mruby/cのRX210 StarterKit(Renesas)へのポーティング手順について

2018.6.10

しまねソフト研究開発センター

1.はじめに

mruby/cを Renesa 社製1chip CISC マイコン RX210の StarterKit 上で動作させるための手順を記述する。

2.使用機材及び開発環境

2.1 使用機材

StarterKit R0K505210S003BE

windows 8.1

2.2 開発環境

cs+ for CC V6.01.00 (Renesas 社製 統合開発環境フレームワーク) cc-rx V.2.08.00 (Renesas 社製 c 言語コンパイラ)

e2 studio Version 6.2.0 (eclipse ベースの統合開発環境) gcc-4.8.4.201701-GNURX-ELF (Renesas 社公式の gcc)

3.開発環境の構築方法

3.1 cs+ for CC 構築

1.ダウンロードサイト

下記 URL のダウンロードから無償評価版を検索し、ダウンロードする。

ダウンロードに際しては、Renesas への ID 登録が必要となる。

https://www.renesas.com/ja-jp/products/software-tools/tools/ide/csplus.html

直接リンクは、下記であるが、バージョン UP 等により変更になる可能性が高いので上記 URLを推奨。

https://www.renesas.com/ja-jp/software/D4000575.html

2.インストール

1にてダウンロードした「CSPlus_CC_Package_V60100.EXE」を実行する事により、統合開発環境及びcコンパイ ラがインストールされる。

このダウンロードファイルでは、RXの開発環境の他に、RH850とRL78の開発環境もインストール出来るが、 今回のポーティングには不要であるので、インストールしない。

なお、「Microsoft .NET Framework 4.5.2」と「Languege Pack 言語パック(日本語)」「Microsoft Visual C++ 2010 SP1 ランタイム」が入っていない場合には、Microsoft 社のサイトからダウンロードしてインストールを行う。これらのインストールに際して、再起動を要求されるため、再起動可能な状況でインストールを行う事を推奨する。

3.2 e2 studio 構築

1.ダウンロードサイト

下記 URL のダウンロードから無償評価版を検索し、ダウンロードする。

ダウンロードに際しては、Renesas への ID 登録が必要となる。

https://www.renesas.com/ja-jp/products/software-tools/tools/ide/e2studio.html

直接リンクは、下記であるが、バージョン UP 等により変更になる可能性が高いので上記 URL を推奨。 https://www.renesas.com/ja-jp/software/D4000675.html c コンパイラに関しては、下記サイトからダウンロードする。 ダウンロード時には、本サイトに ID 登録する 必要が有る。 https://gcc-renesas.com/ja/rx-download-toolchains/ 今回は、「gcc-4.8.4.201703-GNURX-ELF.exe」を選択した。

また、インストールする際にコードが必要となるので同時に取得しておく。

ログイン後に、上の方に出て来る「GNU Tools」にマウスを置くことにより出て来る「ダッシュボード」をクリックする事により、コードが取得できる



図 3.2.1-1 インストールコード取得

2.インストール

1にてダウンロードした「setup_e2_studio_6_2_0.exe」を実行する事により、統合開発環境がインストールされる。

また、「Microsoft Visual C++ 2008 SP1 Runtime」「Microsoft Visual C++ 2010 SP1 Runtime」「Microsoft Visual C++ 2017 Runtime」などが入っていない場合には、Microsoft 社のサイトからダウンロードしてインストールを行う。これらのインストールに際して、再起動を要求される場合があるため、再起動可能な状況でインストールを行行う事を推奨する。

cコンパイラについては、「gcc-4.8.4.201703-GNURX-ELF.exe」を実行する事によりインストールされる。 なお、e2studio インストールの中でダウンロードしインストールする事も可能である。

4.mruby/c インポート 今回は、最新版の mruby/c バージョン 1.1 を RX210 ヘインポートする。 ダウンロードは、下記より行う。 https://github.com/mrubyc/mrubyc/

また、mruby コンパイラは、バージョン 1.3 以降のが必須なため、合わせて下記よりダウンロードする。

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_download/

なお、mruby/cの動作を確認する方法としては、itocサイトにて公開されているチュートリアル1~3が動く事を確認することとする。

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/

4.1 CS+とcc-rxとRX210 の環境への mruby/c インポート

4.1-1 Chapter01「LED 点滅」を CS+ と cc-rx と RX210 の環境で動かす

itoc チュートリアル「Chapter01「LED 点滅」」を参考にしつつ、ペリフェラル周りの設定を簡単に行う為、Renesas 社提供のドライバ FIT を使う

FIT ダウンロードサイト

https://www.renesas.com/ja-jp/products/software-tools/software-os-middleware-driver/software-package/fit-rx210.html

itoc チュートリアル「Chapter01「LED 点滅」」

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/735

1.CS+で新規プロジェクトを作成する。

「ファイル」->「新規作成」->「新しいプロジェクト」を作成を選択し、新規プロジェクトを作成する。



図 4.1-1.1-1 新規プロジェクト作成

2.「プロジェクト作成」画面が出るので、マイクロコントローラ欄に「RX」使用するマイクロコントローラ欄に「RX210」 「R5F5210BBxFP(100pin)」

品種名:R5F5210BBxFP

内蔵 ROM サイズ[K バイト]:1024 内蔵 RAM サイズ[バイト]:98304 追加情報:Package=PLQP0100KB-A

プロジェクトの種類欄に 「アプリケーション(CC-RX)」 プロジェクト名欄に 「FlashLED」(参考) とし、「作成」ボタンを押す。

(マイクロコントロー	ラを選択してください)		~			
∋(<u>M</u>):						
食索できます)	アップデート(山)					
			^			
(プロジェクトの種类	」 夏を選択してください)		· ·			
(ここにプロジェクト	の名称を入力してください)					
C:¥WorkSpace¥		~	参照(<u>R</u>)			
✔ プロジェクト名の	Dフォルダを作成する(<u>A</u>)					
(ルの絶対パスが表示さ	れます)					
イル構成を流用する(<u>S</u>)						
(流用元のプロジ:	ェクト・ファイルを入力してください)	~	▶照(<u>₩</u>)			
□ プロジェクト・フォルダ以下の構成ファイルをコピーして流用する(<u>0</u>)						
	(マイクロコントロー ラ(M): 食素できます) (フロジェクトの種類 (ここにフロジェクト ○¥WorkSpace¥] 「フロジェクト名の (ルの絶対パスが表示さ イル構成を流用する(S) (注知二の寸のご	【マイクロコントローラを選択してください) ラ(M): (文元ジェクトの (プロジェクトの種類を選択してください) 【フロジェクトの種類を選択してください) (ここにフロジェクトの名称を入力してください) (ここにフロジェクトの名称を入力してください) (ジークロジェクトの名称を入力してください) (小の絶対パスが表示されます) イル構成を流用する(S)	【マイクロコントローラを選択してください) ラ(M): 変素できます) アップデードU) 【プロジェクトの種類を選択してください) 【プロジェクトの種類を選択してください) 【プロジェクトの種類を選択してください) 【プロジェクトの種類を選択してください) 【プロジェクトの名称を入力してください) 【・ <td< td=""></td<>			

図 4.1-1.2-1 プロジェクト作成 設定前

	プロジュ	□クト作成			×
マイクロコントローラ(<u>T</u>):	RX				¥
使用するマイクロコントローラ()	<u>M</u>):				
🏦 (マイクロコントローラを検索	きできます)	アップデート(山)			
R5F5210ABxLK(R5F5210BbxFB R5F5210BbxFB R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxLJ(R5F5210BbxFB) R5F5210BbxFB R5F5210BbxLJ(R5F	145pin) 144pin) 100pin) 100pin) 145pin)	品種名:R5F5210BExFP 内蔵ROMサイズ[[X]-1-1024 内蔵RAMサイズ[]X-1-198804 20加情報:Package=PLQP0100K	B-A		^
田小町 RX230 プロジェクトの種類(K):	× 7705-22/00	-BX)			
プロジェクト名(N):	FlashLED				
作成場所(上):	C:¥WorkSpace¥		*	参照(<u>R</u>)	
	☑ プロジェクト名の	フォルダを作成する(<u>A</u>)			
C:¥WorkSpace¥FlashLED¥F	lashLED.mtpj				
🔄 既存のプロジェクトのファイル	,構成を流用する(<u>S</u>)				
流用元のプロジェクト(<u>P</u>):	(流用元のプロジェ	クト・ファイルを入力してください)	V	参照(₩)	
🔲 プロジェクト・フォルダ以下の)構成ファイルをコピーし	,て流用する(<u>0</u>)			
	(°E)	成(<u>C</u>) キャンセル		ヘノルプ(<u>H</u>)	

図 4.1-1.2-2 プロジェクト作成 設定後

3.新規プロジェクト「FlashLED」が作成されたので、CS+が正しく動作する事を確認する為、ビルドを行う。

「ビルド」->「ビルド・プロジェクト」を選択し、出力欄に「終了しました(成功:1 プロジェクト, 失敗:0 プロジェクト)」 が出る事を確認する。

	FlashLED - CS	+ for CC - [プロパテ]		_ 🗆 🗙
ファイル(E) 編集(E) 表示(V) プロジェクト(P) ビルド	3) デバッグ(D) ツール(I) ウインドウ(W) ヘルプ(日)			🤿 🍞 🚔
🔅 🎘 スタート(S) 🛛 🔜 🗿 🗄 🗶 🗈 🖄	🝽 🏭 🍓 🖏 flag_tx_finishe 💌 100% 🖃	😽 🌆 Defay Build	- 🔬 🐻 🖣 🦏 🔘 🔘 🗠	🔞 🖙 द्व 🖕 🔏
	🥋 🗗 : 💋 ソリューション一覧(<u>S</u>)			
😳 לםטֹדאָלי.	1 プロパティー			- X
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	マイル アイル アイル アイル アイル アイル アイル アイル マイル アイル アイル アイル アイル アイル アイル アイル アイル アイン アン アン </td <td>FlashL</td> <td>ED.mtpj Space¥FlashLEDWFlashLED.mtpj</td> <td></td>	FlashL	ED.mtpj Space¥FlashLEDWFlashLED.mtpj	
< >>	温 出力 滲 スマート・ブラウザー 201 エラー一覧			
F1 F2 F3	F4 F5 F6	F7 FB	FB FB 全画面表示	FII FIE

図 4.1-1.3-1 テストビルドの成否確認

4.mruby/c VM ソースコードを、プロジェクトへ追加

・mruby/c ソースコードを展開し、src フォルダー内の全ファイルを src フォルダーごとプロジェクトフォルダーに コピーする。

・src フォルダー内の hal_psoc5lp フォルダーを hal にリネームする。

・src フォルダー内の hal_posix フォルダーを消去する。

・「FlashLED」のフォルダーに配置する

・下記手順で、統合環境にプログラムを登録する。

プロジェクトツリーのファイルの下の所に、src フォルダーをドラッグ&ドロップする。



図 4.1-1.4-1 mruby/cソースコピー

下記の「ファイルとフォルダ追加」のダイアログが出るので、「検索するサブフォルダの階層数」を 10 に変更し 「OK」ボタンを押す。

フォルダとファイル追加	×
ファイルの種類(丁): 複数違択が可能です Cソース・ファイル(*cp): *cp; *cc) ヘッダ・ファイル(*h; *hpp: *inc) アセンブラ・ソース・ファイル(*src: *s) ジャンブ・テーブル・ファイル(*imp) シンボル・アドレス・ファイル(*fsy) ライブラリ・ファイル(*lib) オブジェクト・モジュール・ファイル(*obj) リロケータブル・ファイル(*rcl) テキスト・ファイル(*txt)	
検索するサブフォルダの階層数(<u>S</u>):	10
OK キャンセル	ヘルプ(H)

図 4.1.4-2 mruby/cソースコピー

5.mruby/c プログラムの作成及びコンパイル

チュートリアル「Chapter01「LED 点滅」」を参考に mruby/c プログラムの作成及びコンパイルを行い、 「sample1.c」を作成し、「FlashLED」のフォルダーに配置する。

6.FIT の登録

必要な FIT モジュールを登録する。

今回のデモでは、「ボードサポートパッケージ(BSP)」(R01AN1685)及び「I/O ポート(GPIO)」(「R01AN1721」)が必要となる。

組込み方は、アプリケーションノート「R01AN1826」を参考にする。

7.「Flash.c」プログラムの変更

```
自動生成された「Flash.c」にボード初期化ルーチンとハードウエア対応ルーチン、mruby/c 起動ルーチンを組み込む。
```

Flash.c

#include "src/mrubyc.h"
#include "r_gpio_rx_if.h" //Contains prototypes for the GPIO driver

#include "sample1.c"

void main(void);

}

#define MEMORY_SIZE (1024*10)
static uint8_t memory_pool[MEMORY_SIZE];

/*! オンボード SW 現在状態の読み込み */

static void c_sw1_read(mrb_vm *vm, mrb_value *v)

SET_INT_RETURN(SW1); // SW1 define in FIT BSP rskrx210.h

```
//
/*! オンボード LED ON/OFF
```

*/ static void c_led1_write(mrb_vm *vm, mrb_value *v)

if(GET_INT_ARG(1) == 1) {
 LED1 = LED_ON; // LED1 define in FIT BSP rskrx210.h
 else {
 LED1 = LED_OFF;
 }
}

/*! HAL (別途説明します)
*/
int hal_write(int fd, const void *buf, size_t nbytes)
{
 return 0;
}
int hal_flush(int fd)
{
 return 0;
}
void main()
{
 hardware_setup();
 /* Place your initialization/startup code here (e.g. MyInst_Start()) */

mrbc_init(memory_pool, MEMORY_SIZE); mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read); mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write); mrbc_create_task(sample1, 0); mrbc_run(); return 0; }

/* * assert()関数で使われる。それを受けるため */ void abort(void) { LED0 = LED_ON; // LED0 define in FIT BSP rskrx210.h } 8.hal.hの修正

ハードウエアレイヤーのルーチンである hal.h を修正する。

hal.h

#ifndef MRBC_SRC_HAL_H_ #define MRBC_SRC_HAL_H_

#ifdef __cplusplus extern ″C″ { #endif

#ifndef MRBC_NO_TIMER
define hal_init() ((void)0)
define hal_enable_irq() ((void)0)
define hal_disable_irq() ((void)0)
define hal_idle_cpu() ((void)0)

#else // MRBC_NO_TIMER
define hal_init() ((void)0)
define hal_enable_irq() ((void)0)
define hal_disable_irq() ((void)0)
define hal_idle_cpu() (R_BSP_SoftwareDelay(1, BSP_DELAY_MILLISECS), mrbc_tick())

#endif

#ifdef __cplusplus } #endif #endif // ifndef MRBC_HAL_H_ 9.タイマー未使用の宣言

このチャプターでは、busywait(関数は、FIT 提供)を使っている為、それを宣言します。 CC-RX(ビルドツール)上で、マウスを右クリックし、プロパティを選びます。



開いたプロパティのコンパイルオプションタブを選び、マクロ定義の右側の「…」を押し、テキスト編集ダイアログを開きます。

~	CC-RX のプロパティ	2 / - +
	C++ソース・ファイルの言語	C++(-lang=cpp)
⊳	追加のインクルード・パス	追加のインクルード・パス[12]
Þ	システム・インクルード・パス	システム・インクルード・パス[0]
⊳	コンパイル単位の先頭にインクルードするファイル	コンパイル単位の先頭にインクルードするファイル[0]
⊳	マクロ定義	マクロ定義[1] …
	無効化するプリデファインド・マクロ	
	インフォメーションレベル・メッセージ出力を有効にす。	いいえ(-nomessage)
	抑止するインフォメーションレベル・メッセージ番号	
	表示させない警告メッセージ	
	ウォーニング・メッセージをインフォメーションレベルに3	いいえ
	インフォメーション・メッセージをウォーニングレベルに3	いいえ
	インフォメーション、ウォーニング・メッセージをエラーレ	いいえ
	コメント(/* */)のネストを許す	いいえ(-comment=nonest)
	既存プログラムとの互換性をチェックする	いいえ
	入力ブログラムの文字コード	SJISK(-sjis)
⊿	オブジェクト	
	出力ファイル形式	オブジェクト・モジュール・ファイル(-output=obj)
	デバッグ情報を出力する	(t)(-debug)
	プログラム領域のセクション名	P
	定数領域のセクション名	C
	amilife データ額域のセルションタ	n 🔮
ママゴ	加定義 20定義を「<マクロ名>[=<文字列〉]」の形式で、1行 」パイラのオブションーdefineに相当します。	に1つずつ指定します。
ŧ	も通オプション 🔪 コンパイル・オプション 🖉 アセンプ	『ル・オプション 人 リンク・オプション 人 ヘキサ出 カオプション 人 ライブラリ・ジェネレート・… / 🚽

図 4.1-1.8-2 マクロ定義手順 2

ダイアログに「MBRC_NO_TIMER」を登録します。

	テキスト編集	×
テキスト(<u>T</u>):		
MRBC_NO_TIMER		^
<) OK キャンセル ヘルナ()	Ð

図 4.1-1.8-3 マクロ定義手順 3

10.C 言語仕様の設定

mruby/cは、C言語仕様のC99で描かれているので、変更する。

同じく、C-RX(ビルドツール)のプロパティの「コンパイルオプション」タブを選び「C ソース・ファイルの言語」を 「C99」へ変更する

また、「ライブラリジェネレート」タブの「ライブラリ構成」を「C99」へ変更する。

CC-RX のプロパティ		-	. م
▲ Y-X			
のソース・ファイルの言語	C99(-lang=c99)		[
C++ソース・ファイルの言語	C++(-lang=cpp)		
> 追加のインクルード・パス	追加のインクルード・パス[12]		
シ システム・インクルード・パス	システム・インクルード・パス[0]		
> コンパイル単位の先頭にインクルードするファイル	コンパイル単位の先頭にインクルードするファイル[0]		
> マクロ定義	マクロ定義[0]		
無効化するプリデファインド・マクロ			
インフォメーションレベル・メッセージ出力を有効にする	(.)(.)Ž(-nomessage)		
抑止するインフォメーションレベル・メッセージ番号			
表示させない警告メッセージ			
ウォーニング・メッセージをインフォメーションレベルに変更する	いいえ		
インフォメーション・メッセージをウォーニングレベルに変更する	いいえ		
ジース・ファイルの言語 ソース・ファイルの言語を選択します。コンパイラのオブション-lareにす 共通オプション、コンパイル・オプション、アセンブル・オプション 図 4.1-1.9	≝Ulata。 <u>〈リンンウ・オプション 〈ヘキサ出カオプション 〈 ラィブラリ</u> →1 C 99 設定 1	· ヅ゙ı	ネレー
ジース・ファイルの言語 ソース・ファイルの言語を選択します。コンパイラのオブション-lareにす 共通オプション 、エハパイル・オプション 〈アセンブル・オプション 図 4.1-1.9	lillutt。 <u>〈リンウ・オブション 〈ヘキサ出カオブション 〈 ライブラリ</u> ー1 C99 設定1	·ÿı	×レー
ジース・ファイルの言語 ソース・ファイルの言語を選択します。コンパイラのオブション-lareにす 共通オプション 、コンパイル・オプション 〈アセンブル・オプション 図 4.1-1.9	iiiiLitt。 ノリンク・オプション 〈ヘキサ出カオプション 〈 ライブラリ ー1 C99 設定1	· ジェス	ل م
ジース・ファイルの言語 シソース・ファイルの言語を選択します。コンパイラのオブション-lare(これ 共通オブション、コンパイル・オプション、アセンブル・オブション 図 4.1-1.9	iiilidit。 〈リンク・オブション 〈ヘキサ出カオブション 〈 ライブラリ ー1 C99 設定1	· ÿı 2	×ل– م
>>->->+/ルの言語 >>>->・ファイルの言語を選択します。コンパイラのオブション-lareにす 共通オブション、コンパイル・オプション、アセンブル・オプション 図 4.1−1.5 、 CC-RX のプロパティ モード 標準ライブランの使用・構築方法	 (リンウ・オプション 〈ヘキサ出カオプション 〈 ライブラリ・ ー1 C99 設定1 標準ライブラリ・ファイル作成(オブション実更時) 	· ÿı 2	×レー م
シンース・ファイルの言語 シンース・ファイルの言語を選択します。コンパイラのオブション-langに本 共通オプション、コンパイル・オプション、アセンブル・オプション 図 4.1-1.5 、CC-RX のプロパティ モード 標準ライブブリの使用・構築方法。 産業ライブブリ	 (リンク・オブション 〈 ヘキサ出カオプション 〈 ライブラリ・ ー1 C99 設定1 (標準ライブラリ・ファイル作成(オブション実更時) 	• ŸI 7	×レー م
 ジンース・ファイルの言語 ジンース・ファイルの言語を選択します。コンパイラのオブション-lare(これ 共通オブション、コンパイル・オプション (アセンブル・オブション) 図 4.1-1.5 CO-RX のプロパティ モード 標準ライブラリの使用・構築方法 (数 5/75) 	当じはす。 <u>(リンク・オプション (ヘキサ出カオプション (ライブラリ</u>) -1 C99 設定1 標準ライブラリ・ファイル(作成(オプション実更時) C84(-lang=C90) C94(-lang=C90) C84(-lang	• ŸI?	×レー • •
27-7-740の言語 27-7-740の言語を選択します。コンパイラのオブション-lareに れ 共通オプション ユンパイル・オプション (アセンブル・オプション 図 4.1−1.5 CC-RXのプロパティ モード 低単字イブラジョン 574590455 構築対象のライブラジ	Biblidet。 <u>(リンク・オブション (ヘキサ出力オブション (ライブラリ</u>) ー1 C99 設定1 標準ライブラリ・ファイル(作成(オブション変更時) C99(-lang=c99) カスタム(-head=CSub(Option>)	· ジェス	×レー م
 ジーネ・ファイルの言語 ジーネ・ファイルの言語を選択します。コンパイラのオブション-langに本 共通オブション、コンパイル・オプション (アセンブル・オプション) 図 4.1-1.5 CO-RX のプロパティ モード 標準ライブラリの使用・構築方法。 準準ライブラリの使用・構築方法。 準準ライブラリのも応 構築対象のライブラリ ランタイル・ライブラリを有効にする。 	3当します。 <u>〈リンク・オブション〈へキサ出カオプション〈ライブラリ</u> ー1 C99 設定1 標準ライブラリ・ファイル作成(オブション変更時) C99(-lang=09) カスタム(-head=SubOption>) (X)(-head=subOption>) (X)(-head=suntime)	· ŸI 2	×レ ۹
² >>-ス-ファイルの言語 ² >>-ス-ファイルの言語を選択します。コンパイラのオブション-lare(オ	alliukte。 <u>〈リンク・オブション 〈ヘキサ出力オプション 〈 ライブラリ・</u> ー1 C99 設定1 標準ライブラリ・ファイル作成(オブション変更時) C99(-lang=c99) カスタム(-head=StubOption>) (広いく-head=cnutime) C952	• 971 2	×レ ۹
	当じはす。 <u>〈リンク・オブション 〈へキサ出カオプション 〈 ライブラリ</u> ー1 C99 設定1 標準ライブラリ・ファイル作成(オブション変更時) C99(-lang=c99) 丸スなん(-head=SubOption)) (は、(-head=subOption)) は、(-head=subOption)) は、(-head=subOption)) は、(-head=subOption)) は、(-head=subOption)) は、(-head=subOption))	• ÿı>	×レ ۹
ンソース・ファイルの言語 D ソース・ファイルの言語を選択します。コンパイラのオブション・lare(14 共通オプション、コンパイル・オプション、アセンブル・オプション 民国 4.1−1.5 CO-RXのプロパティ モード 標準ライブフの使用・構築方法 福葉サイブフリ フジタイム・アイブフリ フジタイム・アイブフリ フジタイム・アイブフリ マジックイ、アブリ マジック マジック	 (リンウ・オプション 〈 ヘキサ出 カオプション 〈 ライブラリ・ ー1 C99 設定1 (標準ライブラリ・ファイル(作成(オプション変更時)) C99(-lang=c99) カスタム(-head=cstub()) (は、(-head=crutine)) (は、(-head=crutine)) (は、2) 	<u>، کار،</u>	×レ ۹
2>>>->>>->>>>->>>->>>>->>>->>>>>>>>>	当します。 <u> </u>	- ŸI 2	אר-יי ס -
	当します。 <u>〈リンウ・オブション〈ヘキサ出カオプション〈ライブラリ</u> ー1 C99 設定1 儒学ライブラリ・ファイル作成(オブション変更時) C99(-lang=09) カスタム(-head=StubOption>) はい(-head=runtime) しいえ しいえ しいえ しいえ しいえ しいえ しいえ しいえ	· ŸI >	<u>م</u> ار ا
CO-RX のプロパティ モード 電学・ブラリの使用・構築方法 モード 電学・ブラリの使用・構築方法 モード 電学・ブラリの使用・構築方法 モード 電学・ブラリの使用・構築方法 モード 電学・ブラリの使用・構築方法 電学・ブラリの使用・構築方法 電子・ブラリを ロッグブリンを コンパイル・オプション マングリン マングリン マング ロッグ ロッグ ロッ ロッグ ロッグ ロッグ ロッグ ロッ ロッグ ロッ ロッグ ロッ ロッグ ロッグ ロッ ロッ ロッ ロッグ ロッグ ロッグ ロッグ ロッ ロッ ロッグ ロッ ロッグ ロッグ ロッグ ロッグ ロッグ ロッグ ロッ ロッグ ロッグ ロッグ ロッグ ロッグ ロッ ロッグ ロッ ロッグ ロッグ ロッ ロッグ ロッグ ロッ ロ	当します。 <u>(リンク・オブション (ヘキサ出力オプション (ライブラリ</u>) ー1 C99 設定1 標準ライブラリ・ファイル(作成(オブション変更時) C99(-lang=C99) カスタム(-freed=StubOption>) は、(-freed=StubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) は、(-freed=stubOption>) (は、(-freed=stubOption)) (は、(-freed=stubOption))	• ŸI >	<u>- الا</u> م ا

図 4.1-1.9-2 C99 設定 2

, 共通オプション / コンパイル・オプション / アセンブル・オプション / リンク・オプション / ヘキサ出 カオプション 〉 ライブラリ・ジェネレー.... / ▼

11.mruby/c ソースの変更

今回使用している cc-rx は、C99 への対応が完全でないため、「load.c」を変更する。

```
#define BUF_SIZE 256
```

と定義し、

162 行目及び 170 行目にある

char buf[obj_size+1];

を

char buf[BUF_SIZE];

へ変更する

12.クリーンビルド ビルドメニューのクリーンプロジェクトを行い、 その後ビルドプロジェクトを行う。



図 4.1-1.11-1 ビルドメニュー

13.デバッガの設定

E1 デバッガによる H/W デバッグの設定を行う。

・デバッグツールを E1 に設定する。

「デバッグ」->「使用するデバッグツール」->「RX E1(Serial)」



図 4.1-1.12-1 デバッガ設定設定1

・E1 デバッガのプロパティを変更する

プロジェクトツリーから、「RX E1(Serial)(デバッグツール)」の項目でマウスを右クリックし、プロパティを選択する。



図 4.1-1.12-3 デバッガ設定設定 3

14.デバッガによる書き込み及び実行

デバッグメニューから、「デバッグ・ツールへダウンロード」を選び、ボードにプログラムを書き込む。

デバ	ッグ(<u>D) ツール(T) ウインドウ(W)</u> へル	プ(<u>H</u>)
	デバッグ・ソリューション(<u>S</u>)	
D)	デバッグ・ツールへダウンロード(<u>D</u>)	
6	ビルド&デバッグ・ツールへダウンロード(<u>B</u>)	F6
1	リビルド&デバッグ・ツールへダウンロード(<u>V</u>	<u>/</u>)
00	デバッグ・ツールへ接続(<u>C</u>)	
D)	デバッグ・ツールからアップロード(<u>U</u>)	
5	デバッグ・ツールから切断(<u>N</u>)	Shift+F6
	使用するデバッグ・ツール(<u>し</u>)	
	停止(<u>S</u>)	Shift+F5
D	実行(<u>G</u>)	F5
D	ブレークせずに実行(<u>E</u>)	F8
91	ステップ・イン(<u>I</u>)	F11
ÇΞ	ステップ・オーバー(<u>O</u>)	F10
Č _I	リターン・アウト(<u>R</u>)	Shift+F11
μ	CPUリセット(<u>T</u>)	Ctrl+F5
H=1	リスタート(<u>A</u>)	
•	デバッグ・ツールの状態を巻き戻す(<u>W</u>)	Ctrl+F11
	デバッグ・ツールの状態保存(⊻)	
-	図 4.1-1.13-1 デバ	ッグ 1

デバッグメニューから、「実行」を選ぶと、ボード上でプログラムが実行され、LED1 が1秒おきに点滅する。

デノ	゙゙ゕヷ(<u>D) ツール(T) ウインドウ(W)</u> ∧ル	プ(<u>H</u>)
	デバッグ・ソリューション(<u>S</u>)	•
D,	デバッグ・ツールヘダウンロード(<u>D</u>)	
5	ビルド&デバッグ・ツールへダウンロード(<u>B</u>)	F6
-	リビルド&デバッグ・ツールへダウンロード(<u>M</u>	<u>/</u>)
	デバッグ・ツールへ接続(<u>C</u>)	
D*	デバッグ・ツールからアップロード(<u>U</u>)	
*	デバッグ・ツールから切断(<u>N</u>)	Shift+F6
	使用するデバッグ・ツール(<u>し</u>)	•
	停止(<u>S</u>)	Shift+F5
	実行(<u>G</u>)	F5
0	ブレークせずに実行(<u>E</u>)	F8
9 2	ステップ・イン(<u>I</u>)	F11
Ģ≡	ステップ・オーバー(<u>O</u>)	F10
¢_	リターン・アウト(<u>R</u>)	Shift+F11
۴	CPUリセット(<u>T</u>)	Ctrl+F5
₩Ĵ	リスタート(<u>A</u>)	
۲	デバッグ・ツールの状態を巻き戻す(<u>W</u>)	Ctrl+F11
	デバッグ・ツールの状態保存(⊻)	•

図 4.1-1.13-2 デバッグ 2

LED1 が赤く点滅する。なお、上方の緑の LED は、電源ランプで、電源が入っている事を示す。



図 4.1-1.13-3 デバッグ 3

4.1-2 Chapter02「LED 点滅の速さを変える」を CS+ と cc-rx と RX210 の環境で動かす itoc チュートリアル「Chapter02「LED 点滅の速さを変える」」を参考にして、4.1-1 を修正していく。

1.mruby スクリプトの自動コンパイル処理の追加

CC-RX(ビルドツール)上で、マウスを右クリックし、プロパティを選びます。



図 4.1-2.1-1 自動コンパイルスクリプト追加 1

共通オプションタブを選び、「その他」->「ビルド前に実行するコマンド」に「call mrbc.bat」を追加する。 右側の「…」を押し、テキスト編集ダイアログを開きます。

1	০০ প্রাথিচন	2 2 -	+
	使用するコンパイラ・ハッ・	常にインストール済みの最新版 Mana	^
⊳	記錄	\$2.00.0	
4	その他		
	出力メッセージ・フォーマット	%TargetFiles%	
	ビルド・オブション一覧表示フォ	%TargetFiles% : %Program% %Options%	
	ヒルド前に実行するコマンド	ビルド前に実行するコマンド(1)	
⊳	ビルド復に実行するコマンド	ビルド後に実行するコマンドし	
_	その他の追加オフション		~
UUN2行主%P %0	いかでは、 いたのでは、 いたのでは、 いたのでは、 いたので、 いてので、 いてので、 いてので、 いてので、 いてので、 いてので、 いてので、 いてので、 いてので、 い い い い い い い い い い い い い	指定します。 abat"のようにcall命令を使用してください。また、先頭行に"#python"と記述することで かなっマンドとして実行します。 ダの絶力リズに置換します。 対バスに置換します。	
1		^)) ^ () 10 () 10 () 20 () 10	

図 4.1-2.1-2 自動コンパイルスクリプト追加 2

	テキスト編集	×
テキスト(<u>T</u>):		
call mrbc.bat		^
<		>
プレースホルタ(<u>し</u>): プレースホルダ	値	^
ActiveProjectDir ActiveProjectMicomName ActiveProjectName BuildModeName	C¥WorkSpace¥FlashLED R&F\$210BBxFP FlashLED DefaultBuild	~
<	OK	>)(H)

図 4.1-2.1-3 自動コンパイルスクリプト追加 3

2.浮動小数点の利用

itoc チュートリアル通りに、sample1.rb を修正し、4.1-1の11及び13の手順通りにコンパイルし実行する。 すると、LED1が0.2秒間隔で点滅する。

これは、CS+の環境では、デフォルトで浮動小数点を扱う様に設定されているからである。

3. タイマーの利用

itoc チュートリアル通りに、タイマーを利用するように設定を変更する。

タイマーは FIT ライブラリを利用する。

今回追加が必要な FIT モジュールは、「消費電力低減機能(LPC)」(R01AN2769)「コンペアマッチタイマ(CMT)」 (R01AN1856)の 2 つである。

組込み方は、アプリケーションノート「R01AN1826」を参考にする

また、hal.hに割り込み許可、禁止、cpu休止命令を、メインルーチンにタイマーの初期化及び割り込みルーチンを追加する。

hal.h

#ifndef MRBC SRC HAL H #define MRBC SRC HAL H #ifdef __cplusplus extern "C" { #endif #include <machine.h> #ifndef MRBC_NO_TIMER # define hal_init() ((void)0)
define hal_enable_irq() (set_ipl(0))
define hal_disable_irq() (set_ipl(15))
define hal_idle_cour() (weit()) # define hal_idle_cpu() (wait()) #else // MRBC_NO_TIMER #define hal_init() ((void)0) # define hal_enable_irq() ((void)0) # define hal_enable_irq() ((void)0) # define hal_disable_irq() ((void)0) # define hal_idle_cpu() (R_BSP_SoftwareDelay(1, BSP_DELAY_MILLISECS), mrbc_tick()) #endif int hal_write(int fd, const void *buf, int nbytes); int hal flush(int fd); #ifdef __cplusplus

} #endif #endif // ifndef MRBC_HAL_H_

FlashLED.c

```
#include "src/mrubyc.h"
#include "r_gpio_rx_if.h" //Contains prototypes for the GPIO driver
#include "r_lpc_rx_if.h" // Low Power Consumption (LPC)
#include "r_cmt_rx_if.h" // Compair Much Timer (CMT) API
                                         // Low Power Consumption (LPC) API
                                         // Compair Much Timer (CMT) API
#include "sample1.c"
void cmt_cb(void *pdata);
void main(void);
#define MEMORY_SIZE (1024*10)
static uint8_t memory_pool[MEMORY_SIZE];
/*! オンボード SW 現在状態の読み込み
*/
static void c_sw1_read(mrb_vm *vm, mrb_value *v)
  SET INT RETURN(SW1);
                                        // SW1 define in FIT BSP rskrx210.h
}
/*! オンボード LED ON/OFF
*/
static void c_led1_write(mrb_vm *vm, mrb_value *v)
  if( GET_INT_ARG(1) == 1 ) {
   LED1 = LED_ON; // LED1 define in FIT BSP rskrx210.h
 } else {
   LED1 = LED_OFF;
 }
}
/*! HAL (別途説明します)
*/
int hal_write(int fd, const void *buf, size_t nbytes)
  return 0;
int hal_flush(int fd)
  return 0;
}
void main()
  lpc_err_t lpc_request_status;
  uint32_t cmt_hdl;
 hardware setup();
  /* Configure the MCU for the mode. */
// lpc_request_status = R_LPC_LowPowerModeConfigure(LPC_LP_SW_STANDBY); // STOP CLOCK when no work CMT
lpc_request_status = R_LPC_LowPowerModeConfigure(LPC_LP_ALL_MODULE_STOP);
 if (LPC_SUCCESS != lpc_request_status) {
LED0 = LED_ON;
   while (1) {
     nop();
   }
 }
  /* Create a timer tick with callback at a rate of 1000Hz. */ if( ! R\_CMT\_CreatePeriodic(1000, cmt\_cb, \&cmt\_hdl) ) 
   LEDO = LED_ON;
   while (1) {
     nop();
   }
 }
 /* Place your initialization/startup code here (e.g. MyInst_Start()) */
mrbc_init(memory_pool, MEMORY_SIZE);
 mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);
 mrbc_create_task( sample1, 0 );
 mrbc run():
```

```
return 0;
}
/*
                 assert()関数で使われる。それを受けるため
*
*/
void abort(void)
{
 LED0 = LED_ON; // LED0 define in FIT BSP rskrx210.h
}
/*
                  タイマー割り込みルーチンのコールバック
*
*/
void cmt_cb(void *pdata)
ł
         mrbc_tick();
} /* end tmr0_cmia0_is */
```

4. タイマー未使用宣言を撤回

このチャプターでは、タイマーを使う為、それを撤回します。 CC-RX(ビルドツール)上で、マウスを右クリックし、プロパティを選びます。



図 4.1-2.4-1 マクロ定義削除手順 1

開いたプロパティのコンパイルオプションタブを選び、マクロ定義の右側の「…」を押し、テキスト編集ダイアログ を開きます。

	· · · ·		
4	CC-RX のプロパティ		- +
	C++ソース・ファイルの言語	C++(-lang=cpp)	^
Þ	追加のインクルード・パス	追加のインクルード・パス[12]	
Þ	システム・インクルード・パス	システム・インクルード・パス[0]	
Þ	コンパイル単位の先頭にインクルードするファイル	コンパイル単位の先頭にインクルードするファイル[0]	
Þ	マクロ定義	マクロ定義[1]	
	無効化するプリデファインド・マクロ		_
	インフォメーションレベル・メッセージ出力を有効にす	いいえ(-nomessage)	
	抑止するインフォメーションレベル・メッセージ番号		
	表示させない警告メッセージ		
	ウォーニング・メッセージをインフォメーションレベルに3	いいえ	
	インフォメーション・メッセージをウォーニングレベルに3	いいえ	
	インフォメーション、ウォーニング・メッセージをエラーレ	いいえ	
	コメント(/* */)のネストを許す	いいえ(-comment=nonest)	
	既存プログラムとの互換性をチェックする	いいえ	
	入力ブログラムの文字コード	SJIS コード(-sjis)	
4	オブジェクト		
	出力ファイル形式	オブジェクト・モジュール・ファイル(-output=obj)	
	デバッグ情報を出力する	(\$()(-debug)	
	プログラム領域のセクション名	P	
	定数領域のセクション名	C	
	新聞化 データ編載 Mセカションタ	D	~
7	加定義		
3	20定義を「くマクロ名>[=く文字列>]」の形式で、1行	に1つずつ指定します。	
L-1.	バイラのオフジョンFluennelに作用目します。		
			_
	モ連オフション 🔪 コンバイル・オブション 🖉 アセンブ	フルーオフション 人 リンク・オフション 人 ヘキサ出力オブション 人 ライブラリ・ジェネレート・.	/ 🔻
_			

図 4.1-2.4-2 マクロ定義削除手順 2

ダイアログの「MBRC_NO_TIMER」を削除し登録します。

	テキスト編集
<u>テキフト(T)</u>	
MRBC_NO_TIMER	^
	\sim
<	>
	OK キャンセル ヘルプ(円)

図 4.1-2.4-3 マクロ定義削除手順 3

5.クリーンビルトと実行

4.1-1 の 11 及び 13 の手順通りにコンパイルし実行する。

2.と同様に、LED1 が 0.2 秒間隔で点滅する。

4.1-3 Chapter03「複数の mruby プログラムを同時に動かす」を CS+ と cc-rx と RX210 の環境で動かす itoc チュートリアル「Chapter03「複数の mruby プログラムを同時に動かす」」を参考にして、4.1-2 を修正してい く。

1.mruby プログラムの用意とコンパイル

itoc チュートリアル「Chapter03「複数の mruby プログラムを同時に動かす」」通りに mruby プログラムの用意する。

コンパイルは、4.1-2で準備した自動コンパイル処理にて行われる。

2.ビルド及び実行を行う。

4.1-1の11及び13の手順通りにコンパイルし実行する。

LED1 が 0.2 秒間隔で点滅する。

3.コンソールを有効にする

UART を有効化する為、FIT モジュール「シリアルコミュニケーションインタフェース(SCI:調歩同期式/クロック同期式)」(R01AN1815)及び「Byte Queue Buffer(データ管理)」(R01AN1683)を使用する。

組込み方は、アプリケーションノート「R01AN1826」を参考にする

なお、SCI の方は、アプリケーションノート「R01AN1815」の 14 ページにある CH0 を有効にする設定を行う必要が有る。

4. [FlashLED.c]プログラムの変更

FIT の API「R_SCI_Send()」を利用し、hal_write()を実装する。

また、main()関数に初期化ルーチンを実装する。

FIT モジュール「シリアルコミュニケーションインタフェース(SCI:調歩同期式/クロック同期式)」のコールバック関数を実装する。

FlashLED.c

#include "src/mrubyc.h"

#include "r_gpio_rx_if.h" //Contains prototypes for the GPIO driver #include "r_lpc_rx_if.h" // Low Power Consumption (LPC) API #include "r_cmt_rx_if.h" // Compair Much Timer (CMT) API #include "r_sci_rx_if.h" // The SCI module API interface file. #include "r_byteq_if.h" // The BYTEQ module API interface file. #include "sample1.c" #include "sample2.c" void cmt_cb(void *pdata); void my_sci_callback(void *pArgs);

void main(void);

#define MEMORY_SIZE (1024*10)
static uint8_t memory_pool[MEMORY_SIZE];

/* Handle storage. Needs to persist as long as SCI calls are going to be made.*/ static sci_hdl_t $g_my_sci_handle;$

//-/ *! オンボード SW 現在状態の読み込み */ static void c_sw1_read(mrb_vm *vm, mrb_value *v) { SET_INT_RETURN(SW1); // SW1 define in FIT BSP rskrx210.h }

/*! オンボード LED ON/OFF */ static void c_led1_write(mrb_vm *vm, mrb_value *v) ł if(GET_INT_ARG(1) == 1) { LED1 = LED_ON; // LED1 define in FIT BSP rskrx210.h } else { LED1 = LED_OFF; } } /*! HAL (別途説明します) , */ int hal_write(int fd, const void *buf, size_t nbytes) R_SCI_Send(g_my_sci_handle, buf, nbytes); return nbytes; ł int hal_flush(int fd) return 0; } void main() lpc_err_t lpc_request_status; uint32_t cmt_hdl; sci_cfg_t my_sci_config; sci_err_t my_sci_err; hardware setup(); /* Configure the MCU for the mode. */ // lpc_request_status = R_LPC_LowPowerModeConfigure(LPC_LP_SW_STANDBY); // STOP CLOCK when no work CMT lpc_request_status = R_LPC_LowPowerModeConfigure(LPC_LP_SW_STANDBY); // lpc_request_status = R_LPC_LowPowerModeConfigure(LPC_LP_ALL_MODULE_STOP); if (LPC_SUCCESS != lpc_request_status) { LED0 = LED_ON; while (1) [while (1) { nop(); } } /* Set up the configuration data structure for asynchronous (UART) operation. */ my_sci_config.async.baud_rate = 19200; my_sci_config.async.clk_src = SCI_CLK_INT; my_sci_config.async.clk_src my_sci_config.async.data_size = SCI_DATA_8BIT; my_sci_config.async.parity_en = SCI_PARITY_OFF; my_sci_config.async.parity_type = SCI_EVEN_PARITY; my_sci_config.async.stop_bits = SCI_STOPBITS_1; my_sci_config.async.int_priority = 3; // 1=lowest, 15=highest /* OPEN ASYNC CHANNEL * Provide address of the config structure, the callback function to be assigned, * * and the location for the handle to be stored.*/ my_sci_err = R_SCI_Open(SCI_CH0, SCI_MODE_ASYNC, &my_sci_config, my_sci_callback, &g_my_sci_handle); if (SCI_SUCCESS != my_sci_err) LED0 = LED_ON; while (1) { nop(); } } /* Create a timer tick with callback at a rate of 1000Hz. */ if(! R_CMT_CreatePeriodic(1000, cmt_cb, &cmt_hdl)) { $LED0 = LED_ON;$ while (1) { nop(); } } /* Place your initialization/startup code here (e.g. MyInst_Start()) */ mrbc_init(memory_pool, MEMORY_SIZE); mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read); mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write); mrbc_create_task(sample1, 0); mrbc_create_task(sample2, 0); mrbc_run(); return 0; } /* assert()関数で使われる。それを受けるため */

void abort(void) LED0 = LED ON; // LED0 define in FIT BSP rskrx210.h } * Function Name: my_sci_callback * Description : This is a template for an SCI Async Mode callback function. * Arguments : pArgs pointer to sci_cb_args_t structure cast to a void. Structure contains event and associated data. * Return Value : none void my_sci_callback(void *pArgs) sci_cb_args_t *args; args = (sci_cb_args_t *)pArgs; if (args->event == SCI_EVT_RX_CHAR) /* From RXI interrupt; received character data is in args->byte */ LED3 = ~LED3; // Toggle LED to demonstrate callback execution. nop(); } else if (args->event == SCI_EVT_RXBUF_OVFL) /* From RXI interrupt; rx queue is full; 'lost' data is in args->byte You will need to increase buffer size or reduce baud rate */ nop(): } else if (args->event == SCI_EVT_OVFL_ERR) /* From receiver overflow error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ nop(): else if (args->event == SCI_EVT_FRAMING_ERR) /* From receiver framing error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ nop(); else if (args->event == SCI_EVT_PARITY_ERR) * From receiver parity error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ nop(); } } /* * タイマー割り込みルーチンのコールバック */ void cmt_cb(void *pdata) { mrbc_tick();

} /* end tmr0_cmia0_is */

5.mruby プログラムの修正及び実行

itoc のチュートリアルに有る通り、修正すると、ターミナルに文字が出力される。 なお、文字区切りコードは、LF なので、注意が必要である。



4.2-1 Chapter01「LED 点滅」を CS+ と cc-rx と RX210 の環境で動かす

itoc チュートリアル「Chapter01「LED 点滅」」を参考にしつつ、Renesas 社提供のドライバ FIT を使わずに実装し てみる。

コンパイルの方法及び mruby/c には、FIT を使ったものとの違いはない為、「FlashLED.c」「hal.h」のみ掲載する。

なお、4.1-1 では busywait 関数を、FIT 提供の物を使用していたため、Renesas 社アプリケーションノート 「R01AN1852」掲載を追加する。

R01AN1852

https://www.renesas.com/ja-jp/software/D3014140.html

hal.h

#ifndef MRBC_SRC_HAL_H_ #define MRBC_SRC_HAL_H_

#ifdef __cplusplus extern ″C″ { #endif

#ifndef MRBC_NO_TIMER
define hal_init() ((void)0)
define hal_enable_irq() (set_ipl(0))
define hal_disable_irq() (set_ipl(15))
define hal_idle_cpu() (wait())

#else // MRBC_NO_TIMER
define hal_init() ((void)0)
define hal_enable_irq() ((void)0)
define hal_disable_irq() ((void)0)
define hal_idle_cpu() (R_DELAY_Us(1000, 25000), mrbc_tick())

#endif

#ifdef __cplusplus } #endif #endif // ifndef MRBC_HAL_H_

FlashLED.c

#include <machine.h>
#include "iodefine.h"
#include "src/mrubyc.h"

#include "sample1.c"

/* Local defines */	
#define LED_ON	(0)
#define LED_OFF	(1)
#define SET_BIT_HIGH	(1)
#define SET_BIT_LOW	(0)
#define SET_BYTE_HIGH	(0xFF)
#define SET_BYTE_LOW	(0x00)

```
#define GPIO_OUTPUT
                                  (1)
#define GPIO_INPUT (0)
#define MODE_GPIO (0)
#define MODE_PERIPHERAL
                                 (1)
#define MEMORY_SIZE (1024*10)
static uint8_t memory_pool[MEMORY_SIZE];
void clock_setup( void );
void gpio_setup( void );
/*! オンボード SW 現在状態の読み込み
*/
static void c_sw1_read(mrb_vm *vm, mrb_value *v)
{
   int sw1 = PORT3.PIDR.BIT.B1;
                                             // SW1
   SET_INT_RETURN(sw1);
}
/*! オンボード LED ON/OFF
*/
static void c_led1_write(mrb_vm *vm, mrb_value *v)
{
   if( GET_INT_ARG(1) == 1 ) {
           PORT1.PODR.BIT.B5 = LED_ON;
                                                        // LED1
   } else {
                                                        // LED1
           PORT1.PODR.BIT.B5 = LED_OFF;
   1
}
//=
/*! HAL (別途説明します)
,
*/
int hal_write(int fd, const void *buf, size_t nbytes)
   return 0;
int hal_flush(int fd)
   return 0;
}
int main()
   clock_setup();
   gpio_setup();
   /* Place your initialization/startup code here (e.g. MyInst_Start()) */
   mrbc_init(memory_pool, MEMORY_SIZE);
   mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);
   mrbc_create_task( sample1, 0 );
   mrbc_run();
   return 0;
}
void abort(void)
{
   PORT1.PODR.BIT.B4 = LED_ON;
                                             // LED0
}
void write(void)
{
   PORT1.PODR.BIT.B4 = LED_ON;
                                             // LED0
}
void clock_setup( void )
```

// write protect disable SYSTEM.PRCR.WORD = 0xA50B;

ł

SYSTEM.OPCCR.BYTE = 0x00;

while(SYSTEM.OPCCR.BIT.OPCMTSF == 1)
{

/* Wait for transition to finish. */
}
SYSTEM.MOFCR.BYTE = 0x30;
SYSTEM.HOCOCR.BYTE = 0x01;
SYSTEM.HOCOPCR.BYTE = 0x01;

SYSTEM.MOSCCR.BYTE = 0x01; SYSTEM.SOSCCR.BYTE = 0x01;

SYSTEM.MOSCWTCR.BYTE = 0x0D; SYSTEM.MOSCCR.BYTE = 0x00; SYSTEM.PLLWTCR.BYTE = 0x0C; SYSTEM.PLLCR.BIT.PLIDIV = 2 >> 1; // 20MHz/2 = 10MHz SYSTEM.PLLCR.BIT.STC = 10 - 1; // 10MHz*10 = 100MHz SYSTEM.PLLCR2.BYTE = 0x00; for(int i = 0; i < 160; i++) { /* Wait 13ms. See comment above for why. */ nop(); } $\label{eq:system.sckcr.long} \begin{array}{l} \text{System.sckcr.long} = 0 \\ \text{system.sckcr3.word} = ((\text{uint16_t})4) \\ \text{sys$ SYSTEM.LOCOCR.BYTE = 0x01; // write protect enable
SYSTEM.PRCR.WORD = 0xA500; } void gpio_setup(void) ł // OUT_LED0 DATA // OUT_LED1 DATA // OUT_LED2 DATA PORT1.PODR.BIT.B4 = LED_OFF; PORT1.PODR.BIT.B5 = LED_OFF; PORT1.PODR.BIT.B6 = LED_OFF; PORT1.PODR.BIT.B7 = LED_OFF; // OUT_LED3 DATA PORT1.PDR.BIT.B4 = GPIO_OUTPUT; // OUT_LED0 DIR PORT1.PDR.BIT.B5 = GPIO_OUTPUT; // OUT_LED1 DIR PORT1.PDR.BIT.B6 = GPIO_OUTPUT; // OUT_LED2 DIR PORT1.PDR.BIT.B7 = GPIO_OUTPUT; // OUT_LED3 DIR // SW1 PDR PORT3.PDR.BIT.B1 = GPIO INPUT: // SW2_PDR // SW3_PDR PORT3.PDR.BIT.B3 = GPIO_INPUT; PORT3.PDR.BIT.B4 = GPIO INPUT; PORT3.PDR.BIT.B1 = MODE_GPIO; // SW1_PMR // SW2_PMR // SW3_PMR PORT3.PDR.BIT.B3 = MODE_GPIO; PORT3.PDR.BIT.B4 = MODE_GPIO; // write protect disable
MPC.PWPR.BIT.BOWI = 0; MPC.PWPR.BIT.PFSWE = 1; /* TxD setup */ PORT2.PMR.BIT.B0 = MODE_GPIO; MPC.P20PFS.BYTE = 0x0A; PORT2.PDR.BIT.B0 = GPIO_OUTPUT; // RXD PMR 一旦 GPIO にしてから // TXD0 SELECT TXDを選択し ポート方向を変えて ペリフェラルに設定する。 // TXD PDR PORT2.PMR.BIT.B0 = MODE_PERIPHERAL; // TXD_PMR /* RxD setup * PORT2.PMR.BIT.B1 = MODE_GPIO; // RXD_PMR 一旦 GPIO にしてから // RXD0 SELECT RXD を選択し // RXD_PDR ポート方向を変えて // RXD_PMR ペリフェラルに設定する。、 MPC.P21PFS.BYTE = 0x0A; PORT2.PDR.BIT.B1 = GPI0_INPUT; PORT2.PMR.BIT.B1 = MODE_PERIPHERAL; write protect enable MPC.PWPR.BIT.BOWI = 0; MPC.PWPR.BIT.PFSWE = 0; MPC.PWPR.BIT.B0WI = 1;

}

23

4.2-2 Chapter02「LED 点滅の速さを変える」を CS+ と cc-rx と RX210 の環境で動かす

itoc チュートリアル「Chapter02「LED 点滅の速さを変える」」を参考にしつつ、Renesas 社提供のドライバ FIT を 使わずに実装してみる。

コンパイルの方法及び mruby/c には、FIT を使ったものとの違いはない為、「FlashLED.c」のみ掲載する。

また、自動生成されるソース「intprg.c」の 321 行目に使用している割り込みベクタの設定(下記)が有り、重複す るのでコメントアウトする。

// TMR0 CMIA0

void Excep_TMR0_CMIA0(void){}

FlashLED.c

#include <machine.h> #include "iodefine.h" #include "vect.h"

#include "src/mrubyc.h"

(略)

void clock_setup(void); void gpio_setup(void); void timer_setup(void); void lowpower_setup(void);

(略)

}

{

```
int main()
```

```
clock_setup();
   gpio_setup();
   timer_setup();
  lowpower_setup();
   /* Place your initialization/startup code here (e.g. MyInst_Start()) */
  mrbc_init(memory_pool, MEMORY_SIZE);
  mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);
  mrbc_create_task( sample1, 0 );
  mrbc_run();
  return 0;
(略)
#pragma interrupt (Excep_TMR0_CMIA0(vect=VECT(TMR0, CMIA0)))
void Excep_TMR0_CMIA0(void)
           static int i = 0;
           mrbc_tick();
           if( i >= 500 ) {
                      PORT1.PODR.BIT.B6 = "PORT1.PODR.BIT.B6; // Toggle LED2 with each interrupt
                     i = 0:
           i++•
} /* end Excep_TMR0_CMIA0 */
(略)
void timer_setup( void )
   // TMR0 POWER ON
   // write protect disable
   SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) | 0x02 );
  MSTP(TMR0) = 0; // TMR0 TMR1 ON
// write protect enable
  SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) & (~0x02));
                                 // TMR1 NO FUNCTION
  TMR1.TCCR.BYTE = 0;
                                // cascade 接続 16bit タイマー
// PCLK/8
   TMR1.TCCR.BIT.CSS = 1;
   TMR1.TCCR.BIT.CKS = 2;
                                // CMP A にてカウンターリセット
  TMR1.TCR.BIT.CCLR = 1;
```

TMR0.TCCR.BIT.CSS = 3; // cascade 接続 16bit タイマー TMR0.TCCR.BIT.CKS = 0; TMR0.TCCR.BIT.CKS = 0; // なし TMR0.TCR.BIT.CMIEB = 0; // CMP B 割り込み なし TMR0.TCR.BIT.COMEA = 1; // CMP A 割り込み 有り TMR0.TCR.BIT.OVIE = 0; // オーバーフロー割り込み なし TMR0.TCR.BIT.CCLR = 1; // CMP A にてカウンターリセット TMR01.TCORB = 0xffff; TMR01.TCORA = 3125 - 1; // 1msec * 25MHz / 8 = 3125 TMR01.TCNT = 0; IPR(TMR0, CMIA0) = 1;DTCE(TMR0, CMIA0) =0; IEN(TMR0, CMIA0) = 1;} void lowpower setup(void) SET LOW POWER MODE ALL MODULE STOP // write protect disable SYSTEM.PRCR.WORD = (uint16_t)((0xA500 |SYSTEM.PRCR.WORD) | 0x02); SYSTEM.SBYCR.BIT.SSBY SYSTEM.SBYCR.BIT.SSBY = 0; SYSTEM.MSTPCRA.BIT.ACSE = 1; SYSTEM.MSTPCRA.BIT.MSTPA24 = 1; SYSTEM.MSTPCRA.BIT.MSTPA27 = 1; SYSTEM.MSTPCRA.BIT.MSTPA29 = 1; // write protect enable
SYSTEM.PRCR.WORD = (uint16_t)((0xA500 |SYSTEM.PRCR.WORD) & (~0x02)); } 4.2-3 Chapter03「複数の mruby プログラムを同時に動かす」を CS+ と cc-rx と RX210 の環境で動かす

itoc チュートリアル「Chapter03「複数の mruby プログラムを同時に動かす」」を参考にしつつ、Renesas 社提供のドライバ FIT を使わずに実装してみる。

コンパイルの方法及び mruby/c には、FIT を使ったものとの違いはない為、「FlashLED.c」のみ掲載する。 コンパイル及び実行を行い、LED の点滅及びターミナルへの出力を確認する。

UARTの送受信を簡単にする為、itoc様の「UART wrapper for PSoC5LP」をRX用に移植させて頂いた。

また、自動生成されるソース「intprg.c」の 321 行目,432 行目に使用している割り込みベクタの設定(下記)が有り、 重複するのでコメントアウトする。

// TMR0 CMIA0

void Excep_TMR0_CMIA0(void){}

// SCI0 RXI0

void Excep_SCI0_RXI0(void){ }

// SCI0 TXI0 void Excep_SCI0_TXI0(void){ }

FlashLED.c

```
#include <machine.h>
#include ″iodefine.h″
#include ″src/mrubyc.h″
#include ″uart.h″
#include "sample1.c"
#include "sample2.c"
(略)
void clock_setup( void );
void gpio_setup( void );
void timer_setup( void );
void lowpower_setup( void );
void sci_setup( void );
static uint8_t memory_pool[MEMORY_SIZE];
UART HANDLER uh;
(略)
int main()
  uart_init( &uh );
  clock_setup();
  gpio_setup();
  sci_setup();
  timer_setup();
  lowpower_setup();
  /* Place your initialization/startup code here (e.g. MyInst_Start()) */
  mrbc_init(memory_pool, MEMORY_SIZE);
  mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);
  mrbc_create_task( sample1, 0 );
  mrbc_create_task( sample2, 0 );
  mrbc_run();
  return 0;
}
(略)
void sci_setup( void )
   // SCI0
   // write protect disable
  SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) | 0x02 );
  MSTP(SCI0) = 0; // SCI0 ON
  // write protect enable
SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) & (~0x02));
  SCI0.SCR.BYTE = 0;
  SCI0.SMR.BYTE = 0;
  SCI0.SEMR.BYTE = 0x10;
  SCI0.BRR = 0 \times 50;
  for(int i = 0; i < 160; i++)
         /* Wait 13ms. See comment above for why. */
         nop();
  }
  // SCIO 割り込み設定
                           // 割り込み優先度 3
// 割り込み有効化
// 割り込みフラグクリア
  IPR( SCI0,TXI0 ) = 1;
IEN( SCI0,TXI0 ) = 1;
  IR(SCI0,TXI0) = 0;
                           // 送信割り込み及び送信イネーブル
  SCI0.SCR.BYTE = 0xA0;
  // write protect disable
MPC.PWPR.BIT.BOWI = 0;
  MPC.PWPR.BIT.PFSWE = 1:
   /* TxD setup */
```

// RXD_PMR

PORT2.PMR.BIT.B0 = MODE_GPIO;

一旦 GPIO にしてから

MPC.P20PFS.BYTE = 0x0A; // TXD0 SELECT TXD を選択し PORT2.PDR.BIT.B0 = GPI0_OUTPUT; // TXD_PDR ポート方向を変えて PORT2.PMR.BIT.B1 = MODE_PERIPHERAL; // TXD_PMR ペリフェラルに設定する。 /* RxD setup */ PORT2.PMR.BIT.B1 = MODE_GPI0; // RXD_PMR 一旦 GPI0 にしてから MPC.P21PFS.BYTE = 0x0A; // RXD0 SELECT RXD を選択し PORT2.PDR.BIT.B1 = GPI0_INPUT; // RXD_PDR ポート方向を変えて PORT2.PMR.BIT.B1 = MODE_PERIPHERAL; // RXD_PMR ペリフェラルに設定する。、 // write protect enable MPC.PWPR.BIT.B0WI = 0; MPC.PWPR.BIT.B0WI = 1;

}

4.3 e2studio と gcc for Renesas と RX210 の環境への mruby/c インポート

4.3-1 Chapter01「LED 点滅」を e2studio と gcc for Renesas と RX210 の環境で動かす

FIT ドライバは CC-RX のみ動作対象なので、gcc for Renesas 環境では動作しない。

しかし、4.2 で FIT ドライバを使わずに mruby/c を動作させたので、それを参考にしつつ itoc チュートリアル 「Chapter01「LED 点滅」」を動作させる。

1.実行プログラムプロジェクトを作成する

「ファイル」->「新規」->「C/C++ Project」を選択する。

e ²										mru	ubyc - C/C	2+·
ファイ	ル(<u>E)</u> 編集(<u>E</u>)ソ	- <u>ス(S</u>)	リファクタリング(T)	ナビゲート(<u>N</u>)	検索(<u>A</u>)	プロジョ	<u> / / (P</u>)	Renesas <u>V</u> iews	実行(<u>R</u>)	ウィンドウ(<u>W</u>)	ヘルプ(<u>H</u>)	
	新規(<u>N</u>)			Alt	+シフト+N)	•	Makef	ile Project with E	xisting Co	de		,
	ファイルを開く(<u>.</u>)					C	C/C++	+ Project				
	Open Projects from	n File S	System			Ċ	プロジェ	クト(<u>R</u>)				
	閉じる(<u>C</u>)				Ctrl+W	C++	C/C++	+ プロジェクトに変換	(C/C++ オ	ーチャーを追加)		at
	すべて閉じる(<u>L</u>)			Ctrl-	+シフト+W	63	ソース・コ	フォルダ				
B	保管(S)				Ctrl+S	*	フォルダ	_				a
	別名保存(<u>A</u>)					C	ソース・3	ファイル				Ĭ
G	すべて保管(E)			Ctrl	+シフト+S	h	ヘッダー	・ファイル				n
	前回保管した状態に原	₹す(<u>T</u>)					テンプレ	ートからファイル				
	移動(V)					G	クラス					
-a	名前を変更(M)				F2	1	コード生	成				
s.	更新(F)				F5		タスク					
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	(D)			,	<b></b>	その他(	<u>o</u> )			Ctrl+N	
						-		o._#				

図 4.3-1.1-1 新規プロジェクト作成1

「GCC for Renesas RX C/C++ Executable Project」を選択し「次へ」を押す。



図 4.3-1.1-2 新規プロジェクト作成 2

プロジェクト名を「FlashLED」とし、保存場所を指定(個々の環境で異なる)し「次へ」を押す。

e²		_ 🗆 🗙
GCC for Renesas RX		
New GCC for Renesas F	X Executable Project	
プロジェクト名(P): FlashLE	D	
🗌 デフォルト・ロケーションの	使用(卫)	
ロケーション( <u>L</u> ):	C:¥WorkSpace¥mrubyc	参照( <u>R</u> )
	Create Directory for Project	
ファイル・システムを選択( <u>Y</u> ):	デフォルト 🗸	
ワーキング・セット		
ワーキング・セットにプロジ	江クトを追加( <u>工)</u>	新規( <u>W</u> )
ワーキング・セット(0):	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	選択( <u>E</u> )
?	< 戻る( <u>B</u> ) 次へ( <u>N</u> ) > 終了(E)	キャンセル

図 4.3-1.1-3 新規プロジェクト作成 3

ツールチェインの指定及び開発対象の CPU「RX210」「RX210 - 100pin」「R5F5210BBxFP」を指定する。

					이에요. 이번 것 이 것에는 것		R5F52105AxFP
							R5F52105AxLJ
							R5F52105BxFP
							R5F52105BxLA
1			- • ×		1 概要		R5F52105BxLJ
GCC for Renesas R	<				フィーチャーの概要		R5F52106AxFP
Select toolchain, devic	e & debug settings						R5F52106AxLJ
							R5F52106BxFP
Toolchain Settings				12	チュートリアル		R5F52106BxLA
THE:	●C ○C++				チュートリアルの実行	Ŧ	R5F52106BxLJ
ツールチェーン・	GCC for Renesas RX	7					R5F52107AxFP
					2 ++`/=/IL		R5F52107AxLJ
ツールチェーン・バージョン:	4.8.4.201701	·			・ ワンブルの試行		R5F52107CxFP
	ツールチェーンの管理	<u>.</u>					R5F52107CxL3
Device Settings		Configurations					R5F52108AxFP
ターゲット・デバイス: R5F	51101AxLM	9	bug 構成を生成	6	う 新機能		R5F52108AxLJ
	デバイスのアンロック。	RX100 +		0	新規約について	-	R5F52108CxFP
Tンディアン: Littl		RX200 +	RX210	•	RX210 - 48pin	•	R5F52108CxLJ
		RX600 +	RX21A	•	RX210 - 64pin	•	R5F5210ABxFP
ノロジェクト・タイノ: ナノオ	1//r	RX700 +	RX220	•	RX210 - 69pin	1	R5F5210ABxLJ
			RX230	· _	RX210 - 80pin	•	R5F5210BBxFP
			RX231	•	RX210 - 100pin	•	R5F5210BBxLJ
			RX23T	•	RX210 - 144pin	+	
			RX24T	•	RX210 - 145pin	•	
			RX24U	•			
		64-7 (m)					

図 4.3-1.1-4 新規プロジェクト作成 4

「次へ」を押しても良いが、この先有用なオプションが無いので、「終了」して新規プロジェクトを作成する。

2.新規プロジェクト「FlashLED」が作成されたので,e2studio と GCC for Renesas が正しく動作する事を確認する 為、ビルドを行う。

「ビルド」->「すべてビルド」を選択し、コンソール欄に

「'Finished building target:「プロジェクト名」.mot'」と出力されていれば正しくコンパイルされている。



図 4.3-1.2-1 テストビルドの成否の確認

3.mruby/c VM ソースコードを、プロジェクトへ追加

・mruby/c ソースコードを展開し、src フォルダー内の全ファイルをファイルエキスプローラ等で src フォルダー ごとプロジェクトフォルダーのsrcフォルダー内にコピーする。

・src/src フォルダー内の hal_psoc5lp フォルダーを消去する。

・src/src フォルダー内の hal_posix フォルダーを消去する。

4. Renesas 社提供のドライバ FIT を使わないサンプルをコピーする

4.2-1 で実装した、Renesas 社提供のドライバ FIT を使わない「FlashLED.c」を上書きコピー、「r_delay.c」及び 「r_delay.h」を本プロジェクトにコピーする。また、「hal.h」を、hal フォルダーごとコピーする。

また、コンパイル済みの「sample1.c」をコピーするか、「sample1.rb」をコピーし mbrc にてコンパイルする

5.cc-rxと異なる部分を変更する。

cc-rxとgccでは、c言語仕様で規定されていない部分及びシステム関連の関数の実装状況が異なるので、その部分を置き換える。

まず、インラインアセンブラが異なるので、「r_delay.c」内の R_DELAY 関数を下記の様に変更する。

static void R_DELAY (unsigned long loop_cnt)

パスの通り方も CS+と異なるので、「hal.h」の include を下記の通り変更する。

#include "..¥..¥r_delay.h"

「FlashLED.c」は、

- 「machine.h」が無いので、消去する。
- ・「hal.write」関数の引数「nbytes」が size_t 型となっており、他の関数と異なっているので、int 型に変更する。

int hal_write(int fd, const void *buf, int nbytes)

・nop()関数がないので、インラインアセンブラで定義する。

#define nop() __asm _volatile("nop¥n")

・CS+との違いで「#include "sample1.c"」がうまくいかないので、下記の様に変更する。

extern char sample1[];

### 6.コンパイルオプションを設定する

「プロジェクト」->「プロパティ」でプロパティ画面を開く。



図 4.3-1.6-1 コンパイルオプション設定 1

「C/C++ ビルド」- >「設定」->「ツール設定」タブ ->「Compiler」->「source」を選ぶ。

「Language standard」 プルダウンメニューを「ISO C99」にする。

Chapter01 は、タイマーを使わないので、「User defined compiler options」に「-D MRBC_NO_TIMER」を設定する。



図 4.3-1.6-2 コンパイルオプション設定 2

7.クリーンビルドと実行 「プロジェクト」->「クリーン」を選ぶ。



図 4.3-1.7-1 クリーンビルド 1

すると下記ダイアログがでるので、「OK」ボタンを押す。

2	クリーン	_ 🗆 🗙
クリーンを実行するとすべてのビルドの問題 ロジェクトは初めから再ビルドされます。	とビルド状態が破棄されます	。 次回のビルド時には、プ
○ すべてのプロジェクトをクリーン( <u>A</u> )	<ul> <li>以下で選択した</li> </ul>	:プロジェクトをクリーン( <u>S</u> )
☑	le1	
<ul> <li>✓ビルドを即時に開始(<u>B</u>)</li> <li>●「ワークスペース全体をビルド(<u>W</u>))</li> <li>○ 選択したプロジェクトのみビルド(<u>P</u>)</li> </ul>	OF	< <b>+</b> P>t214

図 4.3-1.7-2 クリーンビルド 2

成功していれば、「2」と同様にコンソール欄に

「'Finished building target:「プロジェクト名」.mot'」と出力されていれば正しくコンパイルされている。

8.デバッガの設定及び実行 E1 デバッガによる H/W デバッグの設定を行う。 「実行」->「デバッグの構成」を選ぶ。



### すると、下記の画面が開くので

「Renesas GDB Hardware Debugging」->「(プロジェクト名) HardwareDebug」を選ぶ。

e ²	デバッグ構成	×
構成の作成、管理、および実行		To a constant of the second se
<ul> <li>○ 油 × ●</li> <li>○ パレタ入力</li> <li>○ (パレキ、ガブリケーション)</li> <li>○ C/C++ リモート・アブリケーション</li> <li>■ EASE Script</li> <li>○ GDB Simulator Debugging (RH850)</li> <li>○ GDB //- ドウェア・デパリキング</li> <li>Java アブリット・ション</li> <li>Java アブリット・ション</li> <li>■ Renesas GDB Hardware Debugging</li> <li>○ Renesas Simulator Debugging (RX, RL78)</li> <li>Uモート Java アブリケーション</li> <li>▶ 起動グルーブ</li> </ul>	名前(N): e2_native_mrubyc_sample1 Hi メイン 参 Debugger	ardwareDebug 二 共通(C) ほ ソース 参照(D) を照(D) 参照(D) 参照(D) 参照(D) 参照(D) 参照(D) 少 〇 目動ビルドを無効にする <u>ワークスペース設定の構成</u>
フィルター一致: 12 / 14 項目		前回保管した状態に戻す(⊻) 適用(Y)
0		デバッグ( <u>D</u> ) 閉じる

図 4.3-1.8-2 デバッガの設定 2

右ペインの「Debugger」タブを選び、「Connection Settings」タブを選ぶ。

2	デバッグ構成		
構成の作成、管理、および実行			Ś
<ul> <li>□ 計</li> <li>□ 計</li> <li>□ (J(L+) アブリケーション</li> <li>□ (J(L+) アブリケーション</li> <li>□ (J(L+) アブリケーション</li> <li>□ EASE Script</li> <li>□ GOB Simulator Debugging (RH850)</li> <li>□ GOB UP = 55×∞ = 50×55</li> </ul>	名前(N): e2_native_mrubyc_sample1 HardwareDebu メイン 参 Debugger Startup 二共道(C) 与 Debug hardware: E1 (RX) V Target I GDB Settings Connection Settings デ/(ッグ・ソール)	ig ソース Device: R5F52108 設定	
	▲ クロック		^
lava 771 wh	メイン・クロック・ソース	EXIAL	~
Renesas GDB Hardware Debugging	EXTAL 周波数[MHZ]	25.0000	
e e2 native mrubyc sample1 HardwareDebug	111/ファッシュメモリー書き換え時にクロック・ソースの姿	(20)	~
■ Renesas Simulator Debugging (RX, RL78) リモート Java アプリケーション	▲ ターグット・ホートとの技術	(Auto)	
	121/-9-	(Auto)	
▶ 起動グループ	授売ゲイノ コTag クロック国連数[Mula]	Fine 16.5	× I
	Jidg グロック周波数[MH2]	2.00	- Y
	+ine //-/[inops]	0.02	Ť
		0.002	~
	FBIA TSコレーターから電源を供給する(MAX 200mA)	(11)	~
	(IAX 20011A) (#絵画圧	5 0V	Ť
		5.07	
		シングルチップ	~
		シングルチップ・モード	
	記動バンクを変更する	いいえ	~
	記動バンク	バンクロ	~
	. 18/ET k		*
7イルター一致: 12 / 14 項目		前回保管した状態に戻す(乂) デバッグ( <u>D</u> )	適用( <u>Y</u> ) 閉じる

図 4.3-1.8-3 デバッガの設定 3

「クロック」の「EXTAL 周波数[MHz]」を「25」

「電源」の「エミュレータから電源を供給する(MAX 200mA)」を「はい」

「供給電圧」を「5.0V」

に変更し、「デバッグ」ボタンを押すと、デバッガに接続され、ボードに書き込まれる。

「実行」->「再開」を選ぶと実行される。

なお、e2 studioのデバッグ環境は、アセンブラによる初期化ルーチンから実行され、一旦 c 言語の main でブレークポイントが掛かる。

今回デバッグ対象となるプログラムを実行するには2回再開する必要がある。

本プログラムでは、LED1 が1秒おきに点滅する。



図 4.3-1.8-4 プログラムの実行

4.3-2 Chapter02「LED 点滅の速さを変える」を e2studio と gcc for Renesas と RX210 の環境で動かす itoc チュートリアル「Chapter02「LED 点滅の速さを変える」」を参考にして、4.3-1 を修正していく。

1.mruby スクリプトの自動コンパイル処理の追加

「プロジェクト」->「プロパティ」でプロパティ画面を開く。

プロ	ت <u>י</u> לעבט	Renesas <u>V</u> iews	<u>実行(R</u> )	ウィンドウ( <u>W</u> )
×	プロジェクトを <b>プロジェクトを</b>	2開く( <u>E)</u> 2 <b>閉じる(<u>S</u>)</b>		
010	すべてビルド ビルド構成 プロジェクトの	( <u>A)</u> Dビルド(B)	Ctrl	+Alt+B
	ワーキング・t クリーン( <u>N</u> ). 自動的にビノ	ビットのビルド( <u>W</u> )  ルド( <u>M</u> )		•
	Make ターク	ቻット		•
e²	C/C++ イン Update Al	ンデックス( <u>i</u> ) l Dependencies		Alt+D
	プロパティ(P	)		

図 4.3-2.1-1 自動コンパイルスクリプトの追加 1

「C/C++ ビルド」->「設定」->「ビルド・ステップ」タブを選ぶ。

ビルド前のステップに、コマンドをフルパスで指定する。

ここでは、プロジェクトの src ディレクトリに「sample1.rb」「mrbc.bat」「mrbc.exe」を入れ、設定する事とする。

e ²	プロパティ: E2_TEST2 -	×
フィルタ入力	設定 🗘 🔹	> • •
<ul> <li>&gt; リソース</li> <li>a C/C++ ビルド Tool chain エディター ビルド変数 ロギング 環境 設定</li> <li>&gt; C/C++ 一般</li> <li>&gt; LinkerScript Renesas QE</li> <li>&gt; タスク・リポジドリー ビルター プロジェクト参照 実行/デバック設定</li> </ul>	構成: HardwareDebug [アクティフ]    構成の管 ● ツール設定 ● Toolchain ● Device ● ビルド・ステップ ● ビルド成果物 副 パイナリー・パーサー ( ビルド前のステップ コマンド: C:¥WorkSpace¥mrubyc¥E2_TEST2¥src¥mrbc.bat 説明: ビルド後のステップ コマンド: 、 説明:	
?	OK ‡7:	ven

図 4.3-2.1-2 自動コンパイルスクリプトの追加 2

2.浮動小数点の利用

itoc チュートリアル通りに、sample1.rbを修正し、4.3-1の7及び8の手順通りにコンパイルし実行する。

すると、LED1が0.2秒間隔で点滅する。

これは、e2_studioとgcc for Renesas の環境では、デフォルトで浮動小数点を扱う様に設定されているからである。

3. タイマーの利用

itoc チュートリアル通りに、タイマーを利用するように設定を変更する。 「プロジェクト」->「プロパティ」でプロパティ画面を開く。



図 4.3-2.3-1 コンパイルオプション設定1

「C/C++ ビルド」- >「設定」->「ツール設定」タブ ->「Compiler」->「source」を選ぶ。

Chapter02 は、タイマーを使うので、「User defined compiler options」の「-D MRBC_NO_TIMER」を消す。



図 4.3-2.3-2 コンパイルオプション設定 2

#### 4.割り込み用スタックの増量

このプログラムにおいては、割り込み用のスタックが足りなくなるので、「Inker_script.ld」を変更する。 「generate」を開いて、「Inker_script.ld」をダブルクリックする。



図 4.3-2.4-1 割り込み用スタックの増量1

「Linker Script Editor」が開くので、「.istack」をマウスで選び、「Vritual Memory Address」を、「0x100」 -> 「0x400」 へ変更する。また、「.data」も「Vritual Memory Address」を、「0x204」 -> 「0x404」へ変更する。

efined Sections		Output Section Details		
Specify linker script sections in the table below	v	Set the properties of the out	put section.	
b .fvectors ( 0xFFFFF80 ) > ROM	Add Section	Output Section Name:	.istack	
<ul> <li>b .text ( 0x+F+00000 ) &gt; ROM</li> <li>b .rvectors &gt; ROM</li> </ul>	Add Assignment	Set Virtual Memory Addre	ss (\//\/	
init > ROM	Remove	Virtual Memory Address	A): 0x400	
JINI > ROM .qot > ROM	nemore	Virtual Memory Region:	RAM	Browse
p .rodata > ROM		Set Load Memory Address	(LMA) explicitly	
b .eh_trame_hdr > ROM b .eh_frame > ROM		Load Memory Address (LMA)		
⊳ 🧽 .jar > ROM		Load Memory Region:		Browse
.ustack ( 0x200 ) > RAM		KEEP (Group will not be or	nitted even if unrefere	nced)
istack ( 0x400 ) > RAM		Mark as "NOLOAD" (Group	will not be loaded at	runtime)
.udid ( 0X404 ) > KAM		Linker Script Commands	Preview:	
⊳ 😓 .bss > RAM		.istack 0x400 : { istack = .; } > RAM		ĺ

図 4.3-2.4-2 割り込み用スタックの増量 2

5.自動生成プログラムの修正や、プログラムを修正する。

自動生成されるソースについて、修正を行う。

・「inthandler.c」の 318 行目に利用している割り込みベクタの設定(下記)があり、重複するので、コメントアウトす る。

Finthandler.c」

// TMR0 CMIA0 // void Excep_TMR0_CMIA0(void) { }

・「start.S」の 90 行目から 98 行目を省電力モードへの移行を行う wait 命令が特権命令のため、ユーザモード に移行しないようにする為コメントアウトする。

「start.S」

/* change PSW PM to user-mode */ /* MVFC PSW,R1 OR #00100000h,R1 PUSH.L R1 MVFC PC,R1 ADD #10,R1 PUSHL R1 RTE NOP NOP

*/

「FlashLED.c」の変更点を記載する。

#### [FlashLED.c]

#include ″iodefine.h″ #include ″src/mrubyc.h″

(略)

#define nop() _asm _volatile("nop¥n")

#define MEMORY SIZE (1024*10) static uint8_t memory_pool[MEMORY_SIZE];

void clock_setup( void ); void gpio_setup( void ); void timer_setup( void ); void lowpower_setup( void );

(略)

int main()

clock_setup(); gpio_setup(); timer_setup(); lowpower_setup();

/* Place your initialization/startup code here (e.g. MyInst_Start()) */ mrbc_init(memory_pool, MEMORY_SIZE); mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read); mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);

mrbc_create_task( sample1, 0 ); mrbc_run();

return 0;

} (略)

void _attribute_ ((interrupt)) Excep_TMR0_CMIA0(void)

static int i = 0;

mrbc_tick();

```
if( i >= 500 ) {
PORT1.PODR.BIT.B6 = "PORT1.PODR.BIT.B6; // Toggle LED2 with each interrupt
         }
         i++;
} /* end Excep_TMR0_CMIA0 */
(略)
void timer_setup( void )
   // TMR0 POWER ON
  // Write protect disable
SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) | 0x02 );
MSTP(TMR0) = 0; // TMR0 TMR1 ON
   // write protect enable
  SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) & (~0x02));
  TMR1.TCCR.BYTE = 0;
                            // TMR1 NO FUNCTION
  TMR1.TCCR.BIT.CSS = 1;
TMR1.TCCR.BIT.CKS = 2;
                           // cascade 接続 16bit タイマー
// PCLK/8
// CMP A にてカウンターリセット
  TMR1.TCR.BIT.CCLR = 1:
  TMR0.TCCR.BIT.CSS = 3;
                           // cascade 接続 16bit タイマー
  TMR0.TCCR.BIT.CKS = 0;
                            // なし
  TMR0.TCR.BIT.CMIEB = 0, // CMP B 割り込み なし
TMR0.TCR.BIT.CMIEB = 0, // CMP B 割り込み なし
TMR0.TCR.BIT.CMIEA = 1; // CMP A 割り込み 有り
TMR0.TCR.BIT.OVIE = 0; // オーバーフロー割り込み なし
TMR0.TCR.BIT.CCLR = 1; // CMP A にてカウンターリセット
  TMR01.TCORB = 0xffff;
  TMR01.TCORA = 3125 - 1; // 1msec * 25MHz / 8 = 3125
  TMR01.TCNT = 0;
  IPR(TMR0, CMIA0) = 1:
  DTCE(TMR0, CMIA0) =0;
IEN(TMR0, CMIA0) = 1;
void lowpower_setup( void )
   // SET LOW POWER MODE ALL MODULE STOP
  // write protect disable 
 SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) | 0x02 );
  SYSTEM.SBYCR.BIT.SSBY = 0;
SYSTEM.MSTPCRA.BIT.ACSE = 1;
SYSTEM.MSTPCRA.BIT.MSTPA24 = 1;
SYSTEM.MSTPCRA.BIT.MSTPA27 = 1;
  SYSTEM.MSTPCRA.BIT.MSTPA29 = 1;
     write protect enable
   SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) & (~0x02));
    「hal.h」を記載する。
「hal.h」
#ifndef MRBC_SRC_HAL_H_
#define MRBC_SRC_HAL_H_
#ifdef _cplusplus
extern "C" {
#endif
#ifndef MRBC_NO_TIMER
# define hal_init() ((void)0)
# define hal_enable_irq() (_builtin_rx_clrpsw(T))
# define hal_disable_irq() (_builtin_rx_setpsw(T))
# define hal_idle_cpu() (_builtin_rx_wait())
                                               // set_ipl(0)
// set_ipl(15)
#else // MRBC_NO_TIMER
# define hal_init()
                   ((void)0)
# define hal_enable_irq() ((void)0)
# define hal_disable_irq() ((void)0)
# define hal_idle_cpu() ( R_DELAY_Us(1000, 50000), mrbc_tick())
#endif
```

ł

}

}

int hal write(int fd, const void *buf, size t nbytes);

```
38
```

int hal_flush(int fd);

#ifdef __cplusplus
}
#endif
#endif // ifndef MRBC_HAL_H_

#ifdef __cplusplus } #endif #endif // ifndef MRBC_HAL_H_

> 6.クリーンインストール及び実行。 4.3-1 の7及び8の手順通りにコンパイルし実行し、、LED1 が 0.2 秒間隔で点滅する事を確認する。

4.3-3 Chapter03「複数の mruby プログラムを同時に動かす」を e2studio と gcc for Renesas と RX210 の環境 で動かす

itoc チュートリアル「Chapter02「LED 点滅の速さを変える」」と 4.2-3 を参考にして、4.3-2 を修正していく。 コンパイルの方法及び mruby/c には、4.3-2 と違いが無い為、、「FlashLED.c」のみ掲載する。 コンパイル及び実行を行い、LED の点滅及びターミナルへの出力を確認する。

UARTの送受信を簡単にする為、itoc様の「UART wrapper for PSoC5LP」をRX用に移植させて頂いた。

また、自動生成されるソース「inthandler.c」の 318 行目,429 行目に使用している割り込みベクタの設定(下記)が 有り、重複するのでコメントアウトする。

// TMR0 CMIA0
void Excep_TMR0_CMIA0(void){}
// SCI0 RXI0
void Excep_SCI0_RXI0(void){ }
// SCI0 TXI0

void Excep_SCI0_TXI0(void){ }

「FlashLED.c」

extern char sample1[]; extern char sample2[];

(略)

UART_HANDLER uh;

(略)

int main()

uart_init( &uh );

clock_setup(); gpio_setup(); sci_setup(); timer_setup(); lowpower_setup();

/* Place your initialization/startup code here (e.g. MyInst_Start()) */ mrbc_init(memory_pool, MEMORY_SIZE); mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read); mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);

 $mrbc_create_task( \ sample1, \ 0 \ ); \\ mrbc_create_task( \ sample2, \ 0 \ ); \\$ 

mrbc_run();

return 0;

}

(略)

{

}

```
void sci_setup( void )
    // SCI0
   // sold
// write protect disable
SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) | 0x02 );
MSTP(SCI0) = 0; // SCI0 ON
// write protect enable
    SYSTEM.PRCR.WORD = (uint16_t)(( 0xA500 |SYSTEM.PRCR.WORD) & (~0x02));
    SCI0.SCR.BYTE = 0;
    SCI0.SMR.BYTE = 0;
SCI0.SEMR.BYTE = 0x10;
    SCI0.BRR = 0 \times 50;
    for(int i = 0; i < 160; i++)
    {
                 /* Wait 13ms. See comment above for why. */
                nop();
    }
    // SCIO 割り込み設定
                                             // 割り込み優先度 3
// 割り込み有効化
// 割り込みフラグクリア
    IPR( SCI0,TXI0 ) = 1;
IEN( SCI0,TXI0 ) = 1;
IR( SCI0,TXI0 ) = 0;
    SCI0.SCR.BYTE = 0xA0;
                                                 // 送信割り込み及び送信イネーブル
    // write protect disable
MPC.PWPR.BIT.B0WI = 0;
MPC.PWPR.BIT.PFSWE = 1;
    /* TxD setup */
   /* TXD Setup */
PORT2.PMR.BIT.B0 = MODE_GPIO; // RXD_PMR 一旦 GPIO にしてから
MPC.P20PFS.BYTE = 0x0A; // TXD0 SELECT TXD を選択し
PORT2.PDR.BIT.B0 = GPIO_OUTPUT; // TXD_PDR ポート方向を変えて
PORT2.PMR.BIT.B0 = MODE_PERIPHERAL; // TXD_PMR ペリフェラルに設定する。
    /* RxD setup */
   /* RXD setup */
PORT2.PMR.BIT.B1 = MODE_GPIO; // RXD_PMR 一旦 GPIO にしてから
MPC.P21PFS.BYTE = 0x0A; // RXD0 SELECT RXDを選択し
PORT2.PDR.BIT.B1 = GPIO_INPUT; // RXD_PDR ポート方向を変えて
PORT2.PMR.BIT.B1 = MODE_PERIPHERAL; // RXD_PMR ペリフェラルに
                                                                                                                    ペリフェラルに設定する。、
    // write protect enable
MPC.PWPR.BIT.B0WI = 0;
MPC.PWPR.BIT.PFSWE = 0;
    MPC.PWPR.BIT.B0WI = 1;
```

# 5.測定

4.1-1、4.1-2、4.2-1、4.2-2、4.3-1、4.3-2 それぞれの LED 点滅時間を計測する。また、CPU の電源にシャント抵抗 (10 Ωを入れ消費電流の時間変化も測定した。



### 図 5.1-1.1 LED の点滅

### 4.1-11秒毎の点滅。周期は、2.0132秒

4		MSO-X 40	)24A, MY5435	50478, 04.00.201	4101303: Fri Mar	23 10:37:42 2018
	■ 1 50mV/ 101.250mV 2	З	4	H 200.0ms/ 0.0s	T F 1 Trig'd'	147m∨ ?
						※ 測定 ※ 目 最大(1): 235m∨ 最小(1):
					a for a second	180m∨ +
⊤►						
ţ			-			

#### 図 5.1-1.2 電流の時間変化



図 5.1-1.3 電流の時間変化及び LED の点滅

### 4.1-11 秒毎の点滅。電流は、21.5mA 平均で、ピークが 23mA が1Hz毎に発生している。



図 5.1-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

4.1-1 1 秒毎の点滅。ピークの形状。LED をトグルするロジックは、224 µsec 程度で動作している。



#### 図 5.1-2.1 LED の点滅

#### 図 5.1-2.2 LED の点滅(拡大)

4.1-2 0.2 秒毎の点滅。周期は、0.39995 秒



図 5.1-2.3 電流の時間変化



図 5.1-2.4 電流の時間変化及び LED の点滅

### 4.1-2 0.2 秒毎の点滅。12mA 平均で、200msec 毎にピークの16mA が発生している。



図 5.1-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大)

^{4.1-2 0.2} 秒毎の点滅。ピークの形状。LED をトグルするロジックは、124 µsec 程度で動作している。

12000m/2 3 4 日 500.0ms/ 日 1 2.93V 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 120.00m/2 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.93V 12.9		MSO-X 4	1024A, MY54350478	3, 04.00.20141013	303: Thu Mar 22	2 15:33:23 2018
	2.00V/ 2 120.00mV	3	4	H 500.0ms/ 15.00us	▼ 1 停止	2.93V
					# PPP 周辺 正 見 の の の の	第定 単 目 (1): 5.22√ 489.87mHz (ルス幅(1): 1.0002s (ルス幅(1): 1.0003s 和): 2.0005s

#### 図 5.2-1.1 LED の点滅

### 4.2-11秒毎の点滅。周期は、2.0005秒



図 5.2-1.2 電流の時間変化



図 5.2-1.3 電流の時間変化及び LED の点滅

### 4.2-11 秒毎の点滅。電流は、21.3mA 平均で、ピークが 23mA が1Hz毎に発生している。



図 5.2-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

^{4.2-11} 秒毎の点滅。ピークの形状。LED をトグルするロジックは、142 µsec 程度で動作している。

	MSO-X 4024A, MY5	4350478, 04.00.20141	L01303: Thu Mar 22 15:34:30 2018		MSO-X 4024A,	MY54350478, 04.00.201410	1303: Thu Mar 22 15:34:41 2018
1 2.00V/ 120.00mV 2	3 4	H 500.0ms/ 15.00us	T 1 2.93∨ □ □ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	1 2.00V/ 120.00mV 2	3 4	H 100.0ms/ 15.00us	T 1 2.93∨ □ 停止 △√
			■ 測定 = p-p(1) 5.22V 別波数(3) 正/い入稿(1) 2000ms 別別(1) 4000ms 小(い入句(1) 4000ms				■ 潮定 ■ p-p0. 5.22V 周速数(1) 2.5003Hz 199.97ms p1(9.2460) 199.98ms 周期(3) 399.96ms ↓

図 5.2-2.1 LED の点滅

図 5.2-2.2 LED の点滅(拡大)

### 4.2-2 0.2 秒毎の点滅。周期は、0.39995 秒



図 5.2-2.3 電流の時間変化



図 5.2-2.4 電流の時間変化及び LED の点滅

### 4.2-2 0.2 秒毎の点滅。11.5mA 平均で、200msec 毎にピークの 15mA が発生している。



図 5.2-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大) 4.2-2 0.2 秒毎の点滅。ピークの形状。LED をトグルするロジックは、120 µsec 程度で動作している。

	MSO-X 4024A, MY54350478,	04.00.2014101303: Thu Ma	r 22 15:38:12 2018
■ 1 2.00V/ ■ 1 120.00mV 2 3	4 H	500.0ms/ T 卍 15.00us (停.	1 2.93V <b>□</b> 止 △√
		15.00us (*.	は 単 携約室 単 ■ 携約室 単 ■ 第2 4 ■ 第3章 499.14mHz 至750-2 %(1): 1.0017s ●750-2 %(1): 1.0018s ○1月(1): 2.0035s ◆

図 5.3-1.1 LED の点滅

## 4.3-11秒毎の点滅。周期は、2.0035秒

4	K		GIES		М	50-X 402	4A, MY54	350478, (	04.00.20	14101303	: Fri	Mar 2	23 12:07	:31 2018
		1 50 101.2	m∨/ 250m∨ (	2	З		4	Н	200.0ms/ 0.0s	[	Ŧ f	2 停止	-1.70V	
	er er beker	dahala Jan Lar	d d d d a ser a		al donate da morte			la sa sa dalah da		terestelan deder		… —— —— - 最	測定 (大(1): (小(1):	∷ 🔳 235m∨ 150m∨
	u l	<u>. 11 11</u>	, lin line ( , d	l de <mark>l'altri paper de la</mark>	n <mark>dagi kili</mark> nga ki	Linealdite	-   			لى بىللى لەر ا			+	
						- - - - - - - - - - - - - - - - - - -	     					+		
Ţ														

図 5.3-1.2 電流の時間変化

1 50mV/ 101.250mV	2 2.00V/ 1.80000V 3	4	H 200.0ms/ 598.0ms	T <u></u> 5 2 下 停止	-1.70V
				Y2 Y1	μ-νμ     Ε <b>f</b> 動       X1(1):       -2.000000000ms       X2(1):       1.002000000000s       ΔX:       1.004000000000s       1/ΔX:       996.02mHz       Y1(1):       229.375mV       ΔY:       14.375mV       ΔΥ/ΔX:       14.3177mV/s
 ×	લે 🗌	-	(X2)		

図 5.3-1.3 電流の時間変化及び LED の点滅

### 4.3-11 秒毎の点滅。電流は、21.5mA 平均で、ピークが 23mA が1Hz毎に発生している。



図 5.3-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

4.3-11 秒毎の点滅。ピークの形状。LED をトグルするロジックは、216 µsec 程度で動作している。

	MSO-X 4024A	, MY54350478, 04.00.20141	101303: Thu Mar 22 15:39:19 2018		M50-X 4024A, MY54	350478, 04.00.20141013	03: Thu Mar 22 15:39:28 2018
2.00V/ 2 120.00mV	3 4	H 500.0ms/ 15.00us	T 〒 1 2.93∨ □□ 停止 △▽	2.00V/ 120.00mV 2	3 4	H 100.0ms/ 15.00us	T 1 2.93V □ 停止 △√
			■ 潮定 ■ = = = = = = = = = =				■ 測定 ■ 目 p=p13 5.22V 刷注数(1) 2.2003H2 正ビ(ルス%(1) 199.97ms 用(1)- 月(ルス%(1) 399.96ms 同時(1) 399.96ms

図 5.3-2.1 LED の点滅

図 5.3-2.2 LED の点滅(拡大)

4.3-2 0.2 秒毎の点滅。周期は、0.39995 秒



図 5.3-2.3 電流の時間変化



図 5.3-2.4 電流の時間変化及び LED の点滅

### 4.3-2 0.2 秒毎の点滅。11.5mA 平均で、200msec 毎にピークの 17mA が発生している。



図 5.3-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大) 4.3-2 0.2 秒毎の点滅。ピークの形状。LED をトグルするロジックは、238 µsec 程度で動作している。

表 5-1 コンパイラ種類及び周期

コンパイラ	CC-RX	CC-RX	gcc for Renesas
ドライバ	FIT	なし	なし
1 周期の時間(busyloop)	2.0132 秒	2.0005 秒	2.0035 秒
1 周期の時間(タイマー)	0.39995 秒	0.39995 秒	0.39995 秒

計測の結果、CC-RX とgccで同じ delay 関数を使った場合でも時間に差が出る事が分かった。

これは、関数呼び出しのオーバヘッドの違いによって起きていると思われる。

また同じコンパイラでも、FIT ドライバを使った方が、時間が掛かっている。

また、タイマーを使った場合は、コンパイラによらず時間は一定であった。

表	5-2	コンパイラ種類及び消費電流	
---	-----	---------------	--

コンパイラ	CC-RX	CC-RX	gcc for Renesas
ドライバ	FIT	なし	なし
電流 平均値(Busyloop)	21.5mA	21.3mA	21.5mA
電流 最大値(Busyloop)	23mA	23mA	23mA
電流 平均値(タイマー)	12mA	11.5mA	11.5mA
電流 平均値(タイマー)	16mA	15mA	17mA

消費電流に関しては、Busyloop とタイマー利用では2倍の差が出ているが、コンパイラやドライバでの差はほぼ見られなかった。

これは、消費電力が今回のテストでは、コンパイラによる違いでは起きなくて、ハードウェアによるものが殆どてあったからだと思われる。

## 6.まとめ

Renesas 社製1 chip CISC マイコン RX210 にて mruby/c のチュートリアルの動作が確認出来た。

gcc for Renesas では、mruby/cのコードの変更は不要であったが、「RX Srandard Toolchain」の C 開発環境は、 C99 対応であるが、可変長配列に対応していないため、mruby/cのコードに修正が必要となった。

CC-RXとgcc for Renesas では、今回のチュートリアルに関して消費電力に対する差は見られなかった。

FIT(Firmware Integration Technology)を使うと、1chipの周辺機器を簡単に扱う事が出来る上、移植性も上がる。 ただし、コンパイラが CC-RX のみの対応となる。