IICA0 ライブラリの解説

IICA0 用に作成したライブラリについて，これまで殆ど説明していなかったので，簡単な説明をしておきます。最後に使用例として BMP180 からの温度データ読み出しを示します。

IICA0 用のライブラリとしては，ハードウェアを制御する基本ライブラリ（r_IIC_lib.c に含まれる）と，基本ライブラリを使用して制御対象に合わせたアプリケーション・ライブラリ（これは，r_cg_main.c に含まれる）があります。基本ライブラリは簡易 I2C と同じインタフェース（及び同じ関数名）にしているので，IICA と簡易 I2C の切り替えが簡単です。

〔1〕IICA0 基本ライブラリ

IICA0 基本ライブラリには以下の関数が含まれています。

・R_IIC_Master_Send()

スレーブ・アドレス，送信データのバッファ，送信データ数を引数として，送信を起動します。

I2C バスが使用可能ならば，スタート・コンディションを発行し，バスの使用権を確保したら，送信用パラメータを作業用のグローバル変数にコピーし，スレーブ・アドレスの LSB をクリア（マスタ送信）にして I2C バスに送信を開始して関数から抜け出します。そのときの戻り値は IIC_SUCCESS（0x00）です。

バスを確保できなければ，戻り値は MD_ERROR1（0x82）となります。

・R_IIC_Master_Receive()

スレーブ・アドレス，受信データのバッファ，受信データ数を引数として，受信を起動します。

I2C バスが使用可能ならば，スタート・コンディションを発行し，バスの使用権を確保したら，受信用パラメータを作業用のグローバル変数にコピーし，スレーブ・アドレスの LSB をセット（マスタ受信）にして I2C バスに送信を開始して関数から抜け出します。そのときの戻り値は IIC_SUCCESS（0x00）です。

バスを確保できなければ，戻り値は MD_ERROR1（0x82）となります。

・R_IIC_check_comstate()

IICA の通信状態（変数 g_iica0_status の値）を戻します。戻り値は以下の通りです（簡易 I2C 用のステータスも含んでいます）。下位 4 ビットが 0b0001 なら通信中，0b0000 なら正常終了，0b0100 なら，スレーブは通信できない状態，0b0101 ならスレーブなし／ビジー（EEPROM では，データの書き込み中のマスタ送信に対して，ACK を戻さない）。

①0x00 (IIC_SUCCESS)　　：通信は正常に終了

②0x81 (IIC_TX_SVADDR)　　：送信用のスレーブ・アドレス送信中（簡易 I2C 用）
③0x01(IIC_USING)　　：データ通信中（簡易 I2C では送信中）
④0xC1(IIC_RX_SVADDR)　　：受信用にスレーブ・アドレス送信中（簡易 I2C 用）
⑤0x41(IIC_RX_MODE)　　：データ受信中（簡易 I2C 用）
⑥0x84(NO_ACK)　　：送信データに対して NACK 応答があった
⑦0x85(NO_SLAVE)　　：スレーブ・アドレスに対して ACK 応答なし

・R_IIC_wait_comend()
IICA の通信完了を HALT 状態で待ちます。通信エラーで終了したならば，ストップ・コンディションを発行して，60μs 待ってから戻ります。

・R_IIC_StopCondition()
I2C バスにストップ・コンディションを発行します。ストップ・コンディションの検出待ちループで WAITTIME（1000）回以内に検出できなかったら，MD_ERROR1（0x82）を戻します。ストップ・コンディションを検出検出したら，直ちに IIC_SUCCESS（0x00）を戻します。コード生成のように指定回数待ってから確認するのではなく，確認しながらループするので，待ち回数が大きくてもバスが使用可能ならすぐにでも戻ってきます。

・IIC_TM03_init()
待ち時間計測用の TM03 を TI03 のエッジ検出を無効にし（ソフトでのトリガのみ有効）TM03 を起動（トリガ待ちに）します。割り込みは禁止にしておきます。

・wait_time()
20μs 単位で指定した時間待ちます。時間は TM03 での 5μs の遅延時間割り込みをポーリングで待ちます。

・wait_20us()
TM03 での 5μs の遅延時間割り込みを 4 回ポーリングで待ちます。

・wait_5us()
TM03 をソフトウェアでトリガして，5μs の遅延時間割り込みをポーリングで待ちます。

〔2〕IICA0 基本ライブラリの内部関数

・R_IICA0_bus_check()

IICA0 のステータス・レジスタ（IICS0）の MSTS0 ビットとフラグ・レジスタ（IICF0）の IICBSY0 ビットを用いて I2C バスの状態をチェックします。コード生成は IICBSY0 ビットだけしかチェックしていなかったので，自身がバスを確保している状態でリスタートができませんでした。

I2C バスが空いている（IICBSY0 ビットが 0）か，自身が既に I2C バスを確保している（MSTS0 ビットが 1）ならば，スタート・コンディションを発行します。スタート・コンディションが検出されて，バスを確保できていれば正常終了とします。なお，自身が I2C バスを確保済の状態でスタート・コンディションが発行できないという状態（スレーブが SDA を Low に引い続けている）には対応していません。そこまでの対応を行うには，SDA 信号が High になるまで，SCL 信号にポート機能を使ってダミー・クロックを出力することで I2C バスを開放させる必要があります。このために，5μs のタイマは準備していますが，I2C バスの開放処理そのものは未実装です。

・r_iica0_interrupt()

INTIICA0 の割り込み処理ルーチンです。INTIICA0 発生時に IICA0 のステータスを確認しながら処理を行っています。

〔3〕アプリケーション・ライブラリ

内部に複数のレジスタをもち，どのレジスタかをレジスタ・アドレスで指定する形式のスレーブを制御するためのライブラリです。通常の I2C バスのスレーブは殆どこの形式です。
基本ライブラリに加えて，レジスタ・アドレスを示す引数が追加されています。

・g_IIC_put_data()

指定したスレーブ・アドレスのデバイスの指定したレジスタ・アドレスからデータを書き込んでいきます。

基本ライブラリと異なり，送信が完了するまで関数からは抜けません。スレーブ・アドレスに対して ACK 応答がなかった場合には，5μs 待ってから 10 回リトライを行います。データに対する NACK 応答に対してはリトライしません。

・g_IIC_get_data()

指定したスレーブ・アドレスのデバイスの指定したレジスタ・アドレスからデータを読み出していきます。

このために，まずレジスタ・アドレス指定のために，1 バイトのデータ（レジスタ・アドレス）をスレーブに対して送信します。送信が完了するのを待ってから，通信ステータスを確認し，スレーブ・アドレスに対して ACK 応答がなければ，5μs 待ってから 10 回までリトライします。スレーブ・アドレスの送信が完了したら，データの受信を起動します。受信

完了を待ち，完了したなら，ストップ・コンディションを発行して I2C バスを開放して処理を終了します。

・g_IIC_get_wdata()

指定したスレーブ・アドレスのデバイスの指定したレジスタ・アドレスから 16 ビット長のデータを読み出していきます。BMP180 から読み出すデータは基本的に 16 ビット長なので，専用の関数として作ってみました。データは上位 8 ビット，下位 8 ビットの順で読み出すものとして，2 バイトのデータを読み出して，16 ビットにマージしてバッファに格納しています。

・g_IIC_get_temp

BMP180 から温度データを読みだすための専用関数です。格読み出したデータを格納するポインタだけを引数で指定するようになっています。この関数内で，スレーブ・アドレスは BMP180 のアドレス（0xEE），BMP180 に対するコマンド・レジスタ（CTRL_MEAS：0xF4）とコマンド（C_TEMP：0x2E）及び温度データのレジスタ・アドレス（A_TEMP：0xF6）を追加して I2C バスをアクセスしています。

最初に，g_IIC_put_data 関数を使用して，BMP180 に温度測定を指示しています。正常に指示できたなら，5ms 以上待ってから g_IIC_get_wdata 関数を使用して，温度データを読み出しています。

5ms の時間は常時動作させている低速内蔵発振回路（LOCO）のクロックをインターバル・タイマでカウントして 5ms のタイミングを生成しています（wait_5ms 関数）。このタイマは常時動作しているので，引数には 5ms のカウント回数＋1 を設定する必要があります。

〔4〕アプリケーションでの使い方

main 関数に記述するアプリケーションとしては，g_IIC_get_wdata 関数を使って，BMP180 の EEPROM から温度補正用のデータをグローバル変数 g_AC6，g_MC，g_MD に格納し，その後に g_IIC_get_temp 関数を使って，温度データを読み出してグローバル変数 g_UT に格納しています。

EEPROM の補正データは一度読み出せば十分なので，後は必要なタイミングで温度データを読み出してください。