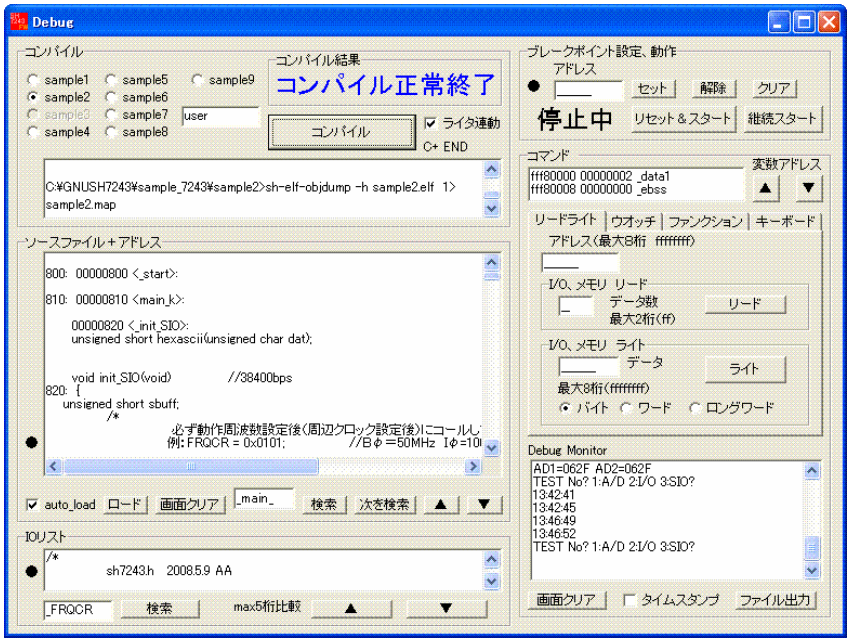
ＢＣＳＨ７２ＸＸ用

フォース®デバッカＳＨ　の使い方

第１版２００８．１２．０５



１．デバッカ機能、使い方

１－１　概要

１－２　コンパイル

１－３　ソースファイル＋アドレス

１－４　ＩＯリスト

１－５　ブレークポイント

１－６　変数アドレス

１－７　コマンド

ａ）リードライト

ｂ）ウオッチ

ｃ）ファンクション

ｄ）キーボード

１－８　タイムスタンプ、デバック状況ファイル出力

１－１　概要

本デバッカはフラッュ ROM 書き込みソフトウエア　フォース®ライタの拡張機能として開発されました。また、従来、単独プログラムとして提供されていた「コンパイラー」がデバック画面に組み込まれています。これにより、プログラムの開発に必要な、コンパイル、フラッシュＲＯＭ書き込み、デバックという一連の作業が有利な連結を伴ってシームレスに行えるようになっています。

デバッカは CPU 内蔵の UBC（ユーザーブレークコントローラ）を使用したリモートデバッカです。インサーキットエミュレータを使用しなくとも、単体でプログラムをデバックできます。

内部フラッシュ ROM に書かれたプログラムが実行している最中に UBC のハードウエアでブレークを掛けるため、デバックと、実動作時で速度が変わらない理想のリアルタイムデバックです。※１

プログラムの変更なしに好きなアドレスにブレークポイントを設定してメモリや I/O の値をリード、ライトすることができます。※２

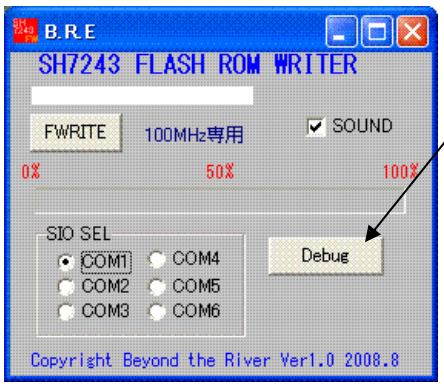
また、この他に特定のメモリ、I/O の値と設定値を比較してアラームを出力するウオッチ機能が使用できます。デバックログのファイル化や、タイムスタンプ機能と相まって頻度が少ないエラー発見、長時間の正常動作確認、ログファイルによるエラー解析等、今までのデバッカを越える、強力な機能も使用できます。

※１　デバック時にデバック機構の制限により CPU の最高性能が低下することはありません。
※２　ブレークポイントの変更の度にフラッシュ ROM を書き換えることがありません。

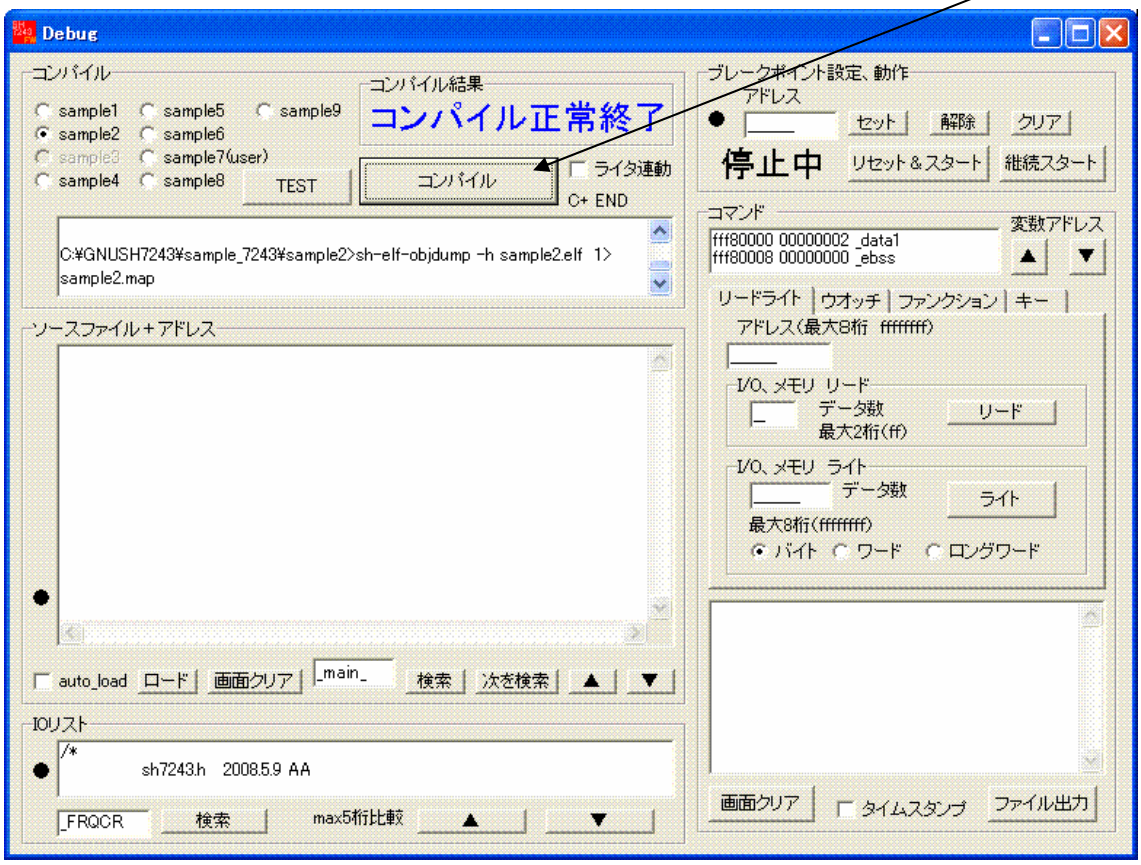
勝手ながら以下の説明は「SH72XX 開発セット」全般、特に「sample6」の取扱説明書を読了されているという前提となっています。以下の説明は SH7243 用フォースライタで行いますが、SH7211 用も使い方は同じです。

１－２　コンパイル

フォース®ライタ　ｆｒｗ７２ＸＸ．ｅｘｅの「デバック」をクリックします。

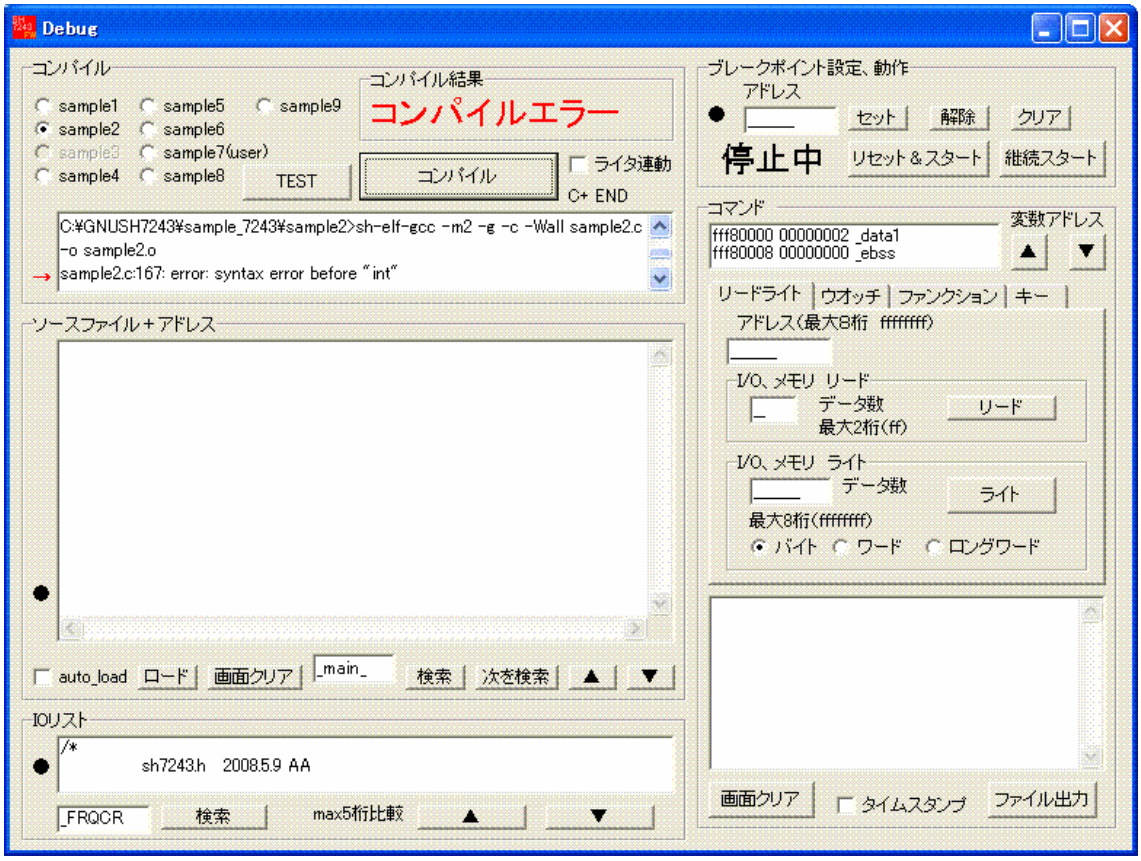


例として開発セットのｓａｍｐｌｅ２をコンパイルしてみます。ｓａｍｐｌｅ２を選択し、コンパイルをクリックします。

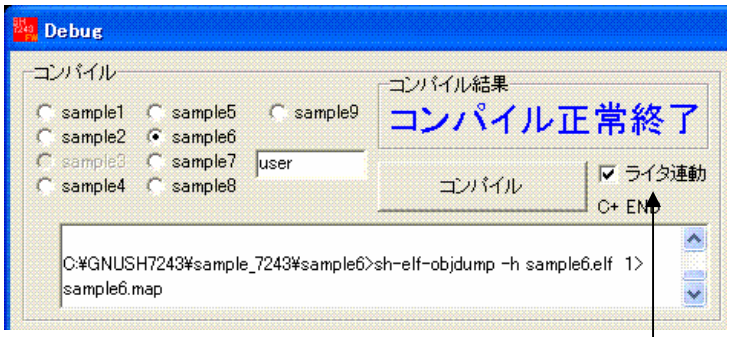


問題なければ「コンパイル結果」欄に「コンパイル正常終了」と表示されます。問題がある場合「コンパイルエラー」「リンクエラー」と赤字で表示されますので、ソースファイルを修正する必要があります。

下記例ではｓａｍｐｌｅ２．ｃにわざと「ｐｐｐｐ」と書き込み、エラーを出しました。
エラー行が矢印→で表示されますので、エディタでその行か、１行上の行を調べてエラーを見つけてください。
例では１６７行目にエラーがあると表示されています。

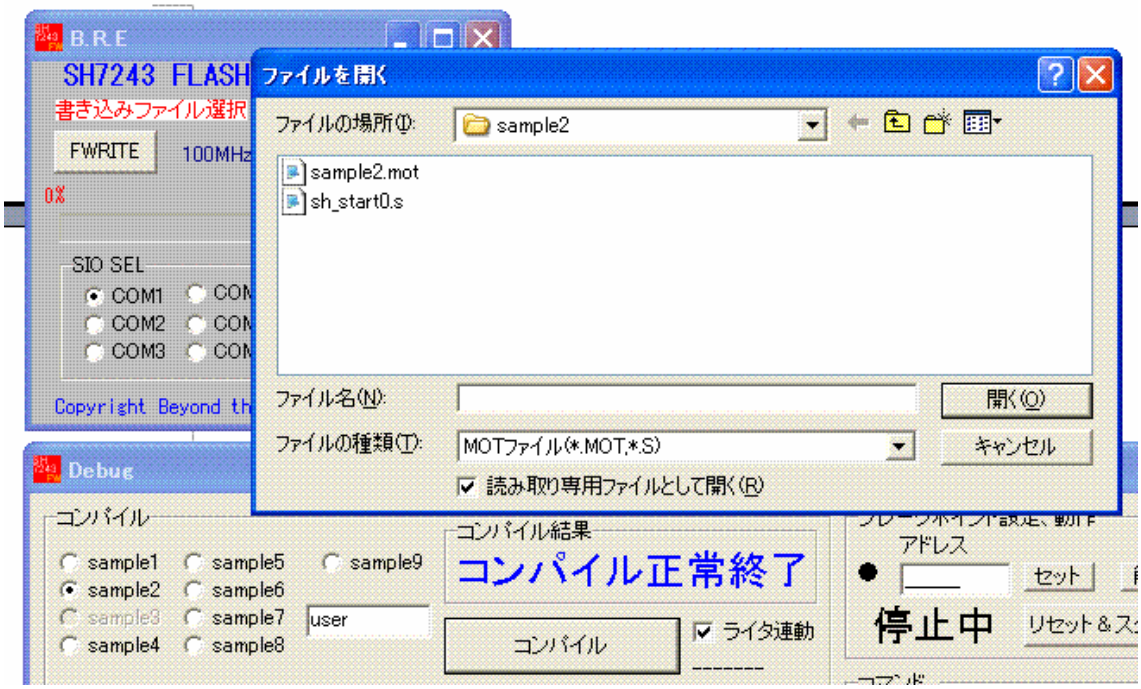


なお、コンパイラーは warning（警告）は通常、問題なしとしますが、「コンパイル正常終了」と表示されても、ｍｏｔファイルが生成されない場合が稀にあります。Ｎｏ　ｓｕｃｈ　ｆｉｌｅ　ｏｒ　ｄｉｒｅｃｔｏｒｙと表示されますので、ソースファイルの修正、再コンパイルが必要です。



プログラムを修正した場合、必ずコンパイル、書き込みが必要です。「ライタ連動」をチェックしておくと、コンパイルが正常終了したときのみ、ライタが起動して書き込み準備に入りますから便利です。
「コンパイル正常終了」→「プログラム書き込み」を行います。

例：sample2 をコンパイルし、「コンパイル正常終了」だったので、自動的に書き込むファイルが表示されているところ。ここでは sample2.mot を選択し、書き込みます。

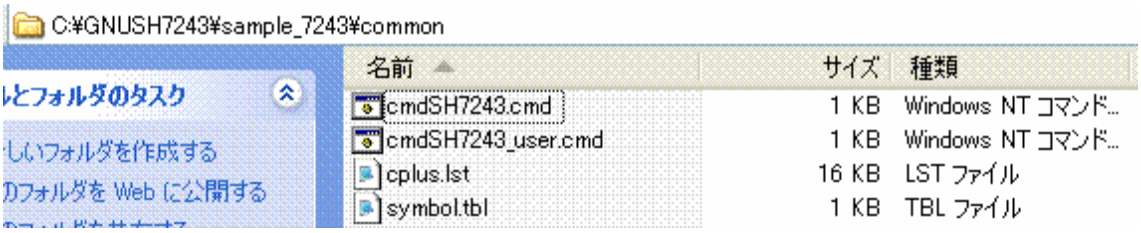


１－３　ソースファイル＋アドレス

　コンパイルをすると、ユーザーが製作したソースファイルに絶対番地が割り振られます。ソースファイル＋アドレス（窓）はそれを表示し、検索等行い、ブレークポイント設定時に参照するための窓です。

※画面はエディタではありません。コピーはできますが、書き込みはできないのでご注意ください。本環境のエディタはユーザーが使い慣れた個々のエディタをご使用いただく前提です。

ソースファイル＋アドレスファイルは c:\GNUSH72XX\bre-sample\common の中にある cplus.lst ファイルです。コンパイルのたびに更新されます。プリンターで印刷できます。



　例としてｓａｍｐｌｅ６のソース＋アドレスファイルが読み込まれた状態を示します。右がユーザーが作成したプログラム、左がコンパイラが割り振ったアドレスです。
「ａｕｔｏ　ｌｏａｄ」をチェックしておくと、コンパイル正常終了時に自動的に読み込まれ便利です。

例えばアドレス 820 番地

```
          void  init_SIO(void)          //38400bps
820:      {
```

は関数 init_SIO の先頭アドレスを示しています。自分で書いた記憶の無いソースファイルですが、

#include "SIO_SH72XX.h"

と書いて include した SIO_SH72XX.h がここに展開されています。ユーザーが書いたｍａｉｎ関数は普通、一番後ろにあります。

下のキーは

「ａｕｔｏ＿ｌｏａｄ」：コンパイルが正常終了すると自動的にソース＋アドレスファイルを読み込み、表示します。

「ロード」　：ソース＋アドレスファイルの読込

「画面クリア」　：画面クリアです。

「検索」　：ユーザーがブレークポイントを掛けたい部分のアドレスを知るために文字検索が行えます。0 番地から検索します。最大 6 文字入力です。デホルトは_main です。

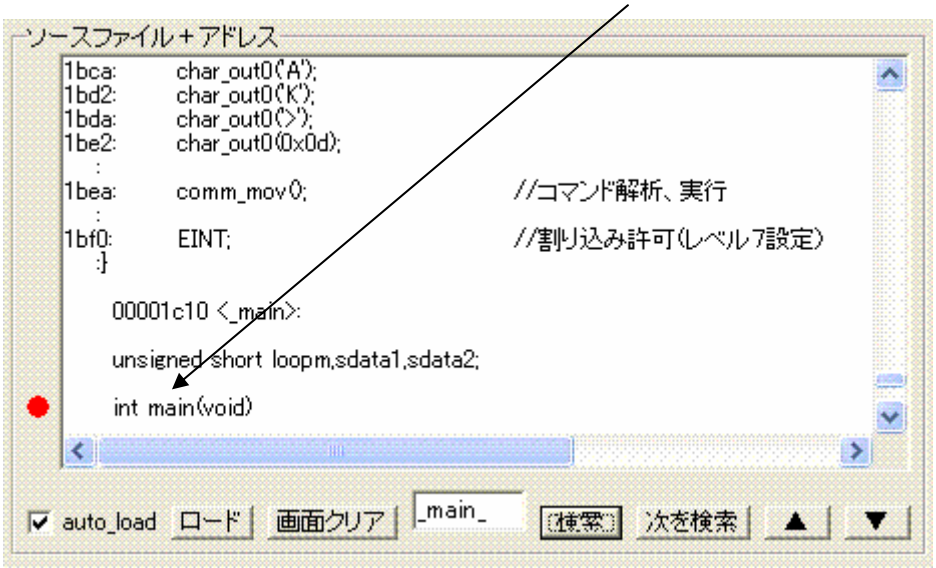
「次を検索」　：0 番地からでなく次を検索します。

「▲」　：１行上を表示します。

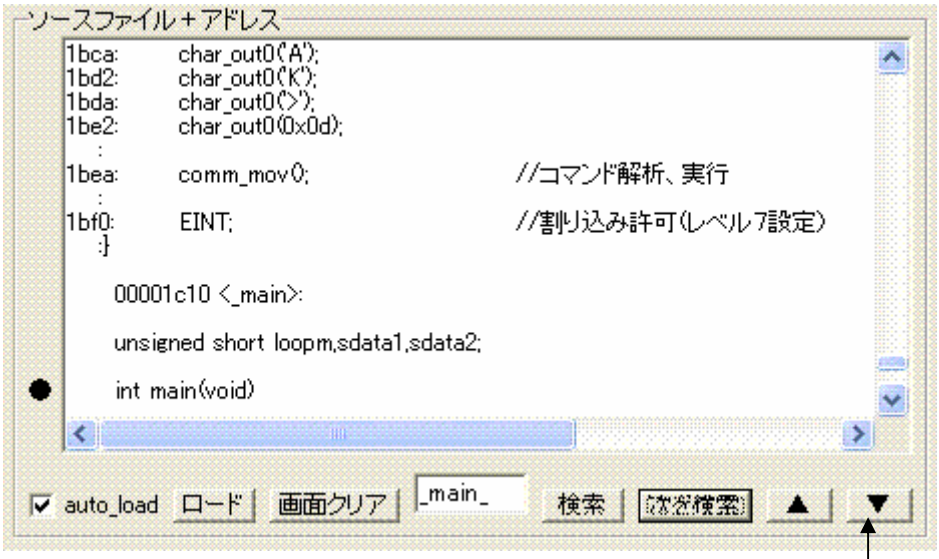
「▼」　：１行下を表示します。

例としてデホルトの_main を検索してみます。

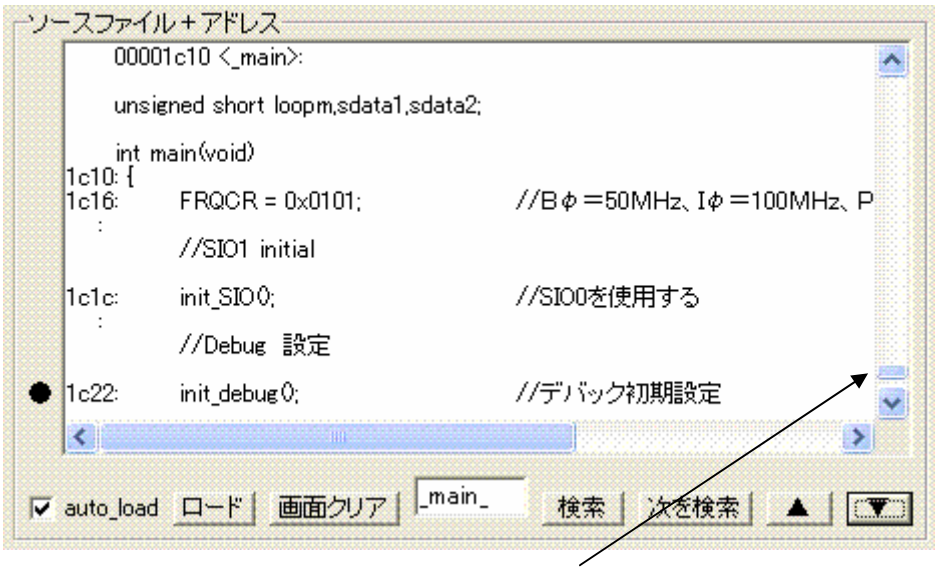
検索して、ヒットすると左の●が●になり、右にヒット文字が表示されます。



「次を検索」をクリックしましたが、__ｍａｉｎは無いようです。●になります。



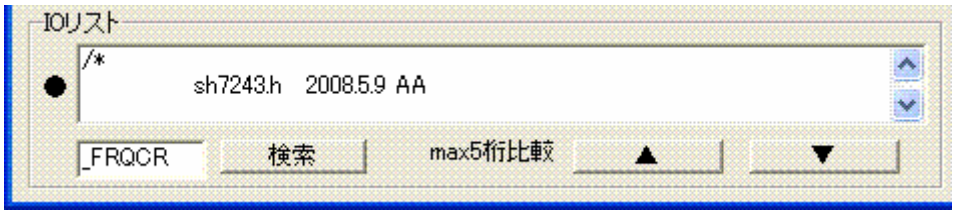
「▼」を何回かクリックして__ｍａｉｎの中を表示させます。



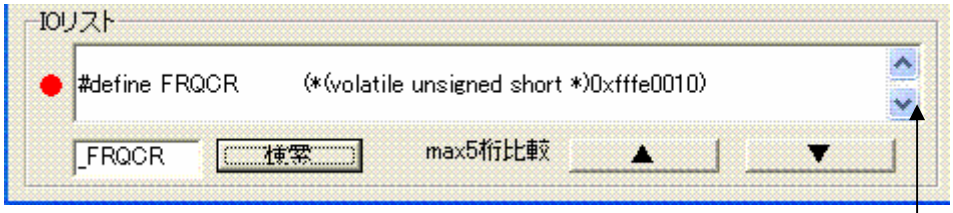
「▲」「▼」は１クリック１行変わります。スクロールバーはマウスで連続して変わります。

１－４　ＩＯリスト

IO リストは CPU の I/O の番地、形を知るための窓です。SH72XX.h ファイルを読み込んでいます。



他の窓と同様に検索ができます。ヒットすると左の●が●になります。最大 5 桁文字や数値が入力できます。



「▲」　：1 行上を表示します。

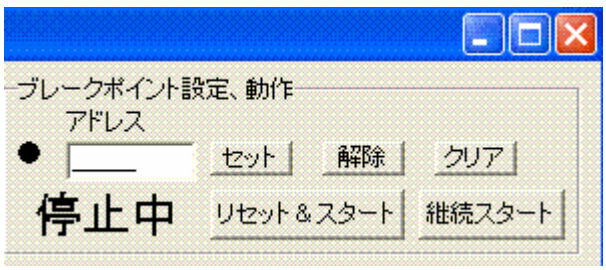
「▼」　：1 行下を表示します。

表示が連続して移動します。

例として FRQCR が表示されています。アドレスが fffe0010H、short なので 2 バイトデータであることがわかります。後述する I/O リード、ライトを行うときに、アドレスをこの表示を参考に書くか、コピー＆ペーストすることができます。

１－５　ブレークポイント

ブレークポイントはプログラムを一時停止させたい部分のアドレスを設定します。プログラムがそのアドレスになると一時停止して、メモリや I/O の読み書き等が出来ます。



「アドレス」　：アドレスを設定　0～ffffffff　８桁　左詰めで設定します。

「セット」　：ブレークポイントをアドレスで示したアドレスに設定します。

「解除」　　：ブレークポイントを解除します。

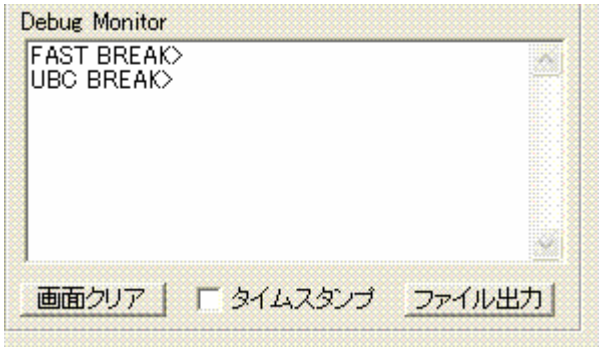
「クリア」　　：アドレスをクリアします。

「リセット＆スタート」　　：CPU ボードにリセットを掛けてプログラムを 0 番地から動作させます。過去に設定したブレークポイントも解除されますので、必要に応じて下記条件時に再度「セット」してください。

DebugMonitor に FAST　BREAK>　と表示されます。

「継続スタート」　　：プログラムが一時停止した番地の次から動作させます。設定等は継続されます。消去されません。

「セット」「解除」「クリア」はいずれも DebugMonitor に　FAST　BREAK＞または UBC　BREAK＞と表示されていて、CPU が「停止中」である時、操作可能です。



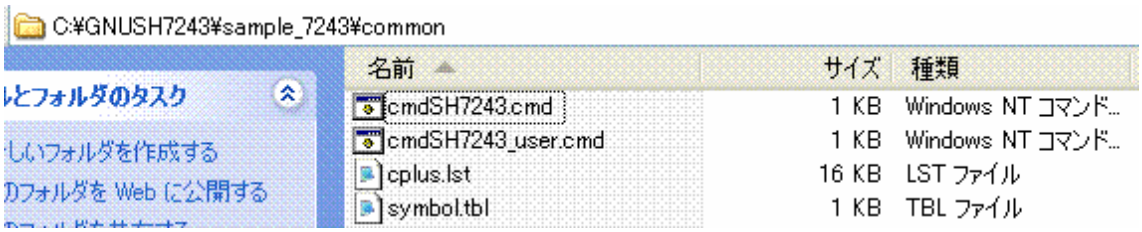
FAST　BREAK＞　　：init_debug（）関数が実行されたときに表示されます。CPU はパソコンからのコマンド待ち状態です。「リセット＆スタート」クリックで表示されます。

UBC　BREAK＞　　：ブレークポイントでブレークした時に表示されます。CPU はパソコンからのコマンド待ち状態です。「継続スタート」でブレークしたアドレスの次から動作を再開します。

１－６　変数アドレス

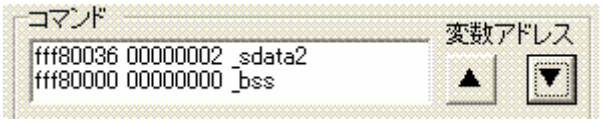
コンパイルすると自動的にシンボルテーブルアドレスが製作され、読み込まれまれ、表示されます。

シンボルテーブルファイルは c:\GNUSH72XX\bre-sample\common の中にある symbol.tbl ファイルです。コンパイルのたびに更新されます。プリンターで印刷できます。

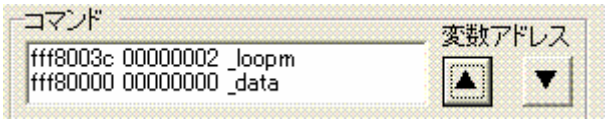


sample6 を例に挙げるとグローバル変数　unsigned short loopm,sdata1,sdata2;
がその振られた絶対番地とデータサイズを確認することができます。
「▲」「▼」で上下できます。検索機能はありません。

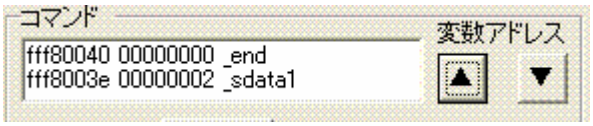
sdata2=fff80036H　2 バイトデータだと分かります。



loopm＝fff8003cH　2 バイトデータだと分かります。



sdata1=fff8003eH　2 バイトデータだと分かります。

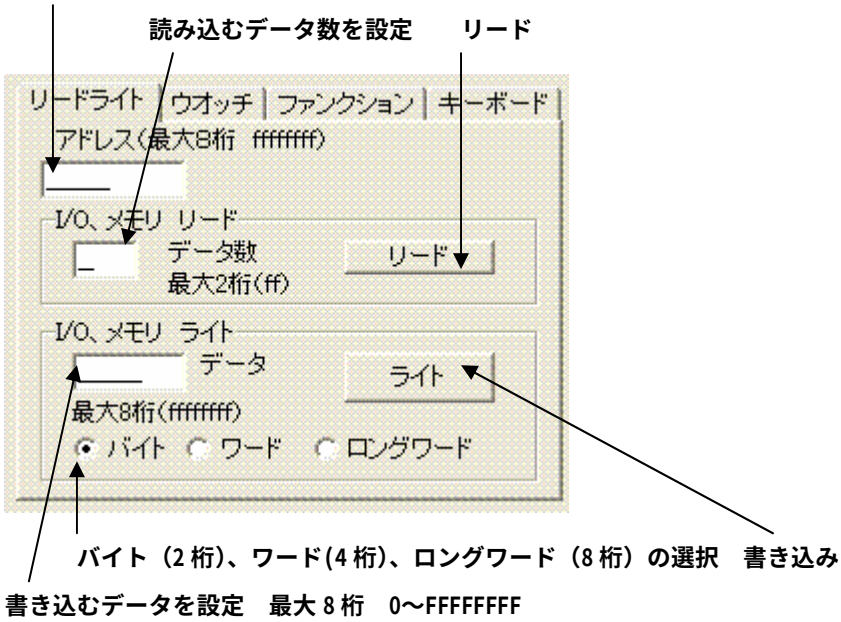


　この窓のアドレスを次項のリード、ライトアドレス等にコピー＆ペーストして変数の内容を読み書きすることができます。本機能はローカル変数は対応していません。調べたい変数がある場合、グローバル変数として設定願います。

１－７　コマンド
a）リードライト

I/O、メモリのリードライトを行います。数値設定は全て左詰めです。

読み込みたい I/O、メモリの先頭アドレスを設定



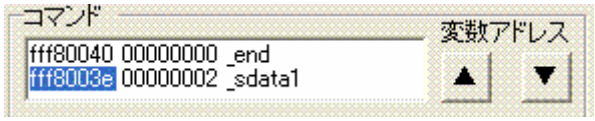
バイト（2 桁)、ワード(4 桁)、ロングワード（8 桁）の選択　書き込み

書き込むデータを設定　最大 8 桁　0～FFFFFFFF

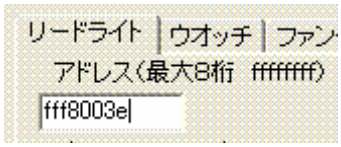
SH や H8 はメモリマップド I/O なので、基本的にアクセスは I/O もメモリも同じです。

例として、sdata1 のデータを読んでみます。

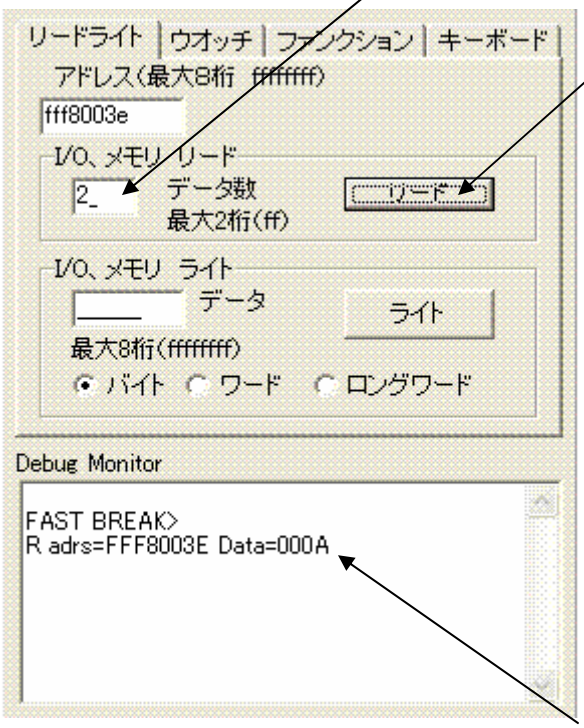
①変数アドレスで sdata1 のアドレスをコピーします。



②I/O、メモリリードライトのアドレスにペーストします。

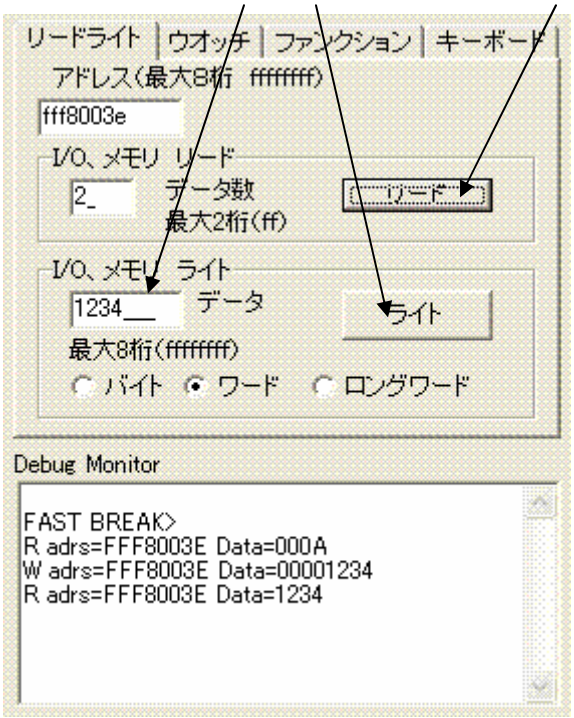


③データサイズは 2 バイトなので、データ数に 2 を設定し、「リード」すると



DebugMonitor にアドレス FFF8003E=sdata1 のデータ AH＝10D が表示されました。(16 進数表示では A、10 進数表示では 10 の意味)

④今度は sdata1 に 1234H を書いてみます。読んでみると確かに 1234 が書き込まれています。

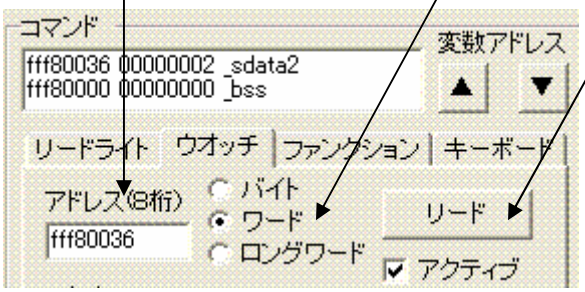


b）ウオッチ

ウオッチコマンドは登録したアドレスをブレークイベントが発生するたびに自動的に読み込み、比較する機能です。

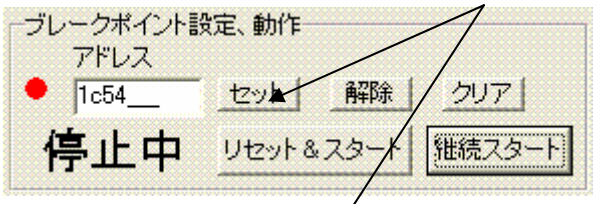
1）自動読込機能

読み込むデータのアドレス　　データサイズ　　読込（これとは別にブレークが発生するたびに自動的に読み込みます）

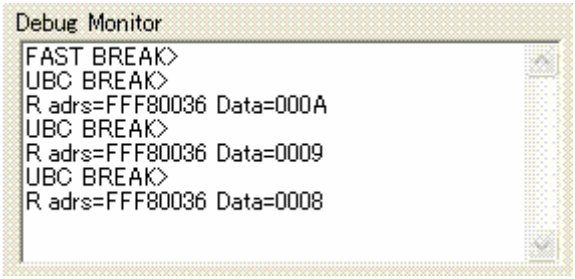


自動読込を有効にします

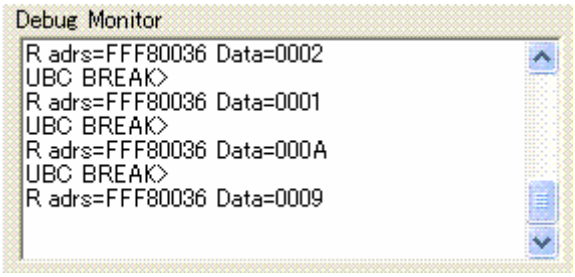
例として sample6 で、ブレークポイントを 1c54 にセットして「継続スタート」をクリックする度に、



読み込むデータのアドレス sdata2＝fff80036 が A、9、8 と減少していくことが確認できます。

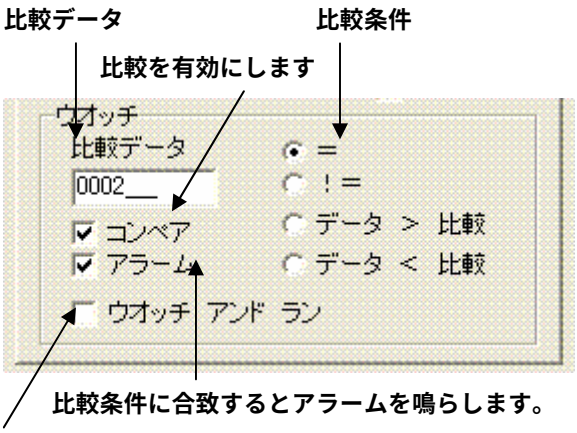


1 まで減少すると、再び 10D(AH)を読み込んで繰り返すのが目視できます。



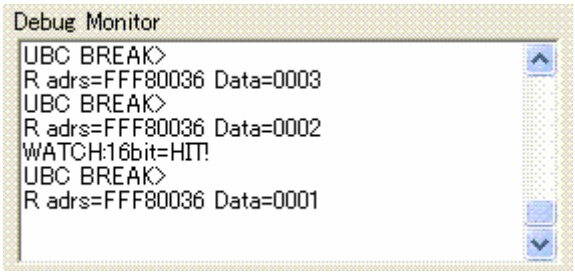
先のリードライトコマンドと比べるとウオッチ機能は 1 回設定すると毎回、ブレーク発生のたびに自動的に読み込むところが異なります。

２）比較機能



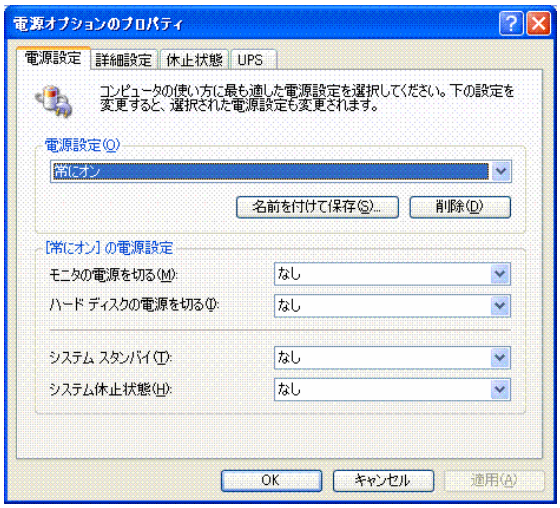
比較が終了すると自動的に動作を再開します。連続動作、ブレークイベント監視ができます。

例として１）の条件(sdata2→１～A まで変化する）で、数値 0002 でイコールという設定にします。「継続スタート」の度に Data は変化していきますが、DebugMonitor の中のように Data=0002 のときに WATCH:16bit=HIT!と表示され、比較データと sdata2 が 0002 で一致した記録が残ります。また、アラームが鳴ります。



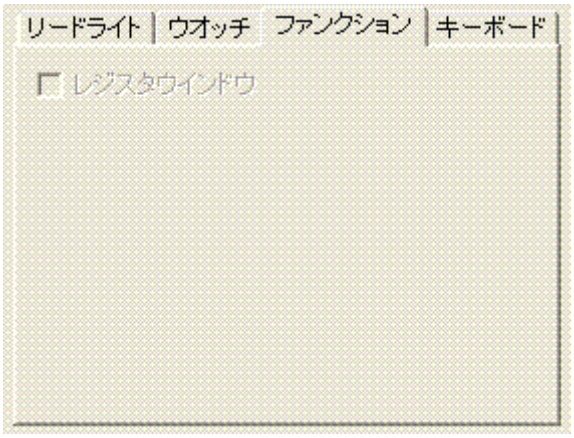
ウオッチ　アンド　ラン機能：通常、ブレークイベントで UBC　BREAK>が表示されると、動作の再開は「継続スタート」の人手によるクリックが必要です。「ウオッチ　アンド　ラン」をアクティブにしておくと「継続スタート」なしで、比較終了後、動作が再開します。無人で連続して比較する場合などに使用します。変数、ポートのデータ、Ａ／Ｄ等アナログ量の比較等、あらゆる事象の解析に使用できます。また、頻度の少ない現象も確実に捉えることができます。

コンピュータを連続で動かすときは、常にフォース®デバッカの画面が見えている状態にしてください。
①コントロールパネル→画面→スクリーンセイバー→なし　にします。
②コントロールパネル→電源オプションのプロパティ→常にオン　にします。　WindowsXP の場合



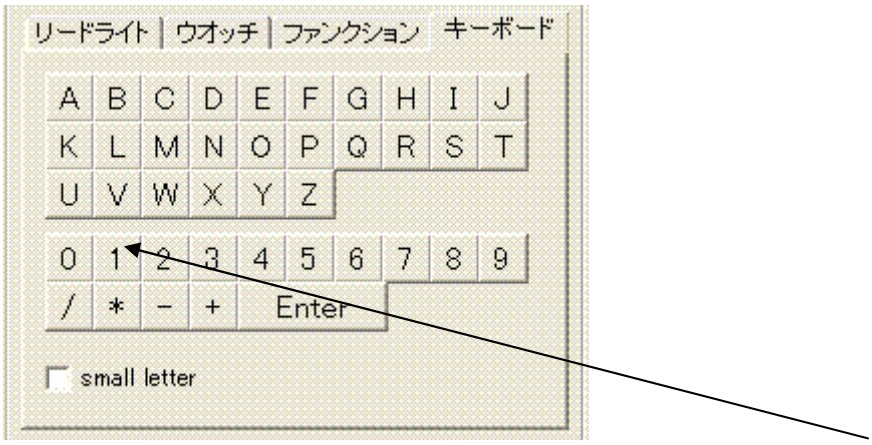
c）ファンクション

現在ファンクションはなにもサポートされていません。

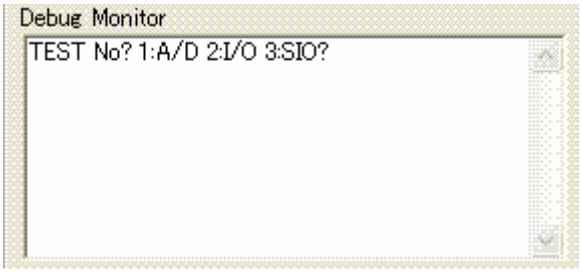


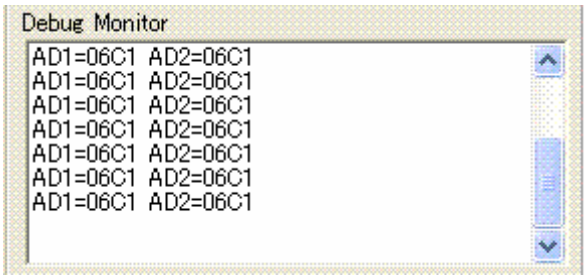
d）キーボード

キーボードはアプリケーションによって、パソコンから CPU ボードへ出力して動作を見たい場合などに使用します。



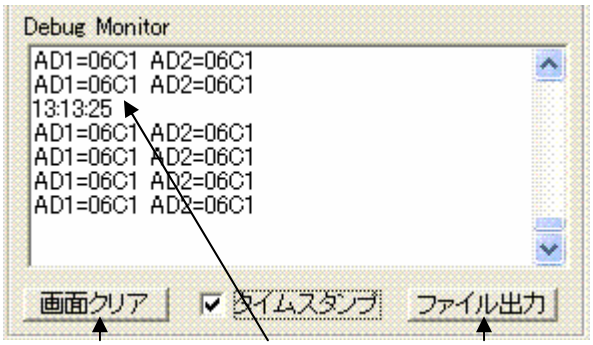
例えば sample2 では動作選択に 1,2,3 のいずれかの入力をまっています。例えば「1」をマウスでクリックするとパソコンから CPU ボードに「１」が出力されます。それを受信した CPU ボードは A/D コンバータデータを連続して出力するプログラムが動作します。





このような無限ループプログラムの初期化は「リセット＆スタート」で行います。

１－８　タイムスタンプ、デバック状況ファイル出力



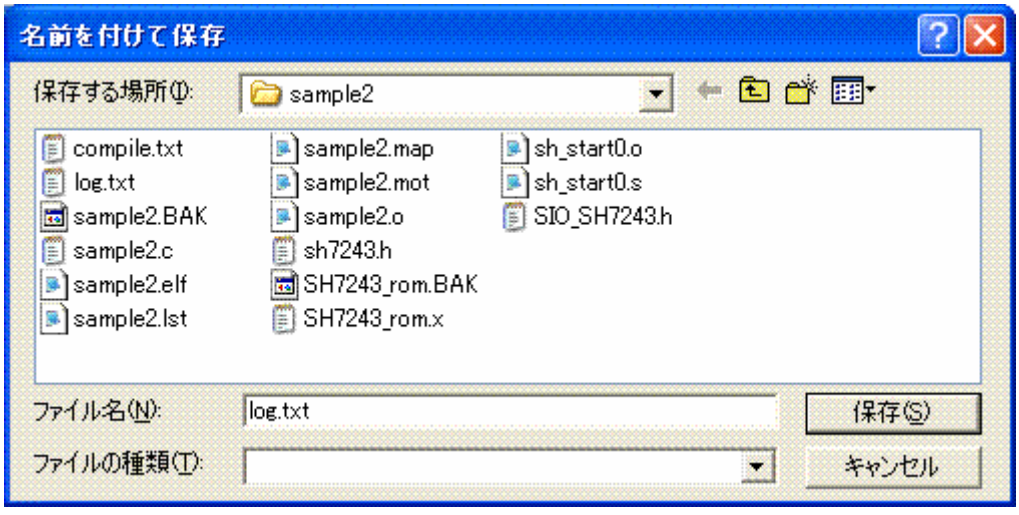
画面クリア　　タイムスタンプ　　デバッグモニタ画面ファイル出力

画面クリア　　：DebugMonitor 画面をクリアします。

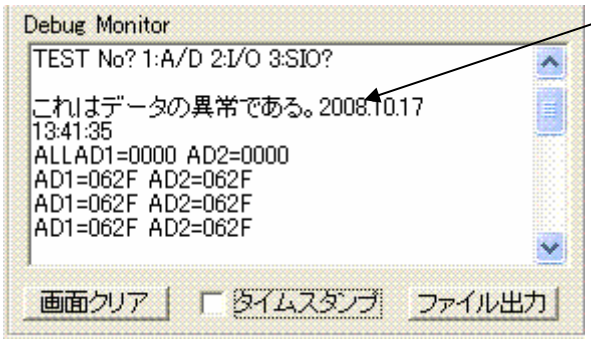
タイムスタンプ　　：約 1 分ごとに DebugMonitor 画面に現在の時刻を入れます。

ファイル出力　　：DebugMonitor 画面の内容をファイルにセーブします。

例えば、log.txt という名前でセーブできます。（お好きな名前、拡張子が付けられます）



また、この窓は書き込みも可能で、デバック中のコメント（下例「これはデータの異常である。2008.10.17」）をユーザーが書き加えることが可能です。



このログファイルにはデバックで行ったこと、時刻（タイムスタンプオン時)、ウオッチ比較のヒットあり、なし、コメント等すべてが記録されています。通常のアスキーファイルですので、プリンターで印刷したり、エディタで検索等掛けられます。

プログラム作成で最後まで残りやすい、頻度の少ない、稀な現象の強力な原因究明ツールになるよう考案されています。

〒350-1213　埼玉県日高市高萩１１４１－１

TEL　042（985）6982

FAX　042（985）6720

Homepage：http://beriver.co.jp　　e-mail：support@beriver.co.jp

有限会社ビーリバーエレクトロニクス　