RL78/G14

簡易 IIC を利用した I2C バス制御 (Arduino API)

要旨

本アプリケーションノートでは、RL78/G14 Fast Prototyping Board (FPB) の Pmod コネクタの I2C バスを用い、Arduino 言語のようなプログラム記述で温湿度センサ HDC1080 を制御します。また、Arduino コネクタの I2C バスを用い、IICA0 を使用して LCD 表示器を制御します。

対象デバイス

RL78/G14

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 仕様

本アプリケーションノートでは、FPB の Pmod コネクタ 1 の I2C バスを用い、ファースト・モード (380 kbps で使用) で温湿度センサ HDC1080 を Arduino 言語のようなプログラム記述で制御します。また、Arduino コネクタの I2C バスを用い、標準モード (85 kbps で使用) で 16 文字×2 行の LCD 表示器に温湿度センサから取得したデータを表示します。

1 分ごとの定周期もしくはスイッチ押下により、温湿度センサからデータを取得し、LCD 表示器に取得データを表示します。

RL78/G14 FPB では、Pmod コネクタ 1 およびコネクタ 2 の I2C バスの SCL 信号と SDA 信号は、標準である Pmod コネクタの SCL 信号と SDA 信号の端子配置が異なります。Pmod に対応した I2C モジュールと接続するために、図 1.1 のように SCL 信号と SDA 信号を入れ替える変換ボードを別途準備します。

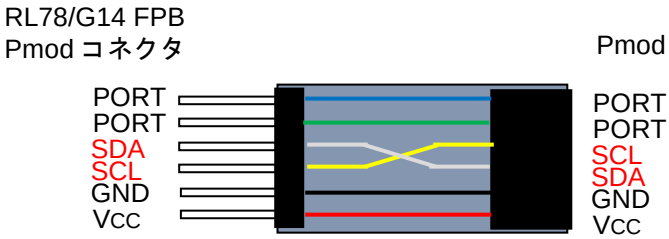

図 1.1 信号変換

RL78/G14 FPB には、2 つの Pmod コネクタがあります。温湿度センサ HDC1080 は Pmod コネクタ 1 に割り当てられた Wire1 を用いて制御しています。

Pmod コネクタ 2 に割り当てられた Wire2 を使用するには、Wire2 を有効にする処理が必要です。r_cg_userdefine.h の図 1.2 の赤で囲んだ 50 行目をコメントアウトし、51 行目のコメントアウトは外して有効にし、AR_SKETCH.c で使用している API 関数名 Wire1 を Wire2 に変更してください。

```
45 /*------------------------------------------------------------↓
46     definition of usage of Wire function↓
47     (if not use the function, commentout the statement)↓
48 ------------------------------------------------------------*/↓
49 ↓
50 #define     USE_WIRE1                   /* use IIC00 on Pmod1 as Wire1              */↓
51 //#define     USE_WIRE2                   /* use IIC20 on Pmod2 as Wire2              */↓
52 ↓
```

図 1.2 使用する WireAPI の定義部

表 1.1 に本プログラムで使用する周辺機能と用途を示します。

表 1.1 使用する周辺機能と用途

周辺機能	用途
デジタル入力	スイッチ (SW_USR) の状態の読み込み
IICA0	I2C バスを利用した LCD 表示器の制御
IIC00	Pmod1 コネクタの I2C バスを利用したセンサの制御
IIC20	Pmod2 コネクタの I2C バスを利用したセンサの制御
タイマ・アレイ・ユニット	時間経過の計測

1.1 プログラム実行環境

本アプリケーションノートでは、RL78 ファミリ固有の開発環境上で、Arduino 言語のようなプログラムを実行させています。プログラム実行環境の概念図を図 1.3 に示します。

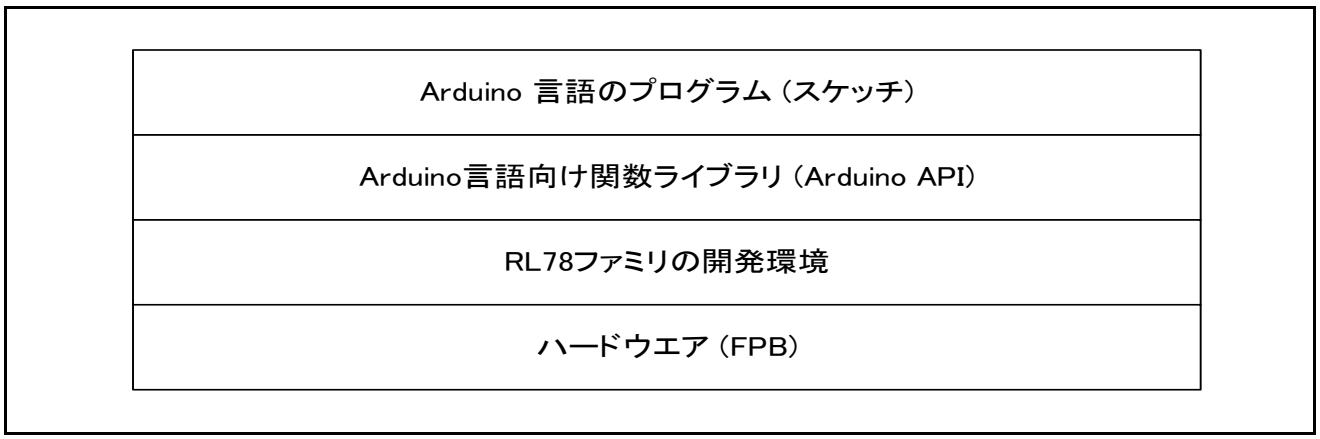

図 1.3 プログラム実行環境

本アプリケーションノートで準備しているライブラリ関数を表 1.2～表 1.4 に示します。

表 1.2 ライブラリ関数 (1/3)

項目	ライブラリ関数	機能
デジタル入出力	pinMode(pin, mode)	pin で指定した端子の動作モード(入力モード / 出力モード / 内蔵プルアップ抵抗を有効にした入力モード) 指定
	digitalWrite(pin, value)	pin で指定した端子を value で指定した状態 (ハイレベル / ロウレベル) にする。
	digitalRead(pin)	pin で指定した端子状態を読み出す。
時間管理	millis()	プログラムの実行を開始した時から現在までの時間をミリ秒単位で返します。
	micros()	プログラムの実行を開始した時から現在までの時間をマイクロ秒単位で返します。
	delay(ms)	ミリ秒単位でプログラムを指定した時間だけ止めます。
	delayMicroseconds(us)	マイクロ秒単位でプログラムを指定した時間だけ止めます。

表 1.3 ライブラリ関数 (2/3)

項目	ライブラリ関数	機能
I2C 制御 (Wire 制御)	Wire.begin()	IICA0 を初期化し、マスタとして I2C バスに接続します。
	Wire.requestFrom(saddr7, bytes, stop) Wire.requestFrom(saddr7, bytes)	指定したスレーブから bytes で指定した数のデータを受信します。 Wire.available()関数でデータ数を求め、Wire.read()関数で読み出します。
	Wire.beginTransmission(saddr7)	指定したスレーブに対して送信準備を行います。 その後、Wire.write()関数でデータをキューに設定します。Wire.endTransmission()で送信を実行します。
	Wire.endTransmission(stop)	キューからスレーブにデータを送信して処理を終了します。
	Wire.write(data)	スレーブに送信するデータをキューに設定します。
	Wire.available()	Wire.read()関数で読み出し可能なデータ数をチェックします。
	Wire.read()	スレーブからの受信データを読み出します。
簡易 IIC 制御 (Wire1 制御)	Wire1.begin()	Pmod コネクタ 1 に接続された IIC00 (Wire1) を初期化し、マスタとして I2C バスに接続します。
	Wire1.requestFrom(saddr7, bytes, stop) Wire1.requestFrom(saddr7, bytes)	指定したスレーブから bytes で指定した数のデータを受信します。 Wire1.available()関数でデータ数を求め、Wire1.read()関数で読み出します。
	Wire1.beginTransmission(saddr7)	指定したスレーブに対して送信準備を行います。 その後、Wire1.write()関数でデータをキューに設定します。Wire1.endTransmission()で送信を実行します。
	Wire1.endTransmission(stop)	Wire1 で、キューからスレーブにデータを送信して処理を終了します。
	Wire1.write(data)	Wire1 のスレーブに送信するデータをキューに設定します。
	Wire1.available()	Wire1.read()関数で読み出し可能なデータ数をチェックします。
	Wire1.read()	Wire1 のキューからスレーブからの受信データを読み出します。

備考 I2C バスのスレーブ機能はサポートしていません。また、一部の関数で、引数または引数の個数が制限されています。

表 1.4 ライブラリ関数 (3/3)

簡易 IIC 制御 (Wire2 制御)	Wire2.begin()	Pmod コネクタ 2 に接続された IIC20 (Wire2) を初期化し、マスタとして I2C バスに接続します。
	Wire2.requestFrom(saddr7, bytes, stop) Wire2.requestFrom(saddr7, bytes)	指定したスレーブから bytes で指定した数のデータを受信します。 Wire2.available()関数でデータ数を求め、Wire2.read()関数で読み出します。
	Wire2.beginTransmission(saddr7)	指定したスレーブに対して送信準備を行います。 その後、Wire2.write()関数でデータをキューに設定します。Wire2.endTransmission()で送信を実行します。
	Wire2.endTransmission(stop)	Wire2 で、キューからスレーブにデータを送信して処理を終了します。
	Wire2.write(data)	Wire2 のスレーブに送信するデータをキューに設定します。
	Wire2.available()	Wire2.read()関数で読み出し可能なデータ数をチェックします。
	Wire2.read()	Wire2 のキューからスレーブからの受信データを読み出します。

備考 I2C バスのスレーブ機能はサポートしていません。また、一部の関数で、引数または引数の個数が制限されています。

1.2 プログラム (スケッチ) の構成

プロジェクトが格納されているフォルダ (workspace) の各統合開発環境フォルダ又は zip ファイル (e2studio の場合) には、RL78 ファミリ開発環境関係のファイルが格納されています。

サブフォルダ AR_LIB には、Arduino API が格納されています。

サブフォルダ sketch には、Arduino 言語のプログラム (スケッチ) である AR_SKETCH.c が格納されています。

スケッチを参照もしくは変更する場合は、sketch の中の”AR_SKETCH.c”ファイルを使用します。

1.3 プロジェクトの起動準備

サンプルコードを圧縮して格納されているアーカイブを解凍すると、3 種の統合開発環境に対応したフォルダ又は zip ファイル (e2studio の場合) が得られるので、使用する統合開発環境用のものを使用してください。

各統合開発環境での手順等については、RL78/G14 FPB 導入ガイド (R01AN5431) アプリケーションノートを参照してください。

1.4 プログラム (スケッチ) の定義

プログラム (スケッチ) の定義内容を図 1.4 に示します。

1)

```
int swPin = 18;                    // assign D18 pin to swPin for SW_USER.
```

2)

```
#define SLADDR_HDC1080    ( 0x40 )        // I2C bus slave address of HDC1080
#define MINUTE           ( 60000/16 )     // 1 minute devided by 16milli sec

unsigned int old_time = 0x0000;           // previous time(milli sec.)

unsigned char hdc1080_buff[4] =           // HDC1080 communication data area
{
    0x00,                                 // high byte of Humidity
    0x00,                                 // low byte of Humidity
    0x00,                                 // high byte of Temp.
    0x00                                  // low byte of Temp.

};

unsigned char humid;                      // Humidity data(unit %)
int   temp;                               // Temperature data (0.1degree unit)
```

3)

```
// LCD display buffer aera 40characters 2lines
// display data is 16 characters/line and 2lines.
// charactor position  0123456789012345
unsigned char disp_line1[40] = " Temp. =  15.0 C";
unsigned char disp_line2[40] = " Humidity =  50%";

int count16ms = 0x0000;                   // for count 1 minute
char sw_work = 0xFF;                      // work for switch check

extern API_Wire Wire;                     // wire API
extern API_Wire Wire1;                    // wire API
```

図 1.4 プログラムの定義部の内容

1) ボード上のスイッチ (SW_USR) を制御する swPin 端子に 18 を指定して D18 に割り当てます。

2) 次に経過時間 (ミリ秒単位) を確認するための 16 ビットの変数 old_time、センサ HDC1080 の制御用として、通信用の 4 バイトの配列 hdc1080_buff と、得られた湿度データ用変数 humid と 0.1 度単位の温度データ変数 temp を定義しています。

3) LCD 表示器の表示データ領域で、1 行目用の変数 disp_line1、2 行目用の変数 disp_line2 の 2 つの 40 バイトの配列を定義しています。

また、16 ミリ秒のインターバルをカウントして 1 分を得るためのカウンタ count16ms とスイッチのチェック用の変数 sw_work を定義しています。

API_Wire 型の構造体 Wire1 は、Pmod コネクタ用の Wire1 関係の API 関数を提供している AR_LIB_WIRE1.c で定義された関数を参照するためのものです。

1.5 初期設定処理

プログラム (スケッチ) の初期設定部分を図 1.5 に示します。

ここでは、setup 関数として、スイッチ入力端子を入力に指定します。さらに、I2C バスのマスタとして IICA0 および IIC00 を設定しています。その後、LCD 表示器に初期表示データを設定しています。

```
void setup(void){
  // put your setup code here, to run once:
    pinMode(swPin, INPUT);                  // set D18pin to input mode

    Wire_begin();                           // set IICA0 for I2C bus master
    Wire1_begin();                          // set IIC00 for I2C bus master

    init_LCD();                             // initialize LCD display

    disp_line1[14] = 0xDF;                  // set degreeC character of LCD
    print_LCD( (uint8_t *__near)disp_line1, (uint8_t *__near)disp_line2);

}
```

図 1.5 初期設定処理部分

1.6 メイン処理部

繰り返し実行されるメイン処理の先頭部分を図 1.6 に示します。「プロジェクトの起動準備」が正しく設定されていると、スケッチがダウンロードされて、loop 関数で停止した状態となります。

```
void loop(void){
  // put your main code here, to run repeatedly:

    static char m_time = 1;
    char work;
    char sw_data;
    unsigned int time_work;
    int work_int;
    unsigned long long_work;

/*-------------------------------------
    wait for 16milli seconds interval.
-------------------------------------*/

    time_work = ( int )( millis() & 0x0FFF0 ); // read milli sec data

    if ( old_time != time_work )        // check 16 milli seconds passed
    {
```

図 1.6 メイン処理の先頭部分

1.7 HDC1080 のデータ処理

HDC1080 は、通常はスリープモードになっています。温度と湿度のデータを取得するためには、測定を要求する必要があります。測定要求は、HDC1080 のスレーブアドレスに続けて 0x00 を送信することで実現できます。

HDC1080 から 14 ビット長の温度と湿度のデータを取得する場合、測定完了までに TYP. 6.5 ミリ秒かかります。本アプリケーションノートでは、測定要求してから 32 ミリ秒後に温度・湿度データを取得します。

HDC1080 から読み出した 4 バイトのデータは図 1.7 のようになります。温度データを赤字で示し、湿度データを青字で示します。

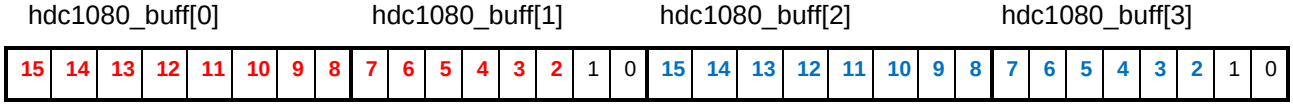

図 1.7 HDC1080 のデータ形式

温度は以下の式で求めます。①で 16 ビットの値を求め、②で 0.1℃単位の値に変換し、③で変換後オフセットとして 40℃分を引くことで 0.1℃単位の温度を得ています。

```
① long_work =  ( hdc1080_buff[0] * 0x100UL + ( hdc1080_buff[1] );
② long_work *= 1650;       // multiply 10 times of Maximum temperature
③ temp = (int)((long_work >> 16) -400); // adjust offset(40degreeC)
```

湿度は以下の式で求めます。①で 16 ビットの値を求め、②で値をパーセントにするために 100 倍し、③でフルスケールに対する値を求めて、100%スケールでの湿度を得ています。

```
① long_work = ( hdc1080_buff[2] * 0x100UL + hdc1080_buff[3] );
② long_work *= 100UL;      // get percentage
③ humid = (unsigned char)(long_work >>16); // get humidity
```

2. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表 2.1 動作確認条件

項目	内容
使用マイコン	RL78/G14 (R5F104MLAFB：RL78G14_FPB)
動作周波数	● 高速オンチップ・オシレータ・クロック (f_{IH})： 32MHz ● CPU／周辺ハードウエア・クロック: 32MHz
動作電圧	3.3V (2.75V～5.5V で動作可能) LVD 動作：リセット・モード LVD 検出電圧 (V_{LVD}) 立ち上がり時 TYP. 2.81V (2.76V～2.87V) 立ち下がり時 TYP. 2.75V (2.70V～2.81V)
統合開発環境	ルネサス エレクトロニクス CS+ for CC V8.05.00 ルネサス エレクトロニクス e² studio V7.7.0 IAR Systems IAR Embedded Workbench for RL78
C コンパイラ	ルネサス エレクトロニクス CC-RL V1.10.00 IAR Systems RL78 用 IAR C/C++ コンパイラ v4.20.1

3. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。

併せて参照してください。

Arduino API 導入ガイド (R01AN5413) アプリケーションノート

オンボード LED 点滅制御 (Arduino API) (R01AN5384) アプリケーションノート

4. ハードウェア説明

4.1 ハードウェア構成例

本アプリケーションノートで使用するハードウェア (FPB) を図 4.1 に示します

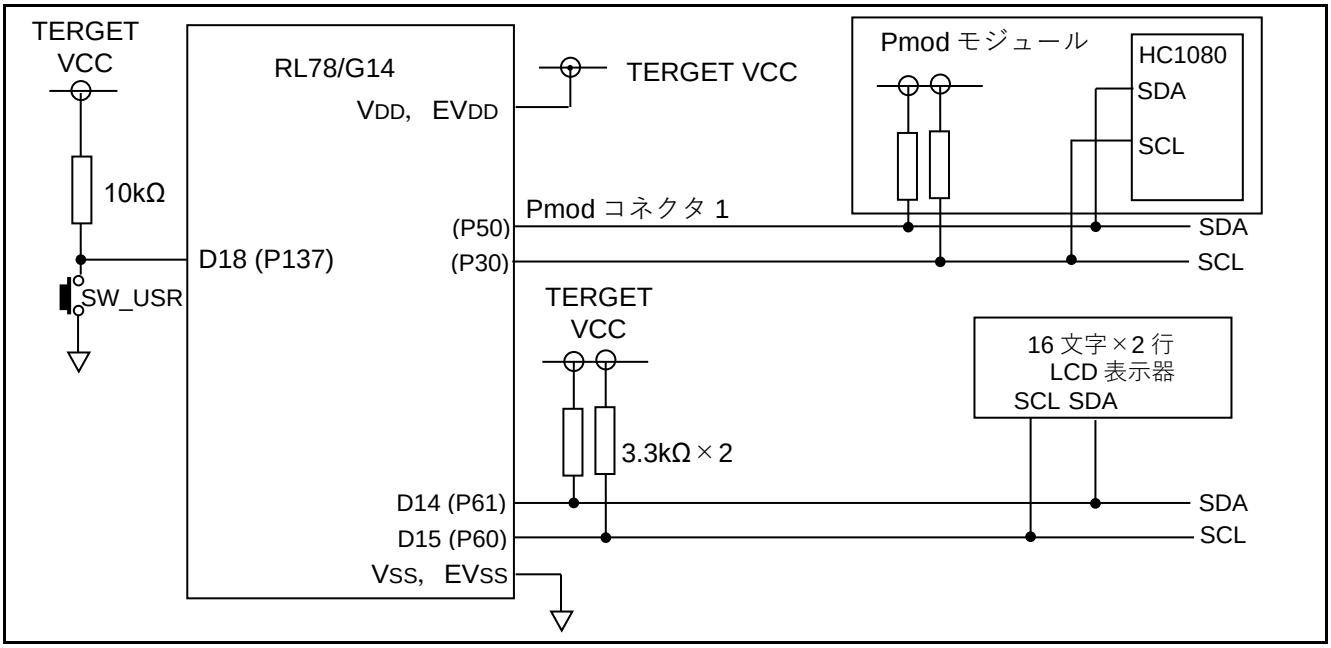

図 4.1 ハードウェア構成例

注意　この回路イメージは接続の概要を示す為に簡略化しています。
電源電圧は USB から 3.3V を供給しています。

4.2 使用端子一覧

使用端子と機能を表 4.1 に示します。

表 4.1　使用端子と機能

端子	ポート名	入出力	機能
D14	P61	入出力	SDA (I2C バスのデータ信号)
D15	P60	入出力	SCL (I2C バスのクロック信号)
-	P50	入出力	Pmod コネクタ 1 の SDA (I2C バスのデータ信号)
-	P30	入出力	Pmod コネクタ 1 の SCL (I2C バスのクロック信号)
-	P14	入出力	Pmod コネクタ 2 の SDA (I2C バスのデータ信号)
-	P15	入出力	Pmod コネクタ 2 の SCL (I2C バスのクロック信号)
D18	P137	入力	スイッチ (SW_USR) 入力

5. ソフトウェア説明

5.1 動作概要

本アプリケーションノートでは、初期設定 (端子の設定) が完了して、メイン処理 (loop) が起動すると、LCD 表示器は初期表示状態になります。

1 分ごとの定周期もしくはスイッチ押下により、温湿度センサ HDC1080 からデータを取得し、取得データから温度と湿度を計算し、温度と湿度を LCD 表示器に表示します。

下記①～②に詳細を記載します。

① setup 関数で、使用する端子の設定を行います。

- オンボードの SW_USR スイッチの読み取り用端子 (swPin) をデジタル入力に設定します。
- D14、D15 端子を使用した I2C バス制御のために IICA0 をマスタに設定します。
- Pmod コネクタ 1 を使用した I2C バス制御のために IIC00 を設定します。
- I2C バス接続の LCD 表示器を初期化し、初期表示状態にします。

② loop 関数で、メイン処理を行います。

- 起動時からのミリ秒単位での経過時間のビット 15～ビット 4 の 12 ビット (16 ミリ秒単位) を得ます。
- 取得データが古いデータ (old_time) から変化したかを確認します。
- 変化していない場合、処理を終了して loop 関数の先頭に戻ります。
- 変化 (16 ミリ秒経過) していた場合、old_time を取得データに変更します。
- 定周期 1 分 (0xEA6) のカウンタをカウントアップします。
- D18 に接続されたスイッチの状態をチェックします。
- スイッチが押されていない、かつ 1 分経過していない場合、loop 関数の先頭に戻ります。[注]
- スイッチが押された、もしくは 1 分経過した場合、温度と湿度を測定します。[注]
- 温湿度センサ HDC1080 のスタンバイを解除して計測を開始します。
  - ◆ データが安定するまで、約 16 ミリ秒待ちます。
  - ◆ センサのデータを読み出します。
  - ◆ 読み出したデータから、温度と湿度を算出します。
- 算出結果を LCD 表示器に転送します。
- loop 関数の先頭に戻ります。

注 起動直後は、センサで温度と湿度を測定します。

5.2 定数一覧

表 5.1 にサンプルコードで使用する定数を示します。

表 5.1 サンプルコードで使用する定数

定数名	設定値	内容
swPin	18	SW_USR を読む端子の番号
DUMMY_DATA	0xFF	マスタ受信時の受信起動用書き込みデータ
RELEASE	1	通信完了でストップコンディション生成を指定
RESTART	0	通信完了でリスタートコンディション生成を指定
SLADDR_HC1080	0x40	センサのスレーブアドレス (7 ビット)
SLADDR_LCD	0x50	LCD 表示器のスレーブアドレス (7 ビット)
COMBYTE	0x00	LCD 表示器へのコマンド転送指定データ
DATABYTE	0x80	LCD 表示器へのデータ転送指定データ
CLRDISP	0x01	LCD 表示器への表示クリアコマンド
HOMEPOSI	0x02	LCD 表示器のカーソルをホームポジションへ移動
LCD_Mode	0x38	文字は 5×8 ドット、表示は 2 行を指定
DISPON	0x0F	カーソル点滅、表示オン
ENTRY_Mode	0x06	1 文字転送で表示位置を移動
LOOPLIMIT	1000	スタートやストップの検出限界を 1000 回に指定
SUCCESS	0x00	I2C バスの処理は正常終了
BUS_FREE	0x00	I2C バスは未使用
BUS_ERROR	0x8F	I2C バスの確保に失敗
GET_BUS	0x10	I2C バスを確保
GET_BUS4TX	0x20	送信のために I2C バスを確保
TX_MODE	0x30	送信モード
TX_END	0x40	送信完了
GET_BUS4RX	0x50	受信のために I2C バスを確保
RX_MODE	0x60	受信モード
RX_END	0x70	受信完了
BUFF_OVER	0x81	送信データ数がバッファ容量を超えた
NO_SLAVE	0x82	該当スレーブなし
NO_ACK	0x83	送信データに NACK 応答
NO_DATA	0x84	受信データ数が 0
MINUTE	60000/16	16 ミリ秒をカウントして 1 分を得る
TX_DELAY1	1	Pmod コネクタ 1 の Wire1 送信間隔 (1μ秒)
RX_DELAY1	10	Pmod コネクタ 1 の Wire1 受信間隔 (10μ秒)
TX_DELAY2	1	Pmod コネクタ 2 の Wire2 送信間隔 (1μ秒)
RX_DELAY2	10	Pmod コネクタ 2 の Wire2 受信間隔 (10μ秒)

5.3 変数一覧

表 5.2～表 5.4 にグローバル変数を示します。

表 5.2 グローバル変数(1/3)

型	変数名	内容	使用関数[注]
unsigned int	old_time	前回の起動からの経過時間 (ミリ秒単位)	loop()
unsigned char	hdc1080_buff[4]	センサからの読出しデータ用バッファ	loop()
unsigned char	humid	湿度データ	loop()
unsigned int	temp	0.1℃単位の温度データ	loop()
char	disp_line1[40]	LCD 表示器への表示データ (1 行目)	loop()
char	disp_line2[40]	LCD 表示器への表示データ (2 行目)	loop()
int	count16ms	16 ミリ秒をカウントして 1 分を作る	loop()
char	sw_work	16 ミリ秒ごとのスイッチの状態確認用	loop()
unsigned char	g_lcd_command[2]	LCD 表示器へのコマンド設定用	set_command()
unsigned char	g_lcd_data[2]	LCD 表示器へのデータ設定用	set_dat()
uint8_t	gp_tx_set	送信用バッファへの書き込みポインタ (最大 255)	Wire_begin(), Wire_beginTransmission(), Wire_write()
uint8_t	gp_tx_get	送信用バッファからの読出しポインタ	Wire_begin(), Wire_beginTransmission(), r_IICA0_interrupt()
uint8_t	g_tx_buff[256]	送信用バッファ	Wire_write(), r_IICA0_interrupt()
uint8_t	gp_rx_set	受信用バッファへの データ書き込みポインタ (最大 255)	Wire_begin(), Wire_requestFrom(), r_IICA0_interrupt()
uint8_t	gp_rx_get	受信用バッファからの データ読み出しポインタ	Wire_begin(), Wire_requestFrom(), Wire_read()
uint8_t	g_rx_buff[256]	受信用バッファ	r_IICA0_interrupt(), Wire_read()
uint16_t	g_rx_num	受信データ数	Wire_requestFrom(), r_IICA0_interrupt()
uint8_t	sladdr8	8 ビットのスレーブアドレス	Wire_beginTransmission(), Wire_requestFromSub(), Wire_requestFromb()
uint8_t	g_stop_flag	終了時のストップコンディション生成フラグ 0：終了時に何もしない 1：終了時にストップコンディション生成	Wire_endTransmission(), Wire_requestFrom(), r_IICA0_interrupt(), r_operation_end()
uint8_t	g_status	IICA0 の状態フラグ 0x00 : BUS FREE 0x8F : BUS Error 0x10 : Get bus 0x20 : Get bus to transmit 0x30 : Transmit operation 0x40 : Transmit end 0x50 : Get bus to receive 0x60 : Receive operation 0x70 : Receive end 0x81 : Data size over buffer size 0x82 : NACK for slave address 0x83 : No ACK for data	r_IICA0_interrupt(), Wire_beginTransmission(), Wire_endTransmission(), Wire_requestFromb(), Wire_requestFromSub(), r_IICA0_interrupt(), r_operation_end()
uint8_t	g_erflag	0x00：成功 0x01：バッファオーバーフロー 0x02：スレーブなし 0x03：送信データに NACK 応答 0x04：その他のエラー	Wire_endTransmission()

注 Arduino の API ではなく内部の処理関数名で示しています。

表 5.3 グローバル変数(2/3)

型	変数名	内容	使用関数[注]
uint8_t	gp_tx1_set	送信用バッファへの書き込みポインタ (最大 255)	Wire1_begin(), Wire1_beginTransmission(), Wire1_write()
uint8_t	gp_tx1_get	送信用バッファからの読出しポインタ	Wire1_begin(), Wire1_beginTransmission(), r_IIC00_interrupt()
uint8_t	g_tx1_buff[256]	送信用バッファ	Wire1_write(), r_IIC00_interrupt()
uint8_t	gp_rx1_set	受信用バッファへの データ書き込みポインタ (最大 255)	Wire1_begin(), Wire1_requestFrom(), r_IIC00_interrupt()
uint8_t	gp_rx1_get	受信用バッファからの データ読み出しポインタ	Wire1_begin(), Wire1_requestFrom(), Wire1_read()
uint8_t	g_rx1_buff[256]	受信用バッファ	r_IIC00_interrupt(), Wire1_read()
uint16_t	g_rx1_num	受信データ数	Wire1_requestFrom(), r_IIC00_interrupt()
uint8_t	sladdr8_1	8 ビットのスレーブアドレス	Wire1_beginTransmission(), Wire1_requestFromSub(), Wire1_requestFromb()
uint8_t	g_stop_flag_1	終了時のストップコンディション生成フラグ 0：終了時リスタートコンディション生成 1：終了時ストップコンディション生成	Wire1_endTransmission(), Wire1_requestFrom(), r_IIC00_interrupt(), r_operation_end_1()
uint8_t	g_status_1	IIC00 の状態フラグ 0x00 : BUS FREE 0x8F : BUS Error 0x10 : Get bus 0x20 : Get bus to transmit 0x30 : Transmit operation 0x40 : Transmit end 0x50 : Get bus to receive 0x60 : Receive operation 0x70 : Receive end 0x81 : Data size over buffer size 0x82 : NACK for slave address 0x83 : No ACK for data	Wire1_beginTransmission(), Wire1_endTransmission(), Wire1_requestFromb(), Wire1_requestFromSub(), r_IIC00_interrupt(), r_operation_end_1()
uint8_t	g_erflag_1	0x00：成功 0x01：バッファオーバーフロー 0x02：スレーブなし 0x03：送信データに NACK 応答 0x04：その他のエラー	Wire1_endTransmission()

表 5.4 グローバル変数(3/3)

型	変数名	内容	使用関数[注]
uint8_t	gp_tx2_set	送信用バッファへの書き込みポインタ (最大 255)	Wire2_begin(), Wire2_beginTransmission(), Wire2_write()
uint8_t	gp_tx2_get	送信用バッファからの読出しポインタ	Wire2_begin(), Wire2_beginTransmission(), r_IIC20_interrupt()
uint8_t	g_tx2_buff[256]	送信用バッファ	Wire2_write(), r_IIC20_interrupt()
uint8_t	gp_rx2_set	受信用バッファへの データ書き込みポインタ (最大 255)	Wire2_begin(), Wire2_requestFrom(), r_IIC20_interrupt()
uint8_t	gp_rx2_get	受信用バッファからの データ読み出しポインタ	Wire2_begin(), Wire2_requestFrom(), Wire2_read()
uint8_t	g_rx2_buff[256]	受信用バッファ	r_IIC20_interrupt(), Wire2_read()
uint16_t	g_rx2_num	受信データ数	Wire2_requestFrom(), r_IIC20_interrupt()
uint8_t	sladdr8_2	8 ビットのスレーブアドレス	Wire2_beginTransmission(), Wire2_requestFromSub(), Wire2_requestFromb()
uint8_t	g_stop_flag_2	終了時のストップコンディション生成フラグ 0：終了時リスタートコンディション生成 1：終了時ストップコンディション生成	Wire2_endTransmission(), Wire2_requestFrom(), r_IIC20_interrupt(), r_operation_end_2()
uint8_t	g_status_2	IIC00 の状態フラグ 0x00 : BUS FREE 0x8F : BUS Error 0x10 : Get bus 0x20 : Get bus to transmit 0x30 : Transmit operation 0x40 : Transmit end 0x50 : Get bus to receive 0x60 : Receive operation 0x70 : Receive end 0x81 : Data size over buffer size 0x82 : NACK for slave address 0x83 : No ACK for data	Wire2_beginTransmission(), Wire2_endTransmission(), Wire2_requestFromb(), Wire2_requestFromSub(), r_IIC20_interrupt(), r_operation_end_2()
uint8_t	g_erflag_2	0x00：成功 0x01：バッファオーバーフロー 0x02：スレーブなし 0x03：送信データに NACK 応答 0x04：その他のエラー	Wire2_endTransmission()

5.4 関数一覧

表 5.5～表 5.6 に関数一覧を示します。

表 5.5 関数一覧 (1/2)

関数名	概要
loop	メイン処理 (スケッチ)
setup	初期化関数 (スケッチ)
pinMode	端子の動作モード (入力モード / 出力モード / 内蔵プルアップ抵抗を有効にした入力モード) 指定
digitalWrite	端子にデータを出力します。
digitalRead	端子状態を読み出します。
micros	プログラム実行開始から現在までの時間をマイクロ秒単位で返します。
millis	プログラム実行開始から現在までの時間をミリ秒単位で返します。
delay	ミリ秒単位でプログラムを指定した時間だけ止めます。
delayMicroseconds	マイクロ秒単位でプログラムを指定した時間だけ止めます。
Wire.begin	I2C ライブラリを初期化してマスタとして接続します。
Wire.requestFrom	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。
Wire_requestFromS	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。受信完了でストップコンディション生成を指定可能です。 Wire.requestFrom の内部処理用関数です。
Wire_requestFromSub	Wire.requestFrom の内部処理用関数です。
Wire.available	Wire.read で読み出せる受信バッファ中のバイト数を返します。
Wire.read	受信バッファからデータを読み出します。
Wire.beginTransmission	指定したスレーブへの送信の準備を行います。
Wire.write	送信するデータを送信バッファに書き込みます。
Wire_writec	1 キャラクタのデータを送信バッファに追加します。 Wire.write の内部処理用関数です。
Wire_writeb	データブロックを送信バッファに追加します。 Wire.write の内部処理用関数です。
Wire.endTransmission	バッファにセットされている送信データを実際に I2C バスで送信します。送信完了時にストップコンディション生成を指定可能です。
init_LCD	LCD 表示器の初期化を行います。
print_LCD	16 文字 2 行分の文字を LCD 表示器に表示します。
move_cursor	LCD 表示器の表示データをセットするカーソル位置を指定します。
set_2digit	1 バイトのデータを 2 桁で表示します。
set_1digit	下位ビットのデータを 1 桁で表示します。
set_command	LCD 表示器にコマンドを送ります。
set_data	LCD 表示器に表示データを送ります。

表 5.6 関数一覧 (2/2)

関数名	概要
Wire1.begin	Pmod コネクタ 1 の I2C ライブラリを初期化してマスタとして接続します。
Wire1.requestFrom	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。
Wire1_requestFromS	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。受信完了でストップコンディション生成を指定可能です。 Wire1.requestFrom の内部処理用関数です。
Wire1_requestFromSub	Wire1.requestFrom の内部処理用関数です。
Wire1.available	Wire1.read で読み出せる受信バッファ中のバイト数を返します。
Wire1.read	受信バッファからデータを読み出します。
Wire1.beginTransmission	指定したスレーブへの送信の準備を行います。
Wire1.write	送信するデータを送信バッファに書き込みます。
Wire1_writec	1 キャラクタのデータを送信バッファに追加します。 Wire1.write の内部処理用関数です。
Wire1_writeb	データブロックを送信バッファに追加します。 Wire1.write の内部処理用関数です。
Wire1.endTransmission	バッファにセットされている送信データを実際に I2C バスで送信します。送信完了時にストップコンディション生成を指定可能です。
Wire2.begin	Pmod コネクタ 2 の I2C ライブラリを初期化してマスタとして接続します。
Wire2.requestFrom	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。
Wire2_requestFromS	指定したスレーブからのデータ読み出しを起動します。読み出しは割り込みで処理します。受信完了でストップコンディション生成を指定可能です。 Wire2.requestFrom の内部処理用関数です。
Wire2_requestFromSub	Wire2.requestFrom の内部処理用関数です。
Wire2.available	Wire2.read で読み出せる受信バッファ中のバイト数を返します。
Wire2.read	受信バッファからデータを読み出します。
Wire2.beginTransmission	指定したスレーブへの送信の準備を行います。
Wire2.write	送信するデータを送信バッファに書き込みます。
Wire2_writec	1 キャラクタのデータを送信バッファに追加します。 Wire2.write の内部処理用関数です。
Wire2_writeb	データブロックを送信バッファに追加します。 Wire2.write の内部処理用関数です。
Wire2.endTransmission	バッファにセットされている送信データを実際に I2C バスで送信します。送信完了時にストップコンディション生成を指定可能です。

5.5 関数仕様

サンプルコードの関数仕様を示します。

[関数名] loop	
概　要	メイン関数
ヘッダ	AR_LIB_PORT.h、AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、AR_SKETCH.h、r_cg_userdefine.h、LCD_LIB.h
宣　言	void loop(void);
説　明	起動後、16 ミリ秒ごとにスイッチの状態を確認する。スイッチが押されている、もしくは 1 分経過したら、センサ HDC1080 を起動する。センサ起動後、約 16 ミリ秒経過したら、センサの計測結果を読み出す。計測結果から湿度と温度を計算して LCD 表示器に湿度と温度を表示する。
引　数	なし
リターン値	なし

[関数名] setup	
概　要	初期化関数
ヘッダ	AR_LIB_PORT.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h
宣　言	void setup(void);
説　明	プログラム (スケッチ) で使用する端子、IICA0 および LCD 表示器の設定を行う。
引　数	なし
リターン値	なし

[関数名] pinMode		
概　要	端子機能設定関数	
ヘッダ	AR_LIB_PORT.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	void pinMode(uint8_t pin,uint8_t mode);	
説　明	第 1 引数で示された端子を第 2 引数で示されたモードに設定する。	
引　数	uint8_t pin	指定する端子の番号
	uint8_t mode	OUTPUT/INPUT/INPUT_PULLUP で端子のモードを指定
リターン値	なし	

[関数名] digitalWrite		
概　要	端子へのデジタル・データ出力関数	
ヘッダ	AR_LIB_PORT.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	void digitalWrite(uint8_t pin, uint8_t value);	
説　明	第 1 引数で示す端子に、第 2 引数で示すデータを出力する	
引　数	uint8_t pin	出力する端子の番号
	uint8_t value	出力するデータ (HIGH/LOW)
リターン値	なし	

[関数名]	digitalRead	
概　要	端子からのデジタル データの読出し関数	
ヘッダ	AR_LIB_PORT.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint8_t digitalRead(uint8_t pin);	
説　明	引数で指定された端子の状態を読み出す。	
引　数	uint8_t pin	読み出す端子の番号
リターン値	uint8_t	読み出したデータ (HIGH/LOW)

[関数名]	micros	
概　要	マイクロ秒単位での経過時間を得る	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint32_t micros(void);	
説　明	起動してからのマイクロ秒単位の経過時間を戻す。	
引　数	なし	
リターン値	uint32_t	マイクロ秒単位の経過時間

[関数名]	millis	
概　要	ミリ秒単位での経過時間を得る	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint32_t millis(void);	
説　明	起動してからのミリ秒単位の経過時間を戻す。	
引　数	なし	
リターン値	uint32_t	ミリ秒単位の経過時間

[関数名]	delay	
概　要	ミリ秒単位でのウエイト関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint32_t delay(uint32_t time);	
説　明	引数で指定したミリ秒単位の時間を待つ。	
引　数	uint32_t time	ウエイト時間 (ミリ秒単位)
リターン値	なし	

[関数名]	delayMicroseconds	
概　要	マイクロ秒単位でのウエイト関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	void delayMicroseconds(uint32_t time);	
説　明	引数で指定したマイクロ秒単位の時間を待つ。	
引　数	uint32_t time	ウエイト時間 (マイクロ秒単位)
リターン値	なし	

[関数名]	Wire.begin	
概　要	I2C バスの使用準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire.begin(void);	
説　明	IICA0 を初期化して、I2C バスの使用準備を行う。	
引　数	なし	
リターン値	なし	

[関数名]	Wire.requestFrom	
概　要	スレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
	uint8_t stop	終了時の処理（省略時はバスを解放する） 0: 通信完了時に何もせず、バス確保を継続 1: ストップコンディション生成 (バスを解放する)
リターン値	uint8_t	0x00: 正常 0x01: バッファ オーバー 0x04: その他のエラー
備　考	g_status：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する。 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗。 実行中に I2C バスへの通信起動処理は禁止。	

[関数名]	Wire_requestFromS	
概　要	スレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire_requestFromS(uint8_t saddr7, uint16_t bytes);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、ストップコンディションを生成してバスを解放する。 （Wire.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
リターン値	uint8_t	0x00: 正常 0x01: バッファ オーバー 0x04: その他のエラー
備　考	g_status：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する。 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗。 実行中に I2C バスへの通信起動処理は禁止。	

[関数名]	Wire_requestFromSub	
概　要	スレーブからの受信準備するための内部関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire_requestFromSub(uint8_t saddr7, uint16_t bytes , uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。 （Wire.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
	uint8_t stop	0: 通信完了時に何もせず、バス確保を継続 1: ストップコンディション生成 (バスを解放する)
リターン値	なし	
備　考	g_status：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する。 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗。 g_erflag：エラーフラグ 0x00：正常、0x01：バッファ オーバー フロー、0x04：その他のエラー 実行中に I2C バスへの通信起動処理は禁止。	

[関数名]	Wire.available	
概　要	読み出せるデータ数を戻す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t Wire.available(void);	
説　明	Wire.requestFrom 関数で受信し、バッファに格納したデータのバイト数を戻す。	
引　数	なし	
リターン値	uint8_t	バッファ中の読み出し可能なデータ数

[関数名]	Wire.read	
概　要	受信バッファからデータを読み出す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t Wire.read(void);	
説　明	バッファに格納されたデータを読み出す。	
引　数	なし	
リターン値	uint8_t	バッファから読み出したデータ (または 0x00)

[関数名]	Wire.beginTransmission	
概　要	スレーブへの送信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire.beginTransmission(uint8_t saddr7);	
説　明	スレーブアドレスを 8 ビットアドレスに変換して変数 sladdr8 に格納し、スタートコンディションを生成してバスを確保する。	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
リターン値	uint8_t	0x00: 正常 0x04: その他のエラー
備　考	g_erflag：通信ステータス 0x00 なら起動成功。 0x04 なら I2C バスの確保失敗。	

[関数名]	Wire.write	
概　要	送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t Wire.write(uint8_t data);　uint8_t Wire.write(uint8_t *buff, uint8_t bytes);	
説　明	引数 1 の 1 キャラクタまたは、引数 2 のデータブロックを送信バッファに格納する。	
引　数 1	uint8_t data	送信するデータ
引　数 2	uint8_t *buff	送信するデータブロック
	uint8_t byte	送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag が 0x01 なら送信バッファがあふれたことを示す。0x04 なら I2C バスが確保できていないことを示す。	

[関数名]	Wire_writec	
概　要	送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t Wire_writec(uint8_t data);	
説　明	引数の 1 キャラクタを送信バッファに格納する。 （Wire.write の 1 キャラクタ処理用内部関数）	
引　数	uint8_t data	送信するデータ
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag が 0x01 なら送信バッファがあふれたことを示す。0x04 なら I2C バスが確保できていないことを示す。	

[関数名]	Wire_writeb	
概　要	送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t Wire_writeb(uint8_t *buff, uint8_t bytes);	
説　明	引数で指定されたブロックのデータを送信バッファに格納する。 （Wire.write のブロック処理用内部関数）	
引　数	uint8_t *buff	送信するデータブロックのアドレス
	uint8_t bytes	送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag が 0x01 なら送信バッファがあふれたことを示す。0x04 なら I2C バスが確保できていないことを示す。	

[関数名]	Wire.endTransmission	
概　要	スレーブへのデータ送信処理関数関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void Wire_ endTransmission(uint8_t STOP);	
説　明	スレーブに送信バッファのデータを送信する。	
引　数	uint8_t STOP	送信完了時の処理 0：送信完了時に何もせず、バス確保を継続 1：送信完了時にバスを解放
リターン値	uint8_t	送信結果 0：成功 1：データ数がバッファサイズを超えた 2：スレーブアドレスに NACK 応答 3：送信データに NACK 応答 4：その他のエラー
備　考		

[関数名]	init_LCD	
概　要	LCD 表示器の初期設定	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t init_LCD(void);	
説　明	LCD 表示器を 16 文字 2 行に設定し表示をクリアする。	
引　数	なし	
リターン値	uint8_t	通信結果 0：成功 2：LCD 表示器からの応答なし
備　考		

[関数名]	print_LCD	
概　要	LCD 表示器へ 1 画面分のデータ (16 文字×2 行) をセット	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void print_LCD(uint8_t *point, uint8_t *point2);	
説　明	引数で渡されたアドレスからの 32 文字を LCD 表示器に 2 行で表示する	
引　数	uint8_t *point	disp_line1 の先頭を指定する
	uint8_t *point2	disp_line2 の先頭を指定する
リターン値	なし	
備　考		

[関数名]	move_cursor	
概　要	LCD 表示器のカーソル位置設定	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void move_cursor(uint8_t col, uint8_t row);	
説　明	引数で渡された位置にカーソルを移動する。	
引　数	uint8_t col	行の何文字目かを指定
	uint8_t row	何行目かを指定
リターン値	なし	
備　考	実行後 60 マイクロ秒は次の書き込みは禁止	

[関数名]	set_2digit	
概　要	LCD 表示器に数値を 2 桁の ASCII コードで表示する	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void set_2digit(uint8_t datacode);	
説　明	引数で渡された 16 進数または BCD のデータを 2 桁の ASCII コードに変換して LCD 表示器に表示データとして送信する。	
引　数	uint8_t datacode	LCD 表示器への 8 ビットのデータコード（16 進数または BCD データ）
リターン値	なし	
備　考		

[関数名]	set_1digit	
概　要	LCD 表示器に数値を 1 桁の ASCII コードで表示する	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	void set_1digit(uint8_t datacode);	
説　明	引数で渡されたデータの下位 4 ビットを ASCII コードに変換して LCD 表示器に表示データとして送信する。	
引　数	uint8_t datacode	LCD 表示器へのデータコード（16 進数または BCD データ）
リターン値	なし	
備　考		

[関数名]	set_command	
概　要	LCD 表示器にコマンドを送信	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t set_command(uint8_t lcd_command);	
説　明	引数で渡されたコマンドコードを LCD 表示器にコマンドとして送信する。	
引　数	uint8_t lcd_command	LCD 表示器へのコマンドコード
リターン値	uint8_t	通信結果 0：成功 2：LCD 表示器からの応答なし
備　考	実行後 60 マイクロ秒は次の書き込みは禁止	

[関数名]	set_data	
概　要	LCD 表示器に表示データを送信	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE.h、r_cg_userdefine.h	
宣　言	uint8_t set_data(uint8_t datacode);	
説　明	引数で渡されたデータコードを LCD 表示器にデータとして送信する。	
引　数	uint8_t datacode	LCD 表示器へのデータコード
リターン値	uint8_t	通信結果 0：成功 2：LCD 表示器からの応答なし
備　考	実行後 60 マイクロ秒は次の書き込みは禁止	

[関数名]	Wire1.begin	
概　要	Pmod1 コネクタの I2C バスの使用準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1.begin(void);	
説　明	IIC00 を初期化して、I2C バスの使用準備を行う。	
引　数	なし	
リターン値	なし	

[関数名]	Wire1.requestFrom	
概　要	Pmod1 コネクタの I2C バスを使用したスレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。	
引　数	uint8_t saddr7 uint16_t bytes uint8_t stop	7 ビットのスレーブアドレス 受信するデータ数 終了時の処理（省略時はバスを解放する） 0: 通信完了時に何もせず、バス確保を継続 1:ストップコンディション生成 (バスを解放する)
リターン値	uint8_t	0x00 ：正常 0x01 ：バッファ オーバー 0x04 ：その他のエラー
備　考	g_status_1：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire1_requestFromS	
概　要	Pmod1 コネクタの I2C バスのスレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1_requestFromS(uint8_t saddr7, uint16_t bytes);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、ストップコンディションを生成してバスを解放する。 （Wire1.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
リターン値	uint8_t	0x00 ：正常 0x01 ：バッファ オーバー 0x04 ：その他のエラー
備　考	g_status_1：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire1_requestFromSub	
概　要	Pmod1 コネクタの I2C バスのスレーブからの受信準備するための内部関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1_requestFromSub(uint8_t saddr7, uint16_t bytes , uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。 （Wire1.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
	uint8_t stop	終了時の処理 0: 送信完了時に何もせず、バス確保を継続 1:ストップコンディション生成 (バスを解放する)
リターン値	なし	
備　考	g_status_1：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 g_erflag_1：エラーフラグ 0x00：正常、0x01：バッファ オーバー フロー、0x04：その他のエラー 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire1.available	
概　要	Wire1 の受信バッファから読み出せるデータ数を戻す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	uint8_t Wire1.available(void);	
説　明	Wire.requestFrom 関数で受信し、バッファに格納したデータのバイト数を戻す。	
引　数	なし	
リターン値	uint8_t	バッファ中の読み出し可能なデータ数

[関数名]	Wire1.read	
概　要	Wire1 の受信バッファからデータを読み出す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	uint8_t Wire1.read(void);	
説　明	バッファに格納されたデータを読み出す。	
引　数	なし	
リターン値	uint8_t	バッファから読み出したデータ (または 0x00)

[関数名]	Wire1.beginTransmission	
概　要	Pmod1 コネクタの I2C バスのスレーブへの送信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1.beginTransmission(uint8_t saddr7);	
説　明	スレーブアドレスを 8 ビットアドレスに変換して変数 sladdr8_1 に格納し、スタートコンディションを生成してバスを確保する。	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
リターン値	uint8_t	0x00 ：正常 0x04 ：その他のエラー
備　考	g_erflag_1：通信ステータス 0x00 なら起動成功。 0x04 なら I2C バスの確保失敗。	

[関数名]	Wire1.write	
概　要	Pmod1 コネクタの I2C バスのスレーブへの送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	uint8_t Wire1.write(uint8_t data);　uint8_t Wire1.write(uint8_t *buff, uint8_t bytes);	
説　明	引数 1 の 1 キャラクタまたは、引数 2 のデータブロックを送信バッファに格納する。	
引　数 1	uint8_t data	送信するデータ
引　数 2	uint8_t *buff	送信するデータブロック
	uint8_t byte	送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_1：通信ステータス 0x00 なら成功 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire1_writec	
概　要	Pmod1 コネクタの I2C バスのスレーブへの 1 バイトの送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	uint8_t Wire1_writec(uint8_t data);	
説　明	引数の 1 キャラクタを送信バッファに格納する。 （Wire1.write の 1 キャラクタ処理用内部関数）	
引　数	uint8_t data	送信するデータ
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_1：通信ステータス 0x00 なら成功 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire1_writeb	
概　要	Pmod1 コネクタの I2C バスのスレーブへのブロックデータ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	uint8_t Wire1_writeb(uint8_t *buff, uint8_t bytes);	
説　明	引数で指定されたブロックのデータを送信バッファに格納する。 （Wire1.write のブロック処理用内部関数）	
引　数	uint8_t *buff	送信するデータブロックのアドレス
	uint8_t bytes	送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_1：通信ステータス 0x00 なら正常 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire1.endTransmission	
概　要	Pmod1 コネクタの I2C バスのスレーブへのデータ送信処理関数関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE1.h、r_cg_userdefine.h	
宣　言	void Wire1.endTransmission(uint8_t STOP);	
説　明	スレーブに送信バッファのデータを送信する。	
引　数	uint8_t STOP	送信完了時の処理 0：送信完了時に何もせず、バス確保を継続 1：送信完了時にバスを解放
リターン値	uint8_t	送信結果 0：成功 1：データ数がバッファサイズを超えた 2：スレーブアドレスに NACK 応答 3：送信データに NACK 応答 4：その他のエラー
備　考		

[関数名]	Wire2.begin	
概　要	Pmod2 コネクタの I2C バスの使用準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2.begin(void);	
説　明	IIC20 を初期化して、I2C バスの使用準備を行う。	
引　数	なし	
リターン値	なし	

[関数名]	Wire2.requestFrom	
概　要	Pmod2 コネクタの I2C バスを使用したスレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2.requestFrom(uint8_t saddr7, uint16_t bytes, uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
	uint8_t stop	終了時の処理（省略時はバスを解放する） 0: 通信完了時に何もせず、バス確保を継続 1:ストップコンディション生成 (バスを解放する)
リターン値	uint8_t	0x00 ：正常 0x01 ：バッファ オーバー 0x04 ：その他のエラー
備　考	g_status_2：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire2_requestFromS	
概　要	Pmod2 コネクタの I2C バスのスレーブからの受信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2_requestFromS(uint8_t saddr7, uint16_t bytes);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、ストップコンディションを生成してバスを解放する。 （Wire2.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
リターン値	uint8_t	0x00 ：正常 0x01 ：バッファ オーバー 0x04 ：その他のエラー
備　考	g_status_2：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire2_requestFromSub	
概　要	Pmod2 コネクタの I2C バスのスレーブからの受信準備するための内部関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2_requestFromSub(uint8_t saddr7, uint16_t bytes , uint8_t stop);	
説　明	引数で示された条件での受信を行うためにスタートコンディションを生成、スレーブアドレスを送信する。以降の処理は割り込みでの処理となる。終了時には、第 3 引数で指定した処理を行う。 （Wire2.requestFrom の内部処理関数）	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
	uint16_t bytes	受信するデータ数
	uint8_t stop	終了時の処理 0: 通信完了時に何もせず、バス確保を継続 1:ストップコンディション生成 (バスを解放する)
リターン値	なし	
備　考	g_status_2：通信ステータス 0x50 なら起動成功。以降 0x60 (受信動作) 、0x70 (受信完了) と変化する 0x81 はバッファエラー、0x84 は受信データなし、0x8F なら起動失敗 g_erflag_2：エラーフラグ 0x00：正常、0x01：バッファ オーバー フロー、0x04：その他のエラー 実行中に I2C バスへの通信起動処理は禁止	

[関数名]	Wire2.available	
概　要	Wire2 の受信バッファから読み出せるデータ数を戻す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	uint8_t Wire2.available(void);	
説　明	Wire.requestFrom 関数で受信し、バッファに格納したデータのバイト数を戻す。	
引　数	なし	
リターン値	uint8_t	バッファ中の読み出し可能なデータ数

[関数名]	Wire2.read	
概　要	Wire2 の受信バッファからデータを読み出す関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	uint8_t Wire2.read(void);	
説　明	バッファに格納されたデータを読み出す。	
引　数	なし	
リターン値	uint8_t	バッファから読み出したデータ (または 0x00)

[関数名]	Wire2.beginTransmission	
概　要	Pmod2 コネクタの I2C バスのスレーブへの送信準備関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2.beginTransmission(uint8_t saddr7);	
説　明	スレーブアドレスを 8 ビットアドレスに変換して変数 sladdr8_2 に格納し、スタートコンディションを生成してバスを確保する。	
引　数	uint8_t saddr7	7 ビットのスレーブアドレス
リターン値	uint8_t	0x00 ：正常 0x04 ：その他のエラー
備　考	g_erflag_2：通信ステータス 0x00 なら起動成功。 0x04 なら I2C バスの確保失敗。	

[関数名]	Wire2.write	
概　要	Pmod2 コネクタの I2C バスのスレーブへの送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	uint8_t Wire2.write(uint8_t data);　uint8_t Wire2.write(uint8_t *buff, uint8_t bytes);	
説　明	引数 1 の 1 キャラクタまたは、引数 2 のデータブロックを送信バッファに格納する。	
引　数 1	uint8_t data	送信するデータ
引　数 2	uint8_t *buff uint8_t byte	送信するデータブロック 送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_2：通信ステータス 0x00 なら成功 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire2_writec	
概　要	Pmod2 コネクタの I2C バスのスレーブへの 1 バイトの送信データ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	uint8_t Wire2_writec(uint8_t data);	
説　明	引数の 1 キャラクタを送信バッファに格納する。 （Wire2.write の 1 キャラクタ処理用内部関数）	
引　数	uint8_t data	送信するデータ
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_2：通信ステータス 0x00 なら成功 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire2_writeb	
概　要	Pmod2 コネクタの I2C バスのスレーブへのブロックデータ設定関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	uint8_t Wire2_writeb(uint8_t *buff, uint8_t bytes);	
説　明	引数で指定されたブロックのデータを送信バッファに格納する。 （Wire2.write のブロック処理用内部関数）	
引　数	uint8_t *buff	送信するデータブロックのアドレス
	uint8_t bytes	送信するデータ数
リターン値	uint8_t	バッファのデータ数
備　考	g_erflag_2：通信ステータス 0x00 なら正常 0x01 なら送信バッファのあふれ、0x04 なら I2C バスが確保できていない	

[関数名]	Wire2.endTransmission	
概　要	Pmod2 コネクタの I2C バスのスレーブへのデータ送信処理関数関数	
ヘッダ	AR_LIB_TIME.h、r_cg_macrodriver.h、AR_LIB_WIRE2.h、r_cg_userdefine.h	
宣　言	void Wire2. endTransmission(uint8_t STOP);	
説　明	スレーブに送信バッファのデータを送信する。	
引　数	uint8_t STOP	送信完了時の処理 0：通信完了時に何もせず、バス確保を継続 1：送信完了時にバスを解放
リターン値	uint8_t	送信結果 0：成功 1：データ数がバッファサイズを超えた 2：スレーブアドレスに NACK 応答 3：送信データに NACK 応答 4：その他のエラー
備　考		

5.6 フローチャート

5.6.1 初期設定関数

図 5.1 に初期設定のフローを示します。

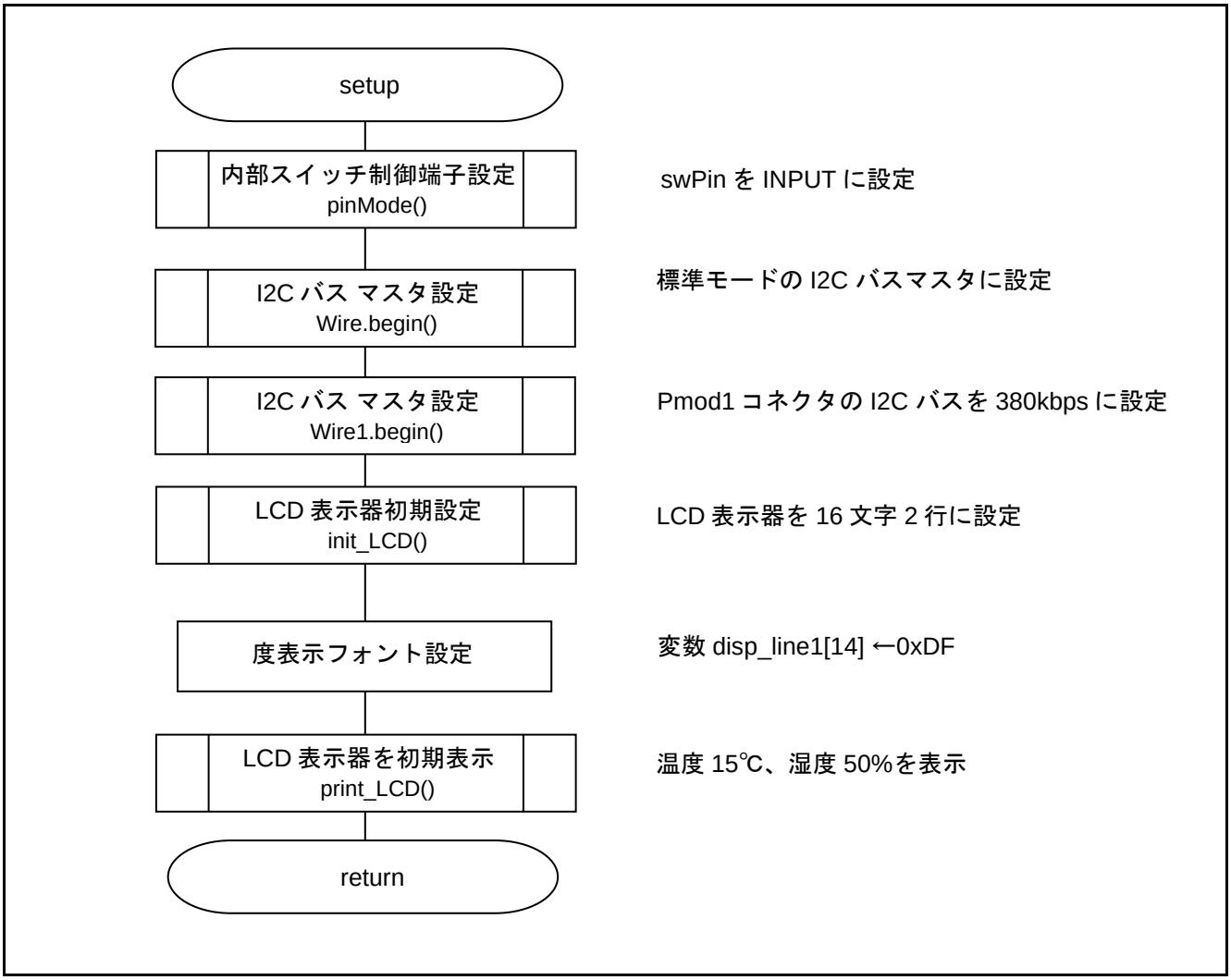

図 5.1 初期設定関数

5.6.2 メイン処理関数

図 5.2～図 5.5 にメイン処理関数のフローチャートを示します。

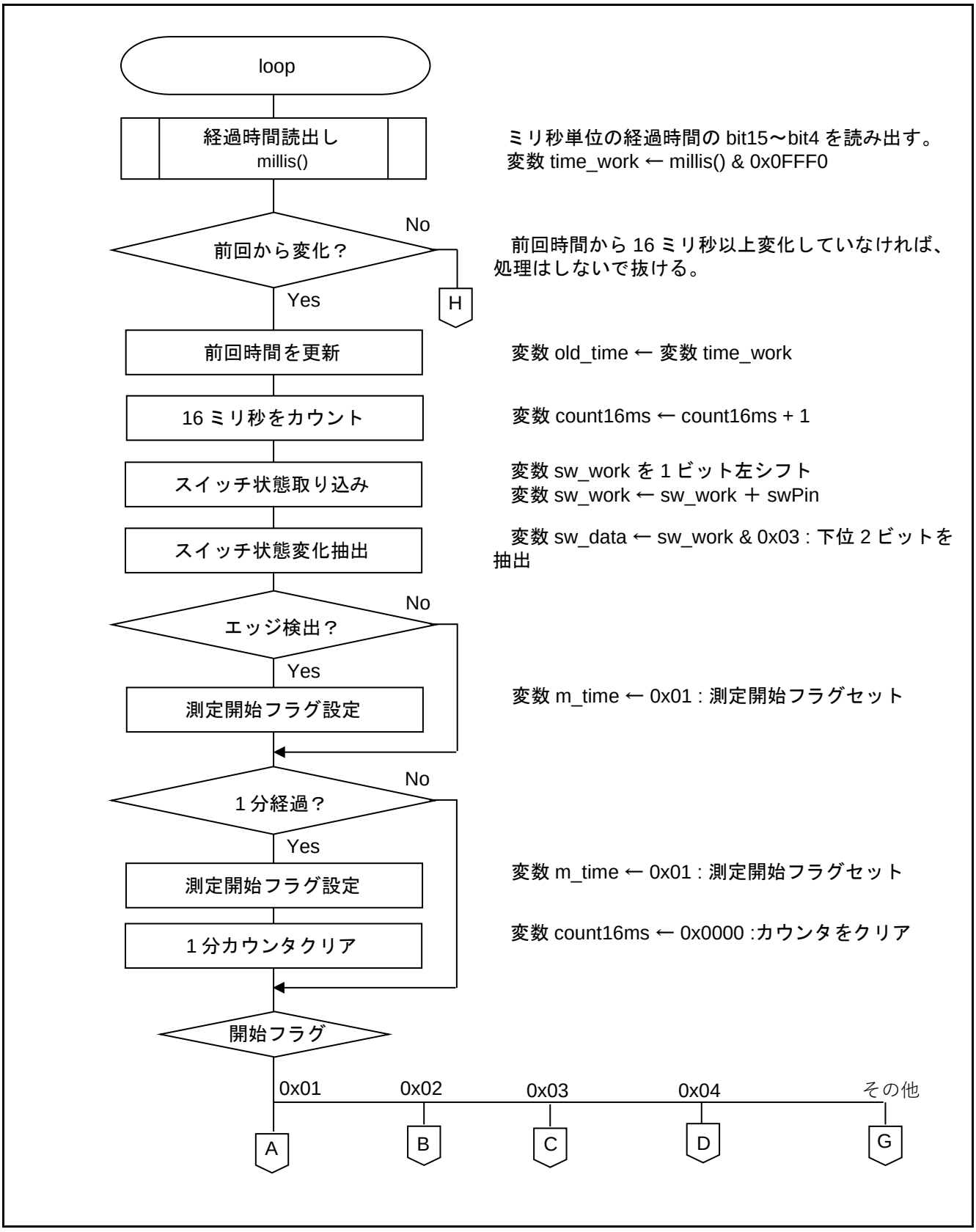

図 5.2 メイン関数 (1/4)

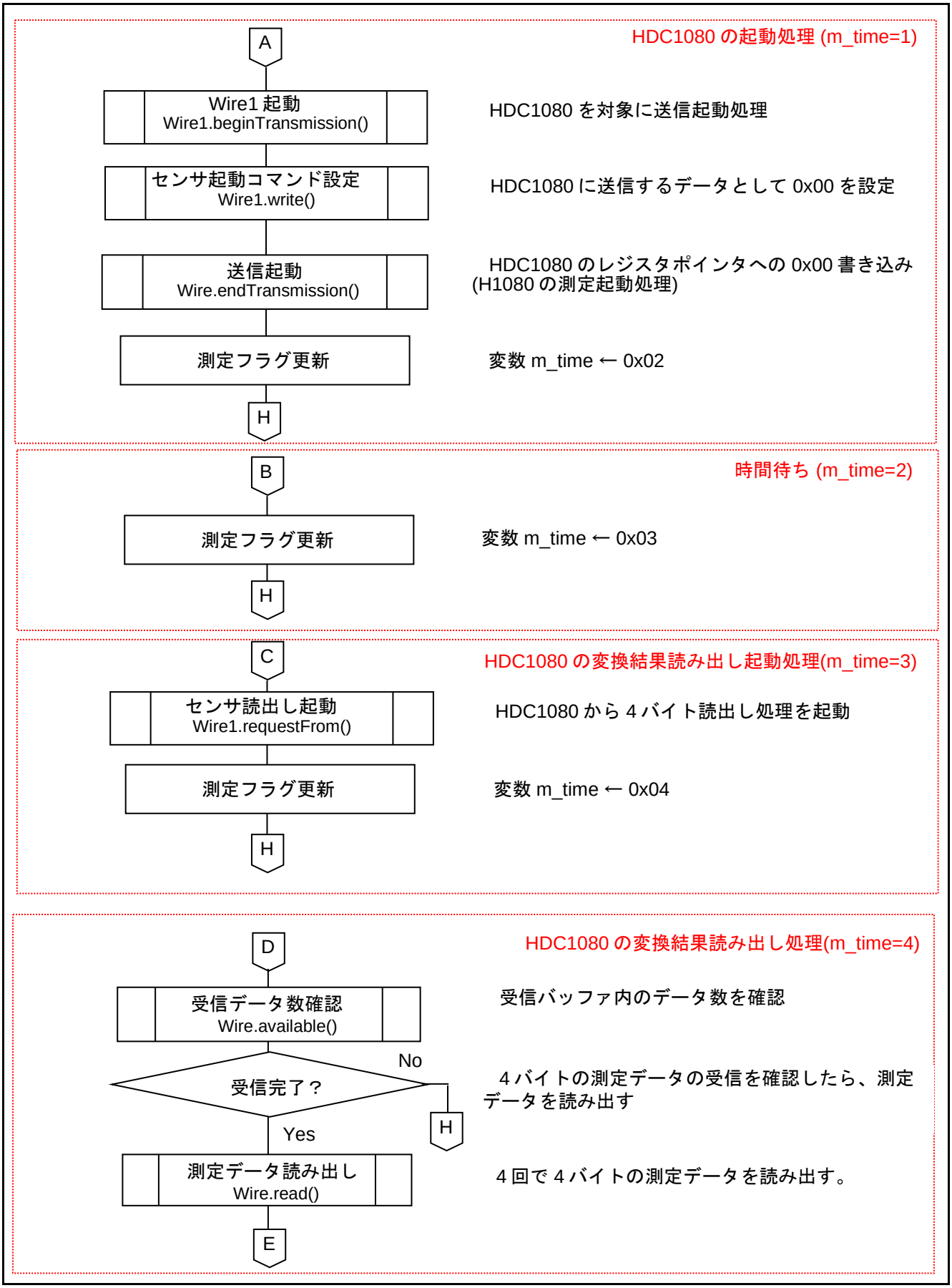
HDC1080 の起動処理 (m_time=1)
A
Wire1 起動
Wire1.beginTransmission()
HDC1080 を対象に送信起動処理
センサ起動コマンド設定
Wire1.write()
HDC1080 に送信するデータとして 0x00 を設定
送信起動
Wire.endTransmission()
HDC1080 のレジスタポインタへの 0x00 書き込み
(H1080 の測定起動処理)
測定フラグ更新
変数 m_time ← 0x02
H
時間待ち (m_time=2)
B
測定フラグ更新
変数 m_time ← 0x03
H
HDC1080 の変換結果読み出し起動処理(m_time=3)
C
センサ読出し起動
Wire1.requestFrom()
HDC1080 から 4 バイト読出し処理を起動
測定フラグ更新
変数 m_time ← 0x04
H
HDC1080 の変換結果読み出し処理(m_time=4)
D
受信データ数確認
Wire.available()
受信バッファ内のデータ数を確認
No
受信完了？
4 バイトの測定データの受信を確認したら、測定データを読み出す
H
Yes
測定データ読み出し
Wire.read()
4 回で 4 バイトの測定データを読み出す。
E

図 5.3 メイン関数 (2/4)

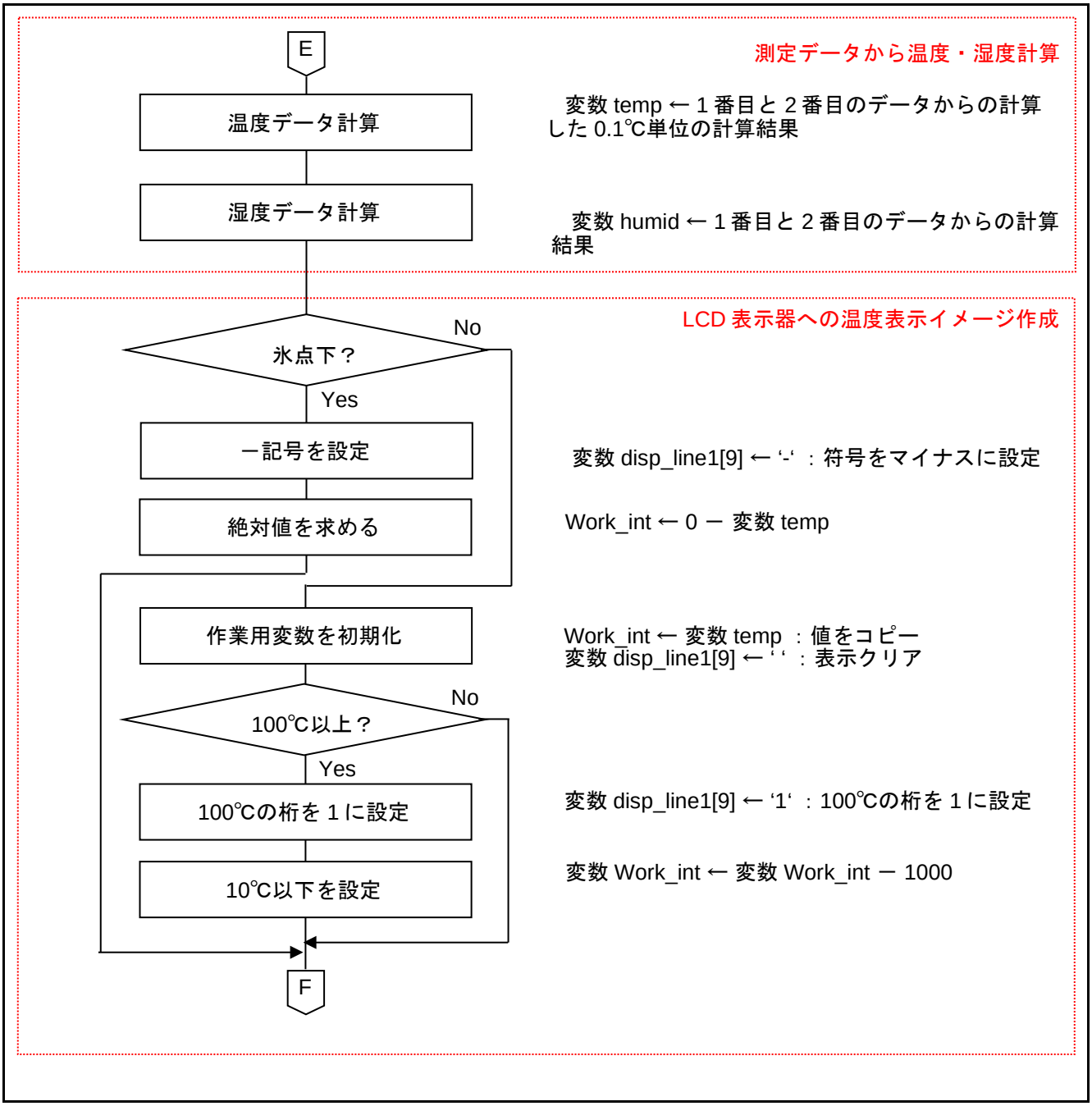
E
測定データから温度・湿度計算
温度データ計算
変数 temp ← 1 番目と 2 番目のデータからの計算した 0.1℃単位の計算結果
湿度データ計算
変数 humid ← 1 番目と 2 番目のデータからの計算結果
LCD 表示器への温度表示イメージ作成
No
氷点下？
Yes
ー記号を設定
変数 disp_line1[9] ← '-' ：符号をマイナスに設定
絶対値を求める
Work_int ← 0 ー 変数 temp
作業用変数を初期化
Work_int ← 変数 temp ：値をコピー
変数 disp_line1[9] ← ' ' ：表示クリア
No
100℃以上？
Yes
100℃の桁を 1 に設定
変数 disp_line1[9] ← '1' ：100℃の桁を 1 に設定
10℃以下を設定
変数 Work_int ← 変数 Work_int ー 1000
F

図 5.4 メイン関数 (3/4)

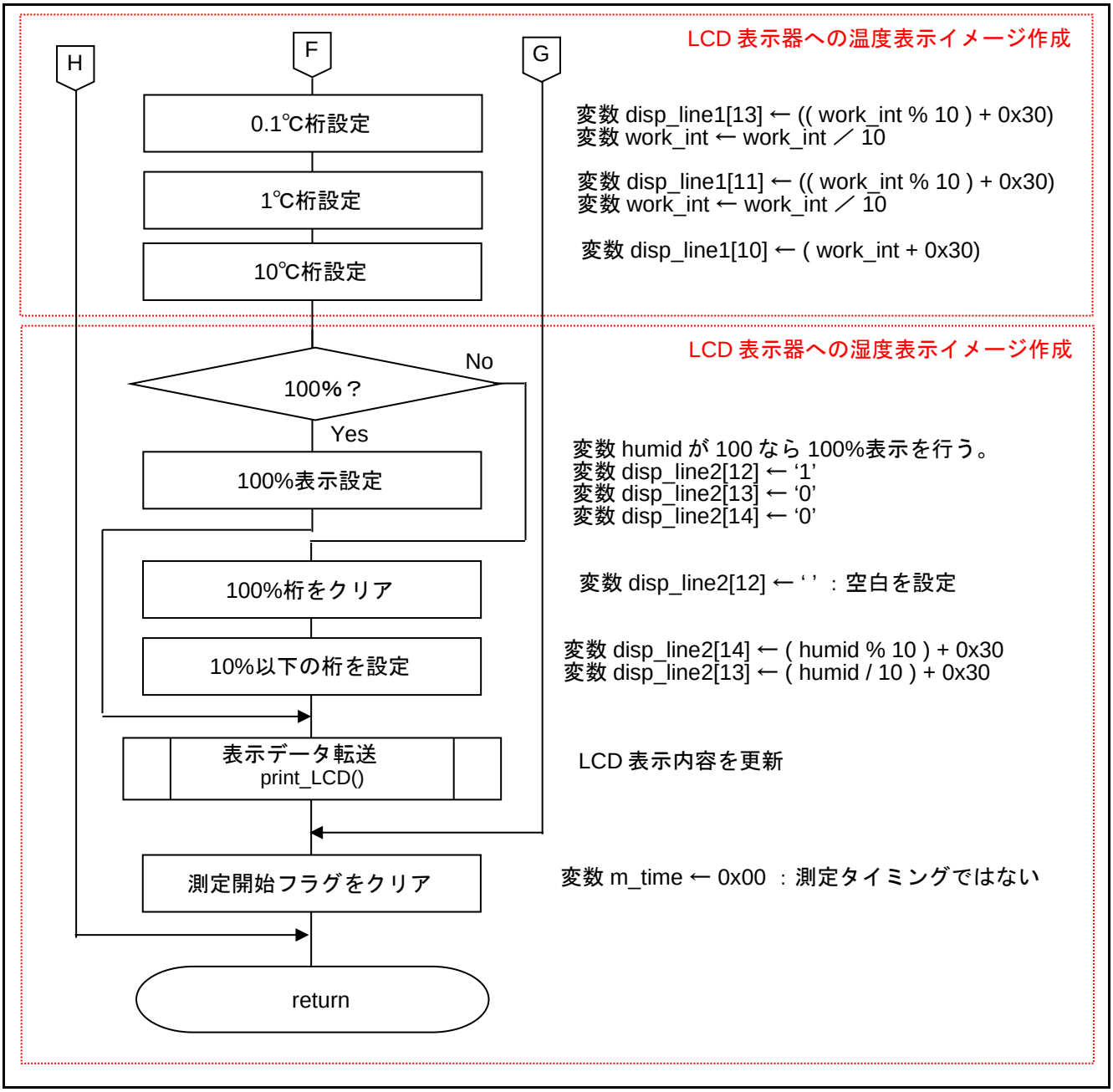
LCD 表示器への温度表示イメージ作成
H
F
G
0.1℃桁設定
1℃桁設定
10℃桁設定
変数 disp_line1[13] ← ((work_int % 10) + 0x30)
変数 work_int ← work_int ／ 10
変数 disp_line1[11] ← ((work_int % 10) + 0x30)
変数 work_int ← work_int ／ 10
変数 disp_line1[10] ← (work_int + 0x30)
LCD 表示器への湿度表示イメージ作成
No
100%？
Yes
100%表示設定
変数 humid が 100 なら 100%表示を行う。
変数 disp_line2[12] ← ‘1’
変数 disp_line2[13] ← ‘0’
変数 disp_line2[14] ← ‘0’
100%桁をクリア
変数 disp_line2[12] ← ‘ ’ ：空白を設定
10%以下の桁を設定
変数 disp_line2[14] ← (humid % 10) + 0x30
変数 disp_line2[13] ← (humid / 10) + 0x30
表示データ転送
print_LCD()
LCD 表示内容を更新
測定開始フラグをクリア
変数 m_time ← 0x00 ：測定タイミングではない
return

図 5.5 メイン関数 (4/4)

5.6.3 LCD 表示器の初期化関数

図 5.6 に LCD 表示器の初期化関数のフローチャートを示します。

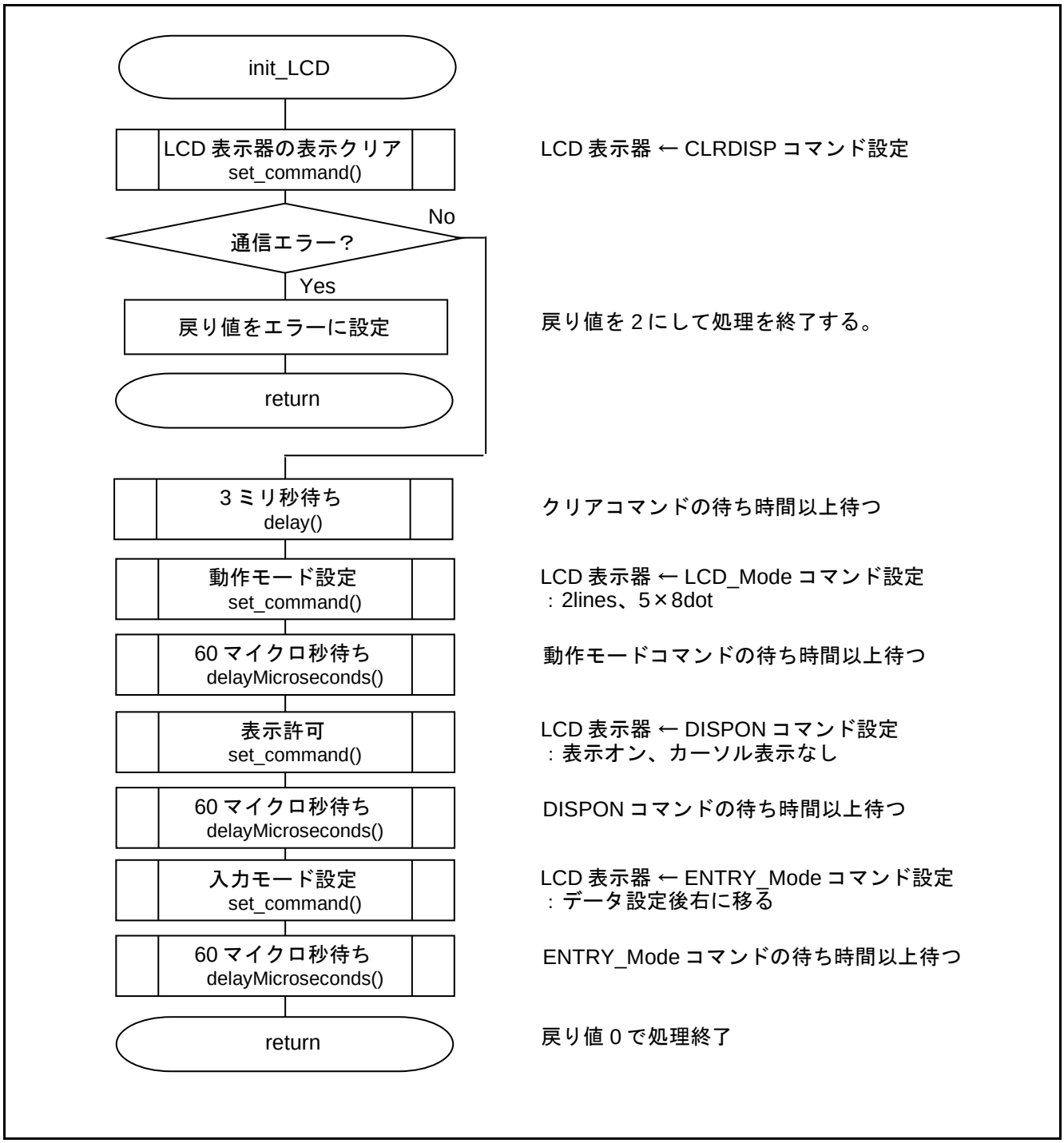

図 5.6 LCD 表示器の初期化関数

5.6.4 LCD 表示器の全画面表示設定関数

図 5.7 と図 5.8 に LCD 表示器の全画面表示設定関数のフローチャートを示します。

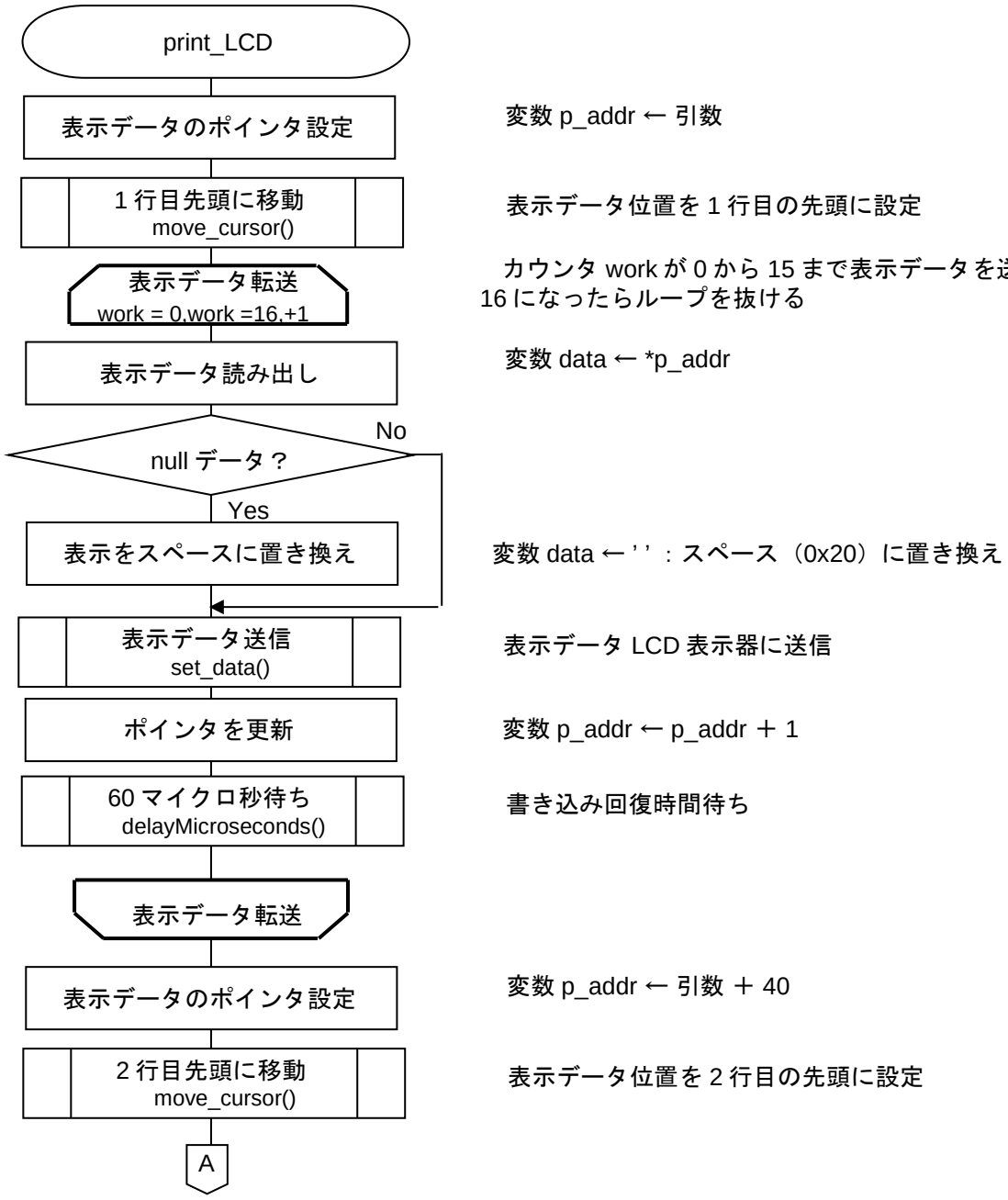

図 5.7 LCD 表示器の全画面表示設定関数 (1/2)

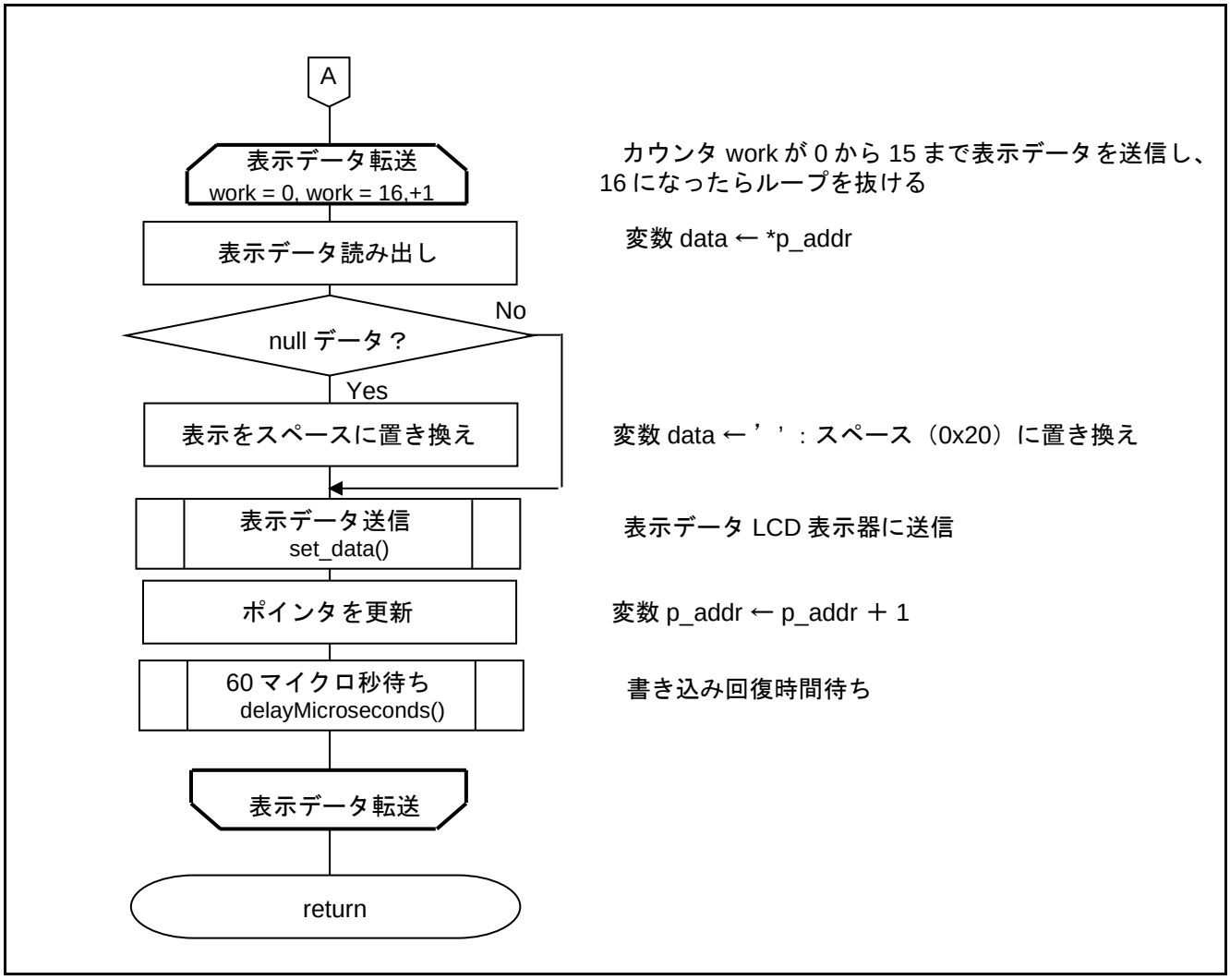

図 5.8 LCD 表示器の全画面表示設定関数 (2/2)

5.6.5 LCD 表示器の表示データ位置設定関数

図 5.9 に LCD 表示器の表示データ位置設定関数のフローチャートを示します。

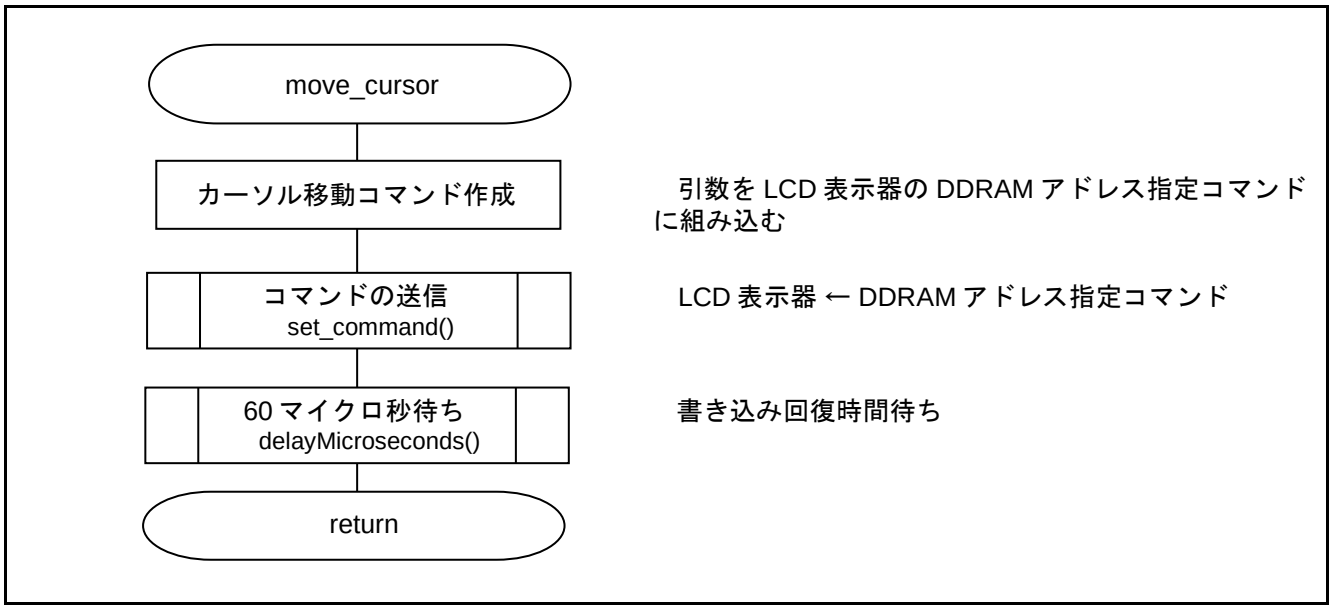

図 5.9 LCD 表示器の表示データ位置設定関数

5.6.6 LCD 表示器のコマンド設定関数

図 5.10 に LCD 表示器のコマンド設定関数のフローチャートを示します。

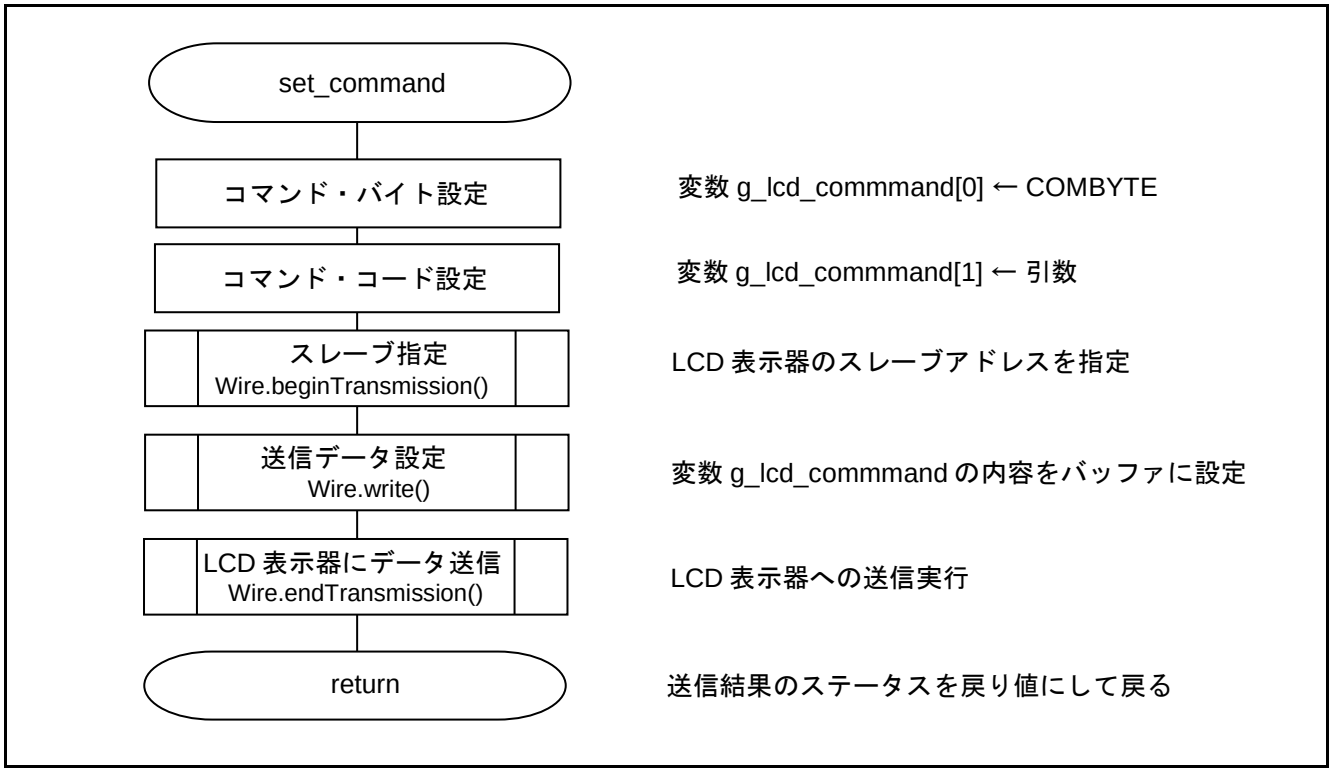

図 5.10 LCD 表示器のコマンド設定関数

5.6.7 LCD 表示器へのデータ設定関数

図 5.11 に LCD 表示器へのデータ設定関数のフローチャートを示します。

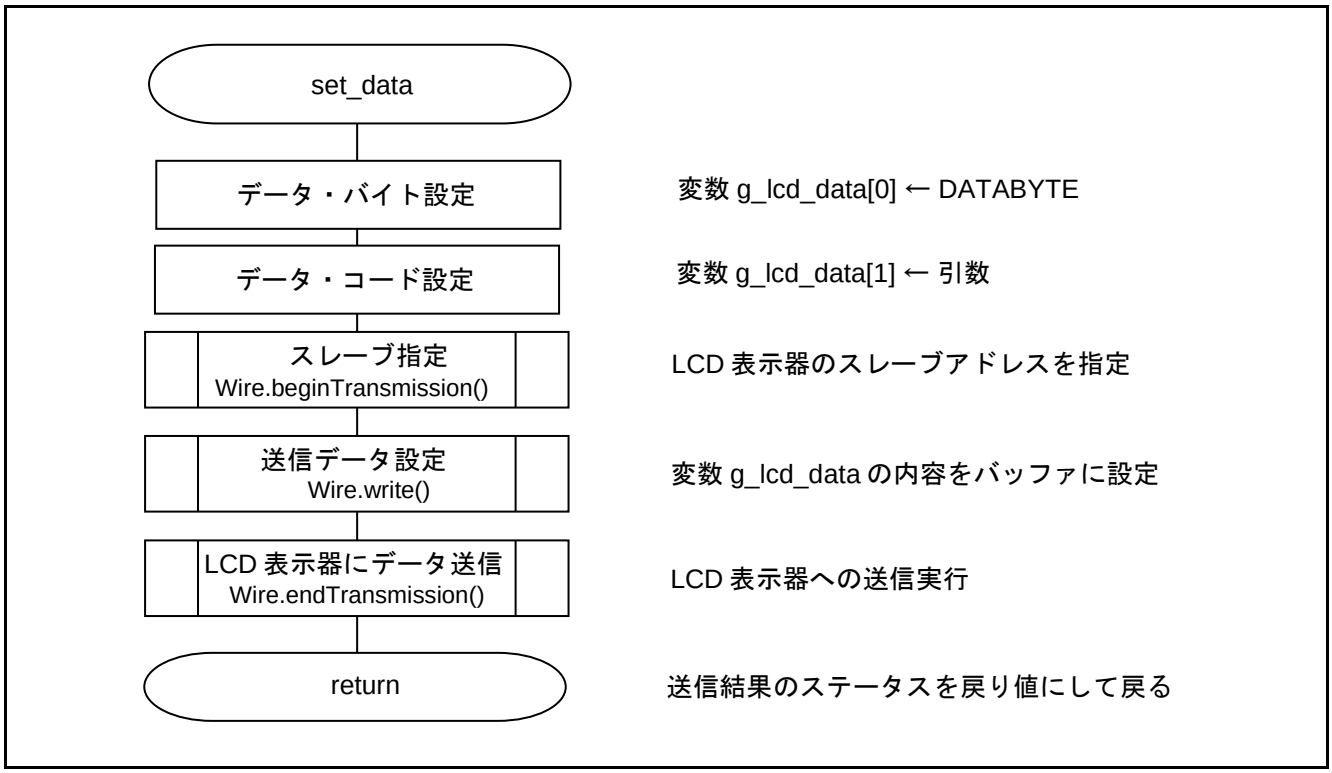

図 5.11 LCD 表示器へのデータ設定関数

6. サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

7. 参考ドキュメント

RL78/G14 ユーザーズマニュアル ハードウェア編 (R01UH0186)
RL78 ファミリ ユーザーズマニュアル ソフトウェア編 (R01US0015)
RL78/G14 Fast Prototyping Board ユーザーズマニュアル (R20UT4573)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

テクニカルアップデート／テクニカルニュース
(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2021.09.01	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1 2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。