お客様各位

ZUD-CD-08-0039 1/127
2008年12月1日
NECエレクトロニクス株式会社
マイクロコンピュータ事業本部
汎用マイコンシステム事業部
開発ツールソリューショングループ
チームマネージャー 安藤 喜成
(担当：鈴木 康之)

78K0R/KG3ターゲット・ボード
QB-78K0RKG3-TB
チュートリアル・ガイド
実践編

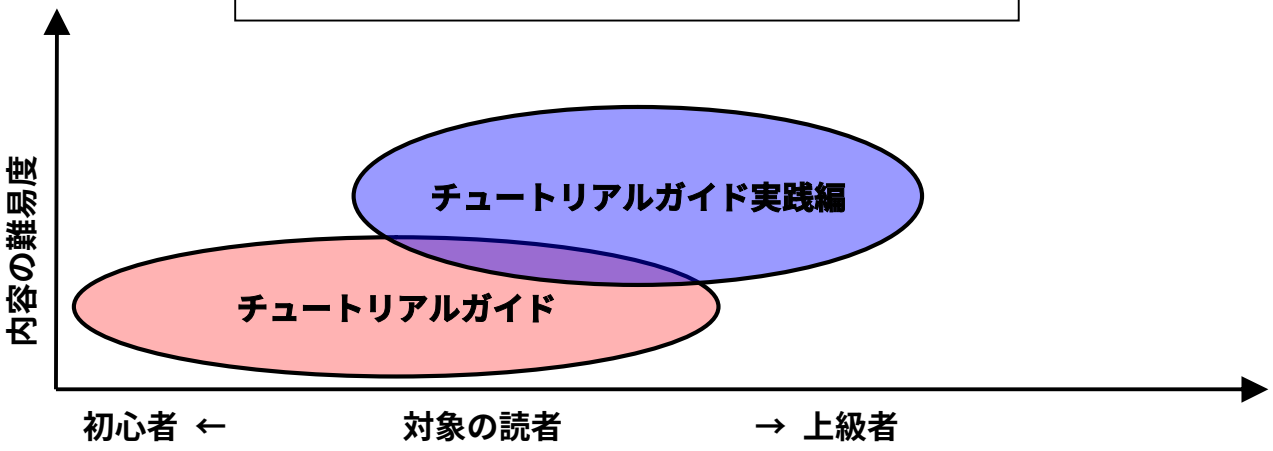
ごあいさつ

QB-78K0RKG3-TBお買い求めいただき、誠にありがとうございます。

本製品(QB-78K0RKG3-TB)は、NECエレクトロニクス社製のプログラミング機能付きオンチップ･デバッグ･エミュレータQB-MINI2(以降MINICUBE2)で使用可能なマイコン搭載のターゲット･ボードです。

チュートリアル･ガイド実践編では、マイコン中級者～上級者を対象としました。78K0Rマイコンの周辺機能を使ったシステム開発例を説明します。A/Dを用いた温度計測、I2Cを用いた外部へのデータ保存、シリアル出力、LCD表示制御を使ったシステム開発例を解りやすく説明しています。本製品と共に活用して下さい。前編の「チュートリアル･ガイド」もお読みになると一層理解が深められます。

チュートリアルガイド実践編の位置づけ



本製品に関する最新情報、必要な無償版開発ツール、およびサンプル･プログラムは、弊社webサイトにて提供しています。

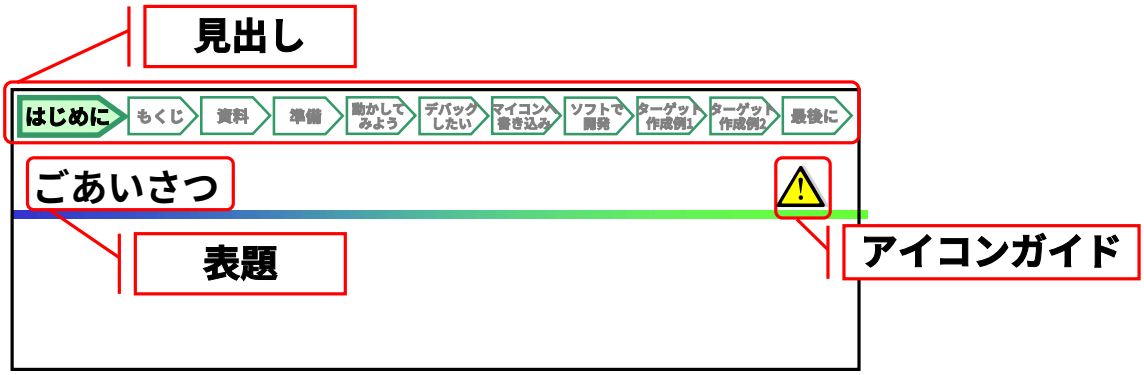
[MINICUBE2関連ツール]
http://www.necel.com/micro/ja/development/asia/minicube2/minicube2.html

[無償版開発ツール]
http://www.necel.com/micro/freesoft/

[サンプル･プログラム]
http://www.necel.com/micro/ja/development/asia/minicube2/minicube2_opt.html

本書の見方(説明)

本書では、わかりやすくするため下記の構成になっています。



[見出し]
本書の章に相当します。太い緑の線で囲まれたところは、現在の章を表します。また、薄い緑の線で囲まれたところは、他の章を表していますので本書全体の構成が、どのページでも一目でわかるようになっています。

[表題]
章の中で伝えたいことをいくつかの項目に分けています。本書中では「xxページを参照して下さい」という表現はしません。例えば「[はじめに]の[本書の見方]を参照して下さい」という表現になります。

[アイコンガイド]
アイコンガイドで、そのページ内容を一目で分かるようにアイコンで表現しています。後で読めばいいのか、必ず読む必要があるのか、このアイコンガイドを参考にして下さい。アイコンガイドには、下記の種類があります。

各章の学習する時間の目安を示します。単位は分です。


注意する内容です。

必ず読む必要のある内容です。

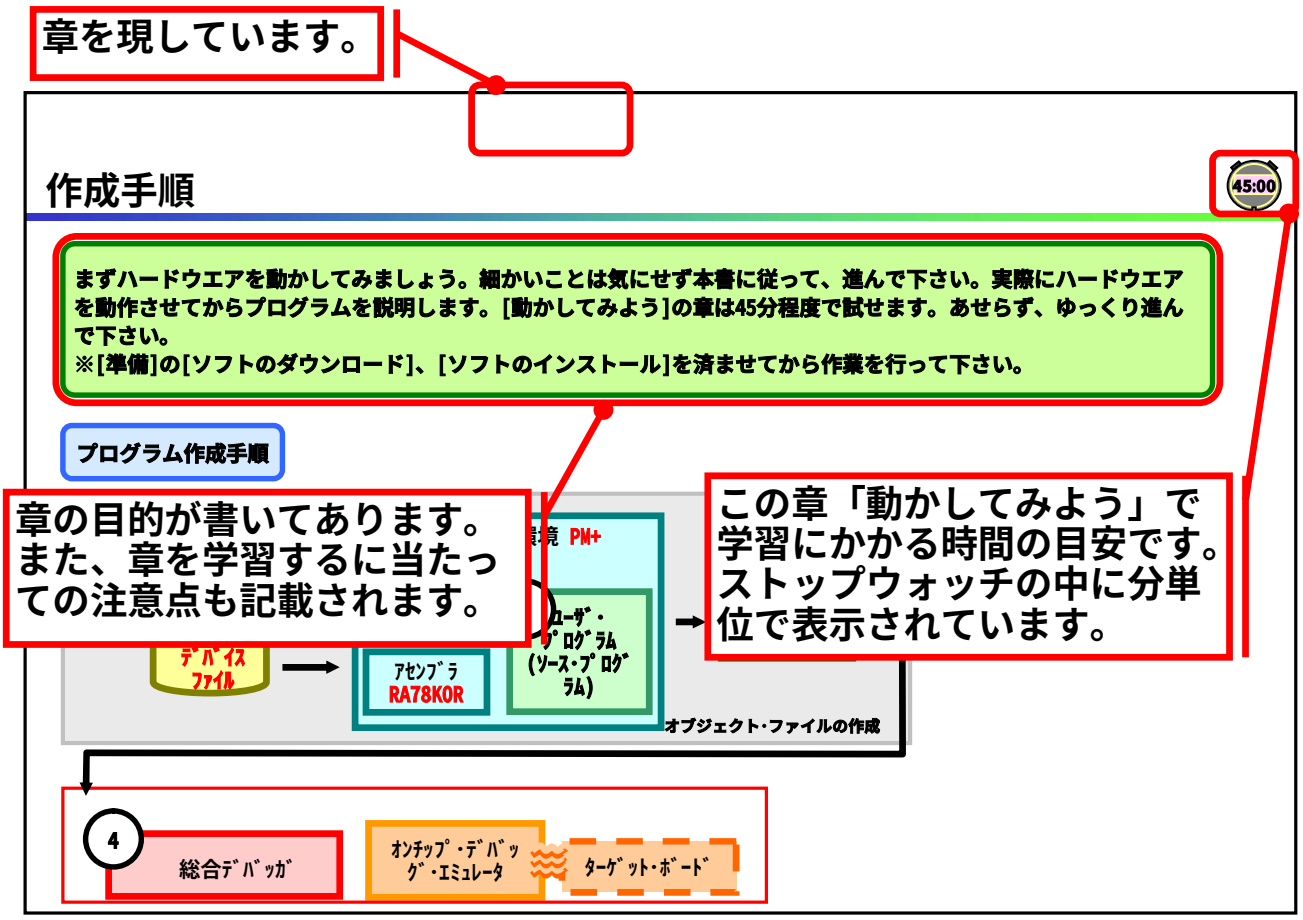

資料として使う内容です。


コラム･ワンポイントなど、関連する事についての内容です。


ハンダごてを使う作業です。

本書の見方(例)

前ページ[本書の見方(説明)]に沿った例を示します。



ワンポイント

ワンポイントについて

そのページに関係する追加情報を載せています。また、深い内容についてはヒントになる語句について記しています。このワンポイントにも注目して下さい。

はじめに

QB-78K0RKG3-TBは、LEDとSWが搭載されています。プログラムを作成しても出力はLED表示のみです。マイコン開発の初級までは、すぐに理解できても中級者、上級者になるための周辺機能/装置への理解は深められません。

本書「チュートリアル･ガイド実践編」では、マイコンへ接続する機器を具体例としてあげ、それらを組み合わせてシステムとして動作するまでの開発手順を示します。これらの実践的なガイドは、次への理解の手助けになるでしょう。本書は全てを読む必要がありません。「周辺機能を理解する」章では、周辺機能例を１つ１つあげており、それらは独立した読み物になっています。そのため必要な周辺機能だけを読んでも、それらを習得できます。では、本書の構成を説明します。

はじめに

[ごあいさつ]
[本書の見方]
[はじめに]
[もくじ]

本書を読むのに必要なガイドを記載しています。

製作するシステム

[LEDを光らせるだけじゃ物足りない!!]
[システム構成図]
[使用部品一覧]

製作するターゲットの概要と必要な部品を説明しています。

製作手順

[準備]
[A/DでSW入力]
[温度を計測]
[RTCで時計]
[LCDへ表示]
[外部へデータ保存]
[RS232Cで送信]

それぞれ独立した章になっています。周辺機能を理解するのに役立ちます。

全体回路図

上記の章「製作手順」を読まなくても、このページの回路図でハードウエアを製作できます。

ソフトウエアの作成

[組み込みプログラムとは]
[基本的なプログラム構成]
[本システムの操作]
[Appliletの設定]
[全ソースファイル説明]
[全体フローチャート]
[プログラム]

システム全体を動作させるプログラムの全てとその説明です。

次へのステップ

[開発前に考えること]
[本書でのコーディング記述]
[更に進んだ設計]
[省電力処理]
[最後に]

プログラムの設計～開発におけるポイントを説明しています。

付録

もくじ

はじめに
ごあいさつ - - - - - - - - - - - - - - - - - - 2
本書の見方(説明) - - - - - - - - - - - - - - - 3
本書の見方(例) - - - - - - - - - - - - - - - - 4
はじめに - - - - - - - - - - - - - - - - - - - 5
もくじ - 6

製作するシステム
製作するシステム - - - - - - - - - - - - - - - 7
LEDを光らせるだけじゃ物足りない!! - - - - - - 8
システム構成図 - - - - - - - - - - - - - - - - 9
開発環境の構成 - - - - - - - - - - - - - - - - 10
使用部品一覧 - - - - - - - - - - - - - - - - - 11

製作手順
製作手順 - - - - - - - - - - - - - - - - - - - 12
準備 - 13
A/DでSW入力 - - - - - - - - - - - - - - - - - 16
温度を計測 - - - - - - - - - - - - - - - - - - 24
RTCで時計 - - - - - - - - - - - - - - - - - - 31
LCDへ表示 - - - - - - - - - - - - - - - - - - 37
外部へデータ保存 - - - - - - - - - - - - - - - 52
RS232Cで送信 - - - - - - - - - - - - - - - - - 67

全体回路図
全体回路図 - - - - - - - - - - - - - - - - - - 74

ソフトウエアの作成
ソフトウエアの作成 - - - - - - - - - - - - - - 75
組み込みプログラムとは - - - - - - - - - - - - 76
基本的なプログラム構成 - - - - - - - - - - - - 78
本システムの操作 - - - - - - - - - - - - - - - 79
Appliletの設定 - - - - - - - - - - - - - - - - 81
全ソースファイル説明 - - - - - - - - - - - - - 86
全体のフローチャート - - - - - - - - - - - - - 87
プログラム - - - - - - - - - - - - - - - - - - 91

次へのステップ
次へのステップ - - - - - - - - - - - - - - - - 115
開発前に考えること - - - - - - - - - - - - - - 116
本書でのコーディング記述 - - - - - - - - - - - 119
更に進んだ設計 - - - - - - - - - - - - - - - - 121
省電力処理 - - - - - - - - - - - - - - - - - - 122
最後に - 123

付録
付録 - 124

製作するシステム

「チュートリアル･ガイド」では、78K0R開発環境の学習に重点を置き、マイコン周辺機能説明は深くしてませんでした。「チュートリアル･ガイド実践編」では**実践的なシステム開発**を例にあげています。ここでは、どのようなシステムの開発を行うかを説明します。

はじめに

製作するシステム

[LEDを光らせるだけじゃ物足りない!!]
[システム構成図]
[開発環境の構成]
[使用部品一覧]

製作手順

全体回路図

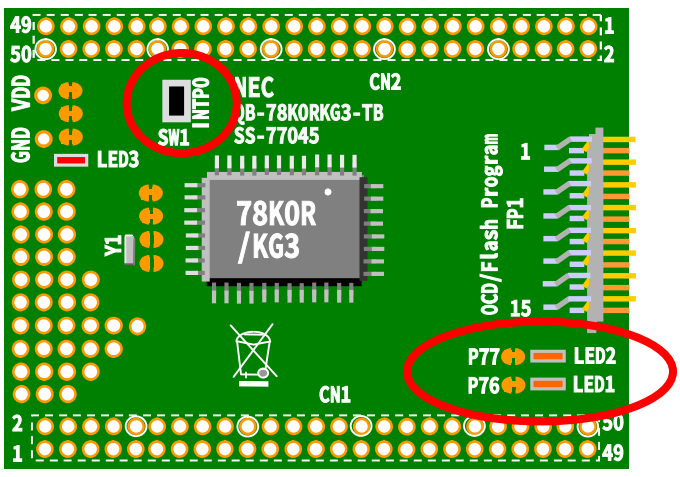
ソフトウエアの作成

次へのステップ

よくある質問

付録

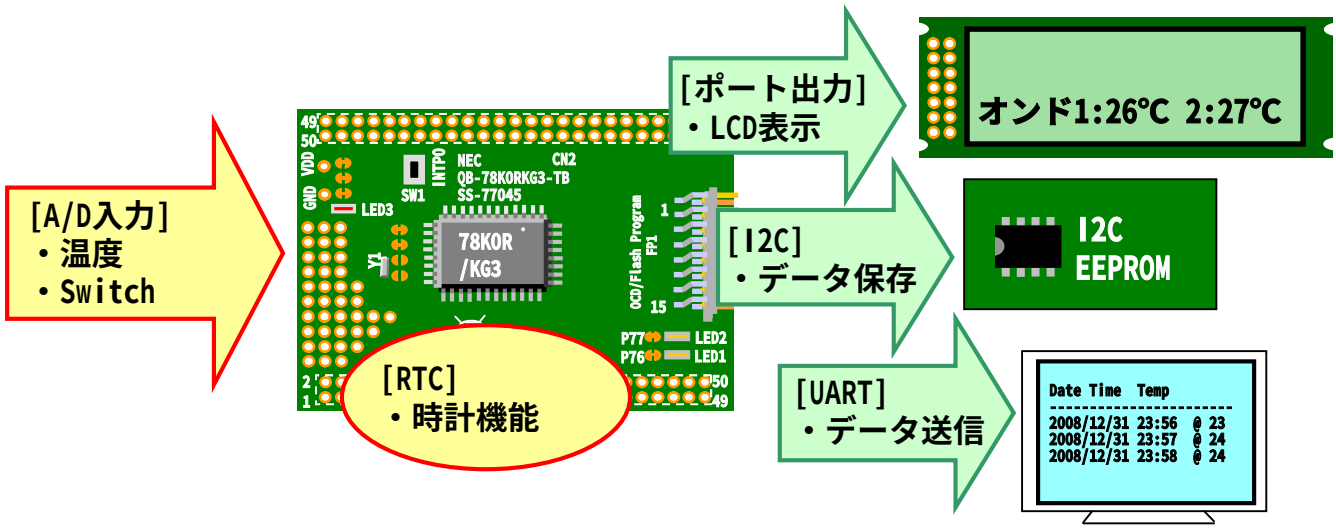
QB-78K0RKG3-TB



SWとLED点滅から
システム開発へ
ステップアップ!!

LEDを光らせるだけじゃ物足りない!!

下記の事を意識してシステムを考えました。
・入力、データ保存、データ表示の基本機能(入力、出力、保存)を備えていること
・実用的なシステムにすること
・あまり複雑な回路にならないこと
・一般的な部品で構成すること
そして考えたのが**「時計表示機能つき温度データロガー」**です。機能を簡単に説明するとこのシステムは温度センサを2つ持っています。温度データはリアルタイムに計測しています。計測した温度データは数分毎にEEPROMへ蓄積します。蓄積された温度データはRS232CでPCへ送信できるようになっています。

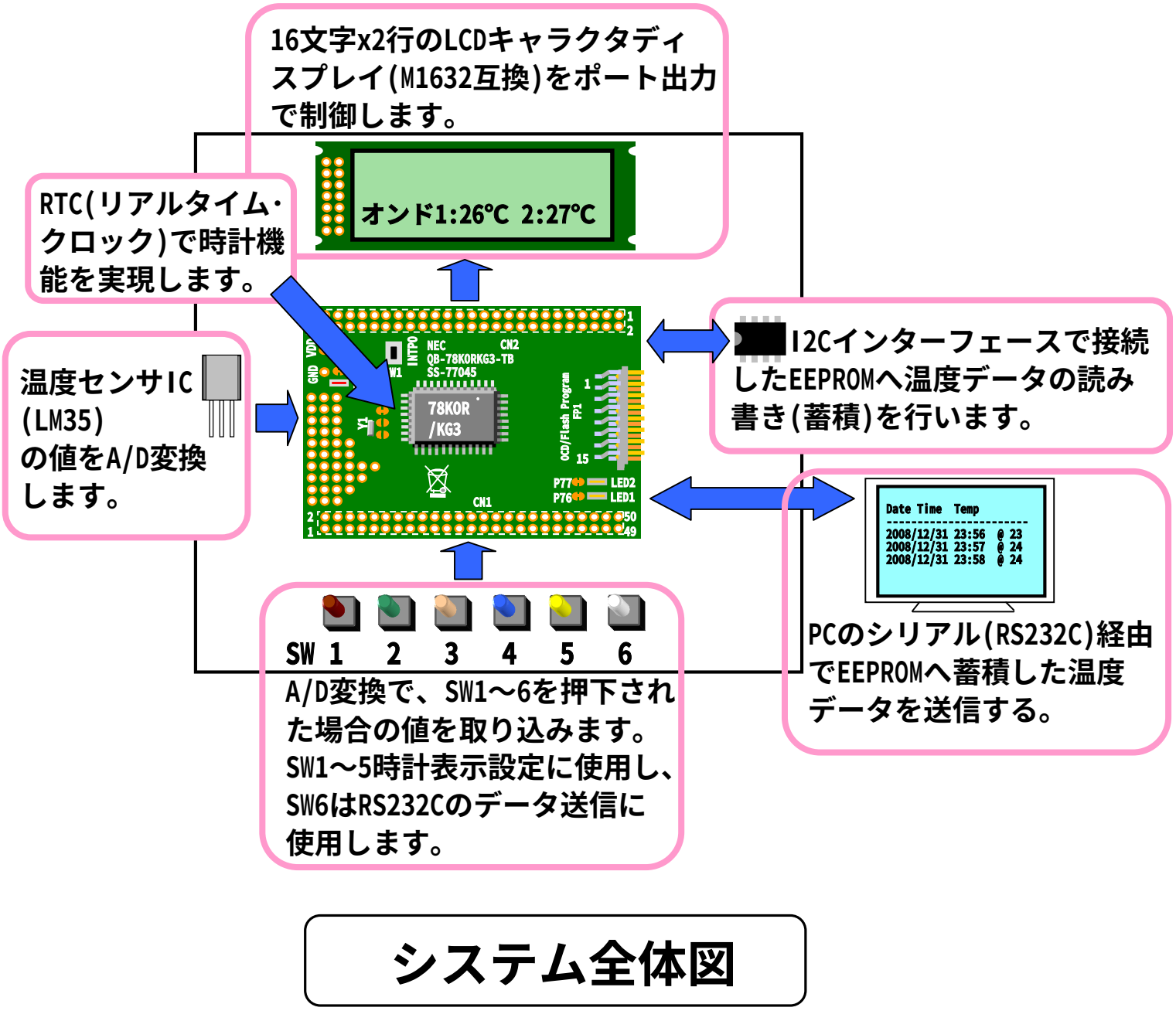


このシステムでは次の周辺機能が学習できます。

- A/Dコンバータを使った温度データの取り込み
- A/Dコンバータを使ったSwitch入力
- RTC(リアルタイム･カウンタ)を使った時計機能
- ポートを使ったLCDキャラクタディスプレイの表示制御
- I2Cを使ったEEPROMへのデータアクセス
- UARTを使ったRS232Cデータ送信

システム構成図

「時計表示機能つき温度データロガー」は下記の機能を持っています
・時計表示機能があります(時刻設定の必要あり)
・2箇所の温度を計測します
・5分毎に計測した温度データをEEPROMへ書き込みます
・SW6を押下するとEEPROM内の温度データをPCへ送信します
下図にシステム全景を示します。

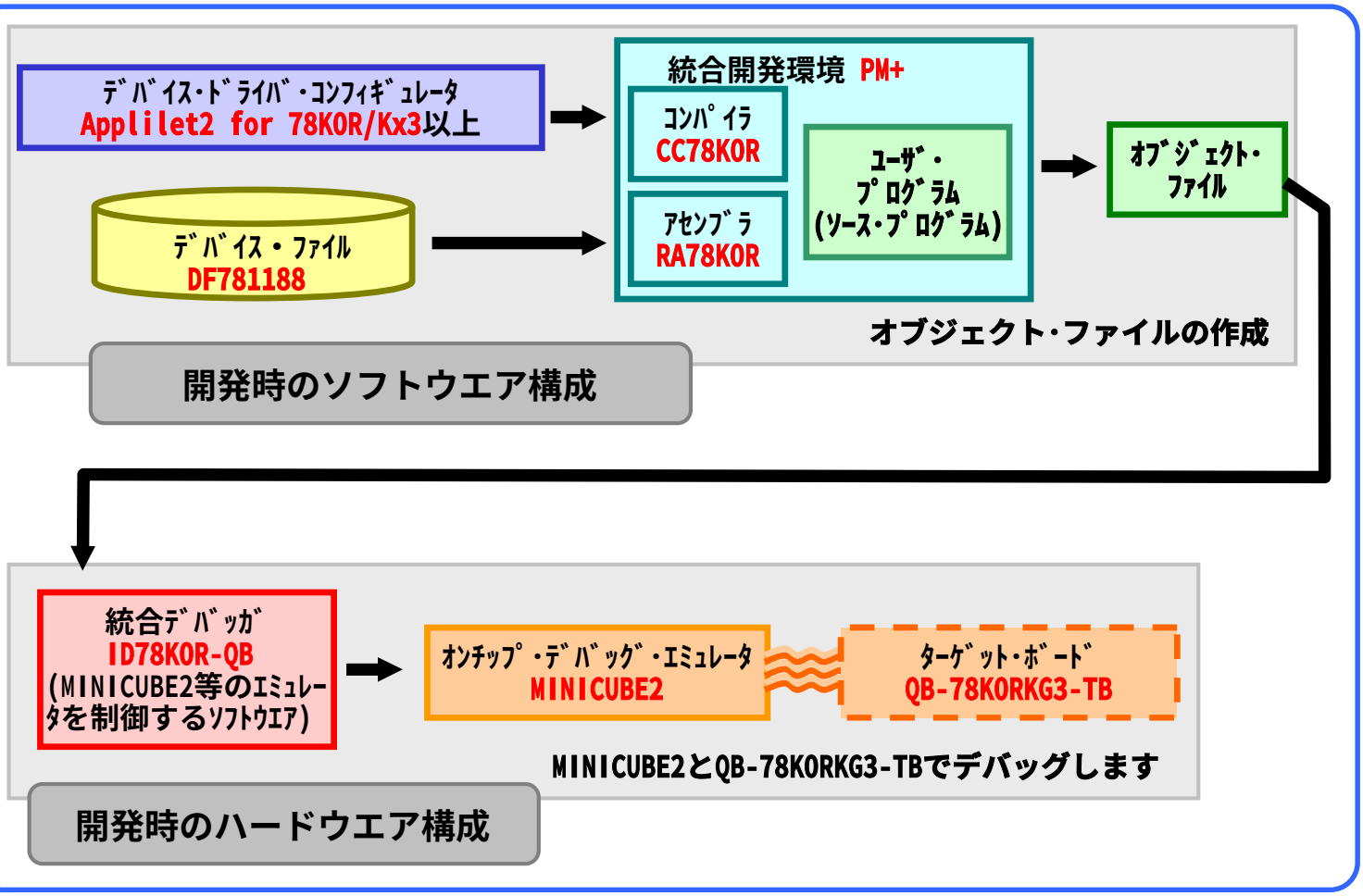


システム全体図

開発環境の構成

前項でシステム構成を説明しましたので、ここではプログラム開発時におけるソフトウエア・ハードウエアの構成について説明します。

プログラム開発～デバッグまでの環境



使用するソフトウエアのバージョンを確認してください。
RA78K0R(W1.10以上)またはRA78K0R(V1.10以上)
CC78K0R(W1.20以上)またはCC78K0R(V1.20以上)
Applilet2 for 78K0R/Kx3 V2.41以上
PM+ V6.31以上

78K0Rのフリーツールは下記URLからダウンロードしてください。
http://www.necel.com/micro/ja/freesoft/78k0r/kx3/

使用部品一覧

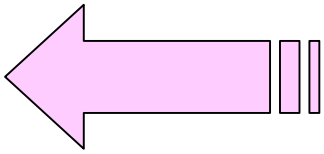
「時計表示機能つき温度データロガー」に必要な部品一覧です。段階を追って章を進めるように章毎に必要な部品をわけて表にしています。なるべく一般的な部品を使用しています。また、表に挙げた部品を必ず使う必要はありません、互換品を使っても大丈夫です。

説明する章	部品の名称	型番	個数	備考
準備	ユニバーサル基板		1	両面スルホール
準備	2x25ピンヘッダ 2x25ピンフレーム		各2	ピンフレームはTBへ搭載
準備	ICソケット	8P 16P	2 1	
準備	スペーサー、ネジ		各4	適宜使用
A/DでSW入力	プッシュスイッチ (タクトスイッチ)		6	
A/DでSW入力	抵抗	1.2KΩ 1.8KΩ 2.7KΩ 3.9KΩ 6.8KΩ 10KΩ	各1	
温度を計測	IC温度センサ	LM35	2	
温度を計測	オペアンプ	LM358N	1	互換品でも可
温度を計測	コンデンサ	0.1uF	1	セラミックで可
温度を計測	抵抗	1KΩ 2.9KΩ	各2	
RTCで時計	水晶振動子	32.768kHz	1	サブクロック用
RTCで時計	コンデンサ	12～15PF	2	サブクロック用
LCDへ表示	16x2キャラクタLCD ディスプレイ	SC1602B	1	M1632ピンコンパチブル
LCDへ表示	2x5ピンヘッダ 2x5ピンフレーム		各1	LCD固定に使用、大きいものをカットして使う。
LCDへ表示	半固定抵抗	10KΩ	1	LCDコントラスト調整用
外部へデータ保存	I2C EEPROM	24C256	1	互換品でも可
外部へデータ保存	抵抗	2KΩ	2	
RS232Cで送信	ADM3202		1	互換品でも可
RS232Cで送信	コンデンサ	0.1uF	5	セラミックで可
RS232Cで送信	DSUB9ピンコネクタ		1	オス
RS232Cで送信	シリアルケーブル		1	9ピンクロス

製作手順

この項では「時計表示機能つき温度データロガー」で扱う周辺機能について1機能毎に説明します。機能毎に学習できるように説明を独立させました。例えば「A/DでSW入力」では、SW入力の確認だけを行えるプロジェクトを作成しています。段階的に学習していき、この章を終える時に「時計表示機能つき温度データロガー」のハードウエア部分が完成しています。
また、学習の必要がない場合は「全体回路図」の章でハードウエアを全て製作してください。

はじめに
製作するシステム
製作手順
[準備]
[A/DでSW入力]
[温度を計測]
[RTCで時計]
[LCDへ表示]
[外部へデータ保存]
[RS232Cで送信]
全体回路図
ソフトウエアの作成
次へのステップ
よくある質問
付録



「製作手順」の章では周辺機能の一つ一つを説明して、ソフトウエアも独立して動作確認していますが、それらを省きたい場合は表の組み立て順(下図を参照)の番号に従ってハードウエアを先に完成させてください。組み立て順は1～11までであります。またタイトルの横に「  」アイコンがありますので組み立ての目印にして、ハードウエア作成してください。その後、「ソフトウエアの作成」章へ進んでソフトウエアを作成してください。

組み立て順	部品の名称	型番	個数	備考
1				
～				
11				

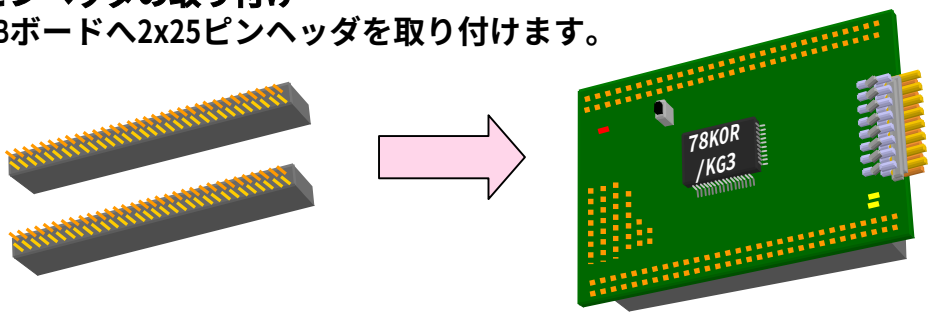
ハードウエアの作成を行う順序を示している表

準備(1)

ハードウエア製作の準備をします。QB-78K0RKG3-TBのベースとなる部分を製作します(以下ベースボードと呼びます)。このベースボードに後からスイッチやIC、LCDキャラクタディスプレイなどを載せます。

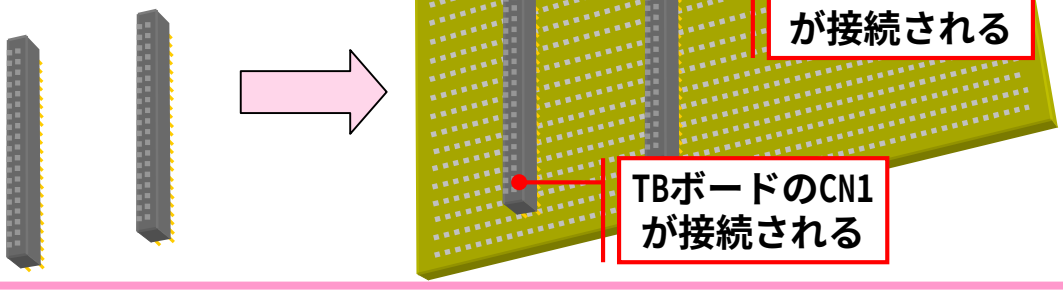
1.ピンヘッダの取り付け

TBボードへ2x25ピンヘッダを取り付けます。



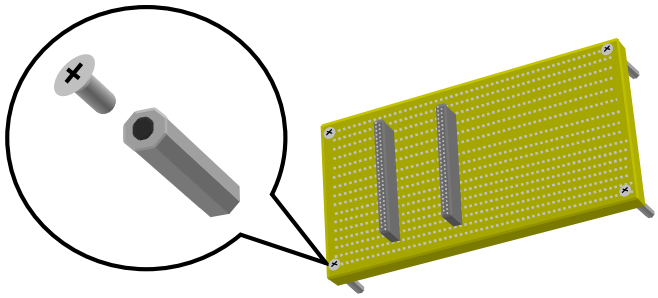
2.ピンフレームの取り付け

ユニバーサル基板へ2x25ピンフレームを取り付けます。
TBボードが搭載できるように配置してください。
中央よりやや端に取り付けると後の部品が載せやすくなります。TBボードを搭載する時のCN1、CN2の接続先も決めます。



3.スペーサーの取り付け

必ずしも必要はないのですが、ベースボードを安定させるためスペーサーを取り付けます。

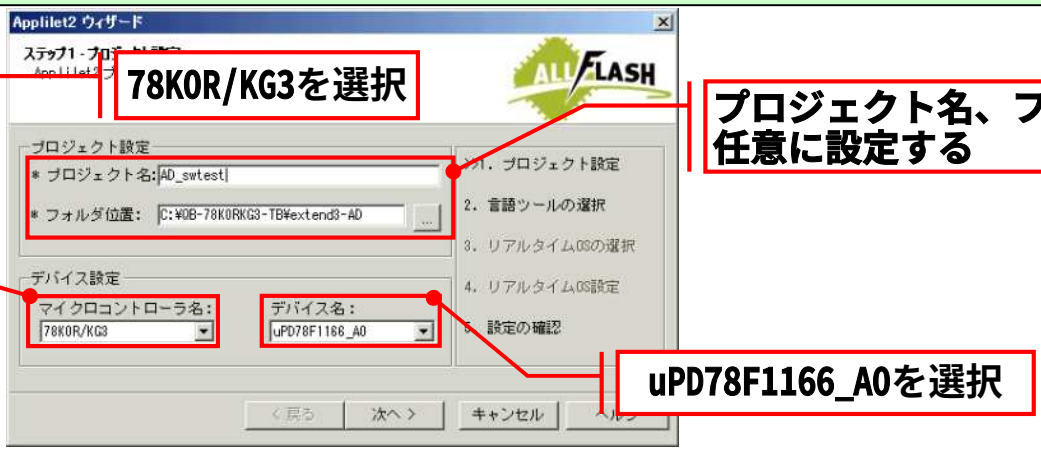


組み立て順	部品の名称	型番	個数	備考
1	2x25ピンヘッダ		各2	ピンヘッダをTBへ搭載
2	2x25ピンフレーム		各2	
3	スペーサー、ネジ		各4	適宜使用

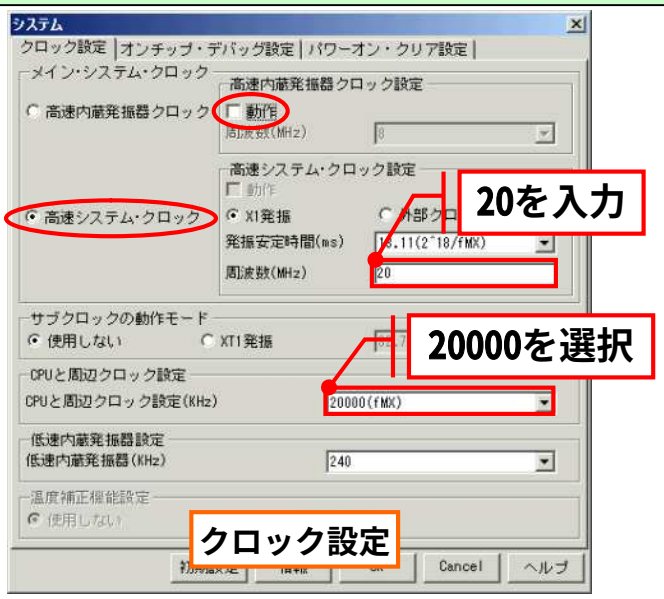
準備(2)

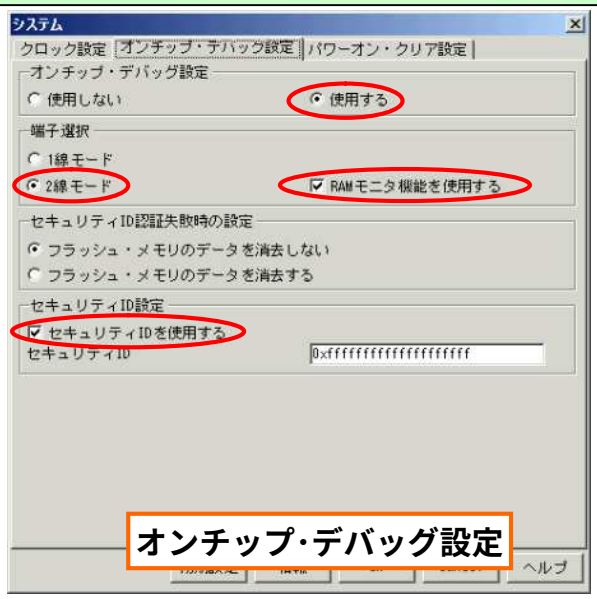
ソフトウエアの準備をします。QB-78K0RKG3-TBに必要な基本的な設定（メイン･クロック、オンチップデバッガ）は、全ての周辺機能で共通ですのでここで説明します。Appliletの初期設定はここを参照してください。この後の章で各周辺機能のApplilet設定が続きます。

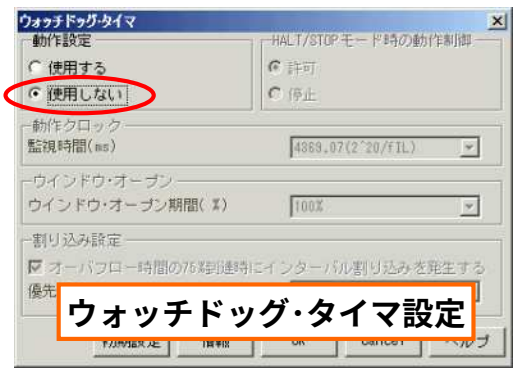
1. Applilet2を起動し、デバイス、プロジェクト名を入力します。
   QB-78K0RKG3-TBのマイクロコントローラ名「78K0R/KG3」、デバイス名「uPD78F1166_A0」を選択します。プロジェクト名は、それぞれ任意につけます。



2. 「システム」でクロック、オンチップ･デバッグ、ウォッチドッグ･タイマを設定します。
   下図に変更点を示します。







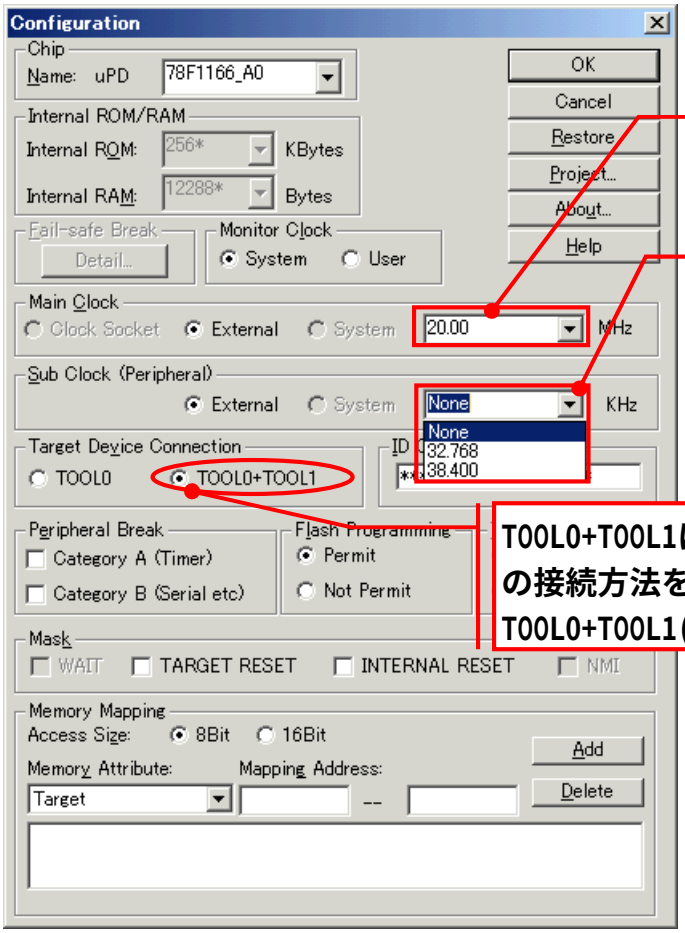
下記の設定を行います。
- CPUの動作は20MHz
- 周辺に供給するクロックも同じ20MHz
- 外付けクロックを使用
- ウォッチドッグ･タイマを使わない
- オンチップ･デバッグは2線式
  RRM(RAMモニタ)機能を使用
- セキュリティIDを使用

準備(3)

QB-78K0RKG3-TBで、デバッグを行う際の基本的なデバッガの使い方をここで説明します。

1.デバッガの起動

PM+のデバッグ・アイコンを押下するか、もしくはPM+よりビルド->デバッグを選択しID78K0R-QB]を起動します。下図のように[Configuration]を設定してください。



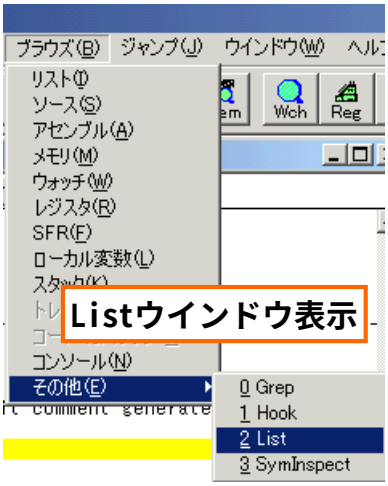
QB-78K0RKG3-TBではシステム・クロックに20MHzが搭載されているので、20を選択します。

サブクロックを設定します。「A/DでSW入力」では使わないので[None]でもよいのですが、後の章「RTC」で時計ではサブクロックを使用します。その際は「32.768」を選択してください。

TOOL0+TOOL1にチェックします。これはターゲットとデバッガの接続方法を設定します。QB-78K0RKG3-TBとMINICUBE2はTOOL0+TOOL1(2線式)で接続されています。

2.デバッガの基本画面

前編「チュートリアルガイド」の「デバッグの基本」章を参考にしてください。下図のウインドウは表示しておくとデバッグ時に便利です。



Listウインドウ表示

RRM表示可能の設定

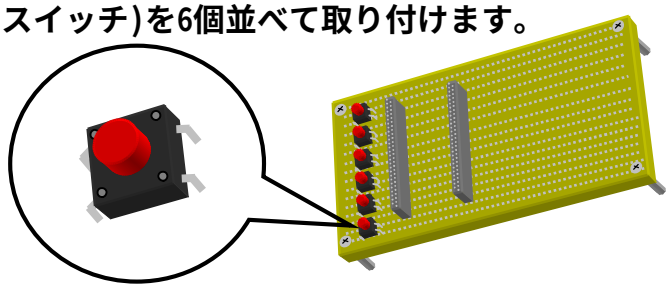
A/DでSW入力

SW入力には下記のような方法で実現可能です
・外部割り込み(INTPx)を使用
・キー割り込み(KRx)を使用
・ポートを使用(キーマトリクス入力)
・A/D入力を使用
それぞれ特徴がありますが、ここではA/D入力を使ってSW入力を実現します。
下記の表は、スイッチ入力を行う時の主な方法の特徴です。

SW入力方法	応答速度	必要端子数	入力可能な数	プログラム難易度	特徴(用途)
外部割り込み	速い	多い	必要端子に同じ	簡単	応答速度が速い
キー割り込み	速い	多い	必要端子に同じ	簡単	応答速度が速い
ポート入力	中	少ない	端子x端子	中	キーボード入力など
A/D入力	遅い	少ない	1～6	難しい	数個の入力に最適

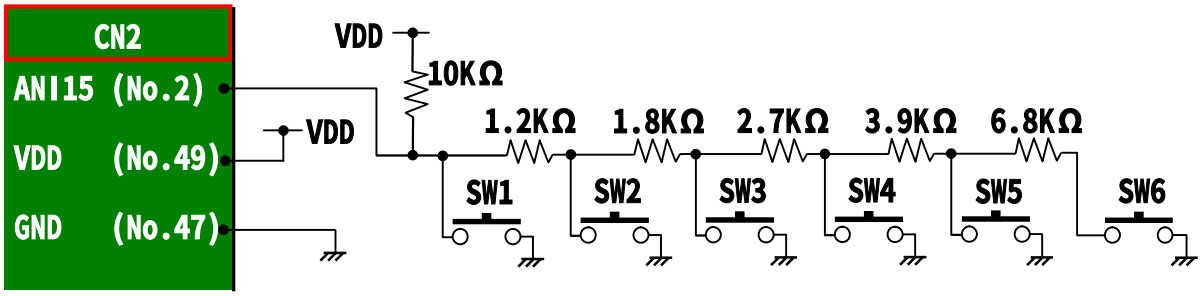
4.プッシュSWの取り付け

プッシュSW(タクトスイッチ)を6個並べて取り付けます。



5.抵抗の取り付け

回路図のように抵抗をつけます。またCN2-15pin(ANI15)へ配線します。SW1～SW6をそれぞれ押下すると、それに応じた抵抗値がANI15へ入力されます。A/D入力を1つのみ使用するので、同時に認識可能なSWは1つです。複数押下された場合は小さい抵抗に接続されているSWが優先されます。



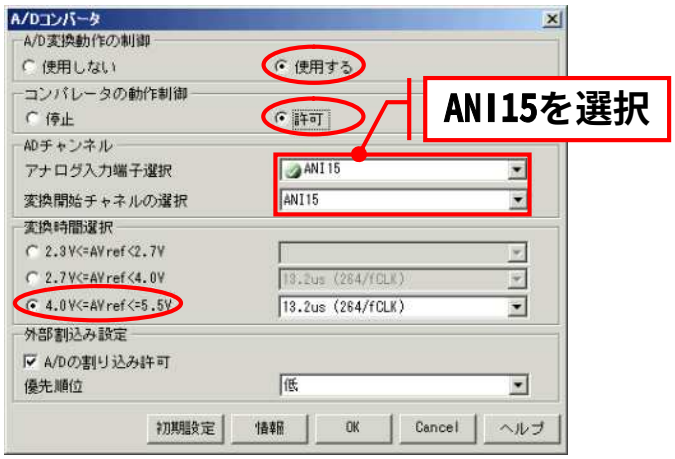
組み立て順	部品の名称	型番	個数	備考
4	プッシュスイッチ		6	タクトスイッチ
5	抵抗	1.2KΩ 1.8KΩ 2.7KΩ 3.9KΩ 6.8KΩ 10KΩ	各1	

A/DでSW入力

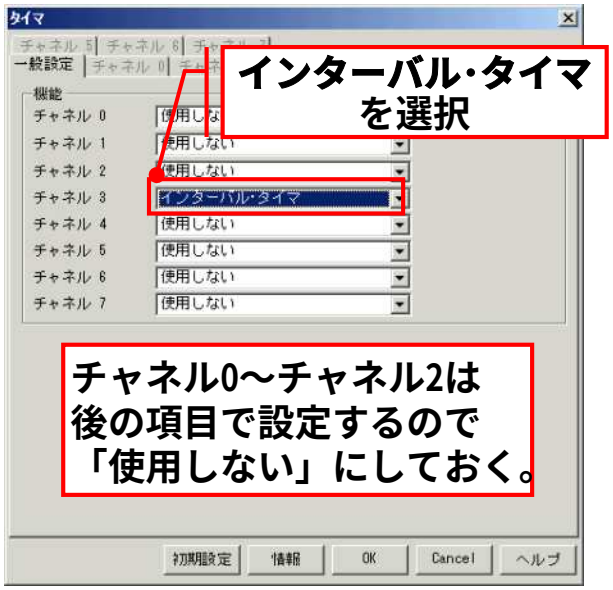
30:00

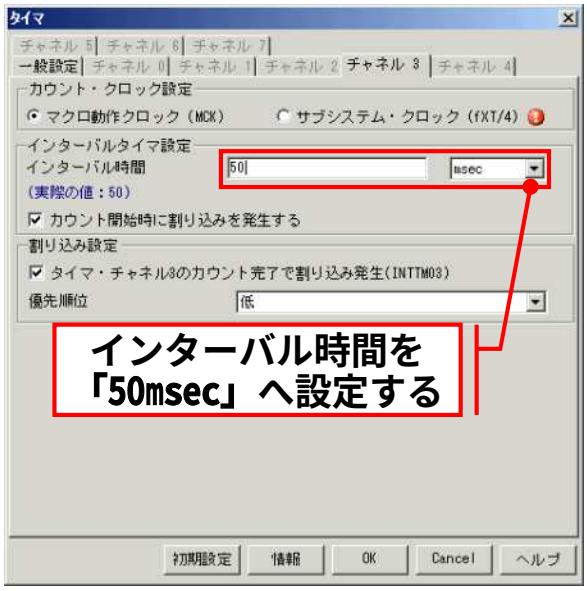
ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。A/DによるSW入力のみをテストするためソフトウエアです。Applilet2による設定から説明します。

a.「A/Dコンバータ」を設定します。
ANI15を使用します。



b.「タイマ」でチャネル3をインターバル･タイマへ設定します。
チャネル3を50msecのインターバル･タイマへ設定します。





資料

A/Dを用いたSW入力について
78K0マイコンサンプルプログラムも参照して下さい。(下記URLの「A/D入力キー・スキャン」)
http://www.necel.com/micro/ja/designsupports/sampleprogram/78k0/
上記の例では、6x8個のキー･スキャン入力を行っていますが、ここでは回路を参考にしてANI15だけを使い6個のSW入力を実現しています。

A/DでSW入力

c.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
TAU.c / h	タイマ処理。
TAU_user.c	ユーザー処理用タイマ割り込みハンドラ定義処理。
Ad.c / h	A/D処理。
Ad_user.c	A/D処理用A/D変換完了割り込みハンドラ定義処理。

上記のソースファイルで赤枠で示したのが「A/DでSW入力」で編集対象のファイルです。次ページより編集するソースを説明します。

A/DでSW入力

d.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

main.c

```
            リスト省略
/*
***********************************************************************************
**  Global define
***********************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
UINT    g50msecCounter;                 /* Interval timer 50msec(ch3使用)        */
USHORT  gPushButtonAd;                  /* ADチャネル15に接続したSWボタンの直値  */
UCHAR   gPushButtonNo;                  /* 押下された SW番号                     */
/* End user code for global definition. Do not edit comment generated here */

void  main( void )
{
    /* Start user code. Do not edit comment generated here */

    g50msecCounter = 0;
    TAU_Channel3_Start();

    while (1) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

インターバルタイマを開始します

TAU_user.c

```
            リスト省略
#include "macrodriver.h"
#include "TAU.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "Ad.h"
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
/*
***********************************************************************************
**  Global define
***********************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
extern UINT    g50msecCounter;
/* End user code for global definition. Do not edit comment generated here */

            リスト省略
__interrupt void MD_INTTM03(void)
{
    /* Start user code. Do not edit comment generated here */

    g50msecCounter++;
    AD_Start();

    /* End user code. Do not edit comment generated here */
}
```

A/D変換を開始します

A/DでSW入力

Ad_user.c

```
            リスト省略
/*
***********************************************************************************
**  Global define
***********************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
/* 押しボタンを判定するADの値 */
#define D_SW1_HIGH          0x0030
#define D_SW2_HIGH          0x00A4
#define D_SW3_HIGH          0x012C
#define D_SW4_HIGH          0x01B0
#define D_SW5_HIGH          0x0234
#define D_SW6_HIGH          0x033B
extern USHORT  gPushButtonAd;
extern UCHAR   gPushButtonNo;
/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------------
**  Abstract:           This function is INTAD interrupt service routine.
**  Parameters:         None
**  Returns:            None
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTAD(void)
{
    /* Start user code. Do not edit comment generated here */

    AD_Stop();
    AD_Read( &gPushButtonAd );
    if ( gPushButtonAd < D_SW6_HIGH )
    {
        if ( gPushButtonAd < D_SW1_HIGH )
        {
            gPushButtonNo = 1;          }
        else if ( gPushButtonAd < D_SW2_HIGH )
        {
            gPushButtonNo = 2;          }
        else if ( gPushButtonAd < D_SW3_HIGH )
        {
            gPushButtonNo = 3;          }
        else if ( gPushButtonAd < D_SW4_HIGH )
        {
            gPushButtonNo = 4;          }
        else if ( gPushButtonAd < D_SW5_HIGH )
        {
            gPushButtonNo = 5;          }
        else
        {
            gPushButtonNo = 6;          }
    }
    else
    {
        gPushButtonNo = 0;
    }
    /* End user code. Do not edit comment generated here */
}
```

電源(AVref)に5Vを印可した場合にA/D入力値と比較する値
D_SW1_HIGHよりA/D値が小さいときはSW1押下とする
…
D_SW6_HIGHよりA/D値が小さいときはSW6押下とする
それ以外はSW押下なしとする

A/D変換を停止します

A/D変換値を読み込むます

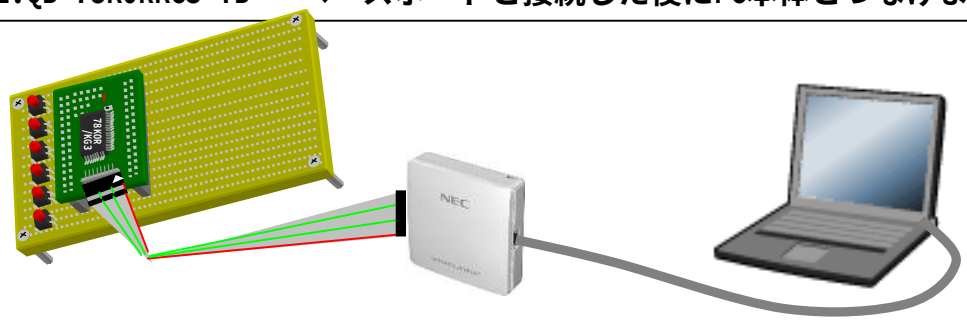
読み込んだ値からSW1～6を求めます

A/DでSW入力

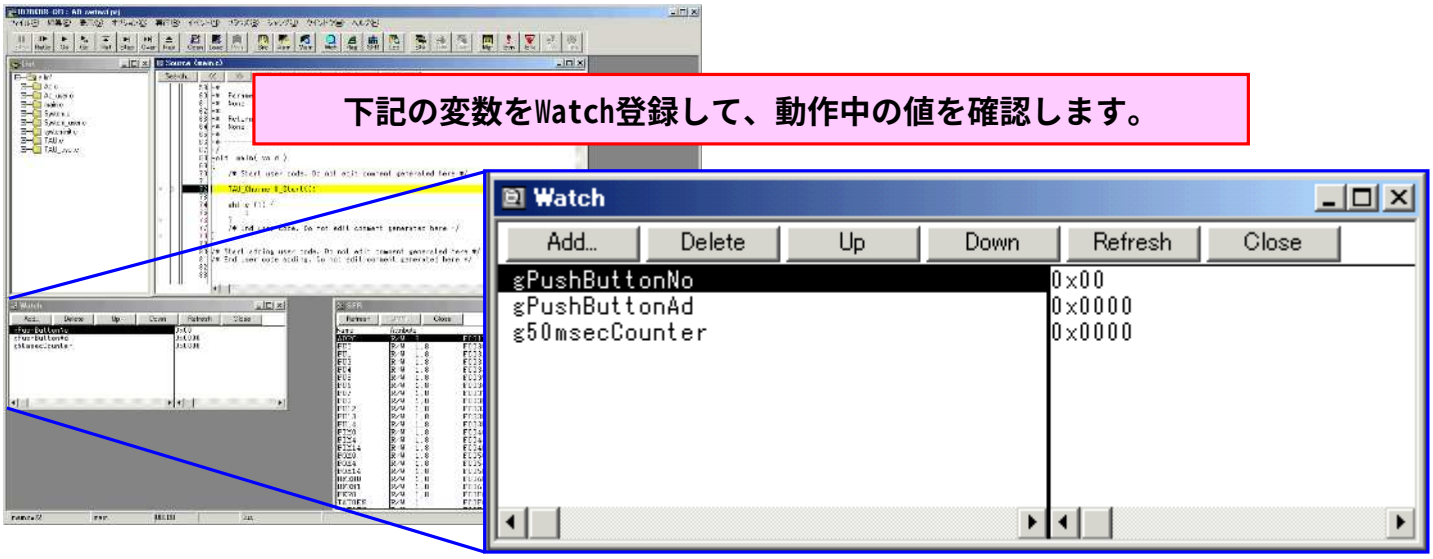
e.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

1.MINICUBE2のスイッチを設定します。

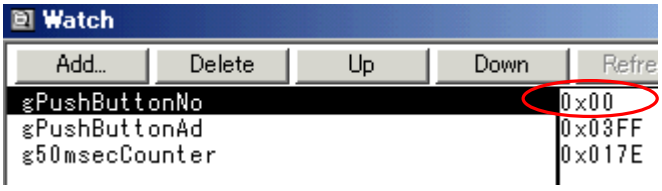
2.QB-78K0RKG3-TB + ベースボードと接続した後にPC本体とつなげます。



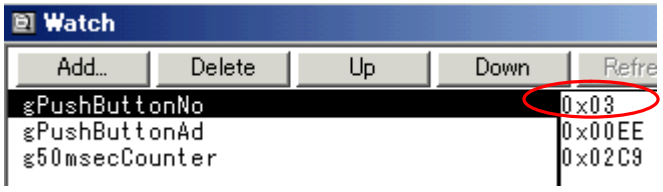
3.PM+ V6.31を起動し、プログラムをビルドしてID78K0R-QB上で実行します。
次ページにID78K0R-QB(統合デバッガ)の使い方を説明していますので参考にしてください。



何も押していないときの動作中の値
SWの値は「0」になっています。

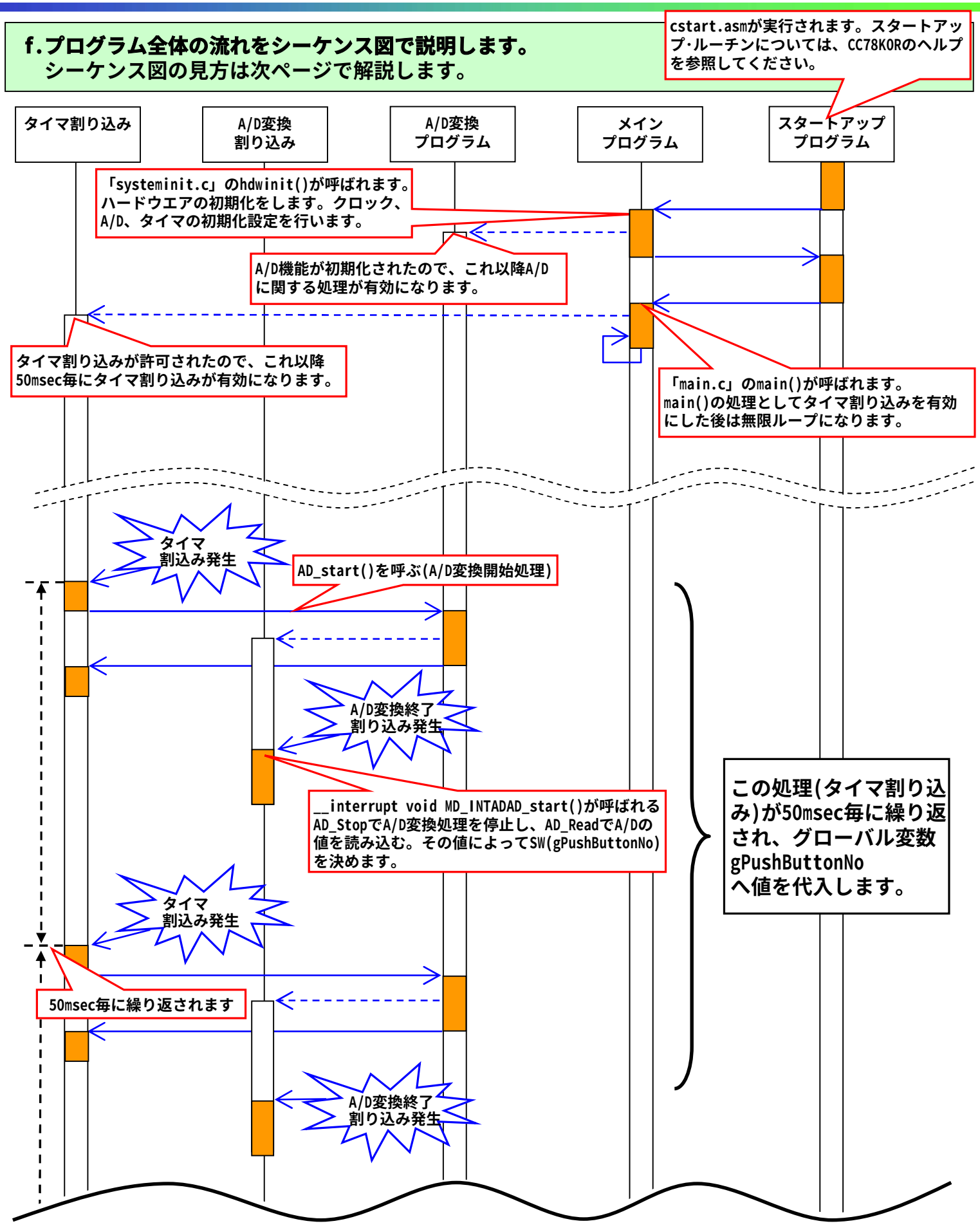


SW3を押したときの動作中の値
SWの値は「3」になっています。



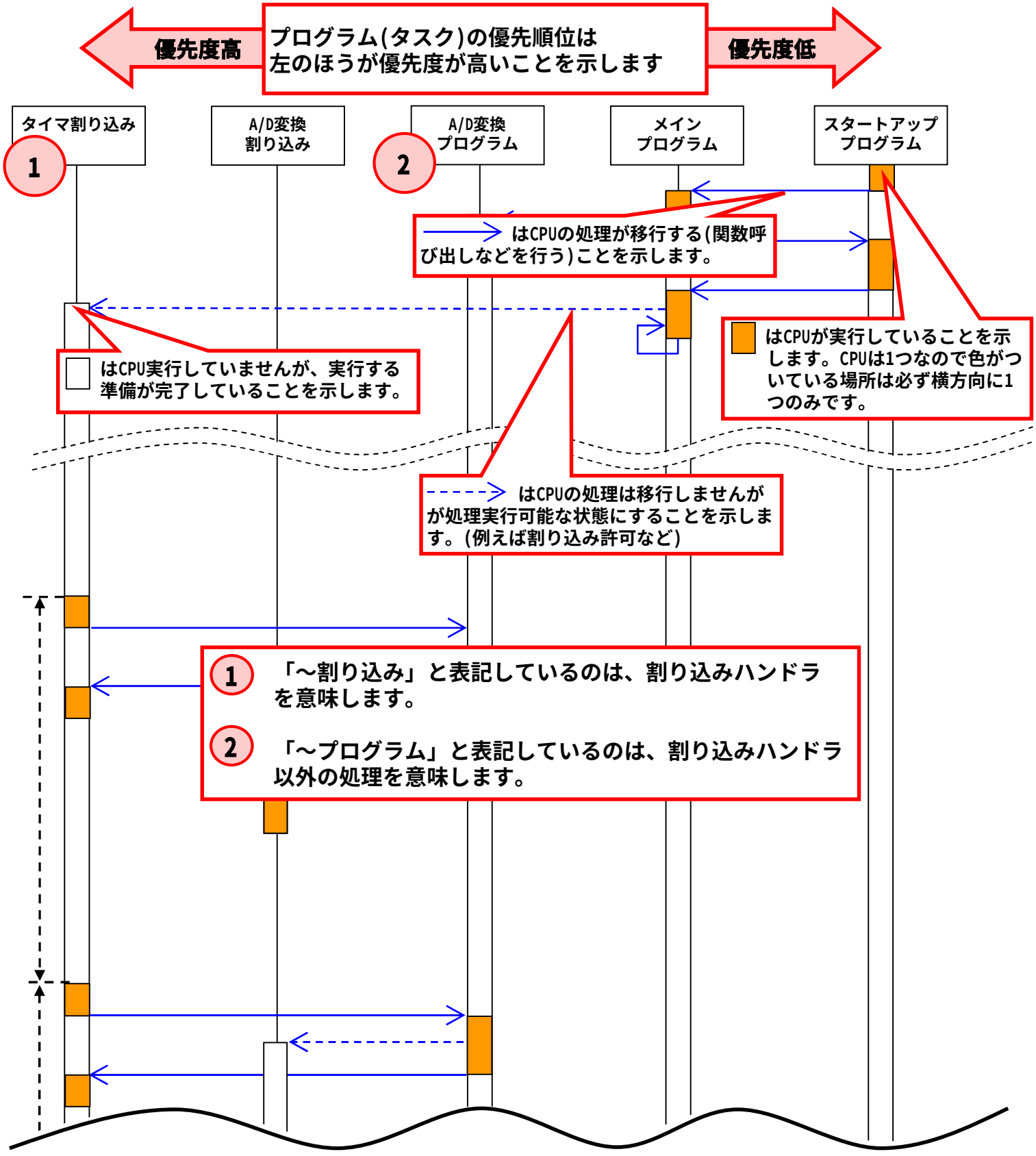
A/DでSW入力

f.プログラム全体の流れをシーケンス図で説明します。
シーケンス図の見方は次ページで解説します。



本書「シーケンス図」の見方

「A/DでSW入力」を例に本書でのシーケンス図の見方を説明します。

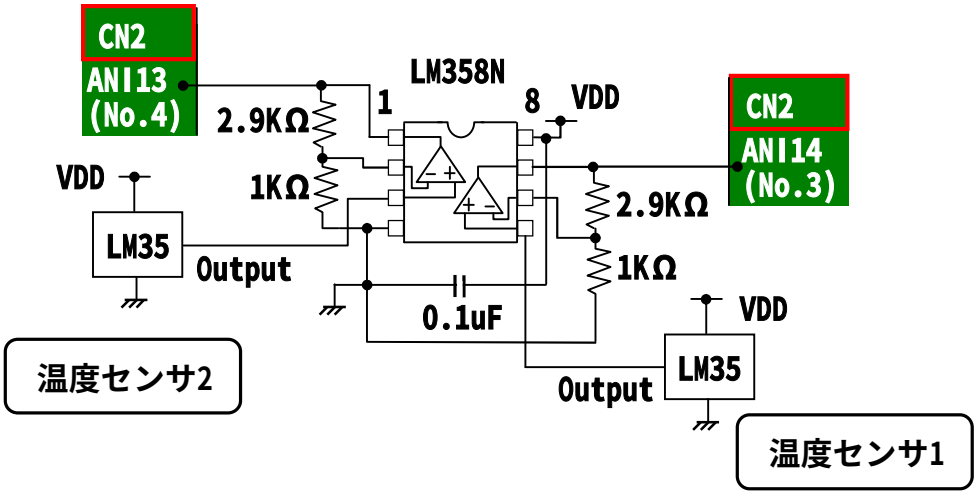


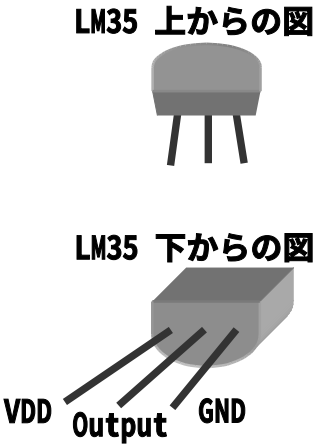
温度を計測

温度計測はA/Dに接続した温度センサのデータ値を読み込みます。
ただ、ここで使用する温度センサのデータ出力値は5V電源を扱う場合25℃で250mV(0.25V)しかありません。250mVではA/D入力しても値が小さいため誤差が大きくなります。そのためオペアンプで4倍に増幅し、25℃の場合で1000mV(1V)を出力するようにします。温度センサはマイナス温度も計測できるのですが回路を簡単にするためプラス電圧のみを計測対象とします。
理論上5V(125℃)まで計測可能です。温度センサは2つ使っていますので、2箇所同時に温度計測が可能です。

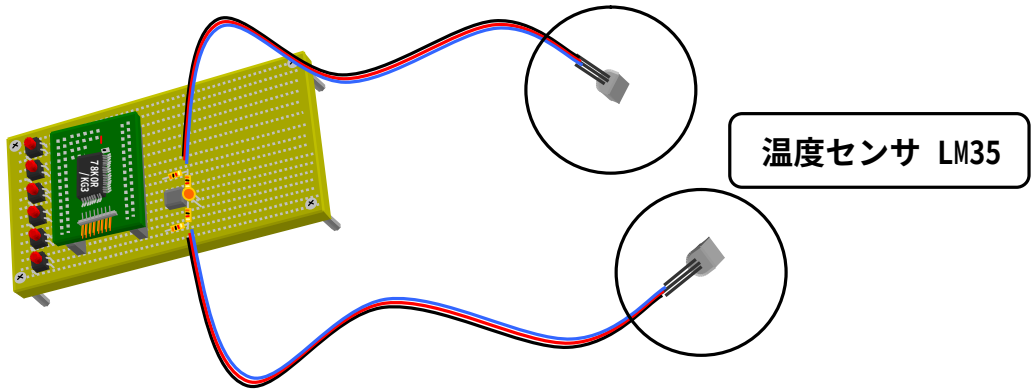
6.温度センサとオペアンプの取り付け

回路図のとおりに配線してください。





LM35は基板上に載せずにケーブルで接続すると、離れたところの温度計測が可能になります。



組み立て順	部品の名称	型番	個数	備考
6	オペアンプ	LM358N	1	互換品でも可
	抵抗	1KΩ 2.9KΩ	各2	抵抗
	コンデンサ	0.1uF	1	セラミックで可
	IC温度センサ	LM35	2	互換品でも可

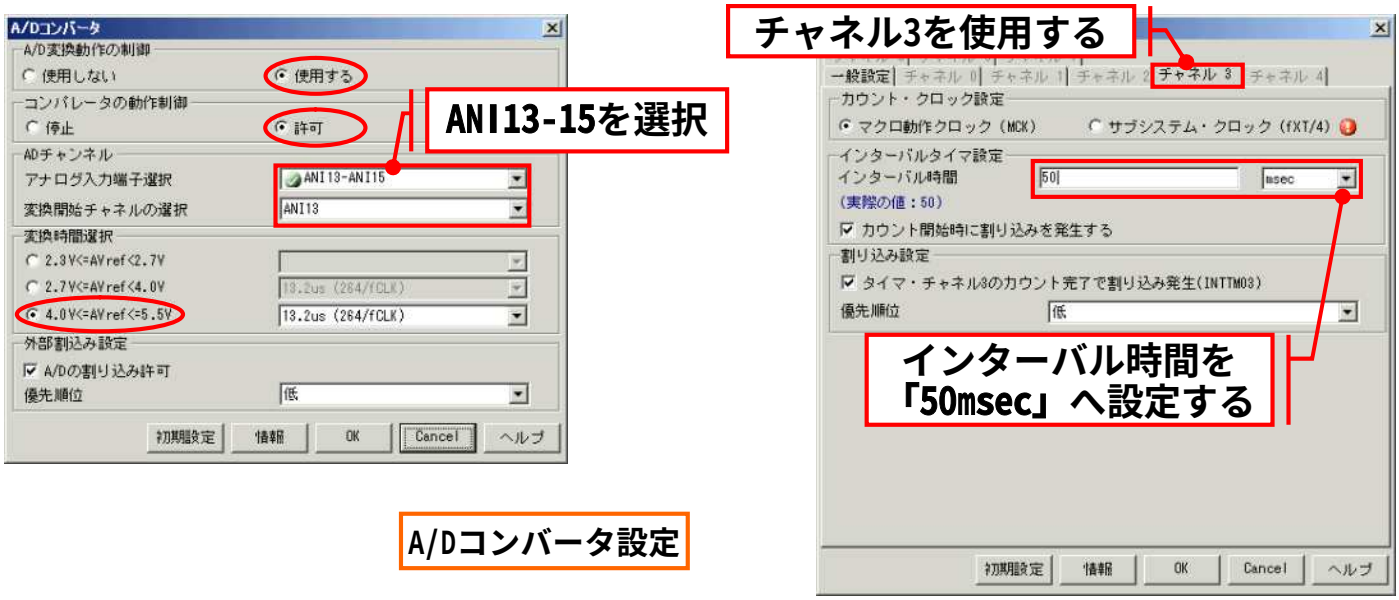
温度を計測

30:00

ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。A/Dによる温度計測のみをテストするためソフトウエアです。Applilet2による設定から説明します。

a. Applilet2を起動し、デバイス、プロジェクト名、システム設定します。
「準備(2)」を参照して、「システム」まで同じに設定します。マイクロコントローラ名「78K0R/KG3」、デバイス名「uPD78F1166_A0」、CPUクロック20MHz、オンチップ･デバッグ、ウォッチドッグ･タイマも同様に設定します。プロジェクト名は「AD_temptest」としました。

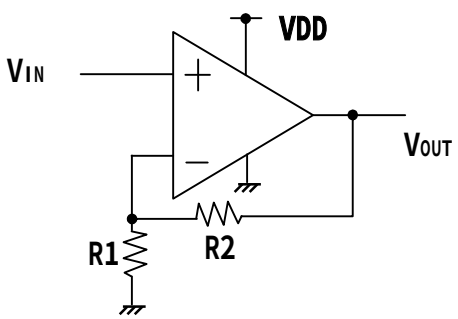
b.「A/Dコンバータ」でANI13-ANI15を設定します。
下図に変更点を示します。また、「タイマ」でチャネル3をインターバル･タイマへ設定します。チャネル3を50msecのインターバル･タイマへ設定します。「タイマ」の設定は「A/DでSW入力」と同じです。



A/Dコンバータ設定

ワンポイント

オペアンプについて
ここでは入力信号を4倍にしています。オペアンプの基本的な使い方は下記URLに説明があります。
http://www.necel.com/ja/faq/f_op.html
上記の「基礎 Q2」に、今回の使用している回路の説明が掲載されています。



今回の回路では、R1が1KΩ、R2が2.9KΩです。計算式では(1 + R2/R1)倍に増幅されるのでVIN → VOUTは約4倍になります。なぜ4倍なのか?それは10倍にしてしまうと、計測する温度が100℃の場合、温度センサは1℃に対して10mVの出力を行うため1000mV(1V)を出力することになります。これを10倍すると10Vとなり、5Vの電源供給で間に合わないので4倍に抑えています。温度100℃の場合、オペアンプの出力は1V x 4 = 4Vになります。

温度を計測

c.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
TAU.c / h	タイマ処理。
TAU_user.c	ユーザー処理用タイマ割り込みハンドラ定義処理。
Ad.c / h	A/D処理。
Ad_user.c	A/D処理用A/D変換完了割り込みハンドラ定義処理。

上記のソースファイルで赤枠で示したのが「温度を計測」で編集対象のファイルです。次ページより編集するソースを説明します。

温度を計測

d.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

main.c

```
            リスト省略
/*
***********************************************************************************
**  Global define
***********************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
UINT    g50msecCounter;                 /* Interval timer 50msec(ch3使用)          */
UCHAR   gSelectAd;                      /* ADチャネルを選択する                    */
USHORT  gTemperatureAd;                 /* ADチャネル14に接続した温度センサ1の直値 */
USHORT  gTemperatureAd2;                /* ADチャネル13に接続した温度センサ2の直値 */
/* End user code for global definition. Do not edit comment generated here */

void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    g50msecCounter = 0;
    gSelectAd = 0;
    TAU_Channel3_Start();

    while (1) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

インターバルタイマを開始します

TAU_user.c

```
            リスト省略
#include "macrodriver.h"
#include "TAU.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "Ad.h"
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
/*
**  Global define
*/
/* Start user code for global definition. Do not edit comment generated here */
extern UINT    g50msecCounter;
extern UCHAR   gSelectAd;
/* End user code for global definition. Do not edit comment generated here */

__interrupt void MD_INTTM03(void)
{
    /* Start user code. Do not edit comment generated here */
    g50msecCounter++;
    if ( gSelectAd == 0 )
    {
        AD_SelectADChannel( ADCHANNEL14 );
    }
    else
    {
        AD_SelectADChannel( ADCHANNEL13 );
    }
    AD_Start();
    /* End user code. Do not edit comment generated here */
}
```

gSelectAdの値によってA/D変換するチャネルを選択します

チャネル14をA/D変換

チャネル13をA/D変換

A/D変換を開始します

温度を計測

Ad_user.c

```
            リスト省略
*
************************************************************************************
**  Global define
************************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
extern UCHAR   gSelectAd;
extern USHORT  gTemperatureAd;
extern USHORT  gTemperatureAd2;
/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------------
**  Abstract:           This function is INTAD interrupt service routine.
**  Parameters:         None
**  Returns:            None
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTAD(void)
{
    /* Start user code. Do not edit comment generated here */
    USHORT adtemp;
    UCHAR  dat;

    AD_Stop();

    AD_Read( &adtemp );
    dat = (UCHAR)((adtemp * 48) / 400);

    if ( gSelectAd == 0 )
    {
        gTemperatureAd = dat;
    }
    else
    {
        gTemperatureAd2 = dat;
    }

    gSelectAd ^= 1;

    /* End user code. Do not edit comment generated here */
}
```

A/D変換を停止する

A/D変換値を読み込む

読み込んだ値から温度を求める

gSelectAdの値によってA/D変換を行ったチャネルを処理する。

次のA/D変換チャネルを決める

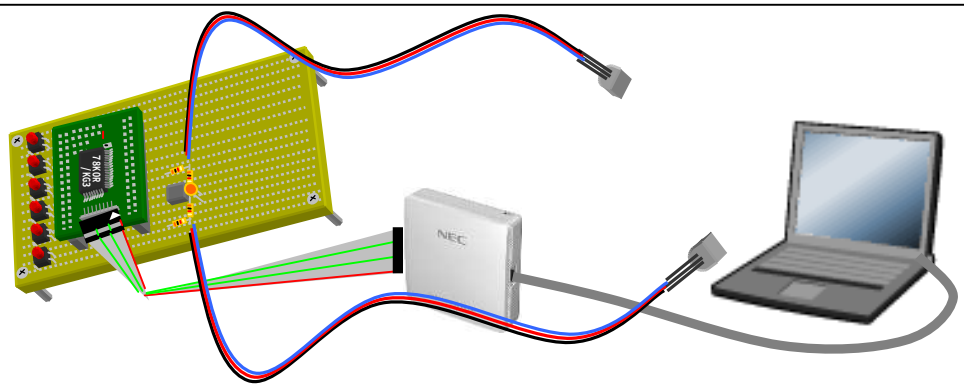
資料

78K0Rでは、最大16チャネルのA/D変換を行えますが、一度に複数のチャネルを変換できません。セレクタと呼ばれる部分で変換するチャネルを選択します。
ここでは、温度センサ1(gTemperatureAd)がANI14(A/Dチャネル14)、温度センサ2(gTemperatureAd2)がANI13(A/Dチャネル13)へ接続されています。gSelectAdが0の時にA/Dチャネル14、1の時にA/Dチャネル13をセレクタで選択しA/D変換を行っています。ですので、一つの温度センサを計測するのは100msec毎に処理されます。

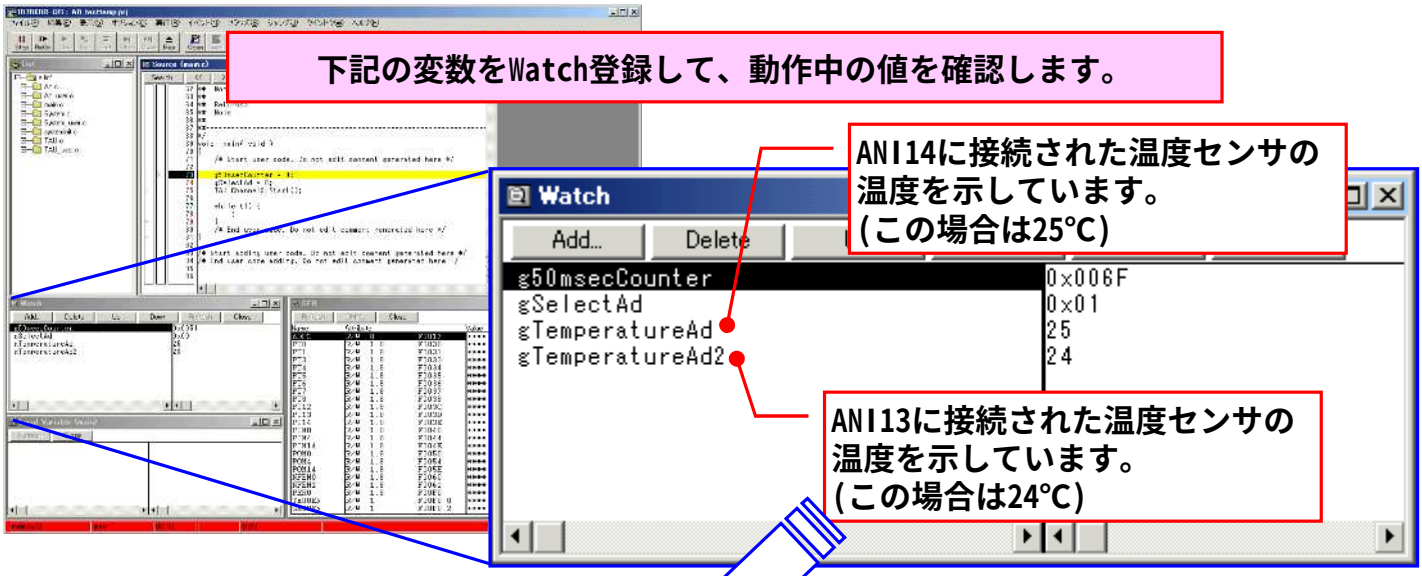
温度を計測

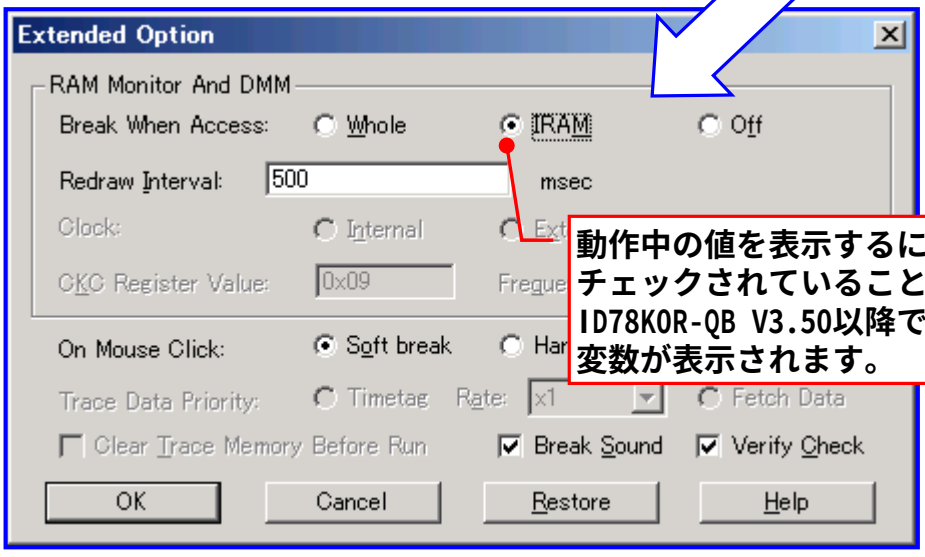
e.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

1.「A/DでSW」と同様にMINICUBE2を設定し、QB-78K0RKG3-TB + ベースボードと接続した後にPC本体とつなげます。



2.PM+ V6.31を起動し、プログラムをビルドしてID78K0R-QB上で実行します。
下図に示すように変数をWatch登録して、動作中の値を確認します。

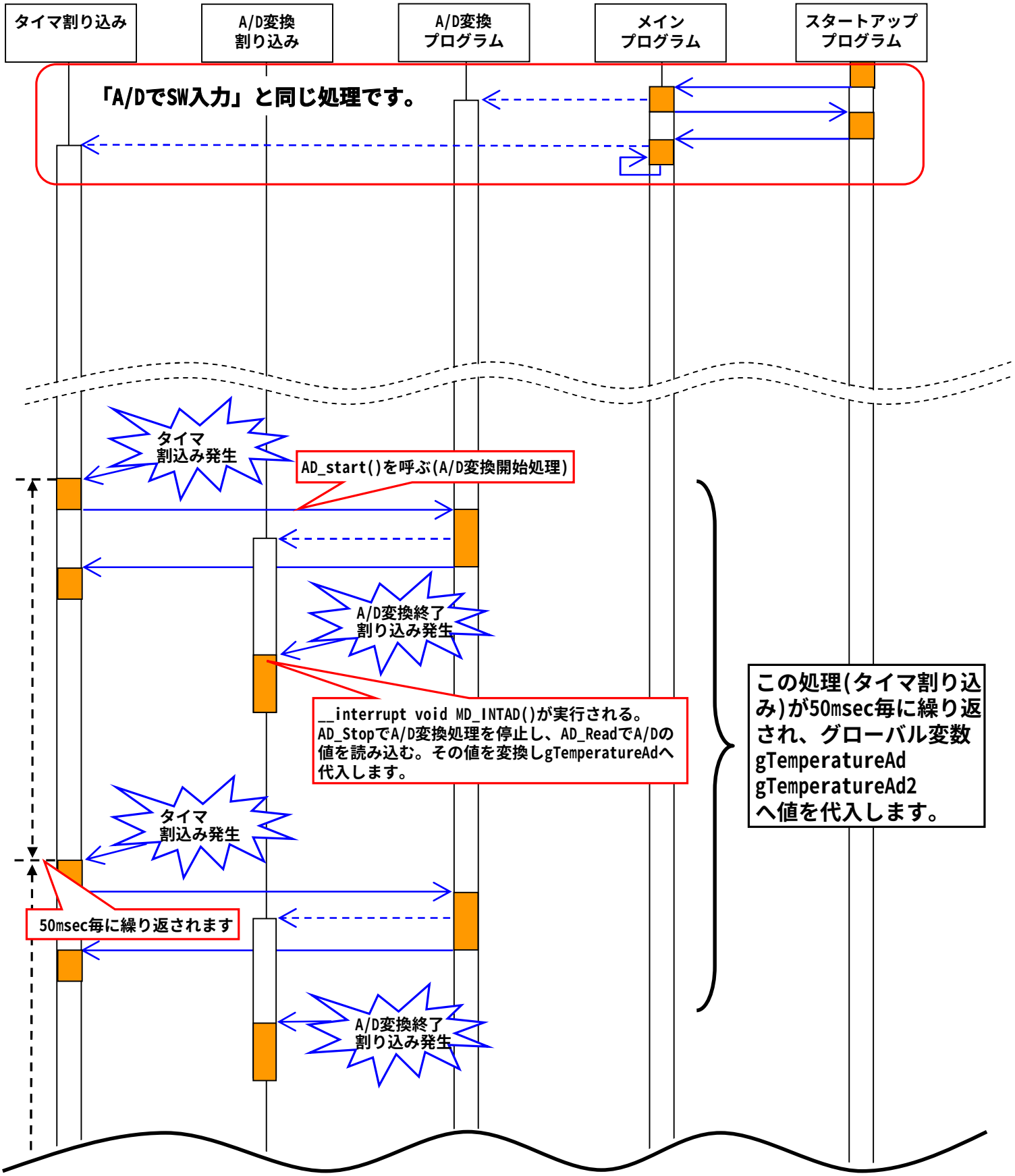




動作中の値を表示するにはデバッガのオプションで[IRAM]にチェックされていることを確認してください。
ID78K0R-QB V3.50以降では、この方法でプログラム実行中の変数が表示されます。

温度を計測

f.プログラム全体の流れを説明します。
シーケンス図で説明します。ただし「A/DでSW入力」と同じ部分は省略します。

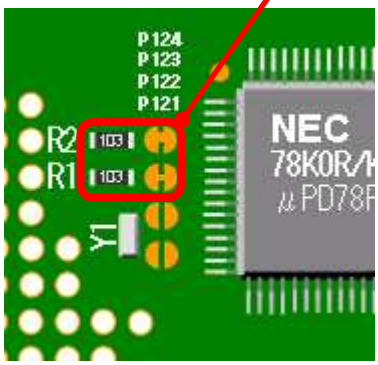


RTCで時計

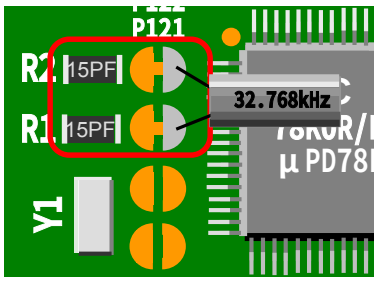
78K0RはRTC(リアルタイム･カウンタ)の機能を持っています。RTCとは簡単にいうと時計機能です。自動的にうるう年などの計算も行うカレンダー機能やアラーム機能、定期割り込み機能などを備えており、簡単に時計を実現可能です。ここではサブクロックを付け加え、現在時刻をRTCへ設定し、1秒毎にボード上のLEDを点滅させてみます。動作確認の時には、現在時刻を変数で表示するようにします。

7.サブクロックの取り付け

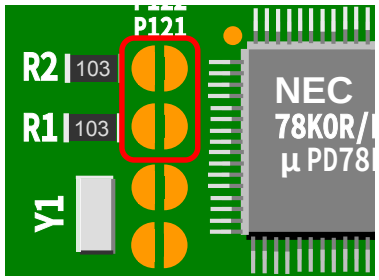
QB-78K0RKG3-TBには、サブクロックが搭載されていません。サブクロックを搭載する方法を説明します。

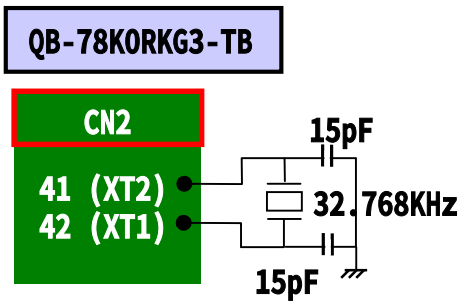


R1とR2を取り外します。替わりにコンデンサ(12～15PF)をつけます。そして、下図のようにサブシステム･クロックを接続して下さい。



実装が難しい場合は、●をパターンカットします。パターンカットすると、R1とR2のプルダウン抵抗が無効になります。CN2の41(XT2)、42(XT1)へサブシステム･クロック(32.768kHz)を接続して下さい。その際、コンデンサも取り付けてください。





組み立て順	部品の名称	型番	個数	備考
7	水晶振動子	32.768kHz	1	サブクロック用
	コンデンサ	12～15PF	2	

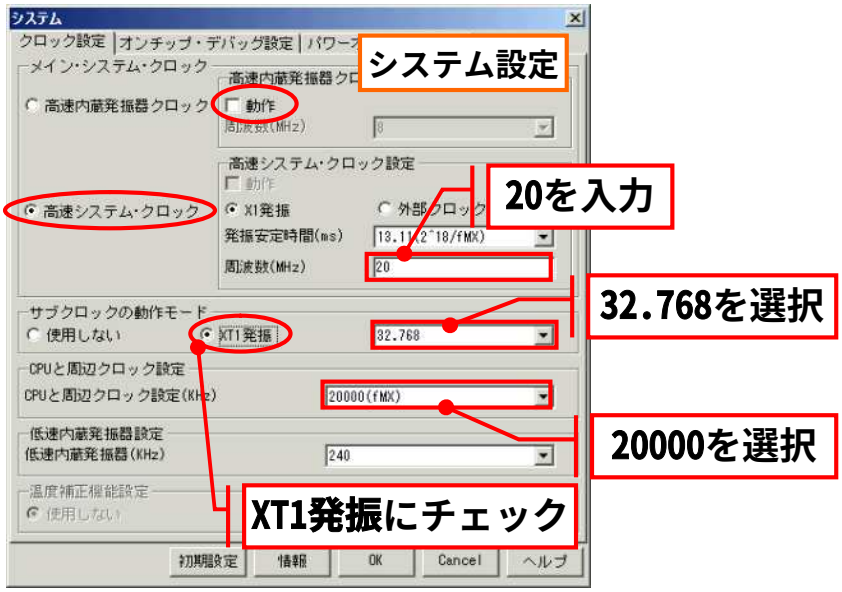
RTCで時計

30:00

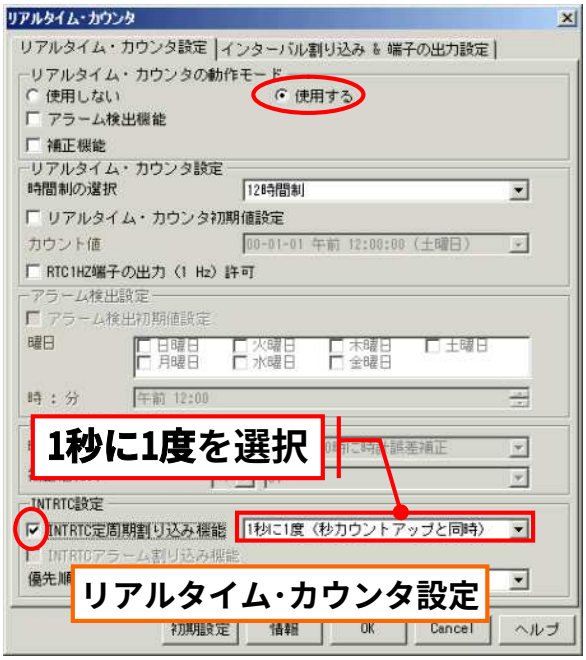
ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。RTCの時計機能をテストするためソフトウエアです。Applilet2による設定から説明します。

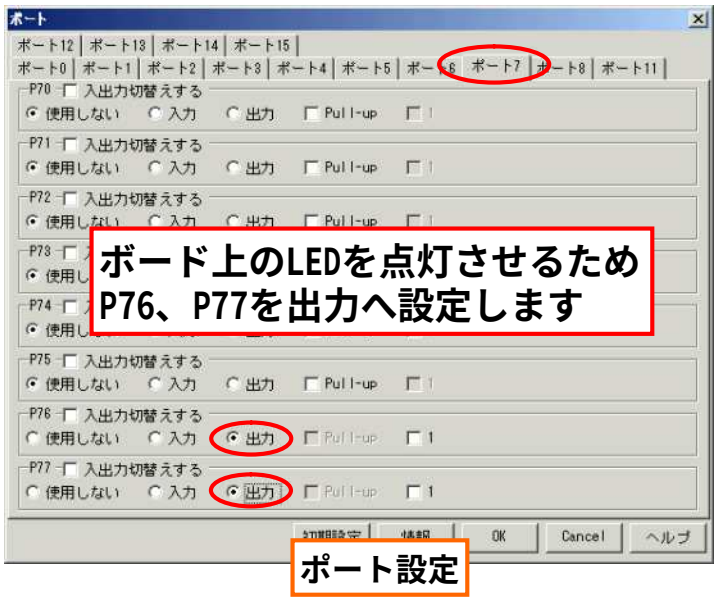
a. Applilet2を起動し、デバイス、プロジェクト名、システム設定します。
「準備(2)」を参照して、「システム」まで同じに設定します。マイクロコントローラ名「78K0R/KG3」、デバイス名「uPD78F1166_A0」、CPUクロック20MHz、オンチップ･デバッグ、ウォッチドッグ･タイマも同様に設定します。プロジェクト名は「RTC_test」としました。

b.「システム」のクロックを設定します。
オンチップ･デバッグの設定は、「A/DでSW入力」と同じですが、クロック設定だけは変更点があります。下図を参考に設定してください。



c.「リアルタイム･カウンタ」、「ポート」を設定します。
下図を参考に設定してください。





RTCで時計

d.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
Port.c /h	ポート初期化ファイル。
RTC.c /h	リアルタイム･カウンタ処理。
RTC_user.c	リアルタイム･カウンタ割り込みハンドラ定義処理。

上記のソースファイルで赤枠で示したのが「RTCで時計」で編集対象のファイルです。次ページより編集するソースを説明します。

RTCで時計

e.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

main.c

```
            リスト省略
/*  Global define
*******************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
UINT    g1secCounter;                   /* Interval timer 1sec(RTC使用)          */
struct  RTCCounterValue gsClockNow;     /* 現在の時間を表す                      */
/* End user code for global definition. Do not edit comment generated here */

void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    g1secCounter = 0;
    P7.6 = 0;
    P7.7 = 1;

    RTC_CounterEnable();
    gsClockNow.Year = 0x08;         /* 初期の日付2008/05/13 00:00を設定する */
    gsClockNow.Month = 0x05;
    gsClockNow.Day = 0x13;
    gsClockNow.Hour = 0;
    gsClockNow.Min = 0;
    gsClockNow.Sec = 0;
    RTC_CounterSet( gsClockNow );

    while (1) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

RTCとやりとりを行う時間設定の構造体

ボード上のLED初期点灯を設定します

RTC動作を開始します

設定した日付をRTCへセットします

RTC_user.c

```
            リスト省略
/*  Global define
*******************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
extern UINT    g1secCounter;
extern struct  RTCCounterValue gsClockNow;
/* End user code for global definition. Do not edit comment generated here */
            リスト省略
/*
**-----------------------------------------------------------------------------
**  Abstract:   This function is  real-time counter constant-period interrupt service handler.
**  Parameters: None
**  Returns:    None
**-----------------------------------------------------------------------------
*/
void CALL_RTC_ConstantPeriodINT( void )
{
    /* Start user code. Do not edit comment generated here */
    UCHAR dat;
    dat = P7.6;
    P7.6 = P7.7;
    P7.7 = dat;

    RTC_CounterGet( &gsClockNow );
    g1secCounter++;
    /* End user code. Do not edit comment generated here */
}
```

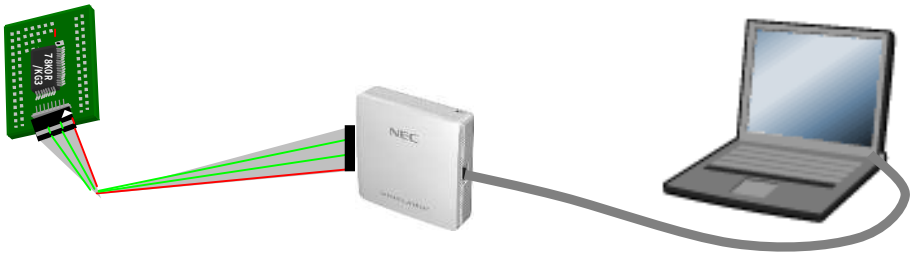
ボード上のLED点灯を入れ替えます

現在の時刻を取得します

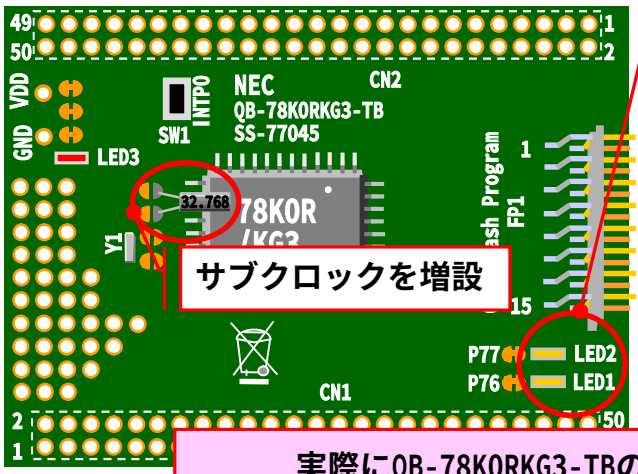
RTCで時計

f.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

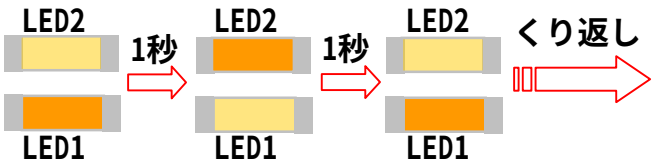
1.「A/DでSW」と同様にMINICUBE2を設定し、QB-78K0RKG3-TB と PC本体とつなげます。
RTCのみを動作確認するのでベースボードに接続する必要はありません。



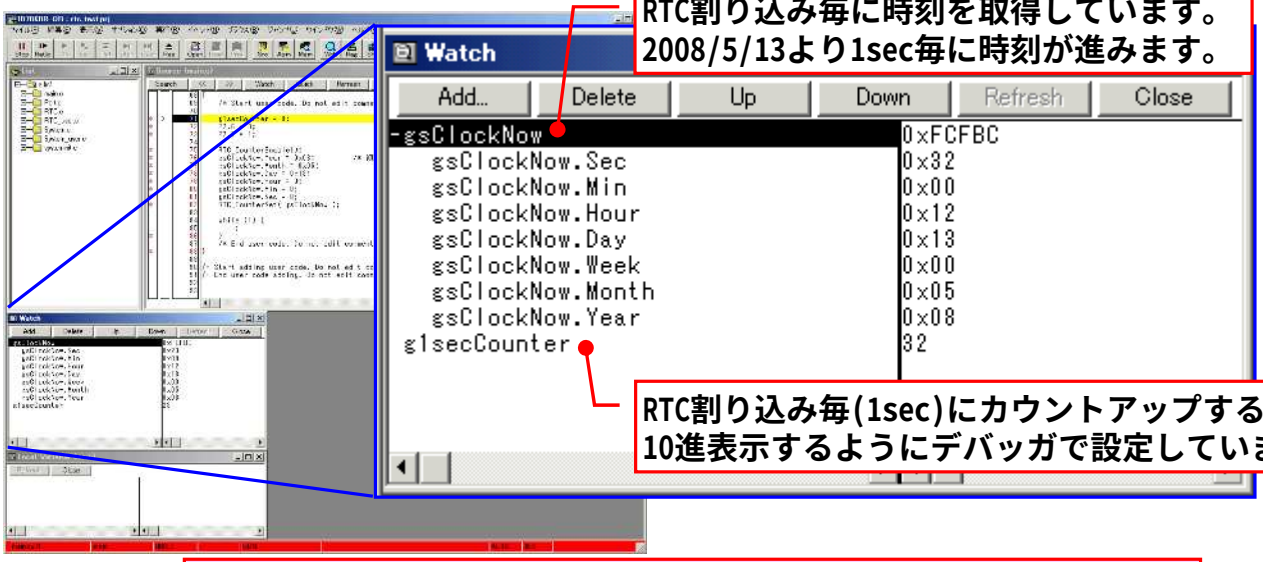
2.PM+ V6.31を起動し、プログラムをビルドしてID78K0R-QB上で実行します。



LED2/LED1が交互に1秒づつ点灯することを確認してください。



実際にQB-78K0RKG3-TBのLEDを見て動作確認します。



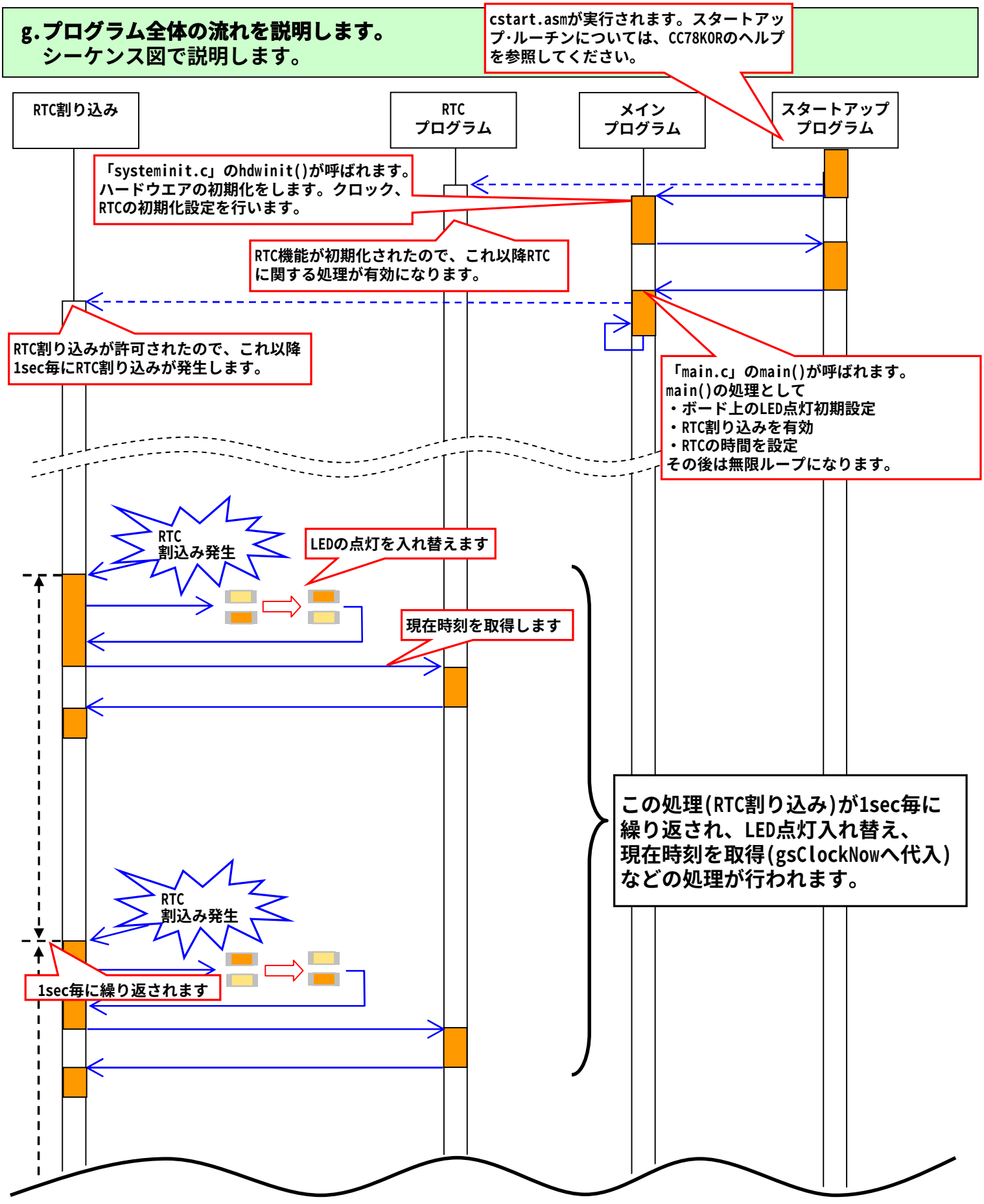
RTC割り込み毎に時刻を取得しています。
2008/5/13より1sec毎に時刻が進みます。

RTC割り込み毎(1sec)にカウントアップする変数です。
10進表示するようにデバッガで設定しています。

デバッガ上で確認してください。変数はRRM設定しており実行中に値が見られるようにしています。

RTCで時計

g.プログラム全体の流れを説明します。
シーケンス図で説明します。



LCDへ表示

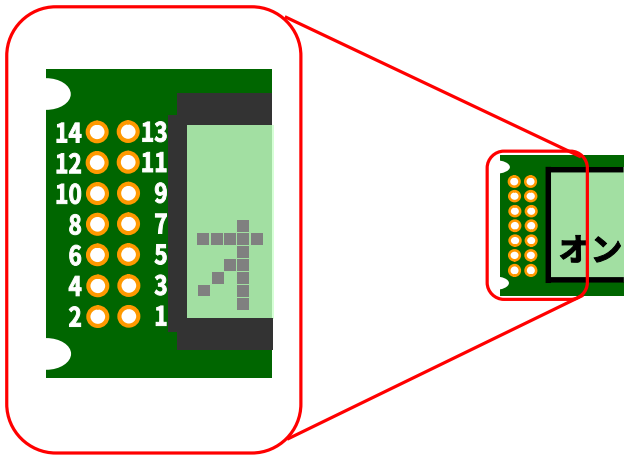
LCDキャラクタディスプレイ(M1632互換)16文字x2行を使った表示を行います。回路を簡易にするためにバックライト非搭載のものを使いました。実際に使用したものは「SC1602BS」です。

[LCDキャラクタディスプレイの信号]

このLCDキャラクタディスプレイはモジュールとなっており、コントローラが内蔵されています。コントローラにコマンドを送ることによって文字を表示します。英数字、記号、カタカナが表示可能です。また、自分でキャラクタ(5x7ドット)を8文字分定義できます。

SC1602BSは14ピンのコネクタを持っています。

No.	信号名	概要
1	VDD	5V
2	Vss	GND
3	VO	コントラスト調整用
4	RS	レジスタ選択
5	R/W	Read/Write
6	E	イネーブル信号
7 : 14	DB0 : DB7	DataBit0 : DataBit7



[LCDキャラクタディスプレイのコマンド]

主なコマンド一覧です。

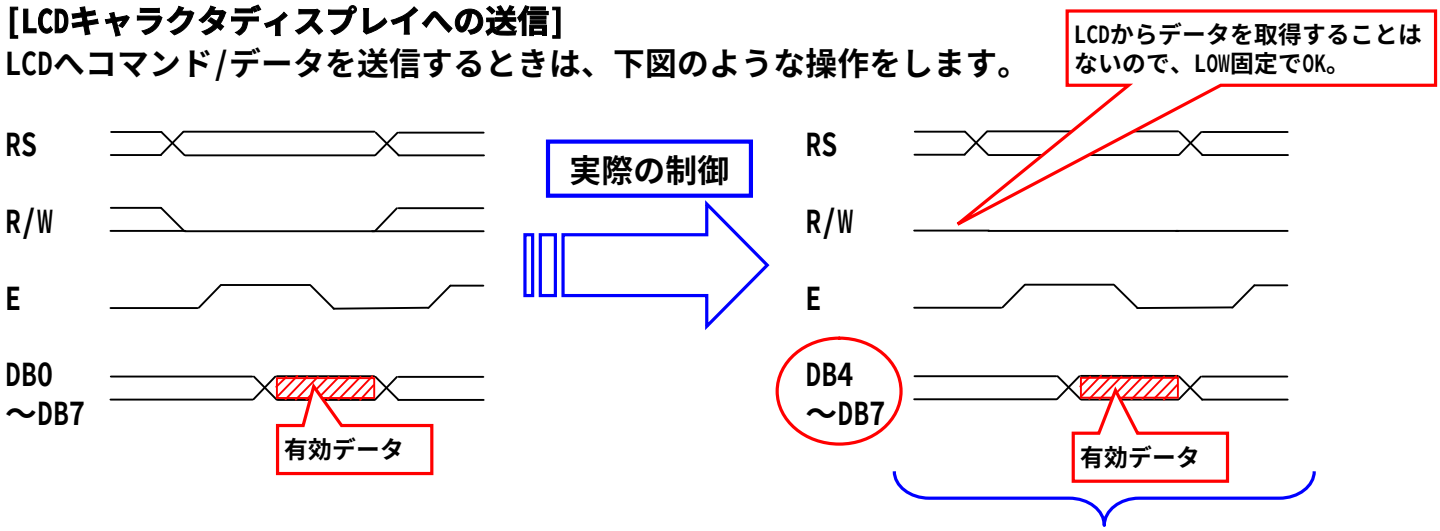
コマンド	コード										概要
	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
表示クリア	0	0	0	0	0	0	0	0	0	1	全表示を消去し、カーソルをホームポジションへ移動します。
カーソルホーム	0	0	0	0	0	0	0	0	1	-	カーソルのみをホームポジションへ移動します。
モードセット	0	0	0	0	0	0	0	1	I/D	S	データ書き込み時のカーソル移動:I/D、シフト表示:Sの設定です。
表示制御	0	0	0	0	0	0	1	D	C	B	.LCD表示:D、カーソル:C、ブリンク(点滅):BのON/OFFを設定します。
カーソル制御	0	0	0	0	0	1	S/C	R/L	-	-	カーソル移動:R/L、シフト表示:S/Cを行います。
ファンクションセット	0	0	0	0	1	DL	N	F	-	-	データ長:DL、表示行数:N、キャラクタドット数:Fを設定します。
CGRAMアドレスセット	0	0	0	1	A5	A4	A3	A2	A1	A0	LCD上のアドレス(A0～A5)を設定します。自作キャラクタを定義するのに使います。
DDRAMアドレスセット	0	0	1	A6	A5	A4	A3	A2	A1	A0	LCD上のアドレス(A0～A6)を設定します。キャラクタ表示に使います。
CGRAM/DDRAM データ書き込み	1	0	D7	D6	D5	D4	D3	D2	D1	D0	LCDへデータを送ります。

その他にLCDキャラクタディスプレイからデータを読み出すコマンドがありますが使用していません。

LCDへ表示

LCDキャラクタディスプレイへのコマンド送信方法を説明します。配線をなるべく簡単にするためにインターフェース長を4bitで行っています。4bitで行うとDB4～DB7のみが使用されるのでDB0～DB3は配線の必要がありません。注

[LCDキャラクタディスプレイへの送信]
LCDへコマンド/データを送信するときは、下図のような操作をします。



DB4～DB7の4bitへデータを設定し、2回送信を行う。これで8bitのデータを実現する。(インターフェース長4bit転送)

※注
DB0～DB3の配線の必要はないと書きましたが、これはLCDのインターフェース長が4bitになってからの操作です。電源投入時は8bitインターフェース長になっており、これを4bitにするには前ページの[LCDキャラクタディスプレイのコマンド]ファンクションセットを実行する必要があります。このファンクションセットDB3[N]を1へ設定しなければなりません。

コマンド	コード										概要
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
ファンクションセット	0	0	0	0	1	DL	N	F	-	-	データ長:DL、表示行数:N、キャラクタドット数:Fを設定します。

そのため、インターフェース長を4bitに設定するコマンドを送ってから、更にDB3[N]を1にするコマンドを送ります。下記のように、2回ファンクションセットを実行することで、電源投入時からでも4bitインターフェース長でLCDキャラクタディスプレイ2行表示が可能になります。

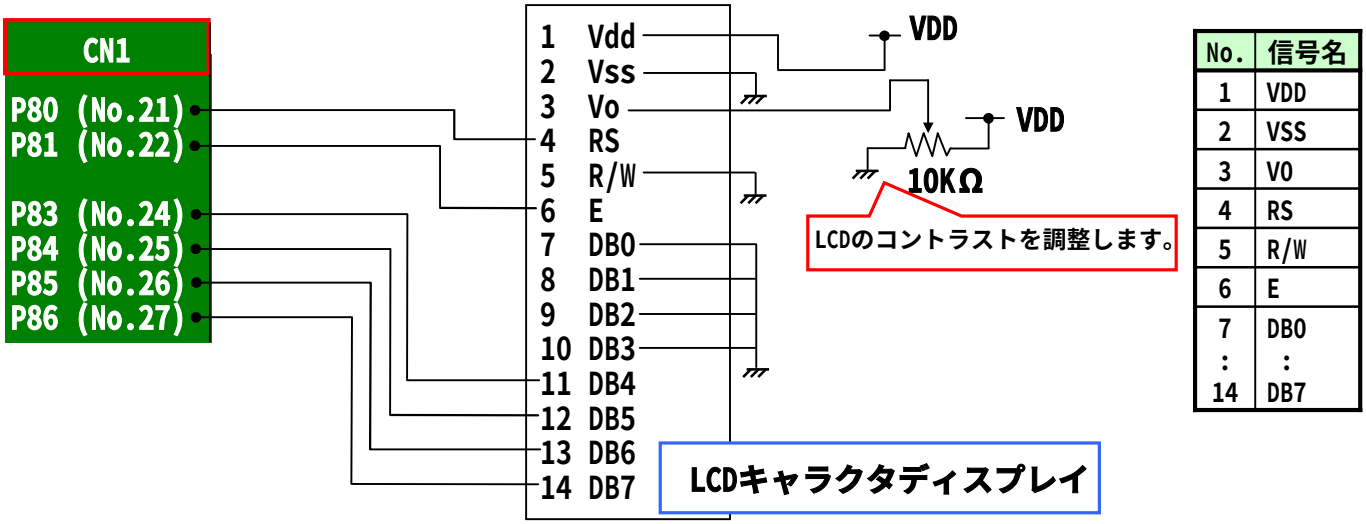
1. 8bitインターフェース長でファンクションセットを実行。
   (DB0～DB3は0固定なのでLCDは1行表示のままだが、このコマンド実行で4bitインターフェース長へ変更される。)
2. 4bitインターフェース長でファンクションセットを実行。
   (全てのコマンドを指定できるので、このコマンド実行でLCDを2行表示にする。)

LCDへ表示

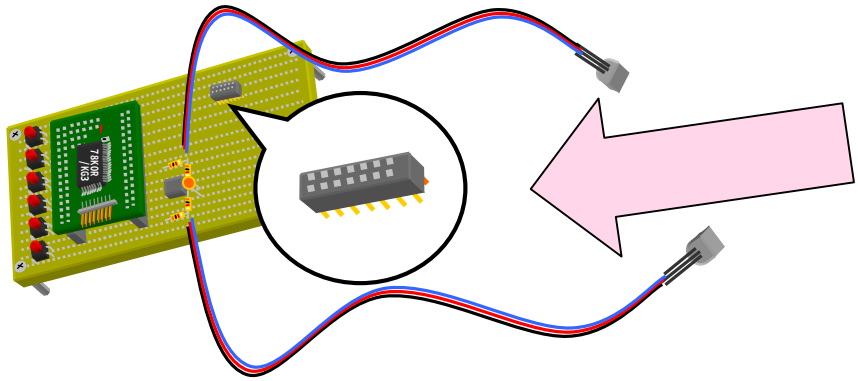
LCDキャラクタディスプレイにピンヘッダコネクタ、ベースボードにピンフレームを取り付けてください。また、LCDキャラクタディスプレイ上にコントラスト調整用の半固定抵抗を搭載すると配線がすっきりまとまります。

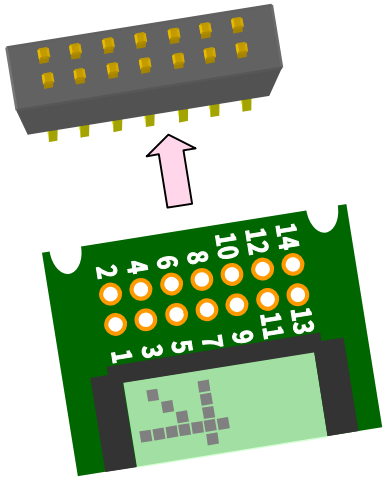
8.LCDキャラクタディスプレイの取り付け

インターフェース長を4bitにすればDB0～DB3の接続は不要ですが、LCDキャラクタディスプレイの初期値はインターフェース8bit長になっているため、インターフェース長を変更するコマンドの送信を行う必要があります。その送信にDB3が必要なため下記の配線図のようになっています。



No.	信号名
1	VDD
2	VSS
3	VO
4	RS
5	R/W
6	E
7 : 14	DB0 : DB7





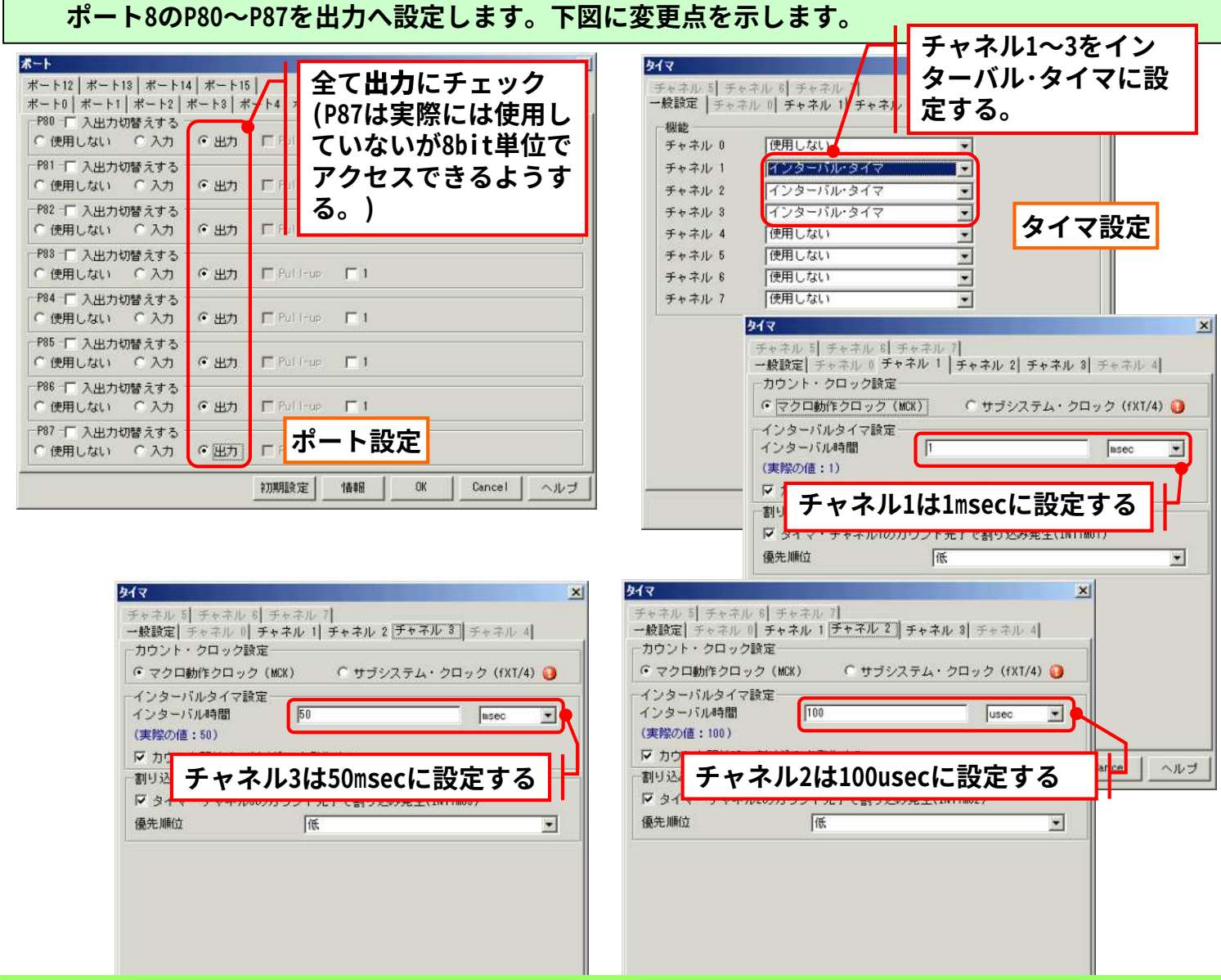
組み立て順	部品の名称	型番	個数	備考
8	16x2キャラクタLCDディスプレイ	SC1602B	1	M1632ピンコンパチブル
	2x5ピンヘッダ 2x5ピンフレーム		各1	LCD固定に使用、大きいものをカットして使う。
	半固定抵抗	10KΩ	1	LCDコントラスト調整用

LCDへ表示

ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。A/Dより入力された温度をLCDへ表示を行うソフトウエアです。Applilet2による設定から説明します。

a. Applilet2を起動し、デバイス、プロジェクト名、システム、A/Dコンバータ設定します。
「準備(2)」を参照して、同様の設定をします。マイクロコントローラ名「78K0R/KG3」、デバイス名「uPD78F1166_A0」、CPUクロック20MHz、オンチップ･デバッグ、ウォッチドッグ･タイマ、A/Dコンバータ(チャネル13～15を使用)も同様に設定します。プロジェクト名は「LCD_test」です。

b.「タイマ」、「ポート」を設定します。
ポート8のP80～P87を出力へ設定します。下図に変更点を示します。



資料

LCDへの表示タイミングについて
「温度計測」を50msec毎に行っていますが、ここではそのタイミングでLCDへの表示も行います。最後にシステムを動作させる場合は時刻を表示させます。その場合は時刻を秒単位で表示しますので、1秒毎にLCD表示更新を行います。

LCDへ表示

c.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
Port.c /h	ポート初期化ファイル。
TAU.c /h	タイマ処理。
TAU_user.c	ユーザー処理用タイマ割り込みハンドラ定義処理。
Ad.c /h	A/D処理。
Ad_user.c	A/D処理用A/D変換完了割り込みハンドラ定義処理。
ctl_lcd.c	LCD制御用ソースファイル。新規に作成します。

上記のソースファイルで赤枠で示したのが「LCDへ表示」で編集対象のファイルです。次ページより編集するソースを説明します。

LCDへ表示

d.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

main.c(リスト1)

```
            リスト省略
/***********************************************************************************
**  Include files
***********************************************************************************/
#include "macrodriver.h"
#include "Ad.h"
#include "TAU.h"
#include "System.h"
#include "Port.h"
/* Start user code for include definition. Do not edit comment generated here */
extern void lcd_init( void );
extern void lcd_vram2lcd( void );
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
UINT    g1msecCounterWait;                  /* Interval timer 1msec for wait function(ch1使用 )     */
UINT    g100usecCounterWait;                /* Interval timer 100usec for wait function(ch2使用)    */
UINT    g50msecCounter;                     /* Interval timer 50msec(ch3使用)                       */
UCHAR   gSelectAd;                          /* ADチャネルを選択する                                 */
USHORT  gTemperatureAd;                     /* ADチャネル14に接続した温度センサ1の直値              */
USHORT  gTemperatureAd2;                    /* ADチャネル13に接続した温度センサ2の直値              */
/* End user code for global definition. Do not edit comment generated here */

/*-----------------------------------------------------------------------------
**  Abstract:          This function  implements main function.
**  Parameters:        None
**  Returns:           None
**-----------------------------------------------------------------------------
*/
void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    g1msecCounterWait = 0;
    g100usecCounterWait = 0;
    g50msecCounter = 0;
    gSelectAd = 0;
    gTemperatureAd = 0;
    gTemperatureAd2 = 0;

    TAU_Channel1_Start();
    TAU_Channel2_Start();
    TAU_Channel3_Start();

    lcd_init();

    while (1) {
        if ( ( g50msecCounter & 0xf ) == 0 )
        {
            lcd_vram2lcd();
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

50msec x 16(800msec)に1回
LCDの表示を更新します。

LCDへ表示

main.c(リスト2)

```
/* Start adding user code. Do not edit comment generated here */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 100usecの時間待ち                                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void wait100usec( UINT para1_ )
{

    /* 100us の時間待ちを para1_ 分だけ行う */
    g100usecCounterWait = 0;
    while ( g100usecCounterWait < para1_ ) ;

}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 1msecの時間待ち                                                            */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void wait1msec( UINT para1_ )
{

    /* 1msec の時間待ちを para1_ 分だけ行う */
    g1msecCounterWait = 0;
    while ( g1msecCounterWait < para1_ ) ;

}
/* End user code adding. Do not edit comment generated here */
```

LCDへ表示

Ad_user.c

```
            リスト省略
*
************************************************************************************
**  Global define
************************************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
extern UCHAR   gSelectAd;
extern USHORT  gTemperatureAd;
extern USHORT  gTemperatureAd2;
/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------------
**  Abstract:           This function is INTAD interrupt service routine.
**  Parameters:         None
**  Returns:            None
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTAD(void)
{
    /* Start user code. Do not edit comment generated here */
    USHORT adtemp;
    UCHAR  dat;

    AD_Stop();

    AD_Read( &adtemp );
    dat = (UCHAR)((adtemp * 48) / 400);

    if ( gSelectAd == 0 )
    {
        gTemperatureAd = dat;
    }
    else
    {
        gTemperatureAd2 = dat;
    }

    gSelectAd ^= 1;

    /* End user code. Do not edit comment generated here */
}
```

A/D変換を停止します

A/D変換値を読み込みます

読み込んだ値から温度を求めます

gSelectAdの値によってA/D変換を行ったチャネルを処理します。

次のA/D変換チャネルを決めます

LCDへ表示

TAU_user.c

```
            リスト省略
#include "macrodriver.h"
#include "TAU.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "Ad.h“
extern void lcd_temperature( void );
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
extern UCHAR    gSelectAd;
extern UINT     g1msecCounterWait;
extern UINT     g100usecCounterWait;
extern UINT     g50msecCounter;
/* End user code for global definition. Do not edit comment g
/*-----------------------------------------------------------------------------
**  Abstract:          This function is INTTM01 interrupt service routine.
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM01( void )
{
    /* Start user code. Do not edit comment generated here */
    g1msecCounterWait++;
    /* End user code. Do not edit comment generated here */
}

/*-----------------------------------------------------------------------------
**  Abstract:          This function is INTTM02 interrupt service routine.
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM02( void )
{
    /* Start user code. Do not edit comment generated here */
    g100usecCounterWait++;
    /* End user code. Do not edit comment generated here */
}

/*-----------------------------------------------------------------------------
**  Abstract:          This function is INTTM03 interrupt service routine.
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTTM03(void)
{
    /* Start user code. Do not edit comment generated here */
    g50msecCounter++;
    if ( gSelectAd == 0 )
    {
        AD_SelectADChannel( ADCHANNEL14 );
    }
    else
    {
        AD_SelectADChannel( ADCHANNEL13 );
    }
    lcd_temperature();    /* 温度表示を行う */

    AD_Start();
    /* End user code. Do not edit comment generated here */
}
```

実際のソースと異なり、コメント類を省略して書いてありますが、青字の追加するコードに注目してください。

1msec毎にカウントアップする変数

100usec毎にカウントアップする変数

gSelectAdの値によってA/D変換するチャネルを選択します

チャネル14をA/D変換

チャネル13をA/D変換

LCDへ温度を表示します

A/D変換を開始します

LCDへ表示

ctl_lcd.c(リスト1)

```
            リスト省略
/* Include files */
#include "macrodriver.h"
#include "System.h"
#include "String.h"
#include "Stdlib.h"

/* LCD定義 */
typedef struct {
    UCHAR rs    : 1;
    UCHAR sts   : 1;
    UCHAR data  : 8;
} ST_LCDCMD;

#define D_LCD_X_SIZE        16          /* LCD表示サイズ X                                          */
#define D_LCD_Y_SIZE        2           /* LCD表示サイズ Y                                          */
#define D_LCD_SIZE          (D_LCD_X_SIZE * D_LCD_Y_SIZE)
#define D_LCD_CGRAMSIZE     (8 * 8)

/* LCDインストラクション設定 */
#define D_LCD_CLR           0x01        /* 表示クリア                                               */
#define D_LCD_HOME          0x02        /* カーソルをホームポジションへ移動                         */
#define D_LCD_ENTRY         0x04        /* エントリー・モード・セット(デフォルトはカーソル左移動,表示シフトなし*/
#define D_LCD_SET           0x08        /* 表示ON/OFF設定(デフォルトは表示OFF,カーソルOFF,ブリンクOFF)  */
#define D_LCD_SHIFT         0x10        /* カーソル/表示の移動(デフォルトはカーソル位置を左へシフト     */
#define D_LCD_FUNC          0x20        /* ファンクションセット(データ長、表示行数、キャラクタドット数) */
#define D_LCD_CGRAM         0x40        /* CGRAMのアドレス設定                                      */
#define D_LCD_DDRAM         0x80        /* DDRAMのアドレス設定                                      */

/* LCDインストラクションのbit設定 */
#define D_LCD_ENTRY_INC     0x02        /* カーソル右移動(インクリメント)                           */
#define D_LCD_SET_ON        0x04        /* 表示ON                                                   */
#define D_LCD_2LINE         0x08        /* 表示行数を2行にする                                      */

/* in main.c */
extern void wait100usec( UINT para1_ );
extern void wait1msec( UINT para1_ );

/* in This file. */
void lcd_temperature( void );

/*****************************************************************************
**  Global define
*****************************************************************************/

/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern USHORT gTemperatureAd;extern USHORT gTemperatureAd2;

/* - - - - - - - - このファイルでの定義 - - - - - - - - */
ST_LCDCMD  gtLcdCmd;                        /* LCDへコマンド送る際の構造体                          */
UCHAR   gLcdUpdate;                         /* LCD表示を更新するカウンタ                            */
UCHAR   gtLcdBuffer[ D_LCD_SIZE + 1 ];      /* LCDのキャラクタ単位のRAMバッファ 16文字x2行           */
UCHAR   gLcdCursor;                         /* LCDカーソル位置                                      */
const UCHAR gLCDTEXT[] = “ ｵﾝﾄﾞ1:    2:    ”;  /* 温度表示時の文字列                                   */

const UCHAR gtCgramData[] =
{   /* LCDへ定義する℃のフォントデータ */
    0x08, 0x14, 0x08, 0x07, 0x08, 0x08, 0x08, 0x07,  0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,  0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f,  0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f,  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f
};
```

LCDにデータ/コマンドを送る際の構造体定義。

LCDへ表示

ctl_lcd.c(リスト2)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDへデータ書き込み(8bitデータ送信)                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_write8bit( UCHAR rs_, UCHAR status_, UCHAR data_ )
{
    UCHAR  lcddata;

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 8bitデータバス */
    lcddata |= (data_ << 2);

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 1 );

    /* 動作起動(イネーブル信号)をLOWにする */
    P8.1 = 0;
    wait100usec( 1 );

}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDへデータ書き込み(4bitデータ送信)                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_write4bit( UCHAR rs_, UCHAR status_, UCHAR data_ )
{
    UCHAR  lcddata, dat;

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 上位4bitデータを送る */
    dat = (data_ & 0xf0) >> 1;
    lcddata |= dat;

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 2 );

    /* 動作起動をLOWにする */
    P8.1 = 0;
    wait100usec( 2 );

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 下位4bitデータを送る */
    dat = (data_ & 0x0f) << 3;
    lcddata |= dat;

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 2 );

    /* 動作起動(イネーブル信号)をLOWにする */
    P8.1 = 0;
    wait100usec( 2 );
}
```

LCDへ表示

ctl_lcd.c(リスト3)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCD初期化                                                                   */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_init()
{
    int i;
    UCHAR dat;

    gLcdUpdate = 0;
    gLcdCursor = 0;
    memset( gtLcdBuffer, ' ', sizeof(gtLcdBuffer) );
    lcd_write4bit( 0, 1, 0x38 );        /* 8bitモードへ設定 */
    wait1msec( 100 );
    lcd_write8bit( 0, 1, 4 );           /* 4bitモードインターフェース設定  */
    gtLcdCmd.rs = 0;                    /* ファンクション行数セット        */
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_FUNC | D_LCD_2LINE;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 100 );

    /* エントリー・モード設定 */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_ENTRY | D_LCD_ENTRY_INC;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 5 );

    /* 表示ON, カーソルOFF, ブリンクOFF */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SET | D_LCD_SET_ON;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 5 );

    /* 表示クリア */
    lcd_write4bit( 0, 1, D_LCD_CLR );
    wait1msec( 10 );

    /* CGRAMへデータを転送する */
    lcd_write4bit( 0, 1, D_LCD_CGRAM );
    for ( i = 0; i < D_LCD_CGRAMSIZE; i++ )
    {
        /* CGRAMデータ書き込み */
        dat = gtCgramData[ i ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 温度表示                                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_temperature( )
{
    int i;
    UCHAR num1[ 7 ], num2[ 7 ];
    UINT temp1, temp2;

    /* 各センサより温度を計算 */
    temp1 = gTemperatureAd;
    temp2 = gTemperatureAd2;

    itoa( temp1, (char *)num1, 10 );
    itoa( temp2, (char *)num2, 10 );
```

電源ON時でもデバッグ中のリセット時でも4bitデータ送信モードへ移行させます。

温度表示する際の「℃」キャラクタ定義を行います。

内部バッファに下記のデータを設定します。
[オンド1:xx℃ オンド2:xx℃]

LCDへ表示

ctl_lcd.c(リスト4)

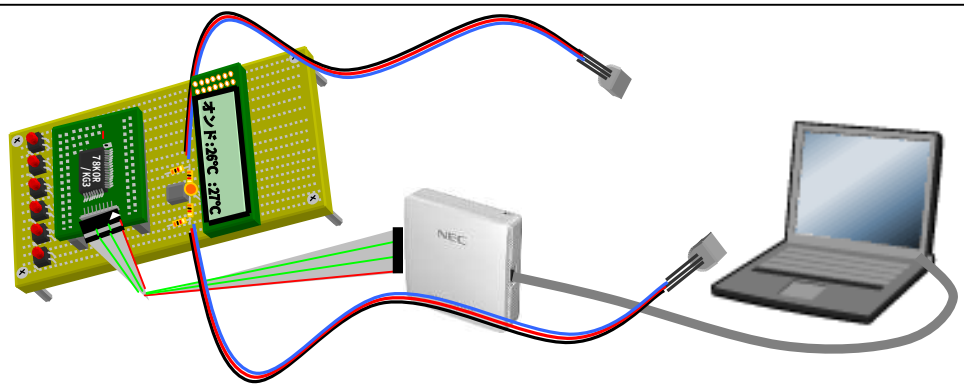
```
    for ( i = 0; i < D_LCD_X_SIZE; i++)
    {
        gtLcdBuffer[ i + D_LCD_X_SIZE ] = gLCDTEXT[ i ];
    }
    gtLcdBuffer[ 7 + D_LCD_X_SIZE ] = num1[ 0 ];
    gtLcdBuffer[ 8 + D_LCD_X_SIZE ] = num1[ 1 ];
    if ( num1[ 1 ] != 0 )
    {
        gtLcdBuffer[ 9 + D_LCD_X_SIZE ] = 0;        /* 「℃」表示 */
    }
    gtLcdBuffer[ 13 + D_LCD_X_SIZE ] = num2[ 0 ];
    gtLcdBuffer[ 14 + D_LCD_X_SIZE ] = num2[ 1 ];
    if ( num2[ 1 ] != 0 )
    {
        gtLcdBuffer[ 15 + D_LCD_X_SIZE ] = 0;       /* 「℃」表示 */
    }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDバッファVRAMからLCD VRAMへ転送                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_vram2lcd()
{
    UINT i;
    UCHAR dat;

    /* DDRAMアドレス設定 */
    lcd_write4bit( 0, 1, D_LCD_DDRAM );
    wait1msec( 1 );

    for ( i = 0; i < (D_LCD_SIZE/2); i++ )
    {
        /* DDRAMデータ書き込み */
        dat = gtLcdBuffer[ i ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }

    /* DDRAMアドレス設定 */
    lcd_write4bit( 0, 1, D_LCD_DDRAM + 0x40 );
    wait1msec( 1 );

    for ( i = 0; i < (D_LCD_SIZE/2); i++ )
    {
        /* DDRAMデータ書き込み */
        dat = gtLcdBuffer[ i + D_LCD_X_SIZE ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }
    wait1msec( 1 );
}
```

内部バッファからデータを転送して実際にLCDへ表示します。

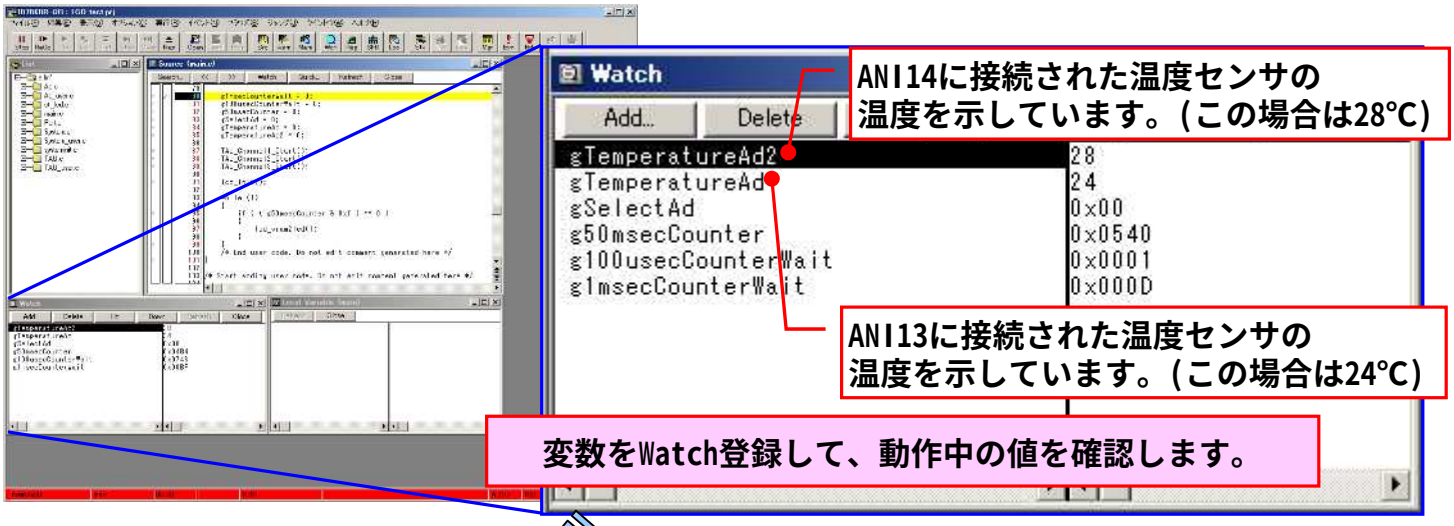
LCDへ表示

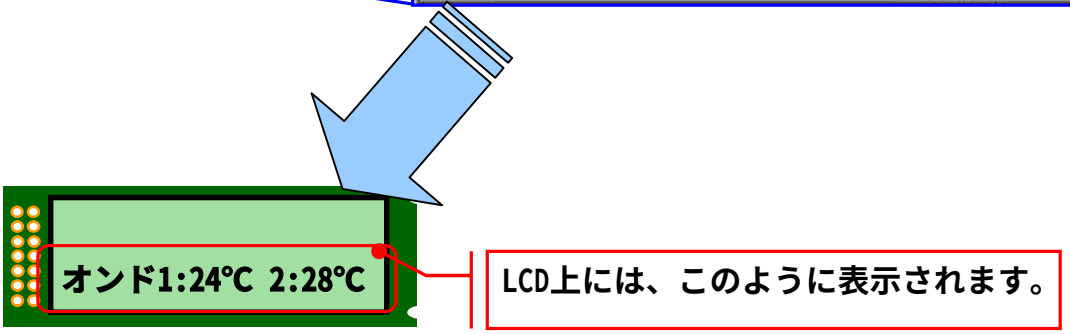
e.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

1.「A/DでSW」と同様にMINICUBE2を設定し、QB-78K0RKG3-TB + ベースボードと接続した後にPC本体とつなげます。



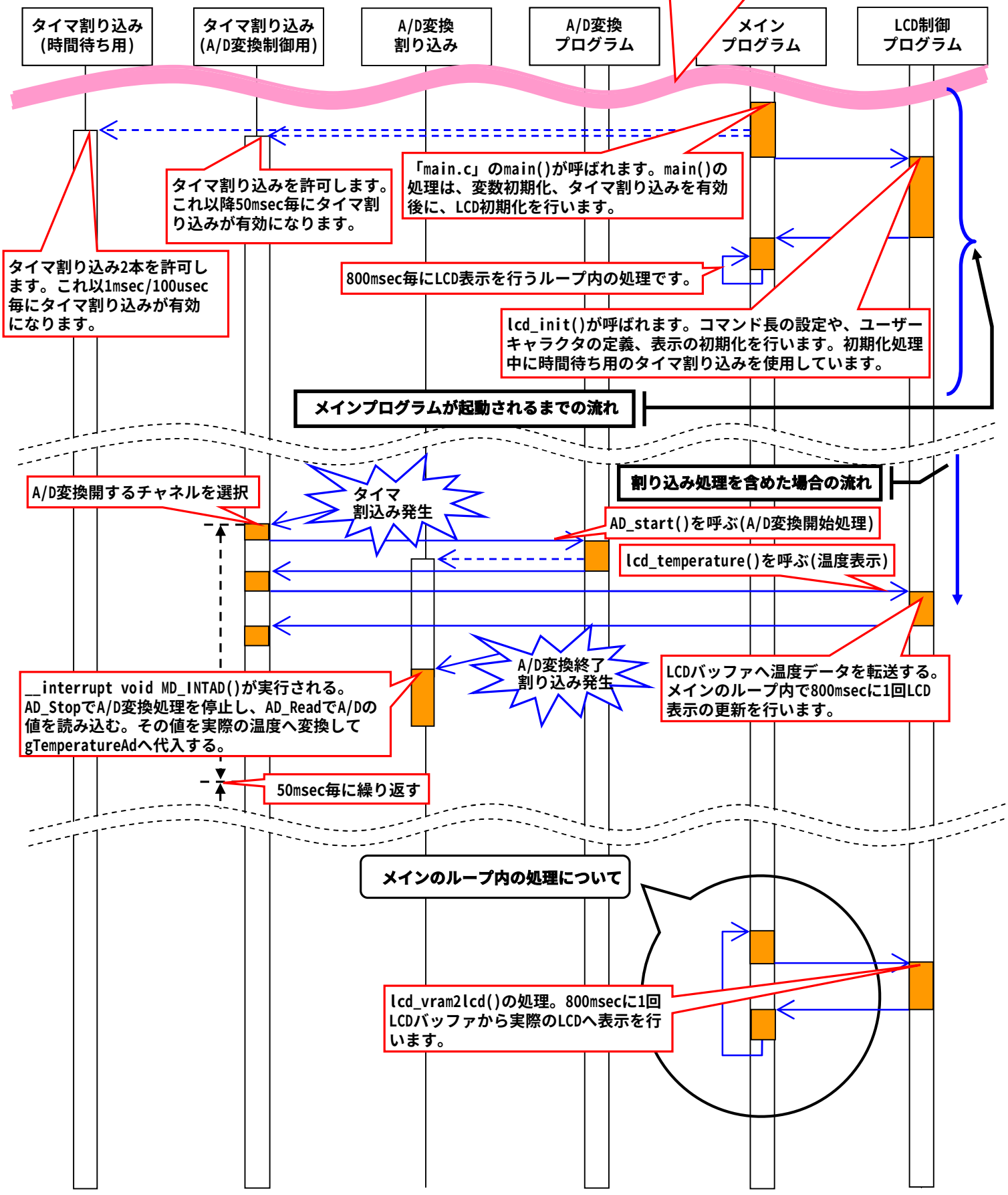
2.PM+を起動し、プログラムをビルドしてID78K0R-QB上で実行します。
変数の登録などは「温度を計測」の章で実行させた場合を参考にしてください。





LCDへ表示

f.プログラム全体の流れを説明します。
シーケンス図で説明します。



外部へデータ保存

I2Cインターフェースを利用して外付けのEEPROMへデータ保存を行います。
データを保存する方法にはいくつかあります。代表的な例を挙げます。

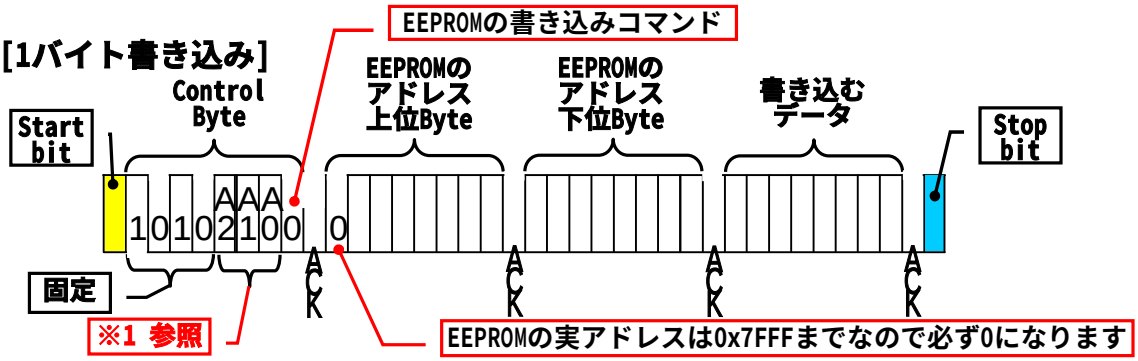
保存媒体	応答速度	通信方法	必要端子	プログラム難易度	特徴(用途)
メモリカード	速い	複雑	多い	高い	CF/SD/MSなど多種ある。大量にデータをやりとりする場合に使う
IC	遅い	簡単	少ない	低い	1バイト単位で書き換え可能。 通信方法としてSPI/I2Cなど

ここでは、入手しやすいI2Cを使いIC(EEPROM 24C256)へデータの書き込みを行います。データが正しく書き込まれたかを検証するプログラムを作成します。

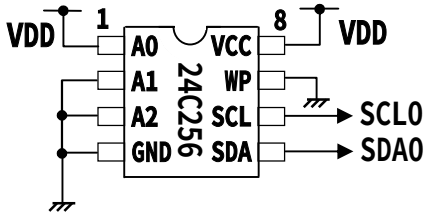
[I2C EEPROMのプロトコル]

I2CはSCL/SDA(78K0RではSCL0/SDA0を使います)の2線で制御を行います。SCLはクロック信号、SDAがデータ信号を扱います。接続は78K0Rがシングルマスタ でEEPROMがスレーブという簡単な構成です。
以下に書き込み/読み込みを行う場合のSDAについて説明します(SCLはクロック信号なので省略)。

[1バイト書き込み]

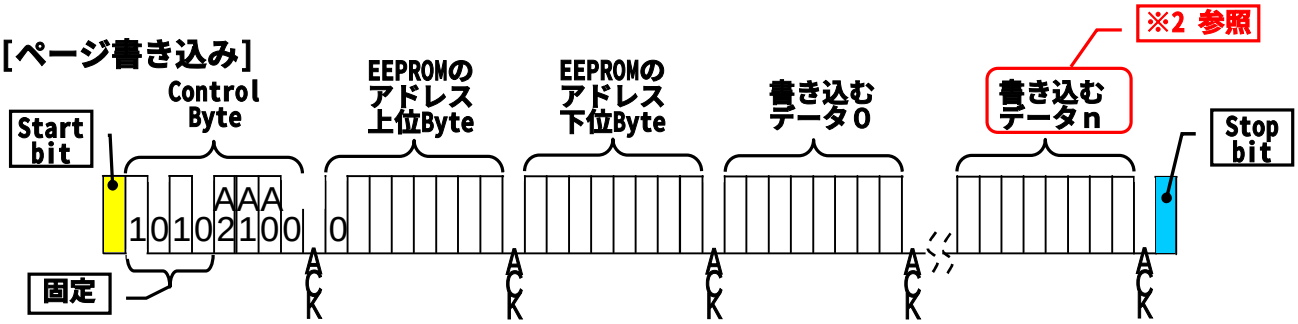


※1 EEPROM(24C256)の設定を反映させます。EEPROMA0～A2の端子状態(アドレス)設定します。A0～A2まで設定できるEEPROMなら、A0～A2の値をユニークな値にして8個まで並列に接続可能です。



左記の回路の場合Control Byteは[10100010]となります
このControl Byteの設定は全てのコマンドに共通です。

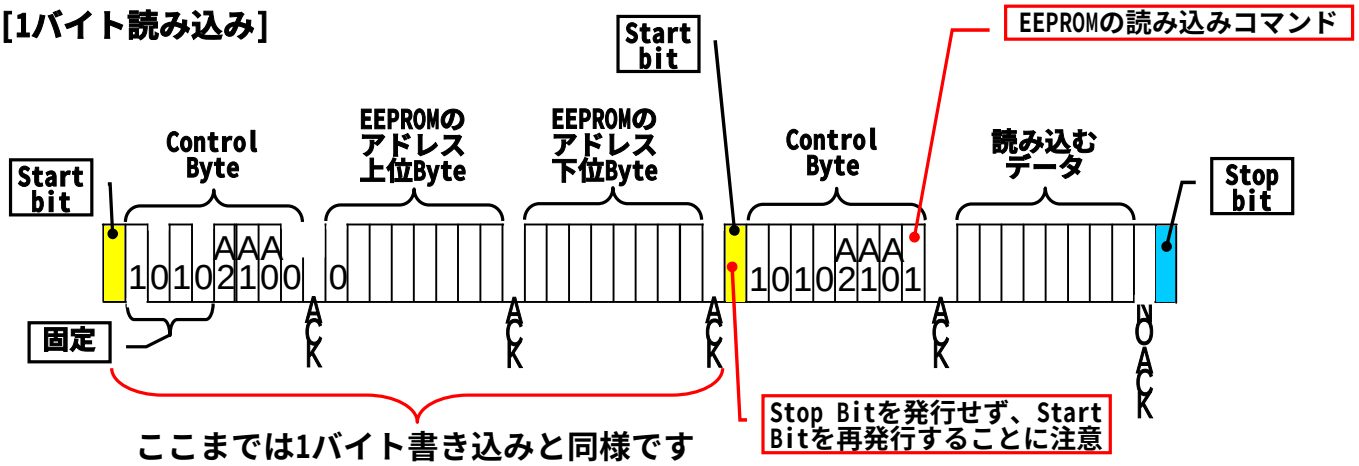
[ページ書き込み]



※2 1度に書き込めるデータ量(n)はEEPROMサイズによって変わります。概ね8～64Byteですが、詳細はデータシートを参照してください。

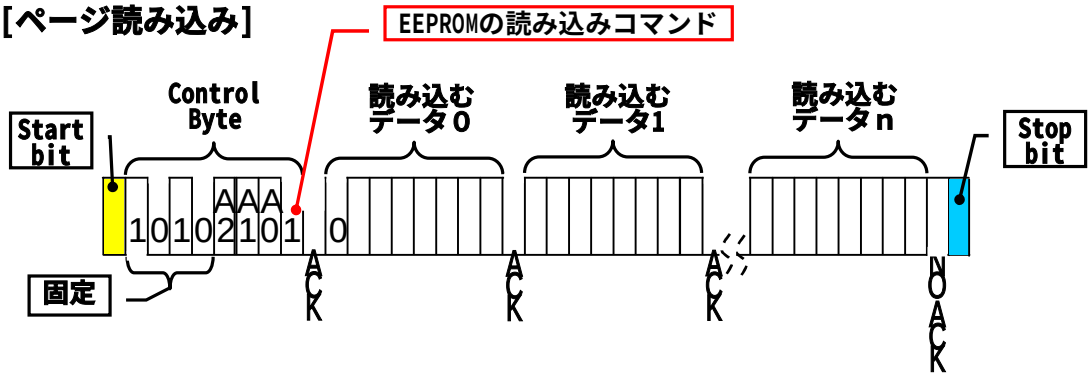
外部へデータ保存

[1バイト読み込み]



ここまでは1バイト書き込みと同様です

[ページ読み込み]



ページ読み込みは読み込むEEPROMのアドレスを指定できません。必ず1バイト読み込みのコマンド実行してアドレス指定後にページ読み込みを行ってください。1バイト読み込みで指定したアドレスは、読み込み後に+1されています。

ワンポイント

EEPROMの書き込み速度について

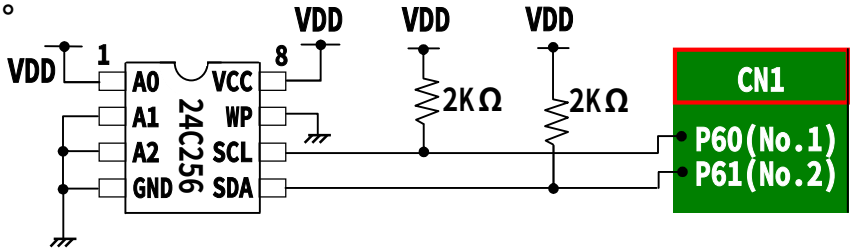
I2C EEPROMでは一般的に書き込みコマンド発行後、処理が終わるまでに5msecほどかかります。(詳細は各社のデータシートを参照してください)。5msecという時間はマイコンの処理の中では、非常に長い時間です。リアルタイムに処理するプログラムの場合には、この待ち時間を考慮しなければなりません。今回のプログラムの場合は、リアルタイムに処理する必要はないためI2C EEPROMへのコマンド発行後に、その場で時間待ち処理を行っています。

外部へデータ保存

I2C EEPROMをベースボードへ配線します。なお、ICソケットを使用しておくと後でI2C EEPROMサイズの変更が容易になります。使う部品はEEPROMとプルアップする抵抗だけですので簡単です。
サンプルプログラムとしてEEPROMへ単純に書き込みと読み込みが可能がどうかをテストします。
QB-78K0RKG3-TB上のSW(INTP0)を押下した時にテストを行うようにします。

9.I2C EEPROMの取り付け

回路は簡単です。下図を参考に配線してください。78K0RのSCL/SDAはN-chオープンドレインになっており、共にプルアップが必要です。EEPROM(24C256)のWPはWriteProtectを表します。WPをプルアップするとWriteProtectが有効になり、EEPROMへ書き込みできなくなるのでプルダウン(GNDへ接続)します。



組み立て順	部品の名称	型番	個数	備考
9	I2C EEPROM	24C256	1	互換品でも可
	抵抗	2KΩ	2	

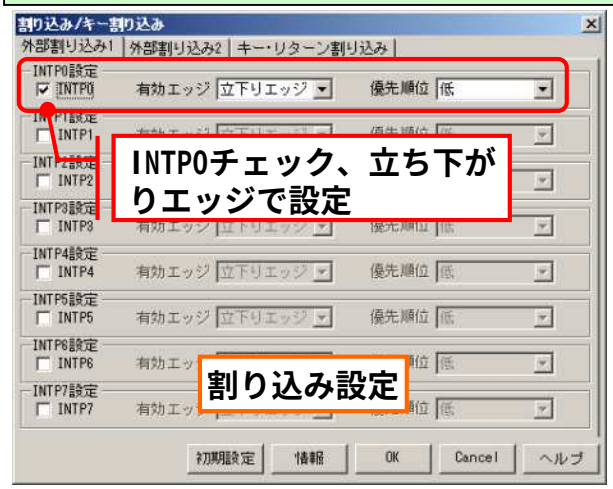
外部へデータ保存

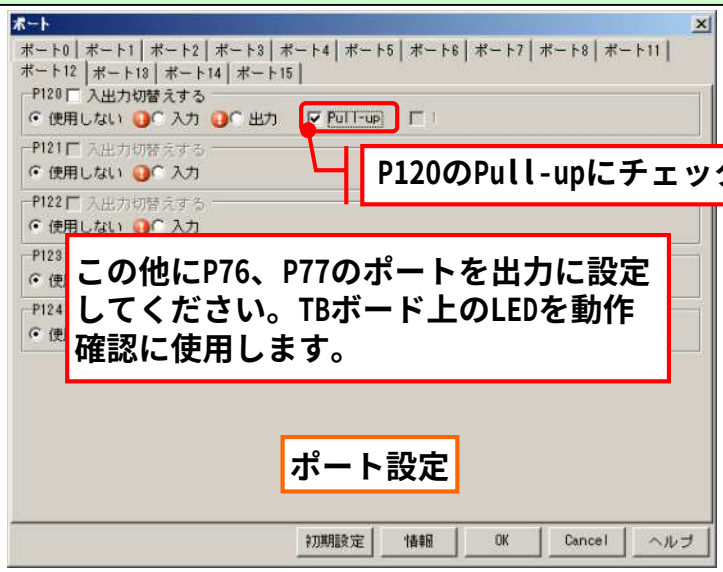
45:00

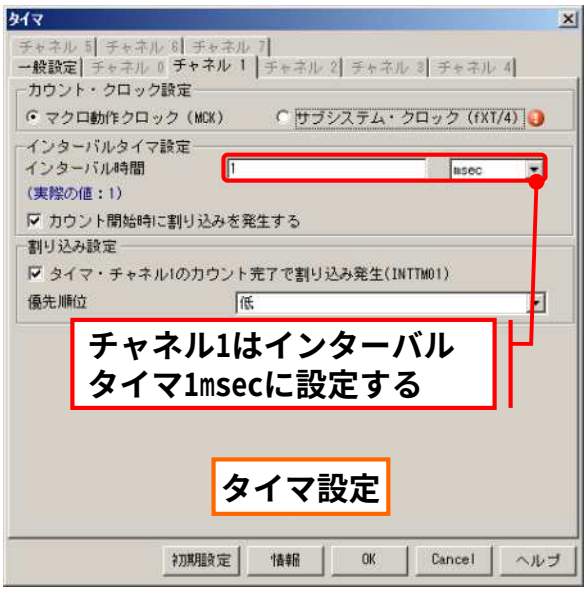
ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。TBボード上のSW(INTP0)が押下された時にプログラムを実行するようにします。

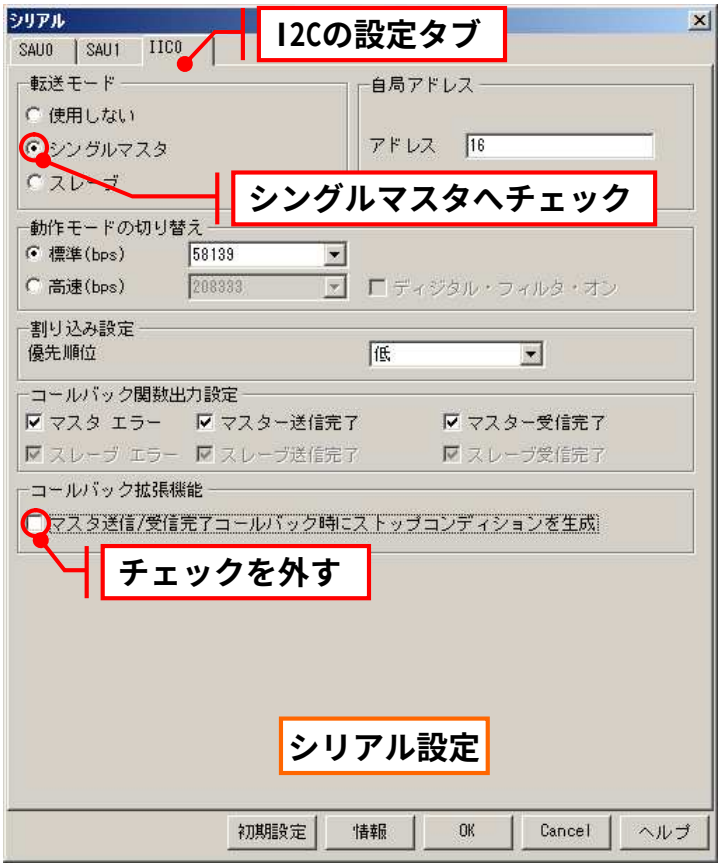
a. Applilet2を起動し、デバイス、プロジェクト名、システム設定します。
「準備(2)」を参照してマイクロコントローラ名、デバイス名「uPD78F1166_A0」、CPUクロック20MHz、オンチップ･デバッグ、ウォッチドッグ･タイマを設定します。プロジェクト名は「EEPROM_test」にします。

b.「割り込み」、「タイマ」、「シリアル」を設定します。
INTP0有効、P120プルアップ、タイマチャネル1msecインターバルタイマ、IIC0の設定をします。









外部へデータ保存

c.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
TAU.c /h	タイマ処理。
TAU_user.c	ユーザー処理用タイマ割り込みハンドラ定義処理。
Int.c /h	外部割り込み処理。
Int_user.c	外部割り込みハンドラ定義処理。
Serial.c /h	シリアル処理。
Serial_user.c	シリアル処理用割り込みハンドラ。
ctl_i2c.c	EEPROM制御用ソースファイル。新規に作成します。

上記のソースファイルで赤枠で示したのが「外部へデータ保存」で編集対象のファイルです。次ページより編集するソースを説明します。

外部へデータ保存

d.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

user_define.h

```
            リスト省略
#ifndef _MD_USER_DEF_
#define _MD_USER_DEF_
/*********************************
**  Macro define
*********************************************************************************/
/* Start user code for definition. Do not edit comment generated here */
/* メインで処理を行うイベントコード */
#define D_EV_INTP0          0x0000001   /* INTP0押下(処理中)    */

/* I2Cステータス定義 */
#define D_I2C_OFF           0
#define D_I2C_WRITE         1           /* write mode (EEPROM データ書き込み)     */
#define D_I2C_READSET       2           /* read set mode (EEPROM アドレス設定)    */
#define D_I2C_READDATA      3           /* read data mode (EEPROM データ呼び出し) */

/* EEPROM関係の定義 */
#define D_EEPROM_CODE       0xA2        /* コントロールバイト値、EEPROMのデバイスアドレスは1に設定している  */
#define D_EEPROM_BUF        20          /* EEPROMを読み込む際のバッファ(書き込みバッファとしても利用)       */
#define D_EEPROM_TIMEOUT    8192        /* EEPROMのタイムアウト時間                                         */
#define D_EEPROM_ACCESSSIZE 16          /* EEPROMを読み書きする際のサイズ                                   */

/* in main.c */
void wait1msec( UINT para1_ );

/* in ctl_i2c.c */
void i2c_initvalue( void );
MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *p_sadrs_, UINT length_ );
MD_STATUS i2c_randomread( USHORT dataadrs_, UCHAR *p_sadrs_ );
MD_STATUS i2c_read( UCHAR *p_sadrs_, UINT length_ );

/* in Serial.c */
MD_STATUS IICO_MasterReceiveStartRandom( UCHAR adr, UCHAR* rxbuf, UINT rxnum, UCHAR wait );

#include "String.h"
#include "Stdlib.h"
#include "Stdio.h"
/* End user code for definition. Do not edit comment generated here */
#endif
```

プログラム内で動作を判断する時のモードを定義しています。

ワンポイント

#ifndef ～ #endifについて

ヘッダファイルではよくこのような書き方をします。これは、ヘッダファイルが多重にincludeされていても、定義を繰り返さないように処理しているのです。上記の場合”_MD_USER_DEF_”が定義されていなければ”_MD_USER_DEF_”を定義してファイルの終わりにある#endifまで有効となるようにしています。ですので、仮に下記のような記述でもエラーにならないのです。また、定義名としてはユニークな名前をつける必要があるため”_ファイル名_”のような名前をつける場合が多いです。

例・・・

```
#include <user_define.h>
#include <user.h>
      :
      :
                    [main.c]
```

```
#ifndef _MD_USER_DEF_
#define _MD_USER_DEF_
      :
      :
#endif     [user_define.h]
```

```
#include <user_define.h>
      :
      :
                    [user.h]
```

user.hで#include<user_define.h>定義されているので二度、読込まれる事になる

外部へデータ保存

main.c(リスト1)

```
                    リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
UINT   g1msecCounterWait;                   /* Interval timer 1msec for wait function */
ULONG  gEventflag;                          /* イベントをチェックするフラグ           */
UCHAR  gData;                               /* EEPROMへ書き込むデータ                 */
UCHAR  gEepromWriteBuf[ D_EEPROM_BUF ];     /* EEPROMへ書き込む際のバッファ           */
USHORT gEepromWriteAdrs;                    /* EEPROMへ書き込むアドレス               */
UCHAR  gEepromReadBuf[ D_EEPROM_BUF ];      /* EEPROMへ読み込み際のバッファ           */
USHORT gEepromReadAdrs;                     /* EEPROMへ読込むアドレス                 */
/* End user code for global definition. Do not edit comment generated here */
                    リスト省略
void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    MD_STATUS retflag;
    int i;

    P7.6 = 1;                   /* TBボード上のLEDを消灯 */
    P7.7 = 1;
    i2c_initvalue();
    gEepromWriteAdrs = 0;
    gEepromReadAdrs = 0;
    g1msecCounterWait = 0;
    gEventflag = 0;
    gData = 0x11;               /* 初期データの値 */
    TAU_Channel1_Start();

    while (1)
    {
        if ( ( gEventflag & D_EV_INTP0 ) != 0 )
        {
            P7.6 = 0;           /* TBボード上のLEDを点灯 */
            P7.7 = 0;

            /* EEPROMへ書き込むデータの設定、検証データの設定 */
            memset( gEepromWriteBuf, 0x00, sizeof(gEepromWriteBuf) );
            memset( gEepromWriteBuf, gData, D_EEPROM_ACCESSSIZE );
            memset( gEepromReadBuf, 0x00, sizeof(gEepromReadBuf) );

            /* I2C EEPROMへデータを書き込み、5msec時間待ちを行う */
            retflag = i2c_write( gEepromWriteAdrs, gEepromWriteBuf, D_EEPROM_ACCESSSIZE );
            wait1msec( 5 );

            /* I2C EEPROMよりデータを読み込み、5msec時間待ちを行う */
            retflag = i2c_randomread( gEepromReadAdrs, gEepromReadBuf );
            wait1msec( 5 );
            retflag = i2c_read( &gEepromReadBuf[ 1 ], (D_EEPROM_ACCESSSIZE - 1) );
            wait1msec( 5 );

            for ( i = 0; i < D_EEPROM_ACCESSSIZE; i++ )
            {
                if ( gEepromWriteBuf[ i ] != gEepromReadBuf[ i ] )
                    break;
            }
            if ( i == D_EEPROM_ACCESSSIZE )
            {
                P7.6 = 1;       /* TBボード上のLEDを消灯 */
                P7.7 = 1;
            }
            gEventflag = 0;
            gData += 0x11;
            gEepromReadAdrs += D_EEPROM_ACCESSSIZE;
            gEepromWriteAdrs += D_EEPROM_ACCESSSIZE;
        }
    }
```

EEPROMに書き込まれたかを検証するためのデータ

EEPROMの0番地から、0x11を16バイト書き込む。

EEPROMの0番地から16バイトgEepromReadBufへ読み込む。

EEPROMへ書き込んだデータと読み込んだデータが異なっている場合TBボード上のLEDが点灯したままになる。

外部へデータ保存

main.c(リスト2)

```
    /* End user code. Do not edit comment generated here */
}
/* Start adding user code. Do not edit comment generated here */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 1msec単位の時間待ち                                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void wait1msec( UINT para1_ )
{

    /* 1msec の時間待ちを para1_ 分だけ行う */
    g1msecCounterWait = 0;
    while ( g1msecCounterWait < para1_ ) ;

}
/* End user code adding. Do not edit comment generated here */
```

Int_user.c

```
                        リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
extern ULONG  gEventflag;
/* End user code for global definition. Do not edit comment generated here */

/*---------------------------------------------------------------------------------
**    Abstract:         This function is INTP0 interrupt service routine.
**    Parameters:       None
**    Returns:          None
**---------------------------------------------------------------------------------*/
__interrupt void MD_INTP0( void )
{
    /* Start user code. Do not edit comment generated here */
    if ( ( gEventflag & D_EV_INTP0 ) == 0 )
    {
        gEventflag |= D_EV_INTP0;
    }
    /* End user code. Do not edit comment generated here */
}
```

TAU_user.c

```
                        リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
extern UINT    g1msecCounterWait;
/* End user code for global definition. Do not edit comment generated here */

/*---------------------------------------------------------------------------------
**    Abstract:         This function is INTTM01 interrupt service routine.
**    Parameters:       None
**    Returns:          None
**---------------------------------------------------------------------------------*/
__interrupt void MD_INTTM01( void )
{
    /* Start user code. Do not edit comment generated here */
    g1msecCounterWait++;
    /* End user code. Do not edit comment generated here */
}
```

外部へデータ保存

Serial.c

リスト省略(ファイルの最後に書きます。)

```
                    •
                    •
                    •
/* Start adding user code. Do not edit comment generated here */

MD_STATUS IIC0_MasterReceiveStartRandom( UCHAR adr, UCHAR* rxbuf, UINT rxnum, UCHAR wait )
{

    STT0 = 1;           /* IIC0 start condition */

    /* wait */
    while( wait-- );

    /* set parameter */
    gIic0RxLen = rxnum;
    gIic0RxCnt = 0;
    gpIic0RxAddress = rxbuf;

    gIic0MasterStatusFlag  = IIC0_MASTER_FLAG_CLEAR;
    adr |= 0x01;        /* receive mode */
    IIC0 = adr;         /* receive address */

    return MD_OK;
}

/* End user code adding. Do not edit comment generated here */
```

Serial.cに含まれる関数
MD_STATUS IIC0_MasterReceiveStartIIC0_MasterReceiveStart
上記の関数から抜粋して、ストップコンディションを生成しないでI2Cの送信するような関数を作成します。

Serial_user.c(リスト1)

リスト省略

```
/***********************************************************************************
**  Include files
***********************************************************************************/
#include "macrodriver.h"
#include "Serial.h"
/* Start user code for include definition. Do not edit comment generated here */

/* in ctl_i2c.c */
extern MD_STATUS i2c_eepromread( USHORT datano_ );

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/***********************************************************************************
**  Global define
***********************************************************************************/
extern                  volatile UCHAR          gIic0MasterStatusFlag;
extern                  volatile UCHAR          gIic0SlaveStatusFlag;
extern                  volatile USHORT         gIic0TxCnt;
extern                  volatile UCHAR*         gpIic0TxAddress;
extern                  volatile UCHAR*         gpIic0RxAddress;
extern                  volatile USHORT         gIic0RxCnt;
extern                  volatile USHORT         gIic0RxLen;
/* Start user code for global definition. Do not edit comment generated here */

/* in ctl_i2c.c */
extern UCHAR   gI2cStatus;
extern MD_STATUS gI2cErrorflag;
extern UCHAR   gI2cReadBuffer[ D_EEPROM_BUF ];

/* End user code for global definition. Do not edit comment generated here */
```

外部へデータ保存

Serial_user.c(リスト2)

```
                    リスト省略
/*
**-----------------------------------------------------------------------------
**  Abstract:          This function callback function open for users operation when IIC0 master error.
**  Parameters:        flag:                status flag
**  Returns:           None
**-----------------------------------------------------------------------------*/
void CALL_IIC0_MasterError( MD_STATUS flag  )
{
    /* Start user code. Do not edit comment generated here */

    gI2cErrorflag = flag;
    gI2cStatus = D_I2C_OFF;

    /* End user code. Do not edit comment generated here */
}

/*
**-----------------------------------------------------------------------------
**  Abstract:          This function callback function open for users operation when IIC0 master receive finish.
**  Parameters:        None
**  Returns:           None
**-----------------------------------------------------------------------------*/
void CALL_IIC0_MasterReceiveEnd( void )
{
    /* Start user code. Do not edit comment generated here */

    SPT0 = 1;
    gI2cErrorflag = MD_MASTER_RCV_END;
    gI2cStatus = D_I2C_OFF;

    /* End user code. Do not edit comment generated here */
}

/*
**-----------------------------------------------------------------------------
**  Abstract:          This function callback function open for users operation when IIC0 master transmit finish.
**  Parameters:        None
**  Returns:           None
**-----------------------------------------------------------------------------*/
void CALL_IIC0_MasterSendEnd( void )
{
    /* Start user code. Do not edit comment generated here */

    /* ストップコンディションの生成チェック */
    if ( gI2cStatus == D_I2C_WRITE )
    {   /* ストップコンディションの生成 */
        SPT0 = 1;
    }

    gI2cErrorflag = MD_MASTER_SEND_END;
    gI2cStatus = D_I2C_OFF;

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

ストップコンディションを生成してステータスをI2C通信終了します。

ステータスをチェックしてI2Cへの書き込みだったらストップコンディションを生成します。

外部へデータ保存

ctl_i2c.c(リスト1)

リスト省略

```
/*****************************************************************************
** Include files
*****************************************************************************/
#include "macrodriver.h"
#include "Serial.h"
#include "user_define.h"

/*****************************************************************************
**  Global define
*****************************************************************************/
UCHAR   gI2cStatus;                         /* I2C の通信ステータス                 */
UCHAR   gI2cBuffer[ D_EEPROM_BUF + 4 ];     /* I2C EEPROMへコマンドを送る際のバッファ */
USHORT  gI2cEepromWriteAdrs;                /* I2C EEPROMの書き込み実アドレス       */
USHORT  gI2cEepromReadAdrs;                 /* I2C EEPROMの読み込み実アドレス       */
MD_STATUS gI2cErrorflag;                    /* I2C 通信エラー時の状態               */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* i2c関連の変数を初期化                                                     */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void i2c_initvalue( void )
{
    gI2cStatus = D_I2C_OFF;
    gI2cErrorflag = MD_OK;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したバイト数、アドレスでEEPROMへ書き込みを行う                        */
/*                                                                           */
/* dataadrs_ :  EEPROMデータ格納アドレス                                     */
/* *p_sadrs_ :  書き込みデータ格納アドレスのポインタ                         */
/* length_   :  データ長                                                     */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *p_sadrs_, UINT length_ )
{
    UINT  cnt;
    MD_STATUS ret;

    memset( gI2cBuffer, 0x00, sizeof( gI2cBuffer ));
    gI2cEepromWriteAdrs = (USHORT)dataadrs_;

    gI2cBuffer[ 0 ] = (UCHAR)(( gI2cEepromWriteAdrs & 0xff00 ) >> 8);
    gI2cBuffer[ 1 ] = (UCHAR)(( gI2cEepromWriteAdrs & 0xff ));
    memcpy( &gI2cBuffer[ 2 ], p_sadrs_, length_ );
    gI2cStatus = D_I2C_WRITE;

    /* EEPROMのデバイスアドレスA0のみHIGH  */
    ret = IIC0_MasterSendStart( D_EEPROM_CODE, gI2cBuffer, (length_ + 2), 1 );
    gI2cErrorflag = ret;

    cnt = D_EEPROM_TIMEOUT;
    while ( gI2cStatus == D_I2C_WRITE )
    {
        cnt--;
        if ( cnt == 0 )
        {
            gI2cErrorflag = MD_RESOURCEERROR;
            ret = gI2cErrorflag;
            break;
        }
    };

    return ret;
}
```

EEPROMへ書き込み後、一定時間経過してもI2Cステータスが終了にならない場合は、書き込みエラーとします。

外部へデータ保存

ctl_i2c.c(リスト2)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したアドレスでEEPROMより1バイト読み込みを行う                             */
/*                                                                             */
/* *dataadrs_ :  EEPROMデータ格納アドレス                                       */
/*               読み込みを行うとEEPROMから読み出すアドレスも加算される         */
/* *p_sadrs_  :  読み込むデータ格納アドレスのポインタ                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_randomread( USHORT dataadrs_, UCHAR *p_sadrs_ )
{
    UINT cnt;
    MD_STATUS ret;

    gI2cEepromReadAdrs = (USHORT)dataadrs_;

    gI2cBuffer[ 0 ] = (UCHAR)(( gI2cEepromReadAdrs & 0xff00 ) >> 8);
    gI2cBuffer[ 1 ] = (UCHAR)(( gI2cEepromReadAdrs & 0xff ));

    ret = IICO_MasterSendStart( D_EEPROM_CODE, gI2cBuffer, 2, 1 );
    gI2cErrorflag = ret;
    gI2cStatus = D_I2C_READSET;
    if  ( ret == MD_OK )
    {
        cnt = D_EEPROM_TIMEOUT;
        while ( gI2cStatus == D_I2C_READSET )
        {
            cnt--;
            if ( cnt == 0 )
            {
                gI2cErrorflag = MD_RESOURCEERROR;
                ret = gI2cErrorflag;
                break;
            }
        };

        if ( !(gI2cErrorflag & MD_ERRORBASE) )
        {
            gI2cStatus = D_I2C_READDATA;

            ret = IICO_MasterReceiveStartRandom( D_EEPROM_CODE, p_sadrs_, 1, 1 );
            if  ( ret == MD_OK )
            {
                cnt = D_EEPROM_TIMEOUT;
                while ( gI2cStatus == D_I2C_READDATA )
                {
                    cnt--;
                    if ( cnt == 0 )
                    {
                        gI2cErrorflag = MD_RESOURCEERROR;
                        ret = gI2cErrorflag;
                        break;
                    }
                };
            }
        }
    }

    return ret;
}
```

EEPROMから読み込むアドレス設定後、一定時間経過してもI2Cステータスが終了にならない場合は、エラーとします。

EEPROMから1バイト読み込み後、一定時間経過してもI2Cステータスが終了にならない場合は、エラーとします。

外部へデータ保存

ctl_i2c.c(リスト3)

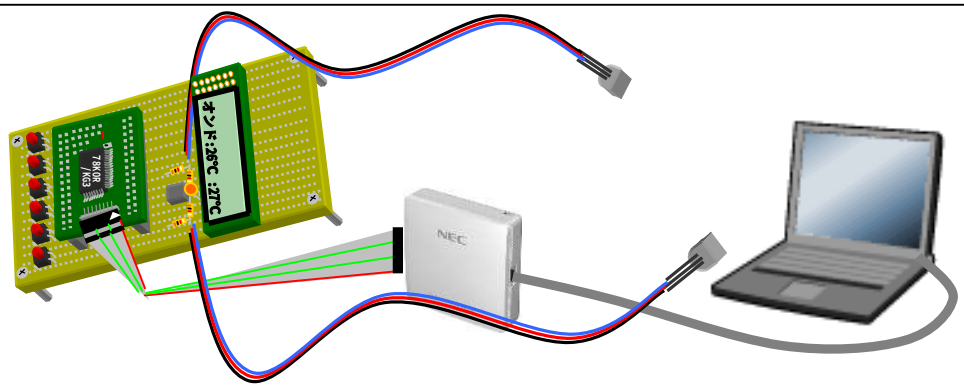
```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したバイト数をEEPROMより読み込みを行う                              */
/*                                                                       */
/* *p_sadrs_ :  読み込むデータ格納アドレスのポインタ                      */
/*              読み込みを行うとEEPROMから読み出すアドレスも加算される    */
/* length_   :  データ長                                                 */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_read( UCHAR *p_sadrs_, UINT length_ )
{
    UINT cnt;
    MD_STATUS ret;

    ret = IICO_MasterReceiveStart( D_EEPROM_CODE, p_sadrs_, length_, 1 );
    gI2cStatus = D_I2C_READDATA;
    if  ( ret == MD_OK )
    {
        cnt = D_EEPROM_TIMEOUT;
        while ( gI2cStatus == D_I2C_READDATA )
        {
            cnt--;
            if ( cnt == 0 )
            {
                gI2cErrorflag = MD_RESOURCEERROR;
                ret = gI2cErrorflag;
                break;
            }
        };
    }

    return ret;
}
```

EEPROMから読み込み後、一定時間経過してもI2Cステータスが終了にならない場合は、エラーとします。

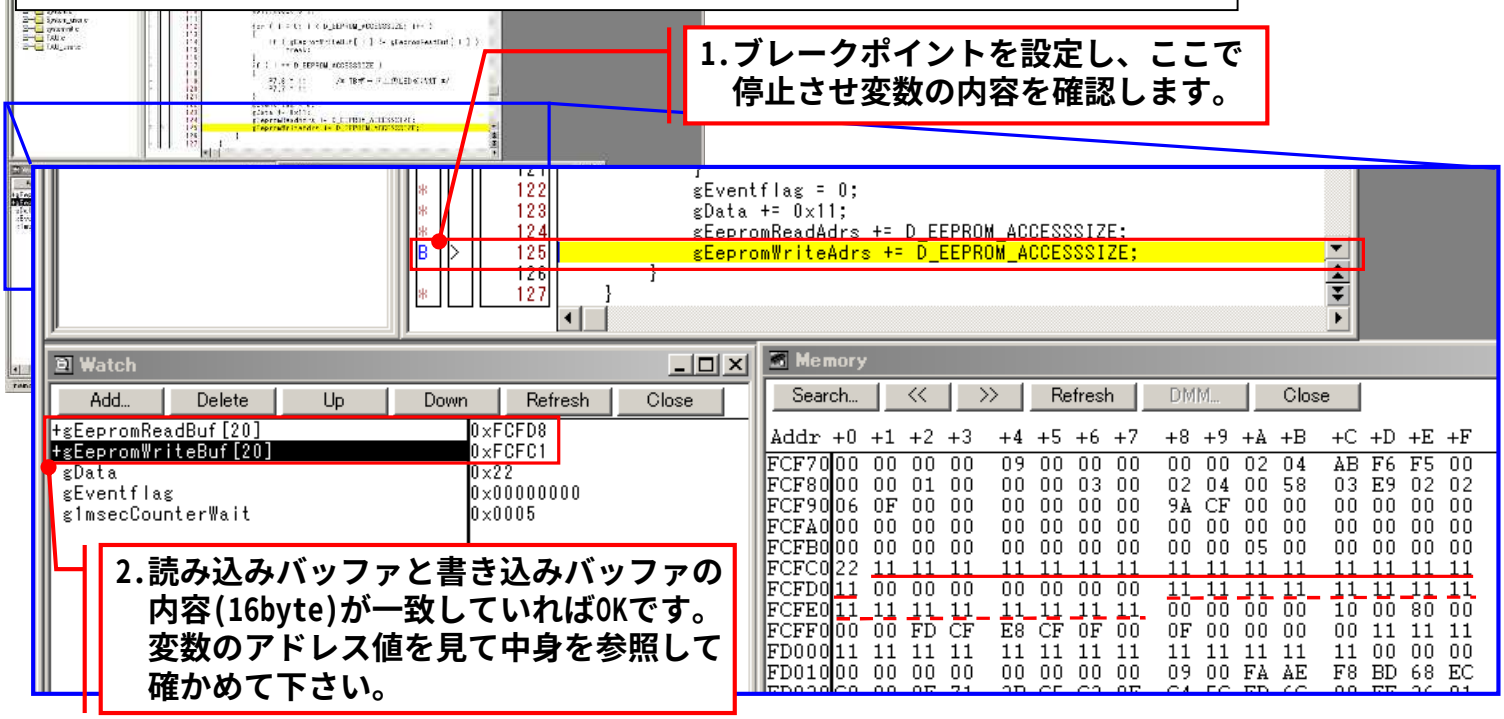
外部へデータ保存

e.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

1.「A/DでSW」と同様にMINICUBE2を設定し、QB-78K0RKG3-TB + ベースボードと接続した後にPC本体とつなげます。



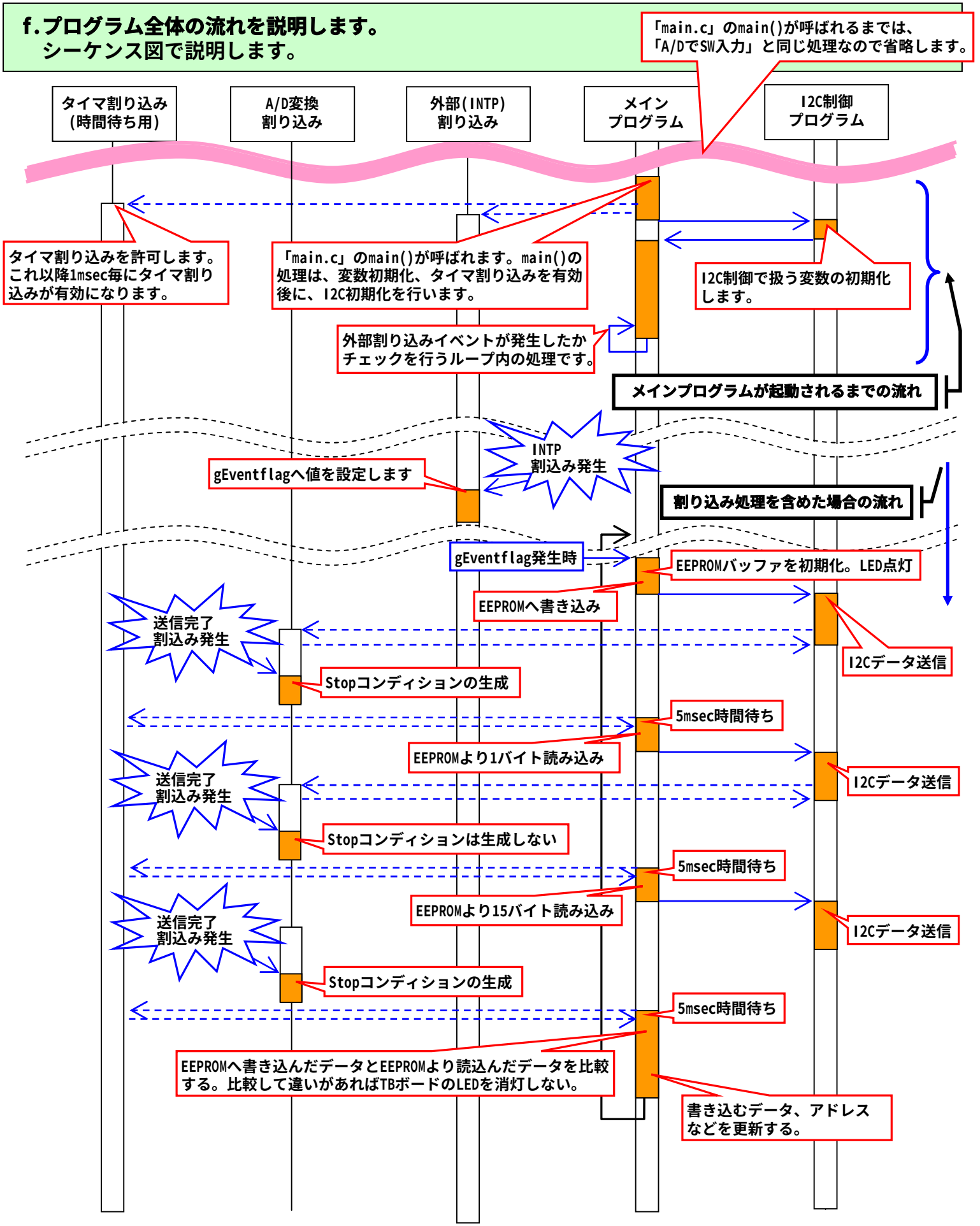
2.PM+を起動し、プログラムをビルドしてID78K0R-QB上で実行します。
変数の登録などは「温度を計測」の章で実行させた場合を参考にしてください。
TBボード上のSWを押下した時にEEPROMへの書き込みを行います。



3.またTBボード上のLEDにも注目してください。正常に書き込み/読み込みが行われていればLEDが一瞬点灯します。エラーがあればLEDが点灯したままになります。

外部へデータ保存

f.プログラム全体の流れを説明します。
シーケンス図で説明します。

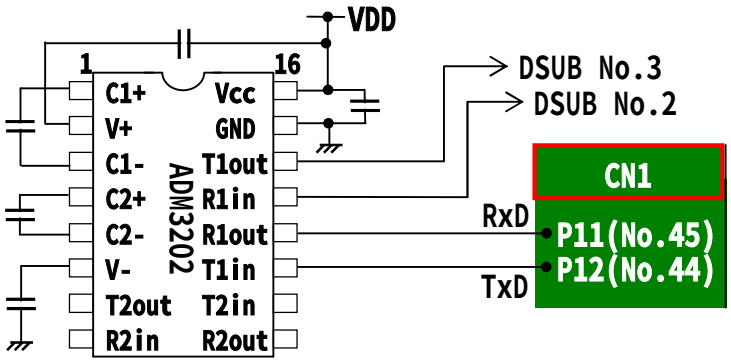


RS232Cで送信

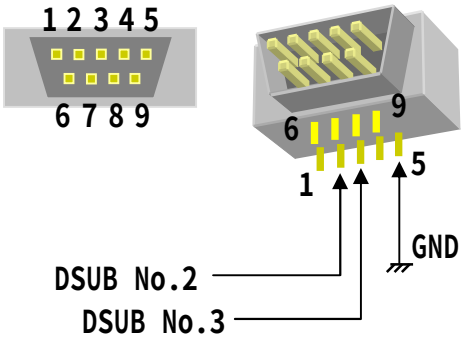
UART(RS232C)を使い、PCへデータを送ります。PC側ではターミナルソフトで動作の確認が可能です。シリアル送信/受信としてなるべく単純にするため、TxD、RxDのみを使用します。RS232Cの規格では、信号線に-5V～+15Vまで流れる場合があります(PCからの信号に限る。マイコンからは最大VDDまでの電圧しか出力しません)。PCからの信号をマイコンにそのまま接続するとマイコンが破壊される可能性があります。そのためADM3202というRS232Cのドライバ ICを使用します。

10.ADM3202の回路図

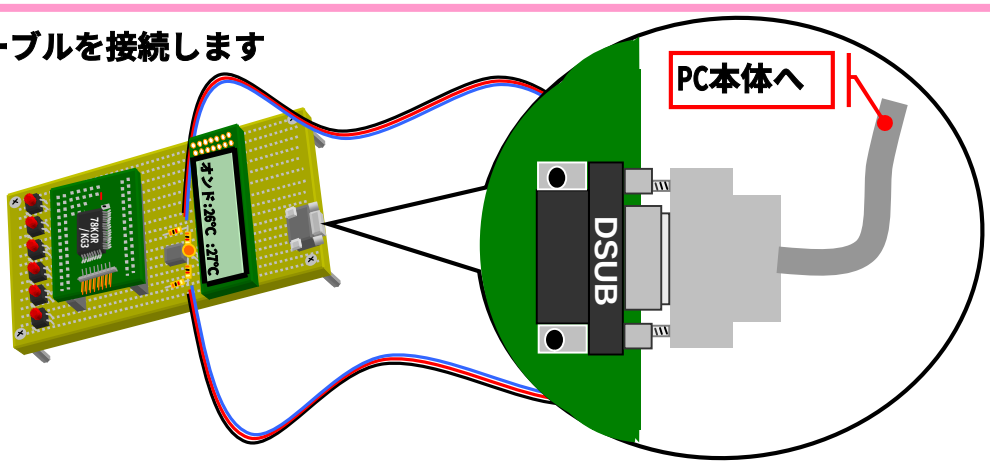
ADM3202をベースボードに配線します。コンデンサ、DSUB9ピンコネクタも配線します。ADM3202は2チャンネル制御可能ですが、ここでは1チャンネルのみ使用します。またDSUBコネクタにはNo.2,No.3の他にNo.5(GND)も接続してください。



※コンデンサは全て0.1μF



11.RS232Cクロスケーブルを接続します



組み立て順	部品の名称	型番	個数	備考
10	ADM3202		1	互換品でも可
	コンデンサ	0.1uF	5	セラミックで可
	DSUB9ピンコネクタ		1	オス
11	シリアルケーブル		1	9ピンクロス

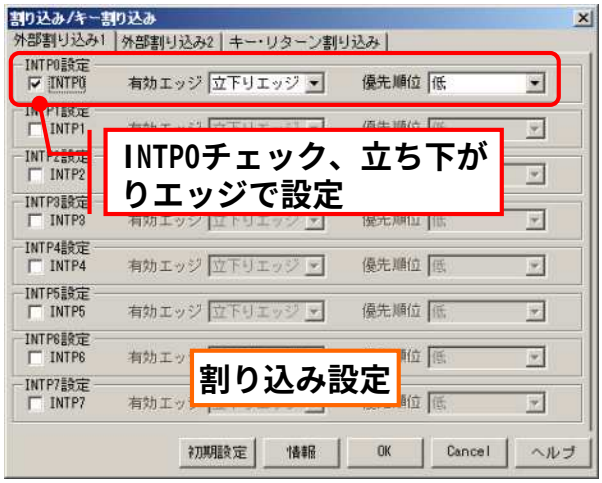
RS232Cで送信

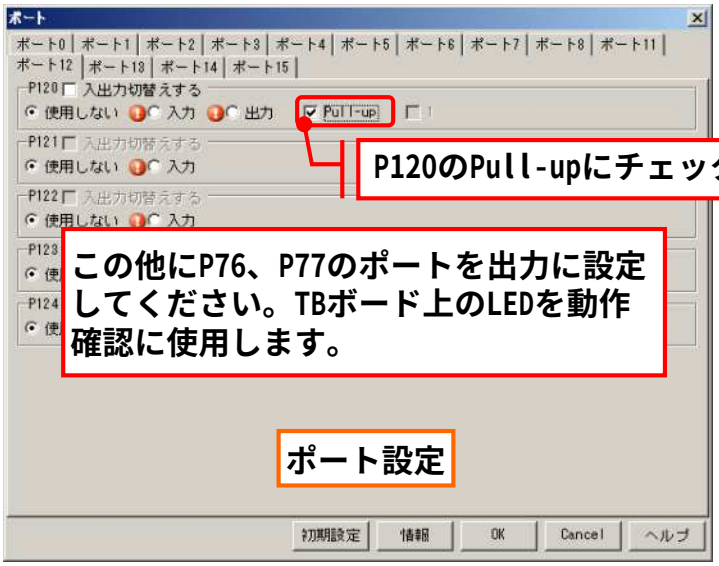
30:00

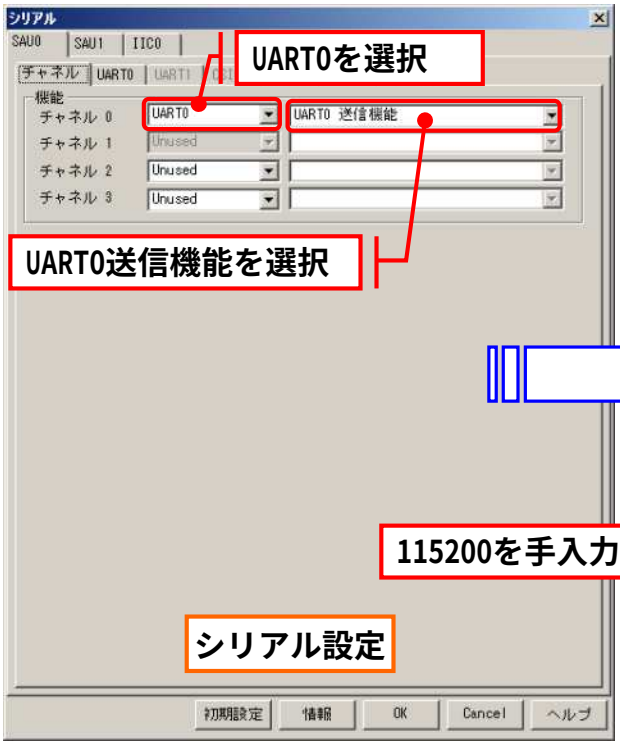
ハードウエアの部分が完成したところで、次にソフトウエアの設定を行います。TBボード上のSW(INTP0)が押下された時にPCへデータ送信を行うようにプログラムを作成します。

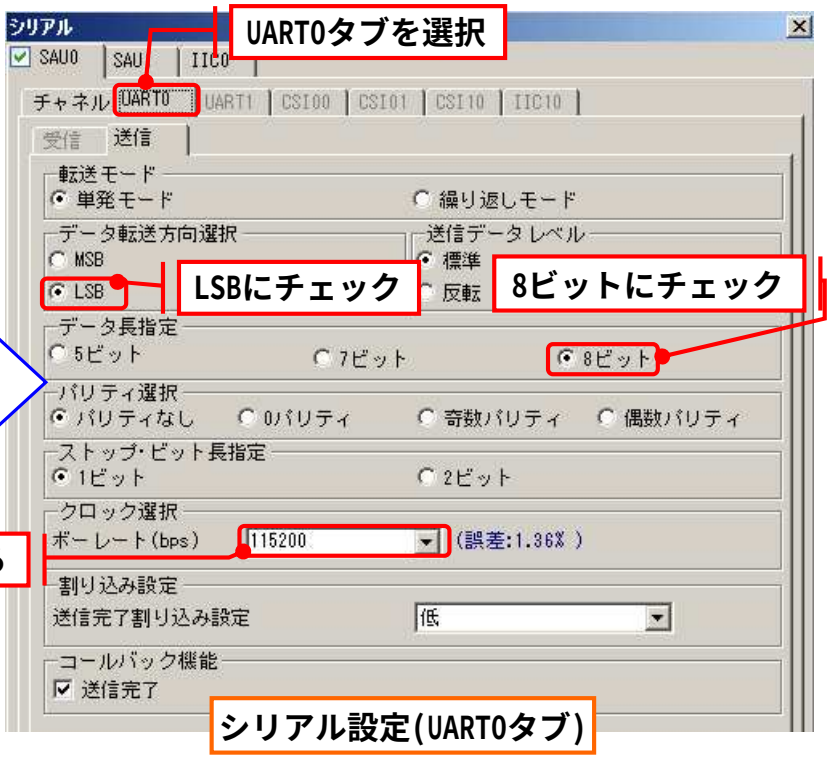
a. Applilet2を起動し、デバイス、プロジェクト名、システム設定します。
「準備(2)」を参照してマイクロコントローラ名、デバイス名「uPD78F1166_A0」、CPUクロック20MHz、オンチップ･デバッグ、ウォッチドッグ･タイマを設定します。プロジェクト名は「RS232C_test」にします。

b.「割り込み」、「シリアル」を設定します。
INTP0有効、P120プルアップ、シリアル(UART0)の設定をします。









RS232Cで送信

c.「GO」ボタンを押下してコード生成します。
以下に生成ソース一覧を説明します。

[ソース･ファイルの構成]

AD_swtest.prx	Applilet2用保存したファイル。

Applilet2が生成するソース一覧

AD_swtest.prw	PM+で使用する環境ファイル。
AD_swtest.prj	PM+で使用する環境ファイル。
lk.dr	リンクディレクティブ･ファイル(シンボル定義ファイル)。今回は未使用。
macrodriver.h	Applilet2用定数定義ソースファイル。
user_define.h	ユーザー追加用定義ソースファイル。
System.c /h	クロック設定ソースファイル。
System_user.c	ユーザー追加用初期化関数ソースファイル。
systeminit.c	各種周辺機能初期化ソースファイル。
main.c	メイン関数。
Int.c /h	外部割り込み処理。
Int_user.c	外部割り込みハンドラ定義処理。
Serial.c /h	シリアル処理。
Serial_user.c	シリアル処理用割り込みハンドラ。

上記のソースファイルで赤枠で示したのが「RS232Cで送信」で編集対象のファイルです。次ページより編集するソースを説明します。

RS232Cで送信

d.プログラムを追加します。下記に示す青字のコードを追加してください。
PM+ でプロジェクトを開きソースファイルを編集してください。

main.c

```
            リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------------
**  Abstract:           This function  implements main function.
**  Parameters:         None
**  Returns:            None
**-----------------------------------------------------------------------------
*/
void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    UART0_Start();
    while (1) {
      ;
    }
    /* End user code. Do not edit comment generated here */
}
```

UARTの送信準備します。

RS232Cで送信

Int_user.c

リスト省略

```
/***********************************************************************************
**  Include files
***********************************************************************************/
#include "macrodriver.h"
#include "Int.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "Serial.h"
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */

UCHAR gtData[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890\n";

/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------------
**  Abstract:          This function is INTP0 interrupt service routine.
**  Parameters:        None
**  Returns:           None
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTP0( void )
{
    /* Start user code. Do not edit comment generated here */
    USHORT len;

    len = sizeof( gtData );
    UART0_SendData( gtData, len );
    /* End user code. Do not edit comment generated here */
}
```

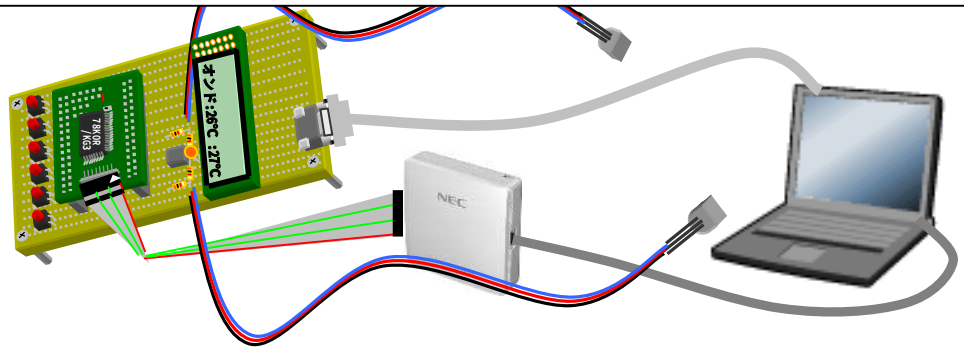
UART送信します。

Appliletを使用するとシリアルでPCへデータ送信するのが、非常に簡単だということがわかると思います。ただ注意点があります。現状のソースはボード上のSWを押下したとたんに送信するようになっていますが、本来はSWを押下したフラグまたはイベントを用意してメインプログラム内で処理すべきです。現状ではSWの連続押下された場合など、期待した処理を行えません。

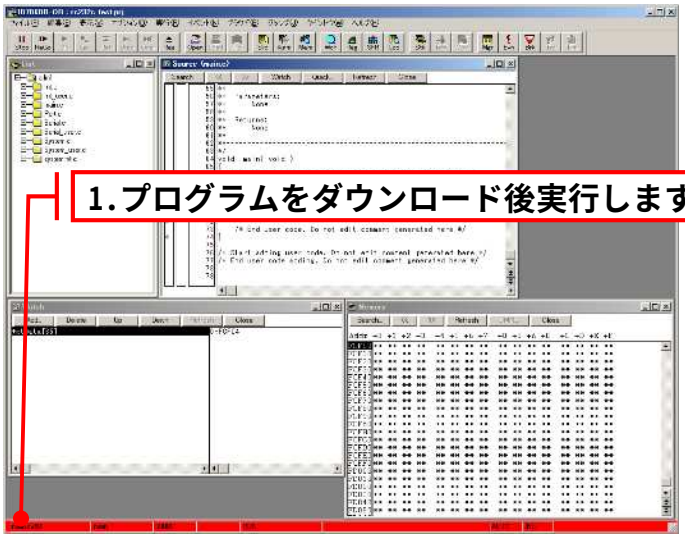
RS232Cで送信

e.プログラムをビルドして実行してみます。
QB-78K0RKG3-TB + ベースボードにMINICUBE2を接続してデバッガを起動します。

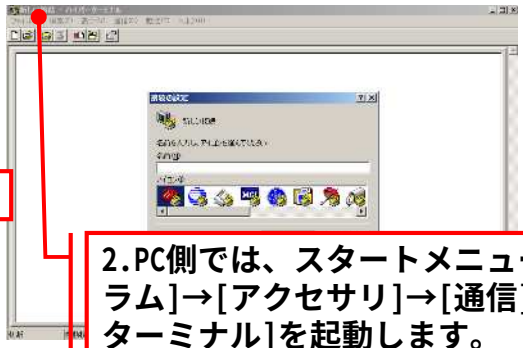
1.「A/DでSW」と同様にMINICUBE2を設定し、QB-78K0RKG3-TB + ベースボードと接続した後にPC本体とつなげます。また、シリアルクロスケーブルも接続します。



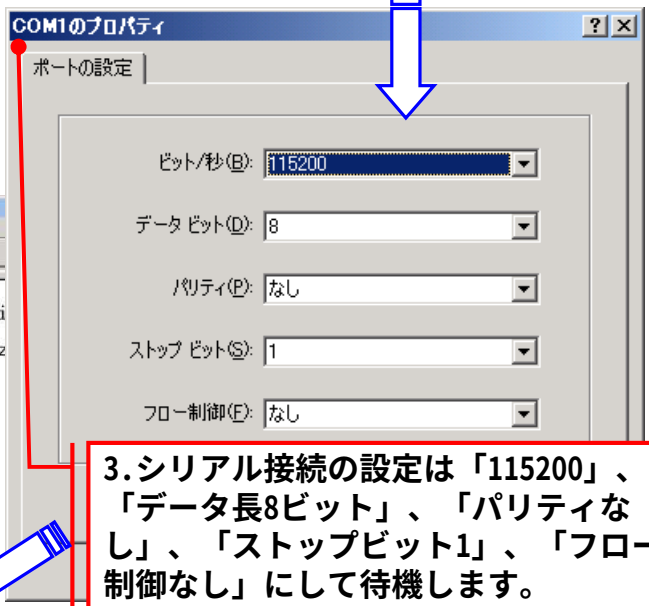
2.PM+を起動し、プログラムをビルドしてID78K0R-QB上で実行します。
TBボード上のSWを押下した時PCへデータ送信します。PC側ではターミナルソフト(ハイパーターミナル)を起動して受信可能状態にしておいてください。



1.プログラムをダウンロード後実行します。



2.PC側では、スタートメニューより[プログラム]→[アクセサリ]→[通信]→[ハイパーターミナル]を起動します。

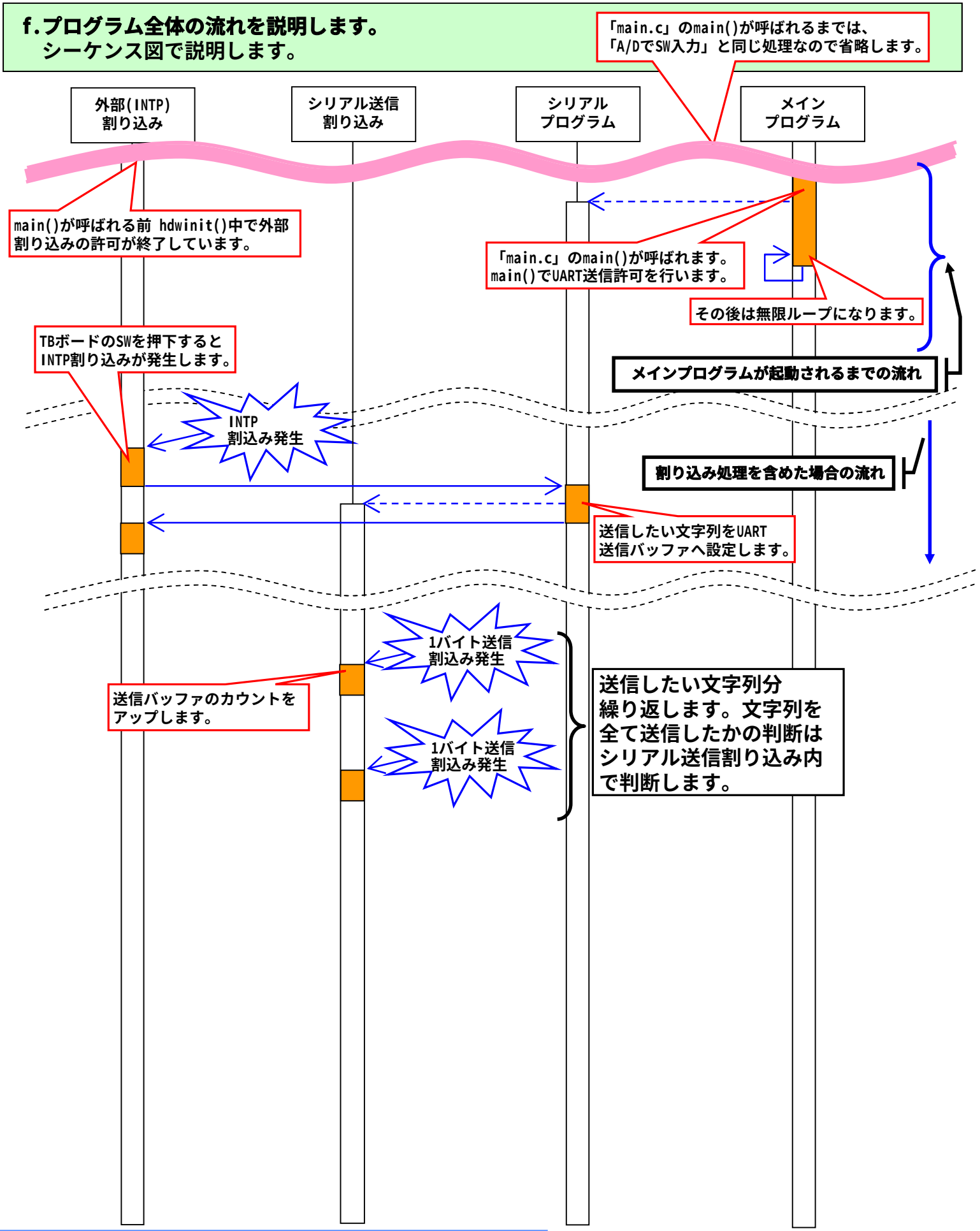


3.シリアル接続の設定は「115200」、「データ長8ビット」、「パリティなし」、「ストップビット1」、「フロー制御なし」にして待機します。

4.TBボード上のSWを押下する毎に文字列が表示されればOKです。

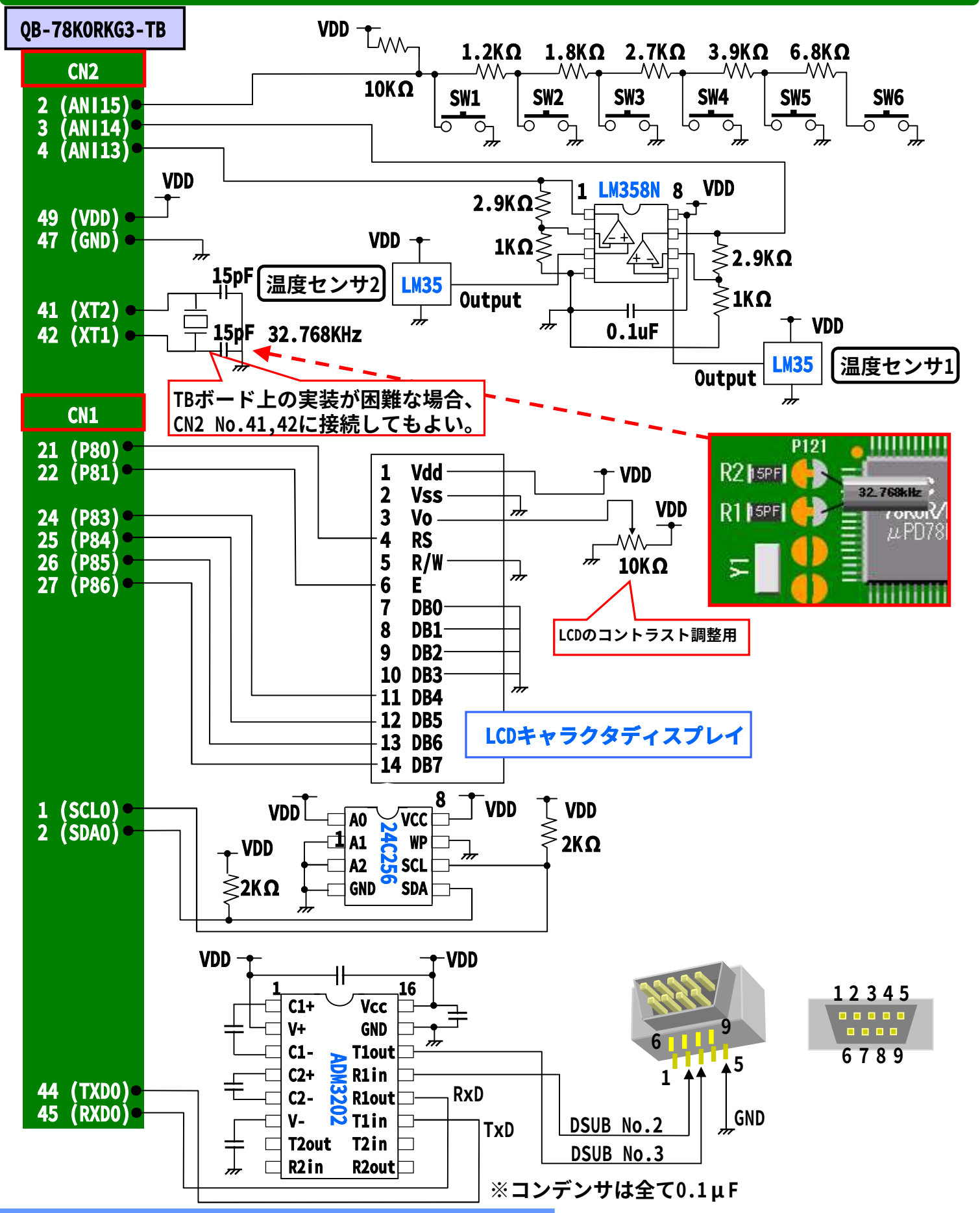
RS232Cで送信

f.プログラム全体の流れを説明します。
シーケンス図で説明します。



全体回路図

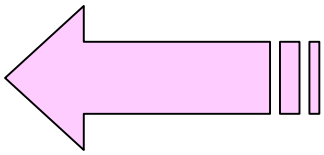
「時計表示機能つき温度データロガー」の全回路図です。



ソフトウエアの作成

この項では「時計表示機能つき温度データロガー」プログラムの作成を行います。作成だけでなくプログラムの動作なども詳細に説明します。この章を通して組み込みプログラムを理解します。

はじめに
製作するシステム
製作手順
全体回路図
ソフトウエアの作成
[組み込みプログラムとは]
[基本的なプログラム構成]
[本システムの操作]
[Appliletの設定]
[全ソースファイル説明]
[全体のフローチャート]
[プログラム]
次へのステップ
付録



「ソフトウエアの作成」章ではターゲットプログラムを作成するだけでなく、マイコンを用いた組み込みプログラムを作成するときの基本などを説明しています。またマイコンについて学習したい場合、弊社WEBも活用してください。

「All Flash 知れば知るほどゼミナール 」
http://www.necel.com/micro/ja/campaign/allflash-zemi/

「マイコンe-Learning」
http://www.necel.com/micro/ja/eLearning/

組み込みプログラムとは

組み込みプログラムとアプリケーション・プログラム（Windows/Unix/Linux上などのOS上で動作するプログラムとします)と大きく異なる点は電源ON～電源OFFまで全てを把握しなければならない事です。これは、一般的なアプリケーション・プログラムを作成するより難易度が高くなります。
組み込みプログラムには次のような特徴があります。

限りある資源の制約を考慮する

1つのCPUで全ての処理を行うため実行速度、RAM/ROMサイズなど一般のPCに比べて貧弱になります。限りある資源を十分に考慮する必要があります。

周辺装置の制御を行う

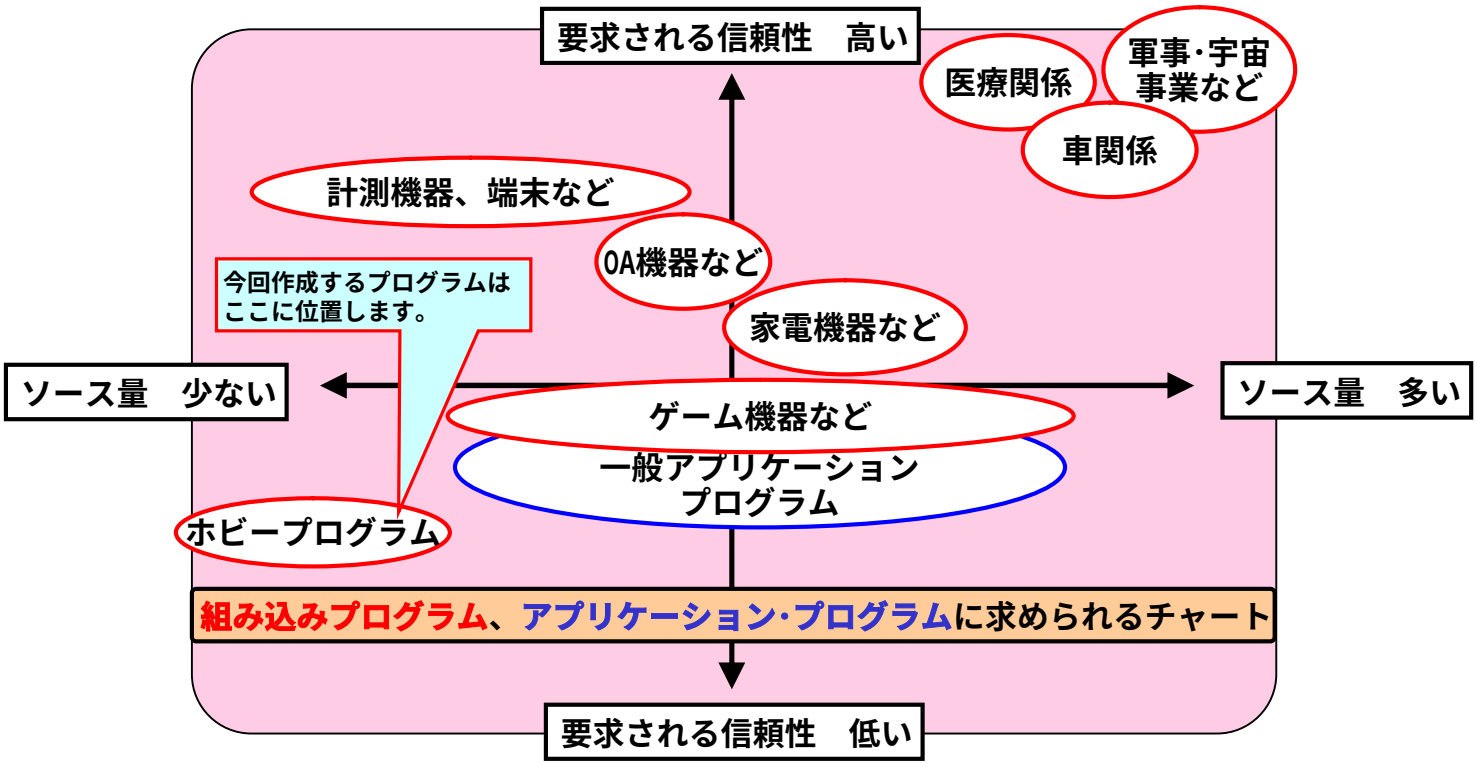
CPUに内蔵された周辺機能でハードウエアの制御も行います。一般のPCだと専用のハードウエアをスロットに搭載します。

要求された時間内に応答するリアルタイム性

リアルタイムとはシステムに要求された時間内に応答して動作しなければならないことを指します。数msec、数μsecで応答しなければならない場合や、モータ制御などは、もっと短い時間で制御する場合もあります。

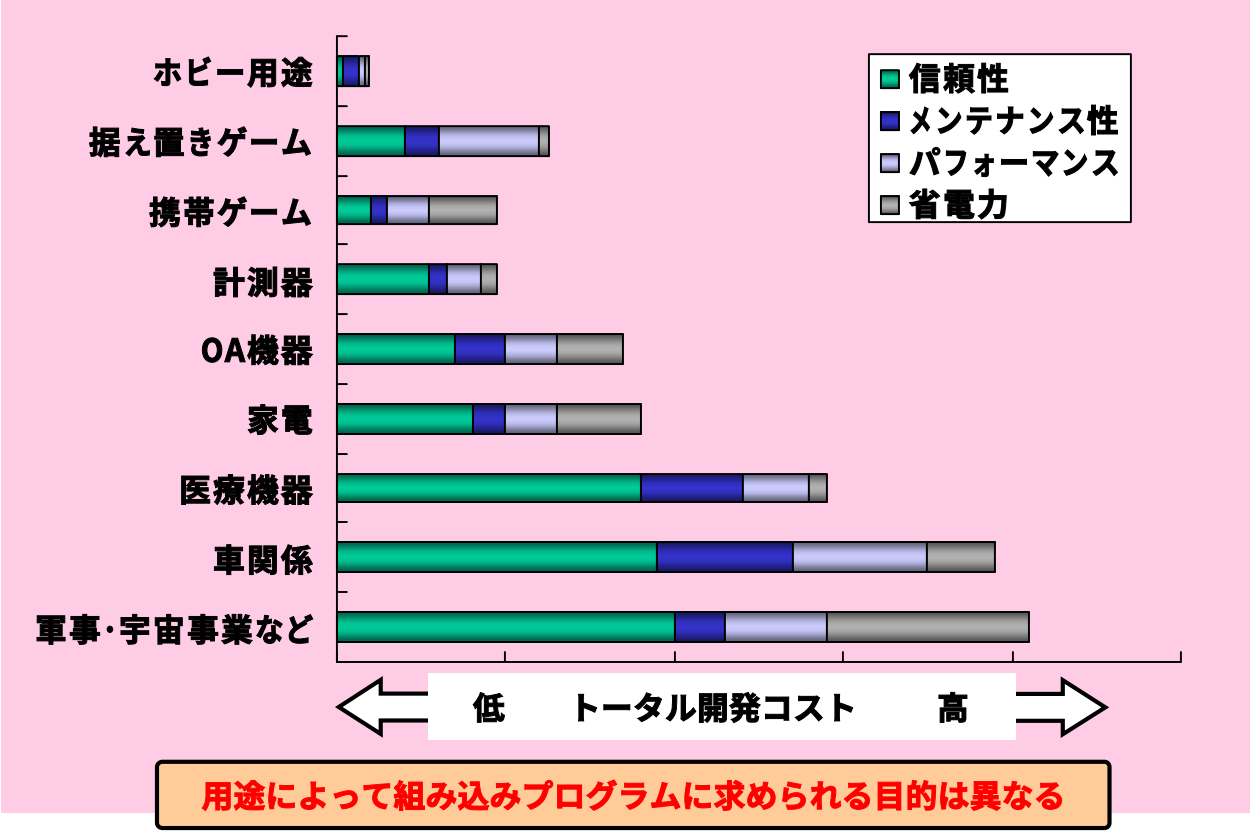
異常系の動作や例外処理にも対応する高い信頼性

例えば動作中に電源が落ちても保存中のデータが破壊されない、ノイズために通信エラーが発生してもリカバリできる、予期しない入力や割り込みなどが起きてもシステムダウンしない、など ありとあらゆる事を想定した処理が求められます。



組み込みプログラムとは

組み込みプログラムの作成方法は一つではありません。プログラムの目的によって求められる用途は変化します。下図は組み込みプログラムの用途によって求められる目的が異なることを説明しています。



- **ホビー用途**
  このガイドで作成するシステムもホビー用途です。ここでは、メンテナンス性を考慮しています。その他のパフォーマンス、省電力については考慮していません。ここでは基本的なプログラム構成に重点をおいて作成しました。

- **ゲーム**
  ゲームといっても携帯するか据え置き型で目的が異なります。携帯するゲームは省電力が最重要となり、据え置き型ではパフォーマンスが重要視されます。

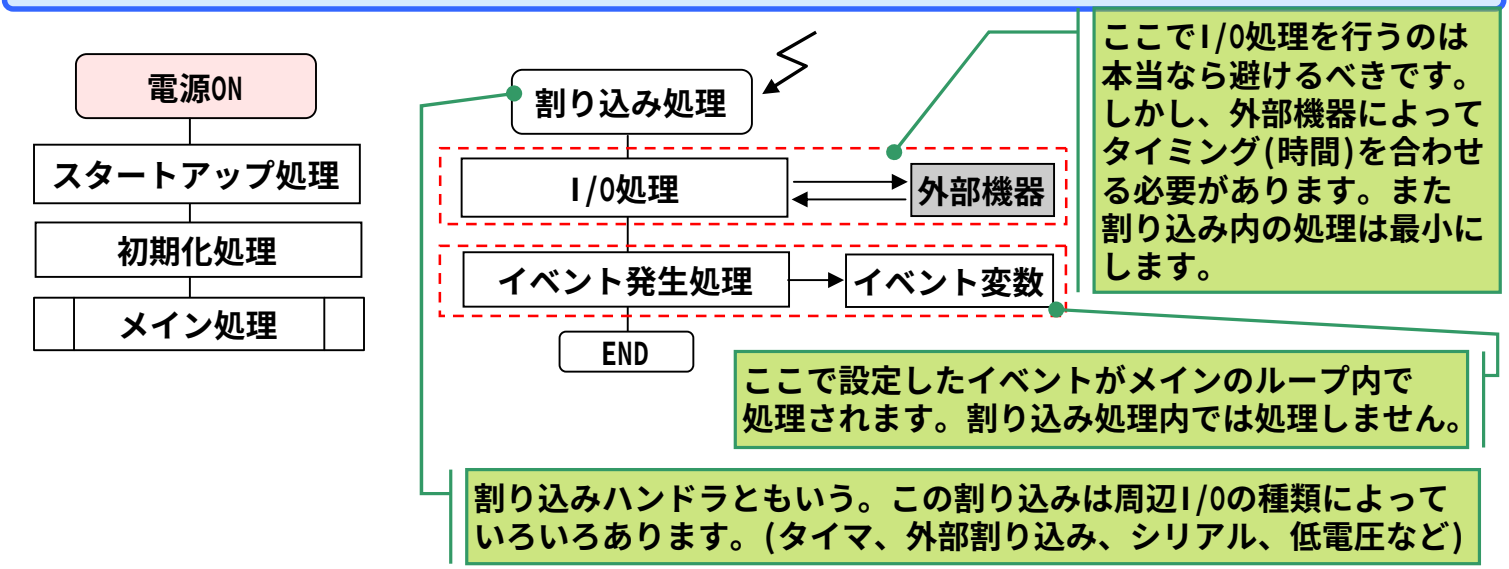
- **計測器、OA機器**
  信頼性が重要です。計測するものに間違いがあれば計測器そのものの存在意味がありません。ただ、種類によってはGHzのパフォーマンスも求められます。

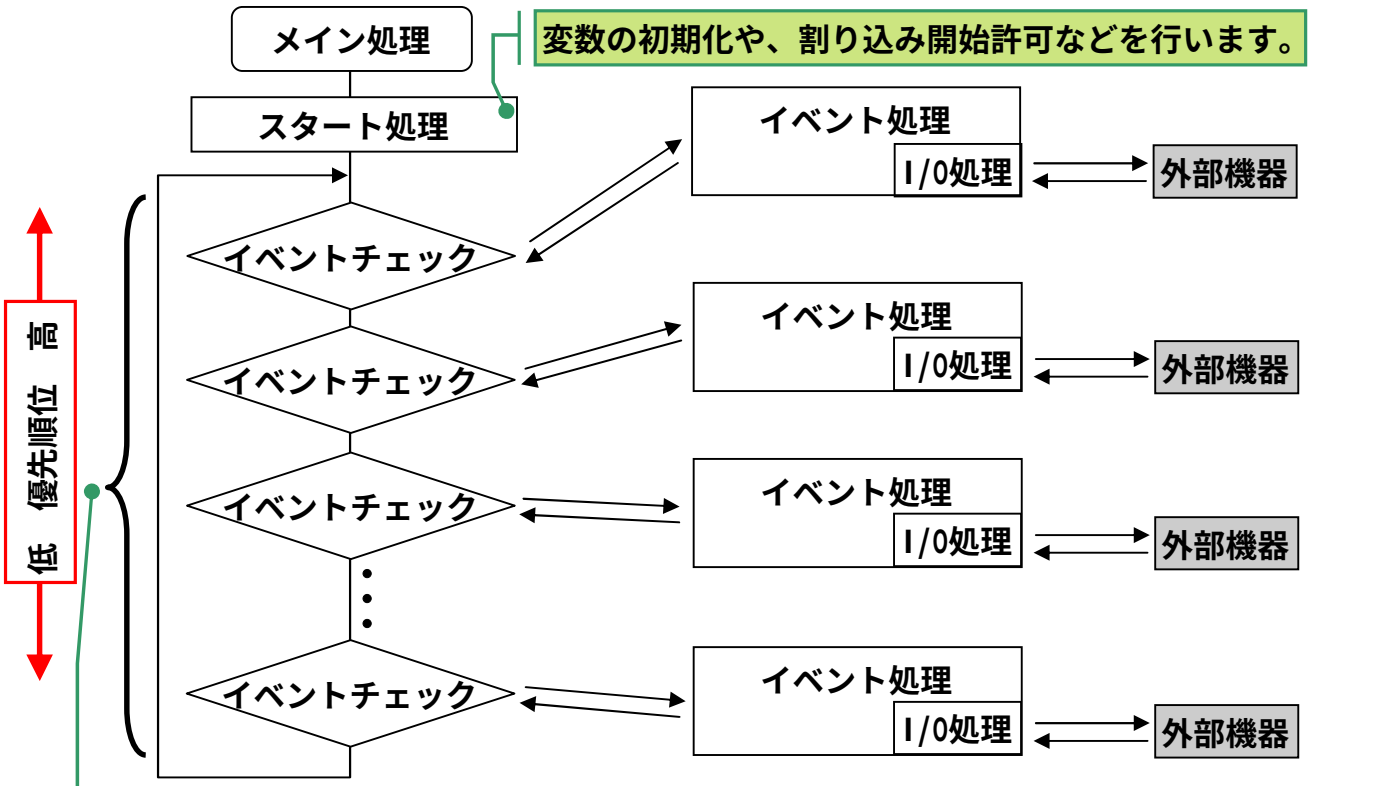
- **家電**
  家電といっても種類が豊富です。高性能を求められるもの、省電力、小型化、など用途に応じて異なります。電池を使うものは省電力、据え置きはパフォーマンスを求める傾向にあります。

- **医療機器/車/軍事･宇宙事業/原子力発電所制御など**
  この分野になると開発コストも増大になります。なぜなら人命にかかわってくるからです。また、開発規模も大きくなり、数百人単位となるでしょう。ここでの医療機器はMRI/PETなど数億円する医療機器を指します。

基本的なプログラムの構成

ここでは組み込みプログラムの基本構成を説明します。もちろん、このガイドで製作するシステムも基本構成に沿ってプログラム作成しています。以下、フローチャートで全体構成を説明します。





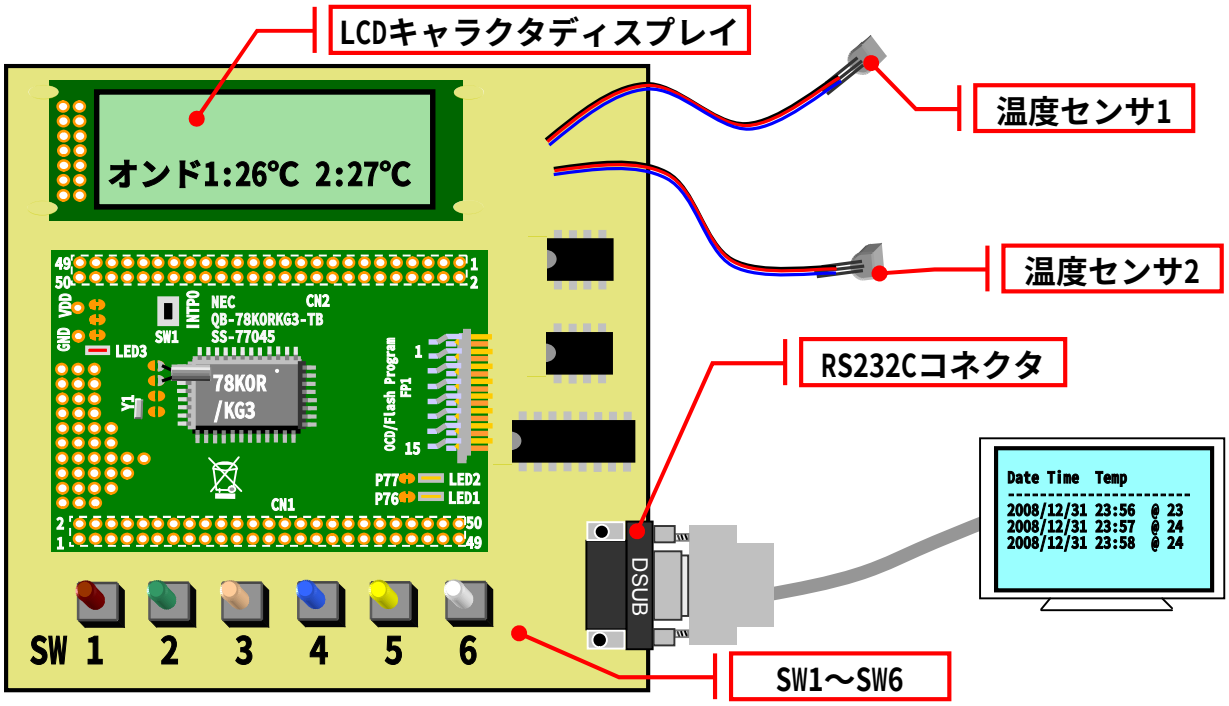
メイン処理内でイベントのチェックを行います。また、イベントチェックの順番を入れ替えることで処理するイベントに優先順位をつけられます。最初にチェックするイベントを記述する程、優先順位が高く処理されます。

ワンポイント

上記は「省電力」に関する処理は掲載していません。基本的な流れのみを説明しています。重要なことは**「割り込み内では処理を最小にする」**、そして割り込みで設定したイベントを**「メインループでイベントチェックをする」**の2点です。

本システムの操作(1)

「時計表示機能つき温度データロガー」の操作及び機能を説明します。下図はシステム全体図です。

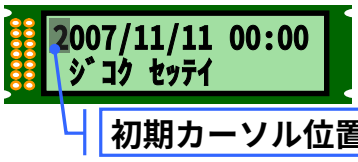


温度表示機能

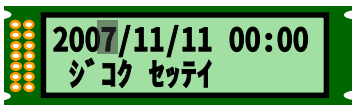
電源ON時に温度表示を開始します。特に設定の必要はありません。

時間表示機能

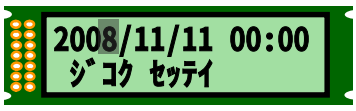
電源ON時では時計表示されません。SW1を押下して現在時刻の設定を開始してください。
SW1押下するとデフォルトの時間が表示されます。カーソルが先頭で点滅します。



変更したい日時にカーソル位置を合わせます。カーソル移動はSW2,SW3の押下で移動します。カーソル位置を合わせたらSW4,SW5で数字を上下させます。時間は24時間制で設定してください。

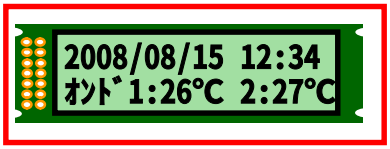


カーソル位置をSW3で移動します。左図の場合「7」位置まで移動します。



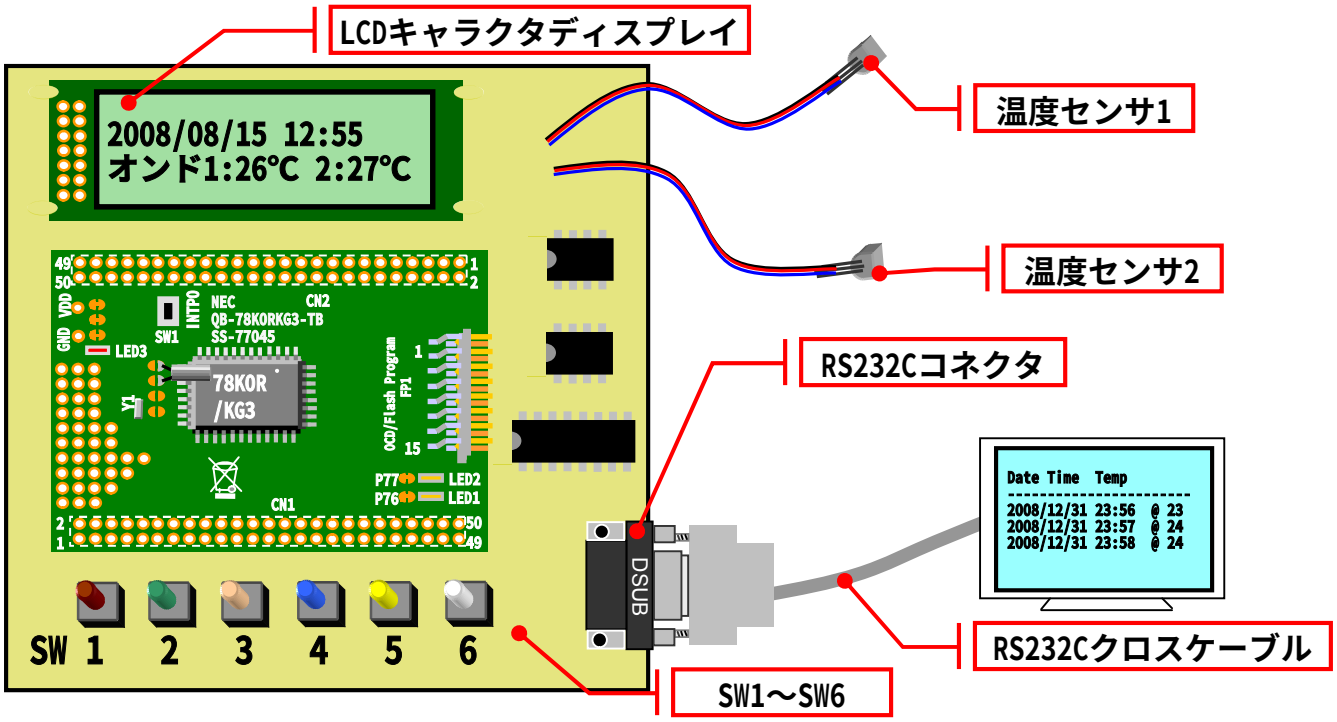
SW5を押下するとカーソル位置の数字が増えます。この要領で全ての時刻を設定します。

全ての時刻設定が終わりましたら、再度SW1を押下してください。



時間表示と共に温度表示が開始されます。また、温度データをEEPROMへ保存します。保存は5分間に1回、そのとき観測された温度データを記録します。

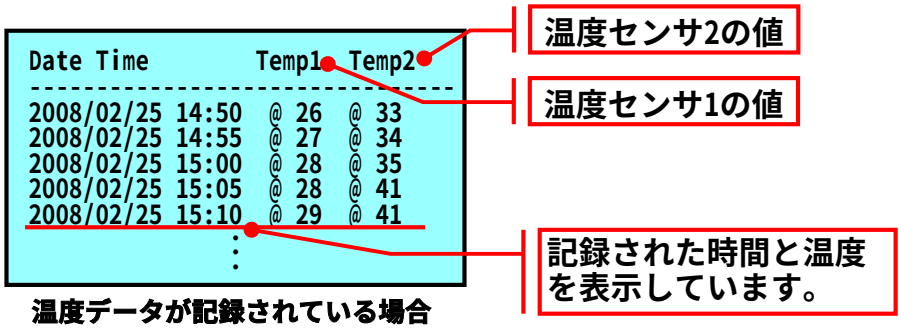
本システムの操作(2)



温度データ送信機能

時計/温度表示時にSW6を押下します。RS232Cコネクタにはクロスケーブルを接続しターミナルソフトを起動しておきます。(「RS232Cで送信」章を参考にしてください)
SW6を押下すると、PC側のターミナルソフトへデータが送られます。温度データの有無でターミナルソフトに表示される画面が異なります。

温度データがない場合



温度データが記録されている場合

記録可能な温度データの最大数は65535個ですが、1つのデータにつき8バイト必要です。そのため256KbitのI2C EEPROMを使用しても記録できるデータは8191個になります。(記録用のヘッダーとして8バイトが必要なため1個分を使用します。)

操作の概要は分かったでしょうか？実際に使うとよりイメージできると思います。時刻の設定ではカーソル移動にリピート処理(SW2～SW5を押下し続けるとカーソル移動が自動で行われる処理)が入っていたり、それなりに使い易さを追求しています。
では、次ページより実際にソフトウェアを作成します。まず、Appliletで行うソフトウエアの初期設定の説明します。

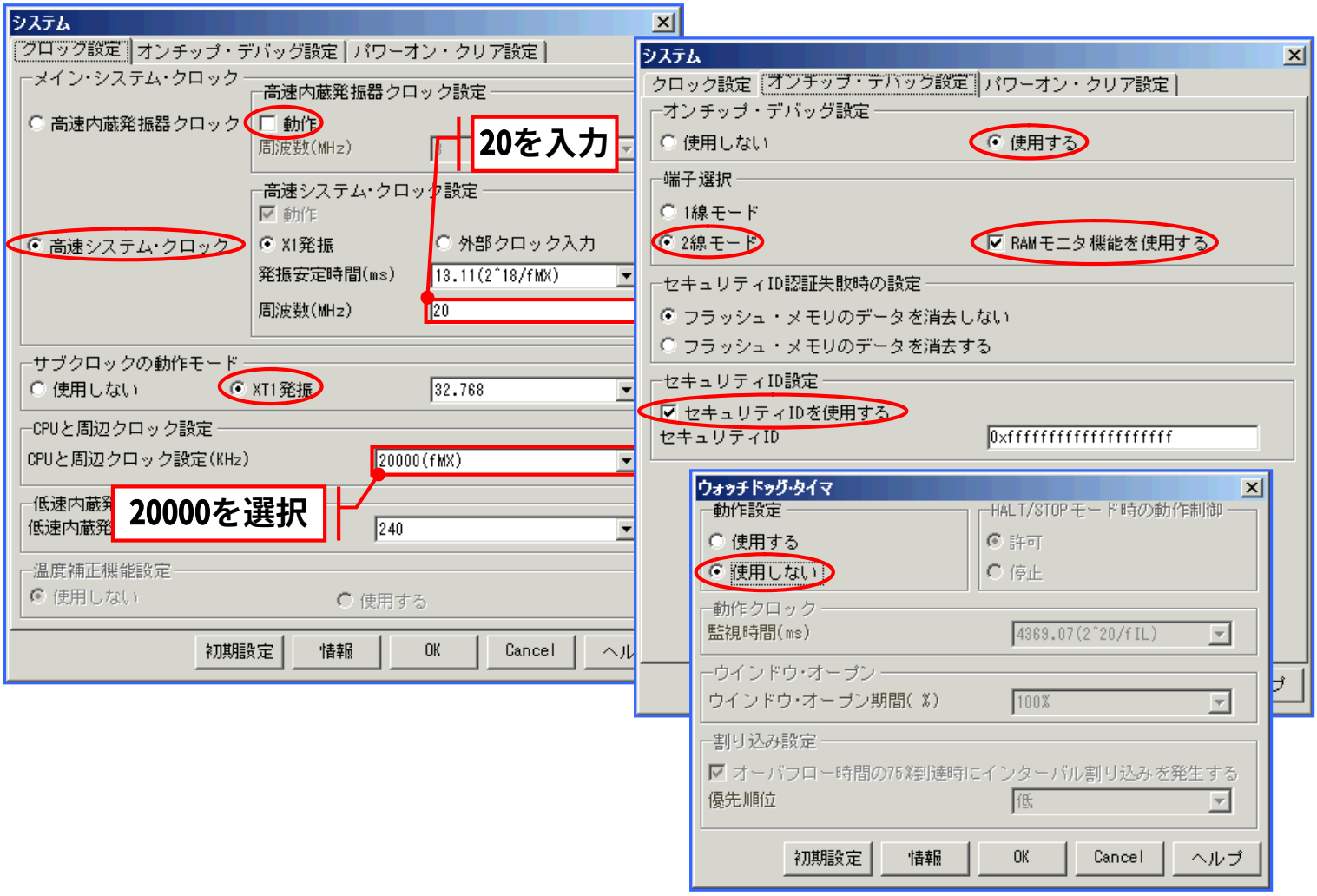
Appliletの設定

「時計表示機能つき温度データロガー」で、必要な周辺機能とAppliletでの設定を説明します。

1.[システム]システムクロック20MHz，サブクロック32.768kHz，オンチップデバッグ2線，RAMモニタあり、ウォッチドッグ･タイマなし
2.[ポート]P76,P77,P80～P87を出力、P120をPull-up
3.[割り込み]INTP0立ち下がりエッジ
4.[シリアル]SAU0チャネル0 UART0送信，LSB，データ長8bit，パリティなし，ストップビット1，115200bps，送信完了割り込みあり、IIC0 シングルマスタ，標準bps，コールバック関数に全てチェック，マスタ送信コールバック時にストップコンディション生成のチェックを外す
5.[A/Dコンバータ]A/D変換使用する，コンパレータ許可，アナログ入力端子ANI13-ANI15，変換開始チャネルANI13，変換時間 4.0V以上の時24us
6.[リアルタイムカウンタ]使用する，24時間制，INTRTC定周期割り込みを1分に1回に設定
7.[タイマ]チャネル0～3をインターバルタイマにする,チャネル1は1msec，チャネル2は100usec，チャネル3は50msecでインターバル割り込みを行う

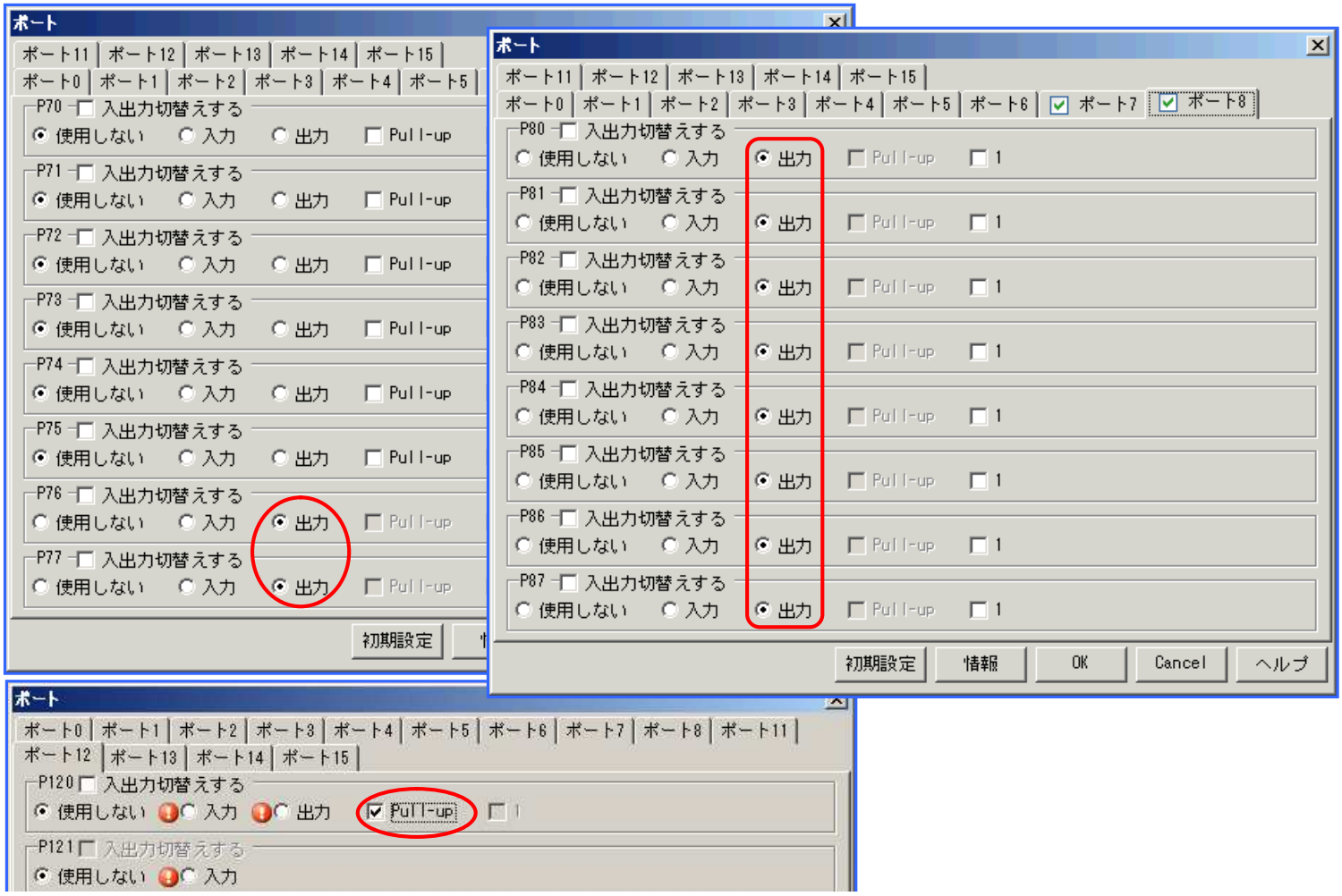
赤で示した場所が設定を行う部分です。それ以外はデフォルトの設定です。初期の設定は、QB-78K0RKG3-TBのマイクロコントローラ名「78K0R/KG3」、デバイス名「uPD78F1166_A0」を選択し、プロジェクト名を「extend3」としています。

1.[システム]システムクロック20MHz，サブクロック32.768kHz，オンチップデバッグ2線，RAMモニタあり、ウォッチドッグ･タイマなし

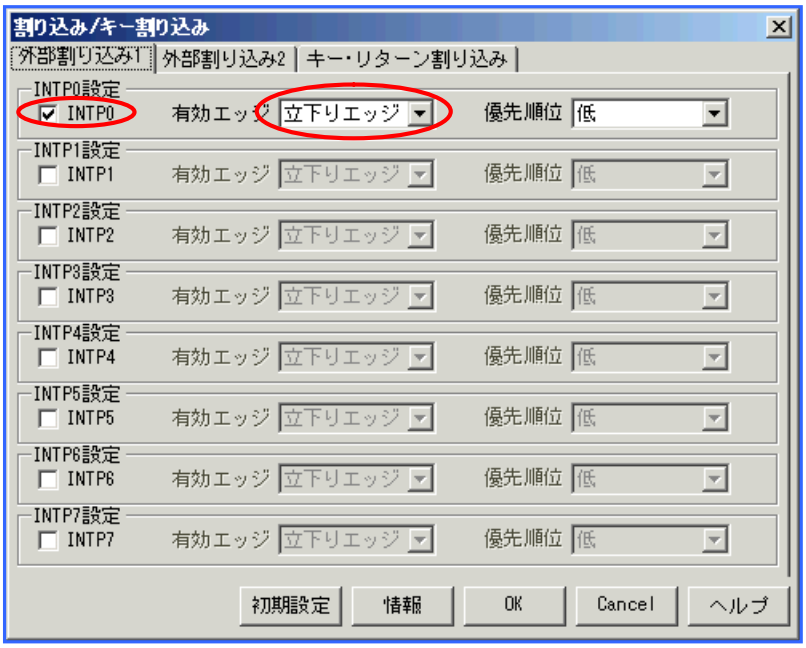


Appliletの設定

2.[ポート]P76,P77,P80～P87を出力、P120をPull-up

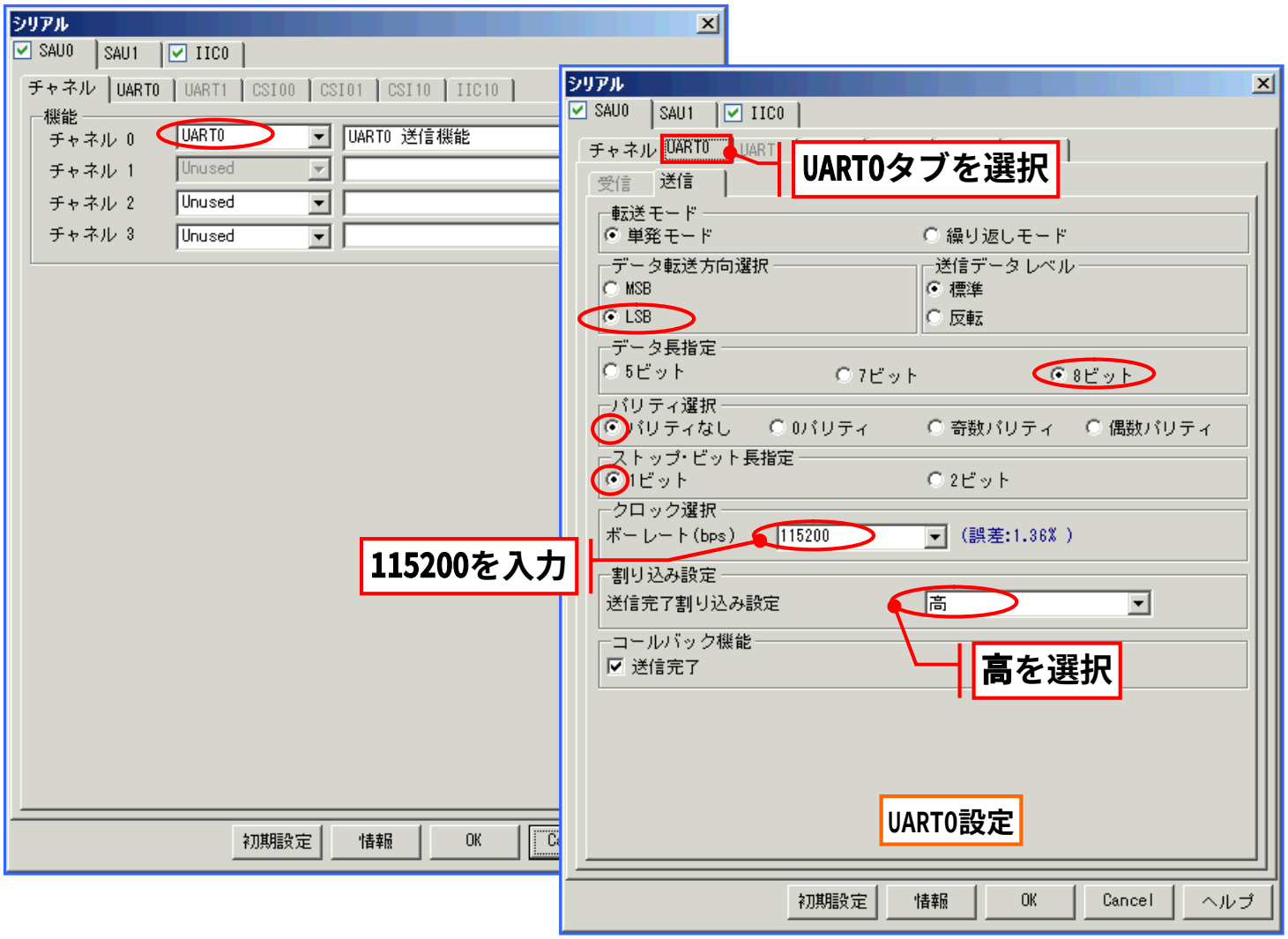


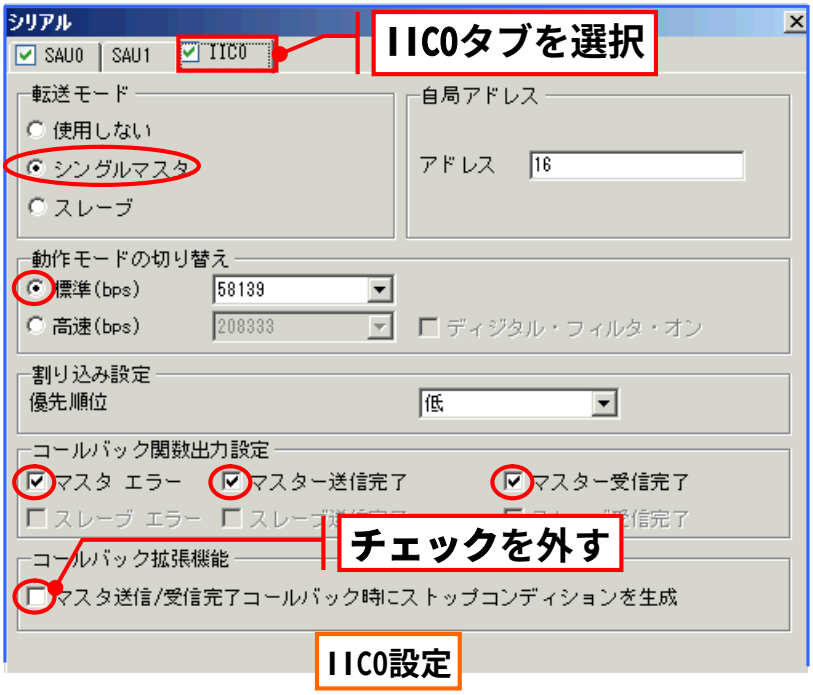
3.[割り込み]INTP0立ち下がりエッジ



Appliletの設定

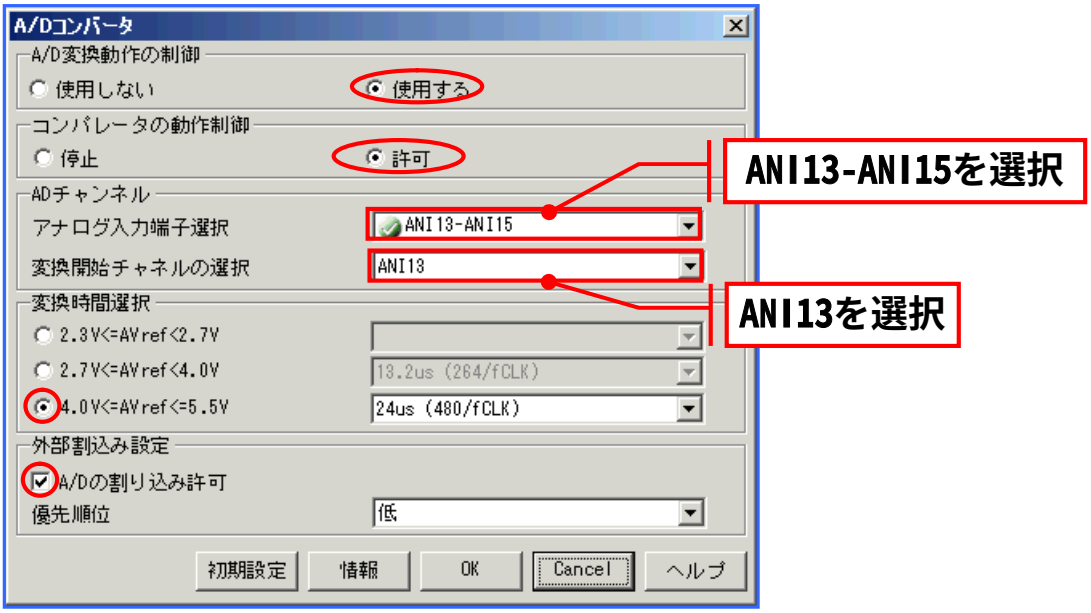
4.[シリアル]SAU0チャネル0 UART0送信，LSB，データ長8bit，パリティなし，ストップビット1，115200bps，送信完了割り込みあり、IIC0 シングルマスタ，標準bps，コールバック関数に全てチェック，マスタ送信コールバック時にストップコンディション生成のチェックを外す



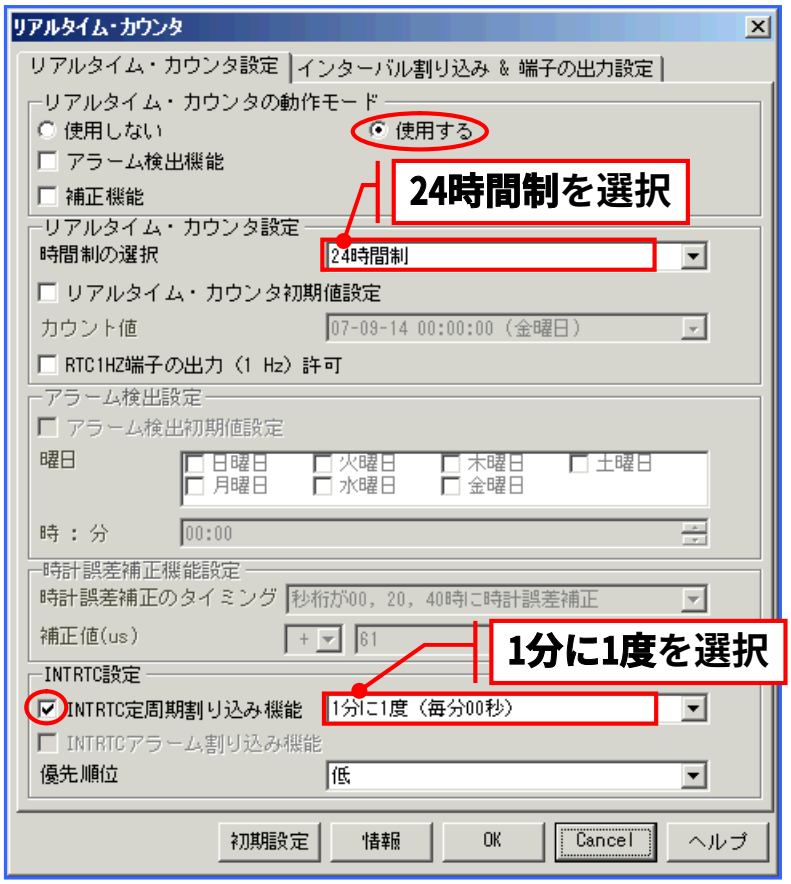


Appliletの設定

5.[A/Dコンバータ]A/D変換使用する，コンパレータ許可，アナログ入力端子ANI13-ANI15，変換開始チャネルANI13，変換時間 4.0V以上の時24us

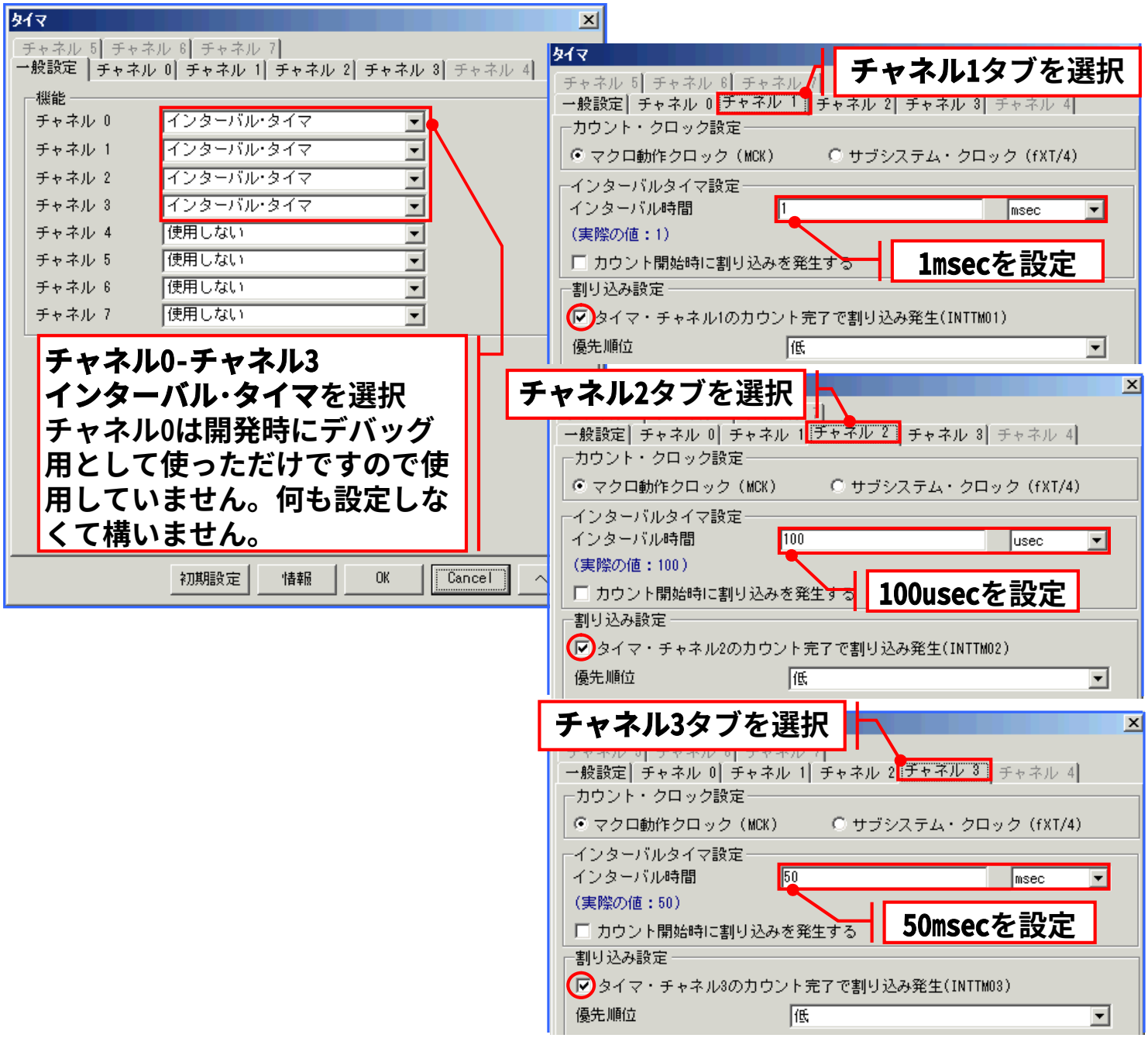


6.[リアルタイムカウンタ]使用する，24時間制，INTRTC定周期割り込みを1分に1回に設定



Appliletの設定

7.[タイマ]チャネル0～3をインターバルタイマにする，チャネル1は1msec，チャネル2は100usec，チャネル3は50msecでインターバル割り込みを行う



以上で、Appliletでの設定は終了です。設定が終わったらコードの生成を行ってください。
次ページより、生成されたファイル一覧の説明と作成するプログラムの全体をフローチャート図を使って説明します。ただし、フローチャート図で全てを説明しているのではなく、全体の流れが解る程度に絞っています。「製作手順」の各章で周辺機能を１つ１つ説明しているので、そちらも合わせて見てください。

全ソースファイルの説明

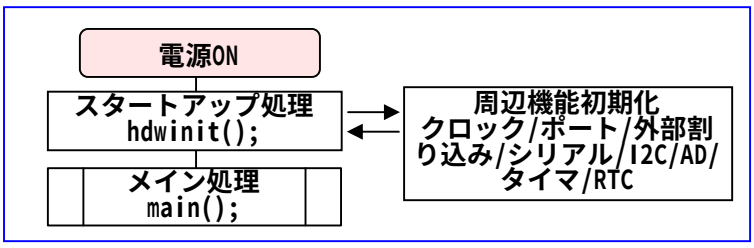
前ページまでのApplilet設定でコードを生成して下さい。下記に出力されるソースファイルと、新規に作成するソースファイルを説明します。

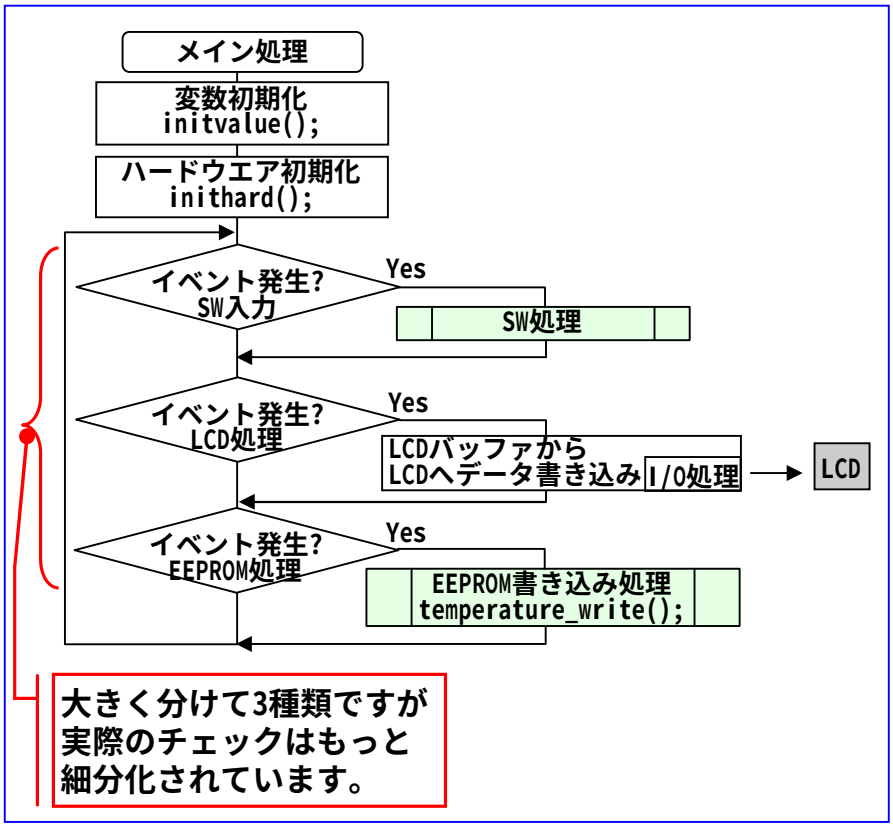
macrodriver.h	AppliletのAPI等で使用される数値が定義されています。
user_define.h	ユーザが自由に使える定義です。全てのCソースファイルにincludeされています。イベント種類やEEPROM/LCDコマンド定義、各種周辺固有の定義を集めています。
System.c System.h systeminit.c System_user.c	システム(クロック)関連に関するソースファイルです。[systeminit.c]には、スタートアップ時にCallされるhdwinit()が含まれています。
main.c	main()処理が含まれるソースファイルです。
Ad.c Ad.h Ad_user.c	A/D処理が含まれるソースファイルです。[Ad_user.c]に割り込みハンドラ処理が書かれてます。
ctl_i2c.c	I2CのEEPROMアクセスに関する処理を行っています。新規作成しています。
ctl_lcd.c	LCD制御やLCDに関係する変数処理を行っています。新規作成しています。
Int.c Int.h Int_user.c	外部割り込み処理が含まれるソースファイルです。[Int_user.c]に割り込みハンドラ処理が書いてあります。
Port.c Port.h	ポートに関する設定です。ポートには割り込みハンドラがありませんので、[port_user.c]は存在しません。
RTC.c RTC.h RTC_user.c	リアルタイム･クロックに関する処理が含まれるソースファイルです。[RTC_user.c]に割り込みハンドラ処理が書かれてます。
Serial.c Serial.h Serial_user.c	シリアル(I2C/UART)処理が含まれるソースファイルです。[Serial_user.c]に各種割り込みハンドラ(UART送信完了、I2C送受信/エラー)処理が書かれてます。
TAU.c TAU.h TAU_user.c	タイマ処理が含まれるソースファイルです。[TAU_user.c]にインターバル割り込みハンドラ処理が書かれてます。

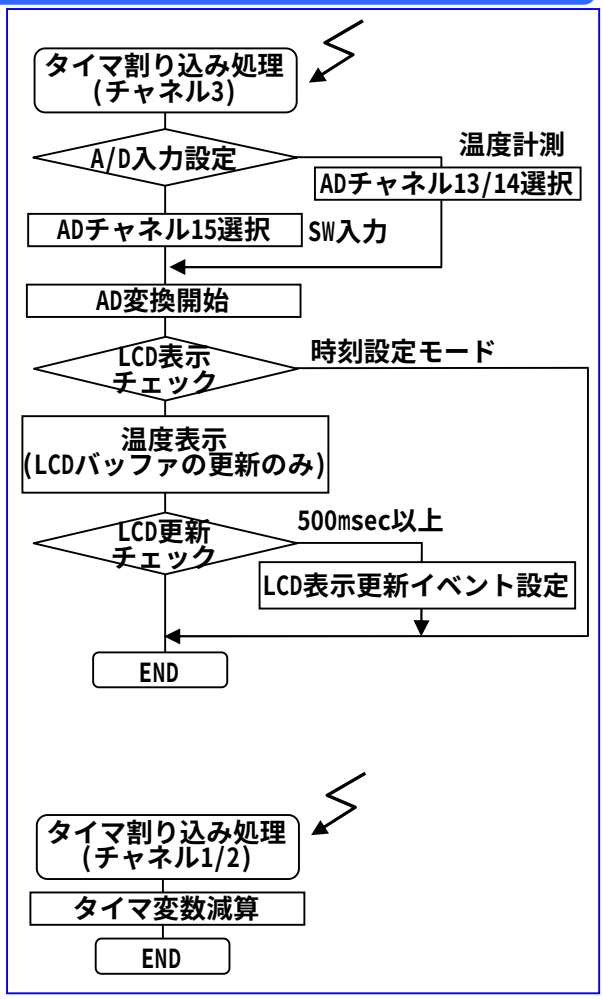
新規に作成するファイルは[ctl_i2c.c],[ctl_lcd.c]の2ファイルのみです。その他ファイルはAppliletが生成したファイルを編集します。

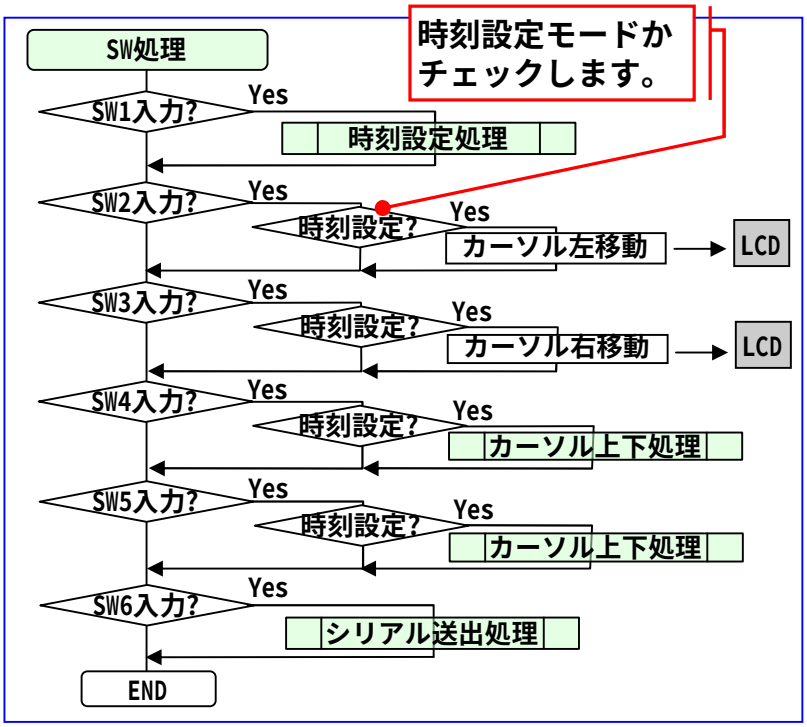
全体のフローチャート

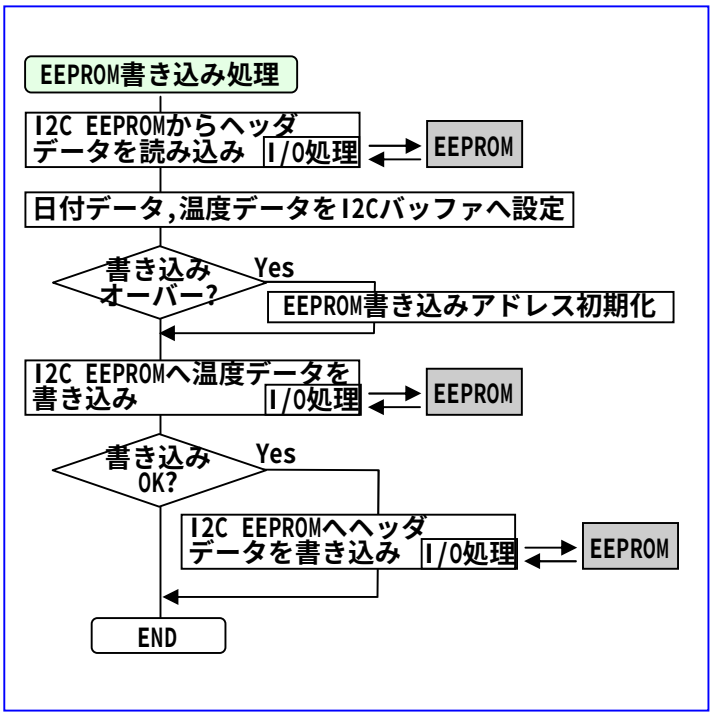
[基本的なプログラムの構成]で示したフローチャートに従って全体の処理を説明します。



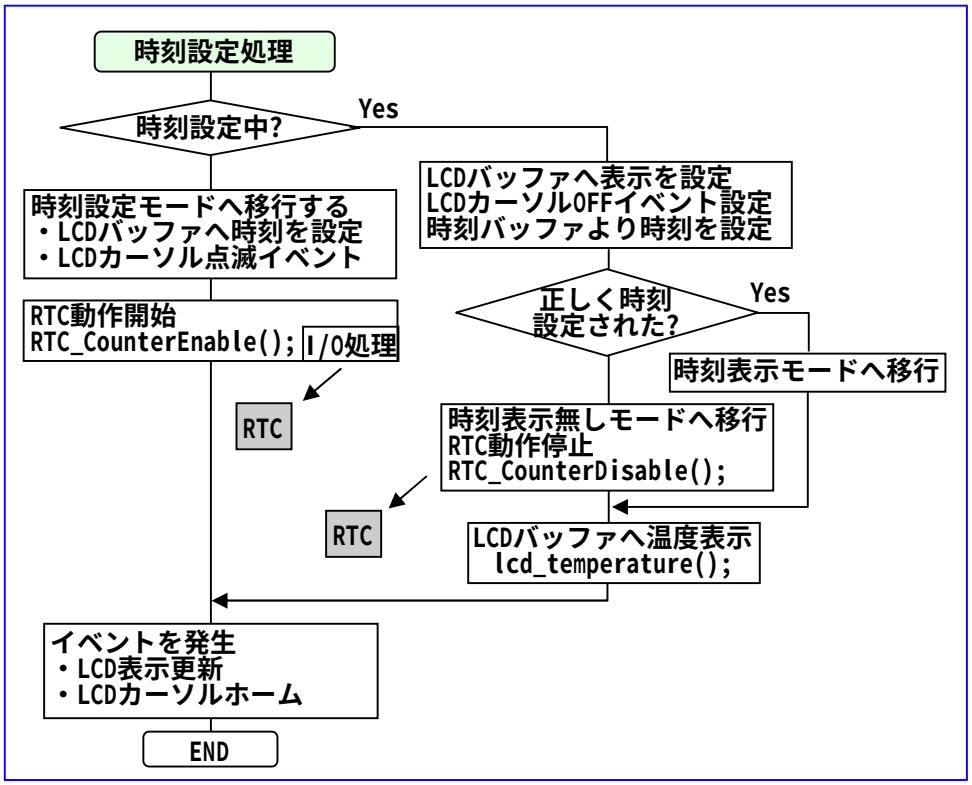


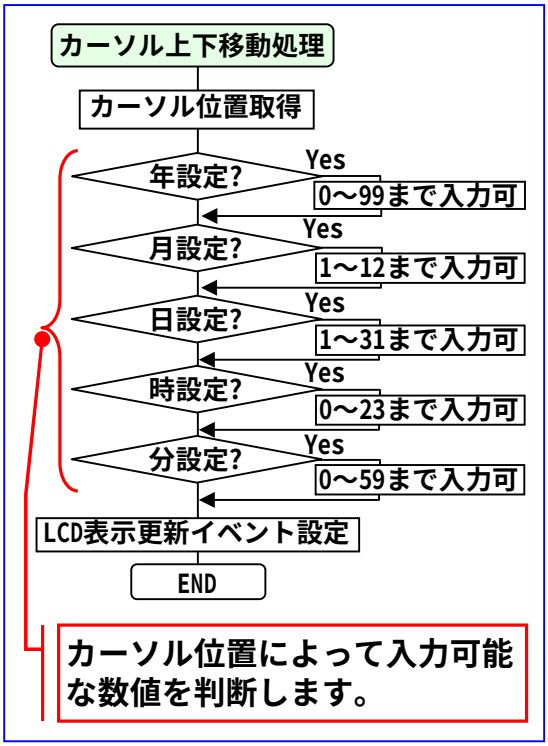


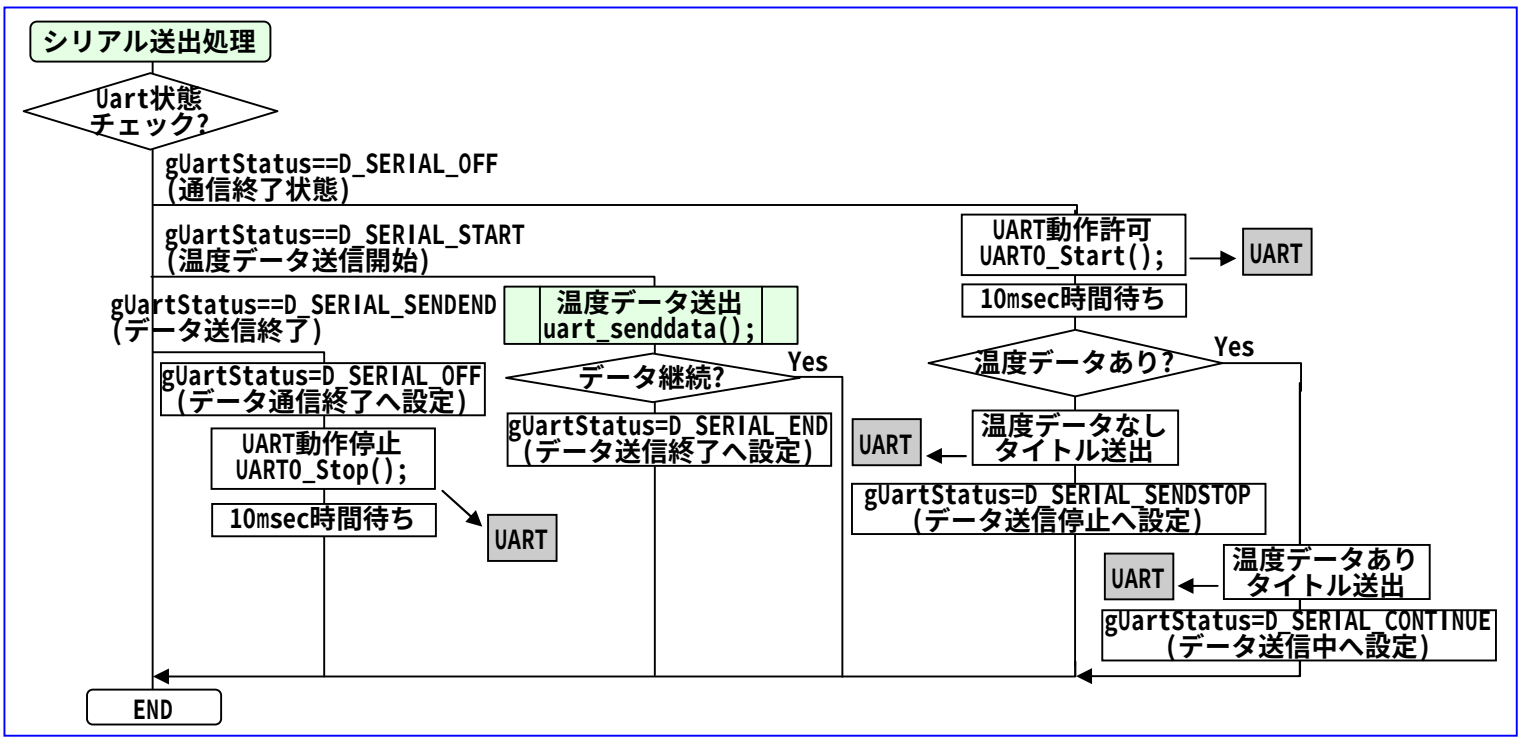


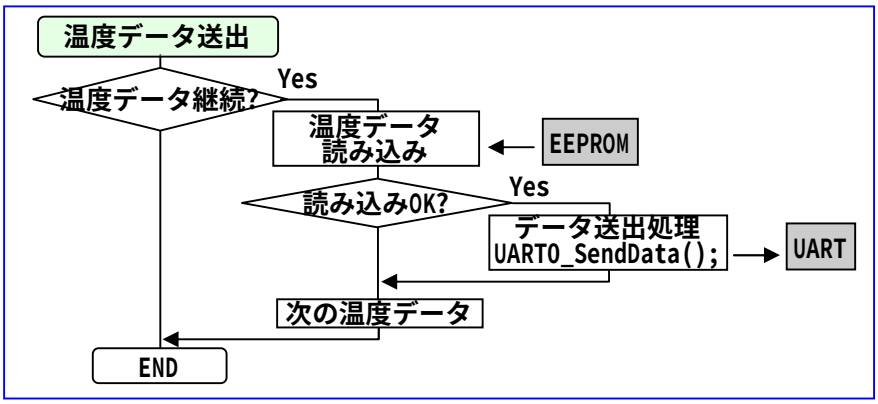


全体のフローチャート

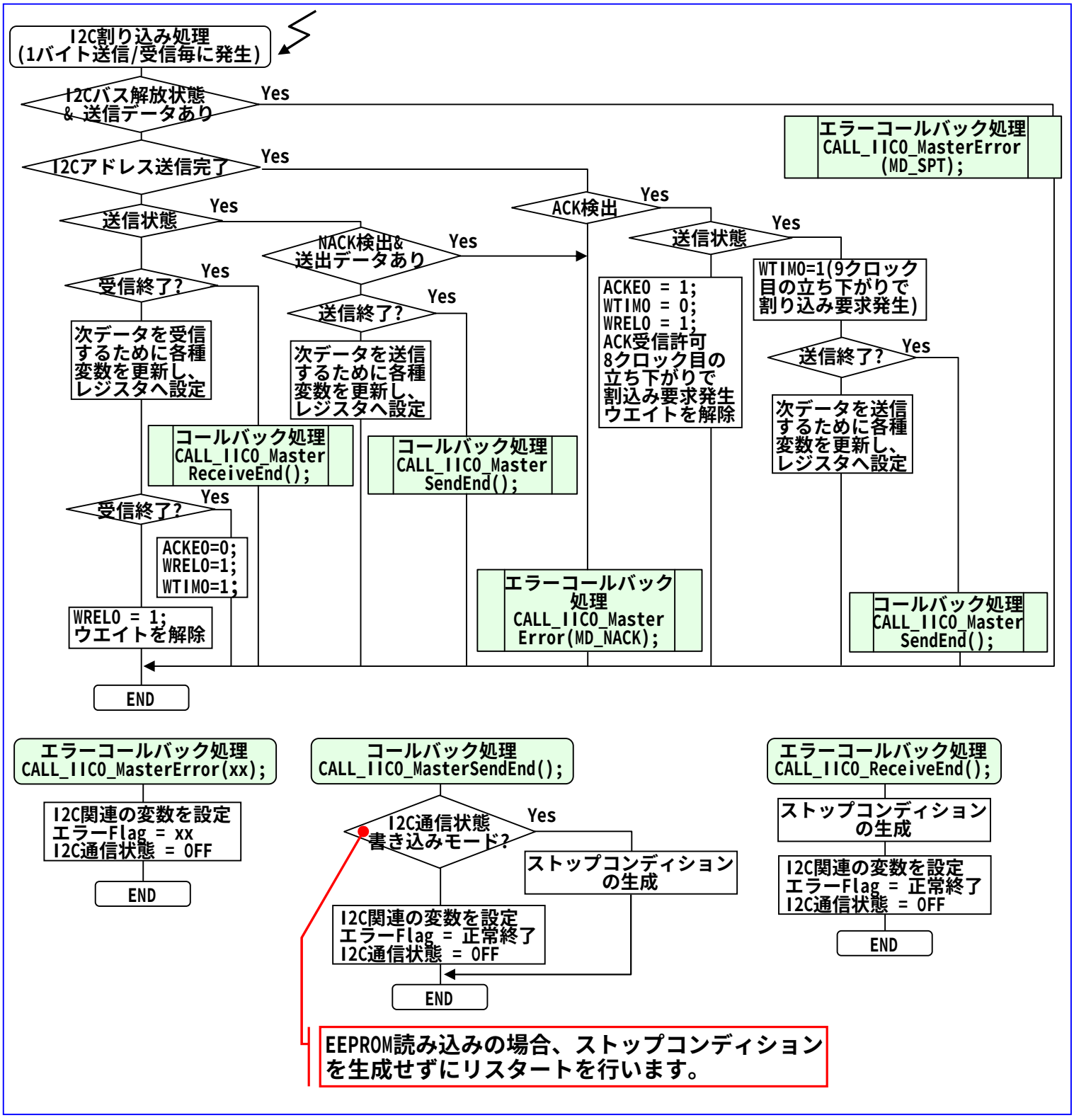

時刻設定処理
時刻設定中?
Yes
時刻設定モードへ移行する
・LCDバッファへ時刻を設定
・LCDカーソル点滅イベント
RTC動作開始
RTC_CounterEnable();
I/O処理
RTC
LCDバッファへ表示を設定
LCDカーソルOFFイベント設定
時刻バッファより時刻を設定
正しく時刻設定された?
Yes
時刻表示モードへ移行
時刻表示無しモードへ移行
RTC動作停止
RTC_CounterDisable();
RTC
LCDバッファへ温度表示
lcd_temperature();
イベントを発生
・LCD表示更新
・LCDカーソルホーム
END


カーソル上下移動処理
カーソル位置取得
年設定?
Yes
0～99まで入力可
月設定?
Yes
1～12まで入力可
日設定?
Yes
1～31まで入力可
時設定?
Yes
0～23まで入力可
分設定?
Yes
0～59まで入力可
LCD表示更新イベント設定
END
カーソル位置によって入力可能な数値を判断します。

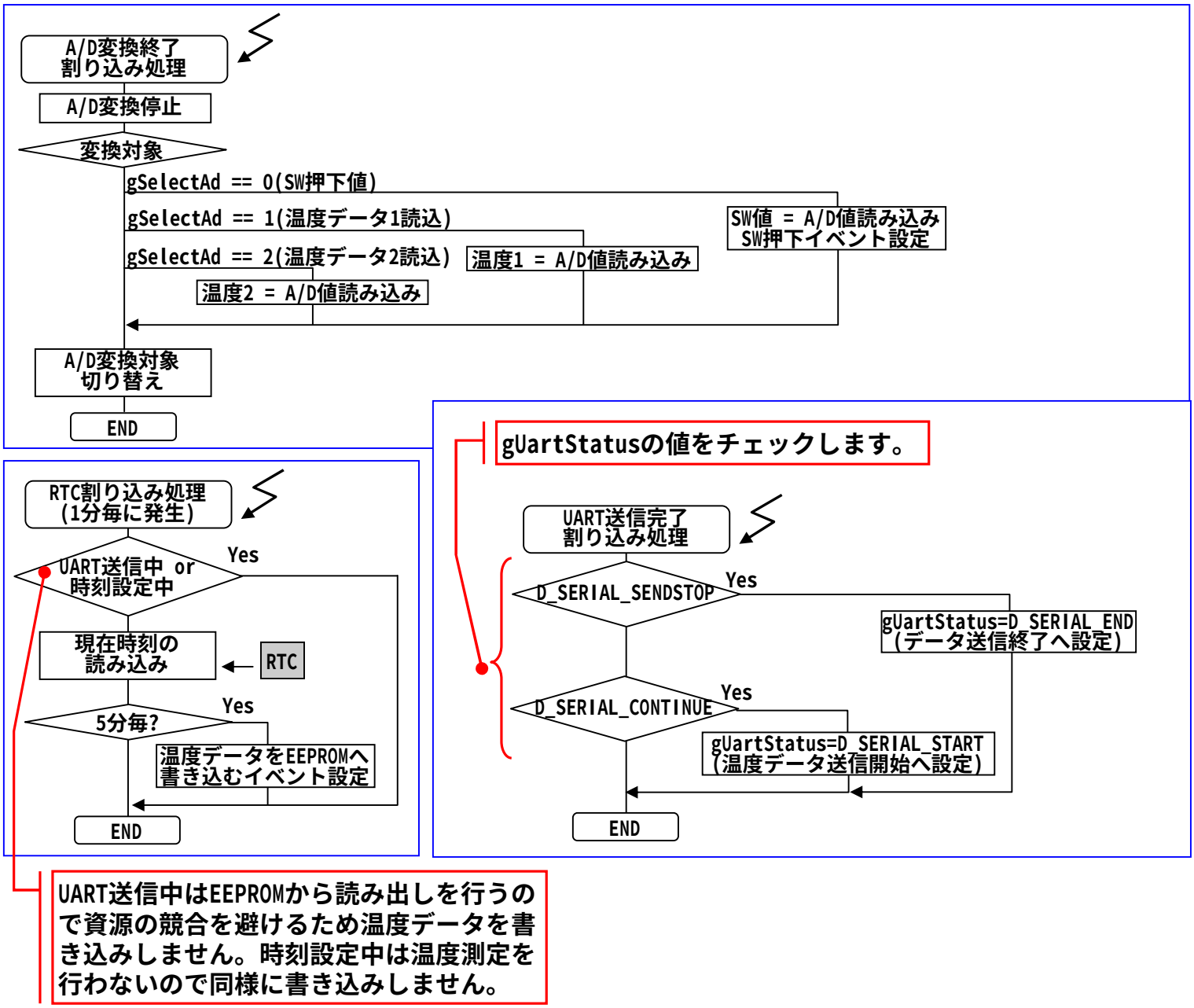

シリアル送出処理
Uart状態チェック?
gUartStatus==D_SERIAL_OFF
(通信終了状態)
gUartStatus==D_SERIAL_START
(温度データ送信開始)
gUartStatus==D_SERIAL_SENDEND
(データ送信終了)
温度データ送出
uart_senddata();
データ継続?
Yes
gUartStatus=D_SERIAL_OFF
(データ通信終了へ設定)
UART動作停止
UART0_Stop();
10msec時間待ち
UART
gUartStatus=D_SERIAL_END
(データ送信終了へ設定)
UART動作許可
UART0_Start();
UART
10msec時間待ち
温度データあり?
Yes
UART
温度データなし
タイトル送出
gUartStatus=D_SERIAL_SENDSTOP
(データ送信停止へ設定)
UART
温度データあり
タイトル送出
gUartStatus=D_SERIAL_CONTINUE
(データ送信中へ設定)
END


温度データ送出
温度データ継続?
Yes
温度データ
読み込み
EEPROM
読み込みOK?
Yes
データ送出処理
UART0_SendData();
UART
次の温度データ
END

全体のフローチャート


I2C割り込み処理
(1バイト送信/受信毎に発生)
I2Cバス解放状態
& 送信データあり
Yes
エラーコールバック処理
CALL_IIC0_MasterError
(MD_SPT);
I2Cアドレス送信完了
Yes
送信状態
Yes
ACK検出
Yes
NACK検出&
送出データあり
Yes
送信状態
Yes
受信終了?
Yes
送信終了?
Yes
ACKE0 = 1;
WTIM0 = 0;
WREL0 = 1;
ACK受信許可
8クロック目の
立ち下がりで
割込み要求発生
ウエイトを解除
WTIM0=1(9クロック
目の立ち下がりで
割り込み要求発生)
送信終了?
Yes
次データを受信
するために各種
変数を更新し、
レジスタへ設定
次データを送信
するために各種
変数を更新し、
レジスタへ設定
次データを送信
するために各種
変数を更新し、
レジスタへ設定
コールバック処理
CALL_IIC0_Master
ReceiveEnd();
コールバック処理
CALL_IIC0_Master
SendEnd();
受信終了?
Yes
ACKE0=0;
WREL0=1;
WTIM0=1;
エラーコールバック
処理
CALL_IIC0_Master
Error(MD_NACK);
コールバック処理
CALL_IIC0_Master
SendEnd();
WREL0 = 1;
ウエイトを解除
END
エラーコールバック処理
CALL_IIC0_MasterError(xx);
I2C関連の変数を設定
エラーFlag = xx
I2C通信状態 = OFF
END
コールバック処理
CALL_IIC0_MasterSendEnd();
I2C通信状態
書き込みモード?
Yes
ストップコンディション
の生成
I2C関連の変数を設定
エラーFlag = 正常終了
I2C通信状態 = OFF
END
エラーコールバック処理
CALL_IIC0_ReceiveEnd();
ストップコンディション
の生成
I2C関連の変数を設定
エラーFlag = 正常終了
I2C通信状態 = OFF
END
EEPROM読み込みの場合、ストップコンディション
を生成せずにリスタートを行います。

全体のフローチャート



ワンポイント

フローチャートについて

商用プログラムでは、フローチャート図やシーケンス図は必須です。ただ、個人で作成するプログラムでも必要でしょうか？答えは「あったほうがいい」です。なぜなら、プログラムを業務で書く場合必ずそれぞれの工程(仕様/設計/コーディング/テストなど)でレビューを行います。が、個人の場合レビューする相手がいません。オープンソースにする手(広く一般公開して意見を聞く)もあるでしょうが、まずフローチャートを書くことを勧めます。フローチャートを書くことでプログラムの見直しを行い、客観的に見ることができます。これはレビューの効果があり、バグも見つけやすく、プログラミングや設計の能力向上につながります。実際このフローチャート図を書いている過程でバグを2件ほど見つけることができました。

プログラム

プログラムの掲載(記述)について

実際のプログラムとここに掲載しているプログラムでは記述に違いがあります。掲載スペースの関係でコメントなどの記述を行数が短くなるように省略しています。そのため、WEBに掲載しているプログラムとコメントの記述部分に差異があります。

[実際のmain.c(抜粋)]

```
/*
***********************************************************************
**  Global define
***********************************************************************
*/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*
**-----------------------------------------------------------------------
**
**  Abstract:
**                      This function  implements main function.
**
**  Parameters:
**                      None
**
**  Returns:
**                      None
**
**-----------------------------------------------------------------------
*/
void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    while ( 1 ) {
        ;
    }

    /* End user code. Do not edit comment generated here */
}
```

意味は同じですが、コメント行が短くなっています。

[チュートリアルガイドでの記述のmain.c(抜粋)]

```
/***********************************************************************
**  Global define
***********************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*-----------------------------------------------------------------------
**  Abstract:           This function  implements main function.
**  Parameters:         None
**  Returns:            None
**-----------------------------------------------------------------------*/
void  main( void )
{
    /* Start user code. Do not edit comment generated here */
    while ( 1 ) {
        ;
    }

    /* End user code. Do not edit comment generated here */
}
```

同様にコメント行が短くなっています。

プログラム

「時計表示機能つき温度データロガー」で必要なプログラムを掲載します。Appliletで出力したコードに次に示す青字のコードをそれぞれのソースファイルへ追加してください。

user_define.h(リスト1)

```
            リスト省略
#ifndef _MD_USER_DEF_
#define _MD_USER_DEF_
/***********************************************************************************
**  Macro define
***********************************************************************************/
/* Start user code for definition. Do not edit comment generated here */
/* メインで処理を行うイベントコード */
#define D_EV_INTP0          0x0000001   /* INTP0押下                              */
#define D_EV_SW1            0x0000002   /* SW1押下                                */
#define D_EV_SW2            0x0000004   /* SW2押下                                */
#define D_EV_SW3            0x0000008   /* SW3押下                                */
#define D_EV_SW4            0x0000010   /* SW4押下                                */
#define D_EV_SW5            0x0000020   /* SW5押下                                */
#define D_EV_SW6            0x0000040   /* SW6押下                                */
#define D_EV_SWCHECK        0x0000080   /* SW値のA/Dを読み込む                    */
#define D_EV_LCD2RAM        0x0000100   /* RAMの内容をLCD表示領域へデータ転送     */
#define D_EV_LCDTEMP        0x0000200   /* LCDに温度表示を行う                    */
#define D_EV_LCDBLINKON     0x0000400   /* LCDブリンクON                          */
#define D_EV_LCDBLINKOFF    0x0000800   /* LCDブリンクOFF                         */
#define D_EV_LCDHOME        0x0001000   /* LCDカーソル位置をホームポジション移動  */
#define D_EV_LCDCURMOVE     0x0002000   /* LCDカーソル位置を移動                  */
#define D_EV_LCDTIME_CHK    0x0004000   /* LCDの時間表示を更新する                */
#define D_EV_UARTSEND       0x0008000   /* UART送出処理中                         */
#define D_EV_EEPROMWRITE    0x0010000   /* EEPROMへ温度データを書き込む           */

/* 押しボタンを判定するADの値 */
#define D_SW1_HIGH          0x0030
#define D_SW2_HIGH          0x00A4
#define D_SW3_HIGH          0x012C
#define D_SW4_HIGH          0x01B0
#define D_SW5_HIGH          0x0234
#define D_SW6_HIGH          0x033B
#define D_SW_REPEAT         7           /* この値を超えた時にキーリピートを行う       */

/* 時計ステータス定義 */
#define D_CLK_OFF           0           /* 時間表示なし(温度データを表示するだけ)     */
#define D_CLK_ON            1           /* 時間設定中(温度計測/EEPROM書き込みを行わない) */
#define D_CLK_ENABLE        2           /* 時間表示中                                 */

/* UARTステータス定義 */
#define D_SERIAL_OFF        0           /* UART通信停止中                             */
#define D_SERIAL_START      1           /* UARTへ日付/温度データを送出中              */
#define D_SERIAL_CONTINUE   2           /* UARTへタイトル文字列を送出中               */
#define D_SERIAL_SENDSTOP   3
#define D_SERIAL_SENDEND    4

/* I2Cステータス定義 */
#define D_I2C_OFF           0
#define D_I2C_WRITE         1           /* write mode (EEPROM データ書き込み)     */
#define D_I2C_READSET       2           /* read set mode (EEPROM アドレス設定)    */
#define D_I2C_READDATA      3           /* read data mode (EEPROM データ呼び出し) */

#define D_EEPROM_WRITETIME  1           /* EEPROMへ温度データを書き込むインターバル時間(分単位)              */
#define D_EEPROM_CODE       0xA2        /* コントロールバイト値、EEPROMのデバイスアドレスは1に設定している */
#define D_EEPROM_BUF        16          /* EEPROMを読み込む際のバッファ(書き込みバッファとしても利用)        */
#define D_EEPROM_TIMEOUT    8192        /* EEPROMのタイムアウト時間                                          */
#define D_EEPROM_HEAD       "K0R "      /* EEPROM識別コード                                                  */
#define D_EEPROM_HEADMAX    4           /* EEPROM識別コードの文字数                                          */
#define D_EEPROM_HEADLEN    8           /* EEPROM識別コードヘッダーの長さ                                    */
#define D_EEPROM_TEMPLEN    8           /* EEPROMへ書き込む温度データの長さ                                  */
#define D_EEPROM_MAXDATA    4095        /* EEPROMに書き込めるデータの数(ROMサイズ/8 - 1)                      */


#undef RTC_WAITTIME
#define RTC_WAITTIME        20000

/* LCD定義 */
typedef struct {
    UCHAR rs    : 1;
    UCHAR sts   : 1;
    UCHAR data  : 8;
} ST_LCDCMD;

/* カーソル移動の定義 */
#define D_LCD_CURUP         1
#define D_LCD_CURDOWN       2
#define D_LCD_X_SIZE        16          /* LCD表示サイズ X                            */
#define D_LCD_Y_SIZE        2           /* LCD表示サイズ Y                            */
```

プログラム

user_define.h(リスト2)

```
#define D_LCD_SIZE          (D_LCD_X_SIZE * D_LCD_Y_SIZE)
#define D_LCD_CGRAMSIZE     (8 * 8)
#define D_LCD_UPDATECOUNT   9           /* LCD表示を更新するカウンタ                                    */

#define D_LCD_X_SIZE        16          /* LCD表示サイズ X                                              */
#define D_LCD_Y_SIZE        2           /* LCD表示サイズ Y                                              */
#define D_LCD_SIZE          (D_LCD_X_SIZE * D_LCD_Y_SIZE)
#define D_LCD_CGRAMSIZE     (8 * 8)
#define D_LCD_UPDATECOUNT   9           /* LCD表示を更新するカウンタ                                    */

/* LCDインストラクション設定 */
#define D_LCD_CLR           0x01        /* 表示クリア                                                   */
#define D_LCD_HOME          0x02        /* カーソルをホームポジションへ移動                             */
#define D_LCD_ENTRY         0x04        /* エントリー・モード・セット(デフォルトはカーソル左移動,表示シフトなし */
#define D_LCD_SET           0x08        /* 表示ON/OFF設定(デフォルトは表示OFF,カーソルOFF,ブリンクOFF)   */
#define D_LCD_SHIFT         0x10        /* カーソル/表示の移動(デフォルトはカーソル位置を左へシフト     */
#define D_LCD_CGRAM         0x40        /* CGRAMのアドレス設定                                          */
#define D_LCD_DDRAM         0x80        /* DDRAMのアドレス設定                                          */

/* LCDインストラクションのbit設定 */
#define D_LCD_ENTRY_INC     0x02        /* カーソル右移動(インクリメント)                               */
#define D_LCD_ENTRY_SHIFT   0x01        /* 表示シフトあり                                               */
#define D_LCD_SET_ON        0x04        /* 表示ON                                                       */
#define D_LCD_SET_CURSOR    0x02        /* カーソルON                                                   */
#define D_LCD_SET_BLINK     0x01        /* ブリンクON                                                   */
#define D_LCD_SHIFT_DISP    0x08        /* 表示全体の移動指定                                           */
#define D_LCD_SHIFT_RIGHT   0x04        /* カーソル/表示の移動を右へシフト(デフォルトは左へシフト)      */

/* End user code for definition. Do not edit comment generated here */
#endif
```

main.c(リスト1)

リスト省略

```
/***********************************************************************************
**  Include files
************************************************************************************/
#include "macrodriver.h"
#include "Ad.h"
#include "TAU.h"
#include "System.h"
#include "RTC.h"
#include "Int.h"
#include "Port.h"
#include "Serial.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "String.h"
#include "Stdlib.h"

/* in ctl_lcd.c */
extern void lcd_init( void );
extern void lcd_temperature( void );
extern void lcd_curhome( void );
extern void lcd_blinkon( void );
extern void lcd_blinkoff( void );
extern void lcd_cursorright( void );
extern void lcd_cursorleft( void );
extern void lcd_cursorupdown( UCHAR direction_ );
extern void lcd_vram2lcd( void );

/* in ctl_i2c.c */
extern void i2c_initvaluie( void );
extern void i2c_eepromheaderwrite( void );
extern void i2c_eepromchk( void );
extern MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *sadrs_, UINT length_ );
extern MD_STATUS i2c_randomread( USHORT dataadrs_, UCHAR *p_sadrs_ );
extern MD_STATUS i2c_read( UCHAR *p_sadrs_, UINT length_ );

/* in Serial_user.c */
extern void uart_init( void );
extern MD_STATUS uart_sendtitle( void );
extern MD_STATUS uart_senddata( void );

/* in This file. */
void initvalue( void );
void inithard( void );
void wait100usec( UINT para1_ );
void wait1msec( UINT para1_ );
void eventcheck( void );
void pushswcheck( UCHAR *p_swnumber_, USHORT *p_repeat_ );
void clocksettime( void );

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
```

プログラム

main.c(リスト2)

```
/* Start user code for global definition. Do not edit comment generated here */
/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in ctl_lcd.c */
extern UCHAR   gLcdUpdate;
extern UCHAR   gtLcdBuffer[ D_LCD_SIZE + 1 ];
extern UCHAR   gLcdCursor;
extern UCHAR   gLCDTIME[ D_LCD_X_SIZE ];
extern const UCHAR gLCDTEXT[][ D_LCD_X_SIZE ];

/* in ctl_i2c.c */
extern UINT    gI2cTemperatureCount;

/* - - - - - - - - このファイルでの定義 - - - - - - - - - */
UINT    g1msecCounter;                  /* Interval timer 1msec(ch1使用)                    */
UINT    g1msecCounterWait;              /* Interval timer 1msec for wait function           */
UINT    g100usecCounterWait;            /* Interval timer 100usec for wait function(ch2使用)*/
UINT    g50msecCounter;                 /* Interval timer 50msec(ch3使用)                   */
UINT    gRtcCounter;                    /* Interval timer 1miniute(rtc使用)                 */

ULONG   gEventflag;                     /* メインループでチェックするイベント。D_EV_xxxxで定義 */
                                        /* bit0 : TBボード上のSW押下(INTP0)                 */
                                        /* bit1 : SW1押下                                   */
                                        /* bit2 : SW2押下                                   */
                                        /* bit3 : SW3押下                                   */
                                        /* bit4 : SW4押下                                   */
                                        /* bit5 : SW5押下                                   */
                                        /* bit6 : SW6押下                                   */
                                        /* bit7 : SW値のA/Dを読み込む                       */
                                        /* bit8 : RAMの内容をLCD表示領域へデータ転送        */
                                        /* bit9 : LCDに温度表示を行う                       */
                                        /* bit10: LCDブリンクON                             */
                                        /* bit11: LCDブリンクOFF                            */
                                        /* bit12: LCDカーソル位置をホームポジション移動     */
                                        /* bit13: LCDカーソル位置を移動                     */
                                        /* bit14: LCDの時間表示を更新する                   */
                                        /* bit15: UART送出処理中                            */
                                        /* bit16: EEPROMへ温度データを書き込む              */
                                        /* bit17 - bit31: reserve(予約)                     */

USHORT  gPushButtonRpeat;               /* 押下され続けると100msec毎にカウントアップする    */
UCHAR   gPushButtonNo;                  /* 押下された SW番号                                */
UCHAR   gPushButtonNoOld;               /* 以前に押下された SW番号                          */
UCHAR   gSelectAd;                      /* ADチャネルを選択する                             */
USHORT  gPushButtonAd;                  /* ADチャネル15に接続したSWボタンの直値             */
USHORT  gTemperatureAd;                 /* ADチャネル14に接続した温度センサ1の直値          */
USHORT  gTemperatureAd2;                /* ADチャネル13に接続した温度センサ2の直値          */
UINT    gSecDispTime;                   /* 時計表示の':'を1秒毎に点滅させるためのカウンタ   */
struct  RTCCounterValue gsClockNow;     /* 現在の時間を表す                                 */
UCHAR   gUartStatus;                    /* UART(シリアル)ステータス                         */
UCHAR   gClockStatus;                   /* 時計ステータス                                   */
UCHAR   gTempWriteBuffer[ D_EEPROM_TEMPLEN ];

/* End user code for global definition. Do not edit comment generated here */

/*
**-------------------------------------------------------------------------
**  Abstract:            This function  implements main function.
**  Parameters:          None
**  Returns:             None
**-------------------------------------------------------------------------
*/
void  main( void )
{
    /* Start user code. Do not edit comment generated here */

    /* 変数初期化 */
    initvalue();

    /* ハードウェア初期化 */
    inithard();

    wait1msec( 5 );

    /* メインループ */
    while ( 1 )
    {
        eventcheck();
    }

    /* End user code. Do not edit comment generated here */
}
```

mainは解りやすく小さくするのが基本です。

プログラム

main.c(リスト3)

```
/* Start adding user code. Do not edit comment generated here */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 変数初期化                                                                  */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void initvalue()
{
    MD_STATUS ret;

    gEventflag = 0;

    g1msecCounter = 0;
    g1msecCounterWait = 0;
    g100usecCounterWait = 0;
    g50msecCounter = 0;
    gRtcCounter = 0;

    gSelectAd = 0;
    gPushButtonRpeat = 0;
    gPushButtonNo = 0;
    gPushButtonNoOld = 0;
    gPushButtonAd = 0x3ff;
    gTemperatureAd = 0;
    gTemperatureAd2 = 0;
    gSecDispTime = 0;
    gClockStatus = D_CLK_OFF;
    gsClockNow.Year = 0x07;                         /* 2007/11/11 00:00 */
    gsClockNow.Month = 0x11;
    gsClockNow.Day = 0x11;
    gsClockNow.Hour = 0;
    gsClockNow.Min = 0;
    gsClockNow.Sec = 0;

    RTC_CounterEnable();
    ret = RTC_CounterSet( gsClockNow );
    if ( ret != MD_OK )
    {
        ret = RTC_CounterSet( gsClockNow );
    }
    RTC_CounterDisable();
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* ハードウェア初期化                                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void inithard()
{
    P7.6 = 1;
    P7.7 = 1;

    TAU_Channel0_Start();
    TAU_Channel1_Start();
    TAU_Channel2_Start();
    TAU_Channel3_Start();

    /* EEPROM内の温度データをチェックする */
    i2c_initvaluie();
    i2c_eepromchk();

    /* UART初期化 */
    uart_init();

    /* LCD初期化 */
    lcd_init();
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 100usecの時間待ち                                                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void wait100usec( UINT para1_ )
{
    /* 100us の時間待ちを para1_ 分だけ行う */
    g100usecCounterWait = 0;
    while ( g100usecCounterWait < para1_ ) ;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 1msecの時間待ち                                                             */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void wait1msec( UINT para1_ )
{
    /* 1msec の時間待ちを para1_ 分だけ行う */
    g1msecCounterWait = 0;
    while ( g1msecCounterWait < para1_ ) ;
}
```

共通に使用される関数を[main.c]においています。

共通に使用される関数を[main.c]においています。

プログラム

main.c(リスト4)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 押しボタンが押下されたチェックする                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void pushswcheck( UCHAR *p_swnumber_, USHORT *p_repeat_ )
{
    UCHAR number;
    ULONG ev;

    if ( gPushButtonAd < D_SW6_HIGH )
    {
        if ( gPushButtonAd < D_SW1_HIGH )
        {
            number = 1;
        }
        else if ( gPushButtonAd < D_SW2_HIGH )
        {
            number = 2;
        }
        else if ( gPushButtonAd < D_SW3_HIGH )
        {
            number = 3;
        }
        else if ( gPushButtonAd < D_SW4_HIGH )
        {
            number = 4;
        }
        else if ( gPushButtonAd < D_SW5_HIGH )
        {
            number = 5;
        }
        else
        {
            number = 6;
        }

        if ( gPushButtonNoOld == number )
        {   /* ボタンを押し続けた場合のリピート処理 */
            (*p_repeat_)++;
            if ( (*p_repeat_) > D_SW_REPEAT )
            {
                *p_swnumber_ = number;

                /* SW1/SW6 以外は、リピート処理を行う */
                if ( ( number != 1 ) && ( number != 6 ) )
                {
                    /* 押しボタンSWのイベントを設定 */
                    ev = 1;
                    ev = ev << number;
                    gEventflag |= ev;
                }
            }
        }
        else
        {
            gPushButtonNoOld = number;
            *p_swnumber_ = number;

            /* 押しボタンSWのイベントを設定 */
            ev = 1;
            ev = ev << number;
            gEventflag |= ev;
        }
    }
    else
    {
        gPushButtonNoOld = *p_swnumber_;
        *p_swnumber_ = 0;
        *p_repeat_ = 0;
    }
}
```

A/DによるSW入力でもボタンを押下しつづけた時のリピート処理を実現しています。

プログラム

main.c(リスト5)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 時計の設定と終了処理                                                        */
/* LCDで変更した時刻 "2007/11/11 00:00" をマイコンへ設定する                   */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void clocksettime()
{
    int i;
    UCHAR udat;
    MD_STATUS rdat;

    if ( gClockStatus == D_CLK_ON )
    {   /* 時間設定モードの場合 */
        gEventflag |= D_EV_LCDBLINKOFF;
        for ( i = 0; i < D_LCD_X_SIZE; i++ )
        {
            gLCDTIME[ i ] = gtLcdBuffer[ i ];
        }
        /* マイコンの時間を変更する */
        gsClockNow.Sec = 0;
        gsClockNow.Week = 0;
        udat = (( gLCDTIME[ 14 ] & 0xf ) << 4) + ( gLCDTIME[ 15 ] & 0xf );
        gsClockNow.Min = udat;
        udat = (( gLCDTIME[ 11 ] & 0xf ) << 4) + ( gLCDTIME[ 12 ] & 0xf );
        gsClockNow.Hour = udat;
        udat = (( gLCDTIME[ 8 ] & 0xf ) << 4) + ( gLCDTIME[ 9 ] & 0xf );
        gsClockNow.Day = udat;
        udat = (( gLCDTIME[ 5 ] & 0xf ) << 4) + ( gLCDTIME[ 6 ] & 0xf );
        gsClockNow.Month = udat;
        udat = (( gLCDTIME[ 2 ] & 0xf ) << 4) + ( gLCDTIME[ 3 ] & 0xf );
        gsClockNow.Year = udat;
        rdat = RTC_CounterSet( gsClockNow );
        if ( rdat == MD_OK )
        {
            gClockStatus = D_CLK_ENABLE;
        }
        else
        {
            gClockStatus = D_CLK_OFF;
            RTC_CounterDisable();
        }
        /* 温度を表示する */
        lcd_temperature();
    }
    else
    {   /* 時間設定モードへ移行する */
        gEventflag |= D_EV_LCDBLINKON;
        for ( i = 0; i < D_LCD_X_SIZE; i++ )
        {
            gtLcdBuffer[ i ] = gLCDTIME[ i ];
            gtLcdBuffer[ i + D_LCD_X_SIZE ] = gLCDTEXT[ 1 ][ i ];
        }
        gClockStatus = D_CLK_ON;
        RTC_CounterEnable();
    }
    gEventflag |= D_EV_LCD2RAM;
    gEventflag |= D_EV_LCDHOME;
    gLcdCursor = 0;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 時刻の取得                                                                  */
/* LCDへ取得した時刻 "20YY/MM/DD HH:MM" を設定する                             */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void clock_gettime()
{
    UCHAR udat;

    gSecDispTime++;
    if ( gSecDispTime & 1 )
    {
        gtLcdBuffer[ 13 ] = ':';
    }
    else
    {
        gtLcdBuffer[ 13 ] = ' ';
    }
    udat = (gsClockNow.Min & 0x0f) + '0';
    gtLcdBuffer[ 15 ] = udat;
    udat = ((gsClockNow.Min & 0x0f0) >> 4 )+ '0';
    gtLcdBuffer[ 14 ] = udat;

    udat = (gsClockNow.Hour & 0x0f) + '0';
    gtLcdBuffer[ 12 ] = udat;
    udat = ((gsClockNow.Hour & 0x0f0) >> 4 )+ '0';
    gtLcdBuffer[ 11 ] = udat;
```

SW6には2通りの機能があるので、それを判断します。1つは時刻設定モードに移行する場合と、もう1つは設定した時刻をRTCへ反映する機能です。

RTC機能があるか(サブクロックが搭載されているか)判断します。

LCDの再表示を行うイベントを設定します。

取得した現在の時刻の構造体(gsClockNow)より、LCDへ表示するためのバッファへ設定します。

プログラム

main.c(リスト6)

```
    udat = (gsClockNow.Day & 0x0f) + '0';
    gtLcdBuffer[ 9 ] = udat;
    udat = ((gsClockNow.Day & 0x0f0) >> 4 )+ '0';
    gtLcdBuffer[ 8 ] = udat;

    udat = (gsClockNow.Month & 0x0f) + '0';
    gtLcdBuffer[ 6 ] = udat;
    udat = ((gsClockNow.Month & 0x0f0) >> 4 )+ '0';
    gtLcdBuffer[ 5 ] = udat;

    udat = (gsClockNow.Year & 0x0f) + '0';
    gtLcdBuffer[ 3 ] = udat;
    udat = ((gsClockNow.Year & 0x0f0) >> 4 )+ '0';
    gtLcdBuffer[ 2 ] = udat;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* EEPROMへ温度データを書き込む処理                                            */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void temperature_write()
{
    USHORT eadrs = 0;
    MD_STATUS ret;
    UCHAR dat;

    memset( gTempWriteBuffer, 0x00, D_EEPROM_TEMPLEN );

    /* EEPROMに登録されている温度データ数をチェックする */
    i2c_eepromchk();

    /* EEPROMへ書き込む際のバッファへ日付を設定。9byte-15byteはリザーブ */
    gTempWriteBuffer[ 0 ] = gsClockNow.Year;
    gTempWriteBuffer[ 1 ] = gsClockNow.Month;
    gTempWriteBuffer[ 2 ] = gsClockNow.Day;
    gTempWriteBuffer[ 3 ] = gsClockNow.Hour;
    gTempWriteBuffer[ 4 ] = gsClockNow.Min;
    dat = (UCHAR)((gTemperatureAd * 48) / 400);
    gTempWriteBuffer[ 5 ] = dat;
    dat = (UCHAR)((gTemperatureAd2 * 48) / 400);
    gTempWriteBuffer[ 6 ] = dat;

    /* 温度データの数をチェックして実EEPROMを超えないようにする */
    if ( gI2cTemperatureCount >= D_EEPROM_MAXDATA )
    {
        gI2cTemperatureCount = 0;
    }

    /* 温度データをEEPROMへ書き込みを行う */
    eadrs = ( gI2cTemperatureCount * D_EEPROM_TEMPLEN ) + D_EEPROM_HEADLEN;
    ret = i2c_write( eadrs, gTempWriteBuffer, D_EEPROM_TEMPLEN );
    wait1msec( 5 );

    if ( ret == MD_OK )
    {
        /* EEPROMにヘッダーを書き込む */
        gI2cTemperatureCount++;
        i2c_eepromheaderwrite();
        wait1msec( 5 );
    }

}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* EEPROMに書き込んだデータをシリアルへ送出する処理                            */
/* ボーレート115200bps データ長8bit パリティなし ストップビット1              */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void serial_send()
{
    MD_STATUS ret;

    if ( gUartStatus == D_SERIAL_OFF )
    {
        UART0_Start();
        wait1msec( 10 );
        /* タイトルを送る */
        ret = uart_sendtitle();
        if ( ret == MD_OK )
        {   /* 温度データ読み込みシーケンスへ */
            gUartStatus = D_SERIAL_CONTINUE;
        }
        else
        {   /* データがない場合 */
            gUartStatus = D_SERIAL_SENDSTOP;
        }
    }
```

EEPROMへ書き込むデータを作成します。

EEPROMへ書き込むアドレスを計算します。

EEPROMへのデータ書き込むが成功した場合、管理ヘッダ部分の書き込みします。

現状の送出状態をチェックします。

タイトル文字列の送出します。

プログラム

main.c(リスト7)

```
    else if ( gUartStatus == D_SERIAL_START )
    {   /* 温度データを送る */
        gUartStatus = D_SERIAL_CONTINUE;
        ret = uart_senddata();
        if ( ret != MD_OK )
        {
            /* 送出終了 */
            gUartStatus = D_SERIAL_SENDEND;
        }
    }
    else if ( gUartStatus == D_SERIAL_SENDEND )
    {
        /* 送出終了 */
        gUartStatus = D_SERIAL_OFF;
        UART0_Stop();
        wait1msec( 10 );
    }
    else
    {   /* 通信中状態 */
        ;
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* D_EV_xxxxで定義したイベント処理を行う                                        */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void eventcheck()
{
    int i;

    /* INTP0チェック */
    if ( gEventflag & D_EV_INTP0 )
    {
        /* イベントクリア */
        gEventflag &= ~D_EV_INTP0;
    }

    /* SW1チェック */
    if ( gEventflag & D_EV_SW1 )
    {
        /* 時計の設定と終了処理 */
        clocksettime();
        /* イベントクリア */
        gEventflag &= ~D_EV_SW1;
    }

    /* SW2チェック */
    if ( gEventflag & D_EV_SW2 )
    {
        if ( ( gLcdCursor > 0 ) && ( gClockStatus == D_CLK_ON ) )
        {
            /* LCDカーソル左移動 */
            gLcdCursor--;
            lcd_cursorleft();
        }
        /* イベントクリア */
        gEventflag &= ~D_EV_SW2;
    }

    /* SW3チェック */
    if ( gEventflag & D_EV_SW3 )
    {
        /* LCDカーソル右移動 */
        if ( ( gLcdCursor < ( D_LCD_X_SIZE - 1 ) ) && ( gClockStatus == D_CLK_ON ) )
        {
            gLcdCursor++;
            lcd_cursorright();
        }
        /* イベントクリア */
        gEventflag &= ~D_EV_SW3;
    }

    /* SW4チェック */
    if ( gEventflag & D_EV_SW4 )
    {
        if ( gClockStatus == D_CLK_ON )
        {
            /* LCDカーソル下処理 */
            lcd_cursorupdown( D_LCD_CURDOWN );
        }
        /* イベントクリア */
        gEventflag &= ~D_EV_SW4;
    }
```

温度データを送出します。

イベントをチェックします。事実上のメインプログラムです。

特に何もしません。デバッグ時に簡単に使用できるようにしてあります。

SW1は時刻設定及び終了の処理を行います。

SW2は時刻設定時のカーソル左移動処理を行います。

SW3は時刻設定時のカーソル右移動処理を行います。

SW4は時刻設定時のカーソル上の数字を-1する処理を行います。

プログラム

main.c(リスト8)

SW5は時刻設定時のカーソル上の数字を+1する処理を行います。

SW6は温度データをUARTへ送出する処理を開始します。

A/D入力からSWの押下へ変換します。

```
/* SW5チェック */
if ( gEventflag & D_EV_SW5 )
{
    if ( gClockStatus == D_CLK_ON )
    {
        /* LCDカーソル上処理 */
        lcd_cursorupdown( D_LCD_CURUP );
    }
    /* イベントクリア */
    gEventflag &= ~D_EV_SW5;
}

/* SW6チェック */
if ( gEventflag & D_EV_SW6 )
{
    /* EEPROMに書き込んだデータを送出する処理 */
    serial_send();

    /* データ送信が完了するまでイベントフラグを立てておく */
    if ( gUartStatus == D_SERIAL_OFF )
    {
        /* イベントクリア */
        gEventflag &= ~D_EV_SW6;
    }
}

/* 押しボタンが押下されたチェックする */
if ( gEventflag & D_EV_SWCHECK )
{
    pushswcheck( &gPushButtonNo, &gPushButtonRpeat );

    /* イベントクリア */
    gEventflag &= ~D_EV_SWCHECK;
}

/* LCDへ現在の時間を設定する */
if ( gEventflag & D_EV_LCDTIME_CHK )
{
    clock_gettime();

    /* イベントクリア */
    gEventflag &= ~D_EV_LCDTIME_CHK;
}

/* RAM→LCD表示領域転送チェック */
if ( gEventflag & D_EV_LCD2RAM )
{
    /* LCDバッファVRAMからLCD VRAMへ転送 */
    lcd_vram2lcd();

    /* イベントクリア */
    gEventflag &= ~D_EV_LCD2RAM;
}

/* LCDカーソル位置を移動チェック */
if ( gEventflag & D_EV_LCDCURMOVE )
{
    /* ホームポジションへ移動 */
    lcd_curhome();

    /* カーソルを指定位置まで移動 */
    for ( i = 0; i < gLcdCursor; i++ )
    {
        lcd_cursorright();
    }
    /* イベントクリア */
    gEventflag &= ~D_EV_LCDCURMOVE;
}

/* LCDカーソル位置をホームポジション移動チェック */
if ( gEventflag & D_EV_LCDHOME )
{
    /* ホームポジションへ移動 */
    lcd_curhome();

    /* イベントクリア */
    gEventflag &= ~D_EV_LCDHOME;
}
```

プログラム

main.c(リスト9)

```
    /* LCDブリンクOFFチェック */
    if ( gEventflag & D_EV_LCDBLINKOFF )
    {
        /* LCDブリンクOFF */
        lcd_blinkoff();

        /* イベントクリア */
        gEventflag &= ~D_EV_LCDBLINKOFF;
    }

    /* LCDブリンクONチェック */
    if ( gEventflag & D_EV_LCDBLINKON )
    {
        /* LCDブリンクON */
        lcd_blinkon();

        /* イベントクリア */
        gEventflag &= ~D_EV_LCDBLINKON;
    }

    /* EEPROMへ温度データを書き込む */
    if ( gEventflag & D_EV_EEPROMWRITE )
    {
        /* EEPROMへ温度データを書き込む */
        temperature_write();

        /* イベントクリア */
        gEventflag &= ~D_EV_EEPROMWRITE;
    }
}
/* End user code adding. Do not edit comment generated here */
```

RTC_user.c

```
                            リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */

/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern UCHAR  gUartStatus;
extern UCHAR  gClockStatus;
extern UINT   gRtcCounter;
extern ULONG  gEventflag;
extern struct RTCCounterValue gsClockNow;

/* End user code for global definition. Do not edit comment generated here */
                                            リスト省略

/*-----------------------------------------------------------------------------
**  Abstract:            This function is  real-time counter constant-period interrupt service handler.
**  Parameters:          None
**  Returns:             None
**-----------------------------------------------------------------------------*/
void CALL_RTC_ConstantPeriodINT( void )
{
    /* Start user code. Do not edit comment generated here */

    /* UART送信中、時刻設定中は何も処理を行わない */
    if ( (gUartStatus == D_SERIAL_OFF) && (gClockStatus == D_CLK_ENABLE) )
    {
        RWAIT = 1;
        while ( RWST == 0 ) ;
        gsClockNow.Min   = MIN;
        gsClockNow.Hour  = HOUR;
        gsClockNow.Day   = DAY;
        gsClockNow.Month = MONTH;
        gsClockNow.Year  = YEAR;
        RWAIT = 0;
        gRtcCounter++;

        if ( (gRtcCounter % D_EEPROM_WRITETIME) == 0 )
        {   /* EEPROMへ温度データを書き込む処理 */
            gEventflag |= D_EV_EEPROMWRITE;
        }
    }

    /* End user code. Do not edit comment generated here */
}
```

UARTよりデータを送出している間はEEPROMへデータを書き込みません。EEPROMからのデータ読み込みと衝突を防ぐためです。

プログラム

Int_user.c

```
                        リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern ULONG  gEventflag;
/* End user code for global definition. Do not edit comment generated here */

/*-----------------------------------------------------------------------------
**  Abstract:            This function is INTP0 interrupt service routine.
**  Parameters:          None
**  Returns:             None
**-----------------------------------------------------------------------------*/
__interrupt void MD_INTP0( void )
{
    /* Start user code. Do not edit comment generated here */

    gEventflag |= D_EV_INTP0;

    /* End user code. Do not edit comment generated here */
}
```

Ad_user.c

```
                        リスト省略
/***********************************************************************************
**  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */

/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern ULONG   gEventflag;
extern UCHAR   gSelectAd;
extern USHORT  gPushButtonAd;
extern USHORT  gTemperatureAd;
extern USHORT  gTemperatureAd2;

/* End user code for global definition. Do not edit comment generated here */

/*-----------------------------------------------------------------------------
**  Abstract:            This function is INTAD interrupt service routine.
**  Parameters:          None
**  Returns:             None
**-----------------------------------------------------------------------------*/
__interrupt void MD_INTAD( void )
{
    /* Start user code. Do not edit comment generated here */

    AD_Stop();

    switch ( gSelectAd )
    {
        case 0 :
            AD_Read( &gPushButtonAd );
            gEventflag |= D_EV_SWCHECK;
            break;

        case 1 :
            AD_Read( &gTemperatureAd );
            break;

        case 2 :
            AD_Read( &gTemperatureAd2 );
            break;

        default :
            break;
    }

    gSelectAd++;
    if ( gSelectAd >= 3 )
    {
        gSelectAd = 0;
    }

    /* End user code. Do not edit comment generated here */
}
```

SW押下を判断するA/D入力値

温度センサ1からのA/D入力値

温度センサ2からのA/D入力値

プログラム

TAU_user.c

```
                            リスト省略
/* Start user code for include definition. Do not edit comment generated here */
#include "AD.h"
/* in ctl_lcd.c */
extern void lcd_temperature( void );
/* End user code for include definition. Do not edit comment generated here */

/*  Global define
***********************************************************************************/
/* Start user code for global definition. Do not edit comment generated here */
/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern UINT    g1msecCounter;
extern UINT    g1msecCounterWait;
extern UINT    g100usecCounterWait;
extern UINT    g50msecCounter;
extern ULONG   gEventflag;
extern UCHAR   gSelectAd;
extern UCHAR   gClockStatus;
/* in ctl_lcd.c */
extern UCHAR   gLcdUpdate;
/* End user code for global definition. Do not edit comment generated here */
                            リスト省略
/*  Abstract: This function is INTTM01 interrupt service routine.
**-------------------------------------------------------------------------------*/
__interrupt void MD_INTTM01( void )
{
    /* Start user code. Do not edit comment generated here */
    g1msecCounter++;
    g1msecCounterWait++;
    /* End user code. Do not edit comment generated here */
}

/*  Abstract: This function is INTTM02 interrupt service routine.
**-------------------------------------------------------------------------------*/
__interrupt void MD_INTTM02( void )
{
    /* Start user code. Do not edit comment generated here */
    g100usecCounterWait++;
    /* End user code. Do not edit comment generated here */
}

/*  Abstract: This function is INTTM03 interrupt service routine.
**-------------------------------------------------------------------------------*/
__interrupt void MD_INTTM03( void )
{
    /* Start user code. Do not edit comment generated here */
    g50msecCounter++;

    switch ( gSelectAd )
    {
        case 0 :
            AD_SelectADChannel( ADCHANNEL15 );
            break;
        case 1 :
            AD_SelectADChannel( ADCHANNEL14 );
            break;
        case 2 :
            AD_SelectADChannel( ADCHANNEL13 );
            break;
        default :
            break;
    }
    AD_Start();

    /* 時計表示をチェックする */
    if ( (gClockStatus == D_CLK_OFF) || (gClockStatus == D_CLK_ENABLE) )
    {   /* 通常は温度表示を行う */
        lcd_temperature();

        /* LCD表示を更新する */
        gLcdUpdate++;
        if ( gLcdUpdate > D_LCD_UPDATECOUNT )
        {
            gLcdUpdate = 0;
            gEventflag |= D_EV_LCD2RAM;
            if ( gClockStatus == D_CLK_ENABLE )
            {
                gEventflag |= D_EV_LCDTIME_CHK;
            }
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

50ms毎に割り込み処理が行われます。

50ms毎に3種類のA/D変換を振り分けます。

時刻設定時にはこの処理は行いません。

プログラム

Serial_user.c(リスト1)

```
                        リスト省略
/* Start user code for include definition. Do not edit comment generated here */
#include "String.h"
#include "Stdlib.h"
#include "Stdio.h“

/* in ctl_i2c.c */
extern MD_STATUS i2c_eepromread( USHORT datano_ );
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
                                                    リスト省略
/* Start user code for global definition. Do not edit comment generated here */
/* - - - - - - - - - - 他参照 - - - - - - - - - - - */
/* in main.c */
extern UCHAR   gUartStatus;
/* in ctl_i2c.c */
extern UCHAR   gI2cStatus;
extern MD_STATUS gI2cErrorflag;
extern UINT    gI2cTemperatureCount;
extern UCHAR   gI2cReadBuffer[ D_EEPROM_BUF + 4 ];

/* in ctl_lcd.c */
extern UCHAR   gLCDTIME[ D_LCD_X_SIZE ];

/* - - - - - - - - このファイルでの定義 - - - - - - - - - */
/* UARTへ表示させる固定データ \n\r は改行を意味する   */
const UCHAR title1[] = { "\n\rDate Time        Temp1   Temp2\n\r--------------------------------\n\r" };
const UCHAR title2[] = { "\n\rNo Data!!\n\r" };
char   gUartBuffer[ 128 ];                          /* UARTより送信を行う際のバッファ */
USHORT gUartSendCount;                              /* UARTへ温度データを送った数     */
/* End user code for global definition. Do not edit comment generated here */

/*-------------------------------------------------------------------------------
**  Abstract:            This function is a callback function of UART0.
**  Parameters:          None
**  Returns:             None
**-------------------------------------------------------------------------------*/
void CALL_UART0_Send( void )
{
    /* Start user code. Do not edit comment generated here */
    if ( gUartStatus == D_SERIAL_SENDSTOP )
    {
        gUartStatus = D_SERIAL_SENDEND;
    }
    else if ( gUartStatus == D_SERIAL_CONTINUE )
    {
        gUartStatus = D_SERIAL_START;
    }
    /* End user code. Do not edit comment generated here */
}

/*-------------------------------------------------------------------------------
**  Abstract:            This function callback function open for users operation when IIC0 master error.
**  Parameters:          flag:                   status flag
**  Returns:             None
**-------------------------------------------------------------------------------*/
void CALL_IIC0_MasterError( MD_STATUS flag  )
{
    /* Start user code. Do not edit comment generated here */
    gI2cErrorflag = flag;
    gI2cStatus = D_I2C_OFF;
    /* End user code. Do not edit comment generated here */
}

/*-------------------------------------------------------------------------------
**  Abstract:            This function callback function open for users operation when IIC0 master receive finish.
**  Parameters:          None
**  Returns:             None
**-------------------------------------------------------------------------------*/
void CALL_IIC0_MasterReceiveEnd( void )
{
    /* Start user code. Do not edit comment generated here */
    SPT0 = 1;
    gI2cErrorflag = MD_MASTER_RCV_END;
    gI2cStatus = D_I2C_OFF;
    /* End user code. Do not edit comment generated here */
}

/*-------------------------------------------------------------------------------
**  Abstract:            This function callback function open for users operation when IIC0 master transmit finish.
**  Parameters:          None
**  Returns:             None
**-------------------------------------------------------------------------------*/
void CALL_IIC0_MasterSendEnd( void )
```

プログラム

Serial_user.c(リスト2)

```
{
    /* Start user code. Do not edit comment generated here */
    /* ストップコンディションの生成チェック */
    if ( gI2cStatus == D_I2C_WRITE )
    {   /* ストップコンディションの生成 */
        SPT0 = 1;
    }
    gI2cErrorflag = MD_MASTER_SEND_END;
    gI2cStatus = D_I2C_OFF;
    /* End user code. Do not edit comment generated here */
}
/* Start adding user code. Do not edit comment generated here */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* UARTで扱う変数初期化                                                        */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void uart_init()
{
    gUartStatus = D_SERIAL_OFF;
    gUartSendCount = 0;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* UARTでタイトルを送信                                                        */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS uart_sendtitle()
{
    MD_STATUS ret;
    USHORT len;
    UCHAR dat;

    if ( gI2cTemperatureCount != 0 )
    {
        gUartSendCount = 0;
        len = sizeof( title1 );
        UART0_SendData( title1, len );
        ret = MD_OK;
    }
    else
    {   /* 送るデータがないので通信終了 */
        len = sizeof( title2 );
        UART0_SendData( title2, len );
        ret = MD_ERROR;
    }
    return ret;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* UARTでデータ送信                                                            */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS uart_senddata()
{
    MD_STATUS ret = MD_ERROR;
    USHORT len;
    UCHAR dat;

    memset( gUartBuffer, 0x00, sizeof(gUartBuffer) );

    if ( gUartSendCount < gI2cTemperatureCount )
    {
        ret = i2c_eepromread( gUartSendCount );
        if ( ret == MD_OK )
        {
            /* 日付のフォーマットを設定 */
            memcpy( gUartBuffer, gLCDTIME, D_LCD_X_SIZE );  /* gLCDTIME = "2007/11/11 00:00" */
            dat = ( (gI2cReadBuffer[ 0 ] & 0xf0) >> 4 ) + '0';
            gUartBuffer[ 2 ] = dat;
            dat = (gI2cReadBuffer[ 0 ] & 0xf) + '0';
            gUartBuffer[ 3 ] = dat;
            dat = ( (gI2cReadBuffer[ 1 ] & 0xf0) >> 4 ) + '0';
            gUartBuffer[ 5 ] = dat;
            dat = (gI2cReadBuffer[ 1 ] & 0xf) + '0';
            gUartBuffer[ 6 ] = dat;
            dat = ( (gI2cReadBuffer[ 2 ] & 0xf0) >> 4 ) + '0';
            gUartBuffer[ 8 ] = dat;
            dat = (gI2cReadBuffer[ 2 ] & 0xf) + '0';
            gUartBuffer[ 9 ] = dat;
            dat = ( (gI2cReadBuffer[ 3 ] & 0xf0) >> 4 ) + '0';
            gUartBuffer[ 11 ] = dat;
            dat = (gI2cReadBuffer[ 3 ] & 0xf) + '0';
            gUartBuffer[ 12 ] = dat;
            dat = ( (gI2cReadBuffer[ 4 ] & 0xf0) >> 4 ) + '0';
            gUartBuffer[ 14 ] = dat;
            dat = (gI2cReadBuffer[ 4 ] & 0xf) + '0';
            gUartBuffer[ 15 ] = dat;
            gUartBuffer[ 16 ] = ' ';
            gUartBuffer[ 17 ] = ' ';
```

プログラム

Serial_user.c(リスト3)

```
            /* 温度データ1を格納 */
            sprintf( &gUartBuffer[ 18 ], "@%3d", gI2cReadBuffer[ 5 ] );
            gUartBuffer[ 22 ] = ' ';
            gUartBuffer[ 23 ] = ' ';
            /* 温度データ2を格納 */
            sprintf( &gUartBuffer[ 24 ], "@%3d\n\r", gI2cReadBuffer[ 6 ] );
            len = strlen( gUartBuffer );

            /* UART送信処理を行う */
            ret = UART0_SendData( (UCHAR *)gUartBuffer, len );
        }
        gUartSendCount++;
    }
    return ret;
}
/* End user code adding. Do not edit comment generated here */
```

ctl_i2c.c(リスト1)

EEPROMへアクセスするための処理がまとめられています。

EEPROMへ書き込みする際の内部管理フォーマットの規定です。

```
                        リスト省略
/* EEPROM に書き込むフォーマット
/* 0000h - 00007h   データヘッダ定義                                                    */
/*                  'KOR',0 識別コード 4 byte                                           */
/*                   xxxxh  データ個数 2 byte ( 0 - 65535 )                             */
/*                   xxxxh  予約       2 byte                                           */
/* 0008h - xxxxh     温度データ(8byte単位)
/*                   yy/mm/dd hh:mm 日付データ 5 byte
/*                   温度データ1 1 byte                                                 */
/*                   温度データ2 1 byte                                                 */
/*-------------------------------------------------------------------------------------*/
/*************************************************************************
** Include files
*************************************************************************/
#include "macrodriver.h"
#include "Serial.h"
#include "System.h"
#include "String.h"
#include "Stdlib.h"
#include "user_define.h"

/* in main.c */
extern void wait1msec( UINT para1_ );

/* in This file. */
MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *p_sadrs_, UINT length_ );
MD_STATUS i2c_randomread( USHORT dataadrs_, UCHAR *p_sadrs_ );
MD_STATUS i2c_read( UCHAR *p_sadrs_, UINT length_ );
MD_STATUS i2c_eepromread( USHORT datano_ );

/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern ULONG  gEventflag;

/* - - - - - - - - このファイルでの定義 - - - - - - - - */
UCHAR   gI2cStatus;                         /* I2C の通信ステータス                          */
UCHAR   gI2cReadBuffer[ D_EEPROM_BUF + 4 ];     /* I2C EEPROMからデータを読み出す際のバッファ    */
UCHAR   gI2cBuffer[ D_EEPROM_BUF + 4 ];         /* I2C EEPROMへコマンドを送る際のバッファ        */
USHORT  gI2cEepromWriteAdrs;                /* I2C EEPROMの書き込み実アドレス                */
USHORT  gI2cEepromReadAdrs;                 /* I2C EEPROMの読み込み実アドレス                */
MD_STATUS gI2cErrorflag;                    /* I2C 通信エラー時の状態                        */
UINT    gI2cTemperatureCount;               /* I2C EEPROMに保存されている温度データの数      */
const UCHAR gHeaderCheckString[] = D_EEPROM_HEAD;   /* EEPROMの最初に書き込むヘッダーの固定データ    */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* i2c関連の変数を初期化                                                   */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void i2c_initvaluie( void )
{
    gI2cStatus = D_I2C_OFF;
    gI2cErrorflag = MD_OK;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* EEPROMに登録されている温度データ数ヘッダを書き込む                      */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void i2c_eepromheaderwrite( void )
{
    UCHAR buf[ D_EEPROM_HEADLEN + 1 ];
    int flag = 0, i;

    memset( buf, 0x00, sizeof( buf ) );
    for ( i = 0; i < D_EEPROM_HEADMAX; i++ )
    {
        buf[ i ] = gHeaderCheckString[ i ];
    }
```

プログラム

ctl_i2c.c(リスト2)

```
    buf[ D_EEPROM_HEADMAX ] = (UCHAR)(gI2cTemperatureCount & 0xff );
    buf[ D_EEPROM_HEADMAX + 1 ] = (UCHAR)((gI2cTemperatureCount & 0xff00) >> 8);
    flag = i2c_write( 0, buf, D_EEPROM_HEADLEN );
    wait1msec( 5 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* EEPROMに登録されている温度データ数をチェックする                            */
/* EEPROMの0000h ～ 0007h をチェックし、温度データ数をチェックする             */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void i2c_eepromchk( void )
{
    UCHAR buf[ D_EEPROM_HEADLEN + 1 ];
    int flag = 0, i;

    memset( buf, 0x00, sizeof( buf ) );
    gI2cTemperatureCount = 0;
    flag = i2c_randomread( 0x0000, buf );
    wait1msec( 5 );
    if ( flag == MD_OK )
    {
        flag = i2c_read( &buf[ 1 ], (D_EEPROM_HEADLEN - 1) );
        wait1msec( 5 );
        if ( flag == MD_OK )
        {
            for ( i = 0; i < D_EEPROM_HEADMAX; i++ )
            {
                if ( buf[ i ] != gHeaderCheckString[ i ] )
                {
                    break;
                }
            }
            if ( i == D_EEPROM_HEADMAX )
            {   /* ヘッダチェックOKの場合 温度データありとする */
                gI2cTemperatureCount = buf[ D_EEPROM_HEADMAX ]
                                     + buf[ D_EEPROM_HEADMAX + 1 ] * 256;
            }
        }
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* EEPROMに登録されている指定された番号の温度データを読み込む                  */
/* 値は gI2cReadBuffer へ格納される                                            */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_eepromread( USHORT datano_ )
{
    MD_STATUS ret;

    memset( gI2cReadBuffer, 0x00, sizeof( gI2cReadBuffer ));

    gI2cEepromReadAdrs = ( datano_ * D_EEPROM_TEMPLEN ) + D_EEPROM_HEADLEN;
    ret = i2c_randomread( gI2cEepromReadAdrs, gI2cReadBuffer );
    wait1msec( 5 );
    if ( ret == MD_OK )
    {
        ret = i2c_read( &gI2cReadBuffer[ 1 ], (D_EEPROM_TEMPLEN - 1) );
        wait1msec( 5 );
    }
    return ret;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したバイト数、アドレスでEEPROMへ書き込みを行う                          */
/*                                                                             */
/* dataadrs_ :  EEPROMデータ格納アドレス                                       */
/* *p_sadrs_ :  書き込みデータ格納アドレスのポインタ                           */
/* length_   :  データ長                                                       */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *p_sadrs_, UINT length_ )
{
    UINT  cnt;
    MD_STATUS ret;

    memset( gI2cBuffer, 0x00, sizeof( gI2cBuffer ));
    gI2cEepromWriteAdrs = (USHORT)dataadrs_;

    gI2cBuffer[ 0 ] = (UCHAR)(( gI2cEepromWriteAdrs & 0xff00 ) >> 8);
    gI2cBuffer[ 1 ] = (UCHAR)(( gI2cEepromWriteAdrs & 0xff ));
    memcpy( &gI2cBuffer[ 2 ], p_sadrs_, length_ );
    gI2cStatus = D_I2C_WRITE;

    /* EEPROMのデバイスアドレスA0のみHIGH  */
    ret = IICO_MasterSendStart( D_EEPROM_CODE, gI2cBuffer, (length_ + 2), 1 );
    gI2cErrorflag = ret;
```

EEPROMから読み込みする際はランダムリードで1バイト読み込みを行ってからブロック読み込みします。

プログラム

ctl_i2c.c(リスト3)

```
    cnt = D_EEPROM_TIMEOUT;
    while ( gI2cStatus == D_I2C_WRITE )
    {
        cnt--;
        if ( cnt == 0 )
        {
            gI2cErrorflag = MD_RESOURCEERROR;
            ret = gI2cErrorflag;
            break;
        }
    };
    return ret;
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したアドレスでEEPROMより1バイト読み込みを行う                            */
/*                                                                             */
/* dataadrs_ :  EEPROMデータ格納アドレス                                       */
/*              読み込みを行うとEEPROMから読み出すアドレスも加算される         */
/* *p_sadrs_ :  読み込むデータ格納アドレスのポインタ                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_randomread( USHORT dataadrs_, UCHAR *p_sadrs_ )
{
    UINT cnt;
    MD_STATUS ret;

    gI2cEepromReadAdrs = (USHORT)dataadrs_;

    gI2cBuffer[ 0 ] = (UCHAR)(( gI2cEepromReadAdrs & 0xff00 ) >> 8);
    gI2cBuffer[ 1 ] = (UCHAR)(( gI2cEepromReadAdrs & 0xff ));

    ret = IICO_MasterSendStart( D_EEPROM_CODE, gI2cBuffer, 2, 1 );
    gI2cErrorflag = ret;
    gI2cStatus = D_I2C_READSET;
    if  ( ret == MD_OK )
    {
        cnt = D_EEPROM_TIMEOUT;
        while ( gI2cStatus == D_I2C_READSET )
        {
            cnt--;
            if ( cnt == 0 )
            {
                gI2cErrorflag = MD_RESOURCEERROR;
                ret = gI2cErrorflag;
                break;
            }
        };

        if ( !(gI2cErrorflag & MD_ERRORBASE) )
        {
            gI2cStatus = D_I2C_READDATA;

            ret = IICO_MasterReceiveStartRandom( D_EEPROM_CODE, p_sadrs_, 1, 1 );
            if  ( ret == MD_OK )
            {
                cnt = D_EEPROM_TIMEOUT;
                while ( gI2cStatus == D_I2C_READDATA )
                {
                    cnt--;
                    if ( cnt == 0 )
                    {
                        gI2cErrorflag = MD_RESOURCEERROR;
                        ret = gI2cErrorflag;
                        break;
                    }
                };
            }
        }
    }

    return ret;
}
```

プログラム

ctl_i2c.c(リスト4)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 指定したバイト数をEEPROMより読み込みを行う                                    */
/*                                                                              */
/* *p_sadrs_ :  読み込むデータ格納アドレスのポインタ                            */
/*              読み込みを行うとEEPROMから読み出すアドレスも加算される          */
/* length_   :  データ長                                                        */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
MD_STATUS i2c_read( UCHAR *p_sadrs_, UINT length_ )
{
    UINT cnt;
    MD_STATUS ret;

    ret = IIC0_MasterReceiveStart( D_EEPROM_CODE, p_sadrs_, length_, 1 );
    gI2cStatus = D_I2C_READDATA;
    if  ( ret == MD_OK )
    {
        cnt = D_EEPROM_TIMEOUT;
        while ( gI2cStatus == D_I2C_READDATA )
        {
            cnt--;
            if ( cnt == 0 )
            {
                gI2cErrorflag = MD_RESOURCEERROR;
                ret = gI2cErrorflag;
                break;
            }
        };
    }

    return ret;
}
```

ctl_lcd.c(リスト1)

LCDキャラクタディスプレイへ表示するための処理がまとめられています。

リスト省略

```
/*****************************************************
** Include files
*****************************************************************************/
#include "macrodriver.h"
#include "Int.h"
#include "RTC.h"
#include "System.h"
#include "String.h"
#include "Stdlib.h"
#include "user_define.h"

/* in main.c */
extern void wait100usec( UINT para1_ );
extern void wait1msec( UINT para1_ );

/* in This file. */
void lcd_temperature( void );

/* - - - - - - - - - - 他参照 - - - - - - - - - - */
/* in main.c */
extern ULONG  gEventflag;
extern USHORT gTemperatureAd;extern USHORT gTemperatureAd2;

/* - - - - - - - - このファイルでの定義 - - - - - - - - */
ST_LCDCMD  gtLcdCmd;                        /* LCDへコマンド送る際の構造体                  */
UCHAR   gLcdUpdate;                         /* LCD表示を更新するカウンタ                    */
UCHAR   gtLcdBuffer[ D_LCD_SIZE + 1 ];      /* LCDのキャラクタ単位のRAMバッファ 16文字x2行   */
UCHAR   gLcdCursor;                         /* LCDカーソル位置                              */
UCHAR   gLCDTIME[ D_LCD_X_SIZE ] = "2007/11/11 00:00"; /* 時刻を設定する際のLCD表示イメージ  */

const UCHAR gLCDTEXT[][ D_LCD_X_SIZE ] =
{
    " ｵﾝﾄﾞ 1:    2:   ",                    /* 文字列 1                                     */
    " ｼﾞｺｸ  ｾｯﾃｲ     ",                    /* 文字列 2                                     */
};

const UCHAR gtCgramData[] =
{   /* LCDへ定義する℃のフォントデータ */
    0x08, 0x14, 0x08, 0x07, 0x08, 0x08, 0x08, 0x07,
    0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x1f,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f
};
```

プログラム

ctl_lcd.c(リスト2)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDへデータ書き込み(8bitデータ送信)                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_write8bit( UCHAR rs_, UCHAR status_, UCHAR data_ )
{
    UCHAR  lcddata;

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 8bitデータバス */
    lcddata |= (data_ << 2);

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 1 );

    /* 動作起動をLOWにする */
    P8.1 = 0;
    wait100usec( 1 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDへデータ書き込み(4bitデータ送信)                                          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_write4bit( UCHAR rs_, UCHAR status_, UCHAR data_ )
{
    UCHAR  lcddata, dat;

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 上位4bitデータを送る */
    dat = (data_ & 0xf0) >> 1;
    lcddata |= dat;

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 2 );

    /* 動作起動をLOWにする */
    P8.1 = 0;
    wait100usec( 2 );

    /* RS : レジスタ選択信号 */
    lcddata = rs_;
    /* E  : 動作起動信号 */
    lcddata |= (status_ << 1);
    /* DB : 下位4bitデータを送る */
    dat = (data_ & 0x0f) << 3;
    lcddata |= dat;

    /* データを出力する */
    P8 = lcddata;
    wait100usec( 2 );

    /* 動作起動をLOWにする */
    P8.1 = 0;
    wait100usec( 2 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCD初期化                                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_init()
{
    int i;
    UCHAR dat;

    gLcdUpdate = 0;
    gLcdCursor = 0;
    memset( gtLcdBuffer, ' ', sizeof(gtLcdBuffer) );

    lcd_write4bit( 0, 1, 0x38 );        /* 8bitモードへ設定 */
    wait1msec( 100 );
    lcd_write8bit( 0, 1, 4 );           /* 4bitモードインターフェース設定  */

    gtLcdCmd.rs = 0;                    /* ファンクション行数セット        */
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_FUNC | D_LCD_2LINE;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 100 );
```

LCDへ送信するコマンドモードを変更します。

LCDへ送信するコマンドモードを変更します。

電源ON時でもデバッグ中のリセット時でも4bitデータ送信モードへ移行させる為の処理です。

プログラム

ctl_lcd.c(リスト3)

```
    /* エントリー・モード設定 */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_ENTRY | D_LCD_ENTRY_INC;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 5 );

    /* 表示ON, カーソルOFF, ブリンクOFF */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SET | D_LCD_SET_ON;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 5 );

    /* 表示クリア */
    lcd_write4bit( 0, 1, D_LCD_CLR );
    wait1msec( 10 );

    /* CGRAMへデータを転送する */
    lcd_write4bit( 0, 1, D_LCD_CGRAM );
    for ( i = 0; i < D_LCD_CGRAMSIZE; i++ )
    {
        /* CGRAMデータ書き込み */
        dat = gtCgramData[ i ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 温度表示                                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_temperature( )
{
    int i;
    UCHAR num1[ 7 ], num2[ 7 ];
    UINT temp1, temp2;

    /* 各センサより温度を計算 */
    temp1 = (gTemperatureAd * 48) / 400;
    temp2 = (gTemperatureAd2 * 48) / 400;

    itoa( temp1, (char *)num1, 10 );
    itoa( temp2, (char *)num2, 10 );

    for ( i = 0; i < D_LCD_X_SIZE; i++)
    {
        gtLcdBuffer[ i + D_LCD_X_SIZE ] = gLCDTEXT[ 0 ][ i ];
    }
    gtLcdBuffer[ 7 + D_LCD_X_SIZE ] = num1[ 0 ];
    gtLcdBuffer[ 8 + D_LCD_X_SIZE ] = num1[ 1 ];
    if ( num1[ 1 ] != 0 )
    {
        gtLcdBuffer[ 9 + D_LCD_X_SIZE ] = 0;        /* 「°C」表示 */
    }
    gtLcdBuffer[ 13 + D_LCD_X_SIZE ] = num2[ 0 ];
    gtLcdBuffer[ 14 + D_LCD_X_SIZE ] = num2[ 1 ];
    if ( num2[ 1 ] != 0 )
    {
        gtLcdBuffer[ 15 + D_LCD_X_SIZE ] = 0;       /* 「°C」表示 */
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDカーソルホームポジション移動                                             */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_curhome()
{
    /* カーソルをホームへ移動 */
    lcd_write4bit( 0, 1, D_LCD_HOME );
    wait1msec( 10 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDブリンクON                                                               */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_blinkon()
{
    /* 表示ON, カーソルOFF, ブリンクON */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SET | D_LCD_SET_ON | D_LCD_SET_BLINK;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 20 );
}
```

LCD表示バッファへ温度データを設定します。

プログラム

ctl_lcd.c(リスト4)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDブリンクOFF                                                              */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_blinkoff()
{
    /* 表示ON, カーソルOFF, ブリンクOFF */
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SET | D_LCD_SET_ON;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 20 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDカーソル右移動                                                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_cursorright()
{
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SHIFT | D_LCD_SHIFT_RIGHT;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 10 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDカーソル左移動                                                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_cursorleft()
{
    gtLcdCmd.rs = 0;
    gtLcdCmd.sts = 1;
    gtLcdCmd.data = D_LCD_SHIFT;
    lcd_write4bit( gtLcdCmd.rs, gtLcdCmd.sts, gtLcdCmd.data );
    wait1msec( 10 );
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 0～9入力処理                                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_0_9_input( UCHAR *num_, UCHAR move_ )
{
    if ( (*num_ >= '0') || (*num_ <= '9') )
    {
        if ( move_ == D_LCD_CURUP )
        {
            if (*num_ != '9')
            {
                (*num_)++;
            }
        }
        else
        {
            if (*num_ != '0')
            {
                (*num_)--;
            }
        }
    }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 0～1入力処理                                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_0_1_input( UCHAR *num_, UCHAR move_ )
{

    if ( (*num_ >= '0') || (*num_ <= '1') )
    {
        if ( move_ == D_LCD_CURUP )
        {
            if (*num_ != '1')
            {
                (*num_)++;
            }
        }
        else
        {
            if (*num_ != '0')
            {
                (*num_)--;
            }
        }
    }
}
```

時刻設定を行う際の入力で0～9までの入力制限します。

時刻(日付)設定を行う際の入力(?0月)で0～1までの入力制限します。

プログラム

ctl_lcd.c(リスト5)

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 0～2入力処理                                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_0_2_input( UCHAR *num_, UCHAR move_ )
{
    if ( (*num_ >= '0') || (*num_ <= '2') )
    {
        if ( move_ == D_LCD_CURUP )
        {
            if (*num_ != '2')
            {
                (*num_)++;
            }
        }
        else
        {
            if (*num_ != '0')
            {
                (*num_)--;
            }
        }
    }
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 0～3入力処理                                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_0_3_input( UCHAR *num_, UCHAR move_ )
{
    if ( (*num_ >= '0') || (*num_ <= '3') )
    {
        if ( move_ == D_LCD_CURUP )
        {
            if (*num_ != '3')
            {
                (*num_)++;
            }
        }
        else
        {
            if (*num_ != '0')
            {
                (*num_)--;
            }
        }
    }
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* 0～5入力処理                                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_0_5_input( UCHAR *num_, UCHAR move_ )
{
    if ( (*num_ >= '0') || (*num_ <= '5') )
    {
        if ( move_ == D_LCD_CURUP )
        {
            if (*num_ != '5')
            {
                (*num_)++;
            }
        }
        else
        {
            if (*num_ != '0')
            {
                (*num_)--;
            }
        }
    }
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDカーソル上下移動処理                                                     */
/* 時刻設定処理 "2007/11/11 00:00" のカーソル位置によって入力値を変える       */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_cursorupdown( UCHAR direction_ )
{
    switch ( gLcdCursor )
    {
        case 2: /* 年の設定 */
        case 3:
            lcd_0_9_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;

        case 5: /* 月の設定(十の位) */
            lcd_0_1_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;
```

時刻設定を行う際の入力(?0時)で0～2までの入力制限します。

時刻(日付)設定を行う際の入力(?0日)で0～3までの入力制限します。

時刻設定を行う際の入力(?0分)で0～5までの入力制限します。

プログラム

ctl_lcd.c(リスト6)

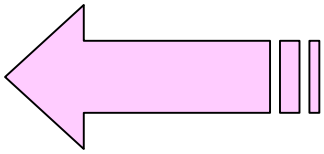
```
        case 6: /* 月の設定(一の位) */
            if ( gtLcdBuffer[ gLcdCursor-1 ] == '0' )
            {
                lcd_0_9_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            else
            {
                lcd_0_2_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            break;

        case 8: /* 日の設定(十の位) */
            lcd_0_3_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;

        case 9: /* 日の設定(一の位) */
            if ( gtLcdBuffer[ gLcdCursor-1 ] == '3' )
            {
                lcd_0_1_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            else
            {
                lcd_0_9_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            break;

        case 11: /* 時の設定(十の位) */
            lcd_0_2_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;

        case 12: /* 時の設定(一の位) */
            if ( gtLcdBuffer[ gLcdCursor-1 ] == '2' )
            {
                lcd_0_3_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            else
            {
                lcd_0_9_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            }
            break;

        case 14: /* 分の設定(十の位) */
            lcd_0_5_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;

        case 15: /* 分の設定(一の位) */
            lcd_0_9_input( &gtLcdBuffer[ gLcdCursor ], direction_ );
            break;
        default :
            break;
    }
    gEventflag |= D_EV_LCD2RAM;
    gEventflag |= D_EV_LCDCURMOVE;
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* LCDバッファVRAMからLCD VRAMへ転送                                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void lcd_vram2lcd()
{
    UINT i;
    UCHAR dat;

    /* DDRAMアドレス設定 */
    lcd_write4bit( 0, 1, D_LCD_DDRAM );
    wait1msec( 1 );

    for ( i = 0; i < (D_LCD_SIZE/2); i++ )
    {
        /* DDRAMデータ書き込み */
        dat = gtLcdBuffer[ i ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }
    /* DDRAMアドレス設定 */
    lcd_write4bit( 0, 1, D_LCD_DDRAM + 0x40 );
    wait1msec( 1 );

    for ( i = 0; i < (D_LCD_SIZE/2); i++ )
    {
        /* DDRAMデータ書き込み */
        dat = gtLcdBuffer[ i + D_LCD_X_SIZE ];
        lcd_write4bit( 1, 1, dat );
        wait100usec( 1 );
    }
    wait1msec( 1 );
}
```

LCDバッファからLCDのVRAMへ転送を行うことで実際に表示されます。

次へのステップ

この章ではプログラムの設計～開発におけるポイントを説明します。前書「チュートリアルガイド」でもプログラムの作り方として説明しましたが、ここでは、もう少し踏み込んで章として独立させました。

はじめに
製作するシステム
製作手順
全体回路図
ソフトウエアの作成
次へのステップ
[開発前に考えること]
[本書でのコーディング記述]
[更に進んだ設計]
[省電力処理]
[最後に]
付録



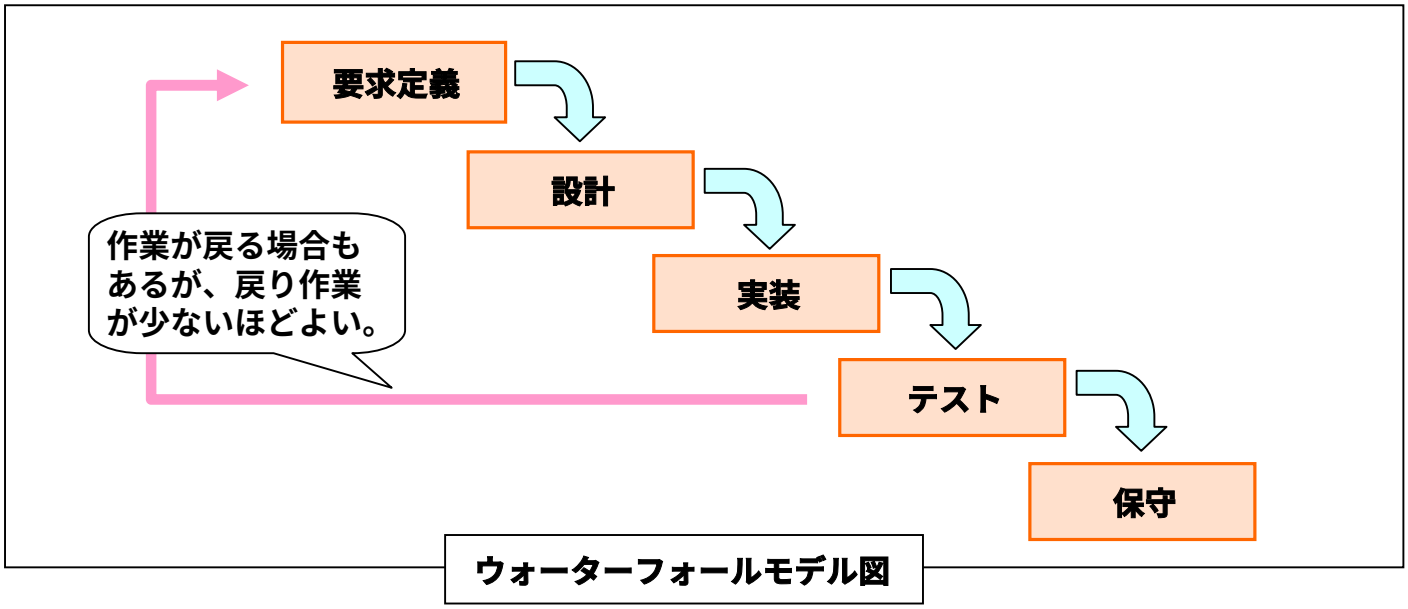
開発前に考えること(1)

開発前に考えることで大事なことはなんでしょう？ここでは、下記の問いを考えてみます。
・開発手法は何を使えばいいですか？
・資源の見積もりが難しいです
・要求の変化に耐えられません

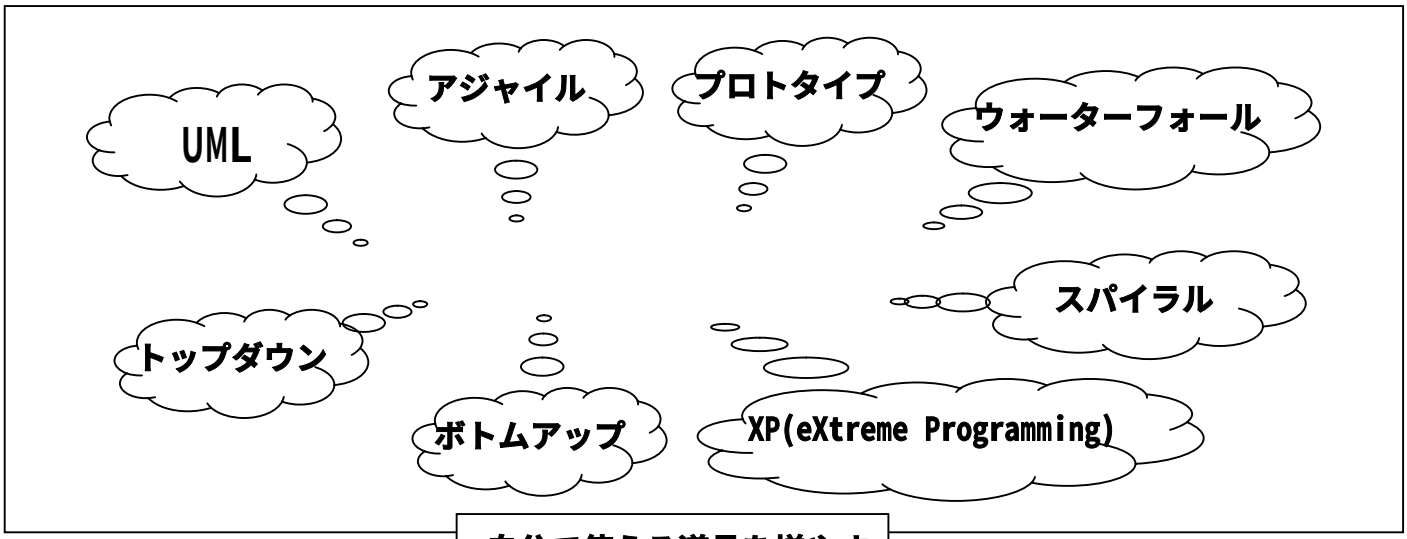
Q.開発手法は何を使えばいいですか？

昔からの開発手法にウォーターフォールモデルがあります。簡単に説明すると次のような順で開発を行うことを示します。「要求定義、設計、実装、テスト、保守」ただ、問題も指摘されており要求定義が曖昧だと戻りの作業が多く発生します。その他の開発手法としてスパイラルモデル、XP、アジャイルソフトウエア開発など多種多様あります。

では何がいいのか？答えは目的によって異なります。**目的、仕様が明確ならウォーターフォールモデルでも充分機能しますし**、1人で最初から最後までの工程を行うなら、それが最適と言えます。重要なのは**自分で使える道具を増やすことと、自らの成功パターンがあるならそれに則って開発する**ことです。



ウォーターフォールモデル図

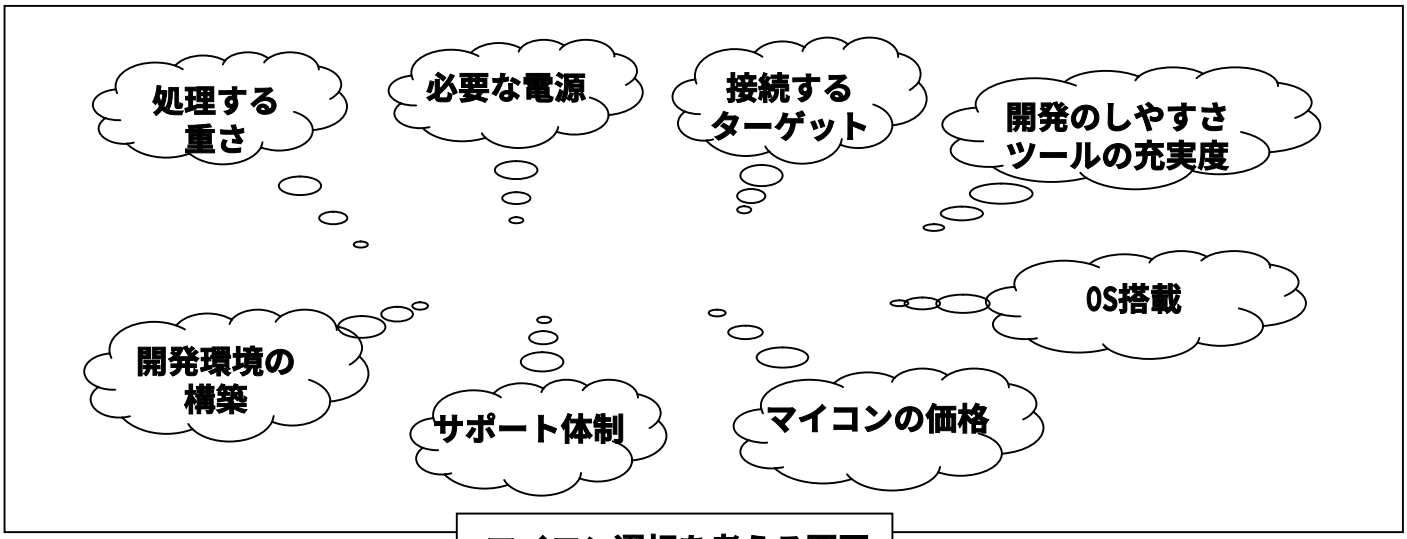


自分で使える道具を増やす

開発前に考えること(2)

Q.資源の見積もりが難しいです

マイコンには種類が豊富で選択に迷うこともしばしばあると思います。ポイントとしては
・必要とする電源(1.8V～5.5V)
・処理の重さ(8/16/32ビット)
・扱う周辺機能
・用意されているツール(本書で扱っているApplilet2等の便利なツールがあるか?)
おおまかには上記を考えればOKです。
まず、処理の重さを考え8/16/32ビットを選択します。開発時には周辺機能の豊富な、ROM容量など大きめのマイコンを使います。開発が進んだ段階で必要ない機能を見極めてROM容量/周辺機能を絞ったマイコンに置き換えればよいのです。



マイコン選択を考える要因

ワンポイント

開発環境の構築について
「MINICUBE2」http://www.necel.com/micro/ja/development/asia/minicube2/minicube2.html
最も簡単に弊社のマイクロコントローラ開発が始められます。また、このMINICUBE2に接続可能なオプションのCPUボードは、8～32ビットマイクロコントローラの代表的なラインアップから1つ選択されています。CPUボードは入手性も優れていますのでマイコン選択に迷ったら、これらのボードを入手して考えるとよいでしょう。

開発前に考えること(3)

Q.要求の変化に耐えられません

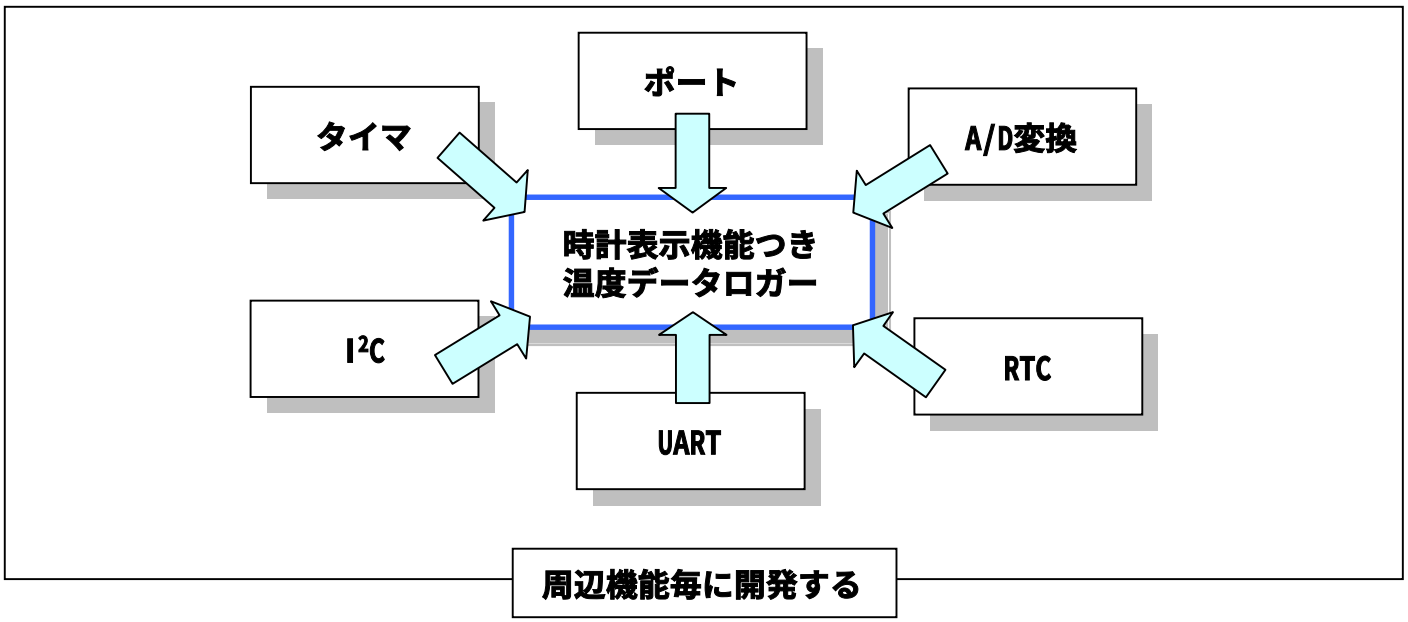
仕様の追加/変更など、開発にはつきものです。開発手法でウォーターフォールモデルについて書きましたが、実践は難しいと思います。なぜなら仕様変更が発生する度に設計、実装の戻り作業が発生するためです。では、どのような開発がよいのでしょうか？チュートリアルガイドで実践した開発手法は、周辺機能の1つ1つについてプログラムを完成させ、テストを行っています。この小さい単位で開発する手法はXP(eXtreme Programming)に近いです。この手法の利点として下記のことがあげられます。

・プログラム単位が小さいので変化に強い
・1つの周辺機能で閉じて開発するのでテストもしやすい
・閉じた開発がいくつもあるので並行して複数の人員で同時開発可能

一方、懸念する点もあります。

・最初の設計が肝心
・複数の人員で開発する場合、意志の疎通、コーディングルールも統一しにくい

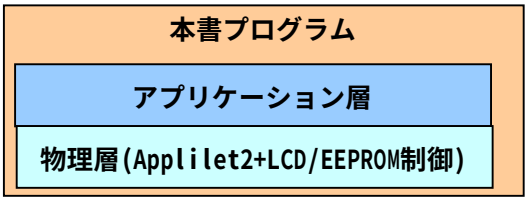
以上に注意すれば、有効な開発手法の1つになるでしょう。



ワンポイント

プログラムの階層構造について

プログラムを作る際は階層構造にするべきです。OSI参照モデルまで細かくする必要はありませんが、少なくとも物理層(ハードウエアにアクセスする層)とアプリケーション層は分けるべきです。そうすれば、作成したプログラムを他のマイコンに移植する際は物理層だけを入れ替えれば大丈夫になります。本書ではApplilet2を使用しているので、基本的な階層構造はできています。



本書でのコーディング記述(1)

コーディング記述は設計と同じように重要です。世の中には設計書のないソフトウエアは数多く存在します。その時に頼りになるのはソースのみです。勿論、設計をせずにソフトウエアを書くことはお勧めしませんが、少なくとも1つのプロジェクトでコーディング記述は統一されなければなりません。ただ、統一されるのではなく、意志を持って統一することが重要です。本書では、どのような考えでコーディング記述を考えているか説明します。

1.マクロ名は大文字にします。
すぐに見分けがつくようにします。また#defineで定義したものは接頭語として”D_”を付加しています。ユーザー定義は[user_define.h]にまとめて記述しています。

```
例･･･ #define D_EV_INTP0          0x0000001    /* INTP0押下 */
      #define D_EV_SWCHECK        0x0000080    /* SW値のA/Dを読み込む */
      #define D_EV_UARTSEND       0x0008000    /* UART送出処理中 */
```

上記の定義はdefineを意味し、EVはイベント系の定義、最後に内容を示しています。

2.変数名の定義。
ローカル変数は全て小文字で構成します。また、グローバル変数には接頭語として”g”を付加します。またポインタ変数には”p”を付加します。構造体/配列変数には”t”をつけます。
これはApplilet2が生成するコードにも共通します。

```
例･･･ g1msecCounter    → グローバル変数
      gpUserBuffer     → グローバルポインタ変数
      gtLcdBuffer[32]  → グローバル配列変数
```

3.改行の定義。
中カッコ{}は独立して改行します。インデントと中カッコの位置で処理の単位を見易くするためです。現在の開発環境において、なるべく改行をつめる意味はなくなっています。

```
例･･･ if ( xx ) {                          if ( xx )
      }                  ──────→           {          → 改行します
                                           }

      for ( xx ) {                         for ( xx )
      }                  ──────→           {          → 改行します
                                           }
```

中カッコのインデントを揃えて、処理の単位を見易くしています。

4.関数の引数についての定義。
関数の引数には最後に”_”(アンダーバー)をつけます。これは、ローカル変数、グローバル変数引数を明確に区別するためです。また、引数に戻り値がある場合は必ずポインタ変数にします。
下記のように関数の戻り値の有無が一目で理解できると思います。

```
例･･･MD_STATUS i2c_write( USHORT dataadrs_, UCHAR *p_sadrs_, UINT length_ )
     dataadrs_ → 戻り値なし
     *p_sadrs_ → 戻り値あり
     length_   → 戻り値なし
```

本書でのコーディング記述(2)

5.関数の作り方について。
関数の入口、出口は1箇所にします。なぜでしょうか？
例･･･メモリリークをしにくくする

```
void function( UCHAR para_ )
{
    size_t *psize;
    psize = malloc(100);
    if ( psize == NULL )
        return;
    if ( para_ == 0 )
        return;
        :
        :
    free( psize );
}
```

関数の引数によって、メモリリーク(メモリ解放のし忘れ)が発生している。

```
void function( UCHAR para_ )
{
    size_t *psize;
    psize = malloc(100);
    if ( psize != NULL )
    {
        if ( para_ != 0 )
        {
            :
            :
        }
        free( psize );
    }
}
```

関数途中で終了することがないので、領域確保した場合は、必ず領域解放を行う処理になっている。

例･･･デバッグしやすくする。下記のように値を返す関数をデバッグすることを考えます。

```
int function( void )
{
    switch ( xx )
    {
        case xx :
            return 1;
            break;
        case xxx :
            return 2;
            break;
        default :
            return 3;
            break;
    }
    retrun 0;
```

関数の戻りを確認するには、それぞれの箇所にブレークを張る必要がある。

```
int function( void )
{
    int ret = 0;
    switch ( xx )
    {
        case xx :
            ret = 1;
            break;
        case xxx :
            ret =2;
            break;
        default :
            ret =3;
            break;
    }
    retrun ret;
```

関数の戻りを確認するには、1箇所のみにブレークを張ればよくなった。

上記の例は少し極端かもしれませんが、とても有効な方法です。また、どうしても関数途中で抜け出す場合は、関数分割や、アルゴリズム変更などで、入口/出口を1つにすることを勧めます。

ワンポイント

プログラムの作り方について
前書「 78K0R/KG3ターゲット･ボード QB-78K0RKG3-TB チュートリアル･ガイド」の[プログラムの作り方」章も参考にしてください。ここに書いていないことにも触れています。「ハードウエアのアクセスは最小限にする」、「1つの関数は多くても100行程度にする」など。

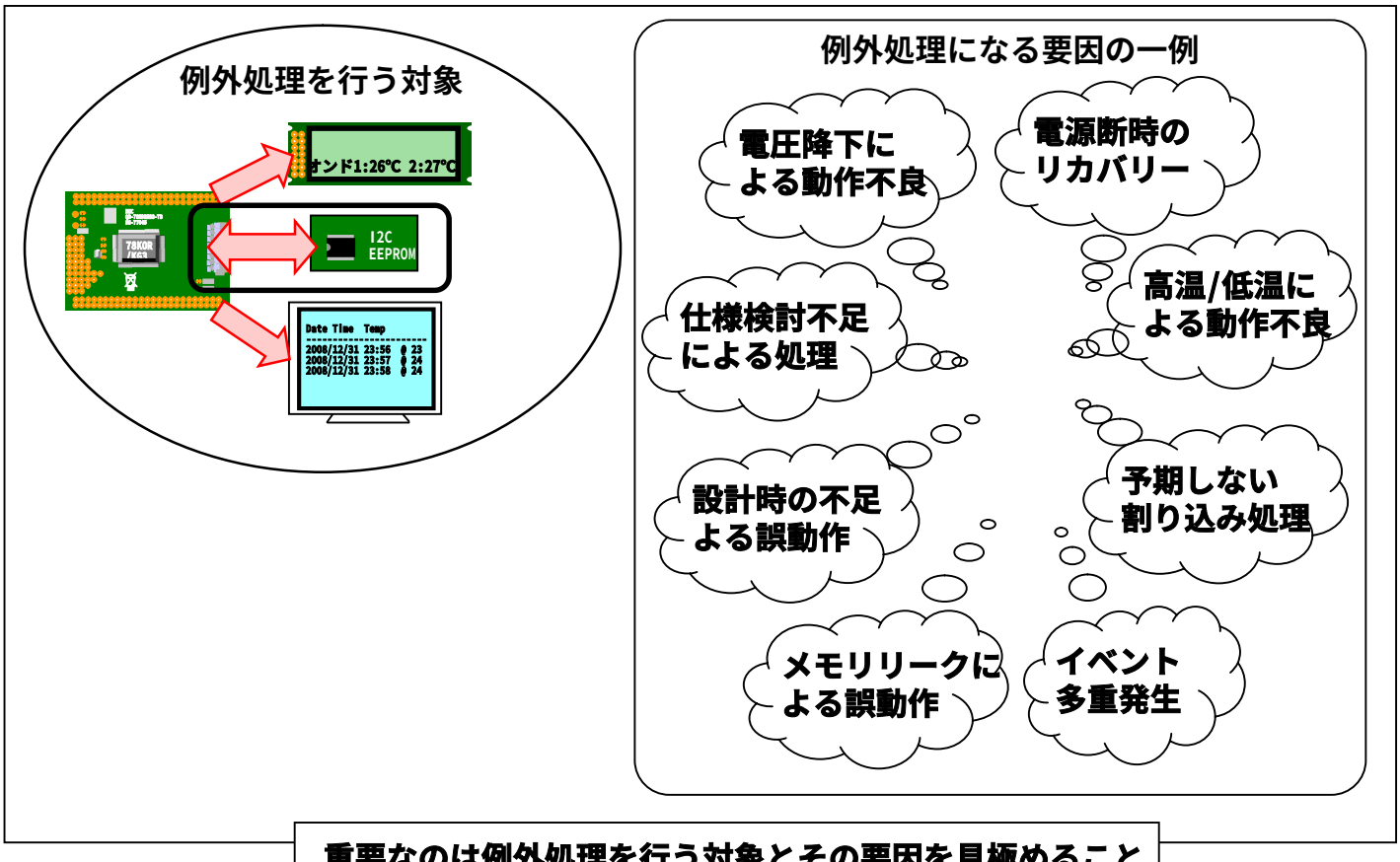
更に進んだ設計

本書のプログラムには考慮していない点が2つあります。
・例外処理
・省電力
これらを考慮して製品としてプログラムが完成したと言えます。本書プログラムの場合、何を考慮すればよいのか具体的に説明します。

例外処理とは

例外処理とは、正常動作以外の処理を指します。本書プログラムでは「EEPROMアクセス中に電源断になった場合のリカバリー処理」が考慮されていません。これを考慮するには
・EEPROM書き込み中に電源断になった場合のデータの整合性チェックする
・データ整合性チェックによってデータ個数など温度のデータヘッダーを修正する
などの処理が必要です。
注意する点はデータ出力時に、捨ててもいいデータかどうかを見極める事です。本プログラムでUART出力時にリカバリー処理は考慮しなくても構いません。それはUARTでデータをPCに送っても確認用にしか利用していないからです。仮にPC側で送られてきたデータを利用する運用になっていたらマイコン側からデータを再送信することも考えられます。
「どの処理をリカバリーする必要があるか？」
は重要です。その他にも電圧降下の監視など製品プログラムなら必要です。

例外処理は、プログラムの中でも最も難しい処理と言えます。この難しい処理を見極めるため、様々な開発手法が考えられていると言っても過言ではありません。



重要なのは例外処理を行う対象とその要因を見極めること

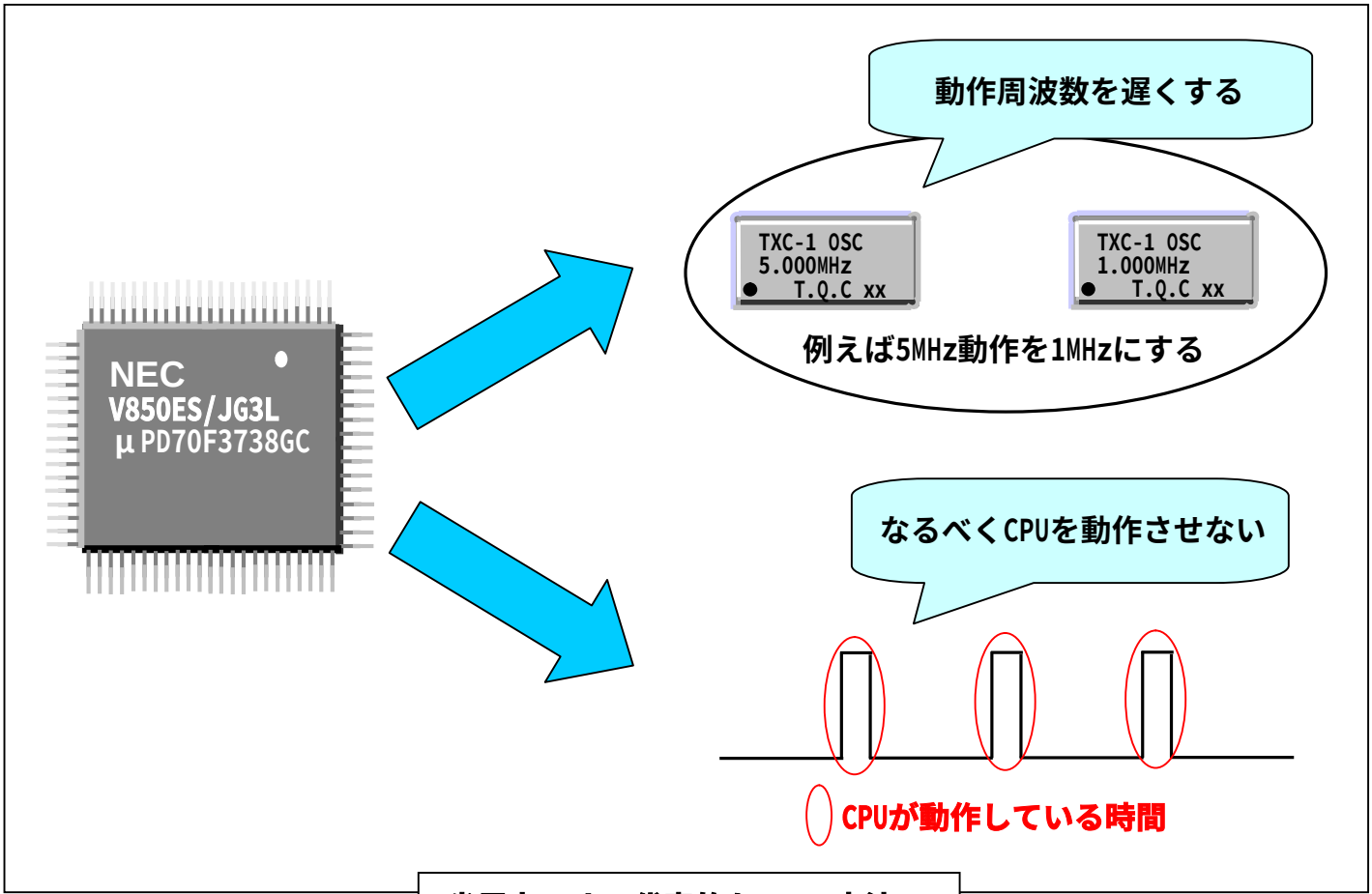
省電力処理

省電力処理とは

マイコンはいつも100%で処理をしていません。ほとんどが割り込み待ち、イベント待ちの処理です。100%連続で使う代表的な処理は、画像の圧縮/展開などです。本書のプログラムでは、そのような処理はありませんので、マイコンの実働時間は非常に短いです。そこで空いてる時間(割り込み待ち、イベント待ち)はスタンバイ機能でHALTモードにすると消費電力を大幅に減らすことができます。このようなスタンバイ処理は、製品化する上では一般的と思ってください。

省電力処理で注意しなければならない点があります。
・HALTモードから復帰するには数マイクロ秒～数ミリ秒かかる場合がある。
・HALTモードに入るタイミングを重要な処理の前にならないように考慮する。

省電力処理は電池を使ったターゲットシステムでは避けて通れません。このプログラムを作成するには構築するシステムの理解を深めることが大切です。マイコンをHALTモードにすれば消費電力を下げられますが、マイコンだけでなく動作させる周辺機能にも注意してください。消費される電力を見極めることが重要です。



省電力にする代表的な2つの方法

最後に

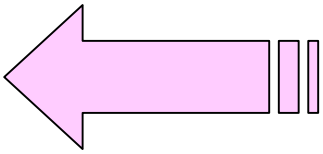
マイコンは組み込みシステムで使われます。この組み込みシステムを作るのにはハードウエア、ソフトウエア両方の知識が不可欠です。本書に掲載しているシステムでさえ、複雑なものになっていると思われた方も多いのではないでしょうか？

これらの知識を深めるのにはどうすればいいのか？近道はありません。ただ、実践する(実際に体験する)とより理解が早まります。「製作手順」の章など項目ごとに独立しているのでシステムの全部を製作しなくとも試せることは多いです。是非、実際に試して知識を深めて頂ければ幸いに思います。

付録

この章では、本書に掲載しているプログラムのURLや参考になるURLなどの資料の一覧を掲載します。

はじめに
製作するシステム
製作手順
全体回路図
ソフトウエアの作成
次へのステップ
付録
[参考資料]

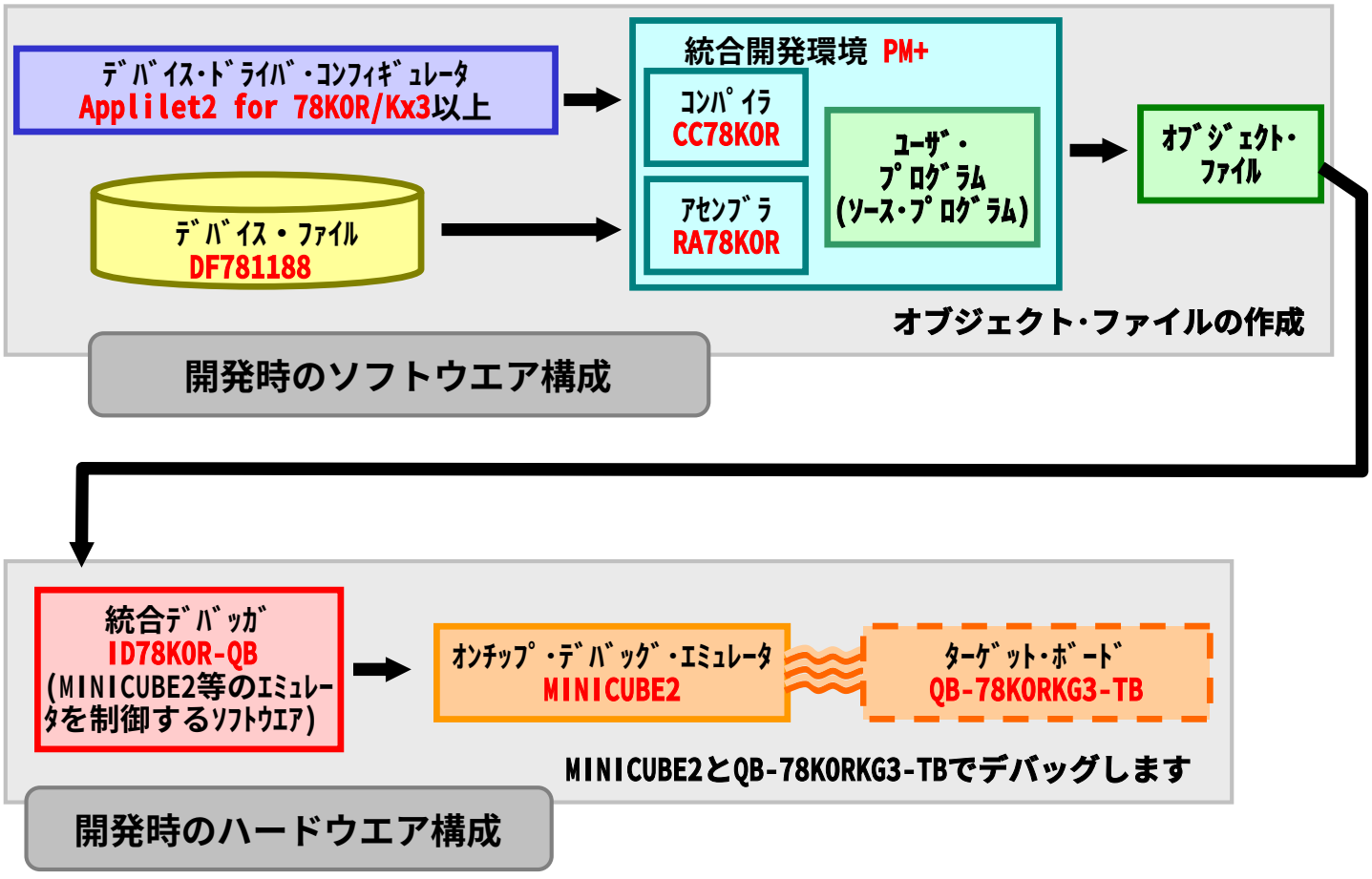


参考資料

78K0R開発環境のダウンロード

http://www.necel.com/micro/ja/freesoft/78k0r/kx3/
下記ファイルがダウンロードできます。
・RA78K0R(統合開発環境PM+を含みます)
・CC78K0R
・78K0R/Kx3用 Applilet2
・78K0R/Kx3用 デバイス・ファイル

https://www5.necel.com/micro/tool_reg/0dsListTool.do?code=460&lang=ja
下記ファイルがダウンロードできます。
・ID78K0R-QB



MINICUBE2の情報について

http://www.necel.com/micro/ja/development/asia/minicube2/minicube2.html
MINICUBE2の情報については、このURLにまとめられています。最新ユーザーズマニュアル、ファームウエア、また各種CPUボード(ターゲット･ボード)のプログラムも掲載しています。
MINICUBE2はWEBを通して情報提供しますのでチェックすることをお勧めします。

参考資料

マイコンについての基礎が理解できます。

http://www.necel.com/micro/ja/campaign/allflash-zemi/

マイコンの基礎知識を解りやすく説明します。講座形式で1回ごとにテーマを決めて説明します。

- 第1回　マイコン
- 第2回　CPU
- 第3回　メモリ
- 第4回　ポート
- 第5回　タイマ
- 第6回　A/Dコンバータ
- 第7回　シリアル・インタフェース
- 第8回　割り込み

マイコンについて基礎～中級の知識が学習できます。

http://www.necel.com/micro/ja/eLearning/

学習形式でマイコンの基本が理解できます。「基礎編」「初級編」「中級編」で構成されております。

「基礎編」

- 基礎編はマイコンを学習するうえで必要となる２進数や論理回路等の基礎知識を習得する事ができるコースです。学習効果を高めるために各章ごとに問題を用意しています。

「初期編」

- 初級編はマイコンをよく知らないかた向けの入門編です。小ピンマイコン（78K0S/Kx1+）を題材に簡単ツールを使ってプログラム作成ができます。

「中級編」

- 中級編はマイコンをより詳しく知りたいかた向けのコースです。78K0/KE2マイコンを題材に、搭載されている周辺機能をどのように使うか学習することができます。章ごとに簡単なプログラム実習がついています。

川崎、名古屋、大阪では「マイコンセミナ」も開催されております。無料セミナもありますので、興味のある方は下記URLをご参照ください。

http://www.necel.com/seminar/ja/

参考資料

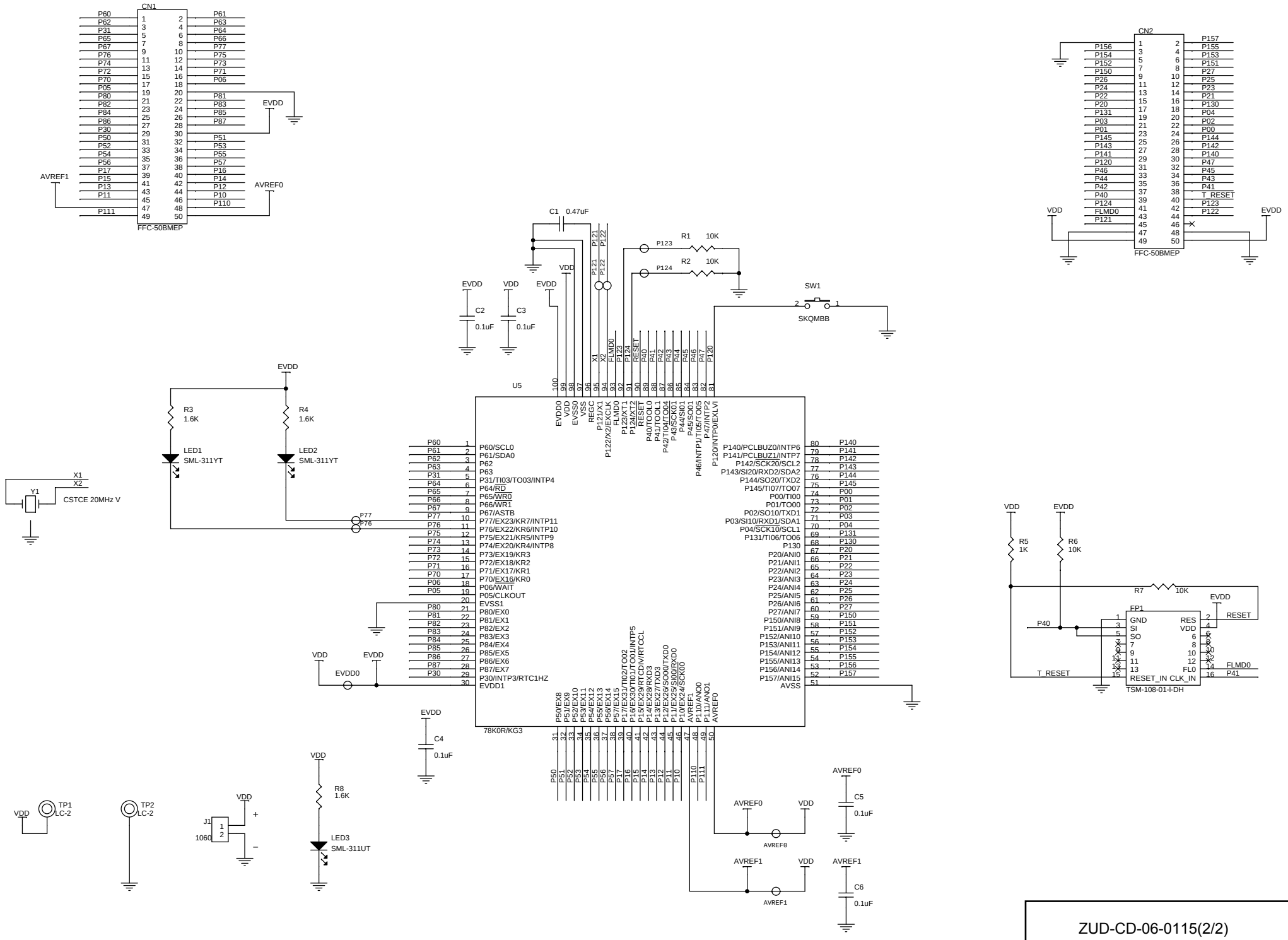
A/Dを用いたキースキャン

http://www.necel.com/micro/ja/designsupports/sampleprogram/78k0/
上記の例では、6x8個のキー･スキャン入力を行っています。「A/DでSW入力」では、この1回路を用いて6個のSW入力を実現しています。

I2Cに関して(FAQより)

http://www.necel.com/ja/faq/mi78k/__78iic.html
弊社のWEB FAQにI2Cバスに関しての現在下記の項目について記述があります。
- I2Cバスについての概要
- I2Cバスのプルアップ抵抗値は？
- I2Cバスの初期設定で STT0と SPT0を同時にセットするとうまくゆかない。
- 外部の EEPROMに I2Cバスで接続する場合の処理例(マスタ)
- I2Cバスが起動できない [78K共通]
- I2Cバスの制御がうまくいかない [共通]
- I2Cのクロック周波数が設定と異なる [共通]
- EEPROMの読み出しはうまくいったが、その後スタート・コンディションがうまくいかない [共通]
- 簡易I2Cバスとは？ [78K0R共通]
- 簡易Ｉ２Ｃ機能（78K0R）
- I2Cバスへの不要な信号出力。[78K0/Kx]

FAQでは、よくある質問などマイコン、開発ツール、またマイコン以外のNECエレクトロニクスの製品の事なら何でも掲載されています。困った時は、まず「FAQ」。
http://www.necel.com/ja/faq/


CN1
FFC-50BMEP
CN2
FFC-50BMEP
U5
78K0R/KG3
C1 0.47uF
C2 0.1uF
C3 0.1uF
C4 0.1uF
C5 0.1uF
C6 0.1uF
R1 10K
R2 10K
R3 1.6K
R4 1.6K
R5 1K
R6 10K
R7 10K
R8 1.6K
SW1
SKQMBB
LED1 SML-311YT
LED2 SML-311YT
LED3 SML-311UT
X1
X2
Y1
CSTCE 20MHz V
FP1
TSM-108-01-I-DH
TP1 LC-2
TP2 LC-2
J1
EVDD
VDD
AVREF0
AVREF1
T_RESET
RESET
FLMD0
ZUD-CD-06-0115(2/2)