# 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 へインポートする。

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。

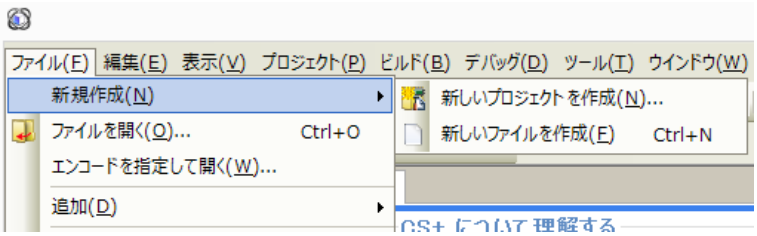


図 4.1-1.1-1 新規プロジェクト作成

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。

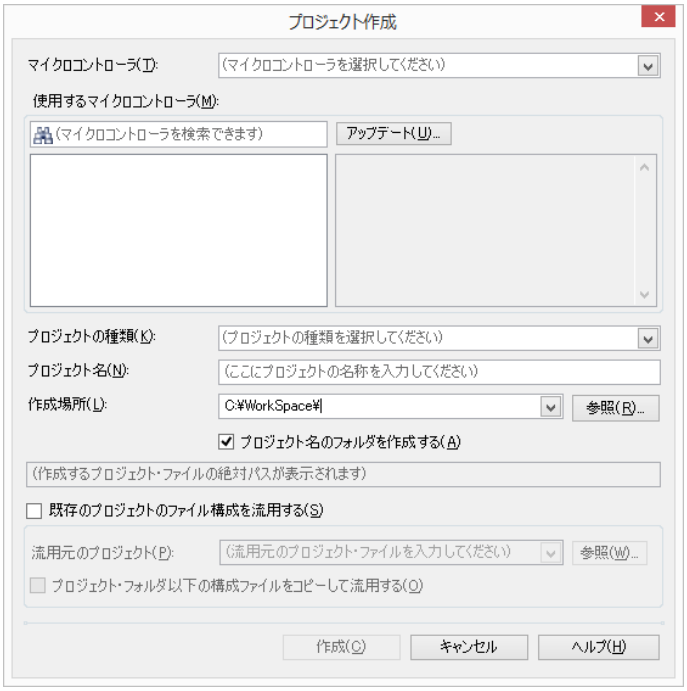


図 4.1-1.2-1 プロジェクト作成 設定前

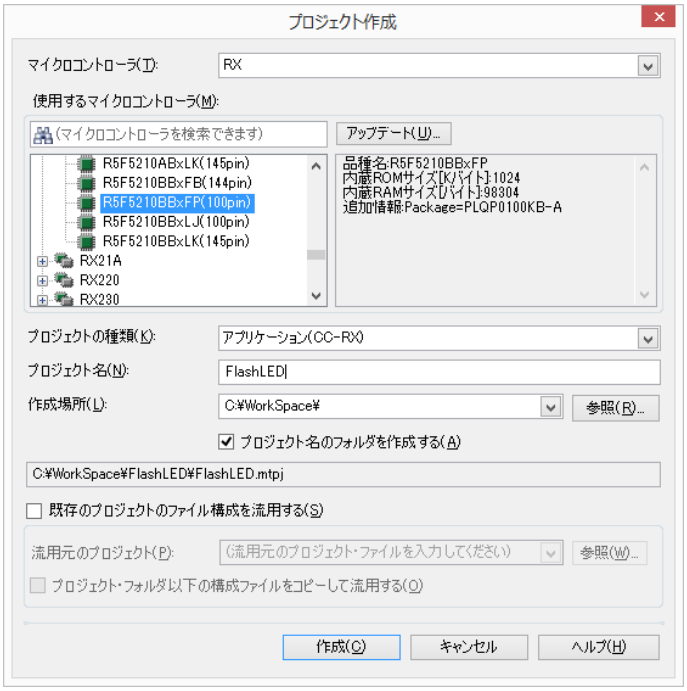


図 4.1-1.2-2 プロジェクト作成 設定後

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。

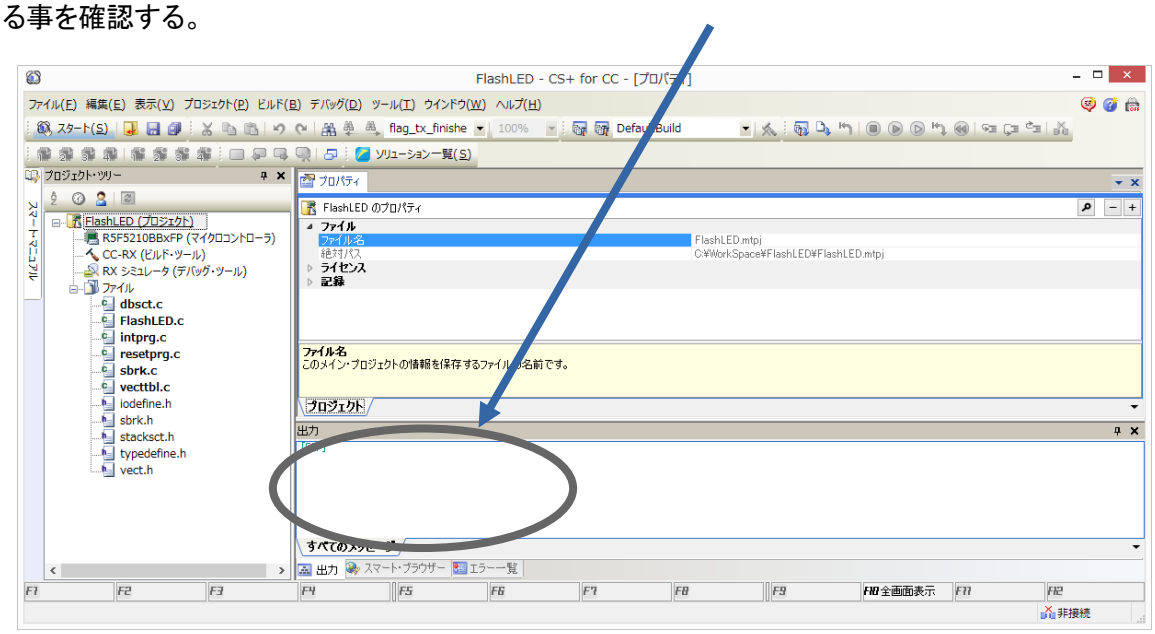


図 4.1-1.3-1 テストビルドの成否確認

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。

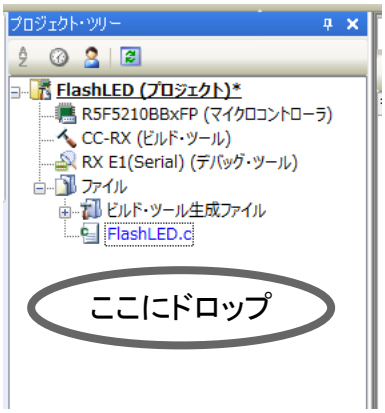


図 4.1-1.4-1 mruby/cソースコピー

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。

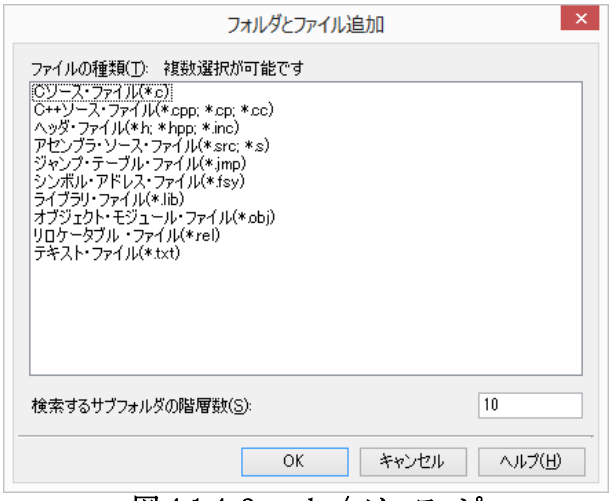


図 4.1.4-2 mruby/cソースコピー

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


/***** Feature test switches ************************************************/
/***** System headers *******************************************************/


/***** Local headers ********************************************************/
#include "r_gpio_rx_if.h"

/***** Constant values ******************************************************/
/***** Macros ***************************************************************/

#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


/***** Typedefs *************************************************************/
/***** Global variables *****************************************************/
/***** Function prototypes **************************************************/
int hal_write(int fd, const void *buf, int nbytes);
int hal_flush(int fd);


/***** Inline functions *****************************************************/


#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。

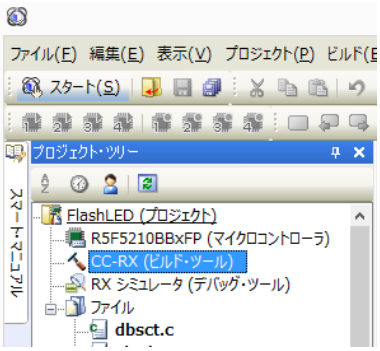


図 4.1-1.8-1 マクロ定義手順 1

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。

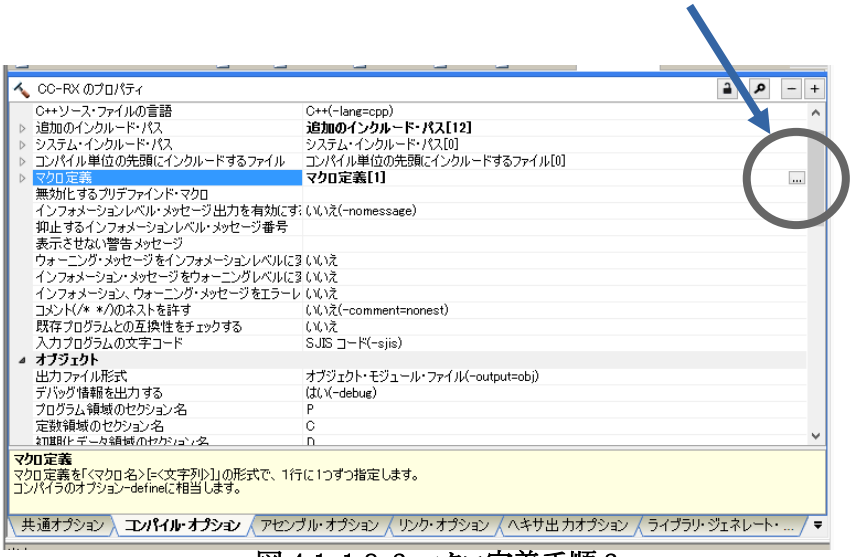


図 4.1-1.8-2 マクロ定義手順 2

ダイアログに「MBRC_NO_TIMER」を登録します。

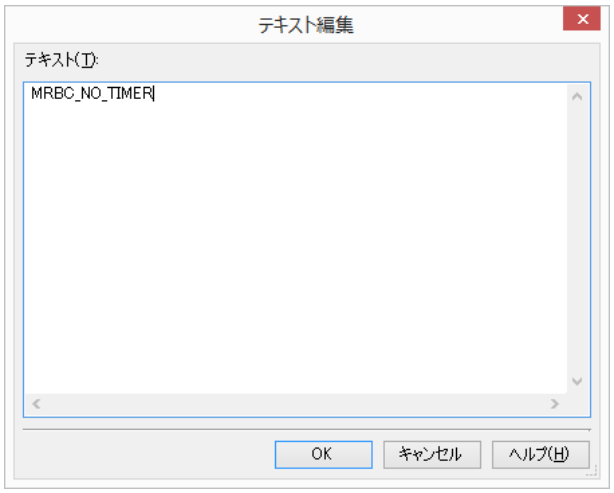


図 4.1-1.8-3 マクロ定義手順 3

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。

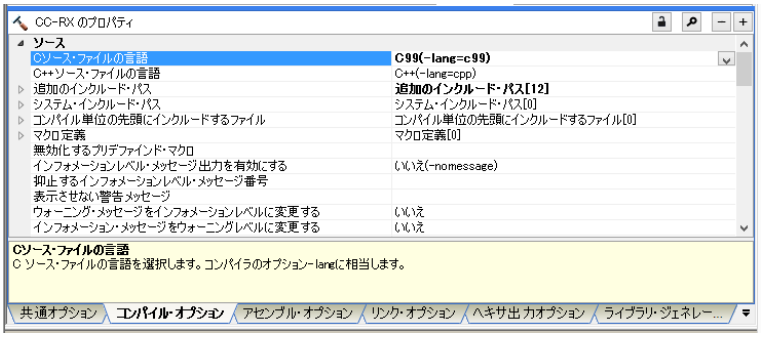


図 4.1-1.9-1 C99 設定1

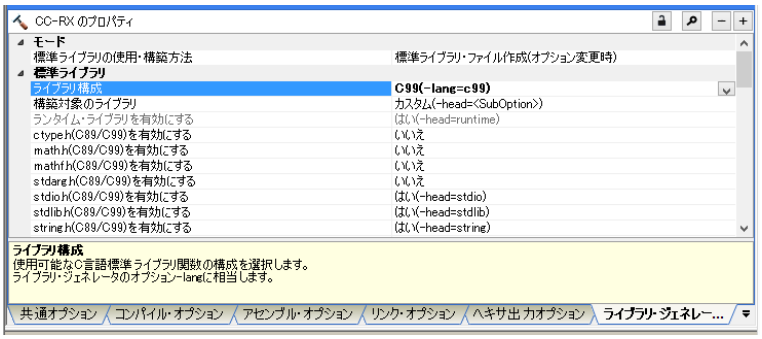


図 4.1-1.9-2 C99 設定 2

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。

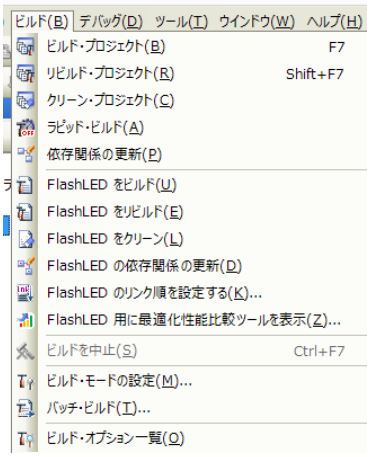


図 4.1-1.11-1 ビルドメニュー

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)」

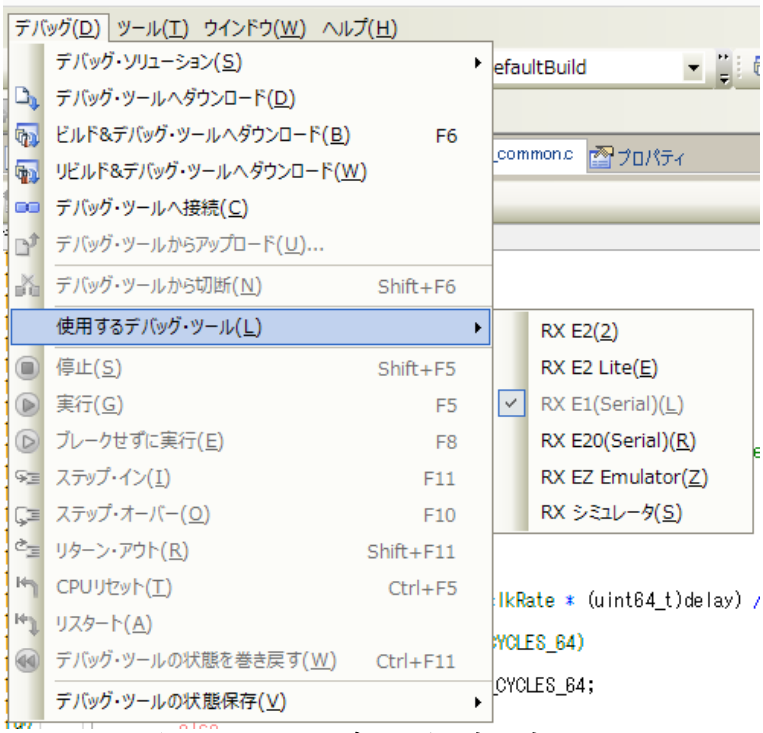


図 4.1-1.12-1 デバッガ設定設定1

・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。

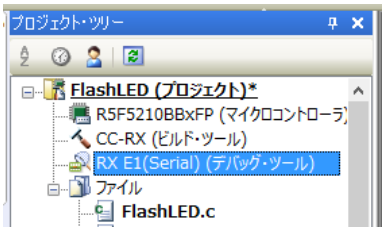


図 4.1-1.12-2 デバッガ設定設定 2

プロパティの画面で、

「メインクロック」を「20」MHz

「動作周波数」を「50」MHz

「エミュレータからの電流供給をする」を「はい」

「供給電圧[V]」を「5.0V」

へ変更する。

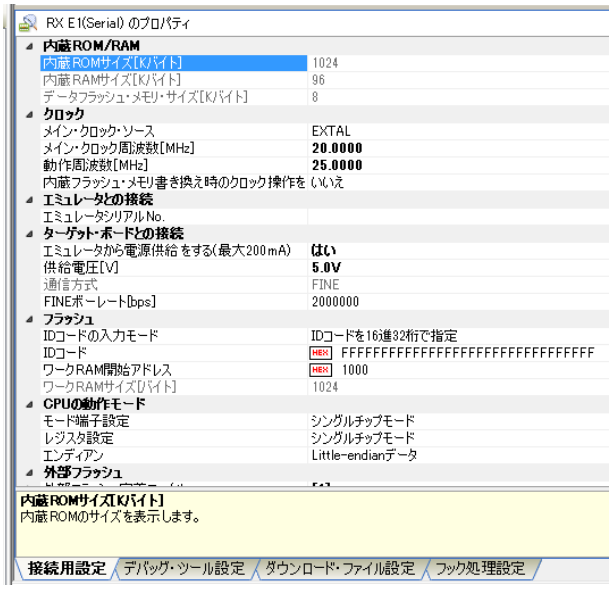


図 4.1-1.12-3 デバッガ設定設定 3

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。

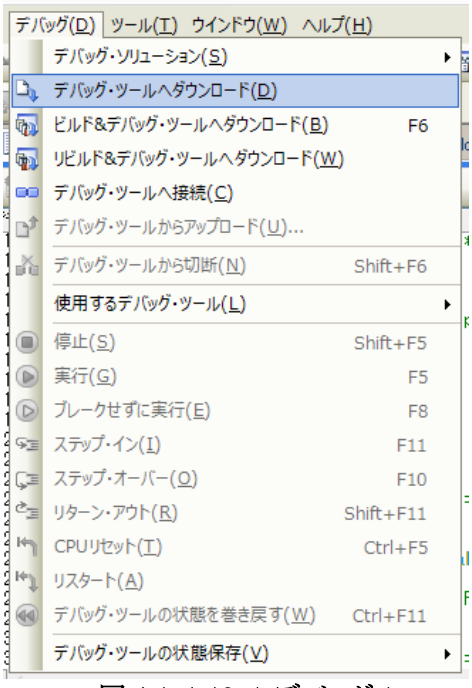


図 4.1-1.13-1 デバッグ 1

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。

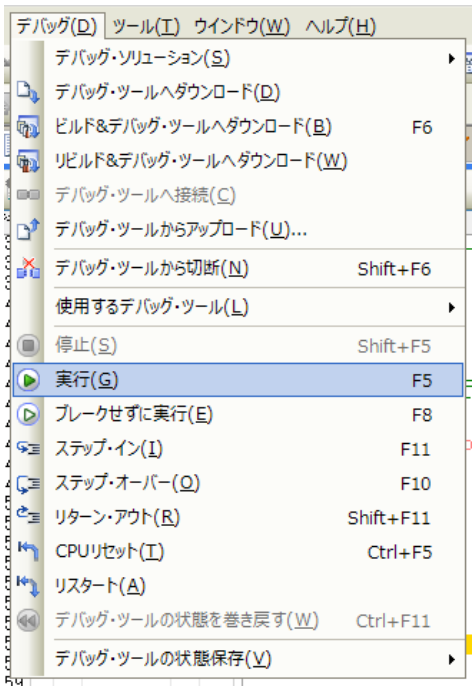


図 4.1-1.13-2 デバッグ 2

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。

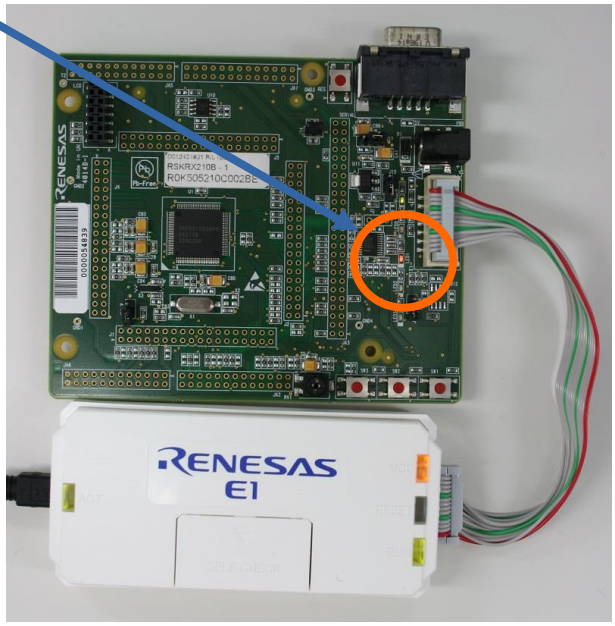


図 4.1-1.13-3 デバッグ 3

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。

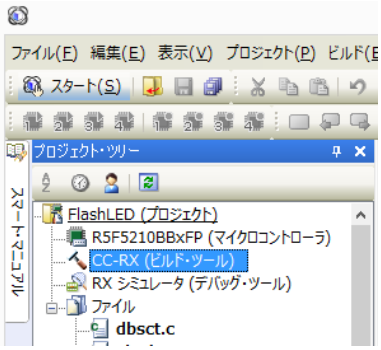


図 4.1-2.1-1 自動コンパイルスクリプト追加 1

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。

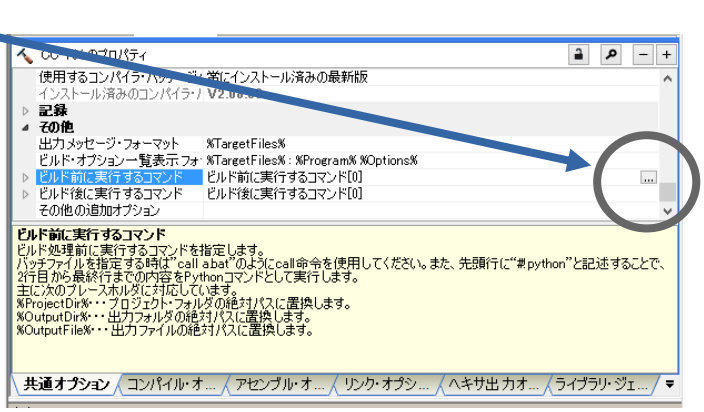


図 4.1-2.1-2 自動コンパイルスクリプト追加 2

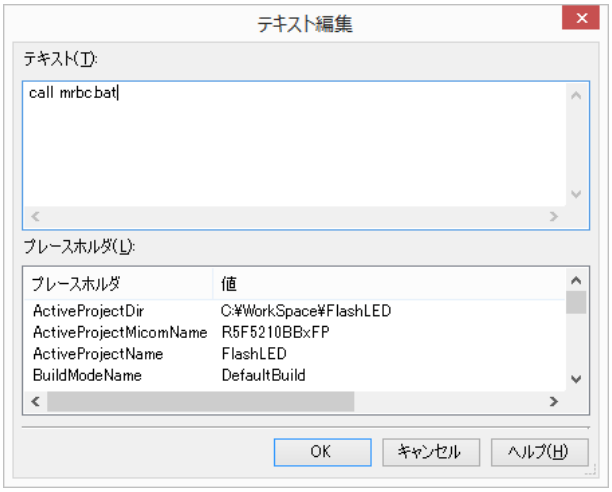


図 4.1-2.1-3 自動コンパイルスクリプト追加 3

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

/***** Feature test switches ************************************************/
/***** System headers *******************************************************/
#include <machine.h>

/***** Local headers ********************************************************/
/***** Constant values ******************************************************/
/***** Macros ***************************************************************/

#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_BSP_SoftwareDelay(1, BSP_DELAY_MILLISECS), mrbc_tick())

#endif


/***** Typedefs *************************************************************/
/***** Global variables *****************************************************/
/***** Function prototypes **************************************************/
int hal_write(int fd, const void *buf, int nbytes);
int hal_flush(int fd);


/***** Inline functions *****************************************************/


#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) API
#include "r_cmt_rx_if.h"            // 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) ) {
    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_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)
{
        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。

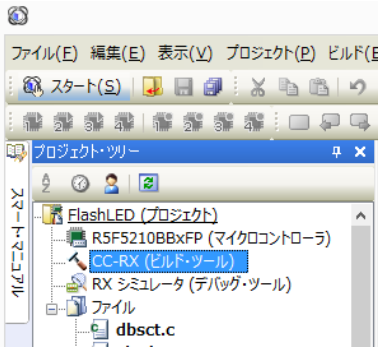


図 4.1-2.4-1 マクロ定義削除手順 1

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。

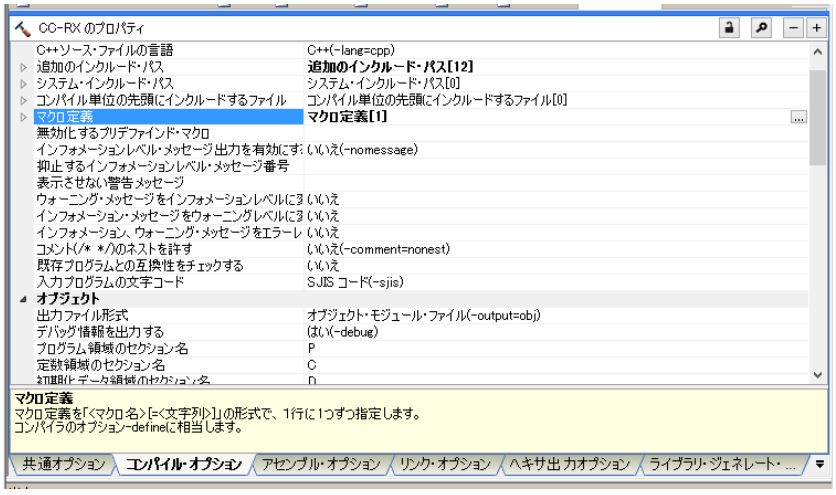


図 4.1-2.4-2 マクロ定義削除手順 2

ダイアログの「MBRC_NO_TIMER」を削除し登録します。

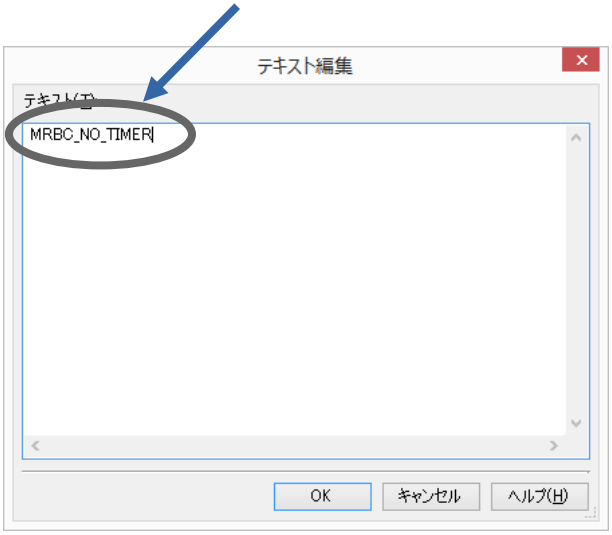


図 4.1-2.4-3 マクロ定義削除手順 3

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)
{
  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;
}

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_ALL_MODULE_STOP);
  if (LPC_SUCCESS != lpc_request_status) {
    LED0 = LED_ON;
    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.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。


図 4.1-3.5-1 ターミナル画面

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


/***** Feature test switches ************************************************/
/***** System headers *******************************************************/
#include <machine.h>

/***** Local headers ********************************************************/
#include "r_delay.h"

/***** Constant values ******************************************************/
/***** Macros ***************************************************************/

#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


/***** Typedefs *************************************************************/
/***** Global variables *****************************************************/
/***** Function prototypes **************************************************/
int hal_write(int fd, const void *buf, int nbytes);
int hal_flush(int fd);


/***** Inline functions *****************************************************/


#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 {
            PORT1.PODR.BIT.B5 = LED_OFF;              // LED1
    }
}


//================================================================
/*! 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;

    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();
    }

    SYSTEM.SCKCR.LONG = 0x21821211;  // PCLKB 100MHz/4 = 25MHz PCLKD 100MHz/2 = 50MHz SYSTEM 100MHz/2 = 50MHz
    SYSTEM.SCKCR3.WORD = ((uint16_t)4) << 8;
    SYSTEM.LOCOCR.BYTE = 0x01;

    // write protect enable
    SYSTEM.PRCR.WORD = 0xA500;
}

void gpio_setup( void )
{
    PORT1.PODR.BIT.B4 = LED_OFF;        // OUT_LED0 DATA
    PORT1.PODR.BIT.B5 = LED_OFF;        // OUT_LED1 DATA
    PORT1.PODR.BIT.B6 = LED_OFF;        // OUT_LED2 DATA
    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

    PORT3.PDR.BIT.B1 = GPIO_INPUT;      // SW1_PDR
    PORT3.PDR.BIT.B3 = GPIO_INPUT;      // SW2_PDR
    PORT3.PDR.BIT.B4 = GPIO_INPUT;      // SW3_PDR

    PORT3.PDR.BIT.B1 = MODE_GPIO;       // SW1_PMR
    PORT3.PDR.BIT.B3 = MODE_GPIO;       // SW2_PMR
    PORT3.PDR.BIT.B4 = MODE_GPIO;       // SW3_PMR

    // write protect disable
    MPC.PWPR.BIT.B0WI = 0;
    MPC.PWPR.BIT.PFSWE = 1;

    /* 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 */
    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          ペリフェラルに設定する。、

    // write protect enable
    MPC.PWPR.BIT.B0WI = 0;
    MPC.PWPR.BIT.PFSWE = 0;
    MPC.PWPR.BIT.B0WI = 1;
}
```

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.TCCR.BYTE = 0;         // TMR1 NO FUNCTION
    TMR1.TCCR.BIT.CSS = 1;      // cascade 接続 16bit タイマー
    TMR1.TCCR.BIT.CKS = 2;      // PCLK/8
    TMR1.TCR.BIT.CCLR = 1;      // CMP A にてカウンターリセット
```

```
    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.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));
}
```

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

```
/***** System headers ********************************************************/
#include <machine.h>

/***** Local headers *********************************************************/
#include "iodefine.h"
#include "src/mrubyc.h"
#include "uart.h"

#include "sample1.c"
#include "sample2.c"

(略)

/***** Function prototypes ***************************************************/
void clock_setup( void );
void gpio_setup( void );
void timer_setup( void );
void lowpower_setup( void );
void sci_setup( void );

/***** Global variables ******************************************************/
/***** Local variables *******************************************************/
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 = 0x50;

    for(int i = 0; i < 160; i++)
    {
            /* Wait 13ms. See comment above for why. */
            nop();
    }

    // SCI0 割り込み設定

    IPR( SCI0,TXI0 ) = 1;           // 割り込み優先度 3
    IEN( SCI0,TXI0 ) = 1;           // 割り込み有効化
    IR( SCI0,TXI0 ) = 0;            // 割り込みフラグクリア

    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 */
    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 */
    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          ペリフェラルに設定する。、

    // write protect enable
    MPC.PWPR.BIT.B0WI = 0;
    MPC.PWPR.BIT.PFSWE = 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。

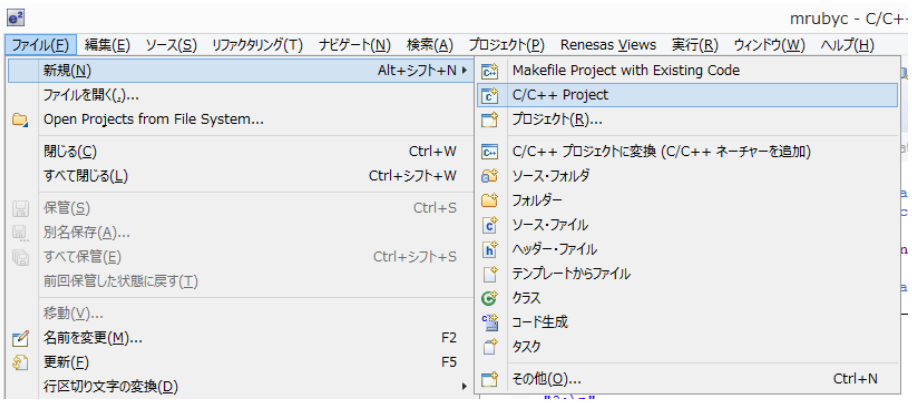


図 4.3-1.1-1 新規プロジェクト作成1

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

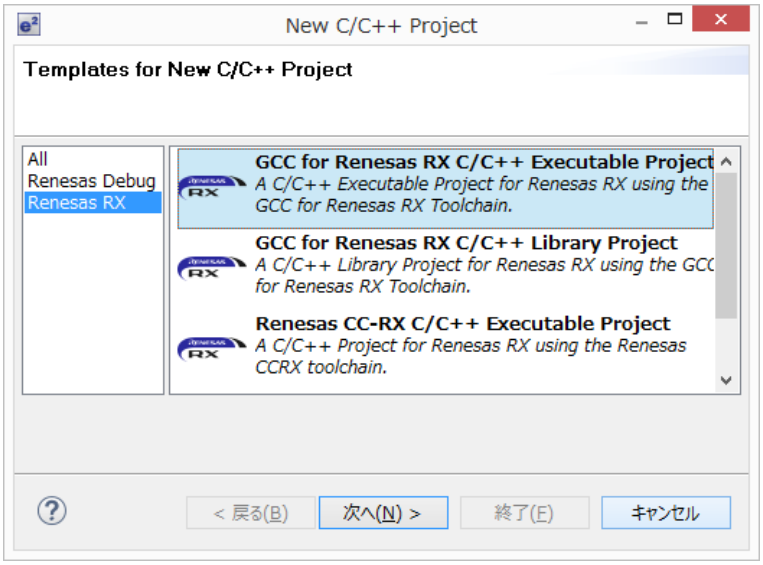


図 4.3-1.1-2 新規プロジェクト作成 2

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。

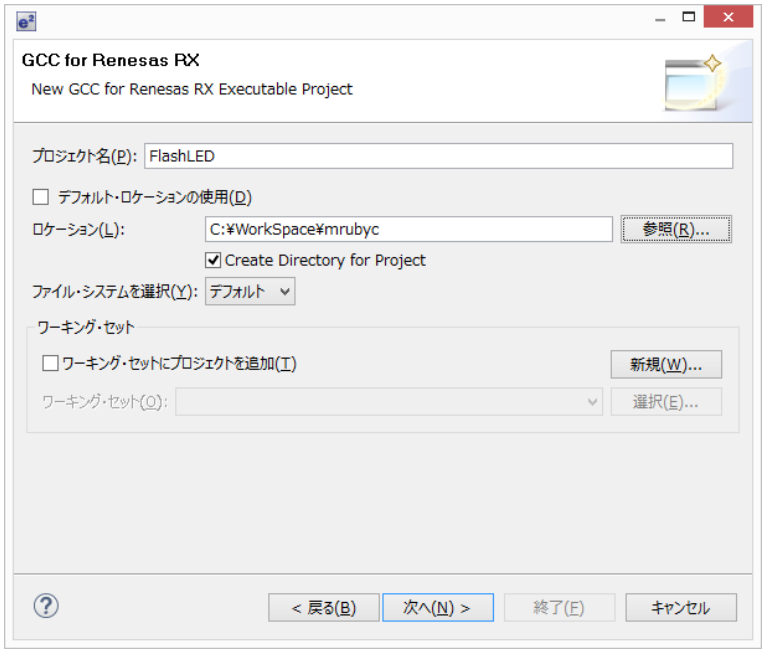


図 4.3-1.1-3 新規プロジェクト作成 3

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。

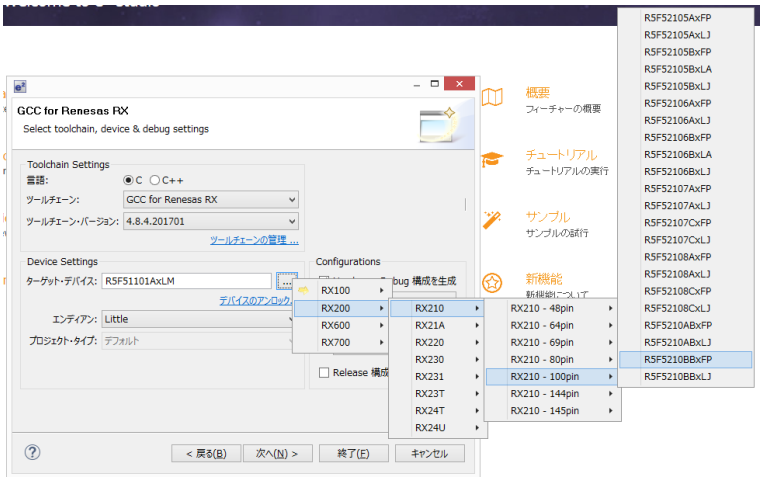


図 4.3-1.1-4 新規プロジェクト作成 4

「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。

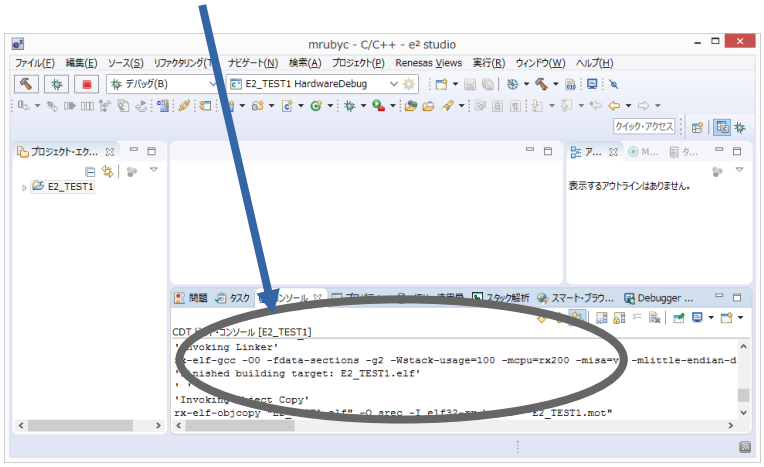


図 4.3-1.2-1 テストビルドの成否の確認

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)
{
_asm _volatile(
    "BRA     ?+¥n"
    "NOP¥n"
"?:¥n"
    "NOP¥n"
    "SUB     #01H,%0¥n"
    "BNE     ?-¥n"
:"=r"(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。

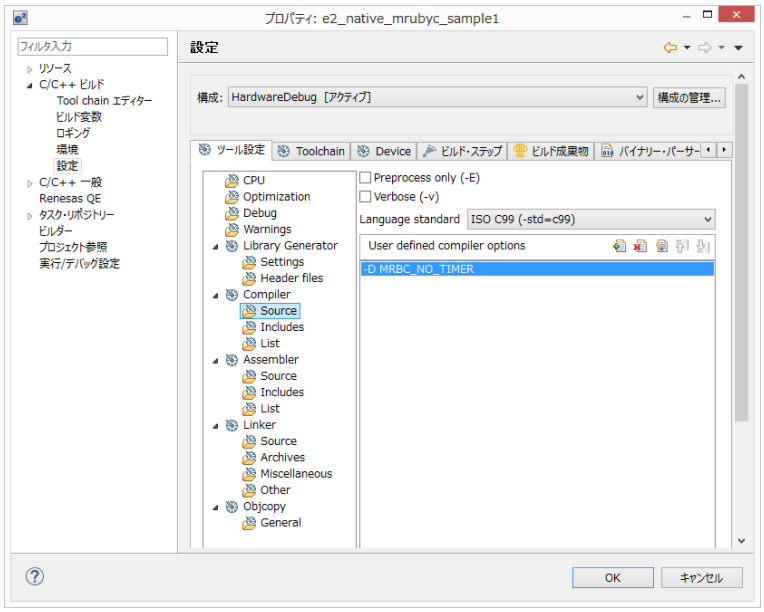


図 4.3-1.6-2 コンパイルオプション設定 2

7.クリーンビルドと実行
「プロジェクト」 -> 「クリーン」を選ぶ。

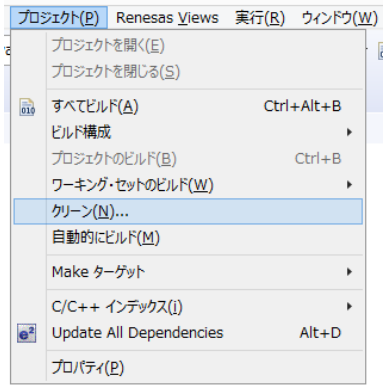


図 4.3-1.7-1 クリーンビルド 1

すると下記ダイアログがでるので、「OK」ボタンを押す。

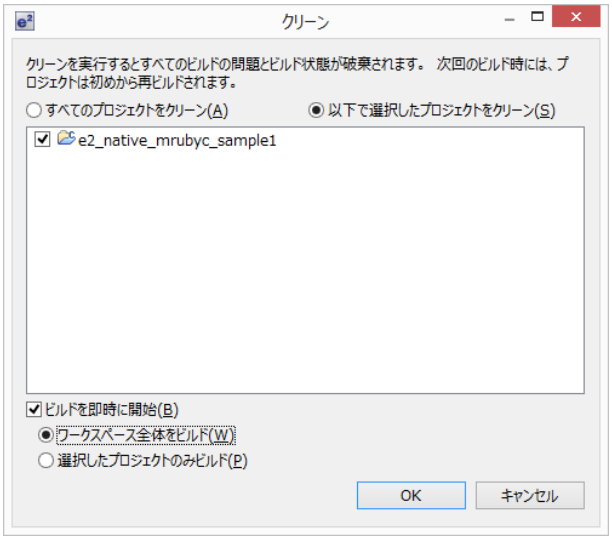


図 4.3-1.7-2 クリーンビルド 2

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。

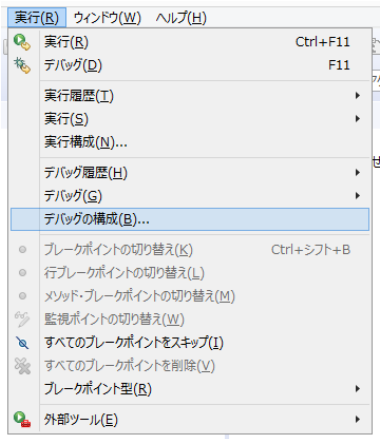


図 4.3-1.8-1 デバッガの設定 1

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。

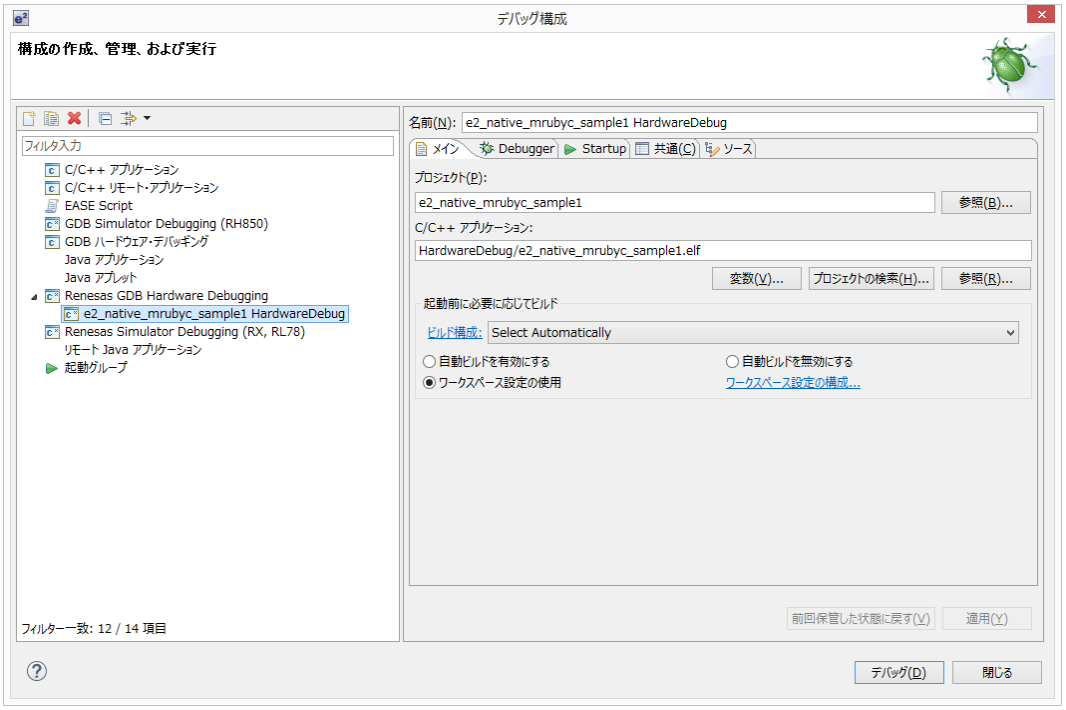


図 4.3-1.8-2 デバッガの設定 2

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

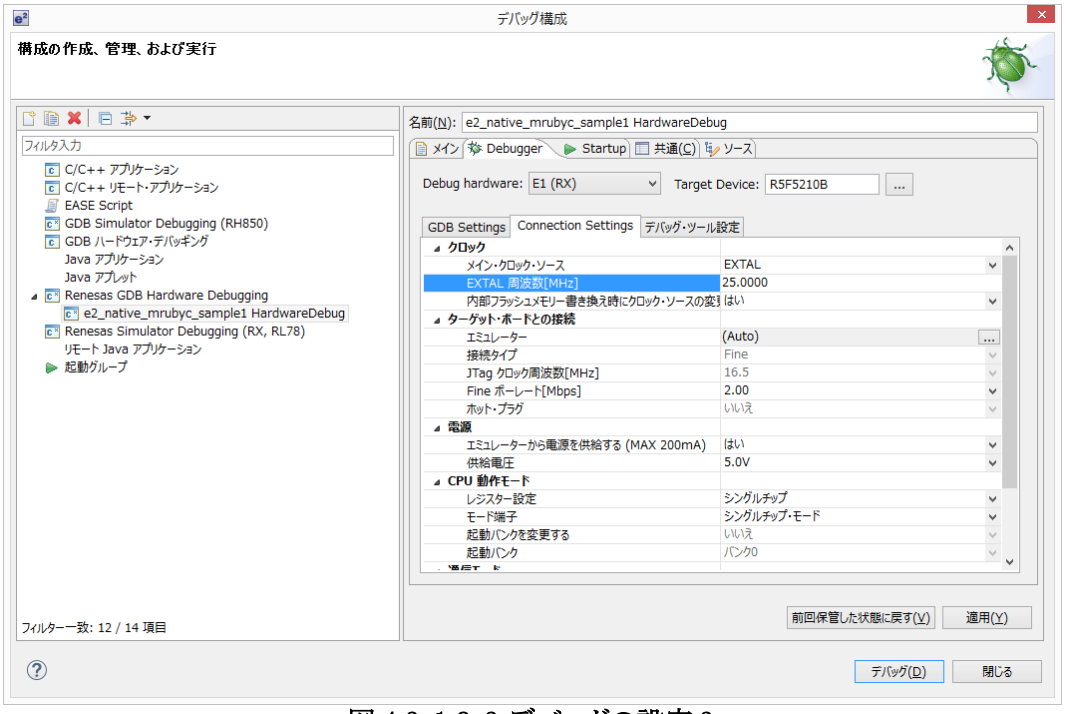


図 4.3-1.8-3 デバッガの設定 3

「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。

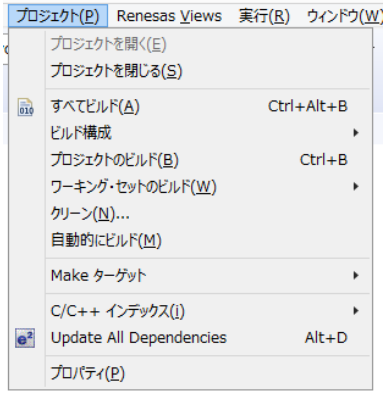


図 4.3-2.1-1 自動コンパイルスクリプトの追加 1

「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。

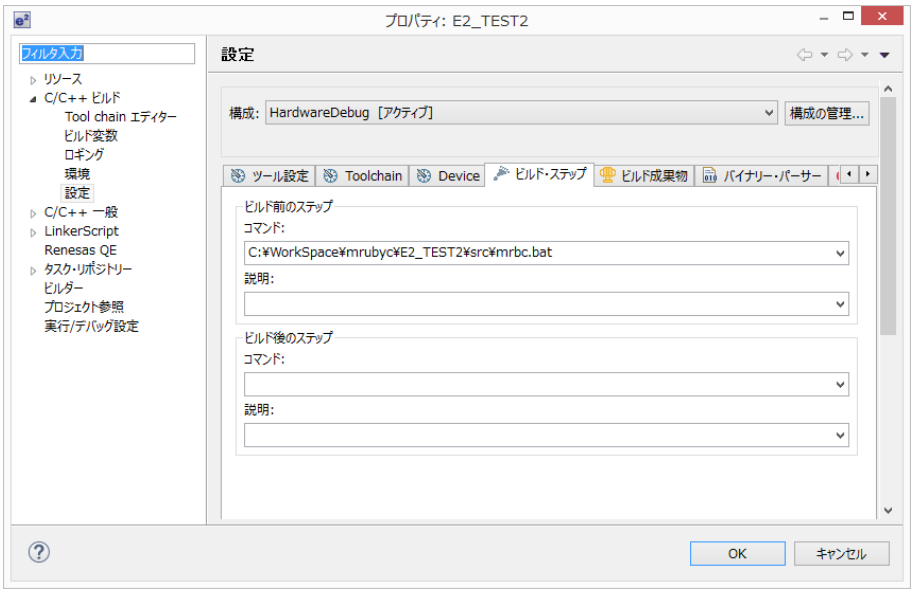


図 4.3-2.1-2 自動コンパイルスクリプトの追加 2

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。

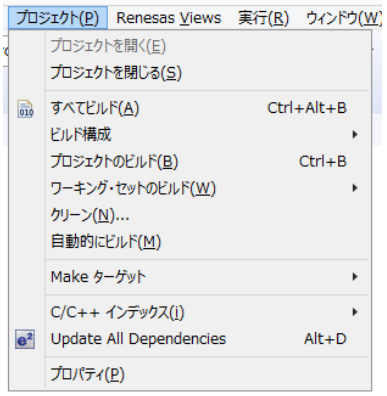


図 4.3-2.3-1 コンパイルオプション設定 1

「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。

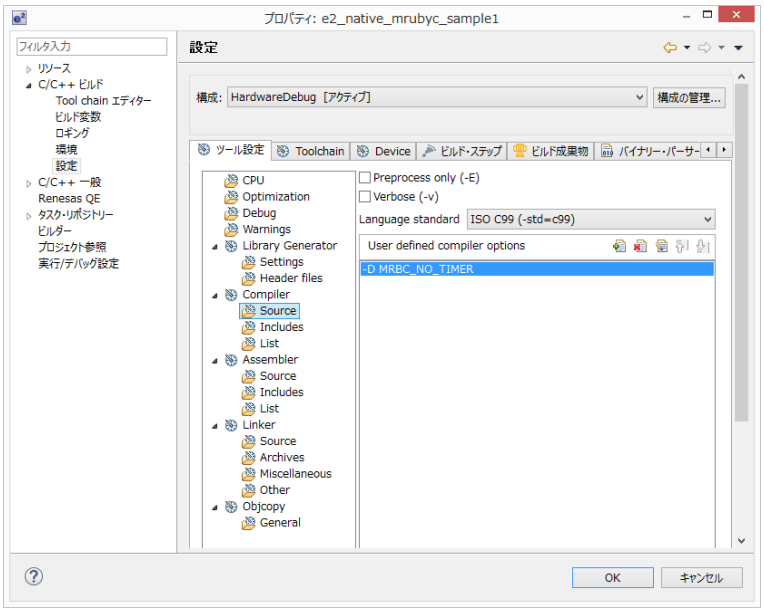


図 4.3-2.3-2 コンパイルオプション設定 2

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、「lnker_script.ld」を変更する。

「generate」を開いて、「lnker_script.ld」をダブルクリックする。

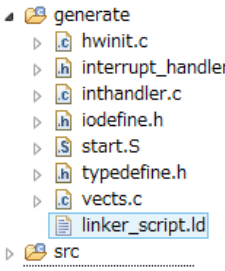


図 4.3-2.4-1 割り込み用スタックの増量1

「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。

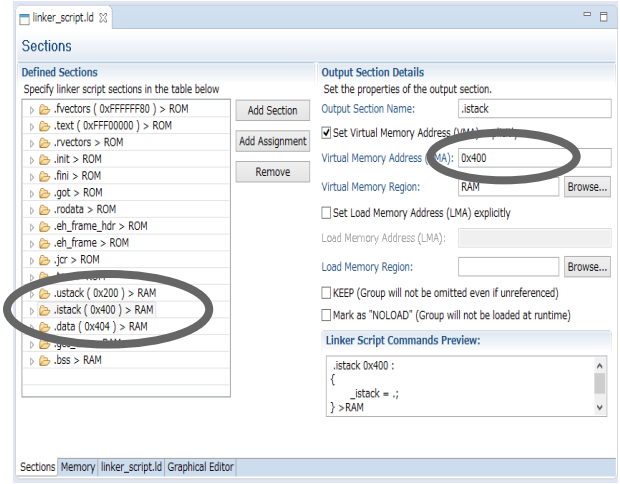


図 4.3-2.4-2 割り込み用スタックの増量 2

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。

「inthandler.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
   PUSH.L 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 = 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.TCCR.BYTE = 0;         // TMR1 NO FUNCTION
    TMR1.TCCR.BIT.CSS = 1;      // cascade 接続 16bit タイマー
    TMR1.TCCR.BIT.CKS = 2;      // PCLK/8
    TMR1.TCR.BIT.CCLR = 1;      // CMP A にてカウンターリセット

    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.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


/***** Feature test switches ************************************************/
/***** System headers *******************************************************/
/***** Local headers ********************************************************/
/***** Constant values ******************************************************/
/***** Macros ***************************************************************/

#ifndef MRBC_NO_TIMER
# define hal_init()        ((void)0)
# define hal_enable_irq()  (__builtin_rx_clrpsw('I'))          // set_ipl(0)
# define hal_disable_irq() (__builtin_rx_setpsw('I'))          // set_ipl(15)
# define hal_idle_cpu()    (__builtin_rx_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, 50000), mrbc_tick())

#endif


/***** Typedefs *************************************************************/
/***** Global variables *****************************************************/
/***** Function prototypes **************************************************/
int hal_write(int fd, const void *buf, size_t nbytes);
```

```
int hal_flush(int fd);

/***** Inline functions ****************************************************/

#ifdef _cplusplus
}
#endif
#endif // ifndef MRBC_HAL_H_

/***** Typedefs ************************************************************/
/***** Global variables ****************************************************/
/***** Function prototypes *************************************************/
int hal_write(int fd, const void *buf, int nbytes);
int hal_flush(int fd);

/***** Inline functions ****************************************************/

#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」

```
/***** Local headers **********************************************************/
#include "iodefine.h"
#include "mruby_c/mrubyc.h"
#include "uart.h"

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

(略)

/***** Function prototypes ****************************************************/
void clock_setup( void );
void gpio_setup( void );
void timer_setup( void );
void lowpower_setup( void );
void sci_setup( void );

/***** Global variables *******************************************************/
/***** Local variables ********************************************************/
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 = 0x50;

    for(int i = 0; i < 160; i++)
    {
            /* Wait 13ms. See comment above for why. */
            nop();
    }

    // SCI0 割り込み設定

    IPR( SCI0,TXI0 ) = 1;           // 割り込み優先度 3
    IEN( SCI0,TXI0 ) = 1;           // 割り込み有効化
    IR( SCI0,TXI0 ) = 0;            // 割り込みフラグクリア

    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 */
    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 */
    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(10Ωを入れ消費電流の時間変化も測定した。

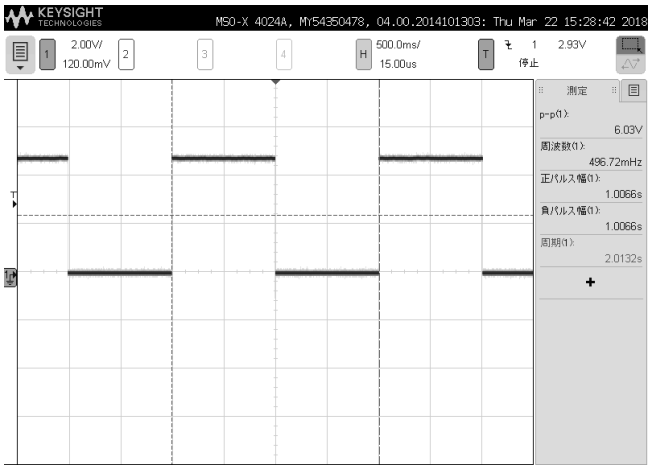


図 5.1-1.1 LED の点滅

4.1-1 1 秒毎の点滅。周期は、2.0132 秒

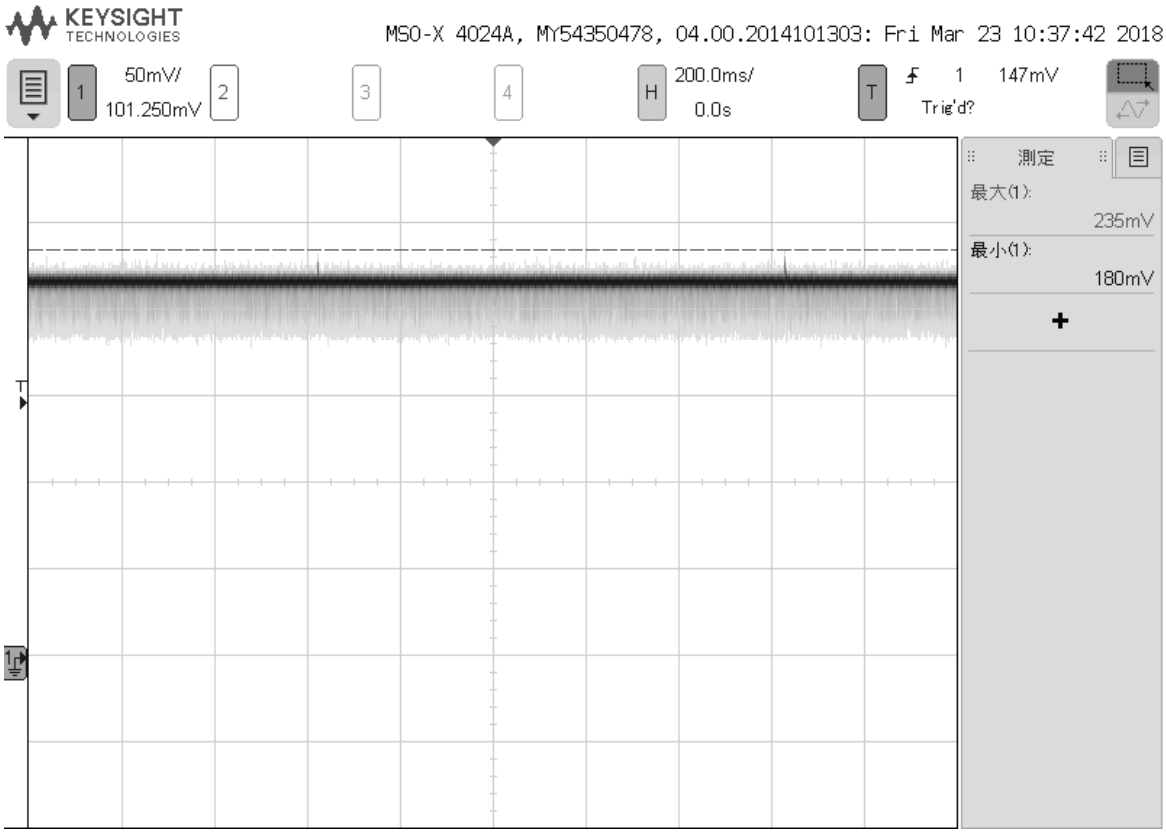


図 5.1-1.2 電流の時間変化

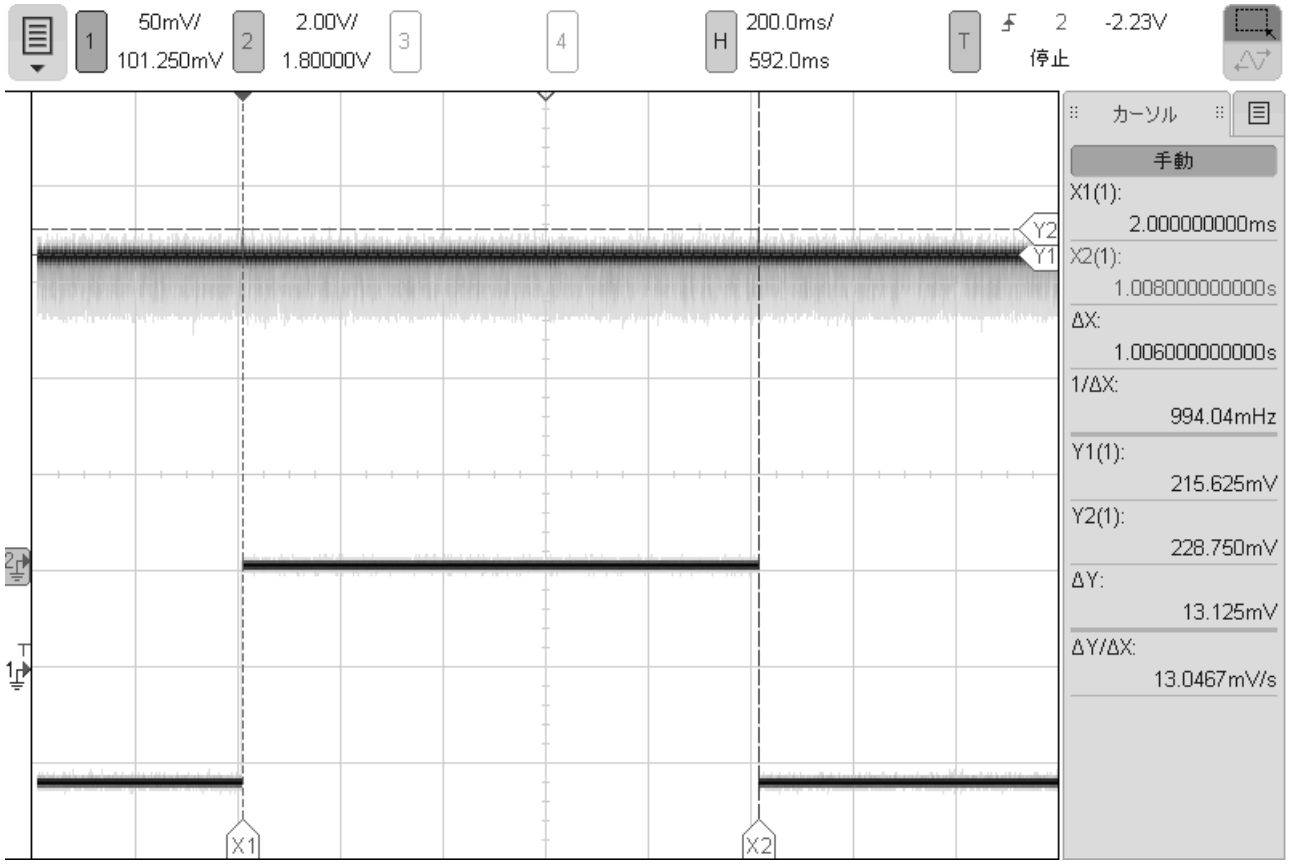


図 5.1-1.3 電流の時間変化及び LED の点滅

4.1-1 1 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。

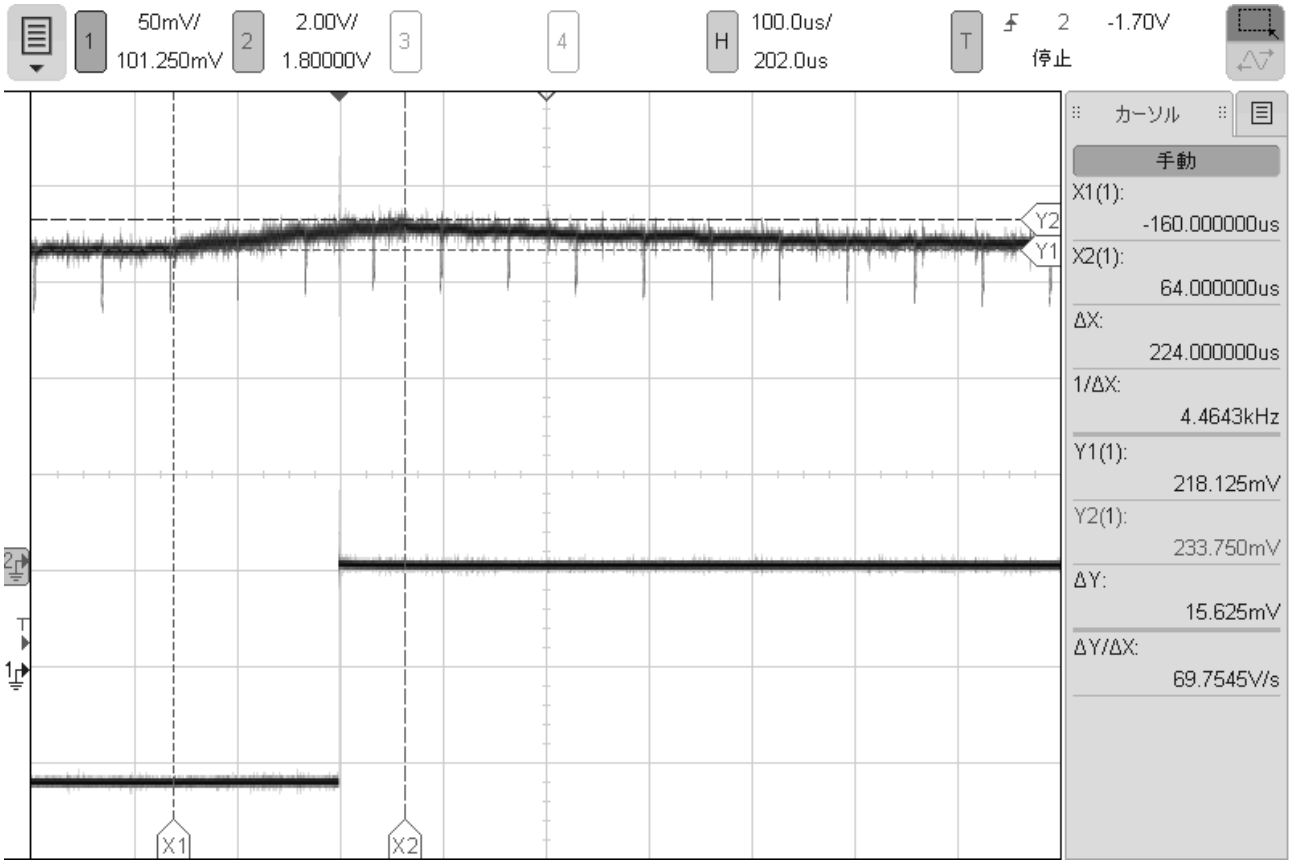


図 5.1-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

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。

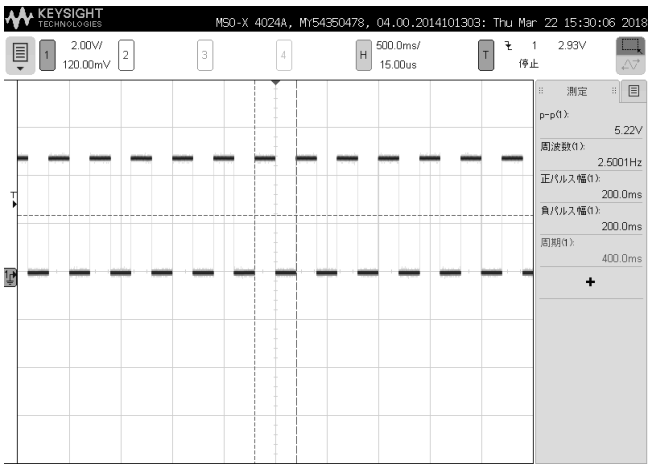


図 5.1-2.1 LED の点滅

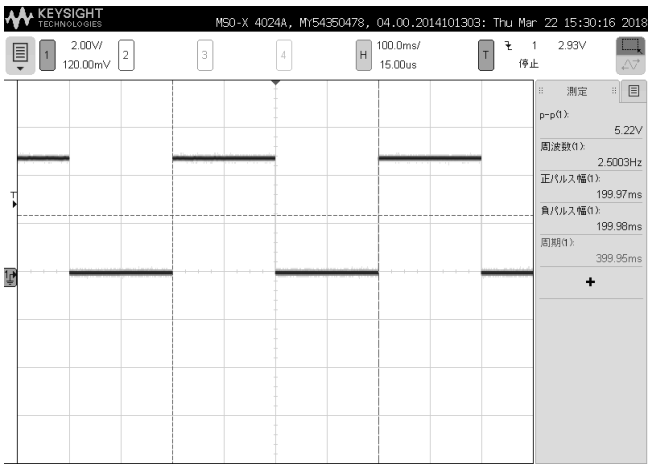


図 5.1-2.2 LED の点滅(拡大)

4.1-2 0.2 秒毎の点滅。周期は、0.39995 秒

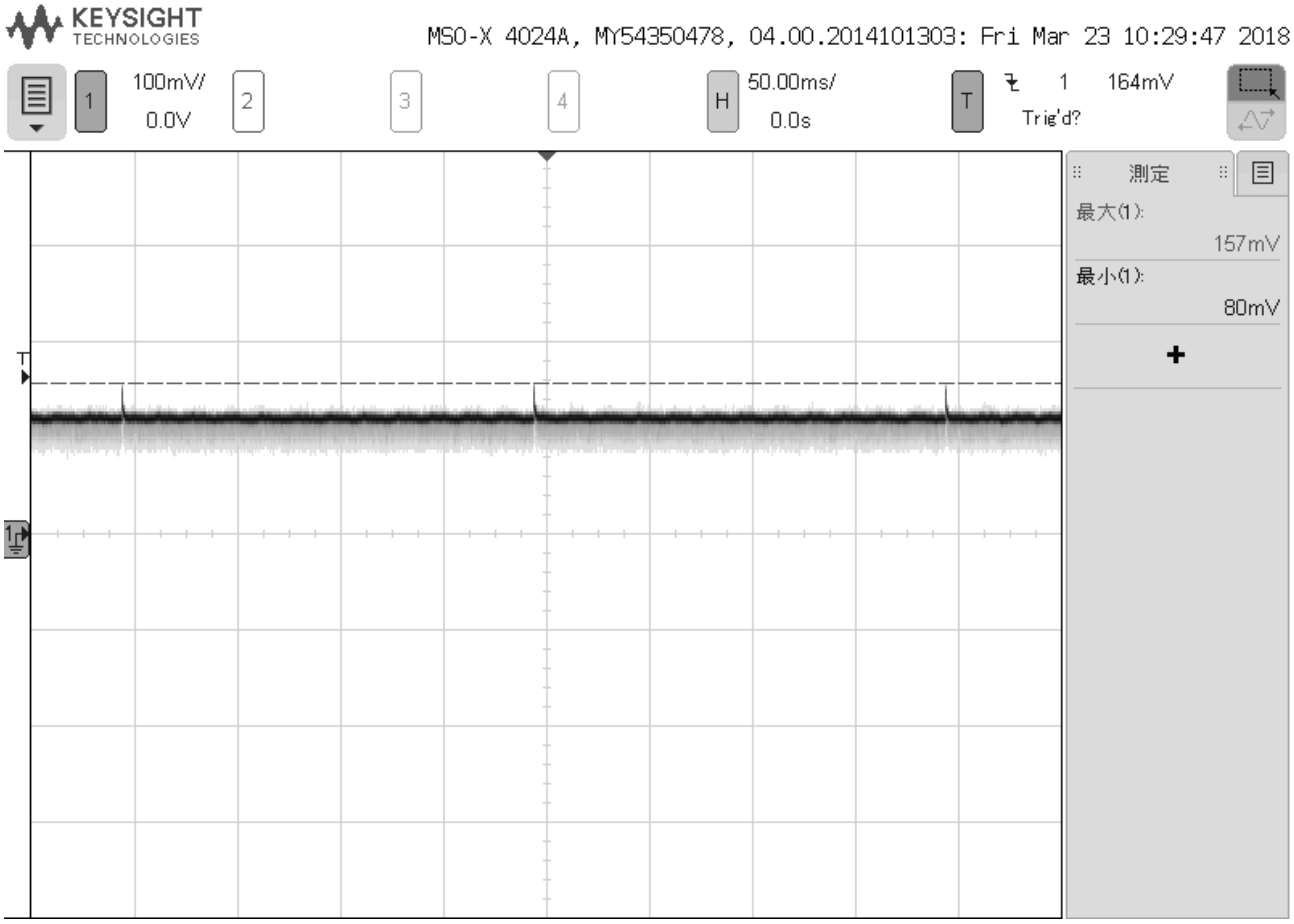


図 5.1-2.3 電流の時間変化

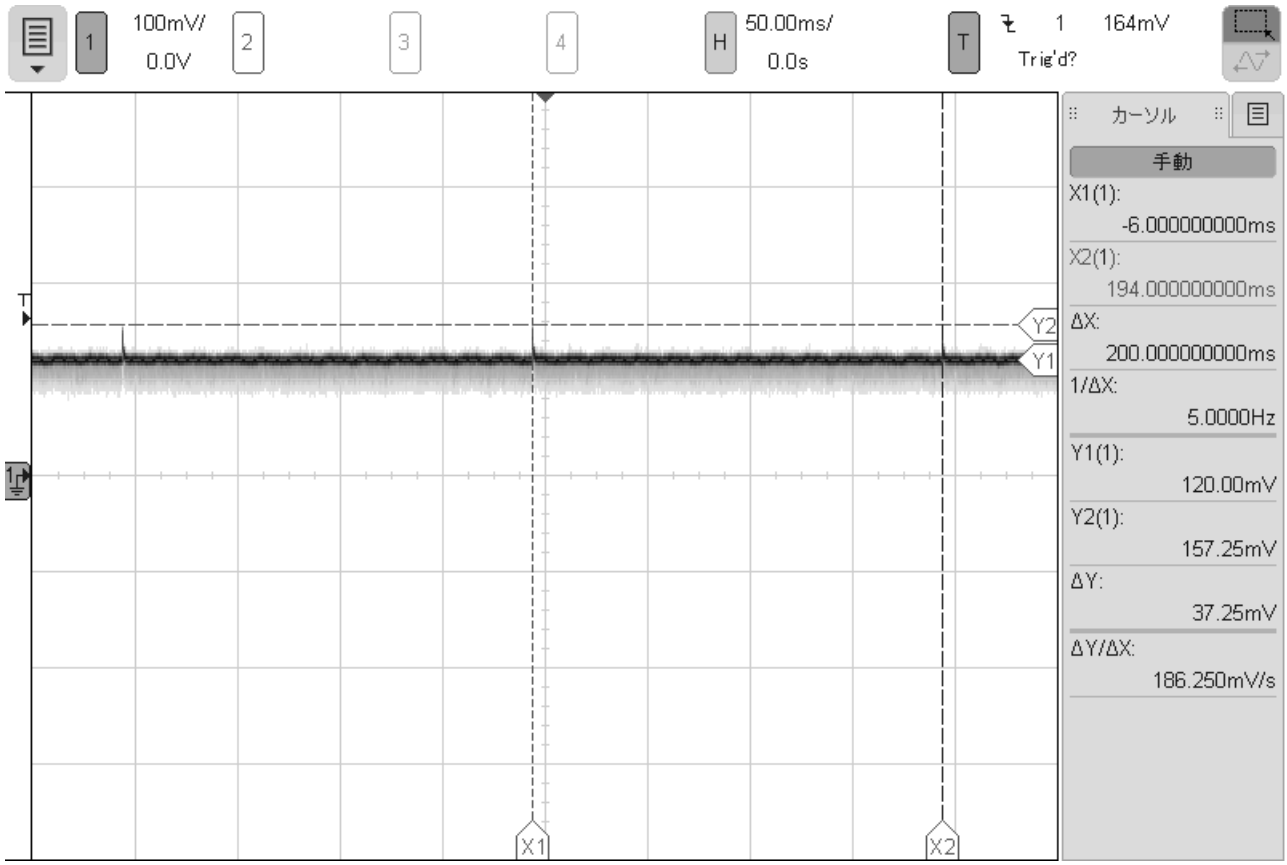


図 5.1-2.4 電流の時間変化及び LED の点滅

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。

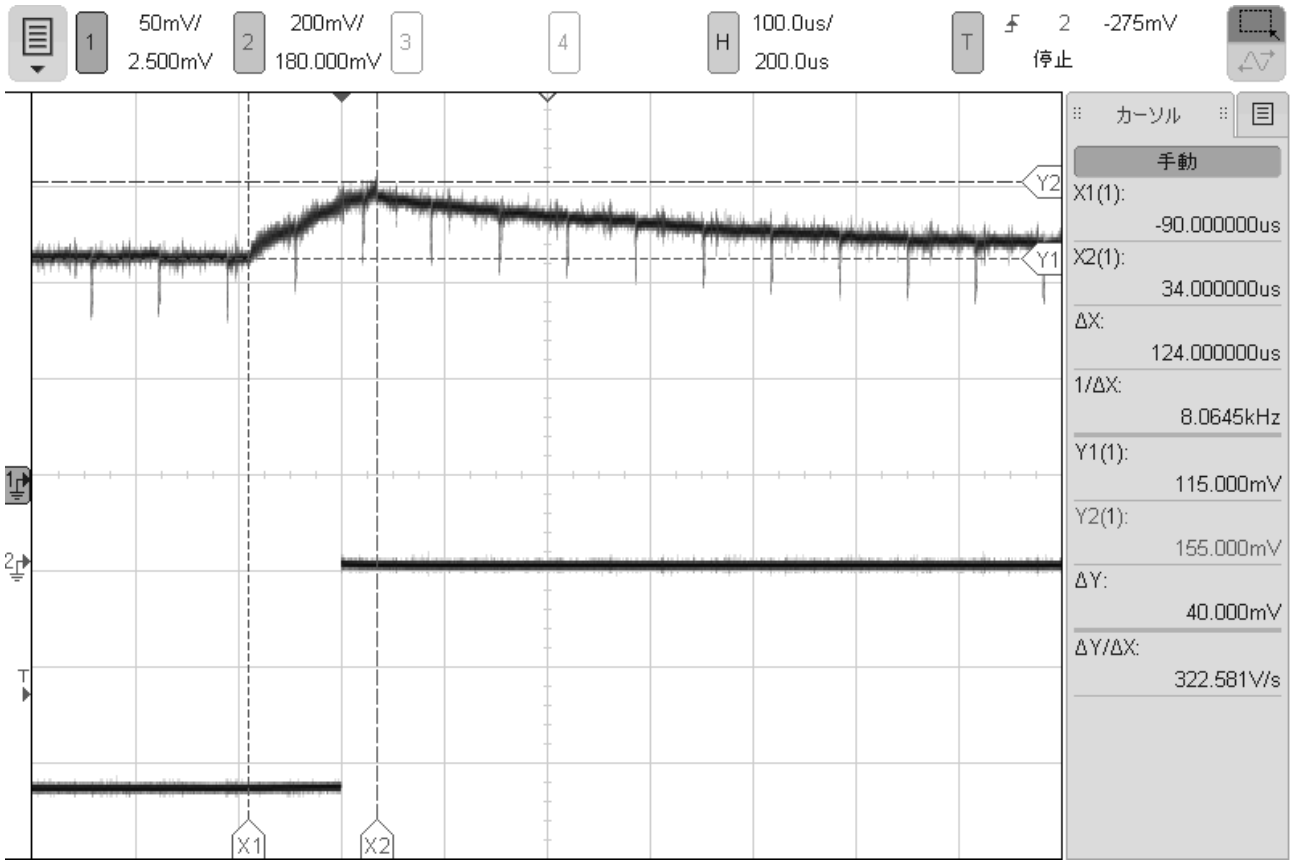


図 5.1-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大)

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。

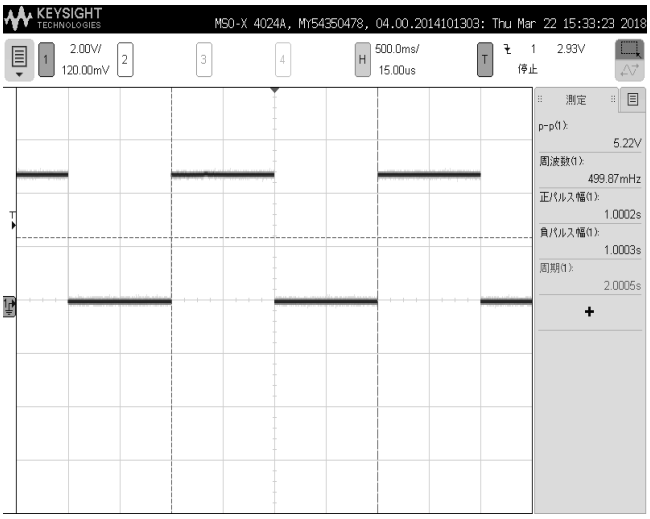


図 5.2-1.1 LED の点滅

4.2-1 1 秒毎の点滅。周期は、2.0005 秒

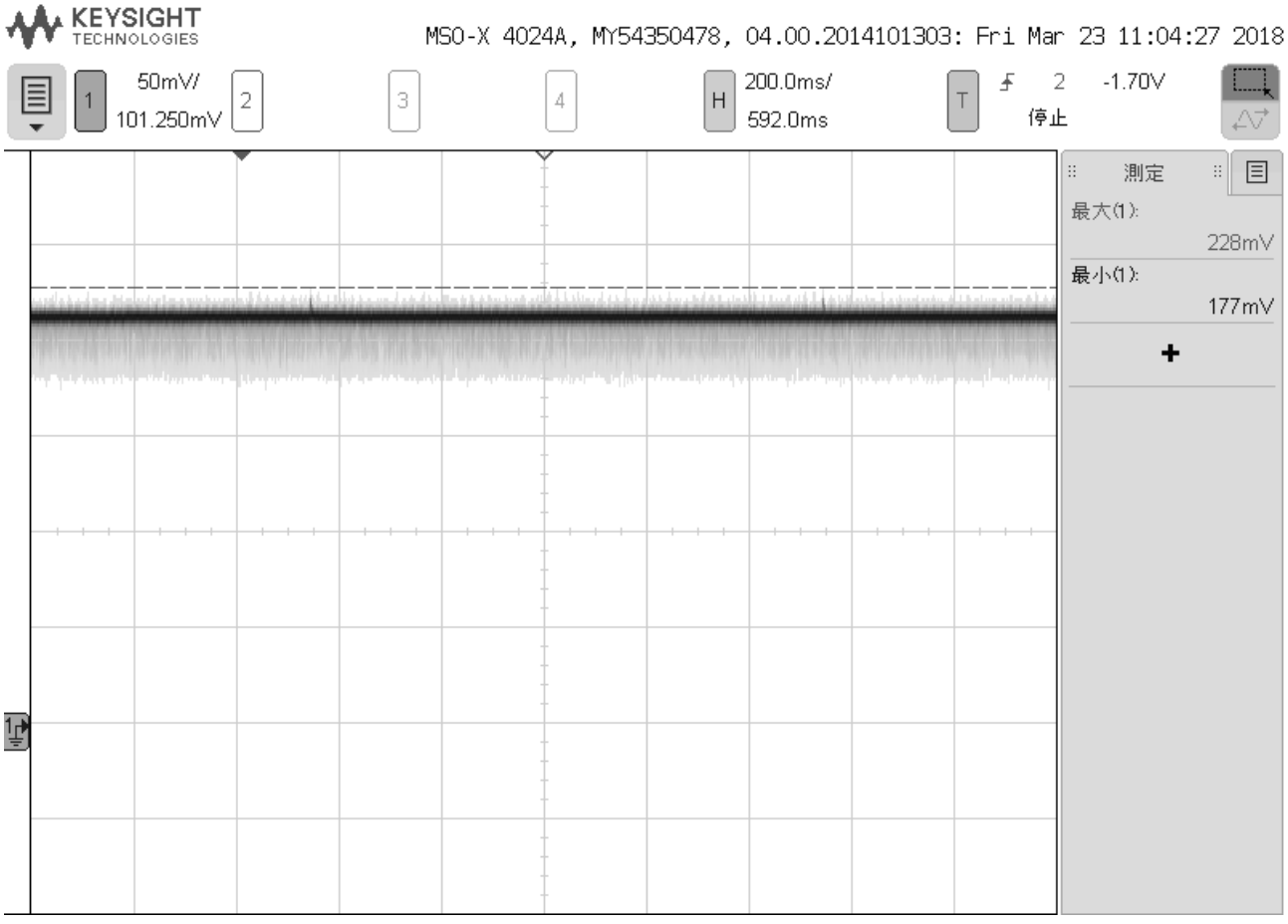


図 5.2-1.2 電流の時間変化

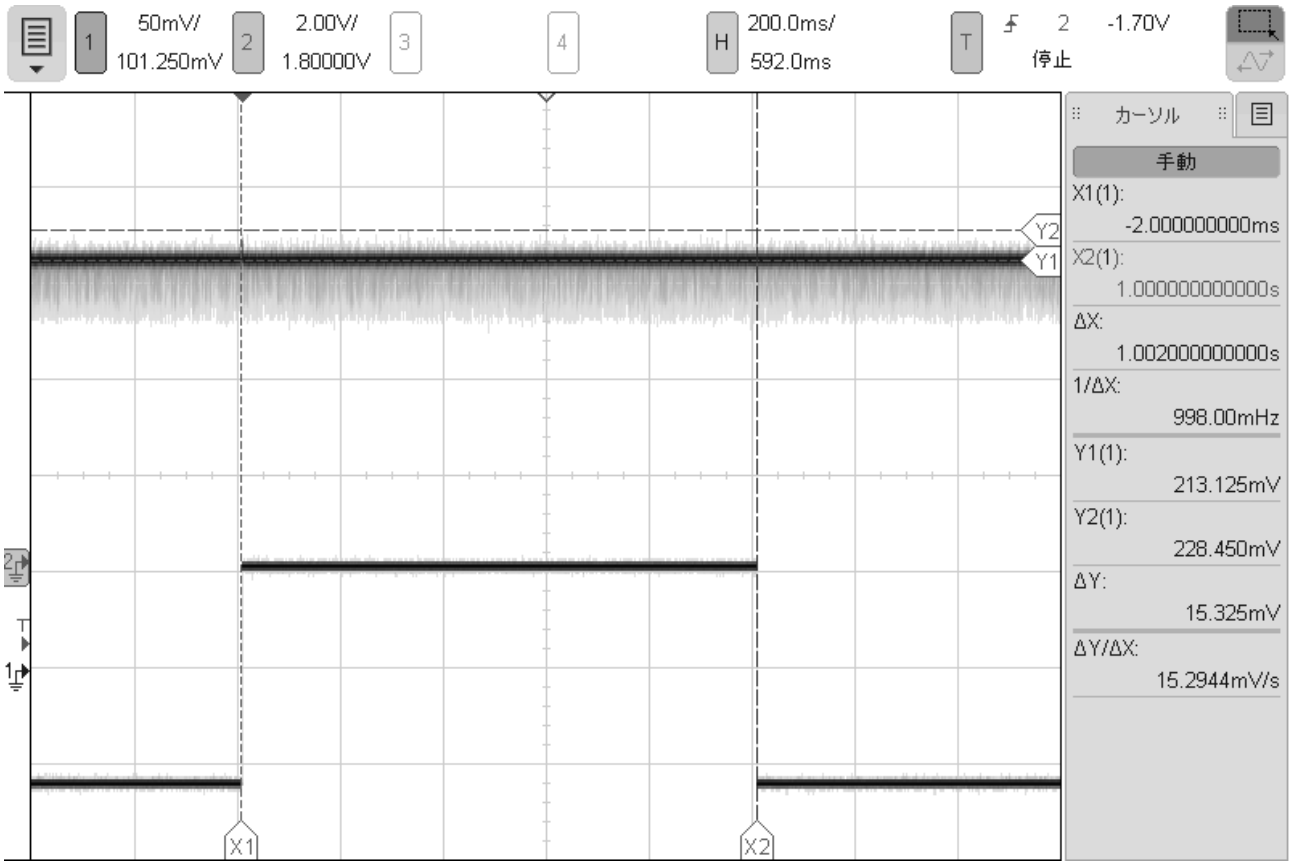


図 5.2-1.3 電流の時間変化及び LED の点滅

4.2-1 1 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。

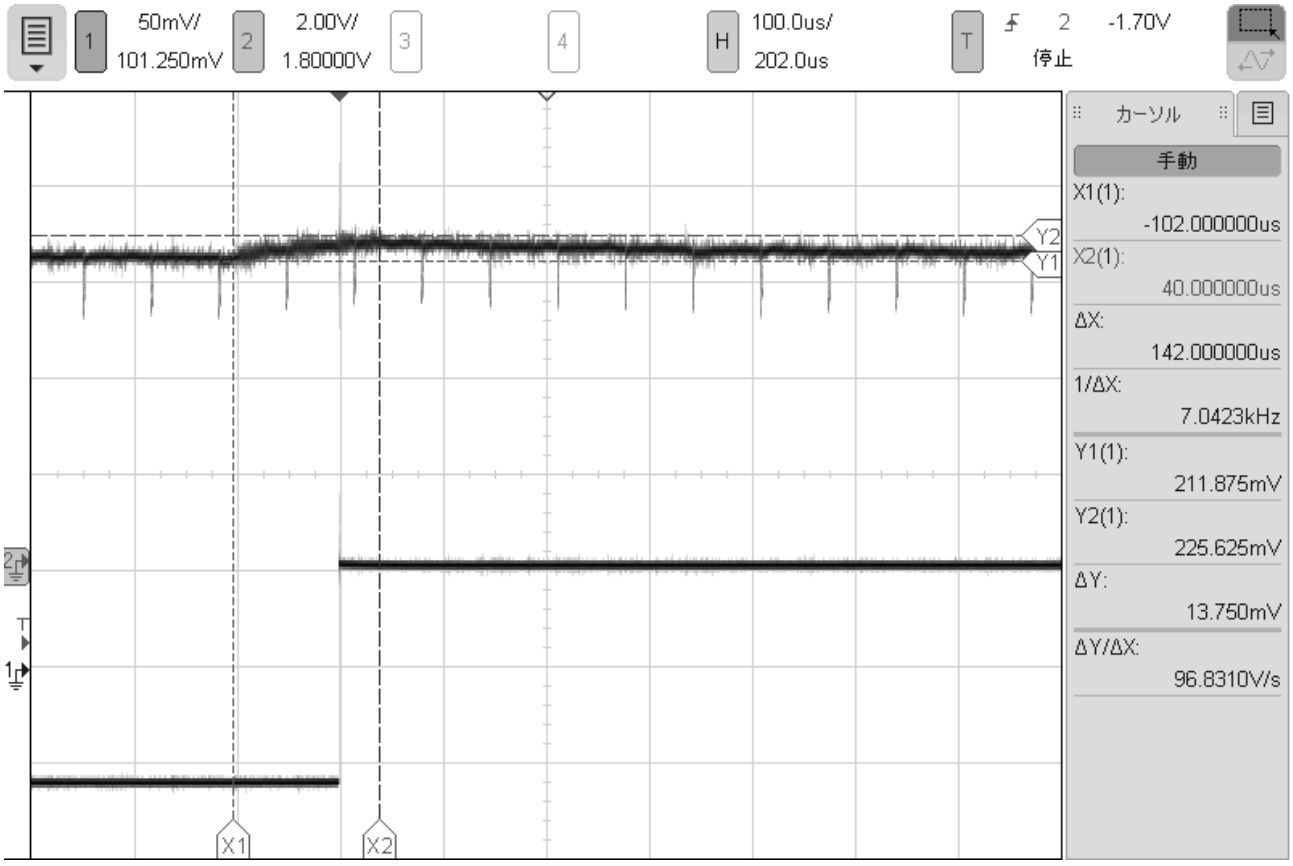


図 5.2-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

4.2-1 1 秒毎の点滅。ピークの形状。LED をトグルするロジックは、142 μsec 程度で動作している。

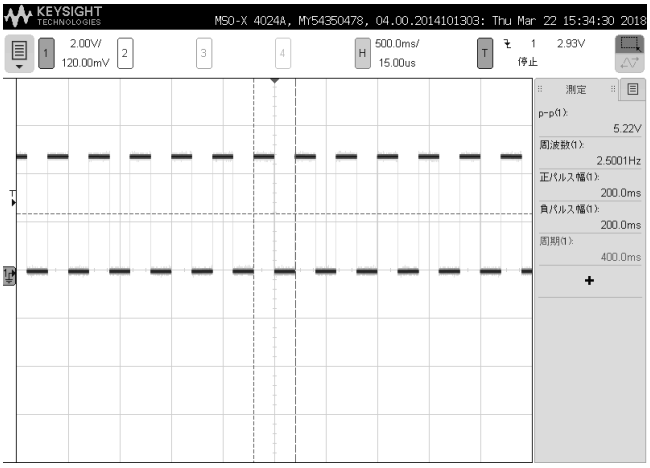


図 5.2-2.1 LED の点滅

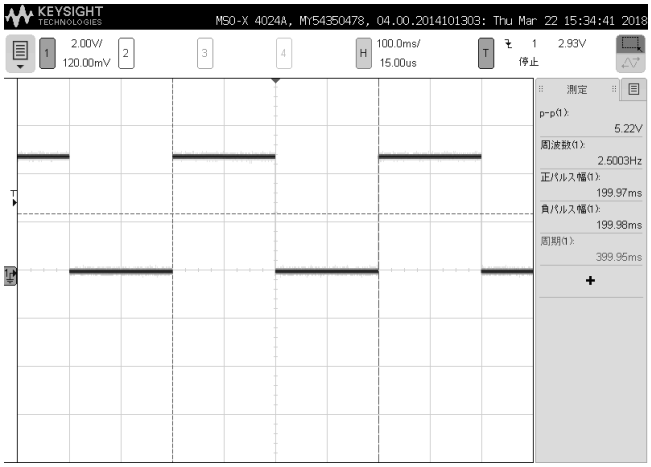


図 5.2-2.2 LED の点滅(拡大)

4.2-2 0.2 秒毎の点滅。周期は、0.39995 秒

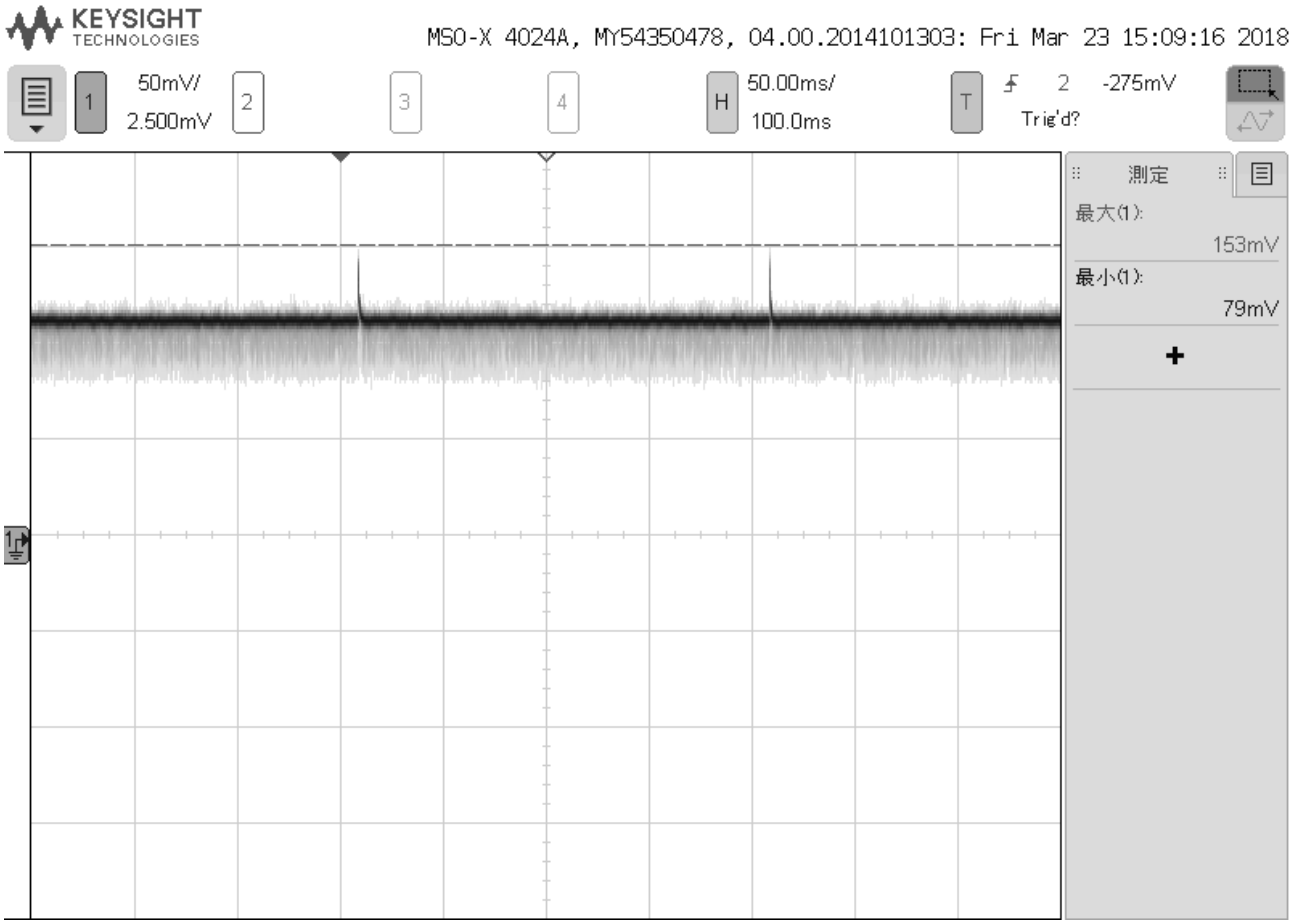


図 5.2-2.3 電流の時間変化

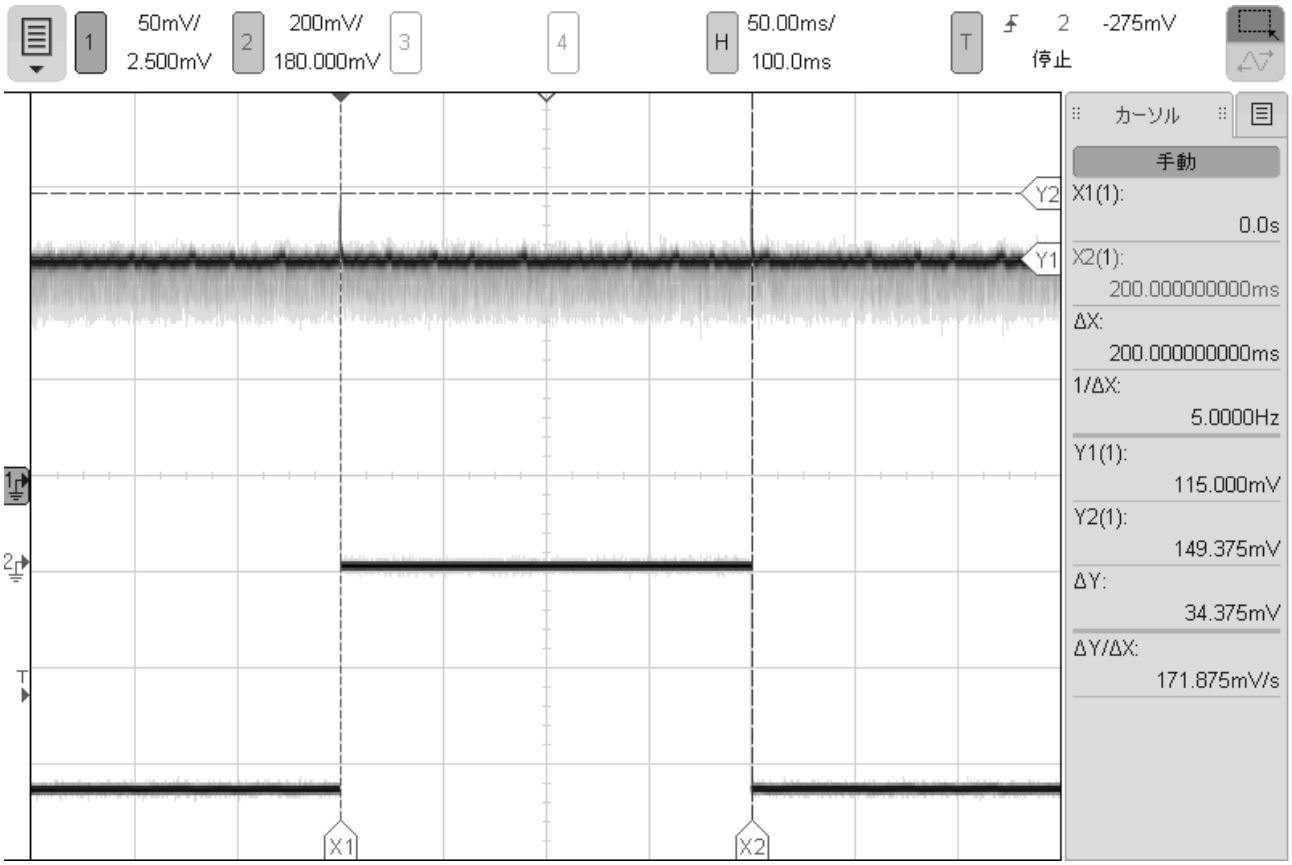


図 5.2-2.4 電流の時間変化及び LED の点滅

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。

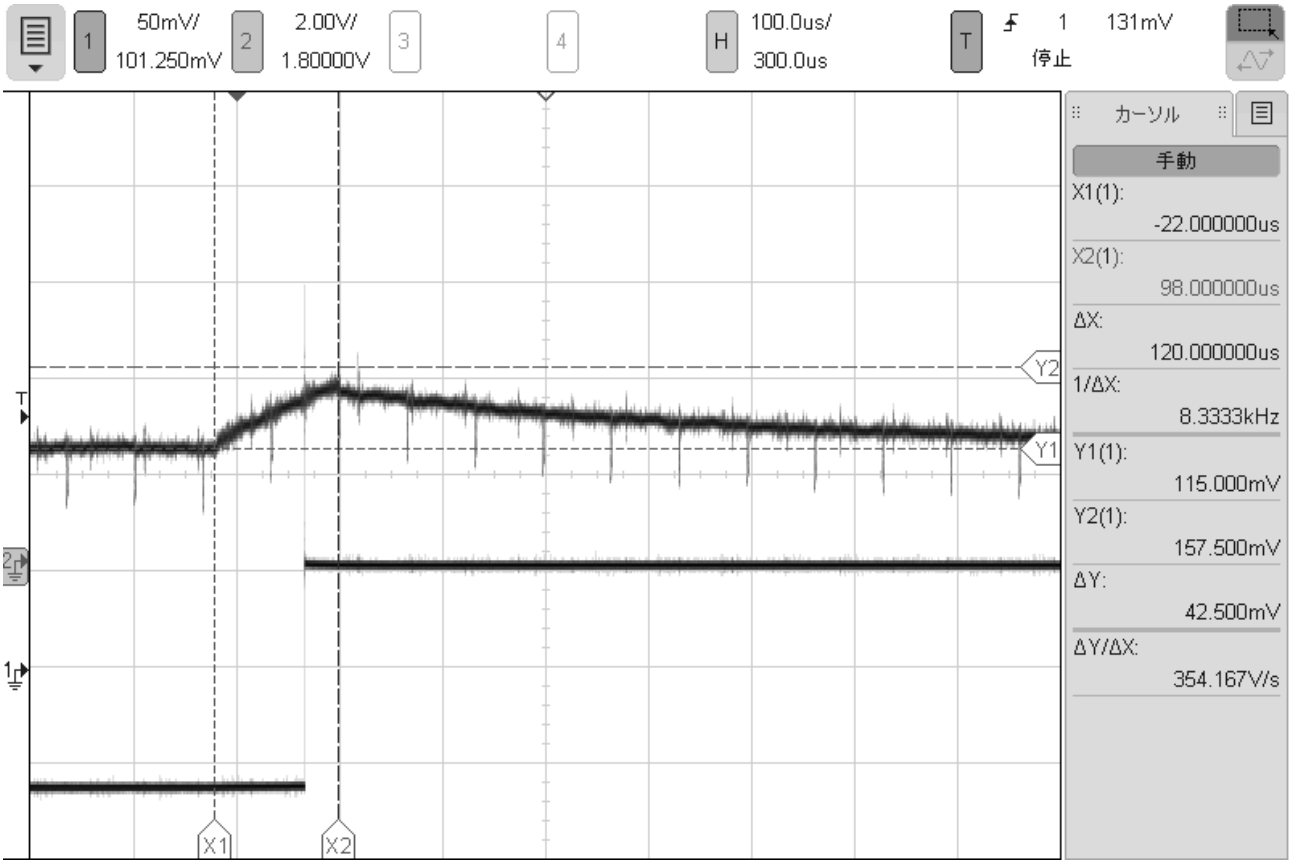


図 5.2-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大)

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。

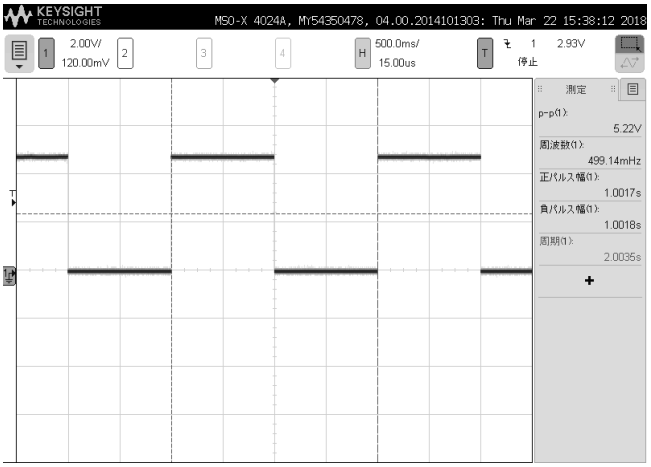


図 5.3-1.1 LED の点滅

4.3-1 1 秒毎の点滅。周期は、2.0035 秒

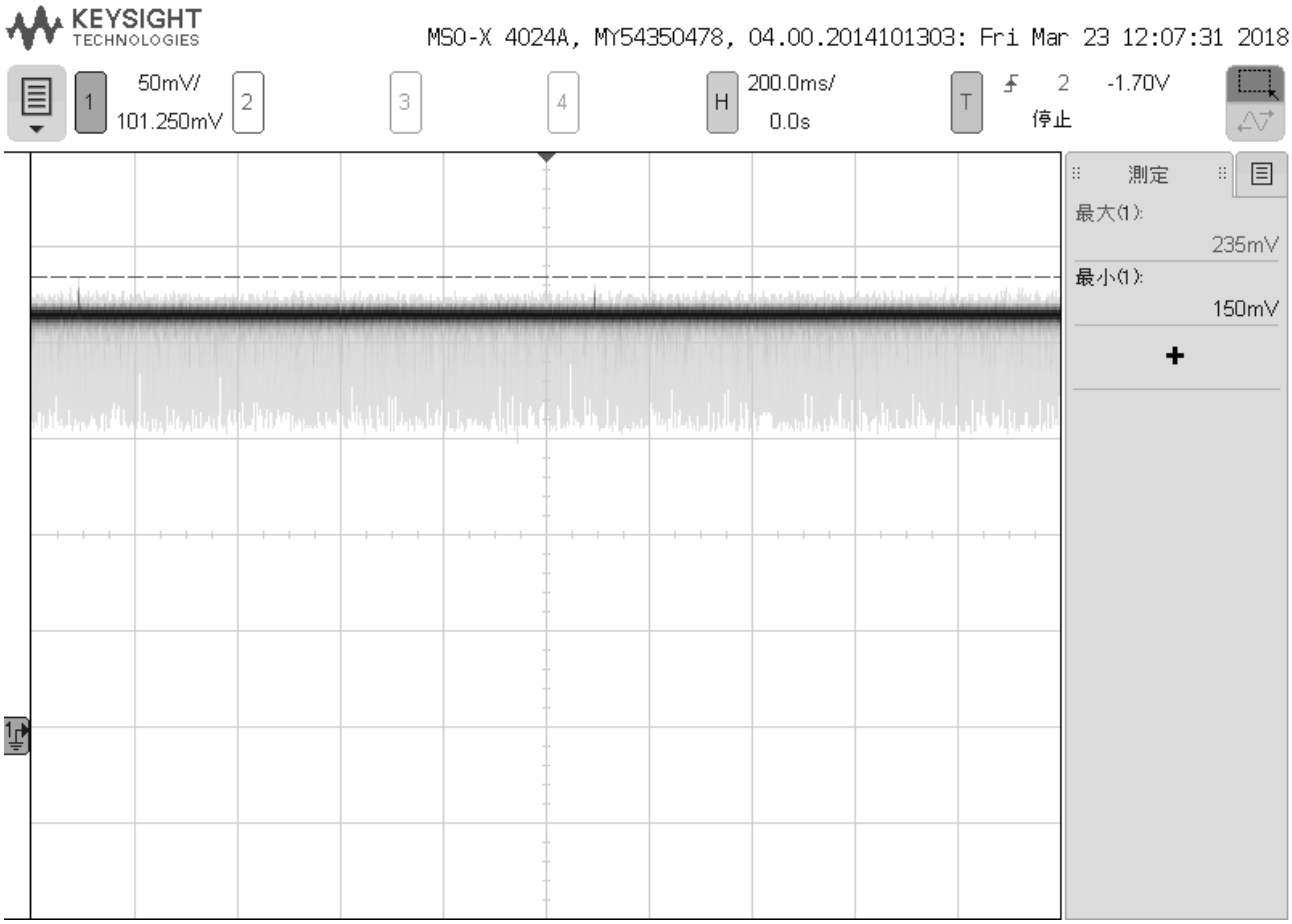


図 5.3-1.2 電流の時間変化

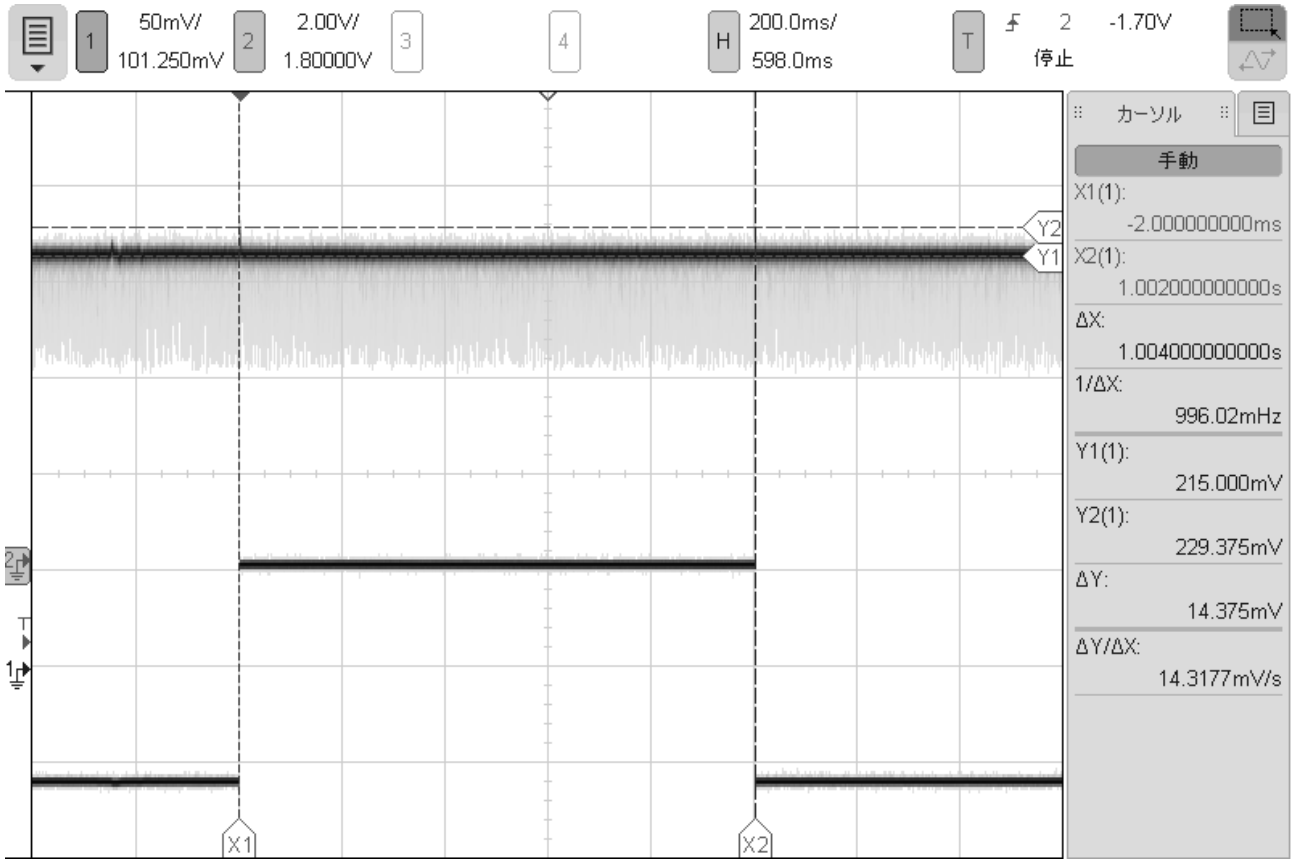


図 5.3-1.3 電流の時間変化及び LED の点滅

4.3-1 1 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。

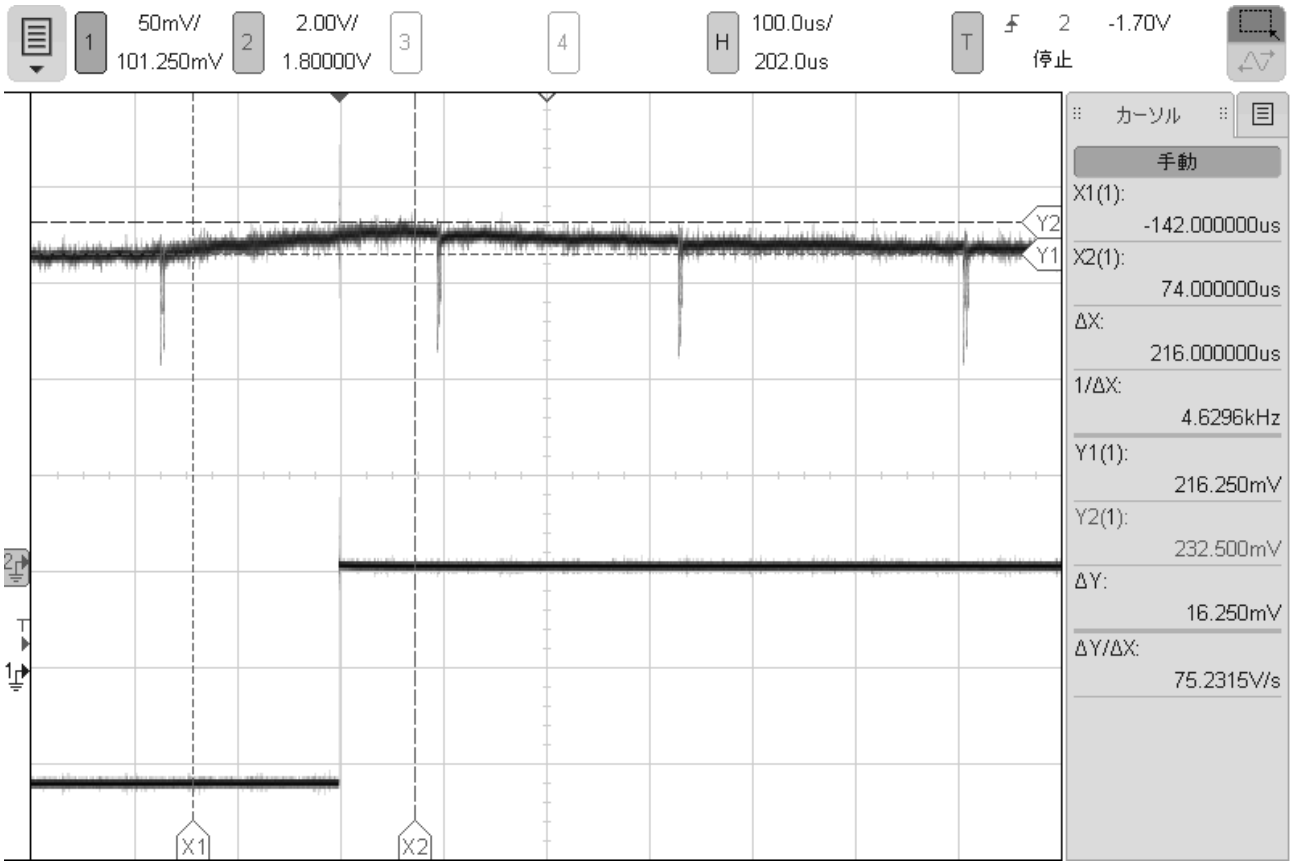


図 5.3-1.4 電流の時間変化及び LED の点滅(ピーク部分拡大)

4.3-1 1 秒毎の点滅。ピークの形状。LED をトグルするロジックは、216 μsec 程度で動作している。

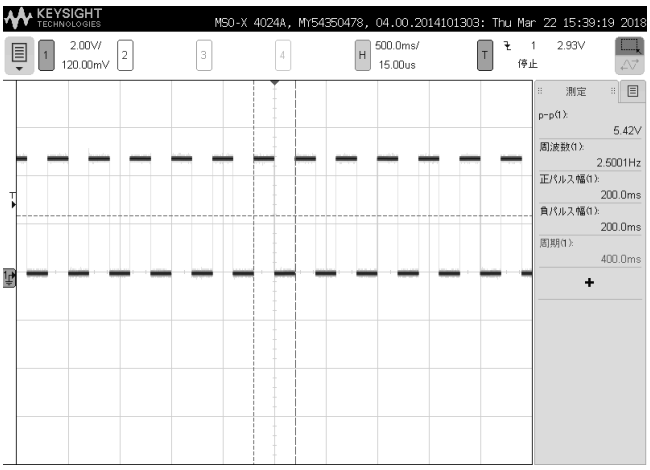


図 5.3-2.1 LED の点滅

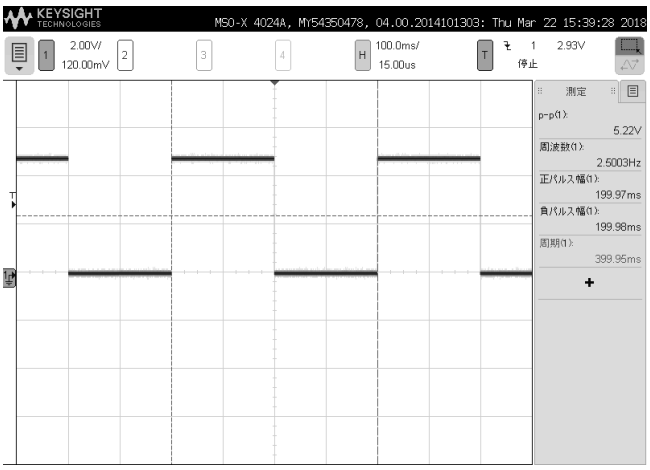


図 5.3-2.2 LED の点滅(拡大)

4.3-2 0.2 秒毎の点滅。周期は、0.39995 秒

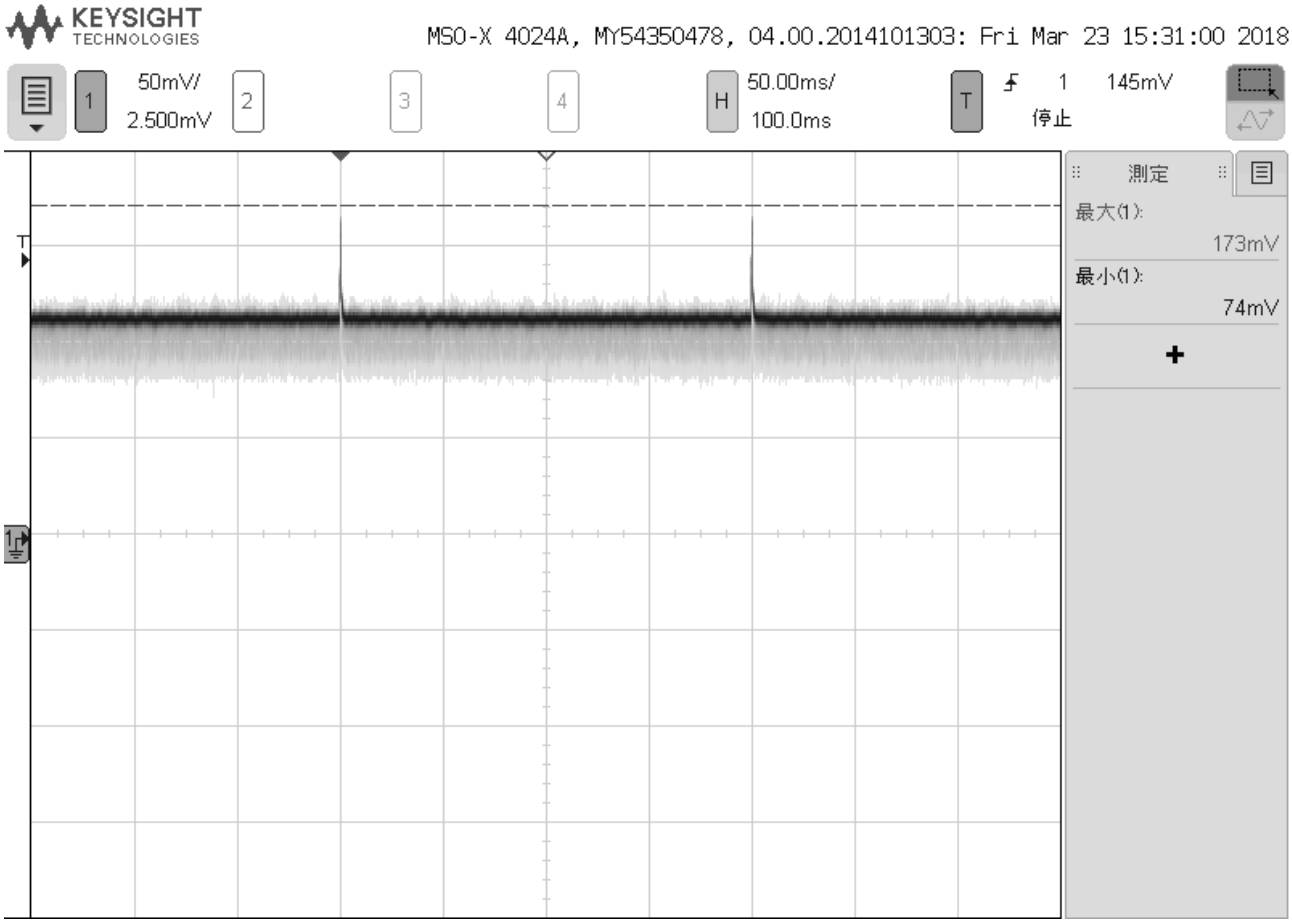


図 5.3-2.3 電流の時間変化

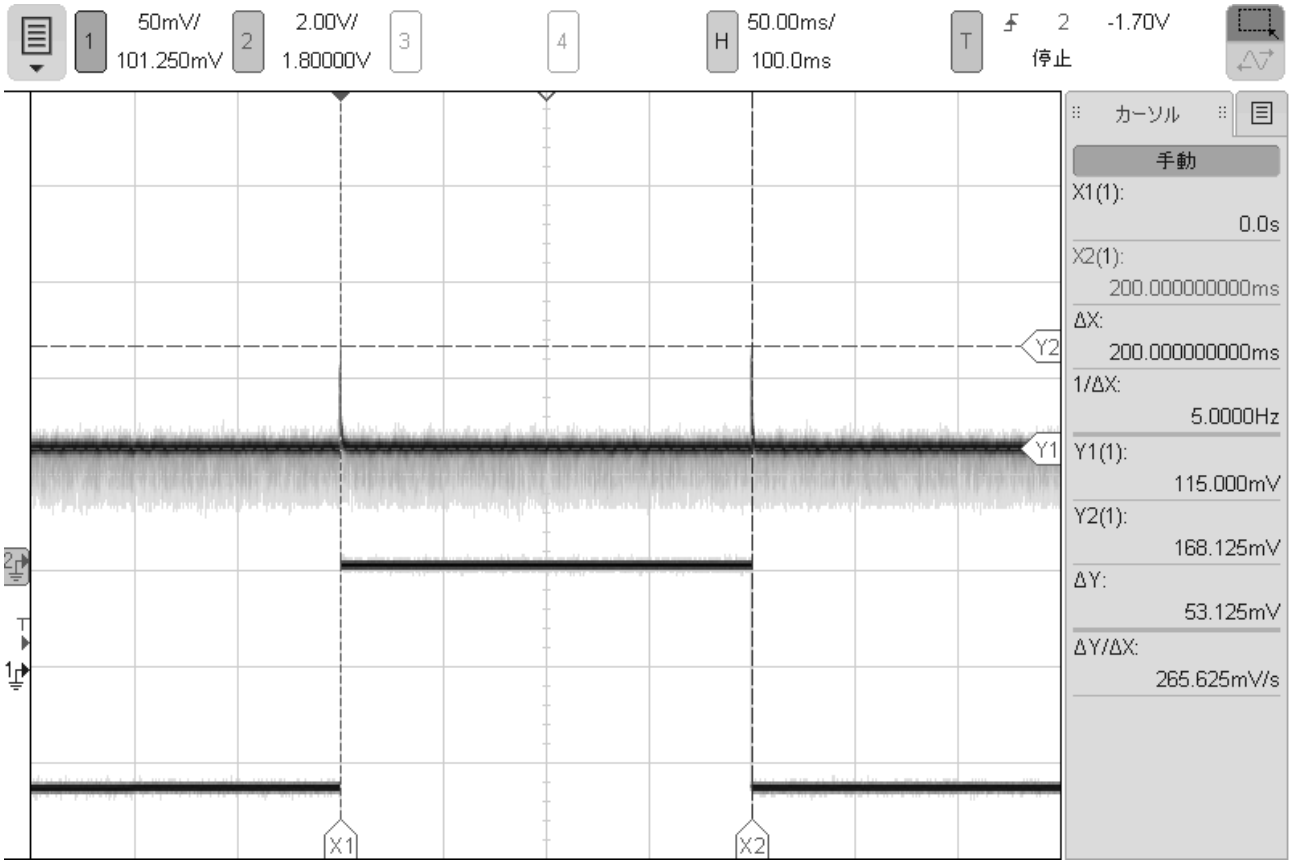


図 5.3-2.4 電流の時間変化及び LED の点滅

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。

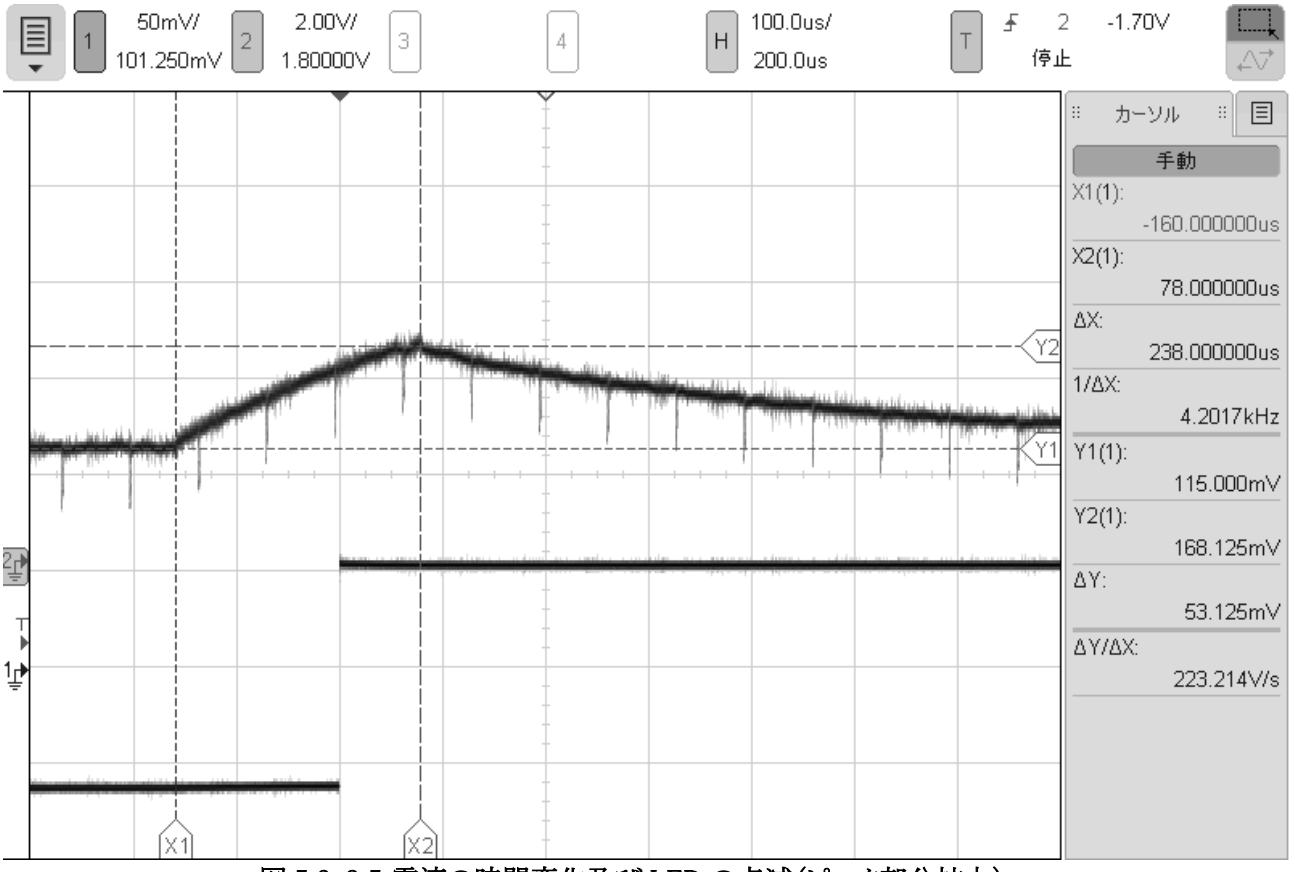


図 5.3-2.5 電流の時間変化及び LED の点滅(ピーク部分拡大)

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1chip 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。