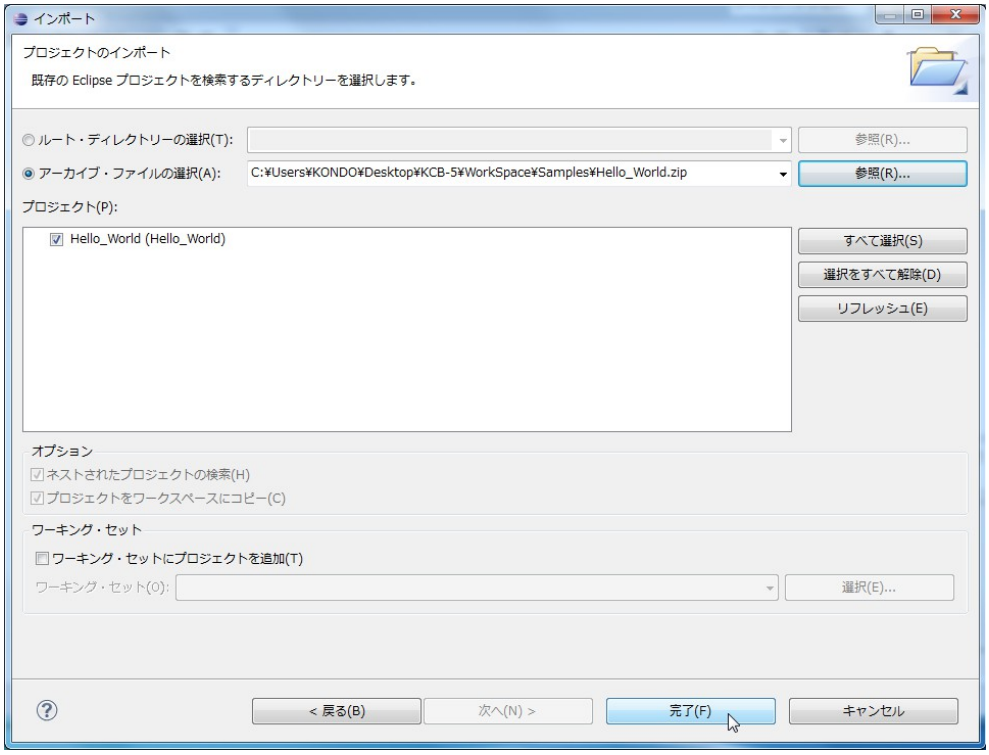
3. 「アーカイブ・ファイルの選択」を選択。「参照」ボタンを押してください。



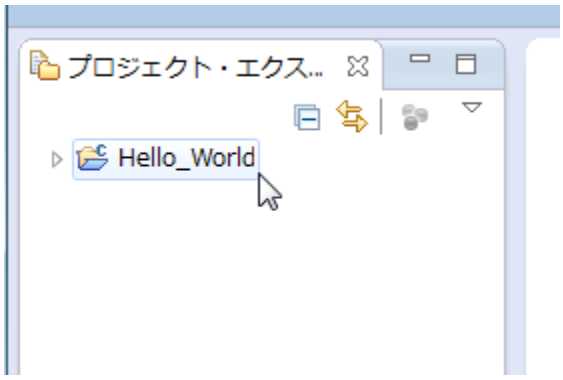
4. 「インポートするプロジェクトを含むアーカイブの選択」ダイアログにて、インポートする zip ファイルのプロジェクトをを選択してください。



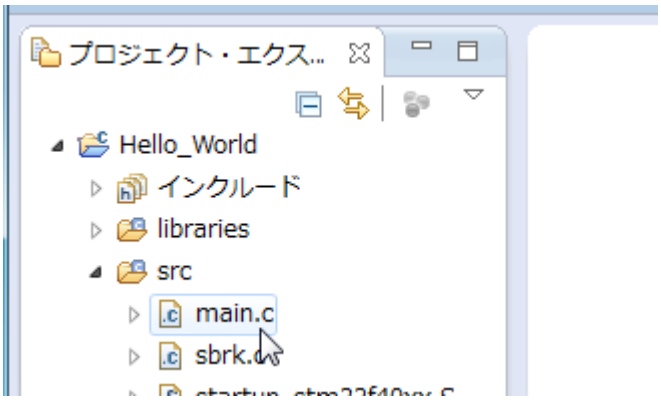
5. 選択が完了したら「プロジェクト」欄に指定のプロジェクト名があり、チェックボックスにチェックが入っているかを確認し、「完了」ボタンを押します。



6. ダイアログが閉じ、「プロジェクト・エクスプローラ」ウィンドウにプロジェクト名がありましたら成功です。



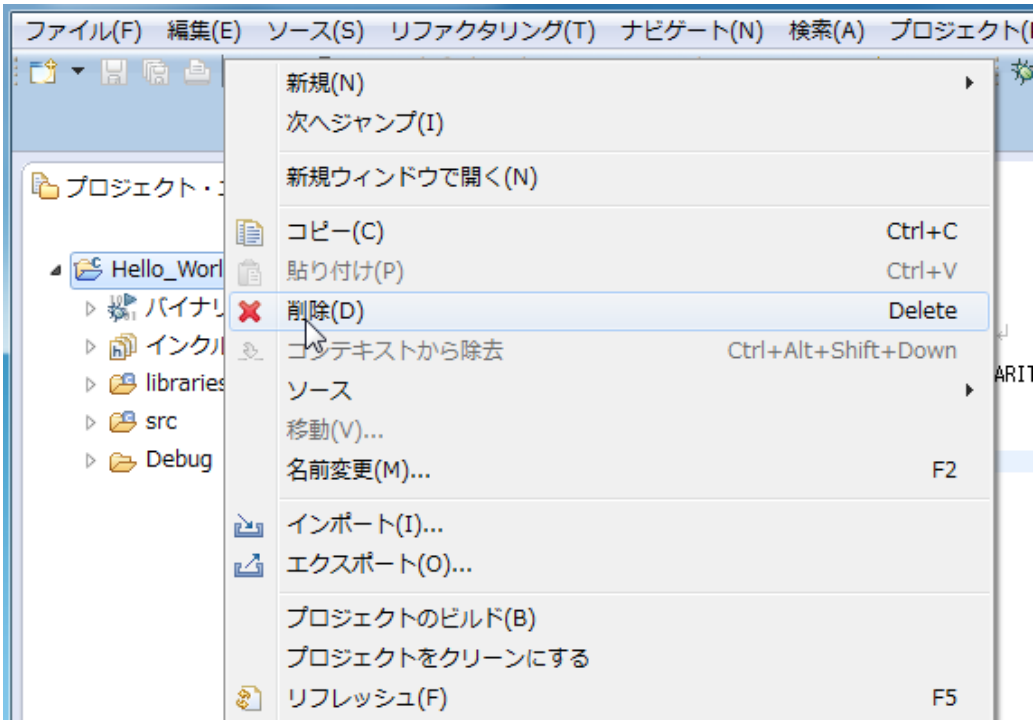
7. プログラムを開始するには、プロジェクト名の横にある矢印マークを押して展開し、「src」フォルダの"main.c"を選択してください。



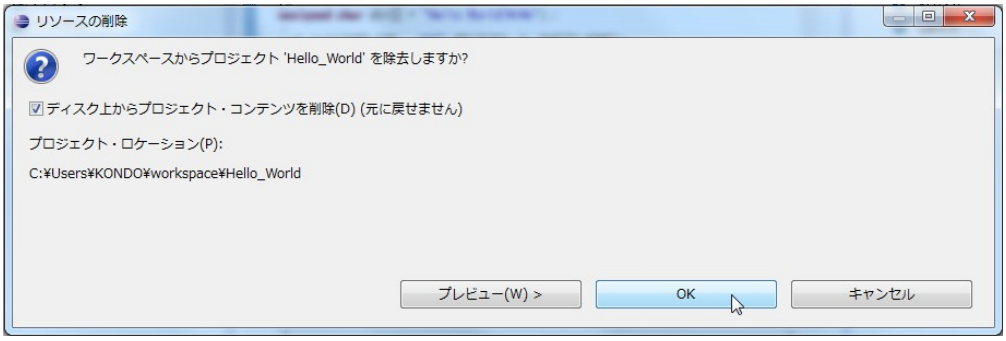
3.7. プロジェクトを削除する

1. プロジェクト・エクスプローラーウィンドウ上にある、削除するプロジェクトを選択し、マウスの右クリックを押します。

2. メニューから「削除」を選択します。



3. 「ディスク上からプロジェクト・コンテンツを削除(D)」にチェックを入れ、OK ボタンを押します。



※こちらの作業で削除したプロジェクトは、元に戻すことはできません。

4. KCB-5 SDK 関数一覧

本節ではユーザーが使用するC言語関数を取り上げ、ヘッダファイルごとに仕様説明を行います。
サンプルをプロジェクトごと配布しますので、そちらも参照してください。

サンプルプロジェクト一覧

機能	名前	内容
ポートの設定関数	PIO_InOut	DIP スイッチを元に LED を点灯させる。
UART 関数	Hello_World	COM 経由で文字列を送信する。
AD 関数	AD_Oneshot	AD1 の変換結果をワンショットで取得する。
	AD_Sweep	AD1 の自動で変換された結果を取得する。
D/A 関数	DAC_Out	のこぎり波を DAC から出力する。
ROM 関数	ROM_ReadWrite	内部 ROM にデータを保存して読み出す。
I2C 関数	I2C_LCD	I2C の LCD に文字を表示する。
	I2C_EEPROM	I2C の外部 EEPROM にデータを保存し読み出す。
TIMER/PWM 関数	PWM_Out	20ms 周期の PWM 波形を出力する。
	Timer_Interrupt	タイマ割り込みを入れ、COM にデータを出力する。
ICS 関数一覧	SIO_Ad_Control	AD1 のデータを元に ID0 のサーボを動かす。
	ICS_Parameter	ICS サーボのパラメータを取得、設定を変更する。

4.1. KCB-5

KCB-5 関数一覧 (kcb5.h)

このファイルは、設定が書かれてますので、必ずインクルードしてください。

4.2. ポートの設定

ポートの設定関数一覧 (pio.h) サンプルプログラム:PIO_InOut

定義

```
#define byte     unsigned char
#define size_t    int                    // 互換性のため

#define HIGH    (1)
#define LOW     (0)
```

ポート名指定用定義

```
// Timer
#define PIO_T1                  (0x00)        // PA0
#define PIO_T2                  (0x01)        // PA1
#define PIO_T3                  (0x06)        // PA6
#define PIO_T4                  (0x07)        // PA7
#define PIO_T5                  (0x10)        // PB0
#define PIO_T6                  (0x11)        // PB1
// DAC
#define PIO_DAC                 (0x04)        // PA4
// AD
#define PIO_AD1                 (0x20)        // PC0
#define PIO_AD2                 (0x21)        // PC1
#define PIO_AD3                 (0x22)        // PC2
#define PIO_AD4                 (0x23)        // PC3
#define PIO_VDD                 (0x24)        // PC4
//SW
#define PIO_SW1                 (0x2E)        // PC14
#define PIO_SW2                 (0x2F)        // PC15
// LED
#define PIO_LED1                (0x1D)        // PB13(Red LED)
#define PIO_LED2                (0x1C)        // PB12(Green LED)
// I2C
#define PIO_I2C_SCL             (0x18)        // PB8
#define PIO_I2C_SDA             (0x19)        // PB9
// SPI
#define PIO_SPI_MOSI            (0x15)        // PB5
#define PIO_SPI_MISO            (0x14)        // PB4
#define PIO_SPI_SCK             (0x13)        // PB3
#define PIO_SPI_NSS             (0x0F)        // PA15
// SIO
#define PIO_SIO1_TX             (0x1A)        // PB10
#define PIO_SIO1_RX             (0x1B)        // PB11
#define PIO_SIO2_TX             (0x2A)        // PC10
#define PIO_SIO2_RX             (0x2B)        // PC11
#define PIO_SIO3_TX             (0x2C)        // PC12
#define PIO_SIO3_RX             (0x32)        // PD2
#define PIO_SIO4_TX             (0x26)        // PC6
#define PIO_SIO4_RX             (0x27)        // PC7
```

pio_init

書式

```
_Bool pio_init(int port,int direction)
```

機能　PIO ポートの初期化と入出力の方向を決めます。

引数　int port　　ポート番号は上記定義より選択してください。通常の機能が割り当てられているポートも PIO ポートに変更できます。

　　int direction　　入出力方向は PIO_SET_IN または PIO_SET_OUT のいずれかより選んでください。

返値　引数の間違いがあった場合に false を返します。

pio_read

書式

```
int pio_read(int port)
```

機能　指定ポートからHIGH/LOWの2値を読み取ります。
引数　int port　　初期化したポートより選択します。
返値　読み取った2値いずれかをHIGH(1)またはLOW(0)で返します。

pio_write

書式

```
int pio_write(int port,int value)
```

機能　指定ポートからHIGH/LOWの2値いずれかを出力します。
引数　int port　　ポート番号。
　　int value　　HIGH(1)またはLOW(0)のいずれかを出力できます。
返値　引数の間違いがあった場合には-1を返します。

4.3. UART

UART 関数一覧 (uart.h) サンプルプログラム:Hello_World

定義

```
#define BR1200                  (1200)
#define BR2400                  (2400)
#define BR4800                  (4800)
#define BR9600                  (9600)
#define BR19200                 (19200)
#define BR28800                 (28800)
#define BR38400                 (38400)
#define BR57600                 (57600)
#define BR115200                (115200)
#define BR625000                (625000)
#define BR1000000               (1000000)
#define BR1250000               (1250000)
#define BR2500000               (2500000)
#define BR3000000               (3000000)
#define BR3125000               (3125000)

#define PARITY_NONE                     (0)
#define PARITY_ODD              (1)
#define PARITY_EVEN                     (2)

#define UART_COM                (1)
#define UART_RX                 (2)
#define UART_SIO1               (3)
#define UART_SIO2               (4)
#define UART_SIO3               (5)
#define UART_SIO4               (6)

typedef enum {
        UART = 0,
        RS485 = 1,
        ICS = 2
} uart_mode;
```

uart_init

書式

```
_Bool uart_init(int port, uart_mode mode, unsigned int baudrate, unsigned
int data, unsigned int parity)
```

機能　通信ポートを UART（TX/RX）として初期化します。

引数　int port　　ポートは UART_COMUART_RX（受信専用）、
UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選択できます。

enum mode　　モードは UART、ICS から選択できます。

int baudrate　　通信速度指定です。上記定義を参照してください。

int data　　データビット長です。

int parity　　パリティは PARITY_NONE、PARITY_ODD、PARITY_EVEN から選択できます。

返値　常に true を返します。

モードについて

UART　　非同期式通信用設定

ICS　　弊社製 ICS サーボ用の設定

※注意：UART_SIO 端子を UART モードで使用する場合は通常のコネクタは使用できません。
電源やグランド端子を確認してください。

uart_tx

書式

```
_Bool uart_tx(int port, unsigned char* tx, int start, int length)
```

機能　データ配列を length だけ送信します。

引数　int port　　ポートは UART_COM、UART_RX（受信専用）、UART_SIO1、UART_SIO2、UART_SIO3、
UART_SIO4 から選択できます。uart_init で初期化したポートを選択してください。

unsigned char* tx　　受信下データを格納する配列のアドレスを指定してください。

int start　　送信開始アドレス（配列インデックス）

int length　　送信数

返値　送信数よりスタートアドレスが大きい場合は false を返します。

uart_rx

書式

```
_Bool uart_rx(int port, unsigned char* rx, int length, unsigned long timeout)
```

機能　指定ポートより length だけデータを受信します。

引数　int port　　ポートは UART_COM、UART_RX（受信専用）、UART_SIO1、UART_SIO2、
UART_SIO3、UART_SIO4 から選択できます。uart_init で初期化したポートを選択し てください。

unsigned char* rx　　受信したデータを格納する配列のアドレスを指定してください。

int length　　受信数

unsigned long timeout　　受信タイムアウト(ループのカウント数)

返値　データを指定数だけ受信したら true、その他の理由により送信失敗時に false を返します。

備考　timeout に 0 を指定すると無制限で待ちます。
timeout 間待ってもデータが受信できなかった場合はグローバル変数 false を返します。

4.4. AD 変換

AD 関数一覧 (ad.h)　サンプルプログラム：AD_Oneshot AD_Sweep

ad_init

書式

```
_Bool ad_init(int port,enum mode)
```

機能　AD 変換方法を決定し、初期化を行ないます。

引数　int port　ポート番号。ポート番号は PIO_AD1、PIO_AD2、PIO_AD3、PIO_AD4、PIO_VDD のうちいずれかより選択します。PIO_VDD は電源の電圧でポートは存在しません。

enum mode　モード。モードは ONESHOT または SWEEP から選択します。ONESHOT は後述の ad_read 関数を呼び出すたびに AD 変換が行われます。SWEEP はメインの処理とは別に自動で AD 変換を行うモードで、ad_read 関数を呼び出したときは保存されている変換値を返すだけになります。

返値　ポート初期化に成功したら true を返します。引数が間違っていた場合には false を返します。

ad_read

書式

```
int ad_read(int port)
```

機能　指定ポートから AD 値を読み取ります。

引数　int port　ポート番号。ポート番号は PIO_AD1、PIO_AD2、PIO_AD3、PIO_AD4、PIO_VDD のうちいずれかより選択します。PIO_VDD は電源の電圧でポートは存在しません。また PIO_VDD は入力値を 1/5 にしています。実際の値は 5 倍にしてください。

返値　読み取った AD 値を返します。AD 値は 12 ビット（0～4095）です。

備考　・5 V 入力で 12bit（4096 段階）で返ってきますが、CPU の関係で分圧を内部で行なっています。
接続する機器によって値が変わる場合もあります。
・PIO_VDD は 16.17V の時 4095 になります。

4.5. D/A 変換

D/A 関数一覧 (dac.h)　サンプルプログラム：DAC_Out

dac_init

書式

```
void dac_init()
```

機能　DAC ポートを有効にします。
引数　なし
返値　なし
備考　デジタル12 bit をアナログ値に変換し、出力するようにポートの初期化を行います。

dac_write

書式

```
int dac_write(unsigned short value)
```

機能　DAC ポートからアナログ電圧を出力します。
引数　unsigned short value　16bit デジタル値（0～65535）が指定できます。内部で 12bit に変換されますので、DAC の解像度は 12bit です。
返値　常に-1 を返します。

4.6. ROM

ROM 関数一覧 (rom.h)　サンプルプログラム：ROM_ReadWrite

rom_erase

書式

```
_Bool rom_erase(unsigned int block)
```

機能　ROM データをブロック消去。
引数　unsigned int block　ROM のブロックです。ROM_BLOCK1 または ROM_BLOCK2 のいずれかを選択します。
返値　消去に失敗した時は false を返します。
備考　KCB-5 の ROM は Flash メモリーですので、消去する際はブロック単位でまとめて消去する必要があります。

rom_read

書式

```
unsigned long rom_read(unsigned int address,unsigned int block,byte*data,size_t size)
```

機能　ROM からデータを読み取ります。
引数　int address　　ROM データアドレス。アドレスは 0～131070 を 2 ブロックから選択できます。
　　unsigned int block　ROM のブロックです。ROM_BLOCK1 または ROM_BLOCK2 のいずれかを選択します。
　　byte *data　　読み出したデータを格納する配列、ユーザーが確保します。
　　size_t size　　読み出すデータ数です。0～131070 の範囲で選べます。
返値　読み出しに成功したデータバイト数が返ります。

rom_write

書式

```
unsigned int rom_write(unsigned int address,unsigned int block,byte*data,size_t size)
```

機能　ROM にデータを書き込みます。
引数　int address　　ROM データアドレス。アドレスは 0～131070 を 2 ブロックから選択できます。
　　unsigned int block　ROM のブロックです。ROM_BLOCK1 または ROM_BLOCK2 のいずれかを選択します。
　　byte *data　　書き込むデータ配列です。
　　size_t size　　書き込むデータ数です。0～131070 の範囲で選べます。
返値　書き込みに成功したデータバイト数が返ります。
備考　一度書き込んだアドレスは消去してから書き込む必要があります。上書きはできません。

4.7. I2C

I2C 関数一覧 (i2c.h)　サンプルプログラム：I2C_LCD　I2C_EEPROM

i2c_init

書式

```
_Bool i2c_init(int clock,i2c_mode mode)
```

機能　i2c ポートを初期化します。

引数　int clock　　デバイスに合わせてクロックを指定します。例として 400kHz の場合は 400000 と指定します。

　　enum mode　　マスターとして使用する場合は I2C_MASTER、スレーブとして使用するには I2C_SLAVE を指定します。

返値　true のみ返ります。

i2c_write

書式

```
int i2c_write(unsigned char i2c_address,unsigned char address,byte*data,syze_t size)
```

機能　i2c ポートからデータを書き込みます。

引数　unsigned char i2c_address　　i2c アドレス

　　int address　　i2c デバイスのアドレス

　　byte *data　　書き込み用データ

　　size_t size　　書き込み用データサイズ

返値　常に 0 が返ります。

備考　デバイスがない場合等で返事が返ってこない場合は関数内部で待ち続けます。

i2c_read

書式

```
int i2c_read(int i2c_address,byte *command,size_t c_size,byte *data,size_t r_size)
```

機能　i2c ポートからデータを読み込みます

引数　int i2c_address　　i2c アドレス

　　byte *command　　データを読むために必要なコマンド配列

　　size_t c_size　　データコマンドのサイズ

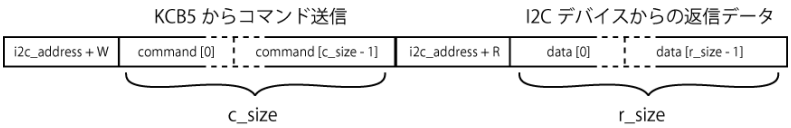
　　byte *data　　読み取ったデータを格納する配列

　　size_t r_size　　読み取りデータサイズ

返値　常に 0 が返ります。

備考　デバイスがない場合等で返事が返ってこない場合は関数内部で待ち続けます。

i2c_read イメージ図



※ACK、NACK、START、STOP は省略してあります

4.8. TIMER/PWM

TIMER/PWM 関数一覧 (timer.h)　サンプルプログラム：PWM_Out　Timer_Interrupt

定義

```
#define TIMER             (0xFF)    // ポート番号に指定するため
#define TIMER_INT         (0xAA) //ポートを指定せず、一定時間割り込みにつかう
#define TIMER1            (0x00) // PA0
#define TIMER2            (0x01) // PA1
#define TIMER3            (0x06) // PA6
#define TIMER4            (0x07) // PA7
#define TIMER5            (0x10) // PB0
#define TIMER6            (0x11) // PB1

// モード1
// モード2
// モード1とORで使用してT端子から出力、入力を決める
// 不可能な組み合わせは初期化関数でエラーを発生させる
// PWMのINPUTはインプット・キャプチャ(開発中)

typedef enum {
        TIMER_MODE_TIMER16  = 0x0100,
        TIMER_MODE_TIMER32  = 0x0200,
        TIMER_MODE_ONEPULSE = 0x0400,
        TIMER_MODE_PWM      = 0x0800,
        TIMER_MODE_OUTPUT   = 0x1000,
        TIMER_MODE_INPUT    = 0x2000,
        TIMER_MODE_INTERRUPT = 0x4000
} timer_mode;
```

timer_init

書式

```
_Bool timer_init(int port,timer_mode, int frequency)
```

機能　タイマーを初期化します。

引数　int port　　ポートは TIMER、TIMER1、TIMER2、TIMER3、TIMER4、TIMER5、TIMER6 から選べます。
TIMER ポートは一般的な内部タイマーのことで、TIMER1～TIMER6 は出力を伴うタイマーを作成する場合に使用します。

enum mode　　モードは
TIMER_MODE_TIMER16、TIMER_MODE_TIMER32、TIMER_MODE_ONEPULSE、TIMER_MODE_PWM、TIMER_MODE_OUTPUT、TIMER_MODE_INPUT から選択できます。
TIMER_MODE_TIMER16/32 は一般的なタイマーでカウンターの最大値が変わります。
TIMER_MODE_PWM は PWM 用のタイマーです。
TIMER_MODE_INPUT/OUTPUT はインプットキャプチャ用タイマーとタイマー出力の指定項目です(開発中)。
TIMER_MODE_INTERRUPT は TIMER4 を一定間隔で割り込みをかけるようにします。

int frequency　　タイマー周期。タイマー周期はマイクロ秒単位で指定します。例えば 1ms のタイマーの場合は 1000 と指定してください。

返値　引数の間違いがあった場合には false を返します。

timer_start

書式

```
int timer_start(int port)
```

機能　指定ポートのタイマーをスタートさせます。
引数　int port　　タイマをスタートさせるポートを指定します。
返値　常に true を返します。

timer_stop

書式

```
int timer_stop(int port)
```

機能　指定ポートのタイマーを停止させます。
引数　int port　　タイマを止めるポートを指定します。
返値　常に true を返します。

timer_read

書式

```
unsigned int timer_read  (int port)
```

機能　指定ポートのタイマー値を読み取ります。
引数　int port　　読み取るポートを選択します。
返値　指定されたポートの現在のタイマ値を返します。

timer_write

書式

```
_Bool timer_write  (int port, unsigned int data)
```

機能　指定ポートのタイマ値を書き込みます。
引数　int port　　指定するポート
　　unsigned int data　　書き込むタイマ値
返値　常に true を返します。

timer_interrupt_set

書式

```
void timer_interrupt_set(void (*address)(void))
```

機能　TIMER4 を用い一定間隔で割り込みがかかる関数を指定します
引数　void (*address)(void)　　割り込みをかける関数のポインタを渡します

4.9. ICS

ICS 関数一覧 (ics.h) サンプルプログラム:SIO_Ad_Control ICS_Parameter

定義

ICS モード定義

```
#define ICS20                   (20)
#define ICS21                   (30)
#define ICS22                   (35)

#define ICS_MAX_ID              (32)
#define ICS_MAX_POS_VALUE       (0x3FFF)
#define ICS_POS_CMD             (0x80) // ポジション設定コマンド
#define ICS_POS_BYTE            (3)
#define ICS_GET_PARAM_CMD       (0xA0) // パラメータ読み書きコマンド
#define ICS_SET_PARAM_CMD       (0xC0)
#define ICS_EEPROM_SC           (0)    // パラメータ読み書きサブコマンド
#define ICS_STRC_SC             (1)    // STRETCH 値読み書きサブコマンド
#define ICS_SPD_SC              (2)    // SPEED 値読み書きサブコマンド
#define ICS_CURNT_SC            (3)    // 電流値・電流制限値読み書きサブコマンド
#define ICS_TMPR_SC             (4)    // 温度値・温度制限値読み書きサブコマンド
#define ICS_GENE_SC             (0x7F) // 汎用データ読み書きサブコマンド
#define ICS_PARAM_BYTE          (2)    // パラメータ読み書きデータサイズ
#define ICS_EEPROM_BYTE         (60)
#define ICS35_EEPROM_BYTE       (66)
#define ICS_ID_CMD              (0xE0) // ID コマンド
#define ICS_GET_ID_SC           (0)    // ID 取得サブコマンド
#define ICS_SET_ID_SC           (1)    // ID 設定サブコマンド
```

sio_init

書式

```
_Bool sio_init(int port, int baudrate)
```

機能　UART_SIO ポートを ICS サーボ用として初期化します。

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

　　int baudrate　　通信速度は 115200、625000、1250000 から選んでください。サーボモーター側の通信速度を確認してください。

返値　ポート初期化に成功したら true を返します。

sio_tx

書式

```
_Bool sio_tx(int port,unsigned char* tx,int length)
```

機能　データ配列を length だけ送信します。

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

　　unsigned char* tx　　送信データ配列

　　int length　　送信データ配列の配列数を指定します。

返値　常に true が返ります。

sio_rx

書式

```
_Bool sio_rx(int port, unsigned char*rx,int length,unsigned long timeout )
```

機能　指定ポートより length だけデータを受信します。

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

　　unsigned char* rx　　受信データ配列

　　int length　　受信データ数を指定します。

　　unsigned long timeout　　受信タイムアウト(ループのカウント数)

返値　timeout 回待っても受信しない場合 false が返ります。

ics_set_pos

書式

```
int ics_set_pos(int port, byte id, unsigned short pos)
```

機能　ICS サーボモーターのポジションを設定し、現在位置を取得します。

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

　　byte id　　デバイスの ID 番号を指定します(0～31)

　　int pos　　3500～11500 の範囲で位置を指定します。ICS サーボモーターの原点(ニュートラル)に合わせるには、7500 と指定します。

　　0 を送るとサーボモーターが脱力し、現在位置を返します。

返値　ICS サーボモーターの現在位置(3500～11500)が返ります。取得失敗したときには-1 が返ります。

備考　返値が-1 だった場合は ICS サーボモーターの ID 番号や通信速度を確認してください。

ics_get_pos

書式

```
int ics_get_pos(int port, byte id)
```

機能　ICS サーボモーターの現在位置を取得します。

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

　　byte id　　デバイスの ID 番号を指定します(0～31)

返値　ICS サーボモーターの現在位置(3500～11500)が返ります。取得失敗したときには-1 が返ります。

備考　この関数を呼ぶと脱力状態になります。

ics_set_param

書式

```
int ics_set_param ( int port, byte id, byte sc, byte param )
```

機能　ICS サーボモーターのパラメータを設定します

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

byte id　　デバイスの ID 番号を指定します（0～31）

byte sc　　ICS_STRC_SC,ICS_SPD_SC,ICS_CURNT_SC,ICS_TMPR_SC のいずれかから選びます（ICS_EEPROM_SC は書き換え防止のためできません）

byte param　　パラメータの値を入れます

返値　sc で指定したパラメータが返ります。データが返ってこない場合は-1 が返ります。

備考　ICS3.5 とICS 3.0 では設定できるパラメータが違うので注意が必要です。

ics_get_param

書式

```
int ics_get_param ( int port, byte id, byte sc, byte *param, size_t size )
```

機能　ICS サーボモーターのパラメータを取得します

引数　int port　　ポートは UART_SIO1、UART_SIO2、UART_SIO3、UART_SIO4 から選びます。

int id　　デバイスの ID 番号を指定します（0～31）

byte sc　　ICS_EEPROM_SC,ICS_STRC_SC,ICS_SPD_SC,ICS_CURNT_SC,ICS_TMPR_SC のいずれかから選びます

byte *param　　取得パラメータの配列

size_t size　　取得パラメータのバイト数

返値　書き換えに失敗したり、接続していなかった場合は-1 が返ります

備考　ICS3.5 とICS 3.0 では設定できるパラメータが違うので注意が必要です。取得パラメータバイト数は ICS の説明書をご覧ください。

KCB-5 SDK 改訂履歴

2014/03/28　　バージョン 1.0.　プレリリースバージョン

2014/10/25　　バージョン 1.0.0

KCB-5 SDK Manual ver.1.0.0