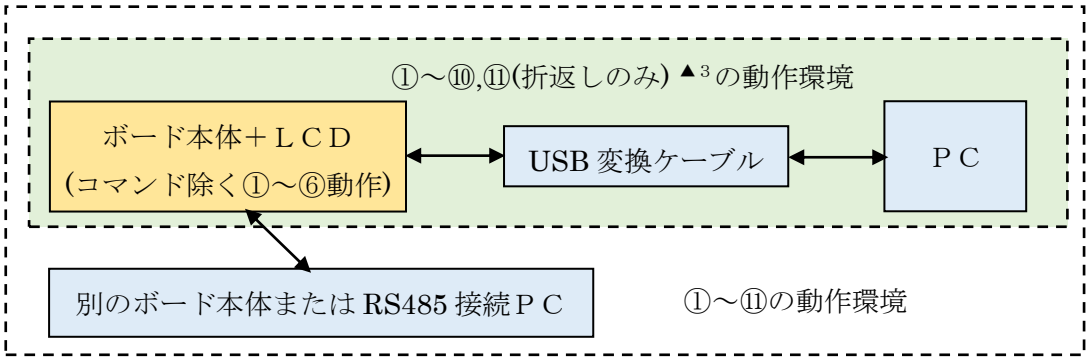
SAMC21-can5 実験キット用
テストプログラム説明書(第 4 版)

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、SAMC21-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、
4 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(16.384kHz)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：パラメータの保存と復元を確認できます。
⑩CAN 通信：２ｃｈ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▲3
⑫拡張基板接続により PWM 調光、I2C EEPROM アクセスを確認できます▲4



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)。

```
[SAMC21_can5_test.X]
 ├ SAMC21_can5_test.hex   ビルド済み実行コード（出荷時にボード書込み済み）
 ├ Makefile               ビルド用プロジェクト・ファイル
 ├ [nbproject]            ビルド用プロジェクト・フォルダ
 └ [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、~~SFR マクロ定義~~▲2）
    ├ device.c            デバイス初期化関数（クロック初期化など）
    ├ init.c              実行環境初期化（main 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]                  ビルド用ソース・コード（機種共通部）
    ├ board.h             ボード識別情報
    ├ common.h            共通定義（変数型、定数、組込み関数、SFR マクロ▲2）
    ├ Font.h/c/inc        漢字フォント検索関数／フォント・データ
    ├ 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(オーバーラン修正▲4)
    ├ task_led.h/c        LED 表示タスク
    ├ task_eep.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 通信関数／ハンドラ
    ├ task_can.h/c        UART 通信関数／ハンドラ
    ├ task_pwm.h/c        PWM 出力関数▲4
    └ task_i2c.h/c        I2C EEPROM 制御タスク▲4
```

3. テスト方法

3.1. ソフトウエアの準備

(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マイクロチップ社の MPLAB X IDE(v5.30 以降)および XC32 コンパイラ(v2.40 以降)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)。最新版以外の XIDE やコンパイラはアーカイブからダウンロードできます。

3.2. ハードウエアの準備

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

JP1：DC ジャック側
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）▲3
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▲4

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▲3

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▲4

拡張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プログラム書き込みの場合

デバッガの USB ケーブルを PC に接続し､次にデバッガ本体を CN7 に接続します。

X IDE を起動し、該当プロジェクトを開き、RUN ボタンで書き込みします。

RUN ボタンによる書き込みの場合は､デバッガは差したままでも外しても良いです。

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、32.768kHz 水晶発振から逓倍によって生成しています。水晶発振の 2 分周出力（16.384kHz）を JP11-7(ver1.31 まで)／JP11-2(ver4.以降)[▲4]に出力しています。

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 256 バイトです。
表示例(コマンド文字含む)：stack consumption = 918 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 CN1-8,9 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

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。

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▲3

(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 コンパイラ用）Font_XC32 を使用します。半角は英数のみで半角カナは表示できません。

(3)データ・フラッシュ読み書き

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 バイトをデータ・フラッシュに書き込みます。格納領域はデータ・フラッシュ領域の先頭 2048 バイトを 8 分割し、巡回的に書き込みます。

pl⏎　パラメータを復元します。巡回書き込みされたデータ・フラッシュの中から最新の 256 バイトを RAM 上に読み込みます。ただし 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 4～254 の範囲です(偶数)。RAM 上のパラメータを符号付 2 バイトのデータ群とみなして NN バイト目から 8 個表示します(10 進形式)。

p2,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 は書き換え開始バイト位置です。AA～HH は各々-32768～32767 の範囲内です。

p4,NN⏎　NN は 4～252 の範囲です(4 刻み)。RAM 上のパラメータを符号付 4 バイトのデータ群とみなして NN バイト目から 4 個表示します(10 進形式)。

p4,NN,AA,・・,DD⏎　RAM 上のパラメータを最大 4 個まで書き換えます。NN は書き換え開始バイト位置です。AA～DD は各々-2147483648～2147483647 の範囲内です。

(4)PWM 調光[▲4]

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[▲4]

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。

拡張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 SAMC21_can5_test.X フォルダ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

Tools→Options→Editor→Formatting の
Language：All Languages
Category：Tabs And Indents
において、

・Expand Tabs to Space のチェックを外します
・インデントおよびタブのサイズを 4 にします

②文字コードの設定

Tools → Options → Embedded → General Settings の一番下までスクロールし Default Charset を Shift_JIS にします。

4.2. 新規にプロジェクトを作成する場合

(1)プロジェクトネームを入力する画面で、下の方にある Encoding を
Shift JIS に設定してください。

(2)プロジェクト生成後のプロパティ設定

①xc32-gcc の Option categories = 「Processing and messages」において、

a)Include directories に..¥COMMON;SRC を指定してください。
これを忘れると COMMON フォルダ内の stdio.h を認識できません。

b)Preprocessor macros に __SAMC21J18__を指定したほうが良いようです。
ビルドには影響しませんが、X IDE のエディタでソースを表示したときに赤線(不認識表示)が減る傾向にあります。

②xc32-gcc の Option categories = 「Optimization」において、
optimization-level を 1 にしてください。

③xc32-ld の Heap size および Minimum stack size には 2048 以上を指定します。

(3)Header Files
COMMON および SRC フォルダの下の*.h および*.inc をすべて指定してください。

(4) Source Files
COMMON および SRC フォルダの下の*.c をすべて指定してください。

4.3. ビルド、書き込み、デバッグ

(1)ビルドのみ行う場合
金槌アイコンを押すか Production → Build Project を選択してください。

(2)ビルドおよび書き込み／実行を行う場合
デバッガを接続していることを確認してください。
右三角アイコンを押すとビルド後に書き込み／実行を行います。

(3)デバッグを行う場合

①書き込み後に自動実行するか停止するかを選択

Tools → Option → Embedded アイコン → Generic Setting タブの一番下までスクロールし、下から 2 番目の Default startup を選択します。

Run　　：停止せず実行します。

Halt at Main：main.c の main 関数先頭で停止します。

Halt at Reset vector：startup_atsamc21j18a.c（XC32 デフォルトコード）のリセット位置で停止します。init.c や device.c の動作を確認をしたい場合はこちらを選択します。

②Debug → Debug project を選択します。

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. 発振器

(1)内部高速発振	24MHz 出力に設定。CPU クロック用。
(2)外部高速発振	不使用。
(3)内部低速発振	32.768kHz。キャリブレーション実施。 デフォルトでは不使用。クロック出力として選択可能。
(4)外部低速発振	32.768kHz。PLL リファレンスとして使用。
(5)PLL	発振周波数 48MHz、出力 24MHz。

5.1.2. クロックジェネレータ

(1)GCLK0	24MHz。CPU クロック用。内蔵高速発振をそのまま出力。
(2)GCLK1	クロック出力用(ver4 以降)、ソースと分周比は API で指定▲4
(3)GCLK2	1MHz。PLL 出力分周。タイマ、AD 変換で使用。
(4)GCLK3	12MHz。PLL 出力分周。UART 用。
(5)GCLK4	8MHz。PLL 出力分周。CAN 用。
(6)GCLK5	12MHz。PLL 出力分周。ブザーの音階生成用。
(7)GCLK6	不使用。
(8)GCLK7	不使用(ver1.31 までクロック出力用)▲4
(9)GCLK8	不使用。

5.1.3. シリアル I/F

(1)SERCOM0	SPI 接続 LCD 用▲4
(2)SERCOM1	UART 用。
(3)SERCOM2	I2C EEPROM 用▲4
(4)SERCOM3	RS-485 用。
(5)SERCOM4	不使用。
(6)SERCOM5	使用禁止（割込みリソースをタスク管理用として使用）。
(7)CAN0	CAN ch0 用。通信速度 500kbps。
(8)CAN1	CAN ch1 用。通信速度 500kbps。

5.1.4. タイマ

(1)TC0	PWM 出力用(ver4 以降)、GCLK5 使用▲4
(2)TC1	不使用（LED4 の PWM 調光に予約）
(3)TC2	5ms 定周期タイマ用(ver4 以降)、GCLK2 使用▲4
(4)TC3	不使用。
(5)TC4	ブザー用方形波出力。GCLK5 使用。
(6)TCC0	不使用。
(7)TCC1	不使用。
(8)TCC2	不使用。
(9)WDT	ウォッチドッグのタイムアウト約 1 秒。

5.1.5. メモリ消費

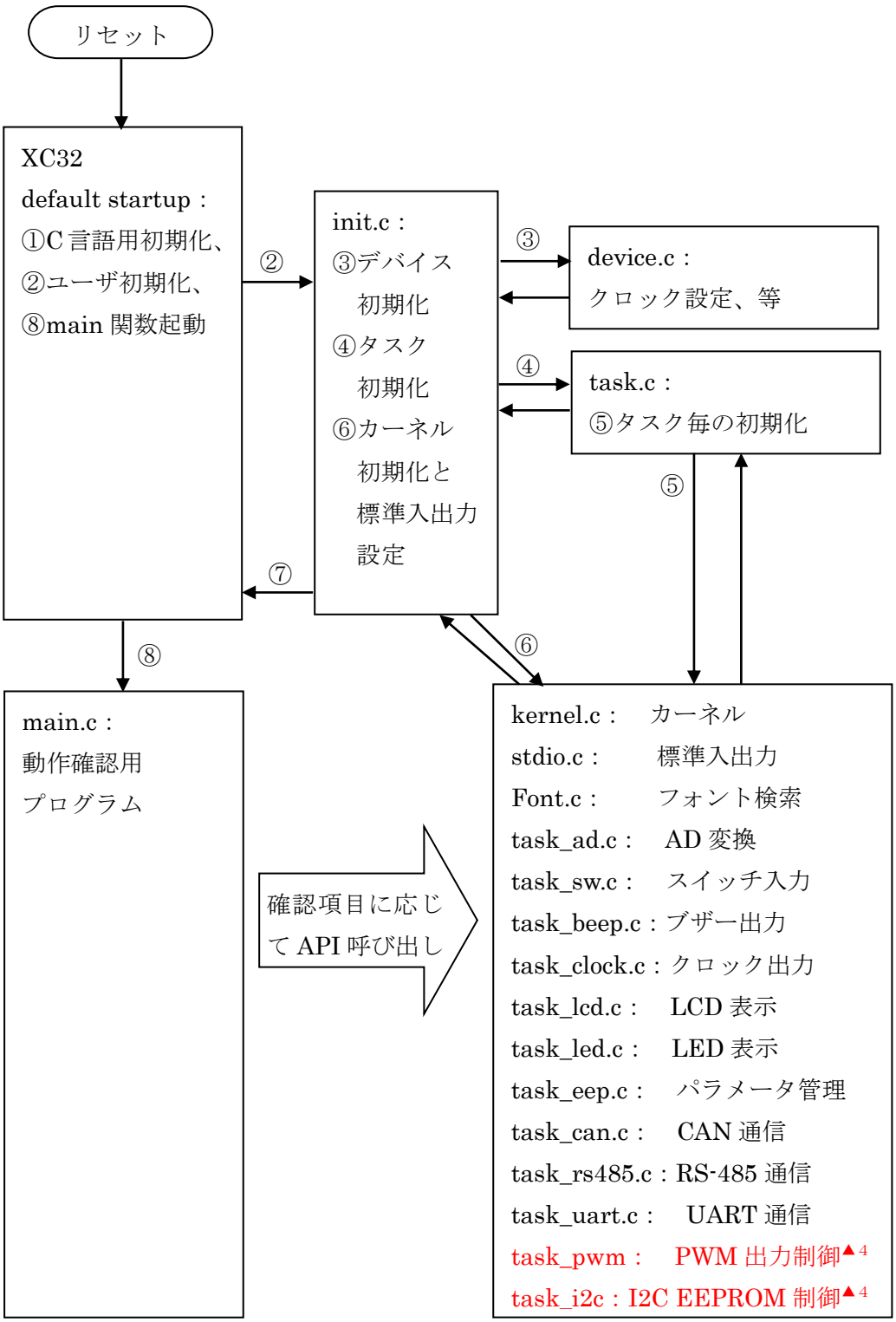
(1)FROM	196KB(XC32 ver4.00) ▲4。漢字フォント 157KB を含む。
(2)RAM	5.9KB(XC32 ver4.00) ▲4。スタック除く。
(3)データフラッシュ	2KB。256B データ x8 巡回書き込み。

5.1.6. ポート

(1)タクト SW1～4	PB06, PB07, PB05, PB04。プルアップ有入力。
(2)DIP SW5-1～4	PB03, PB02, PB01, PB00。プルアップ有入力。
(3)LED1～4	PA10, PA11, PB10, PB11。
(4)クロック出力	PA15 / GCLK_IO[1] ▲4
(5)ブザー出力	PA18 / TC4.WO[0]。
(6)アナログ入力	CN1-7: PA04 / ADC0.AIN[4], CN1-8: PA05 / ADC0.AIN[5], RV1: PA06 / ADC0.AIN[6], RV2: PA07 / ADC0.AIN[7],
(7)UART	TX: PA16 / SERCOM1.PAD[0], RX: PA17 / SERCOM1.PAD[1]
(8)RS-485	TX: PA22 / SERCOM3.PAD[0], RX: PA23 / SERCOM3.PAD[1], EN: PA24 / SERCOM3.PAD[2]
(9)CAN0	TX: PB22 / CAN0.TX, RX: PB23 / CAN0.RX
(10)CAN1	TX: PB14 / CAN1.TX, RX: PB15 / CAN1.RX
(11)LCD 出力	RESET: PB31, CS2: PB30, CS1: PA28, D7: PA27, D6: PA25, D5: PA21, D4: PA20, D3: PB17, D2: PB16, D1: PA19, D0: PB12, E: PA09, DI: PA08
(12)電源出力▲4	PA14。ハイパワー設定。拡張基板用。
(13)PWM 出力▲4	PB13 / TC0.WO[1]
(14)I2C▲4	SDA:PA12/SERCOM2.PAD[0], SCL:PA13/SERCOM2.PAD[1]

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, fprinf, scanf, fscanf

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　　RV1,RV2,CN1-7,CN1-8 に対して、AD_VOLM1, AD_VOLM2, AD_AUX1, AD_AUX2 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）

指定した発振器のクロックを指定分周で出力できます。

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

src	発振器の番号を指定します。main 関数のように device.h または kernel.h がインクルードされていれば、_src(発振器名)で指定することが可能です。発振器名は次の通りです。
	OSC48M 内部高速発振
	XOSC 外部高速発振
	OSC32K 内部低速発振
	XOSC32K 外部低速発振
	FDPLL96M PLL
	OSCULP32K 内部超低電力発振
divsel	0 を指定すると分周比が 1/div となります。
	1 を指定すると分周比が 1/2^(div + 1)となります。
div	divsel が 0 の場合は、1～31、1 の場合は 0～8 を指定します。

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, CA1 の合計数）。受信バッファは ch 毎に 64 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）▲4

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）▲4

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

```
#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（SAM 版）を参照ください。

改訂履歴

版	日付	記事
初版	2021/12/18	新規作成
2 版	2022/1/14	2.テストプログラム一式：device.h 内の SFR マクロ定義を common.h へ移動。
3 版	2022/1/28	・RS-485 の折返し通信について記述追加（1 項, 3.2 項, 3.3.4 項） ・メモリ消費量を XC32 ver4.00 の値に更新(5.1.5 項)
4 版	2022/5/17	・PWM 出力と I2C EEPROM 制御を追加(1 項、2 項、3.2 項、3.3.4 項、5.1.6 項、5.2 項、5.3.8 項、5.3.9 項) ・task_lcd のソース修正(初期化時と改行時のオーバーラン抑制) ・SPI 接続 LCD 基板に関して追記(3.2 項) ・クロック出力ポート変更、基本周期タイマ変更 (3.3.2 項、 5.1.4 項) ・メモリ消費量を更新(5.1.5 項)