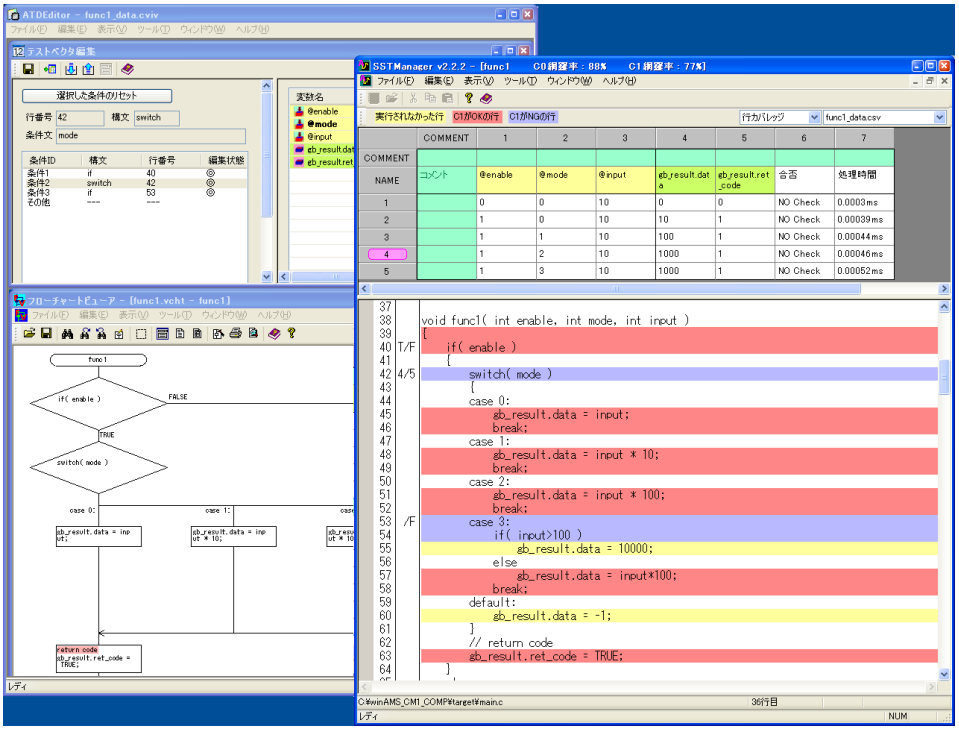
Solution House GAIO TECHNOLOGY



マクロ（シミュレータコマンド）の使い方

2014 年 11 月 04 日
第 1.4.0 版

目次

目次 1

はじめに 3

マイコンシミュレータのデバッグ機能 3

スタートアップコマンドファイル 5

シミュレータコマンド(マクロ)例 6

I/O 制御等で外的要因待ちをしている場合(その1) 6

I/O 制御等で外的要因待ちをしている場合(その2－1) 8

I/O 制御等で外的要因待ちをしている場合(その2－2) 9

Auto 変数の変遷をテスト結果 CSV で確認する場合 11

Auto 変数の値を変更する場合 12

レジスタ値をテスト結果 CSV で確認する場合 13

プログラム中で書き換えられるグローバル変数の値をテスト CSV で指定した値に変更する(戻す)場合 14

特定範囲のメモリ内容を初期化する場合 15

特定範囲のメモリ内容を、特定範囲のメモリにコピーする場合 15

その他の良く使うシミュレータコマンドについて 16

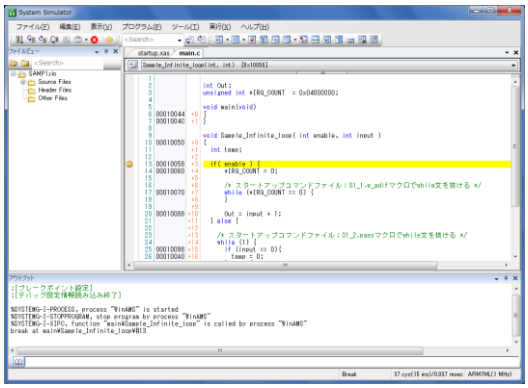
シミュレータコマンドの使い方

はじめに

カバレッジマスターwinAMS は、マイコンシミュレータの機能を利用してテストを実行します。マイコンシミュレータにはデバッグ機能が搭載されており、GUIから変数の値を参照したり、書き換えたりすることが出来ます。マイコンシミュレータの機能を予めファイルに記述して、デバッグ操作を自動化することが可能です。

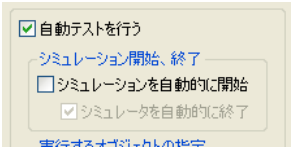
本説明書は、実際に運用で使用するケースが多いシミュレータコマンド(マクロ)についてご説明します。

本説明書は、カバレッジマスターwinAMS(V3.6.2 以降)で使用するマイコンシミュレータ(XAIL V3.1.0 以降)を元にご説明します。

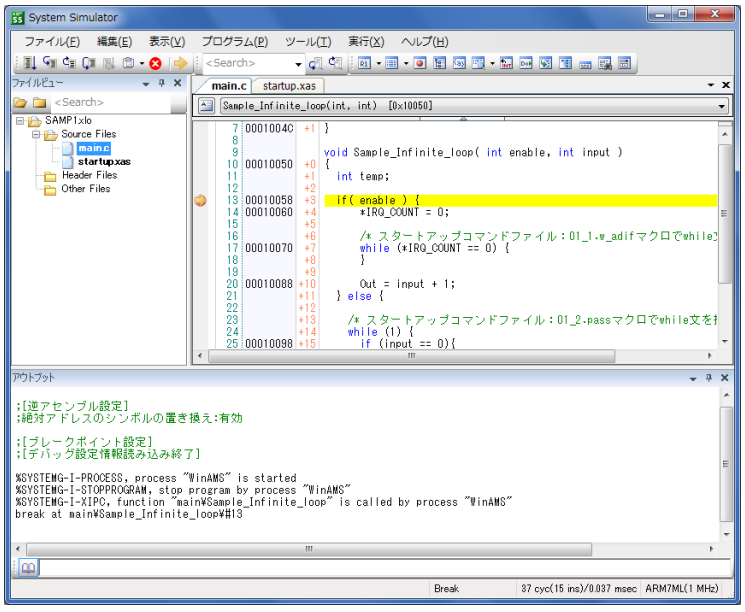


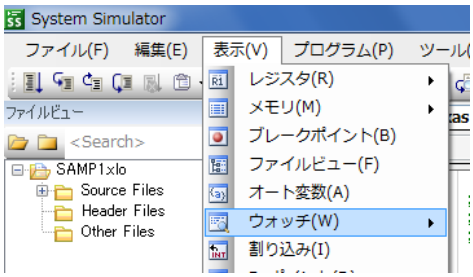
マイコンシミュレータのデバッグ機能

シミュレータコマンドの説明の前に、マイコンシミュレータのデバッグ機能について説明します。

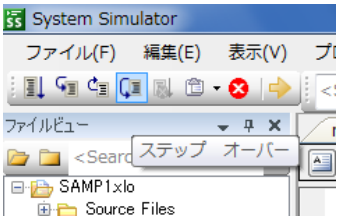


自動実行モードを外すと、テスト実行時にマイコンシミュレータが起動して、デバッガ機能が使用できます。





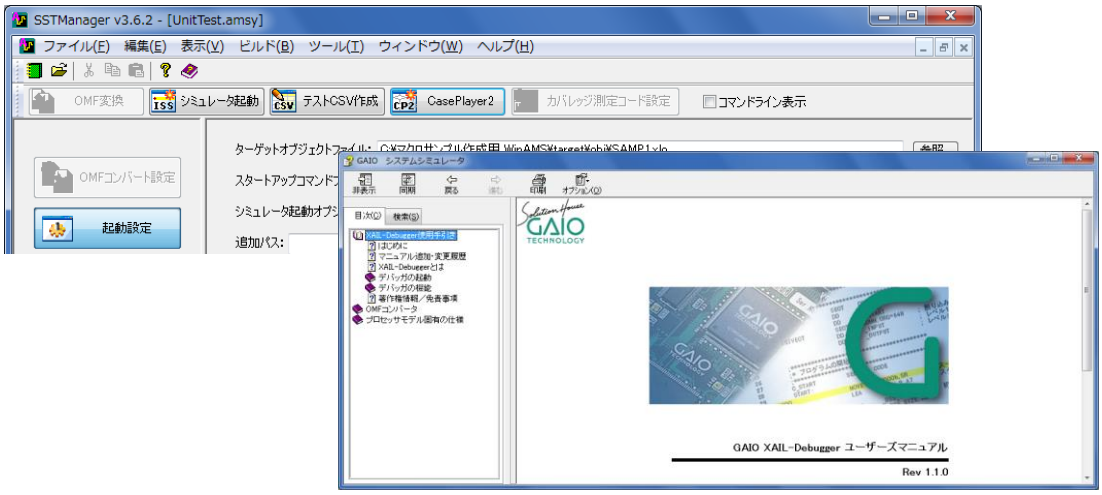
ウオッチウインドウを表示



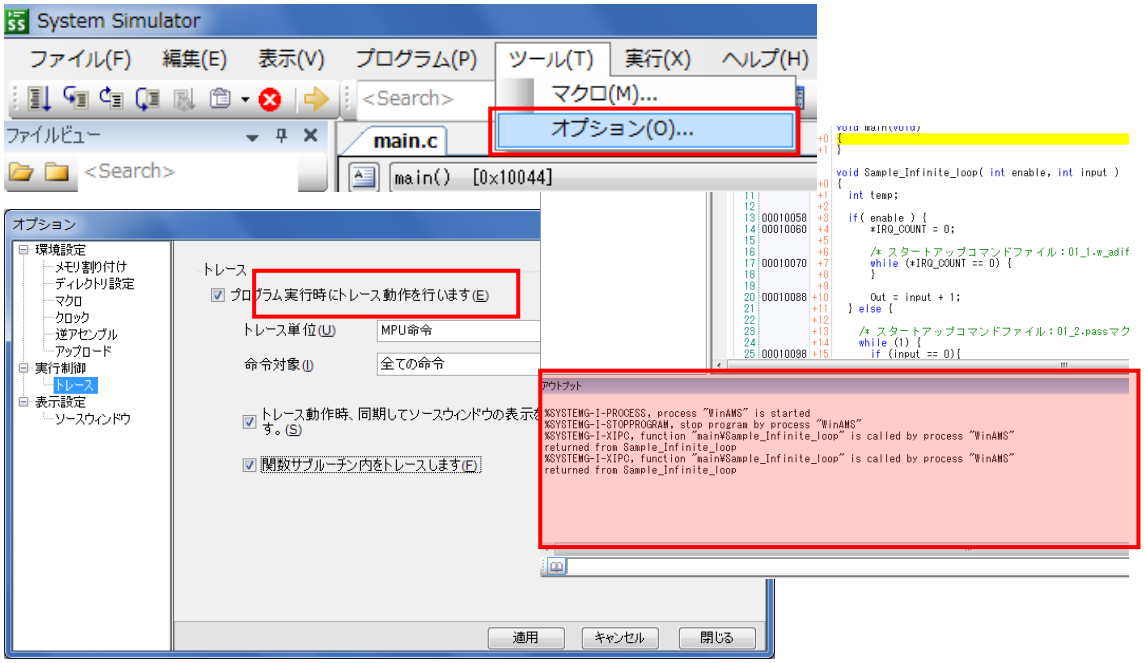
ステップ実行（ステップオーバー）ボタン

- ソースコードデバッグ機能を使用して、変数の値を書き換えたり、PC(プログラムカウンタ)を変更したりできます。
  1. 自動実行モードを外す:「シミュレーションを自動的に開始」をオフにする
  2. SystemSumulator が起動したら「ファイルビュー」からソースを表示
  3. 対象関数にブレークポイントを置く
  4. ウオッチウインドウなどを利用して、様々なデバッグ作業を行います

  デバッガの各機能詳細については、下記からご確認下さい。
  [ヘルプ]→[シミュレータマニュアル]→[目次]→[XAIL-Debugger 使用手引き]



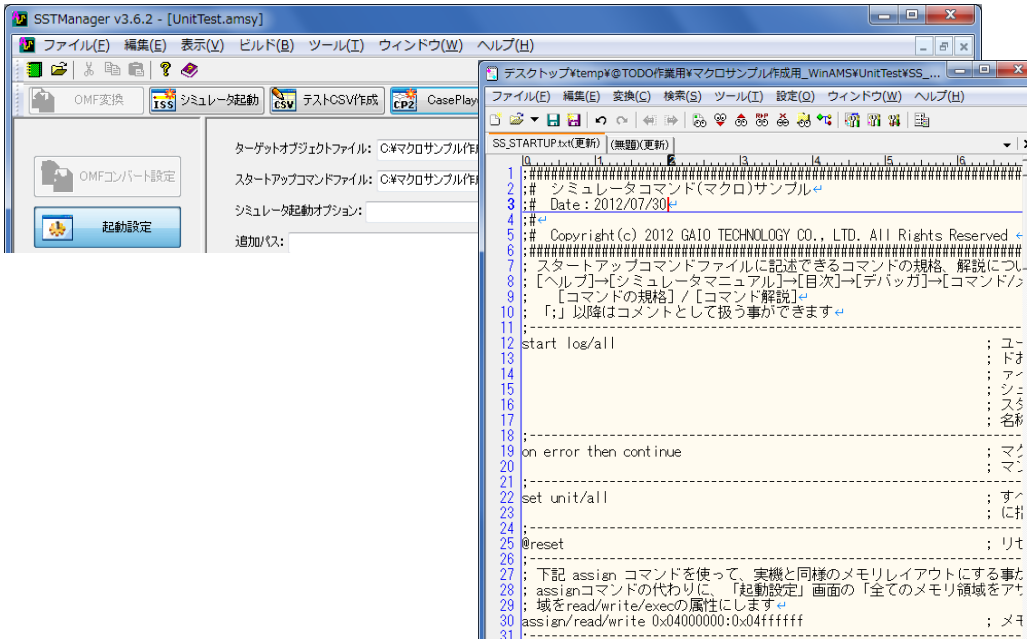
-トレース設定をする事で、MPU デバッグ情報欄にデバック状況を表示する事ができます。
  ※「アウトプット」ビューに表示されるデバック情報(実行ログ)はカバレッジマスターのプロジェクトフォルダに
    ログファイル(system.log)としても出力されます。
  1. 「ツール」メニューから「オプション」を開く
  2. トレース:「プログラム実行時に～」をオンにする
  3. その他の設定を任意で設定する
  4. ウオッチウインドウなどを利用して、様々なデバッグ作業を行います



スタートアップコマンドファイル

マイコンシミュレータで実行するシミュレータコマンドは、SSTManger で指定できるスタートアップコマンドファイルに記述する事で、自動化できます。

スタートアップコマンドファイルに記述できるコマンドの規格、解説については、下記からご確認下さい。
[ヘルプ]→[シミュレータマニュアル]→[目次]→[XAIL-Debugger 使用手引き]→[デバッガの機能]
→[コマンド/メッセージ]→[デバッガカーネルのコマンド/メッセージ]→[SystemG]→[デバッガコマンド一覧]
→[コマンドの規格] / [コマンド解説]
※「;」(セミコロン)以降(右側)の記述はコメントとして扱う事ができます



シミュレータコマンド(マクロ)例

I/O 制御等で外的要因待ちをしている場合(その1)

関数の処理に、I/O 制御でハードウェアの信号待ちを行っている while()ループがあり、そのループを抜けるためには、関数実行(テスト)中に i/o ポート(変数)の値を変化させる必要があります。そのような場合のマクロについて説明します。

■サンプルソース(main.c)

```
  unsigned int *IRQ_COUNT  = 0x04000000;

 1:void Sample_Infinite_loop( int enable, int input )
 2:{
 3:  int temp;
 4:
 5:  if( enable ) {
 6:      *IRQ_COUNT = 0;
 7:
 8:      /* スタートアップコマンドファイル:01_1.w_adif マクロで while 文を抜ける */
 9:      while (*IRQ_COUNT == 0) {
10:      }
11:
12:      Out = input + 1;
13:  } else {
14:      ..........
15:  }
16:}
```

■サンプルマクロ

説明:while 文の条件になっている変数の値を変更する

```
define/address/global ADR_ADIF = 0x04000000  ; ADR_ADIF はマクロで使用する任意の変数名
define/Global COUNT = 0                      ; COUNT はマクロで使用する任意の変数名

macro w_adif_V310                            ; マクロ名は任意で設定
  define/Global COUNT = COUNT + 1
  IF COUNT < 2 THEN goto func_end

  ; 2 回目にマクロが実行されると値のセットを行う
  ; 必要が無ければ、回数カウントの処理は入れないでも良いです
  store ADR_ADIF = 1
  define/Global COUNT = 0

  func_end:                                  ; ラベル名は任意で設定
mend

; IRQ_COUNT(0x04000000)が読み込まれた時にマクロ(w_adif)を実行する
set do/read=0x04000000 w_adif_V310
```

■サンプルマクロ(XAIL V3.2.0 以降)
説明:while 文の条件になっている変数の値を変更する

```
define/address/global ADR_ADIF = 0x04000000  ; ADR_ADIF はマクロで使用する任意の変数名
define/Global COUNT = 0                      ; COUNT はマクロで使用する任意の変数名

macro w_adif_V320                            ; マクロ名は任意で設定
  define/Global COUNT = COUNT + 1

  ; IF ～ THEN ～ ELSE[IF] ～ ENDIF 構文 は、XAIL V3.2.0 以降でサポート
  ; 条件式は()で括る必要があります
  IF (COUNT >= 2) THEN

    ; 2 回目にマクロが実行されると値のセットを行う
    ; 必要が無ければ、回数カウントの処理は入れないでも良いです
    store ADR_ADIF = 1
    define/Global COUNT = 0

  ELSE

  ENDIF
mend

; IRQ_COUNT(0x04000000)が読み込まれた時にマクロ(w_adif)を実行する
set do/read=0x04000000 w_adif_V320
```

I/O 制御等で外的要因待ちをしている場合(その2－1)

マシン語のリターンコードが展開されている場合
任意のコードが実行された後に、リターンコードが展開されているコードに PC を強制的に変更します。

■サンプルソース(main.c)

```
 1:void Sample_Infinite_loop( int enable, int input )
 2(+ 0):{
 3(+ 1):  int temp;
 4(+ 2):
 5(+ 3):  if( enable ) {
 6(+ 4):      ..........
 7(+ 5):} else {
 8(+ 6):
 9(+ 7):    /* スタートアップコマンドファイル:01_2.pass マクロで while 文を抜ける */
10(+ 8):    while (1) {
11(+ 9):      if (input == 0){
12(+10):        temp = 0;
13(+11):      } else {
14(+12):        temp = input * 2;
15(+13):      }
16(+14):      input = temp +1;
17(+15):    }
18(+16):  }
19(+17):}
逆アセンブルコード表示(19 行目)
        RTS
        NOP
```

■サンプルマクロ
説明:強制的に PC(プログラムカウンタ)を変更する

```
macro pass                                  ; マクロ名は任意で設定
  set reg pc = main¥#19                     ; 関数内行オフセットは使用出来ません
mend
; Sample_Infinite_loop の +14 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(pass)を実行する
set do/Aexec=main¥Sample_Infinite_loop¥#+14  pass
```

I/O 制御等で外的要因待ちをしている場合(その2－2)

マシン語のリターンコードが展開されていない場合

PC を強制的に変更しても、リターンコードがない為、関数を終了する事が出来ません。その為、テスト対象関数の実行(初期設定終了アドレスの実行)前に、戻り位置を記録し、テスト対象関数の実行後に、その戻り位置に戻すことで、テストを完了させます。

※マクロだけでは対応出来ませんので、ダミーソースの追加コンパイル・リンクおよび初期設定終了アドレスの変更が必要になります。(本処理対応前の初期設定終了アドレスは main と仮定して説明します。)

■サンプルソース(main.c)

```
 1:void Sample_Infinite_loop( int enable, int input )
 2(+ 0):{
 3(+ 1):  int temp;
 4(+ 2):
 5(+ 3):  if( enable ) {
 6(+ 4):      ..........
 7(+ 5):} else {
 8(+ 6):
 9(+ 7):    /* スタートアップコマンドファイル:01_2.pass マクロで while 文を抜ける */
10(+ 8):    while (1) {
11(+ 9):      if (input == 0){
12(+10):        temp = 0;
13(+11):      } else {
14(+12):        temp = input * 2;
15(+13):      }
16(+14):      input = temp +1;
17(+15):    }
18(+16):  }
19(+17):}
```

■ダミーソース(dummy.c)

```
 1:#include <setjmp.h>
 2:
 3:void dummy_setjmp(void);
 4:void dummy_main(void);
 5:void dummy_longjmp(void);
 6:
 7:int dummy;
 8:jmp_buf env;
 9:
10:void dummy_setjmp(void)
11(+ 0):{
12(+ 1):
13(+ 2):  dummy = 1;        // main 関数実行時に PC を強制変更
14(+ 3):
15(+ 4):  setjmp(env);      // 戻り位置を記録
16(+ 5):
17(+ 6):  dummy_main();     // 初期設定終了アドレスにする
18(+ 7):
19(+ 8):}
20:
```

■ダミーソース(dummy.c 続き)

```
21:void dummy_main(void)
22(+ 0):{
23(+ 1):
24(+ 2):  dummy = 1;
25(+ 3):
26(+ 4):}
27:
28:void dummy_longjmp(void)
29(+ 0):{
30(+ 1):
31(+ 2):  dummy = 1;
32(+ 3):
33(+ 4):  longjmp(env, 1); // setjmp が実行された位置に戻る
34(+ 5):
35(+ 6):}
```

■サンプルマクロ

説明:初期設定終了アドレスの実行前の位置を記録して、テスト対象関数実行後に、その位置に戻します。

```
macro setjmp                                    ; マクロ名は任意で設定
  set reg pc = dummy¥#13                        ; 関数内行オフセットは使用出来ません
                                                ; 初期設定終了アドレスを、「dummy_main」
                                                ; に変更する必要があります。
mend
; main.c / 関数:main の +0 行目に展開されるマシンコードの 1 バイト目が実行される前に
; マクロ(setjmp)を実行する
set do/Bexec=main¥main¥#+0  setjmp

macro longjmp                                   ; マクロ名は任意で設定
  set reg pc = dummy¥#31                        ; 関数内行オフセットは使用出来ません
mend
; main.c / 関数:Sample_Infinite_loop の +14 行目に展開されるマシンコードの 1 バイト目が実
; 行された時にマクロ(longjmp)を実行する
set do/Aexec=main¥Sample_Infinite_loop¥#+14  longjmp
```

Auto 変数の変遷をテスト結果 CSV で確認する場合

Auto 変数の内容をスタブファイルに定義したグローバル変数に保存して、その内容をテスト結果 CSV ファイルで確認する場合のマクロについて説明します。

■サンプルソース(main.c)

```
 1: typedef struct {
 2:  unsigned char value_a;
 3:  unsigned int  value_b;
 4:} InfoType_A;
 5:
 6:/* Auto 変数退避用変数(運用時にはスタブファイルに追加) */
 7:static char Macro_Symbol;
 8:static char Macro_Symbol2;
 9:
10:void Sample_AutoVariables_Evacuation(void)
11(+ 0):{
12(+ 1):  InfoType_A InfoPtr[5];
13(+ 2):
14(+ 3):  InfoPtr[0].value_a = 1;
15(+ 4):  /* スタートアップコマンドファイル:02_1.mymacro マクロで Auto 変数の値を退避 */
16(+ 5):  InfoPtr[0].value_b = 1;
17(+ 6):
18(+ 7):  InfoPtr[0].value_a = 2;
19(+ 8):  /* スタートアップコマンドファイル:02_2.mymacro2 マクロで Auto 変数の値を退避 */
20(+ 9):  InfoPtr[0].value_b = 2;
21(+10):
22(+11):}
```

■サンプルマクロ

説明:Auto 変数の値をグローバル変数に退避する

※マイコンに依っては、Auto 変数をそのまま利用できない場合もあります。

```
macro mymacro                                                    ; マクロ名は任意で設定
  ; プログラムで定義されている変数(Macro_Symbol)に Auto 変数(InfoPtr[0].value_a)の値を設定する
  STORE/Synchronize Macro_Symbol= InfoPtr[0].value_a
mend
; Sample_AutoVariables_Evacuation の +5 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(mymacro)を実行する
set do/Aexec=main¥Sample_AutoVariables_Evacuation¥#+5 mymacro

macro mymacro2                                                   ; マクロ名は任意で設定
  ; プログラムで定義されている変数(Macro_Symbol2)に Auto 変数(InfoPtr[0].value_a)の値を設定する
  STORE/Synchronize Macro_Symbol2= InfoPtr[0].value_a
mend
; Sample_AutoVariables_Evacuation の +9 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(mymacro2)を実行する
set do/Aexec=main¥Sample_AutoVariables_Evacuation¥#+9 mymacro2
```

Auto 変数の値を変更する場合

Auto 変数にスタブファイルで定義したグローバル変数の値を設定する場合のマクロについて説明します。
※グローバル変数の値はテスト CSV で指定します。

■サンプルソース(main.c)

```
 1:typedef struct {
 2:   unsigned char value_a;
 3:   unsigned int  value_b;
 4:} InfoType_A;
 5:
 6:/* Auto 変数設定用変数(運用時にはスタブファイルに追加) */
 7:static char Auto_Symbol;
 8:
 9:int Sample_AutoVariables_Set(void)
10(+ 0):{
11(+ 1):   InfoType_A InfoPtr[5];
12(+ 2):   int ret;
13(+ 3):
14(+ 4):   InfoPtr[0].value_a = 1;
15(+ 5):   /* スタートアップコマンドファイル:05_1.Set_AutoVariables マクロで Auto 変数の値を変更 */
16(+ 6):   InfoPtr[0].value_b = 1;
17(+ 7):
18(+ 8):   if (InfoPtr[0].value_a ==  1){
19(+ 9):     ret = -1;
20(+10):   } else {
21(+11):     ret = InfoPtr[0].value_a;
22(+12):   }
23(+13):}
```

■サンプルマクロ
説明:グローバル変数の値を Auto 変数に設定する
※マイコンに依っては、Auto 変数をそのまま利用できない場合もあります。

```
macro Set_AutoVariables                                          ; マクロ名は任意で設定
  ; プログラムで定義されている変数(Auto_Symbol)の値を Auto 変数(InfoPtr[0].value_a)に設定する
  STORE/Synchronize InfoPtr[0].value_a = Auto_Symbol
mend
;関数:Sample_AutoVariables_Set の +6 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(Set_AutoVariables)を実行する
set do/Aexec=main¥Sample_AutoVariables_Set¥#+6 Set_AutoVariables
```

レジスタ値をテスト結果 CSV で確認する場合

レジスタの内容をスタブファイルに定義したグローバル変数に保存して、その内容をテスト結果 CSV ファイルで確認する場合のマクロについて説明します。

■サンプルソース(main.c)

```
 1:static char Before_Register;
 2:static char After_Register;
 3:
 4:/* Register 退避用変数(運用時にはスタブファイルに追加) */
 5:void Sample_Register_Evacuation(void)
 6(+ 0):{
 7(+ 1):   /* スタートアップコマンドファイル:03_1.b_reg マクロでレジスタの値を退避 */
 8(+ 2):   InfoType_A InfoPtr[5];
 9(+ 3):
10(+ 4):   InfoPtr[0].value_a = 1;
11(+ 5):   InfoPtr[0].value_b = 1;
12(+ 6):
13(+ 7):   /* スタートアップコマンドファイル:03_2.a_reg マクロでレジスタの値を退避 */
14(+ 8):}
```

■サンプルマクロ

説明:レジスタの値をグローバル変数に退避する

```
macro b_reg                                                    ; マクロ名は任意で設定
  ; プログラムで定義されている変数(Before_Register)に R1 レジスタの値を設定する
  STORE/Synchronize Before_Register= %R1
mend
; Sample_Register_Evacuation の +0 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(b_reg)を実行する
set do/Aexec=main¥Sample_Register_Evacuation¥#+0 b_reg

macro a_reg                                                    ; マクロ名は任意で設定
  ; プログラムで定義されている変数(After_Register)に R1 レジスタの値を設定する
  STORE/Synchronize After_Register= %R1
mend
; Sample_Register_Evacuation の +8 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(a_reg)を実行する
set do/Aexec=main¥Sample_Register_Evacuation¥#+8 a_reg
```

プログラム中で書き換えられるグローバル変数の値をテストCSVで指定した値に変更する(戻す)場合

テストCSVで値を設定しても、プログラムで直ぐに値が変更される場合に、テストCSVの値(グローバル変数)を退避して、プログラムで該当変数が変更された直後に、テストCSVの値に戻す場合のマクロについて説明します。

■サンプルソース(main.c)

```
 1:int GlobalA;
 2:int Sample_Global_Set( void )
 3(+ 0):{
 4(+ 1):   int rtn = 0;
 5(+ 2):
 6:       /* スタートアップコマンドファイル:04_1.Evacuation_Variables マクロで GlobalA を退避 */
 7(+ 4):   GlobalA = 0 ;              // グローバル変数クリア
 8(+ 5):
 9(+ 6):   /* スタートアップコマンドファイル:04_2.  Set_Variables マクロで GlobalA を設定 */
10(+ 7):   while ( GlobalA == 0 );   // グローバル変数がセットされるまで待つ
11(+ 8):
12(+ 9):   if ( GlobalA == 99 ){     // セットされたグローバル変数から戻り値を設定
13(+10):     rtn = 1 ;
14(+11):   } else {
15(+12):     rtn = 2 ;
16(+13):   }
17(+14):
18(+15):   return ( rtn ) ;
19(+16):}
```

■サンプルマクロ

```
説明:グローバル変数の値を退避して、その値を戻す
define/Global Evacuation = 0                    ; Evacuation はマクロで使用する任意の変数名
macro Evacuation_Variables                      ; マクロ名は任意で設定
  ; プログラムで定義されている変数(GlobalA)の値を Evacuation に設定する
  define/Global Evacuation = GlobalA
mend
; Sample_Global_Set の +4 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(Evacuation_Variables)を実行する
set do/Aexec=main¥Sample_Global_Set¥#+4 Evacuation_Variables

macro Set_Variables                             ; マクロ名は任意で設定
  ; Evacuation の値をプログラムで定義されている変数(GlobalA)に設定する
  store GlobalA = Evacuation
mend
; Sample_Global_Set の +7 行目に展開されるマシンコードの 1 バイト目が実行された時に
; マクロ(Set_Variables)を実行する
set do/Aexec=main¥Sample_Global_Set¥#+7 Set_Variables
```

特定範囲のメモリ内容を初期化する場合

特定範囲のメモリ内容を任意の値で初期化するマクロについて説明します。

■サンプルマクロ
説明：指定した範囲のメモリ内容に任意の値を設定する
```
FILL MEMORY 0x2000#0x1000 = 0xff                ; XAIL V3.0.1 以降
```

特定範囲のメモリ内容を、特定範囲のメモリにコピーする場合

特定範囲のメモリ内容を、特定範囲のメモリにコピーするマクロについて説明します。ブートローダーを使って、RAM 上で動作するプログラムを ROM から RAM へ展開するような場合などに利用します。

■サンプルマクロ
説明：指定した範囲のメモリ内容を指定した範囲のメモリにコピーする
```
COPY MEMORY 0x2000#0x3000 0xff2000              ; XAIL V3.0.1 以降
```

その他の良く使うシミュレータコマンドについて

ここまでの説明以外に良く使用するシミュレータコマンドについて説明します。

■サンプルマクロ

```
start log/all                          ; ユーザがコマンドウィンドウから実行したデバッガコマン
                                       ; ドおよびウィンドウに表示されるすべてのデータをログフ
                                       ; ァイルへ書き込みます
                                       ; シュミレーション実行後のログファイルは、カバレッジマ
                                       ; スターのテストプロジェクトフォルダに「systemg.log」の
                                       ; 名称で保存されます
on error then continue                 ; マクロ実行中にエラーが生じたときに GO コマンドと STEP コ
                                       ; マンドの次のデバッガコマンドを実行します
set unit/all                           ; すべてのユニットについて、シンボル情報を設定するよう
                                       ; に指定します
@reset                                 ; リセット状態にします（パワーオンリセット）

assign/read/write 0x04000000:0x04ffffff    ; メモリのアサインを行います

set reg pc = 0xffffffff                    ; プログラムカウンタを設定します

set mode source                            ; ソースウィンドウにソースファイルを表示します

set trace/subroutine=yes/display           ; プログラムの実行トレースを実施します
                                           ; ログファイルにも保存されます

; マクロまたはコマンドプロシジャ内から実行される各デバッガコマンドをシミュレータのコマンドウィンド
; ウに表示するようにします（SET VERIFY コマンドより後のコマンドから表示します）
; 作成したマクロが実行されているかどうか確認する時に使います
SET VERIFY
```

シミュレータコマンドの使い方

ガイオ・テクノロジー株式会社

■ユーザーサポートのご案内
http://www.gaio.co.jp/support/support_about.html

■使用方法に関する問い合わせ方法
ご質問は、ユーザーサポート窓口(user@gaio.co.jp)までご連絡下さい。
ユーザーサポート窓口への質問には、ユーザーIDが必要です。
お問い合わせの際に、ユーザーIDをお知らせください。
※保守契約がない場合は、いかなるサポートも提供致しません。