G9006 による CPU 間通信

PCL6046 を遠隔制御

ソースファイル解説

NPM 顧客「満足」から「感動」へ。
日本パルスモーター株式会社

目次

1. はじめに

弊社製 LSI の使用事例をご覧頂き、ありがとうございます。

本書は、G9006 を使用した CPU 間通信の例として、Motionnet システムでデータ通信を使用しない PCL6046 の省配線制御を行うためのソースコードの解説をしたものです。

G9006 側のローカル CPU として Raspberry Pi PICO を使用し、PCL6046 の制御も同じ CPU で行います。

1.1 本書の取扱い

① 本書の全部または一部を無断で転載することは、著作権法によって禁止されています。

② 本書の内容については、性能や品質の向上に伴い、将来予告なく変更することがあります。

③ 本書の内容については、万全を期しておりますが、万一不可解な点や誤り、ならびに記載もれ等お気付きの点がありましたら、弊社営業担当へご連絡をお願いいたします。

1.1.1 記号説明

1.1.1.1 負傷レベル

本書では、次のように負傷レベルを定義します。

- 重傷

  失明、けが、火傷、感電、骨折、中毒等後遺症が残るもの、および治療に入院や長期の通院を要するもの。

- 軽傷

  治療に入院や長期の通院が必要ないもの。(上記「重傷」以外)

1.1.1.2 危険レベル

本製品は、運用者の安全を第一に考え、設計されています。しかし、製品の性質上、どうしても取除けないリスクが存在します。本書では、それらのリスクの重大性および危険性のレベルを、「危険」、「警告」および「注意」事項の 3 段階に分けて表示しています。表示項目をよく読み十分に理解してから、本製品の操作および保守作業を行ってください。

「危険」、「警告」および「注意」事項の表示は、危険性に関する重大性の順(危険＞警告＞注意)で、その内容を下記に説明します。

⚠ 危　険

「危険」項目は、本製品の運用中に、作業者が死亡または重傷に至る切迫した危険性のある場合について記述しています。

⚠ 警　告

「警告」項目は、本製品の運用中に、作業者が死亡または重傷を負う可能性のある場合について記述しています。

⚠ 注　意

「注意」項目は、本製品の運用中に、作業者が軽傷を負う可能性のある場合について記述しています。

注　意
⚠ (警告記号)のない「注意」項目は、作業者が負傷する恐れはないが、本製品、設備、機器等に損害や故障を引き起すことが予想される場合について記述しています。

本書では前述の危険レベル分けのほかに、下記の表記も使用しています。

重　要
「重要」項目は、本製品の操作および保守作業上、特に知っておかなければならない情報や内容がある場合に記述します。

備　考
「備考」項目は、本製品の操作および保守作業上、役立つ情報や内容がある場合に記述します。

1.1.1.3 警告図記号

本書では、「危険」、「警告」、「注意」、「重要」の表記に併せて次のようなシンボル記号を付加し、その警告内容をわかりやすく表現しています。

高電圧が印可される場合があることを表します。
安全確認を怠ったり、取扱いを誤ると感電によるショック、火傷、および死に至る危険を警告します。

表面温度が高くなる部品等があることを表します。
取扱いを誤ると、火傷の危険があることを意味します。

取扱いを誤ると、火災を起こす可能性があることを表します。

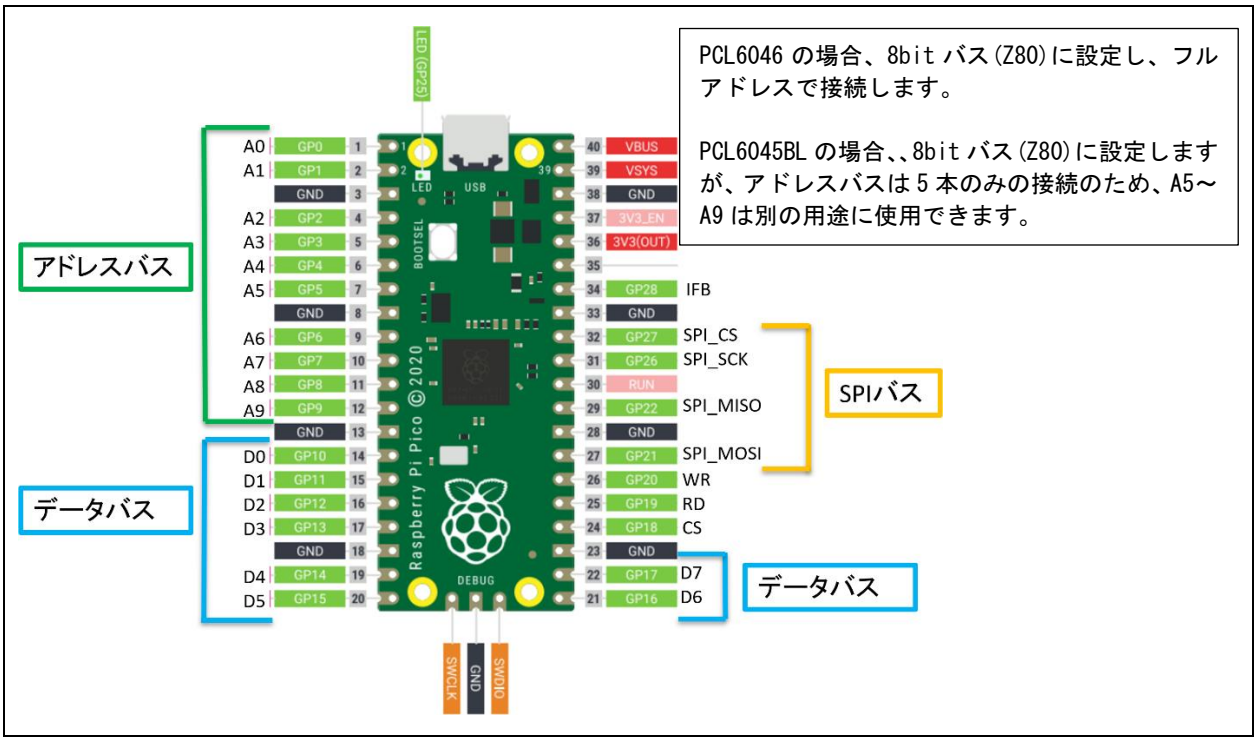
本製品の操作およびメンテナンス作業において、行ってはいけない「禁止」事項を示します。

本製品の操作およびメンテナンス作業において、必ず行っていただく「強制」事項を示します。

2. 概要

2.1 接続情報

Raspberry Pi PICO の端子は次のように使用します。



2.2 事例概要

G9006 を介して Motionnet のローカル側に接続されている PCL6046 を制御します。

ここで G9006 は、G9004A と違ってバスのエミュレーションを行うものではありません。

このため、G9001A 側の CPU に、ローカル側の PCL6046 を直接制御するようなプログラムを実装するわけではありません。

G9006 の役割は、G9001A 側の CPU からのデータを、G9006 側のローカル CPU へ引き渡すことにあります。

よって本サンプルでは、G9001A 側 CPU から、G9006 側のローカル CPU へ、PCL6046 の動作に関する指示を送信します。

指示を受け取った G9006 側のローカル CPU は、指示内容に相当する動作を実行することで、PCL6046 を制御します。

つまり、PCL6046 の制御プログラムは G9006 側のローカル CPU に実装されます。

2.3 開発環境

2.3.1 ソフトウェア類

開発は、OS として Windows10 Pro を使用し、次のソフトウェアを使用しています。

名称	内容
ARM GCC Compiler	コンパイラ
CMake	ビルド自動化ツール
Visual Studio Code	エディタ
Build Tools for Visual Studio	ビルドツール
Python 3.9	Python の実行環境
Git for Windows	GitHub から SDK をダウンロードするツール

各ツール類の入手先やインストール手順、操作方法 当に関しての解説は省略します。

2.3.2 ハードウェア類

使用したハードウェアは次の通りです。

名称	販売元
G9006 評価基板	（日本パルスモーター／貸出品）
Raspberry Pi Pico ボード	（秋月電子通商）
AD1431（ドライバ） （TB67S249FTG ドライバモジュール」でも可）	（日本パルスモーター）
PFC30-24V4（ステッピングモーター） （バイポーラ駆動なら他社製品でも可）	（日本パルスモーター） （秋月電子通商）
PCL6046 ピッチ変換基板 （PCL6045BL ピッチ変換基板でも可）	（日本パルスモーター／非買品） （日本パルスモーター）

3. G9006 への設定内容とローカル CPU の役割

G9006 はローカル CPU により初期設定されます。

3.1 G9006 の初期化内容

今回は次のように初期化します。

この結果、G9001A からは 3 つの G9002A が接続されているように見えます。

設定内容	意味
RADD レジスタ＝4700h	① デバイス番号を 00h に設定。 ② 仮想デバイスを 2 つ追加。 ③ デバイス番号=00h は、4 ポート全て入力ポートに設定。 デバイス番号=01h(仮想デバイス)は、4 ポート全て入力ポートに設定。 デバイス番号=02h(仮想デバイス)は、4 ポート全て出力ポートに設定。
ポート変化割り込み設定メモリー 01h 番地＝0Ah	デバイス番号 2 のポート 3、ポート 1 に変化があれば割り込みを発生。

3.2 汎用ポートの役割

汎用ポートは次の用途で使用します。

デバイス番号	ポート番号	用途
0	PORT1､PORT 0	X 軸 現在位置カウンター下位 16bit
	PORT3､PORT 2	X 軸 現在位置カウンター上位 16bit
1	PORT1､PORT 0	Y 軸 現在位置カウンター下位 16bit
	PORT3､PORT 2	Y 軸 現在位置カウンター上位 16bit
2	PORT1､PORT 0	X 軸の目標位置（絶対座標）
	PORT3､PORT 2	Y 軸の目標位置（絶対座標）

3.2.1 出力ポートの詳細

デバイス番号＝2 の仮想デバイスは、すべてのポートは出力に設定されています。

このポートエリアには、G9001A 側から出力データが送信されてきます。

このデータは X 軸、Y 軸の目標位置をして使用されるため、この値が変化した場合、PCL6046 の各軸も変化に応じて移動させます。

このポートの使い方は次の通りです。

ポート番号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PORT1､PORT 0	Chg	符号	X 軸 の 目 標 位 置													
PORT3､PORT 2	Chg	符号	Y 軸 の 目 標 位 置													

X 軸、Y 軸で、それぞれ 16bit の情報を管理しています。

各データの下位 14bit が各軸の目標位置となります。

ビット 14 は符号として扱います。

ビット 15 は、データ変化の監視のために使用されます。ビット 14～0 のデータを変更する場合、このビットを反転させます。これにより、変化監視の対象をポート 3、ポート 1 に限定できます。

3.3 ローカル CPU の役割

電源投入直後から次のような制御を行います。

① PCL6046 の初期化（X 軸と Y 軸へのレジスタ設定）。

② G9006 の初期設定。

③ PCL6046 の X 軸と Y 軸に対する原点復帰動作。

④ PCL6046 の X 軸と Y 軸から現在位置カウンタを読み出し、デバイス番号 0、1 のポートエリアへ書き込む。

⑤ デバイス番号 2 のポート 3、ポート 1 の状態が変わっていないかを監視。

ポートの状態が変化していた場合、PCL6046 の各軸の目標位置を変更。

（新しい目標位置を書き込まれた PCL6046 は、現在位置と比較し、差分がる場合は差分がゼロになるように指定された軸を移動させる。この動作は、PCL6046 が自動的に行うので、CPU は PRMV レジスタの更新とスタートコマンドの書き込みだけ行うだけです。）

以降、④と⑤を繰り返す。

4. ソースコード説明

ソースコードは次のファイルで構成されています。

ファイル名	内容
main.c	メインとなる動作のシーケンスを記載。
main.h	各種レジスタの定数を記載。
pcl_access.c	パラレルバスアクセス関数を記載。
pcl_access.h	パラレルバスアクセス関数の宣言を記載。 PCL のアドレス、書き込みデータを分割する際に使用する構造体を定義。
g9006_access.c	SPI アクセス関数を記載。
g9006_access.h	SPI アクセス関数の宣言を記載。
parallel_bus_module.pio	パラレルバスアクセスの機能を記載。
spi_module.pio	SPI アクセスの機能を記載。
CmakeLists.txt	コンパイルの定義。
pico_sdk_import.cmake	コンパイルの定義。
stdafx.c	Raspberry Pi PICO 用のソース。
stdafx.h	Raspberry Pi PICO 用のソース。

4.1 Main.c

4.1.1 インクルードファイル

```
#include "stdafx.h"
#include "main.h"
#include "pcl_access.h"
#include "g9006_access.h"
```

4.1.2 関数

4.1.2.1 main

次の処理を行っています。

以降では、PCL6046 の制御に必要な関数の説明を行ってゆきます。

const PIO pio = pio0;
const uint sm1 = 1;
const uint sm2 = 2;	

stdio_init_all();	USB ポートの初期化(Raspberry Pi PICO 用)。
sleep_ms(3000);	
pcl_bus_init(pio, sm1, IFB, PIO_OUT_BASE, PIO_OUT_NUM, PIO_SIDE_BASE, PIO_SIDE_BASE_NUM, PIO_IN_BASE);	パラレルバスの初期化
spi_init(pio, sm2, SPI_MOSI, SPI_MOSI_NUM, SPI_MISO, SPI_MISO_NUM, SPI_SIDE_BASE, SPI_SIDE_BASE_NUM, SPI_SCK_FRQ);	SPI の初期化
pcl_reg_set(pio, sm1, X_SEL);	X 軸の各種レジスタ設定
pcl_reg_set(pio, sm1, Y_SEL);	Y 軸の各種レジスタ設定
g9006_init_set(pio, sm2);	G9006 初期設定
org_run(pio, sm1, X_SEL);	X 軸原点位置移動
org_run(pio, sm1, Y_SEL);	Y 軸原点位置移動
for(;;){	
rcun_update(pio, sm1, sm2);	X 軸と Y 軸の現在位置カウンタを読み出し、デバイス番号 0、1 のポートエリアへ格納。
port_change_chk(pio, sm1, sm2);	デバイス番号 2 のポート 3、1 の状態が変化した場合、PCL6046 の X 軸、Y 軸の位置を変更。
}	

4.1.2.2 pcl_reg_set

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm	使用するステートマシン番号(0～3)
axs	書き込む軸

PCL6046 のレジスタへ、次のような初期設定を行います。

レジスタ	指定軸軸	内容
PRFL	32h	初速度。
PRFH	190h	最高速度。
PRUR	5AAh	加速度。
PRDR	0h	減速度(ゼロを設定した場合、加速度と同じ値が適用される)。
PRMG	12Bh	速度倍率(今回は 1 倍)。
PRUS	0h	S 字加減速動作時の、直線加速領域の定義。
PRDS	0h	S 字加減速動作時の、直線減速領域の定義。
RFA	64h	補助的な速度(今回の動作では未使用)。
PRMV	Ah	移動量。
PRIP	70000000h	円弧補間の中心位置(今回の動作では未使用)。
PRCI	64h	円弧補間歩進数(今回の動作では未使用)。
PRDP	0h	スローダウンポイント。
PRMD	42h	動作モード (位置決め動作 (COUNTER1 絶対位置指定))。
RENV1	80000000h	環境設定 1
RENV3	100000h	環境設定 2

レジスタへの初期設定後、X 軸、Y 軸のメインステータスを読み出し、UART へ出力します。

4.1.2.3 g9006_init_set

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm	使用するステートマシン番号(0～3)

次の動作を実行します。

① G9006 の RADD レジスタ "4700h" を書き込む。

② G9006 の RADD レジスタを読み出し、UART へ出力(デバッグ用)。

③ G9006 のポート変化割り込み設定メモリーの 1 番地に "0Ah" を書き込む。

④ G9006 のポート変化割り込み設定メモリーの 1 番地を読み出し、UART へ出力(デバッグ用)。

4.1.2.4 org_run

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm	パラレルアクセスに使用するステートマシン番号
axs	原点復帰を行う軸

次の動作を実行します。

① PCL6046 の PRMD レジスタ "1Dh" を書き込む(マイナス方向への原点サーチ動作)。

② PCL6046 の指定軸を FL 定速動作でスタート。

③ PCL6046 の PRMD レジスタ "42h" を書き込む(位置決め動作 (COUNTER1 絶対位置指定))。

②でスタートした原点サーチ動作が完了していなくても、プリレジスタへの書き込みのため、②の動作に影響はない。

4.1.2.5 rcun_update

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm1	パラレルアクセスに使用するステートマシン番号
sm2	SPI アクセスに使用するステートマシン番号

次の動作を実行します。

① PCL6046 の X 軸のカウンター1 を読み出す。

② G9006 のデバイス番号＝0 のポート 1、0 エリアに、読み出したデータの下位 16 ビットを書き込む。

③ G9006 のデバイス番号＝0 のポート 3、2 エリアに、読み出したデータの上位 16 ビットを書き込む。

④ PCL6046 の Y 軸のカウンター1 を読み出す。

⑤ G9006 のデバイス番号＝1 のポート 1、0 エリアに、読み出したデータの下位 16 ビットを書き込む。

⑥ G9006 のデバイス番号＝1 のポート 3、2 エリアに、読み出したデータの上位 16 ビットを書き込む。

4.1.2.6 port_change_chk

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm1	パラレルアクセスに使用するステートマシン番号
sm2	SPI アクセスに使用するステートマシン番号

次の動作を実行します。

① G9006 のポート変化割り込みフラグメモリーの 0、1 番地を読み出す。

② デバイス番号＝2 のポート 1 に変化があった場合、次の動作を実行。

・デバイス番号＝2 のポート 1、0 を読み出す。

・符号ビットが "0" の場合、"3FFFh" でマスクする。

・符号ビットが "1" の場合、データのビット 31～14 までを "1" に変更する(符号拡張)。

・PCL6046 の X 軸の PRMV レジスタに、マスク処理(もしくは符号拡張)したデータを書き込む。

・PCL6046 の X 軸に高速スタート 2 を書き込み、目標位置への移動動作を開始する。

③ デバイス番号＝2 のポート 3 に変化があった場合、次の動作を実行。

・デバイス番号＝2 のポート 3、2 を読み出す。

・符号ビットが "0" の場合、"3FFFh" でマスクする。

・符号ビットが "1" の場合、データのビット 31～14 までを "1" に変更する(符号拡張)。

・PCL6046 の Y 軸の PRMV レジスタに、マスク処理(もしくは符号拡張)したデータを書き込む。

・PCL6046 の Y 軸に高速スタート 2 を書き込み、目標位置への移動動作を開始する。

4.1.2.7 port_rmv_update

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm1	パラレルアクセスに使用するステートマシン番号
sm2	SPI アクセスに使用するステートマシン番号
address	ポート読出しメモリアドレスを指定 08h＝デバイス番号 2 ポート 0,ポート 1 を読み出し 0Ah＝デバイス番号 2 ポート 2,ポート 3 を読み出し
axs	処理する軸

次の動作を実行します。

① G9006 のポート変化割り込みフラグメモリーの 0、1 番地を読み出す。

② G9006 のポートデータメモリーの、指定アドレス(address)を読み出す。

③ 符号を確認し、次の動作を実行。

・符号ビットが "0" の場合、"3FFFh" でマスクする。

・符号ビットが "1" の場合、データのビット 31～14 までを "1" に変更する(符号拡張)。

④ PCL6046 の指定された軸(axs)の PRMV レジスタに、③で処理したデータを書き込む。

⑤ PCL6046 の指定された軸(axs)に高速スタート 2 コマンドを書き込む。

4.1.2.8 port_change_flgclr

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1)
sm	使用するステートマシン番号(0～3)
port	フラグクリアするポートを指定 true ：ポート 1 のフラグをクリア false：ポート 3 のフラグをクリア

次の動作を実行します。

① G9006 の指定されたポートに対応するポート変化割り込みをクリア。

② クリアした状態を読み出し、UART へ出力。

4.2 Main.h

4.2.1 マクロ定義

PIO_OUT_BASE	PCL アドレスバス A0 に接続する GPIO の番号を指定。
PIO_OUT_NUM	PCL アドレスバスとデータバスに使用する本数を指定。
PIO_SIDE_BASE	CS,WR,RD を割り当てるピンのベース番号を指定。
PIO_SIDE_BASE_NUM	CS,WR,RD に使用するピンの本数を指定。
PIO_IN_BASE	PCL データバス D0 に接続する GPIO の番号を指定。
IFB	PCL_IFB 信号読み込み端子を指定。
SPI_MOSI	SPI モジュール_MOSI 端子を指定。
SPI_MOSI_NUM	MOSI 端子の本数を指定。
SPI_MISO	SPI モジュール_MISO 端子を指定。
SPI_MISO_NUM	MISO 端子の本数を指定を指定。
SPI_SIDE_BASE	SPI モジュール_SCK,CS 端子のベースピンを指定。
SPI_SIDE_BASE_NUM	SCK,CS 端子合計の本数を指定。
SPI_SCK_FRQ	SPI_SCK 周波数を指定。 (125MHz の 10 分周でステートマシンの動作周波数は 12.5MHz,SCK の周波数は 12.5MHz/4=3.125MHz)
PORTFLGADDRESS	ポート変化割り込み読み出しスタートアドレスを指定。

4.2.2 列挙型

Axis	PCL6046 の軸選択を定義。
Register	PCL6046 用のレジスタアドレス値を定義。
Command	PCL6046 用のコマンドの値を定義。
G9006COMTYPE	G9006 制御コマンドタイプを定義。
G9006OPECOM	G9006 操作コマンドを定義。
G9006REG	G9006 レジスタアドレスを定義。

4.3 pcl_access.c

4.3.1 インクルードファイル

```
#include "stdafx.h"
#include "pcl_access.h"
#include "parallel_bus_module.pio.h"
```

4.3.2 関数

4.3.2.1 pcl_bus_dir_change

パラレルバスの入出力方向を切り替えます。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。

4.3.2.2 address_byte

アドレス値を 1byte ごとに分割します。

Raspberry Pi PICO でパラレルアクセスさせるための前処理です。

引数は次の通りです。

address	PCL6046 アクセス用のアドレス値。
pcl_address_bytes[]	バイト単位に分割されたアドレス値を格納する配列。

4.3.2.3 data_byte

書き込みデータを 1byte ごとに分割します。

Raspberry Pi PICO でパラレルアクセスさせるための前処理です。

引数は次の通りです。

data	PCL6046 書き込み用のデータ。
reg_data_bytes[]	バイト単位に分割されたデータを格納する配列。

4.3.2.4 write_tx_fifo

パラレルバスインターフェースのバッファにデータを書き込みます。

Raspberry Pi PICO でパラレルアクセスさせるための前処理です。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
write_data	バッファに書き込むデータ。

4.3.2.5 read_rx_fifo

パラレルバスインターフェースのバッファからデータを読み出します。

Raspberry Pi PICO でパラレルアクセスさせた後の処理です。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
write_data	ダミー(ゼロを指定) 。

戻り値は次の通りです。

uint32_t	読み出されたデータ。

4.3.2.6 pcl_bus_init

パラレルバスインターフェースを初期化します。

Raspberry Pi PICO でパラレルアクセスさせるための前処理です。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
pio_out_base	PCL アドレスバス A0 に接続する GPIO の番号。
pio_out_num	PCL アドレスバスとデータバスに使用する本数。
pio_side_base	CS,WR,RD を割り当てるピンの番号。
pio_side_num	CS,WR,RD に使用するピンの本数。
pio_in_base	PCL データバス D0 に接続する GPIO の番号。

4.3.2.7 write_reg_pcl

PCL6046 へレジスタデータを書き込みます。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
axs	書き込む軸。
address	書き込みアドレス。
data	書き込みデータ。

4.3.2.8 read_reg_pcl

PCL6046 からレジスタデータを読み出します。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
axs	書き込む軸。
address	書き込みアドレス。

戻り値は次の通りです。

uint32_t	読み出されたレジスタ値。

4.3.2.9 write_com_pcl

PCL6046 へコマンドを書き込みます。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
com_axis	書き込む軸。
address	書き込みアドレス。
com	動作コマンド。

4.3.2.10 read_msts_pcl

PCL6046 からメインステータスを読み出します。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
axs	書き込む軸。
address	書き込みアドレス。

戻り値は次の通りです。

uint32_t	読み出されたステータス値。

4.3.2.11 read_port_substat_pcl

PCL6046 から汎用ポートとサブステータスを読み出し、UART へ出力します。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
axs	書き込む軸。
address	書き込みアドレス。

戻り値はありません。

4.3.2.12 write_port_pcl

PCL6046 へ汎用出力ポートに書き込みます。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
ifb	PCL6046 の IFB 信号読み込み端子。
axs	書き込む軸。
address	書き込みアドレス。
data	書き込みデータ。

4.4 pcl_access.h

4.4.1 マクロ定義

なし。

4.4.2 列挙型

address	アドレスデータを 1byte ごとに分割するために使用。
data	書き込みデータを 1byte ごとに分割するために使用。

4.5 g9006_access.c

4.5.1 インクルードファイル

```
#include "stdafx.h"
#include " g9006_access.h"
#include " spi_module.pio.h"
```

4.5.2 関数

4.5.2.1 spi_access_module

SPI アクセス用のモジュール関数。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
write_data	SPI 書き込みデータ

戻り値は次の通りです。

uint32_t	SPI 読み出しデータ。

4.5.2.2 spi_init

SPI アクセスインターフェースの初期化を行いす。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
spi_mosi	MOSI 端子を指定。
spi_mosi_num	MOSI 端子の本数を指定
spi_miso	MISO 端子を指定。
spi_miso_num	MISO 端子の本数を指定。
spi_side_base	SCK,CS 端子のベースピンを指定。。
spi_side_num	SCK,CS 端子合計の本数を指定
spi_sck_frq	SCK の動作周波数を指定。 ([10]を指定した場合、125MHz の 10 分周でステートマシンの動作周波数は 12.5MHz,SCK の周波数は 12.5MHz/4=3.125MHz)

4.5.2.3 write_reg_G9006

G9006 のアドレス有書き込み制御時に使用する関数。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
comtype	制御コマンドタイプ選択。
address	書き込みアドレスを指定。
data	書き込みデータを指定。

4.5.2.4 read_reg_G9006

G9006 のアドレス有読み出し制御時に使用する関数。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
comtype	制御コマンドタイプを指定。
address	読み出しアドレスを指定。

戻り値は次の通りです。

uint32_t	読み出しデータ。

4.5.2.5 read_str_G9006

G9006 のアドレスなし読み出し制御時に使用する関数。

読み出したデータは UART へ出力するだけで、戻り値はありません。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
comtype	制御コマンドタイプを指定。

4.5.2.6 write_opecom_G9006

G9006 のアドレスなし書き込み制御時に使用する関数。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
comtype	制御コマンドタイプを指定。
opecom	操作コマンドを指定

4.6 g9006_access.h

4.6.1 マクロ定義

なし。

4.6.2 列挙型

なし。

4.7 parallel_bus_module.pio

4.7.1 関数

4.7.1.1 parallel_bus_module_init

パラレルバスインターフェースを初期化します。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
sm	使用するステートマシン番号(0～3) 。
offset	PIO プログラムの先頭アドレス
out_base	OUT 命令で出力するベースピン番号
out_pin_num	OUT 命令で出力するピンの本数(IN 命令の時はこのうち 8 本が IN 方向となる)
side_base	SIDE-SET 命令で使用するピンのベース番号
side_pin_num	SIDE-SET 命令で使用するピンの本数(CS と WR と RD 用)
in_base	IN 命令で使用するベースピン(PCL 読み出しデータの bit0)

4.8 8.10. spi_module.pio

4.8.1 関数

4.8.1.1 spi_module_init

SPI インターフェースを初期化します。

引数は次の通りです。

pio	使用する PIO 番号(PIO0 or PIO1) 。
state_machine	使用するステートマシン番号(0～3) 。
prog_addr	プログラムをロードするプログラムの先頭アドレスを指定
outpin_base	MOSI 端子に使用するピンの番号を指定
num_outpin	MOSI 端子に使用するピンの本数を指定
inpin_base	MISO 端子に使用するピンの番号を指定
num_inpin	MISO 端子に使用するピンの本数を指定
sidepin_base	SCK,CS 端子のベースピン番号を指定
num_sidepin	SCK,CS 端子合計のピン本数を指定
clkdiv	SCK の動作周波数を指定 ([10]を指定した場合、125MHz の 10 分周でステートマシンの動作周波数は 12.5MHz,SCK の周波数は 12.5MHz/4=3.125MHz)

弊社は、弊社ソフトウェアについて著作権を含む一切の知的所有権を保持します。弊社は、弊社ソフトウェアに関するいかなる権利もお客様に譲渡しません。お客様は、弊社の製品を使用する目的でのみ、現状有姿の弊社ソフトウェアを使用することができます。弊社は、弊社ソフトウェアの完全性、正確性、適用性、有用性、第三者知財の非侵害性を含め、明示たると黙示たるとを問わず何らの保証をいたしません。また、弊社ソフトウェアを使用したことで生じる損害（収入または利益の逸失を含む）について、一切の責任を負いません。お客様が、購入国以外で弊社ソフトウェアを使用する場合は、購入国と使用国の輸出管理法や規制を遵守する必要があります。

顧客「満足」から「感動」へ。
日本パルスモーター株式会社

www.pulsemotor.com

お問合せ
www.pulsemotor.com/support

東京 電話 03(3813)8841 FAX 03(3813)8550
大阪 電話 06(6576)8330 FAX 06(6576)8335
お電話受付時間 平日 9:00～17:00

2023 年 12 月発行