DigitShow Modbus のいろは

Makoto KUNO[1)]　Hiroyuki HASHIMOTO[2)]　Kohei MASHITA[3)]

2025 年 10 月 31 日

1)　東京大学生産技術研究所桑野研究室技術職員　https://github.com/mkt-kuno
2)　東京大学生産技術研究所桑野研究室博士課程　https://github.com/Hiroyuki-Hashimoto
3)　東京大学生産技術研究所桑野研究室修士課程　https://github.com/k-mashita

目次

第 1 章　クイックスタート

ここでは、生研式の三軸試験装置を用いて飽和供試体の三軸圧縮試験を行う際の、DigitShow Modbus の基本的な使い方を示します。

1.1　事前準備：ショートカットの作成

まず、実行ファイル（`DigitShowModbus.exe`）のショートカットを作成します。

次に、ショートカットのプロパティを開き、起動時変数の設定を行います。ショートカットの“target（リンク先）”の最後に以下の引数を追加してください。

- （必須）AD/DA ボードと通信する COM ポートを `--port=COM10` や `-p=COM6` のように指定します。AD/DA ボードのポート番号は、デバイスマネージャー等で確認しましょう。
- （試験機によっては必須）動作モードや、クラッチとモータ動作電圧を指定します。詳細はデベロッパーマニュアルを参照してください。

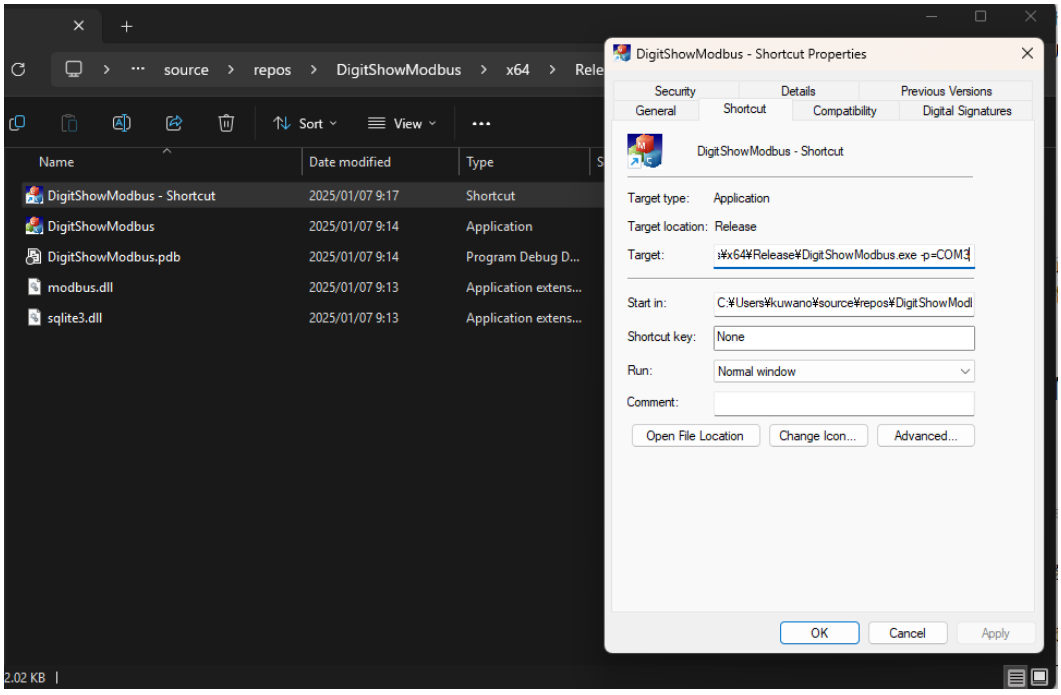

図 1.1　ポート指定の例。`--port=COM10` や `-p=COM6`.

1.2　実験前：アプリを開く

ショートカットをクリックしてアプリを開きましょう。図 1.2 のようなメイン画面が表示されます。値がすべて 0 の場合は、AD/DA ボードとの通信がうまくできていないので、ポート指定を再確認してください。

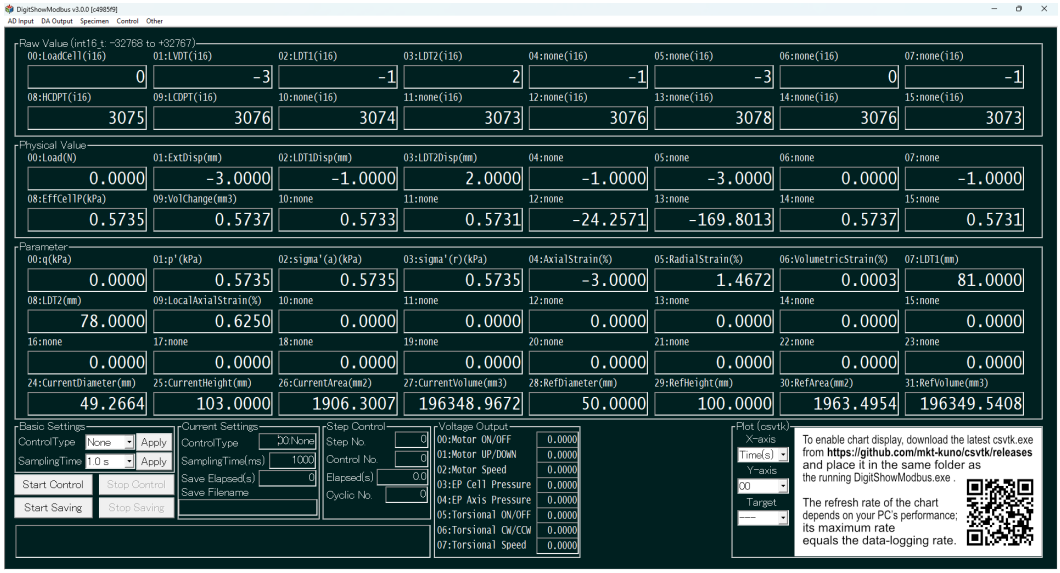

図 1.2　メイン画面。

1.3　供試体を設置する前：ロードセルのキャリブレーション値の入力

1. 左上の “AD Input” タブで “Calibration Factors” をクリックします。
2. キャリブレーション値を設定します。
   - 手動で各チャンネルの値を入力する場合：a, b, c の係数に値を入力し、“Update Varients” を押します。
   - ファイルを読み込む場合：“Load from a file” をクリックして、`.cal` ファイルを読み込みます。現在の設定を保存する場合、“Save to a file” で保存することができます。
3. カウンターバランスを取った状態で、ロードセルのゼロ点を調整をします。ロードセルのチャンネル（CH0）の “Zero” ボタンを押します。そのチャンネルの physical value が 0 になるように、自動的にキャリブレーション値の定数項（c）が変更されます。

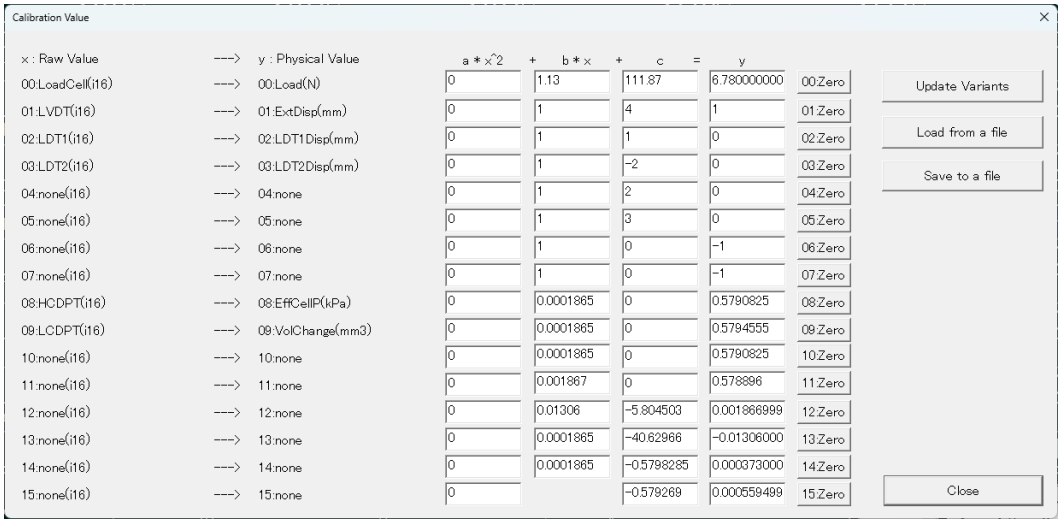

図 1.3　各センサーのキャリブレーション値の入力画面。

1.4　供試体の寸法計測後：供試体初期寸法の入力

1. 左上の “Specimen” タブで “Config” をクリックします。
2. “Initial” 列の “Diameter” 行と “Height” 行に、供試体の初期寸法と高さを入力し、“-> present” ボタンを押します。

1.5　載荷装置と三軸セルを接続した後①：変位計のキャリブレーション値の入力

1.3 節と同様にして、変位計（LVDT）のチャンネル（CH1）キャリブレーション値の入力およびゼロ点の調整を行います。

1.6　載荷装置と三軸セルを接続した後②：Pre-Consolidation

Pre-consolidation は所定の q（軸差応力 [kPa]）を保つ制御コマンドです。主に供試体の飽和過程で使用します。デフォルトの設定では、$q = 0$ なので、等方状態になります。

1. メイン画面左下の “Control ID” を 1 にして、“Set” をクリックします。
2. メイン画面左下の “Start Control” を押すと、制御が始まります。軸受けのクランプを緩め忘れないように気を付けてください。

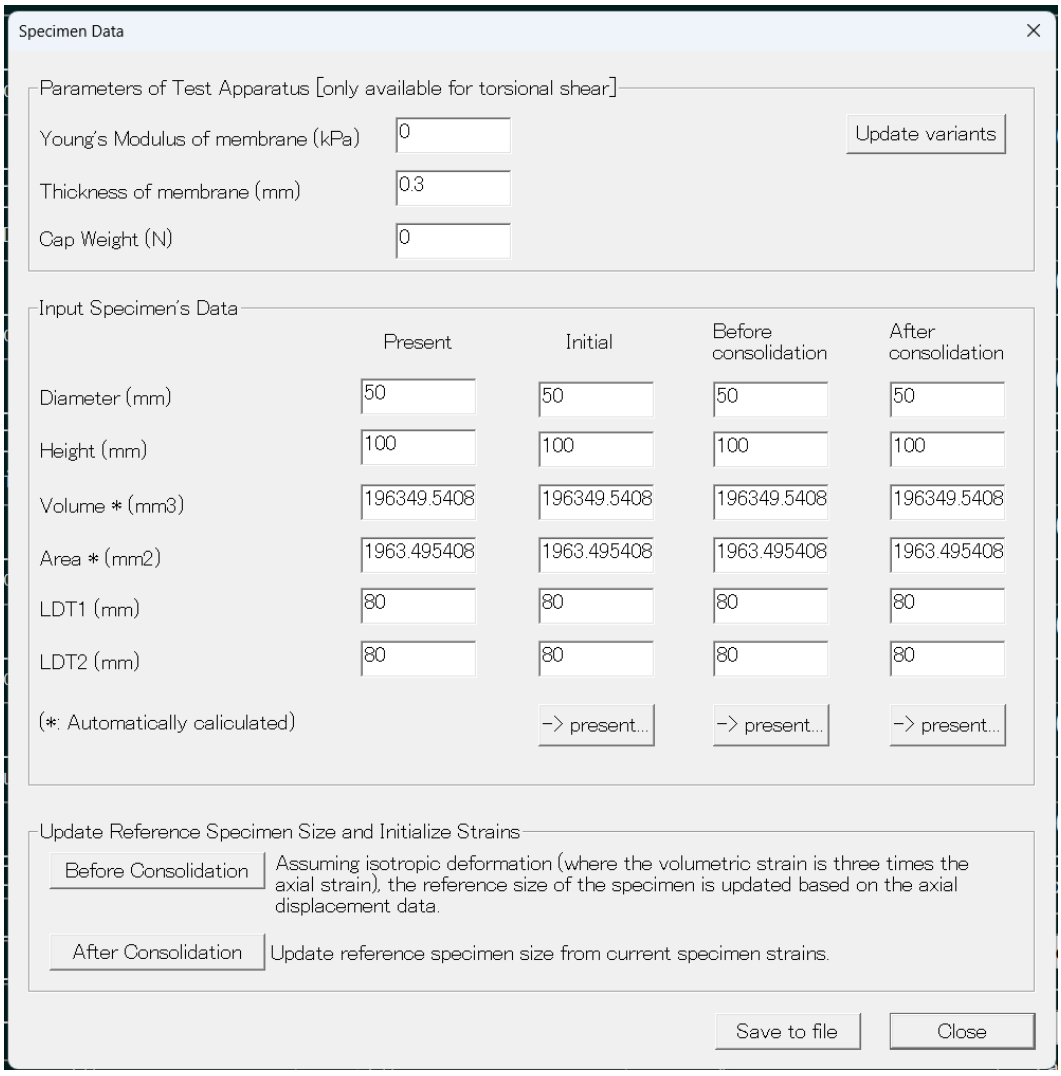

図 1.4　供試体の情報の入力画面。

1.7　供試体の飽和後：差圧計のキャリブレーション値の入力

1.3 節と同様にして、高容量差圧計（HCDPT、CH8）、低容量差圧計（LCDPT、CH9）のキャリブレーション値の入力およびゼロ点の調整を行います。

1.8　圧密①：EP の出力調整

圧密・載荷過程では、EP（電空レギュレータ）を用いてセル圧を自動で制御します。

1. 左上の “D/A Output” タブで “Voltage Output” をクリックします。
2. “CH 3: EP Cell Pressure” に適当な値を入力して “Update” を押します。EP の出力を現在のセル圧に合わせてから、セルに接続します。

1.9　圧密②：ひずみの初期化（Before Consolidation）

1. 左上の “Specimen” タブで “Config” をクリックします。
2. “Before Consolidation” をクリックします。等方的な変形が生じた（軸ひずみの 3 倍の体積ひずみが生じた）と仮定して、変位計から得られる軸ひずみを基に、ひずみの初期化および変位計と LCDPT のゼロ点調整がなされます。

1.10　圧密③：Control from file の設定

1. メイン画面左下の “Stop Control” を押して、制御を停止します。
2. 左上の “Control” タブで “Control from file” をクリックします。
3. 圧密・クリープ・軸圧縮の設定を入力します。“Load” を押すとその Step No. の現在の設定が表示され、“Update” を押すとその Step No. の設定が現在入力している値（パラメータ）に更新されます。
4. Current Step No. が圧密の step（0）になっていることを確認してください。
5. 設定の一例は表 1.1 の通りです（初期拘束圧 20 kPa から、100 kPa で等方圧密し、排水条件で軸ひずみ 15 % まで単調載荷する場合）：
    - Step No.1 の Creep の時間は本来予定している時間よりも長くしておくことを推奨します（意図しないタイミングで次の step に進むのを防ぐため）。
    - 非排水軸圧縮（CU 試験）を行う場合は、載荷中セル圧を変化させる必要がないため、Step No.2 の Monotonic Axial Loading の Args[02] を 0 としてください。
    - Motor RPM と軸変位速度の関係は事前にキャリブレーションしてください（試験機によ

り異なります)。

- Control from file という名前が示すように、タブで区切られた設定ファイルの読み込みや書き出しも可能です。

表 1.1　Control from file の設定例。

Step No.	Control No. (name)	Para0	Para1	Para2	Para3	Para4	Para5	Para6
0	5: Linear Stress Path Loading	20	20	100	100	10	10	1000
1	4: Creep	0	10	1000	10000	100	-	-
2	1: Monotonic Axial Loading	0	300	100	1	15	0	0

図 1.5　制御の設定の入力画面。

1.11　圧密④：データ保存と制御の開始

1. メイン画面左下の “Control ID” を 15 にして “Set” をクリックします。
2. メイン画面左下の “Start Saving” を押して、データの記録を開始します（ファイル名は半角英数字かつ特殊記号を使用しないことを推奨)。
3. メイン画面左下の “Start Control” を押して、圧密を開始します。

1.12　軸圧縮①：データ保存、圧密の停止

メイン画面で “Stop Saving” および “Stop Control” を押して、圧密中のデータの記録と制御を終了します。

1.13 軸圧縮②：ひずみの初期化（After Consolidation）

1. 左上の “Specimen” タブで “Config” をクリックします。
2. “After Consolidation” をクリックします。現在の軸変位や体積変化をもとに、ひずみの初期化および変位計と LCDPT のゼロ点調整がなされます。

1.14 軸圧縮③：Step No の変更

1. 左上の “Control” タブで “Control from file” をクリックします。
2. チェックボックスをクリックしてから、右矢印ボタンをクリックして、Step No を 2（軸圧縮の制御）に進めます。

1.15 軸圧縮④：データの保存、載荷の開始

1. “Start Saving” を押して、データの記録を開始します（ファイル名は半角英数字かつ特殊記号を使用しないことを推奨)。
2. “Start Control” を押して、載荷を開始します（非排水条件で載荷する場合は、バルブを締めるのを忘れずに！)。

1.16 片付け

1. 軸圧縮が終わって Step No が 3 に進んだら、メイン画面左下の “Stop Saving”, “Stop Control” を押して、データの記録と制御を終了します。
2. 左上の “Specimen” タブで “Config” をクリックし、“Save to file” ボタンを押して、供試体の情報をファイルに保存します。
3. すべての片付けが終わったら、最後にアプリを閉じます。お疲れ様でした！

第 2 章　ユーザーマニュアル

ここでは、クイックスタートで書かれなかった、特に試験を実施するうえで必要となりうる内容を含みます。

2.1　動作モードと背景色

ここでは、DigitShowModbus の動作モードと背景色について説明します。DigitShowModbus は、以下の 2 つの動作モードを持ちます。

生研式モータ動作モード　生研式の三軸圧縮試験装置を用いて、飽和供試体の三軸圧縮試験を行うためのモードです。

生研式ねじり試験動作モード　生研式の中空ねじりせん断試験装置を用いて、飽和供試体の中空ねじりせん断試験を行うためのモードです。まだ開発中です。

ねじり試験モードは未完成のため、現在は三軸圧縮試験モードのみが使用可能です。以下では、三軸圧縮試験モードについて説明します。

通常、DigitShowModbus の背景色は深緑色です。背景色が暗い赤色の場合、オリジナル版ではないことを示します。本マニュアルはオリジナル版を対象としたものであり、オリジナル版以外の動作については、サポートできません。

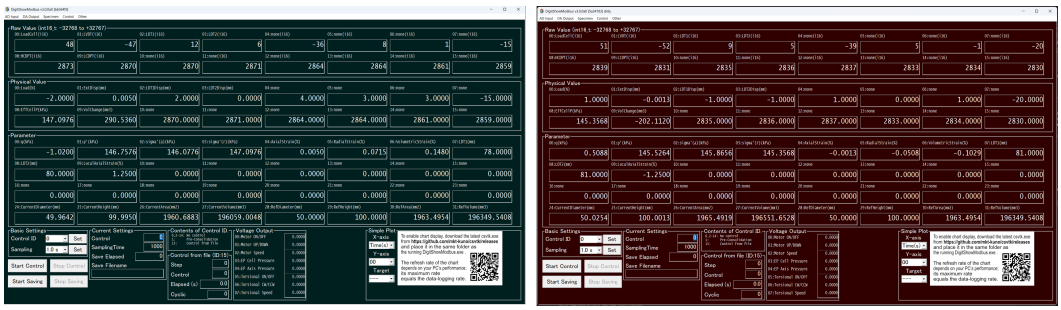

図 2.1　オリジナル版（左）と非オリジナル版（右）の背景色の違い

2.2　起動時変数の設定（ユーザー向け）

ここでは、Windows のショートカットのプロパティを利用した起動時変数の設定について、特に一般ユーザー向けのものに絞って説明します。すべての起動時変数については、3.1 節を参照し

てください。

起動時変数を設定するには、実行ファイル（DigitShowModbus.exe）のショートカットを作成し、ショートカットのプロパティを開いてください。*Target* で、実行ファイルのパスの後に、半角スペースを開けてから、以下に示す各種の起動時変数の指定を行ってください。

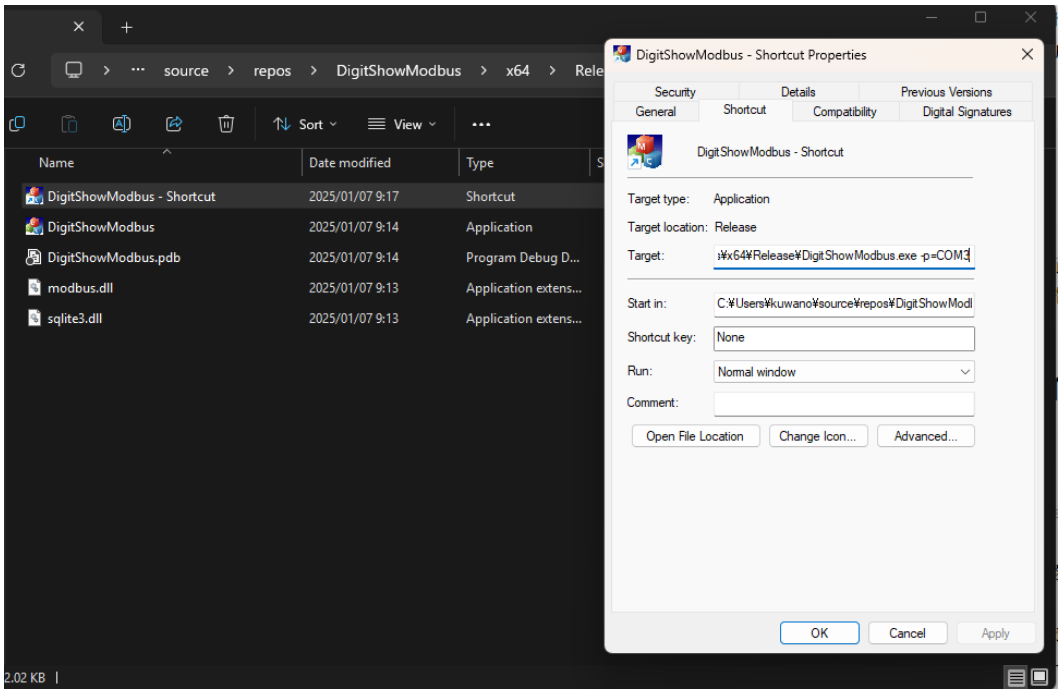

図 2.2　ショートカットのプロパティを利用した起動時変数の設定例（ここでは `-p=COM3` で COM ポートを指定している）。

2.2.1　ModbusRTU COM Port

完全表記が `--port=` 、短縮表記が `-p=` です。

ModbusRTU ボードとの通信に使用する COM ポートを指定するための引数です。`--port=COM10` や `-p=COM6` のように指定します。ModbusRTU ボードのうち、Trio（v1）・Quartet（v2）では、製造時期や互換ボードかどうかによって、FT232 や CH340 など、何かしらの USB シリアル変換 IC を使用しているため、デバイスマネージャー等で確認しましょう。また、動作際してはドライバが必要になります。ArduinoIDE をインストールするか、多くの場合 WindowsUpdate の Additional Update でドライバがインストール可能です。Yamanin（v3）では、USB-CDC（ACM）を使用した高速・高信頼のシリアル通信を採用しており、基本的に Windows 標準のドライバが利用可能なため、追加のドライバインストールは不要です。

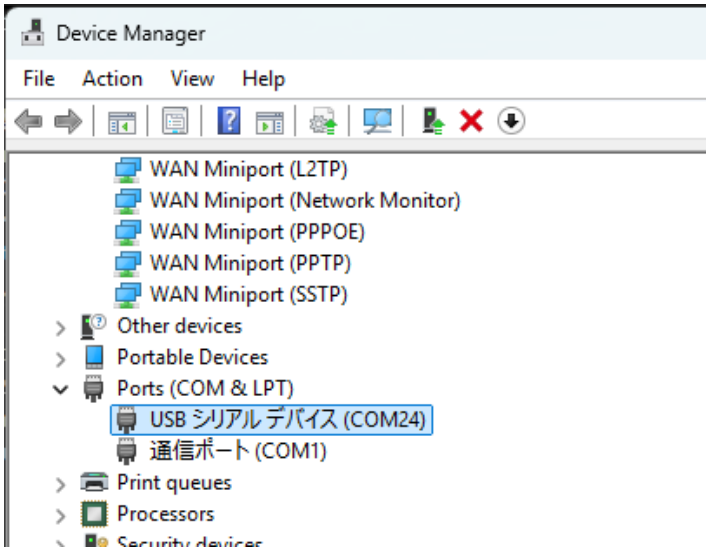

図 2.3　COM ポートの例

2.2.2　動作モード

完全表記が `--mode=` 、短縮表記が `-m=` です。

“0” もしくは “motor” を指定することで生研式モータ動作モードで起動します。

“1” もしくは “torsional” を指定することで生研式ねじり試験動作モードで起動します。

基本的に Motor 動作モードで起動することをお勧めします。ねじりモードは開発中です。Debug ビルドで試用して、安全性を確認してから長期運用してください。

2.2.3　クラッチ&モータ動作電圧

完全表記が `--invert_motor_enable=` 、短縮表記が `-ime=` です。

完全表記が `--invert_motor_direction=` 、短縮表記が `-imd=` です。

“True” もしくは “true”、“1” を指定することで極性が反転します。デフォルト状態はソースコードの確認をしてください。大きな変更が加わっていない限りは、“モータ ON” が “5.0 [V]”、“モータ OFF” が “0.0 [V]”。同じように “モータ UP” が “5.0 [V]”、“モータ DOWN” が “0.0 [V]” です。コードを検索する場合は下記のような記述を探してください。

CDigitShowModbus.cpp より抜粋

```
1  #define AO_DEFAULT_VLT_MOTOR_ON     (5.0f)          // Voltage
   of Axial Motor ON
2  #define AO_DEFAULT_VLT_MOTOR_OFF    (0.0f)          // Voltage
   of Axial Motor OFF
3  #define AO_DEFAULT_VLT_MOTOR_UP     (5.0f)          // Voltage
   of Axial Motor UP
4  #define AO_DEFAULT_VLT_MOTOR_DOWN   (0.0f)          // Voltage
```

```
    of Axial Motor DOWN
```

subsection フォント完全表記が `--font=`、短縮表記が `-f=` です。フォントの指定は、`--font="MS Gothic"` `--font="Source Han Code JP"` のように指定します。デフォルトは “Lucida Sans Typewriter” ですが、他のフォントを指定することも可能です。見やすさもありますが、Windows の環境によっては存在しないフォントを使用しようと、アプリケーションが試行する可能性がありますので、環境ごとに確認してください。

2.3 センサーと入力チャンネル構成

ここでは標準的なセンサーと入力チャンネルの構成について説明します。DigitShow Modbus では、独自開発した Modbus ボードと通信を行い、センサーからアナログ信号を受け取ります。CH0-7 までがひずみゲージ式のセンサーの信号の受け取りに、CH8-15 が 0-5V の直流電圧信号の受け取りに、それぞれ対応しています。なお、センサーのキャリブレーション方法については、別のマニュアルを参照してください。

2.3.1 CH0: ロードセル

ロードセルは、供試体にかかる鉛直荷重を計測するためのセンサーです。4 ゲージ（フルブリッジ）式で、セル室内のキャップ上部に取り付けられることが想定されます。

2.3.2 CH1:LVDT（外部変位計）

外部変位計は、供試体の鉛直変位を計測するためのセンサーです。4 ゲージ（フルブリッジ）式で、セル室の外などに取付られることが想定されています。

2.3.3 （任意）CH2,3:LDT（局所変位計）

LDT は Local Displacement Transducer の略で、Goto et al. (1991) によって開発された供試体の局所軸ひずみを計測するためのセンサーです。供試体側面上の 2 点間の軸圧縮量から局所軸ひずみを求めます。ベディングエラー（「外部変位計などによってキャップや載荷ピストンの軸変位から求めた軸ひずみ」に含まれる測定誤差）の無い正確な軸ひずみを求めることができます。計測できる最大ひずみは 1% 程度ですが、微小なひずみを正確に計測することができます。供試体の直径方向に対称に 1 組設置し、2 つの LDT で計測された値の平均値を用います。なお、LDT の設置は任意であり、使用しなくてもプログラムの動作に影響はありません。

2.3.4 CH4-7: 空きチャンネル

ひずみゲージ式の各種センサーを接続することができる空きチャンネルです。

2.3.5 CH8:HCDPT（高容量差圧計）

供試体の半径方向の有効応力を計測するためのセンサーです。セル水の圧力と供試体の間隙水圧の差圧を測ります。差圧に応じた 4-20mV の DC 信号を伝送する差圧伝送器および、DC 信号を直流電圧信号に変換するディストリビューターを使用することが想定されます。

2.3.6 CH9:LCDPT（低容量差圧計）

供試体の排水量を計測するためのセンサーです。排水量に応じて、供試体に繋がるビュレットの水位が変動するため、ビュレットの基準水位との水位差を差圧として計測します。HCDPT と同じく差圧伝送器とディストリビューターを使用することが想定されます。

2.3.7 CH10-15: 空きチャンネル

直流電圧で入力される各種センサーを接続することができる空きチャンネルです。

2.4 出力チャンネル構成

DigitShow Modbus では、独自開発した Modbus ボードと通信を行い、6 チャンネルまたは 8 チャンネルの 0-10V のアナログ出力を使用することができます。ここではアナログ出力チャンネルの構成について説明します。

2.4.1 CH0: モーター ON/OFF

鉛直載荷モーターのクラッチの ON/OFF を切り替えるための出力チャンネルです。デフォルトで、OFF のときは 0V、ON のときは 5V が出力されます。2.2 節に示す起動時変数の設定で ON/OFF 電圧の極性を反転させることができます。

2.4.2 CH1: モーター UP/DOWN

鉛直載荷モーターの UP/DOWN を切り替えるための出力チャンネルです。デフォルトで、UP のときは 5V、DOWN のときは 0V が出力されます。2.2 節に示す起動時変数の設定で UP/DOWN の極性を変更することができます。試験機により極性を反転させる必要があるので注意してください。

2.4.3 CH2: モーター速度

鉛直載荷モーターの回転速度（RPM）を設定するための出力チャンネルです。多くの生研式三軸試験機では、300RPM/V となっています。

2.4.4 CH3: セル EP 圧力

EP は電空レギュレータ (Electrical-Pneumatic converter) の略称で、電圧に応じた空圧を出力する装置で、セル圧の制御に使用されます。デフォルトでは、0.01275kPa/V の設定となっています。

2.4.5 CH4-7: 空きチャンネル

生研式モータ動作モードでは、空きチャンネルとなります。

2.5 供試体寸法・ひずみ・応力の計算式

ここでは、DigitShow Modbus で計算される供試体寸法・ひずみ・応力の計算式について説明します。

2.5.1 供試体寸法

供試体寸法には、時々刻々と変化する現在の寸法のほかに、レファレンス値という概念が存在します。現在の寸法は、「どこからどのくらい増えた／減ったのか」を計算することによって得られますが、レファレンス値とは、この「どこから」の情報に相当します（「どのくらい増えた／減った」の情報はセンサーから得られます)。また、レファレンス値はひずみの計算の分母に用いられます。レファレンス値は時々刻々の変化はしない定数ですが、値は圧密の前と後で更新されます。

2.5.1.1 供試体寸法のレファレンス値

試験の最初に、実験者は供試体の初期直径を d_{init} と初期高さ h_{init} を測定します。これらをもとに初期体積 V_{init} が次のように計算されます。

$$V_{\mathrm{init}} = \frac{\pi {d_{\mathrm{init}}}^2 h_{\mathrm{init}}}{4} \tag{2.1}$$

Before Consolidation を押す時点での LVDT で計測された変位を δ_{BC} とします。等方的な変形が生じた（軸ひずみの 3 倍の体積ひずみが生じた）という仮定のもと、供試体寸法のレファレンス値（圧密前高さ h_{BC}、圧密前体積 V_{BC}、圧密前直径 d_{BC}）は次のように更新されます。

$$h_{\mathrm{BC}} = h_{\mathrm{init}} - \delta_{\mathrm{BC}} \tag{2.2}$$

$$V_{\mathrm{BC}} = V_{\mathrm{init}} \left(1 - \frac{3\delta_{\mathrm{BC}}}{h_{\mathrm{init}}} \right) \tag{2.3}$$

$$d_{\mathrm{BC}} = \sqrt{\frac{4V_{\mathrm{BC}}}{\pi h_{\mathrm{BC}}}} \tag{2.4}$$

After Consolidation を押す時点での LVDT で計測された変位を δ_{AC} と、LCDPT で計測された排水量を W_{AC} とします。これらをもとに、圧密後の供試体寸法のレファレンス値（圧密前高さ h_{AC}、

圧密前体積 V_{AC}、圧密前直径 d_{AC}）は次のように更新されます。

$$h_{\mathrm{AC}} = h_{\mathrm{BC}} - \delta_{\mathrm{AC}} \tag{2.5}$$

$$V_{\mathrm{AC}} = V_{\mathrm{BC}} - W_{\mathrm{AC}} \tag{2.6}$$

$$d_{\mathrm{AC}} = \sqrt{\frac{4V_{\mathrm{AC}}}{\pi h_{\mathrm{AC}}}} \tag{2.7}$$

2.5.1.2 供試体の現在の寸法

現在のレファレンスの高さを h_{ref}、体積を V_{ref} とします。また、LCDPT で計測された現在の排水量（供試体から出る方向が正）を W と、LVDT で計測された鉛直変位（供試体が圧縮する方向が正）を δ とします。このとき、現在の供試体の高さ h, 体積 V と断面積 A は次のようになります。

$$h = h_{\mathrm{ref}} - \delta \tag{2.8}$$

$$V = V_{\mathrm{ref}} - W \tag{2.9}$$

$$A = \frac{V}{h} \tag{2.10}$$

2.5.2 ひずみ

現在の高さを h、体積を V とし、レファレンス高さを h_{ref}、体積を V_{ref} とします。このとき、軸ひずみ ε_{a} [%], 体積ひずみ ε_{v} [%], 半径方向ひずみ ε_{r} [%] は次のように計算されます。

$$\varepsilon_{\mathrm{a}} = \left(1 - \frac{h}{h_{\mathrm{ref}}}\right) \times 100 \tag{2.11}$$

$$\varepsilon_{\mathrm{v}} = \left(1 - \frac{V}{V_{\mathrm{ref}}}\right) \times 100 \tag{2.12}$$

$$\varepsilon_{\mathrm{r}} = \left(1 - \sqrt{\frac{1-\varepsilon_{\mathrm{v}}}{1-\varepsilon_{\mathrm{a}}}}\right) \times 100 \tag{2.13}$$

なお、昔のプログラム（DigitShowBasic）では、真ひずみが使用されていました。地盤工学会の基準は公称ひずみを採用していたため、DigitShowModbus ではよりシンプルに公称ひずみを採用しました。ひずみ 10% くらいまでであれば、両者の差は小さいですが、変形が大きくなるほど、公称ひずみ（現プログラム）と真ひずみ（旧プログラム）の差が大きくなります。既往研究との比較をする際には注意してください。真ひずみはひずみの足し合わせができることが利点とされるようです。必要に応じて、事後的に自分で計算してください。

2.5.3 応力

ロードセルが計測する荷重を F とします。軸差応力 q は次式で計算されます。

$$q = \frac{F}{A} \tag{2.14}$$

HCDPT の計測する値がそのまま半径方向の有効応力 ${\sigma'}_{\mathrm{r}}$ になります。これらを用いて、軸方向の有効応力 ${\sigma'}_{\mathrm{a}}$ と平均有効応力 p' は次のようになります。

$$ {\sigma'}_{\mathrm{a}} = {\sigma'}_{\mathrm{r}} + q \tag{2.15} $$

$$ p' = \frac{{\sigma'}_{\mathrm{a}} + 2{\sigma'}_{\mathrm{r}}}{3} \tag{2.16} $$

$$ = \frac{q + 3{\sigma'}_{\mathrm{r}}}{3} \tag{2.17} $$

2.5.4 (LDT を使用した場合のみ) 局所鉛直ひずみ

LDT(Local Displacement Transducer) を使用した場合、鉛直方向の局所ひずみを計算することができます。LDT のレファレンス長さを LDT_{ref}、LDT で計測された局所変位を δ_{LDT} とします。このとき、現在の LDT の長さ LDT と局所鉛直ひずみ $\varepsilon_{\mathrm{LDT}}$ は次のように計算されます。

$$ LDT = LDT_{\mathrm{ref}} - \delta_{\mathrm{LDT}} \tag{2.18} $$

$$ \varepsilon_{\mathrm{LDT}} = \left(1 - \frac{LDT}{LDT_{\mathrm{ref}}}\right) \times 100 \tag{2.19} $$

なお、LDT は、供試体のひねり状態の補正するために、供試体の直径方向に対象に 1 組設置し、2 つの LDT で計測された $\varepsilon_{\mathrm{LDT}}$ の平均値を用いることが推奨されます。

2.6 各ウインドウの説明

ソフトウェアの各ウインドウについて説明します。

2.6.1 MainWindow

起動時に表示されるメインウインドウの各部分の説明をします。

1. メイン画面の左上にあるタブから、キャリブレーションや供試体の情報、コントロール設定などに関する各ウィンドウ（次項以降で詳述）を開くことができます。
2. Raw Value では、センサーから AD 変換されたデジタル信号の生データを表示します。最小値は-32768、最大値は 32767 の int16 型の値が表示されます。
3. Physical Value では、Raw Value をキャリブレーション値を用いて物理量 (N や mm など) に変換した値を表示します。
4. Parameters では、応力ひずみや、供試体の現在の寸法、レファレンス寸法などが表示されます。これらの値は、Physical Value や供試体寸法の情報をもとに計算された値です。
5. Basic Settings では、Control ID や、Sampling レートの設定や、制御とサンプリングの開始・停止を行うことができます。なお、Control ID や Sampling レートの設定は、Set ボタンを押すことで反映されます。

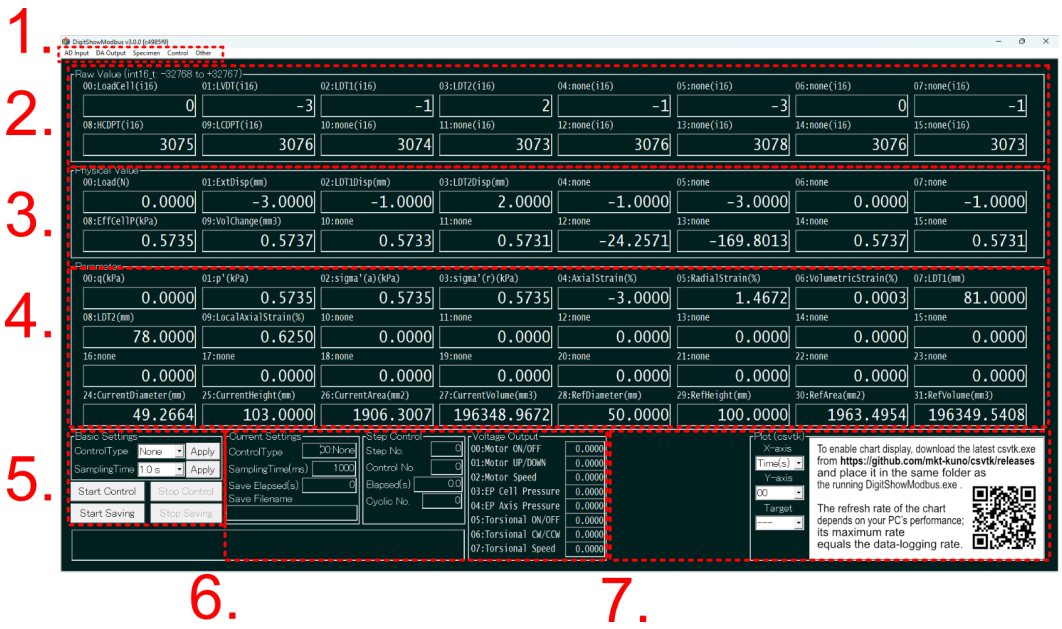

図 2.4　メイン画面。

6. Current Settings & Voltage Output では、現在の各種設定や 2.4 節に示すアナログ電圧出力値の情報が表示されます。
7. csvtk Plot では、Raw Value や Physical Value、Parameters の値の変化をグラフで表示します。グラフの表示を行うには、csvtk.exe を DigitShowModbus.exe と同じフォルダに配置し、Sampling を開始する必要があります。なお、オリジナルの csvtk では応力経路などの一部グラフを正しく表示することができないため、フォーク版（`https://github.com/mkt-kuno/csvtk/releases`）を使用することを推奨します。

2.6.2　DA Output - Voltage Output

アナログ出力について、直接電圧を指定して出力できるウインドウです。EP を三軸セルに接続する前に、EP の出力圧力を調整する際に使用します。また、装置を組み上げたり接続し直した場合に、クラッチやモータなどの動作確認などにも使用されます。

2.6.3　Specimen - Config

供試体の初期寸法を設定するウインドウです。供試体の直径と高さ・LDT の長さなどを入力します。 TDB...

2.6.4　Control - Pre-Consolidation

Pre-Consolidation は先行圧密用の制御コマンドです。主に供試体の飽和過程で使用します。制御の詳細については、2.7 節を参照してください。

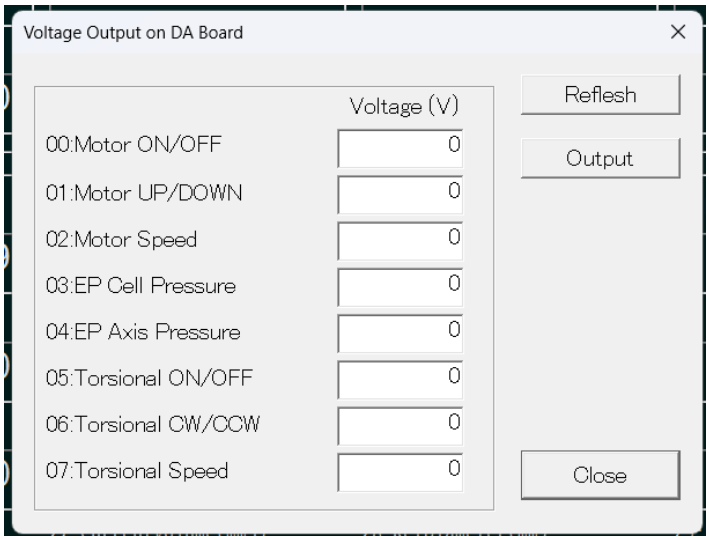

図 2.5　DA Output - Voltage Output ウィンドウ。

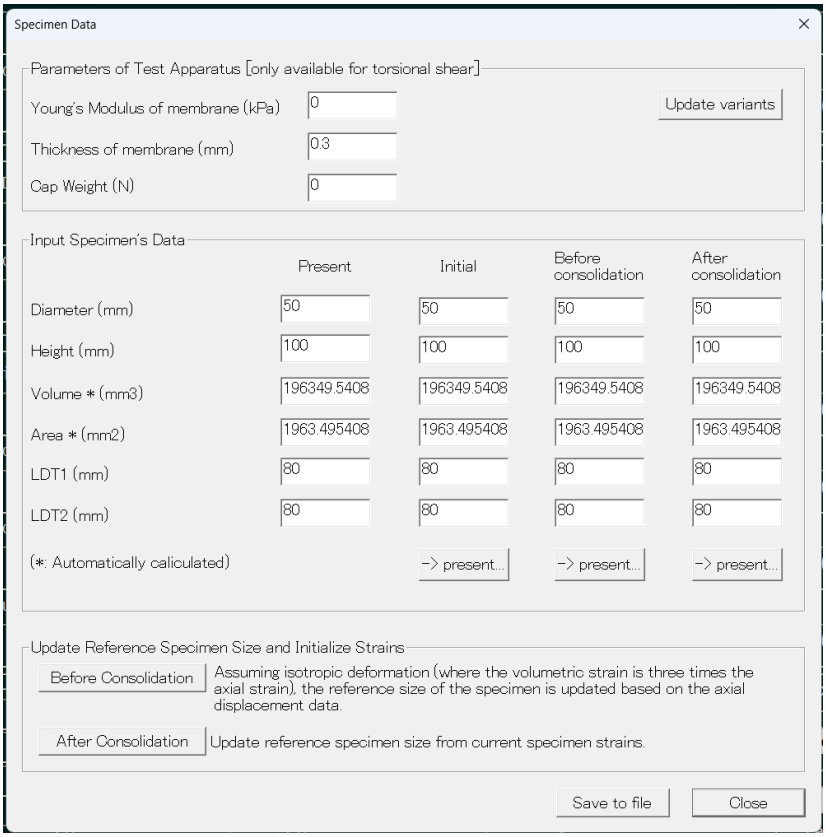

図 2.6　Specimen - Config ウィンドウ。

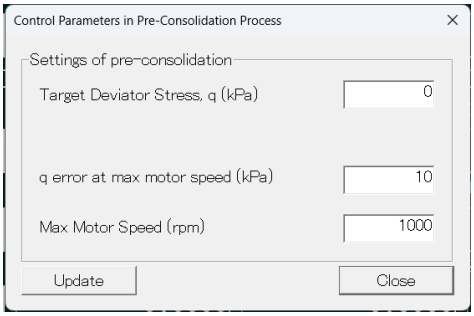

図 2.7　Control - Pre-Consolidation ウィンドウ。

2.6.5　Control - Control from file

複数のコントロールを順序立てて自動的に行うための設定を行うウインドウです。圧密・クリープ・モーター速度・圧縮方向などの設定を入力します。

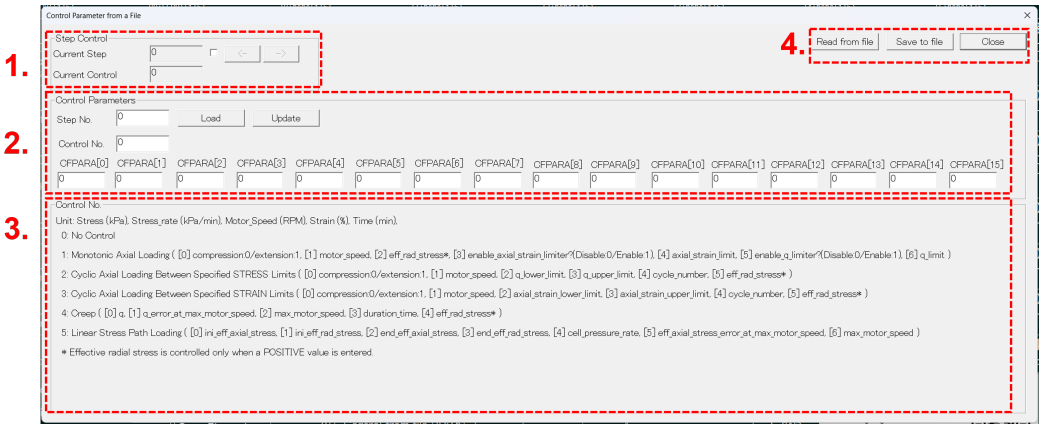

図 2.8　Control - Control from file ウィンドウ。

1. 現在の Control の Step No. が表示されます。チェックボックスを選択した上で、矢印ボタンをクリックすると Step No. のインクリメント・デクリメントができます。なお、設定可能な Step No. は 0 から 1023 までです。
2. 各 Step の Control No. の設定およびそのパラメータの設定を行うことができます。Load ボタンをクリックすると、現在入力中の Step No. の、DigitShowModbus に設定されている Control No. とパラメータが表示されます。Update ボタンをクリックすると、現在入力されている Step No. の Control No. とパラメータが、DigitShowModbus 内に登録されます。
3. 各 Control No. の詳細が記述されています。なお、Control No. の詳細は、2.7 節を参照してください。
4. Read from file ボタンをクリックすると、指定したファイルから Step No., Control No. とパラメータを読み込むことができます。Save to file ボタンをクリックすると、現在設定されている

Step No., Control No. とパラメータの情報を、指定したファイルに保存することができます。

Close ボタンを押すと、ウインドウが閉じます。

制御の詳細については、2.7 節を参照してください。

2.6.6 Other - Version

ソフトウェアのバージョン情報を表示するウインドウです。

図 2.9 Other - Version ウィンドウ。

2.6.7 Other - Open Log Folder

ソフトウェアのログを表示するウインドウです。エラーが発生した場合などに参照します。

2.6.8 Other - Open Config Folder

ソフトウェアのが自動で生成する設定ファイルを表示するウインドウです。

2.7 コントロールについて

ここでは、DigitShowModbus の各コントロール機能について説明します。大きく分けて、供試体の飽和過程で使用する Pre-Consolidation (Control ID:1) と、圧密や軸圧縮時に使用する Control from file (Control ID:15) の 2 つに分かれます。

2.7.1 Pre-Consolidation

Pre-Consolidation は、供試体の飽和過程で使用するコントロールです。図 2.10 に示すように、設定した軸差応力を保つように載荷軸を上下させます。デフォルトの設定では、$q = 0\,\mathrm{kPa}$ なので、等方状態を保とうとします。

軸差応力が目標値から離れれば離れるほど、高速でモーターを回転させて載荷軸を上下させます（いわゆる P 制御のようなもの）。ただし、クラッチの消耗を防ぐため、軸差応力が 0.5 kPa 以下しか離れていないのであれば、載荷軸は動かないようにプログラムされています（この閾値はユーザーからは設定できません）。

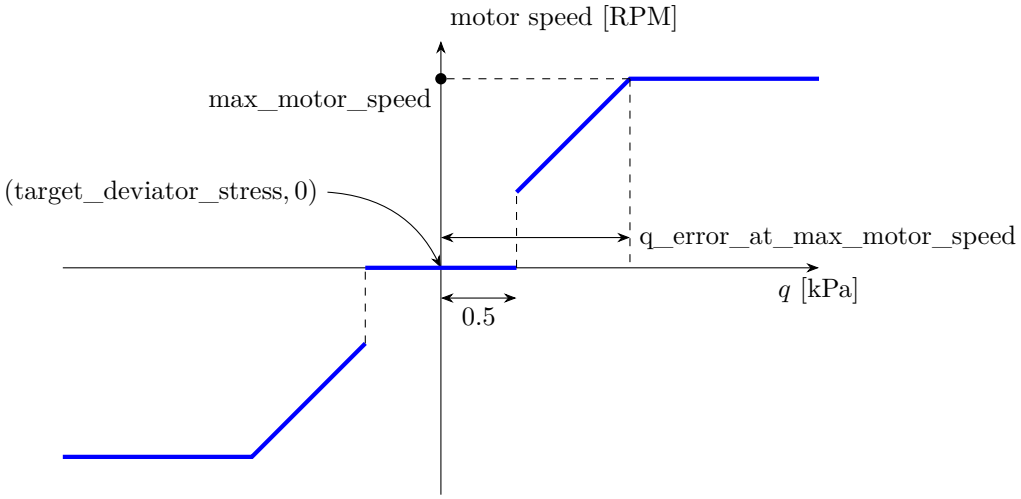

図 2.10　Pre-Consolidation の動作のイメージ

2.7.2　Control from file - 0: No Control

Control No.0: No control は、何の制御も行いません。デフォルトの制御として設定されています。モーターのクラッチは ON のまま、モーターは回転しません（載荷軸は全く動きません）。セル圧を制御する EP の出力電圧は、それまでの値を保持します（セル圧も基本的に変動しません）。

2.7.3　Control from file - 1: Monotonic Axial Loading

Control No.1: Monotonic Axial Loading は、単調軸圧縮・軸伸長を行うコントロールです。
以下のパラメータが設定できます。

- Args[00]: 圧縮 (compression) か伸長 (extension) か。0 が圧縮、1 が伸長です。
- Args[01]: モーターの回転速度。単位は RPM。RPM と実際の載荷速度の関係は試験機および減速機の設定などによって異なるため、予め確認しておく必要があります。
- Args[02]: 半径方向の有効応力。単位は kPa。正の値が入力されたときのみ、EP でセル圧を制御することで、円周方向の有効応力を設定された値に保ちます（0 以下の値が入力された場合、EP の出力圧力は変化しません）。逆に、非排水三軸試験を行う場合には、必ず 0 を入力してください。
- Args[03]: 軸ひずみのリミッター設定を行うか否か。1 で有効化、0 で無効化。リミッターを有効化した場合、Args[04] で設定された値を超えるひずみが計測された場合に次の Step No. に進みます。

- Args[04]: 軸ひずみのリミッター値。単位は %。Args[03] でリミッターを有効化した場合に、Args[04] で設定された値を超えるひずみが計測された場合に次の Step No. に進みます。
- Args[05]: 軸差応力のリミッター設定を行うか否か。1 で有効化、0 で無効化。リミッターを有効化した場合、Args[06] で設定された値を超える軸差応力が計測された場合に次の Step No. に進みます。
- Args[06]: 軸差応力のリミッター値。単位は kPa。Args[05] でリミッターを有効化した場合に、Args[06] で設定された値を超える軸差応力が計測された場合に次の Step No. に進みます。

なお、Args[03] と Args[05] のリミッターは少なくとも 1 つは設定する必要があります。両方とも設定しない場合、載荷が行われません。また、非排水三軸圧縮試験の場合には、Args[04] にロードセルの定格容量を設定することをお勧めします（応力経路で限界状態線上に達したのちに軸差応力が増加し続ける場合があるためです)。

2.7.4 Control from file - 2: Cyclic Axial Loading Between Specified STRESS Limits

Control No.2: Cyclic Axial Loading Between Specified STRESS Limits は、指定した応力範囲での繰り返し軸載荷を行うコントロールです。

以下のパラメータが設定できます。

- Args[00]: 圧縮 (compression) か伸長 (extension) か。0 が圧縮、1 が伸長です。
- Args[01]: モーターの回転速度。単位は RPM。RPM と実際の載荷速度の関係は試験機および減速機の設定などによって異なるため、予め確認しておく必要があります。
- Args[02]: 軸差応力の下限値。単位は kPa。
- Args[03]: 軸差応力の上限値。単位は kPa。
- Args[04]: 何サイクル載荷を行うか。所定のサイクル数の載荷を終えた場合、次の Step No. に進みます。
- Args[05]: 半径方向の有効応力。単位は kPa。正の値が入力されたときのみ、EP でセル圧を制御することで、円周方向の有効応力を設定された値に保ちます（0 以下の値が入力された場合、EP の出力圧力は変化しません)。逆に、非排水三軸試験を行う場合には、必ず 0 を入力してください。

2.7.5 Control from file - 3: Cyclic Axial Loading Between Specified STRAIN Limits

Control No.3: Cyclic Axial Loading Between Specified STRAIN Limits は、指定した軸ひずみの範囲での繰り返し軸載荷を行うコントロールです。以下のパラメータが設定できます。

- Args[00]: 圧縮 (compression) か伸長 (extension) か。0 が圧縮、1 が伸長です。
- Args[01]: モーターの回転速度。単位は RPM。RPM と実際の載荷速度の関係は試験機および減速機の設定などによって異なるため、予め確認しておく必要があります。
- Args[02]: 軸ひずみの下限値。単位は %。

- Args[03]: 軸ひずみの上限値。単位は %。
- Args[04]: 何サイクル載荷を行うか。所定のサイクル数の載荷を終えた場合、次の Step No. に進みます。
- Args[05]: 半径方向の有効応力。単位は kPa。正の値が入力されたときのみ、EP でセル圧を制御することで、円周方向の有効応力を設定された値に保ちます（0 以下の値が入力された場合、EP の出力圧力は変化しません）。逆に、非排水三軸試験を行う場合には、必ず 0 を入力してください。

2.7.6 Control from file - 4: Creep

Control No.4: Creep は、クリープを行うコントロールです。主に、Control No.5: Linear Stress Path Loading を用いて圧密した後に、二次圧密を完了させるために使用されます。

以下のパラメータが設定できます。

- Args[00]: ターゲットとなる軸差応力。単位は kPa。
- Args[01]: 軸差応力が現在の値とどれだけズレたときに、Args[02] で設定したモーターの回転速度となるか。単位は kPa。
- Args[02]: モーターの最大回転速度。単位は RPM。
- Args[03]: 継続時間。単位は分。指定した時間が経過した場合、次の Step No. に進みます。
- Args[04]: 半径方向の有効応力。単位は kPa。正の値が入力されたときのみ、EP でセル圧を制御することで、円周方向の有効応力を設定された値に保ちます（0 以下の値が入力された場合、EP の出力圧力は変化しません）。逆に、非排水条件の場合には、0 を入力してください。

実際の運用では、Args[03] には、十分長い時間を設定しておくことが多いです。これは、圧密→クリープ（二次圧密）の後、軸圧縮に進む前に、供試体のレファレンス値の更新とひずみの初期化の操作を手動で行う必要があるため、勝手に次の Step No. に進まないようにするためです。

また、Args[01] と Args[02] は比例制御に関連するパラメータです。軸差応力のエラー値（目標値と現在の値の差）とモーターの回転方向・速度の関係については、Pre-Consolidation を参照してください。

2.7.7 Control from file - 5: Linear Stress Path Loading

Control No.5: Linear Stress Path Loading は、指定した応力経路に沿った載荷を行う始点となる応力状態と、終点となる応力状態を指定することで、応力経路上でその 2 点を結ぶ直線に沿った載荷を行うコントロールです。一般に圧密時に使用されます。

以下のパラメータが設定できます。

- Args[00]: 初期の軸（鉛直）方向の有効応力。単位は kPa。
- Args[01]: 初期の半径（水平）方向の有効応力。単位は kPa。
- Args[02]: 最終的に目標とする軸（鉛直）方向の有効応力。単位は kPa。

- Args[03]: 最終的に目標とする半径（水平）方向の有効応力。単位は kPa。
- Args[04]: セル圧を変化させる速度。単位は kPa/min。Args[01] から Args[03] に向かって、指定した速度でセル圧が増加または減少します。
- Args[05]: 軸方向の有効応力が現在の目標値とどれだけズレたときに、Args[06] で設定したモーターの回転速度となるか。単位は kPa。
- Args[06]: モーターの最大回転速度。単位は RPM。

なお、Args[05] と Args[06] は比例制御に関連するパラメータです。有効応力のエラー値（現在の目標値と現在の実際の値の差）とモーターの回転方向・速度の関係については、Pre-Consolidation を参照してください。

第 3 章　デベロッパーマニュアル

この章ではハードウェア設計者、ソフトウェア開発者向けの情報を記載します。DigitShowModbus のビルド方法や、専用の ModbusRTU ボードについて、装置開発に必要な情報が含まれています。

3.1　起動時変数

3.1.1　ModbusRTU AD/DA Board Version

完全表記が `--version=` 、短縮表記が `-v=` です。

使用している ModbusRTU ボードのバージョンを指定するための引数です。デフォルトで Trio（v1）ボードを使用します。

“100” を指定することで Trio（v1）ボードを使用します。

“200” を指定することで Quartet（v2）ボードを使用します。

“300” を指定することで Yamanin（v3）ボードを使用します。

大まかな動作の違いを以下に示します。

表 3.1　各ボードの機能概要

BoardName	Version	AD	DA	AD Refresh	Other
Trio（v1）	100	16	6	100 ms	AD/DA 要求後 1 ms Sleep
Quartet（v2）	200	16	8	100 ms	AD/DA 要求後 1 ms Sleep
Yamanin（v3）	300	16	8	10 ms	App 起動時 GreenLED が点灯

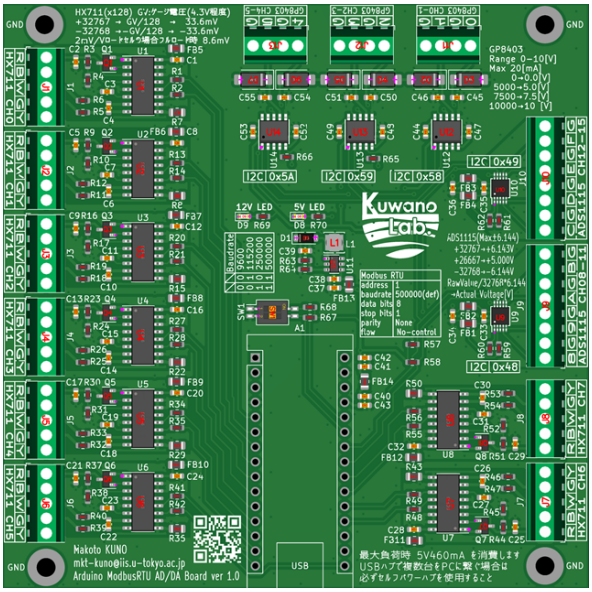

図 3.1　Trio（v1）ボード

ボードは緑で、上部 DA 出力コネクタが 3 個（6ch）なのが特徴です。

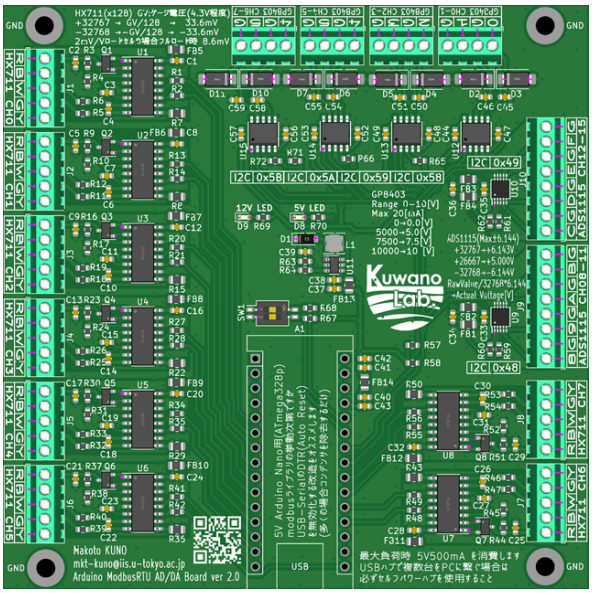

図 3.2　Quartet（v2）ボード

ボードは緑で、上部 DA 出力コネクタが 4 個（8ch）なのが特徴です。

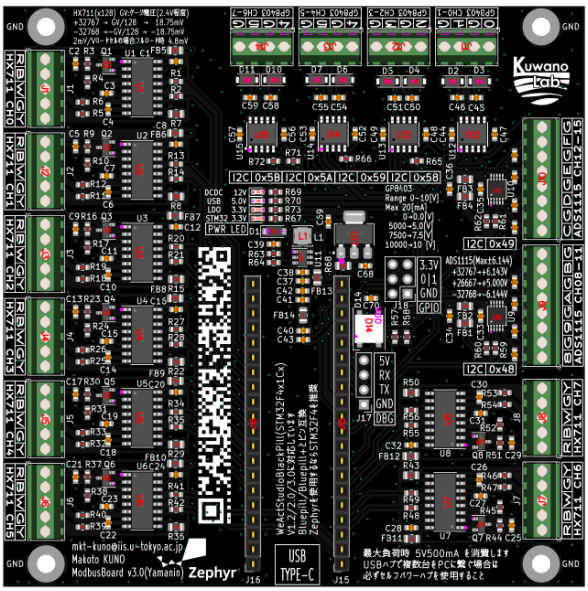

図 3.3　Yamanin（v3）ボード

ボードが黒く、Zephyr のロゴが入っており、NeoPixel LED が搭載されていること、DA 出力の付近の素子が小型化されていることが特徴です。

3.1.2　ModbusRTU COM Port

完全表記が `--port=`、短縮表記が `-p=` です。

ModbusRTU ボードとの通信に使用する COM ポートを指定するための引数です。`--port=COM10` や `-p=COM6` のように指定します。Trio（v1）・Quartet（v2）では、製造時期や互換ボードかどうかによって、FT232 や CH340 など、何かしらの USB シリアル変換 IC を使用しているため、デバイスマネージャー等で確認しましょう。また、動作際してはドライバが必要になります。ArduinoIDE をインストールするか、多くの場合 WindowsUpdate の Additional Update でドライバがインストール可能です。Yamanin（v3）では、USB-CDC（ACM）を使用した高速・高信頼のシリアル通信を採用しており、基本的に Windows 標準のドライバが利用可能なため、追加のドライバインストールは不要です。

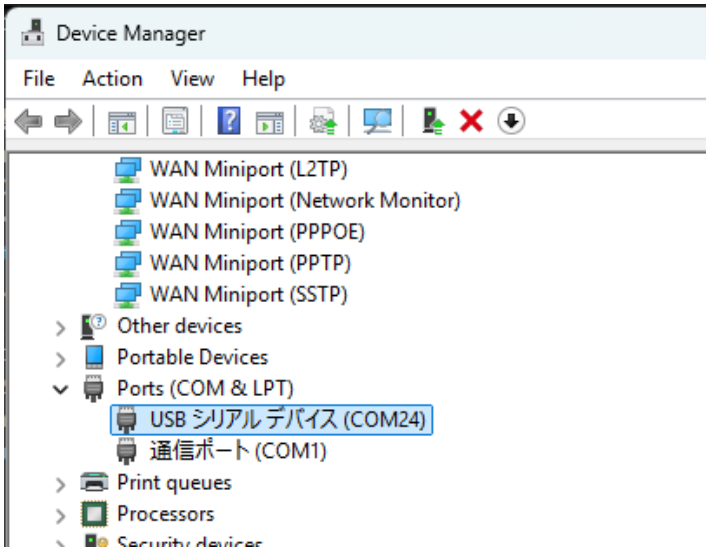

図 3.4　COM ポートの例

3.1.3　ModbusRTU Baudrate

完全表記が `--baudrate=` 、短縮表記が `-b=` です。ModbusRTU ボードとの通信速度を指定するための引数です。基本的に設定する必要はありません。Trio・Quartet・Yamanin ではないオリジナルの Modbus ボードを使用する際に利用してください。

3.1.4　フォント

完全表記が `--font=` 、短縮表記が `-f=` です。フォントの指定は、 `--font="MS Gothic"` `--font="Source Han Code JP"` のように指定します。デフォルトは “Lucida Sans Typewriter” ですが、他のフォントを指定することも可能です。見やすさもありますが、Windows の環境によっては存在しないフォントを使用しようと、アプリケーションが試行する可能性がありますので、環境ごとに確認してください。

3.1.5　動作モード

完全表記が `--mode=` 、短縮表記が `-m=` です。

“0” もしくは “motor” を指定することで生研式モータ動作モードで起動します。

“1” もしくは “torsional” を指定することで生研式ねじり試験動作モードで起動します。

基本的に Motor 動作モードで起動することをお勧めします。ねじりモードは開発中です。Debug ビルドで試用して、安全性を確認してから長期運用してください。

3.1.6　クラッチ&モータ動作電圧

完全表記が `--invert_motor_enable=` 、短縮表記が `-ime=` です。

完全表記が `--invert_motor_direction=` 、短縮表記が `-imd=` です。

“True” もしくは “true”、“1” を指定することで極性が反転します。デフォルト状態はソースコードの確認をしてください。大きな変更が加わっていない限りは、“モータ ON” が “5.0 [V]”、“モータ OFF” が “0.0 [V]”。同じように “モータ UP” が “5.0 [V]”、“モータ DOWN” が “0.0 [V]” です。コードを検索する場合は下記のような記述を探してください。

CDigitShowModbus.cpp より抜粋

```
#define AO_DEFAULT_VLT_MOTOR_ON     (5.0f)          // Voltage of Axial Motor ON
#define AO_DEFAULT_VLT_MOTOR_OFF    (0.0f)          // Voltage of Axial Motor OFF
#define AO_DEFAULT_VLT_MOTOR_UP     (5.0f)          // Voltage of Axial Motor UP
#define AO_DEFAULT_VLT_MOTOR_DOWN   (0.0f)          // Voltage of Axial Motor DOWN
```

3.1.7 Web サーバー機能

完全表記が `--listen=` 、短縮表記が `-l=` です。Web サーバー機能を有効にするための引数です。有効化された場合、実行ファイルと同一ディレクトリにある `www` フォルダ内のファイルをルートとして提供します。引数では公開範囲とポートを指定します。例として、そのパソコンからのみアクセス可能で、通常の HTTP ポート 80 で公開する場合は、`--listen="localhost:80"` と指定します。他のパソコンからもポート 80 をアクセス可能にする場合は、`--listen="0.0.0.0:80"` と指定します。

3.2 VisualStudio 2022 環境構築

Visual Studio 2022 でのビルド方法を記載しています。2025 年移行のバージョンについては、Google で情報を検索し、逐次正しいビルド依存関係を選択してください。

3.2.1 VisualStudio 2022 のインストール

Microsoft から Community 2022 のインストーラを取得し実行します。無料版（Community）の最新版をネットから確実にダウンロードしてください。

図 3.5　Microsoft 公式ホームページ

3.2.2　必要なコンポーネントのインストール

まず、“ワークロード” から “C++によるデスクトップ開発” を選択する。ここでそのままインストールしないこと。“個別のコンポーネント” より、追加のパッケージのインストールが必須です。Linux 向け `C++` や `C#` など似た表記が多いので注意すること。

図 3.6　ワークロード選択画面

次に “個別のコンポーネント” より “Windows 11 SDK” の最新版を追加します。検索欄に “Windows 11 SDK” と入力すると探しやすくて良いでしょう。探した結果、すでに選択されている場合は、最新のものを 2 つ選択するようにしてください。

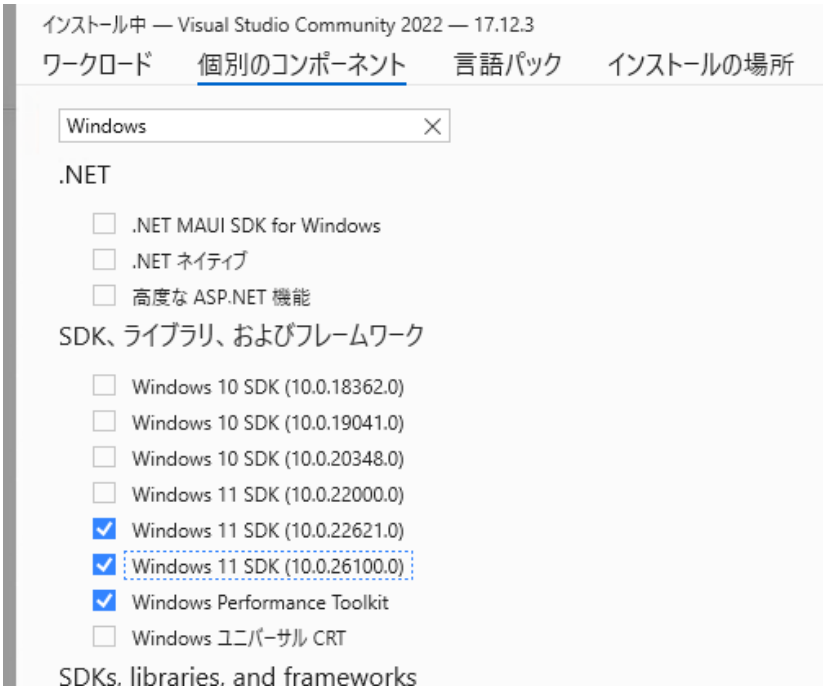

図 3.7　Windows 11 SDK の最新版の選択

次に最新の MFC 用ビルドツールを追加します。検索欄に “C++ MFC” と入力すると探しやすくて良いでしょう。使用しているプラットフォームに合わせて（多くの場合 “x86 および x64”）最新の C++ MFC ビルドツールを選択してください。VisualStudio2022 であれば、 v143 になっているはずです。Snapsragon が乗った Surface 向けにクロスコンパイルしたい場合などは、ARM64/ARM64EC など、環境に応じたツールもインストールしてください。基本的に “x86 および x64 向け” を選択しておけば間違いないですが、まよったら画像のように全部入れてください。

図 3.8　プラットフォームの選択

最終的に、１～５個程度の追加パッケージが選択されているのを確認し、インストールを開始します。

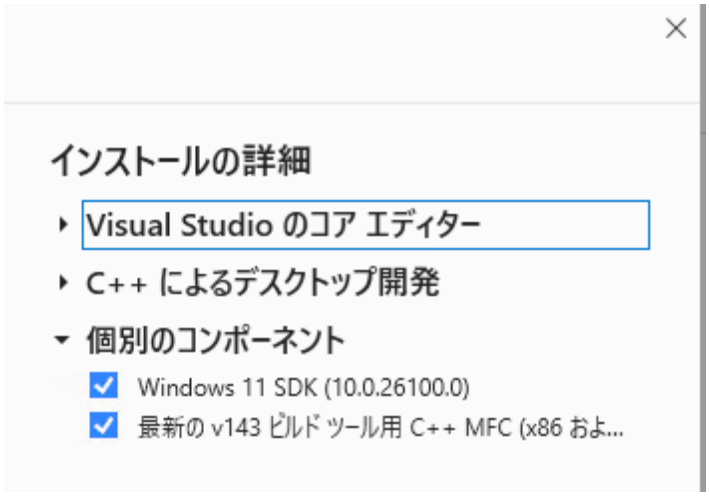

図 3.9　追加のコンポーネント一覧

3.2.3　ソリューションのオープン

VisualStudio のインストールが完了したら、ソリューションファイルを開きます。いくつか方法がありますが、最も簡単なのは VisualStudio を起動した後に “プロジェクトやソリューションを開く” を選択し、“DigitShowModbus.sln” を選択することです。

図 3.10　VisualStudio 起動時初期画面

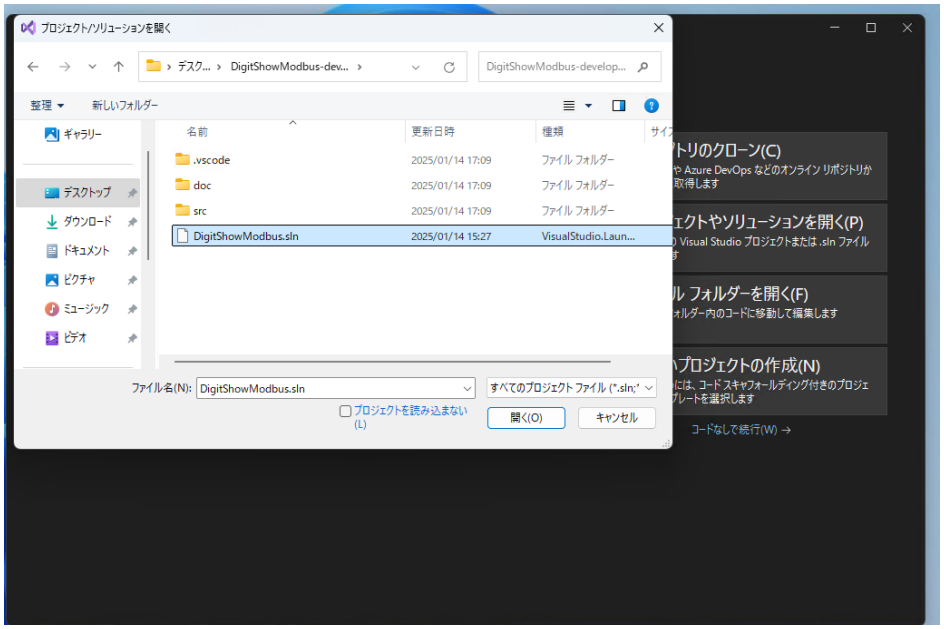

図 3.11　`.sln` ファイルの選択

sln ファイルのダブルクリックでも開けますが、VisualStudio が複数インストールされている場合は、複雑な挙動になる可能性があります。

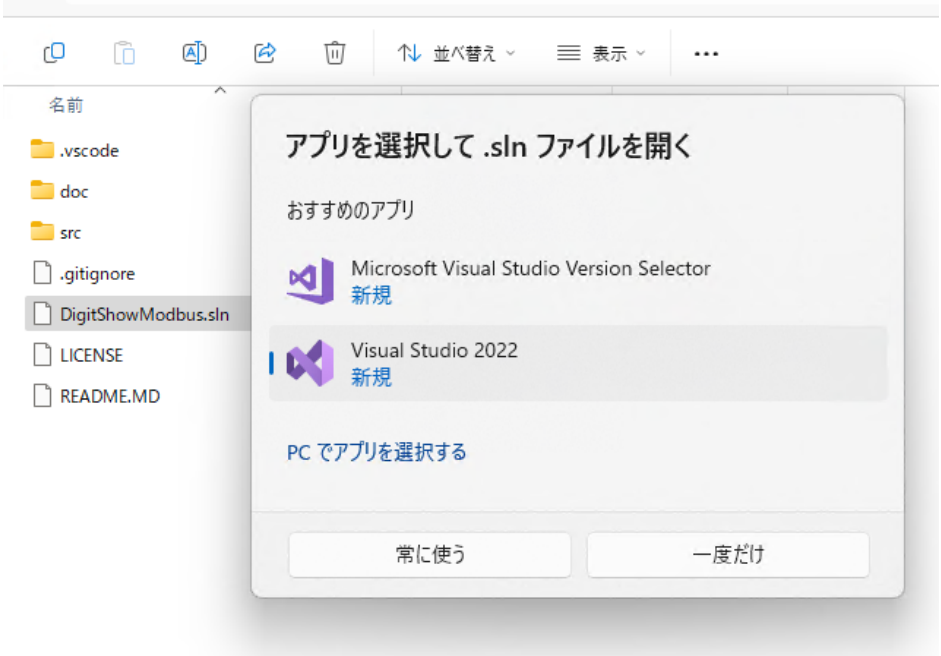

図 3.12　直接選択時の挙動の一例

3.2.4　ソリューションのビルド

コードの追加時や初回運用時などは“Debug”モードでビルドすることをお勧めします。逆にそれ以降では、ログが増えすぎることから“Release”モードでビルドすることをお勧めします。ログの仕組みについては外部ライブラリを使用しているため“spdlog”のドキュメントを参照してください。通常運用時は特別な理由がない限り“x64”の“Release”を選択しておけば間違いはありません。

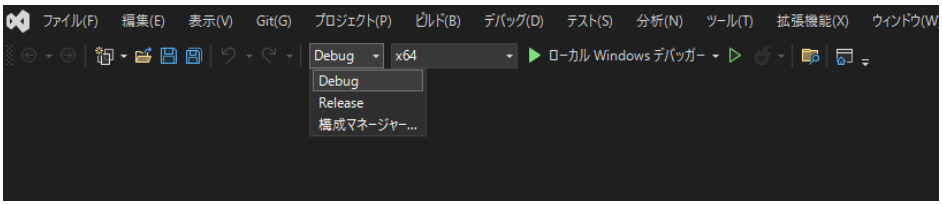

図 3.13　ビルドモードの選択

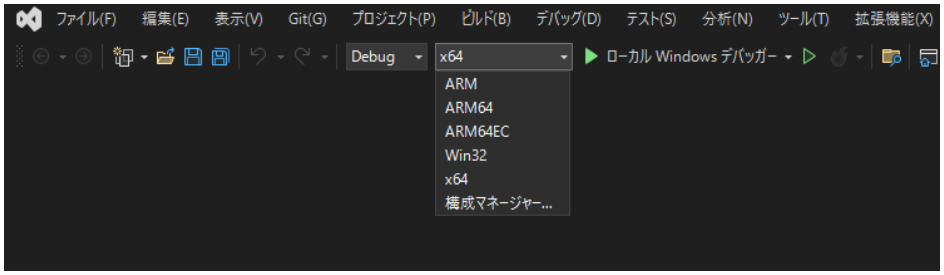

図 3.14　プラットフォームの選択

「なにもしてないのにビルドが通らなくなった」は、よくあることです。ソリューションの“リビルド”もしくは“クリーン”の後に“リビルド”してください。

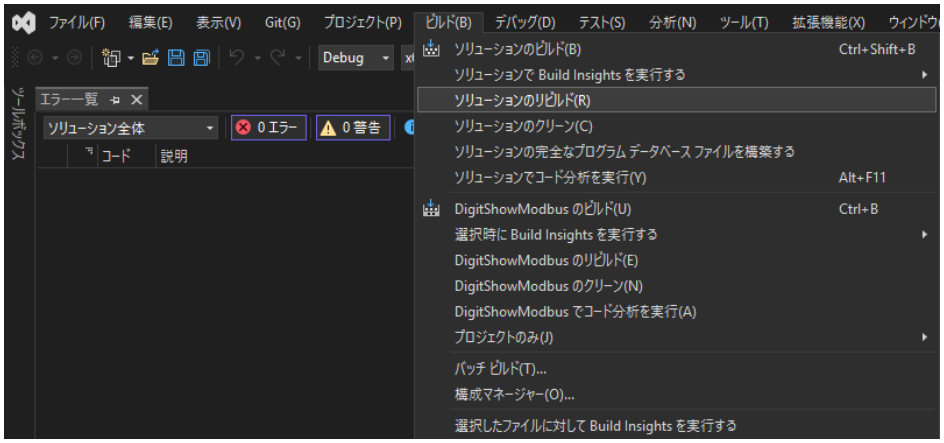

図 3.15　ソリューションのリビルド

3.3　コントロールの追加・修正

`src/DigitShowModbus.cpp` 及び `src/DigitShowModbus.h` を編集しますそもそもの前提知識として、自分にあった `C++` の教材で（なんでもいいです）、ポインタとクラスまで学んでおくことを強くお勧めします。そこまでの知識がないと、何を書いて良いか分からなく可能性が非常に高いです。Python の事前知識があればとっつきやすと思いますが、C++は Python とは異なる部分が多いです。正しくない記述に寄ってプログラムがクラッシュしたとしても、Python のようにエラー箇所がわかりやすく提示されません。

加えて、初心者レベルで良いので、Git 習得を強く推奨します。Git を使っていないと、コードのバックアップや、間違った修正を戻すことが非常に困難です。Git を使用していな場合、コードの変更点がわからなくなったプログラムを持って来ても、救いませんし、救えません。上記の点に留意し、勉強したうえでコントロールを修正してください。

多くの人が修正を加えたいのは、`CDigitShowModbusDoc::ControlMain`での処理になると思います。以下に IIS モーターモードでの処理の例を示します。

ID が 1 の場合は、`MOTOR_Control_PreConsolidation`を呼び出します。これは見ての通り、“Control from File” からではない先行圧密用の関数です。

ID が 15 の場合は、`MOTOR_Control_FileCtrl_MonotonicAxialLoadingStress`などの関数を呼び出します。これがメイン機能の “Control from File” で実行されるプログラム群です。

CDigitShowModbusDoc.cpp より抜粋

```
1          void CDigitShowModbusDoc::MOTOR_ControlMain() {
2      spdlog::trace("{}", __FUNCTION__);
3
4      auto ctx = GetCDSBContext();
5      if (ctx->Flag.control == FALSE) return;
6
7      auto pCD = &ctx->Control.Data[ctx->Control.id];
8
9      spdlog::debug("MOTOR_ControlMain␣Ctrl␣ID:{}", ctx->Control.
       id);
10
11     switch (ctx->Control.id)
12     {
13         case 1: MOTOR_Control_PreConsolidation(pCD); break;
14         case 15:
15         {
16             auto ctrl_step_cc = ctx->Control.ctrl_num_array[ct
               x->Control.current_step_num].load();
17             spdlog::info("ControlMain:␣Control␣from␣file.");
18             spdlog::debug("ctx->Control.current:{}", ctx->Contr
               ol.current_step_num);
19             spdlog::debug("ctx->Control.step_num[ctx->Control.c
               urrent]:{}", ctrl_step_cc);
20             if (ctrl_step_cc == 0) {
21                 // No Control
22                 MOTOR_Set_Speed(0.0f);  // RPM->0
23             }
24             if (ctrl_step_cc == 1) MOTOR_Control_FileCtrl_Monot
               onicAxialLoadingStress();
25             if (ctrl_step_cc == 2) MOTOR_Control_FileCtrl_Monot
               onicAxialLoadingStrain();
26             if (ctrl_step_cc == 3) MOTOR_Control_FileCtrl_Cycli
               cAxialLoadingStress();
27             if (ctrl_step_cc == 4) MOTOR_Control_FileCtrl_Cycli
               cAxialLoadingStrain();
28             if (ctrl_step_cc == 5) MOTOR_Control_FileCtrl_Cree
               p();
29             if (ctrl_step_cc == 6) MOTOR_Control_FileCtrl_Linea
               rStressPathLoading();
30
31             AioUpdateOut();
32             break;
```

```
33          }
34          default:
35              AioUpdateOut();
36              break;
37      }
38  }
```

3.4 Modbus ボードについて

TBD...

3.4.1 Modbus TCP/RTU/ASCII

Modbus とは、1979 年に Modicon（現在の Schneider Electric）が開発した通信プロトコルです。Modbus は、RS-232C や RS-485 などのシリアル通信を使用する Modbus RTU/ASCII と、TCP/IP を使用する Modbus TCP の 2 つのバージョンがあります。TCP/IP を使用するには全台が DHCP 対応でオンラインなネットワークに属しているか、静的 IP で設定されたクローズドネットワークに属している必要があります。今日日、多くの大学、会社でネットワークの管理が厳しくなっているため、本プロジェクトでは物理接続で動作が可能な Modbus RTU/ASCII を選択しました。

次に RTU と ASCII との違いですが、どちらも同じシリアル転送、をベースとした技術です。RTU はバイナリ形式で、ASCII はテキスト形式で通信します。要は人間が通信内容を見たときに分かるか（テキスト形式）どうか、という感じです。RTU はバイナリ形式で通信するため、通信速度が速く、ASCII はテキスト形式で通信するため、通信速度が遅いです。本プロジェクトでは将来的に通信レートが向上することに対応するため、RTU を選択しました。

3.4.2 シリアル通信

最近では Arduino、また多くの電子天秤や制御機器で “Serial” や “シリアル通信”、“UART”、“RS-232C” などの用語を見かけると思います。厳密なことはおいておいて、これらの単語は同じ通信方式を指すと言っていいでしょう。TX（送信）と RX（受信）これらの対によって、ホスト（Windows など）とデバイス（Arduino など）が通信を行います。通信内容は基本的に文字列通信ですが、バイナリ通信も可能です。多くの場合文字列通信が使用されるというだけです。

ここでシリアル通信の種別について、RS-232C、RS-422、RS-485、USB-CDC などがあります。

RS-232C は古い規格で、シングルエンド通信のため最大通信速度が遅いですが、通信距離が長いです。1 対 1 の通信が可能で、通信線には最低限 GND、TX+、RX+の 3 本が必要です。

RS-422 は RS-232C の改良版で、差動通信かつ TX, RX が別れているため通信速度が速く、通信距離も長いです。1 対 1 の通信が可能で、通信線には最低限 GND、TX+、TX−、RX+、RX− の 5 本が必要です。GND を除いた 4 本て大丈夫な、絶縁通信もあります。

RS-485 は RS-422 の改良版で、差動通信で TX/RX をまとめた形のため、マルチポイント通信が可能です。1 対多の通信が可能で、通信線には最低限 GND、D+、D− の 3 本が必要です。GND を除いた 2 本て大丈夫な、絶縁通信もあります。

USB-CDC はシリアル通信を全部 USB に乗せて通信するための規格です。USB-CDC は USB のため、通信速度が速く、通信距離も長いです。1 対 1 の通信が可能で、通信線には最低限 GND、D+、D− の 4 本が必要です。

例えば Arduino Uno/Nano では、PC から Arduino ボードに搭載された FT232/CH340 などの USB シリアル変換 IC までは USB-CDC で USB 通信し、その後は FT232/CH340 などのシリアル変換 IC から RS-232C でマイコンと通信を行います。後述する Yamanin（v3）では PC からマイコンボードまで直接 USB が通信線として使用されており、ロスや遅延、無駄な処理なくシリアル通信が行われます。

3.4.3 シングルエンド信号と差動信号

このような、接続できる機器の数、伝送距離、伝送速度、ノイズ耐性、電源電圧などの要素を考慮して、通信方式を選択する際に出てくるワードが、シングルエンド信号と差動信号です。シングルエンド信号は、信号線と GND の間で信号を送る方式です。例えば Arduino 等の場合、5 V で動作していれば、信号線が 3 V 以上で HIGH、1 V 以下で LOW というように、GND との電圧差で信号を判断します。これは同時に外部から瞬間的に 3V 程度のノイズが乗った際に、もしくはアースやグランドが不安定になると誤動作を起こす可能性があります。なのでテレビのリモコンなんてのは、なんとなくの GND で動作するシングルエンド信号で動作しているため、リモコンをテレビの前に持っていかないと反応しない、方向が悪いと反応しないということがあります。他には液晶の D-SUB なんてのは、シングルエンド信号で動作している上、あろうことかアナログ信号なので、端子の刺さりが甘かったり、端子を触ってノイズが乗ると画面が大きく乱れることがあります。

差動信号は、信号線と信号線の間の電圧差で信号を送る方式です。例えば RS-485 などの場合、TX+と TX− の間で信号を送ります。例えば TX+（例：+2 V）と TX−（例：−2 V）電圧差が 4V 以上なら HIGH、例えば TX+（例：-2V）と TX−（例：+2V）電圧差が-4V 以下なら LOW というように、信号線間の電圧差で信号を判断します。これは先ほどと違い、外部からノイズが乗っても、信号線間の電圧差が変わらない限り、誤動作を起こしにくいです。身の回りだと、有線 LAN や USB、HDMI、DVI などが差動信号で通信しています。有線 LAN は 4 対の通信線。USB 1.0/2.0 は 2 対の通信線です。USB 3.0 は高速化の更に通信対を足しています。要は、高速・安定の通信をするには差動信号が良い、ということです。

3.4.4 フィールドネットワーク

フィールドネットワークとは、工場や建物内の機器やセンサー、アクチュエーターなどを接続するためのネットワークです。我々の用途も、砂・水・油・薬品を利用するため、フィールドネットワークを使用している、と言っていいでしょう。フィールドネットワークには、Profibus、DeviceNet、EtherCAT、CC-Link、Modbus、CAN、PROFINET、EtherNet/IP、POWERLINK、

Sercos、IO-Link、AS-Interface、HART、Bluetooth、Zigbee、Thread などがあります。あまりに多すぎますが、この殆どがライセンス絡みで有料で、ホスト側のデバイスも高価である場合が多いです。この中で比較的安価に利用可能なのが、CAN、Modbus、Bluetooth あたりです。

CAN は車載向けで多く使われてきた実績があり、車体全体のアースを GND とすることで、1 対の差動線のみで通信が可能で、一本の線が切れても通信が可能、加えて複数デバイスを接続することが可能です。この技術は 3D プリンタにも応用され、多くのフラッグシップ 3D プリンタは、高速に動作させたいホットエンド部分につながる線を、通信速度と信頼性を保ったまま減らすため、CAN を採用しています。我々の用途の場合、CAN の通信プロトコルが複雑であることと、1 回あたりの通信（メッセージ）長が短い事が問題になります。

Bluetooth は無線のため、否が応でも予期しない遅延が、かつ電波環境依存で安定しない遅延時間が発生します。計測レートが 1 sample/sec 程度ならば気になりませんが、高速計測を行う場合は、遅延が問題になるので、今回は考慮しませんでした。

残されたのが Modbus、とくに RTU でした。Modbus RTU は、基本的には RS-485 を使用するため、差動信号で通信が行われ、信頼性が高いです。また、RS-485 はマルチポイント通信が可能で、複数のデバイスを接続することが可能です。しかし、RS485 ではなく USB-CDC を使用すれば、1 対 1 通信の限定になりますが、高速通信を行うことが可能です。Arduino などをデバイスとするのであれば、ライブラリも多く存在し、仕様もオープンなため、開発がしやすいというメリットもあります。また Modbus TCP という上位規格があるため、物理接続がいよいよ廃れ始めた頃に TCP/IP への以降もスムーズに進むと考えられます。諸々の理由により、本プロジェクトでは Modbus RTU を採用しました。

3.4.5 Modbus ボード v1（Trio）/v2（Quartet）

- 開発コード：Trio（v1）/Quartet（v2）
- マイコン：Arduino Nano（ATmega328p）
- プラットフォーム：Arduino
- 通信方式：USB シリアル
- 通信プロトコル：Modbus RTU
- 通信速度：38 400 bps
- 入力-HX711：1 ch/ic, 計 8 ch, 16 bit 精度, 128 倍ゲイン,10 Hz 動作
- 入力-ADS1115：4 ch/ic, 計 16 ch, 16 bit 精度, 64 Hz 動作（オプションで 128 Hz）
- 出力-GP8403：2 ch/ic, 計 6/8 ch, 12 bit 精度, オンデマンド動作

3.4.6 Modbus ボード v3（Yamanin）

- 開発コード：Yamanin（v3）
- マイコン：STM32F411CE（BlackPill）
- プラットフォーム：Zephyr RTOS

- 通信方式：USB シリアル
- 通信プロトコル：Modbus RTU
- 通信速度：38 400 bps（USB ダイレクトのため、何でも良い）
- 入力-HX711：1 ch/ic, 計 8 ch, 16 bit 精度, 128 倍ゲイン, 80 Hz 動作, 単純加算平均（8 サンプル）
- 入力-ADS1115：4 ch/ic, 計 16 ch, 16 bit 精度, 128 Hz 動作
- 出力-GP8403：2 ch/ic, 計 8 ch, 12 bit 精度, オンデマンド動作

3.4.7 Modbus ボード v4（Milia）

- 開発コード：Milia（v4）
- マイコン：Raspberry Pi Pico（RP2040）
- プラットフォーム：Arduino
- 通信方式：USB シリアル
- 通信プロトコル：Modbus RTU
- 通信速度：38 400 bps（USB ダイレクトのため、何でも良い）
- 入力-HX711：1 ch/ic, 計 8 ch, 16 bit 精度, 128 倍ゲイン,10 Hz 動作
- 入力-ADS1115：4 ch/ic, 計 16 ch, 16 bit 精度, 64 Hz 動作（オプションで 128 Hz）
- 出力-GP8403：2 ch/ic, 計 6/8 ch, 12 bit 精度, オンデマンド動作

3.5 各 IC の性能と説明

3.5.1 ひずみ入力 IC：HX711

24 bit 精度の ADC を搭載した、内臓で 128 倍のゲインを持ち、10 Hz もしくは 80 Hz のサンプリングレートを持った高精度なアナログデジタルコンバータです。ゲージ電圧 1〜5 V 程度の、フルブリッジ式のひずみセンサーの接続に適しています。出力される値は本来であれば、符号付き 24 bit 精度ですが、ノイズ等により実質的な性能が 16 bit 程度のため、本プロジェクトでは符号付き 16 bit 精度（int16_t/i16）として扱っています。

動作電圧が 2.048 V の場合の物理値と、ゲイン済み電圧、ゲイン前電圧、の関係はを表すと以下の表のようになります。

HX711 の int16_t（i16）物理値から mV/V、もしくは με への定格出力値への変換については、以下の式を使用してください。

表 3.2　HX711 の物理値と換算式

物理値	物理値	差圧（128 倍）[mV]	差圧 [mV]	電圧 [mV]	出力（E=2）[mV/V]
32767	0x7FFF	2048	16	8	3.906
16384	0x4000	1024	8	4	1.953
8192	0x2000	512	4	2	0.977
4096	0x1000	256	2	1	0.488
2048	0x0800	128	1	0.5	0.244
0	0x0000	0	0	0	0
-4096	0xF000	-256	-2	-1	-0.488
-8192	0xE000	-512	-4	-2	-0.977
-32768	0x8000	-2048	-16	-8	-3.906

$$\Delta e = \frac{E}{4} K \varepsilon \quad [\mathrm{V}] \tag{3.1}$$

$$\varepsilon = \frac{4\Delta e}{KE} \quad [\varepsilon] \tag{3.2}$$

$$\frac{\Delta e}{E} = \frac{\mathrm{Raw_{i16}}}{\mathrm{INT16\_MAX} \times \mathrm{HX711\_GAIN} \times \mathrm{DIFF}} \tag{3.3}$$

$$= \frac{\mathrm{Raw_{i16}}}{32767 \times 128 \times 2} \quad [\mathrm{V}] \tag{3.4}$$

$$\mu\varepsilon = \frac{\mathrm{Raw_{i16}}}{\mathrm{INT16\_MAX} \times \mathrm{HX711\_GAIN} \times \mathrm{DIFF}} \times K \times 1\mathrm{E}6 \tag{3.5}$$

$$= \frac{\mathrm{Raw_{i16}}}{32767 \times 128 \times 2} \times 2.0 \times 1\mathrm{E}6 \quad [\mu\varepsilon] \tag{3.6}$$

Δe : 負荷時の出力電圧　[V] (3.7)

E : 印加電圧　[V] (3.8)

K : ゲージ率(殆どの場合2.0) (3.9)

ε : ひずみ (3.10)

3.5.2　汎用入力 IC：ADS1115

シングルエンド入力と差動入力が可能な 16 bit 精度の ADC を搭載した、I2C 接続のアナログデジタルコンバータです。最大 CH はシングルエンドジ 4 つで、差動入力で 2 つです。入力電圧範囲は最大で −6.144〜+6.144 V で、計測レートは 8〜860 sample/sec まで設定可能です。計測レートを上げすぎると精度が下がり、計測レートを下げすぎるとチャンネル間の遅延が大きくなるため、使用される想定レートに合わせ、本プロジェクトでは 7〜8 ms/1ch（32 ms/4 ch） 程度の計測レートで使用しています。

表 3.3　ADS1115 の物理値と換算式

物理値	物理値	電圧
32767	0x7FFF	6.144
16384	0x4000	3.072
8192	0x2000	1.536
4096	0x1000	0.768
0	0x0000	0.000
-4096	0xF000	-0.768
-8192	0xE000	-1.536
-32768	0x8000	-6.144

ADS1115 のの int16_t（i16）物理値の換算表、換算式は以下の表のようになります。

$$V = \frac{\mathrm{Raw_{i16}}}{\mathrm{INT16\_MAX}} \times \mathrm{ADS1115\_RANGE} \tag{3.11}$$

$$= \frac{\mathrm{Raw_{i16}}}{32767} \times 6.114 \quad [\mathrm{V}] \tag{3.12}$$

3.5.3　電圧出力 IC： GP8403

12 bit 精度の DAC を搭載した、I2C 接続のデジタルアナログコンバータです。最大 CH は 2 つで、出力電圧範囲は 0〜10.0 V です。この IC のみ入出力の指定 8bit の倍数にうまく収まらないため、DFRobot 社の GP8403 ライブラリ互換の出力値指定方法となっています。具体的には、指定値が uint16_t の範囲を取り、mV オーダーで出力電圧を指定します。

GP8403 の uint16_t（u16）物理値の換算表、換算式は以下の表のようになります。

表 3.4　GP8403 の物理値と換算式

物理値	物理値	出力電圧 [mV]	出力電圧 [V]
10000	0x2710	10000	10.0
5000	0x1388	5000	5.0
2000	0x07D0	2000	2.0
1000	0x03E8	1000	1.0
100	0x0064	100	0.1
0	0x0000	0	0.0

3.5.4 Arduino Nano

Arduino Uno R3 の小型版で、ATmega328P（一部、ATmega168）を搭載したマイコンボードです。シリアル変換チップを搭載しており、シリアル通信が可能です。8 bit の CPU 命令、16 MHz のクロック、32 KB のフラッシュメモリ（実行ファイル用ストレージ）、2 KB の RAM（メモリ）、1 KB の EEPROM（データ保存用ストレージ）を搭載しています。Trio、Quartet の制御用マイコンとして使用されていますが、HX711 を 80 Hz 駆動で加算移動平均を適用しようとしたところ、処理が追いつかないため、Yamanin に移行する際に使用されなくなりました。8 bit マイコンはその性質上、8 bit の加減算が 1 クロックで、16 bit の加減算が 2 クロック、32 bit の加減算が 4 クロックかかるため、処理が遅くなります。特に剰余算の場合は、更に遅くなるため、単純移動平均であっても処理が間に合わなくなったものと思われます。

3.5.5 WeAct Studio BlackPill STM32F411CE

STM32F411CE を搭載したマイコンボードで、ARM Cortex-M4 の 32 bit マイコンです。100 MHz のクロック、512 KB のフラッシュメモリ、128 KB の RAM を搭載しています。先述の通り、HX711 の 80 Hz 対応、加算移動平均処理に対応するためにマイコンの変更が行われました。また、STM32F411CE は、ZephyerRTOS を使用することによってダイレクトな USB-CDC が利用可能のため、シリアル変換チップが不要で、USB ケーブルを直接接続することが可能です。32 bit マイコンのため、8〜32 bit までの加減算が 1 クロックで行えるため、処理が高速になります。多くの変数が 16 bit 以上であるため、これは強力な高速化になります。加えて、CPU の動作周波数も 6 倍以上であるため、その分処理速度が向上し、パソコンからの処理を遅延なく行えることが出来るようになります。

3.5.6 Raspberry Pi Pico（RP2040）

Raspberry Pi Pico は、RP2040 を搭載したマイコンボードで、ARM Cortex-M0+の 32 bit マイコンです。125 MHz のクロック、2 MB のフラッシュメモリ、264 KB の RAM を搭載しています。RP2040 は、Arduino プラットフォームで使用することができ、Arduino Nano と同様に USB シリアル通信が可能です USB CDC-ACM を使用することにより、シリアル変換チップが不要で、USB ケーブルを直接接続することが可能です。ZephyerRTOS を使用していないため、プロジェクトが簡潔にまとまっています。コアが 2 つあるため、独自のルーチンやユーザーアプリケーションを並列に動作させることが可能です。RP2040 本体の ADC にはエラッタがあり、8 bit の精度しかないため、使用しないことをおすすめします。

3.6 Web サーバー機能

3.6.1 概要

Web サーバー機能を ON にすると、実行バイナリと同一ディレクトリにある www フォルダを静的コンテンツのルートディレクトリとして、Web サーバーが起動します。多くのブラウザでは、`index.html` を自動的に読み込むため、`www/index.html` を作成しておくと、ブラウザでアクセスした際に自動的に表示されます。DigitShowModbus では、brotli を利用した圧縮を透過的（自動）で行うため、よほどの理由がない限りは、開発者は圧縮済みのコンテンツ（gz、zip など）を作成する必要はありません。また、Modbus ボードの計測データを取得したり、CSVTK 画像を取得したり、制御コマンドを送信したりするための WebAPI を用意しています。これらを組み合わせると、独自の HTML や CSS、JavaScript を使用して、リモート監視・可視化用の Web アプリケーションを作成することができます。

3.6.2 計測データの取得 API（v1）

`/v1/` に GET リクエストを送ると、計測データを取得できます。`localhost:80` 運用時であれば `http://localhost/v1/` でアクセスできます。計測データは JSON 形式で返されます。以下は、計測データの例です。

計測データ JSON 一部抜粋


```
{
  "control": {
    "cycle_state": 0,
    "mode": 0,
    "num_cyclic": 0,
    "stepctrl": {
      "args": { "00": 0.0, "01": 0.0, "02": 0.0, "03": 0.0,
      "04": 0.0 },
      "ctrl": 0,
      "current_step": 0
    },
    "type": 0
  },
  "current": {
    "e_p": 1.7425289154052734,
    "e_sa": 5.226467609405518,
    "e_sr": 0.0005594491958618164,
    "ea": 0.0,
    "er": 0.0050961971282958984,
    "ev": 0.010185915976762772,
    "p": 0.0,
    "q": 5.225908279418945,
```

```
    "specimen": {
      "area": 1963.29541015625,
      "diameter": 49.99745178222656,
      "height": 100.0,
      "volume": 196329.546875
    }
  },
  "flag": { "control": false, "cyclic": false, "save_data": fal
  se, "set_board": true },
  "output": { "00": { "label": "00:Motor ON/OFF", "value": 0.0
  } "01": {"label": "01:Motor UP/DWN", "value": 0.0 } },
  "param": { "00": { "label": "00:q(kPa)", "value":
  5.225908279418945 }, "01": { "label": "01:p'(kPa)", "value":
  1.7425289154052734 } },
  "phy": { "00": { "label": "00:Load(N)", "value":
  10.260002136230469}, "01": { "label": "01:ExtDisp(mm)", "valu
  e": 0.0 } },
  "raw": { "00": { "label": "00:LoadCell(i16)", "value": 42.0
  }, "01": { "label": "01:LVDT(i16)", "value": -3.0 } },
  "system": { "color": "#002020" },
  "time": {
    "ctrl_delta_sec": 0.0,
    "ctrl_step_elapsed_sec": 0.0,
    "interval_ms_ctrl": 200,
    "interval_ms_disp": 100,
    "interval_ms_save": 1000,
    "save_elapsed_sec": 0.0
  }
}
```


3.6.3 CSVTK 画像の取得 API（v1）

csvtk 画像は実機で表示されているものと同じものを取得できます。GET リクエストのタイミングによっては、画像更新中のためにエラーが返されることがあります。`/v1/img/chart_a/` に GET リクエストを送ると、CSVTK 画像（左）を取得できます。`/v1/img/chart_b/` に GET リクエストを送ると、CSVTK 画像（右）を取得できます。`localhost:80` 運用時であれば `http://localhost/v1/img/chart_a/` でアクセスできます。

付録 A　LaTeX による本ドキュメントの編集方法

A.1　環境構築

Dev Container によって LaTeX の開発環境をまるごとコンテナ化することで、面倒な LaTeX の環境構築を回避しつつ、本ドキュメントを編集することができます。ただし、依然としてコンテナを動かすための環境構築は必須であり、ここではその方法を説明します。手順さえ踏めば、LaTeX のインストールとは違って、謎のエラーに苦しむこともないはずです。

Dev Container についてのより詳しい解説は Visual Studio Code のウェブサイト（`https://code.visualstudio.com/docs/devcontainers/containers`）を参照してください。ここでの説明はこのサイトの和訳と抜粋にすぎません。

A.1.1　Docker のインストール

A.1.1.1　Windows にインストールする場合

Docker Desktop をインストールしてください。Windows のエディションが Home の場合は WSL2 機能を有効にする必要があります。

A.1.1.2　Windows 上の WSL2（Ubuntu）にインストールする場合

1. 以下 2 つをインストールしてください。具体的な方法はそれぞれのウェブサイトを参照してください。
   - Docker Engine: `https://docs.docker.com/engine/install/ubuntu/`
   - Docker Compose: `https://docs.docker.com/compose/install/linux/`
2. `sudo usermod -aG docker $USER` を実行してください。
3. Ubuntu の再起動をかけてください。

A.1.2　Visual Studio Code（VS Code）のインストール

1. Windows 上に VS Code 本体をインストールしてください。
2. VS Code 上の拡張機能を以下 2 つインストールしてください。偽の拡張機能（マルウェアの可能性があります）に注意してください。
   - Dev Conteiners extension: `https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers`

- Remote Development extension pack: `https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack`

A.1.3 Git の設定

Windows もしくは WSL2（Ubuntu）上に Git がインストールされているのは前提とします。ローカルにユーザー名とメールアドレスが適切に設定されていることを確認してください。

```
git config --global user.name "Your Name"
git config --global user.email "your.email@address"
```

Dev Container では、適切に設定すれば、コンテナの内部でも上手く Git を使うことができます。つまり、ここで使う仮想コンテナは `./doc` に構築されるコンテナですが、そのコンテナは DigitShow Modbus 本体の Git まで見に行ってくれます。よって、VS Code でドキュメントを編集しながら、そのまま push 等が可能です。ここではそのための設定を行います。

詳細は VS Code のウェブサイト（`https://code.visualstudio.com/remote/advancedcontainers/sharing-git-credentials`）にある通りです。「Using a credential helper」というのが楽だと思います。これはつまり GitHub のウェブサイト（`https://docs.github.com/ja/get-started/getting-started-with-git/caching-your-github-credentials-in-git`）を参照せよということなので、これに従ってください。

A.2 作業方法

1. VS Code 上で DigitShow Modbus が格納されたディレクトリを開いてください。
2. Ctrl + Shift + P のショートカットを利用し、「開発コンテナー：コンテナーでフォルダを開く」を選択してください。
3. `./doc` ディレクトリを選択し OK を押してください。
4. 任意の `.tex` ファイルを編集し、保存すると自動でコンパイルが回ります。エディタ右上の▷を押してもコンパイルが回ります。▷の右の、分割画面にルーペがついたボタンを押すと、隣に PDF を参照しながら `.tex` ファイルを編集できます。

A.3 LaTeX を書くときのルール

- 文書の見た目（スタイル）ではなく構造（意味や役割など）を意識してください。あくまで後者が主であり、前者はそれに付随する概念です。例えば、ある部分を太字にしたいと思っても、`\textbf` は使わないでください。太字にしたいのはそこを強調するためであり、そうであれば強調コマンド `\emph` を使うのが理にかなったやり方です。なお、**強調はゴシック体**で定義されています。変更も可能です。

- ダブルクオーテーションマークで括るときは、`"`で括るのではなく`` `` ``と`''`で括るようにするとダブルクオーテーションマークの形が整います。出力例は次の通りです。`"text."` →”text.” `` ``text.'' `` →“text.”
- 好みの問題に近いですが、和文では全角記号を使うと見た目が整うように思います。例えば括弧類（）やコロン：など。和文の中に英数字を含めなければいけない場面では臨機応変に対応しましょう。
- 図表や章にはラベルをつけて参照することができます。ラベルのネーミングは（技術的には）自由ですが、`fig:`, `tab:`, `sec:`のように、何のラベルかを冒頭で示すと便利です。参照する際には`\cref`を使うと、例えば図を参照した際には自動的に「図 1.1」のように、表を参照した際には自動的に「表 1.1」のように出力してくれます（cleveref パッケージ）。
- 単位つきの数値を扱うときには siunitx パッケージの機能を利用すると、数値と単位との間のスペーシングなどのスタイルが整います。例えば 100 kPa は`\qty{100}{\kPa}`とします。あるいは $1 \times 10^5\,\mathrm{kg\,m^{-1}\,s^{-2}}$ は`\qty{1e5}{\kg.\m^{-1}.\s^{-2}}`とします。ほかにも 15〜20 kPa は`\qtyrange{15}{20}{\kPa}`と書けます。
- 数式を書くときは、イタリック体と立体（ローマン体）の使い分けに注意しましょう。例えばイタリック体で ABC と書くと $A \times B \times C$ と解釈されますが、立体で ABC と書くと ABC（一つの数）と解釈されます。また、添え字も h_{init} にようにして立体にすることが多々あるようです。数式モードで立体にするには`\mathrm{}`を使います。
- TikZ パッケージを使うときれいな図が描けます。使い方はオンラインマニュアル（`https://tikz.dev/`）を見てください。
- コードやファイル名、ディレクトリをインラインで書きたいときは`\inlinecode`を定義したので、そちらを使ってください。
- コードブロックもかけます（listings パッケージ）。

DigitShowModbus.cpp

```
1  // DigitShowModbus.cpp :
   アプリケーション用クラスの機能定義を行います。
2  //
3
4  #include "pch.h"
5  #include "framework.h"
6  #include "afxwinappex.h"
7  #include "afxdialogex.h"
8  #include "DigitShowModbus.h"
9  #include "MainFrm.h"
10
11 #include "DigitShowModbusDoc.h"
12 #include "DigitShowModbusView.h"
13
14 #define _USE_MATH_DEFINES
15 #include <math.h>
```

```
16 #include <fstream>
17 #include <filesystem>
18 #include <cstdlib>
19
20 #include "sqlite3.h"
21 #include <spdlog/sinks/rotating_file_sink.h>
22 #include <nlohmann/json.hpp>
23 #include <modbus.h>
24
25 #ifdef _DEBUG
26 #define new DEBUG_NEW
27 #endif
```