RL78/G14

アナログ入出力 (Arduino API)

要旨

本アプリケーションノートでは、RL78/G14 Fast Prototyping Board (FPB) を用いて Arduino 言語のようなプログラム記述で、照度センサからのアナログ入力に応じて LED ライトの明るさを PWM 出力で制御する方法を説明します。

動作確認デバイス

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 を用いて Arduino 言語のようなプログラム記述で照度センサの値をアナログ入力で読み出し、LED ライトを制御するアナログ信号 (PWM 信号) を出力します。

スイッチ押下時もしくは 10 秒ごとに、抵抗で電圧に変換された照度センサの出力値を A/D コンバータで読み出します。その結果に応じて PWM のデューティ比を制御し、LED ライトの明るさを調整します。

照度センサ (NJL7302L-F3) の出力電流は、照度 0.1 lux～1000 lux で 0.2μA～2mA とダイナミックレンジが広いため、51kΩと 1kΩの負荷抵抗を切り替える必要があります。本サンプルコードでは、照度 10 lux～1000 lux に対応し、1kΩの負荷抵抗を使用します。

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
デジタル出力	照度センサの感度選択
A/D コンバータ	照度センサの出力を測定する。
タイマ・アレイ・ユニット	ソフトウェアでの PWM 信号生成 (D5 端子)
タイマ RD	ハードウェアでの PWM 信号生成 (D6 端子)
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 1.1 に示します。

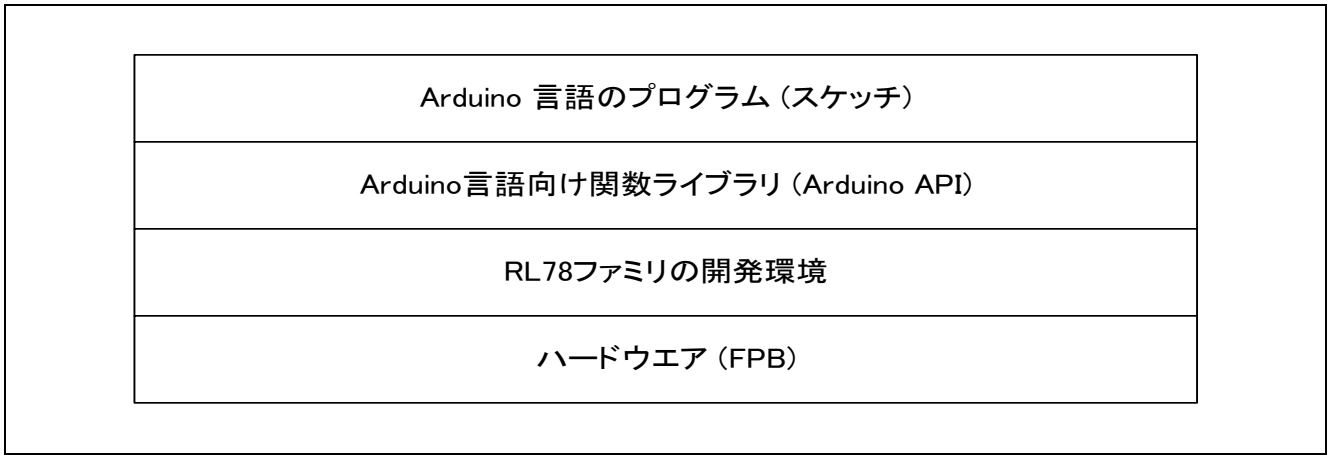

図 1.1 プログラム実行環境

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。
アナログ入出力	analogRead(pin)	アナログ入力ピンからの信号を A/D 変換して取り込みます。
	analogWrite(pin, value)	アナログ出力ピンから 0～255 で指定したデューティ比の PWM 信号を出力します。
	analogReference(type)	アナログ入力の基準電圧を指定します。

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。

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)

項目	ライブラリ関数	機能
簡易 IIC 制御 (Wire1 制御)	Wire1.begin()	Pmod コネクタ 1 に接続された IIC00 (Wire1)を初期化して、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。
簡易 IIC 制御 (Wire2 制御)	Wire2.begin()	Pmod コネクタ 2 に接続された IIC20 (Wire2)を初期化して、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 ファミリの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 1.2 に示します。

1)

```
int swPin       = 18;           // assign D18 pin to swPin for SW_USER.↓
↓
int sensorin    = ANALOGPin0;   // assign A0 pin for sensor data↓
int sensorrefh  = ANALOGPin2;   // assign A2 pin for reference high↓
int sensorrefl  = ANALOGPin3;   // assign A3 pin for reference low↓
↓
int high_range  = 1;            // assign D1 pit to select high illumination↓
int low_range   = 0;            // assign D0 pit to select low illumination↓
↓
int pwm_s       = SOFTPWM2;     // assign SOFTPWM(D5) pin to PWM signal↓
int pwm_h       = HARDPWM1;     // assign HARDPWM(D6) pin to PWM signal↓
```

2)

```
#define SECOND10    ( 10000/16 )       // 10 seconds devided by 16milli sec↓
↓
unsigned int old_time = 0x0000;        // previous time(milli sec.)↓
int count16ms = 0x0000;                // for count 10 seconds↓
char sw_work = 0xFF;                   // work for switch check↓
↓
unsigned int ad_value[2][4];           // 4 times A/D conversion value↓
char ad_time;                          // A/D conversion time count↓
```

図 1.2 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、照度センサのアナログ出力の測定に A0 端子 (ANALOGPin0) を指定し、選択回路のリファレンス電圧の測定に A2 端子 (ANALOGPin2) と A3 端子 (ANALOGPin3) を指定します。照度センサの測定レンジを指定する制御端子に 0 と 1 を指定して D0 と D1 を割り当て、LED の明るさを制御するアナログ出力 (PWM 信号出力) 端子として D5 (ソフトウェア PWM) と D6 (ハードウェア PWM) を割り当てています。

2) 次に経過時間 (ミリ秒単位) を確認するために 16 ビットの変数 old_time を定義し、10 秒をカウントするために 16 ミリ秒のカウンタ count16ms を定義し、スイッチの状態を確認するためにチェック変数 sw_work を定義しています。

また、センサからの測定値を格納する 2 次元の配列 ad_value を定義しています。測定回数は ad_time でカウントします。

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 1.3 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、照度センサを制御する端子を出力に指定します。

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

    pinMode(high_range, OUTPUT);      // set D1 pin to output mode
    pinMode(low_range, OUTPUT);       // set D0 pin to output mode

    analogReference(DEFAULT);         // use VDD as reference for analog in

}
```

図 1.3 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 1.4 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;
    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 1.4 メイン処理の先頭部分

1.7 照度センサの制御

照度センサ (NJL7302L-F3) の出力電流は、照度 0.1 lux～1000 lux で 0.2μA～2mA とダイナミックレンジが広く、10 ビット A/D コンバータでは十分な精度が得られません。そのため、照度 10 lux～1000 lux の範囲は 1kΩの負荷抵抗を使用し、照度 0.1 lux～20 lux の範囲は 51kΩの負荷抵抗を使用する必要があります。

本サンプルコードでは、照度センサからのアナログ入力に応じて LED ライトの明るさを PWM 出力で制御します。PWM のデューディ比は、50 lux 以下のとき 100%、500 lux 以上のとき 0%とし、50 lux～500 lux を 100%～0% (256 分割)で線形制御します。

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

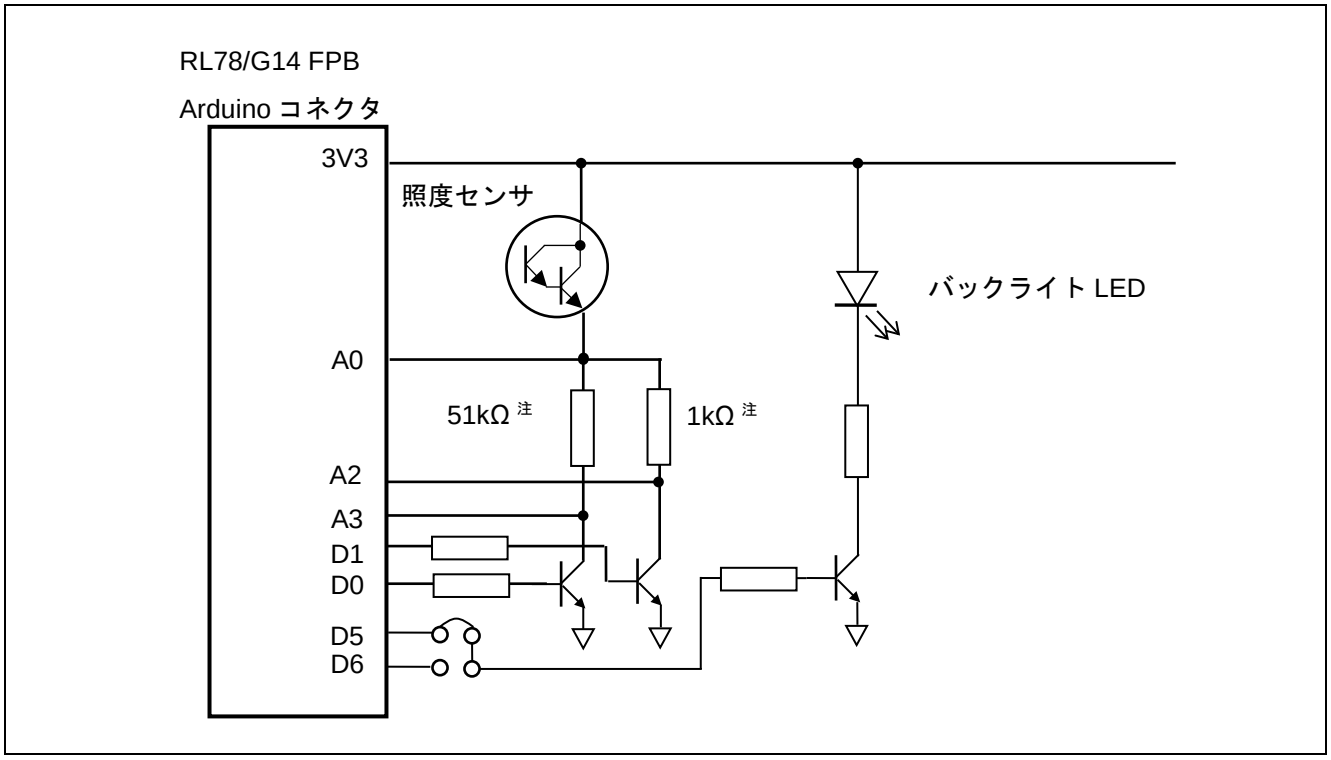

図 4.1 ハードウェア構成例

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。G14 FPB 内部の回路は省略しています。電源電圧には、USB コネクタ経由で 3.3V が供給されています。

注 酸化金属皮膜抵抗

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
D0	P143	出力	低照度選択信号
D1	P144	出力	高照度選択信号
D5	P77	出力	PWM (ソフトウェア) 信号
D6	P10	出力	PWM 信号
A0	P26	アナログ入力	照度信号入力
A2	P24	アナログ入力	高照度用比較電圧入力
A3	P23	アナログ入力	低照度用比較電圧入力
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、照度センサの測定を開始します。測定は 4 回行い、4 回分の平均値から LED をドライブする PWM 信号のデューティ比を求めてアナログ出力端子から PWM 出力します。

以降は、10 秒ごともしくは、スイッチの押下時に測定を行います。

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。
- D1 及び D0 端子の照度センサの負荷抵抗を設定する端子をデジタル出力に設定します。

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。
- 得られたデータが古いデータ (old_time) から変化したかを確認します。
- 変化していなければ、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) していたら、old_time を得られたデータに変更します。
- センサ起動用の 10 秒 (625) 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確認します。
- スイッチが押されておらず、かつ 10 秒経過していなければ、loop 関数の先頭に戻ります。
- スイッチが押された、もしくは 10 秒経過していれば、照度を測定します。
  - ◆ 照度センサの負荷抵抗をオンすることで計測を開始します。
  - ◆ データが安定するまで、約 16 ミリ秒待ちます。
  - ◆ A0 端子のアナログ入力でセンサの出力電圧と、A2 端子のアナログ入力で負荷抵抗のドライブ電圧をそれぞれ 4 回測定して平均値を求めます。
  - ◆ 負荷抵抗を切り離します。
  - ◆ 得られた電位差から電流値を求めます。
  - ◆ 電流値から照度を求めます。
  - ◆ 得られた照度から LED をドライブする PWM 信号のデューティ比を求め、アナログ出力関数に設定します。
- loop 関数の先頭に戻ります。

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
high_range	1	高照度選択端子の番号
low_range	0	低照度選択端子の番号
sensorin	0	照度センサ電圧入力端子番号
sensorrefh	2	照度センサ高照度負荷ドライブ電圧入力端子番号
sensorrefl	3	照度センサ低照度負荷ドライブ電圧入力端子番号
pwm_s	5	端子 5 を使う PWM
pwm_h	6	端子 6 を使う PWM
DEFAULT	0	A/D のリファレンスは VDD
INTERNAL	1	A/D のリファレンスは内部基準電圧
EXTERNAL	2	A/D のリファレンスは外部電源
SECOND10	625	10 秒を計測するための 16 ミリ秒のパルス数

5.3 変数一覧

表 5.2 にサンプルコードで使用する変数を示します。

表 5.2 グローバル変数

型	変数名	内容	使用関数
unsigned int	old_time	前回の起動からの経過時間 (ミリ秒単位)	loop()
Int	count16ms	10 秒用の 16 ミリ秒の回数のカウンタ	loop()
char	sw_work	スイッチの状態チェック用	loop()
unsigned int	ad_value[2][4]	4 回分のアナログ変換結果バッファ	loop()
char	ad_time	アナログ変換回数のカウント用	loop()
unsigned char	g_dachannel [6]	PWM 制御データ (入力用)	analogWrite(), r_tau0_channel0_interrupt()
unsigned char	g_da_time[3]	ソフトウェア PWM 制御データ	r_tau0_channel0_interrupt()

5.4 関数一覧

表 5.3～表 5.4 に関数一覧を示します。

表 5.3 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)
get_PWM_duty	照度センサの電流値から PWM 出力のデューティ比を求めます。
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) を指定します。
digitalWrite	端子へのデータ出力
digitalRead	端子状態の読み出し
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。
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。
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。
analogRead	指定されたアナログ端子の値を読み出します。
analogWrite	指定された端子からアナログ値 (PWM 信号) を出力します。
analogReference	アナログ入力を A/D 変換するための基準電圧を指定します。
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出しを起動します。読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出しを起動します。読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。

表 5.4 関数一覧 (2/2)

関数名	概要
Wire1.begin	Pmod1 コネクタの 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出しを起動します。読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出しを起動します。読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	Pmod2 コネクタの 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出しを起動します。読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出しを起動します。読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	スタートすると起動からの時間を確認し、16 ミリ秒ごとにスイッチをチェックする。押下されているか、もしくは 1 分が経過したらセンサ (HS3001) を起動する。起動後、約 48 ミリ秒経過したら、センサからの計測結果を読み出し、温度と湿度を計算して結果を LCD 表示器で表示する。
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) で使用する端子および A/D コンバータの設定を行う。
引　数	なし
リターン値	なし

[関数名] get_PWM_duty		
概　要	照度センサの電流値から PWM 信号のデューティ比を求める関数	
ヘッダ	r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint8_t get_PWM_duty (uint16_t value);	
説　明	引数で指定された 10 ビットのデータから 255 周期の PWM 信号のデューティ比（0～255）を求める。	
引　数	uint16_t value	:照度センサの電流値
リターン値	uint8_t	:0～255 で示されるデューティ比

[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]	analogRead	
概　要	アナログ入力関数	
ヘッダ	AR_LIB_ANALOG.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	uint16_t analogRead (uint8_t pin);	
説　明	引数で指定された端子の電圧を A/D 変換して戻す。	
引　数	uint8_t	変換する端子
リターン値	uint16_t	A/D 変換結果

[関数名]	analogWrite	
概　要	アナログ出力関数	
ヘッダ	AR_LIB_ANALOG.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	void analogWrite (uint8_t pin, value);	
説　明	引数で指定された端子に指定されたデューティ比の PWM 信号を出力する。	
引　数	uint8_t uint8_t	出力する端子 出力する値
リターン値	なし	A/D 変換結果

[関数名]	analogReference	
概　要	アナログ基準電圧設定関数	
ヘッダ	AR_LIB_ANALOG.h、r_cg_macrodriver.h、r_cg_userdefine.h	
宣　言	void analogReference (uint8_t type);	
説　明	A/D 変換の基準電圧を指定する。	
引　数	uint8_t	基準電圧の設定 DEFAULT：VDD が基準電圧 INTERNAL：内部基準電圧 EXTERNAL：ADREF 端子の電圧
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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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:リスタートコンディション生成 (バスを保持) 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
備　考	g_status：通信ステータス 0x50 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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	終了時の処理 0:リスタートコンディション生成 (バスを保持) 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 の場合は起動成功。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 は起動失敗。 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	受信バッファからのデータ読み出し関数	
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 の場合は起動成功。 0x04 の場合は 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 の場合は送信バッファがあふれたことを示す。0x04 の場合は 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 の場合は送信バッファがあふれたことを示す。0x04 の場合は 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 の場合は送信バッファがあふれたことを示す。0x04 の場合は 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：送信完了時にリスタートでバス確保 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

[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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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:リスタートコンディション生成 (バスを保持) 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 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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:リスタートコンディション生成 (バスを保持) 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 の場合は起動成功。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 は起動失敗。 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 の場合は起動成功。 0x04 の場合は 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 の場合は成功。 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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 の場合は成功。 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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 の場合は正常。 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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：送信完了時にリスタートでバス確保 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

[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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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:リスタートコンディション生成 (バスを保持) 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 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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 の場合は起動成功。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 は起動失敗。 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し、スレーブアドレスを送信する。以降の処理は割り込みを使用した処理となる。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:リスタートコンディション生成 (バスを保持) 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 の場合は起動成功。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 は起動失敗。 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 の場合は起動成功。 0x04 の場合は 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 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_2：通信ステータス 0x00 の場合は成功。 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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。 （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_2：通信ステータス 0x00 の場合は成功 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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。 （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_2：通信ステータス 0x00 の場合は正常。 0x01 の場合は送信バッファのあふれ。 0x04 の場合は 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：送信完了時にリスタートでバス確保 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

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。

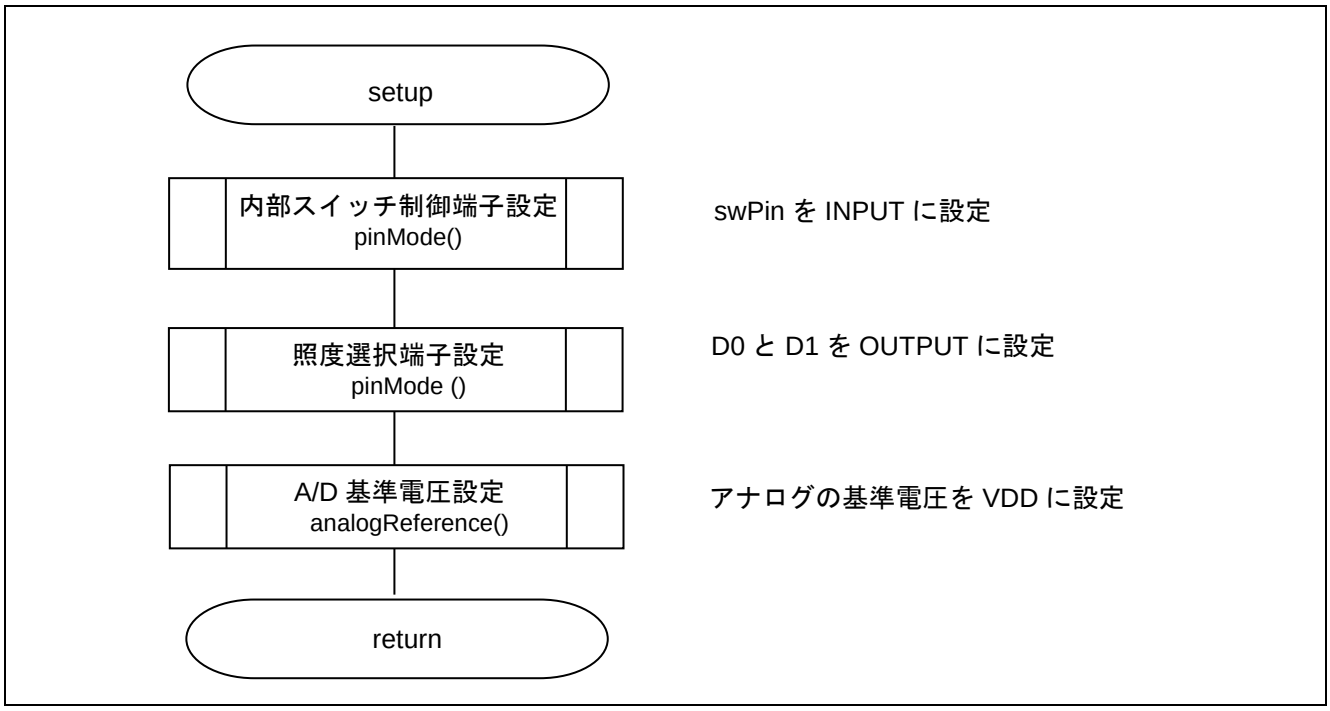

図 5.1 初期設定関数

5.6.2 メイン処理関数

図 5.2～図 5.3 にメイン処理関数のフローチャートを示します。

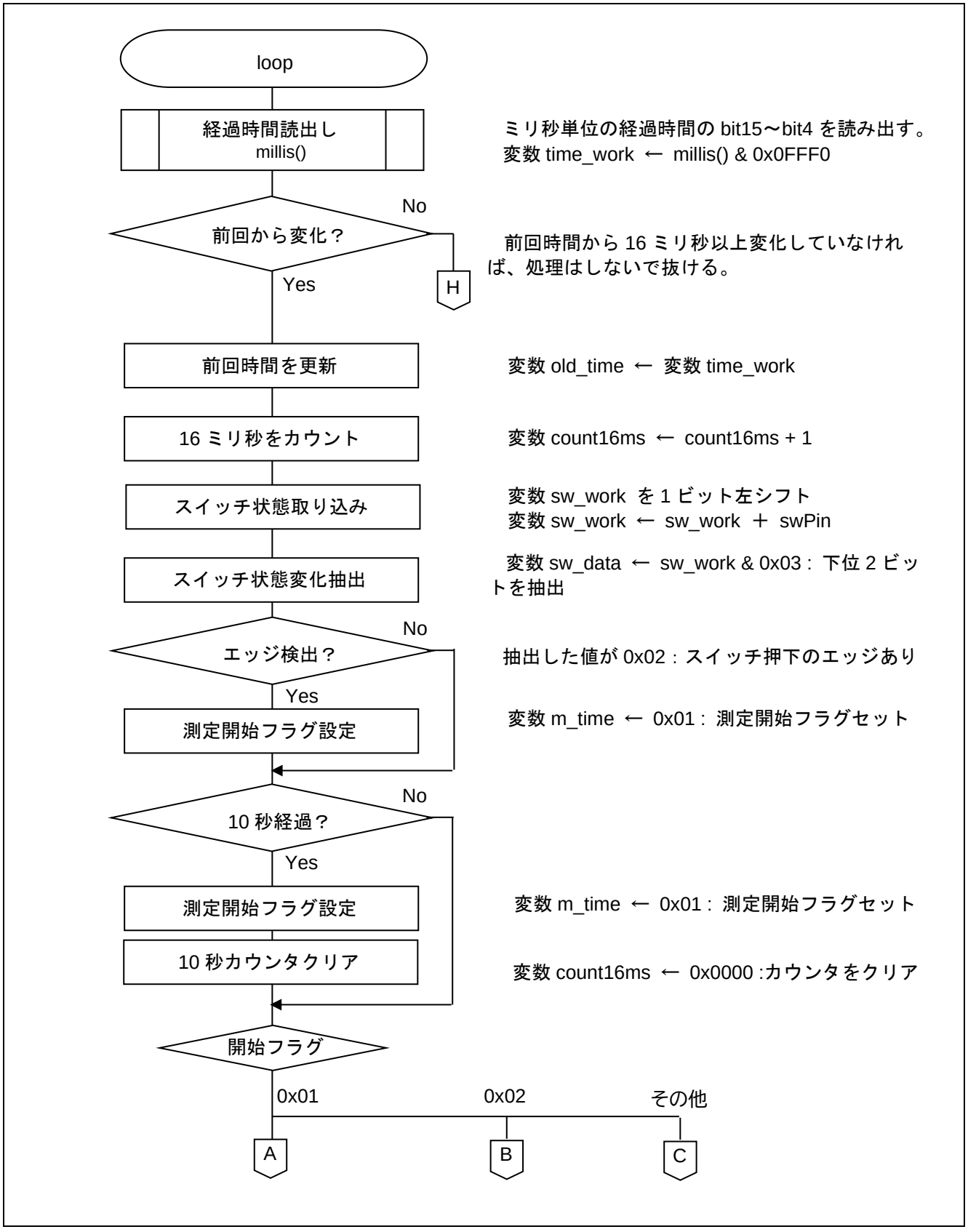

図 5.2 メイン関数 (1/2)

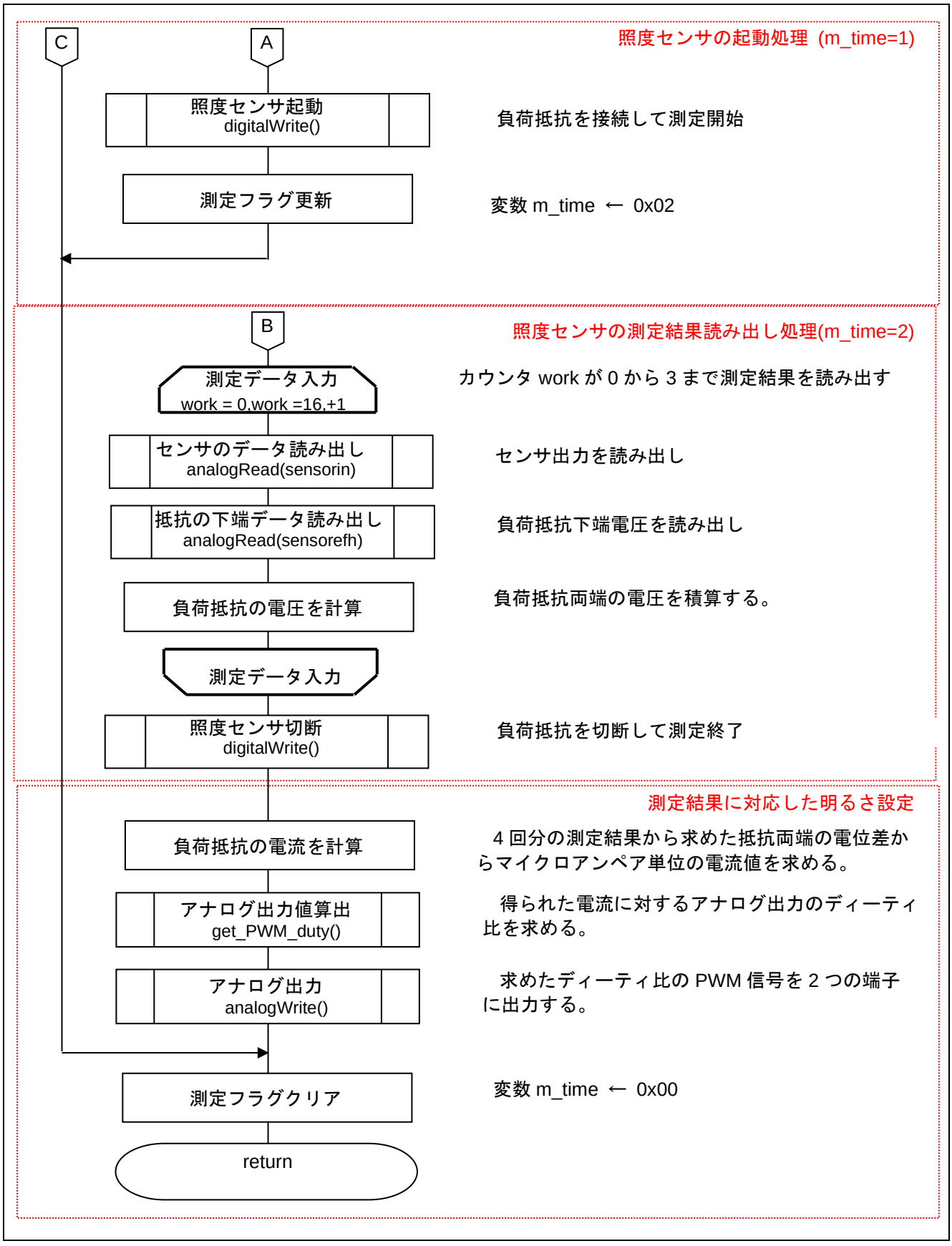
照度センサの起動処理 (m_time=1)
C
A
照度センサ起動
digitalWrite()
負荷抵抗を接続して測定開始
測定フラグ更新
変数 m_time ← 0x02
照度センサの測定結果読み出し処理(m_time=2)
B
測定データ入力
work = 0,work =16,+1
カウンタ work が 0 から 3 まで測定結果を読み出す
センサのデータ読み出し
analogRead(sensorin)
センサ出力を読み出し
抵抗の下端データ読み出し
analogRead(sensorefh)
負荷抵抗下端電圧を読み出し
負荷抵抗の電圧を計算
負荷抵抗両端の電圧を積算する。
測定データ入力
照度センサ切断
digitalWrite()
負荷抵抗を切断して測定終了
測定結果に対応した明るさ設定
負荷抵抗の電流を計算
4 回分の測定結果から求めた抵抗両端の電位差からマイクロアンペア単位の電流値を求める。
アナログ出力値算出
get_PWM_duty()
得られた電流に対するアナログ出力のディーティ比を求める。
アナログ出力
analogWrite()
求めたディーティ比の PWM 信号を 2 つの端子に出力する。
測定フラグクリア
変数 m_time ← 0x00
return

図 5.3 メイン関数 (2/2)

5.6.3 照度センサの電流値から PWM 信号のデューティ比を求める処理関数

図 5.4 に照度センサの電流値から PWM 信号のデューティ比を求める処理関数のフローチャートを示します。

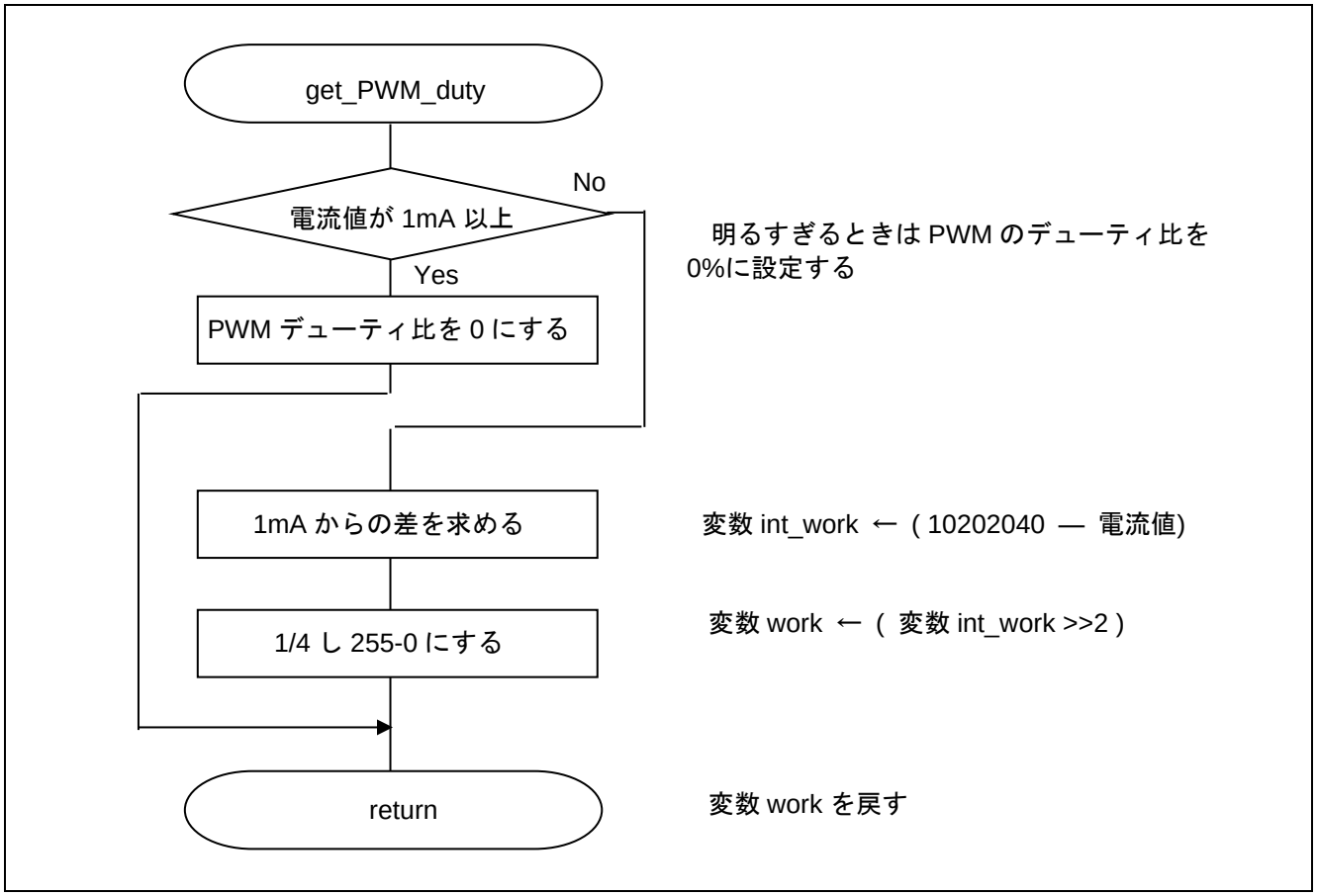

図 5.4 照度センサの電流値から PWM 信号のデューティ比を求める処理関数

備考 計算を簡単にするために上限を 1020μA にしてそこらの差分を求めます。結果を 2 ビット右シフトすることで、0-1020 の値を 0-255 の範囲の値にします。

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.	生成日	改訂内容	
		ページ	ポイント
1.00	2022.03.14	ー	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。