Cortex-M0 開発キット STM32F051

マニュアル

株式会社日昇テクノロジー

http://www.csun.co.jp

info@csun.co.jp

作成・更新日 2013/08/06

• 修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2013/08/06

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。「http://www.csun.co.jp」

目 录

1. 概要

1.1. マイコンの主な仕様

- 2.0 V -3.6 V 電圧動作範囲
- Cortex-M0 CPU (48 MHz max)
- メモリ
  - 64 K bytes Flash
  - 8 Kbytes SRAM
- CRC チェックユニット
- クロック管理
  - 4-32 MHz の外部水晶発振器をサポート
  - 内部 40 kHz と 8 MHz の RC オシレータをサポート
  - 修正機能/カレンダー機能付きの 32K RTC、ストップモードまたはスタンバイモードからプロセッサをウェイクアップ

- リセットと電源管理
  - オン/オフ□電源
  - プログラマブル電圧検出器
- 低消費電力のスリープ、ストップとスタンバイモード
- バッテリ駆動の RTC バックアップレジスタ
- 5 チャネル DMA コントローラ
  - 1 x 12-bit、 1.0 us ADC、0～3.6V の電圧検出範囲
  - 2.4V～3.6V の独立アナログ電源
- 2 チャンネル低消費電力プログラマブル入出力コンパレータ
- 1 チャンネル 12-bit D/A
- 55 チャンネル I/O (最大)
  - 全ては外部割り込みにマッピング可能
  - 36 チャンネル I/O (最大) 5V のフォールトトレランス
- 18 チャンネル (最大)静電容量センシングチャネル、タッチボタンのためのサポート、リニアおよび回転タッチセンサ
- 96-bit 唯一 ID
- SWD シリアルデバッグ
- 11 タイマー (最大)
  - 7 チャンネルの 16 ビットの高度な制御タイマ、6 チャンネルの PWM 出力、デッドタイム生成と緊急ブレーキ機能付
  - 32 ビットと 16 ビットタイマ、4 IC / OC、赤外線リモコンデコード可能
  - 1 つ 16 ビットタイマ、4 つ IC / OC、1 つ OCN、デッドタイム生成と緊急ブレーキ機能付
  - 2 つ 16 ビットタイマ (IC / OC、OCN 機能付き)、デッドタイム生成と緊急ブレーキ機能付、赤外線コントロール用の復調ドア

- １つ 16 ビットタイマ（IC / OC 機能付き）
- 独立したシステム･ウォッチドッグ･タイマ
- １つ 24 ビットダウンカウンタ
- １つ 16 ビット･タイマは、DAC 駆動に使用する

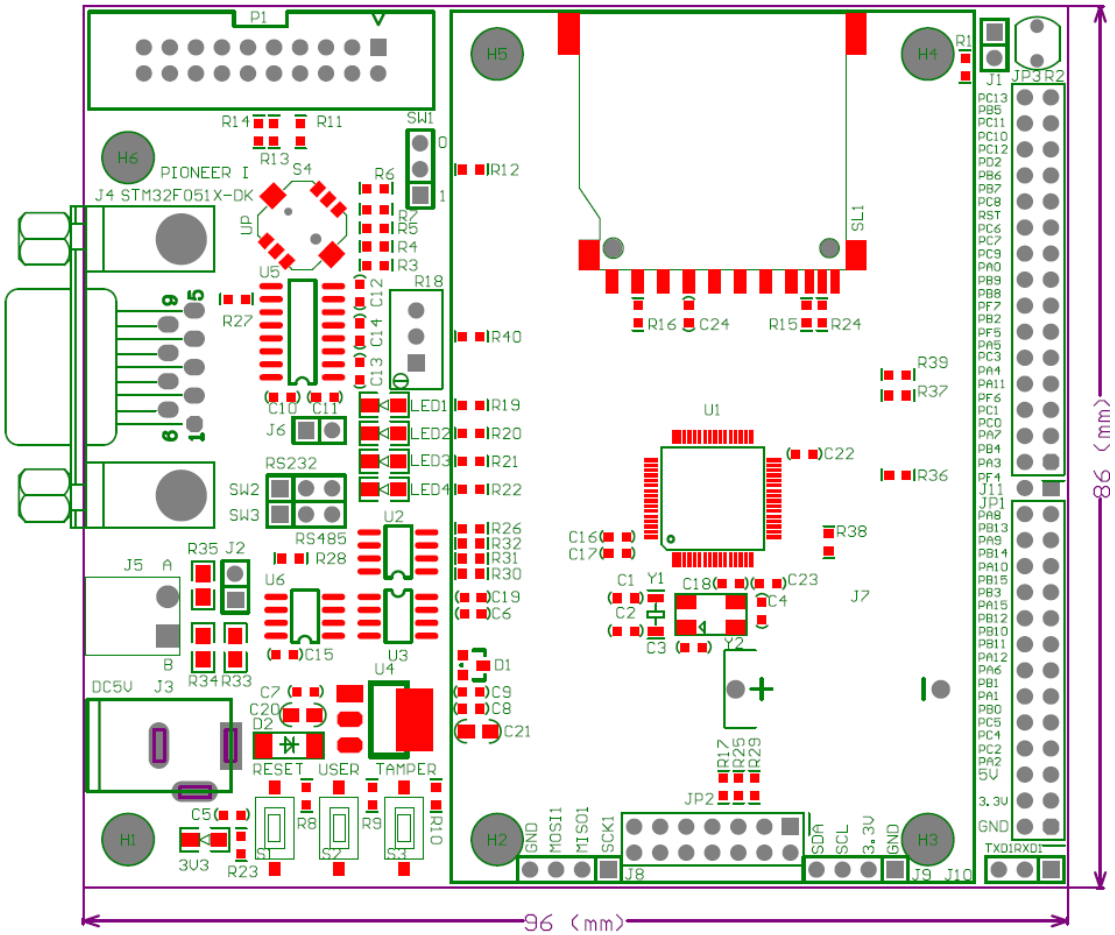
◆ 通信インタフェース

- 2 つ I2C インタフェース、高速モード Plus（1 Mbit/s)、20mA シンク電流をサポート；SMBus/PMBus、ストップモードからウェイクアップをサポート
- 2 つ UARTs、メイン同期 SPI およびモデム制御をサポート、その 1 チャンネルに ISO7816 プロトコル付き、LIN と IrDA サポート、自動ボーレート検出とウェイクアップ機能付
- 2 つ(18 Mbit/s)、4～16 ビットのプログラマブルフレーム、その 1 は I2S とリユース

1.2. ボードの主な仕様

- 20 ピン 2.54 ピッチ JTAG インタフェース
- RS232 インターフェース
- RS485 インターフェース
- SD メモリーカードインターフェース（SPI）
- 4 制御可能な LED
- ２つのユーザーボタン
- １つの 5 方向のナビゲーションボタン
- IIC インタフェースの EEPROM
- IIC インタフェースの温度センサ
- １つの光センサ
- １つのボタン電池ホルダー（ボタン電池なし）
- １つの SPI インタフェース
- １つの IIC インターフェース。
- １つの UART インターフェース
- １つの液晶画面のインターフェース
- すべての I / O は 2.54mm ピッチの 2 配列ピンインタフェースで引出す

2. Hardware layout and configuration



2.1. Power supply

DC5V 電源で開発ボードに給電。

2.2. Boot option

開発ボードの起動モードは下記の 3 つの方法がある：

- ユーザプログラムスペースから起動
- ISP ブートプログラム（Boot loader）スペースから起動
- 内部 SRAM スペースから起動

上記の 3 つの起動モードはコアボードの SW1 とユーザーレジスタのビット 20 で設定できる：

表 1. Boot 関連のジャンパー：

BOOT 0(SW1)	Boot1(Bit20)	起動モード

0	無視	ユーザプログラムスペースから起動
1	1	内部 SRAM スペースから起動
1	0	ISP ブートローダ（Boot loader）スペースから起動

2.3. Clock source

開発ボードに 2 つのクロックソースがある：

- Y1、32.768K　STM32F051R8 チップの RTC 水晶発振器
- Y2、8MHz　　STM32F051R8 チップのメイン頻度水晶発振器

2.4. Reset source

開発ボードのリセット信号は、ローレベルリセットで、下記の方法で実現する：

- RST キー
- JTAG インタフェースの P1-15Pin
- JP2-10Pin
- JP3-21Pin

2.5. SD card

開発ボードは SPI インタフェースで SD カードと接続；通常 I/O PB15 で SD カードスロットの状態を検出する（SD カード）

2.6. Debug Port

開発ボードに 20 ピン 2.54mm ピッチのシミュレーションコンセントを持ち、J-link 或いは U-link2 インタフェースをサポート、通信モードは SW モード。

2.7. Button

開発ボードに 3 つの独立したキーと 1 つの 5 方向のナビゲーションボタンがある、全てプルアップモード。

表 2. 各キーの機能

キー番号	機能
RESET	リセット
USER	押す：ローレベル、通常：ハイレベル、外部割り込みに設定可能
TAMPER	押す：ローレベル、通常：ハイレベル、外部割り込みに設定可能

2.8. LED

開発ボードの 5 つの LED がある。黒色の DC2.1 コンセントの隣には電源 LED、緑色の可変抵抗の隣には 4 つの通常 I/O 制御 LED。

2.9. IIC EEPROM

開発ボードに 1 つの I2C バス EEPROM を持ち、I2C1 を介し接続する、アドレスは 0xA0。

2.10. IIC Temperature Sensor

開発ボードに 1 つの I2C バス温度センサを持ち、I2C1 を介し接続する、アドレスは 0x90。

2.11. IIC Port

J9 は IIC 拡張ポート、外部 IIC モジュールと接続或いは 2 つの開発ボード間の IIC 通信実験に使用する。
表 3 J9 ピン定義

pin 番号	ファンクション
Pin1	GND
Pin2	+3.3V
Pin3	I2C1_SCL
Pin4	I2C1_SDA

2.12. SPI LCD

JP2 は SPI 通信モードの LCD インタフェース、2.8 インチの LCD 拡張ボードと接続に利用。
表 4 JP2 ピン定義

pin 番号	ファンクション
Pin1	PB6、I2C1_SCL、LCD 拡張ボードは使用しない、タッチスクリーン･コントローラ･チップの通信に使用する
Pin2	PB7、I2C1_SDA、LCD 拡張ボードは使用しない、タッチスクリーン･コントローラ･チップの通信に使用する
Pin3	PB9、TP_INT、タッチスクリーン･コントローラ･チップの通信に使用する
Pin4	PF6、TP_CS、タッチスクリーン･コントローラ･チップのチップセレクトに使用する（SPI 通信モード）

Pin5	PB4、SPI1_MISO
Pin6	LCD バックライト制御、プルアップ抵抗
Pin7、8	+3.3V
Pin9	PF4、LCD_CS
Pin10	LCD リセットピン
Pin12	PA7、SPI1_MOSI
Pin13、14	GND

2.13. SPI Port

J8 は SPI 拡張ポット、2 つの開発ボード SPI インタフェースの通信実験に使用する。

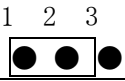
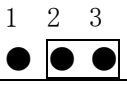
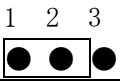
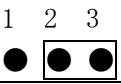
表 5 J8 ピン定義

pin 番号	ファンクション
Pin1	PA5、SPI1_SCK
Pin2	PB4、SPI1_MISO
Pin3	PA7、SPI1_MOSI
Pin4	GND

2.14. RS-232

開発ボードの RS-232 と RS-485 チップは USART1 を共用する。SW2、SW3 上のジャンパーでファンクション設定可能：

表 6、UART 関連ジャンパー

ジャンパー	接続図	説明
SW2		PA9 、TXD と RS-232 チップ接続
		PA9 、TXD と RS-485 チップ接続
SW3		PA10、RXD と RS-232 チップ接続
		PA10、RXD と RS-485 チップ接続

2.15. RS-485

開発ボードの RS-232 と RS-485 チップは USART1 を共用する。設定方法は表 6 を参照。PA12 は RS-485 チップのデータ受送信の制御ピン。PA12＝0、RS-485 受信モード；PA12＝1、RS-485 送信モード。

2.16. Potentiometer

開発ボードに１つの 10K の高精度な可変抵抗があり、ADC11 と接続、対応するコンポーネントは R18。

2.17. LDR

開発ボードに１つの感光抵抗があり、CPU はコンパレータを介して外部の光強度を分析する。J1 ショート時光センサーを使用する。

2.18. Expansion Port

開発ボードにデバッグ用の PA13、PA14 を除き、STM32F051R8 のその他の I/O は全部 JP1、JP3 に引き出し。

3. 開発ツール KEIL の応用

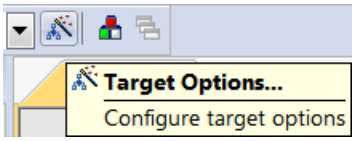
3.1. Keil コンパイル環境

3.1.1. コンパイラ環境構築

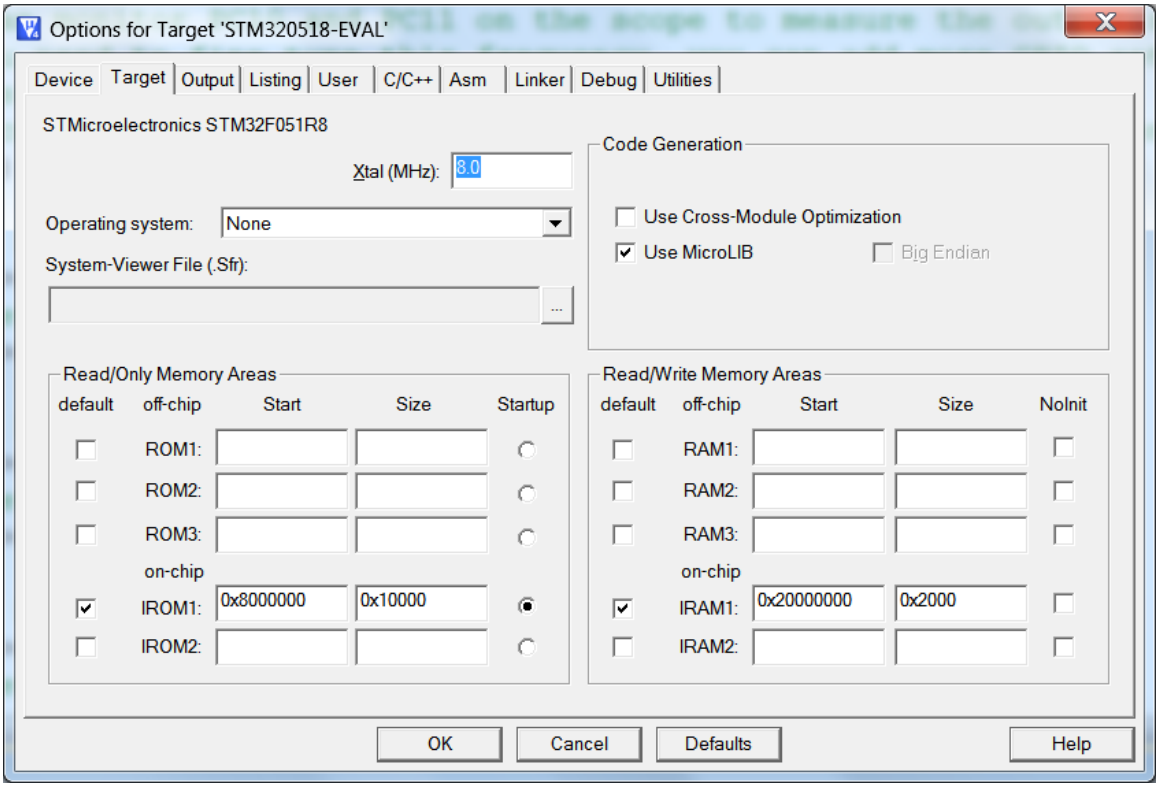
- libを使用する場合バージョン関連エラーがある可能性がありますので、MDK453.exeバージョンをお勧め。
- コンパイラは C ドライブのルートディレクトリ（C:¥Keil¥ARM）下にインストールする事をお勧め、でなければコンパイルエラーの可能性があります。

3.1.2. コンパイル環境設定

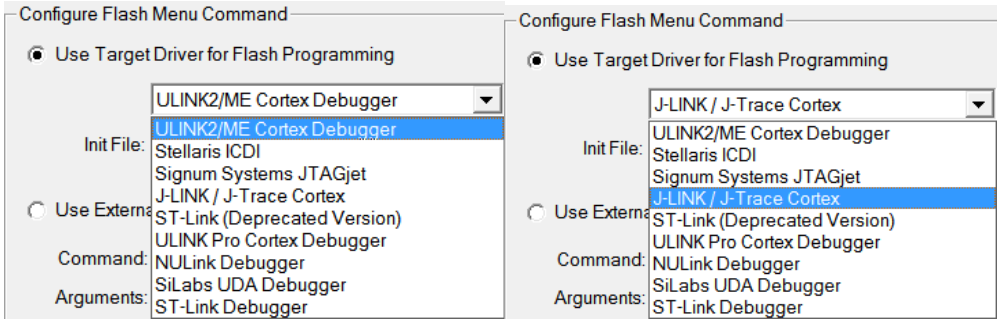
- 一つのサンプルソースをオープンし、｀Options for Target｀をクリック



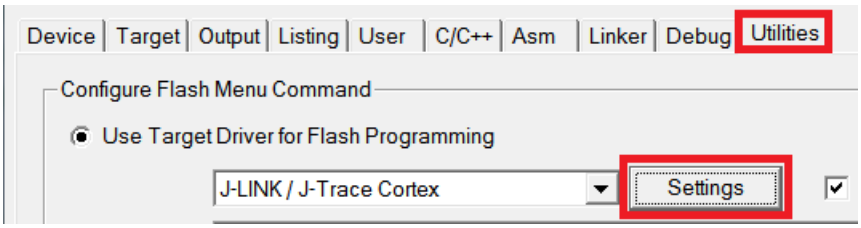
- 下記のウィンドウ通り：

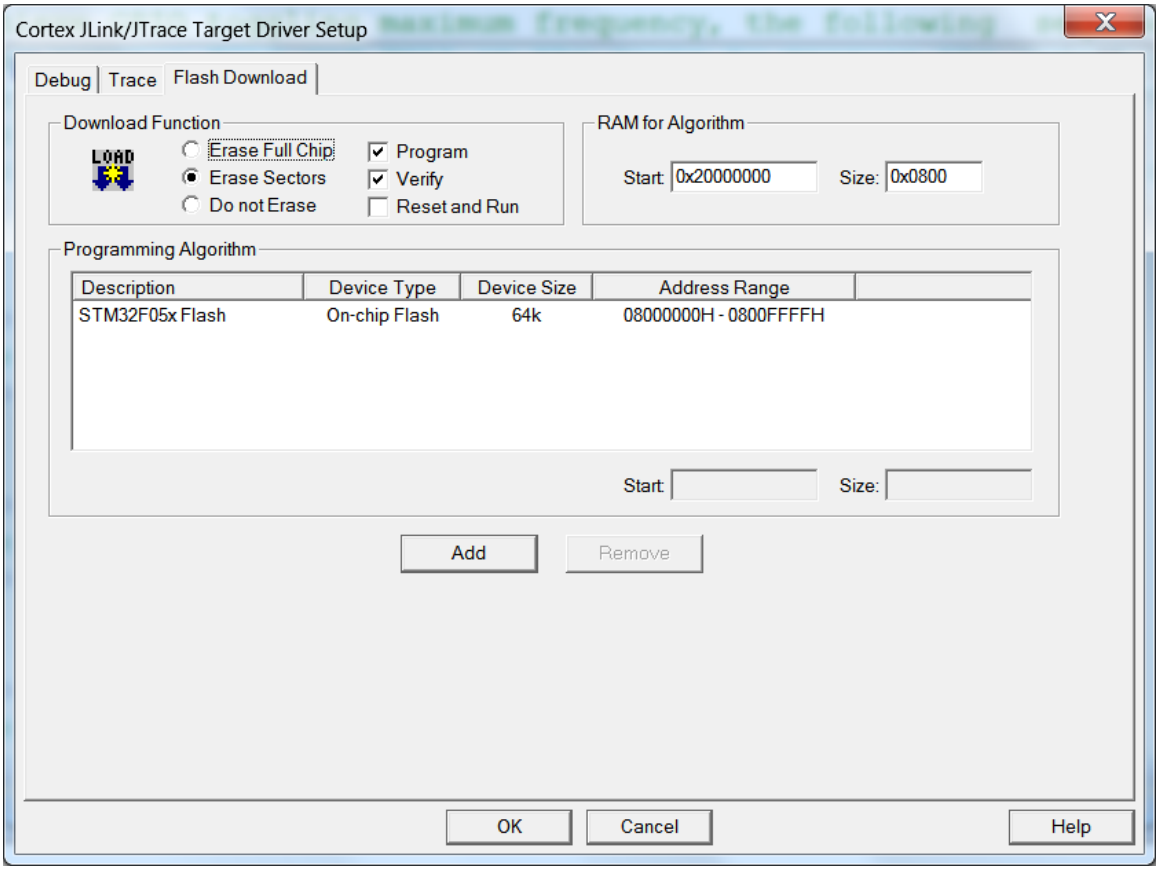


➢ ｀Utilities｀を選択、下記図のように｀Use Target Driver for Flash Programming｀でエミュレータの種類を選択、ULINK2エミュレータを使用する場合、｀ULINK2/ME Cortex Debugger｀を選択する；JLINK V8エミュレータを使用する場合、｀J-LINK/J-Trace Cortex｀を選択する。

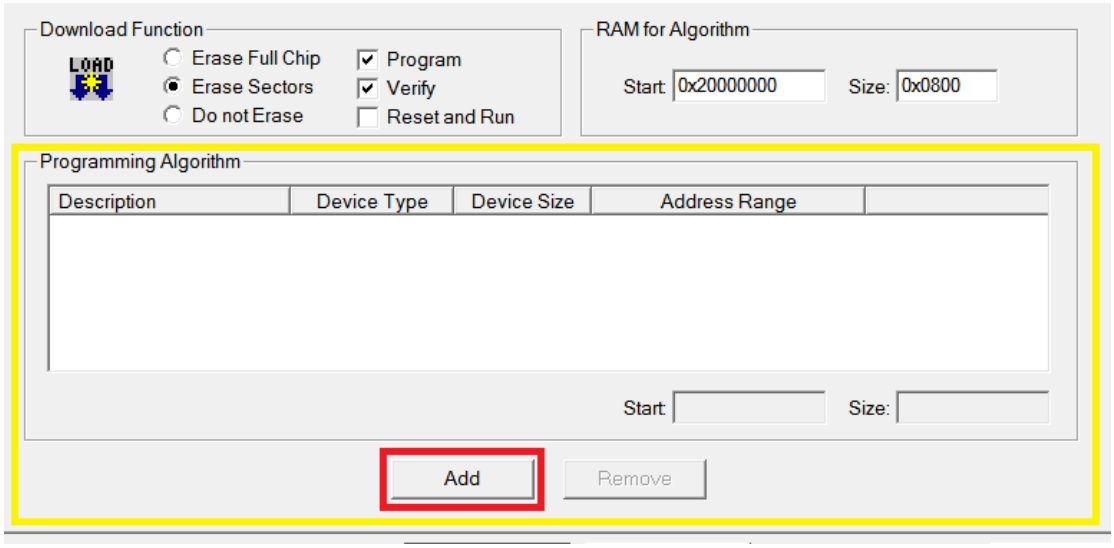


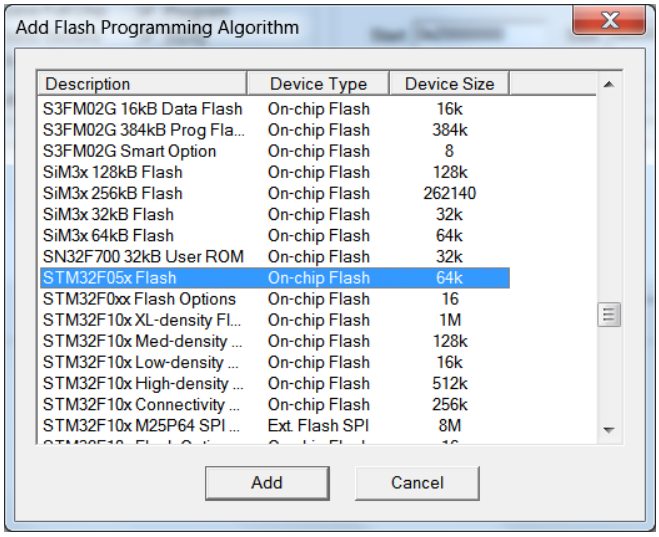
➢ ｀Settings｀をクリック、エミュレータパラメータ設定に入る。エミュレータによって、設定画面が異なるが、原理は同じである。JLINK V8バージョンエミュレータの設定について説明する。



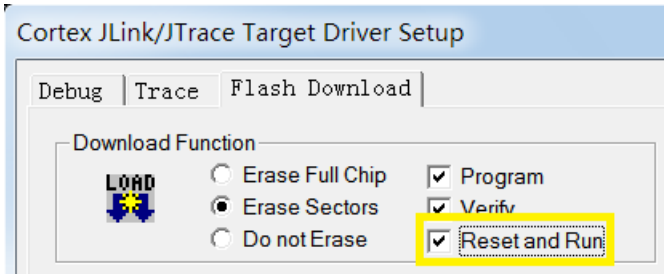


➢ ｀Flash Download｀オプション下の｀Programming Algorithm｀リストに開発ボード必要なデバイスがない場合、｀Add｀で追加する。

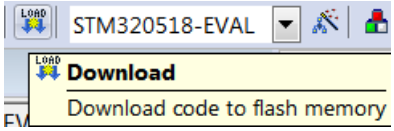




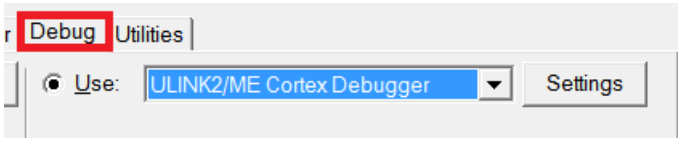
➢ ｀Reset and Run｀有効する場合、JLINK V8バージョンエミュレータはプログラムをダウンロード完了後、直接リセット・実行する。

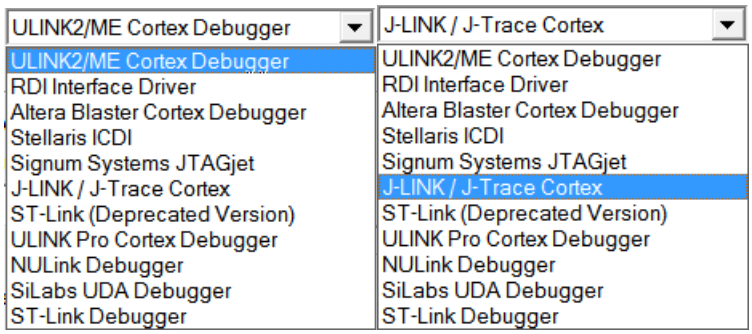


➢ ｀Dowmload to Flash Memory｀設定完了後、エミュレータでプログラムをダウンロードできる。

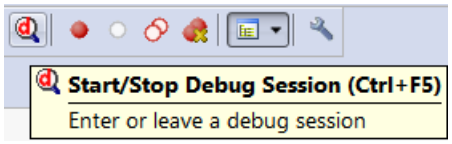


➢次はエミュレータの設定、｀Options for Target｀を選択、ULINK2エミュレータは｀ULINK2/ME Cortex Debugger｀；JLINK V8エミュレータは｀J-LINK/J-Trace Cortex｀を選択する。

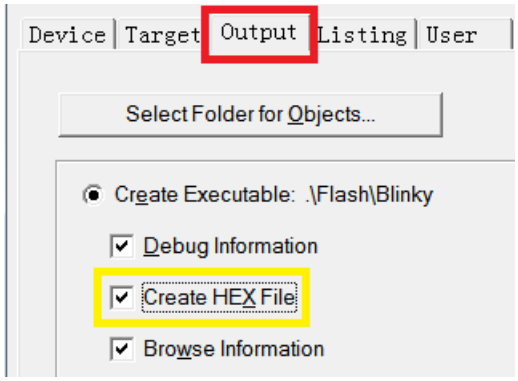




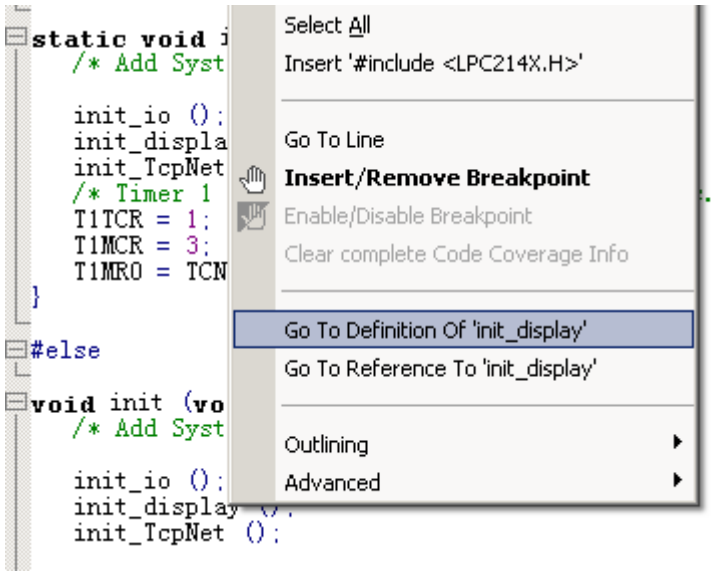
➢ ここまで、エミュレータでシミュレーションプログラムをダウンロードできる。



- hexフォーマットファイルが必要な 場合、`Create HEX File`を有効にする。



- デバッグするため、`Browse Information`も有効にすると勧める。例えばプログラムにinit_display関数を呼び出す時、関数のいずれかのフィールドでマウスを右クリックし、`Go To Definition Of init_display`を選択、ソフトウェアは自動的に当関数の実体にジャンプする。



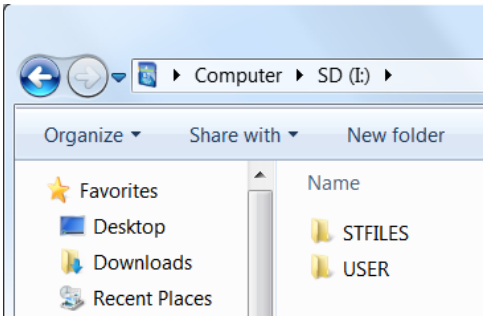

```
/*--------------------------- init_display -------------
static void init_display () {
  /* LCD Module.2x16 init*/

#ifdef USE_4BIT_LCD
  LCD_init ();
  LCD_cur_off ();
  upd_display ();
#endif
}
```

4. サンプルソースについて

4.1. ¥Code¥STM320518-EVAL_FW_V1.0.1¥Project

出荷時は開発ボードに STM320518-EVAL¥MDK-ARM を書き込んでいます。STM32F0518R8 プロセッサの大部分の周辺機能を示す。

- 液晶画面上に表示するアイコンは SD カード内に保存している為、実験の前に、¥Code¥Media にある 2 つのフォルダ（STFILES、USER）を SD カードのルートディレクトリにコピーする。



- SD カードを検出されない場合、下記の情報を表示する：

- SD カード（(STFILES、USER）コピー済み）を SD カードスロットに入れ、画面の提示に従い、5 方向のナビゲーションボタンの真ん中のキーを押し、時間設定に入る、他キーで設定を終了する。

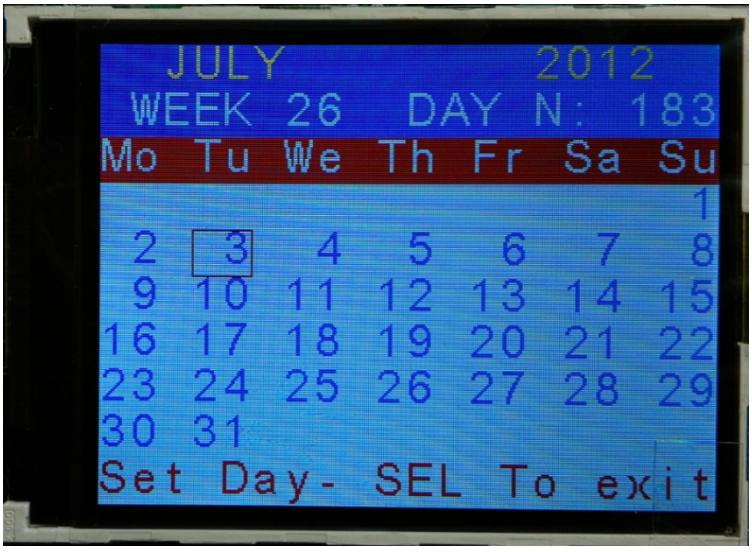
➢ 真ん中のキーで時間設定に入る：

➢ 上方向キーで数字をインクリメント　と下方向キーで数字をデクリメント、中間（確定）キーで確認。

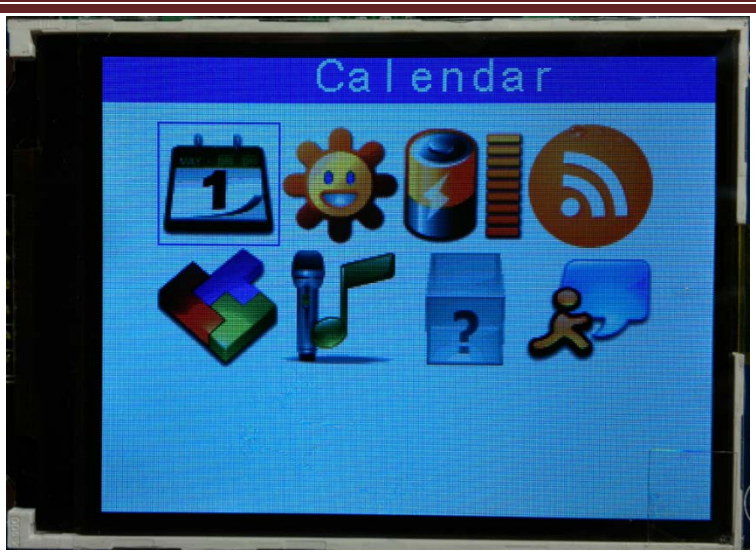
➢ 確定キーで年設定に入り、上/下方向キーで調節し、確認…下記図を 2012 年と設定する：

➢ 月設定に入り、操作は上記と同じ。下記図は JULY と設定する：

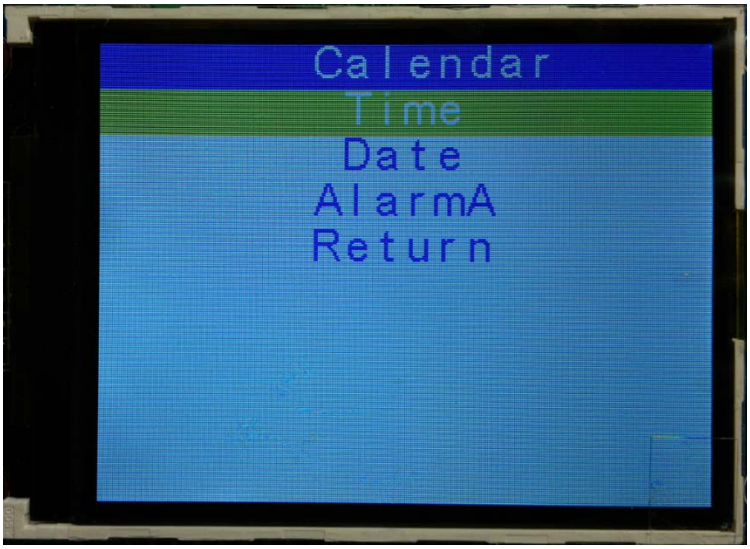
➢ 日時も同じ、上、下、左、右方向キーで調節、確認する。下記図は 3 号 Tu（火曜日）と設定する：



➢ 全ての時間設定完了後、画面の表示は下記の通り：

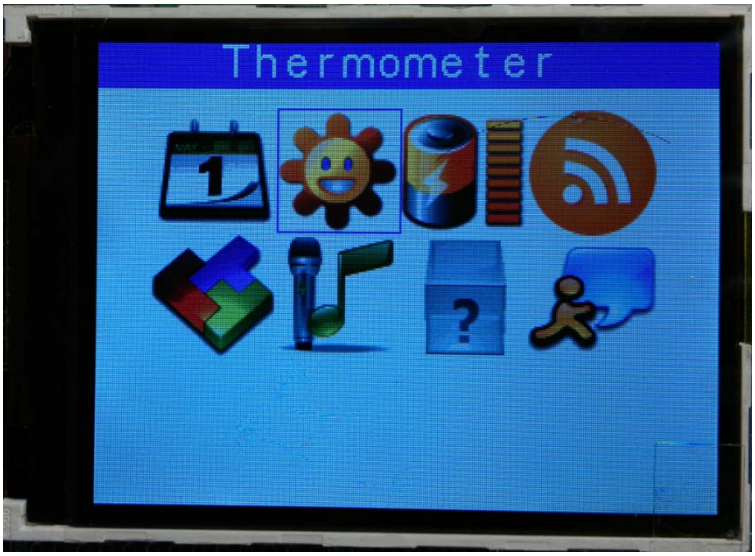


- 方向キーで Calendar（カレンダー）を選択、中間キーでオプションに入る：

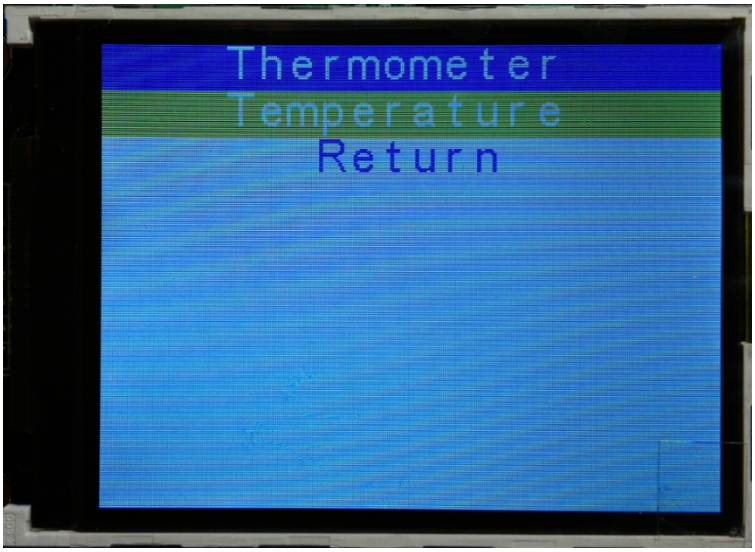


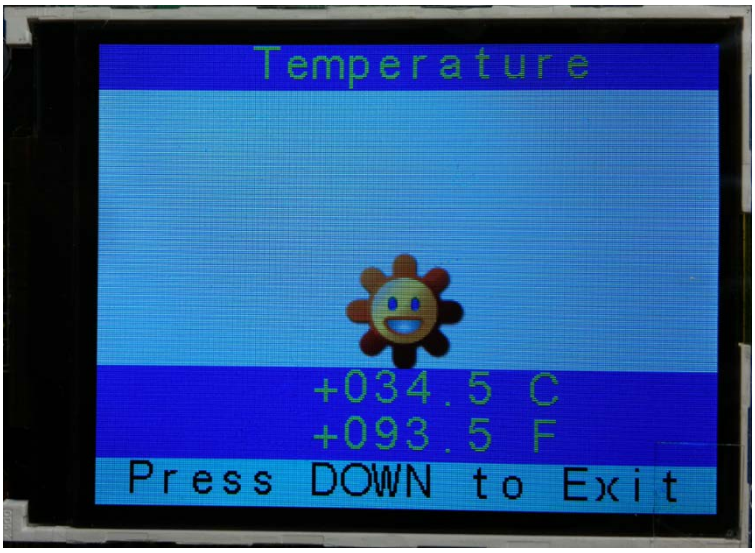
- 上、下方向キーで Time、Date、AlarmA を切り替え、サブメニューに入る。Adjust は現在時間設定、Show は現在時間表示、Return は上級に戻る：

- メインメニューで Thermometer（温度計）を選択、サブメニューに入る：

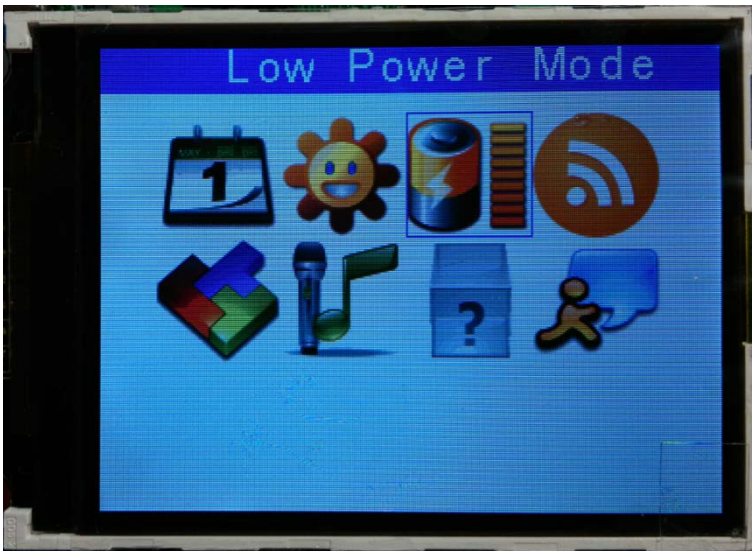


➢ Temperature で現在温度表示、Return で上級に戻る:





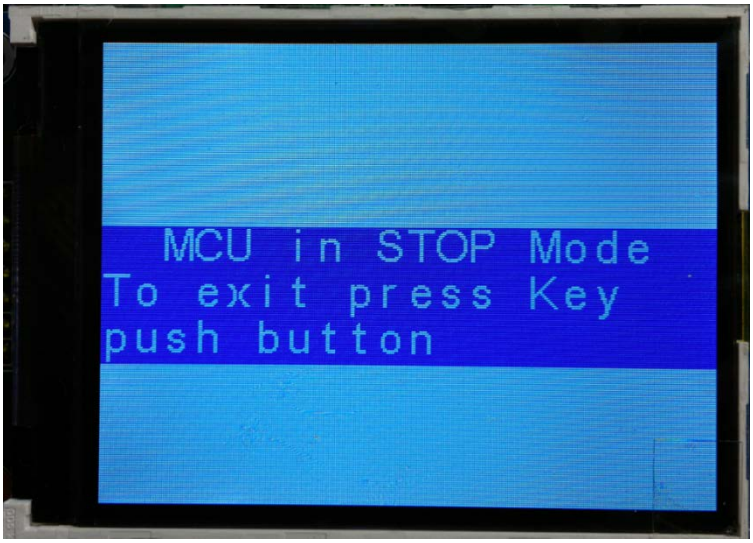
- メインメニューで Low Power Mode（省電力モード）を選択、サブメニューに入る：



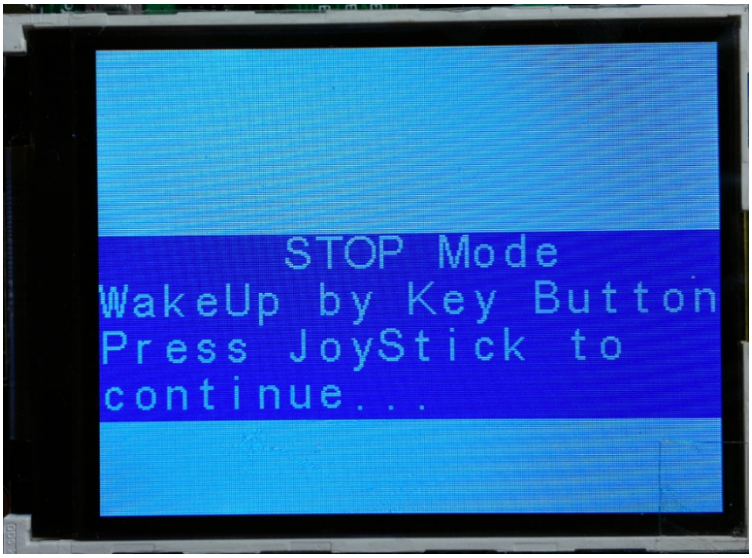
- STOP は停止モード（ストップ）、STANDBY は待機モード（スタンバイ）、Return で上級に戻る：

- モードを選択し、そのサブメニューに入る。例えば STOP モード、画面に下記のオプションが表示する：EXTI は外部キーで STOP モードを終了；RTC Alarm は RTC 割り込みで STOP モードを終了する：

- EXTI を選択、確認すると、プロセッサは STOP モードに入り、開発ボードの 4 つの LED が消灯し、5 方向ナビゲーションボタンも反応しなくなる。

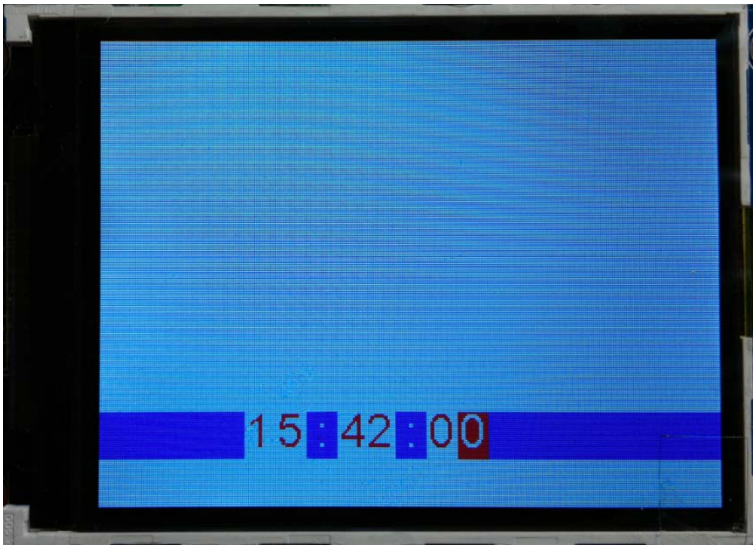


- 開発ボードの USER キーを押し、STOP モードを終了する。4 つ LED が点滅し、画面の提示に従い、5 方向ナビゲーションボタンの任意キーで STOP モードのサブメニューから上級メニューに戻る。



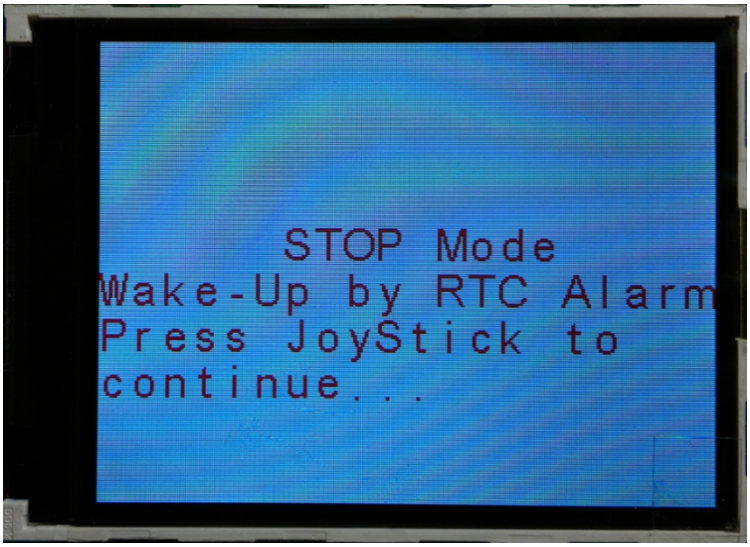
➢ STOP モードのサブメニューで RTC Alarm を選択、RTC ウェイクアップ時間設定に入る：

➢ 上、下、中間キーでウェイクアップ時間を設定する。実験のため、1 分後に設定する：



➢ RTC ウェイクアップ時間設定後、中間キーを押し STOP モードに入り、全てのキーは反応がなくなる。

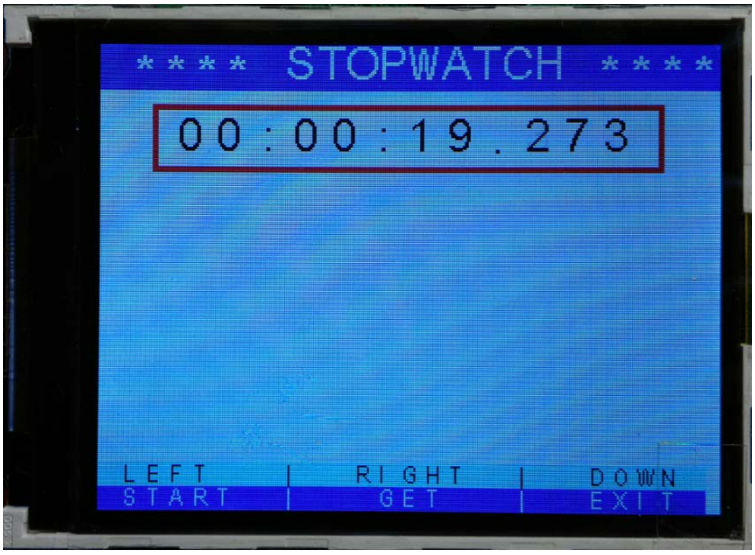
➢ RTC の現在時間が設定時間と一致すると、LED は点滅し、画面の提示に従い、5 方向ナビゲーションボタンの任意キーで STOP モードのサブメニューから上級メニューに戻る。

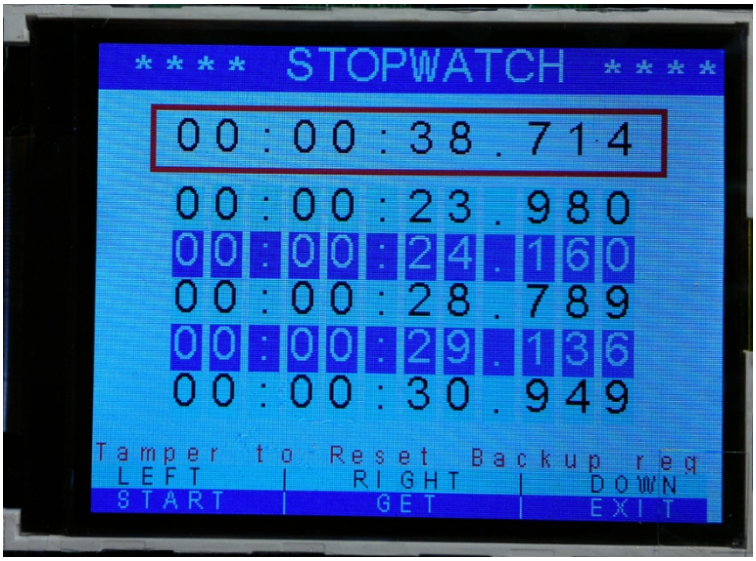


➢ メインメニューに戻り、方向キーで RF EEPROM を選択する（J9 インタフェースに RF EEPROM モジュール（別売）を接続する必要）。

- メインメニューで、方向キーApplications（アプリケーション）を選択、サブメニューに入る。StopWatch はストップウォッチデモプログラム；Timer は時間砂時計プログラム；LDR は光センサーのテストプログラム。

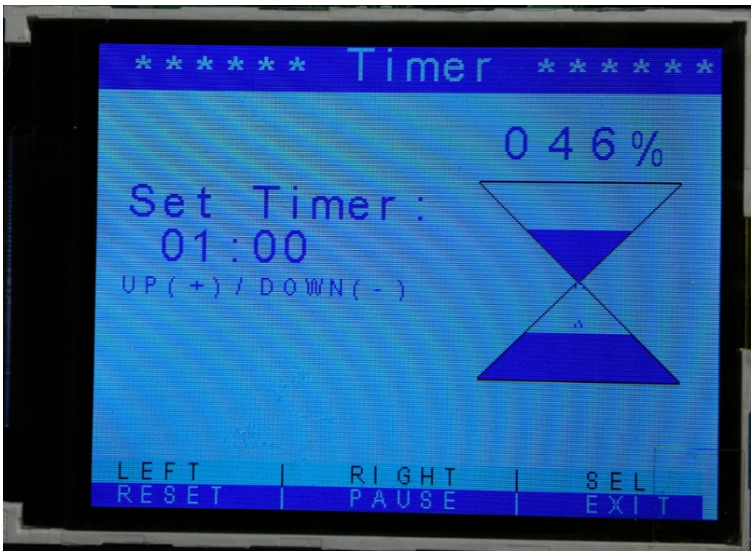
- StopWatch を選択しストップウォッチデモプログラムに入る、左方向キーでカウント開始、右方向キーカウント取り消し、下方向キーで終了する：



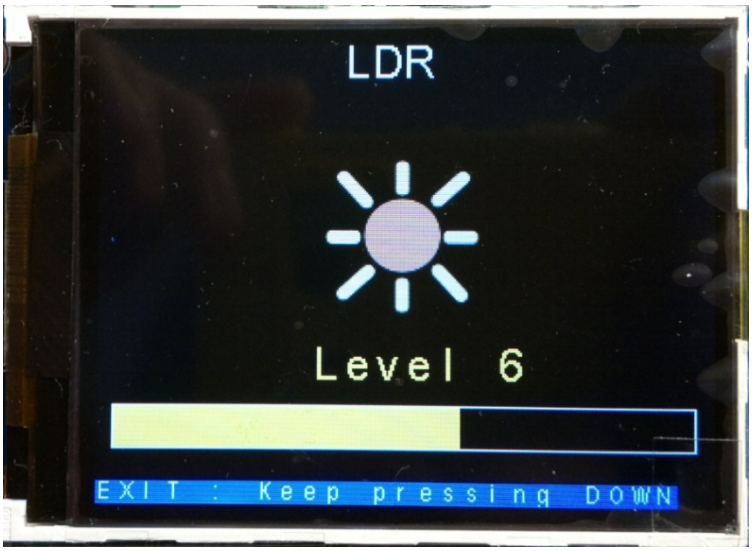


➢ Applications メニューで Timer を選択し、時間砂時計に入る。右方向キーで実行する、左方向キーでリ

セット、中間キーで終了する：



- Applications で LDR を選択、光センサーのテストプログラムに入る。プロセッサは光センサーR2 上の光度の強弱で画面の表示状態を変更する。下方向キーで終了する：



➢ メインメニューで Wave Record は音声の記録や再生機能（拡張ボード（別売）を接続する必要）：

4.2. ¥Code¥STM32F0xx_StdPeriph_Lib_V1.0.0¥Project

4.2.1. ¥IOToggle¥MDK-ARM

開発ボード上の LED1、LED2 をフラッシュを制御するプログラムである。プログラムは交替的に BSRR 及び BRR レジスタに値 0x0C00（0000 1100 0000 0000）を与え、PC10、PC11 ピン状態を 0、1 の交替変化する、I/O＝1、LED 点灯。

4.2.2. ¥USART_Printf¥MDK-ARM

本ディレクトリ下のプログラムは stdio.h の printf 関数を USART にリダイレクトする。ハイパーターミナルにプリントアウト：｀USART Printf Example: retarget the C library printf function to the USART｀、

シリアルポートのボーレートは 115200；ハードウェアフロー制御なし。

➢ リダイレクト動作は下記の通り：

```
#define PUTCHAR_PROTOTYPE int fputc(int ch、 FILE *f)

PUTCHAR_PROTOTYPE
{
  //PUTCHAR_PROTOTYPE 関数にシリアル送信関数をパッケージした
USART_SendData(EVAL_COM1、 (uint8_t) ch);
  // データ送信完了待ち
  while (USART_GetFlagStatus(EVAL_COM1、 USART_FLAG_TXE) == RESET)
  {}
  return ch;
}
```

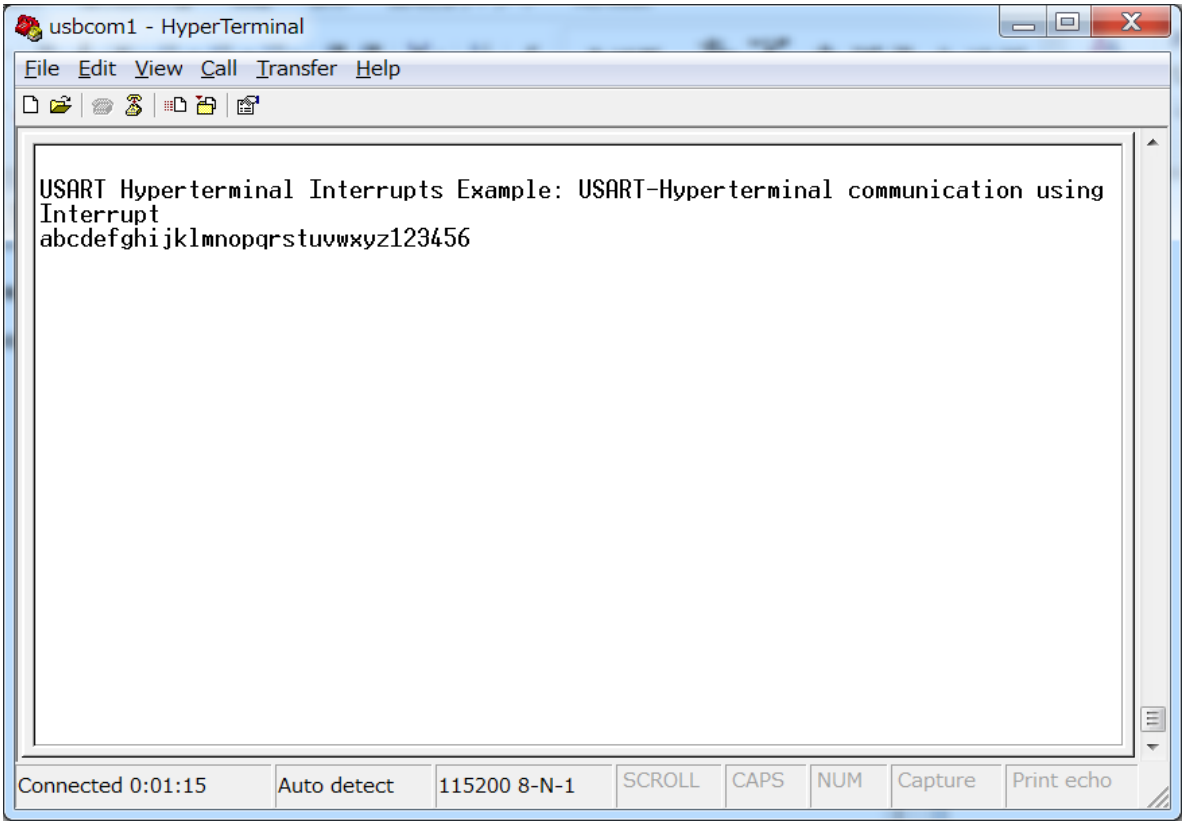
4.2.3. ¥USART_HyperTerminal_Interrupt¥MDK-ARM

シリアルポートテストです。割り込みモードで受送信する。プログラムはハイパーターミナル送信した文字をハイパーターミナルに返送する。受信文字数は 32 個を超えると、LED1、LED2 点灯、プログラムは無限ループに入る。

➢ 割り込みプログラムは stm32f0xx_it.c の USART1_IRQHandler：

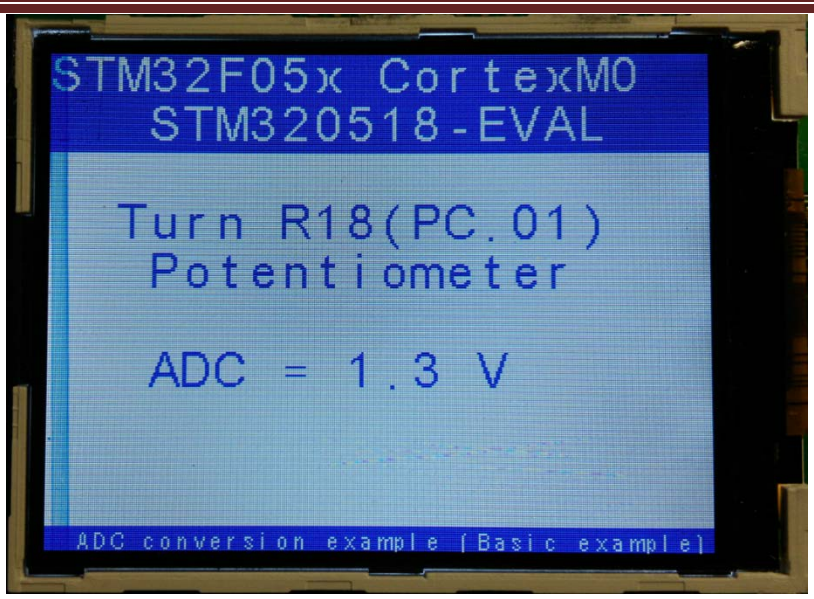
```
void USART1_IRQHandler(void)
{
  if(USART_GetITStatus(EVAL_COM1、 USART_IT_RXNE) != RESET)
  {
    /* シリアルポートからデータを受信、レジスタは１バイトを取り出す */
RxBuffer[RxCount++] = USART_ReceiveData(EVAL_COM1);
/* 受信データを返信 */
USART_SendData(EVAL_COM1、 RxBuffer[RxCount-1]);

    if(RxCount == NbrOfDataToRead) //受信データ長は、受信バッファの長さに等しい場合
    {
      /* 受信割り込みオフ*/
      USART_ITConfig(EVAL_COM1、 USART_IT_RXNE、 DISABLE);
    }
  }

  if(USART_GetITStatus(EVAL_COM1、 USART_IT_TXE) != RESET)
  {
    /* １バイトのデータを送信レジスタに書き込む */
    USART_SendData(EVAL_COM1、 TxBuffer[TxCount++]);
```

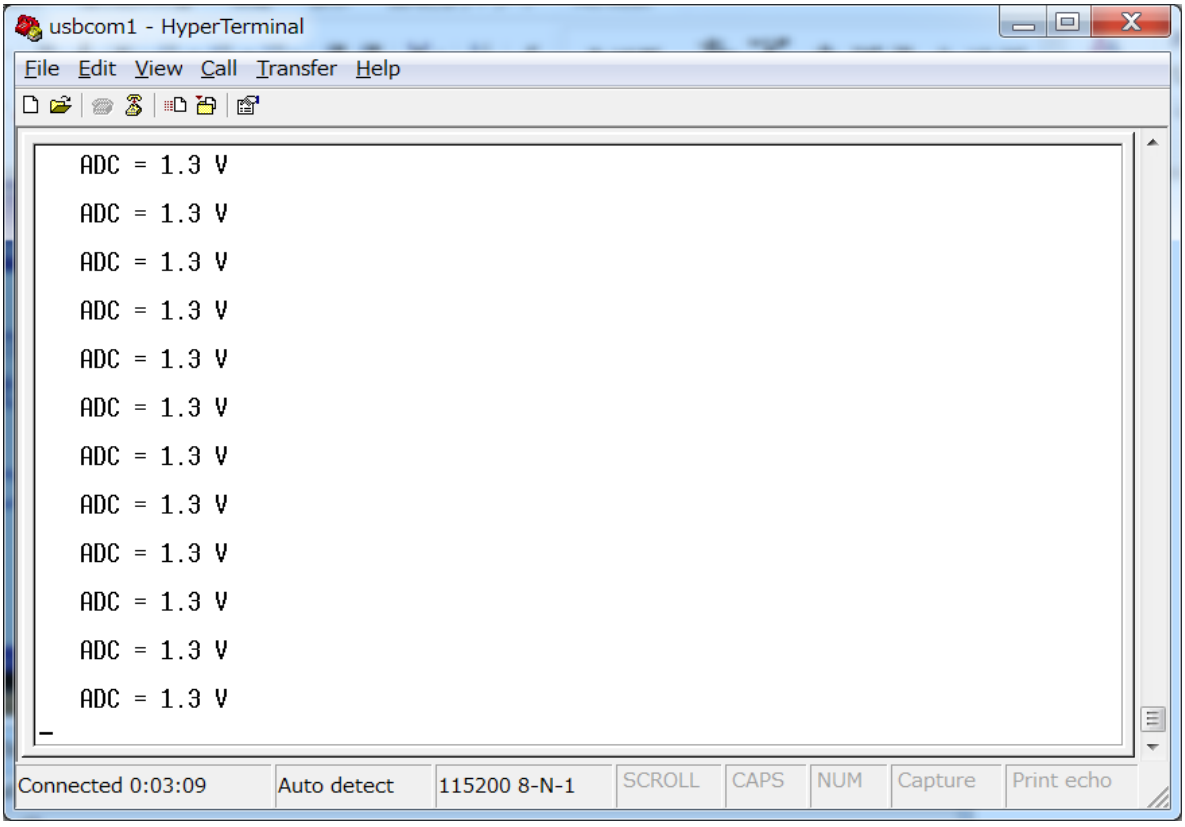
```
    if(TxCount == NbrOfDataToTransfer)// 送信データ長は、送信バッファの長さに等しい場合
    {
      /* 送信割り込みオフ*/
      USART_ITConfig(EVAL_COM1、 USART_IT_TXE、 DISABLE);
    }
  }
}
```



4.2.4.　¥ADC_Basic_Example¥MDK-ARM

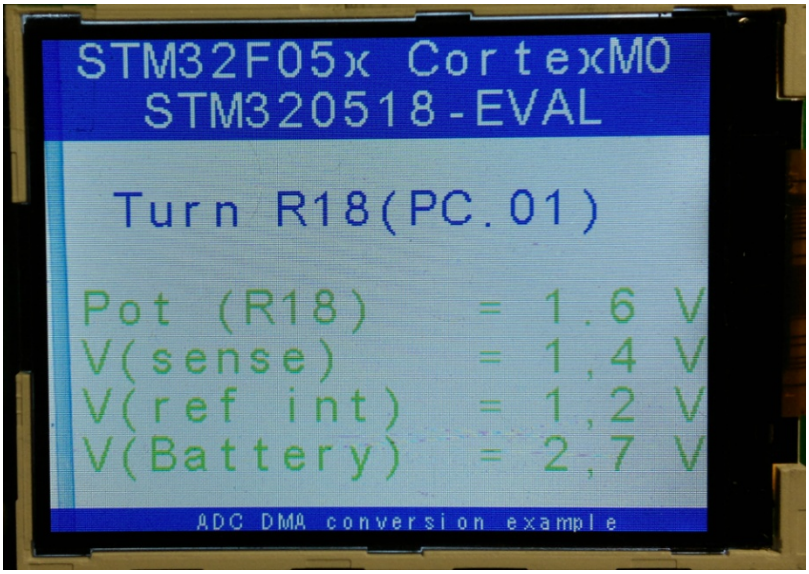
本ディレクトリ下のプログラムは ADC1 チャンネル 11 を使用し、PC1 ピン接続している可変抵抗の電圧をスクリーンに表示する。ハイパーターミナルのボーレートは 115200、ハードウェアフロー制御なし、ハイパーターミナルの方も同じ電圧値をプリントアウトする。

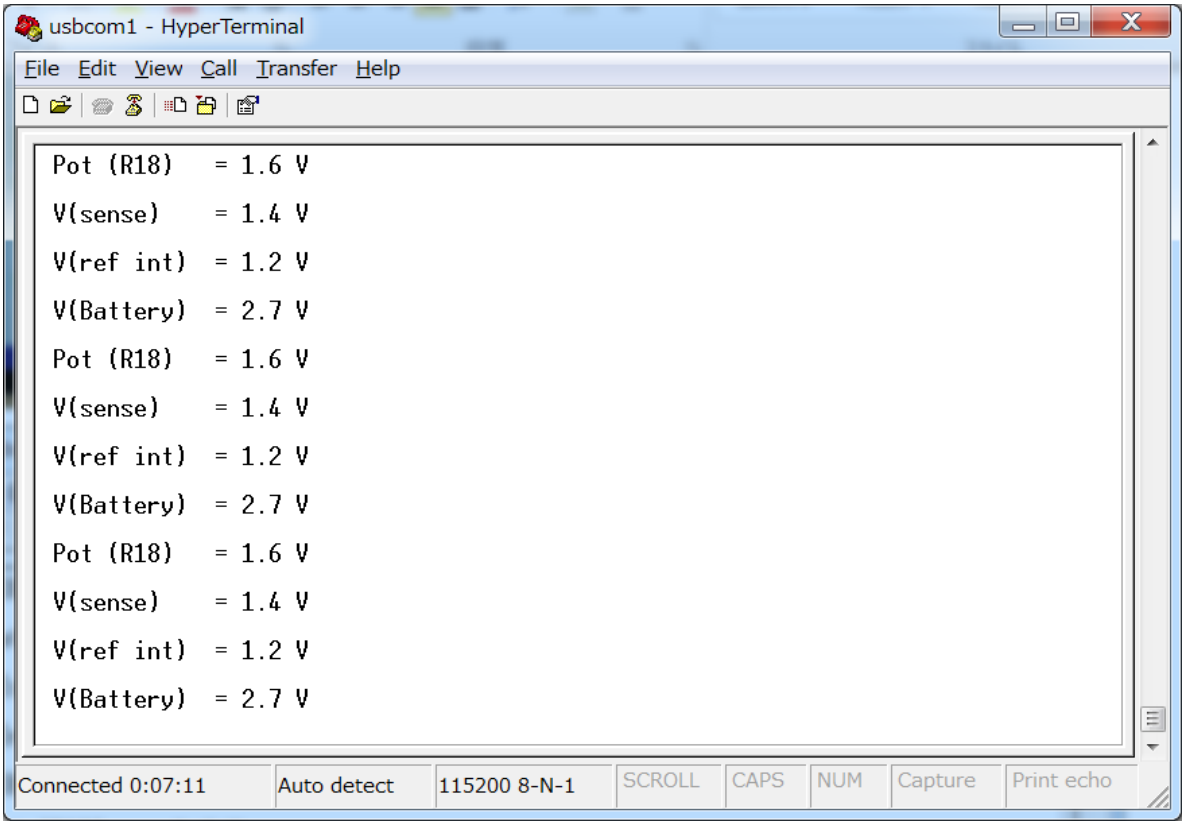




4.2.5. ¥ADC_DMA¥MDK-ARM

STM32F051R8 には 16 外部アナログ入力チャネル、1 チャネル内部温度センサ、1 チャネル内部基準電圧、1 外部 VBAT 電源ピンを監視するためのチャネルがある。本ディレクトリ下のプログラムは上記のセンサ、ピンの電圧値をメモリに DMA し、スクリーンにプリントアウトする。





- ADC1 の設定は main.c 中の ADC1_DMA_Config 関数で実行する。
- ①外部アナログ入力チャネル 11 対応する PC1 ピンを初期化：

```
/* GPIOC のクロックを有効する */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC、 ENABLE);

   /* ADC1 の クロックを有効する  */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1、 ENABLE);
```

```
  /* DMA1 の クロックを有効する  */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1 、 ENABLE);

  /* ADC チャンネル 11 をアナログ入力に設定する */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC、 &GPIO_InitStructure);
```

➢ ②DMA チャンネル 11 を設定する：

```
/* DMA1 チャンネル 11 の設定をリセット */
  DMA_DeInit(DMA1_Channel1);

  /* ADC1 のデータレジスタアドレスを DMA チャンネル 11 に割り当てる*/
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_Address;

  /* DMA チャンネル 11 のメモリベースアドレスを割り当て */
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)RegularConvData_Tab;

  /*周辺はソース/ターゲットを指定、ここではソース */
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;

  /*指定されたチャネルデータユニットのバッファサイズに割り当て*/
  DMA_InitStructure.DMA_BufferSize = 4;

  /*インクリメンタルペリフェラルアドレスレジスタ判断*/
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;

  /*インクリメンタルメモリアドレスレジスタ判断*/
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;

  /* 周辺データの幅設定、ハーフバイトの幅に設定する */
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;

  /*メモリデータの幅設定、ハーフバイトの幅に設定する  */
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;

  /* DMA の動作モード、ループモードに設定する*/
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;

/* 優先レベルを設定する */
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
```

```
/*メモリ - メモリ転送モードに設定する*/
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

/* DMA_InitStructure データ構成により DMA を初期化 */
DMA_Init(DMA1_Channel1、 &DMA_InitStructure);
/* DMA1 有効する*/
DMA_Cmd(DMA1_Channel1、 ENABLE);

/* ADC DMA 請求 ループモードに設定する */
ADC_DMARequestModeConfig(ADC1、 ADC_DMAMode_Circular);

/* ADC_DMA 有効する*/
ADC_DMACmd(ADC1、 ENABLE);
```

- ③ADC 初期化、データ構造の中にコンフィギュレーションデータを充填する。

```
/* ADC データ構造を初期化 */
ADC_StructInit(&ADC_InitStructure);

/*変換解像度を設定する、12bit */
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;

/*連続モードまたはシングルモード設定、ここでは連続モード*/
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;

/*トリガモード設定、ここではトリガーなし*/
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;

/*データアライメント設定、ここでは右揃えを選択*/
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

/*走査方向設定、Backward （後ろ向きに）*/
ADC_InitStructure.ADC_ScanDirection = ADC_ScanDirection_Backward;
ADC_Init(ADC1、 &ADC_InitStructure);
```

- ④ADC1 各チャンネルのサンプリング周波数設定する：

```
/* ADC1 外部アナログ入力チャンネル 11 のサンプリング周波数を 55.5 の ADC クロックサイクルに設定する*/
ADC_ChannelConfig(ADC1、 ADC_Channel_11 、 ADC_SampleTime_55_5Cycles);

/* ADC1 内部温度センサチャンネルのサンプリング周波数を 55.5 の ADC クロックサイクルに設定する
```

```
  */
    ADC_ChannelConfig(ADC1、 ADC_Channel_TempSensor、 ADC_SampleTime_55_5Cycles);
    /* 温度センサチャンネル有効する */
    ADC_TempSensorCmd(ENABLE);

    /* 内部基準電圧チャンネルのサンプリング周波数を 55.5 の ADC クロックサイクルに設定する */
    ADC_ChannelConfig(ADC1、 ADC_Channel_Vrefint 、 ADC_SampleTime_55_5Cycles);
    ADC_VrefintCmd(ENABLE);

    /*内部基準電圧 VBAT 電源ピンチャンネルのサンプリング周波数を 55.5 の ADC クロックサイクルに
設定する
    ADC_ChannelConfig(ADC1、 ADC_Channel_Vbat 、 ADC_SampleTime_55_5Cycles);
    ADC_VbatCmd(ENABLE);
```

4.2.6.　¥DAC_ADC¥MDK-ARM

プログラムは DAC と ADC を使用する、プログラムは割り込みで、ADC1 チャンネル 11(PC.01)の対応ピンの電圧値を DAC 方式で DAC_OUT1 (PA4)ピンで表現できる。

➢　プログラムは ADC_Config、DAC_Config２つの設定関数と１つの割り込み関数で組み合わせる。ADC 変換設定は変換完成後割り込みを生成し(EOC)、ADC 変換したデジタル量をアナログ量に変換し、出力する。可変抵抗 R18 の抵抗値を調節し、マルチメータで PA4 ピンの対応変化を測定する。

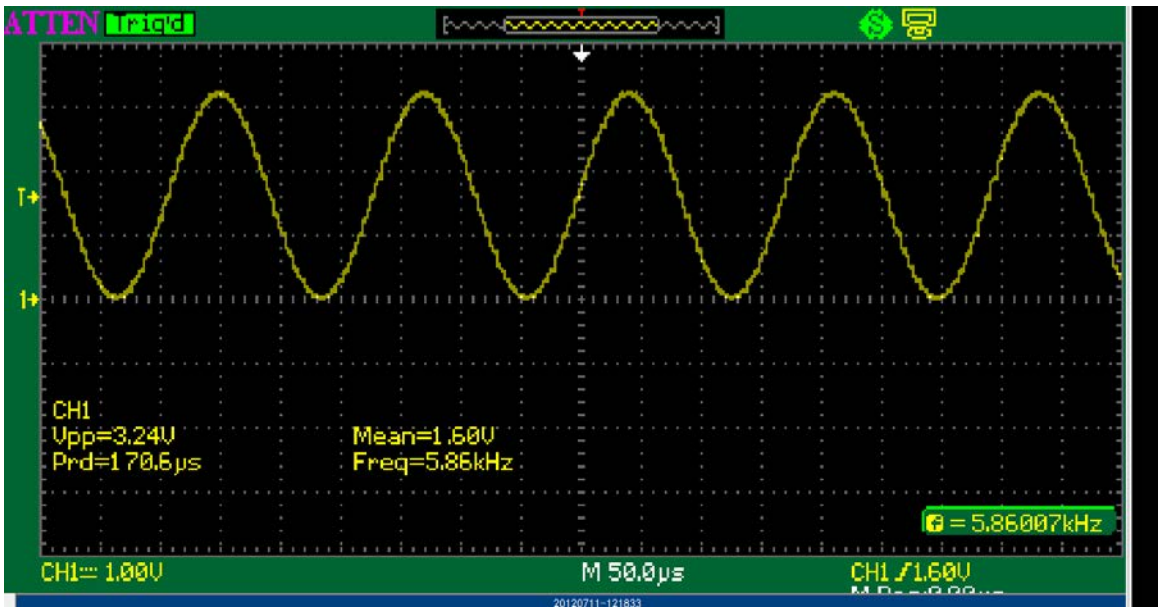
```
  void DAC_Config(void)
  {
    DAC_InitTypeDef    DAC_InitStructure;
    GPIO_InitTypeDef   GPIO_InitStructure;

    /* Enable GPIOA clock */
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA、 ENABLE);

    /* Configure PA.04 (DAC_OUT1) in analog mode -------------------------*/
    GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init(GPIOA、 &GPIO_InitStructure);

    /* DAC クロック有効する、DAC ピン PA4 */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC、 ENABLE);

    /* DAC 設定、トリガーなし */
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
    /* バッファ出力有効する */
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
```

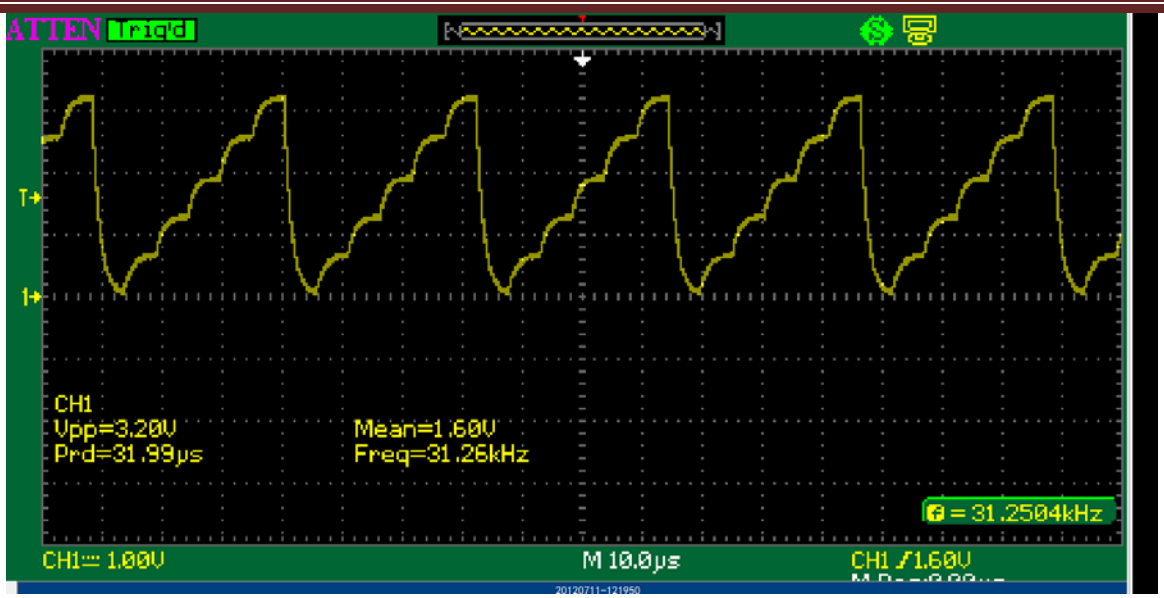
```
  /* DAC チャンネル 1 初期化 */
  DAC_Init(DAC_Channel_1、 &DAC_InitStructure);

  /* DAC チャンネル 1 有効する */
  DAC_Cmd(DAC_Channel_1、 ENABLE);
}

void ADC_Config(void)
{
  ADC_InitTypeDef    ADC_InitStructure;
  GPIO_InitTypeDef   GPIO_InitStructure;
  NVIC_InitTypeDef   NVIC_InitStructure;

  /* GPIOC クロック有効する、ADC ピン PC0.1 */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC、 ENABLE);

  /* ADC1 クロック有効する */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1、 ENABLE);

  /*ADC1 チャンネル 11 を入力モードに設定する */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;       // GPIO 模拟/输出模式
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;//内部电阻没有上拉或者下拉
  GPIO_Init(GPIOC、 &GPIO_InitStructure);

  /*ADC1 設定取り消し */
  ADC_DeInit(ADC1);
  ADC_StructInit(&ADC_InitStructure);

  /* 12 bits 精度  */
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  /* 連続変換モード */
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  /* 外部トリガーなし */
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  /*変換されたデータは右揃え*/
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  /*上向きに走査*/
  ADC_InitStructure.ADC_ScanDirection = ADC_ScanDirection_Upward;
  ADC_Init(ADC1、 &ADC_InitStructure);

  /* ADC1 外部アナログ入力チャンネル 11 サンプリング周波数を 239.5 ADC クロックサイクルに設定
```

```
する*/
    ADC_ChannelConfig(ADC1、 ADC_Channel_11 、 ADC_SampleTime_239_5Cycles);

    /* 変換完了後割り込み生成すると設定する */
    ADC_ITConfig(ADC1、 ADC_IT_EOC、 ENABLE);

    /* 割り込み設定と有効する、割り込み関数 ADC1_COMP_IRQn */
    NVIC_InitStructure.NVIC_IRQChannel = ADC1_COMP_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    /* ADC1 有効する */
    ADC_Cmd(ADC1、 ENABLE);

    /* ADC 変換開始、関数は下記の通り */
    ADC_StartOfConversion(ADC1);
}
```

4.2.7. ku¥DAC_SignalsGeneration¥MDK-ARM

プログラムは DAC を使用し、DMA 方式で正弦波またはエスカレーターの波形を生成する。USER キーで波形を切り替える。





➢本プログラムは外部割り込み用の USER キー(PB8) 、DA 変換のタイマーTIM2、伝送 DMA チャンネル 13 と DAC チャンネル 1 を使用する。

➢先ず DAC の初期化を行う：

```
void DAC_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* DMA を使用するため、DMA1 のクロック有効する */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1、 ENABLE);

  /* DAC のクロック有効する  */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC、 ENABLE);

  /* DAC チャンネル 1 は PA.04 を使用 、GPIOA のクロック有効する */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA、 ENABLE);

  /* PA.04 (DAC_OUT1)をアナログ入力に設定する、内部にプルダウン抵抗なし */
  GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA、 &GPIO_InitStructure);
}
```

➢次は TIM2 設定、TIM2 を DAC 変換のトリガーソースとして、STM32F051 プロセッサは ADC_CFGR1 のレジスタの TSELx[2:0]ビットの設定により、トリガーソースを選択する (TIM2、TIM3、TIM6、TIM16、外部割り込み 9 とソフト制御ビット(Reference manual *13.3.6 DAC trigger selection*))。

Source	Type	TSEL[2:0]
Timer 6 TRGO event	Internal signal from on-chip timers	000
Timer 3TRGO event		001
Reserved		010
Timer 15 TRGO event		011
Timer 2 TRGO event		100
Reserved		101
EXTI line9	External pin	110
SWTRIG	Software control bit	111

```
/* TIM2 クロック有効する */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2、 ENABLE);

  /*構造 TIM_TimeBaseStructure 内部のメンバーに値を設定する、デフォルト設定は：
TIM_TimeBaseInitStruct->TIM_Period = 0xFFFFFFFF;
  TIM_TimeBaseInitStruct->TIM_Prescaler = 0x0000;
  TIM_TimeBaseInitStruct->TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseInitStruct->TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInitStruct->TIM_RepetitionCounter = 0x0000;
*/
  TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);

/*オートリロード値を設定する、カウンターは 0 からカウントする、オートリロード値と一致する場合、割り込みを生成し、0 にリセットする。(0x0000-0xFFFF 中の任意値に設定できる。) */
  TIM_TimeBaseStructure.TIM_Period = 0xFF;

  /*プリスケーラ値を設定する、範囲は 0x0000-0xFFFF */
  TIM_TimeBaseStructure.TIM_Prescaler = 0x0;

  TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
  /*計算器はインクリメントモード */
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2、 &TIM_TimeBaseStructure);

  /* TIM2 が生成する割り込みを出力トリガーとする */
  TIM_SelectOutputTrigger(TIM2、 TIM_TRGOSource_Update);

  /* TIM2 有効する */
  TIM_Cmd(TIM2、 ENABLE);
```

➢USER キーを外部割り込みモードに設定する、割り込み関数は波形出力ビットを反転する。

```
STM_EVAL_PBInit(BUTTON_KEY、 BUTTON_MODE_EXTI);
void EXTI4_15_IRQHandler(void)
```

```
  {
    if(EXTI_GetITStatus(KEY_BUTTON_EXTI_LINE) != RESET)
    {
      /* メインプログラムにキー押された状態を告知し、メインプログラムが対応動作完了後、値 0
を与える*/
      WaveChange = !WaveChange;

      /* メインプログラムに波形需要変更を告知する */
      SelectedWavesForm = !SelectedWavesForm;

      /*割り込みフラグクリア*/
      EXTI_ClearITPendingBit(KEY_BUTTON_EXTI_LINE);
    }
  }
```

- 最後に、メインプログラム実行中正弦波やエスカレーター波形（2 つのサブプログラム）を生成する

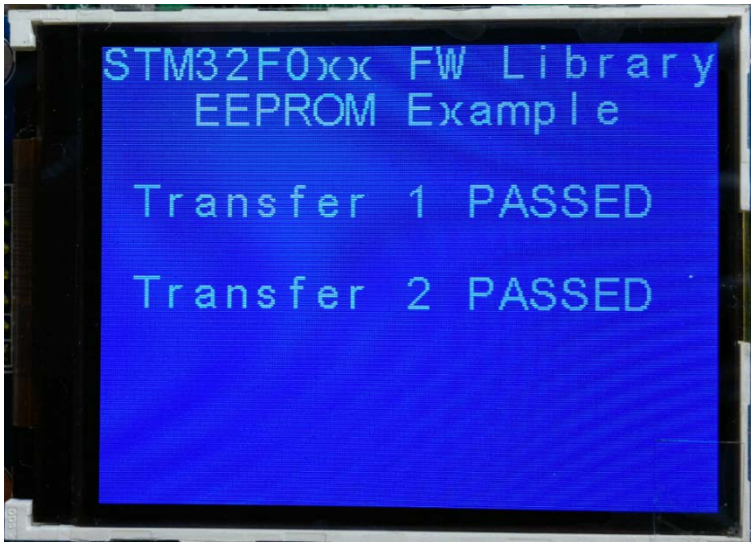
```
  /* 正弦波を生成するプログラム */
  DAC_DeInit();

  /* DAC 設定、タイマオーバフロー割り込みを DAC 変換トリガーに関連する*/
  DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;
  /* 出力バッファなし */
  DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;

  /* DMA チャンネル 3 クリア、再設定する */
  DMA_DeInit(DMA1_Channel3);

  /* DMA ベースアドレスを設定、ここでは 12 ビット右揃えデータ保存レジスタアドレス*/
  DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR12R1_ADDRESS;

  /*メモリベースアドレスを設定、ここでは、正弦波配列のアドレスで、12 ビットのデータ */
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&Sine12bit;

  /*データ転送の方向を設定、ここでは、メモリからデータ読み取り(Read from memory) */
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;

  /*指定されたチャネルのバッファサイズを設定*/
  DMA_InitStructure.DMA_BufferSize = 32;

  /*レジスタ･アドレスはインクリメント/ デクリメント設定、ここでは、インクリメント*/
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;

  /* メモリアドレスはインクリメント/ デクリメント設定、ここでは、インクリメント*/
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
```

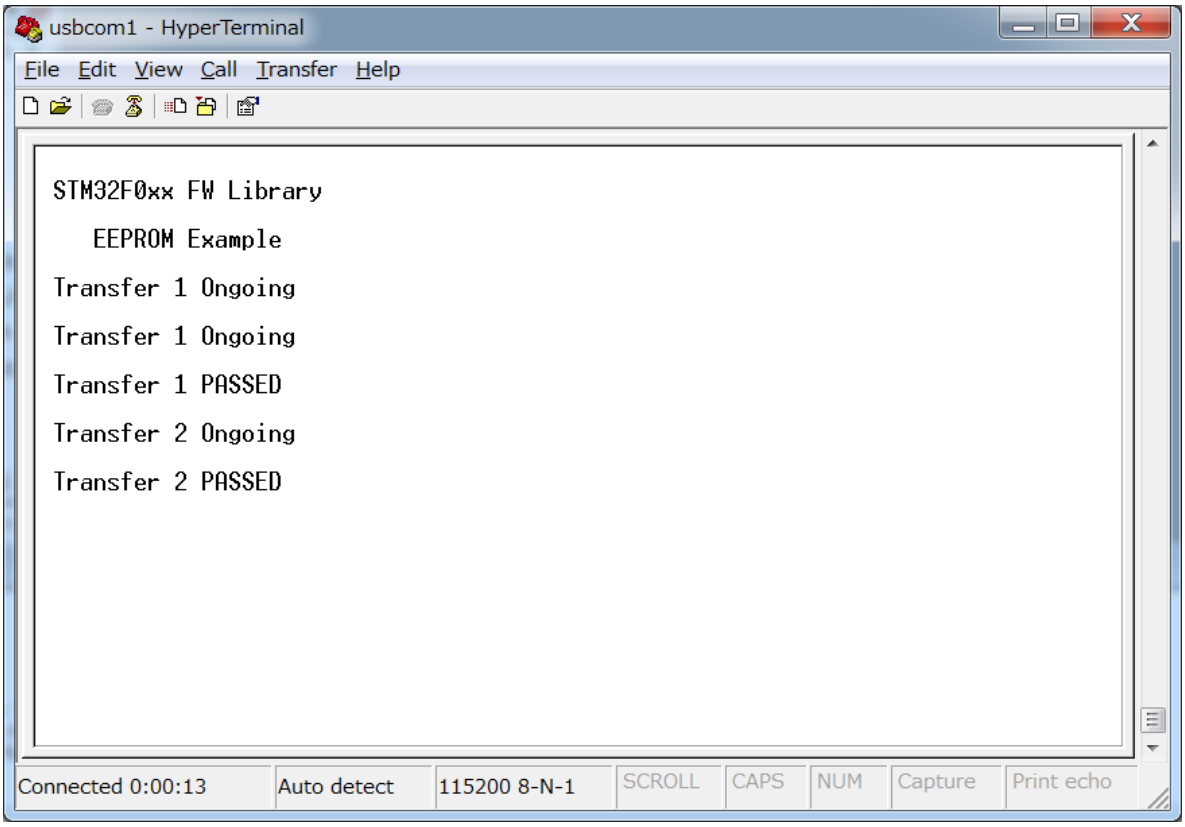
```
    /* 周辺とメモリのデータ幅をハーフワイドに設定*/
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;

  /*ループモードに設定*/
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;

    /*高優先度に設定*/
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;

    /*メモリ - メモリ転送モード禁止*/
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel3、 &DMA_InitStructure);

    /* DMA1 チャンネル 3 有効する*/
    DMA_Cmd(DMA1_Channel3、 ENABLE);

    /* DAC チャンネル 1 初期化 */
    DAC_Init(DAC_Channel_1、 &DAC_InitStructure);

    /* DAC チャンネル 1 有効する、PA.04 自動的に DAC 変換器に接続する */
    DAC_Cmd(DAC_Channel_1、 ENABLE);

    /* DAC チャンネル 1 対応 DMA 有効する */
    DAC_DMACmd(DAC_Channel_1、 ENABLE);

/* エスカレーター波形生成プログラム*/
DAC_DeInit();

/* DAC 設定、タイマオーバフロー割り込みを DAC 変換トリガーに関連する */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_T2_TRGO;

/*出力バッファなし */
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;

/* DMA チャンネル 3 クリア、再設定する*/
DMA_DeInit(DMA1_Channel3);

/* DMA ベースアドレスを設定、ここでは 8 ビット右揃えデータ保存レジスタアドレス */
DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR8R1_ADDRESS;

/* メモリベースアドレスを設定、ここでは、エスカレーター波形配列のアドレスで、8 ビットのデータ */
```

```
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&Escalator8bit;

/*指定されたチャネルのバッファサイズを設定、エスカレーター波形は正弦波のバッファより小さい */
DMA_InitStructure.DMA_BufferSize = 6;

/*周辺とメモリのデータ幅を 1 バイトワイドに設定 */
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
DMA_Init(DMA1_Channel3、 &DMA_InitStructure);

/* DMA1 チャンネル 3 有効する */
DMA_Cmd(DMA1_Channel3、 ENABLE);

/* DAC チャンネル 1 初期化*/
DAC_Init(DAC_Channel_1、 &DAC_InitStructure);

/* DAC チャンネル 1 有効する、PA.04 自動的に DAC 変換器に接続する */
DAC_Cmd(DAC_Channel_1、 ENABLE);

/* DAC チャンネル 1 対応 DMA 有効する */
DAC_DMACmd(DAC_Channel_1、 ENABLE);
```

4.2.8. ¥I2C_EEPROM¥MDK-ARM

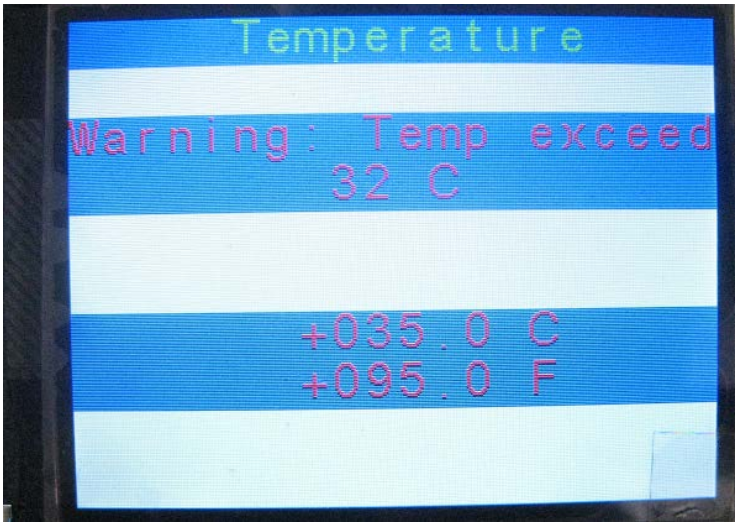
開発ボードの I2C インタフェースの EEPROM のテストプログラム。プログラムは EEPROM に固定文字列を書き込み、そしてそれを読み出し、比較する。書き込みデータと読み出しデータが一致する場合、スクリーンに Transfer PASSED を表示、でないと、Transfer FAILED を表示する。ハイパーターミナルにも同じ情報をプリントアウトする。ハイパーターミナルのボーレートは 115200、ハードウェアフロー制御なし。

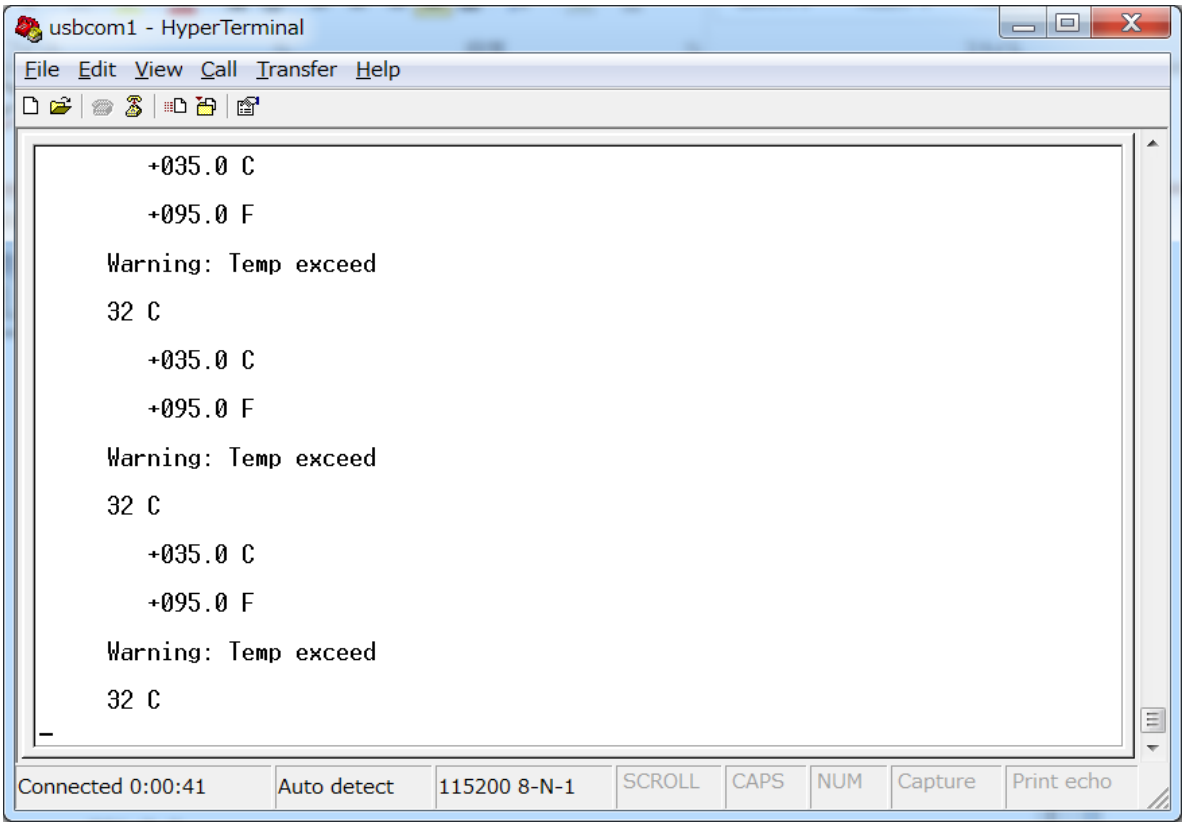




4.2.9. ¥I2C_TSENSOR¥MDK-ARM

開発ボードの I2C インタフェースの温度センサのテストプログラム。プログラムは収集温度をスクリーンに表示する。ハイパーターミナルも同じ情報をプリントアウトする。ハイパーターミナルのボーレートは 115200、ハードウェアフロー制御なし。



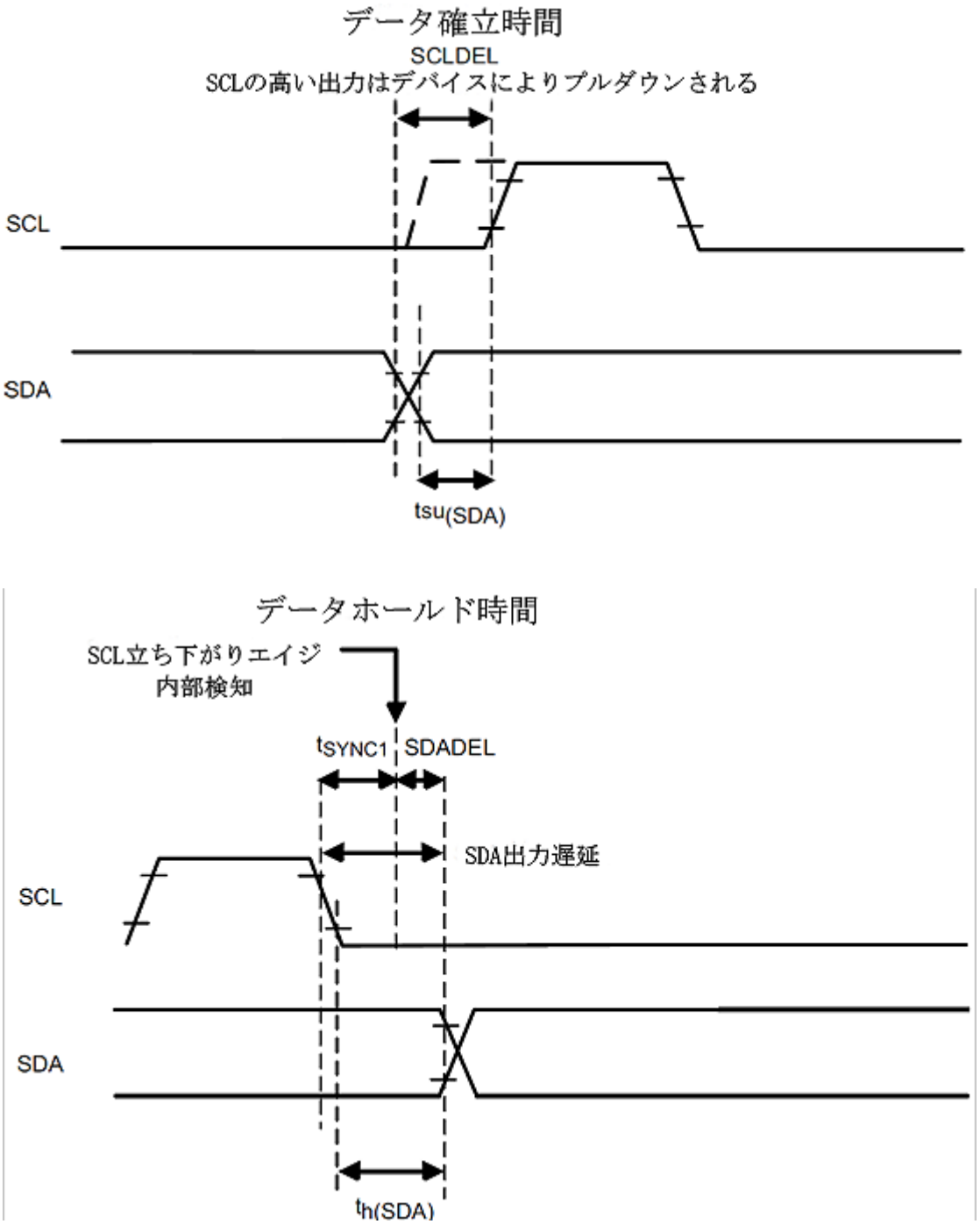


➢ I2C デバイスについて、LM75_Init 関数内の I2Cx_TIMINGR レジスタの設定は重要で、I2C 信号データ確立/ホールド時間及び SCL クロックハイ/ローレベル時間を設定する。

```
void LM75_Init(void)
{
  I2C_InitTypeDef  I2C_InitStructure;

  LM75_LowLevel_Init();

  /* LM75_I2C configuration */
  I2C_InitStructure.I2C_Mode = I2C_Mode_SMBusHost;
  I2C_InitStructure.I2C_AnalogFilter = I2C_AnalogFilter_Enable;
  I2C_InitStructure.I2C_DigitalFilter = 0x00;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_Timing = LM75_I2C_TIMING; //0x1045061D

  /* Apply LM75_I2C configuration after enabling it */
  I2C_Init(LM75_I2C、 &I2C_InitStructure);

  /* LM75_I2C Peripheral Enable */
  I2C_Cmd(LM75_I2C、 ENABLE);
}
```

➢ データ確立時間：SCL の立ち上がりエッジが到着前に、SDA ピンのデータ状態が不安定から安定状態になる必要な時間；データホールド時間：SCL の立ち下がりエッジの到着後、SDA ピン上のデータの状態は、変更前必要な維持時間。

図 1. データ確立時間とデータホールド時間



➢ データ確立/ホールド時間及び SCL クロックハイ/ローレベル時間は I2C デバイスの最小許容時間よりを長く設定する必要がある。下記図で NXP の LM75A の最小データ確立時間は 100nS(LM75A データマニュアル　P16)、最小データホールド時間 10nS、SCL クロック最小ローレベル時間 1.3uS、SCL クロック最小ハイレベル時間

0.6uS、I2Cx_TIMINGR レジスタの設定値は上記の I2C デバイスパラメータを上回る必要がある。

➢

➢ 最小数据保持时间为 10nS、SCL 时钟的低电平最小时间为 1.3uS、SCL 时钟的高电平最小时间为 0.6uS、

I²C 总线接口的动态特性 [1]

V_{CC} = 2.8 V to 5.5 V; T_{amb} = −55 °C to +125 °C; unless otherwise specified.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
T_{CLK}	SCL clock period	see Figure 12		2.5	-	-	µs
t_{HIGH}	HIGH period of the SCL clock			0.6	-	-	µs
t_{LOW}	LOW period of the SCL clock			1.3	-	-	µs
$t_{HD;STA}$	hold time (repeated) START condition			100	-	-	ns
$t_{SU;DAT}$	data set-up time			100	-	-	ns
$t_{HD;DAT}$	data hold time		[2]	0	-	-	ns
$t_{SU;STO}$	set-up time for STOP condition			100	-	-	ns
t_f	fall time	SDA and OS outputs; C_L = 400 pF; I_{OL} = 3 mA		-	250	-	ns

[1] These specifications are guaranteed by design and not tested in production.

[2] The data hold time minimum value is 10 ns for the SCL clock period of 10 µs or higher.

➢ I2Cx_TIMINGR レジスタのビットファンクションは下記の通り（Reference manual P526）：Bits 31:28 PRESC[3:0]はプリスケーラ値設定；Bits 23:20 SCLDEL[3:0]はデータの確立時間設定；Bits 19:16 SDADEL[3:0]はデータホールド時間設定；Bits 15:8 SCLH[7:0]は SCL クロックのハイレベル時間設定、Bits 7:0 SCLL[7:0]は SCL クロックのローレベル時間設定。

23.7.5 Timing register (I2Cx_TIMINGR)

Address offset: 0x10
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PRESC[3:0]				Res.	Res.	Res.	Res.	SCLDEL[3:0]				SDADEL[3:0]			
rw								rw				rw			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCLH[7:0]								SCLL[7:0]							
rw								rw							

➢ Bits 31:28 PRESC[3:0]フィールドを設定、クロックサイクル tPRESC を得る、それを基づき、tSCLDEL、tSDADEL、tSCLH、tSCLL を得る。対応関係は下記の通り：

t_{PRESC} = (PRESC+1) x t_{I2CCLK} (=HIS=1/8MHz=125nS)

t_{SCLDEL} = (SCLDEL+1) x t_{PRESC}

t_{SDADEL} = SDADEL x t_{PRESC}

t_{SCLH} = (SCLH+1) x t_{PRESC}

t_{SCLL} = (SCLL+1) x t_{PRESC}

➢ 上記の LM75_Init 関数に基づき、I2Cx_TIMINGR レジスタの値は 0x1045061D で、16 進数をバイナリにスプリットすると、0001 0000 0100 0101 00000110 00011101 を得る。従って、PRESC=1、SCLDEL=4、SDADEL=5、SCLH=6、SCLL=29、データの確立、データホールド時間を算出する。比較計算で得られる各時間は LM75A データマニュアルでの最小許容時間より長い。

t_{PRESC} = (PRESC+1) x t_{I2CCLK} (=HIS=1/8MHz=125nS) = (1+1)x 125 = 250 nS

t_{SCLDEL} = (SCLDEL+1) x t_{PRESC} = (4+1)x250 = 1250nS

t_{SDADEL} = SDADEL x t_{PRESC} = 5x250 = 1250nS

t_{SCLH} = (SCLH+1) x t_{PRESC} = (6+1)x250 = 1750nS

tSCLL = (SCLL+1) x tPRESC = (29+1)x250 = 7500nS

4.2.10. ¥SPI_MSD¥MDK-ARM

SPI インタフェースの SD カードドライバテストプログラム。プログラムはデータを SD カードのアドレス 0 からの 512 bytes スペースに書き込み、また読み取り、比較する。データ正確 LED1 点灯；データエラーLED2 点灯。

➢ SPI と SD カードの接続について、stm320518_eval.c 内の SD_LowLevel_Init 関数を参照。

```
void SD_LowLevel_Init(void)
{
  GPIO_InitTypeDef  GPIO_InitStructure;
  SPI_InitTypeDef   SPI_InitStructure;

  /* SD カードチップセレクト有効する、SD カード挿入検出、MISO、MOSI、SCK 対応ピンクロック */
  RCC_AHBPeriphClockCmd(SD_CS_GPIO_CLK | SD_SPI_MOSI_GPIO_CLK | SD_SPI_MISO_GPIO_CLK
|SD_SPI_SCK_GPIO_CLK | SD_DETECT_GPIO_CLK、 ENABLE);

  /* SPI クロック有効する */
  RCC_APB2PeriphClockCmd(SD_SPI_CLK、 ENABLE);

  /* SCK ピンプロパティ設定*/
  GPIO_InitStructure.GPIO_Pin = SD_SPI_SCK_PIN;        //対応ピン PA.05
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;         // GPIO オルタネート機能モード
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;    // GPIO 駆動速度 50MHz
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;       //プッシュプル･モード、push-pull
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;        //内部プルアップ、pull-up
  GPIO_Init(SD_SPI_SCK_GPIO_PORT、 &GPIO_InitStructure);

  /* MISO ピンプロパティ設定*/
  GPIO_InitStructure.GPIO_Pin = SD_SPI_MISO_PIN;       //対応ピン PB.04
  GPIO_Init(SD_SPI_MISO_GPIO_PORT、 &GPIO_InitStructure);

  /* MOSI ピンプロパティ設定*/
  GPIO_InitStructure.GPIO_Pin = SD_SPI_MOSI_PIN;       //対応ピン PA.07
  GPIO_Init(SD_SPI_MOSI_GPIO_PORT、 &GPIO_InitStructure);

  /* SD チップセレクトピンプロパティ設定*/
  GPIO_InitStructure.GPIO_Pin = SD_CS_PIN;                 // 対応ピン PF.05
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;        //出力モード設定
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;       // プッシュプル･モード、push-pull
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;         //内部プルアップ、pull-up
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;    // GPIO 駆動速度 50MHz
```

```
    GPIO_Init(SD_CS_GPIO_PORT、 &GPIO_InitStructure);

    /* SD 挿入設定、ピンプロパティ検知*/
    GPIO_InitStructure.GPIO_Pin = SD_DETECT_PIN;        // 対応ピン PB.15
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;        // 入力モード
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;        // 内部プルアップ、pull-up
    GPIO_Init(SD_DETECT_GPIO_PORT、 &GPIO_InitStructure);

    /* PA.05 は SPI の SCK 機能 に設定 */
    GPIO_PinAFConfig(SD_SPI_SCK_GPIO_PORT、 SD_SPI_SCK_SOURCE、 SD_SPI_SCK_AF);

    /* PB.04 は SPI の MISO 機能 に設定 */
    GPIO_PinAFConfig(SD_SPI_MISO_GPIO_PORT、 SD_SPI_MISO_SOURCE、 SD_SPI_MISO_AF);

    /* PA.07 は SPI の MOSI 機能 に設定*/
    GPIO_PinAFConfig(SD_SPI_MOSI_GPIO_PORT、 SD_SPI_MOSI_SOURCE、 SD_SPI_MOSI_AF);

    /* SPI プロパティ設定、SPI デバイスは 4 つの通信モードがある*/
    /* SPI の単方向/双方向データ･モード設定、ここでは双方向モード*/
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;       // SPI ホストモード
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;   // 8bits データ長

    /*シリアル･クロックは、定常状態はハイレベル*/
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;

    /*ビットキャプチャのクロックエッジを 2 つクロックエッジと設定*/
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;       //ソフトウェアモード SPI チップセレクト
    /* SPI ボーレートの分周値を設定する*/
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;

    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;  //最上位ビット優先送信
    SPI_InitStructure.SPI_CRCPolynomial = 7;            //
    SPI_Init(SD_SPI、 &SPI_InitStructure);

    SPI_RxFIFOThresholdConfig(SD_SPI、 SPI_RxFIFOThreshold_QF);

    SPI_Cmd(SD_SPI、 ENABLE); /*!< SD_SPI enable */
}
```

以上。