SmartRX 学習キット
チュートリアル 1

ルネサス エレクトロニクス社　RX マイコン搭載
HSB シリーズマイコンボード　評価キット

-本書を必ずよく読み、ご理解された上でご利用ください

株式会社北斗電子

REV.1.0.1.0

ー目　次ー

注意事項

本書を必ずよく読み、ご理解された上でご利用ください

【ご利用にあたって】

1. 本製品をご利用になる前には必ず取扱説明書をよく読んで下さい。また、本書は必ず保管し、使用上不明な点がある場合は再読し、よく理解して使用して下さい。
2. 本書は株式会社北斗電子製マイコンボードの使用方法について説明するものであり、ユーザシステムは対象ではありません。
3. 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
4. 弊社のマイコンボードの仕様は全て使用しているマイコンの仕様に準じております。マイコンの仕様に関しましては製造元にお問い合わせ下さい。弊社製品のデザイン・機能・仕様は性能や安全性の向上を目的に、予告無しに変更することがあります。また価格を変更する場合や本書の図は実物と異なる場合もありますので、御了承下さい。
5. 本製品のご使用にあたっては、十分に評価の上ご使用下さい。
6. 未実装の部品に関してはサポート対象外です。お客様の責任においてご使用下さい。

【限定保証】

1. 弊社は本製品が頒布されているご利用条件に従って製造されたもので、本書に記載された動作を保証致します。
2. 本製品の保証期間は購入戴いた日から1年間です。

【保証規定】

保証期間内でも次のような場合は保証対象外となり有料修理となります

1. 火災・地震・第三者による行為その他の事故により本製品に不具合が生じた場合
2. お客様の故意・過失・誤用・異常な条件でのご利用で本製品に不具合が生じた場合
3. 本製品及び付属品のご利用方法に起因した損害が発生した場合
4. お客様によって本製品及び付属品へ改造・修理がなされた場合

【免責事項】

弊社は特定の目的･用途に関する保証や特許権侵害に対する保証等、本保証条件以外のものは明示・黙示に拘わらず一切の保証は致し兼ねます。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任を負いません。損害の発生についてあらかじめ知らされていた場合でも保証は致し兼ねます。
ただし、明示的に保証責任または担保責任を負う場合でも、その理由のいかんを問わず、累積的な損害賠償責任は、弊社が受領した対価を上限とします。本製品は「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。
保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。
本製品を使った二次製品の保証は致し兼ねます。

安全上のご注意

製品を安全にお使いいただくための項目を次のように記載しています。絵表示の意味をよく理解した上でお読み下さい。

表記の意味

警告 取扱を誤った場合、人が死亡または重傷を負う危険が切迫して生じる可能性がある事が想定される

注意 取扱を誤った場合、人が軽傷を負う可能性又は、物的損害のみを引き起こすが可能性がある事が想定される

絵記号の意味

	**一般指示** 使用者に対して指示に基づく行為を強制するものを示します		**一般禁止** 一般的な禁止事項を示します
	**電源プラグを抜く** 使用者に対して電源プラグをコンセントから抜くように指示します		**一般注意** 一般的な注意を示しています

警告

以下の警告に反する操作をされた場合、本製品及びユーザシステムの破壊・発煙・発火の危険があります。マイコン内蔵プログラムを破壊する場合もあります。

1. 本製品及びユーザシステムに電源が入ったままケーブルの抜き差しを行わないでください。
2. 本製品及びユーザシステムに電源が入ったままで、ユーザシステム上に実装されたマイコンまたはIC等の抜き差しを行わないでください。
3. 本製品及びユーザシステムは規定の電圧範囲でご利用ください。
4. 本製品及びユーザシステムは、コネクタのピン番号及びユーザシステム上のマイコンとの接続を確認の上正しく扱ってください。

発煙・異音・異臭にお気付きの際はすぐに使用を中止してください。

電源がある場合は電源を切って、コンセントから電源プラグを抜いてください。そのままご使用すると火災や感電の原因になります。

⚠注意

以下のことをされると故障の原因となる場合があります。

1. 静電気が流れ、部品が破壊される恐れがありますので、ボード製品のコネクタ部分や部品面には直接手を触れないでください。
2. 次の様な場所での使用、保管をしないでください。

   ホコリが多い場所、長時間直射日光があたる場所、不安定な場所、衝撃や振動が加わる場所、落下の可能性がある場所、水分や湿気の多い場所、磁気を発するものの近く
3. 落としたり、衝撃を与えたり、重いものを乗せないでください。
4. 製品の上に水などの液体や、クリップなどの金属を置かないでください。
5. 製品の傍で飲食や喫煙をしないでください。

ボード製品では、裏面にハンダ付けの跡があり、尖っている場合があります。

取り付け、取り外しの際は製品の両端を持ってください。裏面のハンダ付け跡で、誤って手など怪我をする場合があります。

CD メディア、フロッピーディスク付属の製品では、故障に備えてバックアップ(複製)をお取りください。

製品をご使用中にデータなどが消失した場合、データなどの保証は一切致しかねます。

アクセスランプがある製品では、アクセスランプの点灯中に電源を切ったり、パソコンをリセットをしないでください。

製品の故障や、データ消失の原因となります。

本製品は、医療、航空宇宙、原子力、輸送などの人命に関わる機器やシステム及び高度な信頼性を必要とする設備や機器などに用いられる事を目的として、設計及び製造されておりません。

医療、航空宇宙、原子力、輸送などの設備や機器、システムなどに本製品を使用され、本製品の故障により、人身や火災事故、社会的な損害などが生じても、弊社では責任を負いかねます。お客様ご自身にて対策を期されるようご注意ください。

本書で説明する内容

本書は、最初にマイコンボードのプログラムを行う方を対象にしています。

マイコンチップ自体やボードに搭載されている機能を動かすためにはどうすればよいかを、順を追って解説するマニュアルとなります。

章毎に、解説している内容が異なりますので、順番にお読み頂く必要はありません。興味のある章、使用したい機能を解説している章、チュートリアルを動かしてみてください。

マイコンと組み込みプログラム

マイコンとは？

本製品は、マイコンを搭載したボードです。マイコンとは一体何者なのでしょう？

SmartRX!!!マイコンボードを見ると、黒くて足が出ている部品がありますが、それがマイコンです。

よく見ると、黒くて足が出ている部品は、実はたくさんあります。そのうち、ボードの中央部に配置されており、4 方向に細かな足が出ている正方形の部品がマイコンです。

マイコンは、一般的にマイコンと呼ばれていますが、正しくは「マイクロコントローラ」の略です。「小さくて、何かを制御するもの」という意味でしょうか。

マイコンに良く似たものとして、CPU(Central Processing Unit)があります。一般的にパソコンに入っている、銀色の金属製のふたが付いていて、裏面には端子が格子状に並んでいる部品です（2018 年現在）。

また、最近のスマートフォンに搭載されているプロセサチップの事を、SoC(System On Chip)などと呼ぶこともありますが、基本的にこれら、マイコンと CPU, SoC は本質的には何も変わらないと思います。敢えて異なる点を挙げるとすると、「得意分野が異なる」といったところでしょうか。

マイコンは外部の機器を制御するという目的に適しています。

例えば、身近に存在するさまざまな機器（掃除機、洗濯機等の家電や、自動車や航空機等の乗り物、身近にある大多数の機器と言っても良いでしょう）に、マイコンは使用されています。

組み込みプログラムとは？

プログラムというのは、コンピュータにとっての予定表のようなもので、「行うこと」が「行う順番」に記載されているものです。コンピュータは、プログラムに沿って動作を行います。

マイコンもコンピュータ（計算機）の一種ですので、プログラムに従った動作を行うという点は変わりません。

プログラムはパソコンやスマートフォンを動かしたり、至る所で使われていますが、マイコン向けのプログラムを「組み込みプログラム」と呼ぶ事があります。「組み込みプログラム」というと、何か通常のプログラムと別なものになるのでしょうか？

パソコン上で動いているアプリケーションプログラムと組み込みプログラムは、本質的な違いはないと思います。どちらもコンピュータ上で実行する命令を書き下したものです。

組み込みプログラムは、機器に組み込まれたプログラムという意味で、機器（ハードウェア）に近い立ち位置のプログラム（ソフトウェア）となるかと思います。外から見ると、ハードウェアの一部に見えるかも知れません。

パソコンやスマートフォンで動作するプログラムに慣れた方でも、マイコン向けの組み込みプログラムというと、ちょっと（かなり？）勝手が違うのは事実で、「組み込み」というキーワードが付いただけで、異世界の存在に感じてしまう方も少なくないかも知れません。

本キットは、マイコンや組み込みプログラムが身近ではないという方をターゲットにしてますので、あまり身構えずに、マイコンを自由に動かす事を楽しんで頂ければと思います。

マイコンのプログラムの特徴って？

マイコン向けのプログラム（組み込みプログラム）は、何か特別な点があるわけではありませんが、いくつか特徴のようなものはあるかも知れません。

・ハードウェアとソフトウェアを切り離して考える事ができない

マイコンは、ハードウェアを制御する目的で使用されることが多いため、ソフトはソフト、ハードはハードのような切り分けができないのかと思います。

パソコンのアプリケーションプログラム開発のときに、マザーボードや CPU の回路図を見ながらというケースはまず無いと思いますが、マイコンのプログラムでは、ボードの回路図を見ながら行うというのが普通です。

要は回路図が読めないとプログラムが書けないといった側面があるのです。

・マイコンの中にタイマや通信モジュール等のハードウェアが備わっており、それらを制御する必要がある

マイコンの中には、種々のモジュールが内蔵されており、マイコンを使いこなすためにはそれらのモジュールを制御する必要があります。

この時は、マイコンの「ハードウェアマニュアル」という資料を読む事になります。文字通り、「ハードウェア」について記載されているマニュアルですので、純粋なソフト屋さんにはピンと来ない点があるかも知れません。

・メモリにデータを書き込むとランプが点灯する？

マイコンには特定のアドレスにデータを書き込むと、端子（マイコンの足）の電圧が 0→3.3V に変化したりします。

通常メモリにデータを書き込むといった処理は、メモリ中のデータが書き換わるというイメージですが、マイコンのハードウェアを制御する場合もメモリのアクセスと同じような手法となります。

・プログラムを間違えると全体が止まってしまう

パソコンのアプリケーションですと、プログラムに致命的なエラーがあった場合、「ハードディスクのブートセクタを壊す」「他のアプリケーションや OS の使用しているメモリ領域を壊す」といった様な処理は OS がブロックしてくれたりもします。OS の動作を含め完全に止まってしまうケースはそう多くないかと思います。

それに対し、組み込み系のプログラムでは、暴走時の保護機構等はありますが、基本的にはプログラムで「何でも」できてしまいます。ブートセクタに相当する領域を上書きする事も容易にできます。場合によっては、縛りがなく自由という考え方もできるかと思います。

・プログラム全体を手の内に

パソコンのアプリケーション作成時は、OS や API が提供する機能を使わずにプログラムを行うという事は現実的ではないかと思います。自分以外の人が書いたコードと協調して動かす事が少なからず求められるかと思います。

それに対し、組み込みプログラムは、本当の意味での「フルスクラッチ」「ゼロベース」で、プログラミングできるということが言えるかと思います。

・メモリが潤沢には使用できない

SmartRX!!!マイコンボード搭載マイコンのメモリ(RAM)は、32KB です。プログラムを格納するコード領域(ROM)は、128KB です。

一昔前の組み込み環境から見ると、RAM も ROM も潤沢にあり、メモリを節約したプログラムコードを書く事に専念しなくても良いレベルですが、パソコンやスマートフォンから見るとかなり少ないですので、この点は、節約の意識が必要でしょうか。

・処理速度がコントロールできる

パソコンやスマートフォン用のアプリは、搭載 CPU のクロック速度もまちまちで、他のタスクとの関係で、自分のアプリケーションにどの程度の CPU リソースが割り振られるかも決まってはいません。

それに対し、組み込み系では、クロック周波数を（ある程度）プログラマが指定できますので、命令の実行速度はプログラマの手の内にあります。また、外からの影響のないプログラムであれば、一連のルーチンの処理速度はほぼ決まるので、1 秒以内に必ず処理を終わらせたいといった要求に応える事ができます。

・開発言語は？

組み込みのプログラムで使用されるのは、

C 言語(C++)
アセンブリ言語

が主です。もちろん、他の言語向けのコンパイラやインタプリタを用意しているマイコンもありますが、基本はこの 2 種の言語となると思います。本チュートリアルでは、C 言語をベースにします。

1. TUTORIAL0

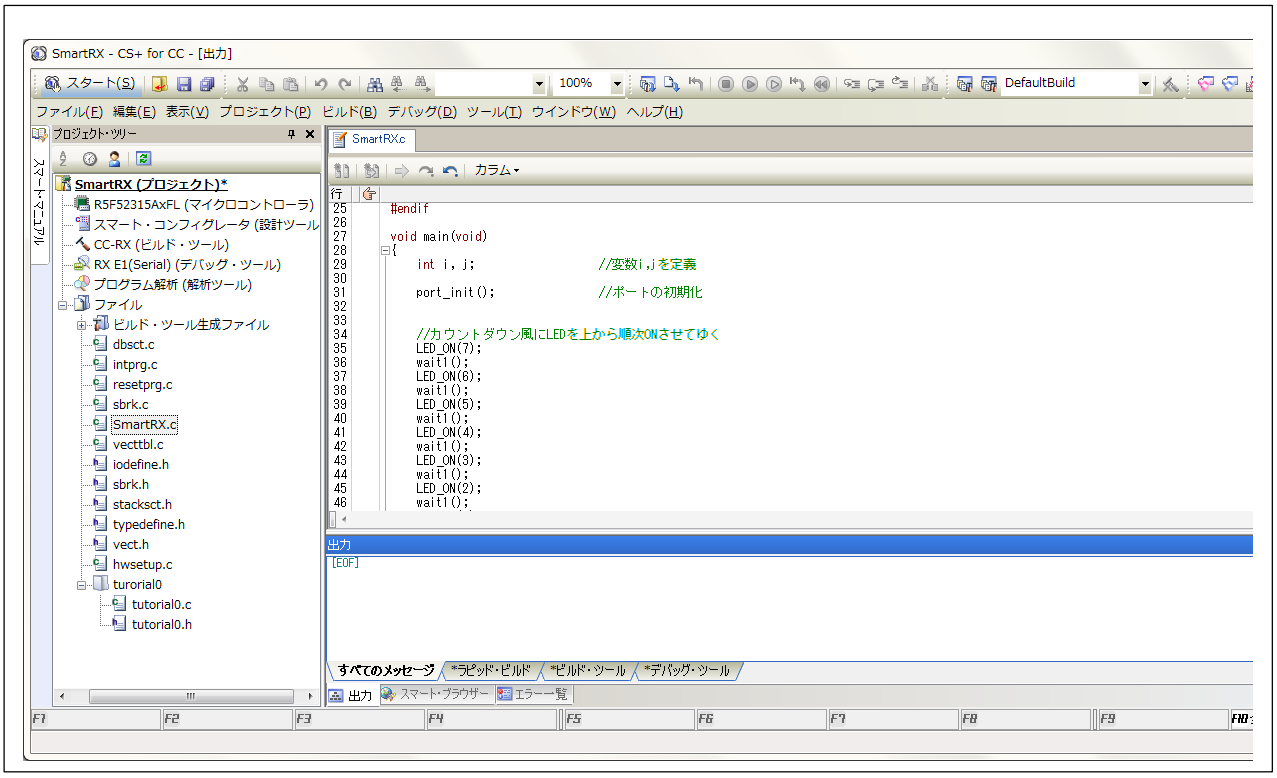
最初のチュートリアルとなります。

CD 内の、TUTORIAL¥TUTORIAL0

以下を、PC のドライブにコピーしてください。

[コピー先フォルダ]¥TUTORIAL0¥SmartRX.mtpj

をダブルクリックで CS+を起動してください。



色々なファイルがありますが、

SmartRX.c

が、メイン関数を含むファイルとなります。それ以外にも重要なファイルはいくつかあるのですが、とりあえず他のファイルは「おまじない」とここでは考えてください。

SmartRX.c[抜粋]

```
void main(void)
{
    int i, j;//変数i,jを定義

    port_init();//ポートの初期化

    //カウントダウン風にLEDを上から順次ONさせてゆく
    LED_ON(7);
    wait1();
    LED_ON(6);
    wait1();
    LED_ON(5);
    wait1();
    LED_ON(4);
    wait1();
    LED_ON(3);
    wait1();
    LED_ON(2);
    wait1();
    LED_ON(1);
    wait1();
    LED_ON(0);
    wait1()
```

この部分では、ボード上の LED を LED7 から LED0 の方向に、順次点灯させていきます。

最初に LED7 が点灯し、ちょっと待つ。次に、LED6 が点灯して…という処理です。

port_init()

は関数で、初期化のために、最初に呼びだす事が必要です。

wait1()

も関数で、この関数を呼び出すと、1 秒程度ウェイトが入ります。

LED_ON(n)
LED_OFF(n)

は、マクロと呼ばれるもので、n=0~7 を入れて呼び出すと、ボード上の LED が点灯・消灯します。

プログラムのビルドと書き込みに関しては、「SmartRX 学習キット　スタートアップマニュアル（始めにお読みください）」を参考に行ってください。

この部分を書き換えて、LED の点灯のパターンが変わる（制御できている）事を確かめてみてください。

※TUTORIAL0 では、「ファイルの書き換え」「ビルド」「書き込み」のフローを理解頂くため、意図的に構文エラーとなるコードを入れていますので、ファイル内のコメントを見て正しい構文に修正後ビルドを行ってください

本チュートリアルでは、LED の点灯のパターンや制御が実際に行える事を理解するという所までとなります。実際に、どの様なプログラムを書けば LED が制御できるかは、次のチュートリアルとなります。

2. LED_SW

ボード上の

LED（LED0~7 の 8 つの LED）

DIP スイッチ(SW2, 4ch)

プッシュスイッチ(SW3, SW4)

を制御する方法を学ぶチュートリアルとなります。

まず、LED がマイコンとどの様に接続されているか（回路図）を理解する必要があります。

・LED とマイコンの接続

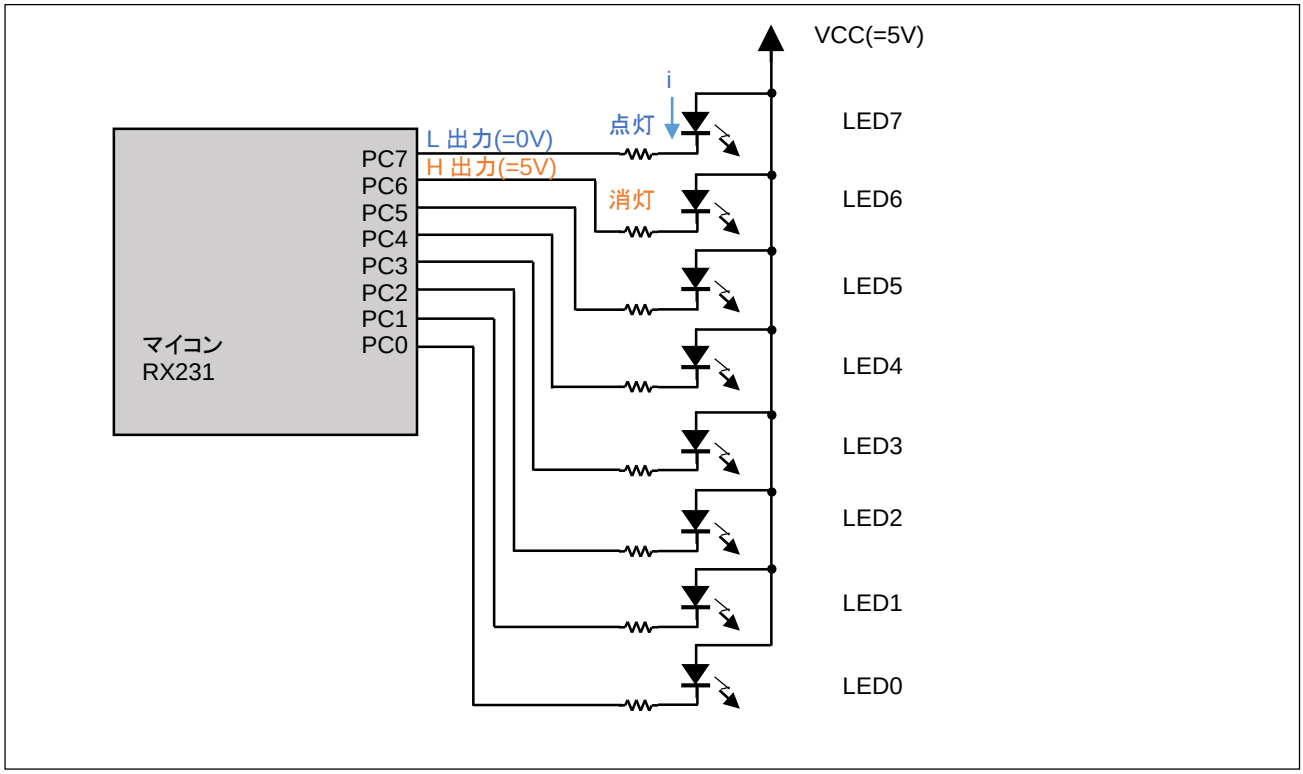


図 2-1 LED の接続

LED とマイコンは図の様に接続されています。ボードに 5V の電源を印加した場合、VCC と書かれている場所は 5V となります。

ここで、マイコンの PC7 端子を L 出力制御すると、LED7 の両端に電圧の差が生じ、電流（図中の i）が流れますので、LED7 が点灯する事となります。

・端子と LED の関係

PC7=L 出力	LED7 は点灯
PC7=H 出力	LED7 は消灯

ここで、注意したいのが、LED とマイコンの端子の接続状態です。

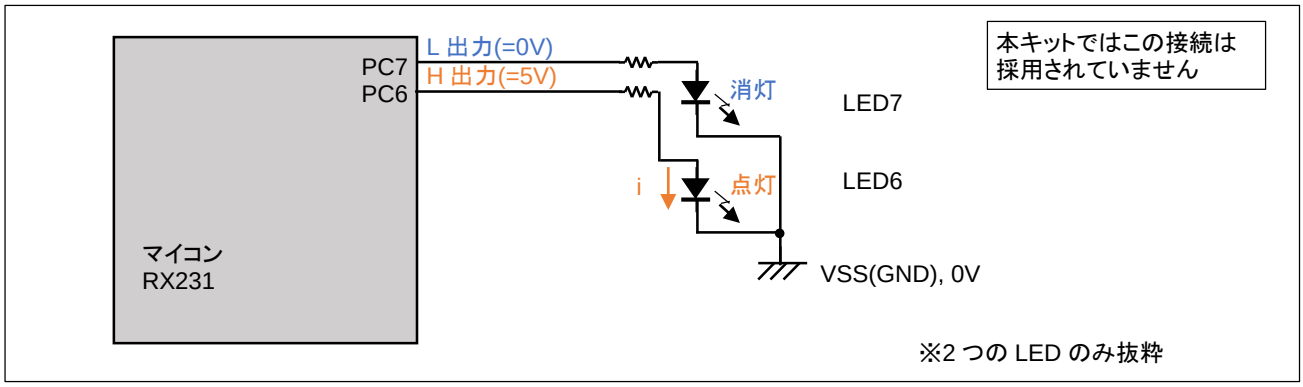


図 2-2 LED の接続(2)

回路図が LED の接続(2)の様になっていた場合は、まったく逆の結果（L 出力で消灯、H 出力で点灯）となります。要は回路図に応じて、制御を考えなくてはならないという事です。

LED の点灯・消灯を制御する場合に使用するマイコンの機能は「I/O ポート」です。「I/O ポート」の機能では、マイコンの端子を

・出力モード
　出力は H とする
　出力は L とする
・入力モード

に設定可能です。

出力モードと入力モードの切り替えは、マイコンの特定のアドレスにデータを書き込む事で行われます。

アドレスとしては、0x0008 C00C　です。このアドレスを操作すれば良いのですが、アドレスというのは数字の羅列で覚えにくいし、取り違いを起こしやすいものだと思います。

そこで、開発環境側で予め、アドレスと「名前」の対応表を用意してくれています。対応表はファイルになっており、ファイル名は、iodefine.h　です。

プログラムでは、

SmartRX.c[抜粋]

```
#include"iodefine.h"
```

という 1 行を書いておけば良いです。このファイルを使用すると、先程のアドレスが

PORTC.PDR.

という名称でアクセスできる様になります。端子名が PC7（PC6...PC0, ポートの C）なので、PORTC。Port Direction Register 略で、PDR です。ポートの方向（入力か出力か）を決めるレジスタ（記憶領域）です。

出力の L と H を決めるのは、

PORTC.PODR.

となります。Port Output Direction Register の略です。

PORTC.PDR, PORTC.PODR は、8bit(1Byte)のレジスタとなっており、Byte 単位でアクセスする場合は

PORTC.PODR.BYTE = 0x1F;

の様に記載します。

0x1F = 0b00011111

ですので、

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	1	1	1	1	1
PC7 =L	PC6 =L	PC5 =L	PC4 =H	PC3 =H	PC2 =H	PC1 =H	PC0 =H
LED7 点灯	LED6 点灯	LED5 点灯	LED4 消灯	LED3 消灯	LED2 消灯	LED1 消灯	LED0 消灯

となります。当該ビットを 0 にした場合端子からは L が出力され、1 にした場合は端子からは H が出力されます。

入出力の設定は、

PORTC.PDR.BYTE = 0xFF;

で、PC7~PC0 が出力設定となります。（0 が入力、1 が出力です）

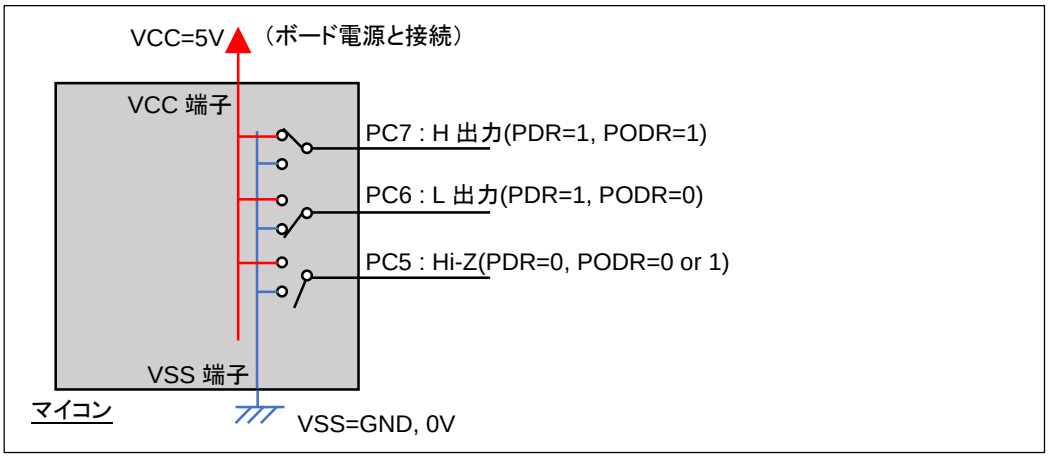
なお、PORTC.PDR, PORTC.PODR レジスタは、ビット単位でのアクセスも定義されており、

PORTC.PODR.BIT.B3 = 1;

と書けば、bit3, PC3 を H にするという意味となります。

コラム　L 出力, H 出力とは

デジタル回路では、L/H,0/1 という表現を良く使います。



H 出力(1)とは PC7 等の端子につながっているマイコンの中のスイッチが VCC に接続されている状態です。

ボード電源電圧（=マイコン VCC 端子電圧）が 5V のときは、5V となります。ボード電源電圧が 3.3V のときは、H 出力は 3.3V となります。

L 出力(0)とは、マイコンの中のスイッチが VSS(GND)に接続されている状態で、0V となります。

PDR を 0 に設定した場合は、スイッチが VCC にも VSS にもつながらない状態となります。この様な状態を、ハイ・インピーダンス状態といい、記号では、Z または Hi-Z と表します。端子を入力端子として使用する場合はこの設定にします。

H,L,Z の 3 状態を設定できる回路を、トライステート（3 条件）バッファなどと呼ぶ事があり、マイコンの I/O ポートは、ほとんどがトライステートバッファの構成となっています。

H 出力は、デジタル的には 1 と表現しますが、ボード（システム）の電源電圧により、電気的には 5V だったり 3.3V だったりしますので、注意が必要です。

端子の真理値表を示します。端子を入力モードに設定した場合は、LED は消灯となります。

真理値表

PDR	PODR	端子	備考
0	X	Z	LED は消灯
1	0	L	LED は点灯
1	1	H	LED は消灯

次に、スイッチがマイコンとどの様に接続されているか（回路図）を理解する必要があります。

・スイッチとマイコンの接続

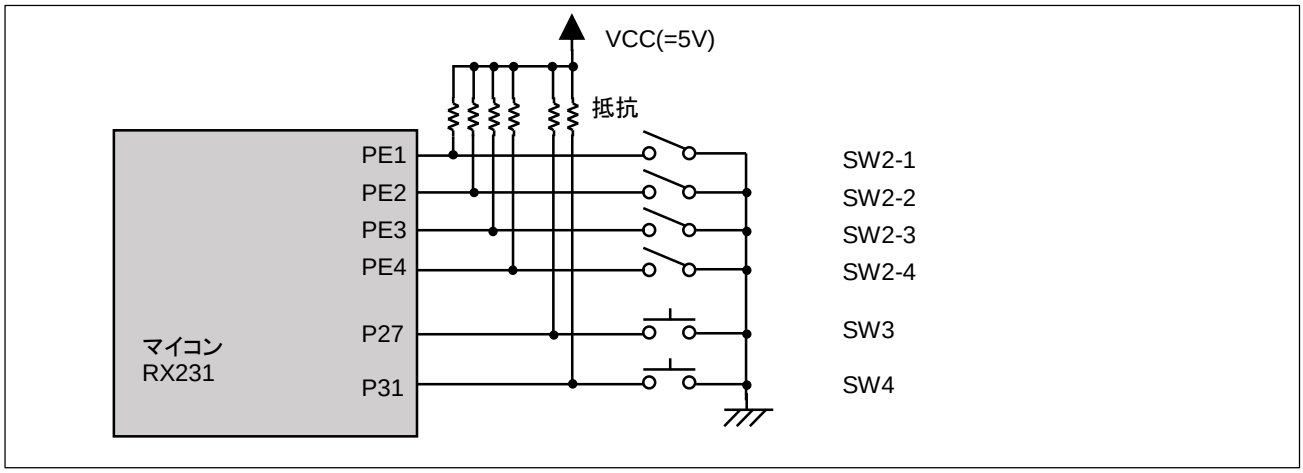


図 2-3 スイッチの接続

マイコンとスイッチ(SW2:DIP-SW, SW3,SW4:Push-SW)は、上図の様に接続されています。スイッチをオフ状態とした場合は、マイコンの端子には H が入力されます。スイッチをオン状態とした場合は、端子は L となります。

この様な接続を、抵抗で上（電源側）に引き上げることから、プルアップと呼びます。

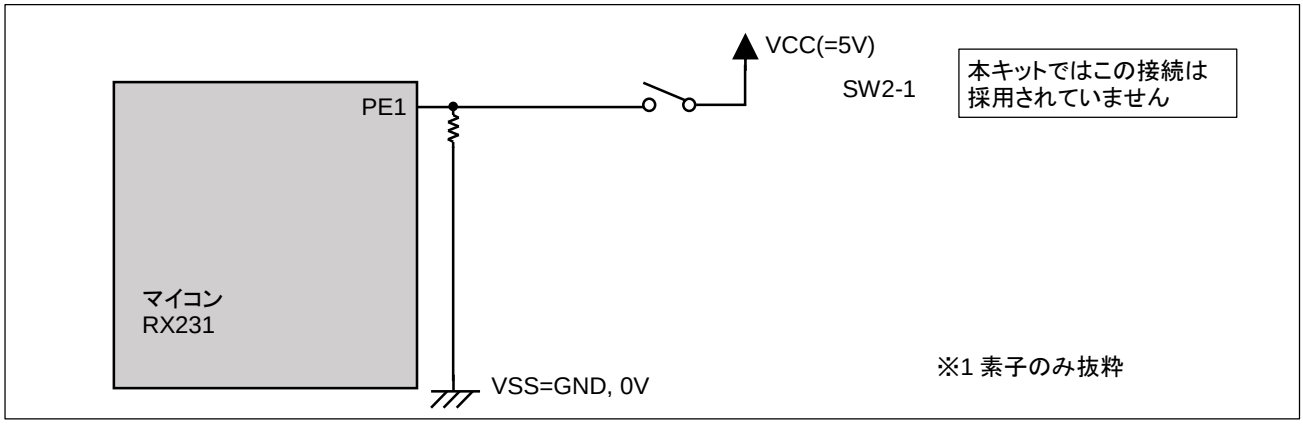


図 2-4 スイッチの接続(2), プルダウン

図 2-4 の様にスイッチとマイコンを接続した場合は、スイッチオン時、端子は H。スイッチオフ時、端子は L となります。図 2-3 の接続とは、極性が逆になります。マイコンとスイッチがどの様に接続されているかは、プログラムを作成する上でも必要な情報となります。

マイコンの端子をスイッチ入力として使用するためには、マイコン側から出力するモードに設定してはいけません。マイコン側からは、コラム「L 出力、H 出力とは」の Z の状態にする必要があります。

出力するかどうかを決めるレジスタは、PDR でしたので、SW2 向けに PORTE.PDR を

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
X	X	X	0	0	0	0	X
PE7 =?	PE6 =?	PE5 =?	PE4 =Z	PE3 =Z	PE2 =Z	PE1 =Z	PE0 =?

に設定する必要があります。この場合は、ビット単位での設定

```
PORTE.PDR.BIT.B4 = 0;
PORTE.PDR.BIT.B3 = 0;
PORTE.PDR.BIT.B2 = 0;
PORTE.PDR.BIT.B1 = 0;
```

が使えます。

P27 と P31 の設定は、

```
PORT2.PDR.BIT.B7 = 0;
PORT3.PDR.BIT.B1 = 0;
```

で良いです。

コラム　特定のビットのみ変更する演算

2 進数(0b00010010)、10 進数(18)、16 進数(0x12)に関しては、本書では解説を省略します。2 進数と 16 進数は、本書でも頻繁に出てきますので、ピンと来ないという方は、プログラムの入門書等で勉強してください。

先程の解説で、PORTE.PDR を

0bXXX0000X　　(1)

に設定しましたが、バイト単位で設定する場合どのように行えば良いのでしょう。

(1)で、0 のところは 0 に設定したい。X のところは、現在の設定値を変えたくないとします。

ここで、(1)の X のビットを、1 とした値を考えます

0b11100001

です。16 進数では、0xE1 となります。

PORTE.PDR.BYTE = PORTE.PDR.BYTE & 0xE1;

とすれば、（&は、ビット単位での AND 演算）

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
元の値 0xD9 とする	1	1	0	1	1	0	0	1
0xE1	1	1	1	0	0	0	0	1
0xD9 & 0xE1	1	1	0	0	0	0	0	1

bit1,2,3,4 を 0 に変更する事ができます。上記式は

PORTE.PDR.BYTE &= 0xE1;

と演算、代入を 1 つの演算子(&=)で行う記載でも等価で、こちらの方がスマートです。

今度は、逆の操作

0bXXX1111X　　(2)

bit1,2,3,4 のみ 1 にしたい場合はどうすれよいのでしょうか。(2)の X のところは 0, 1 のところは 1 の値を作り、

0b00011110 = 0x1E

元の数と OR を取れば良いです。

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
元の値 0xD9 とする	1	1	0	1	1	0	0	1
0x1E	0	0	0	1	1	1	1	0
0xD9 | 0x1E	1	1	0	1	1	1	1	1

PORTE.PDR.BYTE |= 0x1E;

です。

特定ビットを 0 にしたい時は、変更したくないところを 1 とした値を作り、特定のビットを 0 にしたい時は、変更したいところを 1 にした値を作りましたが、頭の切り替えが面倒という場合、

常に変更したい値を 1 （この場合は、0x1E）

で考え、

PORTE.PDR.BYTE |= 0x1E;　　b4-1 を 1 にしたい場合
PORTE.PDR.BYTE &= ~0x1E;　　b4-1 を 0 にしたい場合

~（ビット毎の反転）

を使うという手もあります。

本チュートリアルでのサンプルプログラムは、大抵 "&= ~ " の形式で特定のビットを 0 に変更しています。

スイッチの入力ポートを、PDR で入力設定にした後、端子の状態を読み込む場合、

PORTE.PIDR.

というレジスタにアクセスします。端子のレベルが、L であれば 0。端子のレベルが H であれば、1 となります。

PORTE.PIDR.BYTE

で、PORTE の 8 端子全ての状態。

PORT2.PIDR.BIT.B7

で、P27 の状態のみ読み込めます。

LED_SW チュートリアルのプログラムは以下となります。スイッチの読み取りと、LED の制御を行っているプログラムです。

SmartRX.c[抜粋]

```
#include "iodefine.h"
#include "led.h"
#include "sw.h"

void main(void)
{
    unsigned char dip_sw_state;

    led_init();//LEDポートの初期化
    sw_init();//SWポートの初期化

    while(1)//無限ループ
    {

        if(PORT2.PIDR.BIT.B7 == 0)//SW3が押されていれば
        {
            PORTC.PODR.BYTE = 0x00;//LEDを全部点灯とする
        }
        else if(PORT3.PIDR.BIT.B1 == 0)//SW4が押されていれば
        {
            PORTC.PODR.BYTE = 0xFF;//LEDを全部消灯とする
        }
        else
        {
            dip_sw_state = PORTE.PIDR.BYTE;//DIPスイッチの読み取り

            PORTC.PODR.BYTE = dip_sw_state;//LEDをDIPスイッチの状態と合わせる
        }
   }

    //マイコンのプログラムでは、main関数から抜けないように
    //（この場所には来ないように）
    //する
}
```

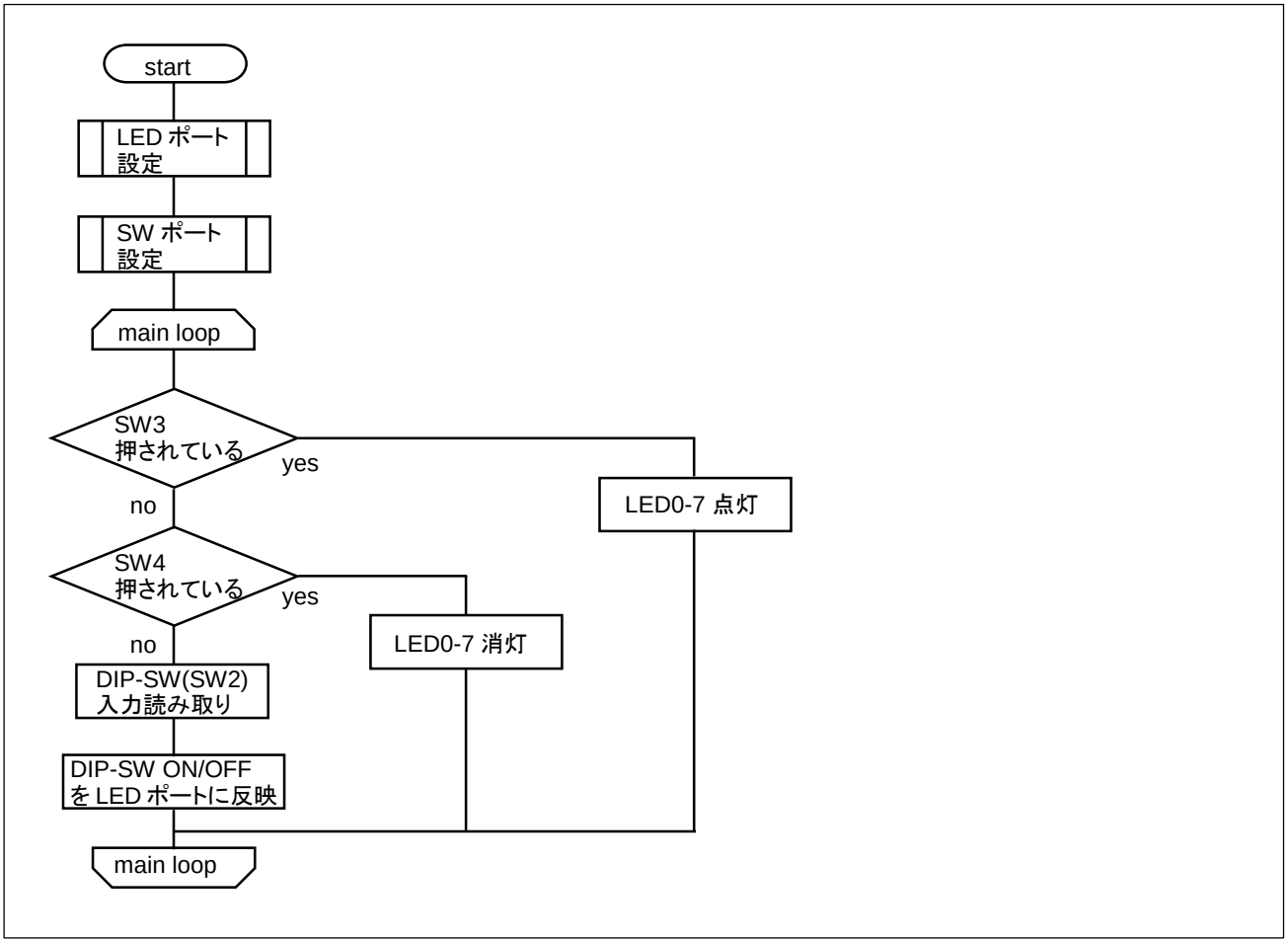
led_init(), sw_init() 内では、PDR の初期設定をしています。

初期設定後は、無限ループになっており、プッシュスイッチと DIP スイッチを読み取り、LED の ON/OFF を行うプログラムになっています。出力の設定である、PODR，入力の値である PIDR レジスタを使用しています。

LED_SW チュートリアルのフローチャートを示します。

ーフローチャートー

メイン関数　main()



SW3 を押している間は、LED 全点灯。SW4 を押している間は、LED 全消灯。SW3,4 が押されていないときは、DIP-SW(SW2)と LED が連動します。

本プログラムでは、スイッチの状態により、LED が点灯・消灯するだけですが、一般的な家電製品では、LED のところが他のもの（Bru-ray ディスクの再生等）になっているだけで、基本的にはこのチュートリアルのような処理が行われているとお考えください。

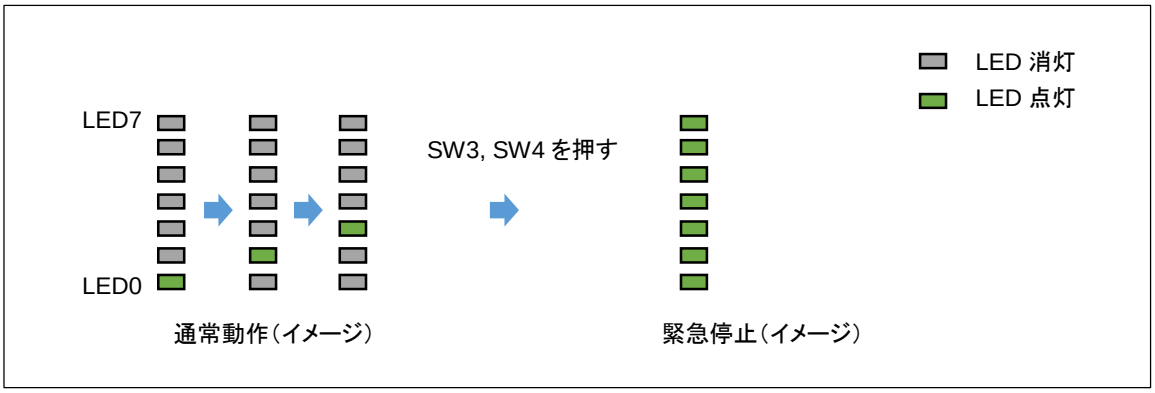
3. IRQ

このチュートリアルでは、割り込みの基礎を学びます。

IRQ は Interrupt ReQuest の略で、割り込み要求といった意味合いです。

割り込みとは、言葉の意味と同じで、予め決められている順番をすっ飛ばす処理の事です。

・本プログラムの動作



ボードにプログラムを書き込み電源を投入すると、LED0→LED1→...→LED7 の順で 1 つだけ LED が点灯します。LED7 までいくと、点灯する LED が LED6→LED5...と折り返すような動作をします。この状態が通常動作のイメージです。

SW3, SW4 は緊急停止ボタンのイメージで、SW3, SW4 のいずれかを押すと、LED が全点灯となり、システムが停止します。（実際プログラムの動作は停止します）

SW3 は、LED_SW のチュートリアルで使用した手法（I/O ポート）として読み取りを行っています。

それに対し、SW4 は本チュートリアルで初出となる、割り込みの機能を使っています。

ボタンを押してからシステムが停止するまでの時間が SW3 と SW4 で異なる事を確かめてください。（SW3 はしばらく押したままにしないと反応しません）

※停止状態から再度プログラムを走らせるためには、SW1(RESET)を押してください

SW3 は、LED が端（LED0 か LED7）に来たタイミングでのみ、ボタンを押したのが有効になります。それに対し、SW4 はどの段階でも、ボタンに反応するはずです。

プログラムの処理で、SW3 は数秒に 1 回しか読み取りの処理が行われません。それに対し、SW4 は押したタイミングでマイコンに割り込みが入ります。

マイコンは、SW4 が押されたタイミングで、今行っていた処理（LED を順次点灯させる）を一旦保留にして、割り込みの処理を実行します。そのため、SW4 にはいつでも反応するという事となります。

－本チュートリアルで使用している関数（抜粋）－

irq1_init

概要：IRQ1 初期化関数
宣言：
　　void irq1_init(void)
説明：
　・IRQ1, P31 の初期化
を行います

引数：
　　なし
戻り値：
　　なし

　P31 を IRQ1 として設定します。
　割り込み優先度=10
　fall エッジ検出
　デジタルフィルタ有効（PCLK/64）

irq1_start

概要：IRQ1 開始関数
宣言：
　　void irq1_start(void)
説明：
　・IRQ1 の動作開始を行います

引数：
　　なし
戻り値：
　　なし

irq1_stop

概要:IRQ1 停止関数

宣言:

　　void irq1_stop(void)

説明:

　・IRQ1 の動作停止を行います

引数:

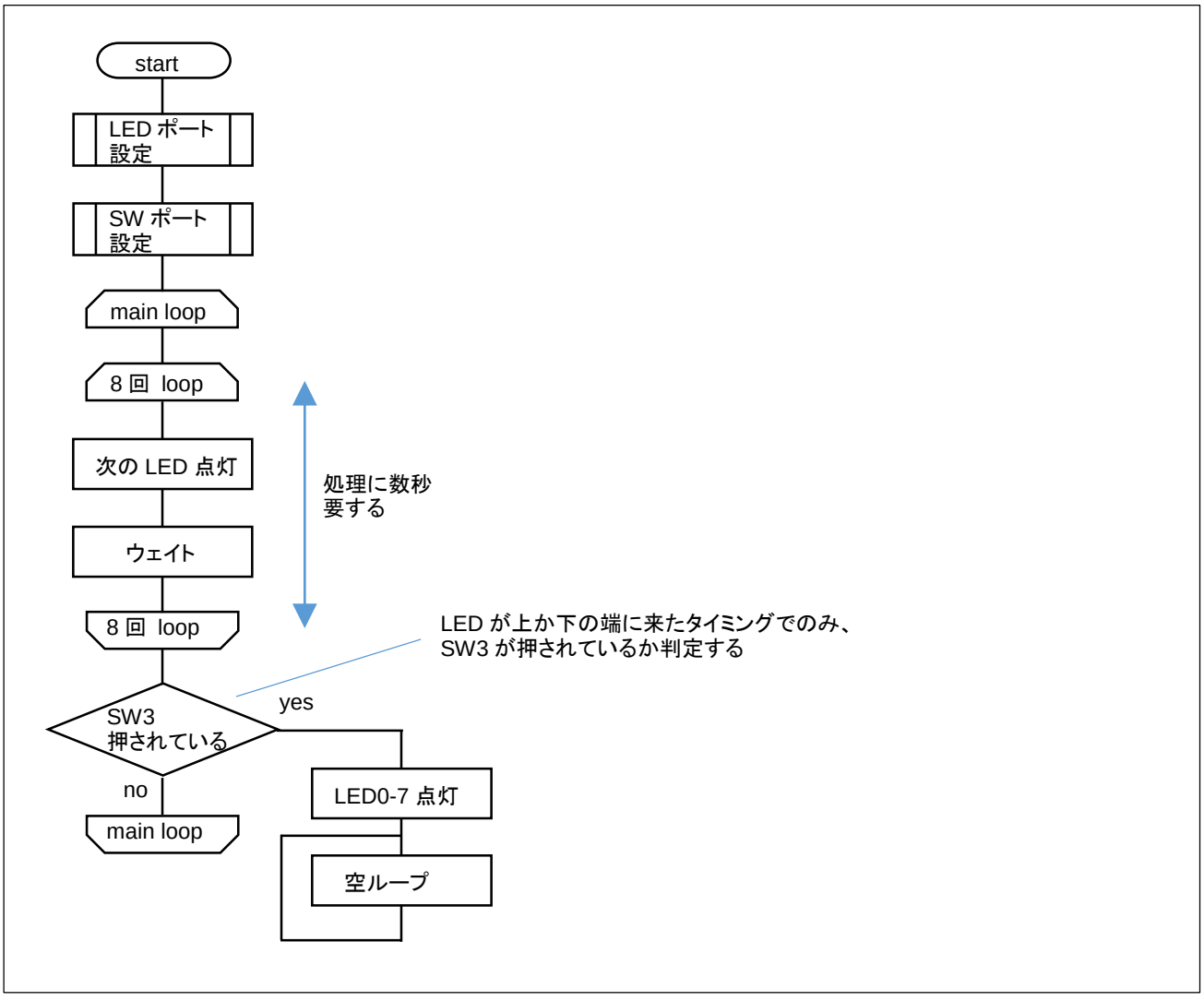
　　なし

戻り値:

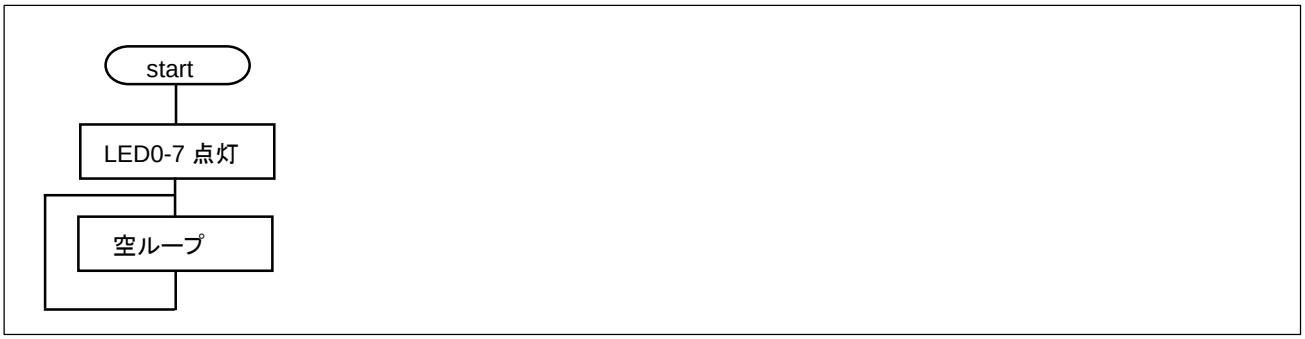
　　なし

ーフローチャートー

メイン関数　main()



割り込み関数 Intr_ICU_IRQ1()



4. LCD

このチュートリアルでは、キャラクタ型 LCD(SC1602)の制御を行います。
LCD は、製品に付属しています。

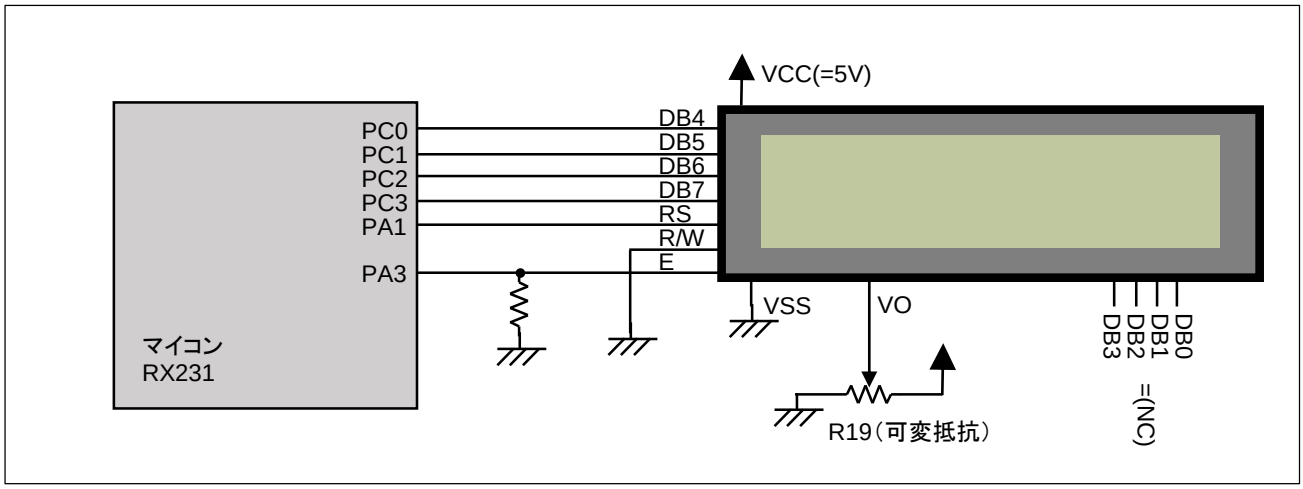


図 4-1 LCD の接続（回路図）

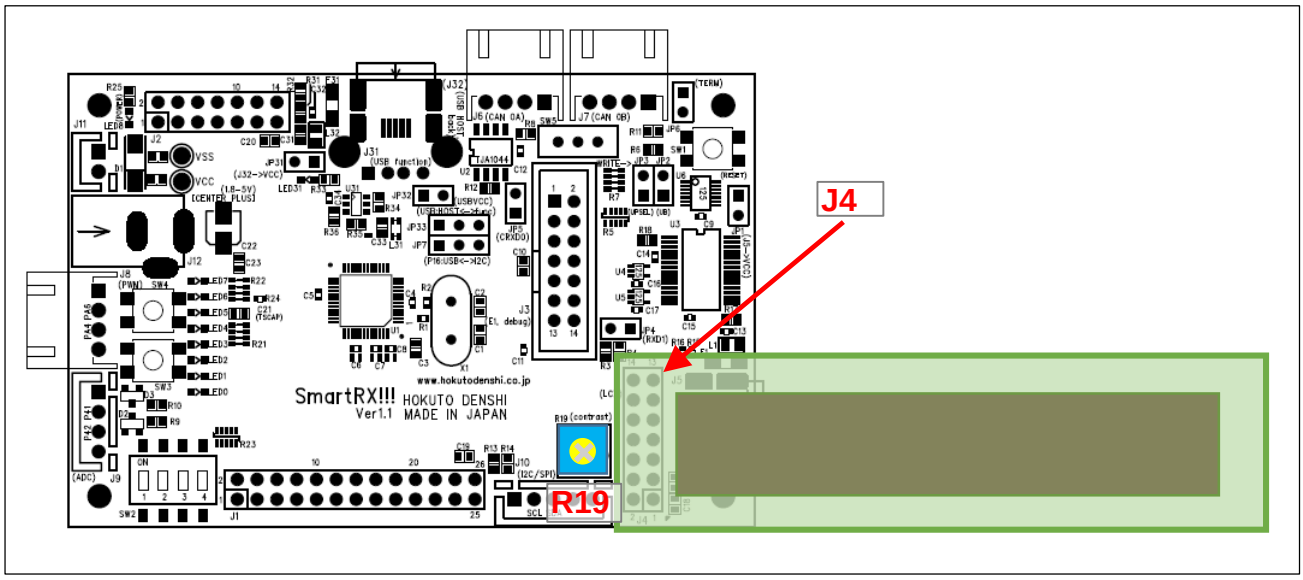


図 4-2 LCD の接続

製品に付属している LCD は、裏面に 14P のピンコネクタが実装されていますので、SmartRX!!!マイコンボード J4（ピンヘッダ実装）に、差し込む様に接合してください。

(NC) = No Connect, どこにも接続されていないという意味です

表 4-1 SC1602 LCD インタフェース信号表 (J9)

No	信号名	接続先	備考
1	VDD	VCC	
2	VSS	VSS	
3	VO	コントラスト電位	R19(可変抵抗) 時計回り:表示薄い 反時計回り:表示濃い
4	RS	PA1	
5	R/W	VSS	
6	E	PA3	プルダウン
7	DB0	(NC)	
8	DB1	(NC)	
9	DB2	(NC)	
10	DB3	(NC)	
11	DB4	PC0	LED ポートと兼用
12	DB5	PC1	LED ポートと兼用
13	DB6	PC2	LED ポートと兼用
14	DB7	PC3	LED ポートと兼用

(NC)は未接続です。

SC1602 タイプの LCD は、マイコンからコマンドを送る事により、表示や画面のクリアを行います。
DB0~DB7 の 8bit 幅でアクセスする手法と、DB4~DB7 の 4bit 幅でアクセスする手法がありますが、本ボードでは後者の手法としています。(マイコンからの信号は、合計 6 本でのアクセスとなります)

基本的に、キャラクタ(A-Z, a-z, 0-9)は、8bit の文字コードを持っていますので、1 文字を送るのに、4bit で 2 回に分けて送る方式となります。

LCD の仕様は、巻末に付録として記載しますが、

・R/W=L のときは、マイコン→LCD へデータを送る
・R/W=H のときは、LCD→マイコンにデータを送る(R/W はボードで L 固定しているのでこのモードは使用できない)

・RS=L のときは、コマンド(画面のクリアやカーソルの移動)を送るモード
・RS=H のときは、データ(画面に表示する文字)を送るモード

・E(イネーブル)が、クロックの役割を行う

・DB4-7 が送信データ

となります。

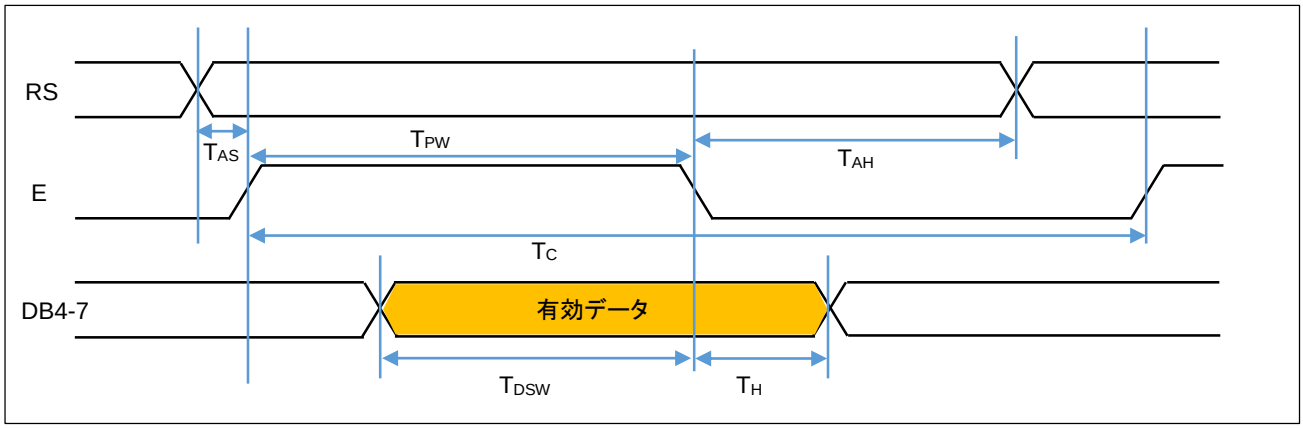


図 4-3　SC1602 LCD ユニット 制御波形

表 4-2 SC1602 LCD ユニット タイミング

シンボル	項目	min	typ	max	Unit
T_{AS}	RS セットアップ時間	0			[ns]
T_{AH}	RS ホールド時間	10			[ns]
T_C	E(イネーブル)周期	1200			[ns]
T_{PW}	E(イネーブル)パルス幅	140			[ns]
T_{DSW}	データセットアップ時間	40			[ns]
T_H	データホールド時間	10			[ns]

規定されている波形と、タイミングを示します。ここで見るべきポイントですが、

・DB4-7 にデータをセット(L/H を確定)させてから、E を H→L に変化させると、LCD ユニットはデータを取り込む
・E の H 期間は、140ns 以上必要(TPW の規定)
・E の周期は、1200ns 以上必要(TC の規定)
・RS を確定させてから、E を L→H に変化させるタイミングは 0ns 以上であればよい(TAS の規定)(*1)
・E を H→L に変化させてから、RS は最低 10ns は変化させてはいけない(TAH の規定)(*2)
・DB4-7 を確定させてから、E を H→L に変化させるタイミングは最低 40ns 以上取らなければならない(TDSW の規定)(*1)
・E を H→L に変化させてから、DB4-7 は最低 10ns は変化させてはいけない(TH の規定)(*2)
・上記タイミング規定は mim 側の規定なので、時間を長く取る分には問題はない

となります。ちょっと約束事が多い気もしますが、機械(電子回路)の動作なので、多少のルールが必要になってきます。

ー本チュートリアルで使用している関数（抜粋）ー

lcd_init

概要：LCD 初期化関数
宣言：
　　void lcd_init(void)
説明：
　・LCD の初期化を行います

引数：
　　なし
戻り値：
　　なし

lcd_cmd

概要：LCD コマンド送信関数
宣言：
　　void lcd_cmd(unsigned char c)
説明：
　・LCD にコマンドを送信します

引数：
　　c: 送信コード
戻り値：
　　なし

lcd_hs1, lcd_hs2

概要：1 行目にカーソルを移動させる関数, 2 行目にカーソルを移動させる関数
宣言：
　　void lcd_hs1(void), void lcd_hs2
説明：
　・LCD にコマンドを送信します

引数：
　　なし
戻り値：
　　なし

lcd_clear

概要:LCD 画面クリア関数
宣言:
　　void lcd_clear(void)
説明:
　・LCD の画面をクリアします

引数:
　　なし
戻り値:
　　なし

lcd_write_char

概要:LCD キャラクタ送信関数
宣言:
　　void lcd_write_char(unsigned char c)
説明:
　・LCD にキャラクタ(1 文字)を送信します

引数:
　　c: キャラクタコード
戻り値:
　　なし
使用例:
lcd_write_char('A');
　現在のカーソル位置に、A を表示させる。

lcd_write_hex, lcd_write_byte_int, lcd_write_short_int

概要:LCD 数値表示関数
宣言:
　　void lcd_write_hex(unsigned char c)
　　void lcd_write_byte_int(unsigned char num)
　　void lcd_write_short_int(unsigned char num)

説明:
　・LCD にキャラクタ(1 文字)を送信します

引数:

c, num: 表示する数値

戻り値:

なし

使用例:

lcd_write_hex(0x34);

現在のカーソル位置に、34 を表示させる。

lcd_write_byte_int(120);

現在のカーソル位置に、120 を表示させる。

lcd_write_hex(12345);

現在のカーソル位置に、12345 を表示させる。

lcd_write_str

概要:LCD 文字列表示関数

宣言:

void lcd_write_str(unsigned char *str)

説明:

・LCD に文字列を送信します

引数:

*str: 表示する文字列

戻り値:

なし

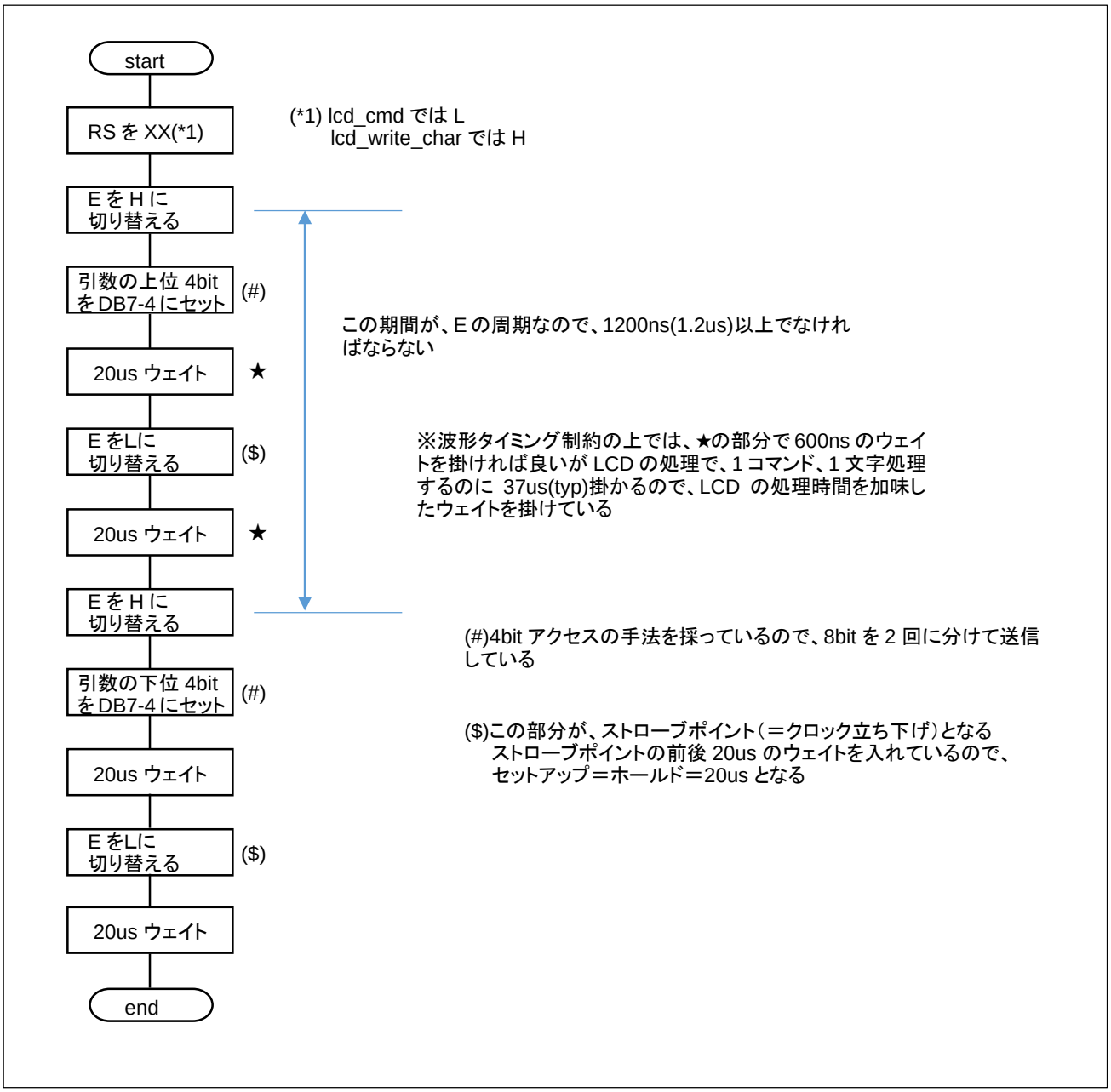
使用例:

lcd_write_str("LCD SAMPLE");

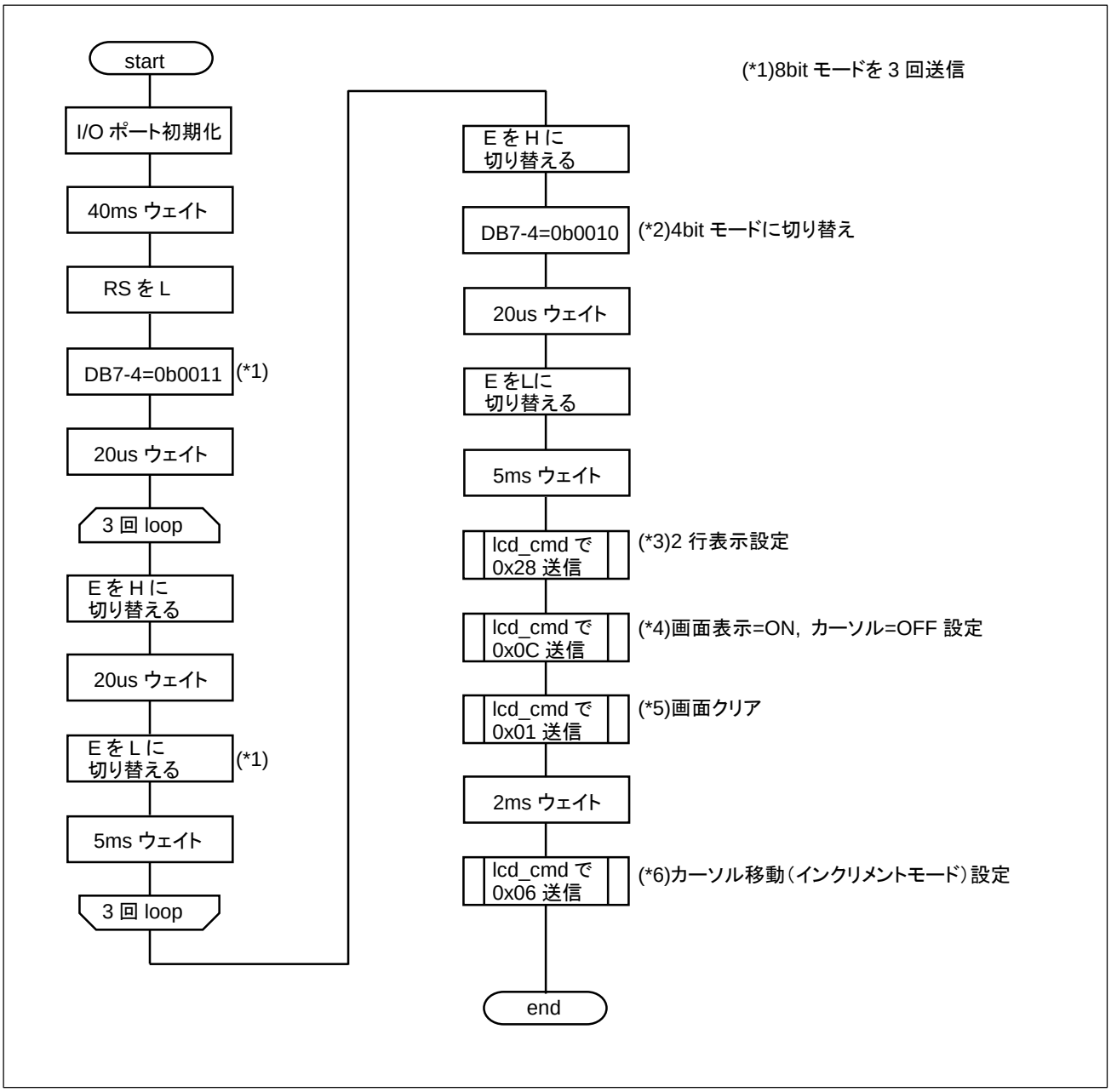
現在のカーソル位置に、LCD SAMPLE を表示させる。

ーフローチャートー

コマンド送信関数 lcd_cmd(), キャラクタ送信関数 lcd_write_char()



LCD 初期化関数 lcd_init()



LCD の初期化関数のフローチャートを示します。マイコンと LCD は 4bit の接続となっているので、最終的には、4bit モードに設定したいのですが、電源投入後は一旦 8bit モードに設定しています。3 回 8bit モード設定のコードを送信していますが、それは、以下の理由となります。

・(1)初期化時 LCD が 8bit モードだった場合

	LCD が受け取るデータ	LCD が認識するコマンド
1 回目	0b0011XXXX	8bit モード切替コマンド
2 回目	0b0011XXXX	8bit モード切替コマンド
3 回目	0b0011XXXX	8bit モード切替コマンド

※XXXX は、マイコンと未接続の DB3-0 のレベルとなります（未接続の場合 LCD 内蔵のプルアップが有効となり実際は 1 となります）

・(2)初期化時 LCD が 4bit モードだった場合　上位 4bit のデータ受信シーケンスからのスタート

	LCD が受け取るデータ	LCD が認識するコマンド
1 回目	0b0011（上位 4bit）	8bit の内半分の上位 4bit のデータの受信完了
2 回目	0b0011（下位 4bit）	8bit で 0b00110011 データの受信→8bit モード切替コマンド
3 回目	0b0011XXXX（8bit モード）	8bit モード切替コマンド

・(3)初期化時 LCD が 4bit モードだった場合(3)　下位 4bit のデータ受信シーケンスからのスタート

	LCD が受け取るデータ	LCD が認識するコマンド
1 回目	0b0011（下位 4bit）	受信済みの上位 4bit と組み合わせたデータ 0b????0011 を受信
2 回目	0b0011（上位 4bit）	8bit の内半分の上位 4bit のデータの受信完了
3 回目	0b0011（下位 4bit）	8bit で 0b00110011 データの受信→8bit モード切替コマンド

※LCD の動作モードが 4bit モードで、上位 4bit の受信が終わっており、下位 4bit のデータを待っている状態で、マイコンのリセットが掛かり、LCD 初期化を開始したケース

LCD の状態（8bit モード、4bit モード、4bit モードで上位 4bit のデータを受信済みの場合）に拘わらず、0b0011 のコマンドを 3 回送る事により、LCD は 8bit モードへ設定ができます。その後、8bit モードでの状態で、4bit モード切替コマンドを送信する事により、確実に 4bit モードに入れる事ができます。
（一度 8bit モードに入れる事により、4bit の上位・下位のどちらから受信を開始するかという問題をクリアできます）

－サンプルプログラムの動作－

LCD 使用時は、電源電圧を 5V としてください。(LCD の動作電圧が 4.5~5.5V となっているためです)
(※市販の 3.3V の LCD モジュールを使用した場合は、3.3V でも LCD が使用可能です)
LCD 接続、電源投入後、可変抵抗を精密ドライバ等で回して、液晶の画面の濃さを調整してください。

サンプルプログラムを起動すると、
LED0 が点灯
の状態となります。
SW3 :点灯している LED が一つ上に移動
SW4 :点灯している LED が一つ下に移動

LCD 画面に下記表示が出力されます。

```
LED,SW DEMO
DIP-SW:■■□□
```

2 行目は DIP-SW の ON/OFF に連動しており、ON:■, OFF:□となります。

※■□はユーザ定義文字で、文字コード 0x00, 0x01 にドット単位でパターンを定義しています
(SC1602 は、8 文字分のユーザ定義パターンを作成可能なので、簡単な図形等の作成、表示が可能です)

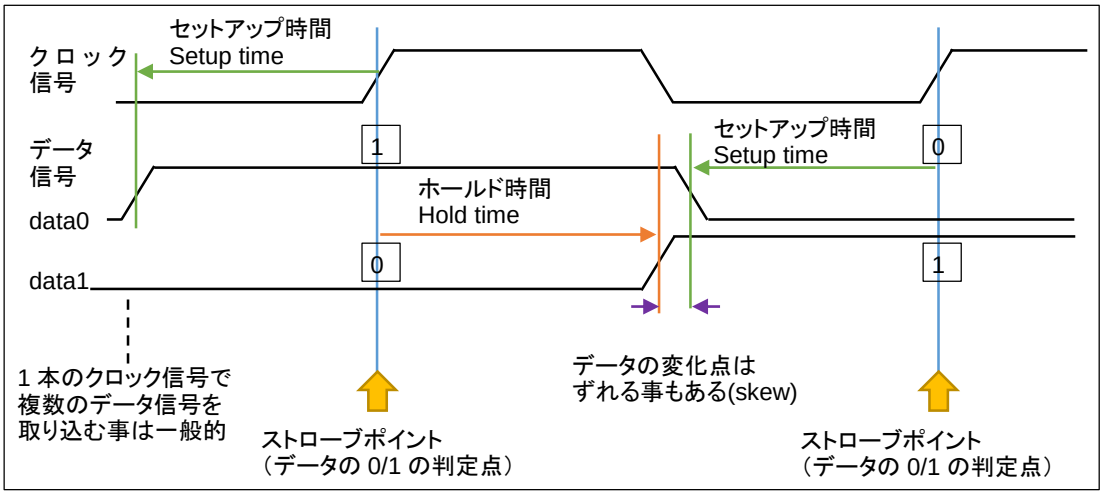
※SW3, SW4 はチャタリング(ボタンを押した際、電気的な接点がバウンドして何回か ON/OFF が切り替わる事)をキャンセルする簡単な例となっています
プログラムでは、5 回連続して同じ状態(ON or OFF)が繰り返されたとき、入力が有効になる様にしています

※LCD にコマンドやキャラクタを送信するのに、PC0-PC3 を使用しています
PC0-PC3 は、LED の制御にも使用されていますので、LCD 制御の際、一瞬 LED0-3 が点滅します
(LCD にコマンドやキャラクタを送信した後は、LCD アクセス前の LED の状態に戻します)

なお、LED 切り替え時に LCD にも信号が送られている事となりますが、LCD クロック(E 端子)を入力していないため、LCD はデータとして受け取りません。

コラム　セットアップ時間とホールド時間

電子回路では、セットアップ時間とホールド時間という概念が良く出てきます。
2 者間でデータ転送を行う際、守らなければならないタイミングの規定です。



この図では、ポジティブエッジトリガ（クロックの立ち上がりのタイミングでデータを取り込む）としています。（クロックの立下りでデータを取り込むネガティブエッジトリガや、両方のエッジでデータを取り込む DDR(Double Data Rate)というシステムもあります。）
※図 4-3 では、E がクロック、DB4-7 がデータですが、E と DB の関係は、E の立下りでデータを取り込むネガティブエッジトリガです

セットアップ時間は、データ信号が変化してから、データを取り込むポイント（上図のストローブポイント）までの時間です。この時間は、規定時間以上のタイミングを確保する必要があります。規定されているセットアップ時間より短い場合（データが変化してから直ぐにクロックが変化する場合）は、正しいデータを取り込めない可能性があります。

ホールド時間は、クロックが変化してから、一定時間データを変化させてはいけない時間です。データが変化して良いタイミングは、クロック変化後・ホールド時間経過後となる必要があります。

セットアップ時間やホールド時間は最低の時間が規定されていますので、その時間以上のタイミングが確保できるよう波形変化のタイミングを設計する必要があります。

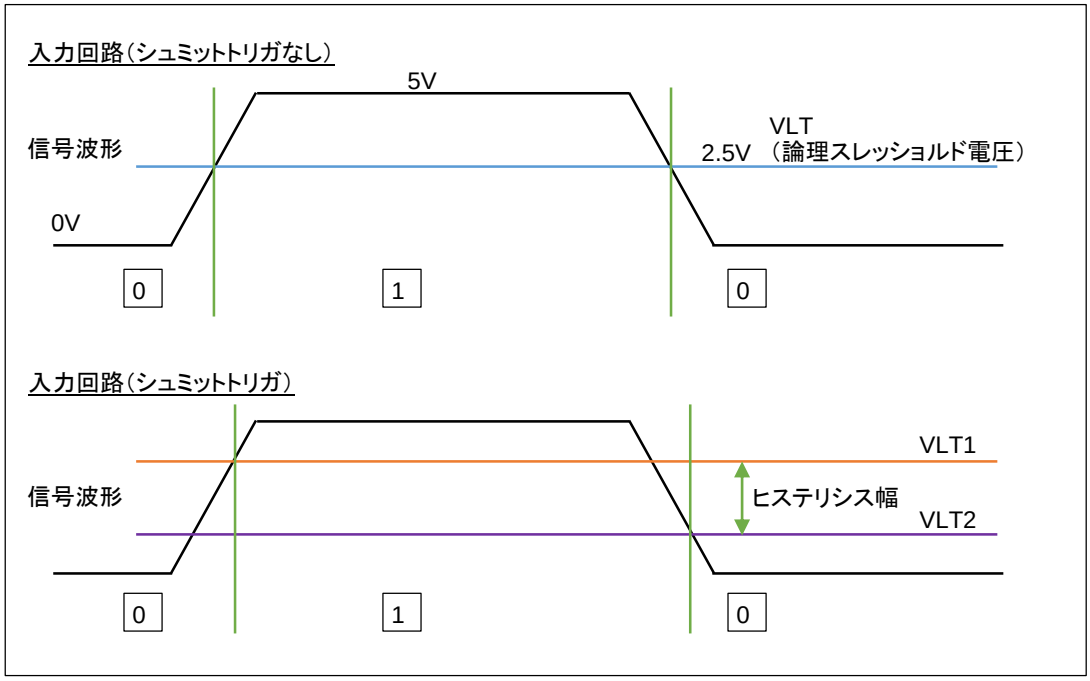
一般的には、1 本のクロック信号で複数のデータ信号を送るというケースが多いですが、データの変化点に時間的なずれが生じる場合でも、全てのデータ信号が規定のセットアップ時間、ホールド時間を守る必要があります。

一般的にはデータの変化点を、ストローブポイントの真ん中とすると、一番マージンが確保できます。（例えばホールド時間を長く取ると（=データの波形変化をクロックの変化から十分に時間が経過したタイミングに設定すると）、次のセットアップ時間が短くなり、結局どちらかにしわ寄せが来ます）

コラム　デジタル信号の L/H

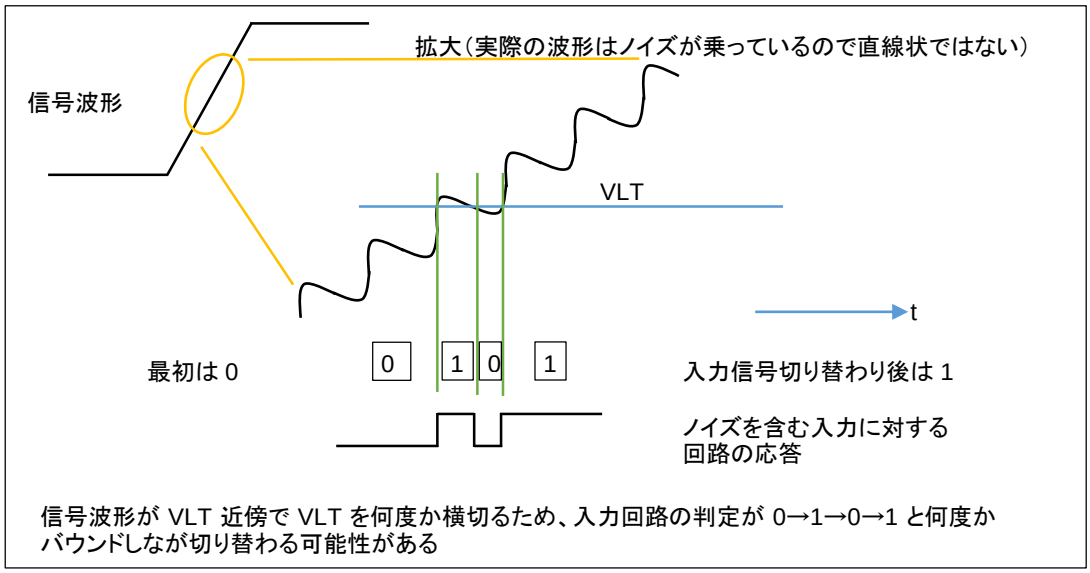
デジタル回路では、L/H のレベルで信号をやり取りしますが、信号を受け取る側の回路は電圧が 0V の時が L レベル。5V の時に、H レベルといった判断をします。

・回路の動作



デジタル回路では、L(0V), H（例えば 5V）の間に、スレッショルド（閾値）があり、閾値より低い場合デジタル的な 0。高い場合、デジタル的な 1 と判断されます。

シュミットトリガといい、閾値を 2 値設け、入力信号が L→H に変化する際は、高い閾値(VLT1)で 0/1 を判定し、入力信号が H→L に変化する際は低い閾値(VLT2)で 0/1 を判定する方式のものもあります。

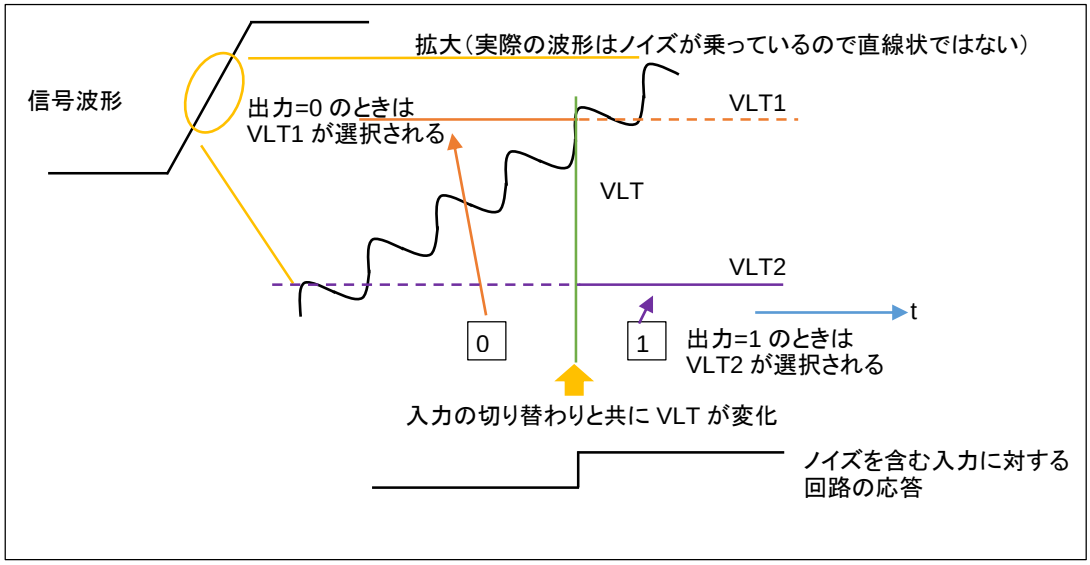


シュミットトリガなしの回路の場合、入力が 0→1 の切り替わりではなく、変化の途中で何度か 0 と 1 の間を行ったり来たりする可能性があります。

コラム　デジタル信号の L/H

入力のバウンドは、データの信号であれば問題ないのですが、クロック信号では問題となります。
※クロック信号と共に使用される、データ信号はあくまでクロックの切り替わりの際にデータがどちらかに確定していれば良いので切り替わりの過程で 0/1 の切り替わりが起こっても（望ましくはないかもしれませんが）問題はありません

シュミットトリガ系の回路を使用した場合は以下となります。



シュミットトリガの回路では、入力が 0→1 と判定された段階で、閾値が下がる（VLT2 に切り替わる）ので、1 に切り替わった後で信号波形が VLT1 を横切っても入力が 0 と判定される事はありません。

そのため、
・信号波形の傾きが鈍っている場合
・ノイズが多い場合
では、シュミットトリガ付きの入力回路が良く使用されます。

RX231 マイコンも、一部の端子を除いて入力回路にはシュミットトリガが採用されています。

コラム　ViH/ViL と VOH/VOL（一見問題なさそうに見えるが…）

今回、マイコンの出力回路とLCD の入力回路を接続して信号のやり取りを行っています。そもそも、マイコンの出力と LCD の入力の電気的特性は整合しているのでしょうか。

データシートを見ると、以下の様になっています。

・LCD 側の入力のスペック

Item	Symbol	min	typ	max	Unit
Input Voltage	V_{IL}	0		0.6	V
	V_{IH}	2.2		VDD	V

これは、0~0.6V までは、入力=0 と見なします。2.2~VDD(=5V)までは、入力=1 と見なします。ということを意味しています。では、0.6~2.2V のときはどうなるのでしょうか？これは、不定（0 か 1 かは保証しないが、0 か 1 のどちらか）となります。

・マイコン側の出力スペック

項目	記号	min	max	単位	測定条件
出力 Low レベル	V_{OL}	-	0.8	V	IOL=2mA, 4.0<=VCC<=5.5V
出力 High レベル	V_{OH}	VCC-0.8	-	V	IOL=-2mA, 4.0<=VCC<=5.5V

ここで、VCC=VDD=5V 時、H レベルは

出力側：4.2V 以上（5-0.8）が保証れており　>　入力側：2.2V 以上であればよい

の関係が満たされています。L レベルは、

出力側：0.8V 以下を保証　<　入力側：0.6V 以下でなければならない
↑条件を満たしていない！

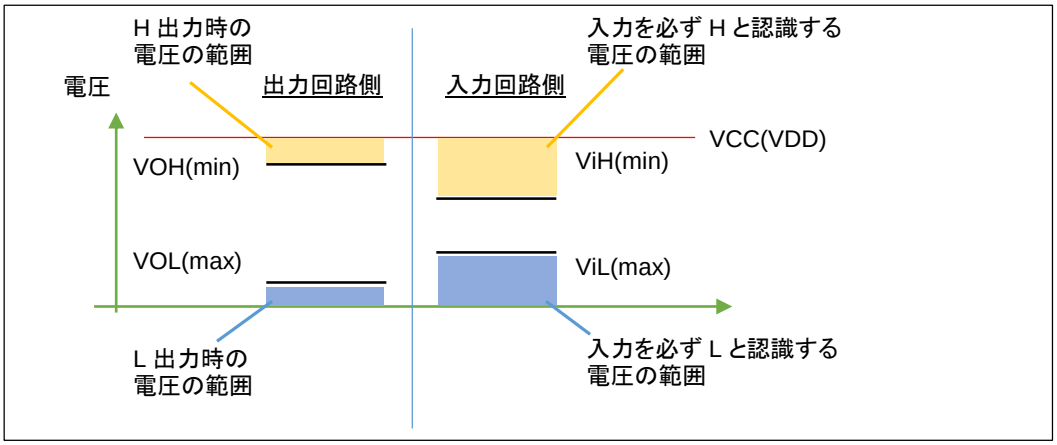
マイコン側は L レベルとして、0.8V 以下を保証しているので、L=0.7V でもスペックの範囲内です。それに対し、LCD 側は、L レベルとして 0.6V 以下を要求（0.7V の時は、H と見なすかもしれないよ）と言っているので、整合が取れていません。

SmartRX!!!学習キットでは、「現実的には問題とならない」と判断して、マイコンの出力と LCD の入力を直接接続していますが、製品設計の世界では、スペックを満たしているかという確認は常に行う必要があるかと考えます。

※本来は「マイコン側出力」と「LCD 入力」の間にレベル変換回路が必要です

コラム　ViH/ViL と VOH/VOL（一見問題なさそうに見えるが…）

ViH, ViL と VOH/VOL は、以下の関係となっている事が必要です。



5. 付録

5.1. SmartRX 学習キット付属 LCD(SC1602)の仕様

＜LCD 資料＞

資料 1　液晶部について　特長

- 5×7 ドットマトリックス+カーソル、16 桁×2 の液晶表示
- 1/16 デューティ
- 192 種のキャラクタジェネレータ ROM
  文字フォント：5×7 ドットマトリクス
- プログラム書込み可能な 8 種のキャラクタジェネレータ RAM
  文字フォント：5×7 ドットマトリクス
- 80×8 ビットの表示データ RAM（最大 80 文字）
- 4ビット及び8ビットの MPU とのインタフェース可能
- 表示データ RAM、キャラクタジェネレータ RAM ともに MPU からの読み出しが可能
- 豊富なインストラクション機能
  表示クリア　他　資料 3　インストラクションについて参照
- 発振回路内蔵
- 5V 単一電源　・　動作温度範囲　0～50℃
- 電源投入時自動リセット回路内蔵
- CMOS プロセス使用

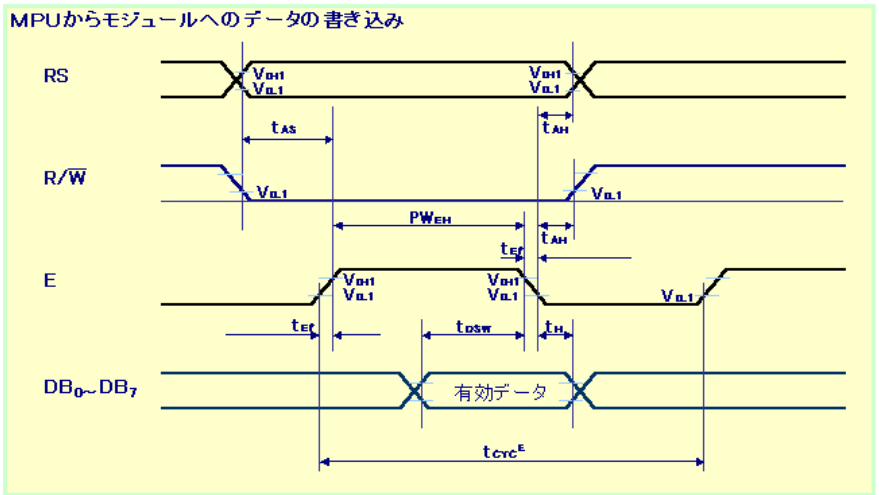
資料 2　タイミング特性について

＜タイミング＞

項目	記号	MIN	MAX
イネーブルサイクル時間	tCYCE	1200	-
イネーブルパルス幅 "High"レベル	PWEH	140	-
イネーブル立上がり・立下り時間	tEr・tEf	-	25
セットアップ時間 RS、R/*W→E	tAS	0	-
アドレスホールド時間	tAH	10	-
データセットアップ時間	tDSW	40	-
データホールド時間	tH	10	-

■書込み動作　単位：ns

VDD=5.0V±10% VSS=0V Ta=0～50



資料 3　インストラクションについて

＜機能コード一覧＞

インストラクション	コード										機能	実行時間(typ)
	RS	R/*W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
表示クリア	0	0	0	0	0	0	0	0	0	1	全表示クリア後、カーソルをホーム位置（0番地）へ戻す	1.52ms
カーソルホーム	0	0	0	0	0	0	0	0	1	*	カーソルをホーム位置へ戻し、シフトしていた表示も元へ戻る（DDRAM の内容は変化無し）	1.52ms
エントリーモード	0	0	0	0	0	0	0	1	I/D	S	カーソルの進む方向、表示をシフトするかどうかの設定（データ書込み及びデータ読み出し時に上記動作が行われます）	37μs
表示 ON/OFF コントロール	0	0	0	0	0	0	1	D	C	B	全表示の ON/OFF[D]、カーソル ON/OFF[C]、カーソル位置の文字のブリンク[B]をセット	37μs
カーソル/表示シフト	0	0	0	0	0	1	S/C	R/L	*	*	DD RAM の内容を変えずカーソルの移動、表示シフト	37μs
ファンクションセット	0	0	0	0	1	DL	N	F	*	*	インタフェースデータ長[DL]、表示行数[N]、文字フォント[F]を設定	37μs
CG RAM アドレスセット	0	0	0	1	ACG						CG RAM のアドレスセット(以後送受するデータは CG RAM データ)	37μs
DD RAM アドレスセット	0	0	1	ADD							DD RAM のアドレスセット(以後送受するデータは DD RAM データ)	37μs
BF/アドレス読出し	0	1	BF	AC							ﾓｼﾞｭｰﾙが内部動作中であることを示す BF 及び AC の内容を読出し(CG RAM/DD RAM 双方可)	37μs
CG RAM/DD RAM データ書込み	1	0	書込みデータ								CG RAM または DD RAM にデータを書込む	37μs tADO=5.6μs
CG RAM/DD RAM データ読出し	1	1	読出しデータ								CG RAM または DD RAM にデータを読出す	37μs tADO=5.6μs

*	:	無効のビット
ACG	:	CGRAM のアドレス
ADD	:	DDRAM のアドレス
AC	:	アドレスカウンタ

	=1	=0
R/L	右シフト	左シフト
S	表示をシフトさせる	表示をシフトしない
N	2 行表示	1 行表示
F	5×10 ドットマトリックス	5×7 ドットマトリックス
BF	内部動作中	インストラクション受付可
S/C	表示のシフト	カーソル移動

	=1	=0
I/D	インクリメント	デクリメント
DL	8ビット	4ビット
D	表示ON	表示OFF
C	カーソルON	カーソルOFF
B	ブリンクON	ブリンクOFF

■クロック発振周波数（fOSK）が変化すると実行時間も変化します
例　fOSK＝190kHz の場合　37μs×270/190＝53μs

■tADO 時間はクロック発振周波数（fOSK）によって変化します
tADO＝1.5/（fOSK）（s）

資料 4　文字コードと文字パターンについて

文字コードと文字パターンは下記例の通りの関係となっております　（対応一覧は次の資料 5 文字コード一覧をご覧下さい）

＜CG RAM アドレスと文字コード・文字パターン＞

- CGRAM データは"1"が表示上の選択、"0"が非選択に対応します
- 文字コードビット 0-2 と CGRAM アドレスビット 3-5 が対応します（3ビット8種）
- CGRAMアドレスビット 0-2 が文字パターンの行位置を指定します
- 文字パターンの8行目はカーソル位置で、カーソルとCGRAMデータの論理和をとって表示されますので、カーソル表示を行う際は8行目のCGRAMデータを0にして下さい
- 8行目のデータを1にするとカーソルの有無に関係なく1ビットが点灯します
- 文字パターンの列位置はCGRAMデータビット 0-4 に対応し、ビット4が左端になります
- CGRAMデータビット 5-7 は表示されませんが、メモリは存在しているので、一般のデータRAMとして使用できます
- CGRAM の文字パターンを読み出すときは文字コードの 4-7 ビットは全て"0"を選択します
- どのパターンを読み出すかは 0-2 のビットで決定しますが、ビット3は無効なので"00H"と"08H"では同じ文字が選択されます

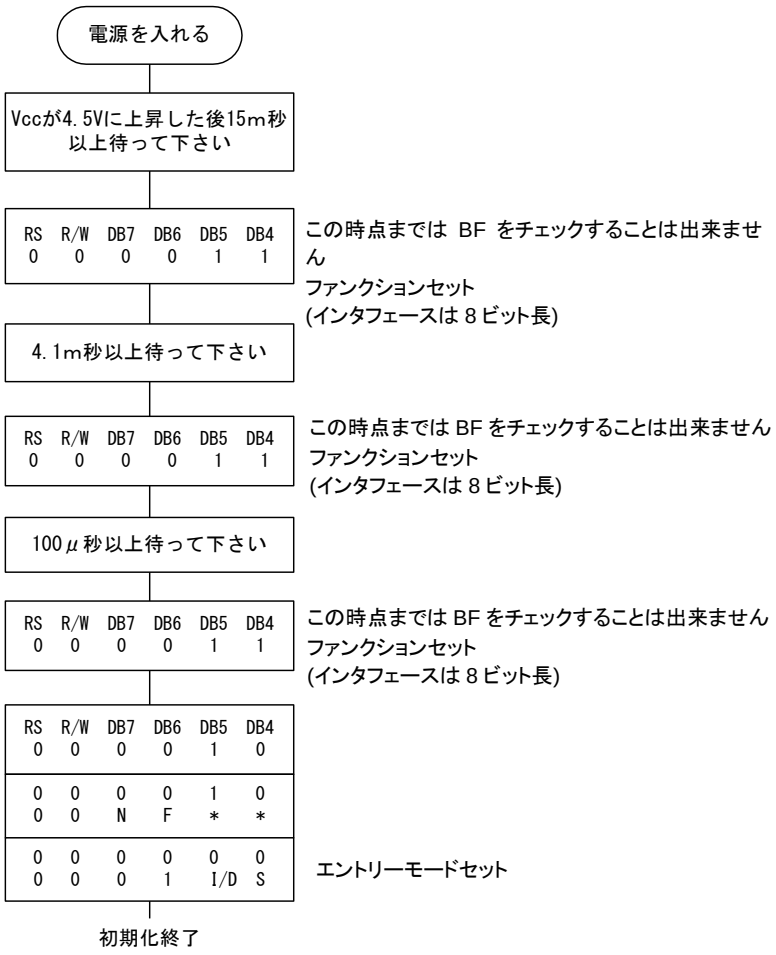
文字コード(DDRAMデータ) 上位ビット 7 6 5 4	文字コード(DDRAMデータ) 下位ビット 3 2 1 0	CG RAMアドレス 上位ビット 5 4	CG RAMアドレス 下位ビット 3 2 1	文字パターン(CGRAMデータ) 上位ビット 7 6 5	文字パターン(CGRAMデータ) 下位ビット 4 3 2 1 0	
0 0 0 0	0 0 0	0 0	0 0 0	* * *	1 1 1 1 0	文字パターン例「R」
			0 0 1	* * *	1 0 0 0 1	
			0 1 0	* * *	1 0 0 0 1	
			0 1 1	* * *	1 1 1 1 0	
			1 0 0	* * *	1 0 1 0 0	
			1 0 1	* * *	1 0 0 1 0	
			1 1 0	* * *	1 0 0 0 1	
			1 1 1	* * *	0 0 0 0 0	←カーソル位置
0 0 0 0	0 0 1	0 0	0 0 0	* * *	1 0 0 0 1	文字パターン例「¥」
			0 0 1	* * *	0 1 0 1 0	
			0 1 0	* * *	1 1 1 1 1	
			0 1 1	* * *	0 0 1 0 0	
			1 0 0	* * *	1 1 1 1 1	
			1 0 1	* * *	0 0 1 0 0	
			1 1 0	* * *	0 0 1 0 0	
			1 1 1	* * *	0 0 0 0 0	←カーソル位置
			0 0 0	* * *		
0 0 0 0	1 1 1	1 1	0 1 1	* * *		
			1 0 0	* * *		
			1 0 1	* * *		
			1 1 0	* * *		
			1 1 1	* * *		←カーソル位置

資料 5　文字コード･文字パターン対応一覧

＜文字コードと文字パターン対応表 ＞

上位4ビット 下位4ビット		0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx	0000	CG RAM (1)		0	@	P	`	p		−	タ	ミ	α	p
xxxx	0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx	0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	ß	θ
xxxx	0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx	0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx	0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx	0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx	0111	(8)		7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx	1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx	1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	$^{-1}$	y
xxxx	1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx	1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	^{x}	万
xxxx	1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx	1101	(6)	−	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx	1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx	1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	■

資料 6　LCD 初期化フロー



取扱説明書改定記録

バージョン	発行日	ページ	改定内容
REV.1.0.0.0	2018.9.27	—	初版発行
REV.1.0.1.0	2022.5.10	P35	誤記訂正

お問合せ窓口

最新情報については弊社ホームページをご活用ください。

ご不明点は弊社サポート窓口までお問合せください。

株式会社 北斗電子

〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

TEL 011-640-8800　FAX 011-640-8801

e-mail：support@hokutodenshi.co.jp (サポート用)、order@hokutodenshi.co.jp (ご注文用)

URL:http://www.hokutodenshi.co.jp

ルネサス エレクトロニクス RX マイコン搭載
HSB シリーズマイコンボード 評価キット

SmartRX 学習キット
チュートリアル 1

株式会社 北斗電子

 2022 年 5 月 10 日改訂 REV.1.0.1.0 (220510)