



Solution House
GAIO
TECHNOLOGY

Copyright (c) 2007 GAIO TECHNOLOGY CO., LTD. All rights reserved.

CoverageMaster winAMS
GAIO Unit Test Simulator for Embedded Software

Macro User Guide

Revised October 2014

Table of Contents

Table of Contents	2
Introduction	3
The MPU simulator debugger.....	3
Startup command file	7
Macro (simulator command) examples	8
I/O control or other external wait condition (1)	8
I/O control or other external wait condition (2)	9
Verify auto variable changes in the test results CSV file	10
Set auto variable values.....	11
Verify register value in the test results CSV file	12
Store and return global variable value	13
Initialize a specified memory region to a set value	14
Copy content of memory region to another memory region	14
Other common simulator commands	15

Using macros

Introduction

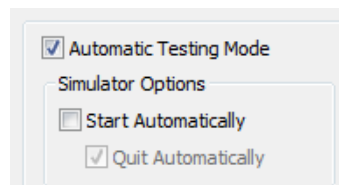
CoverageMaster winAMS uses a MPU simulator in order to perform testing. Using the built in debugger GUI, variable values may be referenced as well as rewritten. It is also possible to write macros (simulator commands) to a file in order to automate various tasks. This document contains examples of macros that may be written in order to perform several simulator operations.

This document is for use with the new System Simulator (XAIL v3.0 or newer) included with CoverageMaster winAMS v3.6.2 or newer.

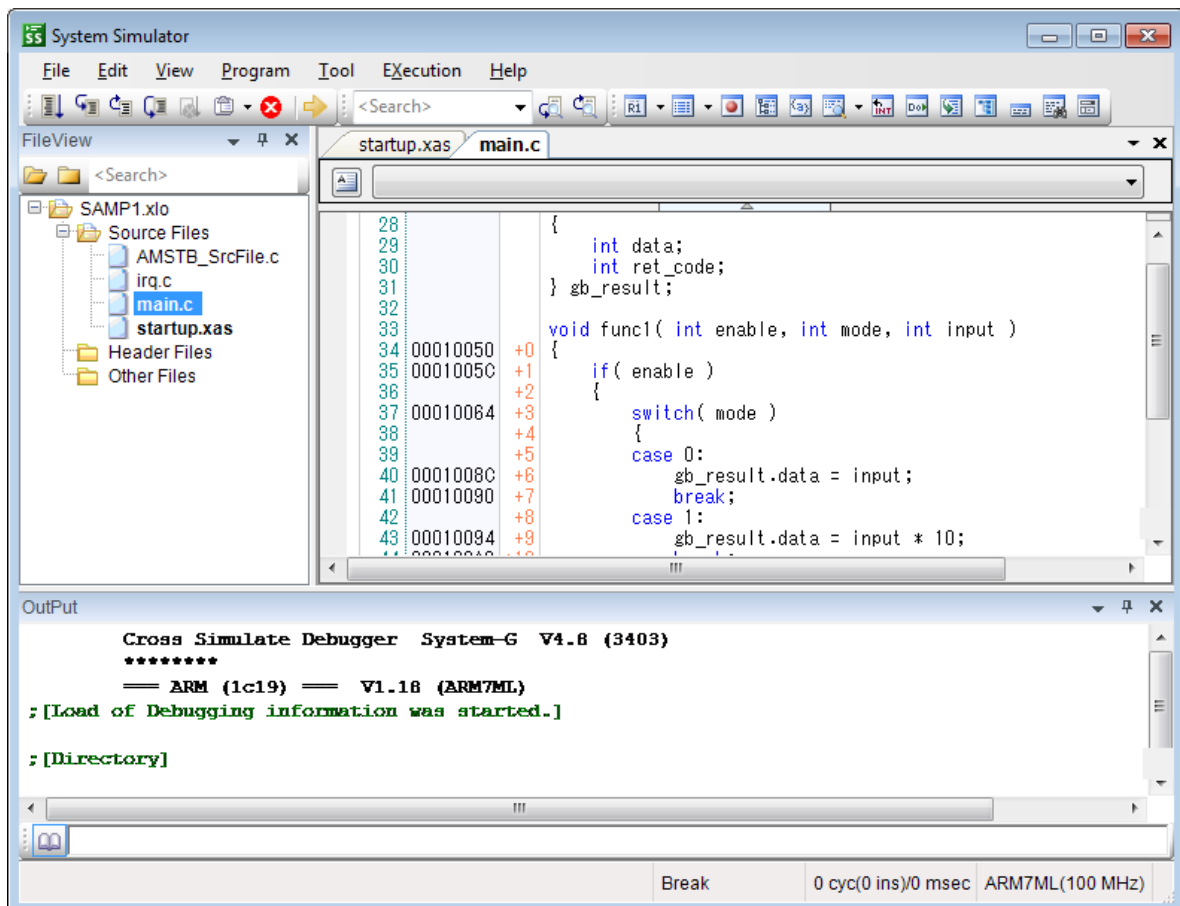
The MPU simulator debugger

An introduction of the MPU simulator debugger is described in this section prior to the macro examples.

From SSTManager's "Target Settings" screen, uncheck the "Start Automatically" setting.



After clicking the "Start Simulator" button, the MPU Simulator debugger GUI will be shown.

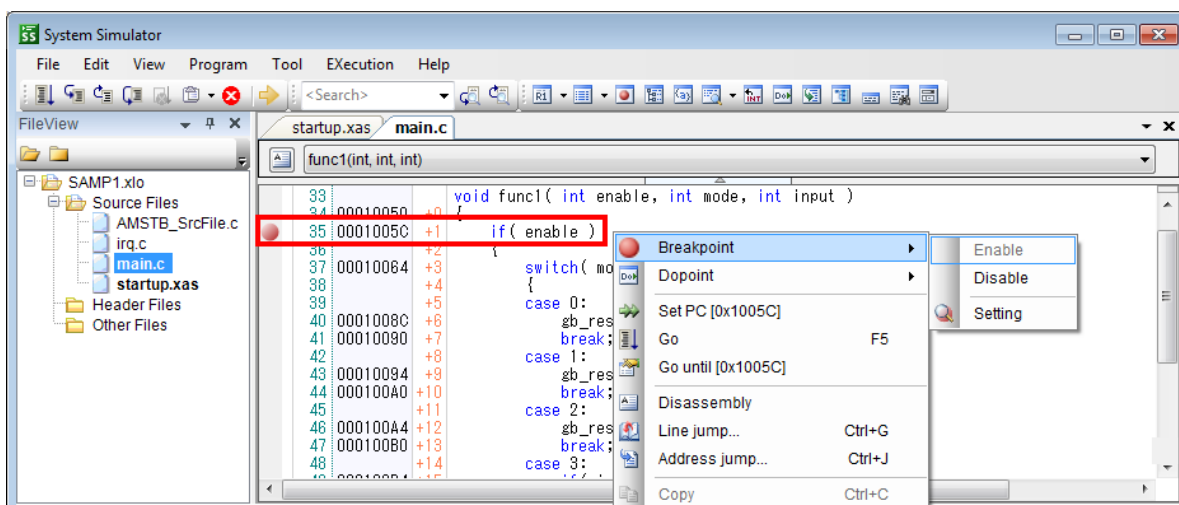


From the System Simulator a variety of debug related commands can be performed. Examples of some of the more common features are shown below.

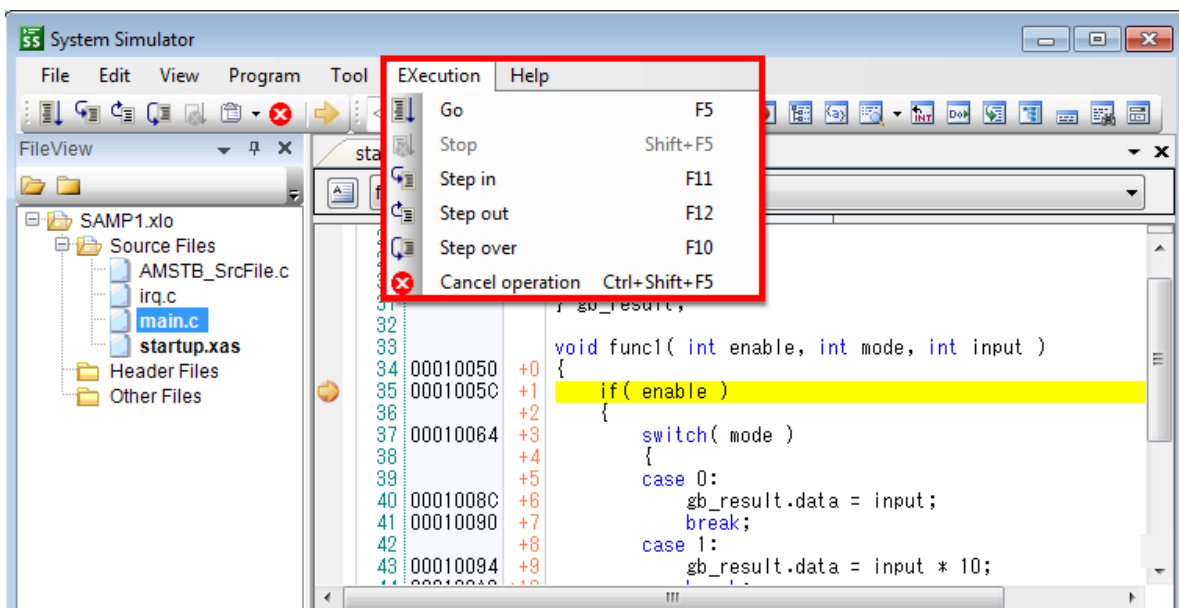
Breakpoints can be set to stop simulation at a specified point in the program.

To create a breakpoint:

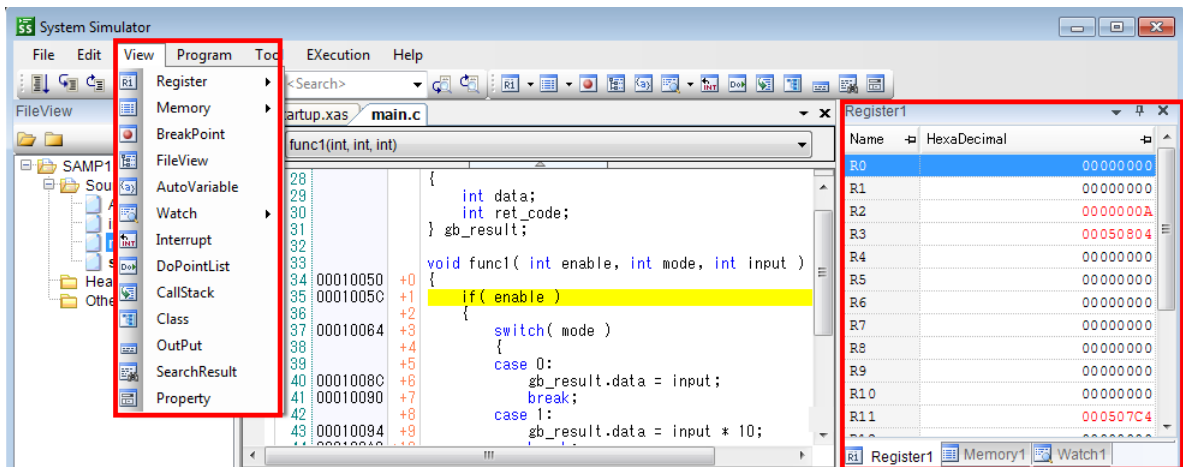
1. After starting the System Simulator, from the “View” menu, click “File View”, then double-click on a source file to show it in the source view
2. Right-click on a line of code that has an address allocated to it (address shown on the left-side), then select “Breakpoint” – “New Setting”
 - a. Note: the simulation will not stop at breakpoints set on the very first line with an address allocated to it (this is due to a limitation)
3. After setting a breakpoint, a red-dot will be shown on the farthest left-column of the code line
 - a. Alternatively, left-clicking on the red-dot will enable/disable the breakpoint setting
4. A list of all set breakpoints can be viewed from the “View” menu, “Breakpoint” selection



Starting the Simulation and **Step Execution** can be performed from the “EXecution” menu. The shortcut key displayed (F5, F10, F11, etc.) to the right of the option can be used for easy access.

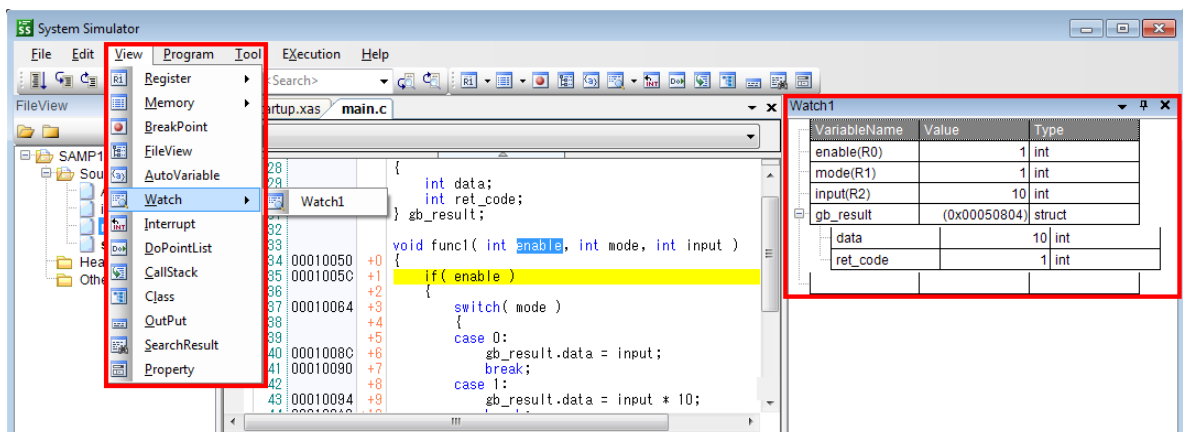


Additional debug related windows can be displayed from the “View” menu. These include a **Register** view, **Memory** view, **Watch** window, etc.



In order to use the **Watch** window:

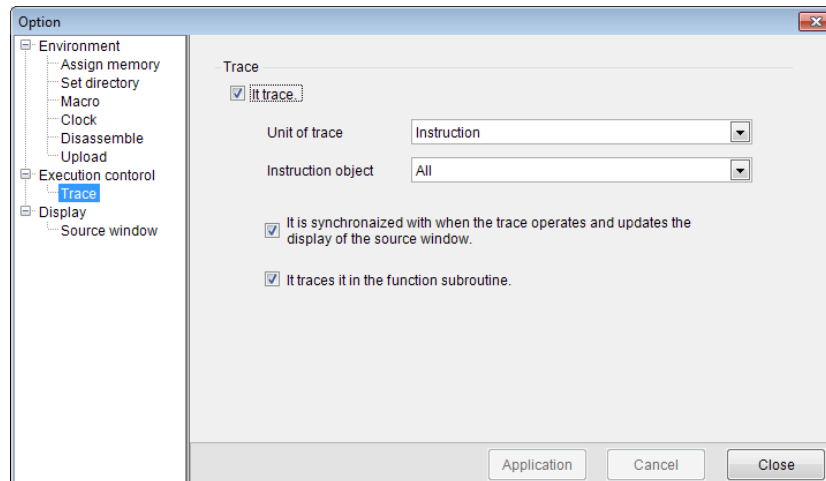
1. First, create a breakpoint within the function you wish to watch variable changes for, and run the simulator till that point
 - a. Note: items that are outside the current scope will be shown in grey
2. From the “View” menu, select “Watch”
3. Double-click on a variable, structure instance, or pointer in the code view to select it, then drag-and-drop the item into the Watch window
 - a. Right-click on the Watch window to display values in Hexadecimal or Decimal
 - b. Note: in some cases items cannot be added to the Watch window due to insufficient debug information in the object file



The **Trace** setting can be enabled to highlight the line of code being executed during the simulation and output trace information to the Output window. In addition, the trace output information will be output in the “systemg.log” file saved in the CoverageMaster project folder.

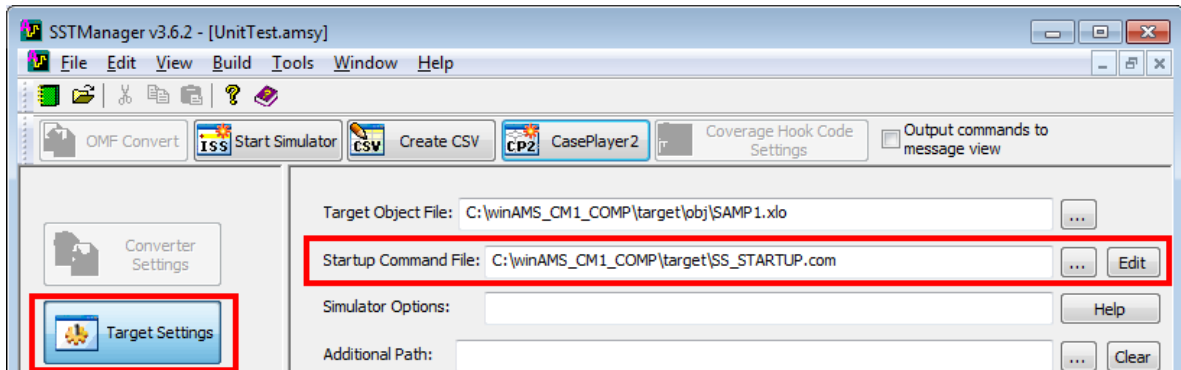
1. From the “Tool” menu, click “Option...”
2. Under the “Execution Control” section, select “Trace”
3. Enable all settings

Note: The trace settings takes additional time to output data so will slow down the simulation. It is recommended to turn off the trace settings for normal test operation and only enable the trace temporarily for debugging.



Startup command file

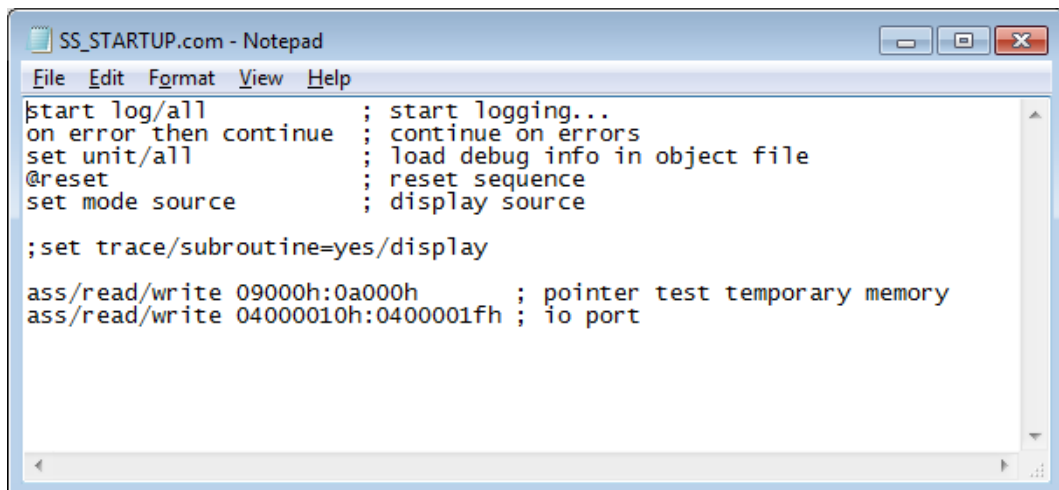
Macros (simulator commands) may be entered into the startup command file to run when the MPU simulator is started. The startup command file is set within SSTManager's Target Settings screen. Examples of macros to perform various operations are found later in this document.



Click the "Edit" button to the right of the startup command file to open the startup command file for editing.

A startup command file example is shown below.

Note: as semicolon ";" indicates a comment row within the startup command file and will not be executed.



```
File Edit Format View Help
|start log/all ; start logging...
|on error then continue ; continue on errors
|set unit/all ; load debug info in object file
|@reset ; reset sequence
|set mode source ; display source
|
|;set trace/subroutine=yes/display
|
|ass/read/write 09000h:0a000h ; pointer test temporary memory
|ass/read/write 04000010h:0400001fh ; io port
```

Macro (simulator command) examples

I/O control or other external wait condition (1)

Use this macro to exit an I/O control or other external signal wait related while() loop found within a test function.

■ Sample source code

```

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:     /* use the "w_adif" macro in sample shown below to exit the loop */
9:     while (*IRQ_COUNT == 0) {
10:    }
11:
12:     Out = input + 1;
13:   } else {
14:     ...
15:   }
16: }

```

■ Sample macro

For modifying a variable in order to exit a loop.

```

define/address/global ADR_ADIF = 0x04000000 ; ADR_ADIF is a var defined for macro use
define/Global COUNT = 0 ; COUNT is a var defined for macro use

```

```

macro w_adif ; macro definition
define/Global COUNT = COUNT + 1
IF COUNT < 2 THEN goto func_end

```

```

; the value is set on the 2nd time the macro is run
; if unnecessary the COUNT operation may be removed
store ADR_ADIF = 1
define/Global COUNT = 0

```

```

func_end: ; label name
mend

```

```

; run w_adif macro when IRQ_COUNT(0x04000000) is read
set do/read=0x04000000 w_adif

```


I/O control or other external wait condition (2)

■Sample source code

```

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):     /* use the "pass" macro in sample shown below to exit the loop */
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): }

```

■Sample macro

For modifying the PC (program counter) to pass over the loop.

```

macro pass          ; macro definition
  set reg pc = main¥#19
mend
; run pass macro after executing Sample_Infinite_loop's line +14
set do/Aexec=main¥Sample_Infinite_loop¥##+14 pass

```

Verify auto variable changes in the test results CSV file

Use this macro to store auto variable values in global variables (defined in a stub file) in order to display the auto variable values in the test results CSV file.

■ Sample source code

```

1: typedef struct {
2:   unsigned char value_a;
3:   unsigned int value_b;
4: } InfoType_A;
5:
6: /* vars for storing auto var values (add to stub file) */
7: static char Macro_Symbol;
8: static char Macro_Symbol2;
9:
10: void Sample_AutoVariables_store(void)
11 (+ 0): {
12 (+ 1): InfoType_A* InfoPtr;
13 (+ 2):
14 (+ 3): InfoPtr[0].value_a = 1;
15 (+ 4): /* store auto var value using "mymacro" macro in sample shown below */
16 (+ 5): InfoPtr[0].value_b = 1;
17 (+ 6):
18 (+ 7): InfoPtr[0].value_a = 2;
19 (+ 8): /* store auto var value using "mymacro2" macro in sample shown below */
20 (+ 9): InfoPtr[0].value_b = 2;
21 (+10):
22 (+10): }
```

■ Sample macro

For storing auto variable values into global variables.

Note: depending on the MPU, adjustments may need to be made.

macro mymacro ; macro definition

```

; store auto var value into static var defined in program
STORE/Synchronize Macro_Symbol= InfoPtr[0].value_a
mend
```

```

; run mymacro after executing Sample_AutoVariables_store's line +5
set do/Aexec=main¥Sample_AutoVariables_store¥##+5 mymacro
```

macro mymacro2 ; macro definition

```

; store auto var value into static var defined in program
STORE/Synchronize Macro_Symbol2= InfoPtr[0].value_a
mend
```

```

; run mymacro2 after executing Sample_AutoVariables_store's line +9
set do/Aexec=main¥Sample_AutoVariables_store¥##+9 mymacro2
```

Set auto variable values

Use this macro to set auto variable values using global variables (defined in a stub file).
 Note: values for the global variables are set in the test CSV file.

■ Sample source code

```

1: typedef struct {
2:   unsigned char value_a;
3:   unsigned int value_b;
4: } InfoType_A;
5:
6:   /* var for setting auto var values (add to stub file) */
7:   static char Auto_Symbol;
8:
9:   int Sample_AutoVariables_Set(void)
10 (+ 0): {
11 (+ 1):   InfoType_A* InfoPtr;
12 (+ 2):   int ret;
13 (+ 3):
14 (+ 4):   InfoPtr[0].value_a = 1;
15 (+ 5):   /* set auto var values using "Set_AutoVariables" macro in sample below */
16 (+ 6):   InfoPtr[0].value_b = 1;
17 (+ 7):
18 (+ 8):   if (InfoPtr[0].value_a == 1){
19 (+ 8):     ret = -1;
20 (+10): } else {
21 (+11):   ret = InfoPtr[0].value_a;
22 (+12): }
23 (+13): }
```

■ Sample macro

Set global var value to auto variable.

Note: depending on the MPU, adjustments may need to be made.

macro Set_AutoVariables ; macro definition

; store global var (Auto_Symbol) value into auto var (InfoPtr[0].value_a).

STORE/Synchronize InfoPtr[0].value_a = Auto_Symbol

mend

; run Set_AutoVariables macro after executing Sample_AutoVariables_Set's line +6

set do/Aexec=main¥Sample_AutoVariables_Set¥##+6 Set_AutoVariables

Verify register value in the test results CSV file

Use this macro to store register values in global variables (defined in a stub file) in order to display the register values in the test CSV file.

■ Sample source code

```

1:  /* vars for storing register values (add to stub file) */
2:  static char Before_Register;
3:  static char After_Register;
4:
5:      void Sample_Register_store(void)
6 (+ 0):  {
7 (+ 1):  /* store register value using "b_reg" macro in sample shown below */
8 (+ 2):  InfoType_A* InfoPtr;
9 (+ 3):
10 (+ 4): InfoPtr[0].value_a = 1;
11 (+ 5): InfoPtr[0].value_b = 1;
12 (+ 6):
13 (+ 7): /* store register value using "a_reg" macro in sample shown below */
14 (+ 8): }
```

■ Sample macro

For storing register values into global variables.

```

macro b_reg ; macro definition
    ; store R1 register value into Before_Register variable
    STORE/Synchronize Before_Register= %R1
mend
    ; run b_reg macro after executing Sample_Register_store's line +0
set do/Aexec=main¥Sample_Register_store¥##+0 b_reg

macro a_reg ; macro definition
    ; store R1 register value into After_Register variable
    STORE/Synchronize After_Register= %R1
mend
    ; run a_reg macro after executing Sample_Register_store's line +8
set do/Aexec=main¥Sample_Register_store¥##+8 a_reg
```

Store and return global variable value

Use this macro to store a global variable value to a temp variable, and then later return the stored value to the global variable. This may be useful for programs such as below that immediately modify a global variable value.

■ Sample source code

```

1:      int GlobalA;
2:      int Sample_Global_Set( void )
3 (+ 0): {
4 (+ 1):   int rtn = 0;
5 (+ 2):
6 (+ 3):   /* store global variable value using "Store_Variables" macro in sample shown below */
7 (+ 4):   GlobalA = 0 ; /* clear global variable */
8 (+ 5):
9 (+ 6):   /* set global variable value using "Set_Variables" macro in sample shown below */
10 (+ 7):  while ( GlobalA == 0 ); /* wait for global variable to be set */
11 (+ 8):
12 (+ 9):  if ( GlobalA == 99 ){ /* set return value based on global variable */
13 (+10):   rtn = 1 ;
14 (+11):  } else {
15 (+12):   rtn = 2 ;
16 (+13):  }
17 (+14):
18 (+15):  return ( rtn ) ;
19 (+16):  }
```

■ Sample macro

Store and return global variable value

```

define/Global StoreVar = 0                ; variable definition for macro use
macro Store_Variables                    ; macro definition
    ; store GlobalA value in StoreVar variable
    define/Global StoreVar = GlobalA
mend
    ; run Store_Variables macro after executing Sample_Global_Set's line +4
set do/Aexec=main¥Sample_Global_Set¥##+4 Store_Variables

macro Set_Variables                      ; macro definition
    ; set GlobalA to StoreVar value
    store GlobalA = StoreVar
mend
    ; run Set_Variables macro after executing Sample_Global_Set's line +7
set do/Aexec=main¥Sample_Global_Set¥##+7 Set_Variables
```

Initialize a specified memory region to a set value

Use this macro to initialize a specified memory region to a set value.

■ Sample Macro

initialize a specified memory region to a set value

```
FILL MEMORY 0x2000#0x1000 = 0xff ; XAIL V3.0.1 or later
```

Copy content of memory region to another memory region

Use this macro to copy the content of a specified memory region to another memory region. Use this macro for instances such as when using a boot loader to deploy a program from ROM to RAM.

■ Sample Macro

Copy content of memory region to another memory region

```
COPY MEMORY 0x2000#0x3000 0xff2000 ; XAIL V3.0.1 or later
```

Other common simulator commands

Other commonly used simulator commands.

■ Sample

start log/all

; Write data displayed in the simulator window and user entered commands in the
; command window to the log file. The log file named "systemg.log" will be saved
; in the test project folder.

on error then continue

; Run the next debugger command when an error is encountered during macro execution.

set unit/all

; Set the symbol information for all units

@reset

; Set to the device reset state (power-on reset)

assign/read/write 0x04000000:0x04ffffff

; assign memory with the indicated attribute

set reg pc = 0xffffffff

; set the program counter to the specified value

set mode source

; display the source file in the source window

set trace/subroutine=yes/display

; enable the program execution trace and save details to log file

SET VERIFY

; output debug commands within macros or command procedures to the simulator output window

; Note: commands after the SET VERIFY command will be output

; Note: this can be useful for verifying that macros have been set

GAIO TECHNOLOGY CO.,LTD.

3 Tennouzu First Tower 25F

2-2-4 Higashi-Shinagawa, Shinagawa-ku, Tokyo 140-0002 Japan

TEL: (03) 4455-4767

Email: info@gaio.co.jp

- * Company names and product names that appear in this presentation are trademarks of their respective company.
- * Unauthorized distribution or duplication of this presentation material is prohibited.