RL78/ G12

IICA による IIC バスのマスタ制御

要旨

本サンプルコードでは、シリアル・インタフェース IICA を用いて、IIC バスを介した LED 表示／A/D 変換結果の読み出し／データの送受信の制御方法を示します。

対象デバイス

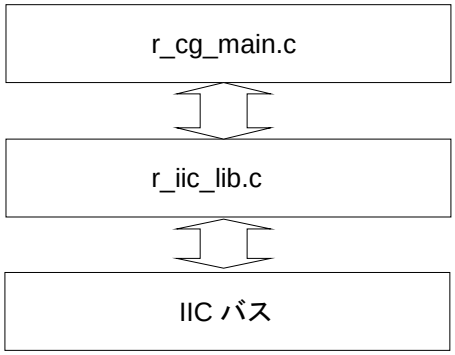
RL78/G12

本サンプルコード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。

1. 本サンプルコードの概要

本サンプルコードでは、IIC バスを使用する上で知っていたほうがよいと思われる基本的な項目および、RL78/G シリーズに内蔵されているシリアル・インタフェース IICA0 を用いた IIC バスで接続した周辺機能の制御方法を示します。

実際に IIC バスの物理層を制御する処理関数部は”r_iic_lib.c”として独立させています。メインの処理を行う r_cg_main.c の IIC 通信を行う関数からは、”r_iic_lib.c”で提供されている関数を使用して IIC バス使用します。”r_iic_lib.c”部分とコード生成での内蔵周辺機能の初期設定を変更することで、簡易 IIC への置き換えができることを考慮しています。



IIC バスを使用するアプリケーションと IIC 通信関数

IICA0 を使用して IIC バスを制御する関数群

IIC バスの物理層

なお、IICA0 の制御には割り込み（ベクタ割り込み）を使用します。

本サンプルコードで対象としているスレーブは、LED 表示／A/D 変換機能／128 バイトの RAM をサポートしているものとします。

2. IIC バスの基本的な内容

2.1 はじめに

IIC バスはいろいろな制約があって，使い方が難しそうだという声をよく耳にします。これは，通信に対して，送りたいときに送信したら相手は勝手に受信してくれ，データが欲しいときには通信相手が都合よく送ってきてくれると考えているからではないかと思われます。これは通信では「無手順」と呼ばれていて，古くはマイコンとテレタイプやＣＲＴターミナルのような端末との通信で一般に使われていました。

2.1.1 通信とプロトコル

しかし、マイコンと周辺機能を内蔵した外部の IC または他のマイコンを接続して通信を行う場合には、これではきちんとした通信とは言えません。

きちんとした通信は，通信相手の状態を考慮しながら，決まったルール（プロトコル）に基づいてデータのやり取りを行うものです。そのため、前もってルールを明確にしておき、そのルールに従って通信を行う必要があります。

その点で，IIC バスでは通信の基本的なプロトコルが規定されているので，それに従うだけで済むと考えるべきです。

他の通信方法，例えば３線式シリアル通信（CSI）でもプロトコルは必要です。CSI では，マスタが通信を始める前にスレーブの通信準備ができていないと，正常な通信は行えません。これを満足させるには，スレーブからビジー状態であることを示す信号をマスタに戻す必要があります。（このためには，３線式ではなく，４線式になってしまいます。）このようなことをきちんと対応しないと，ビットずれが発生してしまい，この状態が CSI のインタフェースを初期化するまで続いてしまいます。この制御は厳密に考えると結構面倒な処理が必要になります。ここらは，RL78/G13 の CSI 通信のアプリケーションノートに例が記載されています。

実際に，マイコンが CSI（または SPI）を用いて制御するような周辺はかなりのものが EEPROM や D/A と言った単機能のハードウェアで，スレーブは常に通信が可能な状態になっているので，マイコン側では通信の手順を特に意識しなくても済んでいるだけなのです。

なお，EEPROM では単純なデータのやり取りだけでは制御できませんが，それは通信ではなく EEPROM の制御方法と考えられているようです。

一方，IIC バスはスレーブが準備できているかをチェックする手段が組み込まれていて，他に信号を追加することなく，２本の信号線だけで通信が可能です。

もうひとつの一般的な通信方式である調歩同期通信ですが，UART 機能を用いて比較的簡単に通信を行えるので，初心者に好まれているようです。上記の端末との通信方式はほとんどがこの調歩同期通信による無手順通信です。しかし，調歩同期通信でもきちんとした通信を行うには，それなりのプロトコルが必要になります（実際に，ほとんどの端末はフロー制御をサポートしています）。UART のレジスタを見ると，３つのエラー・フラグがあります。つまり，これは使用者がそのようなエラーに対する処理を行う必要があることを表しています。しかも，このエラー・フラグは当然ながら受信側にしかありません。そのために，受信側でエラーが発生したことをデータの送信側に連絡する必要があります。ここで良く用いられるのが再送要求です。データの送信に対して，受信側は正しくデータを受け取ったことを示すために，送信側に ACK 応答を行います。データが正しく受け取れなかったときには NACK 応答を行って，送信側に再送を要求します。これらの処理を使用者がソフトで実現する（場合によっては通信用のプロトコル・スタックを利用する）必要があります。

IIC バスには受信側から送信側に受信可能であることを通知する ACK 応答機能が組み込まれているので，この点では使用者が頭を悩ます必要はありません。

IIC バスでは，CAN や LIN のように上位階層までプロトコルが決められていない分，ある程度自由に使うことが可能な通信手段です。だからこそ，ボード内でのいろんな周辺機能との通信に用いられています。

2.1.2　IIC バスの仕様のとらえ方

IIC バスを難しくしている要因のひとつにマルチマスタ機能があります。マルチマスタ機能は IIC バスを完璧に使いこなすには重要な機能です。しかし，どこでアービトレーションが発生し，どこでアービトレーションに負けたかを判断するのは大変です。RL78 のシリアル・インタフェース IICA でもかなりのページが割かれています。しかも、負けたことはわかっても、勝ったことは最後までわかりません。

しかし，ほとんどのアプリケーションはシングルマスタで済み，マルチマスタまでは使いません。マルチマスタかシングルマスタかはシステムの最初の仕様検討段階で決まると考えられます。そこで，シングルマスタで十分な場合には，マルチマスタ機能は無視して使いたい範囲だけで使えば十分です。マルチマスタ機能を使わなければ，アービトレーションや通信予約のような項目は考慮する必要はありません。

同様のことは拡張コードについても言えます。通常の使用方法では拡張コードもほとんど使いません。マスタは単に拡張コードを送信しなければ済みますし，スレーブは拡張コードを受信したら通信から抜けてしまえば済みます。

このように割り切ってしまえば，IIC バスは手軽な通信手段といえます。特に，スレーブはいくつかのポイントさえ抑えれば簡単です。この場合に注意すべきことは，どの機能はサポートして，どの機能はサポートしていないということを明確にしておくことです。

ここではシングルマスタのシステムを対象にします。

2.1.3　IIC バスのプロトコルのとらえ方

IIC バスの通信は全てマスタが管理します。スレーブが独自にできることは，ウエイトをかけることと，NACK 応答をすることだけです。それ以外はマスタからの指示に従って通信を行う必要があります。ただし，IIC バスでは，基本的なプロトコルだけを規定していて，上位のプロトコルは規定していないので，そこはユー

ザが規定する必要があります。つまり，この部分にスレーブの仕様を反映する，スレーブとしてそのような場合にどうするかを定義することになります。

(a) IIC バスの仕様として決まっていること

仕様書には「通常、アドレス指定されたレシーバは、（省略）、各バイトが終了するたびにアクノリッジを生成しなければなりません。

スレーブ・レシーバがアドレス確認を行なうことができない場合（例えば、リアルタイム機能を実行しているために受信できないような場合）、そのスレーブはデータ・ラインを“H”の状態に保持しなければなりません。　（省略）　スレーブがデータを受信することができない場合、スレーブは次に送られてくる最初のバイトに対してアクノリッジを生成しないことによってそのことを示します。　（省略）　マスタがレシーバとなる場合､スレーブから送信された最後のデータ･バイトに対してアクノリッジをしないことによって、マスタはスレーブ・トランスミッタにデータの終わりを知らせます」と記載されています。

つまり，「マスタ受信時の NACK 応答はエラーではなく通信の終了」で，「スレーブからの NACK 応答は通信を行えない状況にある」ことを表します。決して、スレーブからの NACK 応答は受信データにエラーがあったことを意味してはいません。

(b)　コントローラで決まっていること

RL78 に搭載されている IICA 機能については，ハードウェアマニュアルを参照してください。

この中で誤解しやすいのが，スレーブとして動作中にリスタートがかかって，別のスレーブが選択されたときです。通常のスタート・コンディション後のアドレス受信では，アドレスが一致したときだけ割り込みが発生します。しかし，既にスレーブとして通信を行っていたときに，マスタがリスタートで別のスレーブを選択したときにも，割り込みが発生します。これは，それまで行っていた通信が終わったことをスレーブが認識して，通信から抜けるために使用させるためです。このため，マスタがリスタートを行う可能性がある場合には，スレーブは必ず IICS レジスタの状態を確認しながら処理を進める必要があります。

(c)　ユーザが定義すること

例えば，IIC バスの仕様書の「7 ビット・アドレスのフォーマット」には「以前にアクセスされたメモリー・ロケーションの自動加算または自動減算についての決定は、全てデバイスの設計者が行ないます。」と明記されていますし，これ以外にも定義しておいた方がよいことがあります。

例えば，RL78／G13 の IIC バスのアプリケーションノートの例のように，16 バイトのデータしか扱わないのに 17 バイト目のデータの読み出し指示が来た場合にどうするかです。ひとつの考え方として，17 バイト目以降にはまた 1 バイト目からのデータを送信していく（これは EEPROM がそのようになっています）ことがあり，別の考え方としては，16 バイト目を送信したら，17 バイト目からは何も送信しないことがあります。どちらの場合も，スレーブの仕様として定義するなどして，状態と動作を明確化すれば十分です。逆に途中でマスタが通信を打ち切った（NACK 応答，STOP コンディション発行，START コンディション発行）場合，次のマスタからの読み出し指示に対してどのデータを送信するかもスレーブの自由です。前回のアドレスの次データから送信するか，先頭から送信するかのどちらかになるかと思いますが，それはスレーブの仕様として定義すれば済みます。

このように，IIC バスのプロトコルに従って通信を行えば，IIC バスのスレーブとして十分です。このような IIC バスの手順を満足した以外に，ローカルな通信であれば，さらに IIC バスの仕様の一部をサポートしないことも考えられます。

IIC バスのマスタはそのような仕様のスレーブの制御を実現するためにはどうすればよいか検討すればよいわけです（細かいところはマスタの指示（＝IIC バスのプロトコル）に従うが，大きな動作はスレーブの仕様に合わせるという訳です）。

つまりは，最初にスレーブの仕様を明確化するのが重要です。

2.2　対象とするスレーブの仕様

今回の制御対象のスレーブの仕様を示します。

スレーブとして、以下の 3 つの機能を提供します。3 つの機能はレジスタのアドレスと送信か受信かで切り替わります。

・LED 表示機能　　：表示データは 2 個の 8 ビットのデータで、SW で切り替えて表示します。

・A/D 変換機能　　：4 チャネルのアナログ入力の 16 サンプル分の移動平均を送信します。

・RAM 機能　　：128 バイトのデータの任意のアドレスからの読み書きができます。

2.2.1　通信仕様

IIC バスでの通信仕様を以下に示します。

- 接続する IIC バス　　：ファースト・モード
- スレーブ・アドレス　　：0x60
- 拡張コード対応　　：対応しない（無視して通信から退避）
- アドレス・レジスタ方式　：スレーブ・アドレスに続く 8 ビットで使用する機能を指定します。

レジスタ・アドレス	R/W	機　能
0x00 ～ 0x01	Write	LED表示データ書き込み
0x00 ～ 0x03	Read	A/D変換結果読み出し
0x80 ～ 0xFF	Read/Write	RAMへの読み書き

2.2.2　LED 表示機能

LED 表示器としては、8 ビットのデータを表示するために 8 個の LED を使用しています。表示可能なデータは 2 バイトで、SW 入力により 2 バイトのデータのどちらかを指定できるようにします。SW を押していないときには、レジスタ・アドレス 0x00 のデータ、押しているときにはアドレス 0x01 のデータを表示します。

2.2.3　A/D 変換機能

アナログ入力を変換し、最新の 16 回分の移動平均を得ることが可能です。RL78/G12 での代表的な A/D 変換の仕様は以下の通りです。

- ・アナログ入力　　：チャネル 0～3 の 4 チャネル
- ・変換方式　　：スキャン・モードで連続変換モード
- ・変換分解能　　：10 ビット
- ・変換時間　　：6.33μ秒／チャネル

変換結果はレジスタ･アドレス 0x00～0x03 で読み出すことができます。10 ビットの変換結果を上位 2 ビット、下位 8 ビットの順で読み出せます。チャネル 3 の下位 8 ビットを読み出したあとは、図 2.1 に示すようにチャネル 0 に戻ります。

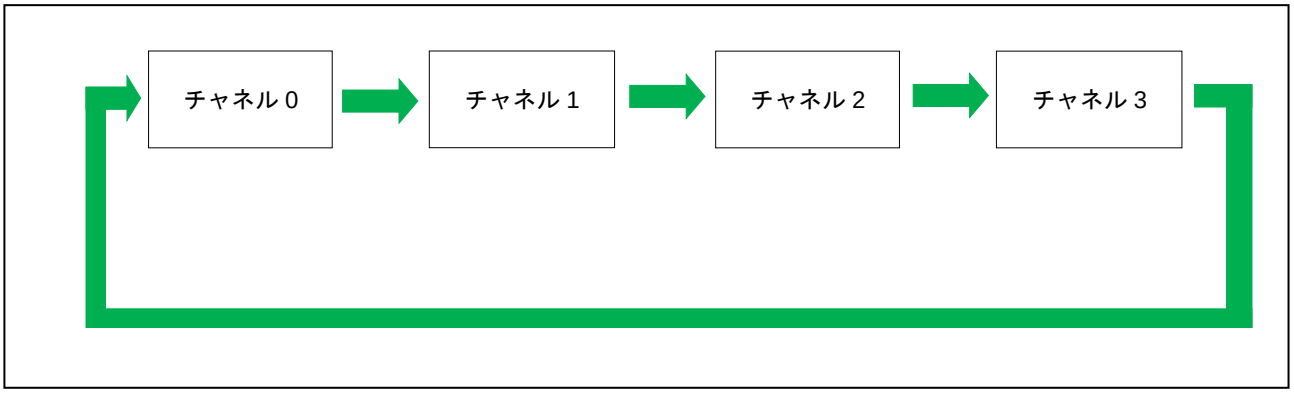


図 2.1 A/D 変換結果の読み出し

2.2.4　RAM 機能

128 バイトの一時保管に使用できる 128 バイトの領域です。初期状態では、0x00～0x7F のデータが格納されています。スレーブ・アドレスに続けて 8 ビットで指定するレジスタのアドレスが 0x80～0xFF の場合に RAM へのアクセスとなります。受信したデータは直ちに RAM に書き込まれます。RAM へのアクセスは自動的にアドレスが更新されていきます。RAM のアドレス 0x7F へのアクセスの次はアドレス 0x00 へのアクセスとなります。

2.3　スレーブをアクセスするためのプロトコル

2.3.1　LED への表示

LED に表示する場合のアクセス方法を図 2.2 に示します。

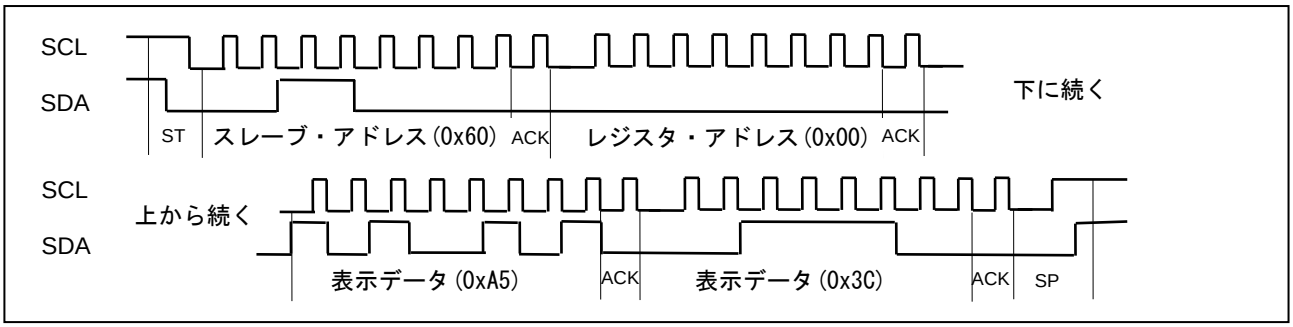


図 2.2 LED 表示データ書き込みタイミング

この例では、スタート・コンディション（ST）、スレーブ・アドレス（0x60）に続けて、レジスタ・アドレス 0x00 に 0xA5、0x01 に 0x3C を書き込んでいます。最後に、ストップ・コンディション（SP）でスレーブに送信完了を通知しています。

これに対して、スレーブは ACK 応答を行っています。

2.3.2　A/D 変換結果の読み出し

A/D 変換結果を読み出す場合のアクセス方法を図 2.3 に示します。

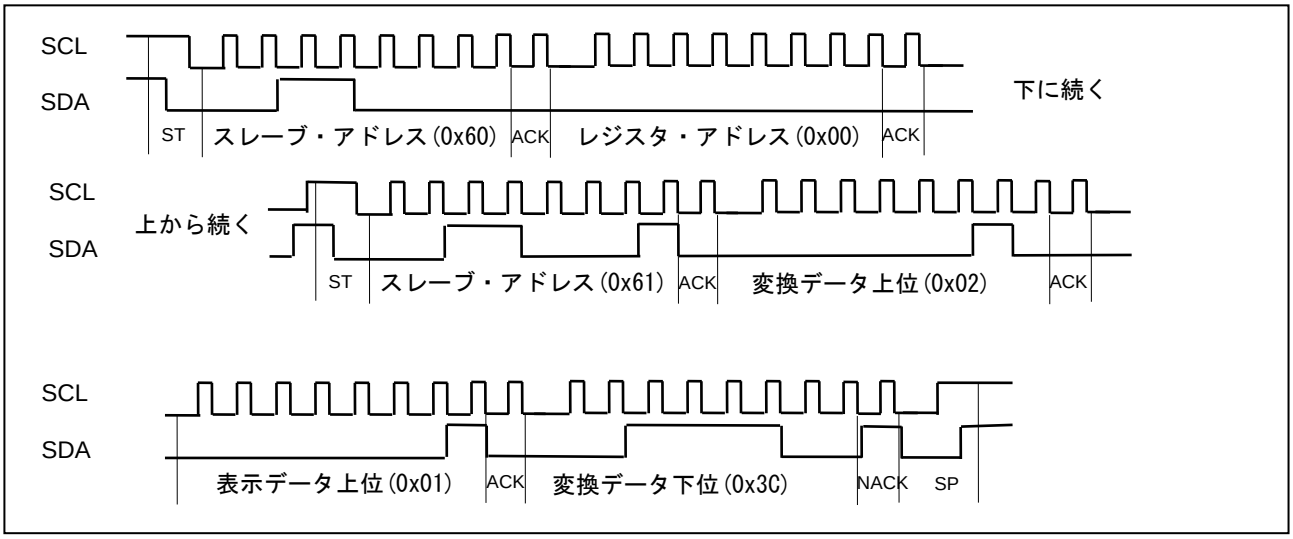


図 2.3 A/D 変換結果読み出しタイミング

この例では、最初にスタート・コンディション（ST）、スレーブ・アドレス（0x60）に続けて、レジスタ・アドレス 0x00（＝チャネル 0）を指定しています。

その後に、リスタートして、読み出しでスレーブを選択（0x61）することで、チャネル 0 の A/D 変換結果を上位、下位の順に読み出します。この図では、チャネル 0 の変換結果の上位が 0x02 となっています

一番下のタイミングでは変換結果（0x013C）を読み出したところで、マスタが NACK 応答を戻してきたので、スレーブは通信完了として通信から退避します。最後に、ストップ・コンディション（SP）で IIC バスを開放して通信を完了します。

4 チャネル分の A/D 変換が完了したら、得られた移動平均値を IICA0 の変換結果送信用バッファに設定します。一方、A/D 変換結果の読み出しはチャネル 3 の下位の読み出し後はチャネル 0 の上位になるので、A/D 変換結果の読み出しを繰り返すことで、最新の変換結果を得ることが可能です。

2.3.3　RAM のデータ読み出し

RAM のデータを読み出す場合のアクセス方法を図 2.4 に示します。

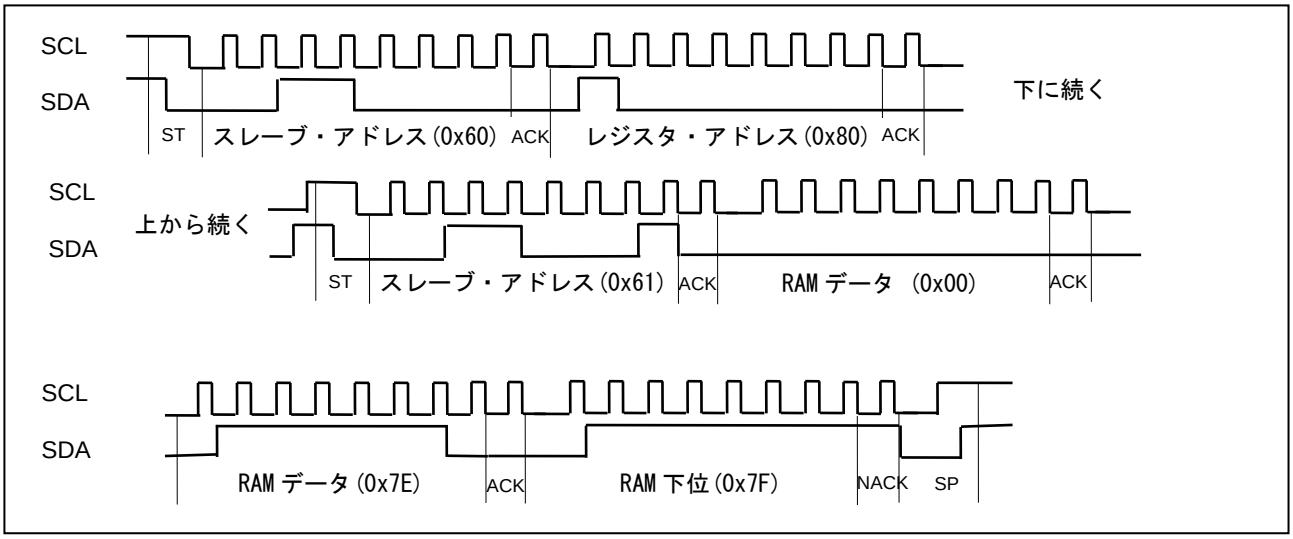


図 2.4 RAM データの読み出しタイミング

この例では、最初にスタート・コンディション（ST）、スレーブ・アドレス（0x60）に続けて、レジスタ・アドレス 0x80（＝RAM のアドレス 0x00）を指定しています。

その後に、リスタートして、読み出しでスレーブを選択（0x61）することで、指定したアドレス 0x00 番地の RAM の値を読み出します。この図では、アドレス 0x00 の値は 0x00 なっています

一番下のタイミングでは RAM アドレス 0x7E の値を読み出し、0x7F の値を読み出したところで、マスタが NACK 応答を戻してきたので、スレーブは通信完了として通信から退避します。最後に、ストップ・コンディション（SP）で IIC バスを開放して通信を完了します。

2.3.4 RAM へのデータ書き込み

RAM にデータを書き込む場合のアクセス方法を図 2.5 に示します。

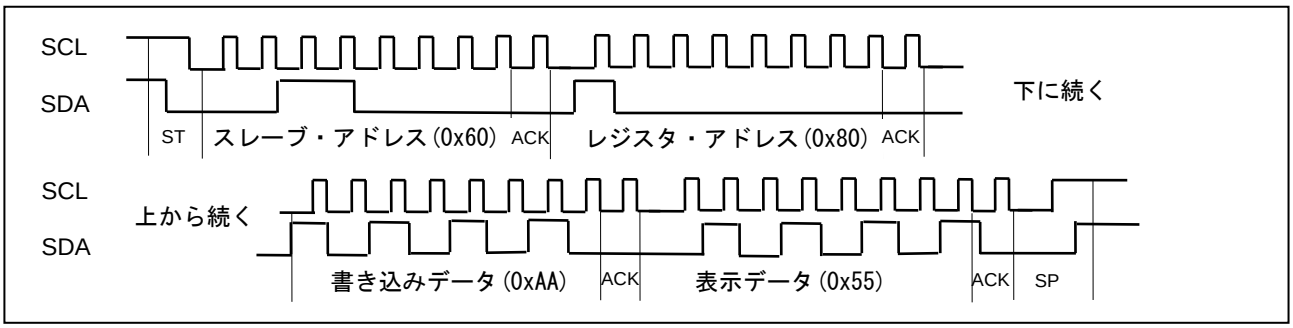


図 2.5 RAM へのデータ書き込みタイミング

この例では、最初にスタート・コンディション（ST）、スレーブ・アドレス（0x60）に続けて、レジスタ・アドレス 0x80（＝RAM のアドレス 0x00）を指定しています。

その後、0x00 番地への書き込みデータとして 0xAA を次の番地への書き込みデータとして 0x55 を送信しています。2 バイトのデータを送信して通信を完了し、ストップ・コンディションを発行して IIC バスを開放しています。

3. 動作確認条件

本サンプルコードは、下記の条件で動作を確認しています。

表 3.1　動作確認条件

項目	内容
使用マイコン	RL78/G12（R5F1026A）
動作周波数	● 高速オンチップ・オシレータ（HOCO）クロック：24MHz ● CPU／周辺ハードウエア・クロック： 24MHz
動作電圧	3.3V（2.9V～5.5V で動作可能） LVD 動作モード：リセット・モード、電圧： 2.75V
統合開発環境	ルネサス エレクトロニクス製 CS+ V3.03.00
アセンブラ	ルネサス エレクトロニクス製 CC-RL V1.02.00
使用ボード	RL78/G12 ターゲット･ボード（QB-R5F1026A-TB） （+ スレーブ・ボード）

4. ハードウェア説明

4.1 ハードウェア構成例

図 4.1 に本サンプルコードで使用するハードウェア構成例を示します。

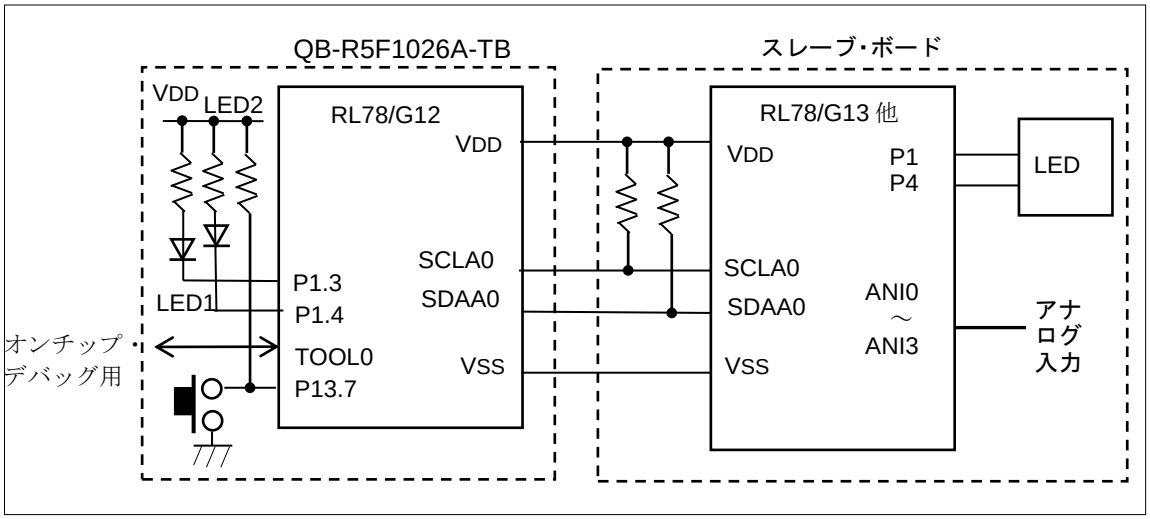


図 4.1　ハードウェア構成

注意 1　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。実際に回路を作成される場合は、端子処理などを適切に行い、電気的特性を満たすように設計してください（入力専用ポートは個別に抵抗を介して V_{DD} 又は V_{SS} に接続して下さい）。

2　V_{DD} は LVD にて設定したリセット解除電圧（V_{LVD}）以上にしてください。

4.2 使用端子一覧

表 4.1に使用端子と機能を示します。

表 4.1　使用端子と機能

端子名	入出力	内容
SDAA0	入出力	IIC 通信データ信号
SCLA0	入出力	IIC 通信クロック信号
P13	出力	動作中を示す LED1 ドライブ信号
P14	出力	RAM チェック・エラーを示す LED2 ドライブ信号
P137	入力	SW 入力

5. ソフトウェア説明

5.1 動作概要

本サンプルコードでは、内蔵周辺機能の初期設定だけは CS+のコード生成機能を利用します。

内蔵周辺機能の初期設定が完了したら、P137 に接続された SW が押されるのを待ち、IIC 通信を開始します。最初に、フラッシュに設定してある LED 点灯用データを 50ms 間隔で送信します。50ms のインターバルがあるので、点灯パターンの動きを目で確認できます。

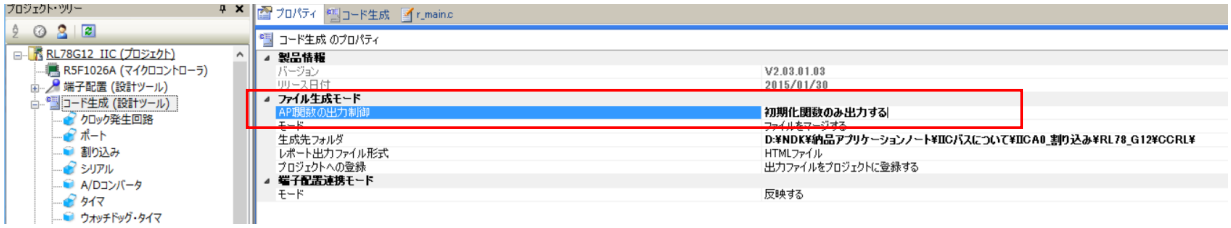
次は、スレーブの RAM からのデータ読み出しです。読み出しは 16 バイトを 8 回合計 128 バイト読み出します。2 回目は以降では読み出したデータの値をチェックし、期待値と異なると LED を点灯してループします。

次は、RAM への書き込みを行います。書き込みは、フラッシュに準備された 64 バイトのデータを 2 個書き込みます。

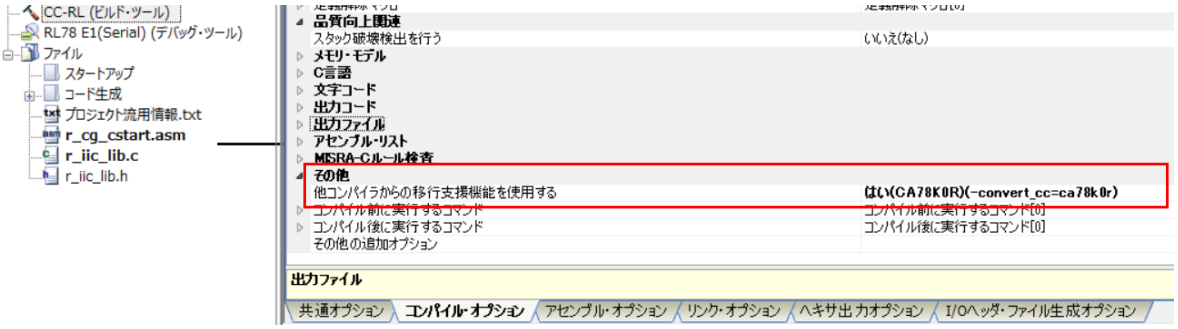
最後に、A/D 変換結果を読み出します。変換結果の読み出しは 4 チャネル分の合計 8 バイトを SW が押されるまで繰り返します。SW が押されたら、LED 点灯処理に戻ります。

5.2 コード生成での設定内容

プロパティの「ファイル生成モード」の「API 関数の出力制御」を「初期化関数のみ出力する」に設定し、以下の設定を行います。



CA78K0R からの継承性で、CC-RL78 のプロパティの「コンパイル・オプション」の「その他」の「他コンパイラからの以降支援機能を使用する」を「はい（CA78K0R）（-convert_cc=ca78k0r）」に設定しておきます。



(1) クロック発生回路の設定

(a) 端子割り当て設定：そのまま確定

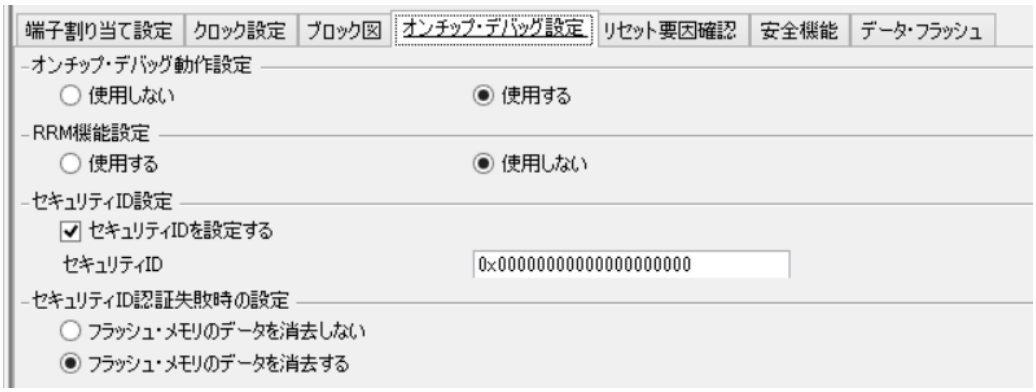
(b) クロック設定

・動作モード設定：高速メイン・モード 2.7(V) ≦ VDD ≦ 5.5(V)
・EVDD 設定：2.7(V) ≦ EVDD ≦ 5.5(V)

・メイン・システム・クロック（fMAIN）設定：高速オンチップオシレータクロック（fIH）
・高速オンチップオシレータクロック設定：24（MHz）
・高速システム・クロック設定：動作をチェックしない
・低速内蔵発振クロック（fIL）設定：周波数　15（kHz）
・インターバル・タイマ動作クロック設定：15（fIL）（kHz）
・CPU と周辺クロック設定：24000（fIH）（kHz）
・RESET 端子設定：使用する（P125）

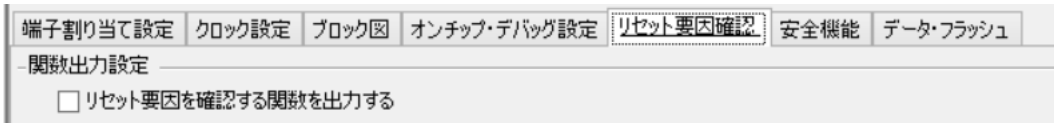
(c) オンチップ・デバッグ設定

・オンチップ・デバッグ動作設定：使用する
・RRM 機能設定：使用しない
・セキュリティ ID 設定：セキュリティ ID を設定する
・セキュリティ ID 認証失敗時の設定：フラッシュ・メモリのデータを消去する



(d) リセット要因確認

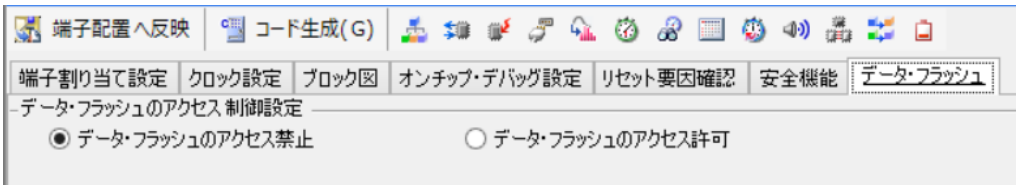
・リセット要因を確認する関数を出力する：「チェック」を外す



(e) 安全機能

全て初期値（「使用しない」チェック）のまま

(f) データ・フラッシュ：データ・フラッシュのアクセス禁止



(2) ポートの設定

P13 と P14 を出力ポートに設定し、「1」にチェックする

その他のポートは初期値（使用しない）のまま

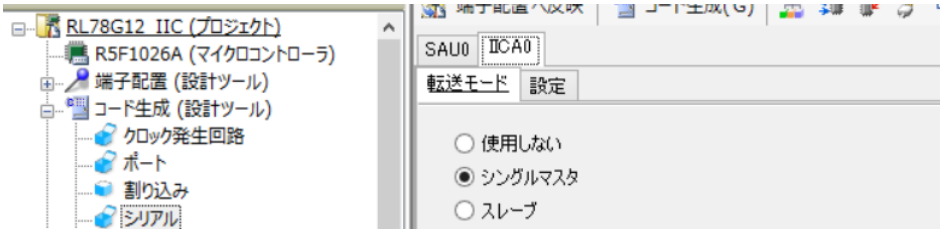
(3) 割り込みの設定

全て初期値（チェックなし）のまま

(4) シリアルの設定

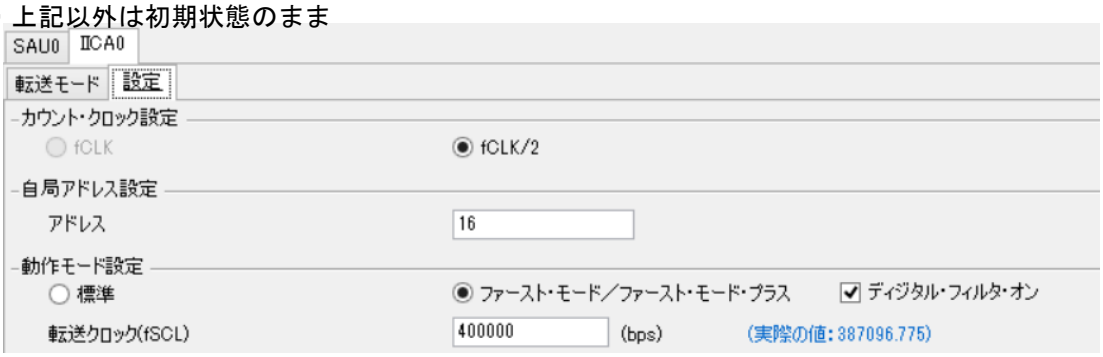
SAU 関係は全て初期値（使用しない）のまま

(a) IICA0 の転送モード：シングルマスタを選択



(b) IICA0 の設定

・カウント・クロック設定：fCLK/2（RL78/G10 の場合は fCLK）
・自局アドレス設定：16
・動作モード設定：ファースト・モードを選択、ディジタル・フィルタ・オンをチェック
・転送クロック（fSCL）：400000（bps）
・上記以外は初期状態のまま



(5) A/D コンバータの設定

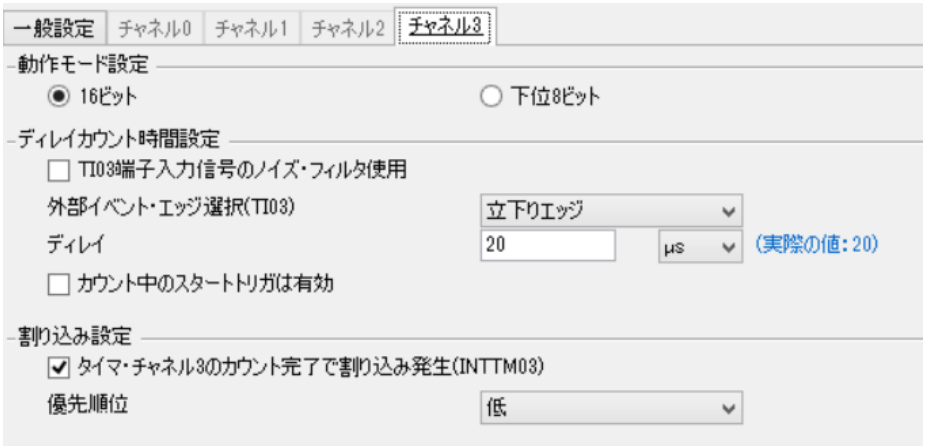
初期値（使用しない）のまま

(6) タイマの設定

(a) 一般設定　チャネル 3：ディレイカウント機能

(b) チャネル 3

・動作モード設定：16 ビット
・ディレイカウント時間設定：20μs
これ以外は初期状態のまま



(7) ウォッチドッグ・タイマの設定

・HALT/STOP/SNOOZE モード時の動作設定：停止
・ウォッチドッグ・タイマ動作設定：使用しない

(8) インターバル・タイマの設定

・インターバル・タイマ動作設定：使用する
・インターバル時間設定：50　ms
上記以外は初期設定のまま

(9) クロック出力／ブザー出力の設定

初期値（使用しない）のまま

(10) DMA コントローラの設定

初期値（使用しない）のまま

(11) 電圧検出回路の設定

・電圧検出動作設定：使用する
・動作モード設定：リセットモード
・検出電圧設定：2.75（V）

5.3　オプション・バイトに反映される設定一覧

表 5. 1 にオプション・バイトに反映される値の例を示します。

表 5.1　オプション・バイト反映値

アドレス	設定値	内容
0x000C0	0b11101110	ウォッチドッグ・タイマ 動作停止 （リセット解除後、カウント停止）
0x000C1	0b01111111	LVD リセット・モード 2.81V (2.76V～2.87V)
0x000C2	0b11100000	HS モード、HOCO ：24MHz
0x000C3	0b10000100	オンチップ・デバッグ許可

5.4　定数一覧

表 5.2 にサンプルコードで使用する定数を示します。

表 5.2　サンプルコードで使用する定数

定数名	設定値	内容
TRUTH	1	真、セット
FALSE	0	偽、クリア
INT_MASK	1	割り込み禁止
INT_ENABLE	0	割り込み許可
MD_ERROR1	0x82	IIC バスがビジー状態
MD_ERROR3	0x84	アービトレーション負け
MD_OK	0x00	正常終了
MAX_DATA	64	一度に扱うデータ数の上限
SLAVE_ADDR	0x60	アクセス対象のスレーブ・アドレス
RAM_TOP	0x80	スレーブの RAM の先頭アドレス
LED_ON	0	LED を点灯するデータ
dispdata[22][2]		LED の点灯を制御するデータ・パターン
ram_data[8][64]		スレーブの RAM に書き込むデータ
IIC_STS_MASK	0x0F	g_iic00_status の下位 4 ビットを抽出するマスク
IIC_USING	0x01	IIC バスを使用中
IIC_SUCCESS	0x00	IIC 通信を正常終了
WAITTIME	1000	スタート・コンディションの発行待ち時間
STS_MASK	0x3F	ステータスのマスクデータ（ライブライ内部用）
COM_ERROR	0xFF	通信エラーのステータス
ON_COMMU	0x01	通信中ステータス
DUMMY_DATA	0xFF	受信起動時のダミー・データ

5.5　変数一覧

表 5.3 にサンプルコードで使用する変数一覧を示します。g_iica0_status 以下の変数は、IICA0 のハードウェア制御用ライブラリで使用する変数です。

表 5.3　サンプルコードで使用する変数

Type	Variable Name	Contents	Function Used
uint8_t	g_read_ram[8][16]	RAM 受信用バッファ	main()
uint8_t	g_read_data[16]	受信データ用バッファ	main()
uint8_t	g_write_data[MAX_DATA+1]	送信用バッファ （送信データの頭にレジスタの値（アドレス）を追加	g_IIC_put_data g_IIC_get_data
uint8_t	g_025s_status	0.25s 経過フラグ	R_MAIN_UserInit() wait_50ms() wait_250ms() r_it_interrupt()
uint8_t	g_025s_count	50ms カウンタ	R_MAIN_UserInit() wait_50ms() r_it_interrupt()
uint8_t	g_iica0_status	IICA0 通信状態	R_IIC_check_comstate() R_IIC_wait_comend() R_IICA0_bus_check() r_iica0_interrupt()
uint8_t	gp_iica0_rx_address	受信データ書き込みポインタ	R_IIC_Master_Receive() r_iica0_interrupt()
uint16_t	g_iica0_rx_len	受信するデータ数	R_IIC_Master_Receive() r_iica0_interrupt()
uint16_t	g_iica0_rx_cnt	受信したデータ数	R_IIC_Master_Receive() r_iica0_interrupt()
uint16_t	gp_iica0_tx_address	送信データ読み出しポインタ	R_IIC_Master_Send() r_iica0_interrupt()
uint16_t	g_iica0_tx_cnt	残り送信データ数	R_IIC_Master_Send() r_iica0_interrupt()

5.6　関数一覧

表 5.4 に IIC バス制御で使用する関数一覧を示します。関数 IIC_TM03_init 以下は IICA0 のハードウェア制御用ライブラリの関数です。

表 5.4　関数一覧

関数名	概要
g_wait_SW	起動時の SW 入力待ち
g_IIC_put_data	IIC ライブラリを使用してデータを送信する
g_IIC_get_data	IIC ライブラリを使用してデータを受信する
wait_50ms	LED 表示データ更新用に 50ms 待つ
wait_250ms	初期化後に 250ms 待つ
r_it_interrupt	100ms のインターバル・タイマ割り込み処理を行う
IIC_TM03_init	TM03 の起動準備を行う
R_IIC_Master_Send	指定したデータのスレーブへの送信を起動する
R_IIC_Master_Receive	スレーブからのデータ受信を起動する
R_IIC_wait_comend	IIC バスでの通信完了を待つ
R_IIC_check_comstate	IIC バスの通信状態を確認する
R_IIC_StopCondition	IIC バスにストップ・コンディションを発行する
wait_time	LCD モジュールのコマンド処理時間を待つ
wait_20us	LCD モジュールのタイミング調整で 20us 待つ
R_IICA0_bus_check	IIC バスにスタート・コンディションを発行する
r_iic00_interrupt	IIC00 の転送完了処理を行う