訓練手引書

生産する製造業者から：ArduinoからAVRへの間隙の橋渡し

AN-12077

事前必要条件

- **ハードウェア事前必要条件**
  - ATmega328P Xplained Mini基板
  - IO1 Xplained Pro拡張基板
  - Arduino Xplained Pro基板
  - マイクロUSBケーブル
- **ソフトウェア事前必要条件**
  - Atmel® Studio 6.2またはそれ以降版
  - Arduino IDE 1.6.0
  - Atmel Studio用Arduino拡張
  - 端末ウィンドウ拡張
- **予想完了時間**
  - 2時間

序説

この実践は豊富な使用者インターフェースとそれが提供する他の優れた開発ルールと共にAtmel Studioを使ってArduinoを開発する方法を実演します。

Arduinoは柔軟性、使い易いハードウェアとソフトウェアに基づく解放ソースの電子回路試作基盤です。それは芸術家、設計者、趣味人、そして相互作用的ば物や環境を作成することに興味がある誰かのために意図されています。Arduinoについての重要な事実はこれらの基板がAtmelマイクロ コントローラ系統に基づき、基本的なソフトウェアはAtmel開発ツールに基づきます。

‘何故Aruduinoから切り替えるべきか?’に対する答えは、Arduino IDEが以下だからです。

- 経験豊富な書き手の能力に対するかなりの制限
- コンパイラ警告とデバッグ能力の不足
- (全ての命令文の後のSerial.println()は数ず)、高度なプロジェクトでの作業時に重荷
- Atmel Studioは統合されたArduino IEDを脱した使用者に対する素晴らしい選択
- それらのArduinoライブラリのいくつかは正に殆ど価値のない簡便性

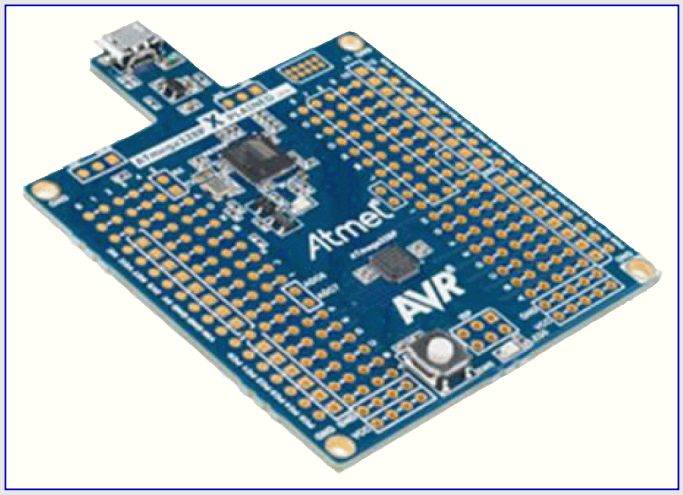
従って何故両世界を出来るだけ利用しないのか?。Arduinoはデバッグ能力を持つC/C++の頂上の包装部(ラッパー)。どんなArduinoのスケッチやライブラリでも高度なプロジェクトであなた自身の独自コードと結合することが可能です。

Atmelは趣味人から開発者への移行の橋渡し、助けを提供するための“going pro”共同体についてのArduinoでの独特で特権的位置と責任を持ちます。

Atmel Xplained Mini系統はCとC++への容易な移行に対する完璧な”橋渡し”です。それはArduinoに対して非常に類似した構造を持ち、Arduino基板のも飛渡の機能を提供します。IDEが正しく構成設定された時にArduinoスケッチの走行が可能です。基板上のハードウェア デバッガ/書き込み器も利用可能です。それは非常に一般的なAVR®マイクロ コントローラ系統使い、相対的に安価です。

Rev. 42439A-08/2015, 42439AJ1-01/2021

図1. ATmega328P Xplained Mini基板



この訓練手引き書は4つの課題から成ります。

最初の課題ではATmega328P Xplained MiniをArduino IDEに接続L方法を理解します。

第2の課題ではAtmel StudioのArduino拡張を使ってAtmel StudioでArduinoｽｹｯﾁをを作成します。

第3の課題では直接ｽｹｯﾁを取り入れるようにAtmel Studioを構成設定します。既存Arduinoｽｹｯﾁをを Atmel Studioへ移行する方法を検討します。また、Atmel Studioで中断点(ﾌﾞﾚｰｸﾎﾟｲﾝﾄ)を挿入する方法とﾃﾞﾊﾞｯｸﾞする方法を調べます。

第4の課題ではﾌｧｲﾙを編集して周辺機能(ADC,I^{2}C,SPI)を持つ簡単な応用を作成します。この応用はA/D変換器(ADC)を通し光感知器を読み、I^{2}Cｲﾝﾀｰﾌｪｰｽを通して温度感知器を読み、そしてSPIｲﾝﾀｰﾌｪｰｽを通してSDｶｰﾄﾞ内にﾃﾞｰﾀを格納します。

目次

アイコン基本識別子

 **情報** 特定の話題について文脈上の情報を提供

 **助言** 有用な助言と技法を強調

 **実施事項** 完了されるべき目標を強調

 **結果** 課題段階の予測される結果を強調

 **警告** 重要な情報を表示

 **実行** 必要時に目的対象の実行されるべき活動を強調

1. 訓練単位部構造

この訓練素材は以下として各種Atmel配給物を通して取得することができます。

- 通常、Atmel展示室ｳｪﾌﾞ ｻｲﾄ(http://gallery.atmel.com/)で、またはAtmel Studio拡張管理部を使って得られるAtmel Studio拡張(.vsixﾌｧｲﾙ)
- 通常、Atmel訓練作業中に提供されるAtmel訓練実行物(.exeﾌｧｲﾙ)

配布形式に依存して、この訓練素材によって必要とされる各種資料(実践訓練資料、ﾃﾞｰﾀｼｰﾄ、応用記述、ﾂｰﾙ)は種々の場所で得られます。

1.1. Atmel Studio拡張 (.vsix)

一旦この拡張がｲﾝｽﾄｰﾙされると、Atmel Studioで"New Example Project from ASF..."を使って種々のﾌﾟﾛｼﾞｪｸﾄを開いて作成することができます。

この拡張でｲﾝｽﾄｰﾙされたﾌﾟﾛｼﾞｪｸﾄは一般的に"Atmlel Training⇒Ateml Cort. Extension Name"下で得られます。

拡張に依存して利用可能な各種ﾌﾟﾛｼﾞｪｸﾄがあります。

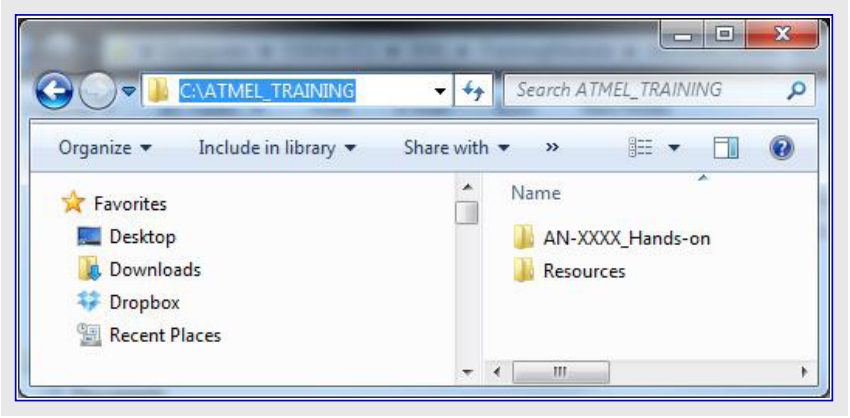
- Hands-on Documentation (実践訓練資料) ：必要とされる資源としての資料を含みます。
- Hands-on Assignment (実践訓練課題) ：開始に必要とされるかもしれない初期ﾌﾟﾛｼﾞｪｸﾄを含みます。
- Hands-on Solution (実践訓練解決策) ：この実践訓練に対する解決策である最終的な応用を含みます。

以降の頁でいくつかの資源に対して参照が行われる度に、使用者はHands-on Documentation(実践訓練資料)ﾌﾟﾛｼﾞｪｸﾄ ﾌｫﾙﾀﾞを参照しなければなりません。

1.2. Atmel訓練実行物 (.exe)

実行物がｲﾝｽﾄｰﾙされた場所に依存して、2つの主なﾌｫﾙﾀﾞから成る以下の基本構造を見つけるでしょう。

- AN-12077_Hands-on ：開始と解決策に必要とされるかもしれない初期ﾌﾟﾛｼﾞｪｸﾄを含みます。
- Resources ：必要とされる資源(ﾃﾞｰﾀｼｰﾄ、ｿﾌﾄｳｪｱ、ﾂｰﾙなど)を含みます。



特定の位置が指定されない限り、以降の頁でいくつかの資源に対して参照が行われる度に、使用者はこのResources(資源)ﾌﾟﾛｼﾞｪｸﾄ ﾌｫﾙﾀﾞを参照しなければなりません。

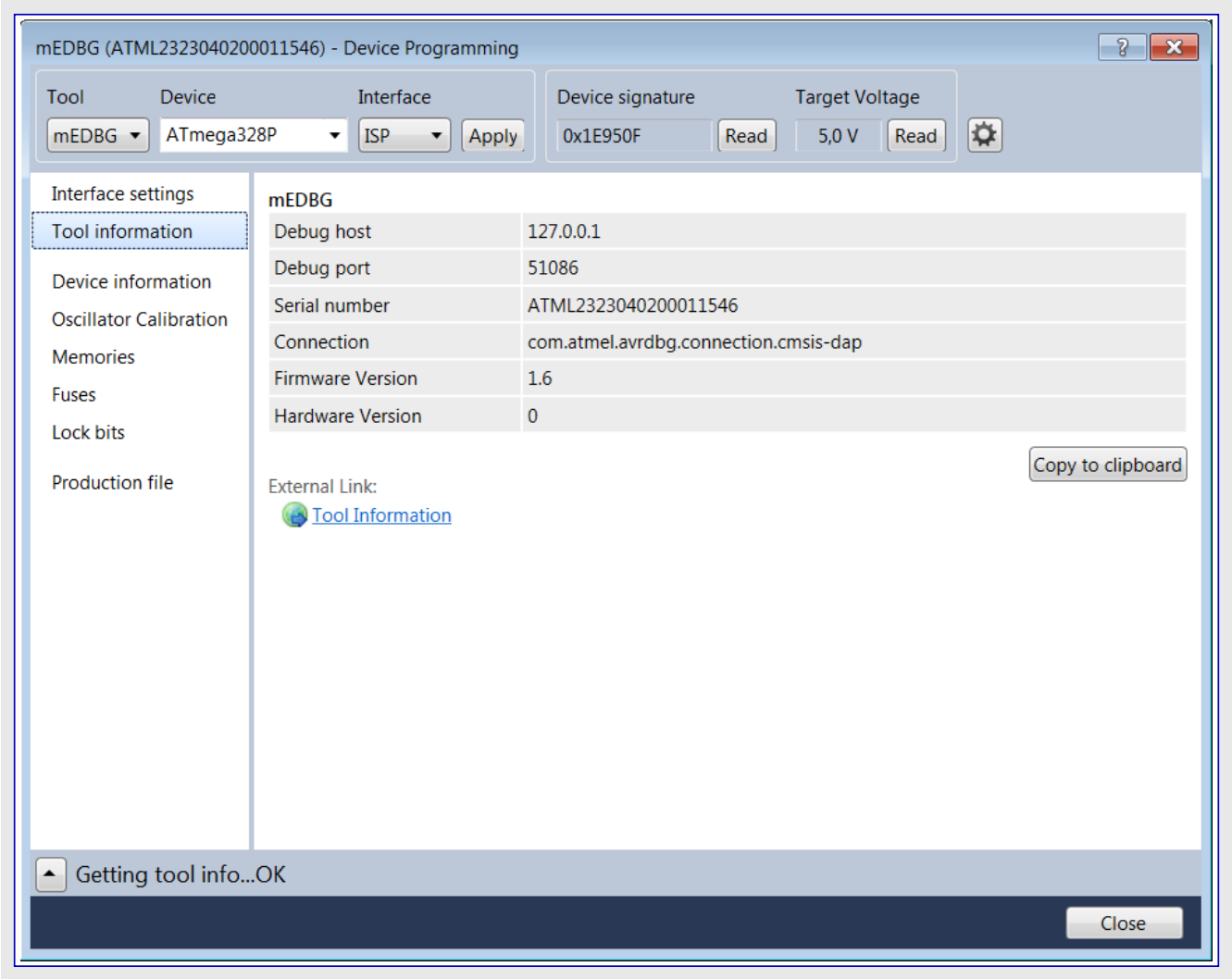
2. 課題1 : Arduino IDEへのATmega328P Xplained Mini接続方法

2.1. Arduino IDE

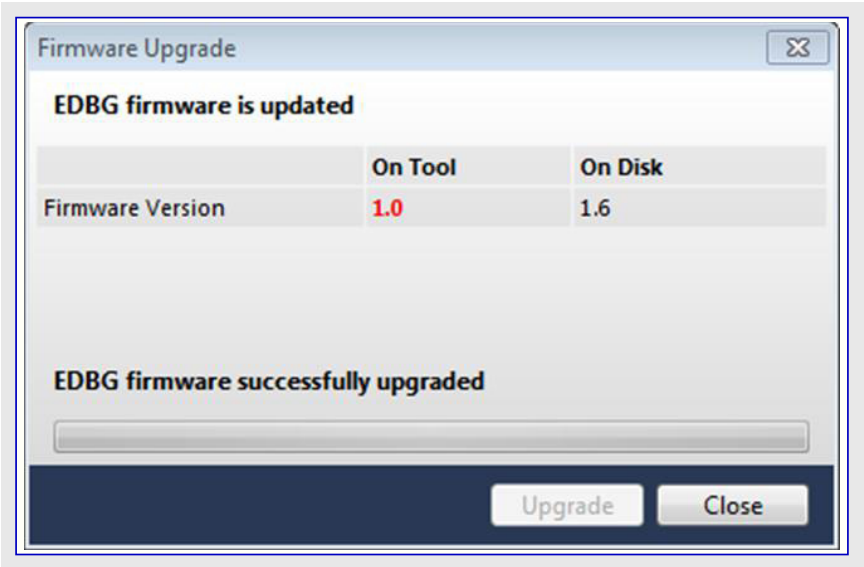
www.arduino.ccからArduino IDEをﾀﾞｳﾝﾛｰﾄﾞしてください。

2.2. ATmega328P Xplained MiniでのmEDBGﾌｧｰﾑｳｪｱ格上げ(更新)

1. Atmel作業空間(http://spaces.atmel.com/gf/project/avr_xp_mini/frs/)へ行ってください。"medbgdebugger"一括からmedbg_fw.zipを選んでﾀﾞｳﾝﾛｰﾄﾞしてください。
2. Atmel Studioｲﾝｽﾄｰﾙ ﾌﾙﾀﾞ(例えば、C:¥Program Files (x86)¥Atmel¥Atmel Studio 6.2¥tools¥mEDBG)にmedbg_fw.zipのZIP一括を上書きしてください。
3. Atmel Studioを開始してください。
4. ATmega328P Xplained Miniをｺﾝﾋﾟｭｰﾀに接続してください。
5. Atmel Studioに於いて、Tools⇒Device programming (alt.:Ctrl+Shift+P)を選んでください。
6. ﾃﾞﾊﾞｲｽ ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ｳｨﾝﾄﾞｳで、Toolにmedbgを設定して"Apply"をｸﾘｯｸしてください。



情報 次にAtumel Studioがﾌｧｰﾑｳｪｱを格上げ(更新)を望むかを尋ねるでしょう。



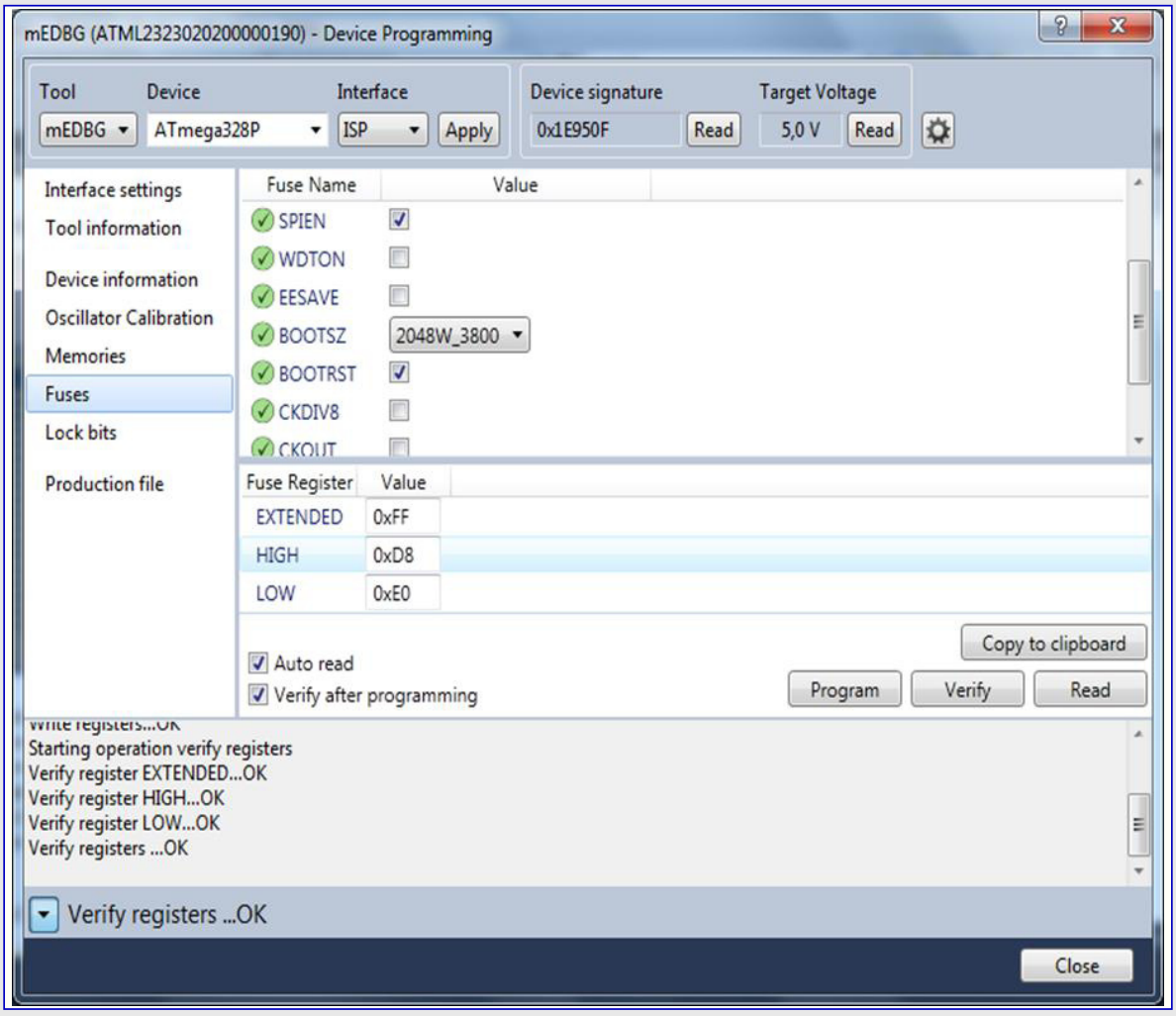
7. Upgradeを選んでください。

情報 Atmel Studioのﾌﾟﾛｸﾞﾗﾐﾝｸﾞ/ﾃﾞﾊﾞｯｸﾞによる認証をできなくする、いくつかのATmega328P Xplained Mini基板での通番のﾊﾞｸﾞがあります。あなたの基板が通番内に未知の文字を持つ場合、"Releases"ﾌｫﾙﾀﾞ内に修正があります。あなたの基板の通番を見るには、Atmel Studioを開始してTools⇒Device programmingでツールを選択してください。これは通番と共にmEDBGを一覧にし、いくつかの文字が黒背景で"?"を持つ場合、通番修正(serial number fix)から一括をﾀﾞｳﾝﾛｰﾄﾞしてHow_to_change_serial_Number.pdfからの指示に従ってください。

2.3. ATmega328Pでのﾌﾞｰﾄﾛｰﾀﾞ ﾋｭｰｽﾞ設定

1. 次に’Device Programming’ｳｨﾝﾄﾞｳで、’Fuses’を選択してください。
2. 下で示されるようにEXTENDED,HIGH,LOWの値を変更してProgramをｸﾘｯｸしてください。
   EXRENDED = $FF
   HIGH =$D8
   LOW = $E0

図2-1. Device Programming : Fuses



2.4. ﾌﾞｰﾄﾛｰﾀﾞ書き込み

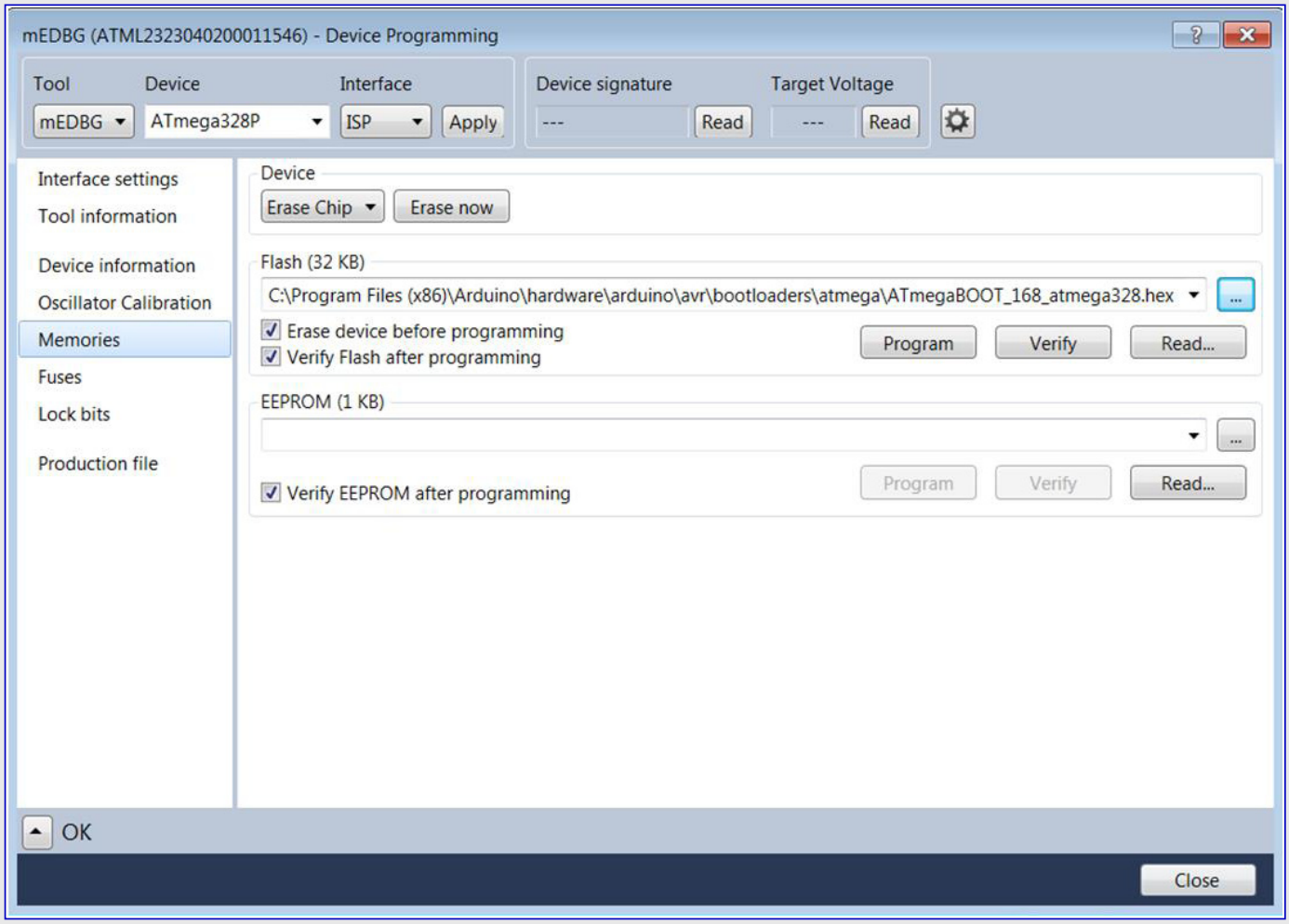
情報 ﾌﾞｰﾄﾛｰﾀﾞhexﾌｧｲﾙはC:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥bootloaders¥atmega¥*.hexのArduino IDEﾌｫﾙﾀﾞに置かれます。ﾌﾞｰﾄﾛｰﾀﾞは**表2-1.**で一覧されるように基板構成設定に従って選択することができます。

表2-1. ﾌﾞｰﾄﾛｰﾀﾞ

Xplained Mini	ﾌﾞｰﾄﾛｰﾀﾞ
ATmega328P/5V/16MHz	ATmegaBOOT_168_atmega328.hex
ATmega168P/5V/16MHz	ATmegaBOOT_168_ng.hex
ATmega168P/3.3V/8MHz	ATmegaBOOT_168_pro_8MHz.hex

1. ‘Tools⇒Device Programming’でのｳｨﾝﾄﾞｳで、’Memories’ﾀﾌﾞを選んでください。
2. C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥bootloaders¥atmega¥ ATmegaBOOT_168_atmega328.hexを閲覧してください。
3. Programをｸﾘｯｸしてください

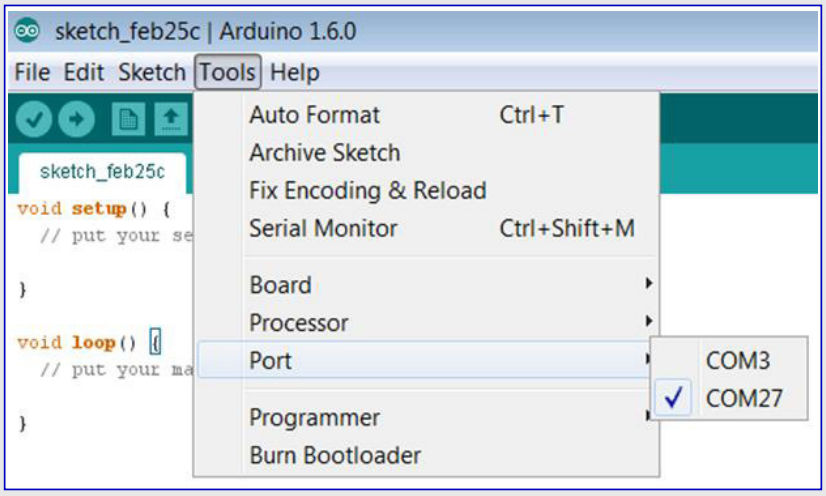
図2-2. Device Programming : Memories



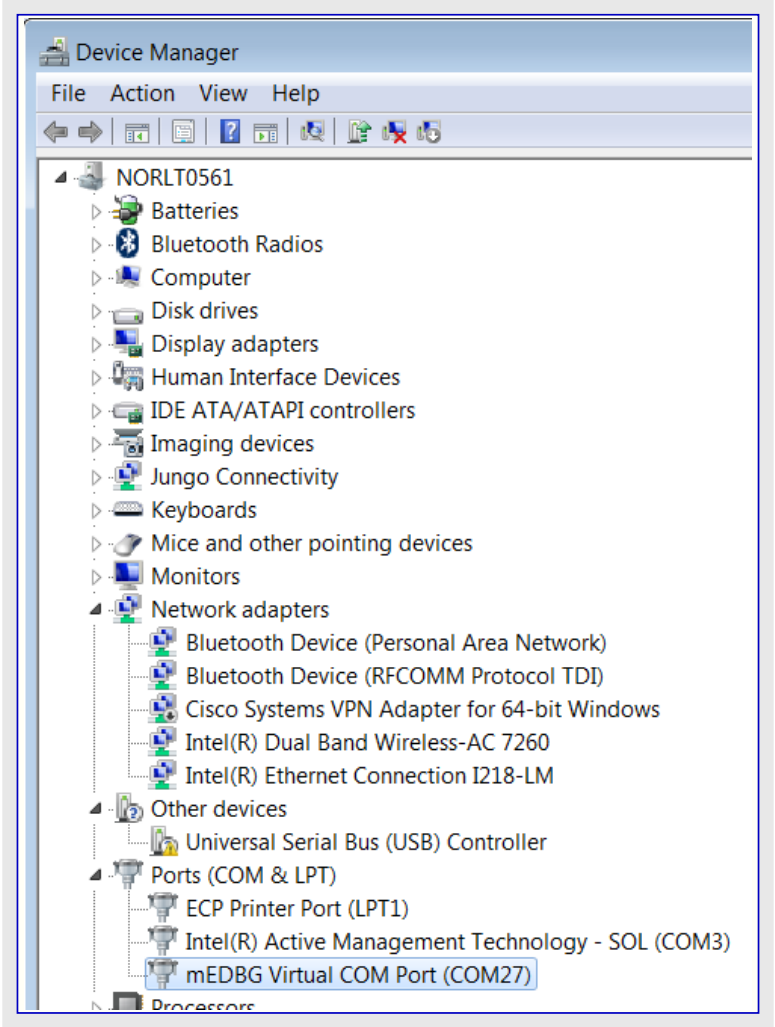
2.5. Arduino IDE構成設定

1. Windows®のスタート メニューまたはArduino IDEがインストールされたフォルダからArduino IDEを開始してください。
2. メニューからのTools⇒Portからで mEDBG用の正しいCOMポートを選択してください(mEDBGに使われるCOMポートを確認する方法を次の助言でご覧ください)。

図2-3. Ardduino IDE : (Serial) Port



助言　mEDBGのCOMポートはスタート⇒コントロール パネル⇒デバイス マネージャ⇒ポートから下のように眺めることができます。



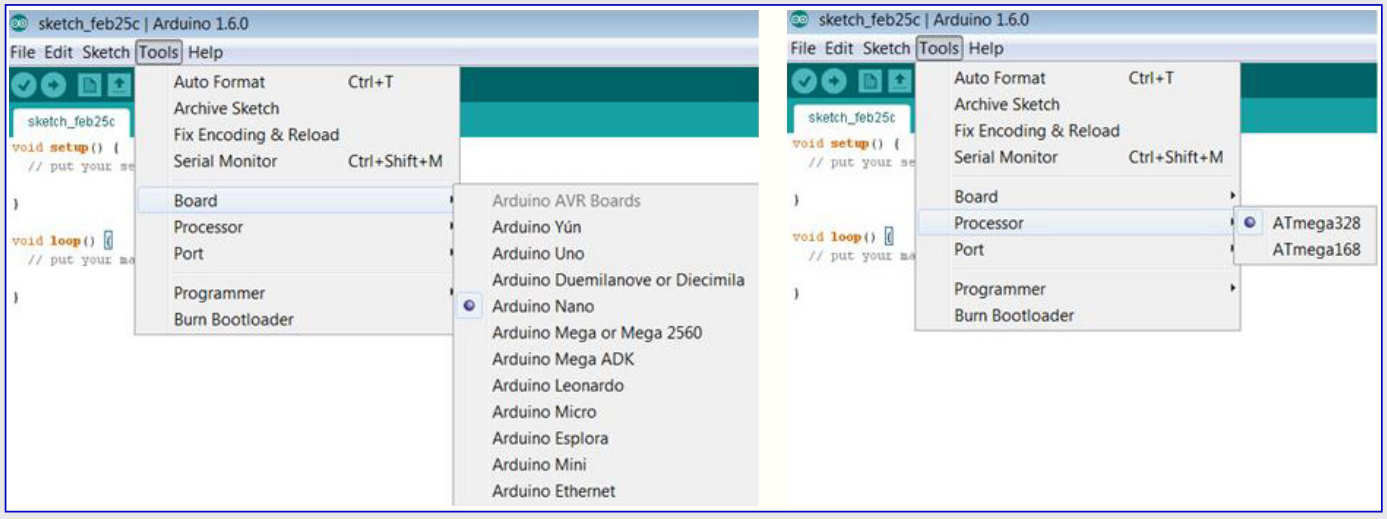
3. 基板(Board)は**表2-2.**に従って選択することができます。

表2-2. Arduino IDE : Board

Xplained Mini	Arduino IDEで選択する基板
ATmega328P Xplained Mini	Arduino Nano : ATmega328
ATmega168PB Xplained Mini	Arduino Nano : ATmega168

ここで、 選択：Tools⇒Boards⇒Arduino Nano
選択：Tools⇒Processor⇒ATmega328

図2-4. Ardduino IDE : BoardとProcessor

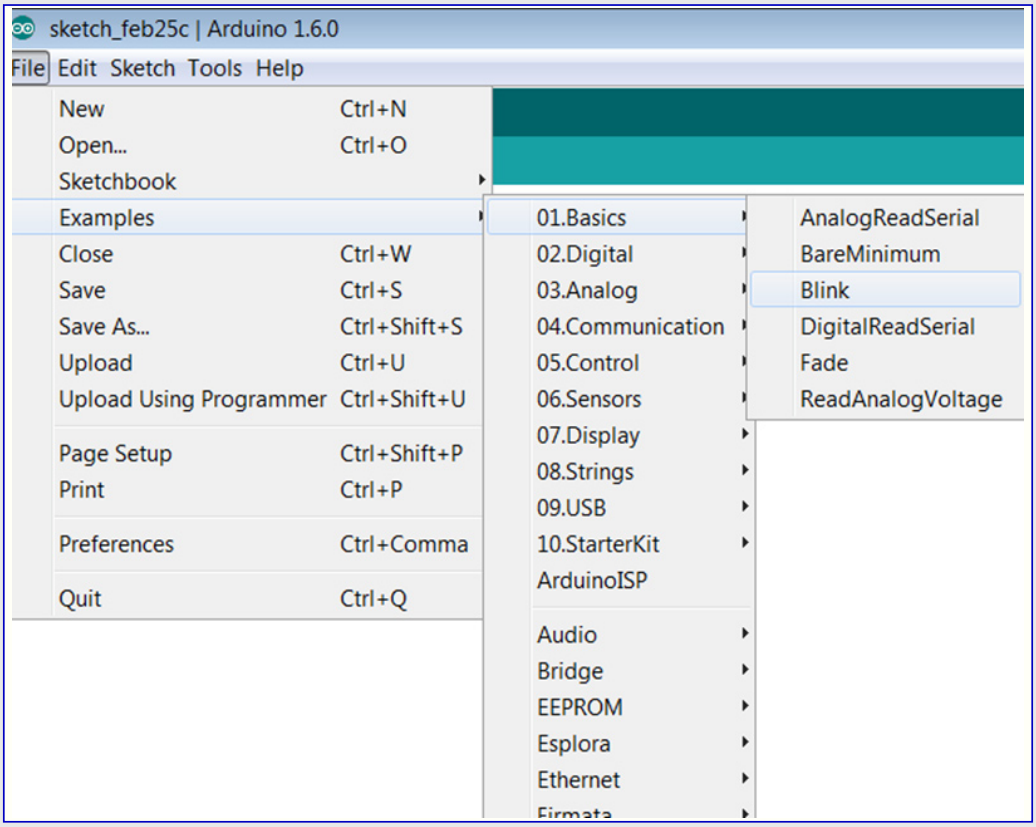


2.6. ﾌﾟﾛｸﾞﾗﾑのｱｯﾌﾟﾛｰﾄﾞ

結果 ATmega328P Xplained Mini基板はArduino IDEへ接続され、LED点滅が始まるでしょう。

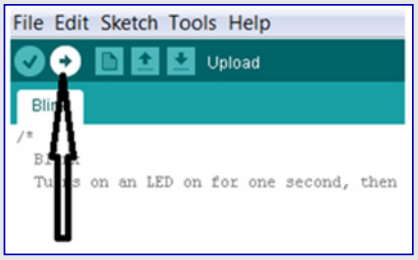
1. Arduino IDEで、Files⇒Examples⇒01.Basics⇒Blinkを選択してください。

図2-5. Ardduino IDE : Examples-Blink



2. Arduino IDEで“Upload”釦をｸﾘｯｸすることによってｽｹｯﾁをｱｯﾌﾟﾛｰﾄﾞしてください。

図2-6. Ardduino IDE : Upload



結果 ｳｨﾝﾄﾞｳの下部で’Compiling Sketch(ｽｹｯﾁをｺﾝﾊﾟｲﾙ中)’、’Uploading(ｱｯﾌﾟﾛｰﾄﾞ中)’、’Done uploading(ｱｯﾌﾟﾛｰﾄﾞ終了)’のﾒｯｾｰｼﾞが出現されるでしょう。

3. File⇒Save As...を選んで’Blink’ｽｹｯﾁを保存してください。ｽｹｯﾁを保存するのにどのﾊﾟｽをも選ぶことができます。

4. ATmega328P Xplained Mini基板でLED点滅を観察してください。

これで成功裏にArduino IDEへ接続されたATmega328P Xplained Mini基板があります。

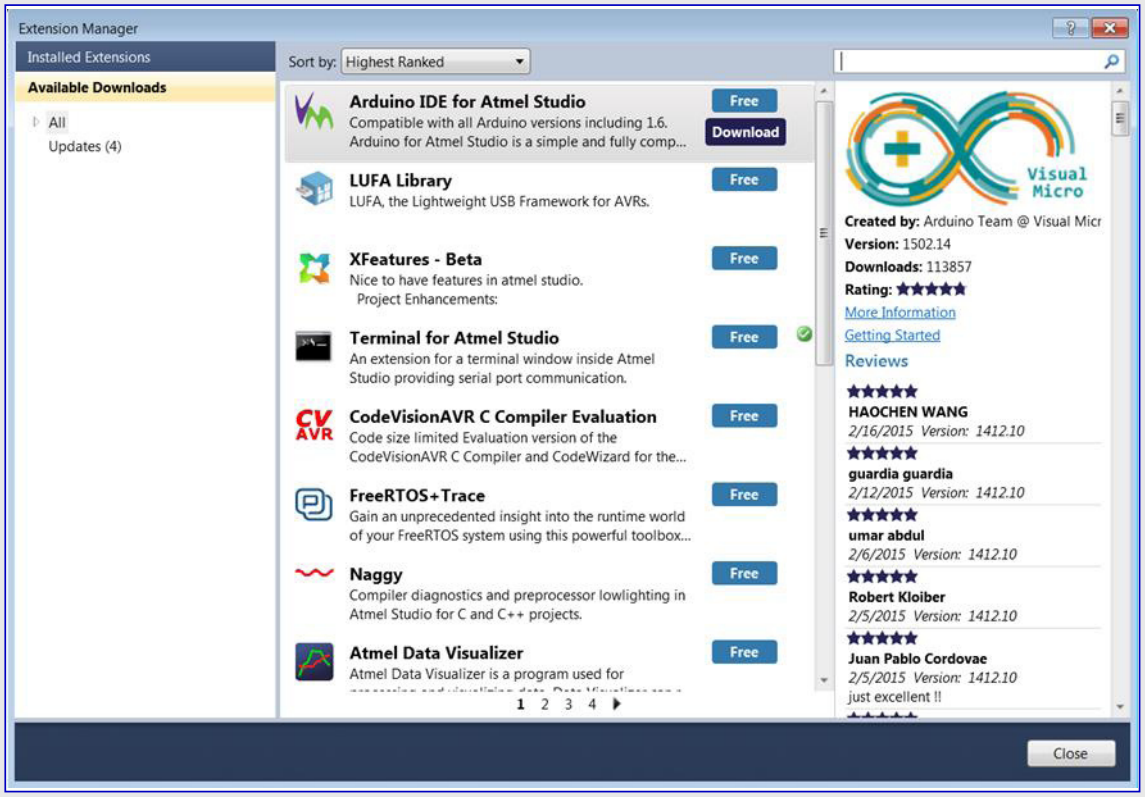
3. 課題2：Atmel StudioでのArduinoスケッチ作成

情報 Atmel StudioのArduino拡張はそれが提供する豊富な使用者インターフェースや専門的な機能を持つAtmel Studio内で書いてコンパイルし、そしてどのArduinoへもアップロードすることをArduinoのスケッチに許します。

3.1. 拡張のダウンロード

1. Atmel Studioを開いてください。
2. Tools⇒Extension Managerを選んでください。拡張管理部(Extension Manager)ウィンドウが開き、既定によってインストールされている拡張を示します。
3. “Available Download”任意選択をクリックしてください。

図3-1. Atmel Studio : Extension Manager



4. “Arduino IDE for Atmel Studio”を選んで”Download”アイコンをクリックしてください。
5. “Sign in to Extension Manager Dashboard”ウィンドウが開き、サインイン(sign-in)/登録(register)を尋ねます。

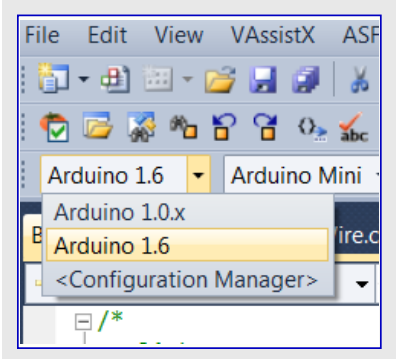
情報 既に登録していないなら、どうぞ登録してください。その後に“Sign in to Extension Manager Dashboard”ウィンドウを閉じてください。Atmelは提供されるEメールIDのために、あなたへ確認Eメールを送ります。あなたのEメールIDを確認するためにリンクをクリックしてください。“Sign in to Extension Manager Dashboard”で手順2.～4.を繰り返してください。Eメール IDと登録中に提供されるパスワードを使ってサインインしてください。

6. “Arduino IDE for Atmel Studio”拡張がダウンロードされ、その後にそれをインストールしてください。

3.2. スケッチ作成

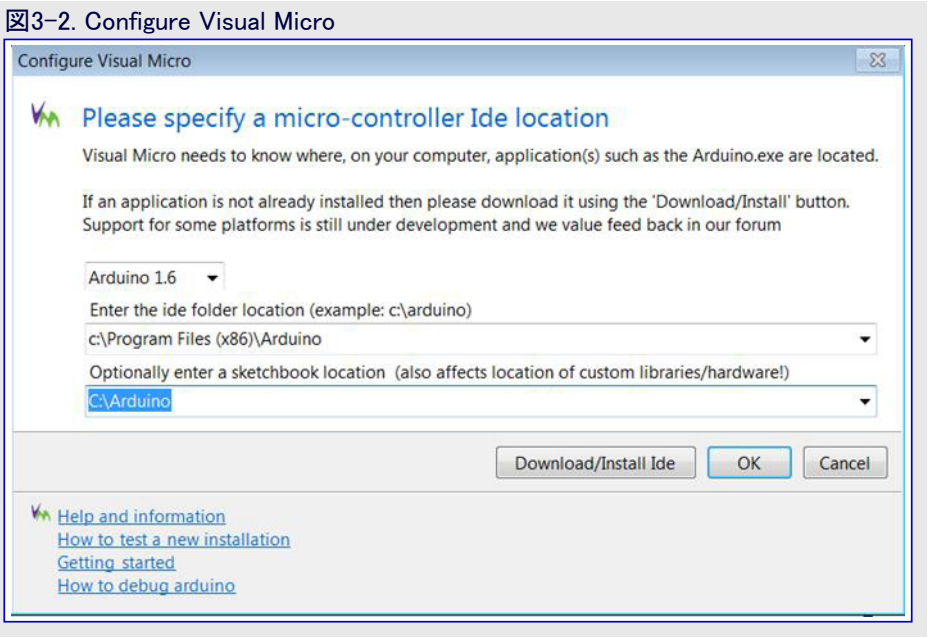
1. Atmel Studioを開いてください。

実施事項 Atmel StudioでArduino 1.6が一覧にされていることを確認し、それを選んでください。

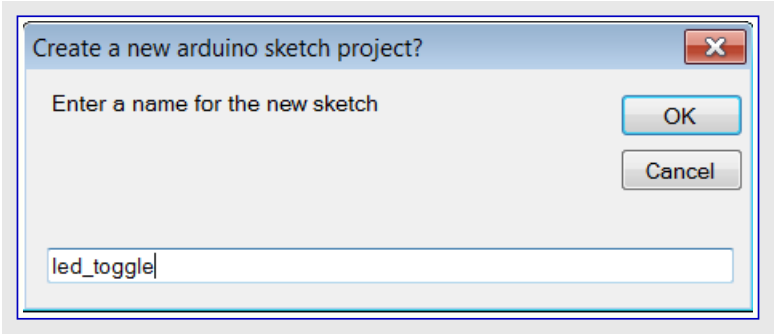


情報 それが一覧にされてなければ、<Configuration Manager>をクリックしてください。Arduino 1.6を選んでください。下図で示されるようにArduinoインストール フォルダのパスを追加してください。'OK'を選んでください。

図3-2. Configure Visual Micro



2. File⇒New⇒Sketch Projectを選んでください。
3. 下図で示されるようにスケッチ プロジェクト用の名前を入力して'OK'を選んでください。

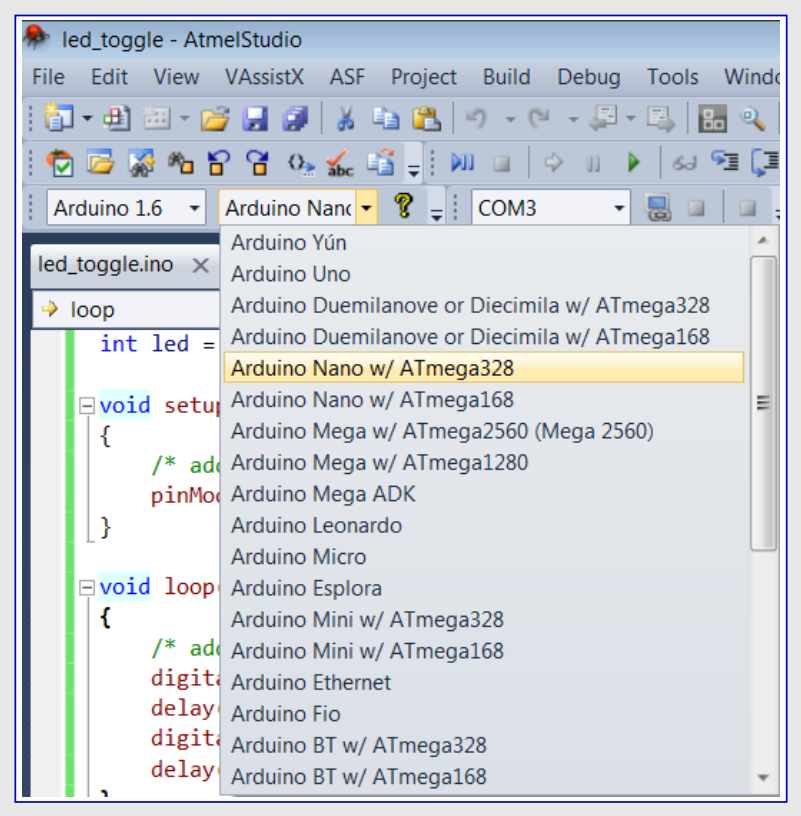


情報 led_toggle.inoスケッチは図3-2.で示されるように既定位置としてc:Arduinoに作成されるでしょう。

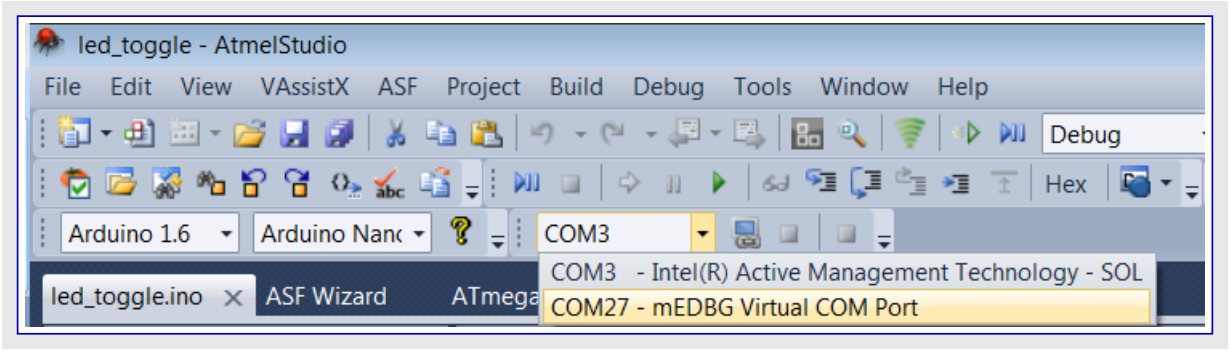
4. LEDを交互ON/OFFするようにコードを編集してください。

```
int led = 13;

void setup()
{
    /* 準備コードをここに追加してください。 */
    pinMode(led, OUTPUT);
}

void loop()
{
    /* 主プログラム コードをここに追加してください。 */
    digitalWrite(led,HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
}
```

5. 基板をArduino Nano w/ATmega328として選んでください。



6. mEDBGのCOMポート番号を選んでください。



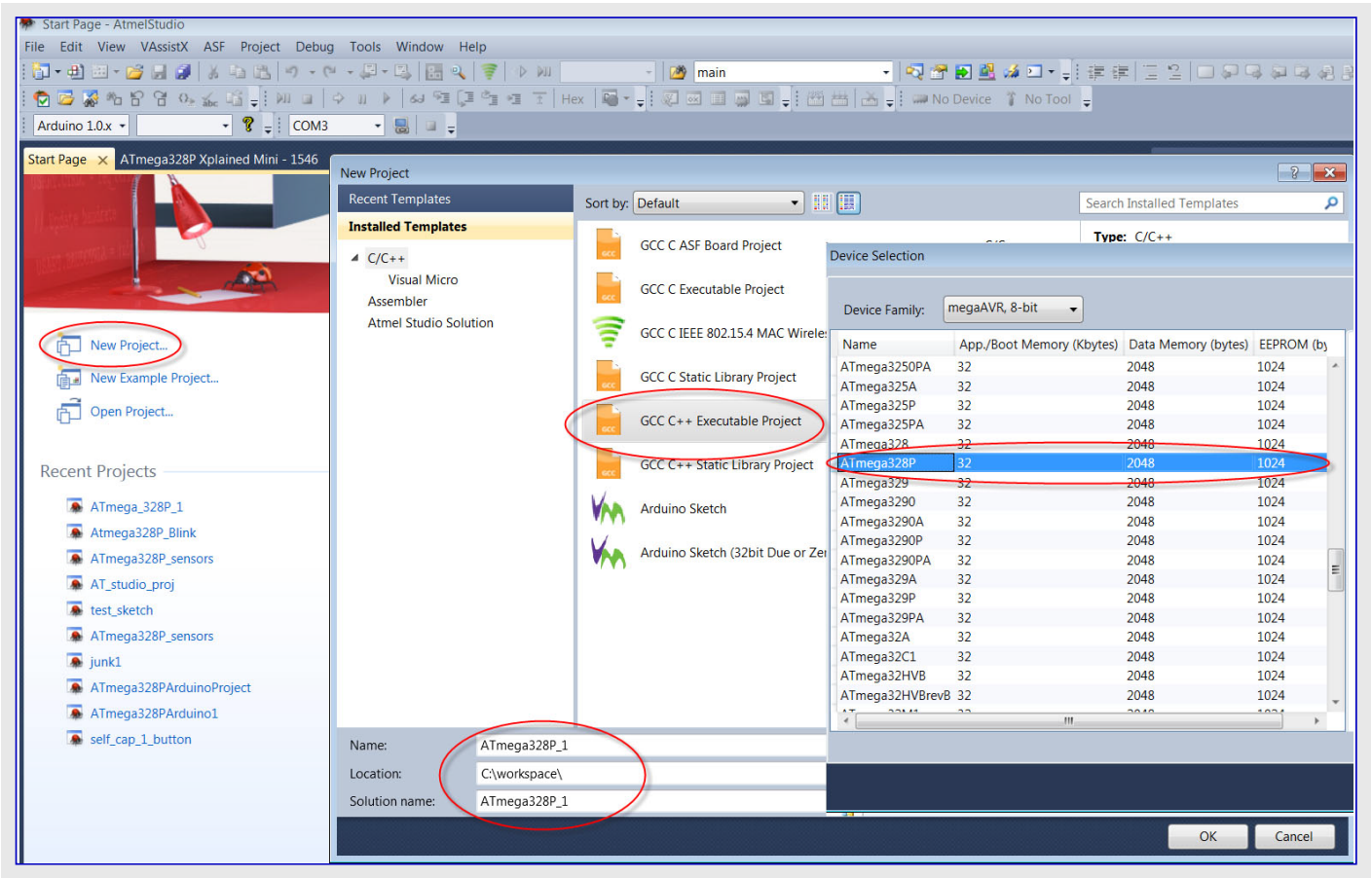
7.  アイコンをクリックすることによってプログラムをダウンロードしてください。

 **結果** ATmega328P Xplained Mini基板上のLEDが交互ON/OFFを始めるでしょう。

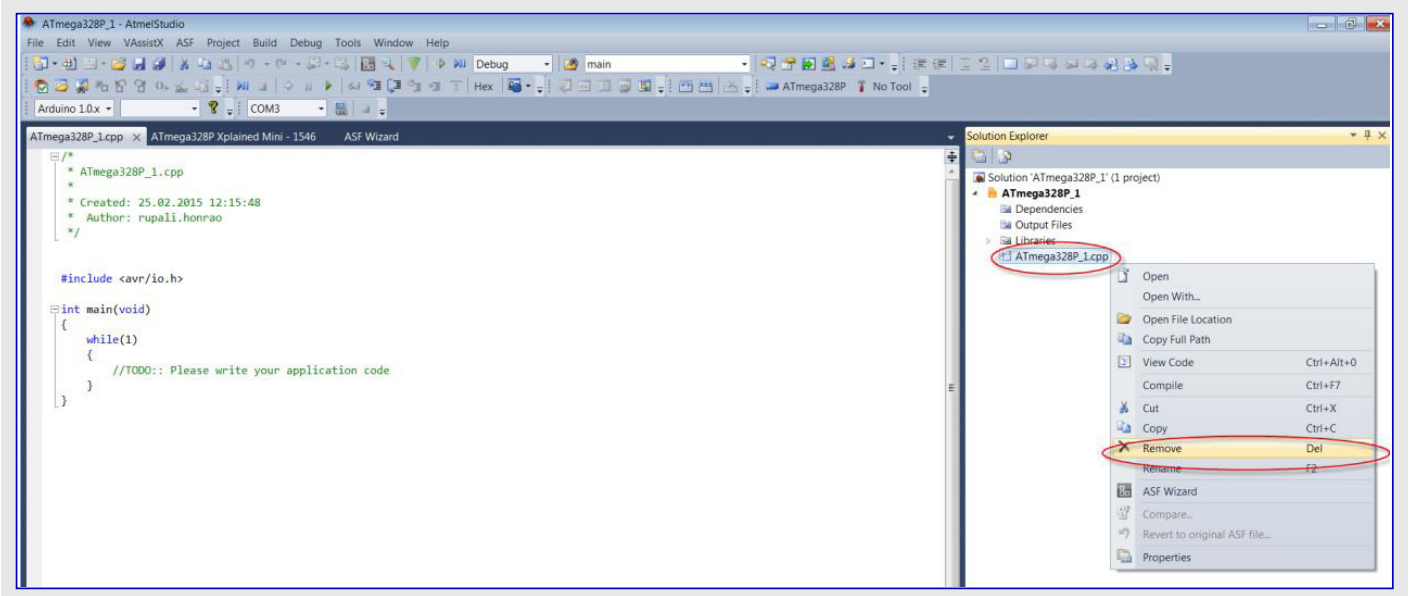
4. 既存ArduinoスケッチのAtmel Studio IDEへの移行

4.1. プロジェクト作成

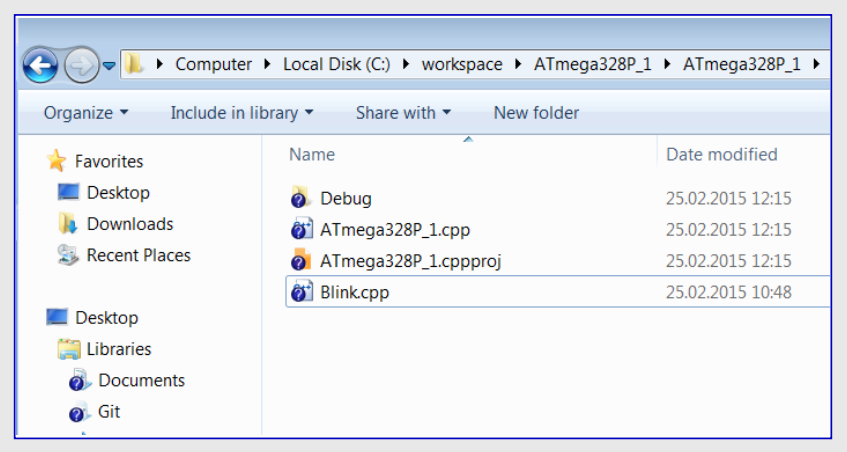
1. Atmel Studioを開き、”GCC C++ Executable Project”で新規プロジェクトを作成し、妥当な名前を与え、デバイスとしてATmega328Pを選んでください。
2. プロジェクトが作成されたフォルダの場所を記録することを確実にしてください。



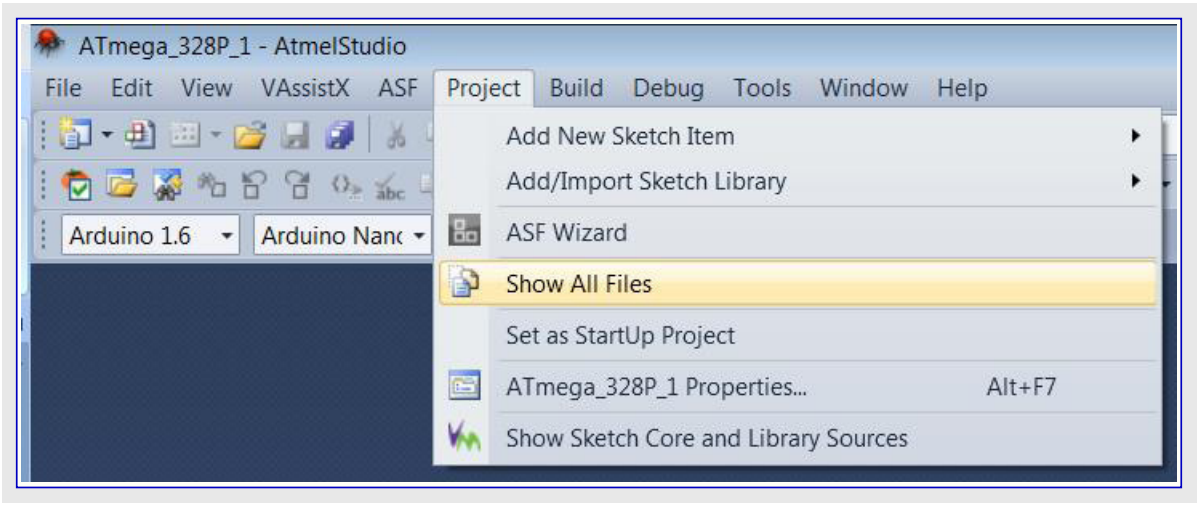
3. 結果のSolution Explorerで、下図で示されるように右クリックしてプロジェクトCPPファイル(ここではATmega328P_1.cpp)を削除してください。



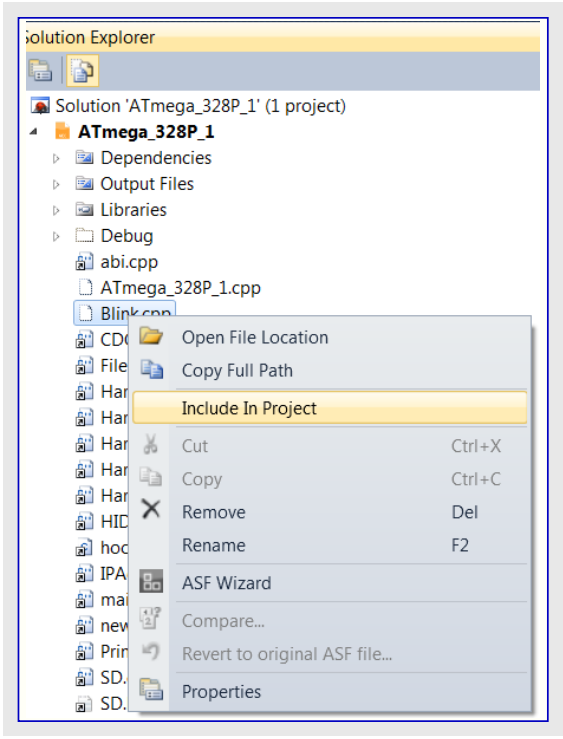
4. Windowsのエクスプローラで、
   a. あなたのスケッチ(先に保存したArduino点滅スケッチ/Blink.ino)を複写し、(例えAtmel Sudioのプロジェクトから取り去られたとしても、未だエクスプローラにある)ATmega328P_1.cppのあるAtmel Studioのプロジェクト副フォルダにそれを置いてください。
   b. あなたのスケッチ拡張を.cppに(Blink.ino⇒Blink.cpp)改名してください。下図をご覧ください。



5. Atmel StudioでProject⇒Show All Filesを選んでください。



6. Solution Explorerウィンドウで、Blink.cppファイルを右クリックし右図で示されるように”Include In Project”を選んでください。



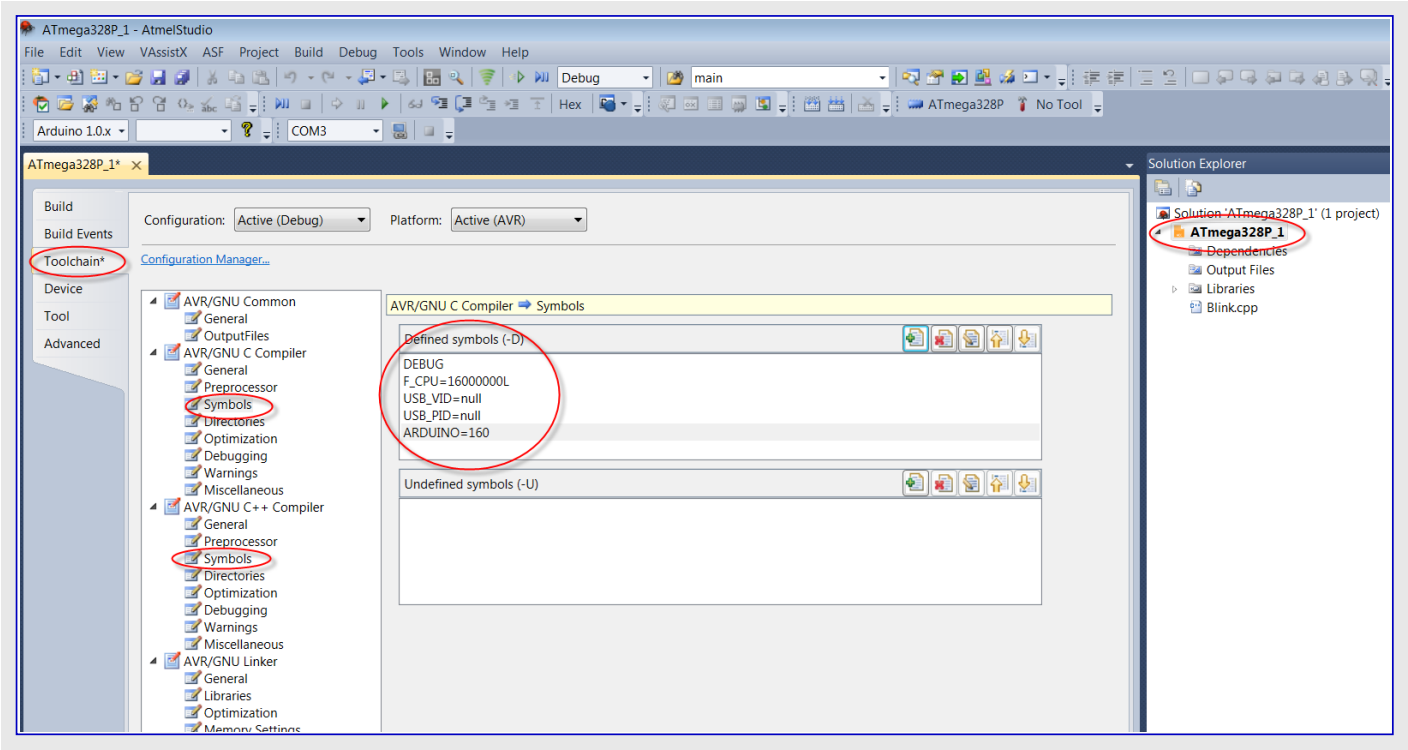
4.2. コンパイラ シンボル構成設定

1. Solution Explorerでプロジェクトを右クリックし、Propertiesを選ぶか、またはファイル メニュー(Alt+F7)でProject⇒プロジェクト名 Properties…へ行ってください。
2. Toolchain⇒AVR/GNU C Compiler⇒Symbols下に以下を追加してください。

   F_CPU=16000000L
   USB_VID=null
   USB_PID=null
   ARDUINO=160

   情報 これらはArduino特有シンボルです。

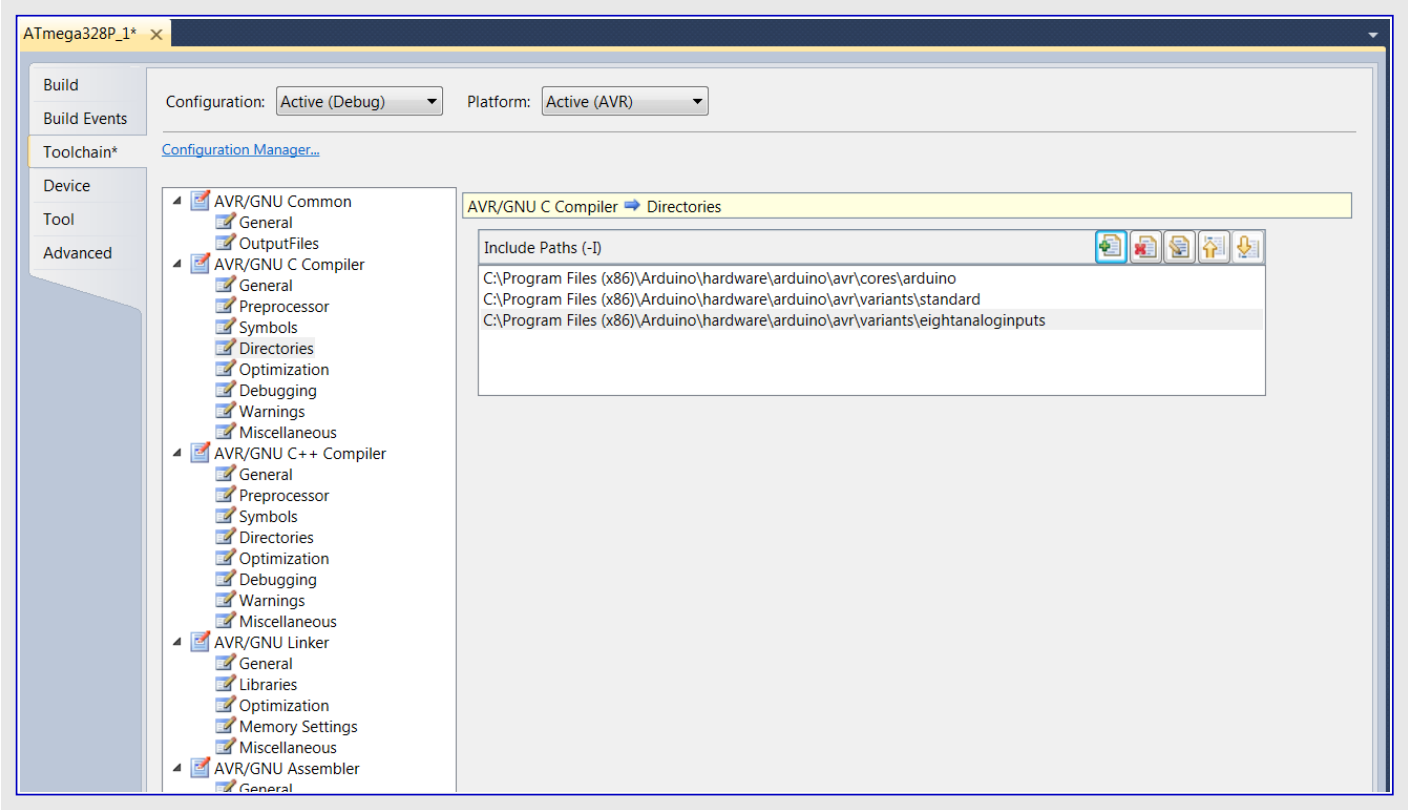
3. Toolchain⇒AVR/GNU C++ Compiler⇒Symbolsに対して繰り返してください。



4.3. コンパイラ ディレクトリ構成設定

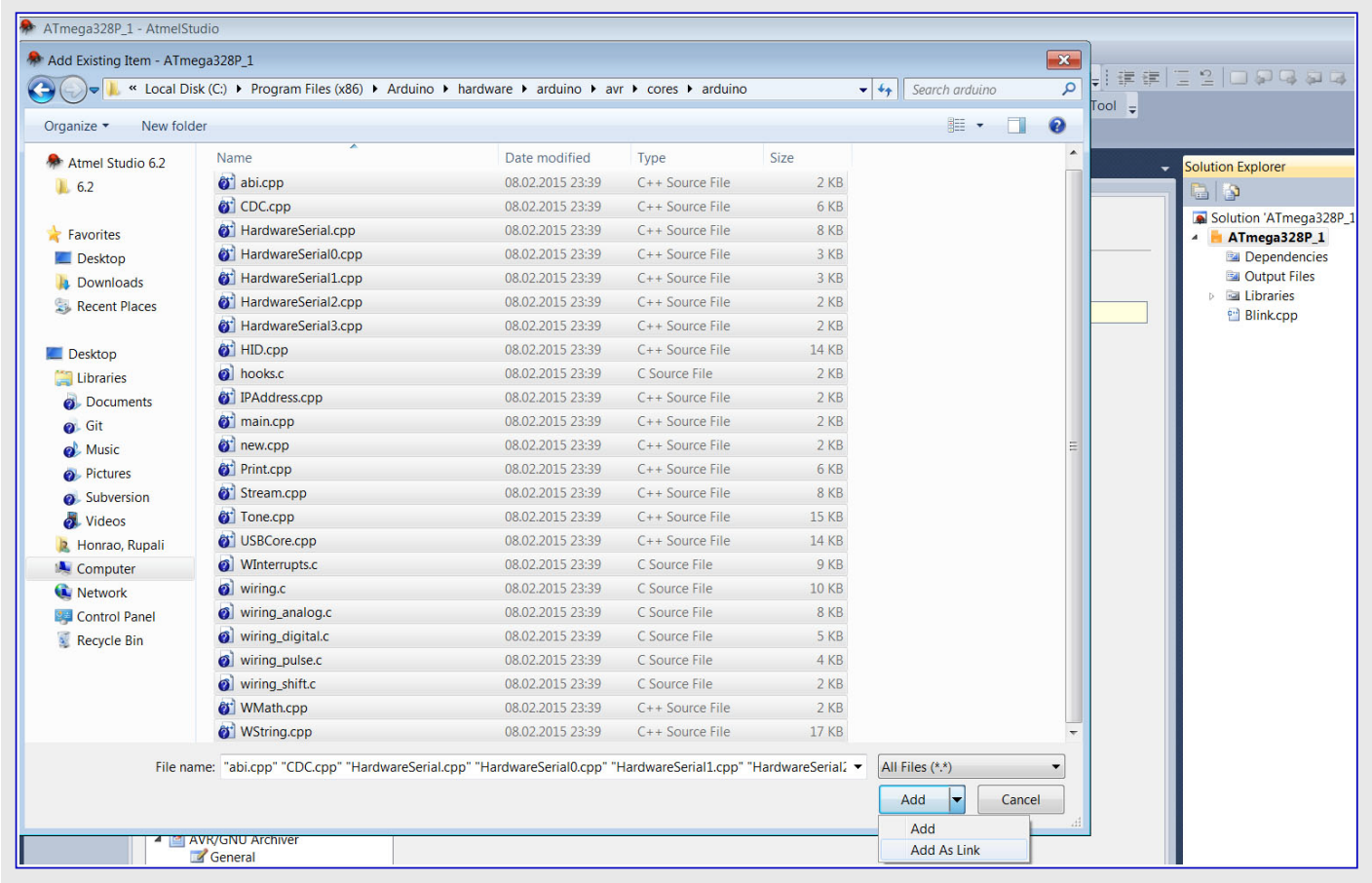
1. Toolchain⇒AVR/GNU C Compiler⇒Directories下に以下を追加してください。
2. C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥cores¥arduino
3. C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥variants¥standard
4. C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥variants¥eightanaloginputs
5. Toolchain⇒AVR/GNU C++ Compiler⇒Directoriesに対して繰り返してください。

警告 これらのパス追加中に”Relative Path”の選択解除を確実にしてください。



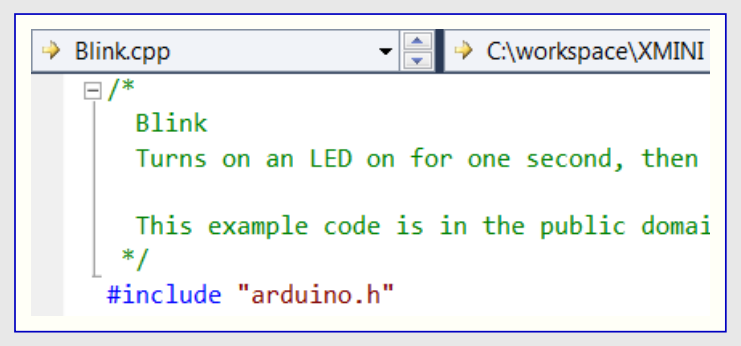
4.4. Arduino依存ファイル追加

1. プロジェクトを右クリックして”Add⇒Existing Item…”に行ってください。
2. 開いたダイアログ枠で閲覧部のパスC:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥cores¥arduinoを狙ってください。
   a. “File name”枠で*.c*<Enter>を入力してください。
   b. 全ファイルを多選択して”Add as Link”を選んでください。



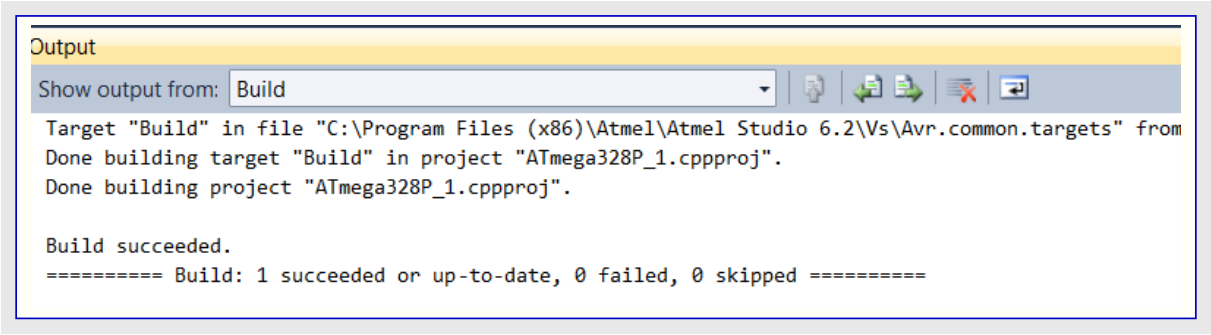
4.5. 解決策構築

1. Blink.cppの先頭に# include “arduino.h”を追加してください。



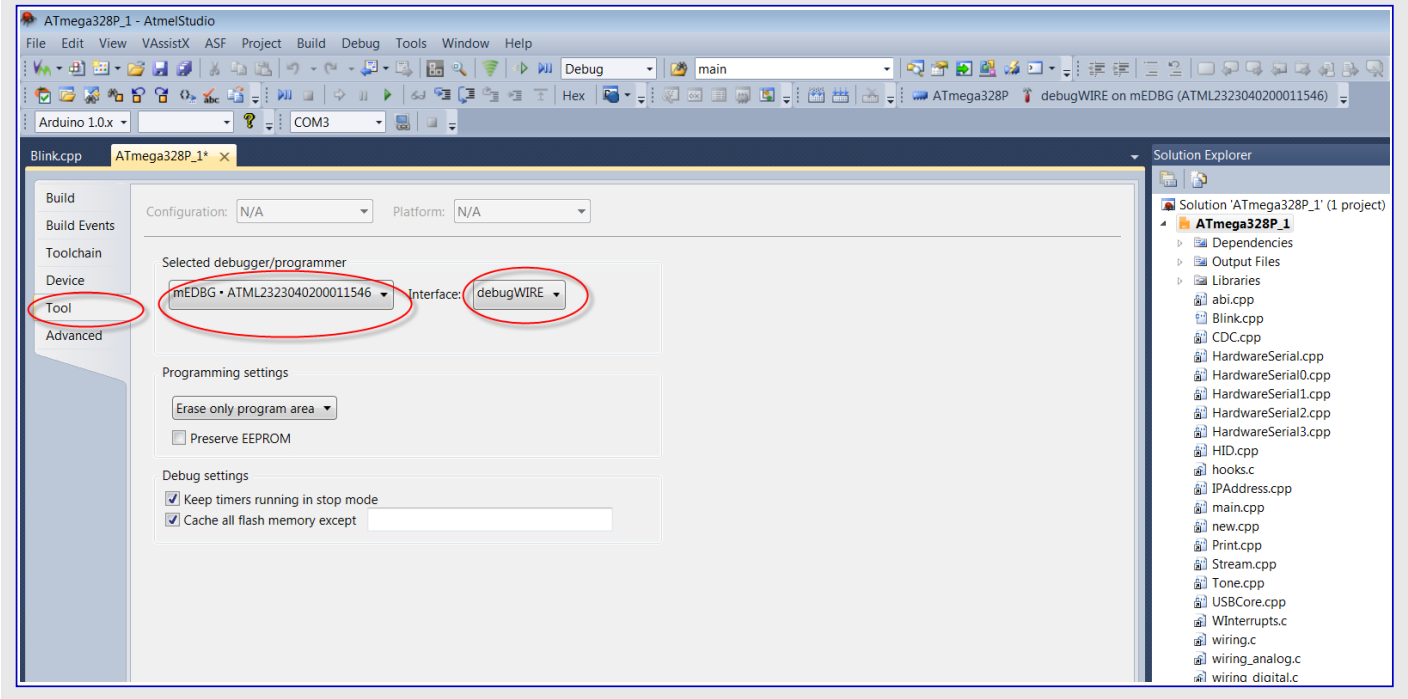
情報 今やArduino依存物が追加され、プロジェクトをコンパイルして目的対象内に読み込むことができます。

2. ファイル メニューで、Build⇒Build Solutionを選んでください。この構築は誤りなしで成功裏に終わるべきです。



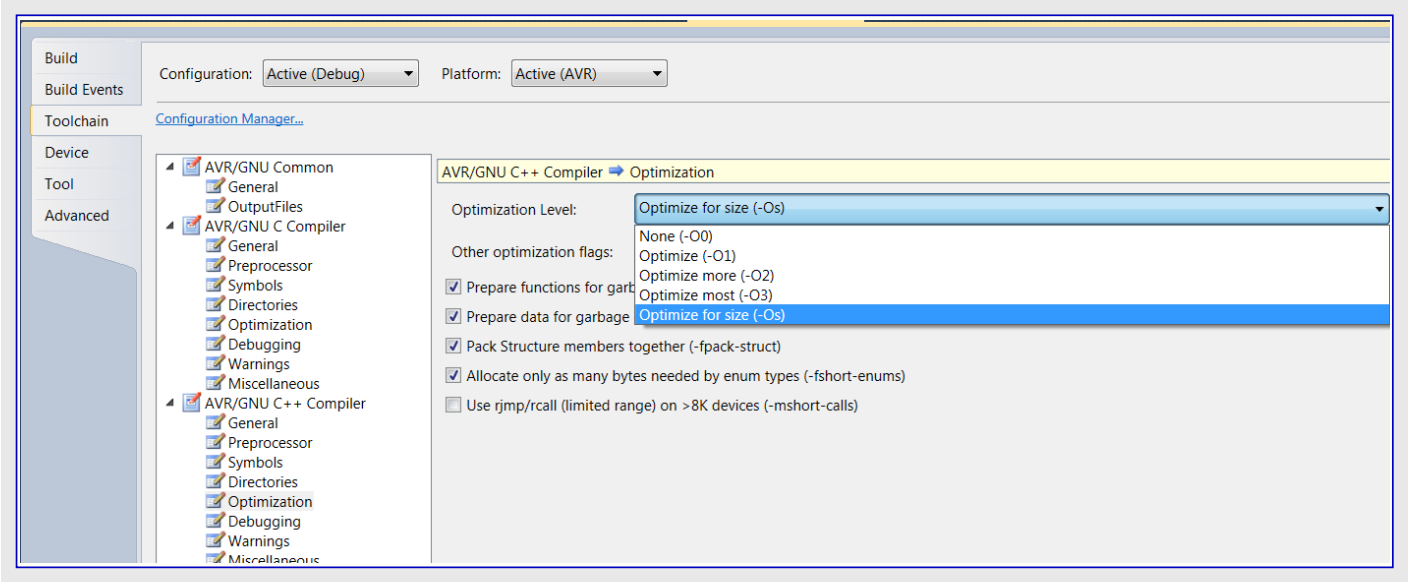
4.6. ATmega328P Xplained Mini基板差し込み

1. Solution Explorerでプロジェクトを右クリックしてファイル メニュー(Alt+F7)でPropertiesを選ぶか、またはProject⇒プロジェクト名 Properties…へ行ってください。
2. Tool下で、InterfaceとしてmEDBGとdebugWIREを選んでください。



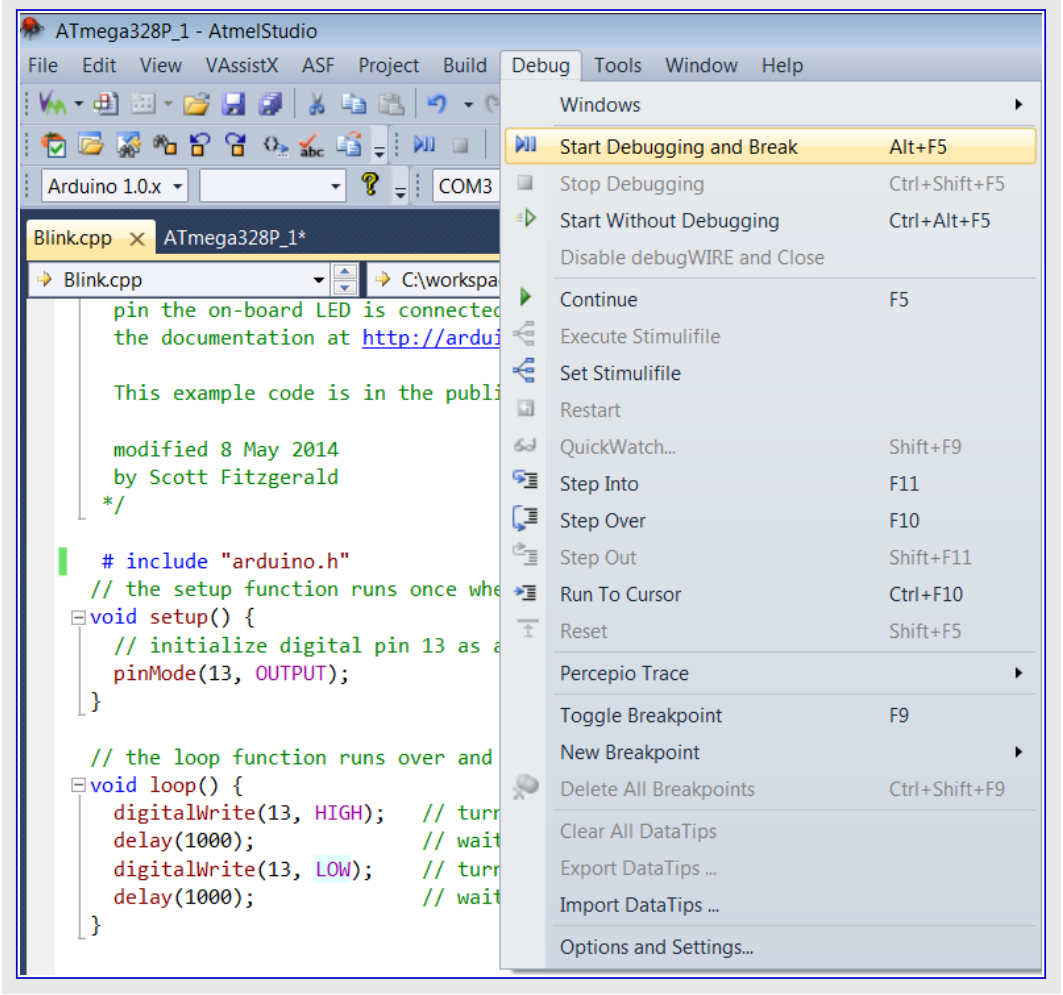
情報 いくつかの箇所でシステムはデバッガの更新を望むかもしれません。それを更新しましょう。

3. Toolchain⇒AVR/GNU C Compiler⇒Optimization下で最適化ﾚﾍﾞﾙ’Optimize for size(-Os)’を選んでください。
4. Toolchain⇒AVR/GNU C++ Compiler⇒Optimization下で最適化ﾚﾍﾞﾙ’Optimize for size(-Os)’を選んでください。

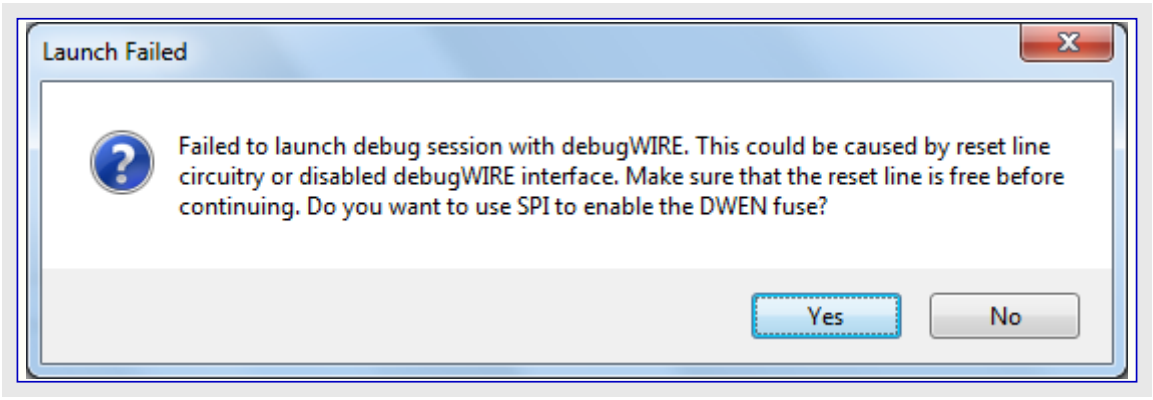


4.7. ﾃﾞﾊﾞｯｸﾞ

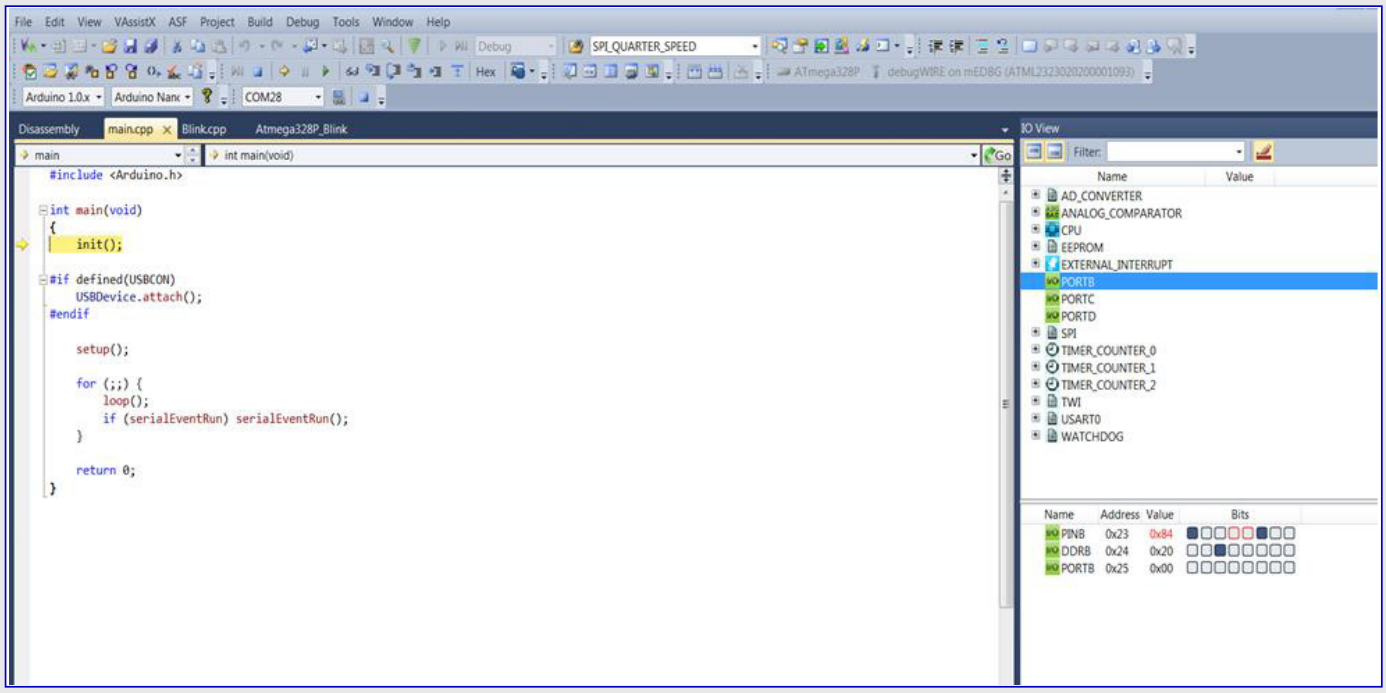
1. Debugを選び、’Start Debugging and Break’をｸﾘｯｸしてください。



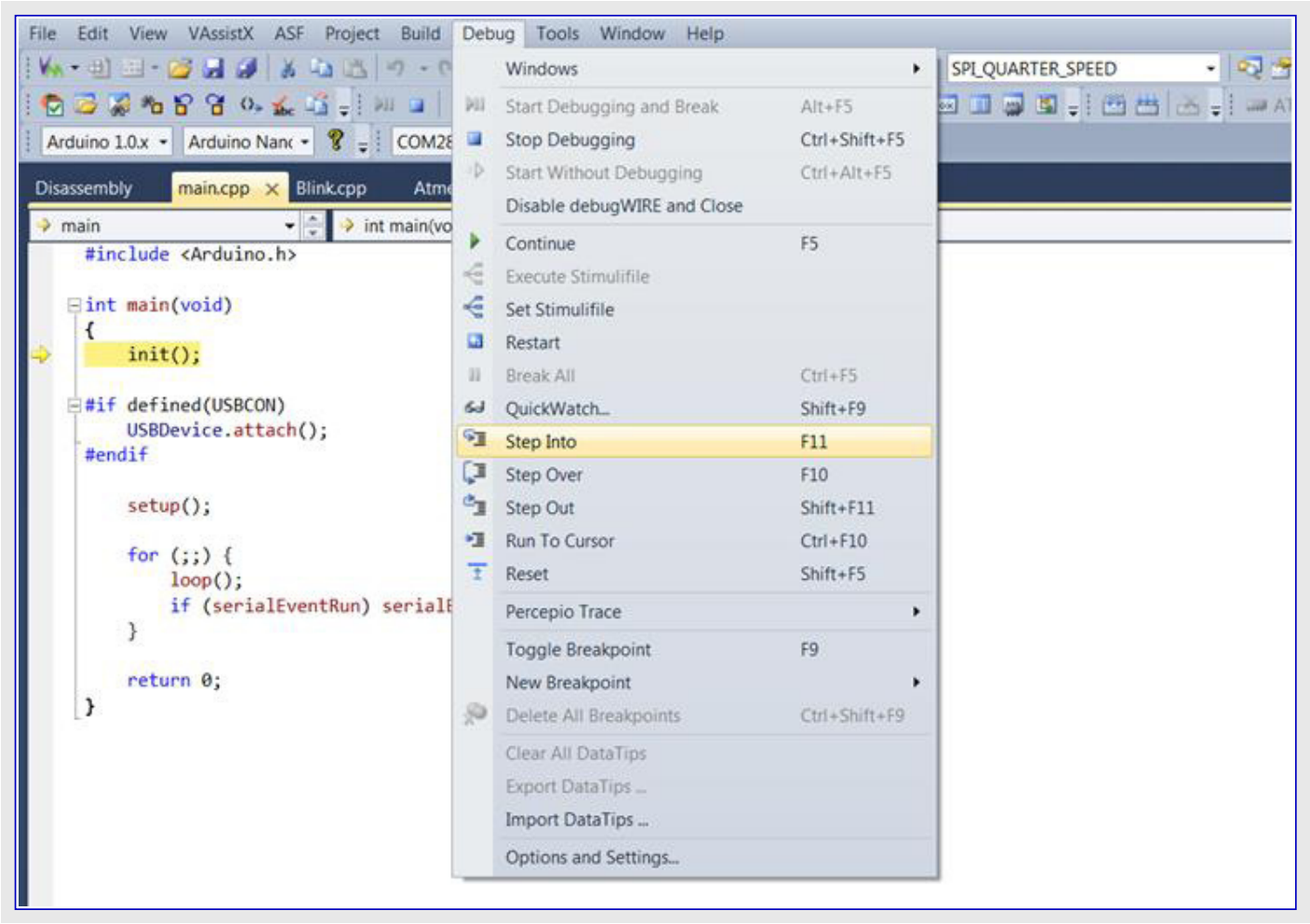
警告 DWENﾋｭｰｽﾞが許可されていない場合に異常ﾒｯｾｰｼﾞが表示されます。下で示されるように'Yes'をｸﾘｯｸしてﾋｭｰｽﾞを設定するのにAtmel StudioはISPを使います。



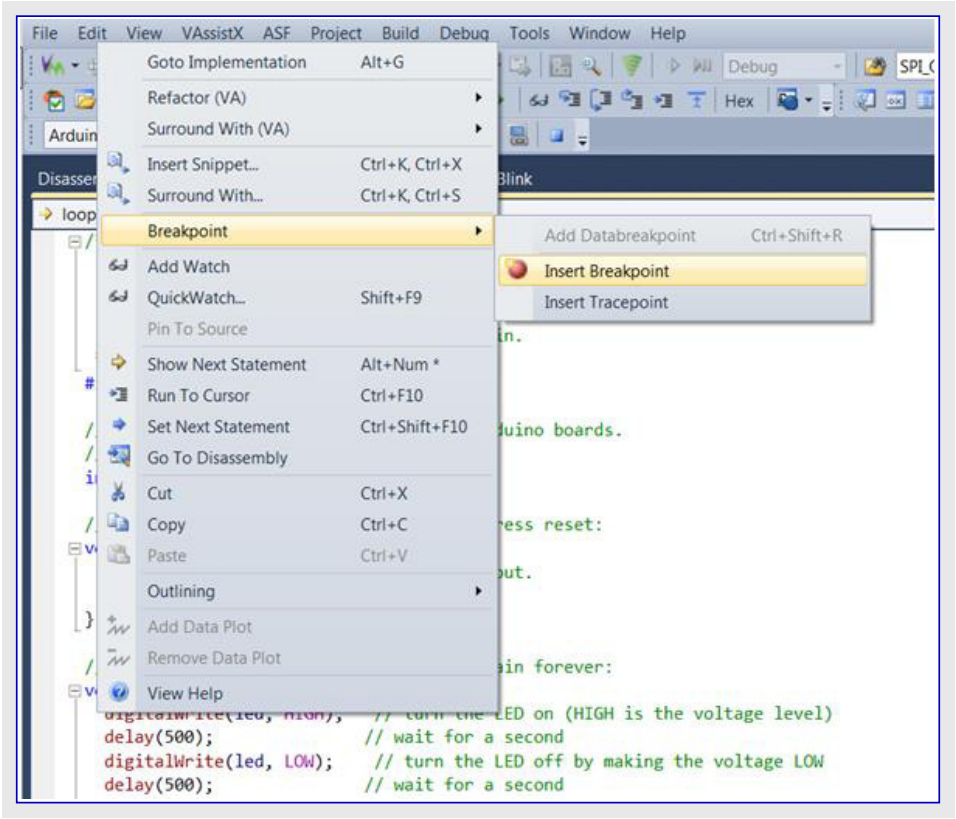
結果 ﾃﾞﾊﾞｯｶﾞが開始され、mainで中断されます。今やﾃﾞﾊﾞｯｸﾞを開始する準備が整いました。



情報 Debugメニューでは種々のﾃﾞﾊﾞｯｸﾞ任意選択が利用可能です。



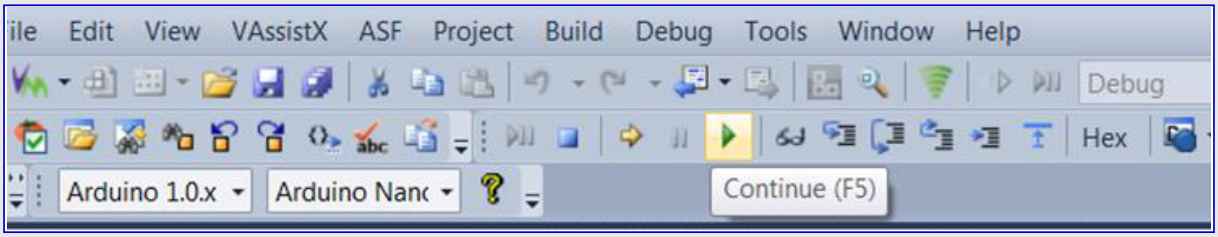
2. (‘Solution Explorer’のBlink.cppﾌｧｲﾙで)中断点(ﾌﾞﾚｰｸ ﾎﾟｲﾝﾄ)の挿入を望む場所であるｿｰｽ ｺｰﾄﾞ内の行へ行き、右ｸﾘｯｸしてBreakpoint⇒Insert Breakpointを選んでください。



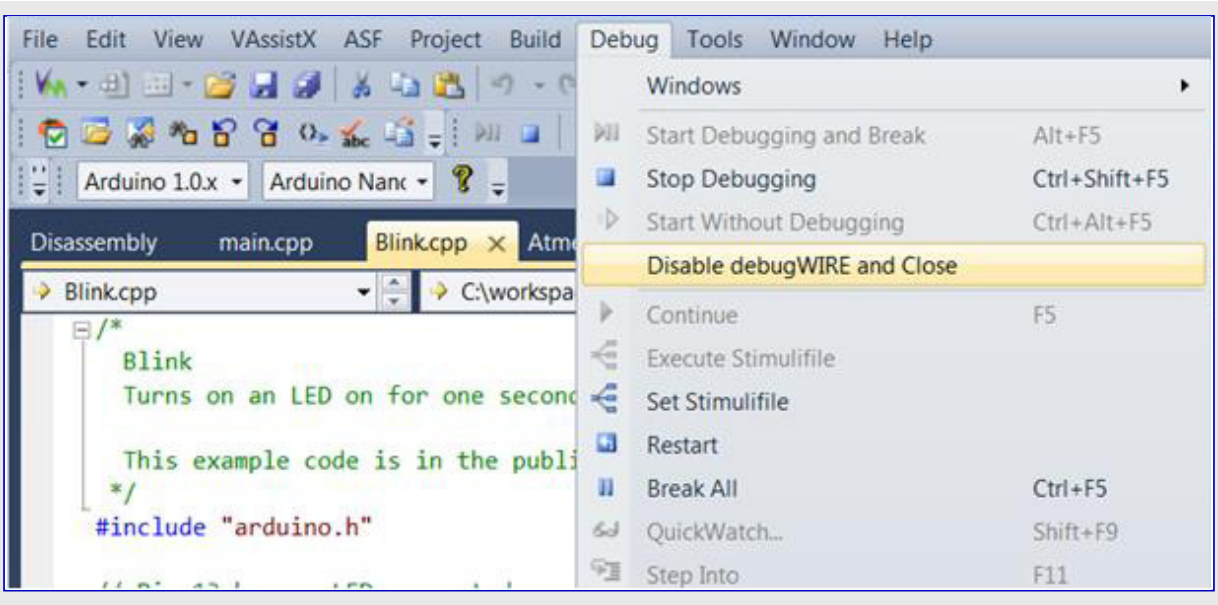
結果 中断点(ブレーク ポイント)が挿入されます。

```
  // the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

3. “Continue”をクリックすることによって中断点へ走行してください。必要な度毎に実行を一時停止と継続をすることができます。



4. Debug⇒Disable debugWIRE and Closeを選ぶことによってﾃﾞﾊﾞｯｸﾞ動作を抜け出してください。



警告 ﾃﾞﾊﾞｯｸﾞWIREを禁止することが重要です。

情報 ﾃﾞﾊﾞｯｸﾞWIREの禁止は目的対象をﾘｾｯﾄしてDWENﾋｭｰｽﾞをﾘｾｯﾄし、再びISPｲﾝﾀｰﾌｪｰｽを使うことができます。ﾌﾟﾛｸﾞﾗﾑ(0)されたDWENﾋｭｰｽﾞを持つことは、ｸﾛｯｸ系のいくつかの部分に全休止動作形態で動くことを許します。これは休止動作中にAVRの消費電力を増します。従ってﾃﾞﾊﾞｯｸﾞWIREが使われない時にDWENﾋｭｰｽﾞは常に禁止されるべきです。

5. 基板をﾘｾｯﾄしてLEDの点滅を観察してください。

5. ATmega328P応用

我々はA/D変換器(ADC)を使って光感知器を読み、I²Cインターフェースを使って温度感知器を読み、そしてSPIインターフェースを使ってSDカードにデータを格納する簡単な応用を作成しつつあります。また、光感知器値と温度感知器値はmEDBG COMポートを通して送信されます。

I²CについてはArduinoからの”Wire”ライブラリが必要とされ、SDカードについてはインクルードされるべき”SD library”が必要とされます。

5.1. コンパイラ構成設定

1. プロジェクト(ここではATmega328P_1)を右クリックしてPropertiesを選んでください。
2. Toolchain⇒AVR/GNU C Compiler⇒Directories下に以下を追加してください(前もって追加したディレクトリが未だ一覧にあるべきです)。
   C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥libraries¥Wire
   C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥libraries¥Wire¥utility
   C:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥libraries¥SPI
   C:¥Program Files (x86)¥Arduino¥libraries¥SD¥src
   C:¥Program Files (x86)¥Arduino¥libraries¥SD¥src¥utility
3. Toolchain⇒AVR/GNU C++ Compiler⇒Directoriesに対して繰り返してください。

警告 これらのパス追加中に”Relative Path”のチェック外しを確実にしてください。

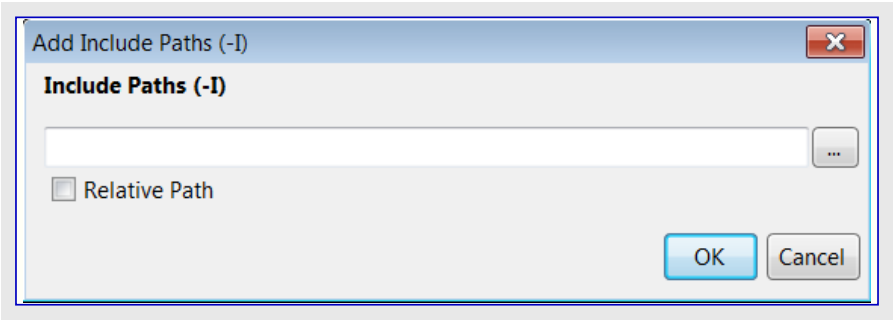
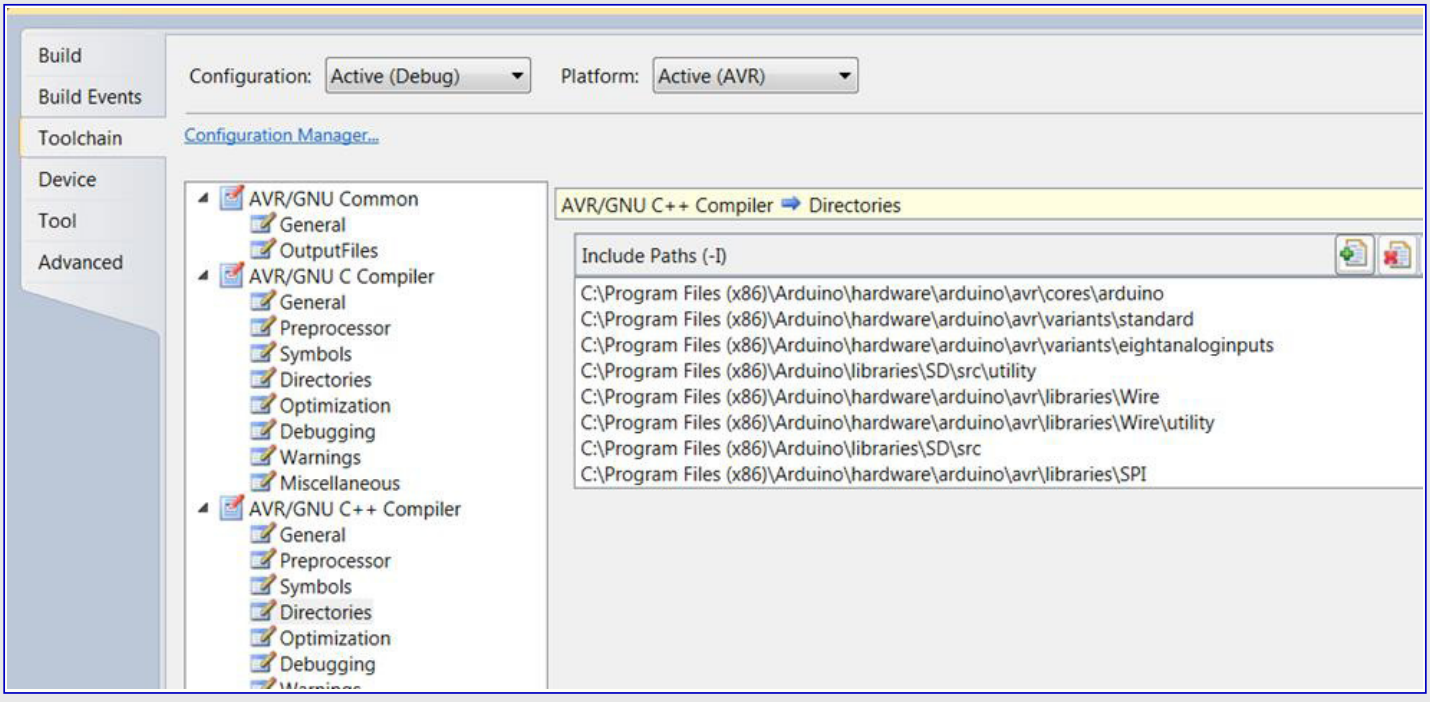


図5-1. Atmel Studio : Compiler Setup

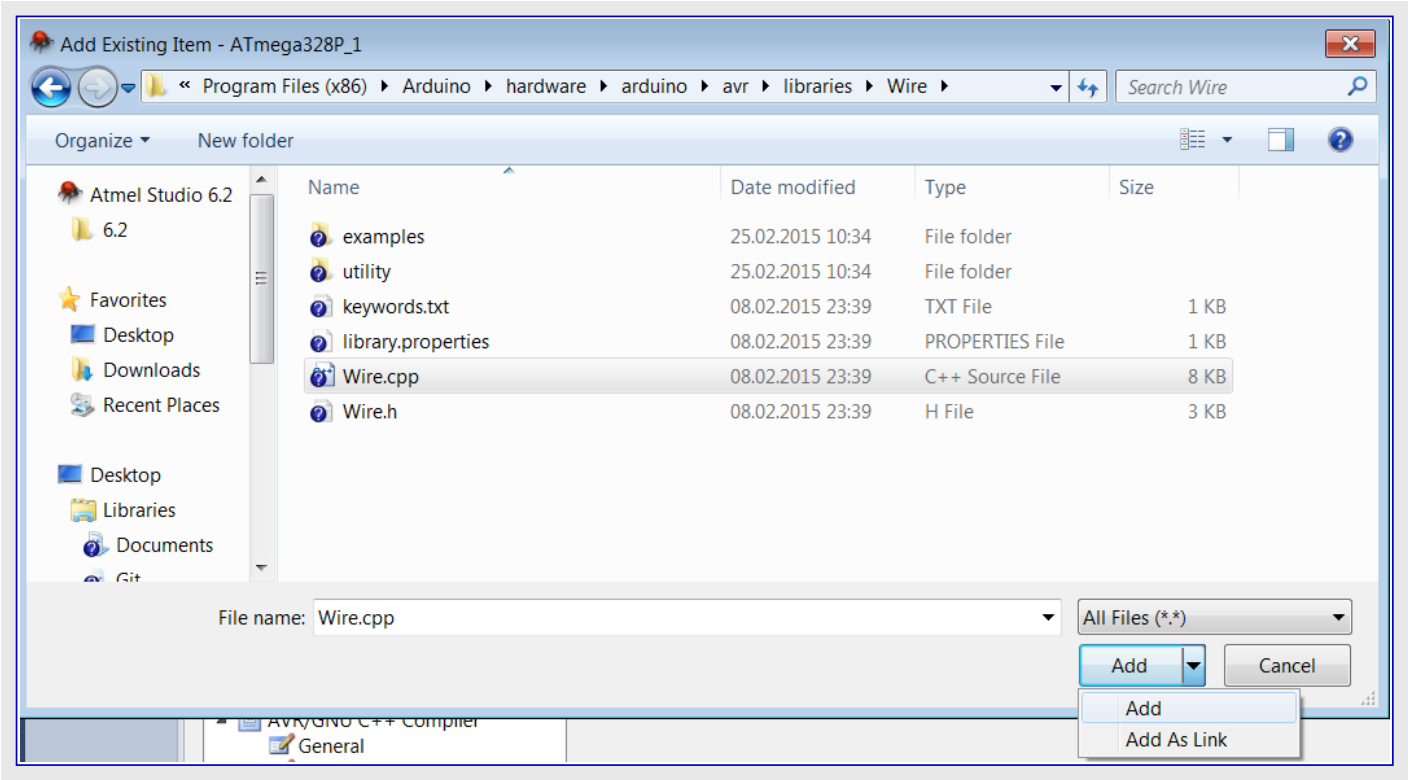


5.2. 依存ファイル追加

実施事項 ライブラリから実際のAtmel Studio プロジェクトへ.ccpソース ファイルを追加してください。

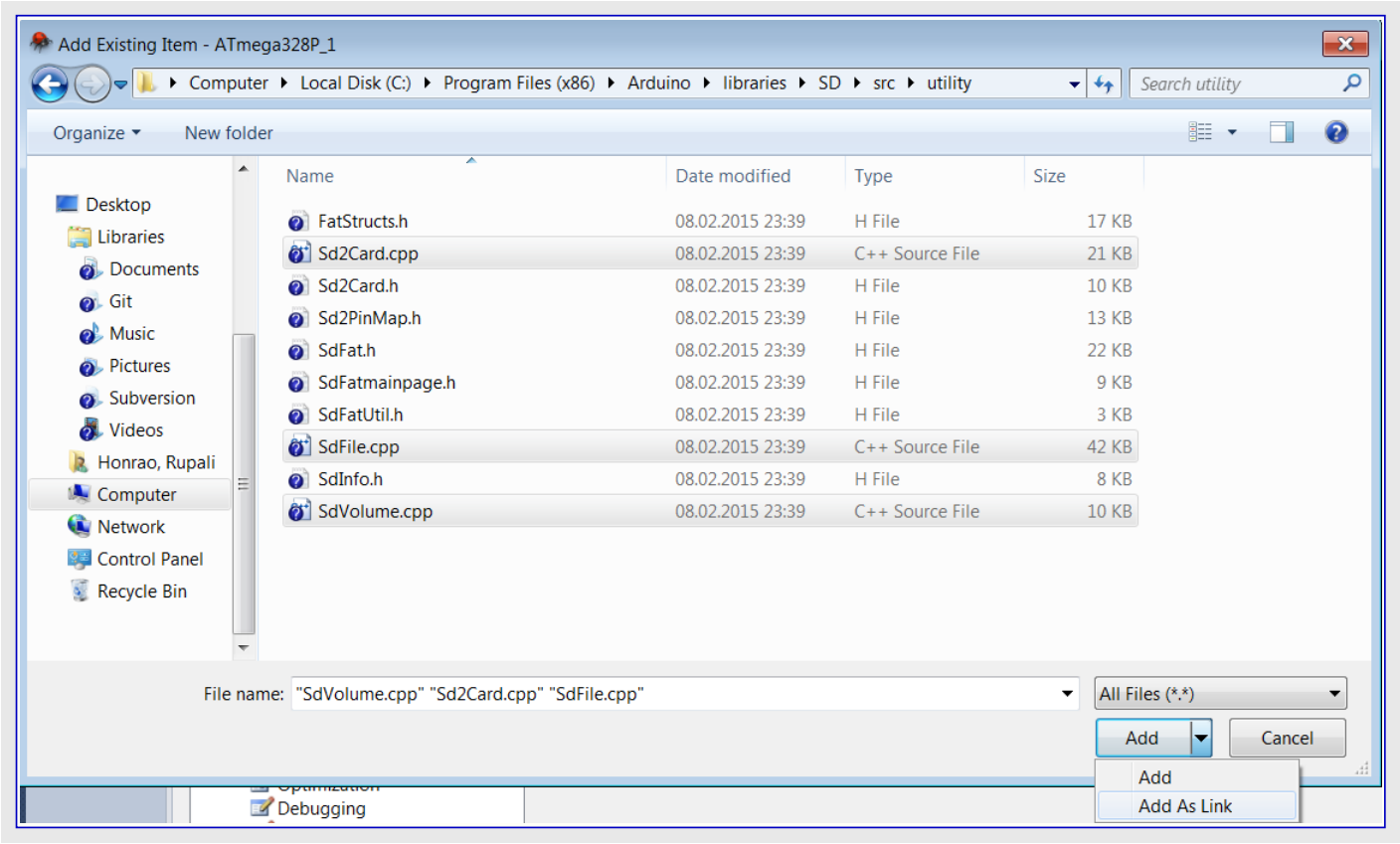
5.2.1. Wireライブラリに対して

1. プロジェクトを右クリックし、“Add⇒Existing Item…”へ行ってください。
2. 開いたダイアログ枠に於いて、閲覧部でC:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥libraries¥Wireを狙ってください。
3. Wire.ccpファイルを選択して‘As a link(リンクとして)’追加してください。
4. “wire¥utility”副ディレクトリに対して同様に繰り返し、twi.cファイルを選択して‘As a link(リンクとして)’追加してください。



5.2.2. SDライブラリに対して

1. プロジェクトを右クリックし、“Add⇒Existing Item…”へ行ってください。
2. 開いたダイアログ枠に於いて、閲覧部でC:¥Program Files (x86)¥Arduino¥hardware¥arduino¥avr¥libraries¥SPIを狙ってください。
3. SPI.ccpファイルを選択して‘As a link(リンクとして)’追加してください。
4. C:¥Program Files (x86)¥Arduino¥libraries¥SD¥srcに対して同様に繰り返してください。
   a. SD.cppとFile.cppのファイルを複数選択してください。
   b. “SD¥utility”副ディレクトリに対して同様に繰り返し、SdVolume.cpp, Sd2Card.cpp, SdFile.cppを選択して‘As a link(リンクとして)’追加してください。



5.3. 応用開発

実行 現在のBlink.cppﾌｧｲﾙをsensors.cppﾌｧｲﾙに改名し、LED点滅ｺｰﾄﾞ(即ち、setup()とloop())を削除してください。

実行 sensors.cppﾌｧｲﾙの先頭でSDとWireのﾗｲﾌﾞﾗﾘ用のﾍｯﾀﾞ ﾌｧｲﾙをｲﾝｸﾙｰﾄﾞください。

```
#include <Wire.h>
#include <SD.h>
```

実行 温度感知器に関する定義を追加してください。

```
#define AT30TSE_TEMPERATURE_TWI_ADDR    0x4F
#define AT30TSE_TEMPERATURE_REG         0x00
#define AT30TSE_TEMPERATURE_REG_SIZE    2
#define AT30TSE_NON_VOLATILE_REG        0x00

#define AT30TSE_CONFIG_RES_9_bit        0
#define AT30TSE_CONFIG_RES_10_bit       1
#define AT30TSE_CONFIG_RES_11_bit       2
#define AT30TSE_CONFIG_RES_12_bit       3
uint16_t resolution = AT30TSE_CONFIG_RES_12_bit;
```

実行 使用ﾋﾟﾝに対する定数を定義してください。

```
const int analogInPin = A0;             // ｱﾅﾛｸﾞ入力ﾋﾟﾝ
const int chipSelect = 10;              // SPI従装置選択ﾋﾟﾝ
```

実行 SDﾗｲﾌﾞﾗﾘを使う変数を定義してください。

```
File myFile;
```

実行 A/D変換結果と温度を格納する変数を定義してください。

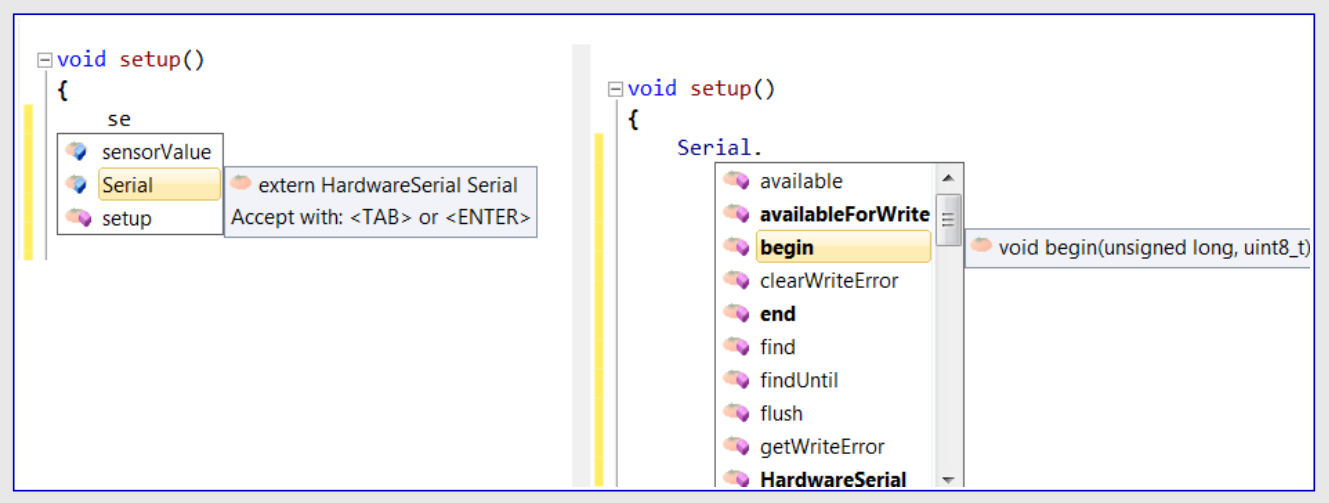
```
double temp_result;
int sensorValue = 0;                    // A/D変換器A0からの読み込み値
```

実行 温度感知器を読む関数を追加してください。

```
uint16_t at30tse_read_register(uint8_t reg, uint8_t reg_type, uint8_t reg_size)
{
    uint8_t buffer[2],i=0;
    buffer[0] = reg | reg_type;
    buffer[1] = 0;

    /* AT30TSEの内部ﾚｼﾞｽﾀ ﾎﾟｲﾝﾀ */
    Wire.beginTransmission(AT30TSE_TEMPERATURE_TWI_ADDR);
    Wire.write(buffer[0]);
    Wire.endTransmission();

    Wire.requestFrom(AT30TSE_TEMPERATURE_TWI_ADDR, reg_size);

    while(Wire.available())
    {
        buffer[i] = Wire.read(); // 文字としてﾊﾞｲﾄ受信
        i++;
    }

    return (buffer[0] << 8) | buffer[1];
}

double at30tse_read_temperature()
{
    /* 16ﾋﾞｯﾄ温度ﾚｼﾞｽﾀ読み込み */
    uint16_t data = at30tse_read_register(AT30TSE_TEMPERATURE_REG, AT30TSE_NON_VOLATILE_REG,
                                          AT30TSE_TEMPERATURE_REG_SIZE);

    double temperature = 0;
    int8_t sign = 1;

    /* 負検査と符号ﾋﾞｯﾄ解除 */
    if (data & (1 << 15)){
        sign *= -1;
        data &= ~(1 << 15);
    }

    /* 温度へ変換  */
    switch (resolution){
        case AT30TSE_CONFIG_RES_9_bit:
            data = (data >> 7);
            temperature = data * sign * 0.5;
            break;
        case AT30TSE_CONFIG_RES_10_bit:
            data = (data >> 6);
            temperature = data * sign * 0.25;
            break;
        case AT30TSE_CONFIG_RES_11_bit:
            data = (data >> 5);
            temperature = data * sign * 0.125;
            break;
        case AT30TSE_CONFIG_RES_12_bit:
            data = (data >> 4);
            temperature = data * sign * 0.0625;
            break;
        default:
            break;
    }
    return temperature;
}
```

実行 setup()初期化関数を追加してください。

情報 Atumel Studioで関数名を書き始めると直ぐに下で示されるように可能な関数名が一覧にされます。

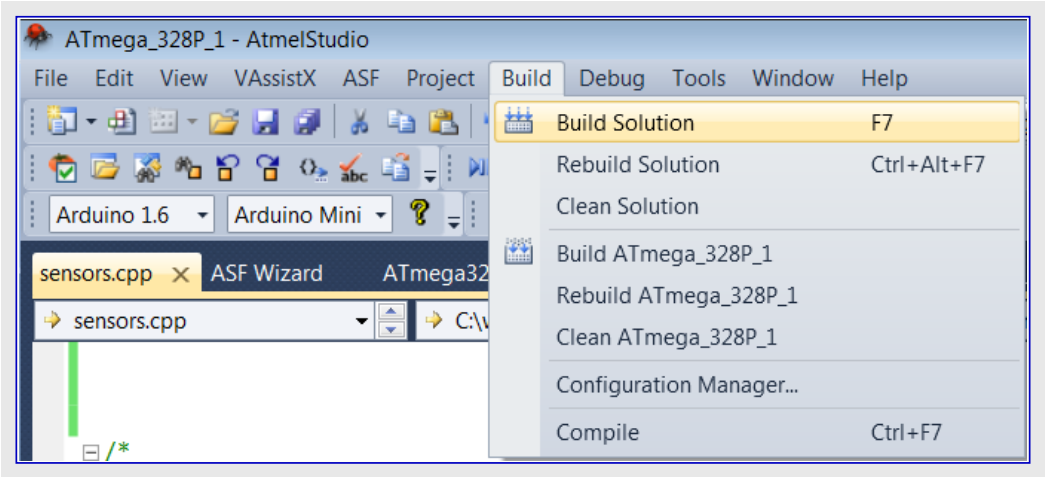



```
void setup()
{
    Serial.begin(9600);         // 直列通信を開く

    if (!SD.begin(chipSelect)) {
        Serial.println("SD Card initialization failed!");
        return;
    }
    Serial.println("SD Card initialization done.");

    SD.remove("data.txt");

    Wire.begin();
}
```

実行 変数とloop()関数追加してください。

```
bool sensor_flg=0;

void loop() {

    sensorValue = analogRead(analogInPin);          // A/D変換器(ADC)読み込み
    temp_result = at30tse_read_temperature();       // 温度読み込み

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)              // 特殊な感知器値でファイル内にデータ書き込み
    {
        sensor_flg=0;

        myFile = SD.open("data.txt", FILE_WRITE);

        if (myFile)                 // ファイルが開いていればそれに書き込み
        {
            myFile.print("sensor = " );
            myFile.print(sensorValue);

            myFile.print("    temp = ");
            myFile.print(temp_result);
            myFile.print("¥n");

            myFile.close();         // ファイルを閉じる
        }
        else
        {
            // ファイルが開かなかった場合は異常表示
            Serial.println("error opening data.txt");
        }

        // 読むために改めてファイルを開く
        myFile = SD.open("data.txt");
        if (myFile)
        {
            // 何も無くなるまでファイルから読み込み
            while (myFile.available()) {
                Serial.write(myFile.read());
            }
            // ファイルを閉じる
            myFile.close();
        } else {
            // ファイルが開かなかった場合は異常表示
            Serial.println("error opening data.txt");
        }
    }
    delay(500);
}
```

実行 ファイルを保存し、'Build'タブを選択するか、またはキーボードでF7キーを押すことによって解決策を構築してください。



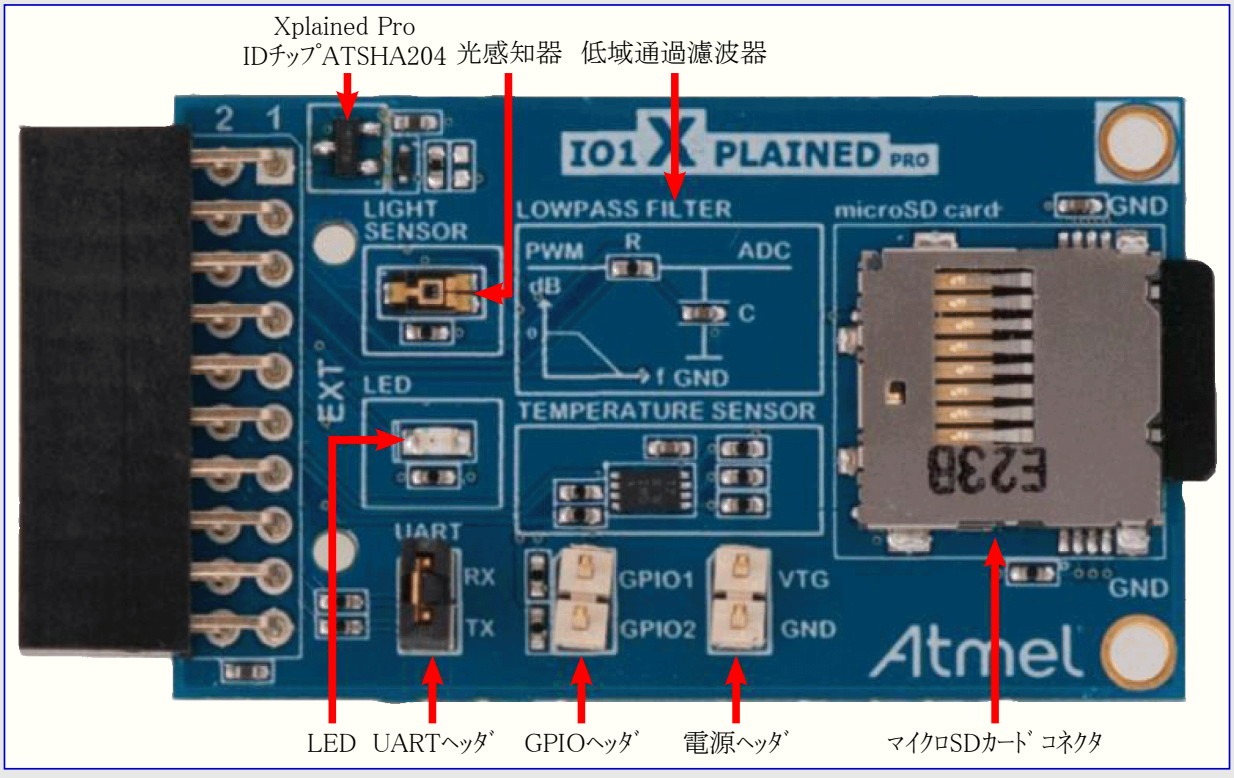
結果 この構築は異常なしで成功裏に終了すべきです。

5.4. ハードウェア接続

光感知器と温度感知器はIO1 Xplained Pro基板上に存在します。

IO1 Xplained ProはAtmel Xplained Pro評価基盤に対する拡張基板です。それはマイクロSD、温度感知器、光感知器、その他色々を含むXplained Pro MCU基板に対して広く多様な機能を与えるように設計されます。IO1拡張基板は図5-2.で示されます。

図5-2. IO1 Xplained Pro



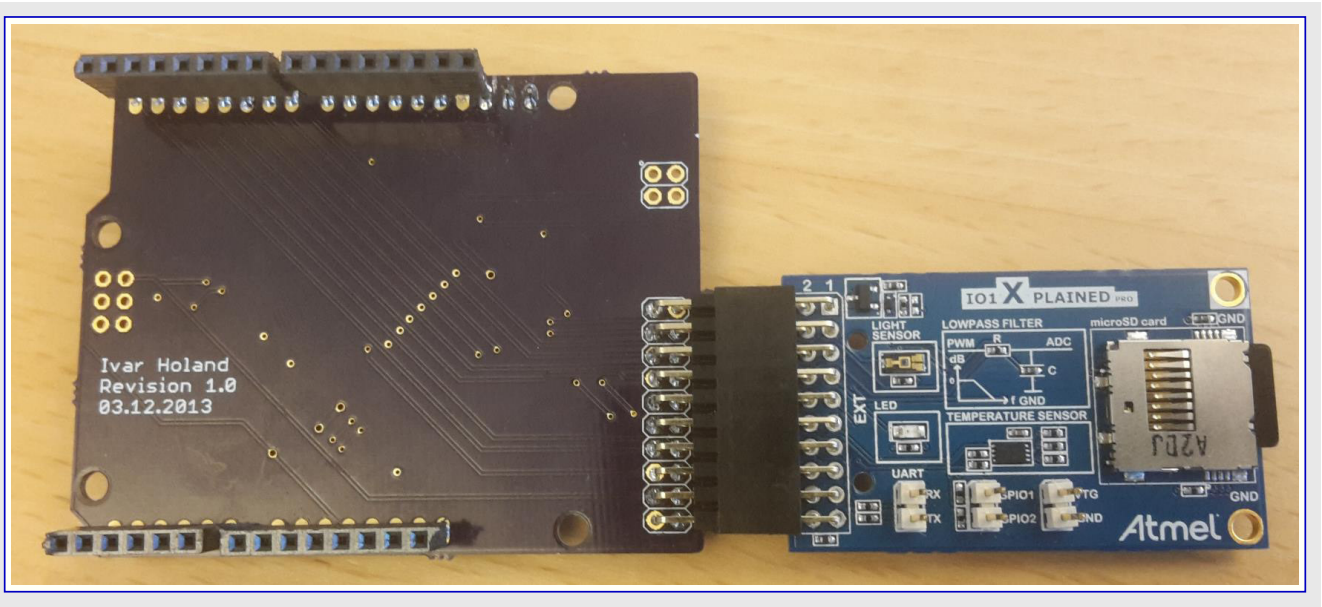
ATmega328P Xplained MiniへのIO1 Xplained Pro基板の実際の接続は表5-1.で示されます。

表5-1. 接続 : IO1 Xplained Pro – ATmega328P Xpalined Mini

IO1 Xplained Pro		ATmega328P Xplained Mini	
ピン番号	名前	ピン名	MCUピン
3	ADC	PC0	A0
11	TWI_SDA	PC4	SDA
12	TWI_SCL	PC5	SCL
2	GND	GND	–
20	VCC	3V3	–
15	SPI_SS_A	PB2	SPI_SS (D10)
16	SPI_MOSI	PB3	SPI_MOSI
17	SPI_MISO	PB4	SPI_MISO
18	SPI_SCK	PB5	SPI_SCK

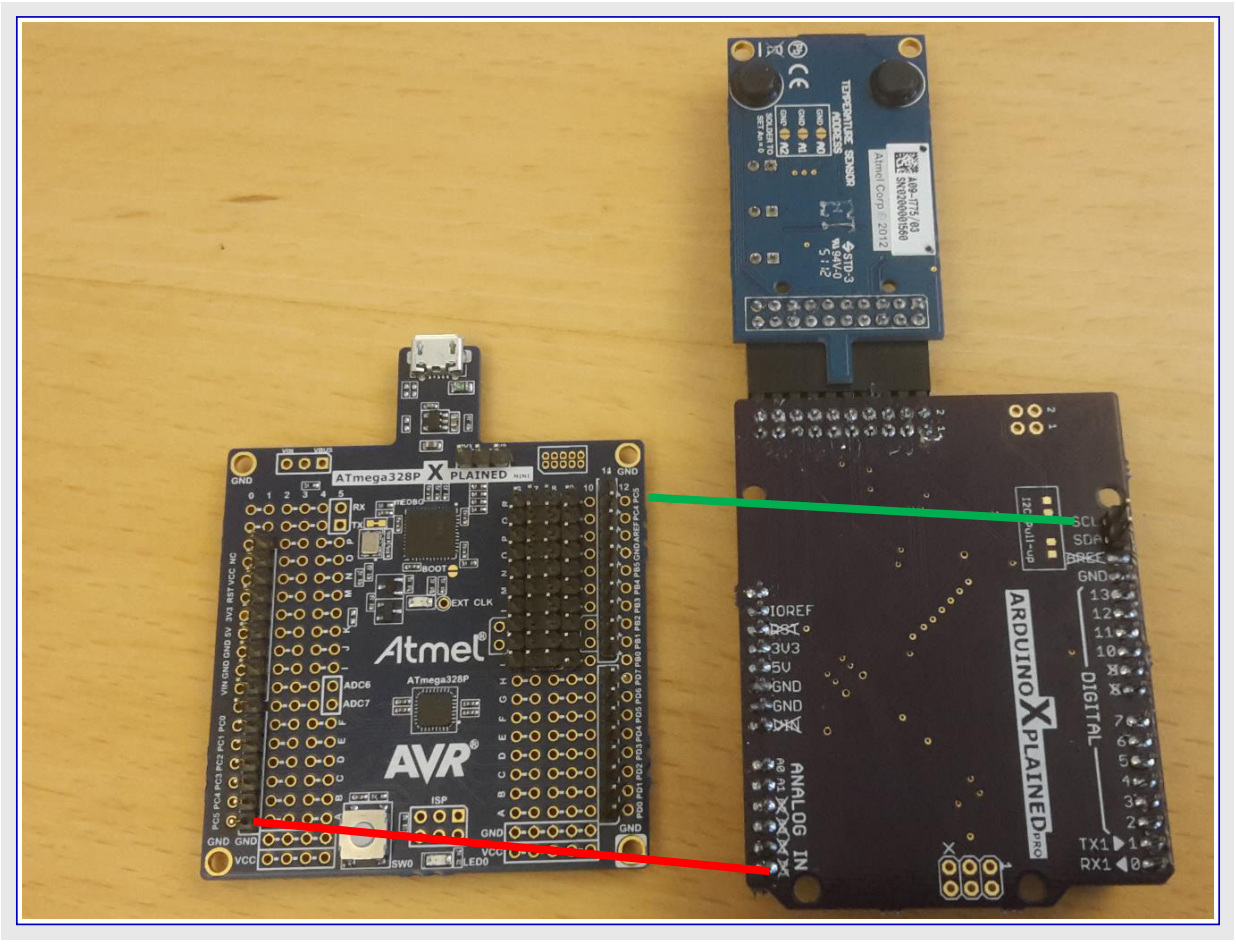
5.4.1. 接続 : IO1 Xplained Pro – Arduino Xplained Pro

下図で示されるようにIO1 Xplained ProをArduino Xplained Proへ接続してください。



5.4.2. 接続 : ATmega328P Xplained Mini – Arduino Xplained Pro

次にATmega328P Xplained MiniをArduino Xplained Pro基板に接続することが必要です。Xplained Mini基板の列1からのPC5をArduino Xplained ProのA5に(赤色接続)とXplained Mini基板の列11からのPC5をArduino Xplained ProのSCLに(緑色接続)行くように接続してください。



右のように接続してください。

5.4.3. 接続 : USBケーブル

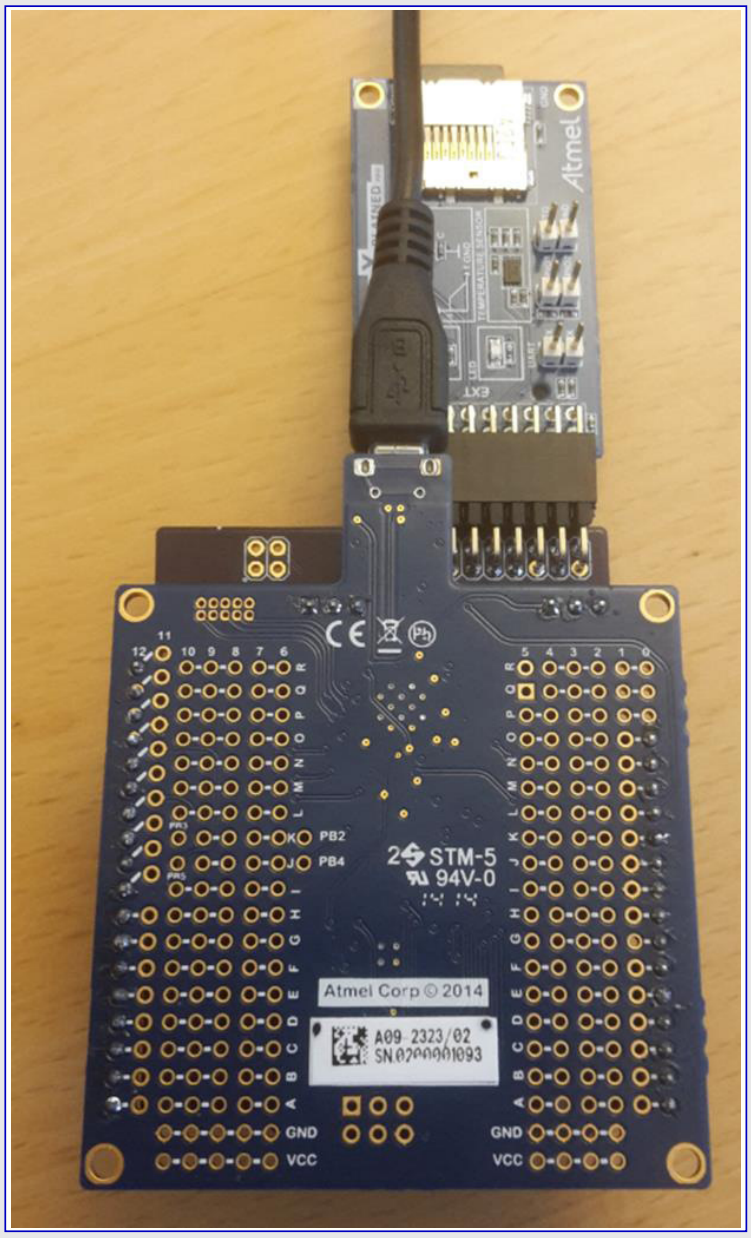
ATmega328P Xplained Mini基板にUSBケーブルを接続し、基板を図5-3.のように接続してください。

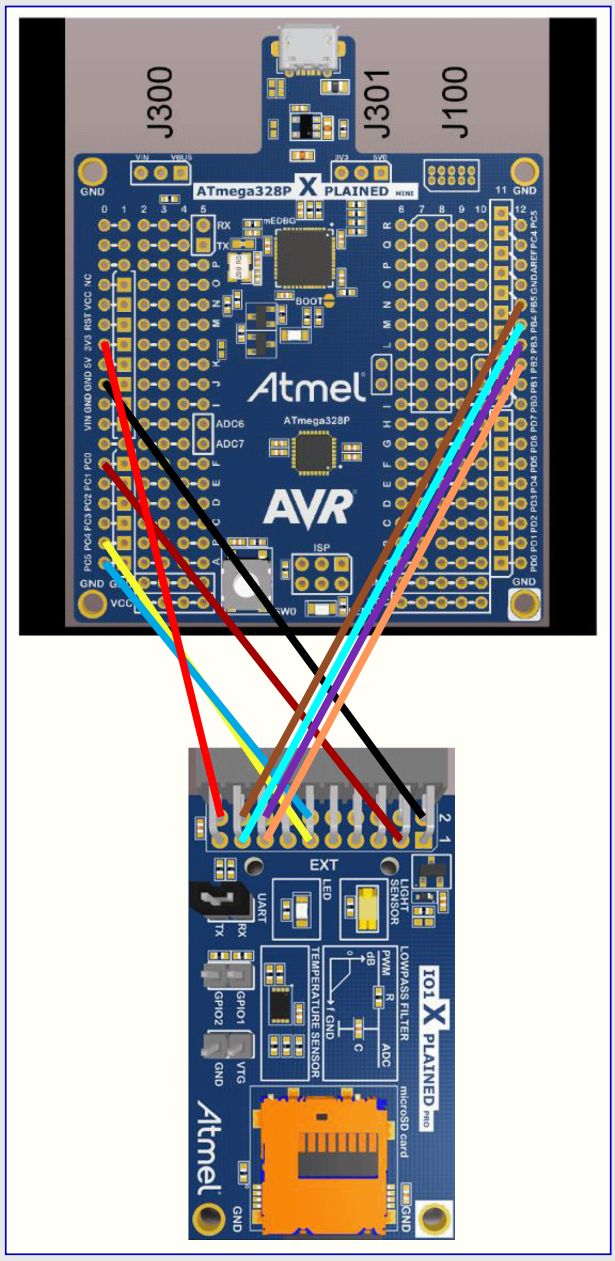
警告 ソケットにSDカードが正しく挿入されているのを確実にしてください。

情報 基板は光感知器が暗がりにないこの方法で置かれます。

情報 代替接続法：使用者は右下の図で示されるようにArduino Xplained Pro基板を使わずに9PのオスーメスコネクタをつかってIO1 Xplained Pro基板を接続することもできます。

図5-3. 課題4に対するハードウェア接続の視覚的表現

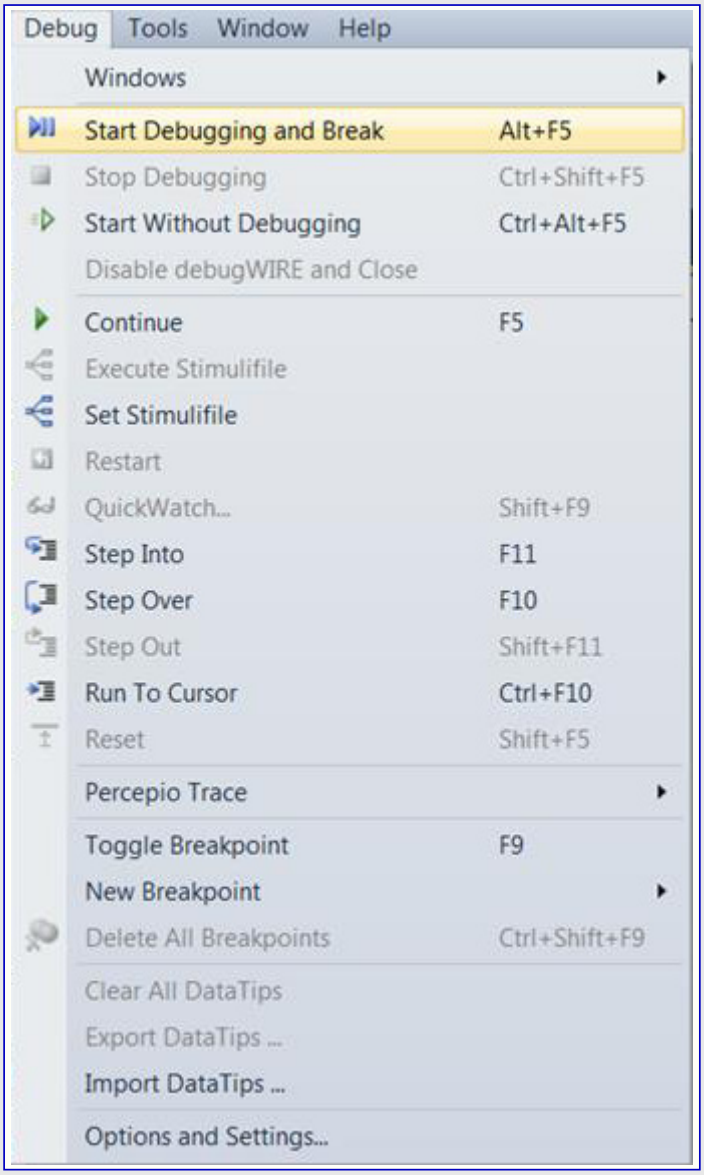




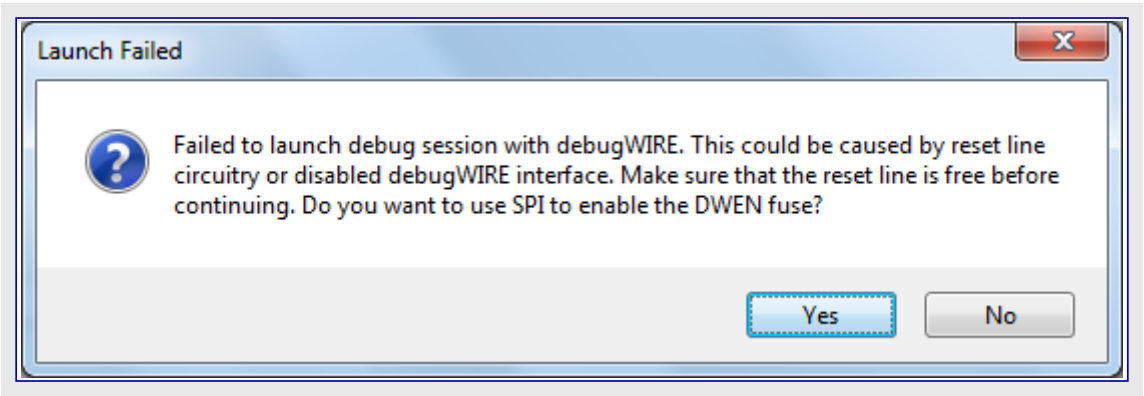
5.5. 応用のﾃﾞﾊﾞｯｸﾞ

次に応用をﾃﾞﾊﾞｯｸﾞします。ｺｰﾄﾞは特定条件(感知器値>500時)で光感知器値と温度感知器値がSDｶｰﾄﾞに格納されるように書かれています。この条件に中断点(ﾌﾞﾚｰｸﾎﾟｲﾝﾄ)を配置して中断点に当たることを調べてください。

実施事項 Debugを選んで'Start debugging and Break'をｸﾘｯｸしてください。

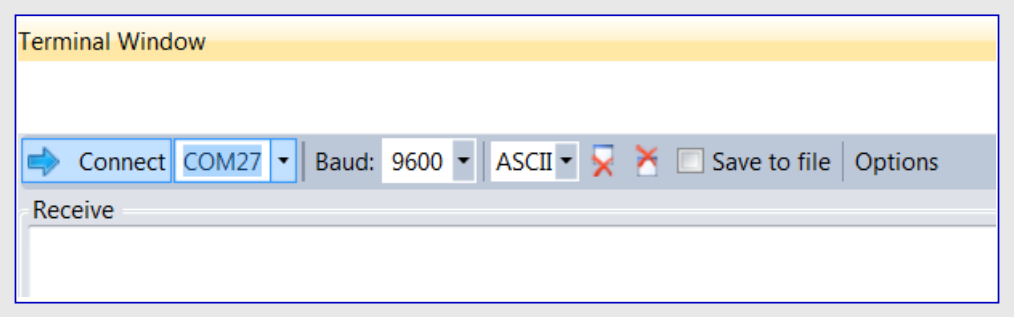


警告 DWENﾋｭｰｽﾞが許可されていない場合に異常ﾒｯｾｰｼﾞが表示されます。下で示されるように'Yes'をｸﾘｯｸしてﾋｭｰｽﾞを設定するのにAtmel StudioはISPを使います。



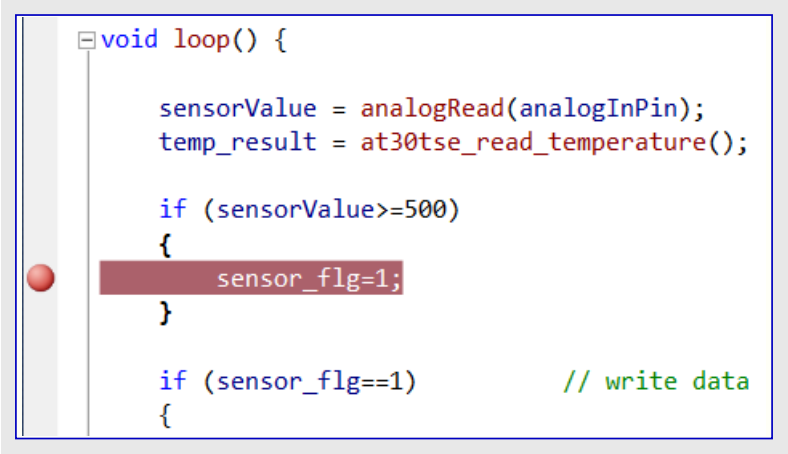
結果 ﾃﾞﾊﾞｯｶﾞが開始され、mainで中断されます。今やﾃﾞﾊﾞｯｸﾞを開始する準備が整いました。

実施事項 Atmel StudioでView⇒Terminal Windowを選んでください。mEDBG COMﾎﾟｰﾄ、Baud:9600を選び、”Connect”を選んでください。



情報 感知器ﾃﾞｰﾀはmEDBG COMﾎﾟｰﾄを通して受信されます。

実施事項 Solution Explorerからsensors.cppを開き、下図で示されるように中断点を配置してください。



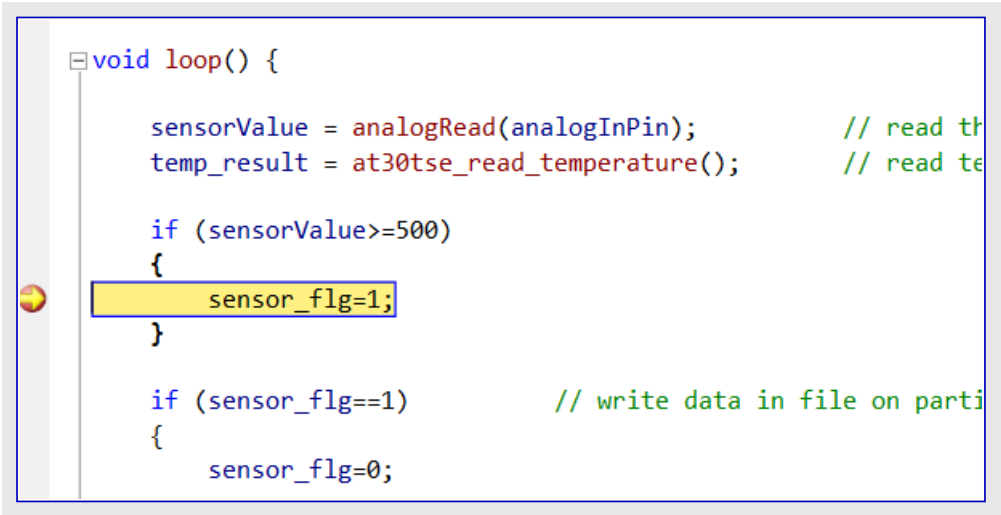
```
void loop() {

    sensorValue = analogRead(analogInPin);
    temp_result = at30tse_read_temperature();

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)          // write data
    {
```

実施事項 次に ▶ 印の選択、またはｷｰﾎﾞｰﾄﾞでのF5ｷｰ押下によってﾃﾞﾊﾞｯｸﾞを始めてください。

結果 端末(Terminal)ｳｨﾝﾄﾞｳに’SD Card initialization done’ﾒｯｾｰｼﾞが現れるでしょう。

情報 光感知器はIO1 Xplained Pro上に存在し、感知器値は暗い時に増加し、明るい時に減少します。

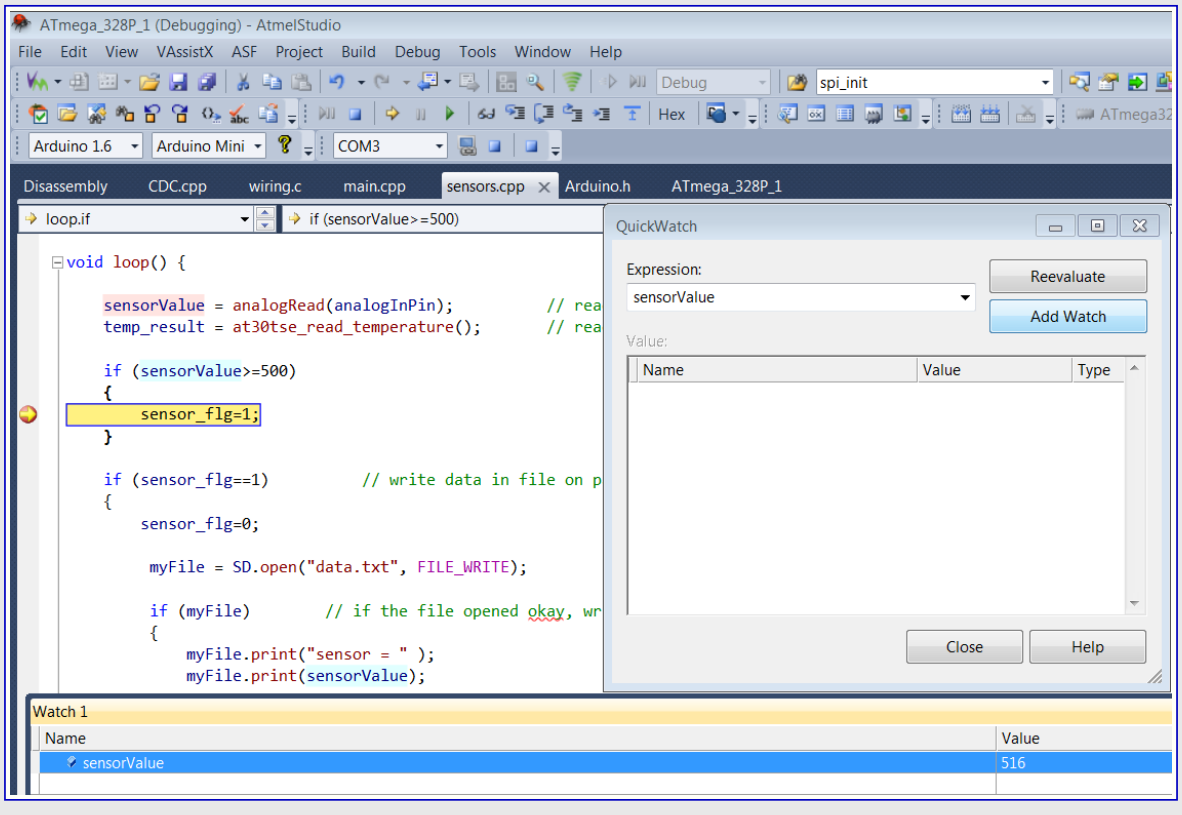
実施事項 感知器値が増すように指で感知器を覆ってください。感知器値が増加されたために条件(sensorValue≧500)が真なら、下で示されるようにﾌﾟﾛｸﾞﾗﾑ実行が中断点(ﾌﾞﾚｰｸﾎﾟｲﾝﾄ)に行き当たるでしょう。



```
void loop() {

    sensorValue = analogRead(analogInPin);          // read th
    temp_result = at30tse_read_temperature();       // read te

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)          // write data in file on parti
    {
        sensor_flg=0;
```

実施事項 ﾃﾞﾊﾞｯｸﾞ中に感知器値を見るために、Debug⇒QuickWatch…を選んでください。式’sensorValue’を入力して’Add Watch’釦を選んでください。

結果 監視1(Watch1)ｳｨﾝﾄﾞｳに’sensorValue’変数の値が表示されるでしょう。

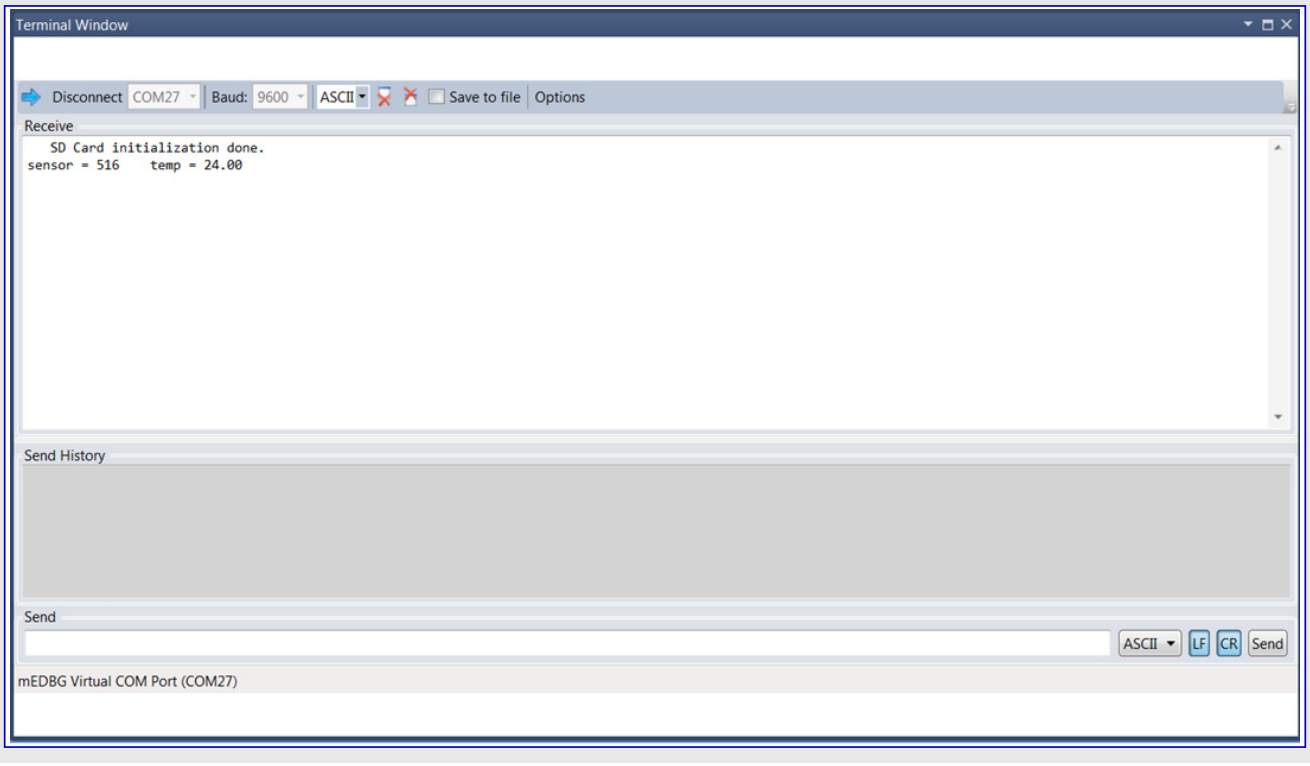


実施事項 ‘Watch1’ｳｨﾝﾄﾞｳと’QuickWatch’ｳｨﾝﾄﾞｳを閉じてください。

実施事項 ｷｰﾎﾞｰﾄﾞでF5を押すことによって実行を続けてください。

結果 感知器の値はSDｶｰﾄﾞに格納されmEDBG COMﾎﾟｰﾄを通して送信もされます。端末(Terminal)ｳｨﾝﾄﾞｳは光感知器値と温度値を示すでしょう。

図5-4. Atmel Studio：端末(Terminal)ｳｨﾝﾄﾞｳ



実施事項　Debug⇒Disable debugWIREを選ぶことによってﾃﾞﾊﾞｯｸﾞ動作を向けだしてください。

警告　ﾃﾞﾊﾞｯｸﾞWIREを禁止することが重要です。

Atmel StudioでATmega328P Xplained MiniでのArduinoﾌﾟﾛｼﾞｪｸﾄを成功裏に追加しました。

追補A. 課題4に対する完全な解決策

sensors.cppファイル

```
#include "arduino.h"
#include <Wire.h>
#include <SD.h>

#define AT30TSE_TEMPERATURE_TWI_ADDR  0x4F
#define AT30TSE_TEMPERATURE_REG       0x00
#define AT30TSE_TEMPERATURE_REG_SIZE  2
#define AT30TSE_NON_VOLATILE_REG      0x00

#define AT30TSE_CONFIG_RES_9_bit      0
#define AT30TSE_CONFIG_RES_10_bit     1
#define AT30TSE_CONFIG_RES_11_bit     2
#define AT30TSE_CONFIG_RES_12_bit     3

uint16_t resolution = AT30TSE_CONFIG_RES_10_bit;
double temp_result;
int sensorValue = 0;          // A/D変換器(ADC) A0から読んだ値

const int analogInPin = A0;   // アナログ入力ピン
const int chipSelect = 10;

// SDユーティリティ ライブラリ関数を使うための変数を準備
File myFile;

void setup()
{
   Serial.begin(9600);        // 直列通信を開く

   if (!SD.begin(chipSelect)) {
       Serial.println("SD Card initialization failed!");
       return;
   }
   Serial.println("SD Card initialization done.");

   SD.remove("data.txt");

   Wire.begin();
}

uint16_t at30tse_read_register(uint8_t reg, uint8_t reg_type, uint8_t reg_size)
{
   uint8_t buffer[2],i=0;
   buffer[0] = reg | reg_type;
   buffer[1] = 0;

   /* AT30TSEでの内部レジスタ ポインタ */
   Wire.beginTransmission(AT30TSE_TEMPERATURE_TWI_ADDR);
   Wire.write(buffer[0]);
   Wire.endTransmission();

   Wire.requestFrom(AT30TSE_TEMPERATURE_TWI_ADDR, reg_size);

   while(Wire.available())
   {
       buffer[i] = Wire.read();
       i++;
   }

   return (buffer[0] << 8) | buffer[1];
}
```

```
#include <Wire.h>
double at30tse_read_temperature()
{
    /* 16ﾋﾞｯﾄ温度ﾚｼﾞｽﾀ読み込み */
    uint16_t data = at30tse_read_register(AT30TSE_TEMPERATURE_REG,  AT30TSE_NON_VOLATILE_REG,
                                          AT30TSE_TEMPERATURE_REG_SIZE);

    double temperature = 0;
    int8_t sign = 1;

    /* 負検査と符号ﾋﾞｯﾄ解除 */
    if (data & (1 << 15)){
        sign *= -1;
        data &= ~(1 << 15);
    }

    /* 温度に変換 */
    switch (resolution){
        case AT30TSE_CONFIG_RES_9_bit:
            data = (data >> 7);
            temperature = data * sign * 0.5;
            break;
        case AT30TSE_CONFIG_RES_10_bit:
            data = (data >> 6);
            temperature = data * sign * 0.25;
            break;
        case AT30TSE_CONFIG_RES_11_bit:
            data = (data >> 5);
            temperature = data * sign * 0.125;
            break;
        case AT30TSE_CONFIG_RES_12_bit:
            data = (data >> 4);
            temperature = data * sign * 0.0625;
            break;
        default:
            break;
    }
    return temperature;
}
```

```
bool sensor_flg=0;

void loop() {
    sensorValue = analogRead(analogInPin);          // A/D変換器(ADC)読み込み
    temp_result = at30tse_read_temperature();       // 温度読み込み

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)          // 特定感知器値をファイルに書き込み
    {
        sensor_flg=0;

        myFile = SD.open("data.txt", FILE_WRITE);

        if (myFile)             // ファイルが開いていれば、それに書き込み
        {
            myFile.print("sensor = " );
            myFile.print(sensorValue);

            myFile.print("    temp = ");
            myFile.print(temp_result);
            myFile.print("¥n");

            myFile.close();    // ファイルを閉じる
        }
        else
        {
            // ファイルが開かなければ異常を表示
            Serial.println("error opening data.txt");
        }

        // 読むために再びファイルを開く
        myFile = SD.open("data.txt");
        if (myFile)
        {
            // 何も無くなるまでファイルから読み込み
            while (myFile.available()) {
                Serial.write(myFile.read());
            }
            // ファイルを閉じる
            myFile.close();
        } else {
            // ファイルが開かなければ異常を表示
            Serial.println("error opening data.txt");
        }
    }
    delay(500);
}
```

6. 結び

この実施訓練で以下の作業が達成されました。

- Atmel StudioへのArduino風ｺｰﾄﾞ書き形式支援を追加
- Arduinoﾌﾟﾛｼﾞｪｸﾄへﾃﾞﾊﾞｯｸﾞ能力を追加
- どんなｺｰﾄﾞ変更もなしにAtmel Xplained MiniとAtmel StudioでのArduinoﾌﾟﾛｼﾞｪｸﾄを置く
- もっと込み入った複雑なﾌﾟﾛｸﾞﾗﾑを書くことができる能力を追加

7. 改訂履歴

文書改訂	日付	注釈
42439A	2015年8月	初版文書公開

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA TEL:(+1)(408) 441-0311 FAX: (+1)(408) 436-4200 | www.atmel.com

---

本応用記述はAtmelのAN12077応用記述(Rev.42439A-08/2015)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はリンクとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。