デバッグモニター
演習マニュアル

H8/300シリーズ用

第1.02 版
2011. 08. 21
タカシステム

Ver	改訂日	改訂内容
1.00	2011.07.07	初版
1.01	2011.08.05	“11.6. 割り込み関数処理の補足説明”を追加 “(3)gccがインストールが出来ないとき”を追加
1.02	2011.08.21	“(1)cygwin起動に″invalid fstab option – ‘0’ ”が表示する時の対処法”を追加

注 意 事 項

著作権
・本マニュアルに関する著作権はタカシステムに帰属します。

禁止事項
ユーザーは以下の内容を行うことはできません。
・第三者に対して、本マニュアルを販売、販売を目的とした宣伝、使用、営業、複製などを行うこと
・第三者に対して、本マニュアルの使用権を譲渡または再承諾すること
・本マニュアルの一部または全部を改変、除去すること
・本マニュアルを無許可で翻訳すること
・本マニュアルの内容を使用しての、人命や人体に危害を及ぼす恐れのある用途での使用

転載、複製
本マニュアルの転載、複製については、文書によるタカシステムの事前の承諾が必要です。

責任の制限
デバッグモニター実行による損失・損害が生じた場合でも、タカシステムはその責任を負いません。
又、本マニュアルに記載した内容は慎重に記述・制作したものですが、万一本マニュアルの記述誤りに起因する損失・損害が生じた場合でも、タカシステムはその責任を負いません。
あらかじめご了承の上でご使用下さい。

その他
本マニュアルに記載の情報は本マニュアル発行時点のものであり、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。

連絡先
E-mail：takasystem@nifty.com

評価期間
使用開始から90日間は評価期間です。
評価期間中は機能制限は有りません。評価期間後は登録できるプロジェクトは1プロジェクトで”組み込み関数機能”が使用できなくなります。

本マニュアルの内容を予告なく変更することがあります。

■開発コンセプト

1. どのようなデバッグモニターにするか
   ・組み込み機器開発のソースデバッグができて、結果が即確認できること。
   ・プログラム作成中の不具合発見が容易にできること。
   ・周辺機器を接続したときに簡単な確認ができること。
   ・対象MCUのシリーズに最大限対応できること。
   ・学習から本格的な開発までに対応できること。

2. 誰を対称にするか
   ・プログラム作成とC言語の知識が多少有り、組み込み機器の知識を習得したい方。
   ・MCUの基本知識を習得したい方。
   ・実際に組み込み機器開発している方で、Cソース/アッセンブラソースデバッグをしたい方。

3. メリットは
   ・最小限度のコストで組み込み機器の学習が出来ること。
   ・ソースデバッグが可能になること。

4. 導入コストは
   ソフト/ハードを合わせたトータルコストが小額の出費ですむ様にする（1万～2万円程度）。

5. 対象MCUは
   ・市販されている完成品ボードのMCUを対象にする。（接続するれば即開始できる）
   ・雑誌等の付録のMCUを対象にする。
   ・過去に多く市販されたMCUを対象にする。（10年位前まで）
   ・今後メーカが出すMCUも対象にする。

6. 開発環境の整備は
   ・コンパイラは”cygwin”の”gcc”を使用する。
   ・対象MCUの″gcc”をインストールするための、スクリプトファイルを供給する。
     基本はソースをコンパイルして、開発者の環境を整備してもらう。
   ・ユーザーのPCの使用状況に影響を与えず導入評価ができる様にする。
     ”cygwin”を導入すると使用中の”make”が使用できなくなることがある。

目　次

■開発コンセプト

1. はじめに

デバッグモニタは、ターゲットボード上の外部RAM又MPUの内臓RAMに、デバッグ対象プログラムをダウンロードして実行解析するツールです。
外部RAM(数十Kバイト～数百Kバイト)が有れば、プログラムを外部RAMにダウンロードして本格的な開発が可能になります。
ボード毎に異なる外部RAM仕様に対応する為に、組込み関数と呼ぶ機能を持っています。
この機能を使用する為には内臓RAMが4KB以上のMPUが対象になります。

2. 開発に必要なもの

2.1. システム構成

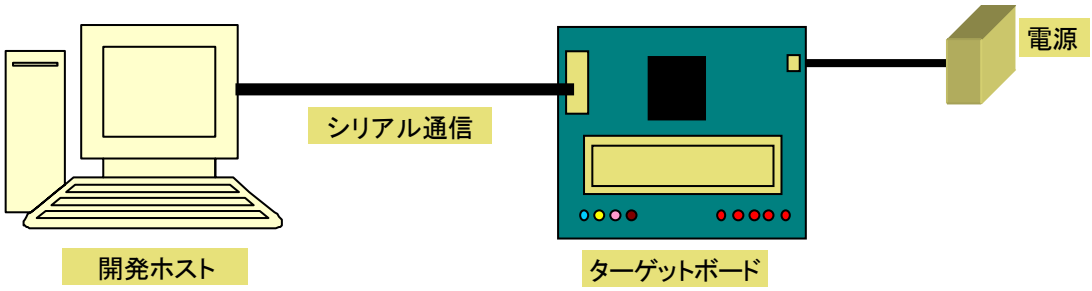


図 1 システム構成

2.2. ソフトウエア

以下のソフトウェアと動作環境が必要です。

開発ホスト	OS	Windows 2000/Windows XP
	メモリ	640Mバイト(1GB上を推奨)
デバッグモニタパッケージ(本ソフト)	ホスト側	デバッグモニタソフト
	ターゲット側	H8/300H 8F,3048B,3052,3062,3067,3068,3069のMCUに対応したデバッグモニタプログラムをROMに書き込んでおきます。 (書き込みソフト例:H8WriteTurbo、h8w.exe等)
コンパイラ	gcc (elf)	本デバッグモニターは“h8300-elf-gcc”の出力情報に対応しています。 “h8300-hms-gcc”の出力情報には対応していません。

表2.1 動作環境

3. MCU機能条件

3.1. デバッグモニターが使用しているMCUの機能

デバッグモニタは、表3.1に示す様にMCUの機能を使用しています。その為、これらの機能をユーザープログラムで使用する事は出来ません。これらの機能以外は、自由にユーザプログラムで使用することができます。

使用機能	内容
エミュレーションのベクター領域の制約事項	ROM上のベクター領域をRAM上に写像するフラッシュメモリのエミュレーション設定をしています。ベクタ番号1,2,3以外ユーザーはベクターテーブル領域の書き換えは出来ません。
SCI1（シリアルコミュニケーションインタフェースチャネル1）	SCI1 を使った通信により、命令のやり取りを行っています。 そのため、ユーザプログラムでSCI1 は使えません。
ベクタ番号4,5,6	ブレークポイントやステップイン、ステップオーバ、ステップアウトなどのシングルステップ動作を実現するために、ベクタ番号4,5,6のメモリ間接を利用しています。 そのため、ユーザプログラムでベクタ番号4,5,6 は使用できません。
ベクタ番号56,57,58,59	SCI1のデータ受信、受信エラー発生の割り込みを利用しているためベクタ番号56,57,58,59を使用しています。 そのため、ユーザプログラムでベクタ番号56,57,58,59 は使用できません。
内臓RAM	内臓RAMの2KBを作業領域として使用しています。

表3.1 使用機能

3.2. MCU 動作モード

MCUの動作可能なモードを示します。基本的に内臓ROM/RAMが有効でなければ成りません。

MCU / モード	3048F(16MHz) 3048B(25MHz)	3052(25MHz)	3062(20MHz)	3067(20MHz)	3068(20MHz)	3069(25MHz)
モード1	×	×	×	×	×	×
モード2	×	×	×	×	×	×
モード3	×	×	×	×	×	×
モード4	×	×	×	×	×	×
モード5	×	×	○	○	○	○
モード6	○	○	×	×	×	×
モード7	×	×	×	×	×	×

表3.2 MCU 動作モード

○ 可能
× 実行不可能
＊実行不可能でも外付けROMでデバッグモニターが実行でき、内臓RAMとSFRのアドレスが同じなら可能です。

3.3. メモリーマップ

内臓ROM/RAMの使用状態を示します。

メモリ / MPU		3048 F/B	3052	3062	3067	3068	3069	備考
内臓ROM	モニタープログラム	000000	000000	000000	000000	000000	000000	
		00FFFF	00FFFF	00FFFF	00FFFF	00FFFF	00FFFF	
	未使用	010000	010000	010000	010000	010000	010000	
		07FFFF	07FFFF	07FFFF	07FFFF	07FFFF	07FFFF	
外部アドレス空間（外部RAMの有無とサイズはボード毎に違います）		080000	080000	080000	080000	080000	080000	
		FFDF0F	FFDF0F	FFDF0F	FFDF0F	FFDF0F	FFDF0F	
内臓RAM	ユーザ使用可能	FFEF10	FFDF10	FFEF20	FFEF20	FFBF20	FFBF20	内臓RAMだけでデバッグする時は、プログラム、変数、スタックポインタはこの領域内に収めて下さい。 外部RAMでデバッグする時は、変数領域として使用出来ます。 エミュレーション領域はベクターテーブル領域(256バイト)以外は使用出来ます。
		FFEFFF	FFDFFF	FFEFFF	FFEFFF	FFDFFF	FFDFFF	
	フラッシュメモリのエミュレーション	FFF000	FFE000	FFF000	FFF000	FFE000	FFE000	
		FFF3FF	FFEFFF	FFF3FF	FFF3FF	FFEFFF	FFEFFF	
	ユーザ使用可能	FFF400	FFF000	FFF400	FFF400	FFF000	FFF000	
		FFF70F	FFF70F	FFF70F	FFF70F	FFF70F	FFF70F	
	モニター使用	FFF710	FFF710	FFF710	FFF710	FFF710	FFF710	モニターの変数領域、スタックポインタに使用します。
		FFFF0F	FFFF0F	FFFF1F	FFFF1F	FFFF1F	FFFF1F	

表1.4 メモリーマップ

3.4. MCUの動作周波数別のデバッグモニターMOTファイル

ターゲットボード上のMCU動作周波数に対応したデバッグモニターのMOTファイルを作成します。
基本のデバッグモニターMOTファイルから、MCUグループと動作周波数の組み合わせで、デバッグモニターMOTファイルを作成出来ます。
対象MCUが換わったとき、基本のデバッグモニターMOTファイルから、何度も目的のMCUデバッグモニターMOTファイルを作成することが出来ます。

MCU動作周波数（MHz）	周波数
2	2MHz
2.097152	
2.4576	
3	3MHz
3.6864	4MHz
4	
4.9152	5MHz
5	
6	6MHz
6.144	
7.3728	7MHz
8	8MHz
9.8304	9MHz
10	10MHz
12	12MHz
12.288	
13	13MHz
14	14MHz
14.7456	15MHz
16	16MHz
18	18MHz
20	20MHz
25	25MHz

表3.4 動作周波数表

図3.4 MOTファイル作成

4. 作成手順

4.1. 作成手順フロー

導入から完成プログラムをフラッシュROMに書き込むまでに必要な作業手順を記載します。

（1）デバッグモニターMOTファイルをMCUのフラッシュROMに書き込む
（2）プロジェクトの作成とプロジェクトへのソースの登録
（3）プログラムデバッグ
（4）完成プログラムをMCUのフラッシュROMに書き込む
（5）完成プログラムをフラッシュROMで実行確認

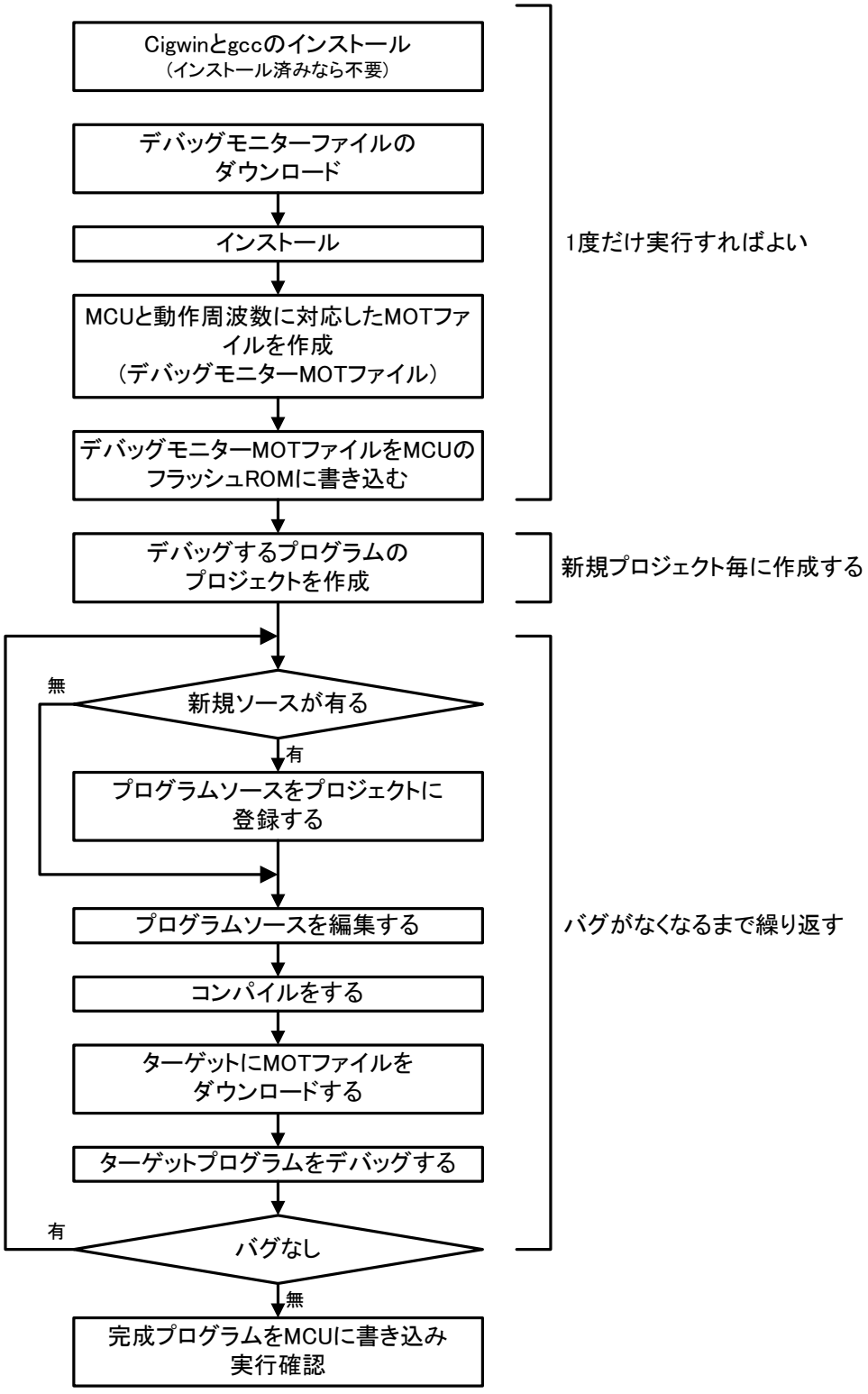


図4.1 作成手順フロー

4.2. プロジェクトとモジュールとファイルの関係

1つのプロジェクトを作成すると指定された作業スペース直下にプロジェクトが作成されなす。
プロジェクト直下にはコンパイルに必要なメイクファイルとモジュールフォルダ、デバッグ情報フォルダを作成します。
モジュールフォルダ直下にソースファイルを作成します。下記の図の様にプロジェクトのファイルを階層的に関連付けることができます。

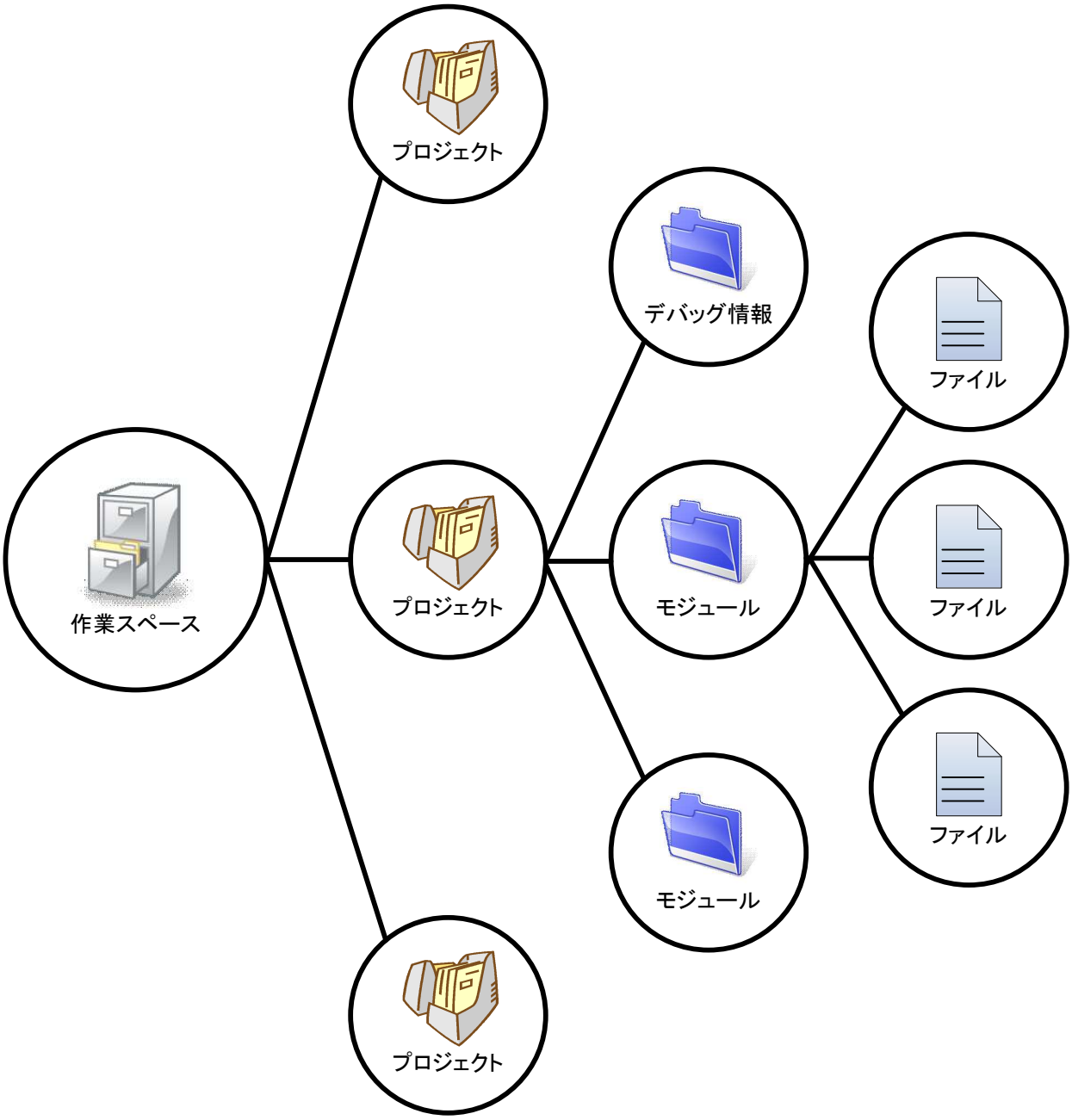


図4.2 プロジェクトファイル構成

5. ターゲットプログラム解析方式

デバッグモニタには二つの実行方式が有ります。一つはユーザプログラムを1行ずつ取得、実行をするインタープリター方式で、1ステップ毎の実行状態を見て変数変化の状態を知るのに適した方式と、もう一つは、直接ユーザプログラムを実行する(フラッシュROMに書いた状態に近い速度)システム実行方式が有ります。

5.1. インタープリター方式とシステム実行方式

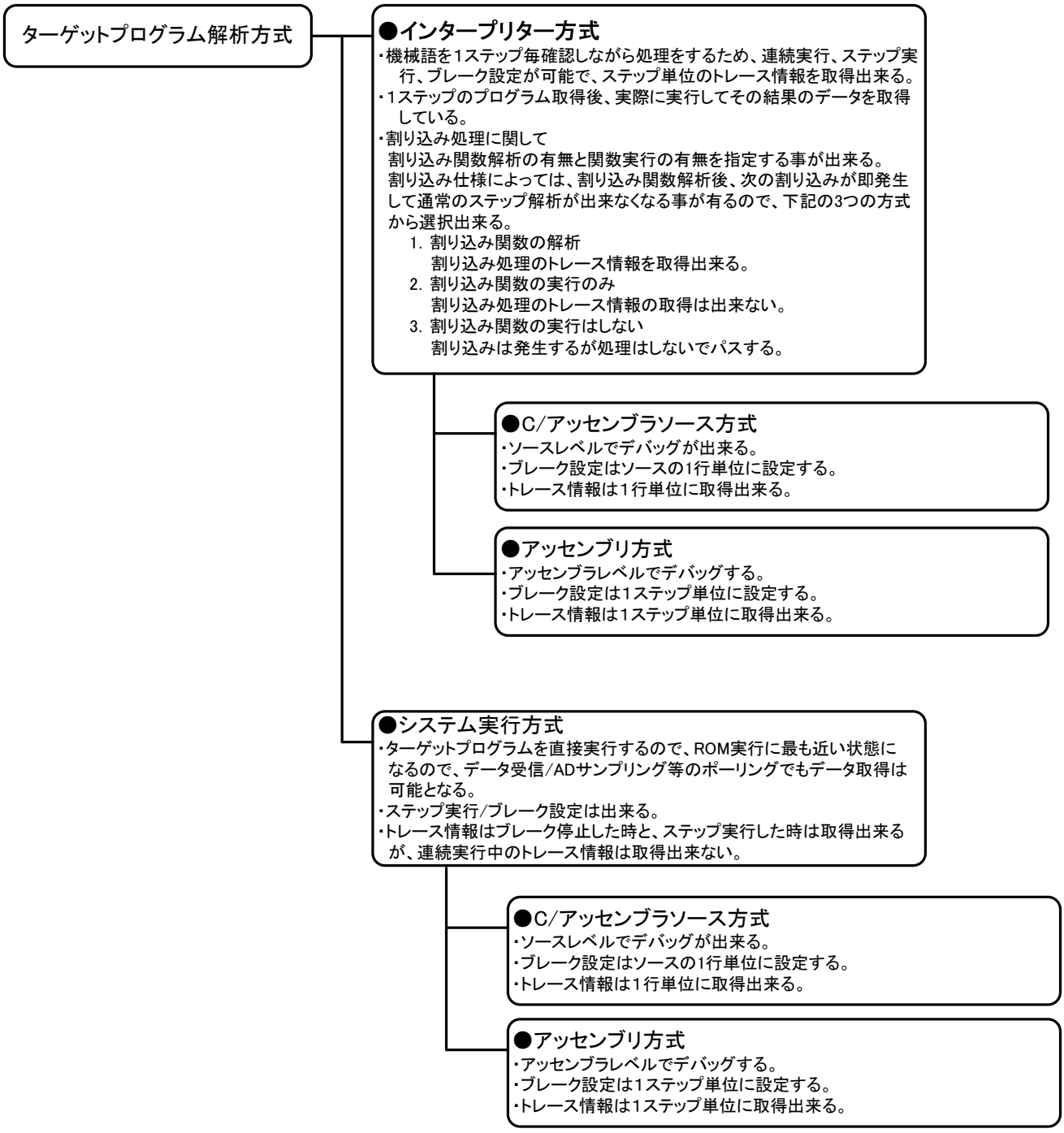


表5.1 実行方式構成

5.2. デバッグ時のメモリーイメージ

デバッグをするにはRS-232Cで接続して、ターゲットボードとプログラムやデータのやり取りを行います。
ユーザプログラムをRAM上に転送するので、修正の度にフラッシュROMに書き込む必要がないので、完成プログラムのフラッシュROMへの書き込み回数を減らすことが出来ます。

（1）外部RAMを使用してデバッグする時

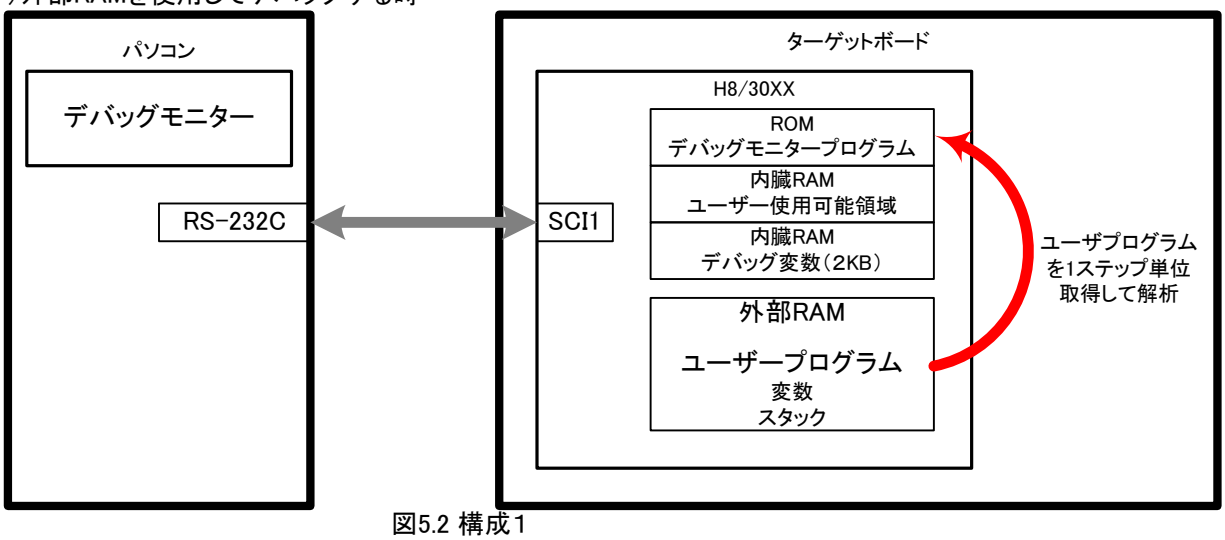


図5.2 構成1

（2）内臓RAMを使用してデバッグする時

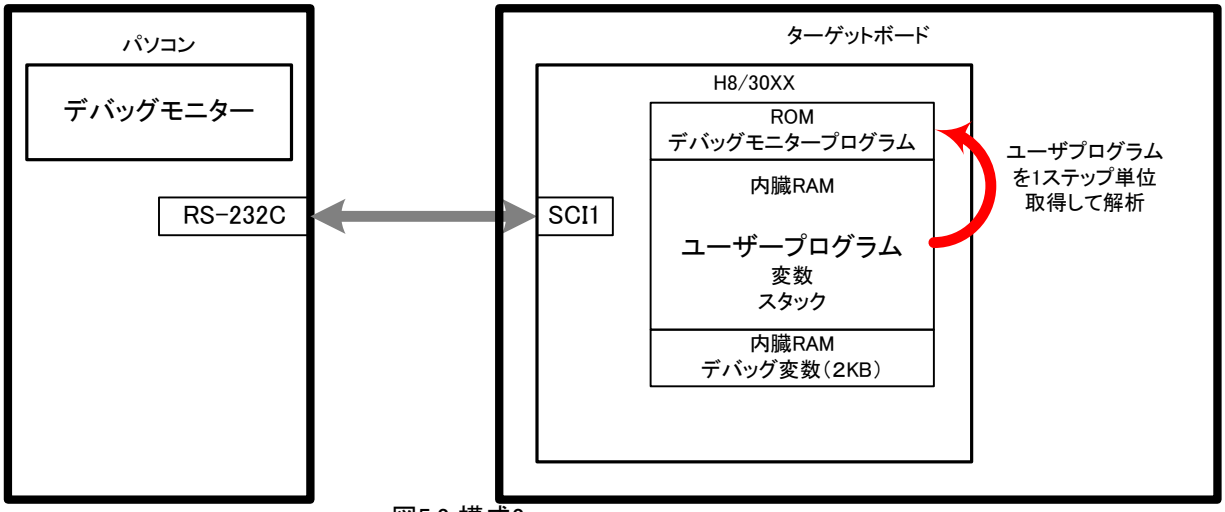


図5.2 構成2

（3）ターゲットプログラムを直接実行する時

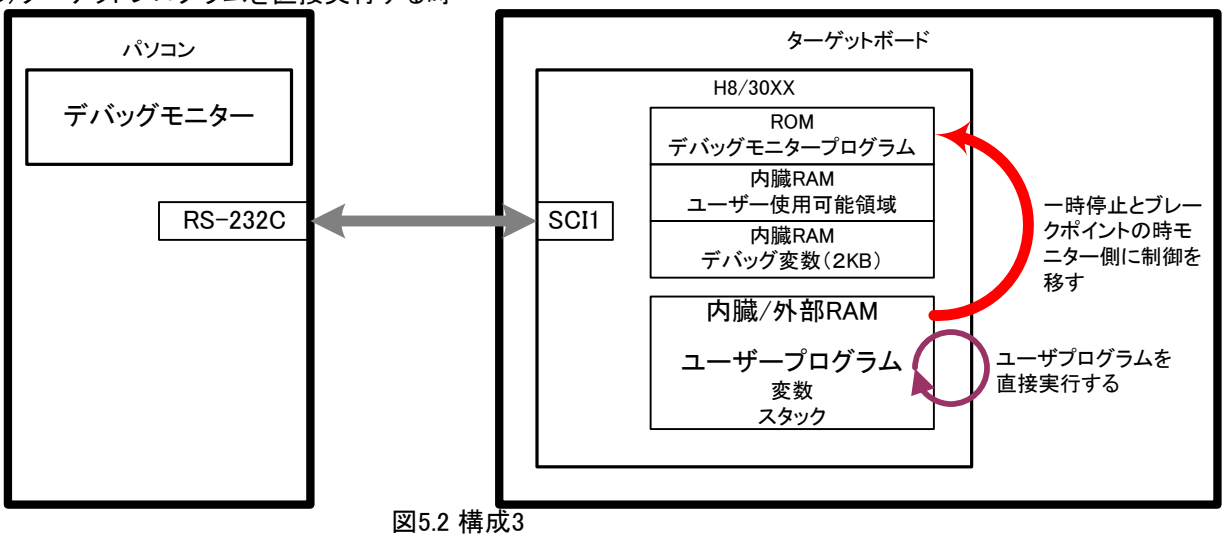


図5.2 構成3

5.3. デバッグ中のインタープリター方式とシステム実行方式の切り替え

デバッグ中の関数の直前にブレークポイントを設定して、実速度が速い”システム実行方式”でブレークポイントまで進め、以降インタープリター方式に切り替えて詳細なデバッグが出来ます。

実行方式を切り替える時は、必ずブレークポイント設定して、その停止後に実行方式を切り替える様にして下さい。

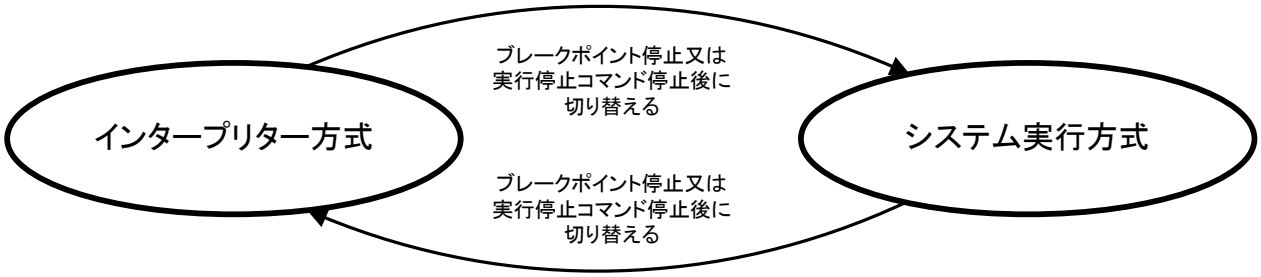


図5.2 実行方式切り替え遷移図

5.3. ユーザが割り込み関数をデバッグする時

ユーザが割り込みを使う時は、通常ベクタアドレスが有る領域を「RAMによるフラッシュメモリのエミュレーション」機能を使いベクター領域を書き換える必要が有ります。
本モニターはその処理をモニター内で処理をしているので、ユーザーは割り込み関数を記述するだけですむ様にしてありますので、ROMやエミュレーション領域を書き換える必要は有りません。
H8/3052を例に「RAMによるフラッシュメモリのエミュレーション」の概念を記載します。

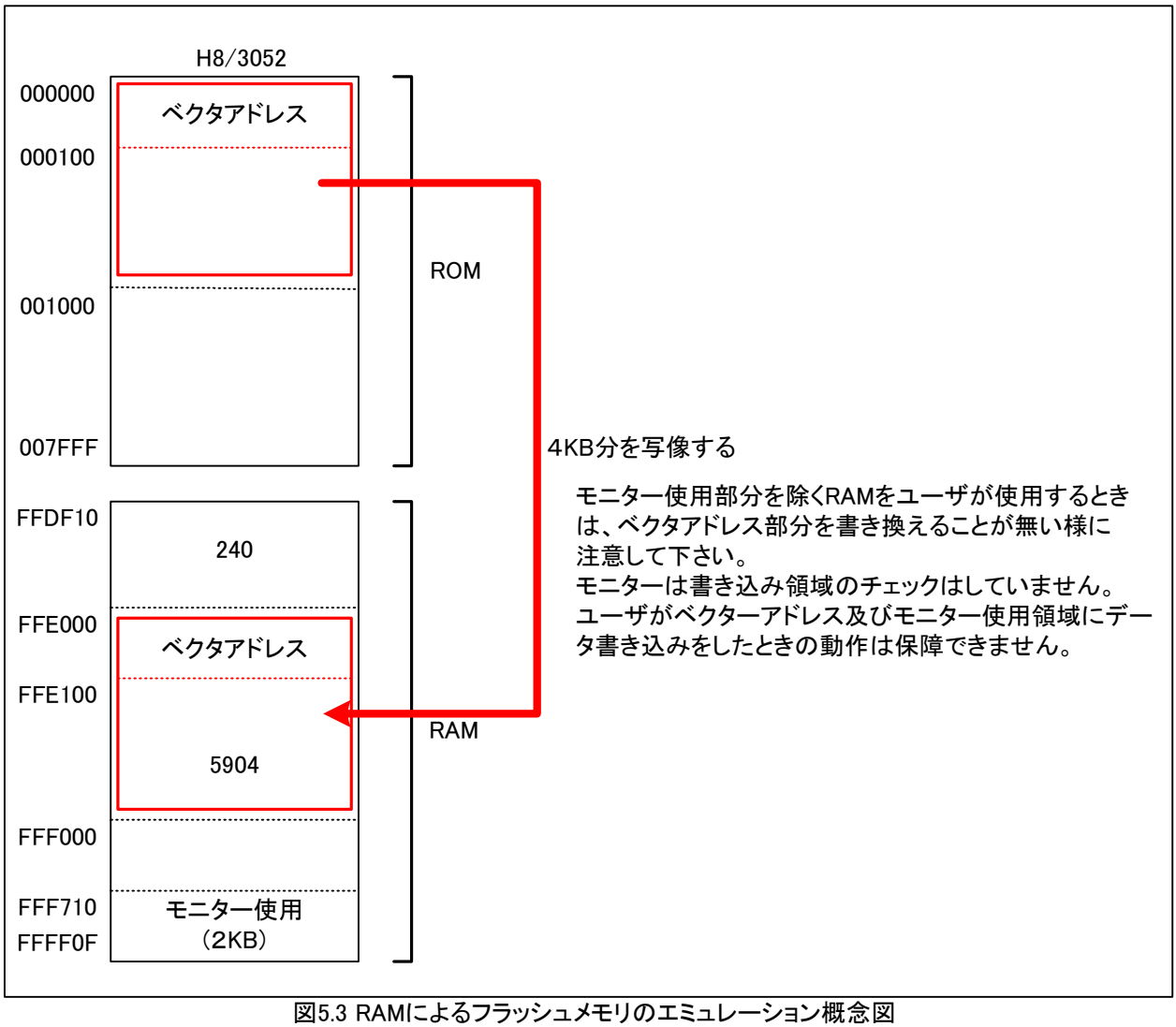
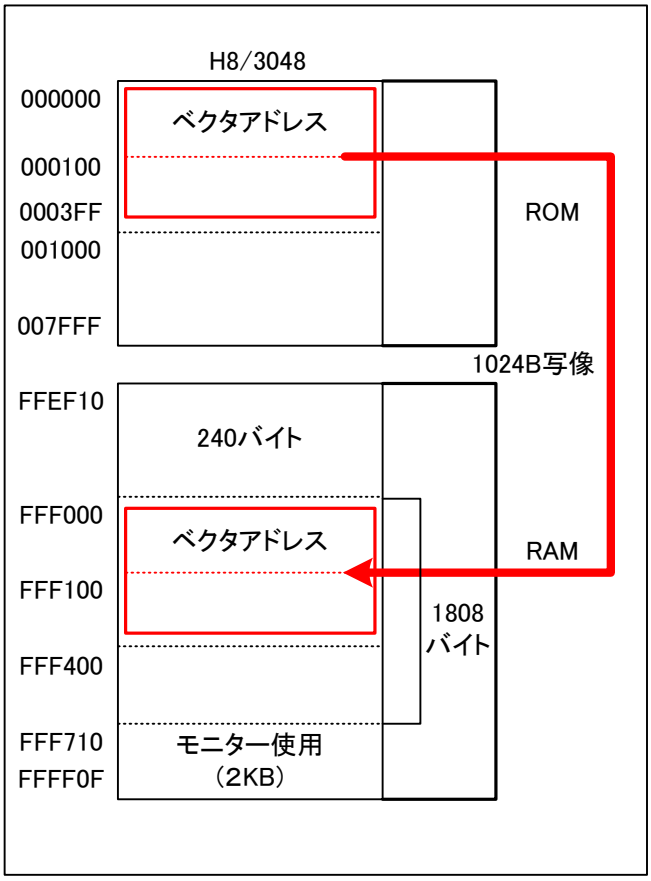
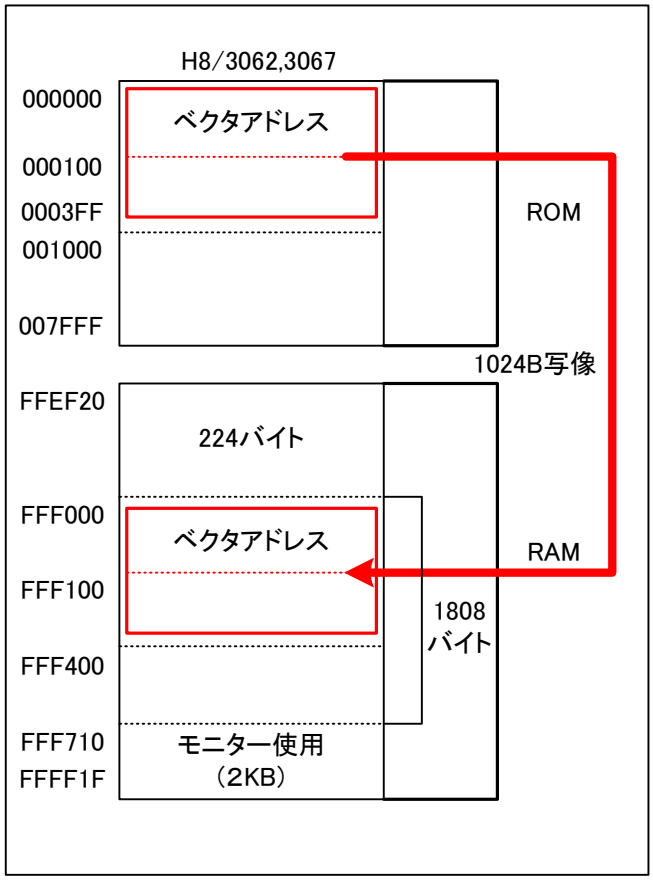


図5.3 RAMによるフラッシュメモリのエミュレーション概念図





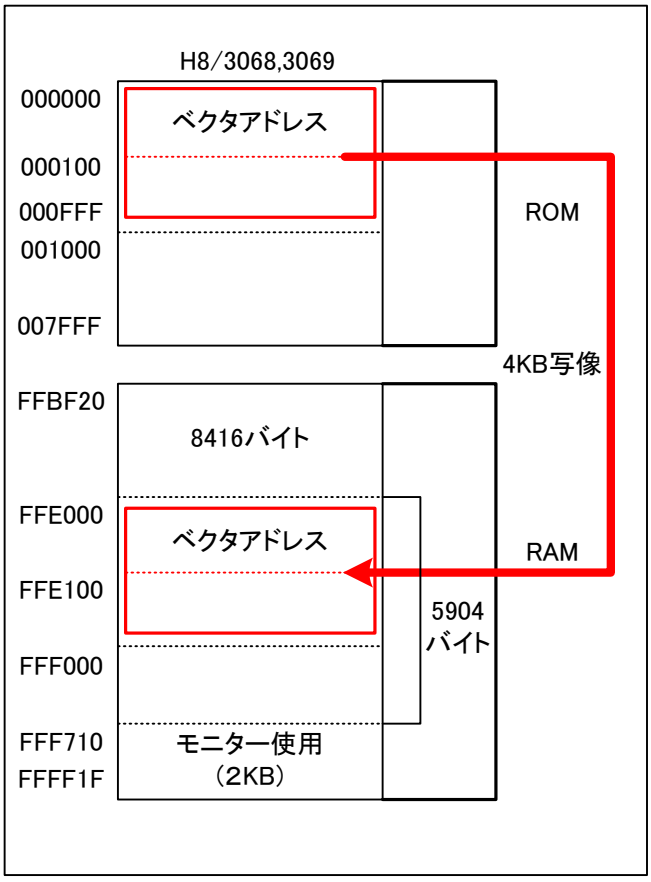
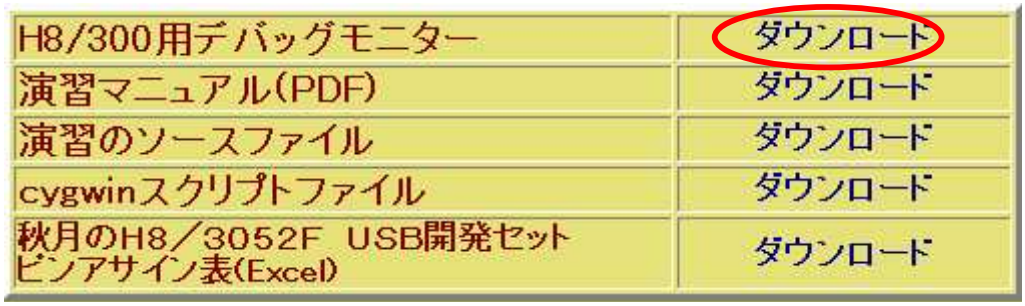


図5.4 その他のMCUのRAMによるフラッシュメモリのエミュレーション概念図

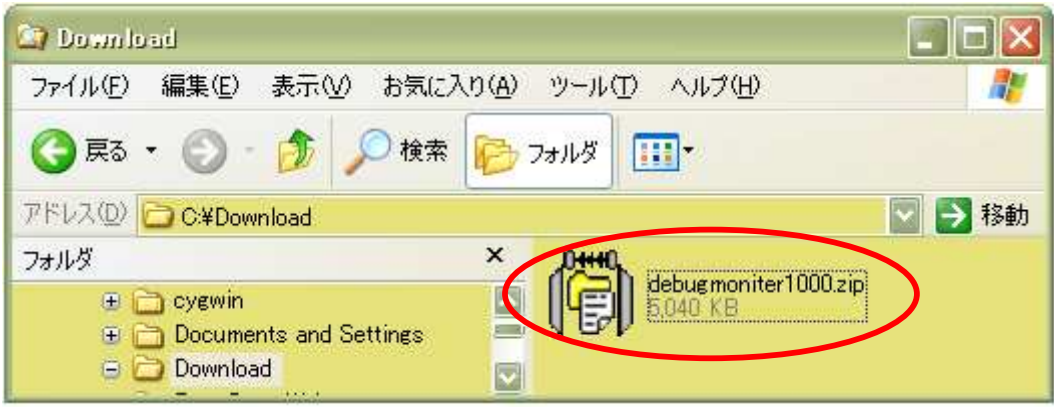
6. インストール

6.1. ファイルのダウンロード

(1)トップページからダウンロードをする
デバッグモニターのダウンロードをクリックして。”debugmoniter1000.EXE”をダウンロードします。
数字の1000はバージョンにより異なります。



(2)ダウンロードファイルを解凍する
ダウンロードファイルをダブルクリックして解凍します。アイコンは使用するアーカイブソフトで変わります。



(3)解凍されたファイル状態
フォルダー階層は使用するアーカイブソフトで変わります。

6.2. デバッグモニターの設定

(1)setup.exeを実行します。

(2)セットアップ開始

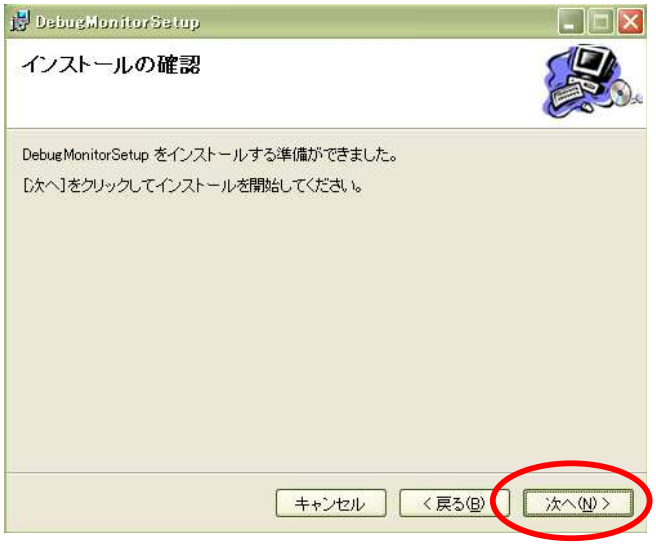
次へをクリックします。

(3)セットアップフォルダと使用者の設定

フォルダと使用ユーザを設定後、次へをクリックします。

(4)準備確認

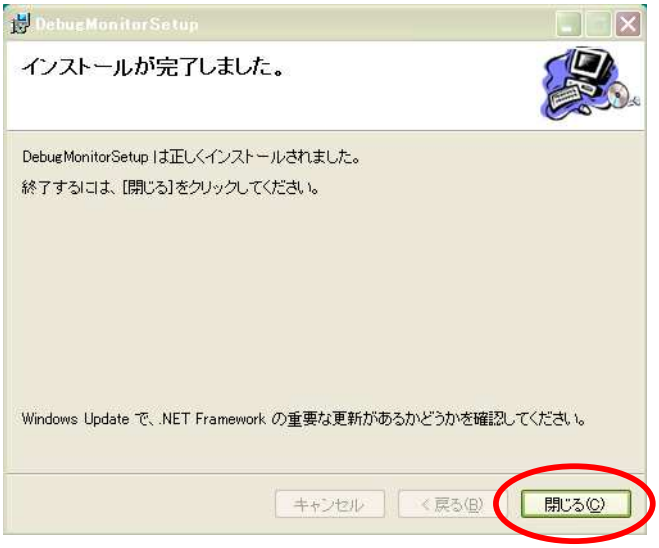
次へをクリックします。



(5)使用許諾の確認

同意するをクリック後、次へをクリックします。

(5)インストール完了
　閉じるをクリックします。



(6)インストールされたデバッグモニターとMOTファイル作成のショートカット

6.3. アンインストールするとき

コントロールパネルの「アプリケーションの追加と削除」を使ってアンインストールしてください。

7. ターゲットMCUに合ったMOTファイルを作る

7.1. MOTファイルを作成するプログラムを起動する

”MakeMCUMotFile”をダブルクリックして起動します。

7.2. 使用するMCUに合ったデバッグモニターMOTファイルを作成する

使用するMCUのグループと動作周波数に合ったMOTファイルを作成します。このMOTファイルをフラッシュROMに書き込みます。

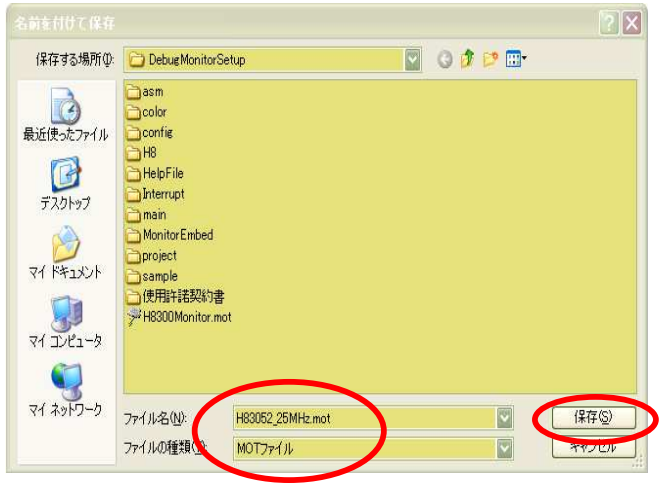
例としてMCUのグループがH8/3052で動作周波数が25MHzのときのデバッグモニタMOTファイルを作成します。”？”をクリックするとヘルプを表示しますので、別のMCUグループの例が記載されていますので、合わせて参考にして下さい。

(1)MCUグループのラジオボタンをクリックすます。

(2)動作周波数のラジオボタンをクリックすます。

(3)作成をクリックすます。

(4)保存をクリックします
　ファイル名を”H8300Monitor”にはしないで下さい。



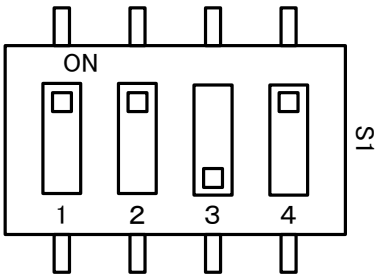
7.3. MOTファイルをフラッシュROMに書き込む

(1)パソコンとターゲットボードをRS-232Cケーブルで接続して下さい。
接続するパソコン側とターゲットボードのポートはユーザの状況に応じて選択して下さい。
例として"H8WriteTurbo,exe"で書き込む方法を記載します。"H8WriteTurbo,exe"の詳細はヘルプを参照して下さい。
この例はパソコンは"COM1"でターゲットは"SCI1"に接続して書き込みます。

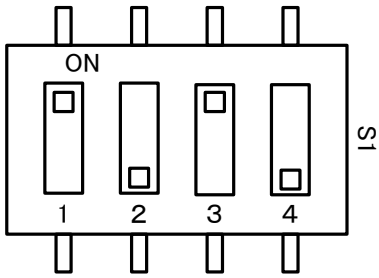
(2)書き込むMOTファイルを"H8WriteTurbo,exe"にドラッグ・アンド・ドロップする

・秋月電子の"H8／3069F USBホストボード"に書き込むときの注意点
動作モード設定するDIP SWは以下の様になっています。

書き込み（ブートモード7）



動作（モード5）



・転送ソフトについて
h8write.exe、H8WriteTurboあるいはH8Write（Masahiro Ochiai氏）等が有ります。

「h8write.exe」みついわゆきお氏のオープンツールです。次の場所から入手できます。
http://mes.sourceforge.jp/h8/writer-j.html

H8WriteTurboHは次の場所から入手できます。
http://strawberry-linux.com/h8/

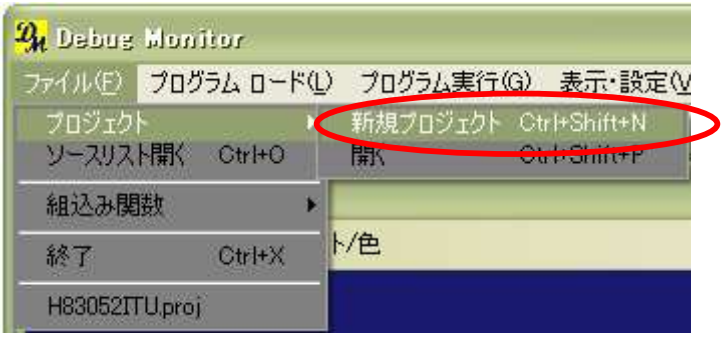
8. プロジェクトの作成

8.1. デバッグモニターを起動する

”DebugMonitor”を左ダブルクリックして起動させます。

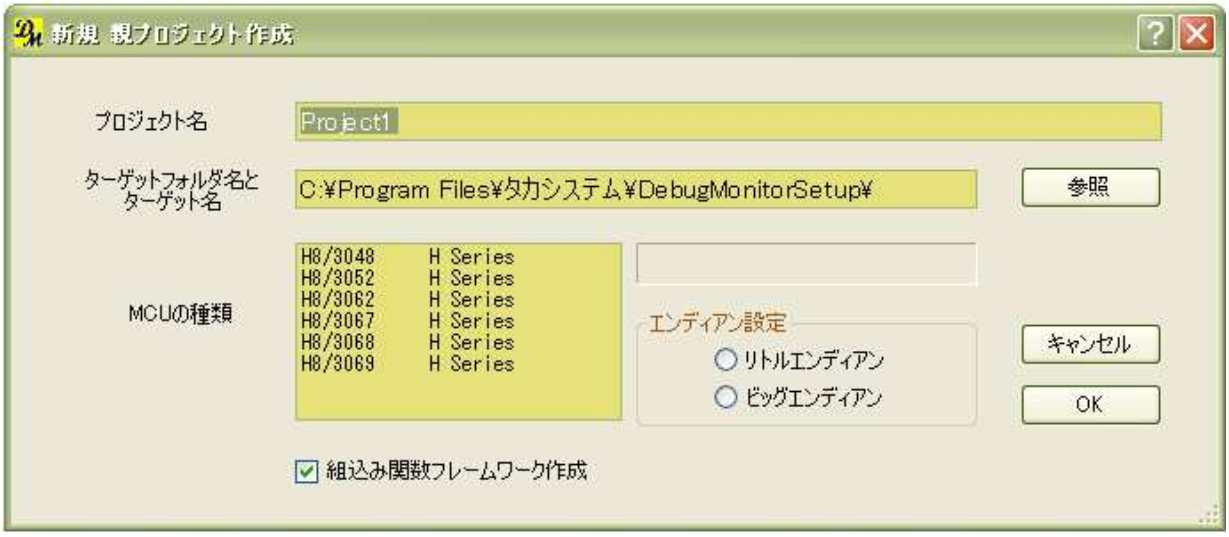
8.2. 新規プロジェクトの作成

メニューから新規プロジェクトを選択してクリックします。



8.3. 新規プロジェクトの作成ウィンドウ

新規プロジェクトを作成する初期ウィンドウです。
プロジェクト名、保存先とターゲット名、MCUの種類、組み込み関数フレームワーク作成を設定します。
設定例としてプロジェクト名は”TEST”、MCUはH8/3052で設定して行きます。
ユーザはプロジェクト名は作成するプロジェクトに、、MCUの種類は使用するMCUに置き換えてください。

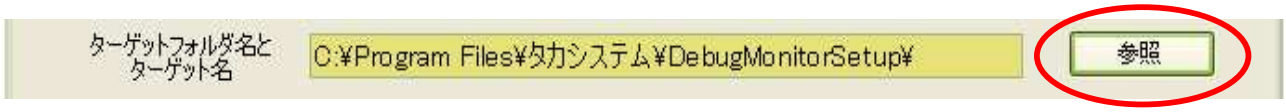


(1)プロジェクトを入力

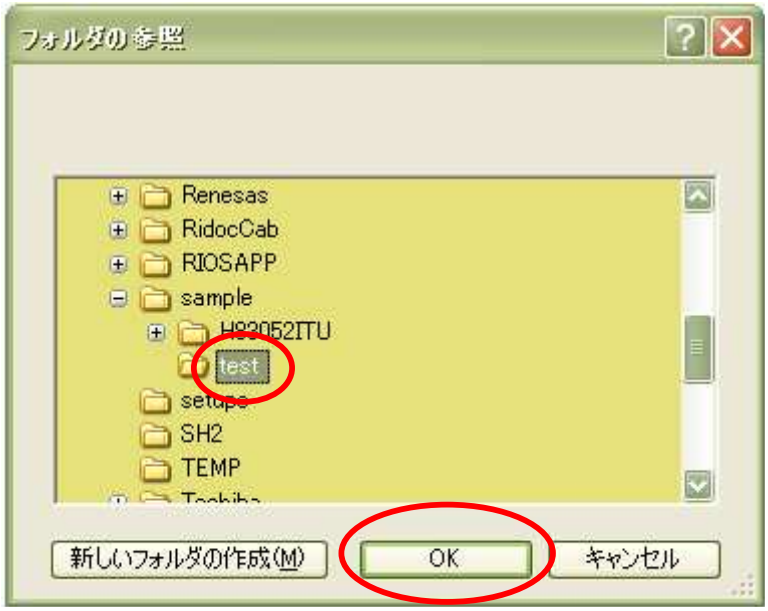
ここではプロジェクト名を”TEST”と入力します。

(2)保存先を入力

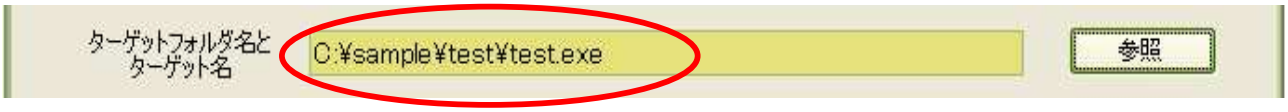
保存先とターゲット名をフルパス指定で入力するか、参照を左クリックして保存先を指定して下さい。
参照をクリックして保存先を指定したときの状態を示します。



1)保存先のホルダまで移動し、OKをクリックします。

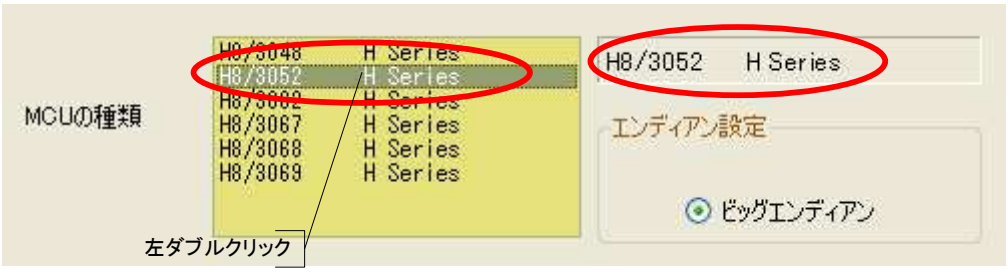


2)保存先のホルダが書き換わります。



(3)MCUの種類を指定

使用するMCUの種類を選択します。
MCUの種類を左ダブルクリックすると、選択したMCU名を表示します。
他のMCUの時は該当するMCUを選択して下さい。



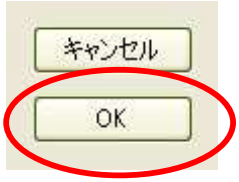
(3)組み込み関数フレームワーク作成

デバッグモニターは外部RAMを有効にする方法をしらないので、ターゲットボードの外部RAMを有効にするプログラムを組み込むことが出来る機能です。
この組み込み関数は、デバッグするプログラム本体を外部RAMにダウンロードする前に実行して、外部RAMを使用出来る様にするためのプログラムのフレームワークを作成します。
外部RAMを使用するときは必要になるはずなので、作成をチェック状態にして下さい。

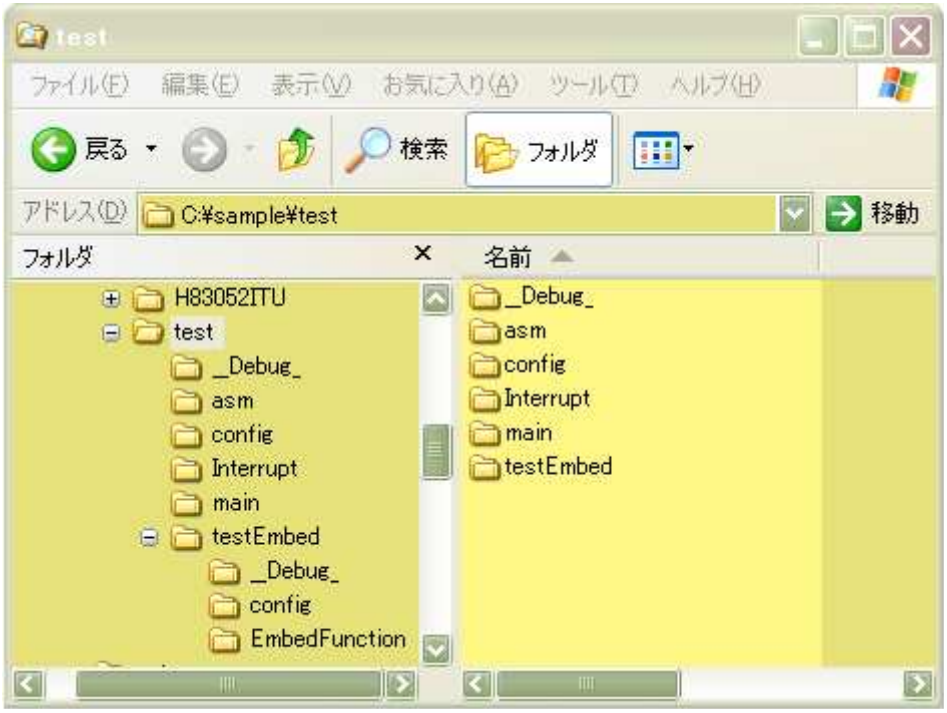
内臓RAMだけデバッグするプロジェクトは、作成のチェック状態をはずしても構いません。後から追加することがでます。

(4)プロジェクトの保存

OKボタンをクリックするとプロジェクトを保存します。



作成されたプロジェクトフォルダ状態です。



(5)ソース/ヘッダの環境設定

既存のソースや新規のソースファイルがあるとき、プロジェクトに追加します。
詳細は環境設定のヘルプを参照して下さい。
今回は設定しません。

(6)ツールの環境設定

ユーザ環境に応じた設定をして下さい。
cygwinは"C"ドライブ直下にインストールしていることを前提にしていますので、ユーザのインストール先に変更してください。

1)ソースやテキストファイルを編集する時に使用するテキストエディタを設定します。
参照を左クリックするとフォルダとエディタの有る所を簡単に設定できます。
使いなれたテキストエディタに変更して下さい。

2)cygwinの"make.exe"が有るフォルダを設定します。
参照を左クリックすると"make.exe"が有るフォルダを簡単に設定できます。
今回は変更しません。

3)cygwinのH8300のユーティリティが有るフォルダを設定します。
参照を左クリックすると"h8300-elf-gcc.exe"が有るフォルダを簡単に設定できます。
今回は変更しません。

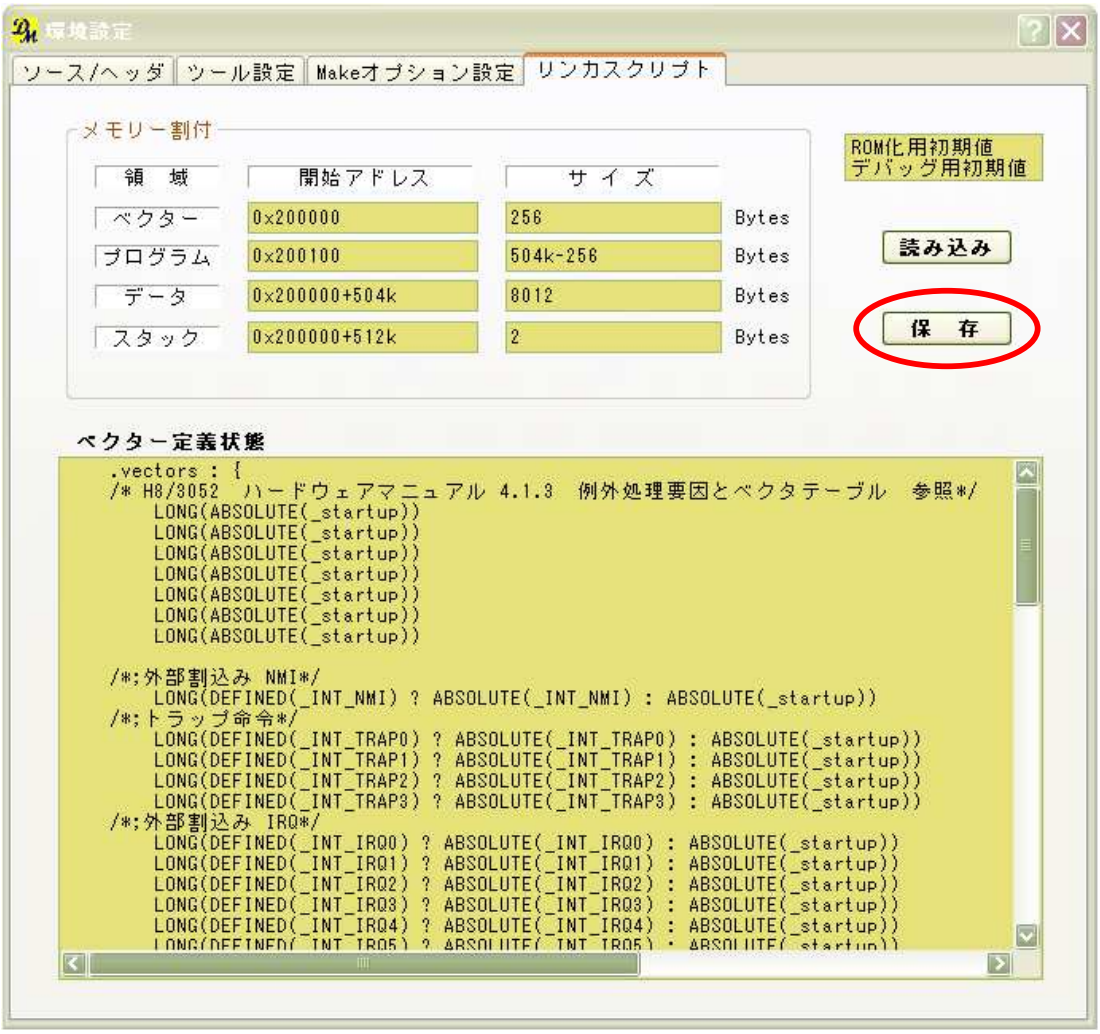
・gccの接頭語はそのままにします。

(7)Makeオプションの環境設定

コンパイラオプションの設定をします。
組み込み環境に応じて設定します。今回は以下の様に設定して保存をクリックして下さい。

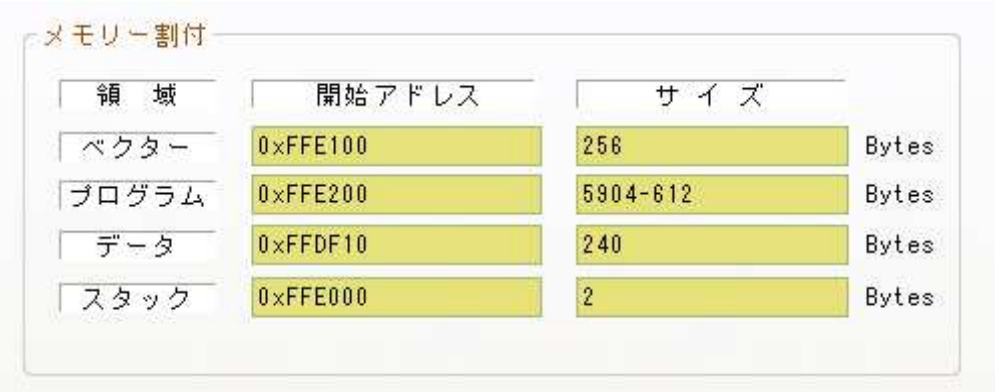
(8)リンカスクリプトの環境設定

ターゲットプログラムを実行させるRAM領域の各セクションの開始アドレスとバイト数を設定します。
使用するターゲットボードの外部RAMアドレスマップに応じて設定して下さい。
初期値は外部RAMが0x200000番地から512KB分のサイズ有ることを前提にしています。
変更したら必ず保存をクリックして下さい。

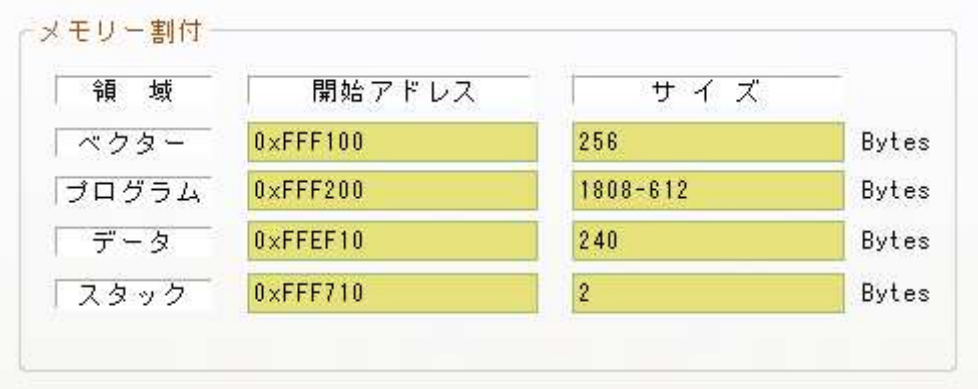


(9)内臓RAMだけでデバッグするときのアドレス設定例
外部RAMが無いときはこの設定にして下さい。
"3.3. メモリーマップ"も参考にして下さい。

H8/3052



H8/3048



H8/3062,3067

H8/3068,3069

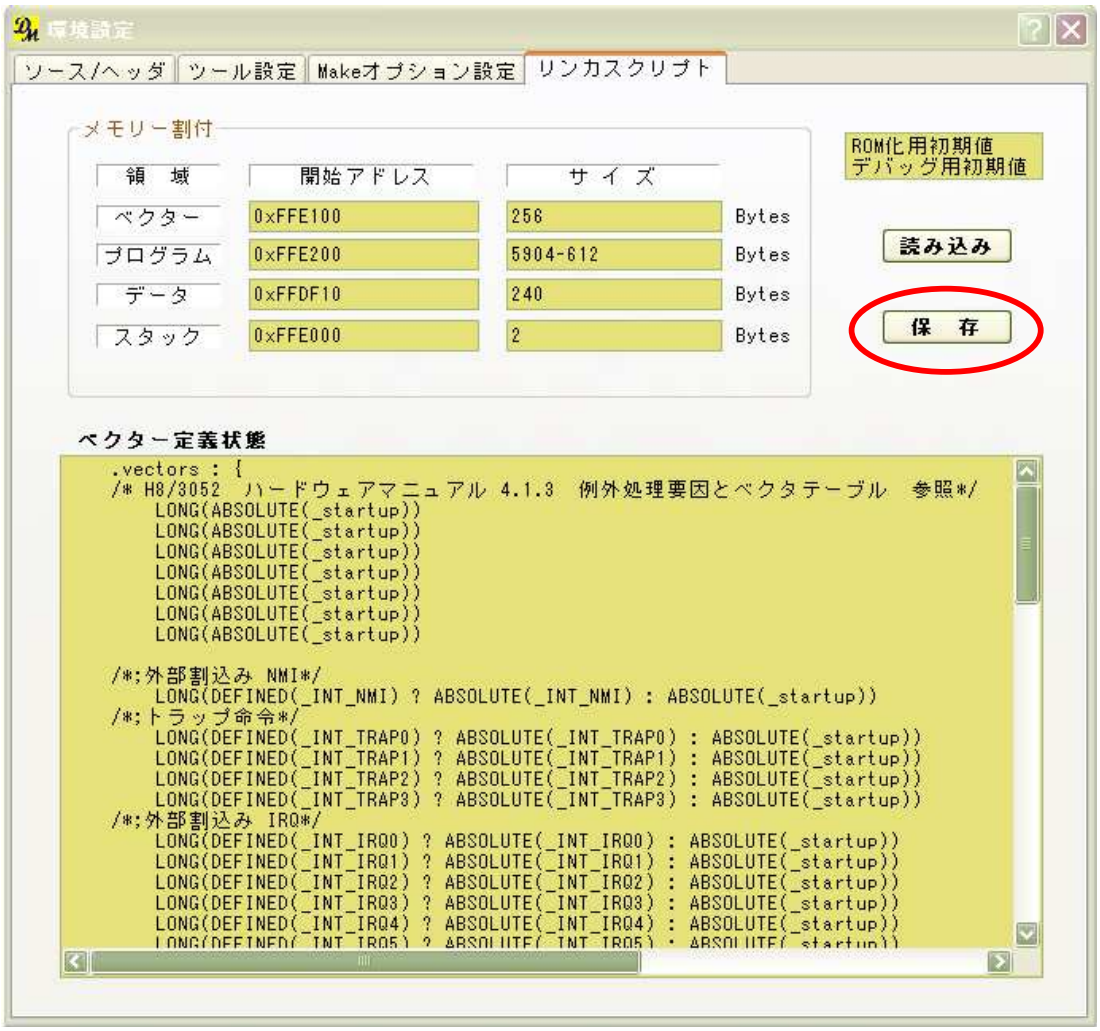
これで準備完了です。

9. プログラムの編集

外部RAMが無いボードで動作する、ウォッチドッグタイマをインターバルタイマにして、割り込み発生させるプログラムを組みます。

9.1. サンプルプログラムの編集をする

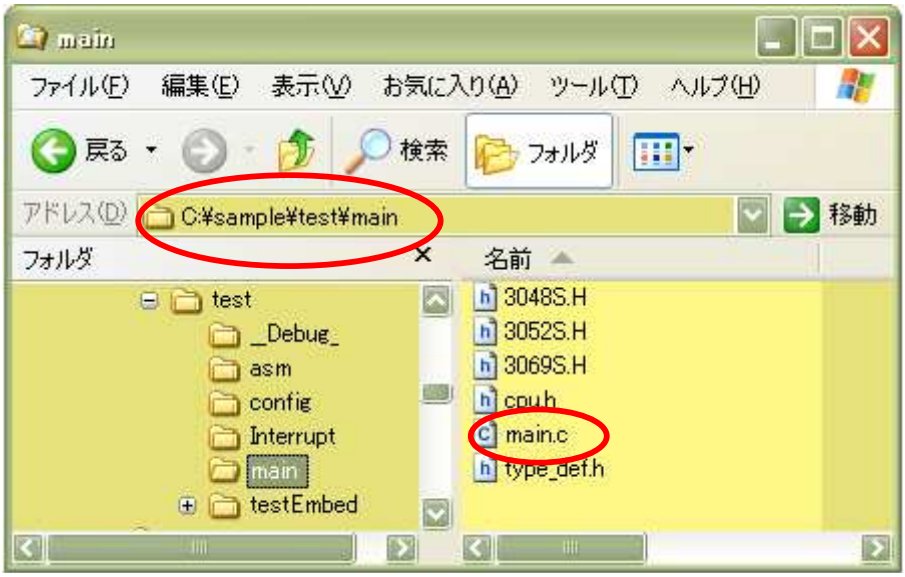
(1)内臓RAMだけで実行出来る様にリンカスクリプトを設定する



(2)main関数を記述する

記述するソースファイルがコンテンツ上に有りますのでダウンロードできます。

プロジェクトを作成したホルダ内の"main.c"を編集します。



ここから"main.c"です。

```
/************************************************************************/
/*                                                                      */
/*    FILE        :main.c                                               */
/*    DATE        :                                                     */
/*    DESCRIPTION :Main Code                                            */
/*    CPUTYPE     :                                                     */
/*                                                                      */
/* 変更履歴 :                                                           */
/*    変更日        変更者        変更内容                              */
/*                                                                      */
/************************************************************************/
#include "type_def.h"
#include "cpu.h"


void    StartIntTimer( void );
void    StopIntTimer( void );
void    LEDInitPort( void );


UCHAR       cLedData;       // LEDの点灯・消灯するデータ
UCHAR       befflag;        // 割り込み発生前の状態
UCHAR       aftflag;        // 割り込み発生後の状態
```

次ページに続く

前ページの続き

```
/************************************************************************/
/* MODULE  : LEDInitPort                                                */
/* ABSTRACT: LED点灯・消灯ポートの設定と初期化                          */
/* FUNCTION: ポートBを出力に設定し点灯・消灯を初期化する                */
/*                                                                      */
/* NOTE    :                                                            */
/* ARGUMENT: void                                                       */
/* RETURN  : void                                                       */
/*                                                                      */
/************************************************************************/
void    LEDInitPort( void )
{
    PBDDR.PORT.all = 0xff;              // ポートB出力に設定
    cLedData = CLR_imm;                  // 点灯・消灯データ初期化
    PBDR.PORT.all = ~cLedData;          // LED全消灯(0で点灯する場合)
}

/************************************************************************/
/* MODULE  : StopIntTimer                                               */
/* ABSTRACT: インターバルタイマーカウント停止処理                       */
/* FUNCTION: インターバルタイマーにして、クロックセレクトをφ/4096にて  */
/*             カウント停止をする                                       */
/*                                                                      */
/* NOTE    :                                                            */
/* ARGUMENT: void                                                       */
/* RETURN  : void                                                       */
/*                                                                      */
/************************************************************************/
void    StopIntTimer( void )
{
    TCNT_WT = 0x5A00;
    TCSR_WT = 0xA51F;
    RSTCSR_WT = 0xA500;
}


/************************************************************************/
/* MODULE  : StartIntTimer                                              */
/* ABSTRACT: インターバルタイマーカウント開始処理                       */
/* FUNCTION: インターバルタイマーにして、クロックセレクトをφ/4096にて  */
/*             カウント開始をする                                       */
/*                                                                      */
/* NOTE    :                                                            */
/* ARGUMENT: void                                                       */
/* RETURN  : void                                                       */
/*                                                                      */
/************************************************************************/
void    StartIntTimer( void )
{
    TCNT_WT = 0x5A00;
    TCSR_WT = 0xA53F;
    RSTCSR_WT = 0xA500;
}
```

次ページに続く

前ページの続き

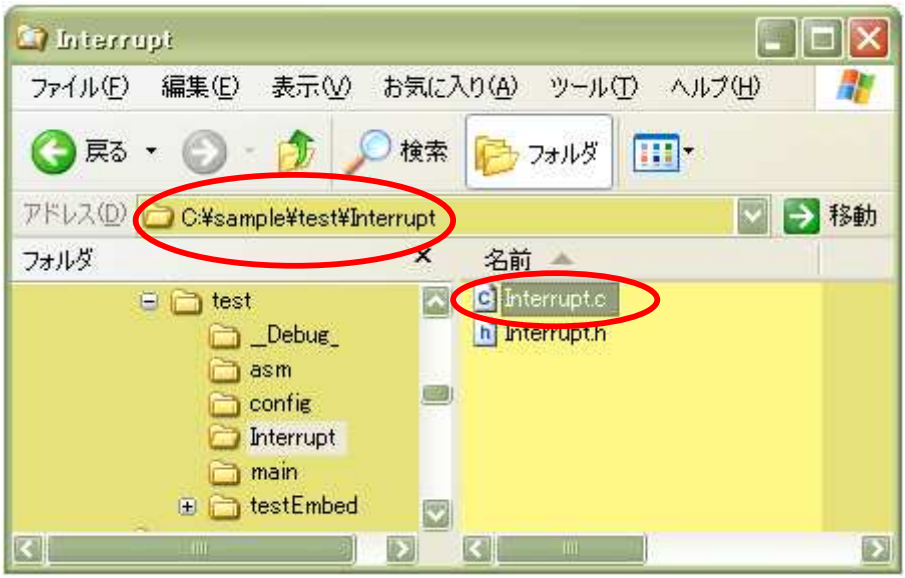
```
/************************************************************************/
/* MODULE  : main                                                       */
/* ABSTRACT: メイン処理                                                 */
/* FUNCTION: 割り込み発生毎にLED点灯のデータを変化させ点消灯を無限に   */
/*           処理する                                                   */
/*                                                                      */
/* NOTE    : LEDはポートBに4つ接続されていれば点消灯をする             */
/* ARGUMENT: void                                                       */
/* RETURN  : int                                                        */
/*                                                                      */
/************************************************************************/
int        main( void )
{
    aftflag = OFF_imm;                          // 割り込み発生状態の初期化
    befflag = OFF_imm;                          // 割り込み発生状態の初期化
    __asm volatile("andc    #0x7f,ccr");        // 割り込み許可
    LEDInitPort();
    StartIntTimer();

    while (1)
    {
        if (aftflag != befflag) // 割り込み先で"befflag"で変化するのを待つ
        {
            aftflag = befflag;      // 状態を差し替える
            cLedData++;             // LED点灯位置を変化させる
            if (cLedData == 0x11)
            {
                cLedData = 0x00;    // 全点灯したので全消灯する
            }
            StartIntTimer();        // カウント開始
        }
    }
    return (0);
}
```

“main.c”はここまでです。

(3)インターバルの割り込み関数を記述する

プロジェクトを作成したホルダ内の"Interrupt.c"を編集します。
"Interrupt.c"にたの割り込み関数も記述してありますが、下記の部分だけ記述しても問題は有りません。



ここから“Interrupt.c”です。

```
#include    "type_def.h"
#include    "cpu.h"


extern    void    StartIntTimer( void );
extern    void    StopIntTimer( void );
extern    UCHAR   cLedData;       // LEDの点灯・消灯するデータ
extern    UCHAR   befflag;        // 割り込み発生前の状態



// WOVI(ウォッチドッグタイマ)
#pragma interrupt
void    INT_WOVI( void )
{
    StopIntTimer();
    befflag ^= 1;
    PBDR.PORT.all = ~cLedData;
}
```

“Interrupt.c”はここまでです。

10. プログラムのコンパイル

(1)メニューバーの”コンパイル”をくりっくする。

(2)ビルドをクリックする。

(3)コンパイルの終了
エラーがなければ”>コンパイル終了”のメッセージが表示されます。
エラーが有るときはエラーメッセージのテキストファイルを表示します。

コンパイルエラーがなければ次はデバッグです。

11. サンプルプログラムのデバッグをする

11.1. プログラムをターゲットボードにダウンロードする

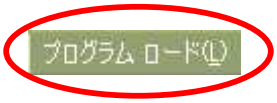
(1)ターゲットボードと通信できるか確認する

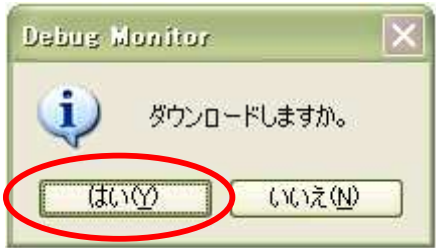
・ターゲットボードの電源を入れる

・"test.proj"をクリックする。　・ターゲットボートと正常に通信できるとバージョンメッセージが表示される。

(2)プログラムをダウンロードする

1)メニューバーからプログラムロードをクリックする。　2)はいをクリックする。

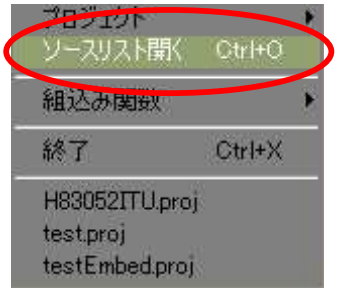




3)ダウンロード終了が表示される。

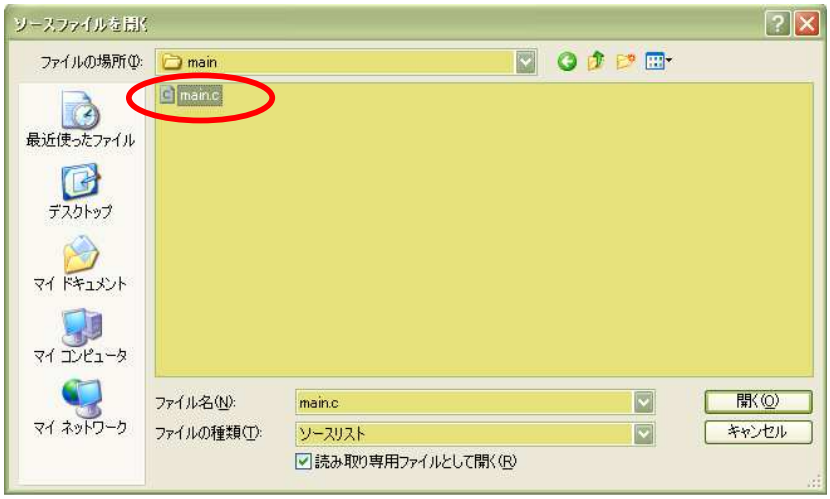
11.2. ソースリストを表示する

(1)メニューバーからソースリスト開くをクリックする。



(2)mainフォルダをクリックする

(3)main.cをクリックしてソースを表示する



(4)Interrupt.cをクリックしてソースを表示する

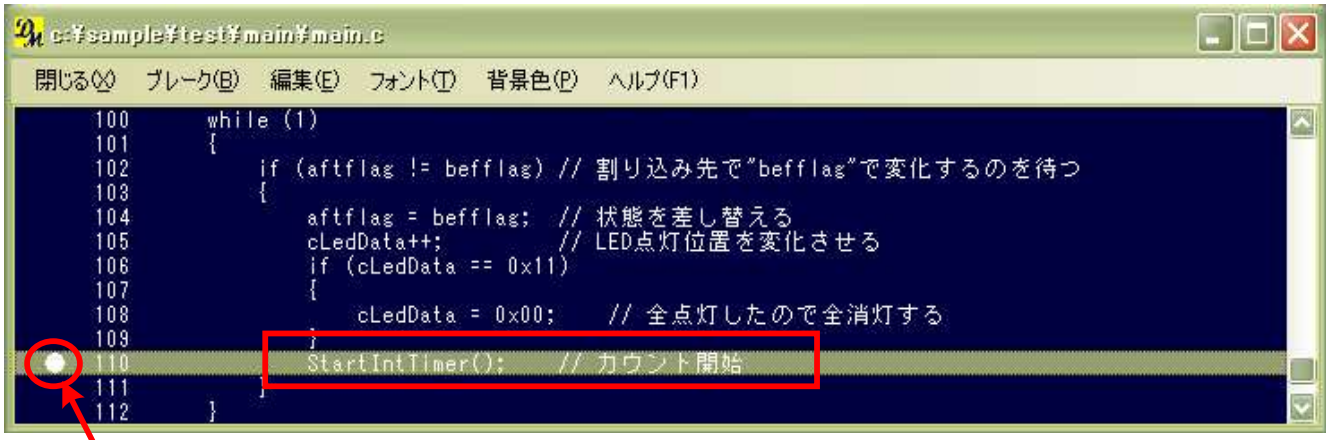
main.cと同じ操作で、"interrupt"フォルダ内のInterrupt.cを表示する

11.3. ブレークポイントの設定と解除をする

停止させてい行にブレークポイント設定すると、実行中ブレークポイント設定された行で停止させることが出来ます。

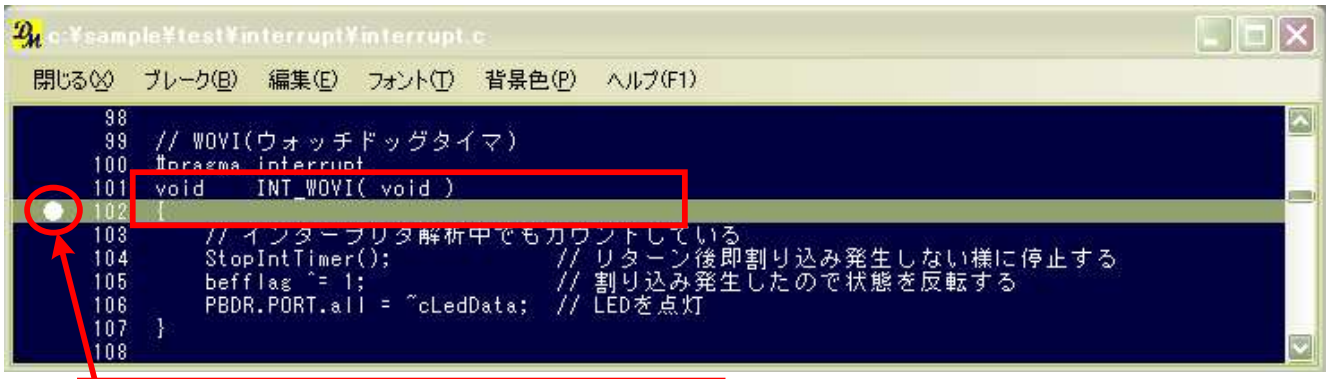
(1)ブレークポイント設定する行までカーソル移動をする。
(2)ブレークポイント設定する選択行で左ダブルクリックする。

プログラムの実行が110行目きたとき停止します。再実行で停止した行から実行します。



左ダブルクリックするとカーソル行に白丸を表示する。
再度、左ダブルクリックすると解除する。

(3)割り込み側にもブレークポイント設定する。

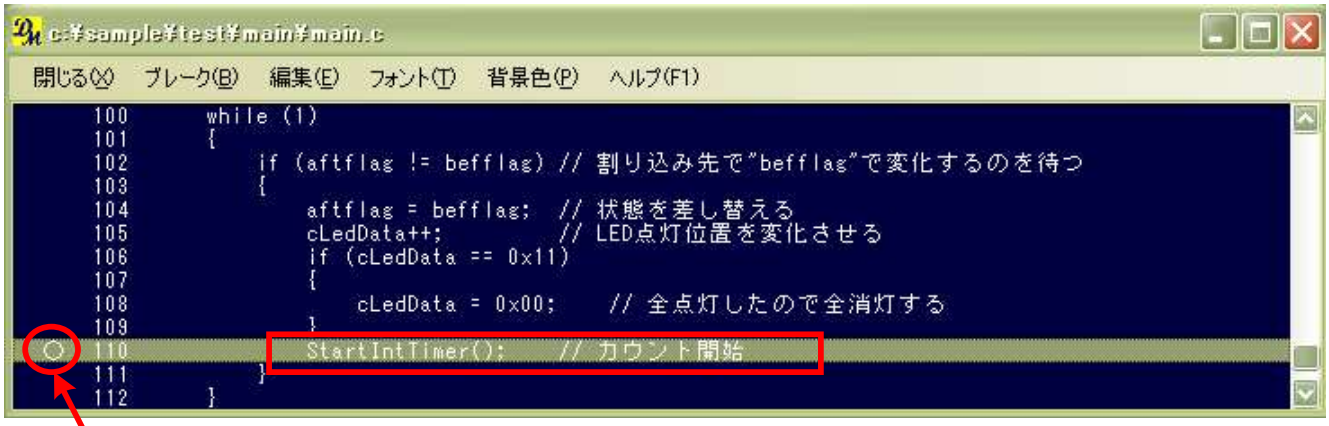


割り込みの先頭にブレークポイントを設定する。

(4)ブレークポイントの有効/無効の設定をする

ブレークポイント設定はそのままで、ブレークポイントで停止させたくない時に設定します。無効化になっていると、その行に来ても停止しないで実行を続けます。

ブレークポイント設定した行で、右ダブルクリックするとブレーク設定が無効になり"白抜き丸"になります。
ブレークポイント無効化した行で、右ダブルクリックするとブレーク設定が有効になり"白丸"になります。



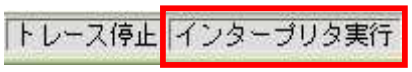
右ダブルクリックするとブレーク設定が無効になり"白抜き丸"になる
再度、右ダブルクリックするとブレーク設定が有効になり"白丸"になる

11.4. 実行方式を決定する

 をクリックして実行方式を切り替えます。停止して”インタープリンタ実行方式”から”システム実行方式”に切り替えができます。又、その逆へ切り替えることもできます。

(1)インタープリンタ実行方式

1ステップ毎に取得して実行するの、詳細なトレース結果を得られます。
今回はこの方式でデバッグします。ステータスバーに”インタープリンタ実行”が表示されます。

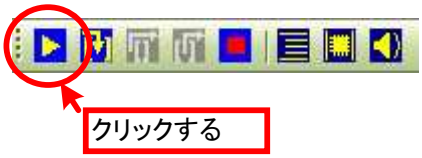


(2)システム実行方式

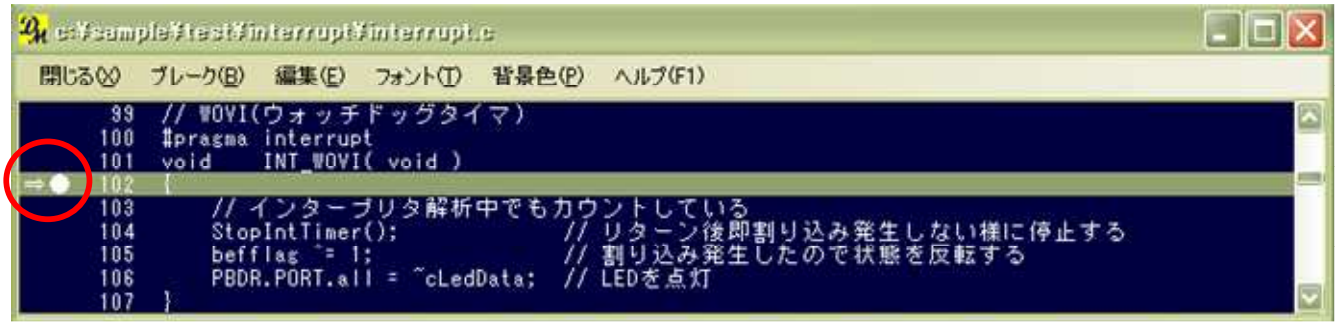
直接ユーザプログラムを実行するので、処理速度はROM化した速度に近い状態で処理できます。但し、実行中の詳細なトレース結果を得ることはできません。(ステップ実行又はブレークポイントで停止した時点のトレース結果は取得でます)
ステータスバーに”システム実行”が表示されます。

11.5. プログラムの実行をする

(1)実行開始をクリックします。

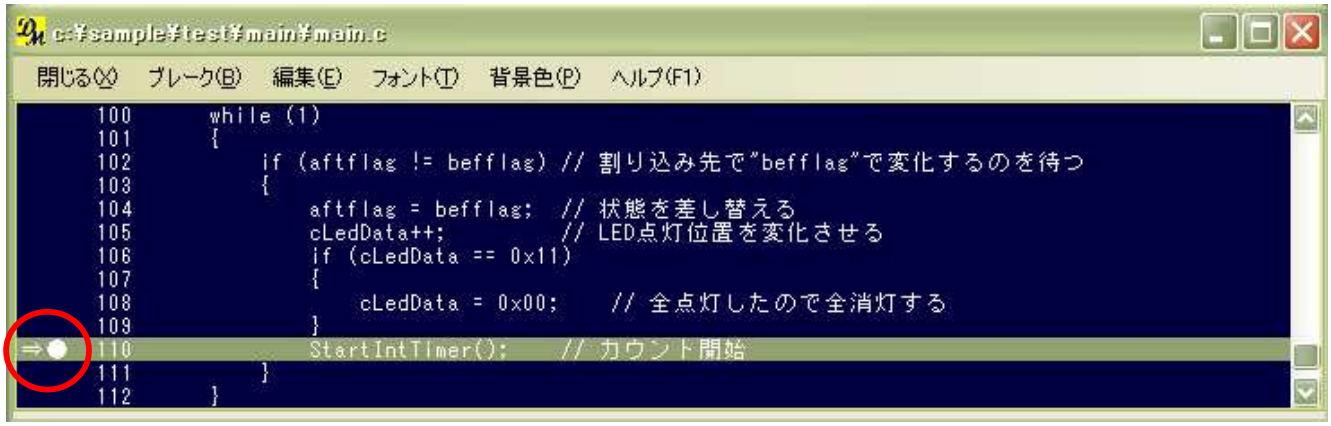


(2)ブレークポイント設定した行で停止します。カールが停止行に移動し”⇒”が表示されます。



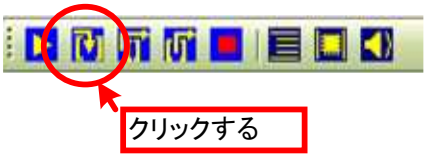
(3)実行開始をクリックします。
(1)と同じボタンをクリックします。

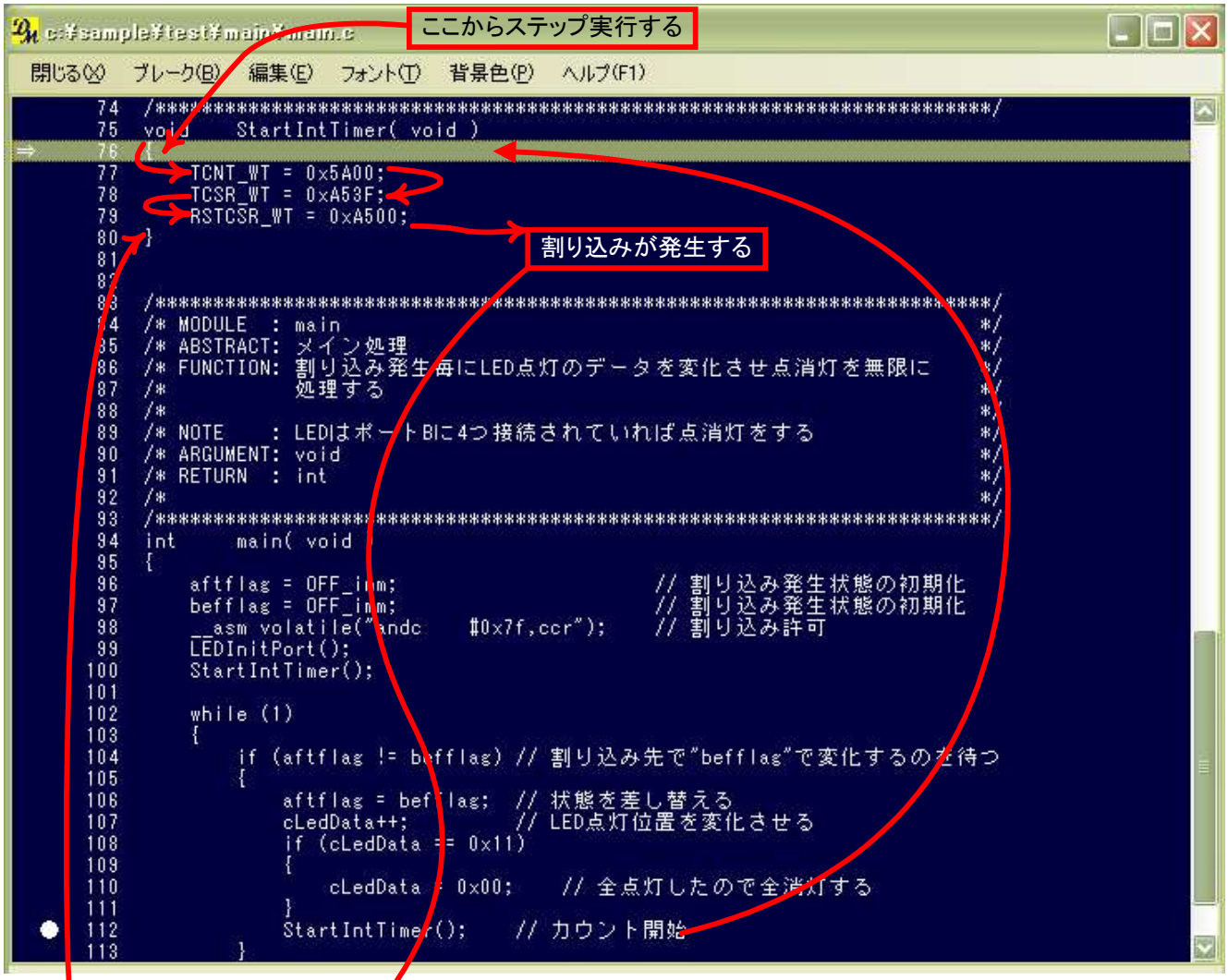
(4)2箇所目ブレークポイント設定した行で停止します。カールが停止行に移動し”⇒”が表示されます。

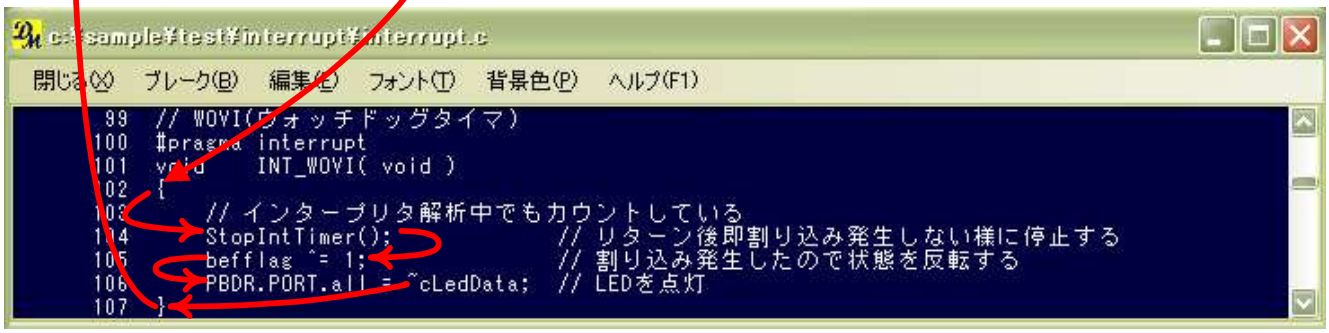


(5)1行毎実行する

1行毎に実行してみます。
ステップインをクリックして、停止行から実行を再開します。







以上の様に他のステップも動作を確認してみて下さい。

別のプログラムをデバッグするときは、前のプログラムが使用したハード状態、レジスタ、スタック状態を初期状態に戻す必要が有るため、一度ハードリセットをして下さい。

11.6. 割り込み関数処理の補足説明

タイマー関係の割り込み処理があるプログラムをデバッグ中でブレーク繰り返すと、割り込み発生しなくなることが有ります。
対処方法としてカウント開始の前後どちらかに割り込み許可処理を入れてください。

12. 組み込み関数機能とは

組込み関数とは、親プロジェクトのモジュール関数群と別に単独で実行する関数です。
デバッグモニタは外部RAMを有効にする方法をしらないので、後付でデバッグモニタにターゲットボードの外部RAMを有効にさせる為の機能です。
この組み込み関数は、デバッグするプログラム本体を外部RAMにダウンロードする前に実行して、外部RAMを使用出来る様します。この関数はデバッグモニタの一部分として機能するため、デバッグモニターのスタック領域を使用しているので大きいプログラムを組み込むことは出来ません。

この機能を使用する為には内臓RAMが4KB以上のMCUが必要になります。

等価処理として、内臓RAMだけで動作する外部RAMを有効にするプログラムを実行後、本来のデバッグするプログラムをダウンロードして、デバッグする方法が有ります。

12.1. 組み込み関数の使用条件

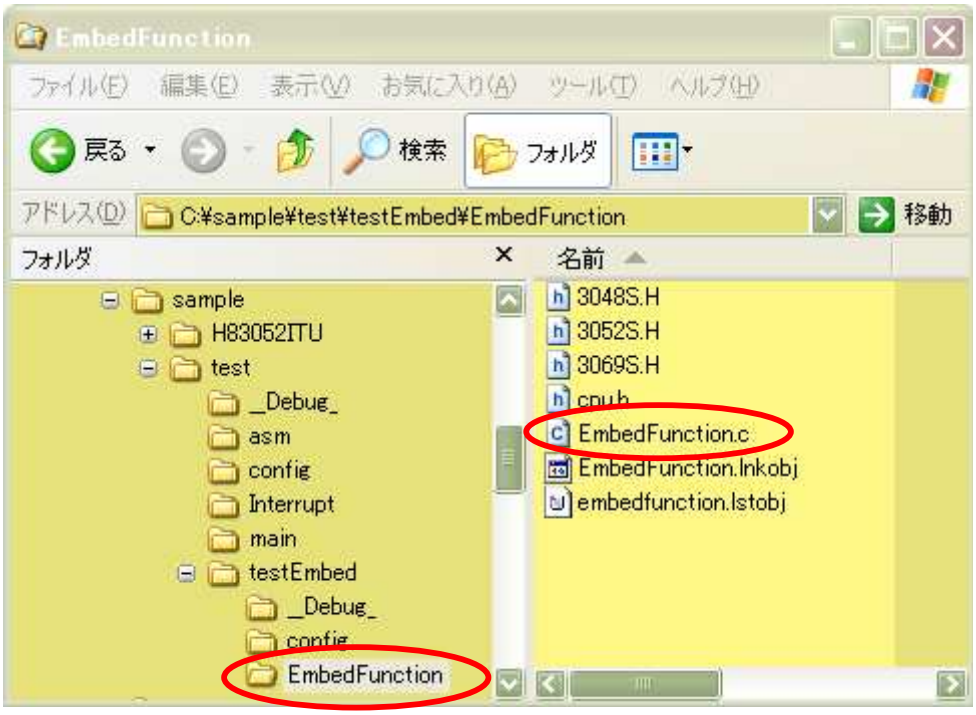
組込み関数は、デバッグモニターに後から追加する(一時的に追加)機能のプログラムで、一つの独立したプログラムと同様です。
デバッグはデバッグモニタのスタック領域を使用して実行するためにAUTO変数は使用しないでください。
又、外部RAMを有効にする処理に絞り込んでください。

12.2. デバッグ方法

デバッグは、サンプルのデバッグと同様の要領で操作して下さい。
デバッグ完了後は、一度ハードリセットをして下さい。

12.3. 記述するソースファイル

作成したプロジェクトフォルダ内の”xxxxxxEmbed”フォルダの”EmbedFunction”フォルダにある”EmbedFunction.c”に記述します。



12.4. サンプルコード

サンプルコードは秋月電子の、”AKI－H8／3052F USB開発セット”、”H8／3069F USBホストボード”及び”H8／3069Fネット対応マイコンLANボード”を記載して置きます。詳細仕様は秋月電子の穂ムーページを参照して下さい。
他にも色々と有りますが、使用するボード仕様に合う様に書き換えて下さい。

(1)AKI－H8／3052F USB開発セット

```
void    EmbedFunction(void)
{
        P1DDR.PORT.BYTE = 0xFF;   /* A0~A7  address                */
        P2DDR.PORT.BYTE = 0xFF;   /* A8~A15 address               */
        P2PCR.PORT.BYTE = 0x00;   /* ポート2をPull Up Offに設定     */
        P3DDR.PORT.BYTE = 0x00;   /* D0~D8 Data                   */
        P4DDR.PORT.BYTE = 0x00;   /* ポート4入力に設定              */
        P4PCR.PORT.BYTE = 0xFF;   /* ポート4をPull Up Onに設定      */
        P5DDR.PORT.BYTE = 0xFF;   /* A16~A19 Address              */
        P5PCR.PORT.BYTE = 0x00;   /* ポート5をPull Up Offに設定     */
        P6DDR.PORT.BYTE = 0xF8;   /* bit6~3 OUT bit2~0 IN         */
        P8DDR.PORT.BYTE = 0xFF;   /* ポート8出力に設定              */
        P9DDR.PORT.BYTE = 0xDF;   /* Bit5 IN                      */
        PADDR.PORT.BYTE = 0xF0;   /* bit7~4 OUT bit3~0 IN         */
        PBDDR.PORT.BYTE = 0xFF;   // ポートB出力に設定
}
```

(2)H8／3069F USBホストボード

```
void    EmbedFunction(void)
{
        P1DDR.PORT.BYTE = 0xff;   // A0-7  is enable
        P2DDR.PORT.BYTE = 0xff;   // A8-15 is enable
        P8DDR.PORT.BYTE = 0xfc;   // CS1-3 is enable
        RTCOR = 0x0a;             // リフレッシュタイムコンスタントレジス
        RTMCSR = 0x1f;            // リフレッシュタイマコントロール(φ／32 でカウント)
        DRCRB = 0x98;             // 8bitアクセス空間A23～ A10/リフレッシュサイクルを禁止
        DRCRA = 0x30;             // DRAM 空間~CS2
}
```

(3)H8／3069Fネット対応マイコンLANボード

```
void    EmbedFunction(void)
{
        P1DDR.PORT.BYTE = 0xff;   // A0-7  is enable
        P2DDR.PORT.BYTE = 0x07;   // A8-10 is enable
        P8DDR.PORT.BYTE = 0xec;   // CS1-2 is enable
        RTCOR = 0x9b;             // リフレッシュタイムコンスタントレジス(64KHz refresh)
        RTMCSR = 0x0f;            // リフレッシュタイマコントロール(φ／2 でカウント)
        DRCRB = 0x98;             // 8bitアクセス空間A23～ A10/リフレッシュサイクルを禁止
        DRCRA = 0x10;             // DRAM 空間~CS1
}
```

13. 機能一覧

デバッグモニタの機能一覧を記載します。

13.1. ファイルメニュー

メニュー	サブメニュー	機能
ファイルプロジェクト	新規プロジェクト	プロジェクトのフレームワークを作成します。 ここで作成したプロジェクトが親プロジェクトに成ります。
	開く	既存のプロジェクト開きます。
ソースリスト開く	ー	ソースリストを開きます。 ソースリスト上にブレーク設定が出来ます。
組込み関数	関数プロジェクト	親プロジェクトに付属するサブプロジェクトのフレームワーク作成をします。
	関連付け	既存のサブプロジェクトを親プロジェクトに付属させる様関連付けします。
	関連解除	親プロジェクトに関連付けられている、サブプロジェクトの関連付けを解除します。 サブプロジェクトのフォルダーは残ります。
終了	ー	デバッグモニターを終了します。
作成したプロジェクト名	ー	既存のプロジェクト開きます。親プロジェクト、サブプロジェクトの全てを表示します。

13.2. プログラムロードメニュー

メニュー	サブメニュー	機能
プログラムロード	ー	プログラムをターゲットボードに送信します。組み込み関数があれば同時に処理します。

13.3. プログラム実行メニュー

メニュー	サブメニュー	機能
連続実行	ー	実行指示アドレスからブレークポイント又は実行停止が指示されるまで実行します。
ステップ	ー	ステップイン 実行指示アドレスの1ステップを実行します。
実行アドレス指定	ー	実行アドレスを指定します。
実行停止	ー	実行停止します。
強制停止	ー	強制的に実行停止をします。 システム実行方式では、ターゲットを優先するので停止できなくなることがあります。 停止位置のソースリストはアッセンブラ表示に成ります。

13.4. 表示・設定メニュー

メニュー	サブメニュー	機能
変数ウォッチ	ー	変数内容を表示します。
メモリダンプ	ー	開始アドレスから最大2000バイトまでメモリ内容を表示します。
メモリライト	ー	メモリ内容を表示し変更します。
ブロック コピー/移動	ー	コピー元開始アドレスから指定バイト数分コピー先開始アドレスにコピーします。 移動はコピー後元のメモリを”0x00”で埋め込みます。
同一文字埋め[Fill]	ー	開始アドレスから最大64Kバイト分指定文字で埋め込みます。
レジスター	ー	レジスター内容を表示します。
ソースリスト開く	ー	ソースリストを開きます。“ソースリスト開く” と同じ機能です。
ログ表示	ー	ロギングフォームを表示します。

13.5. ブレークポイントメニュー

メニュー	サブメニュー	機能
プログラムブレーク	ー	プログラムのブレークポイント状態を表示します。 プログラムを一時停止したいソース行を指定します。 ブレーク有効状態はチェックボックスがマークされます。無効のときはマークが無くなります。 ここで設定したブレーク情報は、ソースリスト上に表示されます。 ＊ブレークポイント設定には上記の他に、ソースリストからブレークポイント設定が出来ます。

13.6. 履歴メニュー

メニュー	サブメニュー	機能
トレース開始・停止	開始	実行状態をトレースします。 ・インタープリター解析の時は実行した全てを記録します。 ・システム実行の時は停止したステップのみ記録します。
	停止	トレースを停止します。下記のトレース区間設定の停止もします。
トレース区間設定	ー	インタープリター解析中で且つCソース上のみ指定可能です。 ソースリストでトレース開始から停止までの区間を設定し、その区間の実行状態をトレースします。
トレース消去	ー	記録したトレース結果を消去します。
トレース表示	ー	記録したトレース結果を表示します。
カバレッジ表示	ー	記録したトレース結果から、指定モジュールのカバレッジを表示します。

13.7. 環境設定メニュー

メニュー	サブメニュー	機能
ソース/ツール	ー	親プロジェクトを作成した後に、ソースファイル、コンパイラーの登録フォルダー、ソースをコンパイルするためのMakeファイルのオプション定義、テーゲットのメモリ配置を定義するリンカースクリプト等を定義します。 ・プロジェクトのソース/ヘッダーを登録します。 ・プロジェクトで使用するコンパイラー、エディター等のツールを定義します。 ・ソースのコンパイルオプション設定をして、makeファイルを作成します。 ・コンパイル・リンクするとき、ターゲットのメモリー配置を定義するリンカースクリプトを作成します。
シリアルポート	シリアルポート	ターゲットボードと通信するシリアルポート番号の指定をします。 (ディフォルト:COM1)
	接続	ターゲットボードとの通信のシリアルポートをオープンします。 (ディフォルト)
	切断	ターゲットボードとの通信のシリアルポートをクローズします。
デバッグ要件	ー	実行方式、割り込み解析等のデバッグするため環境を設定します。

13.8. コンパイルメニュー

メニュー	サブメニュー	機能
コンパイル	ー	ターゲットソースのコンパイルをします。

13.9. ウィンドウメニュー

メニュー	サブメニュー	機能
重ねて表示	ー	重ねて表示します。
上下に並べて表示	ー	上下に並べて表示します。
左右に並べて表示	ー	左右に並べて表示します。

13.10. ヘルプメニュー

メニュー	サブメニュー	機能
DebugMonitorのヘルプ	ー	本デバッグモニターのヘルプを表示します。
アッセンブルリスト生成	ー	コンパイル後のアツセンブリリストにアドレスと機械語の生成をします。
ソースディリミッタ	Windows(CR LF)	ソースのディリミッタを“CR・LF”に変更します。
	Unix(LF)	ソースのディリミッタを“LF”に変更します。
バージョン情報	ー	本デバッグモニターのバージョンとシリアル番号を表示します。
登録番号	ー	登録番号を表示します。

13.11. ツールボタン

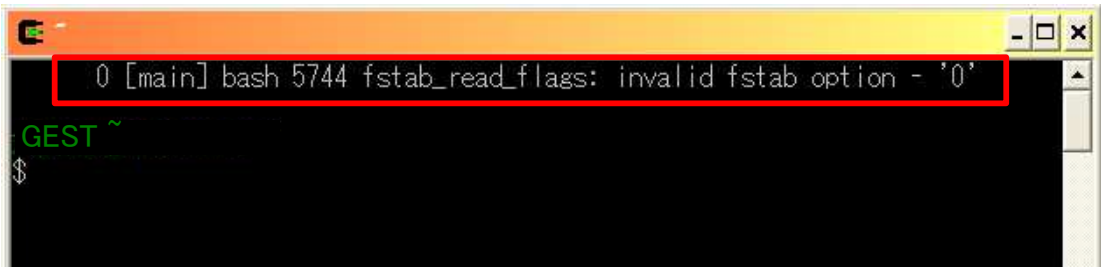
ボタン名	ツールボタン	機能
実行開始		実行指示アドレスからブレークポイント又は実行停止が指示されるまで実行します。
ステップイン		実行指示アドレスの1ステップを実行します。
ステップオーバー		関数実行して次のステップで停止します。
ステップアウト		実行中の関数から抜けます。
実行停止		実行停止をします。
ソースコート切替え		Cソース・アッセンブラソース/コンパイラ生成アッセンブラの切替えをします。
実行方式切り替え		インタープリタ実行/システム実行の実行方式の切り替えをします。
実行停止時のブザー		実行停止時のブザーのON/OFF切り替えをします。

14. 補足

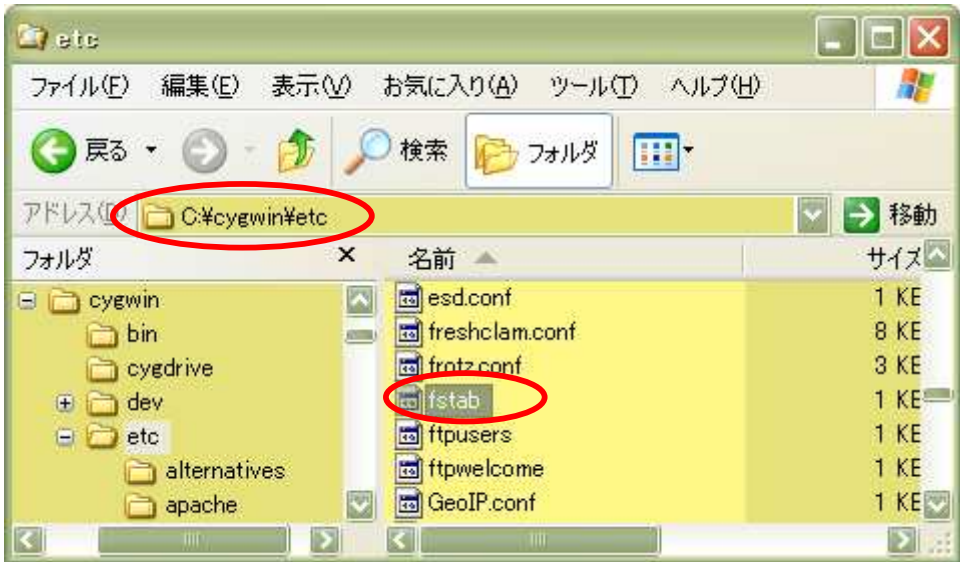
14.1. Cigwinのインストール

ttp://www.cygwin.com/ で、「Install or update now!」をクリックして、セットアッププログラム setup.exe をダウンロードしてセットアップして下さい。
詳細なセットアップ方法は、インターネットで”cygwin インストール”で検索し、判りやすいホームページを参照して下さい。

(1)cygwin起動に”invalid fstab option – ‘0’ ”が表示する時の対処法



●etcフォルダー内の ”fstab”を変更する



---------- インストール直後の記述内容 ----------
For a description of the file format, see the Users Guide
http://cygwin.com/cygwin-ug-net/using.html#mount-table

/usr/X11R6/lib/X11/fonts some_fs binary 0 0
none /cygdrive cygdrive text,posix=0,user 0 0
---------- 上記行まで----------

↓

---------- 変更後の記述内容 ----------
For a description of the file format, see the Users Guide
http://cygwin.com/cygwin-ug-net/using.html#mount-table

/usr/X11R6/lib/X11/fonts some_fs binary 0 0
#none /cygdrive cygdrive text,posix=0,user 0 0

none /cygdrive cygdrive noacl,posix=0 0 0
c:/cygwin/home/GEST /home/GEST ntfs noacl,posix=0 0 0
c:/cygwin/usr/local /usr/local ntfs noacl,posix=0 0 0
c:/cygwin/tmp /tmp ntfs noacl,posix=0 0 0
---------- 上記行まで----------

インストール内容によって若干内容が違うと思いますが、使用するフォルダをマウントしています。

グレーはコメントにした行、赤色は追加した行です。さらにマウントしたいフォルダがあれば追加して下さい。

参考ホームページ:
http://www.cygwin.com/
cygwinのホームページの”Cygwinのバージョン1.5.xからの更新”
http://unixlife.jp/unixlife/linux/sys-fstab.jsp

14.2. gccのインストール

”cygwin インストール”後に”C:¥cygwin¥home¥xxxxxxxx¥”直下に”install”フォルダを作り、そのフォルダ内に ソースコードパッケージファイルを保存して下さい。保存後”cygwin”を起動してgccのインストールをして下さい。

cygwinはCドライブ直下にインストールしたことを前提に記述しています。
xxxxxxxx:ユーザー別のフォルダ名

(1)ソースコードを取得する

必要なソースは gcc, binutils, newlib です。以下の場所からそれぞれ取得します。
又、各ソースの最新バージョンでないので、最新バージョンを取得して下さい。
以下のバージョンはデバッグモニタをコンパイルしているバージョンです。

●ダウンロードページとソースコードパッケージファイル名

http://ftp.gnu.org/pub/gnu/binutils/
　binutils-2.17.tar.gz
http://quox.org/install/gnu/gcc.html -> http://quox.org/install/gnu/gcc-4.1.1.html
　gcc-4.1.1.tar.bz2
ftp://sources.redhat.com/pub/newlib/
　newlib-1.15.0.tar.gz

(2)インストールする

1)以下のスクリプトファイルを作成して、”C:¥cygwin¥home¥xxxxxxxx¥install”フォルダに保存する。
　ファイル名は”elf.sh”として下さい。
2)”cygwin”を起動する
3)コンソールに以下のコマンドを入力する
$cd install
$bash elf.sh
インストールを開始します。

インストールスクリプトファイル”elf.sh”

```
#!/bin/sh
tar zxvf binutils-2.17.tar.gz
cd binutils-2.17
./configure --target=h8300-elf --prefix=/usr/local
make
make install
cd ..

export PATH=/usr/local/bin:$PATH
tar jxvf gcc-4.1.1.tar.bz2
cd gcc-4.1.1
./configure --target=h8300-elf --prefix=/usr/local --enable-languages=c,c++
make
make install
cd ..

tar zxvf newlib-1.15.0.tar.gz
mkdir newlib-1.15.0-h8300
cd newlib-1.15.0-h8300
../newlib-1.15.0/configure --target=h8300-elf --prefix=/usr/local
make
make install
cd ..
```

(3)gccがインストールが出来ないとき

ソースを以下のバージョンからインストールして下さい。
　binutils-2.17.tar.gz
　gcc-4.1.1.tar.bz2
　newlib-1.15.0.tar.gz