LD Cv2!のROM化開発手順(第5版)

■はじめに

この文書は、ラダー図開発ツールLD Cv2!を使って生成したプログラムを、実機へ組み込む事例について説明しています。

■その他

・個別のマイコンの開発ツールの操作方法については、ベンダーのドキュメント、市販の参考書などをご覧ください。

・LD Cv2!のROM化開発の考え方については、LD Cv2!に同梱の取扱説明書(特に第2章)をご覧ください。

・本文書で使用する製品名は、各社、各組織の登録商標、または商標です。

第1章 mbed NXP LPC1768
(ARM社のCPUコアを採用したNXP社マイコンへの適用例)

■概要

NXP LPC1768はARM社のCPUコアを採用したNXP社のマイコンで、mbed NXP LPC1768は このマイコンと周辺回路から成るマイコンボードです。

mbedではプログラム開発作業はウェブブラウザ上でC++を使ってコーディングを行い、コンパイルするとROM書込データは自動的にパソコンにダウンロードされます。パソコンとUSB接続されたマイコンはパソコン画面上ではMBEDという名前のディスクドライブに見えるので、このファイルをMBEDにコピーするとマイコンへの書込が完了します。

■ハードウェア、制御仕様

入出力は、スイッチ1点、LED1点のみです。

図1.1の通り、mbed LPC1768ボードのポートp5に外付のスイッチSW1、抵抗２個を配線します。330Ωの抵抗は短絡でもかまいません。

LEDはボード上のLED1を使用します。

ボード上のUSB端子はパソコンのUSB端子にケーブルで接続し、電源供給とプログラム書込に使用します。

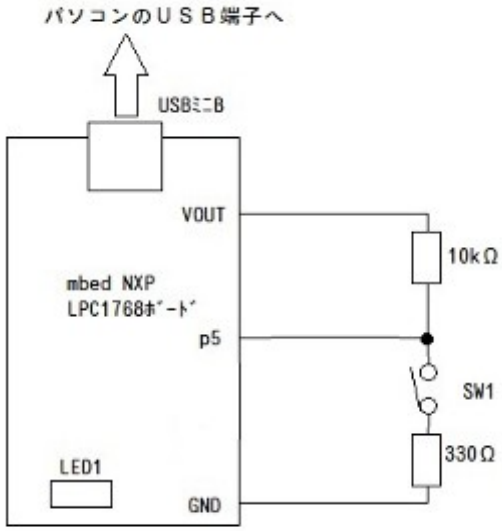


図１.1 マイコンボードの配線

このハードウェアで「スイッチSW1の開閉状態により、LED1の点滅速度を切り替える」制御をおこなうこととします。

■開発作業

では、実際に開発をおこなってみます。

【手順1】ラダー図の作成

LD Cv2!を使ってラダー図を作成します。

LD Cv2!の仮想PLCには、１０００スキャンまたは１００００スキャン実行毎にON/OFFが反転する特殊なメモリS13とS14が用意されています。

ラダー図上でSW1のスイッチ信号のA接点に接点S14を、B接点に接点S13を直列接続してやれば、必ずどちらか一方が点滅信号源になり、これらを並列接続してマイコンボード上のLED1に出力(コイルを接続)してやれば目的の動作を行います。

ただし、 LD Cv2!の仮想PLCはマイコンのI/Oポート(SW1とLED1)を直接操作できないため、とりあえずここでは

適当に、SW1=M000、LED1=M010

と決めて、ラダー図を作成します。解りやすいようにI/Oコメントも定義しました。(図1.2)

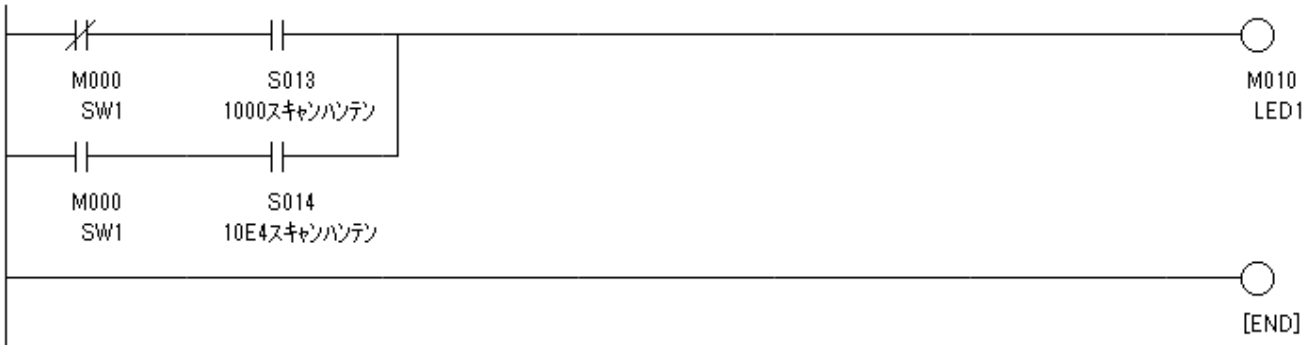


図1.2 作成したラダー図

【手順2】C言語への変換

LD Cv2!のデフォルトの変換条件はC言語ですが、RAM容量が小さくて済む「C言語(Smallモデル)」に変更しておきます。(LPC1768はRAM容量が大きいので「C言語」のままでもかまいません)

次にLD Cv2!で[変換：変換実行]メニューを実行し、変換結果をファイル名 plc1.cとして保存します。

これで、図1.2のロジックを含むC言語プログラム(関数plc)が作成されました。

【手順3】マイコン開発環境上での作業

以後は、mbedの開発環境を使っての作業となります。ウェブブラウザでhttps://os.mbed.com/

にアクセスし、右上の「コンパイラ」(英語表示の場合「Compiler」)を押下すると、ウェブブラウザのウィンドウにmbedのIDEが立ち上げります。

IDE上で、左上の[新規：新しいプログラム...]メニューを実行すると「新しいプログラムの作成」ウィンドウが表示されます。(図1.3)

【注意】 ここでは、ユーザアカウントの登録が済んでいる、ターゲットのマイコン(mbed NXP LPC1768)の登録が済んでいるものとします。

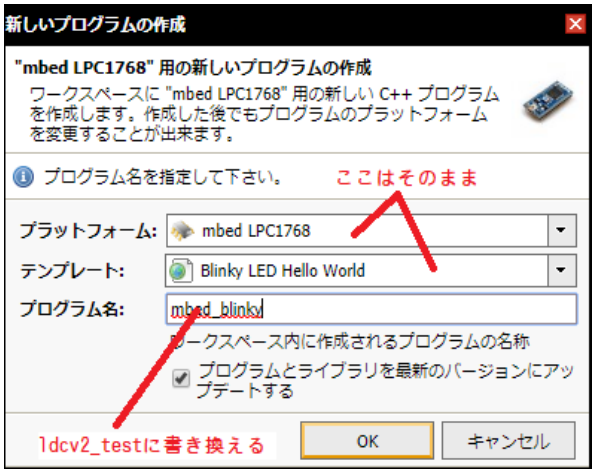


図1.3 「新しいプログラムの作成」ウィンドウの書き換え

プログラム名のみ「ldcv2_test」に書き換えて、OKボタンを押します。

この状態では、ldcv2_test にはテンプレートのBlinky LED Hello Worldのコード(サンプルのLチカプログラム)が入っています。(図1.4)

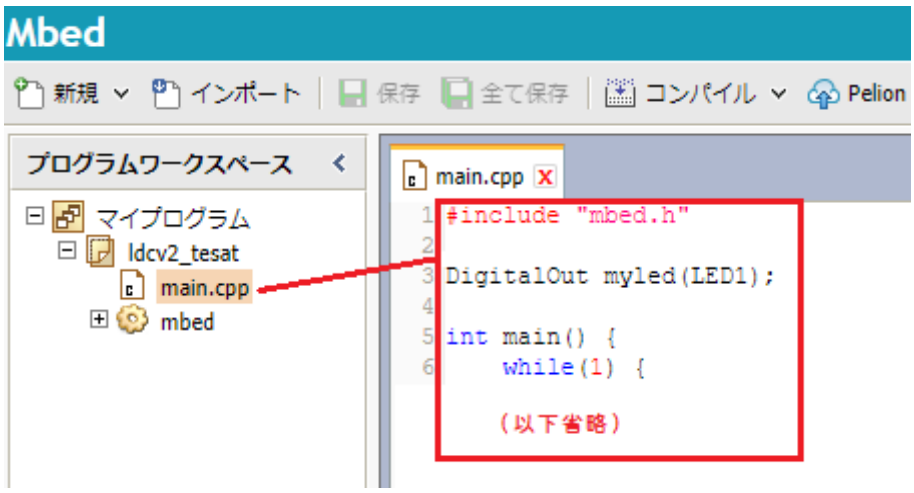


図1.4　main.cppのプログラム(変更前)

このmain.cppのプログラム(赤枠内)を、図1.5のフローチャートのプログラム(リスト1.1)に差し替えます。

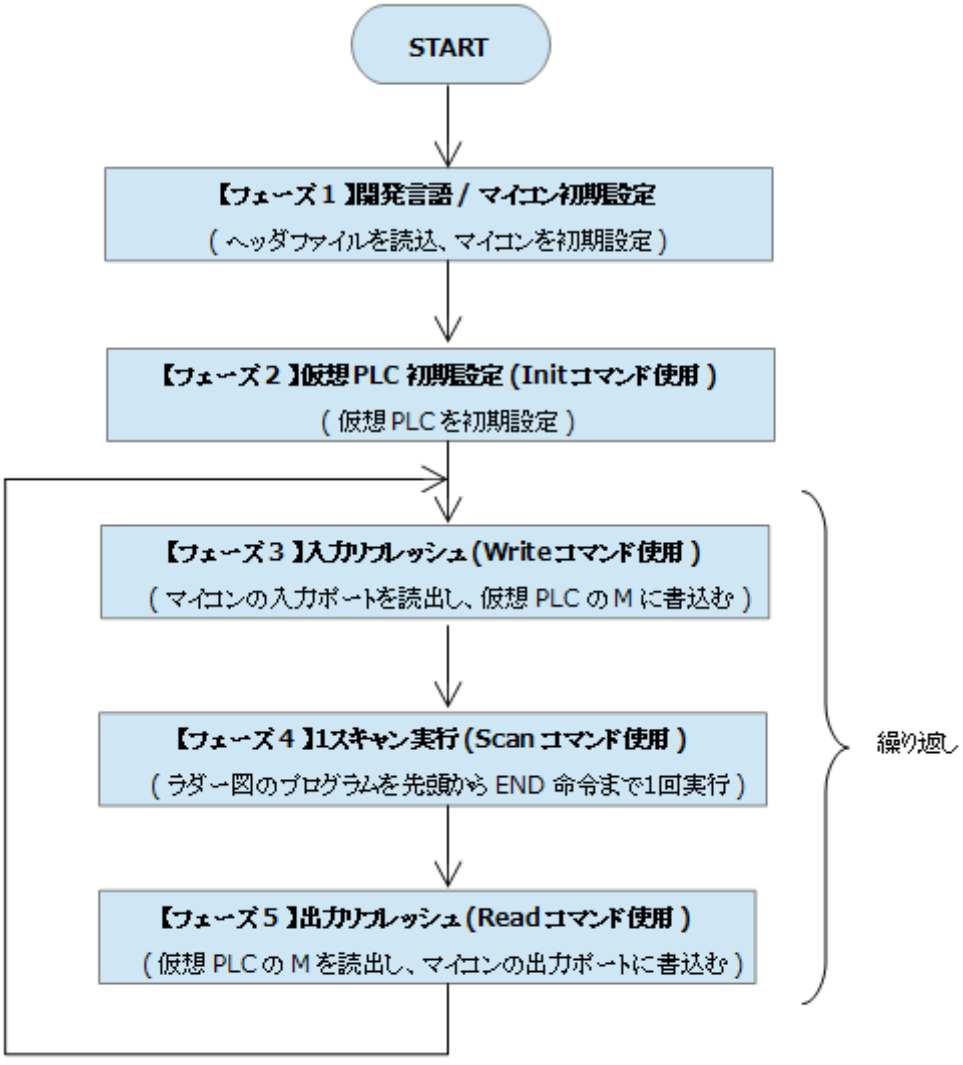


図1.5 仮想PLCの制御ロジック

```
//----------------------------------------
//【フェーズ1】開発言語/マイコン初期設定
//(ヘッダファイルを読込、マイコンを初期設定)
//----------------------------------------
#include "mbed.h"
#include "plc1.c"        //LDCv2!が生成したCプログラムをインクルード

DigitalIn  sw1(p5);      //ポート(p5)を入力に設定
DigitalOut led1(LED1);   //ポート(LED1)を出力に設定

int main()
{
  //----------------------------------------------
  //【フェーズ2】仮想PLC初期設定(Initコマンド使用)
  //(仮想PLCを初期設定)
  //----------------------------------------------
  plc('I',0,0);

  while(1){
    //-------------------------------------------------
    //【フェーズ3】入力リフレッシュ(Writeコマンド使用)
    //(マイコンの入力ポートを読出し、仮想PLCのMに書込む)
    //-------------------------------------------------
    plc('W',0x000,sw1);          //SW1→M000

    //-------------------------------------------------------
    //【フェーズ4】１スキャン実行(Scanコマンド使用)
    //(ラダー図のプログラムを先頭からEND命令まで１回実行)
    //-------------------------------------------------------
    plc('S',0,0);                //１スキャン実行

    //-------------------------------------------------
    //【フェーズ5】出力リフレッシュ(Readコマンド使用)
    //(仮想PLCのMを読出し、マイコンの出力ポートに書込む)
    //-------------------------------------------------
    led1=plc('R',0x010,0);       //M010→LED1
  }
}
```

リスト1.1 変更後のプログラム(main.cpp)

main.cppのコードの最初で、LD Cv2!が生成した関数plcのファイル(plc1.c)をインクルードしているので、このファイルもldcv2_testに加えます。(図1.6)

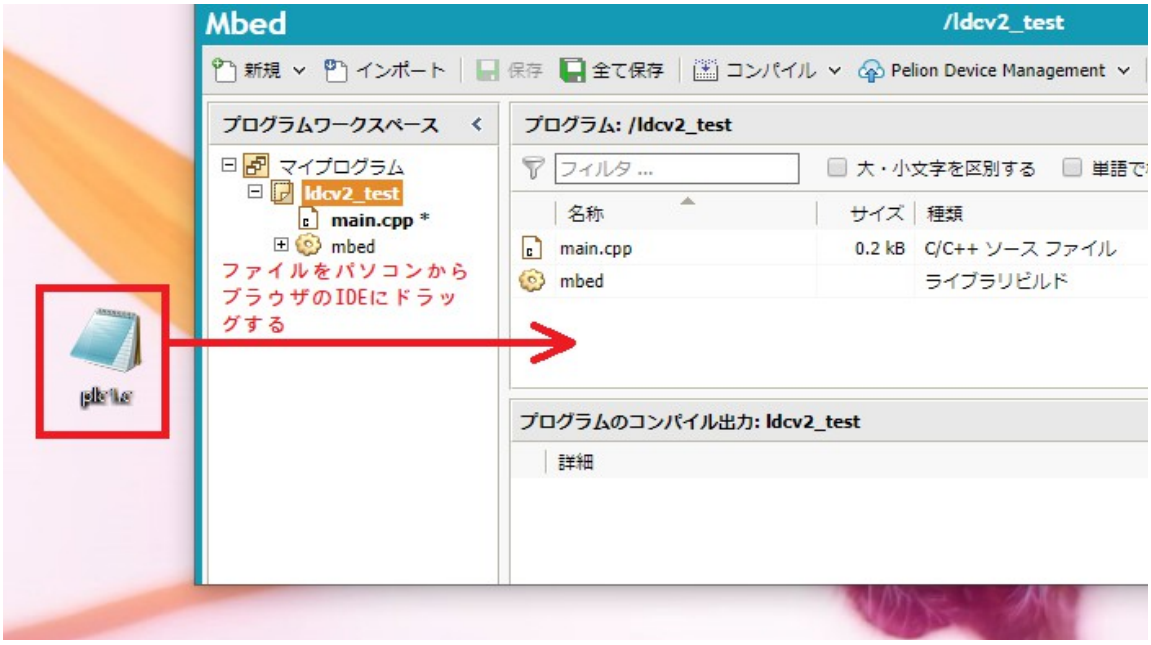


図1.6 LD Cv2!が生成した関数plcのファイルを加える

これで、コンパイルの準備ができました。IDEの上側のあるコンパイルメニューをクリックすると、コンパイル結果がIDEの下側に表示されます。(図1.7)

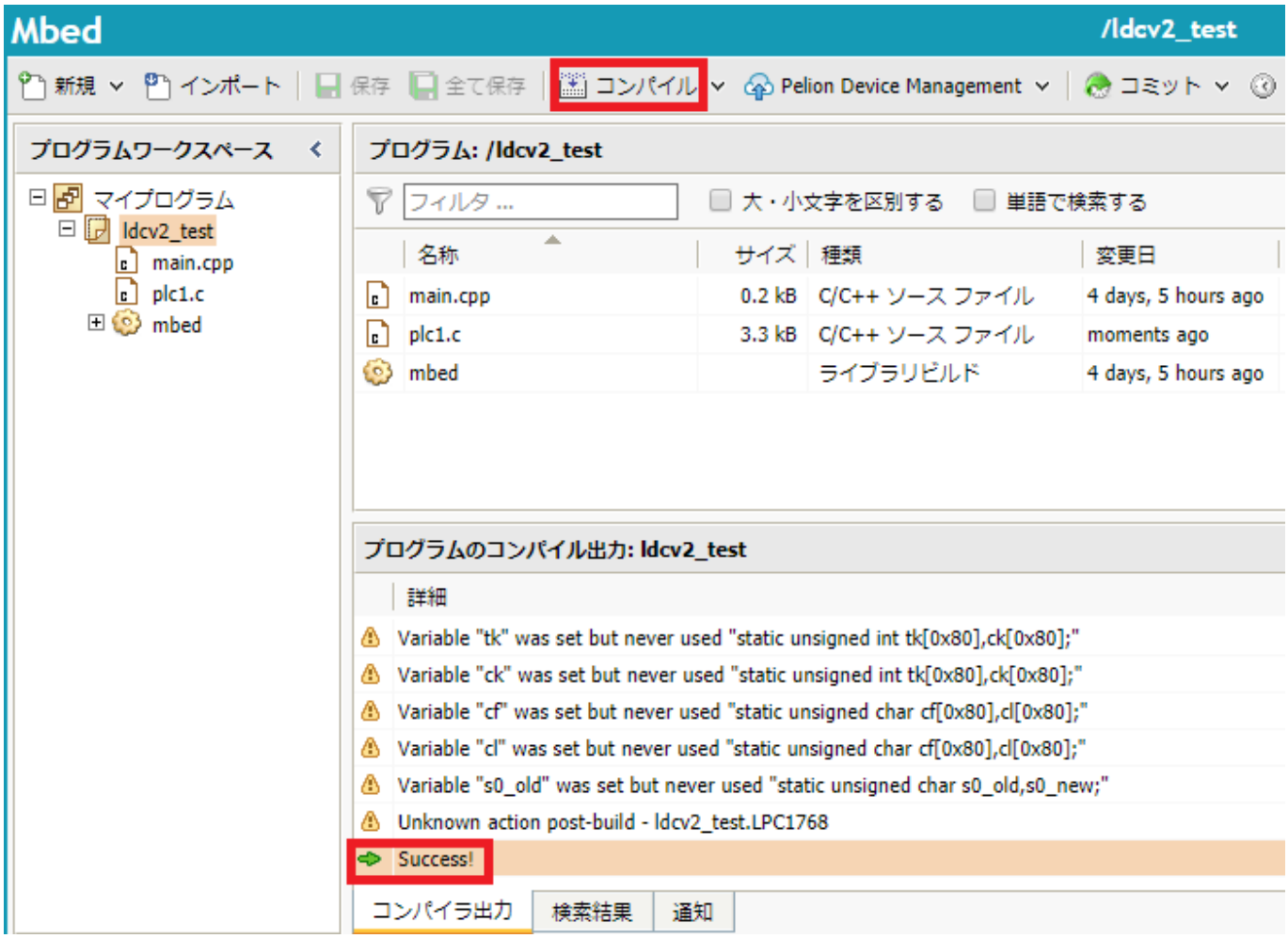


図1.7 コンパイルを実行し、結果を表示

コンパイルが成功すれば、この時点でブラウザのダウンロード先フォルダに

ldcv2_test.LPC1768.bin

というマイコンへの書き込みデータファイル(.bin)がダウンロードされています。(２回以上コンパイルすると、ファイル名末尾に連番が付きます。)

【手順4】マイコンへの書込み、動作

パソコンとマイコンボードをUSB接続すると、パソコン画面上でマイコンはMBEDという名前のディスクドライブに見えます。

書き込みデータファイル(.bin)をMBEDにコピー(ドラッグ&ドロップ)するとマイコンへの書込が完了します。(図1.8)

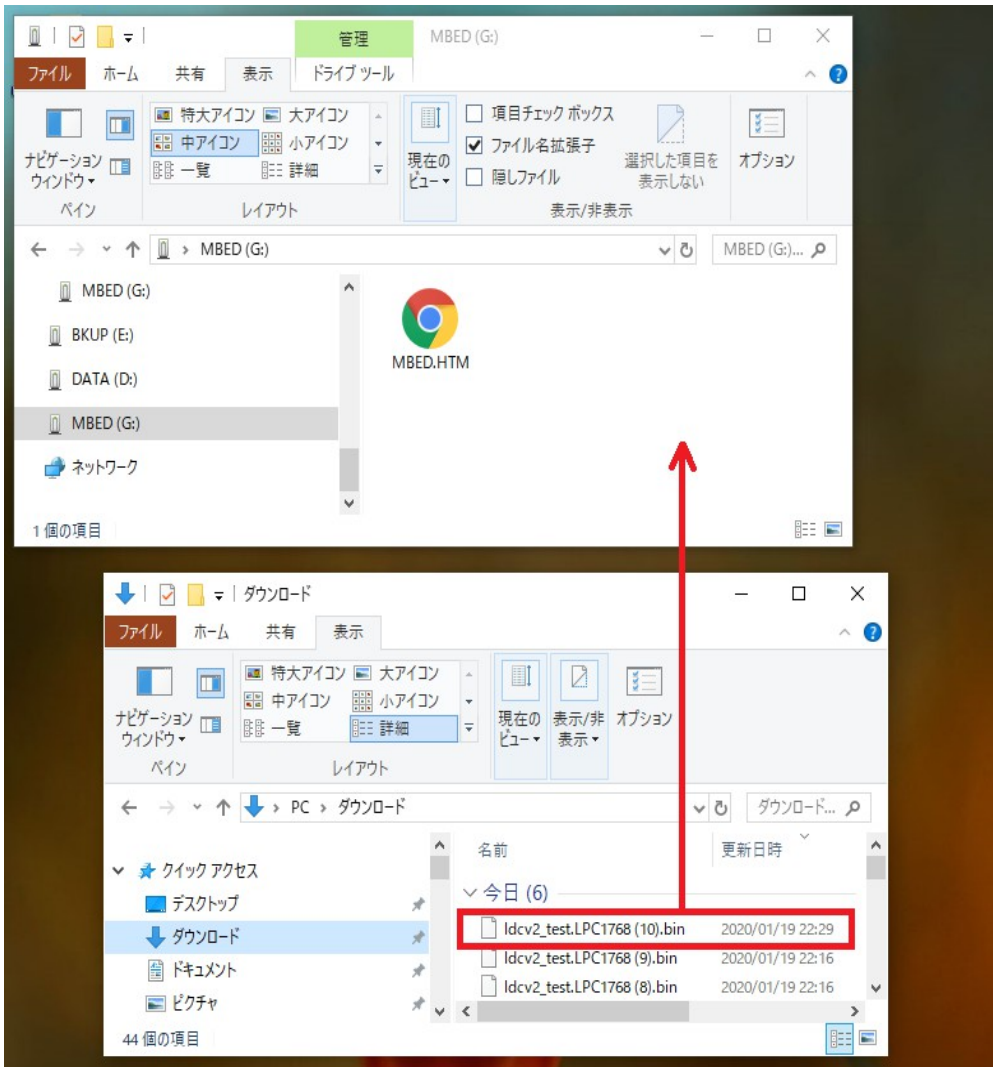


図1.8　書き込みデータファイルをMBEDにコピー

この時点でボード上のリセットスイッチを押すとＬＥＤが点滅し、SW1スイッチの操作で点滅速度が切り替わります。(スイッチが閉の場合、点滅速度が速すぎて連続点灯に見えますが、ボードを振ると点滅していることが解ります。)

プログラムはマイコンのROM(フラッシュメモリ)に書き込まれているので、パソコンと切り離しても電源さえ供給していれば動作します。

第2章 Arduino
(タイマ基準クロック(S0)を外部から駆動する方法)

■概要

LD Cv2!の仮想PLCのタイマT0～T7Fは、特殊メモリS0(タイマ基準クロック)が変化する毎に経過値が+1します。

ラダー図プログラムでタイマを1点でも使用する場合は、S0を定期的に変化するクロック信号で駆動させる必要があります。

簡易的には図2.1のようにスキャンタイムの倍数(S10～S14)を利用することができますが、この方法はタイマ時間がスキャンタイムによって変動することになります。

図2.1 タイマ基準クロックとして スキャンタイムを使用した例

変動しないタイマ時間を得る手っ取り早い方法は、S0にマイコンのシステムクロック由来の信号を与えることです。

今回はS0に5Hzのクロック信号(0.1秒毎に変化する信号)を与えて、タイマT0～T7Fが起動すると0.1秒毎に経過値が＋１増加するようにしてみます。

ハードウェアとしてArduino(Duemilanove)を使用しましたが、他のマイコンボードでも考え方は同じです。

■ハードウェア、制御仕様

Arduinoのポートにスイッチ(SW1)とLED(LED1)を配線します。(接続ポートと極性については表2.2の通り)

このハードウェアで、スイッチを押すと5秒後に0.5秒間だけEDが点灯する(途中でスイッチを離したときは点灯しない)制御を行うこととします。

■タイマ基準クロックの実現方法

タイマ基準クロックには、Arduino がサポートしているmillis関数を使用することとします。

この関数は1ms毎に値が+1する32ビットのフリーランニングカウンタ(FRC)なので、１スキャン実行毎に

今回スキャンのFRC値ー前回スキャンのFRC値

を読出・計算すれば経過時間が得られます。この経過時間を変数に累積してゆき、変数の値が100を超える毎に別の変数(5Hzクロック信号用)をビット反転させればこれが5Hz信号となります。

このロジックをArduino言語(≒C言語)で表現すると、一例ですがリスト2.1のようになります。

```
//(5Hz信号を生成し、仮想PLCのM70に書込む)
Frc_Old = Frc_New;                  //前回スキャンのFRC値を保存
Frc_New = millis();                 //今回スキャンのFRC値を読込
Frc_Sum +=  Frc_New-Frc_Old;        //経過時間を累積
if(Frc_Sum>=100){                   //累積値が100msを超えたら
  Clk_5Hz = Clk_5Hz ^ B00000001;    //5Hz信号のb0を反転
  Frc_Sum = Frc_Sum % 100 ;         //累積値を100未満に正規化
}
```

リスト2.1 5Hzクロック信号生成

この5Hz信号でラダー図上のS0を駆動すればよいのですが、関数plcのWriteコマンドはラダー図上のS0を直接操作できません。そこでWriteコマンドではM70(適当に決めたI/O番号)を操作し、ラダー図上でM70の接点でコイルS0を駆動することにします。(図2.2)

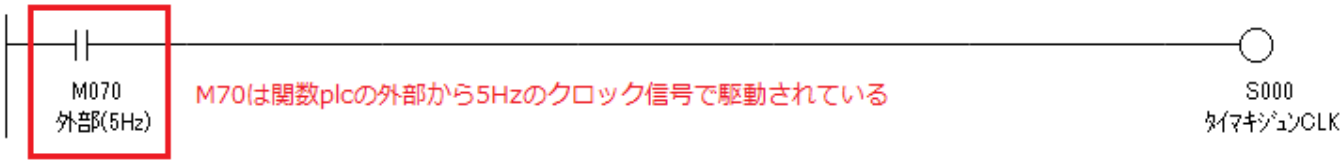


図2. 2 接点M70でコイルS0を駆動

■開発作業

では、実際に開発をおこなってみます。

【手順1】ラダー図の作成

ラダー図上で使うI/O番号の割付は表2.1 の通りに決めました。

I/O番号はS0以外は適当に決めた番号で、別の番号を使ってもかまいません。

表2.1 I/O番号の割付

I/O番号(ラダー図)	ポート番号	内容
M0	P12(入力)	スイッチ(SW1)を接続 (スイッチを押すと入力がONになる)
M10	P13(出力)	LED(LED1)を接続 (出力をONするとLEDが点灯する)
M70	-	ここに、Writeコマンドで５Hzのクロック信号を与える
T70	-	ラダー図中で、5秒タイマとして使用する
T7F	-	ラダー図中で、0.5秒タイマとして使用する
S0	-	LD Cv2!の仕様で決められたタイマ基準クロック (番号は変更不可)

このI/O番号でラダー図を作成します。解りやすいようにI/Oコメントも定義しました。(図2.3)

タイマの設定値はT70が50×0.1秒= ５秒、T7Fが5×0.1秒=0.5秒となります。

図2.3 作成したラダー図

【手順2】C言語への変換

LD Cv2!のデフォルトの変換条件はC言語ですが、ArduinoはRAM容量が小さいため「C言語(Smallモデル)」に変更しておきます。

次にLD Cv2!で[変換：変換実行]メニューを実行し、変換結果をファイル名 plc2.cとして保存します。

これで、図2.3のロジックを含むＣ言語プログラム(関数plc)が作成されました。

【手順3】マイコン開発環境上での作業

以後は、Arduinoの開発環境を使っての作業となります。

まず、Arduino IDEを立ち上げます。

リスト２.2で、黒字がArduino IDEが自動的に生成したフレームで、 フェーズ１～２のプログラムはsetup() 関数の中(またはプログラム先頭)に記述し、フェーズ３～５のプログラムはloop() 関数の中に記述します。各フェーズの意味については、第1章の図1.5を参照ください。

リストが長いのでリスト2.2全体をコピペすることをお勧めします。

プログラムを入力したら、スケッチを保存し、手順２で作成したファイルplc2.cもスケッチと同じフォルダに保存します。

```
//-----------------------------------------
//【フェーズ1(1)】開発言語/マイコン初期設定
//(ヘッダファイルを読込、マイコンを初期設定)
//-----------------------------------------

//LDCv2!が生成したCプログラムをインクルード
#include "plc2.c"

//ピンに名前を付与
#define p12 12
#define p13 13

//変数定義
unsigned long Frc_New;
unsigned long Frc_Old;
unsigned long Frc_Sum;
byte Clk_5Hz;

void setup() {
  //-----------------------------------------
  //【フェーズ1(2)】開発言語/マイコン初期設定
  //(マイコンを初期設定)
  //-----------------------------------------
  pinMode(p12, INPUT);    //p12(SW1)を入力に設定
  pinMode(p13, OUTPUT);   //p13(LED1)を出力に設定
  Frc_New = millis();
  Frc_Old = millis();
  Frc_Sum =0;
  Clk_5Hz = B00000000;

  //-----------------------------------------------
  //【フェーズ2】仮想PLC初期設定(Initコマンド使用)
  //(仮想PLCを初期設定)
  //-----------------------------------------------
  plc('I',0,0);
}

void loop() {
  //-------------------------------------------------
  //【フェーズ3】入力リフレッシュ(Writeコマンド使用)
  //(マイコンの入力ポートを読出し、仮想PLCのMに書込む)
  //-------------------------------------------------
  plc('W', 0x00, digitalRead(p12));     //p12(SW1)→M000

  //(5Hz信号を生成し、仮想PLCのM70に書込む)
  Frc_Old = Frc_New;                    //前回スキャンのFRC値を保存
  Frc_New = millis();                   //今回スキャンのFRC値を読込
  Frc_Sum +=  Frc_New-Frc_Old;          //経過時間を累積
  if(Frc_Sum>=100){                     //累積値が100msを超えたら
    Clk_5Hz = Clk_5Hz ^ B00000001;      //5Hz信号のb0を反転
    Frc_Sum = Frc_Sum % 100 ;           //累積値を100未満に正規化
  }
  plc('W', 0x70,Clk_5Hz);               //5Hz信号→M070

  //-------------------------------------------------
  //【フェーズ4】１スキャン実行(Scanコマンド使用)
  //(ラダー図のプログラムを先頭からEND命令まで１回実行)
  //-------------------------------------------------
  plc('S',0,0);                         //１スキャン実行

  //-------------------------------------------------
  //【フェーズ5】出力リフレッシュ(Readコマンド使用)
  //(仮想PLCのMを読出し、マイコンの出力ポートに書込む)
  //-------------------------------------------------
digitalWrite(p13, plc('R', 0x10, 0));   //M010→p13(LED1)

}
```

リスト2.2 作成したArrduiinoプログラム

【手順4】マイコンボードに転送、実行

[スケッチ:マイコンボードに書き込む]で、ROM書込が行われます。

エラーが無ければ、この時点でスイッチを押すと5秒後に約0.5秒間LEDが点灯します。

制御プログラムは、マイコンボードにROMに書き込まれているのでパソコンと切り離しても電源さえ供給していれば動作します。

第3章 Raspberry Pi Zero
(LD CV2!の命令語プログラムをインタプリタで実行)

■概要

PLCのプログラムをラダー図で作成する場合、通常、プログラマはPLCをラダー図の通り動作する装置と見ていて、命令語(PLCにおける機械語みたいなもの)で動作していることを意識していません。

これは、一般のコンピュータでプログラマがコンパイル後の機械語プログラムを意識しないのと同じです。

LD Cv2!は、カーソル位置の回路がどのような命令語に変換されているかを見ることができます、(図3.1)

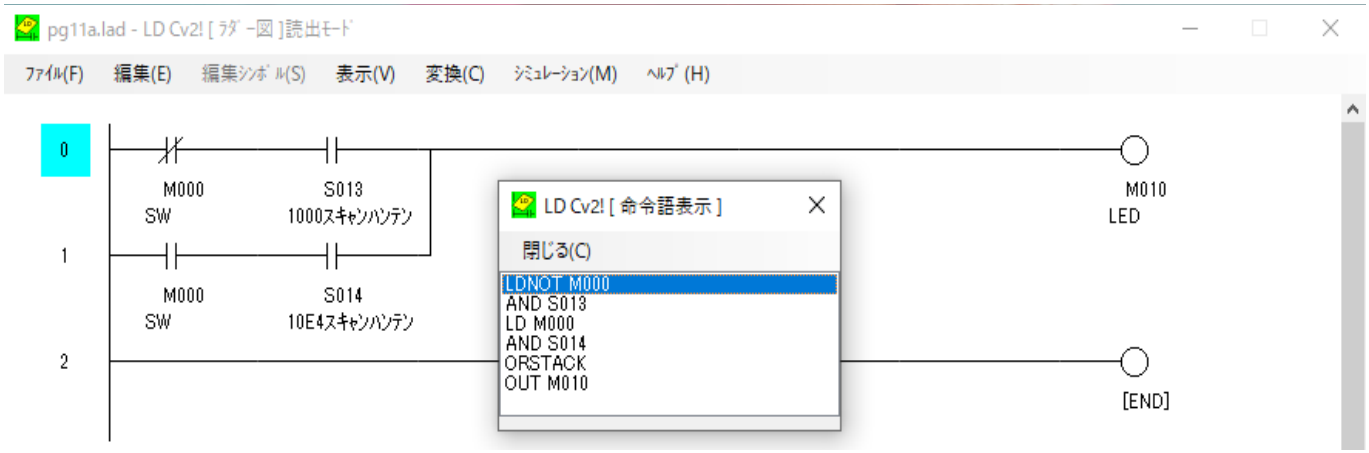


図3. 1 カーソル位置の回路を命令語表示

また、ラダー図全体の命令語プログラム(テキストファイル)を出力することも可能です。ただし、この命令語プログラムは、仮想PLCの命令語であるため実行する実機(PLC)は存在しません。

そこで、Raspberry Pi Zeroマイコンボード上で仮想PLCの命令語インタプリタを走らせることにしました。

具体的には、Raspberry Pi ZeroのLinux上でThonny(IDE) を起動し、ここでPythonで記述した命令語インタプリタを走らせます。インタプリタは、命令語プログラム(ファイル **plc.il**)を読み込み後、運転を開始し、プログラムに従ってボード上のGPIOを制御します。

なお、今回の実施例は「遊び・実験」に近いもので、処理速度や異常処理などの面で産業機器であるPLCとはレベルが違うこと、インタプリタのコードの検証が十分でないことをご承知おきください。

■ハードウェア、制御仕様

入出力は、スイッチ1点、LED1点のみです。

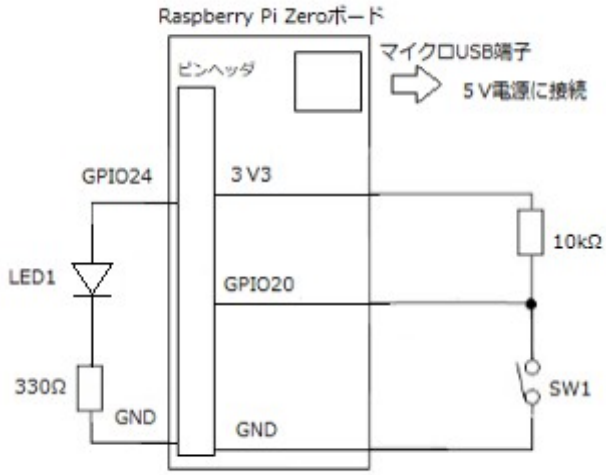


図3. 2 マイコンボードの配線

図3.2の通り、マイコンボードのGPIO20にはスイッチ(SW1)、GPIO24にはLED(LED1)を接続します。マイクロUSB端子は電源用です。
ボードとWindowsパソコンとはWiFi経由でリモート接続しました。(Raspberry Pi(サーバー)側はVNCを有効にし、Windowsパソコン(クライアント)側はVNC Viewerをインストールします。ここら辺の設定は市販の参考書などをご覧ください。)
このハードウェアで「スイッチSW1の開閉状態により、LED1の点滅速度を切り替える」制御をおこなうこととします。

■開発作業
では、実際に開発をおこなってみます。

【手順1】ラダー図の作成
LD Cv2!を使ってラダー図を作成します。
LD Cv2!の仮想PLCには、１００スキャンまたは１０００スキャン実行毎にON/OFFが反転する特殊なメモリS12とS13が用意されています。
ラダー図上でSW1のスイッチ信号のA接点に接点S13を、B接点に接点S12を直列接続してやれば、必ずどちらか一方が点滅信号源になり、これらを並列接続してマイコンボード上のLED1に出力(コイルを接続)してやれば目的の動作を行います。
今回ボード上で動作させる命令語インタプリタは、入出力を表3.1の通り対応付けてあります。

表3.1 マイコンボードのGPIOとMの対応

マイコンボードのGPIO	ラダー図上のメモリM
GPIO20 (入力ポート)	M20
GPIO21 (入力ポート)	M21
GPIO22 (入力ポート)	M22
GPIO23 (入力ポート)	M23
GPIO24 (出力ポート)	M24
GPIO25 (出力ポート)	M25
GPIO26 (出力ポート)	M26
GPIO27 (出力ポート)	M27

従って、図3.2の配線の場合、SW1=GPIO20=M20、LED1=GPIO24=M24となるため、ラダー図は図3.3となります。解りやすいようにI/Oコメントも定義しました。(インタプリタの都合で、I/OコメントにはASCII文字しか使っていません。)

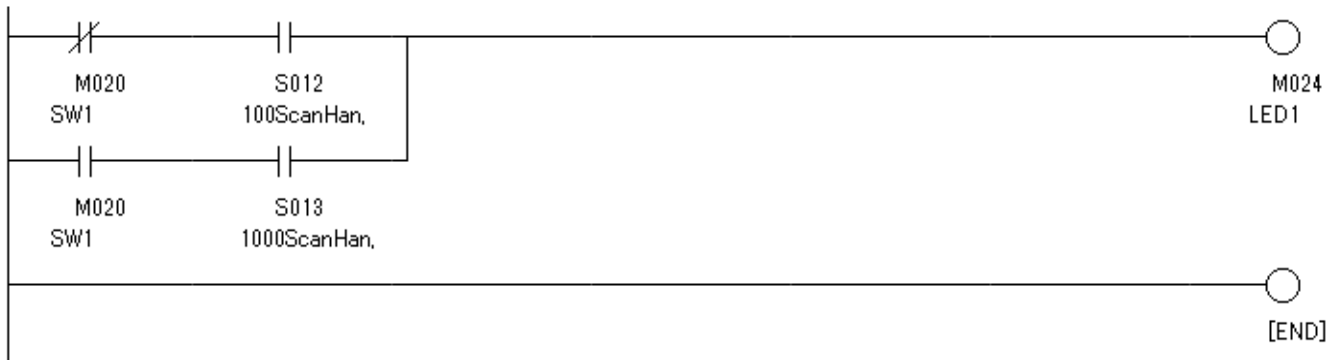


図3.3 作成したラダー図

【手順2】命令語への変換
LD Cv2!で[変換：変換条件ウィンドウを開く]メニューを実行し、変換言語を「命令語」に変更します。(図3.4)

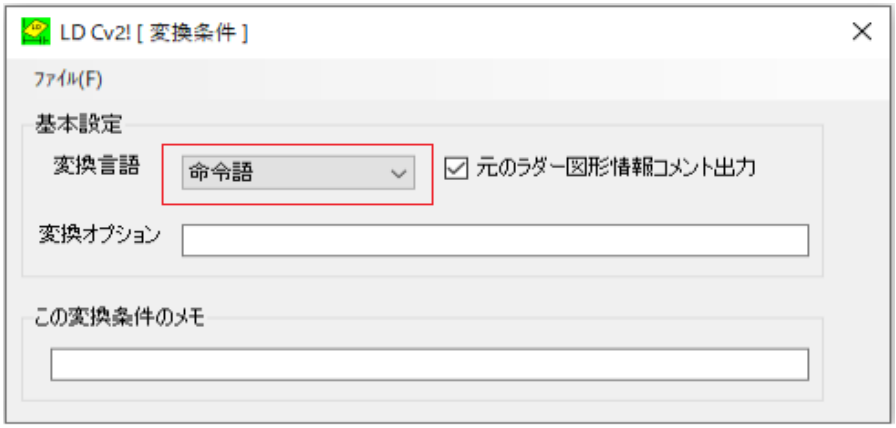


図3.4 変換条件を命令語に変更

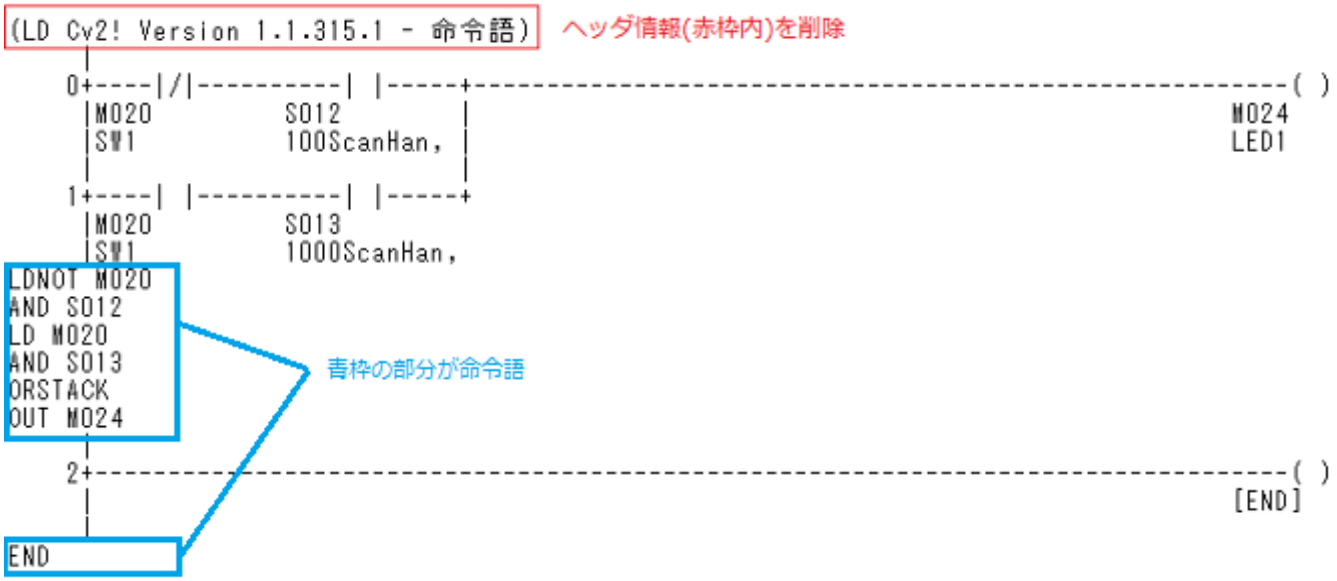
次にLD Cv2!で[変換：変換実行]メニューを実行し、変換結果をファイル名 **plc.il**として保存します。(リスト3.1)

なお、命令語プログラムにシフトJISコードを含んでいると命令語インタプリタでエラーになるので、赤枠内の１行は削除し、**plc.il**で上書き保存しておいてください。

【補足説明】

文字コードの変換機能のあるエディタなどでUTF-8に変換すれば、ヘッダ情報を削除する必要はありません。

LD Cv2!の将来バージョンで、変換オプション(-NH)の指定で(命令語の場合も)ヘッダ情報を出力しないようにする予定です。



リスト3.1 生成された命令語プログラム

LD Cv2!では、命令語プログラムの行の先頭の空白または括弧をコメントと扱うルールにしていますので、リスト3.1 のラダー図形情報は実行に影響を与えません。

【手順3】命令語インタプリタの導入

ここからは、リモート接続したRaspberry Pi ZeroのLinux上での作業となります。

Windowsディスクトップ上に、リモート接続したRaspberry Pi ZeroのLinuxディスクトップを開き、LinuxのWebブラウザで

https://www.takami.com/ldcv/

から、命令語インタプリタ(**ilip□□.zip**)をダウンロードして、解凍し、解凍後のファイル(**ilpi.py**)を、**/home/pi/**に置きます。

メニュー[プログラミング：Thonny Python IDE]を実行し、Thonnyを立ち上げ、 Thonnyのメニュー [ファイル：Open]を実行し、ファイル(**ilpi.py**)を開くと、Editビューに命令語インタプリタのコードが表示されます。(図3.5)

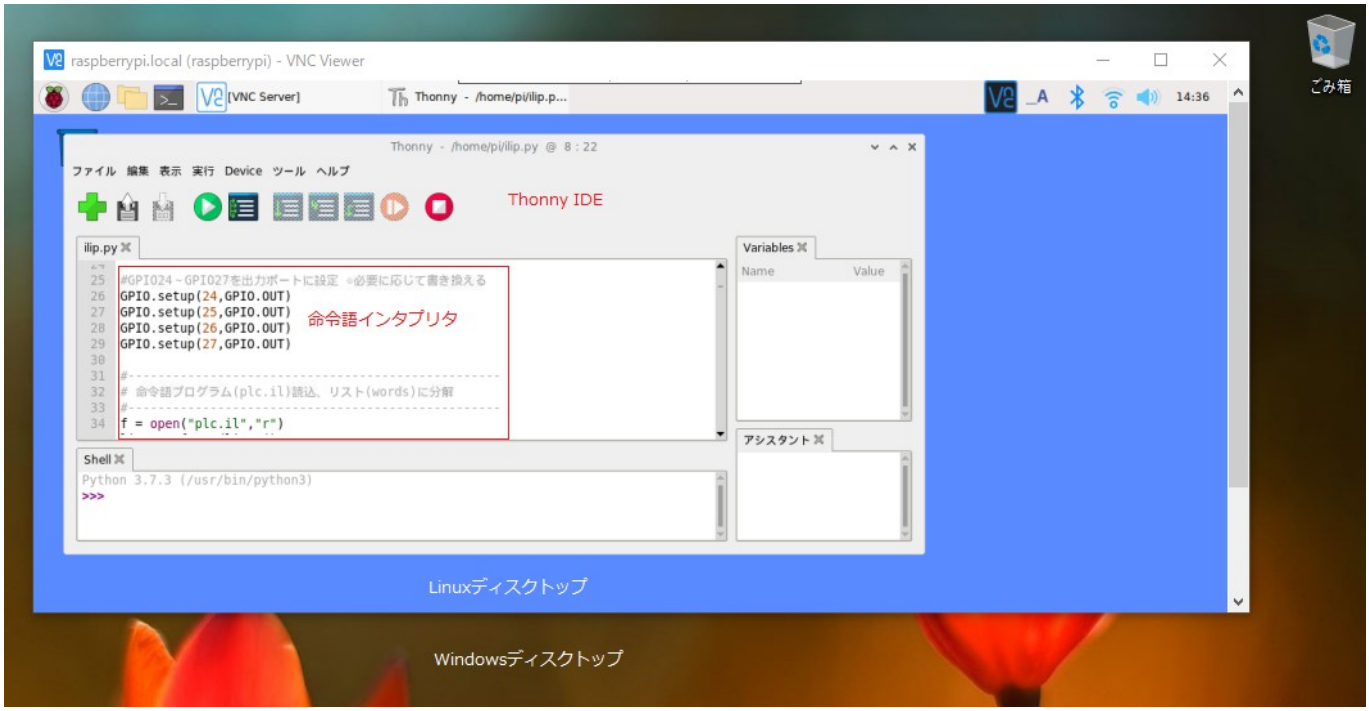


図3.5 ThonnyのEditビューに命令語インタプリタのコードを表示

【手順5】マイコンボードに転送、実行

手順３で作成した命令語プログラム(リスト3.1)を、WindowsパソコンからRaspberry Pi Zeroマイコンボードに転送します。具体的には図3.6で①を選択し、開いたウィンドウの②(Send Files)を押して、転送するファイルを聞かれたら③を指定します。これで、Linuxのディスクトップ上に命令語プログラム(**plc.il**)が転送されます。

転送後のファイルをLinuxのディストップから命令語インタプリタと同じ**/home/pi/**に置きます。

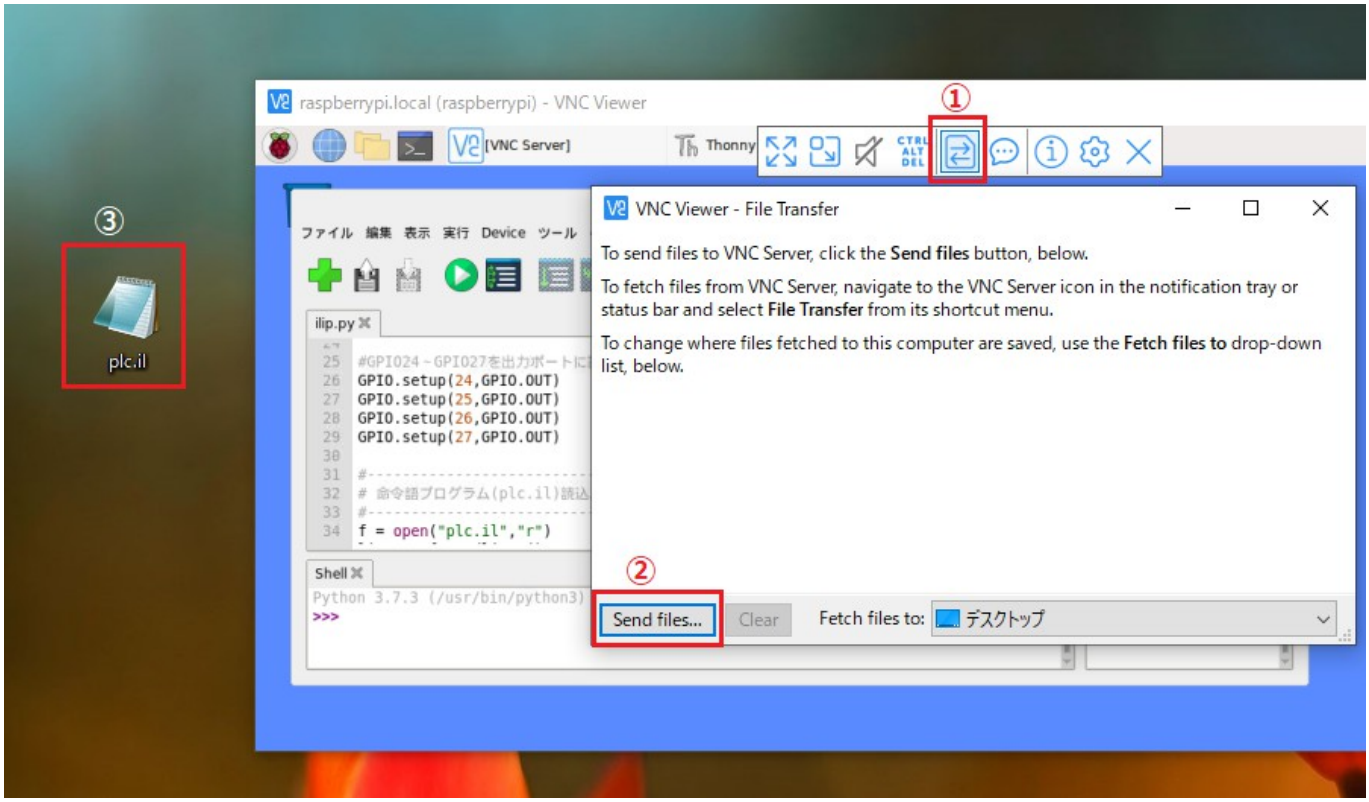


図3.6 命令語プログラムをマイコンに転送

以上で、実行の準備ができました。

Thonnyの実行ボタン(▷)を押下すると、インタプリタが起動し、命令語プログラム(**plc.il**)を読み込んだ後、運転開始します。

運転開始すると、Shellビューに読み込んだ命令語プログラムを表示した後、「PLC START!」と表示します。この時点でSW1の状態によって、LED1の点滅速度が切り替わるはずです。

図3.7は、高速に点滅している時のLED1の駆動ポート(GPIO24)の電圧波形です。約54msで電圧が変化していますが、この信号はS12(100スキャン反転信号)なので、PLCのスキャンタイムは約0.54msであることがわかります。かなりの低速ですが、インタプリタ(Python)で書いたインタプリタ(命令語インタプリタ)プログラムを実行しているので仕方ありません。

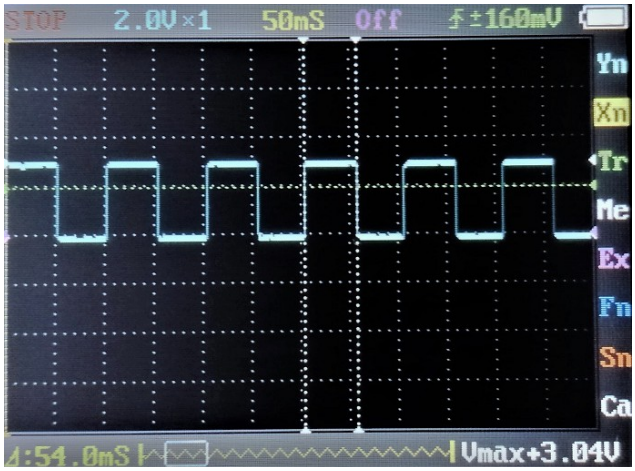


図3.7 LED1の駆動ポートの電圧波形

以上

来歴(LD Cv2!のROM化開発手順)
2020/01/19 新規作成、第１版とする
2020/03/01 表記を一部変更、第2章追加、第2版とする
2020/03/03 ラダー図(図2.3)などの誤記を訂正、第3版とする
2020/03/15 マイコンボードの配線(図1.1)を差し替え、第4版とする
2020/05/06 表記を一部変更、第3章追加、第5版とする