



# HSBRX65-I0-BOARD

## サンプルソフトウェア マニュアル(2)

---

ルネサス エレクトロニクス社 RX651/RX671(QFP-144 ピン)搭載 HSB シリーズマイコンボード向け I/O ボード

-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.1.0.0.0

注意事項 .....	1
安全上のご注意 .....	2
概要 .....	4
ファイルの構成 .....	4
ソースコードの構造 .....	5
スマート・コンフィグレータの設定 .....	7
サンプルプログラムの動作説明(main_etc.c) .....	8
サンプルプログラムの動作説明(main_timer.c) .....	15
1. サンプルプログラムに含まれる関数の説明 .....	18
1.1. io_board.c .....	18
1.1.1. プッシュスイッチ(SW0-SW7) .....	18
1.1.2. 割り込み対応プッシュスイッチ(SW8-SW10) .....	19
1.1.3. LED(LED0-LED7) .....	20
1.1.4. ブザー(B1) .....	20
1.1.5. タイマ連動 LED(LED8) .....	21
1.1.6. ステッピングモータ(J5) .....	23
1.1.7. A/D 変換(R29) .....	25
1.1.8. 7セグメント LED(SEG1) .....	26
1.1.9. マトリックススイッチ(SW11~SW26) .....	29
1.2. lcd_1602.c .....	31
1.3. sci.c .....	35
2. アプリケーションプログラムの作成方法 .....	41
2.1. io_board.cに含まれる関数の使用方法 .....	45
2.1.1. プッシュスイッチ(SW0-SW7) .....	45
2.1.2. 割り込み対応プッシュスイッチ(SW8-SW10) .....	45
2.1.3. LED(LED0-LED7) .....	46
2.1.4. ブザー(B1) .....	46
2.1.5. タイマ連動 LED(LED8) .....	47
2.1.6. ステッピングモータ(J5) .....	48
2.1.7. A/D 変換(R29) .....	49
2.1.8. 7セグメント LED(SEG1) .....	49
2.1.9. マトリックススイッチ(SW11~SW26) .....	51
2.2. lcd_1602.cに含まれる関数の使用方法 .....	52
2.3. sci.cに含まれる関数の使用方法 .....	53
3. サンプルプログラムで定義されている定数など .....	56



3.1. io_board.h.....	56
3.2. sci.h .....	56
<b>4. マイコン使用機能.....</b>	<b>57</b>
4.1. 使用機能一覧.....	57
4.2. I/O ボードの各機構に割り当てられているマイコン機能.....	57
4.3. 使用割り込み一覧.....	58
<b>付録.....</b>	<b>59</b>
取扱説明書改定記録 .....	59
お問合せ窓口.....	59



## 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



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



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

	<b>一般指示</b> 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		<b>一般禁止</b> 一般的な禁止事項を示します
	<b>電源プラグを抜く</b> 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		<b>一般注意</b> 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があります。

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・異臭にお気づ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、長時間直射日光が当たる場所、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。

## 概要

HSBRX65-IO-BOARD は、主にプログラム学習向けのキットとなりますので、ユーザがプログラムを作成するのが主目的ですが、回路図やハードの仕様を見ながらゼロからプログラムを作成するのは、意外と手間が掛かるものです。

とりあえず、ボード搭載機能を組み合わせて何かアプリケーションプログラムを作ってみたいという場合に使用可能な、ベースとなるプロジェクトを用意しましたので、その内容を説明する資料となります。

ボード搭載の機能は関数化されていますので、まずはこちらで用意した関数を呼び出す事で動作見てみる。その上で、関数の内容を変更したり、拡充したりして、マイコンのプログラミングを学習する事を想定しています。

## ファイルの構成

・RX651 向け

SAMPLE2\_HSBXR65\_IO\_BOARD.zip

SAMPLE2\_HSBXR65\_IO\_BOARD¥csplus¥HSBRX65-IO-BOARD\_SAMPLE2.zip CS+プロジェクト  
¥e2studio¥HSBRX65\_IO\_BOARD\_SAMPLE2.zip e2studio アーカイブ

・RX671 向け

SAMPLE2\_HSBXR65\_IO\_BOARD\_RX671.zip

SAMPLE2\_HSBXR65\_IO\_BOARD\_RX671¥csplus¥HSBRX65-IO-BOARD\_SAMPLE2\_RX671.zip CS+プロジェクト  
¥e2studio¥HSBRX65\_IO\_BOARD\_SAMPLE2\_RX671.zip e2studio アーカイブ

Web よりダウンロードできるファイルは、RX651 向けと RX671 向けの 2 種類となっています。

それぞれの zip ファイルの中に、CS+向けと e2studio 向けが格納されていますので、お使いの開発環境に合わせて使用してください。

CS+プロジェクトの方は、zip ファイルを展開して、展開先に含まれる、プロジェクト名.mtpj ファイルをダブルクリックして、CS+を起動してください、

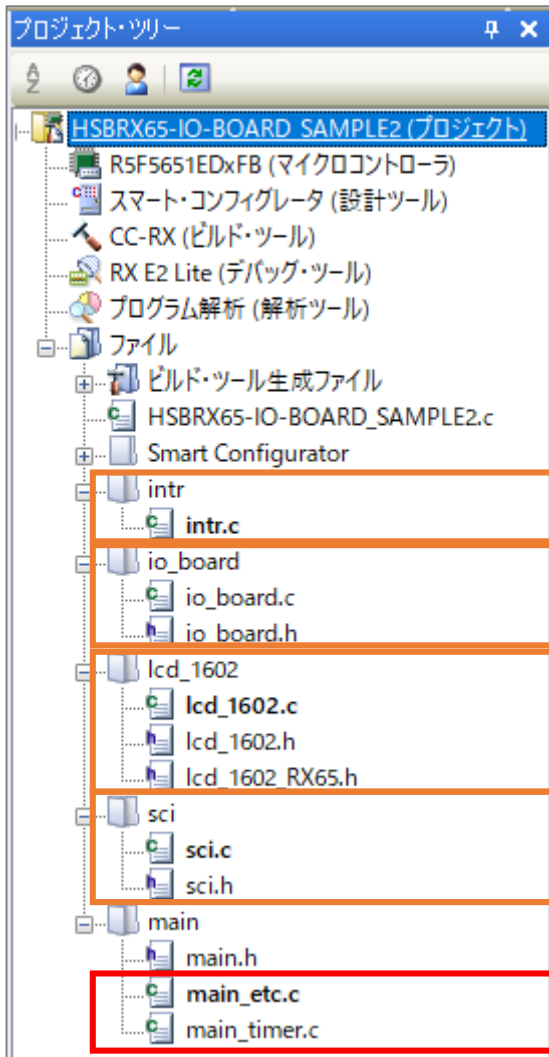
e2studio の方は、

ファイル→インポート→既存プロジェクトをワークスペースへ  
アーカイブファイルの選択

で、e2studio のアーカイブ zip ファイルを指定してください。



## ソースコードの構造



intr	割り込み処理をまとめたソース
io_board	HSBRX65-IO-BOARD 搭載機能をまとめたソース
lcd_1602	キャラクタ LCD 関数
sci	UART(SCI)関数
main	アプリケーションプログラムの例

カテゴリ	説明	備考
Smart Configurator	スマート・コンフィグレータ生成ファイル	
intr	割り込み関数をまとめたフォルダ	
io_board	HSBRX65-IO-BOARD 搭載機能を関数化	
lcd_1602	キャラクタ LCD	
sci	UART(SCI)通信	
main	メイン関数(アプリケーションプログラム例)	

サンプルプログラムは、ソースコードの形で提供されていますので、自由に書き換えが可能です。

ユーザアプリケーション作成時は、main 以下にメインの処理を記載したソースコードを追加する予定です。  
(main\_etc.c, main\_timer.c は参考のために用意しているサンプルですので、ユーザアプリケーション作成時は削除頂いて構いません。)

ユーザ作成のアプリケーションプログラム本体を、main 以下に配置して、

io\_board.c

lcd\_1602.c

sci.c

内の関数を呼び出す形で、アプリケーションプログラムの作成を行います。

予め用意されている関数で機能が不足しているものや、挙動を変更したい場合は、上記フォルダ内のソースコードを変更してビルドしてください。

## スマート・コンフィグレータの設定

本サンプルプログラムでは、スマート・コンフィグレータで、以下のコンポーネントを使用しています。

▼ 現在の設定状態

使用しているボード/デバイス: R5F5651EDxFB (ROM size: 2MB, RAM size: 640KB, Pin count: 144)

生成先ロケーション (PROJECT\_LOC#):

使用しているコンポーネント:

コンポーネント	バージョン	設定
✔ 8ビットタイマ	1.10.0	Config_TMR1(TMR1: 使用中)
✔ Board Support Packages. (r_bsp)	7.50	r_bsp(使用中)
✔ PWMモードタイマ	1.12.0	Config_TPU2(TPU2: 使用中)
✔ SCI(SCIF)調歩同期式モード	1.12.0	Config_SCI5(SCI5: 使用中)
✔ コンペアマッチタイマ	2.3.0	Config_CMT3(CMT3: 使用中)
✔ シングルスキャンモードS12AD	2.5.0	Config_S12AD0(S12AD0: 使用中)
✔ リアルタイムクロック	1.8.0	Config_RTC(RTC: 使用中)
✔ 割り込みコントローラ	2.3.0	Config_ICU(ICU: 使用中)

	機能	用途	備考
TMR1	タイマ	ブザーの駆動	P17 でブザーを鳴らす設定 (JP2 は P17 側を選択)
TPU2	タイマ	LED8 の駆動 TPU2 のインターバルタイマ	P15 で LED8 を駆動する設定 (JP1 は P15 側を選択)
SCI5	UART(SCI)	通信	J7(USB-miniB)で PC と通信
CMT3	タイマ	A/D 変換の起動 7 セグメント LED の駆動 マトリックススイッチの読み取り	ボード搭載機能の定期処理 (1ms のインターバルタイマ)
S12AD0	A/D 変換	ボリューム(R29)の A/D 変換	
RTC	リアルタイム クロック	時計・カレンダー機能の使用 1 秒のインターバルタイマ	
ICU	割り込み	SW8, SW9, SW10 の検出	IRQ4, IRQ13, IRQ15

サンプルプログラムで設定済みなのは、上記コンポーネントです。上記以外は、ユーザ側で追加して使用可能です。  
(例えば、コンペアマッチタイマ CMT0~CMT2 などは、ユーザ側で使用可能です。)

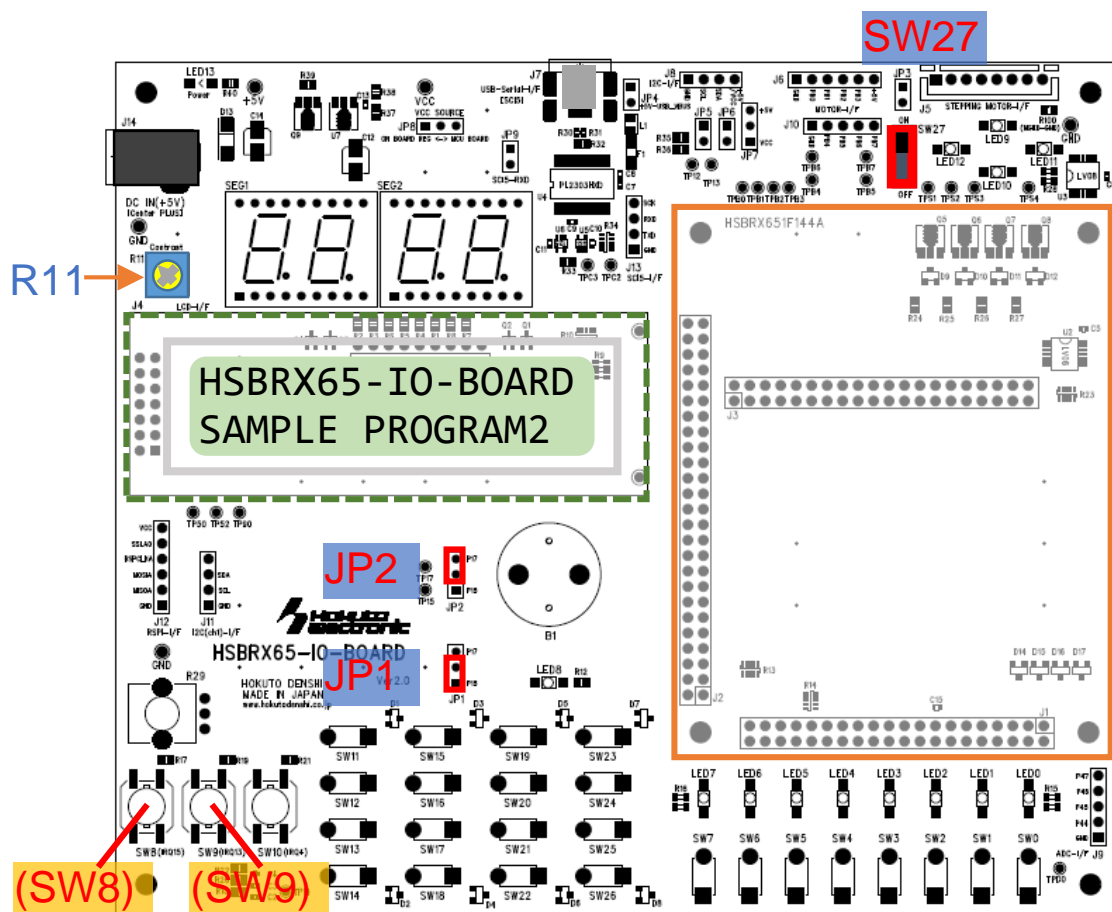
## サンプルプログラムの動作説明(main\_etc.c)

ボード搭載機能を一通り紹介するデモプログラムです。

プログラムの書き込みを行い、電源を投入すると、LCD 画面に

**HSBRX65 - IO - BOARD  
SAMPLE PROGRAM2**

と、約 3 秒間表示されます。表示が見難い場合は、R11 を調整して、適切なコントラストに設定してください。



SW27 は ON(上側)、JP1 は下側、JP2 は上側に設定してください。

SW8 を押すと、次の項目

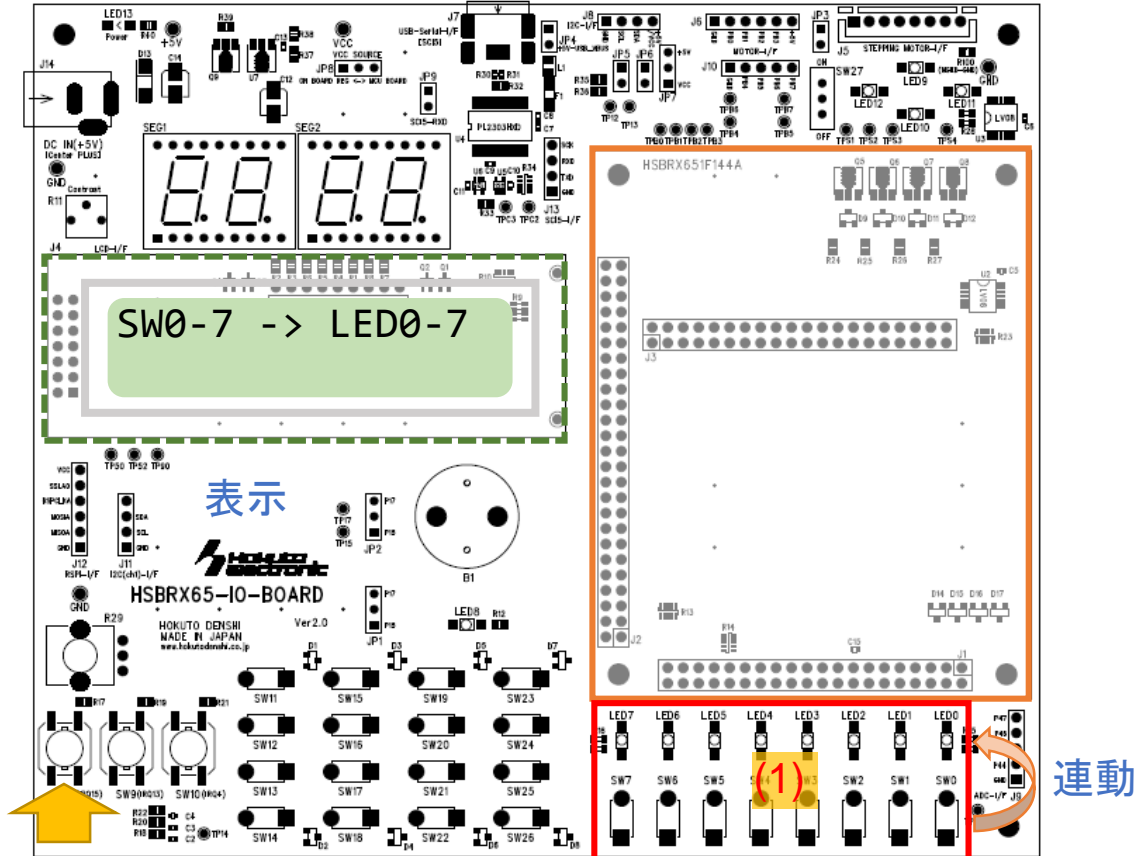
SW9 を押すと、前の項目

となります。

(1)SW-LED 連動

ボード上のプッシュスイッチ SW0-7 を押すと、押している間、LED0-7 が点灯します。

(これは、SW0-7 と LED0-7 がボード上で接続されているわけではなく、マイコンボードで実行されているプログラムが、SW の読み取りと、LED の制御を行っています。)

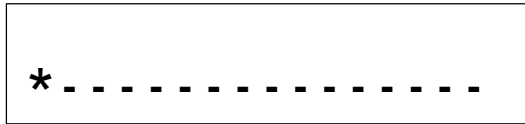


SW8 を押すと、次の動作項目へ移動します。

## (2)マトリクススイッチの読み取り

マトリクススイッチ SW11-26 を押すと、LCD 画面に押ししているスイッチの状態が表示されます。(複数押しも認識します。)

LCD 2 行目の表示

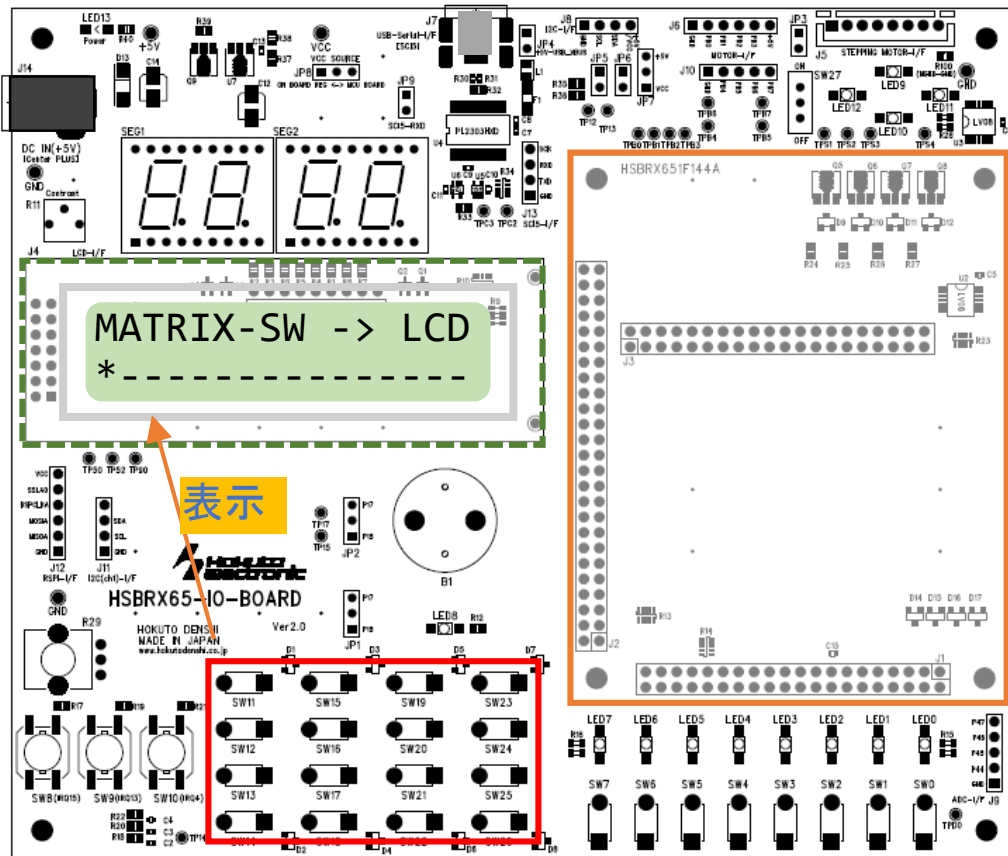


SW11

SW26

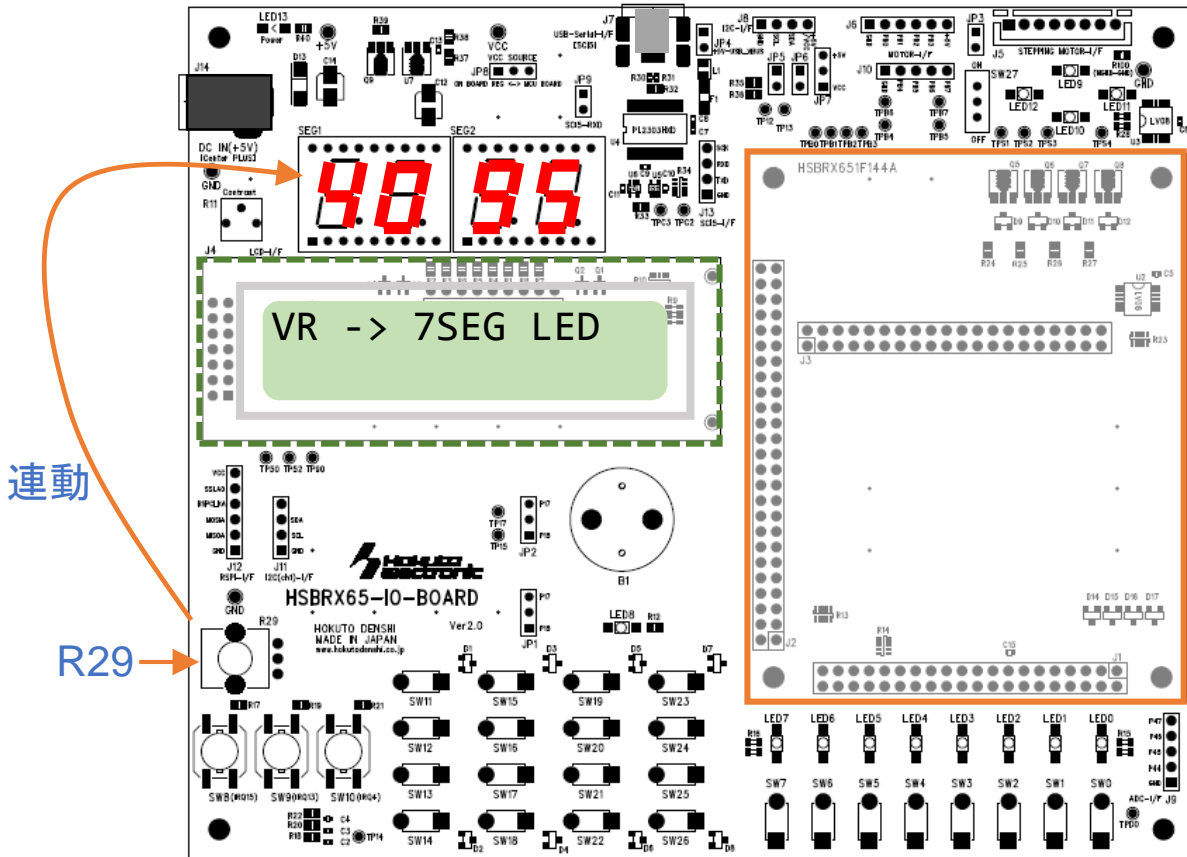
-: キーを押していない

\*: キーを押している



(3)7セグ LED, VR

R29(VR, 可変抵抗)を回すと、回転角度に応じた数値が、7セグメントLEDに表示されます。(0~4095の値)

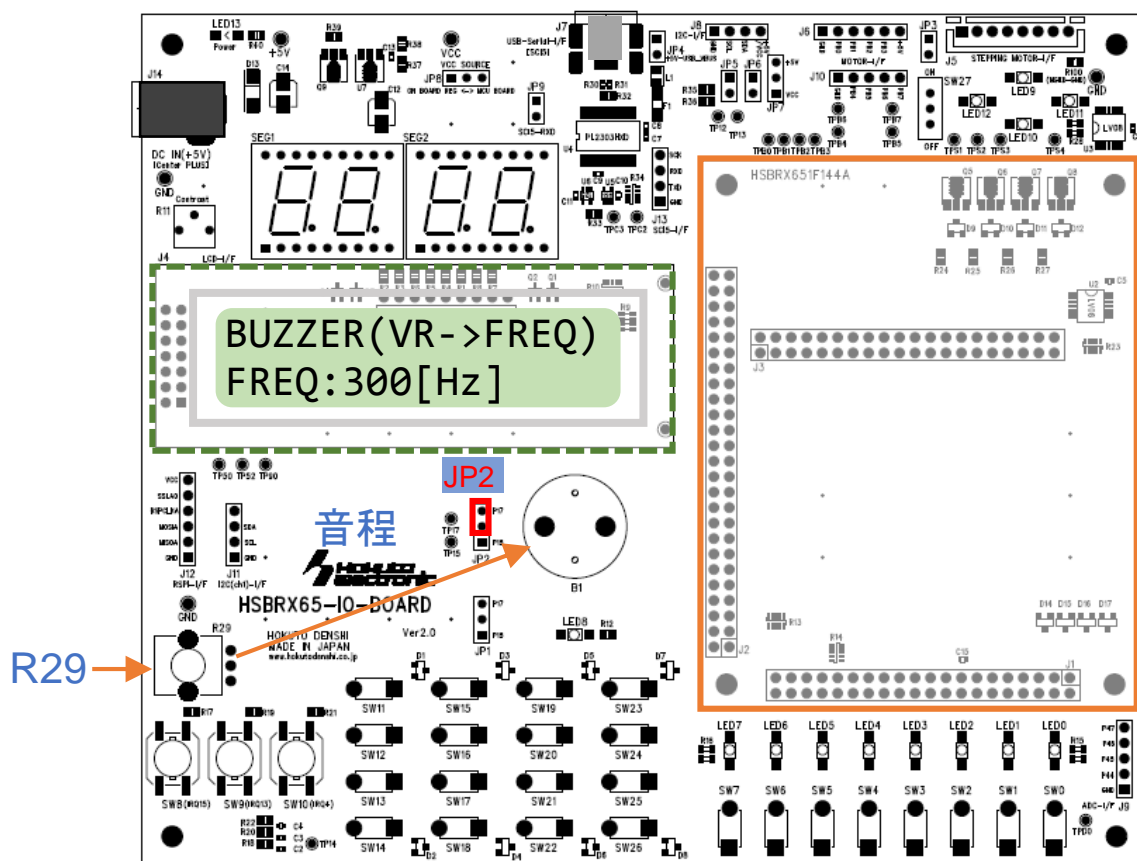


#### (4)ブザー鳴動

ブザーが鳴ります。

R29 を回すと、約 300Hz から 8000Hz の間で音の高さが変わります。

ブザーは、TMR1 タイマ(8 ビットタイマ)を使用して、大体デューティ比が 50 となる矩形波でブザーを駆動しています。



※TMR1 タイマは、PCLKB(60MHz)の 1/1024 分周で駆動しており、

タイマの 1 周期 17.07us

矩形波の周期の最小(2 サイクル) 34.13us, 29.3kHz

矩形波の周期の最大(256 サイクル) 4.37ms, 229Hz

となります。プログラム上では、ブザーを鳴らす関数は、周波数を指定する関数となっており、300~8000(Hz)の引数の指定が可能です。(関数仕様は後述)



### (5)LED 点滅

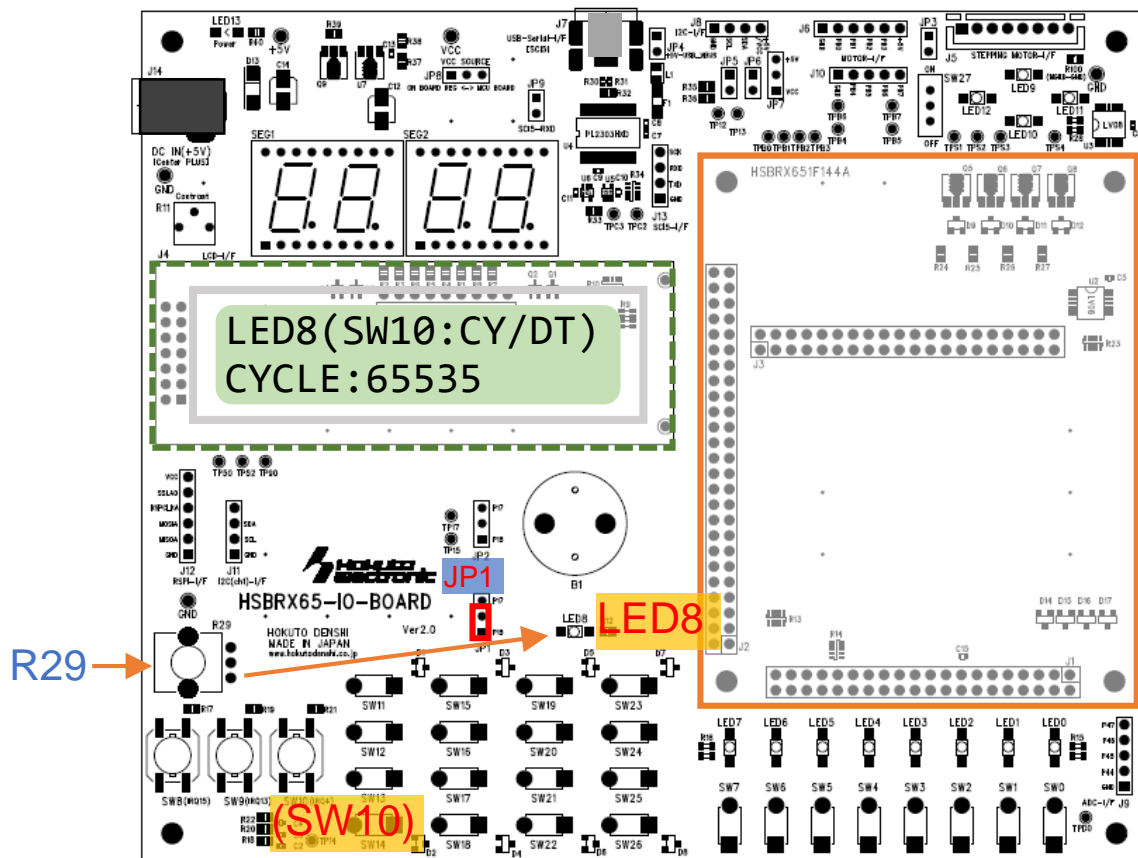
LED8 が点滅します。R29 で

- ・点滅の周期
- ・点滅のデューティ比(点灯と消灯の割合)

を変更可能です。周期とデューティ比の切り替えは、SW10 で行います。

SW10 を押すと LCD の 2 行目が「DUTY:60.1」の様な表示となり、R29 を回すとデューティ比が変わります。

LED8 の点滅は、TPU2 タイマ(16 ビットタイマ)を使っています。



※TPU2 タイマは、PCLKB(60MHz)の 1/1024 分周で駆動しており、

タイマの 1 周期 17.07us

ON/OFF の周期の最小(2 サイクル) 34.13us, 29.3kHz

ON/OFF の周期の最大(65536 サイクル) 1.12s, 0.89Hz

(最大で 1 秒強周期の点滅となります。)

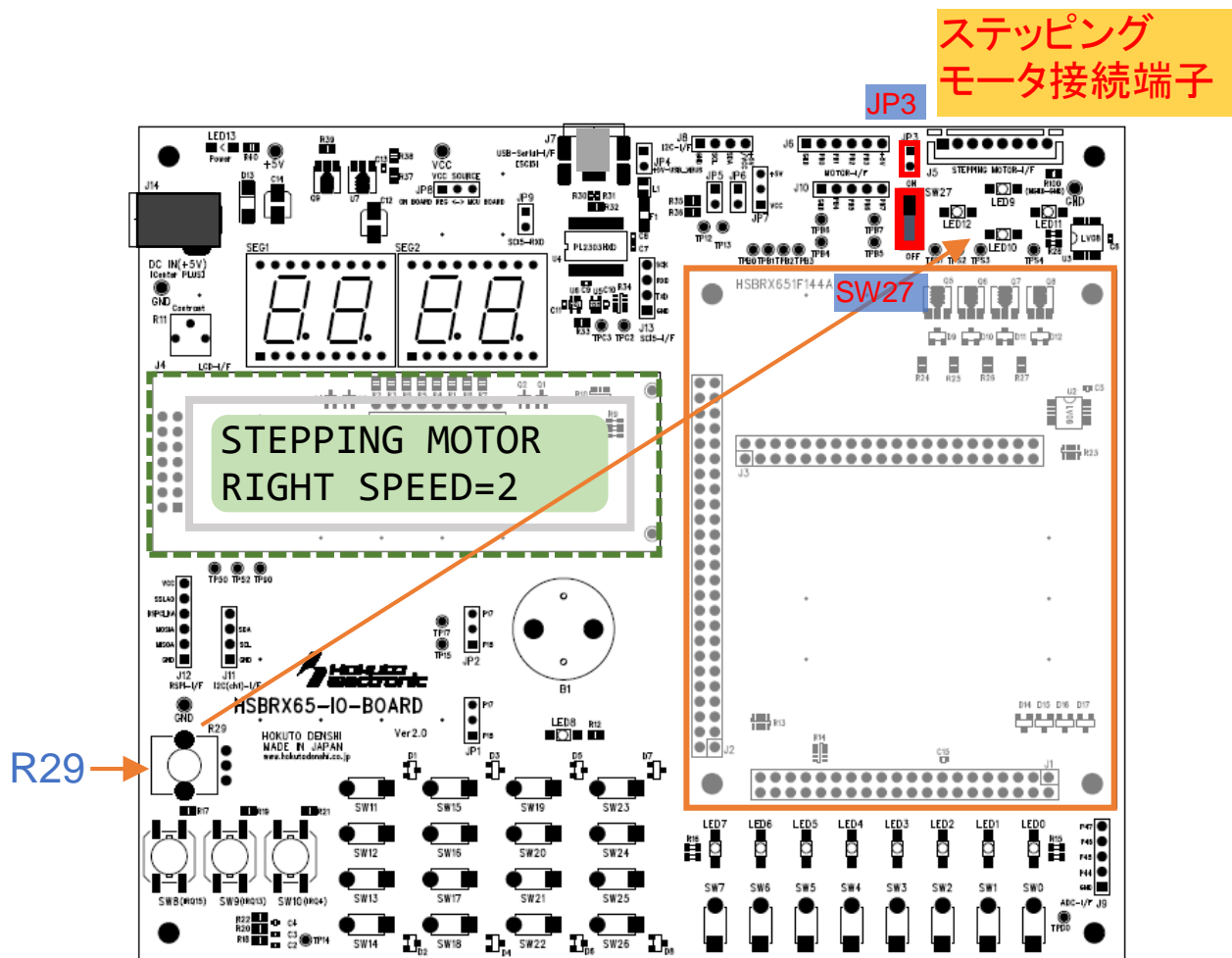
### (6)ステッピングモータ駆動

※モータをつなげなくても LED の点灯パターンで動作イメージは掴めます。

R29 を回すと、

左回転←停止→右回転

の様に、回転方向、回転速度を変えられます。

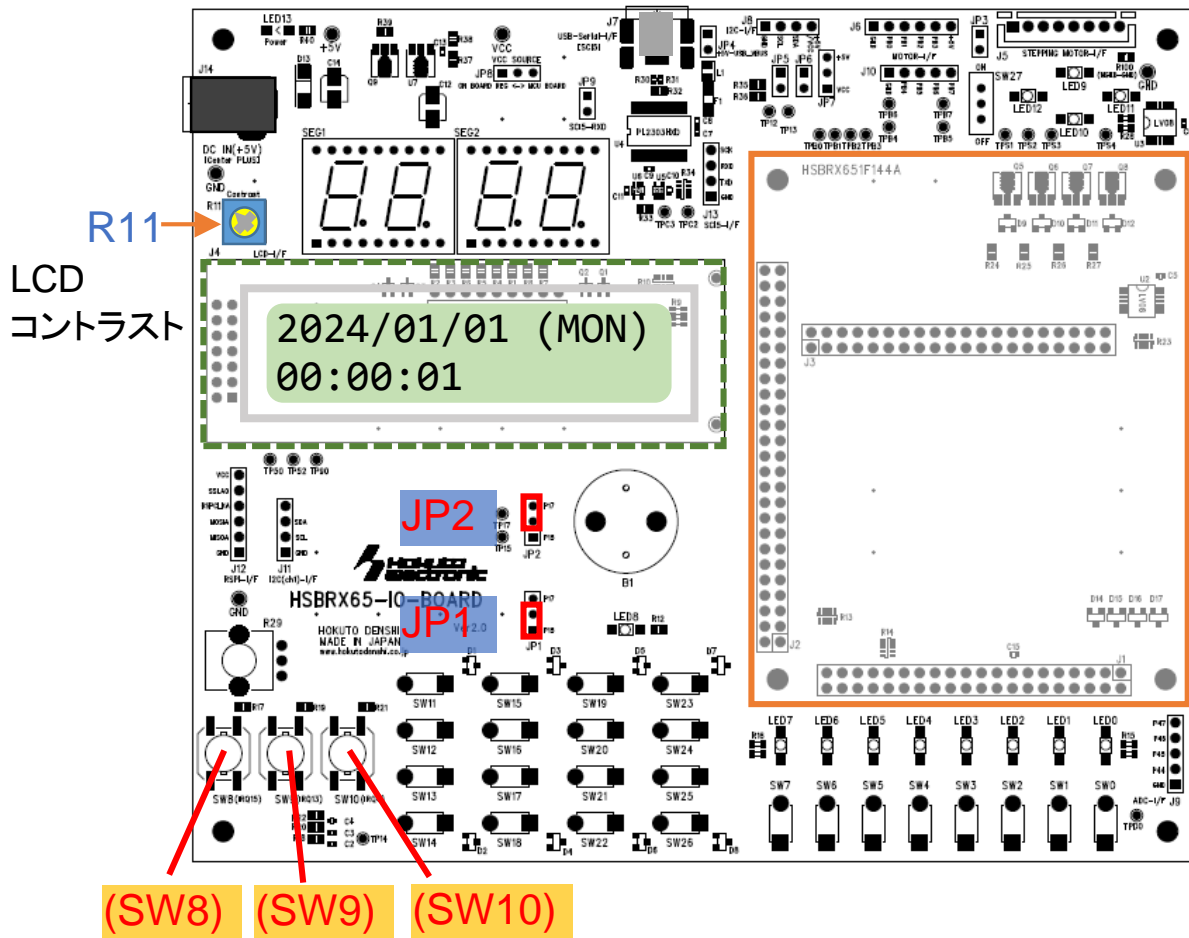


※ステッピングモータの回転パターンを変えるタイミングは TPU2 タイマを使用しています。

## サンプルプログラムの動作説明(main\_timer.c)

RTC(リアルタイムクロック)の機能を使用して、時計を表示させるのと、アラーム機能(指定した秒数後にアラームを鳴らす)のサンプルプログラムです。

起動時に、SW8を押したまま電源を投入。または、SW8を押したままマイコンボードのリセットスイッチを押してください。(起動には、3秒程度掛かります。SW8はLCDに時計が表示されるまで押したままとしてください。)

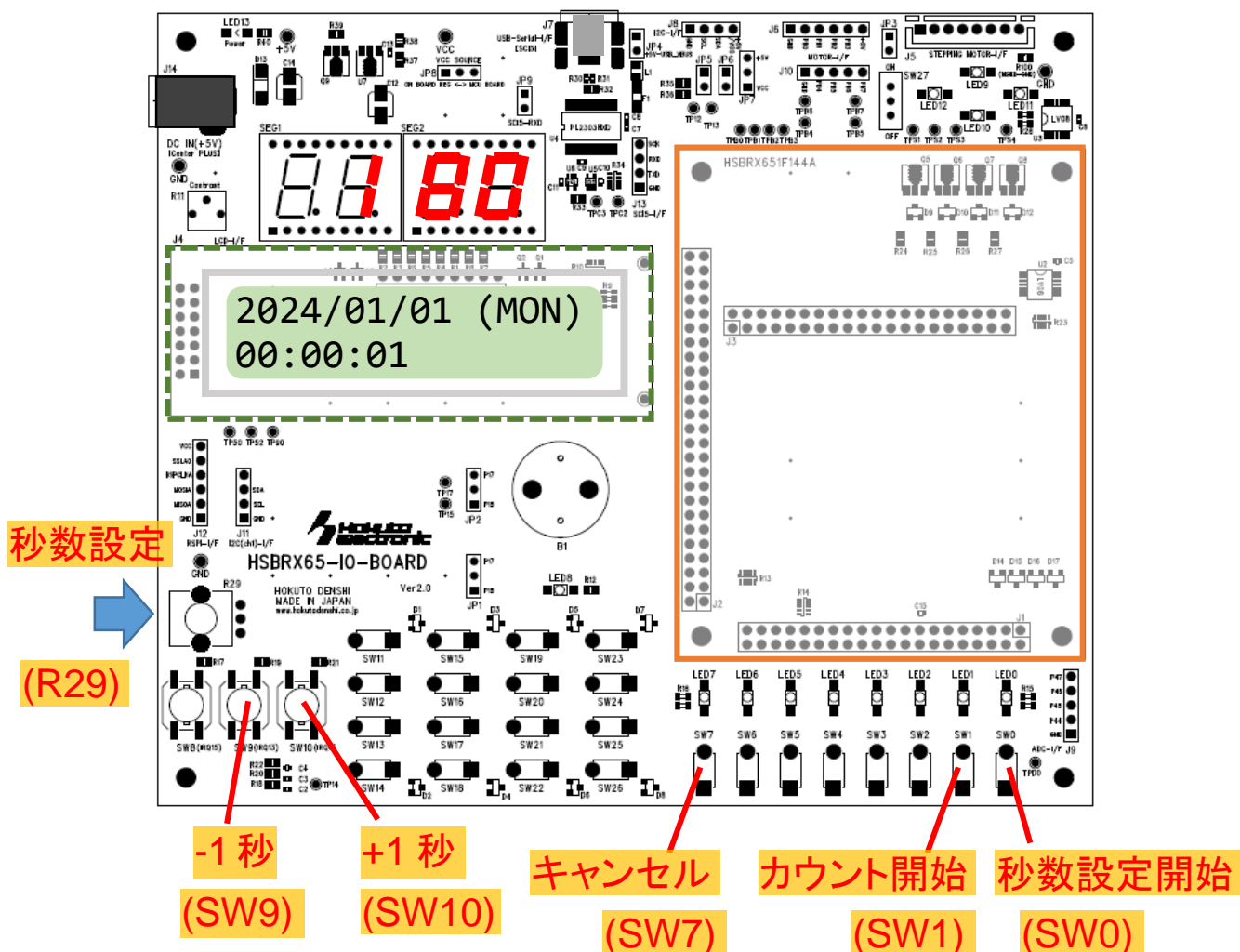


JP2は上側に設定してください。(JP1は未使用ですが、下側ショートで問題ありません。)

### ○時計の時間変更

- ・SW8 時間の設定  
年→月→日→時間→分→秒→設定終了(時計動作開始)
- ・SW9 値を小さい方向に動かす  
年設定時、2024→2023 TP→2022 の様に値が変わります。
- ・SW10 値を大きい方向に動かす  
年設定時、2024→2025→2026 の様に値が変わります。

### ○タイマ(アラーム)機能



SW0 を押すと、タイマ時間の設定となり、R29 で時間(1~1000 秒)の設定ができます。タイマ時間の設定値は、7セグメント LED に表示されます。

タイマ時間は、SW9, SW10 で微調整が可能です。

時間の設定後、SW1 を押すとカウントダウンが始まります。時間が 0 になると、アラームが鳴ります。アラームは、SW0~SW7 のいずれかを押す事で止まります。

アラームは、鳴り始めてから 30 秒経過すると、連続音に変わります。

タイマ時間設定中、カウントダウン中、アラーム鳴動後、いずれのタイミングでも、SW7 を押すことでタイマ機能を停止することができます。

## 1. サンプルプログラムに含まれる関数の説明

### 1.1. io\_board.c

io\_board.c 内には、7 セグメント LED やマトリックススイッチなどのボード搭載機能を制御する関数が含まれます。  
(キャラクタ LCD の制御と、UART(SCI)通信は別ソースになっています。)

#### 1.1.1. プッシュスイッチ(SW0-SW7)

##### push\_sw\_init

概要: プッシュスイッチ初期化関数

宣言:

```
int push_sw_init(void);
```

説明:

SW0-SW7 が接続されているポートを入力端子に設定します

引数:

なし

戻り値:

正常終了(0)

補足:

マイコンのリセット後は全ポート入力端子に設定されますので、明示的に本関数を実行しなくても問題ありません

##### push\_sw\_read

概要: SW0-7 読み取り関数

宣言:

```
unsigned char push_sw_read(void);
```

説明:

SW0-SW7 を読み取ります

引数:

なし

戻り値:

bit7, bit6 ... bit0 = SW7, SW6, ...SW0 の状態

スイッチが押されている場合は 0b0

補足:

SW3, SW1 が押されている場合は、戻り値は 0xF5(bit3=bit1=0b0, 他は 0b1)

## 1.1.2. 割り込み対応プッシュスイッチ(SW8-SW10)

### intr\_push\_sw\_init

概要: SW8-SW10 設定関数

宣言:

```
int intr_push_sw_init(int timing);
```

説明:

SW8-SW10 の応答するタイミングを設定します

引数:

timing: 0 ボタンを押したときに割り込みが掛かる[起動時のデフォルト]

1 ボタンを離したときに割り込みが掛かる

2 押した場合と離した場合の両方割り込みが掛かる

戻り値:

正常終了(0)

引数エラー(-1)

補足:

本関数を実行しない場合は、デフォルト値: ボタンを押したときに割り込みが掛かる、となります

### intr\_push\_sw\_start

概要: SW8-SW10 動作開始関数

宣言:

```
int intr_push_sw_start(void);
```

説明:

SW8-SW10 の割り込みを開始します

引数:

なし

戻り値:

正常終了(0)

### intr\_push\_sw\_stop

概要: SW8-SW10 動作停止関数

宣言:

```
int intr_push_sw_stop(void);
```

説明:

SW8-SW10 の割り込みを停止します

引数:

なし

戻り値:

正常終了(0)

### 1.1.3. LED(LED0-LED7)

#### led\_init

概要: LED0-LED7 初期化関数

宣言:

```
int led_init(void);
```

説明:

LED0-LED7 の初期化(出力ポート設定と全消灯)を行います

引数:

なし

戻り値:

正常終了(0)

#### led\_set

概要: LED0-LED7 設定関数

宣言:

```
int led_set(unsigned char led);
```

説明:

LED0-LED7 の点灯状態の制御を行います

引数:

led: LED の点灯状態, bit7, ...,bit0 = LED7, ...LED0

0b0 で消灯, 0b1 で点灯

戻り値:

正常終了(0)

### 1.1.4. ブザー(B1)

#### buzzer\_freq\_set

概要: ブザー周波数設定関数

宣言:

```
int buzzer_freq_set(unsigned short freq);
```

説明:

ブザーの周波数設定を行います

引数:

freq: [Hz]単位の周波数、300~8000 の範囲の値が有効

戻り値:

正常終了(0)

補足:

300 未満の値は 300、8000 以上の値は 8000 に設定されます



### buzzer\_on

概要: ブザー鳴動関数

宣言:

```
int buzzer_on(void);
```

説明:

ブザーを鳴らします

引数:

なし

戻り値:

正常終了(0)

### buzzer\_off

概要: ブザー停止関数

宣言:

```
int buzzer_off(void);
```

説明:

ブザーを止めます

引数:

なし

戻り値:

正常終了(0)

※本関数使用時は、JP2 は P17 側(上側をショート)としてください

## 1.1.5. タイマ連動 LED(LED8)

### led8\_cycle\_set

概要: LED8 周期設定関数

宣言:

```
int led8_cycle_set(unsigned short cycle);
```

説明:

LED8 の点滅周期の設定を行います

引数:

cycle: LED8 の点滅の周期のサイクル数(1-65535)の値

戻り値:

正常終了(0)

説明:

1 サイクルは、 $1/60\text{MHz} \times 1024 (=17.07\mu\text{s})$ です

本関数で周期を変更した場合でもデューティ比は周期変更前の値が維持されます

補足:

実際の動作としては、設定した値+1 の周期(cycle=1 とした場合は、34.13us の周期)となります  
引数として 0 を与えた場合は、引数=1 に変換されます

### led8\_duty\_set

概要: LED8 点滅デューティ比設定関数

宣言:

```
int led8_duty_set(float duty);
```

説明:

LED8 の点滅のデューティ比の設定を行います

引数:

duty デューティ比(0-1)

戻り値:

正常終了(0)

補足:

引数として 1 以上の値を与えた場合は、1(100%)に変換されます  
引数として負の値を与えた場合は、0(0%)に変換されます

### led8\_on

概要: LED8 点滅動作開始関数

宣言:

```
int led8_on(void);
```

説明:

LED8 の点滅を開始します

引数:

なし

戻り値:

正常終了(0)

### led8\_off

概要: LED8 点滅動作停止関数

宣言:

```
int led8_off(void);
```

説明:

LED8 の点滅を停止します

引数:

なし

戻り値:

正常終了(0)

※本関数で設定した周期、デューティ比は TPU2 タイマの値に反映されます

※TPU2 タイマの割り込みを有効化しているため、TPU2 タイマの周期やデューティ(コンペアマッチ値)に応じて他の機能呼び出すような使い方も可能です

### 1.1.6. ステッピングモータ(J5)

#### stepping\_motor\_init

概要: ステッピングモータ初期化関数

宣言:

```
int stepping_motor_init(void);
```

説明:

ステッピングモータの初期化を行います

引数:

なし

戻り値:

正常終了(0)

補足:

ステッピングモータで使用している PB4-PB7 端子を出力設定、及び L 出力とします

#### stepping\_motor\_stop

概要: ステッピングモータ初期化関数

宣言:

```
int stepping_motor_stop(void);
```

説明:

ステッピングモータを停止します

引数:

なし

戻り値:

正常終了(0)

補足:

ステッピングモータで使用している PB4-PB7 端子を L 出力とします

#### stepping\_motor\_step\_move

概要: ステッピングモータ動作関数

宣言:

```
int stepping_motor_step_move(int direction);
```

説明:

ステッピングモータを 1 ステップ動かします

引数:

direction: 1:右回転, -1:左回転

戻り値:

正常終了(0)

引数エラー(-1)

### stepping\_motor\_drive

概要: ステッピングモータ位置指定関数

宣言:

```
int stepping_motor_drive(unsigned char pattern);
```

説明:

ステッピングモータの印加パターンを指定します

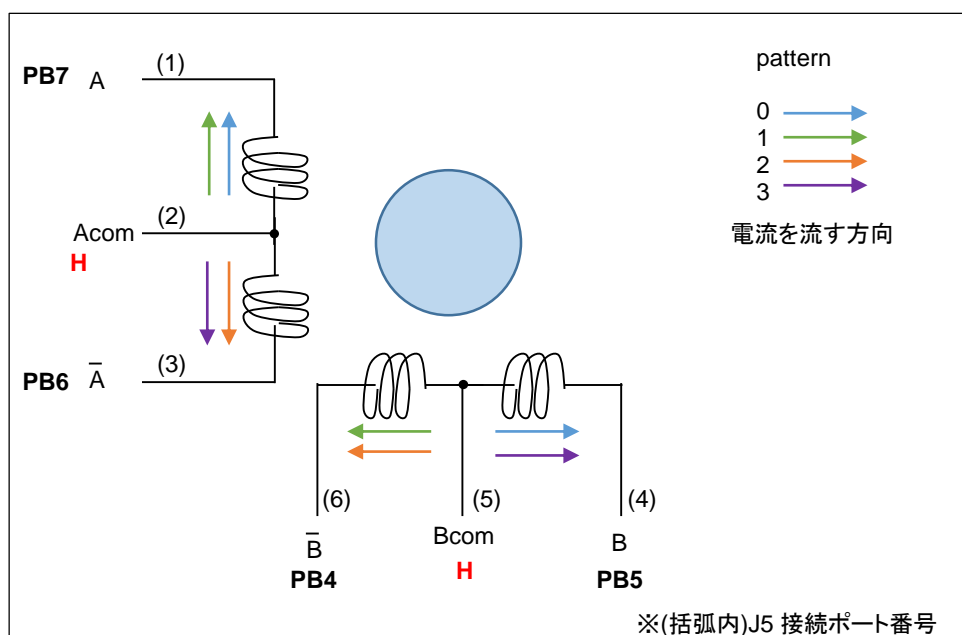
引数:

pattern: 印加パターン(0~3)

戻り値:

正常終了(0)

— 指定する pattern とステッピングモータに流す電流の方向の関係 —



pattern	PB7	PB6	PB5	PB4
	A	$\bar{A}$	B	$\bar{B}$
0	L		L	
1	L			L
2		L		L
3		L	L	

pattern 引数に、0~3 の値を入れて本関数を呼び出した場合、上記の方向の電流を流します。

### 1.1.7. A/D 変換(R29)

#### adc\_start

概要: A/D 変換動作開始関数

宣言:

```
int adc_start(void);
```

説明:

A/D 変換の定期的な実行を開始します

引数:

なし

戻り値:

正常終了(0)

補足:

CMT3(1ms)毎に、A/D 変換を実行し結果をグローバル変数(g\_adc\_result)に格納します

#### adc\_stop

概要: A/D 変換動作停止関数

宣言:

```
int adc_stop(void);
```

説明:

A/D 変換の定期的な実行を停止します

引数:

なし

戻り値:

正常終了(0)

#### adc\_val

概要: A/D 変換値取得関数

宣言:

```
unsigned short adc_val(void);
```

説明:

A/D 変換の結果の値を返します

引数:

なし

戻り値:

A/D 変換値(0-4095)

補足:

A/D 変換結果は移動平均が計算されます(デフォルトは 16 値の移動平均)

## 1.1.8. 7 セグメント LED(SEG1)

### led7seg\_init

概要: 7 セグメント LED 初期化関数

宣言:

```
int led7seg_init(void);
```

説明:

7 セグメント LED の初期化を行います

引数:

なし

戻り値:

正常終了(0)

### led7seg\_start

概要: 7 セグメント LED 動作開始関数

宣言:

```
int led7seg_start(void);
```

説明:

7 セグメント LED の駆動動作を開始します

引数:

なし

戻り値:

正常終了(0)

補足:

CMT3(1ms)毎に、表示桁を切り替えて全ての桁が点灯しているように表示します

### led7seg\_stop

概要: 7 セグメント LED 動作停止関数

宣言:

```
int led7seg_stop(void);
```

説明:

7 セグメント LED の駆動動作を停止します

引数:

なし

戻り値:

正常終了(0)

## led7seg\_segment\_data\_set

概要: 7 セグメント LED セグメント制御関数

宣言:

```
int led7seg_segment_data_set(unsigned char column, unsigned char seg, unsigned char seg_on_off);
```

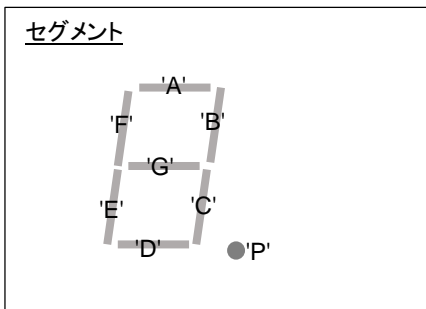
説明:

7 セグメント LED のセグメント単位での点灯・消灯制御

引数:

column: 点灯条件を変える桁を指定(1-4)(一番右側が 1, 一番左側が 4)

seg: 点灯条件を変えるセグメントを指定('A' ~ 'G', 'P')(キャラクタの'A'~'G', 'P'で指定)



seg\_on\_off: 0 の場合はセグメントの消灯、1 の場合はセグメントの点灯





戻り値:

正常終了(0)





引数エラー(-1)

使用例:

```
led7seg_segment_data_set(4, 'G', 1); //4 桁目に '-' を表示 (セグメント G を ON)
led7seg_segment_data_set(4, 'P', 1); //4 桁目の小数点を表示 (セグメント P を ON)
led7seg_segment_data_set(3, 'B', 1); //3 桁目に 'H' を表示 (セグメント B を ON)
led7seg_segment_data_set(3, 'C', 1); //3 桁目に 'H' を表示 (セグメント C を ON)
led7seg_segment_data_set(3, 'E', 1); //3 桁目に 'H' を表示 (セグメント E を ON)
led7seg_segment_data_set(3, 'F', 1); //3 桁目に 'H' を表示 (セグメント F を ON)
led7seg_segment_data_set(3, 'G', 1); //3 桁目に 'H' を表示 (セグメント G を ON)
```

column	4	3	2	1
7 セグ表示				

```
led7seg_segment_data_set(3, 'G', 0); //3桁目のセグメント G を OFF
```

column	4	3	2	1
7セグ表示				

### led7seg\_disp\_hex

概要: 7セグメントLED セグメント表示関数

宣言:

```
int led7seg_disp_hex(unsigned short data);
```









説明:









7セグメントLEDに16進表示を行います

引数:

data: 表示させるデータ(0x0 ~ 0xFFFF)

dataに応じて16進4桁で以下の様なパターンを表示します

data	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
7セグ表示								

data	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
7セグ表示								

戻り値:

正常終了(0)

### led7seg\_disp\_num

概要: 7セグメントLED セグメント表示関数

宣言:

```
int led7seg_disp_num(short data, unsigned char dp);
```

説明:

7セグメントLEDに数値表示を行います

引数:

data: 表示させる数値(-999 ~ 9999)

dp: 小数点位置(0-4), 0:小数点を表示しない, 1~4:指定した桁に小数点表示

戻り値:

正常終了(0)

引数エラー(-1)



補足:

```
led7seg_disp_num(1234, 2);  
→123.4 を表示する
```

### 1.1.9. マトリックススイッチ(SW11~SW26)

#### matrixsw\_init

概要: マトリックススイッチ初期化関数

宣言:

```
int matrixsw_init(void);
```

説明:

マトリックススイッチの初期化を行います

引数:

なし

戻り値:

正常終了(0)

#### matrixsw\_start

概要: マトリックススイッチ動作開始関数

宣言:

```
int matrixsw_start(void);
```

説明:

マトリックススイッチの定期的な読み取りを開始します

引数:

なし

戻り値:

正常終了(0)

補足:

CMT3(1ms)毎に、読み取る列を変えていき、4 サイクル(4ms)で全てのキーのスキャンが一巡します

#### matrixsw\_stop

概要: マトリックススイッチ動作停止関数

宣言:

```
int matrixsw_stop(void);
```

説明:

マトリックススイッチの定期的な読み取りを停止します

引数:

なし

戻り値:

正常終了(0)

### matrixsw\_read

概要: マトリックススイッチ読み出し関数

宣言:

```
int matrixsw_stop(unsigned char key);
```

説明:

マトリックススイッチの状態を読み出します

引数:

key: 0:SW11 の読み出し, 1:SW12 の読み出し..., 15:SW26 の読み出し

戻り値:

0: スイッチが押されている

1: スイッチが押されていない

引数エラー(-1)

## 1.2. lcd\_1602.c

lcd\_1602.c 内には、キャラクタ LCD の制御関数が含まれます。

### lcd\_init

概要: LCD 初期化関数

宣言:

```
void lcd_init(void);
```

説明:

LCD(SC1602)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: 送信するコマンド

戻り値:

なし

使用例:

```
lcd_cmd(0x0f);  
カーソルを表示、カーソルをブロックで表示  
lcd_cmd(0x0c);  
カーソルを非表示
```

### lcd\_hs1

### lcd\_hs2

概要: LCD カーソル移動関数

宣言:

```
void lcd_hs1(void);
```

```
void lcd_hs2(void);
```

説明:

LCD のカーソル(文字を表示する位置)を変更します

lcd\_hs1 は、1 行目 1 文字目、lcd\_hs2 は、2 行目 1 文字目です

引数:

なし

戻り値:

なし

### lcd\_clear

概要: LCD 表示クリア関数

宣言:

```
void lcd_clear(void);
```

説明:

LCD 表示をクリアします

引数:

なし

戻り値:

なし

### lcd\_write\_char

概要: LCD 文字表示関数

宣言:

```
void lcd_write_char(unsigned char c);
```

説明:

LCD に 1 文字表示させます(表示は、現在のカーソル位置です)

引数:

c: 表示する文字を指定

戻り値:

なし

### lcd\_write\_hex

概要: LCD 表示関数

宣言:

```
void lcd_write_hex(unsigned char c);
```

説明:

LCD に 1 バイトの hex コードを表示させます

引数:

c: 表示するコード

戻り値:

なし

使用例:

```
lcd_write_hex(0x2d); //LCD画面に"2D"を表示させます
```

### lcd\_write\_byte\_int

概要: LCD 表示関数

宣言:

```
void lcd_write_byte_int(unsigned char num);
```

説明:

LCD に 1 バイトを数値で表示します

引数:

num: 表示する数値(0~255)

戻り値:

なし

使用例:

```
lcd_write_byte_int(125); //LCD画面に"125"を表示させます
```

### lcd\_write\_short\_int

概要: LCD 表示関数

宣言:

```
void lcd_write_short_int(unsigned short num);
```

説明:

LCD に 2 バイトを数値で表示します

引数:

num: 表示する数値(0~65535)

戻り値:

なし

使用例:

```
lcd_write_byte_int(1200); //LCD画面に"1200"を表示させます
```

### 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 に文字列を表示します

引数:

\*str: 表示する文字列のポインタ(¥0 終端)

戻り値:

なし

使用例:

```
lcd_write_str("ABC"); //LCD画面に"ABC"を表示させます
```

## lcd\_cursor\_move

概要: LCD カーソル移動関数

宣言:

```
void lcd_cursor_move(unsigned char lines, unsigned char pos);
```

説明:

LCD のカーソル位置を指定します

引数:

lines: 行(1-2)

pos: カーソル位置(0-0x3f) ※表示は 16 桁~0x0f までですが、LCD 内の RAM は~0x3f までです  
(スクロール機能を使う事で 17 桁目以降のデータも表示させる事ができます)

戻り値:

なし

使用例:

```
lcd_cursor_move(2, 3);
```

```
lcd_write_str("ABC"); //2行目の4桁目から"ABC"を表示させます
```

## 1.3. sci.c

sci.c 内には、UART(SCI)の制御関数が含まれます。

### sci\_start

概要: SCI 初期化関数

宣言:

```
void sci_start(void);
```

説明:

SCI の動作開始を行います

引数:

なし

戻り値:

なし

### sci\_stop

概要: SCI 停止関数

宣言:

```
void sci_stop(void);
```

説明:

SCI の動作停止を行います

引数:

なし

戻り値:

なし

### sci\_write\_char

概要: SCI 1 文字出力関数

宣言:

```
void sci_write_char(unsigned char c);
```

説明:

SCI に 1 文字出力行います

引数:

c: 表示する文字の文字コード(1 を表示する場合は、0x31 または'1')

戻り値:

なし

## sci\_write\_uint8\_hex

## sci\_write\_uint16\_hex

## sci\_write\_uint32\_hex

概要: SCI hex 出力関数

宣言:

```
void sci_write_uint8_hex(unsigned char c);  
void sci_write_uint16_hex(unsigned short s);  
void sci_write_uint32_hex(unsigned long l);
```

説明:

SCI に hex コードを出力します

引数:

c: hex コード(8bit)  
s: hex コード(16bit)  
l: hex コード(32bit)

戻り値:

なし

使用例:

```
sci_write_str("0x");  
sci_write_uint8_hex(0x5a);  
端末に、"0x5A"が表示されます。
```

## sci\_write\_uint8

## sci\_write\_uint16

## sci\_write\_uint32

概要: SCI 数値出力関数

宣言:

```
void sci_write_uint8(unsigned char c);  
void sci_write_uint16(unsigned short s);  
void sci_write_uint32(unsigned long l);
```

説明:

SCI に符号なしで数値を出力します

引数:

c: 表示させる数値(8bit)  
s: 表示させる数値(16bit)  
l: 表示させる数値(32bit)

戻り値:

なし

使用例:

```
unsigned short s = 0x8000;  
sci_write_uint16(s);
```



端末に、"32768"が表示されます。

### sci\_write\_int8

### sci\_write\_int16

### sci\_write\_int32

概要: SCI 数値出力関数

宣言:

```
void sci_write_int8( char c );  
void sci_write_int16( short s );  
void sci_write_int32( long l );
```

説明:

SCI に(負の数値の場合)符号付きで数値を出力します

引数:

c: 表示させる数値(8bit)  
s: 表示させる数値(16bit)  
l: 表示させる数値(32bit)

戻り値:

なし

使用例:

```
short s = 0x8000;  
sci_write_int16(s);  
端末に、"-32768"が表示されます
```

### sci\_write\_str

概要: SCI 文字列出力関数

宣言:

```
void sci_write_str(const char *str);
```

説明:

SCI に文字列を出力します

引数:

\*str: 文字列(¥0 終端)

戻り値:

なし

補足:

文字列に、"¥n"(0x0a)が含まれる場合は、"¥r¥n"(0x0d,0x0a)に変換されます

### sci\_write\_flush

概要: SCI 出力待ち関数

宣言:

```
void sci_write_flush(void);
```

説明:

バッファ内の出力が完了するまで待ちます

引数:

なし

戻り値:

なし

補足:

SCI 表示関数は、バッファメモリに表示内容をコピーして、割り込み(もしくは DTC)で出力を行います  
バッファ内の表示内容を空になるまで表示させるのが本関数です

使用例:

```
sci_write_flush();  
sci_stop();
```

### sci\_read\_char

概要: SCI 1 文字読み出し関数

宣言:

```
unsigned short sci_read_char(unsigned char *c);
```

説明:

SCI から 1 文字読み出しを行います

引数:

c: 読み出し結果(ポインタ)

戻り値:

0: 受信データがあり、\*c に読み出した文字(ポインタ)を格納  
0xffff(SCI\_RECEIVE\_DATA\_EMPTY): バッファに溜まっている受信データがない

### sci\_read\_str

概要: SCI 文字列読み出し関数

宣言:

```
unsigned short sci_read_str(char *str, unsigned short size);
```

説明:

SCI から文字列を読み出します

引数:

str: 読み出し結果(ポインタ)  
size: 読み出すバイト数

戻り値:

0: 受信データが size バイト以上あり、\*str に読み出した文字の先頭ポインタを格納  
0xffff(SCI\_RECEIVE\_DATA\_EMPTY): バッファに size バイトの受信データが溜まっていない

### sci\_read\_data\_size

概要: SCI 受信データサイズ取得関数

宣言:

```
unsigned short sci_read_data_size(void);
```

説明:

受信バッファに溜まっているデータサイズ(バイト数)を返します

引数:

なし

戻り値:

0: 受信データなし

>0: 溜まっているデータサイズ

### sci\_read\_buf\_clear

概要: SCI 受信データクリア関数

宣言:

```
void sci_read_buf_clear(void);
```

説明:

受信バッファに溜まっているデータをクリアします(全て読み出し済みとします)

引数:

なし

戻り値:

なし

### float2str

### float2str\_eformat

### double2str

### double2str\_eformat

概要: 浮動小数点数一文字列変換関数

宣言:

```
void float2str(float value, int num, char *str);
```

```
void float2str_eformat(float value, int num, char *str);
```

```
void double2str(double value, int num, char *str);
```

```
void double2str_eformat(double value, int num, char *str);
```

説明:

浮動小数点数を文字列に変換します

(\_eformat は 1.23e-3 の様に e 形式に変換を行う関数です)

引数:

value: 表示させる数値(float/double 型)

num: 小数点以下の桁数

\*str: 文字列変換後の先頭ポインタ

戻り値:

なし

使用例:

```
float f = 1.23456f;
```

```
char buf[20];
```

```
float2str(f, 2, buf);
```

```
sci_write_str(buf);
```

端末に、"1.23"が表示されます。

## 2. アプリケーションプログラムの作成方法

本プロジェクトを使用して、アプリケーションプログラムを作成する方法に関して示します。

main フォルダ以下に、

main\_????.c

を作成する(???は任意の名称)。

※ここでは、説明上 main\_etc.c を作成する事とします

main\_etc.c

```

/*-----
インクルード
-----*/
#include "r_smc_entry.h"

#include "main.h"
#include "io_board.h"
#include "lcd_1602.h"
#include "sci.h"

/*-----
定数定義
-----*/
//[必要に応じて]

/*-----
グローバル変数 (ファイル内)
-----*/
//[必要に応じて]

/*-----
関数プロトタイプ宣言
-----*/
void main_etc(void);

//割り込みコールバック関数
void cmt3_callback_etc(void);
void rtc_callback_etc(void);
void adc_callback_etc(void);
void sw8_callback_etc(void);
void sw9_callback_etc(void);
void sw10_callback_etc(void);
void tpu2a_callback_etc(void);
void tpu2b_callback_etc(void);

```

(a)インクルード

(b)割り込みコールバック関数定義

main\_etc.c(続き)

```

/*-----
関数定義
-----*/
void main_etc(void)
{
    //I/Oボードの各種機能を紹介するサンプルプログラム

    int i;
    //[変数定義]

    //割り込みコールバック関数の割り当て
    cmt3_callback = cmt3_callback_etc;
    rtc_callback = rtc_callback_etc;
    adc_callback = adc_callback_etc;
    sw8_callback = sw8_callback_etc;
    sw9_callback = sw9_callback_etc;
    sw10_callback = sw10_callback_etc;
    tpu2a_callback = tpu2a_callback_etc;
    tpu2b_callback = tpu2b_callback_etc;

    //プログラム本体

```

(c)割り込みコールバック関数割り当て

(a)インクルードは、

r\_smc\_entry.h

main.h

io\_board.h

lcd\_1602.h

sci.h

をインクルードしてください。

(b)割り込みコールバック関数定義

サンプルプログラムで予め使用されている割り込み関数を使用する際は、割り込みコールバック関数を定義してください。割り込み処理の最後のタイミングで、ここで定義したコールバック関数が呼び出されるようになります。コールバック関数内には、ユーザ側で行いたい処理を記載します。

コールバック関数	割り込み内容	呼ばれるタイミング	備考
cmt3_callback	コンペアマッチタイマ(CMT3)	1ms 毎	
rtc_callback	リアルタイムクロック(RTC)	1 秒毎	
adc_callback	A/D 変換(ADC12)	A/D 変換が終わったタイミング	CMT3 で A/D 変換が実行されるので 1ms に実行
sw8_callback	IRQ15	SW8 が押されたタイミング	intr_push_sw_init()関数で、割り込みが掛かるタイミングは変更可能
sw9_callback	IRQ13	SW9 が押されたタイミング	
sw10_callback	IRQ4	SW10 が押されたタイミング	
tpu2a_callback	TPU2 コンペアマッチ A	TPU2 の周期終わり	led8_start()で LED8 が動作中に有効
tpu2b_callback	TPU2 コンペアマッチ B	TPU2 のコンペアマッチ	

## (b) 割り込みコールバック関数割り当て

割り込みコールバック関数を、ユーザ定義関数に割り当てます。

割り込みコールバック関数は、関数ポインタとなっていますので、いつでも任意の関数に割り当てが可能です。

(一度割り当てた後で、別の関数に割り当て直す事も可能です。)

## main\_etc.c(続き)

```

//各種初期化
push_sw_init();           //プッシュスイッチ (SW0-SW7) 初期化
intr_push_sw_init(0);    //0:スイッチを押した際に反応する様にする
led7seg_init();          //7セグLEDの初期化
matrixsw_init();         //マトリックススイッチの初期化
led_init();              //LED(LED0-LED7) 初期化
stepping_motor_init();   //ステッピングモータ初期化
lcd_init();              //LCD初期化

//動作スタート
intr_push_sw_start();    //割り込みスイッチ動作開始
R_Config_CMT3_Start();   //ADC, 7セグLED, マトリックススイッチのタイマ動作開始
R_Config_RTC_Start();    //リアルタイムクロック動作開始
adc_start();             //A/D変換動作開始
led7seg_start();         //7セグLED動作
matrixsw_start();        //マトリックススイッチ読み取り

//J7 (USB-miniB) への情報表示 (115,200bps)
sci_start();            //SCI初期化&動作開始
sci_write_str("%n%nCopyright (C) 2024 HokutoDenshi. All Rights Reserved.%n");
sci_write_str("HSBRX65-IO-BOARD SAMPLE PROGRAM.%n%n");

sci_write_str("Board function demo.%n");

//LCDへの情報表示
lcd_hs1();
//      1234567890123456
lcd_write_str("HSBRX65-IO-BOARD");
lcd_hs2();
lcd_write_str("SAMPLE PROGRAM2");

```

(d)初期化

(e)動作スタート

(f)SCI(USB-Serial)への情報表示

(g)キャラクタ LCD への情報表示

## (d)各種初期化

使用したい機能の初期化を行ってください。

## (e)動作スタート

使用したい機能の動作を開始してください。

R\_Config\_CMT3\_Start(); は、「A/D 変換」「7 セグメント LED」「マトリックススイッチ」のいずれかを使用する場合は実行が必要です。

(f)(g)は、USB-Serial や LCD に表示を行う場合に使用してください。

main.h には

```

/*-----
関数プロトタイプ宣言
-----*/
extern void main_etc(void);
extern void main_timer(void);

```

ユーザが追加したメイン関数のプロトタイプ宣言を書いてください。

※main\_etc(), main\_timer()はサンプルとして用意している関数なので、ユーザ側でアプリケーションを作成する際は、これらの関数(及び main\_etc.c, main\_timer.c のファイル)を消してしまっても問題ありません。

HSBRX65-IO-BOARD\_SAMPLE2.c に、本来のメイン関数がありますので、このメイン関数からユーザ側で定義したメイン関数を呼び出す様にしてください。

```

#include "r_smc_entry.h"

#include "main.h"
#ifdef __cplusplus
// #include <ios> // Remove the comment when you use ios
// _SINT ios_base::Init::init_cnt; // Remove the comment when you use ios
#endif

void main(void);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif

void main(void)
{
// 起動するアプリの選択 (SW8-SW10を押した状態で起動, SW番号の若い方優先)
if (PORT0.PIDR.BIT.B7 == 0)
{
// リセット解除時にSW8が押されている場合は、タイマアプリを起動
main_timer();
}
else if (PORT0.PIDR.BIT.B5 == 0)
{
// リセット解除時にSW9が押されている場合は、
}
else if (PORT1.PIDR.BIT.B4 == 0)
{
// リセット解除時にSW10が押されている場合は、
}

// 起動時にSW8-SW10が押されていない場合は、ボード機能紹介デモアプリを起動
main_etc();
}

```



ダウンロードしたプロジェクトは上記のようになっており、起動時に SW8 が押されている場合と、そうでない場合で呼び出すユーザ定義のアプリケーションプログラムを変えています。1 種類のアプリケーションを動かす場合は、

```
void main(void)
{
    //ユーザ側で定義したアプリケーションを実行
    main_user();
}
```

上記の様な形で問題ありません。

## 2.1. io\_board.c に含まれる関数の使用方法

### 2.1.1. プッシュスイッチ(SW0-SW7)

sw\_read()関数で、スイッチの状態を読み出せます。

```
unsigned char sw;
```

```
sw = sw_read();
```

SW0-SW7 が押されていない場合は、sw=0xFF となります。SW0 のみ押されている場合は、sw=0xFE。SW0, SW1, SW2 が押されている場合は、sw=0xF8 です。押されているキーに対応するビットが 0 となり、押されていないキーに対応するビットは 1 となります。

### 2.1.2. 割り込み対応プッシュスイッチ(SW8-SW10)

```
intr_push_sw_init(0);
```

```
intr_push_sw_start();
```

上記を実行すると、SW8~SW10 が押されたタイミングで、SW8:IRQ15, SW9:IRQ13, SW10:IRQ4 の割り込みが入るようになります。

```
intr_push_sw_init(1);
```

とすると、SW8~SW10 が押されている状態から、離されたタイミングで割り込みが入るようになります。

```
intr_push_sw_init(2);
```

とすると、SW8~SW10 を押したタイミングと離れたタイミングの両方で割り込みが入るようになります。

割り込みが掛かった際は、割り込みコールバック関数

```
sw8_callback_etc()
```

(sw8\_callback = sw8\_callback\_etc; でsw8\_callback を sw8\_callback\_etcに割り付けた場合)

```
sw9_callback_etc()
```

```
sw10_callback_etc()
```

が呼ばれますので、この関数の中でスイッチを押したときの処理、もしくはフラグの処理を行ってください。

main\_etc()や main\_timer()では、スイッチを押した時の動作は、main\_etc(), main\_timer()内に記載し、割り込みコールバック関数内では、フラグを変更する様にしています。

### 2.1.3. LED(LED0-LED7)

LED0~LED7 の点灯状態を変える場合は、

```
led_init(); //最初に実行
```

```
led_set(0xFF); //全点灯
```

```
led_set(0x01); //LED0のみ点灯
```

```
led_set(0x80); //LED7のみ点灯
```

上記関数で行います。

bit=0 で消灯、bit=1 で点灯です。(回路としては、PA0=L で点灯です。led\_set()関数に、led\_set(0x01)とすると、PORTE は PORTE.PODR.BYTE = 0xFE に設定されます。)

### 2.1.4. ブザー(B1)

```
buzzer_freq_set(3000);
```

```
buzzer_on();
```

でブザーを鳴らします。JP2 は上側をショートに設定してください。

buzzer\_freq\_set()は、ブザーの周波数を設定します。300~8000[Hz]の範囲で設定が可能です。ブザーが鳴っている場合でも、周波数の設定は可能です。

buzzer\_off(); でブザーは停止します。

## 2.1.5. タイマ連動 LED(LED8)

LED8 は、JP1 を下側ショートに設定した場合、タイマ TPU2 と連動して、点滅します。点滅の周期と、点灯・消灯のタイミングの設定が可能です。

```
led8_cycle_set(58593);
led8_duty_set(0.50f);
led8_on();
```

点滅の周期を 1 秒。点滅の時間を、点灯:消灯=50:50 に設定し、LED8 を点滅させる場合の設定です。

周期設定の 1 サイクルは、 $1/60\text{MHz} \times 1024 = 17.067[\text{us}]$ です。led8\_cycle\_set()で、58593 を指定した場合は

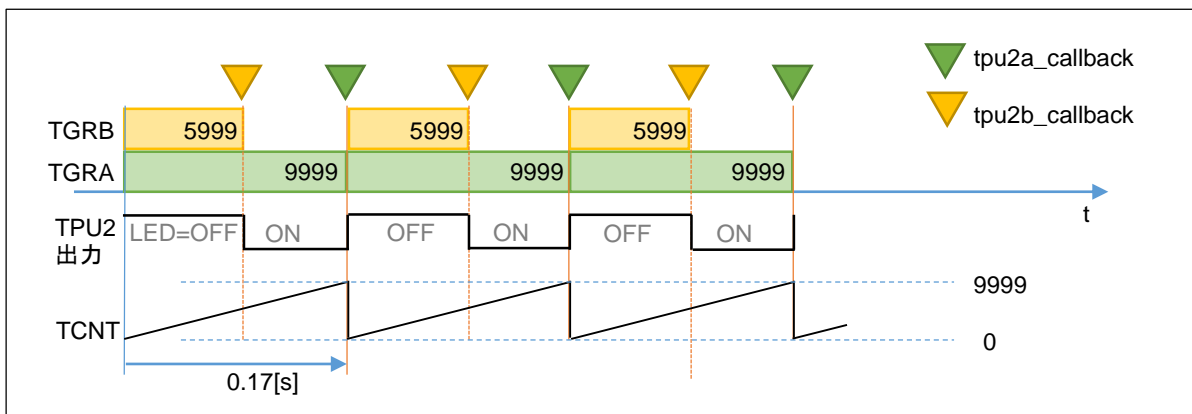
周期:  $(58593 + 1) \times 17.067[\text{us}] = 1.000004[\text{s}]$

となります。

duty は、0.4 を指定した場合は、全体の 40%の時間が点灯、60%の時間が消灯となります。

### ・TPU2 のタイミング

```
led8_cycle_set(9999);
led8_duty_set(0.40f);
```



周期を、9999とした場合は、0~9999カウントの10,000周期(0.17s)となります。

LED8は、LでONとなりますので、 $(1 - \text{led8\_duty\_set}())$ で指定した値が、実際のTPU2タイマの(正の)デューティ比となります。

led8\_off(); で LED8 の点滅は止まります。TPU2 タイマも止まります。

LED8 (TPU2 タイマ) のタイミングで、何か別な処理を行わせる事が可能です。

(`tpu2a_callback = tpu2a_callback_etc;`で `tpu2a_callback` を `tpu2a_callback_etc` に、  
`tpu2b_callback = tpu2b_callback_etc;`で `tpu2b_callback` を `tpu2b_callback_etc` に割り付けた場合)

例えば、

```
void tpu2a_callback_etc(void)
{
    buzzer_on();
}
```

```
void tpu2b_callback_etc(void)
{
    buzzer_off();
}
```

上記で、ブザーを間欠的に鳴らす(ピー ピー ピー の様に、鳴ったり休んだり)ことが可能です。鳴っている時間と無音時間の比は、`led8_duty_set()`で設定可能ですが、上記の場合は、 $(1 - \text{led8\_duty\_set}())$ で設定した値がブザーが鳴っている時間となります。

`main_timer()`では、アラームを鳴らす際に、`main_etc()`ではステッピングモータを回す際に、TPU2 のコールバック関数を使用しているので、参考にしてみてください。

## 2.1.6. ステッピングモータ(J5)

```
stepping_morter_init();
```

で初期化後、

```
stepping_motor_step_move(1);
```

ステッピングモータを右方向に 1 ステップ動かす。

```
stepping_motor_step_move(-1);
```

ステッピングモータを左方向に 1 ステップ動かす。

```
stepping_motor_drive(3);
```

ステッピングモータに加える電流のパターンを、パターン 3 (PB5=PB6=L) とする。

※電流の向きとパターンの関係は、1.1.6 ステッピングモータ(J5)の項を参照

```
stepping_motor_stop();
```

ステッピングモータに印加する電流を止める。

※実際にステッピングモータを接続しなくても、LED9-LED12 の点灯パターンで回転のイメージは掴めます

### 2.1.7. A/D 変換(R29)

```
adc_start();
```

で、A/D 変換動作を開始してください。

```
unsigned short val;
```

```
val = adc_val();
```

で、R29 の回転角度に応じた、A/D 変換値が得られます。

val は、R2 を右に目一杯回した場合は 0。左に目一杯回した場合は、4095 となります。

(時計回りに回した際に値を増加させたい場合は、 $(4095 - \text{adc\_val}())$ の値を計算に使ってください。)

A/D 変換は、CMT3 の 1ms のタイミングで実行され、A/D 変換が終わった時点で、A/D 変換割り込みが入ります。A/D 変換割り込み内では、過去の 16 回の A/D 変換結果の平均値 (移動平均) を取り、`adc_val()` の戻り値としています。

(移動平均の回数は、`io_board.h` 内で変更可能です。)

### 2.1.8. 7 セグメント LED(SEG1)

7 セグメント LED を使用する場合、

```
led7seg_init();
```

```
led7seg_start();
```

を実行してください。7 セグメント LED は、ダイナミック点灯(全てのセグメントを静的に ON/OFF 制御ができない。時間を切って、セグメントの点灯を制御する必要がある)のため、上記関数でタイマ(CMT3, 1ms)を使い、動的にセグメントを ON/OFF しています。

7 セグメント LED に数値を表示する関数は用意されており、

```
led7seg_disp_num(1234, 0);
```

で、"1234"を表示させる事が出来ます。

```
led7seg_disp_num(-453, 2);
```

で、"-45.3"の表示が可能です。第一引数は、-999 ~ 9999 まで。第二引数は、小数点を表示しない場合は 0。一番右の桁(1 桁目)の後ろに小数点表示する場合は 1。一番左の桁(4 桁目)の後ろに小数点表示する場合は 4 です。

また、16 進数 4 桁の表示を行いたい場合は、

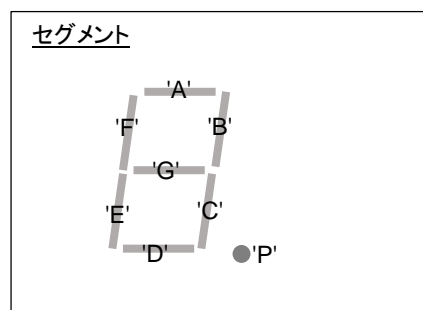
```
led7seg_disp_hex(0xABCD);
```

の様に関数を呼び出せば、表示が可能です。

その他、セグメント単位で ON/OFF させることも可能です。

現在の点灯状態から、4 桁目を全て消灯にしたい場合は、

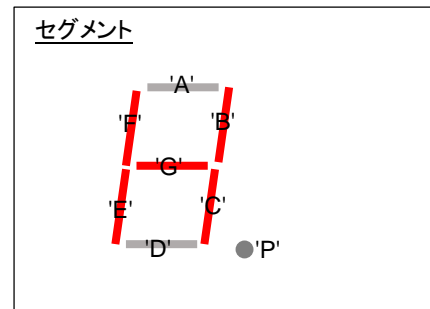
```
lcd7seg_segment_data_set(4, 'A', 0);
lcd7seg_segment_data_set(4, 'B', 0);
lcd7seg_segment_data_set(4, 'C', 0);
lcd7seg_segment_data_set(4, 'D', 0);
lcd7seg_segment_data_set(4, 'E', 0);
lcd7seg_segment_data_set(4, 'F', 0);
lcd7seg_segment_data_set(4, 'G', 0);
lcd7seg_segment_data_set(4, 'P', 0);
```



で、4 桁目のセグメント(A~G と小数点の P)を、消灯とする事が出来ます。

3 桁目に、アルファベットの H を表示させたい場合は、

```
lcd7seg_segment_data_set(3, 'A', 0);
lcd7seg_segment_data_set(3, 'B', 1);
lcd7seg_segment_data_set(3, 'C', 1);
lcd7seg_segment_data_set(3, 'D', 0);
lcd7seg_segment_data_set(3, 'E', 1);
lcd7seg_segment_data_set(3, 'F', 1);
lcd7seg_segment_data_set(3, 'G', 1);
lcd7seg_segment_data_set(3, 'P', 0);
```



で可能です。

### 2.1.9. マトリックススイッチ(SW11~SW26)

```
matrixsw_init();
matrixsw_start();
```

でマトリックススイッチの読み取りを有効化します。マトリックススイッチは、タイマ(CMT3, 1ms)で読み取りを行っています。1 回毎に、1 列のキーの状態を読み取るので、4 サイクル(4ms)でキーの読み取りが一巡します。

タイミング的には、同時に 16 個のキーを読み出している訳ではなく、あるタイミングで SW11, SW12, SW13, SW14 の読み取り。1ms 後に、SW15, SW16, SW17, SW18 の読み取りという様に、時分割でスイッチの読み取りを行っています。

```
int sw;
sw = matrixsw_read(IOBOARD_SW11);
```

SW11 が押されている場合は、sw=0。押されていない場合は、sw=1 となります。

#define IOBOARD\_SW11 (0) で定義されています。

matrixsw\_read()の引数は、0~15 で、SW11~SW26 の読み取りを行います。

## 2.2. lcd\_1602.c に含まれる関数の使用方法

```
lcd_init();
```

で最初に、初期化してください。

```
unsigned char a = 0x12;
```

```
unsigned char b = 0x34;
```

```
lcd_hs1();
```

```
lcd_write_str("MESSAGE");
```

```
lcd_hs2();
```

```
lcd_write_str("VAL=0x");
```

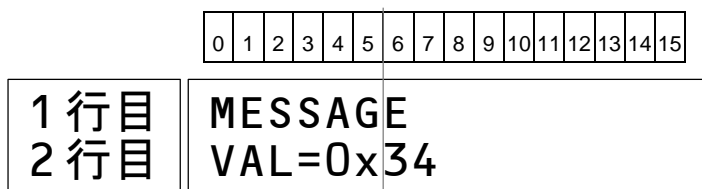
```
lcd_wrt_hex(a);
```

<b>MESSAGE</b>
<b>VAL=0x12</b>

1 行目に、"MESSAGE"を表示して、2 行目に 16 進で変数 a の中身を表示する場合は、上記の様に行えます。

```
lcd_move_cursor(2, 6);
```

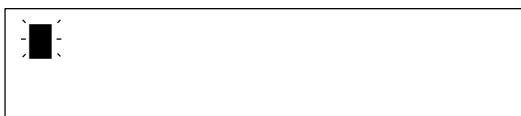
```
lcd_wrt_hex(b);
```



特定の位置から、変数 b の内容を出力したい場合は、上記の様に行えます。

```
lcd_clear();
```

```
lcd_cmd(0x0f);
```



LCD 画面をクリアして、カーソルを点滅表示。

lcd\_cmd()は、任意のコマンドを送ることが出来ます。



LCDにはコマンドを送ることで、表示をスクロールさせたり、カーソルの表示・非表示。アルファベット、記号、カタカナ以外の文字パターンの定義などが行えます。LCDが受け付けるコマンドに関しては、「HSBRX65-IO-BOARD 取扱説明書 の付録 キャラクタ LCD(SC1602)参考資料」や、SC1602タイプのLCDのデータシートをご確認ください。

### 2.3. sci.c に含まれる関数の使用方法

UART(SCI)の通信先は、ボードの USB-miniB コネクタ(J7)となります。J7をPCと接続すると、PCからは、COMポート COMn(nは環境によって異なる)として認識されます。PC上では仮想端末ソフト(Teraterm や putty など)を使用してください。通信速度は、115,200bps。その他の条件は、8bit, パリティなし, 1ストップビットです(基本的には速度以外は、通常の仮想端末ソフトのデフォルト値から変更不要です)。

```
unsigned short a = 0x1234;
```

```
sci_start();
```

```
sci_write_str("MCU Board boot message!\n");
```

```
sci_write_str("\nHEX: 0x");
```

```
sciw_write_uint16_hex(a);
```

```
sci_write_str("\nDEC: ");
```

```
sciw_write_uint16(a);
```

```
MCU Board boot message!
```

```
HEX: 0x1234
```

```
DEC: 4660
```

文字列と、数値の表示は上記の様にできます。

数値表示は、

	8bit	16bit	32bit
符号付き 10 進数	sci_write_int8	sci_write_int16	sci_write_int32
符号なし 10 進数	sci_write_uint8	sci_write_uint16	sci_write_uint32
16 進数	sciw_write_uint8_hex	sciw_write_uint16_hex	sciw_write_uint32_hex

10 進の符号の有無、16 進表示。8bit, 16bit, 32bit で上記関数が用意されています。

float や double の値の表示は、

```
float f = 1.2345f;
```

```
double = 0.987654;
```

```
char buf[20];

float2str(f, 2, buf); //小数点以下 2 桁まで表示
sci_write_str(buf);
sci_write_str("¥n");
double2str(d, 4, buf); //小数点以下 4 桁まで表示
sci_write_str(buf);
```

1.23 0.9876
----------------

浮動小数点数を文字列に変換する関数を使用して、上記の様に行えます。

なお、sci.c に含まれる関数を使用しなくても

```
#include <stdio.h>
```

```
sprintf(buf, "%.2f¥n", f);
sci_write_str(buf);
```

でも同様の事が行えます。(標準ライブラリ関数の方が様々な書式表示に対応しています。)

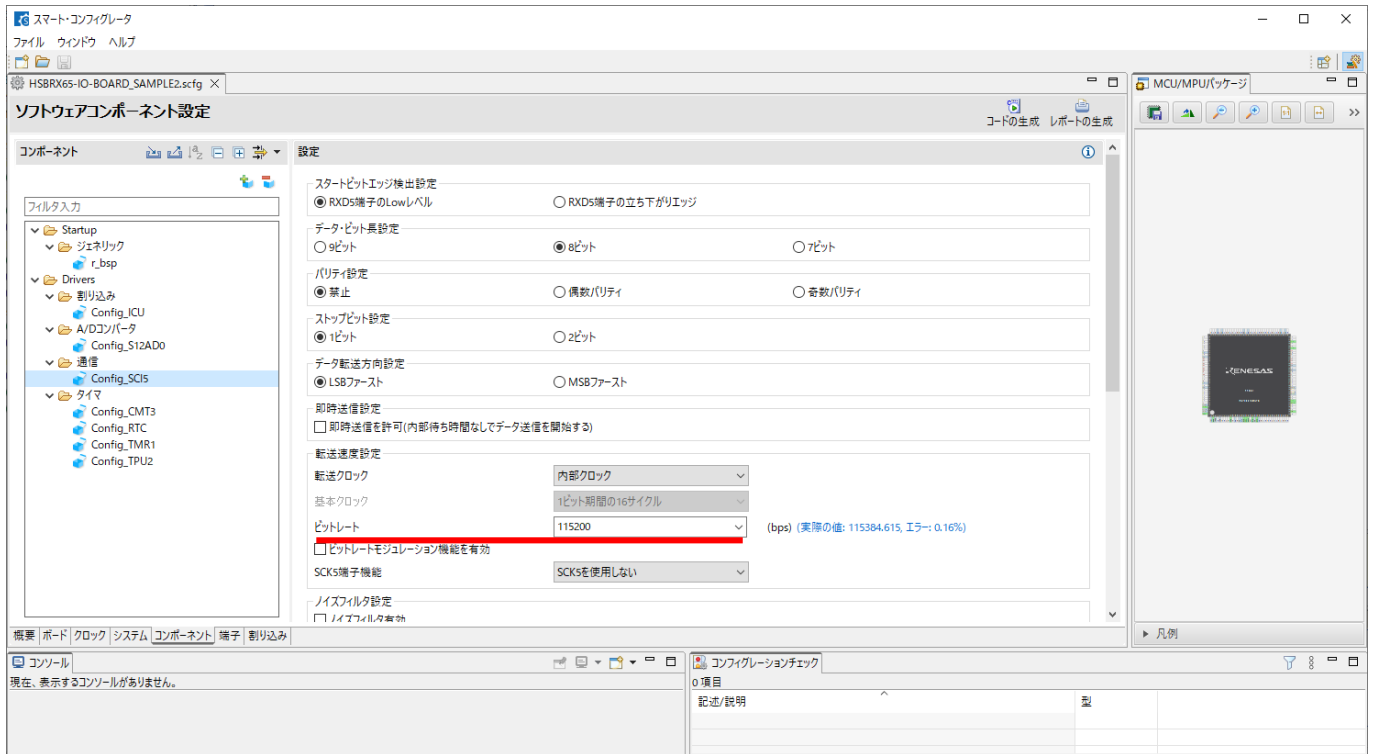
sci.h 内の関数を使用する事で、PC のキーボードからの文字入力も可能です。キーボードからのコマンドで、マイコン側が応答するようなプログラムを組む場合は、

```
unsigned char c;
unsigned short ret;

while (1)
{
    ret = sci_read_char(&c);
    if (ret == SCI_RECEIVE_DATA_EMPTY) continue;
    switch (c)
    {
        case 'A':
            sci_write_str("input is A¥n");
            break;
```

上記の様に、sci\_read\_char()関数で、キーボードからの入力の読み取りが行えます。

デフォルトで設定されている、通信速度(115,200bps)を変更する場合は、スマート・コンフィグレータの設定画面から、コンポーネントタブ



ビットレートの欄を選択する事で、9,600bps やその他の通信速度に設定が可能です。  
 (プルダウンで予め用意されている通信速度に変更する事や、ボックスに直接数値を入力する事で、任意の速度にも設定が可能です。)

※ビットレートの右側に青文字で記載されている、エラーが概ね 3%未満となる通信速度を設定してください

ビットレート誤差が大きいと通信が出来なくなります(誤差が大きい場合は、その下のビットモジュレーションを有効のチェックを入れると、誤差を小さくすることも出来ます)

## 3. サンプルプログラムで定義されている定数など

### 3.1. io\_board.h

```
#define IOBOARD_ADC_AVERAGE_NUM (16)
```

R29 のボリュームの A/D 変換値の移動平均値を取ります。デフォルトでは、1ms 毎に A/D 変換値を取り 16 値の平均値を、変換結果とします。この値を、0 または 1 にすると移動平均の計算は行われないようになります。

### 3.2. sci.h

```
#define SCI_SEND_BUF_SIZE 1024
```

送信で使用するバッファサイズ(バイト数)を定義。送信バッファは、2 面(送信用と格納用)使用しているので、ここで定義した 2 倍のメモリを消費します。

```
#define SCI_RECV_BUF_SIZE 16
```

受信で使用するバッファサイズを定義。デフォルトでは、キーボードからの入力を想定して、16バイトと小さな値となっています。プログラムからのデータを受信する場合等は、サイズを増やしてください。

```
#define SCI_USE_FLOAT
```

浮動小数点の関数(float2strなど)を使用する場合は定義。

```
#define SCI_USE_TX
```

送信系の関数を使う場合は定義。

```
#define SCI_USE_RX
```

受信系の関数を使う場合は定義。

関数の無効化は、ROMの小さなマイコンで無効化できる様にしている定数なので、大容量のROM(2MB)を搭載しているRXでは変更は不要かと思えます。

## 4. マイコン使用機能

### 4.1. 使用機能一覧

	機能	用途	備考
TMR1	タイマ	ブザーの駆動	P17 でブザーを鳴らす設定 (JP2 は P17 側を選択)
TPU2	タイマ	LED8 の駆動 TPU2 のインターバルタイマ	P15 で LED8 を駆動する設定 (JP1 は P15 側を選択)
SCI5	UART(SCI)	通信	J7(USB-miniB)で PC と通信
CMT3	タイマ	A/D 変換の起動 7 セグメント LED の駆動 マトリックススイッチの読み取り	ボード搭載機能の定期処理 (1ms のインターバルタイマ)
S12AD0	A/D 変換	ボリューム(R29)の A/D 変換	
RTC	リアルタイム クロック	時計・カレンダー機能の使用 1 秒のインターバルタイマ	
ICU	割り込み	SW8, SW9, SW10 の検出	IRQ4, IRQ13, IRQ15

上記は、スマート・コンフィグレータで設定している機能です。プログラムの中でも、マイコン周辺機能の設定に関してはスマート・コンフィグレータ上で設定する事で、完結してしまいます。

### 4.2. I/O ボードの各機構に割り当てられているマイコン機能

I/O ボード機構	端子	マイコン機能
7 セグメント LED(SEG1-2)	PA0-7, P74-77	汎用ポート
キャラクタ LCD(J4)	P50,52,P90-93	汎用ポート
ブザー(B1)	P17 (P15)	TMO1
LED(LED8)	(P17) P15	TIOCB2
ボリューム(R29)	P40	AN000
割り込み用スイッチ(SW8)	P07	IRQ15
割り込み用スイッチ(SW9)	P05	IRQ13
割り込み用スイッチ(SW10)	P14	IRQ4
マトリックススイッチ (SW11-26)	P60-67	汎用ポート
プッシュスイッチ(SW0-7)	PD0-7	汎用ポート
LED(LED0-7)	PE0-7	汎用ポート
ステッピングモータ(J5)	PB4-7	汎用ポート
USB-Serial(J7)	PC2-3	RXD5, TXD5

### 4.3. 使用割り込み一覧

本サンプルプログラムで使用している割り込みの一覧を示します。

割り込み(機能名)	優先度	用途
S12ADI(S12AD) A/D 変換完了	2	A/D 変換値の移動平均の算出
CMI3(CMT3) コンペアマッチタイマ	3	A/D 変換の起動, 7 セグメント LED の処理, マトリックススイッチの処理
RPD(RTC) リアルタイムクロック	4	1 秒毎のインターバルタイマ
IRQ4(ICU) 端子割り込み	5	SW10 押下検出
IRQ13(ICU) 端子割り込み	5	SW9 押下検出
IRQ15(ICU) 端子割り込み	5	SW8 押下検出
TGI2A(TPU2) TPU2 コンペアマッチ A	6	LED8(TPU2)周期終わりの割り込み
TGI2B(TPU2) TPU2 コンペアマッチ B	6	LED8(TPU2)が OFF→ON に切り替わる際の割り込み
RXI5(SCI5) SCI5 受信割り込み	7	USB-Serial 受信
TXI5(SCI5) SCI5 送信データエンプティ	7	USB-Serial 送信処理
TEI5(SCI5) SCI5 送信完了割り込み	7	USB-Serial 送信処理
ERI5(SCI5) SCI5 エラー割り込み	7	USB-Serial エラー処理

※優先度は、数値の大きなほうが優先です

## 付録

### 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	2024.12.18	—	初版発行

### 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: <https://www.hokutodenshi.co.jp>

### 商標等の表記について

- ・ 全ての商標及び登録商標はそれぞれの所有者に帰属します。
- ・ パーソナルコンピュータを PC と称します。

---

ルネサス エレクトロニクス社 RX651/RX671(QFP-144 ピン)搭載 HSB シリーズマイコンボード向け I/O ボード

# HSBRX65-IO-BOARD

## サンプルソフトウェア マニュアル(2)

株式会社 **北斗電子**

©2024 北斗電子 Printed in Japan 2024 年 12 月 18 日改訂 REV.1.0.0.0 (241218)

---