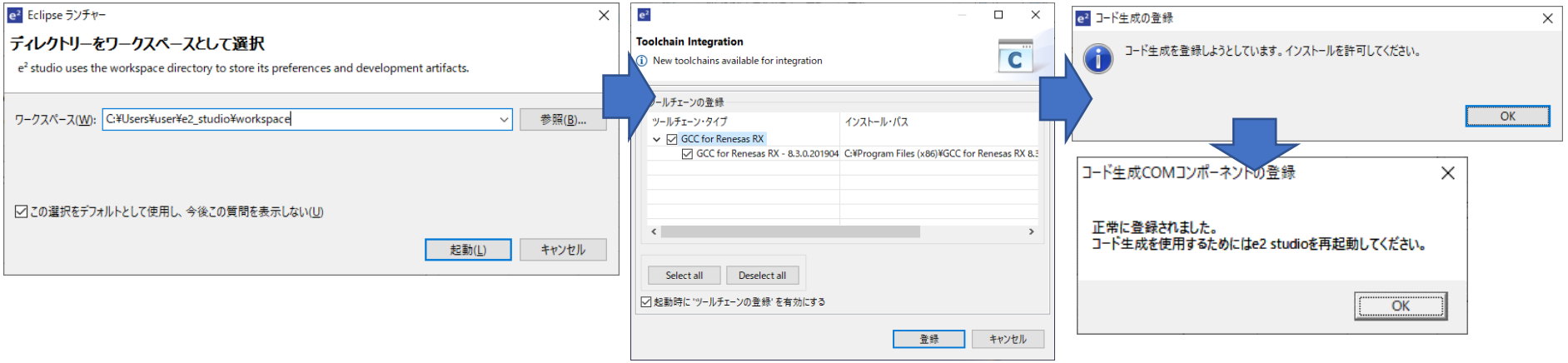
Kernel Software Engineering

iCar - e2 studio入門編

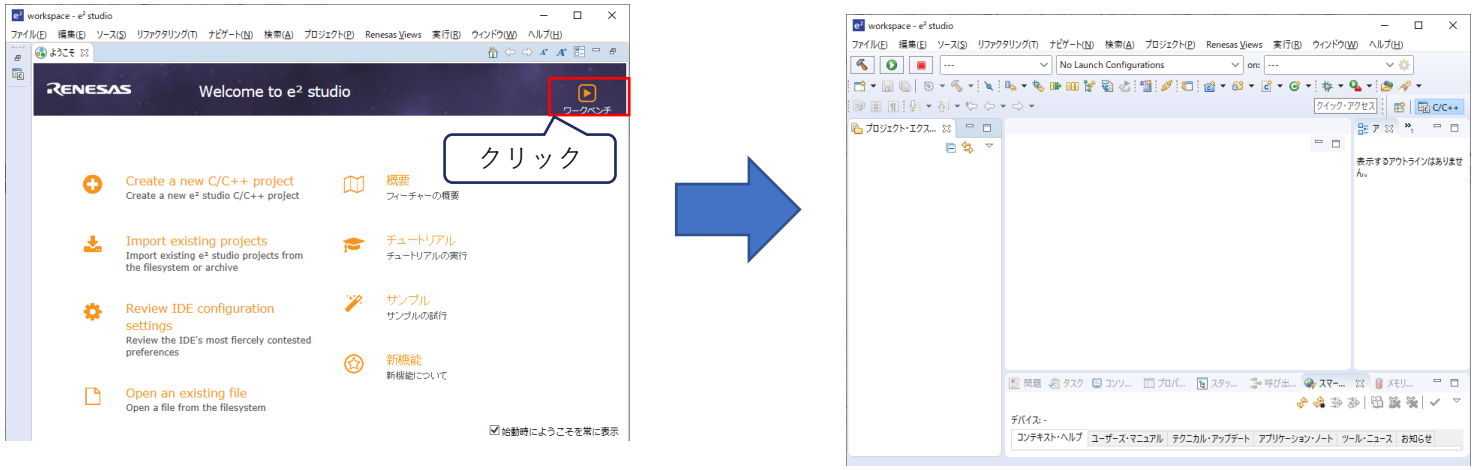
株式会社 カーネル・ソフト・エンジニアリング

e2 studio – 起動

e2 studioのインストールが完了したら、スタートメニューまたはデスクトップよりe2 studioを起動します。
※初回起動時は、下記のような画面が表示されることがありますので、画面の指示に従って、
ワークスペースの作成、ツールチェイン(コンパイラ)の登録を行ってください。



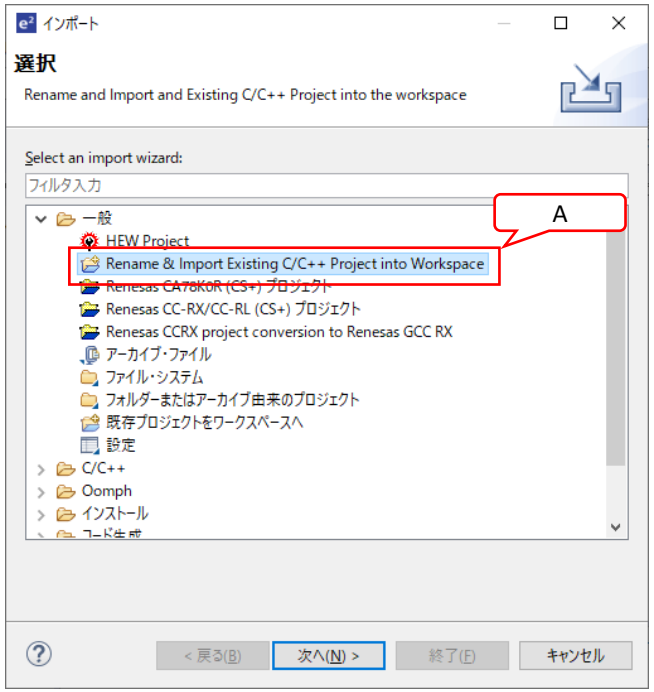
e2 studioを起動したら、右上のワークベンチボタンをクリックして、ワークベンチを開きます。

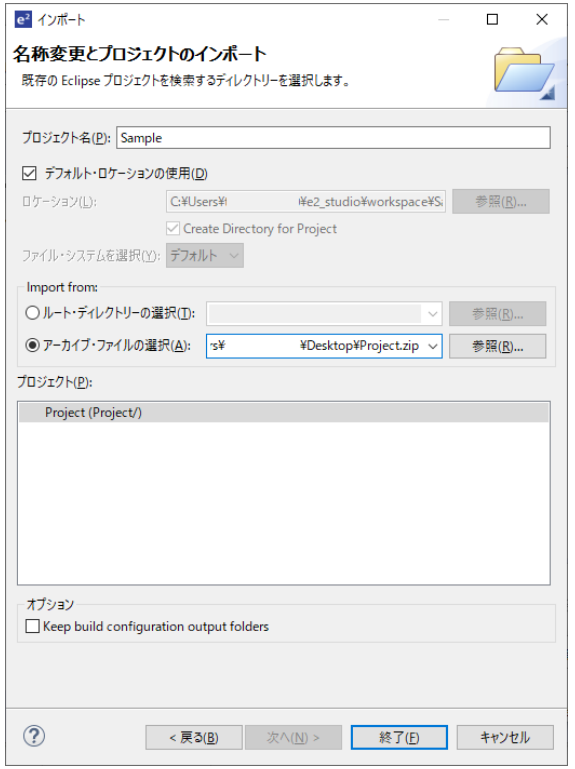


プロジェクトを作成する (1/2)

1. ベースプロジェクト(Project.zip)のダウンロード
   e2 studio導入編手順のiCar用のベースプロジェクトのダウンロードがまだ完了していない場合、
   iCar製品ホームページ(https://icar.kernel-se.co.jp/download/)よりベースプロジェクト(Project.zip)を
   ダウンロードして、デスクトップ等、任意のフォルダに保存しておきます。

2. e2 studioのメニューより、「ファイル」->「インポート」をクリックし、「一般」の中にある、
   「Rename & Import Existing C/C++ Project into Workspace」(画像-A)を選択し、「次へ」ボタンをクリックして、
   「インポート-名称変更とプロジェクトのインポート」画面を開きます。



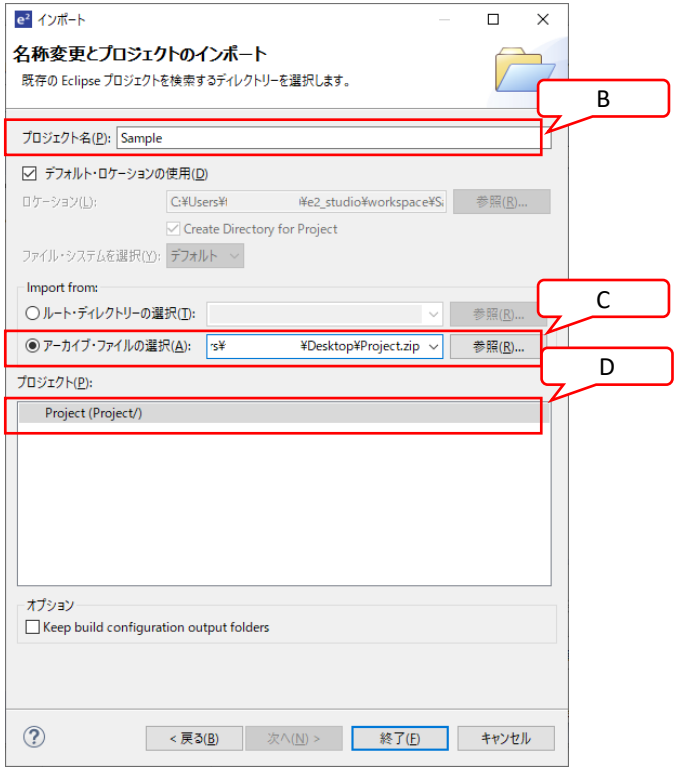


プロジェクトを作成する (2/2)

3. 「インポート-名称変更とプロジェクトのインポート」画面が開いたら、
「プロジェクト名」の項目に、新しく作成するプロジェクト名を入力します。(画像-B)

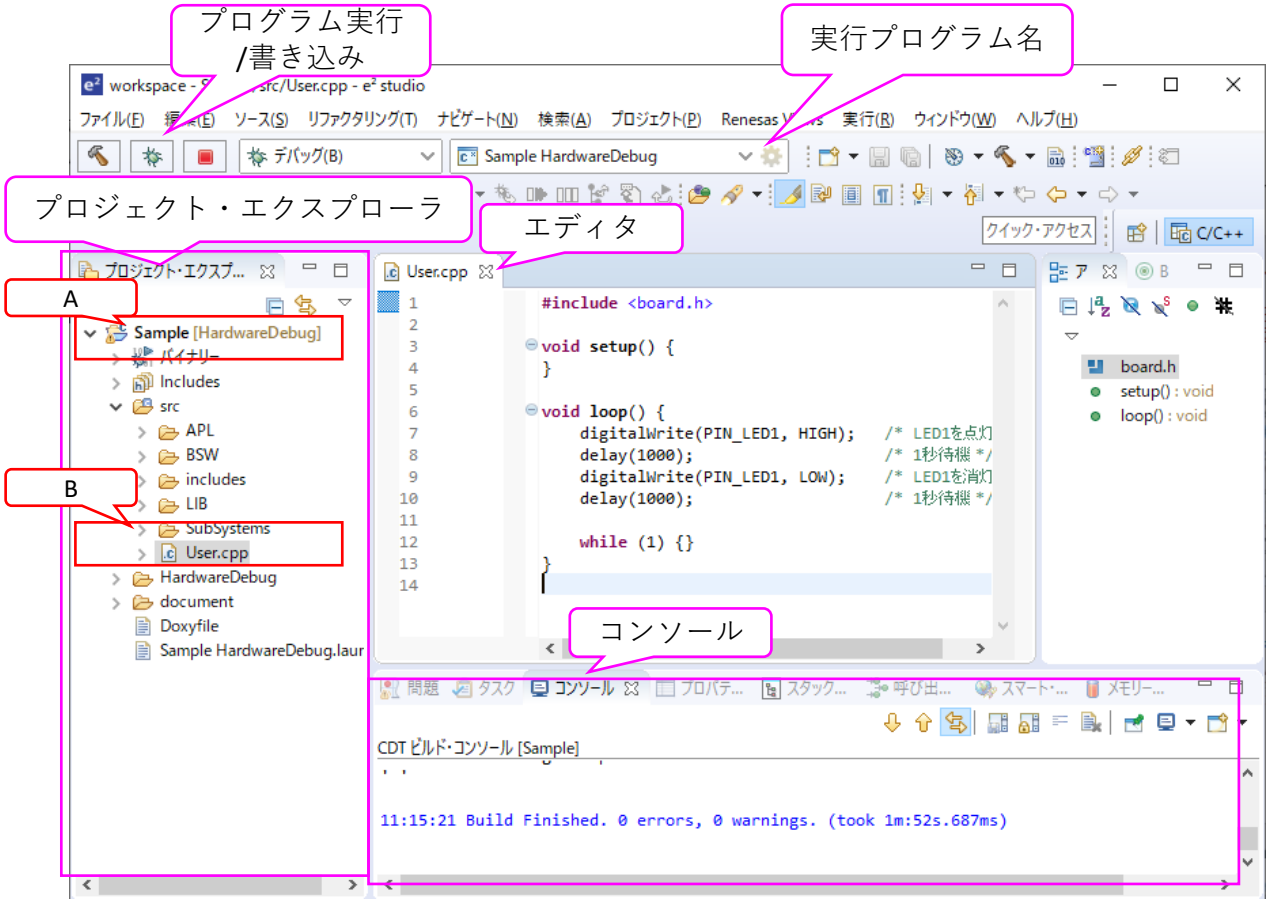
4. Import fromの項目で、「アーカイブ・ファイルの選択」にチェックを入れて「参照」ボタンをクリックし、
ダウンロードしておいた、ベースプロジェクト(Project.zip)を選択します(画像-C)

5. 「プロジェクト」欄に表示されるプロジェクトを選択し(画像-D)、
「終了」ボタンをクリックすると、プロジェクトが作成されます。



プログラムを作成する

プロジェクトを作成した後は、プログラムを作成します。
「プロジェクト・エクスプローラ」より、作成したプロジェクト名をダブルクリックして開き(画像-A)、srcフォルダの中のUser.cppをダブルクリック(画像-B)して開いてUser.cppにプログラムを作成します。
前ページまでのプロジェクト作成手順にて、プロジェクトを作成した直後の状態では、User.cppの内容は、LED1が1秒点灯, 消灯するプログラムとなっています。



【各部の説明】
プロジェクト・エクスプローラ：
開いているプロジェクト,
ファイルが表示されます

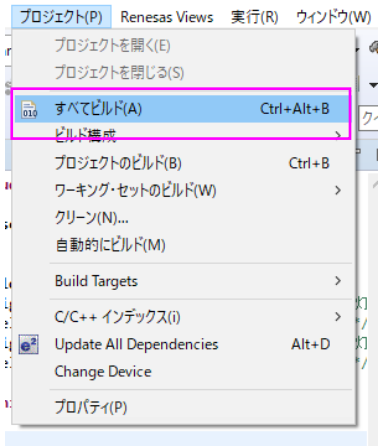
エディタ：
プログラムを編集する
テキストエディタです

プログラム実行/書き込み：
プログラムが書き込みされます

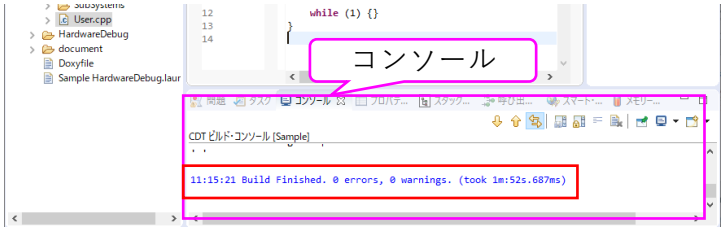
起動設定：
複数プロジェクトを
開いている場合に、
実行/ 書き込みする
プロジェクトを選択します

プログラムをビルドする

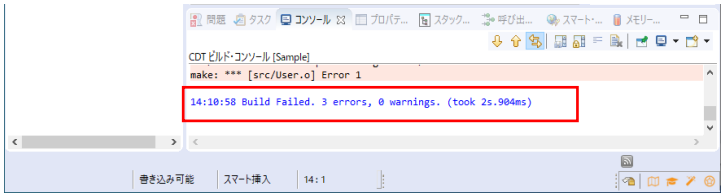
プログラムを作成したら、プログラムをiCarが実行できるようにするために、ビルドを行います。
ビルドを行うには、メニューより、「プロジェクト」->「すべてビルド」をクリックします。
ビルドに成功すると、コンソールに「Build Finished.」と表示されるので、プログラム実行/書き込みに進みます。
失敗した場合は、「Build Failed.」と表示されるので、コンソールタブや問題タブを選択してエラー箇所を確認し、
プログラムを修正してから再度ビルドを行います。

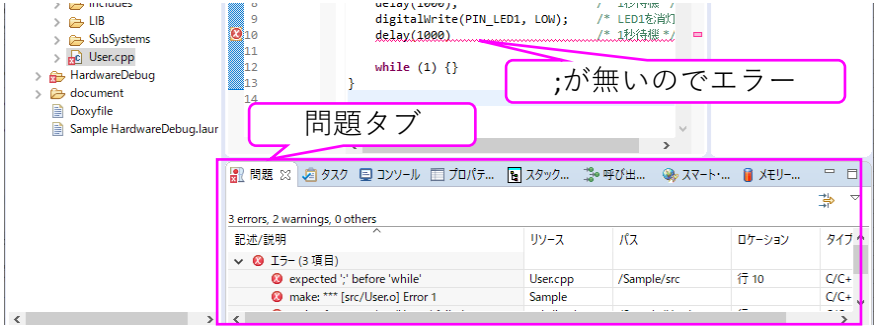


■ビルド成功時



■ビルド失敗時





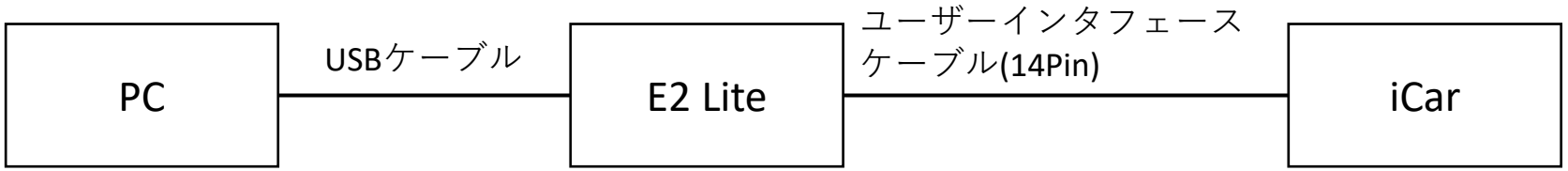
Note よくあるエラーの例:

・行の最後に; セミコロンが無い
expected ‘;’ before ‘****’
・{の数と}の数が一致していない
expected } at end of input
・(の数と)の数が一致していない
expected ‘)’ before ****
・関数名・変数名が間違っている
*** was not declared in this scope
・英数字やカッコが日本語入力になっている

iCarとPCを接続する

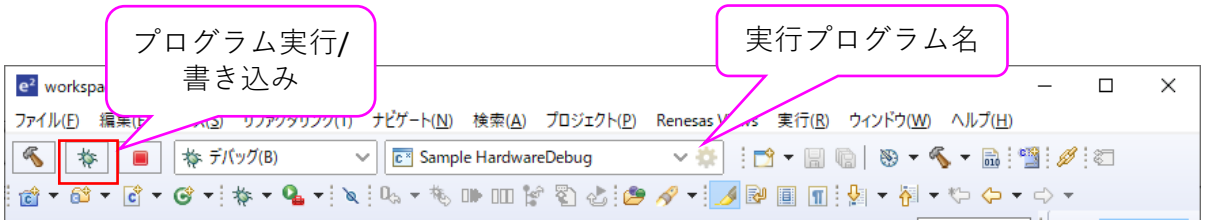
プログラムのビルドが完了したら、プログラムを書き込みするために、iCarとPCを接続します。
iCarの電源をOFFにした状態で、PCとE2エミュレータLite(以降 E2 Lite)、iCarを接続します。

※注意：iCarとE2 Lite、PCとE2 Liteを接続または取り外しをする時は、iCarの電源を切って行ってください。

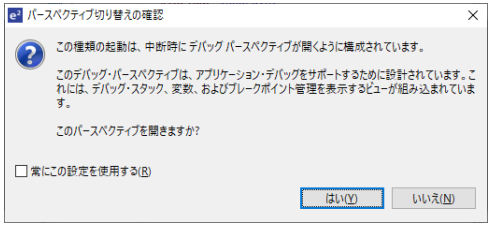


プログラムを書き込みする

iCarとPCを接続したら、iCarの電源を入れて、「プログラム実行/書き込み」ボタンをクリックして
プログラムを書き込みします。
Note: ここで複数実行プログラムがある場合は実行プログラム名を選択して変更します。

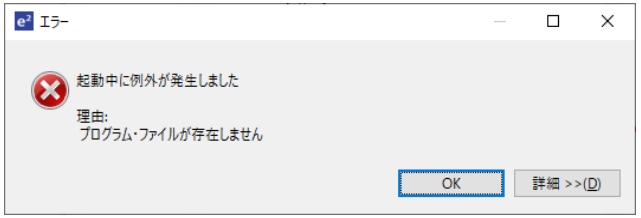


パースペクティブの切り替えの確認ダイアログが出てきたら、「はい」ボタンをクリックします。

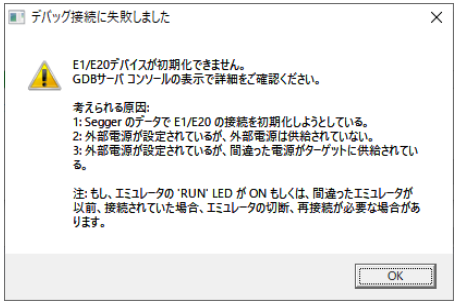


エラー発生時は、エラー内容に従い処置を行ってください。
接続が出来ない場合はiCarとPCの接続を確認、再起動して試行してください。

ビルドエラーが解消されていないのに
実行開始しようとしている・実行プログラム名が間違っている



iCarとPCが接続できない
(iCarの電源が入っていない、iCarの電池が切れている、
PCと接続されていない)

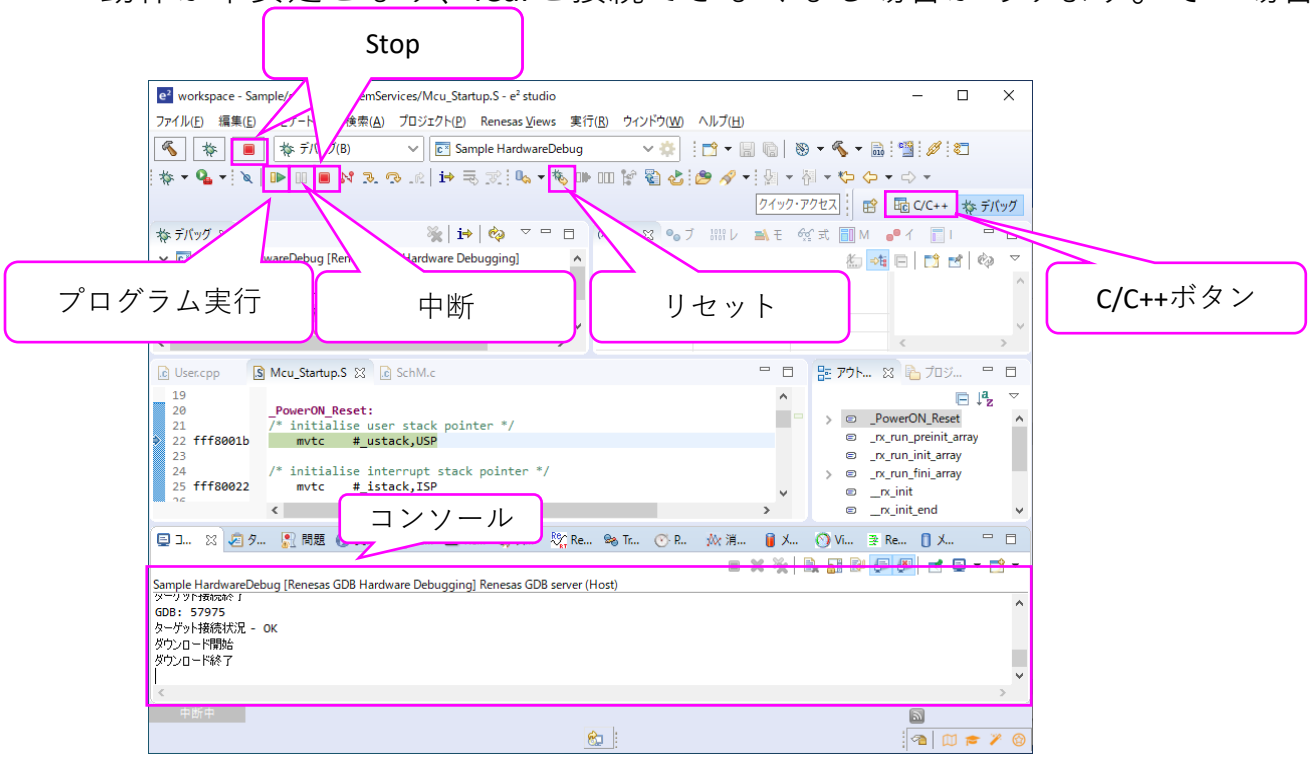


プログラムを実行する

iCarにプログラムの書き込みが完了すると、コンソール画面にダウンロード終了と表示されデバッグ画面となります。
Note: プログラムの書き込みのみ行う場合、ここで「Stop」ボタンをクリックして、デバッグを終了し、
iCarの電源を切ってE2 Liteを取り外します。その後、iCarの電源を再度入れると、iCar単体で動作出来ます。

・「プログラム実行」ボタンをクリックすると、プログラムの実行が開始されます。
・プログラムを終了する時や、 iCarの電源切る時には「Stop」ボタンをクリックします。
・編集画面に戻るには、「C/C++」ボタンをクリックします。
・最初からプログラムを実行するには、「中断」「リセット」「プログラム実行」ボタンを順番にクリックします。

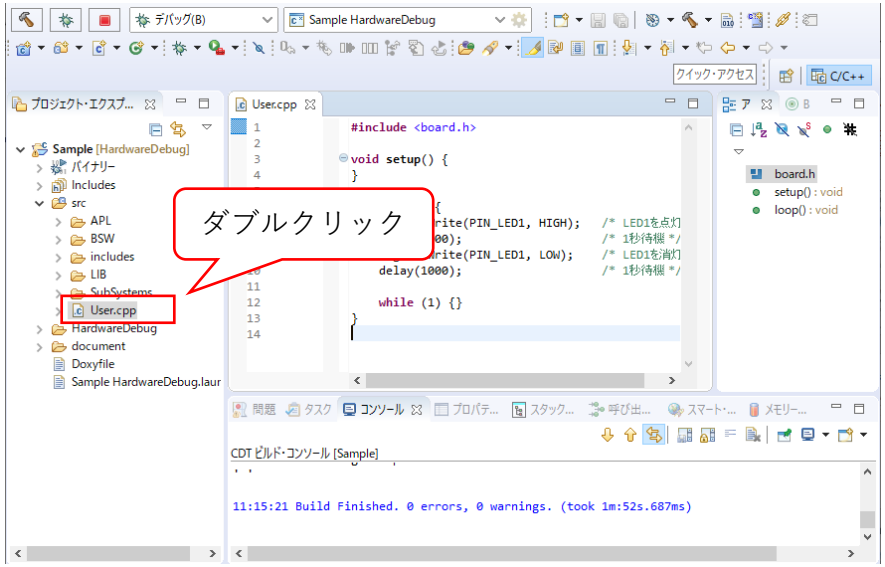
注意：iCarの電源を切る時や、PCから取り外す場合は先にStopボタンをクリックしてプログラムを終了してください。
動作が不安定となり、iCarと接続できなくなる場合があります。その場合PCとiCarを再起動してください。



LED1秒点灯プログラムを作成しよう

e2studioの導入が出来たら、プログラムを作成しましょう。
※このプログラムは、「1_LED1秒点灯」というファイル名でサンプルプログラムに、User.cppに記述する内容が保存されています。

(1) e2 studioを起動してプロジェクト・エクスプローラから User.cppを開きます。



(2) 開いたUser.cppに下記の内容を記述します

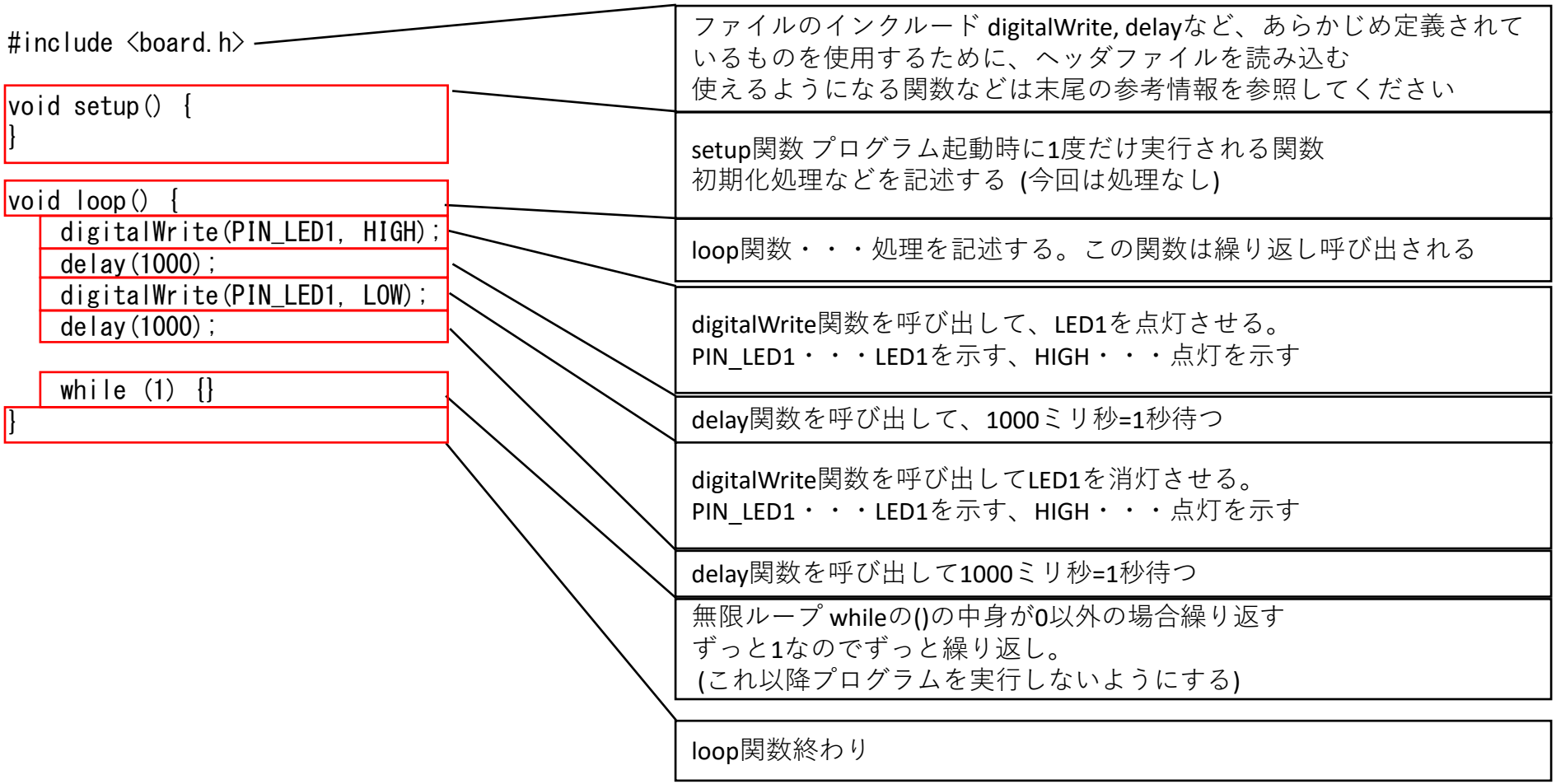
```
#include <board.h>

void setup() {
}

void loop() {
    digitalWrite(PIN_LED1, HIGH);
    delay(1000);
    digitalWrite(PIN_LED1, LOW);
    delay(1000);

    while (1) {}
}
```

プログラムが完成したら、ビルドをして実行してみましょう。
作成したプログラムは、プログラムが起動したらLED1を1秒間ONして、OFFするプログラムなので、
e2Studioのデバッグボタン をクリックしてプログラムを開始して動作を確認してみましょう。
再開ボタン をクリックすると、iCarのLED1が1秒点灯して、消灯する動作となるはずです。

もう一度実行するには、中断ボタン、リセットボタン、再開ボタン を順にクリックします。

LED1秒点灯プログラムを作成しよう(解説1)

前のページで作成したLED1秒点灯プログラムを解説します。



プログラムを変更しよう その1(while(反復)を使う)

前のページで作成した、LED1秒点灯プログラムを、反復処理を使って、
LEDがずっと点滅するプログラムに変更してみましょう。
※このプログラムは、「2_LED点滅」というファイル名で、サンプルプログラムに含まれています。

User.cppを下記のように変更します

```
#include <board.h>

void setup() {
}

void loop() {
    while (1) {
        digitalWrite(PIN_LED1, HIGH);
        delay(1000);
        digitalWrite(PIN_LED1, LOW);
        delay(1000);
    }
}
```

追加

行頭に半角スペース4つ追加
(もしくはTAB追加)

追加

while (1) {}削除

プログラムが完成したら、ビルドして実行してみましょう。 LED1が1秒点灯、1秒消灯を繰り返します。

プログラムを変更しよう その1(while(反復)を使う) (解説)

前のページで作成したLEDがずっと点滅するプログラムを解説します。

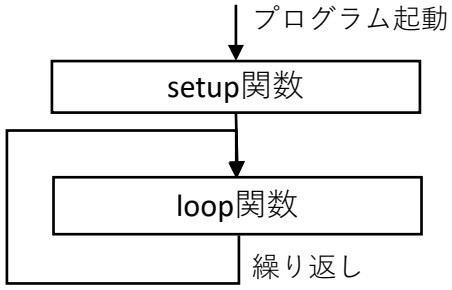
```
#include <board.h>

void setup() {
}

void loop() {
    while (1) {
        digitalWrite(PIN_LED1, HIGH);
        delay(1000);
        digitalWrite(PIN_LED1, LOW);
        delay(1000);
    }
}
```

ファイルのインクルード (iCarで関数を呼び出してプログラミングを行うのに必要なおまじないと考えてください)
setup関数 プログラム起動時に1度だけ実行される関数 初期化処理などを記述する (今回は処理なし)
loop関数・・・処理を記述する。この関数は繰り返し呼び出される
while (1) { ・・・ 無限ループ開始 {から}までの間がループ処理の対象
LED1を点灯→1秒待つ→LED1を消灯→1秒待つの処理 行頭にスペースを入れる変更をしたのは、 インデント(字下げ)して、プログラムを見やすくするためです。 プログラムが見やすくなり、{ }の閉じ忘れを発見するのに有用です。 【ルール】 {が出たら、次の行から行頭に記述するスペースの数を増やします。 }が出たら、その行以降行頭に記述するスペースの数を減らします。 ※iCarではスペース4つでインデントを行っています スペース2つやTABであらわしているプロジェクトもあります。
無限ループ終わり
loop関数終わり

プログラムを変更しよう その1(while(反復)を使う) (別解)

実は前のページまでに作成した、LEDがずっと点滅するプログラムは、whileを使わずに単に作成することも出来ます。
※このプログラムは、「2_LED点滅_別解」というファイル名で、サンプルプログラムに含まれています。

iCarの起動フロー



iCarのプログラムは、プログラム起動時にsetup関数が1度呼び出された後、
loop関数が呼び出しされます。
loop関数の処理が終わっても、すぐに再び呼び出しされるため、
whileによる無限ループ処理を書かなくても、
LEDの点滅処理だけ書くことで、ずっとLED点滅を実現することが出来ます。
逆に、loop関数最後まで実行したら、それ以降プログラムを実行してほしくない場合は
loop関数最後に無限ループを記述します。(使用例： LED1秒点灯プログラム)

※本動作は、上位のモジュールで、while (1) {loop();}のような処理があることで実現しています。

```
#include <board.h>

void setup() {
}

void loop() {
    digitalWrite(PIN_LED1, HIGH);
    delay(1000);
    digitalWrite(PIN_LED1, LOW);
    delay(1000);
}
```

ファイルのインクルード (iCarで関数を呼び出してプログラミングを行うのに必要なおまじないと考えてください)

setup関数 プログラム起動時に1度だけ実行される関数
初期化処理などを記述する (今回は処理なし)

loop関数・・・処理を記述する。この関数は繰り返し呼び出される

LED1を点灯→1秒待つ→LED1を消灯→1秒待つの処理

loop関数終わり

プログラムを変更しよう その2(if(分岐)を使う)

前のページで作成したLEDがずっと点滅するプログラムを、分岐処理を使って、
ボタン1を押している時だけ、LEDが点滅するプログラムに変更してみましょう。
※このプログラムは、「3_ボタンを押している間LED点滅」というファイル名で、サンプルプログラムに含まれています。

```
#include <board.h>

void setup() {
}

void loop() {
    while (1) {
        if (digitalRead(PIN_BTN1) == HIGH) {
            digitalWrite(PIN_LED1, HIGH);
            delay(1000);
            digitalWrite(PIN_LED1, LOW);
            delay(1000);
        }
    }
}
```

追加

行頭に半角スペース4つ追加
(もしくはTAB追加)

追加

プログラムが完成したら、ビルドして実行してみましょう。 ボタン1を押している間だけ、
LED1が1秒点灯、1秒消灯を繰り返します。

プログラムを変更しよう その2(if(分岐)を使う)

前のページで作成したボタン1を押している時だけ、LEDが点滅するプログラムを解説します。

```
#include <board.h>

void setup() {
}

void loop() {
    while (1) {
        if (digitalRead(PIN_BTN1) == HIGH) {
            digitalWrite(PIN_LED1, HIGH);
            delay(1000);
            digitalWrite(PIN_LED1, LOW);
            delay(1000);
        }
    }
}
```

ファイルのインクルード (iCarで関数を呼び出してプログラミングを行うのに必要なおまじないと考えてください)

setup関数 プログラム起動時に1度だけ実行される関数
初期化処理などを記述する (今回は処理なし)

loop関数・・・処理を記述する。この関数は繰り返し呼び出される

while (1) { ・・・ 無限ループ開始 {から}までの間がループ処理の対象

もしボタン1がHIGH(押されていたら)、{}の中の処理を実行する

LED1を点灯→1秒待つ→LED1を消灯→1秒待つ処理

}閉じ (もしボタン1が押されていたら)

無限ループ終わり

loop関数終わり

iCar関数一覧 (1/2)

iCarで使用できる関数の一覧 (使用頻度が高いものを抜粋して説明しています)
使用可能な全関数については、iCar製品ホームページで公開している、関数リファレンスを参照してください。
(関数リファレンスを解凍し、indexファイルを開くと、リファレンスをブラウザで開くことが出来ます)

関数	機能	引数
digitalWrite(PIN_LEDn, HIGH/LOW);	LEDを点灯・消灯させます	1: Pin (操作するLED番号) PIN_LEDn n=LED番号 2: HIGH/LOW・・・ (HIGH=点灯, LOW=消灯)
analogWrite(PIN_BUZZER, freq);	ブザーを鳴らします	1: Pin (PIN_BUZZER固定) 2: freq・・・周波数 (8～16000) ※範囲外は停止
analogWrite(PIN_MOTOR_n, spd);	モータL, Rを指定出力で動かします	1: Pin (動かすモータ) PIN_MOTOR_n n=L=左, n=R=右 2: spd・・・速度(%) (-100=後退～0=停止～100=前進) ※範囲外指定でブレーキをかける
analogWrite(PIN_FLED_n, duty);	フルカラーLEDの明るさを設定します	1: Pin (操作するLED) PIN_FLED_n n=R=赤, n=G=緑, n=B=青 2: duty・・・明るさ(0(消灯)～100(%))
LcdDrv_update();	LCDを更新します ※本関数を呼び出して初めて更新が反映される	なし
LcdDrv_clear();	LCDをクリアします	なし
LcdDrv_print("text");	LCDに文字を表示します	1: 文字列 (char *型)
LcdDrv_setNum(num, digit);	LCDに正の10進数の値を表示します ※表示桁を超えた桁は切り捨てされる	1: num・・・表示値 2: digit・・・表示桁
LcdDrv_setHex(num, digit);	LCDに16進数の値を表示します ※表示桁を超えた桁は切り捨てされる	1: num・・・表示値 2: digit・・・表示桁
LcdDrv_setCursor(y, x);	LCDに書き込む文字の位置を指定します	1: y=縦位置 2: x=横位置
delay(n);	nミリ秒間待機します	1: n・・・待機時間 (ミリ秒)

iCar関数一覧 (2/2)

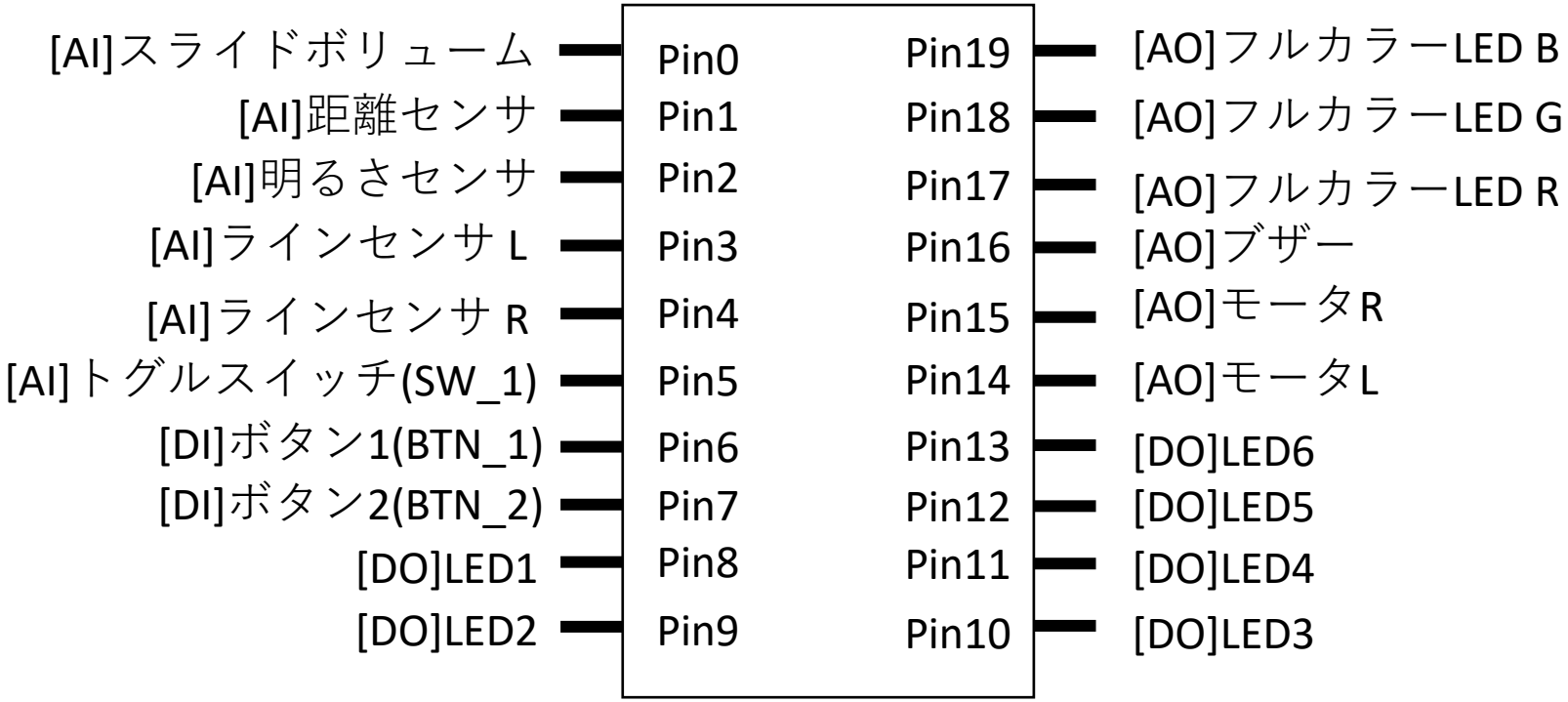
iCarで使用できる関数の一覧 (使用頻度が高いものを抜粋して説明しています)
使用可能な全関数については、iCar製品ホームページで公開している、関数リファレンスを参照してください。
(関数リファレンスを解凍し、indexファイルを開くと、リファレンスをブラウザで開くことが出来ます)

関数	機能	戻り値
analogRead(PIN_DISTANCE)	前方の物との距離を取得する	前方障害物との距離 [mm] (6027・・・未検出)
analogRead(PIN_BRIGHTNESS)	周囲の明るさを取得する	周囲の明るさ (0=明るい～4095=暗い) (※未確定時も4095)
analogRead(PIN_VOLUME)	スライドボリュームの位置を取得	スライドボリュームの位置 (0=後方～100=前方)
analogRead(PIN_LINE_L)	ラインセンサLの明るさを取得	ラインセンサLの直下の明るさ (0=明るい～4095=暗い)
analogRead(PIN_LINE_R)	ラインセンサRの明るさを取得	ラインセンサRの直下の明るさ (0=明るい～4095=暗い)
analogRead(PIN_TOGGLE)	スイッチの状態を取得	スイッチの位置 (0=後退方向, 1=中点, 2=前進方向)
digitalRead(PIN_BTN1)	ボタン1の状態を取得	1=押されている, 0=押されていない
digitalRead(PIN_BTN2)	ボタン2の状態を取得	1=押されている, 0=押されていない
millis()	プログラム起動からのミリ秒を返す	プログラム起動時からの時間 (ミリ秒)
tone(PIN_BUZZER, freq, period)	ブザーを一定時間鳴らします	1: Pin (PIN_BUZZER固定) 2: freq・・・周波数 (8～16000) ※範囲外は停止 3: period・・・鳴らす時間 ミリ秒 (0=ずっと)
noTone(PIN_BUZZER)	ブザーを止めます	1: Pin (PIN_BUZZER固定)

iCar Pin一覧

iCarでは前のページの関数一覧に記載されているように、
各デバイスを種類に応じた制御関数(digitalWrite, digitalRead, analogWrite, analogRead)呼び出すことで制御します。
関数呼び出し時に指定するPinについて下記のようになっています。

[AI]・・・analogRead関数で操作、 [DI]・・・digitalRead関数で操作
[AO]・・・analogWrite関数で操作、[DO]・・・digitalWrite関数で操作



Note: 実際のマイコンのPinと変換する処理をsrc/SubSystems/arduino/iCar/hardware/iCar/trunk/variants/iCar/port.cで
行っているため、マイコンのPin番号とは異なります。

(参考) 時間待ちと他の処理を並行動作させる方法(1)

1秒待つ処理などの一定時間待つ処理について、サンプルプログラムの多くではdelay関数を使って実現しています。
しかし、delay関数を使用すると、待ち時間の間は他の処理を何も実行出来ない問題があります。

例えば、時間待ちがある処理と他の処理を同時に行う以下の制御は、delay関数を使って実現するのが困難です。
・LEDを1秒間隔で点滅させる
・ボタン1を押している間、ブザーを鳴らす(ボタン入力にすぐに(1秒待たず)ブザーを鳴らす)

このような制御を実現するためには、下記のようにmillis関数を使用してプログラム起動時からの時間を取得し、定期的に処理を行います。
※本プログラムは、「8_ボタンを押したらブザーを鳴らす(2)」というファイル名で、サンプルプログラムに含まれています。

```
#include <board.h>

int LedOn;
uint32_t LedTime;

void setup() {
    LedTime = millis();
    LedOn = LOW;
}

void loop() {
    /* BTN1 ON */
    if (digitalRead(PIN_BTN1) == HIGH) {
        analogWrite(PIN_BUZZER, 2000);
    } else {
        analogWrite(PIN_BUZZER, 0);
    }

    /* LEDを最後に制御してから1秒たった */
    if ((millis() - LedTime) >= 1000) {
        /* ONとOFFを反転する */
        LedOn = !LedOn;

        /* LEDを更新する */
        digitalWrite(PIN_LED3, LedOn);

        /* 最後にLEDを制御した時間を覚える */
        LedTime = millis();
    }
}
```

uint32_t型で、最後にLED点灯/消灯した時間を記憶する変数を作成

setup関数内にて、
現在時刻で、最後にLED点灯/消灯した時間を記憶する変数を初期化

常に行いたい処理を記述
(ボタン1を押したらブザーを鳴らす処理)

現在時刻 - 最後にLED点灯/消灯した時間を求めて、
実行したい間隔(1秒)以上経過しているかを判定
※必ず、以上(>=)で判定します。 (一致(==)で判定すると、他の制御の処理時間の影響を受けて、現在時刻 - 最後にLED点灯/消灯した時間が1000を超えてしまった場合、二度と処理が実行されなくなってしまいます)

指定時間間隔で行う処理を記述する

現在時刻で、最後にLED点灯/消灯した時間を記憶する変数を更新

Note: millis()関数はプログラム起動からのミリ秒を0～ 4294967295の範囲で返します(4294967295の次は0) →約49日まで計測可能

(参考) 時間待ちと他の処理を並行動作させる方法(2)

前のページで、millis関数で時間を測定する処理をして、以下のプログラムを作る例を紹介しましたが、
他には、短い時間(10ミリ秒など)で、定期的に実行する関数を作り、カウンタを用いて1秒を計測する方法もあるので紹介します。
・LEDを1秒間隔で点滅させる
・ボタン1を押している間、ブザーを鳴らす(ボタン入力にすぐに(1秒待たず)ブザーを鳴らす)

※本プログラムは、「9_ボタンを押したらブザーを鳴らす(3)」というファイル名で、サンプルプログラムに含まれています。

```
#include <board.h>

void period_10ms();

int LedOn;
int LedTime;
uint32_t PeriodTime;

void setup() {
    PeriodTime = millis();
    LedTime = 0;
    LedOn = LOW;
}
void loop() {
    if (digitalRead(PIN_BTN1) == HIGH) {
        analogWrite(PIN_BUZZER, 2000);
    } else {
        analogWrite(PIN_BUZZER, 0);
    }
    if ((millis() - PeriodTime) >= 10) {
        period_10ms();
        PeriodTime = millis();
    }
}
void period_10ms() {
    LedTime++;
    if (LedTime >= 100) {
        LedTime = 0;
        LedOn = !LedOn;
        digitalWrite(PIN_LED3, LedOn);
    }
}
```

関数のプロトタイプ宣言を記述

時間を数える変数を作成 (ここではint型でLedTime)
uint32_t型で、10ミリ秒定期処理を実行した時間を記憶する変数を作成

setup関数内にて、
10ミリ秒定期の時間変数をmillis()で初期化、
時間を数える変数を0初期化

常に行いたい処理を記述
(ボタン1を押したらブザーを鳴らす処理)

現在時刻 - 10ミリ秒定期処理を実行した時間を求めて、
実行したい間隔(10ミリ秒)以上経過しているかを判定
10ミリ秒毎にperiod_10ms関数を呼び出しする

10ミリ秒毎に実行される関数を記述する

10ミリ秒 × 100回 = 1秒をLedTime変数で数えて、1秒ごとに処理を実行

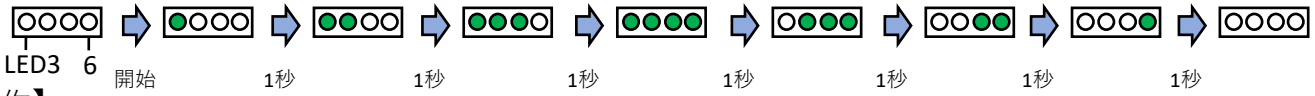
1秒ごとに実行する処理を記述 (LED点滅)

(参考) プログラムの実行を一時停止・1行ずつ実行する(1)

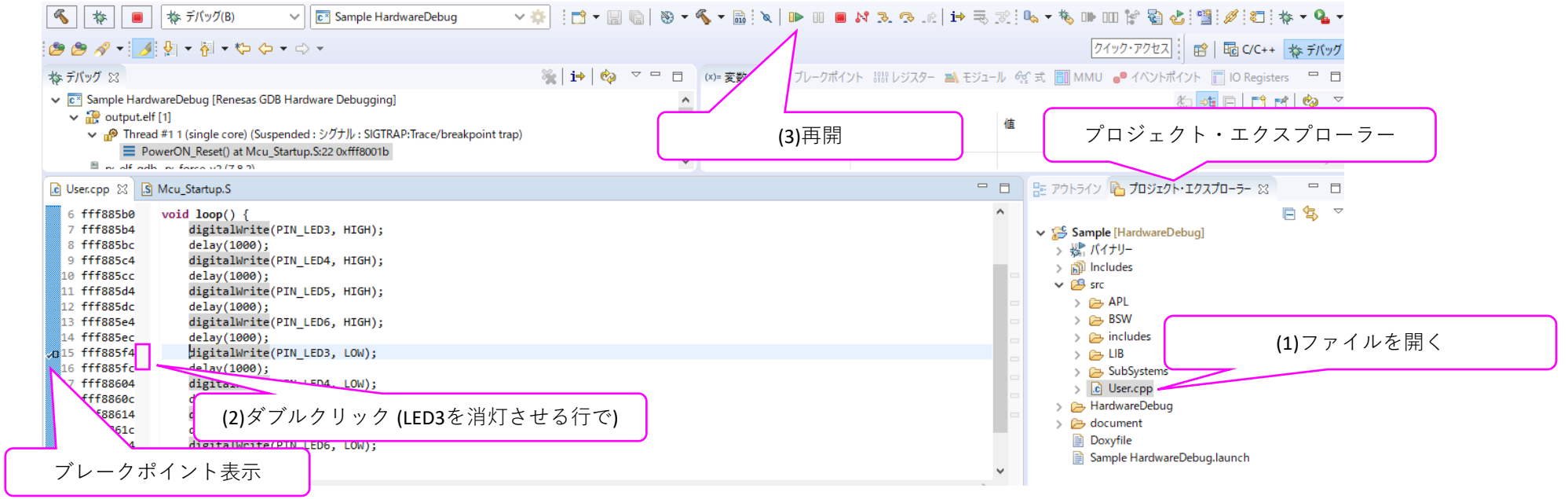
デバッガ(e2 studio)には、「ブレークポイント」という、設定した行でプログラムを停止する機能や、
「ステップ実行」という1行ずつプログラムを実行する機能があります。
ここでは、ブレークポイントを使用して、プログラムを一時停止し、動作を確認してみましょう。

【準備】サンプルプログラムの「5_順番にLEDを点灯(2)」を書き込みします

【サンプルプログラムのプログラムの動作】LED3～6を順番に点灯した後、LED3～6の順に消灯

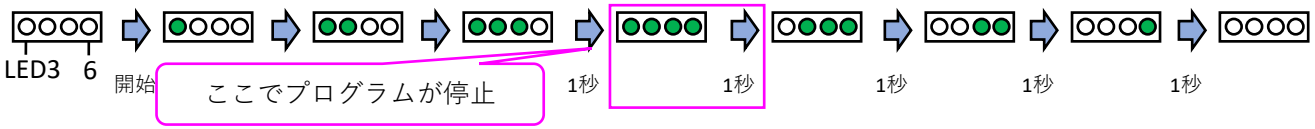


【操作】
(1) プログラムを書き込み後、デバッグ画面のプロジェクト・エクスプローラーより、実行を止めたい行があるファイルを開きます
(2) 実行を停止したい行(今回はLED3を消灯する処理で設定)の左側をダブルクリックしてブレークポイントを設定します。
　設定するとブレークポイント表示が出ます
(3) 再開 (プログラム実行ボタン)をクリックすると、指定行で実行が止まります (次ページ)

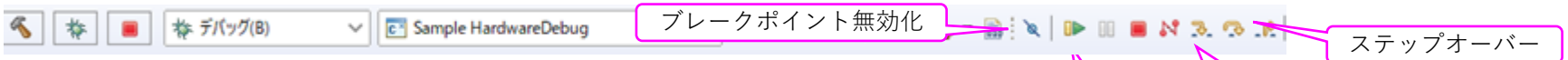


(参考) プログラムの実行を一時停止・1行ずつ実行する(2)

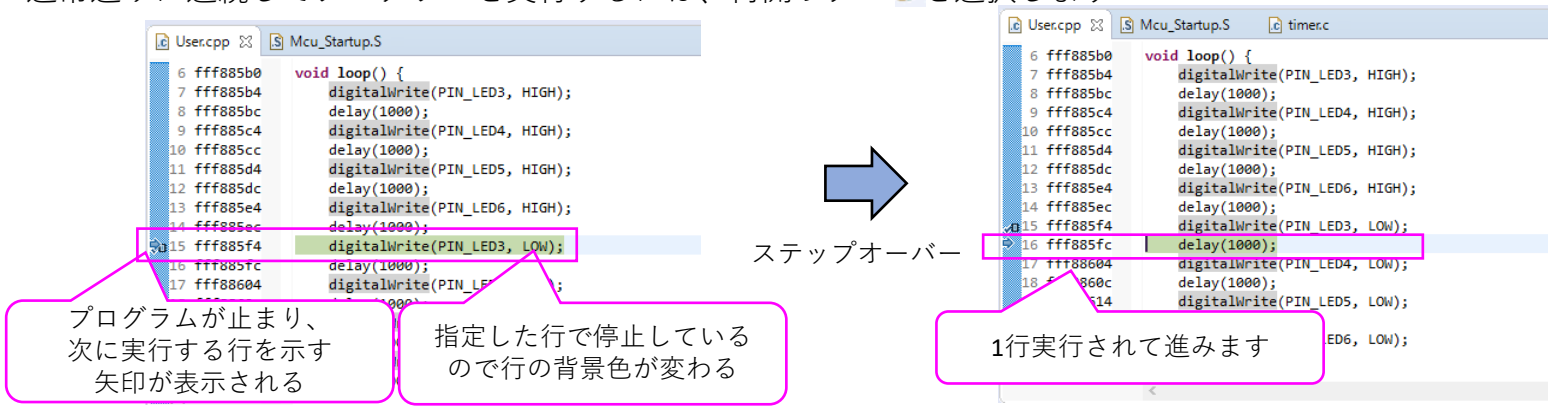
前ページで、ブレークポイントの設定とプログラムの実行を開始しました。
プログラムがブレークポイントを設定したところまで実行されると、プログラムの動作が停止します。



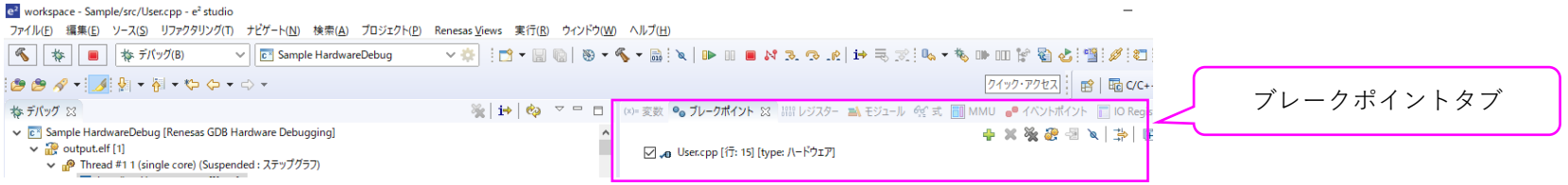
プログラムの動作が停止したら、そのあとの実行方法をメニューバーより選択することが出来ます。



1行だけプログラムを実行するには、「ステップオーバー」ボタン を選択します
関数の中身を細かく1行ずつ見る場合は、「ステップイン」ボタン を選択します
通常通りに連続してプログラムを実行するには、再開ボタン を選択します



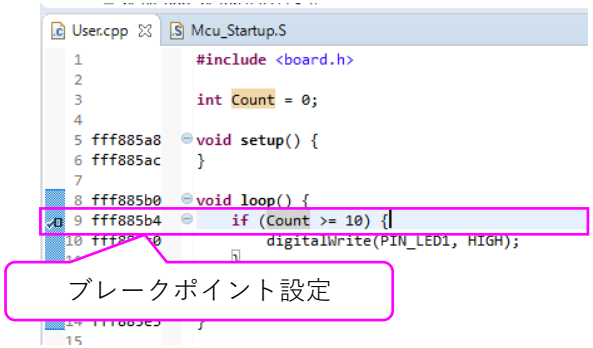
Note: ブレークポイントの設定可能数は8個です。(8個を超えるとエラーとなります)
また、ブレークポイント無効化ボタンを選択すると一括で無効化できます。
ブレークポイントタブで、設定しているブレークポイント一覧の表示や削除ができます。

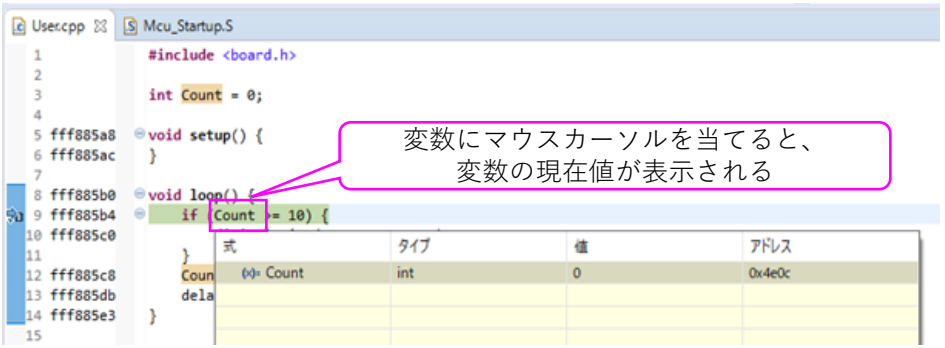


(参考) 変数の値を確認する

デバッガ(e2 studio)には、変数の値を確認する機能があります。
ここでは、ブレークポイントと組み合わせて変数の値を確認してみましょう。

【準備】サンプルプログラムの「21_10秒後にLED点灯」を書き込みします
【サンプルプログラムの動作】プログラム起動後10秒後にLED1が点灯します
【操作】
・User.cppのif (Count >= 10)の行にブレークポイントを設定し、再開ボタンを押してプログラムを実行します
・ブレークポイントでプログラムが停止したら、見たい変数 (ここではCount)の上にマウスカーソルを当てると変数の値が表示されます





・式タブに変数名を入力することでも値を確認することが出来ます。また、変数であれば値の変更も可能です。

Note: ローカル変数(関数内で宣言)の場合は、スタックに配置される都合上、正しく値が表示されないことがあります。
どうしても値を確認したい場合はグローバル変数で宣言することで値を確認することができます。