

RH850-can5 実験キット用 テストプログラム説明書(初版)

商標等の表記について

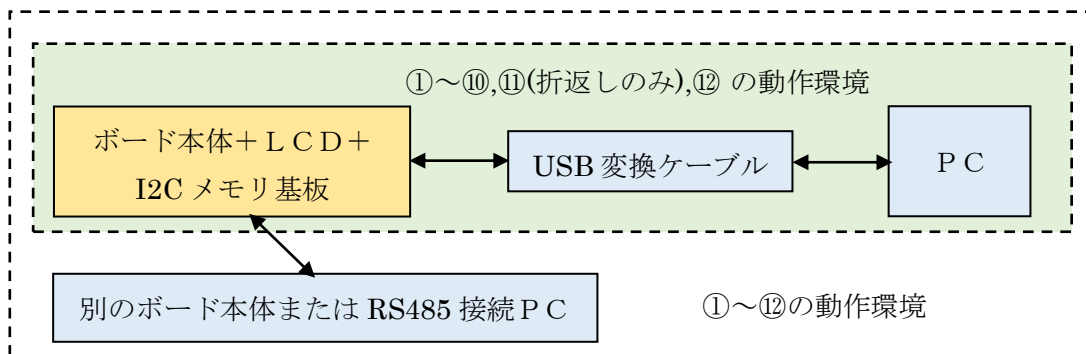
- ・全ての商標および登録商標はそれぞれの所有者に帰属します。
- ・パーソナルコンピュータの略号を PC とします。

1. 概要

本プログラムは、RH850-can5 実験キットの動作確認用です。

出荷時にボードに書き込みしてありますので、すぐに動作確認を行えます。

- ①LED 表示：タクト SW に応じて LED の点灯／点滅の確認ができます。
- ②ブザー鳴動：タクト SW に応じてブザーの鳴動が確認できます。
ホスト PC からのコマンドによりメロディ演奏が確認できます。
- ③SW 動作：タクト SW の有効／無効を DIP SW で指定できます。
有効なタクト SW の押下により、LED 点灯状態の変更ができます。
- ④アナログ入力：タクト SW1 の押下またはホスト PC からのコマンドにより、
6 つのアナログ入力値をホスト PC に定期的に送信します。
- ⑤発振確認：クロック出力(デフォルト 100kHz)を測定して精度を確認できます。
- ⑥LCD 表示：LCD モジュールが接続してあれば、漢字表示を確認できます。
- ⑦UART 通信：USB 変換ケーブルにより、ホスト PC と通信できます。
- ⑧ウォッチドッグ機能：タイムアウトによるリセット動作を確認できます。
- ⑨データフラッシュ読み書き動作：本版では未対応
- ⑩CAN 通信：2 c h の CAN ポート同士を接続し、ホスト PC からのコマンドにより
CAN 通信動作を確認できます。
- ⑪RS-485 通信：実験ボード 2 枚の RS-485 ラインを対向して接続した上で、ホスト PC
からのコマンドにより RS-485 通信を確認できます。実験ボード 1 枚での
折返し通信も可能です。
- ⑫I2C メモリ基板接続により PWM 調光、I2C EEPROM アクセスを確認できます。



2. テストプログラム一式

動作確認内容を変更したい場合は、「テストプログラム一式」をダウンロードし、`main.c`を書き換えてビルドします（ビルド方法は「4.ビルド方法」を参照）。

「テストプログラム一式」の内容は以下の通りです(はフォルダ名)。

RH850_can5_tp	
└ RH850_can5_tp.mot	ビルド済み実行コード（出荷時にボード書込み済み）
└ RH850_can5_tp.mtpj	ビルド用プロジェクト・ファイル
└ iodef.h	IO 定義（CS+生成ファイル）
└ SRC	ビルド用ソース・コード（機種依存部）
└ device.h	デバイス定義（リソース割付け）
└ device.c	デバイス初期化関数（クロック初期化など）
└ Font.inc	漢字フォント・データ
└ init.c	実行環境初期化（ <code>main</code> 関数の起動準備）
└ main.c	main 関数（動作確認用プログラム）
└ task.h/c	タスク定義、タスク初期化
COMMON	
└ board.h	ボード識別情報
└ bootup.asm	リセット後起動プログラム
└ common.h	共通定義（変数型、定数、組込み関数、 <code>SFR</code> マクロ）
└ Font.h/c	漢字フォント検索関数、半角フォント
└ kernel.h/c	カーネル
└ melody.h/inc	メロディ定義／演奏データ
└ stdio.h/c	標準入出力関数
└ task_ad.h/c	AD 変換タスク
└ task_sw.h/c	スイッチ入力タスク
└ task_beep.h/c	ブザー出力、メロディ演奏タスク
└ task_clock.h/c	クロック出力用関数
└ task_lcd.h/c	LCD 表示タスク
└ task_led.h/c	LED 表示タスク
└ task_eep_i2c.h/c	パラメータ保存／復旧タスク
└ task_can.h/c	CAN 通信用関数
└ task_rs485.h/c	RS-485 通信用関数／ハンドラ
└ task_uart.h/c	UART 通信用関数／ハンドラ
└ task_pwm.h/c	PWM 出力関数
└ task_i2c.h/c	I2C EEPROM 制御タスク

3. テスト方法

3.1. ソフトウェアの準備

(1)汎用通信ソフト

ホスト PC からコマンド操作する場合は、汎用通信ソフト（例えば Tera Term）を PC にインストールします。通信速度等の設定は以下のようにします。

- ・改行コード：受信 LF、送信 LF
- ・ローカルエコー：あり
- ・漢字コード：SJIS (シフト JIS)
- ・スピード：460800 [bps]
- ・データ：8bit
- ・パリティ：none
- ・ストップビット：1bit
- ・フロー制御：none

注意：Tera Term のデフォルトキーマップではテンキーの/*+が別キーになっています。テンキーを使って操作したい場合はキーマップの変更も行います。

(2)USB ドライバのインストール

[FTDI のサイト](#)から VCP ドライバをインストールします。

(3)ビルド環境

実験ボードに改めてオブジェクトを書き込みたい場合や、ソースを修正して書き込みたい場合はルネサスエレクトロニクス社の CS+ (v8 以降)および CC-RH コンパイラを PC にインストールします（ビルド方法は「4.ビルド方法」を参照）。

3.2. ハードウェアの準備

最初は AC アダプタを接続せず、以下の手順でセットアップします。

① ジャンパ設定

JP2：ショート（付属 LCD モジュール接続）

JP3：ショート（ブザー鳴動）

JP5：ショート（RS-485 ドライバは送信時のみ受信オン）。ただし本ボード 1 枚で折返し通信する場合はオープン（RS-485 ドライバは常時受信オン）

JP8：ショート（ボリューム RV1 を接続）

JP9：ショート（ボリューム RV2 を接続）

②LCD 表示テストを行う場合

CN2 に LCD 基板を接続します。出荷時はパラレル LCD 用となっているため、SPI 接続 LCD 基板を接続する場合は、device.h の TARGET_LCD 定義を下記に変更して再ビルド／書き込みが必要です。

```
#define TARGET_LCD      LCD_TYPE_AQM1248
```

③CAN 通信テストを行う場合

CN5,CN6 の 2 ピン同士、3 ピン同士を接続します。

④RS-485 対向通信テストを行う場合

2 枚の実験ボードの CN4 の 2 ピン同士、3 ピン同士を接続します。

DIP SW5-4 は、主にテストする側はオン、対向側はオフにします。

ただし本ボード 1 枚で折返し通信する場合はオフにします。

注意：DIP SW 設定を間違えると無限ループに陥る可能性があります。

⑤PWM 調光、I2C EEPROM 制御テストを行う場合

I2C メモリ基板を予備ポートに接続してください。

⑥AC アダプタ接続

2 枚の実験ボードを使用する場合は、主に試験する方を先に接続します。

⑦USB 変換ケーブル接続（ホスト PC 接続）

PC 側を先に接続し、次に CN3 を接続します。

⑧汎用通信ソフトを起動

2 枚の実験ボードを使用する場合は、それぞれのウインドウを開きます。

⑨新たにプログラムを書き込む場合

E1/E2 エミュレータの USB ケーブルを PC に接続し、次にターゲット用のケーブルまたはソケットを CN7 に接続します。

CS+を起動し、該当プロジェクトを開き、「デバッグ」→「ビルド&ダウンロード」で書き込みします。実行ボタン(▶)を押すと動作を開始します。

3.3. テスト手順

3.3.1. スイッチ、LED、ブザーのテスト

DIP SW5 の 1~4 のオン／オフにより、タクト SW1~4 の有効／無効を指定できます。

- ・DIP SW5-n をオフにした状態ではタクト SWn は反応しません。
- ・DIP SW5-n をオンにした状態でタクト SWn を押すと、ブザーが鳴り、LEDn の点灯状態が変化します。ブザーの音程および長さ、LED の点滅周期は 1~4 で異なります。

なおタクト SW1 はアナログ入力テストのスタート／ストップも兼ねています（テスト内容は後述の a コマンド参照）。

3.3.2. 発振周波数の確認

マイコンの動作に必要な各種クロックは、16MHz 水晶発振から逡倍によって生成しています。精度確認用として PLL で逡倍したクロックを分周して JP11-2 (I2C メモリ基板のテストピン) に出力しています。デフォルトでは 100kHz を出力します。

3.3.3. 1 文字コマンドによるテスト

ホスト PC から 1 文字送るだけで動作するコマンドを以下に示します。複数を並行して動作させることもできます (ウォッチドッグ除く)。

(1) スタック消費量の表示

- s 4 バイト単位で消費量を表示します。ただし最小値は 128 バイトです。
表示例(コマンド文字含む) : `stack consumption = 916 bytes.`

(2) アナログ入力の AD 変換開始/停止

- a 変換を開始します。1.28 秒周期でボリューム RV1,RV2 および CN1-3,4,5,6 の電圧を AD 変換し、変換結果をホスト PC に表示します。0V~5V の電圧が 0~40950 に対応します。
表示例 : `AD test 9811, 40222, 20300, 20500, 20300, 20500,`
- q AD 変換を停止します。

(3) メロディ演奏開始/停止

- m ブザーからメロディを出力します。
- q 演奏を停止します。

(4) CAN 通信開始/停止

- c ch0 と ch1 間で相互に通信を行い、受信結果をホスト PC に表示します。
表示内容 : `CANc:i:l:v,s`
c=受信チャンネル、i=受信 ID(16 進数)、l=受信データ長、v=数値、s=文字
- d CAN 通信を停止します。

(5) ウォッチドッグのタイムアウト動作確認

- w 意図的にタイムアウトを引き起こします。結果、マイコンはリセットされます。リセット起動時はリセット要因を判定し、通常は `RESET start.`、ウォッチドッグのタイムアウト時は `WDT timeout.` とホスト PC に表示します。

(6)ヘルプ

h コマンドの一覧を表示します。

3.3.4. 行コマンドによるテスト

行コマンドはリターンキー(Enter キー)を押してから有効になります。リターンキーを押す前まではバックスペースキーによる修正も可能です。バックスペースキー以外の編集キーは無効です。

(1) RS-485 通信

[AA・・・AA↵] 任意文字列 AA・・・AA を通信相手に送ります。相手側はホスト PC の相手側ウインドウに \$RS485test: AA・・・AA を表示します。ただし折返し通信の場合は自ウインドウに表示されます。

(2) LCD 表示

]AA・・・AA↵] 任意文字列 AA・・・AA を LCD に表示します。リターンの直前に ¥n を付けると改行します。表示できる全角文字セットは 5362 字です (Font.inc で定義)。文字セットを変更したい場合は、[ベクター](#)に掲載されている漢字フォント・データ生成 (XC32,CC-RH コンパイラ用) Font_XC32 (R1.1 以降) を使用します (本版は K14 フォントのみ対応)。半角は英数のみで半角カナは表示できません。

(3) パラメータ読み書き (本版では I2C メモリに保存)

ps↵] パラメータを保存します。管理データ 4 バイトと、RAM 上でパラメータとして管理している 252 バイトの合計 256 バイトを I2C メモリに書き込みます。格納領域は 768 バイト目から 256 バイトです。

pl↵] パラメータを復元します。I2C メモリから RAM 上に読み込みます。1 回もパラメータを保存していない場合は、オールゼロとなります。

p2,NN↵] NN は 0~77 の範囲です。符号付 2 バイトのデータ群を NN 目から 16 個表示します(10 進形式)。

p2,NN,AA,・・・,PP↵] RAM 上のパラメータを最大 16 個まで書き換えます。NN は書き換え開始位置です。AA~PP は-32768~32767 の範囲内です。

p4,NN↵] NN は 0~23 の範囲です。符号付 4 バイトのデータ群を NN 目から 8 個表示します(10 進形式)。

p4,NN,AA,・・・,HH↵] RAM 上のパラメータを最大 8 個まで書き換えます。NN は書き換え開始位置です。AA~HH は-2147483648~2147483647 の範囲内です。

(4)PWM 調光

sin 関数により周期的に明滅させています。下記コマンドにより周期や明滅の範囲を変更できます。

- p, c, w, g ↵
 - p : 周期を ms 単位で指定します。初期値 2000。
 - c : 明るさの中心値を%単位で指定します。初期値 50。
 - w : 明るさの変化の幅を%単位で指定します。初期値 50。
 - g : 変化率の係数を 1000 以上の数値で指定します。初期値 2500。

(5)I2C EEPROM 制御

文字列の書き込みおよび読み出しを行います。

- iw n,aaa ↵
 - n : 書き込み開始アドレスを 10 進数または 16 進数指定します。最大は 130,816 (0x1ff00)です。
 - aaa : 文字列を 240 バイト以内で記述します。EEPROM には文字列および終端用ヌル文字を書き込みます。
 - ir n ↵
 - n : 読み出し開始アドレスを 10 進数または 16 進数指定します。最大は 130,816 (0x1ff00)です。読み出した文字列を表示します。
- I2C メモリ基板を接続していなくて応答がない場合は w を 2 回押してください。

4. ビルド方法

4.1. 既存プロジェクトを使用する場合

- (1) 「プロジェクトを開く」で RH850_can5_tp.mtpj ファイルを指定してください。
- (2) ソースを閲覧／編集する場合は、下記設定を確認してください。

①エディタ設定

「ツール」→「オプション」→の

「全般」→「表示」→「タブの桁数」は 4

「テキストエディタ」→「エンコーディング」のデフォルトはシフト JIS

4.2. 新規にプロジェクトを作成する場合

(1)新規プロジェクト作成画面

マイクロコントローラとして RH850/F1KM の R7F701690(LQFP64)を選択してください。

(2)CC-RH(ビルドツール)のプロパティ設定

①共通オプション

- a) コンパイルの追加のインクルードパスとして下記3つを追加してください。

SRC

..¥COMMON

.

- b) レジスタ・モード→「ep レジスタの扱い」を「プロジェクト全体で固定する」にします。

②コンパイル・オプション

a)出力コード

- ・データのデフォルト・セクションを **ep_disp16** にします。
- ・定数のデフォルト・セクションを **zconst** にします。

③リンク・オプション

a)セクション

- ・セクションの開始アドレスを以下にします。

```
RESET/%ResetVectorPE1%,.zconst,.text,.const,.edata,.data/0000200,.stack.bss,EIINTT  
BL.bss,.ebss,.edata.R,.bss,.data.R/FEDE0000
```

- ・ROM から RAM へマッピングするセクションを以下にします。

.data=.data.R

.edata=.edata.R

(3)E1(LPD)もしくは E2(LPD)のプロパティ設定

①接続用設定

- ・メイン・クロック周波数 **16.00**

②フラッシュ・オプション設定(「デバッグ」→「デバッグ・ツールへ接続」の後)

- ・OPBT0 **BA2AFDCF**

(4)ファイル

- ①デフォルトで生成された **boot.asm**, **cstart.asm**, **main.c** を「プロジェクトから外し」ます。**iodef.h** はそのまま残します。

②ファイル指定

「ファイル」→「追加」→「新しいカテゴリを作成」で、**SRC** と **COMMON** を作成します。**COMMON** および **SRC** フォルダの下のファイルをすべて登録します。

4.3. ビルド、書き込み、デバッグ

(1) ビルドのみ行う場合

「ビルド」 → 「ビルド・プロジェクト」 を選択します。

(2) ビルドおよび書き込み／実行を行う場合

① E1 または E2 エミュレータを接続していることを確認してください。

② 「デバッグ」 → 「ビルド&デバッグ・ツールヘダウンロード」 を選択します。

ダウンロード後 **main** 関数の最初で止まります。

リセット位置で止めたい場合は、「デバッグ・ツール」のプロパティを開き、

「ダウンロード・ファイル設定」 → 「デバッグ情報」 → 「指定シンボル」を

`__reset` に変更します。これにより **main** 関数前に実行する `init.c` や `device.c` で

ブレークをかけることができます。

③ 「デバッグ」 → 「実行」 で動作を開始します。

5. プログラム詳細

5.1. マイコン内のリソース割り当て

リソースの割り当ては、device.h に記述してあります。

5.1.1. クロック周波数

(1)MainOSC	16MHz
(2)CPU クロック	120MHz
(3)PPLLCLK	80MHz
(4) PPLLCLK2	40MHz
(5)内蔵低速発振	240kHz

5.1.2. 周辺クロック入力設定

(1)ADC	PPLLCLK2
(2)UART	PPLLCLK2
(3)CAN	PPLLCLK
(4)CSI	PPLLCLK
(5)I2C	PPLLCLK2
(6)TAUD	PPLLCLK → CK0,CK1:不使用, CK2:20MHz, CK3:1MHz

5.1.3. シリアル I/F

(1)RLIN30	UART 用 (USB-シリアル変換用)
(2)RLIN31	RS-485 用
(3)CSIH	SPI 接続 LCD 選択時に使用
(4)CSIG	不使用
(5)I2C0	EEPROM 接続 (JP11 経由)
(6)I2C1	不使用
(7)CAN0	CAN ch0 用
(8)CAN1	CAN ch1 用
(9)CAN2	不使用 (CN2 コネクタ接続)

5.1.4. タイマ

(1)TAUD00ch0~1	不使用
(2)TAUD00ch2	定常クロック出力 (JP11 経由)
(3)TAUD00ch3	スケジューラで使用 (割込みのみ使用)
(4)TAUD00ch4~7	不使用
(5)TAUD00ch8~9	ブザー用
(6)TAUD00ch10~11	PWM 調光 (JP11 経由)
(7)TAUD00ch12~13	不使用
(8)TAUD00ch14~15	定周期タイマ用
(9)TAUJ	全チャンネル不使用
(10)WDTA	ウォッチドッグのタイムアウト約 1 秒。
(11)OSTM	不使用
(12)RTC	不使用
(13)ENCA	不使用

5.1.5. メモリ消費

(1)FROM	194KB (漢字フォント 157KB を含む)
(2)RAM	8.9KB (内スタック 4KB, ベクターテーブル 2KB)
(3)データフラッシュ	本版では不使用

5.1.6. ポート

(1)RS-485	TX: P10_12 / RLIN31TX, RX: P10_11 / RLIN31RX, EN: P10_8
(2)CAN0	TX: P10_1 / CAN0TX, RX: P10_0 / CAN0RX
(3)CAN1	TX: P10_7 / CAN1TX, RX: P10_6 / CAN1RX
(4)UART	TX: P10_10 / RLIN30TX, RX: P10_9 / RLIN30RX
(5)LED1~4	P9_0, P9_1, P9_2, P9_3
(6)アナログ入力	CN1-3: AP0_0 / ADCA0I0, CN1-4: AP0_1 / ADCA0I1, CN1-5: AP0_2 / ADCA0I2, CN1-6: AP0_3 / ADCA0I3, CN1-7(RV1): AP0_4 / ADCA0I4, CN1-8(RV2): AP0_5 / ADCA0I5, CN1-9(DIPSW5-1): AP0_6 / ADCA0I6, CN1-10(DIPSW5-2): AP0_7 / ADCA0I7
(7)DIP SW5-1~4	AP0_6, AP0_7, AP0_8, AP0_9

(8)クロック出力	JP11-2: P0_0 / TAUD0O2
(9)PWM 出力	JP11-7: P10_5 / TAUD0O11
(10)I2C	JP11-5(SDA): P10_2 / RIIC0SDA, JP11-3(SCL): P10_3 / RIIC0SCL
(11)LCD 出力	RESET: P8_6, CS2: P8_5, CS1: P8_4, D7: P8_3, D6: P8_1, D5: P8_0, D4: P8_2, D3: P0_6, D2: P0_5, D1: P0_4, D0: P0_1, E(SCK): P0_2 / CSIH0SC, DI(DO): P0_3 / CSIH0SO
(12)ブザー出力	P10_4 / TAUD0O9
(13)タクト SW1~4	P10_13, P10_14, JP0_3, JP0_4。内蔵プルアップ ON。

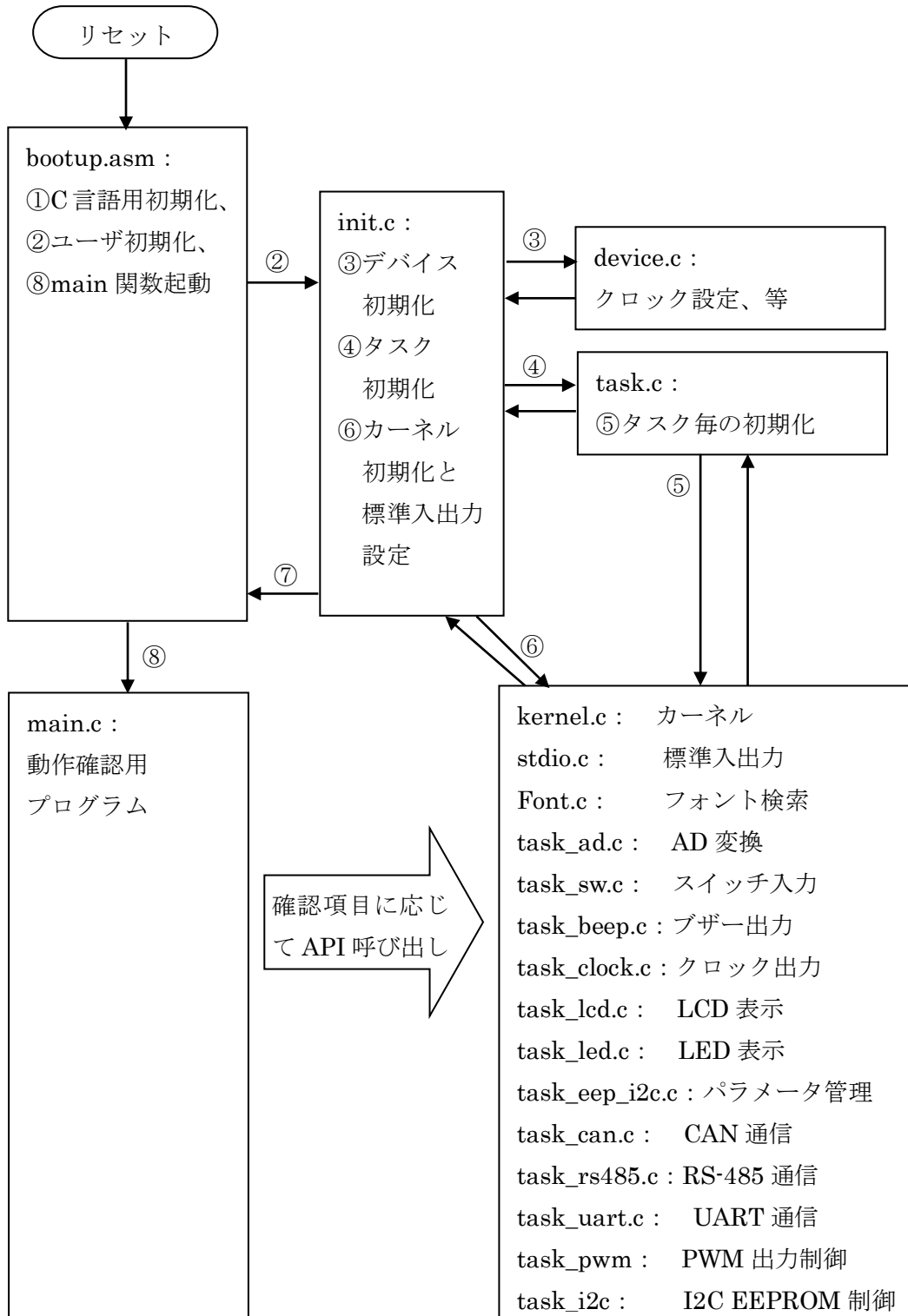
5.1.7. 割込み、優先度

優先度 0~7	未使用
優先度 8	UART/RS485 受信 (ch35: ICRLIN30UR1, ch122: ICRLIN31UR1)
優先度 9	未使用
優先度 10	I2C 受信 (ch78: ICRIIC0RI)
優先度 11	未使用
優先度 12	一般の割込み 定周期タイマ (ch15: ICTAUD0I14, ch55: ICTAUD0I15) UART 送信/エラー (ch34: ICRLIN30UR0, ch36: ICRLIN30UR2) RS-485 送信 (ch121: ICRLIN31UR0) I2C 送信/バッファ空き/エラー (ch79: ICRIIC0TEI, ch76: ICRIIC0TI, ch77: ICRIIC0EE)
優先度 13	未使用
優先度 14	スケジューラ (ch49: ICTAUD0I3) およびタスク処理
優先度 15	未使用

参考：割込みベクタは RAM 上に保持しており、動的変更可能です。

5.2. 動作順序

リセット後の動作順序を丸番号で示します。



5.3. 主な関数の使用方法

5.3.1. 標準入出力関数 (stdio.h)

UART, RS-485, LCD については、C 言語のストリーム型標準入出力関数が使用できます。

オープン／クローズ	fopen, fclose (注：バイナリモードは使用できません)
1 バイト入出力	putchar, getchar, putc, getc, fputc, fgetc
文字列入出力	puts, gets, fputs, fgets
書式付き入出力	printf, fprintf, scanf (fscanf は非対応)

UART, RS-485, LCD を使用可能とするため、init.c の中で以下の登録を行っています。

```
STDIO_set_driver("COM1:", UART_open);
STDIO_set_driver("COM3:", RS485_open);
STDIO_set_driver("LCD:", LCD_open);
```

さらに標準入力(stdin)と標準出力(stdout)を init.c の中で開いています。

```
stdin = fopen("COM1:", "r");
stdout = fopen("COM1:", "w");
```

したがって、main 関数では fopen することなく UART が使用できます。

5.3.2. アナログ入力取得関数 (task_ad.h)

10ms 周期で AD 変換を繰返しており、直近 10 回分の加算値を次の関数で取得できます。

```
uint AD_value(uint ch);
ch      device.h で定義されている論理チャンネルを指定します。
        RV1,RV2,CN1-3～CN1-6 に対して、AD_VOLM1, AD_VOLM2,
        AD_AUX1～AD_AUX4 を指定します。
```

結果(符号なし 2 バイト整数)を 10 で割ると移動平均値が得られます。

5.3.3. スイッチ入力関数 (task_sw.h)

現在の状態を次の関数で取得できます。

```
int SW_value(void);
```

結果の最下位ビットから、SW1～4, DIP SW5-1～4 の順に各ビットが ON/OFF 状態を示します。ON が 1 です。ビット位置を示すマスク値は次のように定義しています。

```
#define PUSH1_MASK          0x01
#define PUSH2_MASK          0x02
#define PUSH3_MASK          0x04
#define PUSH4_MASK          0x08
#define DIPSW1_MASK         0x10
```

```
#define DIPSW2_MASK          0x20
#define DIPSW3_MASK          0x40
#define DIPSW4_MASK          0x80
```

例えば DIPSW5-1 の状態を知るには、次のように記述します。

```
j = SW_value();
if (j & DIPSW1_MASK) /*ON 状態*/;
else /*OFF 状態*/;
```

タクト SW が押されたかどうかは、次の関数で取得できます。

```
int SW_geton(int mask);
    mask    前述のマスク値
```

結果が非ゼロならば押下があったことを示します。この関数を呼び出すと、当該 SW の押下検出をクリアします。従って、次また押されるまでは結果がゼロになります。

5.3.4. ブザー出力関数 (task_beep.h)

次の関数で、ブザーを一定時間鳴らすことができます。

```
void BEEP_out(int freq , int time);
    freq    周波数を Hz 単位で指定します (おおむね 100~10,000Hz 範囲)。
    time    鳴動時間を 10ms 単位数で指定します。
            0 を指定すると鳴動途中でも強制停止します。
```

メロディ・データがある場合は、次の関数で演奏ができます。

```
int BEEP_play(ONPU *p, int offset, int tempo, int ichou);
    p        メロディ・データへのポインタを指定します。
    offset    演奏開始位置を指定します。通常は 0(最初から)を指定します。
    tempo     演奏速度を 1 分間の四分音符の個数で指定します。
    ichou     音程を変更する場合は、半音単位で±の数値を指定します。
```

結果が 0 以上なら演奏を開始しています。演奏中であれば結果が EOF となります。

メロディ演奏を停止したい場合は、次の関数を使用します。

```
int BEEP_stop(void);
```

結果として演奏停止位置が得られます。すでに演奏を終わっている場合は EOF が得られます。

5.3.5. クロック出力関数 (task_clock.h)

指定した周波数の方形波クロックを出力できます。

```
void CLOCK_output(int src, int divsel, int div);
```

src 方形波クロックを出力するかどうかを指定します。

CLOCK_OUT_DISABLE 出力せず

CLOCK_OUT_ENABLE 出力する

divsel 分周入力周波数を指定します。

CLOCK_INPUT_20MHz 20MHz

CLOCK_INPUT_1MHz 1MHz

div 出力周波数の希望値を Hz 単位で指定します。

入力周波数/2/div が割り切れない場合は、実際の出力周波数がずれま
す。

5.3.6. LED 点滅指示関数 (task_led.h)

LED の点灯、消灯、点滅を次の関数で指示できます。

```
void LED_blink(int ch, int mode, int ontime, int offtime);
```

ch 対象の LED1~4 に対して LED_CH0~4 を記述します。

mode LED_MODE_ON(点灯)、LED_MODE_OFF(消灯)、
LED_MODE_LOOP(点滅継続)、もしくは点滅回数(1~253)を指定します。

ontime LED 点滅の場合、点灯時間を 10ms 単位数で指定します。

offtime LED 点滅の場合、消灯時間を 10ms 単位数で指定します。

5.3.7. CAN 通信関数 (task_can.h)

CAN 通信用のデータは、送信/受信とも次の構造体に格納します。

```
typedef struct {  
    int          id:16; // ID (有効範囲は下位 11 ビット)  
    int          port:8; // 0(CAN0)または 1(CAN1)  
    int          dlc:8; // データ長(0~8)  
    uchar       data[8]; // データ 8 バイト(dlc 分のみ有効)  
} CAN_PACKET;
```


送信する場合は、構造体にデータを格納後、次の関数を呼び出します。

```
int CAN_send(CAN_PACKET *pp);
```

pp 構造体へのポインタ

結果が EOF の場合はバッファ満杯のため送信予約が出来なかったことを示します。

結果が 0 以上の場合は送信予約が成功し、空きバッファ数を示します。送信バッファは ch 毎に 32 パケット分あります。

受信する場合は、格納先構造体を用意し、次の関数を呼び出します。

```
int CAN_receive(CAN_PACKET *pp);
```

pp 構造体へのポインタ

結果が EOF の場合は CAN0, CAN1 とも受信パケットが無かったことを示します。

結果が 0 以上の場合は受信パケットの内容が指定構造体へ格納され、さらに引き取りが必要な受信パケット数を示します (CAN0, CAN1 の合計数)。受信バッファは ch 毎に 32 パケットあり、本関数を呼び出すごとに CAN0, CAN1 の受信パケットを交互に引き取ります。片方の ch しか受信が無い場合は片方だけ連続で引き取ります。

5.3.8. PWM 出力関数 (task_pwm.h)

PWM 出力を開始するには次の関数で指示します。PWM 出力 0%で開始します。

```
int PWM_start(int freq, int nega);
```

freq PWM 周波数を Hz 単位で指定します(750~180,000 範囲)。

nega 出力極性を指定します(1=正, -1=負)。

戻り値は PWM100%時のカウント値になります。

PWM デューティは次の関数で指示します。

```
void PWM_ratio(float r);
```

r PWM 比率を 0.0(0%)~1.0(100%)の範囲で指定します。

5.3.9. I2C EEPROM 制御関数 (task_i2c.h)

EEPROM の I2C アドレス(7bit)とページサイズを device.h 内で定義してあります。

```
#define I2C_ADDR_EEPROM(0x50) // EEPROM の I2C アドレス(7bit).
```

```
#define I2C_PAGE_SIZE 256 // EEPROM ページサイズ.
```

次の関数でデータ書き込みを開始します。ページ単位の分割は自動で行います。

```
int I2C_EEP_write(int addr, char *buff, int num);
```

addr 書き込み開始アドレス(byte 単位)

buff 書き込みデータへのポインタ

num 書き込みバイト数.

戻値 正常に受け付けられると I2C_STATUS_ACCEPT が戻ります。

次の関数でデータ読み出しを開始します。

```
int I2C_EEP_read(int addr, char *buff, int num);
```

addr 読み出し開始アドレス(byte 単位)

buff 読み出しデータの格納先ポインタ

num 読み出しバイト数

戻値 正常に受け付けられると I2C_STATUS_ACCEPT が戻ります。

次の関数で書き込みや読み出しの完了を判定します。

```
int I2C_status(void);
```

戻値 完了すると I2C_STATUS_READY が戻ります。

5.4. カーネル仕様

TSOS カーネル説明書 (RH850 版) を参照ください。

改訂履歴

版	日付	記事
初版	2023/11/26	新規作成