RX600 シリーズ用リアルタイム OS　RI600/4

リアルタイム OS デバッグ入門

R20AN0109JJ0100
Rev.1.00
2011.05.31

本アプリケーションノートは、リアルタイム OS RI600/4 をご使用になられるユーザのための、デバッガを使用したデバッグ手法の入門ガイドです。

目次

1. 概要

本アプリケーションノートは、RX62N を搭載した CPU ボードをターゲットとして、リアルタイム OS RI600/4 を使用したシステムのデバッグ手法を解説するものです。デバッグには High-performance Embedded Workshop（以下 HEW と略します）に標準搭載されているリアルタイム OS 対応デバッグ機能を使用します。

1.1 開発環境

本アプリケーションノートでは、下記の開発環境を使用しています。

1.1.1 ハードウェア

・ホスト PC

・RX62N スタータキット CPU ボード

・E1 エミュレータ

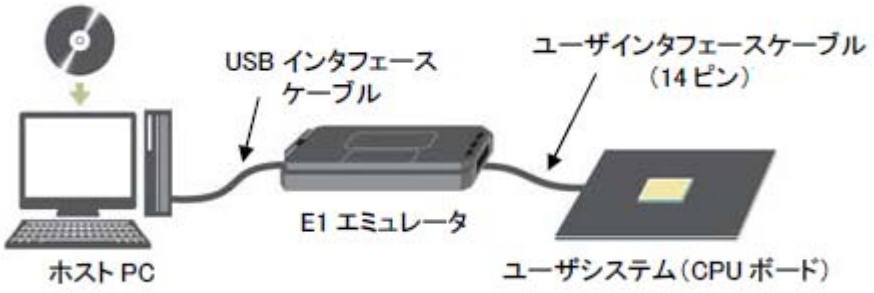


図 1-1 ハードウェア構成

1.1.2 ソフトウェア

・リアルタイム OS RI600/4 （V.1.00 Release 02）

・GUI コンフィギュレータ （Ver. 1.00.00.002）

・統合開発環境 High-performance Embedded Workshop （Version 4.08.00.011）

・RX ファミリ用 C/C++コンパイラパッケージ （V.1.00.01.003）

1.2 メモリマップ

CPU ボードのメモリマップを以下に示します。

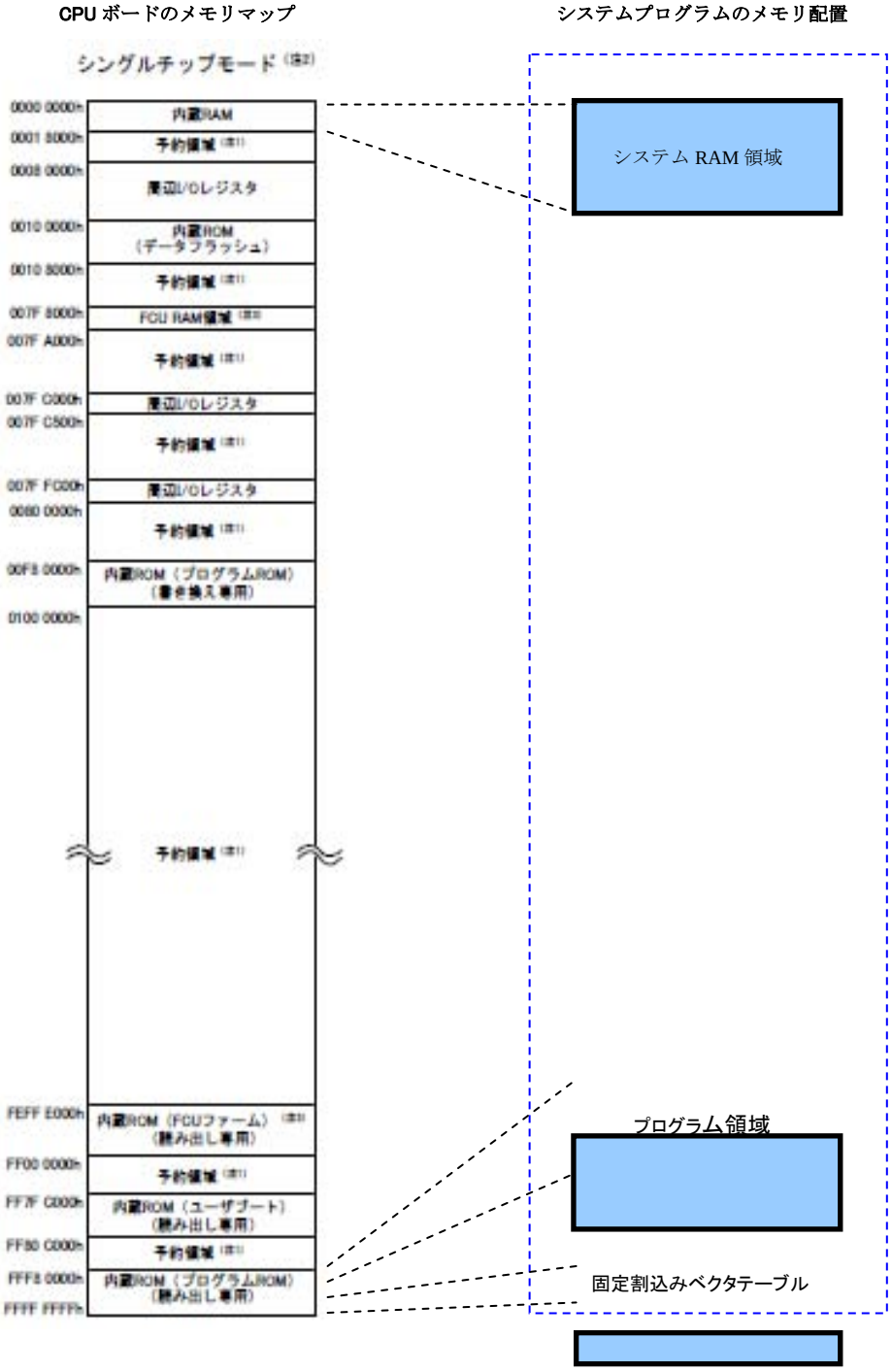


図 1-2 メモリマップ

注 1.予約領域は、アクセスしないでください。
注 2.ブートモード、ユーザーブートモードは、シングルチップモードと同じアドレス空間となります。
注 3.FCU についての詳細は、RX62N グループハードウェアマニュアル「37. ROM(コード格納用フラッシュメモリ)」、「38. データフラッシュ(データ格納用フラッシュメモリ)」をご参照ください。

サンプルプログラムのメモリ配置を以下に示します。

表 1-1　サンプルプログラムのメモリ配置

No.	名称 （配置アドレス）	セクション名	内容
1	システム RAM 領域 （0x00000000）	SI	システムスタック
2		SURI__STACK	タスクのスタック
3		B*	未初期化データ領域
4		R*	初期化データ領域
5	プログラム領域 （0xFFFF0000）	INTERRUPT_VECTOR	可変割込みベクタテーブル
6		P*	プログラム領域 （サンプルプログラムが配置されます。）
7		C*	定数領域
8		D*	初期化データ領域
9		W*	switch 文分岐テーブル領域 （サンプルでは未使用です。）
10	固定割込みベクタテーブル領域（0xFFFFFF80）	FIX_INTERRUPT_VECTOR	固定割込みベクタテーブル

2. デバッグ手順

2.1 概要

RI600/4 を使用したソフトウェア開発では、エミュレータで提供されるデバッグ機能と HEW に標準搭載されているリアルタイム OS 対応デバッグ機能を組み合わせて効率よくデバッグすることが可能です。

例えば、エミュレータのブレーク機能を使用して、プログラムで問題が発生しているポイントで停止させ、タスクの状態などを確認しながらデバッグを行い、問題点の抽出、プログラムの修正を行います。

2.2 デバッグ機能

図 2-1 にリアルタイム OS 対応デバッグ機能のウィンドウイメージを示します。

RI600/4 と E1 エミュレータを使用した場合に、使用できるリアルタイム OS 対応デバッグ機能について説明します。

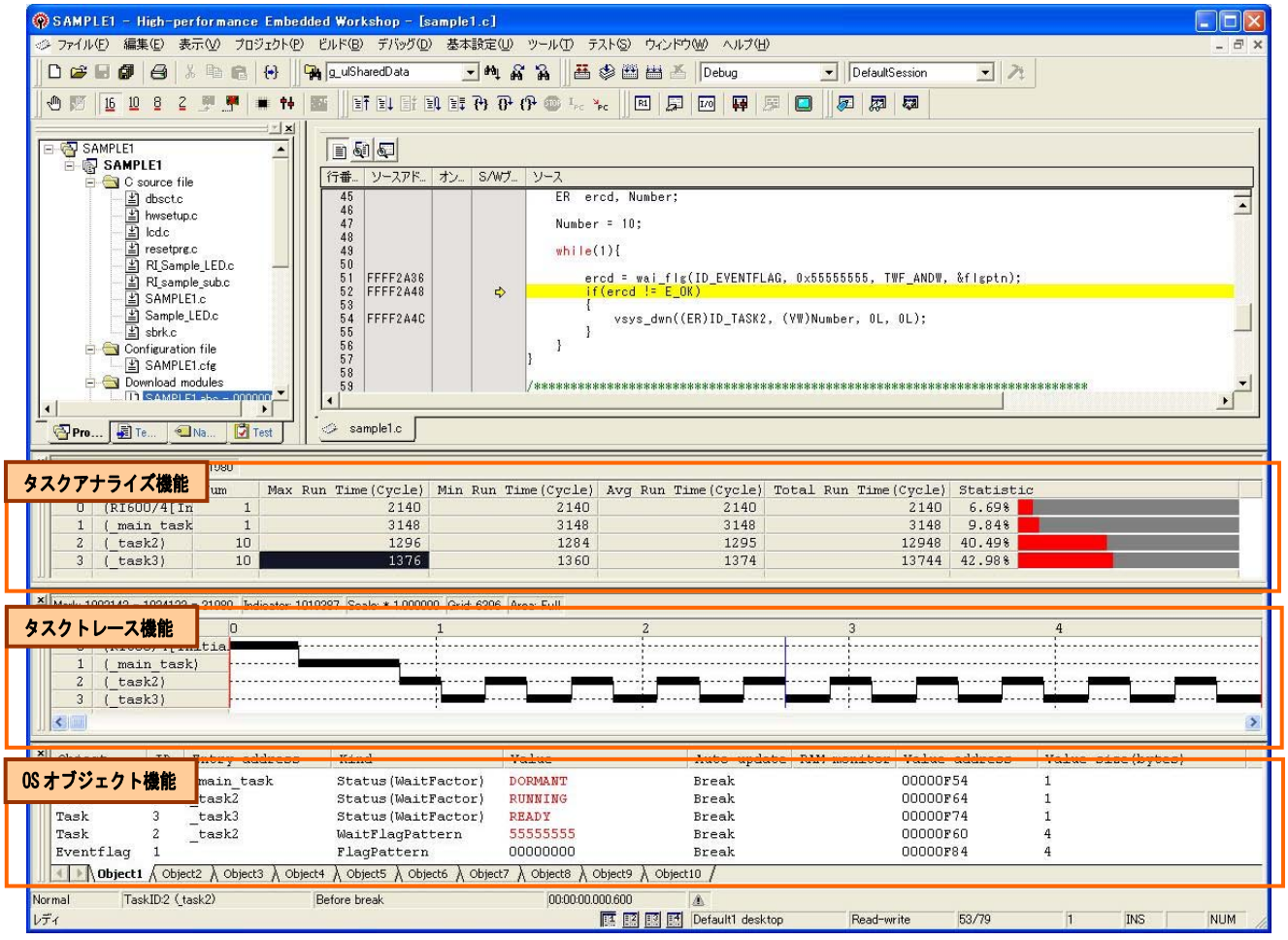


図 2-1 リアルタイム OS 対応デバッグ機能使用イメージ

(1) OS オブジェクト機能

RI600/4 が提供するタスク、イベントフラグ、セマフォなど、オブジェクトの情報が参照できます。表 2-1 に表示可能なオブジェクトの情報一覧を示します。

表 2-1 RI600/4 用 OS オブジェクト表示情報一覧

オブジェクト	取得情報	備考
タスク	状態	待ち状態時は待ち要因も表示
	現在優先度	
	イベントフラグの待ちパターン	イベントフラグ待ち状態時のみ有効
	起動要求の回数	
	起床要求の回数	
	現在のスタックポインタ	休止状態、実行状態時は不定値
システム	実行状態のタスク ID	
	システム状態	
	システム時刻（上位）	
	システム時刻（下位）	
セマフォ	現在のセマフォカウント	
イベントフラグ	現在のイベントフラグパターン	
データキュー	データキュー内に格納されているデータ数	
メールボックス	メッセージキューの先頭メッセージのアドレス	
ミューテックス	ミューテックスをロックしているタスク ID	
メッセージバッファ	メッセージバッファ領域の残りサイズ	
	メッセージバッファ領域に格納されている メッセージ数	
固定長メモリプール	空きブロック数	
周期ハンドラ	状態	
	起動までの残り時間	
アラームハンドラ	状態	
	起動までの残り時間	

OS オブジェクト機能を使用するには、HEW の[表示]－[RTOS]－[OS オブジェクト]で OS オブジェクトウィンドウを開きます。

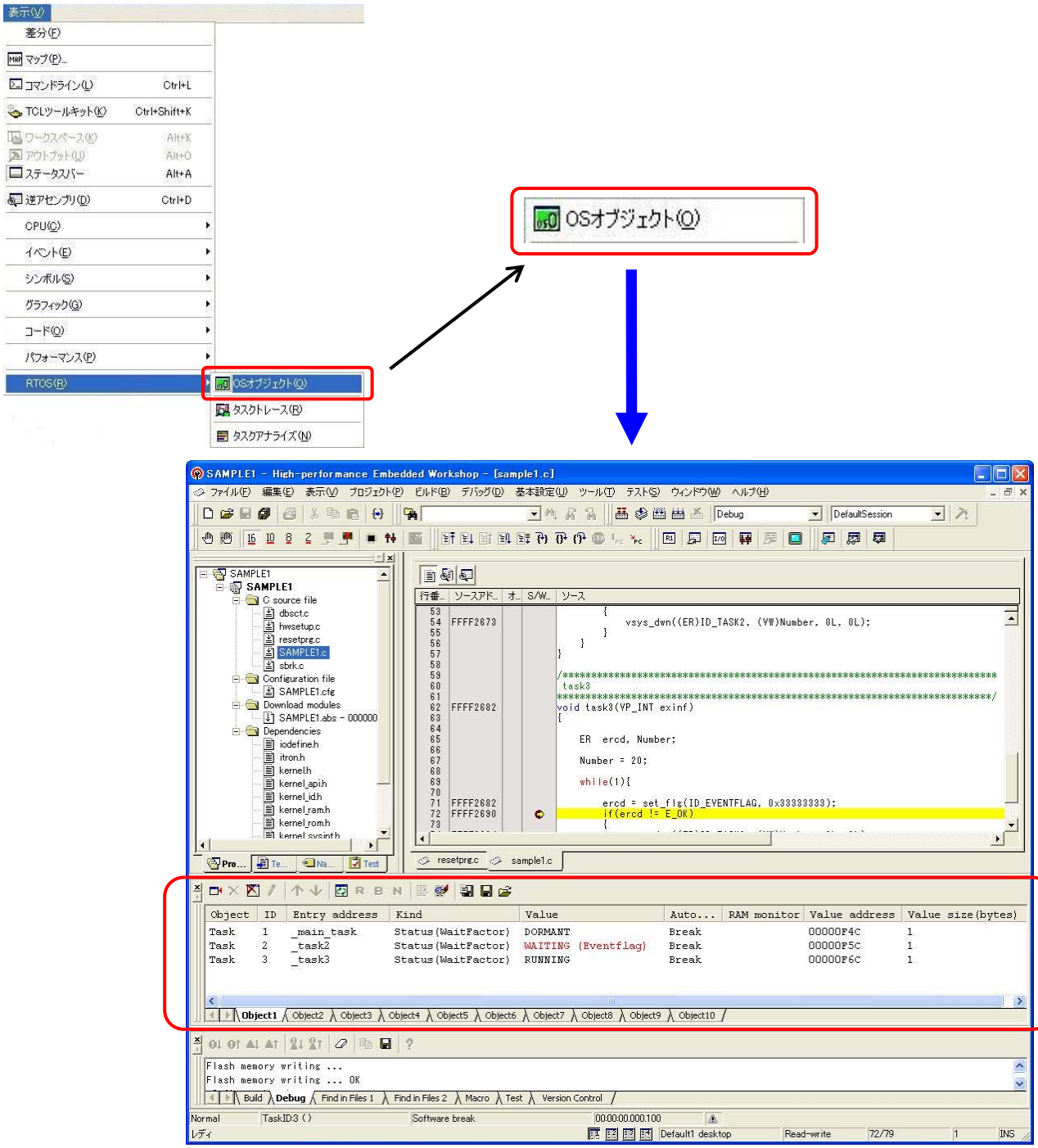


図 2-2　OS オブジェクトウィンドウの表示

OS オブジェクトウィンドウは、”Object1”から”Object10”まで 10 枚の表示シートがあり、各シートの表示内容は必要に応じてオブジェクトデータの追加や削除を行い使用します。例えば、”Object1”にはタスクの「状態」を表示、”Object2”にはタスクの「優先度」やセマフォの「状態」を表示する、というように表示情報をシートごとに分けて使うことができます。設定したオブジェクトデータを保持しますので、開くたびに再設定する必要はありません。

（初回表示の時は、”Object1”シートに全タスクの「状態」を自動的に追加し表示されます。それ以降は、任意にオブジェクトデータの追加や削除を行い使用することになります）

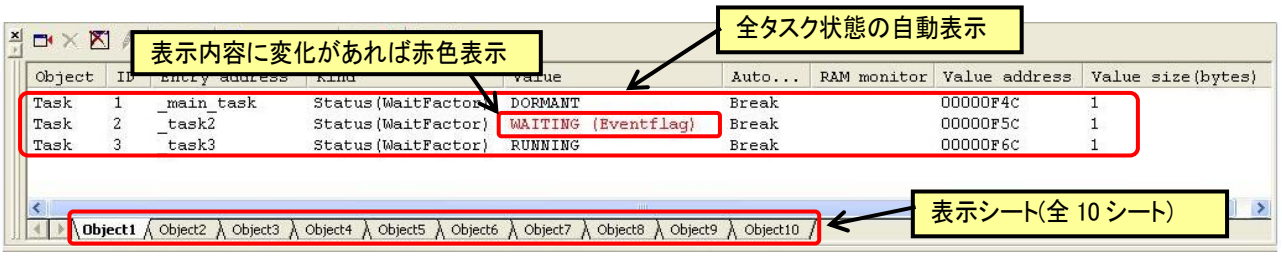


図 2-3　OS オブジェクトウィンドウの表示仕様

表示更新時にオブジェクトデータの情報に変化があった場合は、赤色文字で表示されます。

表示するオブジェクトデータの追加や削除、表示順序の変更などの設定方法は、OS オブジェクトウィンドウ上で右クリックをして、ポップアップメニューから行います。

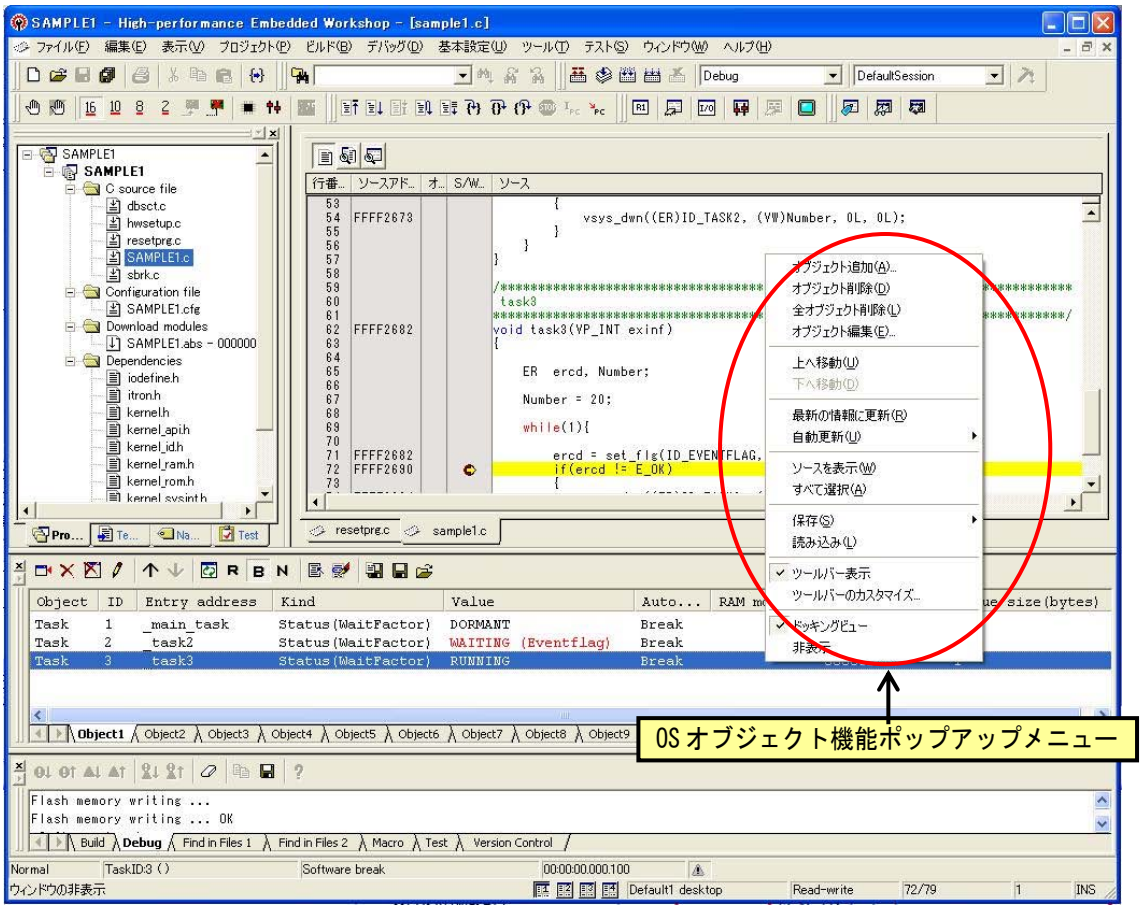


図 2-4　OS オブジェクトウィンドウのポップアップメニュー

ポップアップメニューの[オブジェクト追加]や[オブジェクト編集]では、以下のダイアログボックスが表示されます。表 2-2 に示すオブジェクトデータの各項目を設定します。

図 2-5 オブジェクトデータの選択ダイアログボックス

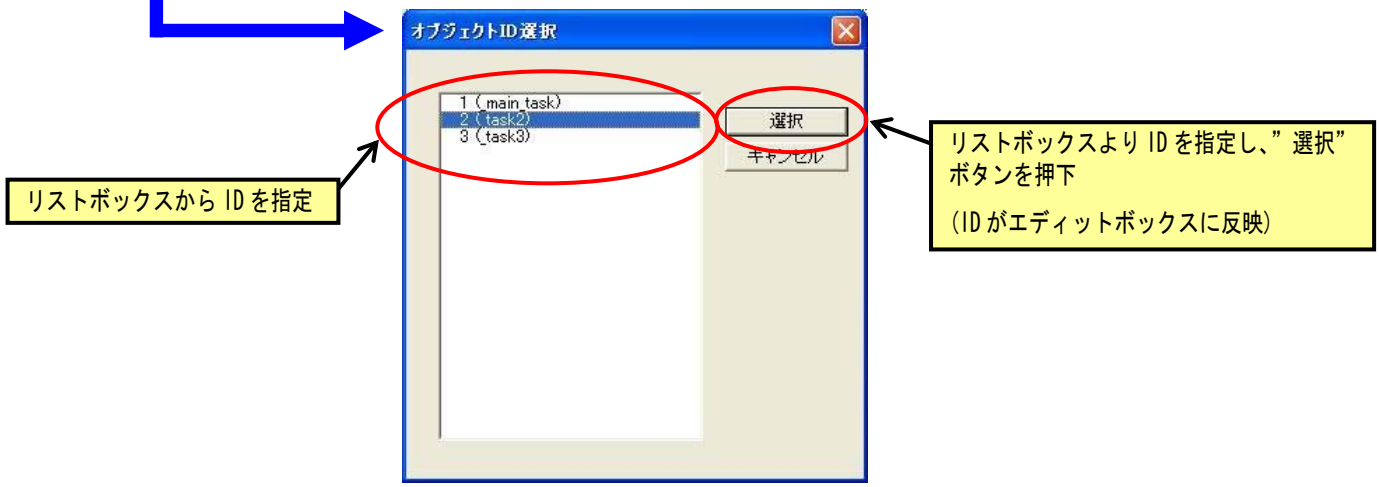


図 2-6 オブジェクト ID の選択ダイアログボックス

表 2-2 OS オブジェクトデータの設定項目

設定項目		内容
オブジェクト		プルダウンリストで選択(表示されるものが対象)
ID 番号	すべて	最大オブジェクト ID 分
	選択	エディットボックスに直接入力、または選択ボタン押下でリストダイアログボックスを表示、選択
取得情報		プルダウンリストで選択(表示されるものが対象)
表示条件	実行中更新	プログラム実行中もデータ更新表示
	停止持のみ更新	プログラム停止時にデータ更新表示
	更新しない	プログラム実行中／停止に関わらず非更新

(2) タスクトレース機能

タスクの実行履歴をグラフィカル表示、タスクの実行時間を計測する機能です。

タスクトレース機能を使用するには、HEW の[表示]－[RTOS]－[タスクトレース]でタスクトレースウィンドウを開きます。

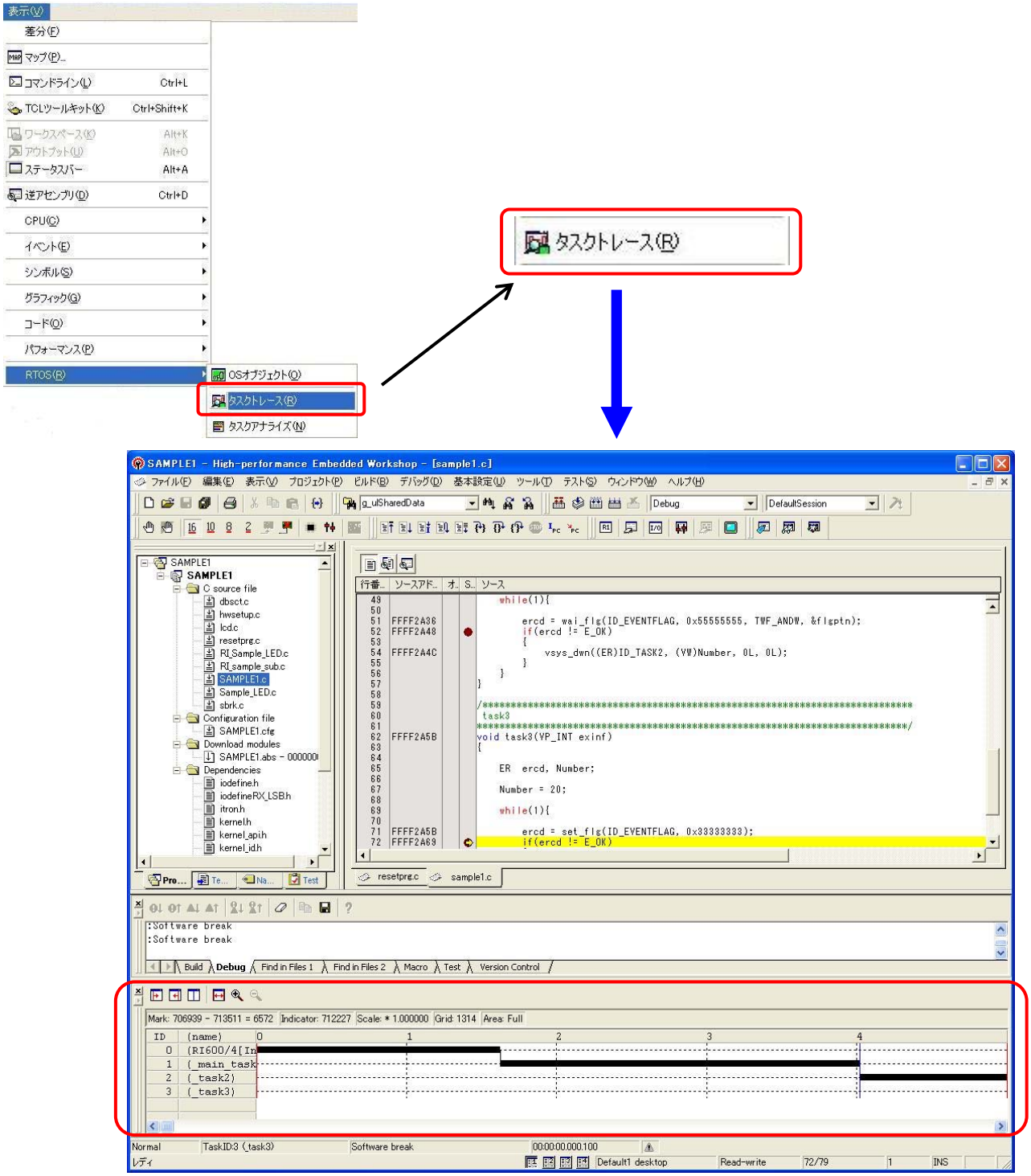


図 2-7 タスクトレースウィンドウの表示

タスク実行の結果を表示するには、タスクトレースウィンドウを開いてからプログラムの実行／停止を行います。表示されたタスク遷移図の上にマウスを移動すると詳細情報が表示されます。

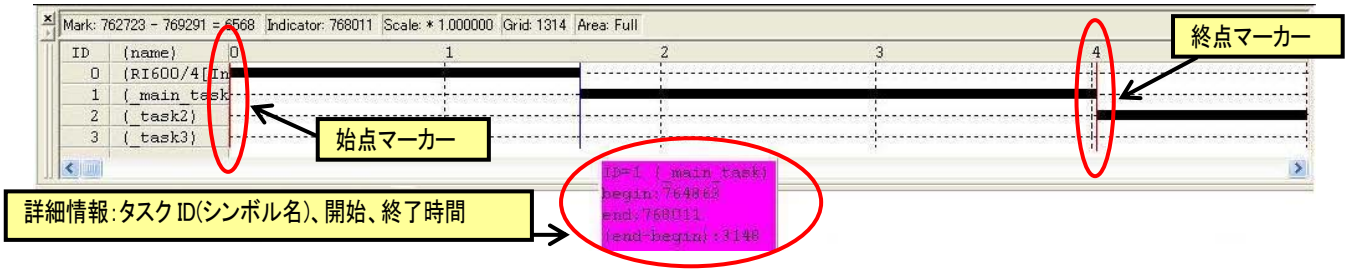


図 2-8　タスク遷移の詳細表示

タスク遷移図の拡大／縮小表示、タスクの実行時間を計測位置（始点／終点）の設定などを行うことができます。

設定方法は、タスクトレースウィンドウ上を右クリックして、ポップアップメニューから行います。

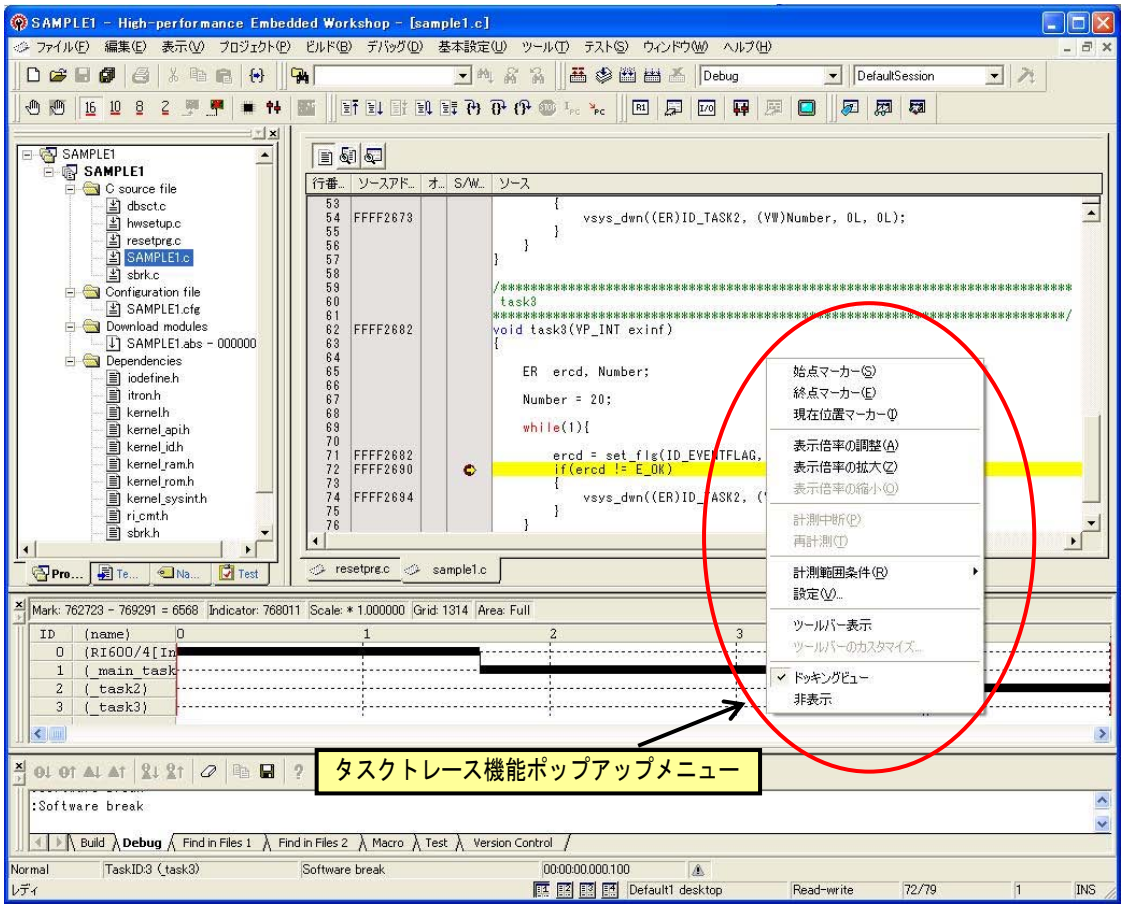


図 2-9　タスクトレースウィンドウのポップアップメニュー

(3) タスクアナライズ機能

各タスクの CPU 占有状況を表示する機能です。

タスクアナライズ機能を使用するには、HEW の[表示]－[RTOS]－[タスクアナライズ]でタスクアナライズウィンドウを開きます。

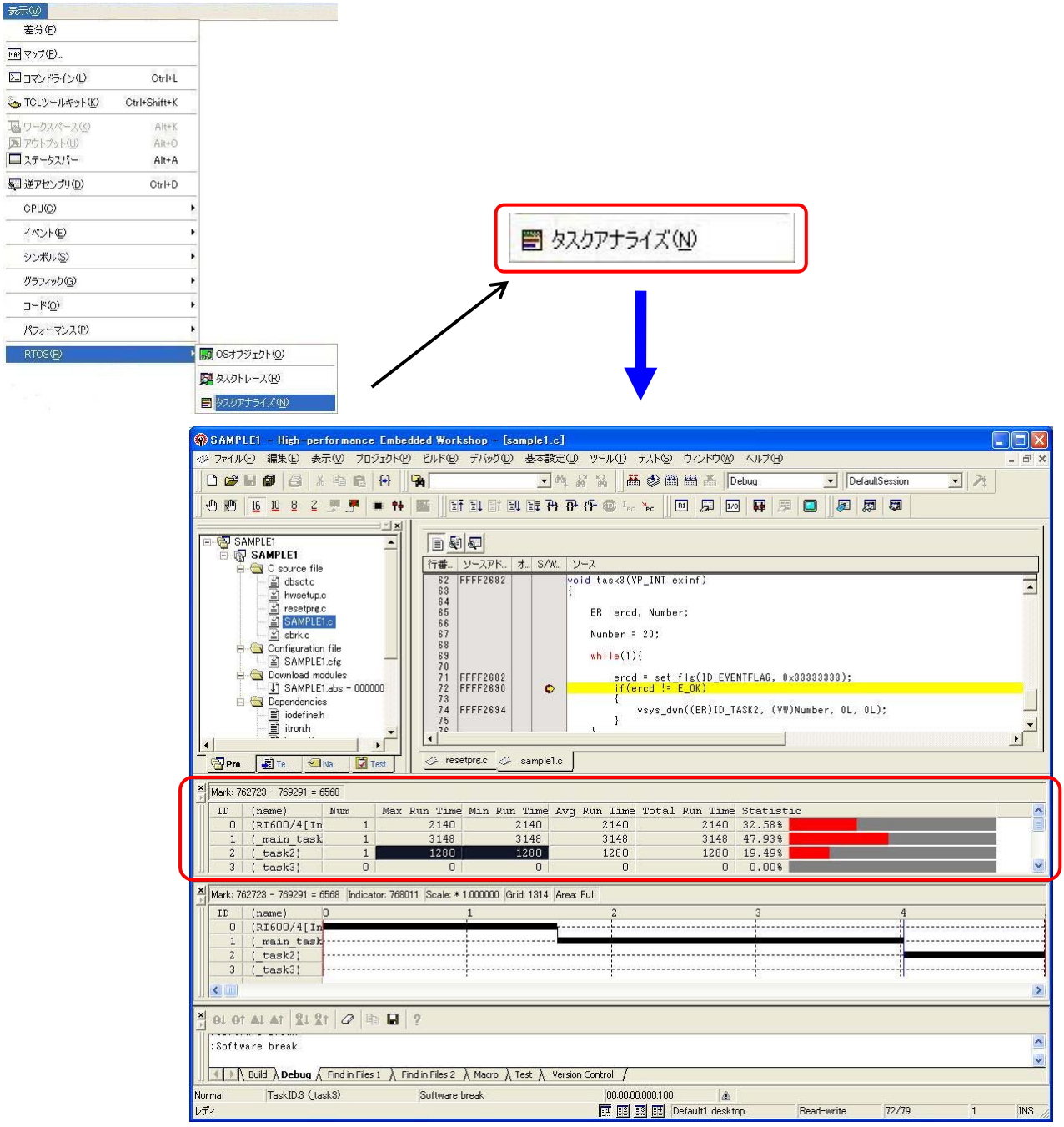


図 2-10 タスクアナライズウィンドウの表示

タスクアナライズウィンドウでは、タスクトレースウィンドウで指定した始点マーカーと終点マーカーの範囲を計測し表示します。

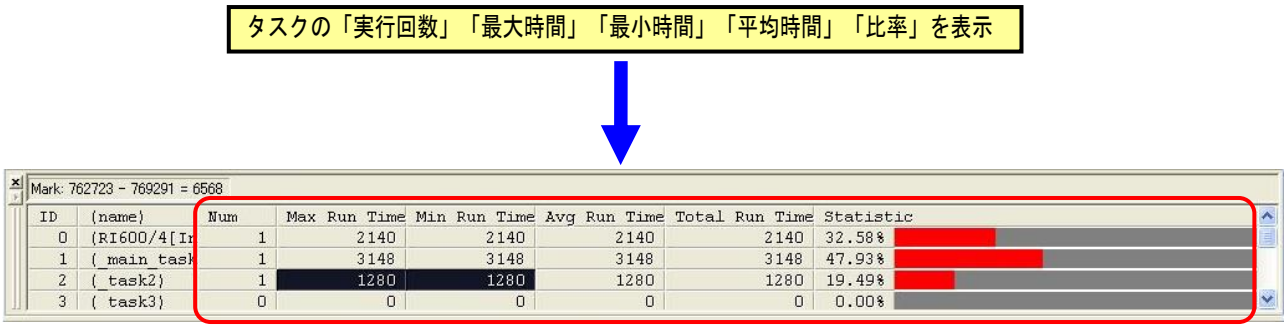


図 2-11　始点マーカーから終点マーカーまでの計測表示

タスクアナライズで表示された情報をファイルに保存することができます。

この操作はタスクアナライズウィンドウ上で右クリックをしてポップアップメニューから行います。

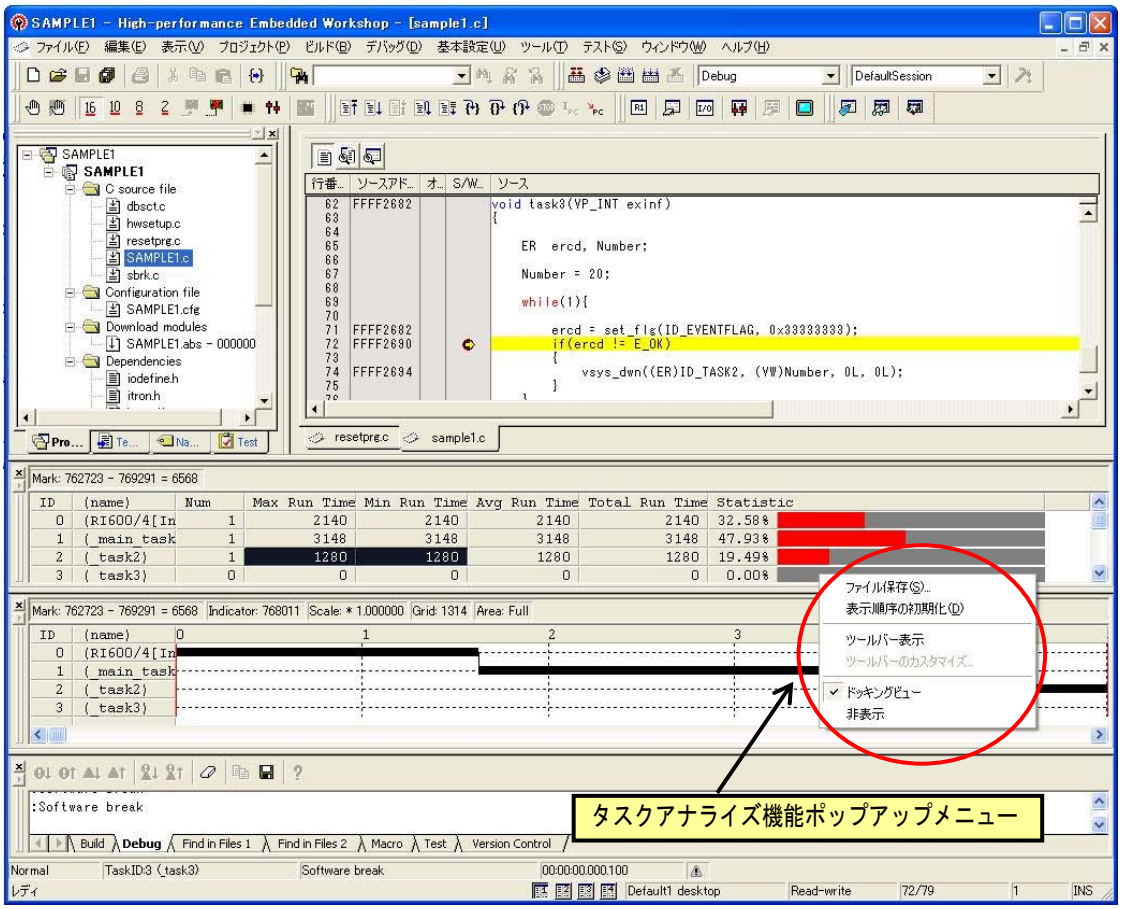


図 2-12　タスクアナライズウィンドウのポップアップメニュー

(4) タスクステップ機能

特定のタスクに着目してステップ実行する機能です。

タスクステップ機能は、HEW の[デバッグ]－[RTOS デバッグ]の各メニューで実行します。

タスクステップは、実行状態のタスクに対して行われます。

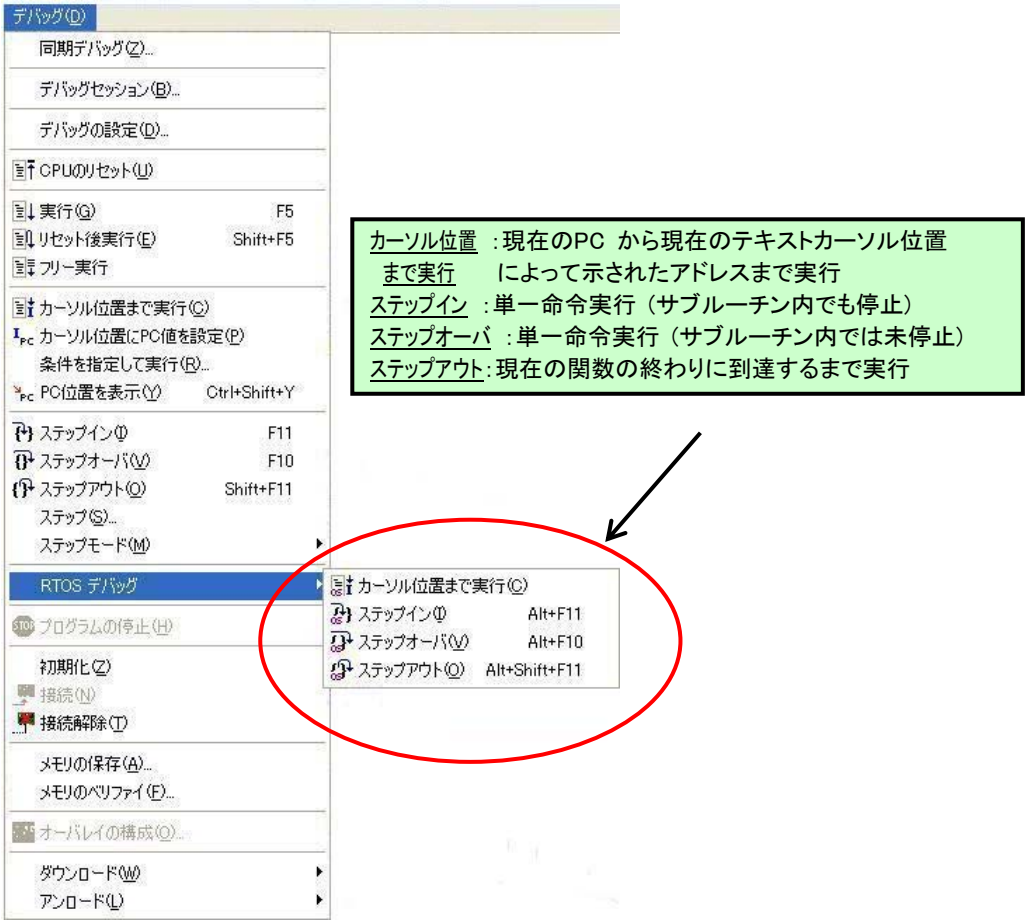


図 2-13 タスクステップ機能メニュー

(5) 実行状態タスク表示機能

実行状態のタスク ID をステータスバーに表示する機能です。

HEW の左下に実行状態のタスク ID とシンボル名を自動的に表示します。

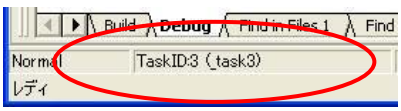


図 2-14 現在実行中のタスク ID 表示

3. サンプルプログラムを用いたデバッグ手法

RI600/4 カーネルの機能を使用したサンプルプログラム SAMPLE1～SAMPLE10 を使って、デバッグ手法を説明します。

SAMPLE1. イベントフラグを使用した同期のデバッグ

SAMPLE2. メールボックスの先頭メッセージ内容確認

SAMPLE3. 固定長メモリプールの空きメモリブロック数確認

SAMPLE4. タスク動作順序のデバッグ

SAMPLE5. イベント発生の周期時間確認

SAMPLE6. セマフォ資源獲得に関する不正動作のデバッグ

SAMPLE7. デッドロック状態のデバッグ

SAMPLE8. 共通関数でのタスク ID 確認

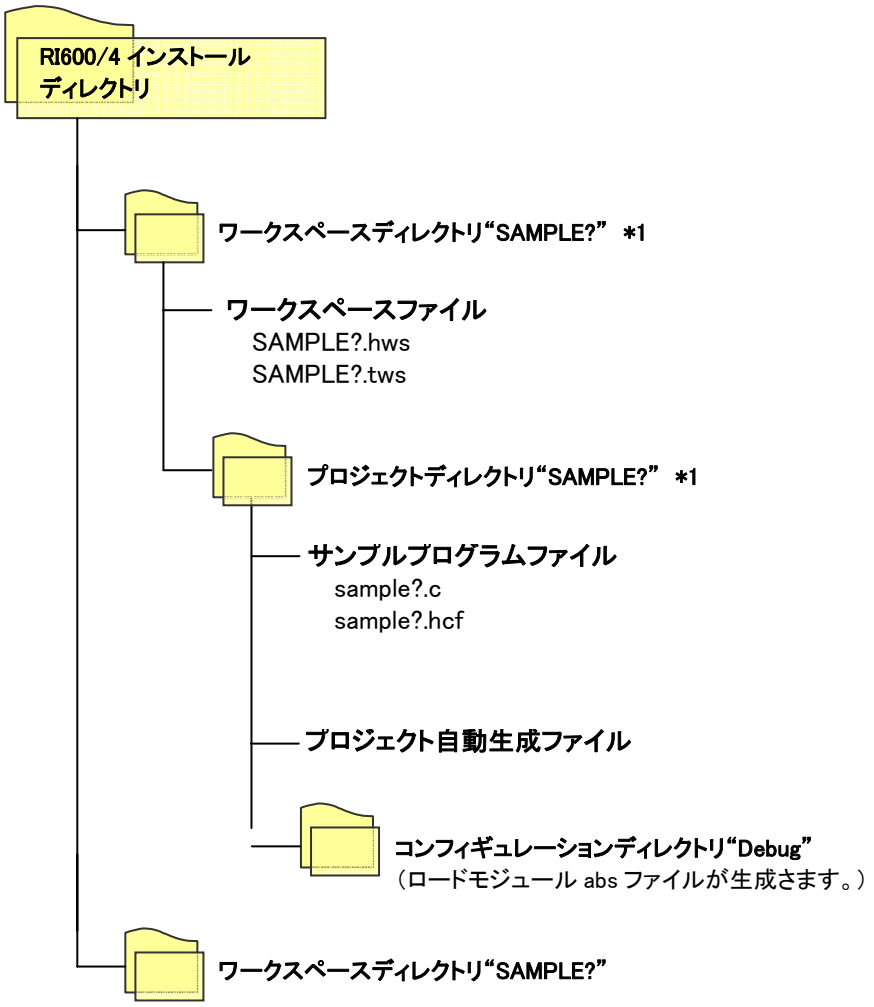
SAMPLE9. システム異常終了のデバッグ

SAMPLE10. システムのスループット向上策

3.1 サンプルプログラムの構成

図 3-1 にサンプルプログラムのディレクトリ構成を示します。

本アプリケーションノートで提供するサンプルプログラムは、RI600/4 のインストールディレクトリに格納してください。



【注】*1 SAMPLE?の“?“はサンプルプログラムの番号を意味します。

図 3-1 サンプルプログラムディレクトリ構成

3.2 システムの起動

システムを起動するには、サンプルとして提供しています HEW のワークスペースファイル「sample?.hws」(”?”はサンプルプログラムの番号)ファイルをダブルクリックします。

(1) ターゲットシステムの接続

ターゲットシステムに接続するために起動設定を行います。

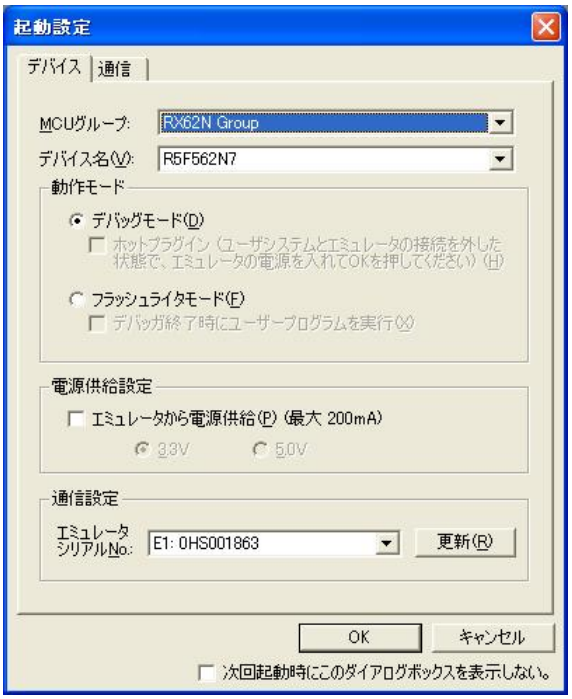


図 3-2 E1 エミュレータ起動設定ダイアログ

[MCU グループ]	RX62N Group を選択
[デバイス名]	R5F562N7 を選択
[動作モード]	“デバッグモード”ラジオボタンを選択
[電源供給設定]	“エミュレータから電源供給”チェックボックスを未チェック
[通信設定]	エミュレータシリアル No. を確認

以下、接続中のダイアログボックスが表示されます。

図 3-3 E1 エミュレータ接続中ダイアログ 1

次にコンフィギュレーションプロパティを設定します。

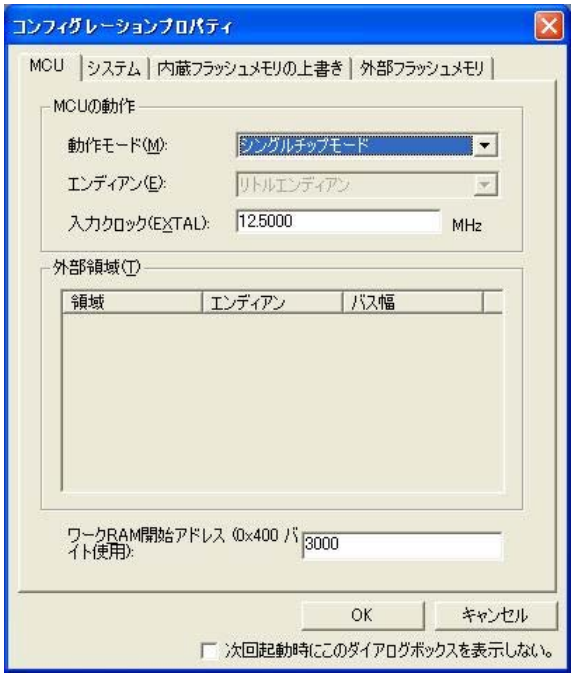


図 3-4 E1 エミュレータコンフィグレーションプロパティダイアログ

[動作モード]	シングルチップモード を選択
[入力クロック]	12.5000MHz を指定

以下、接続中のダイアログボックスが表示されます。

図 3-5 E1 エミュレータ接続中ダイアログ 2

これで、ターゲットシステムが接続されました。

(2) ダウンロード

HEW の[デバッグ]－[ダウンロード]でロードモジュール(「sample?.abs」ファイル)をダウンロードします。

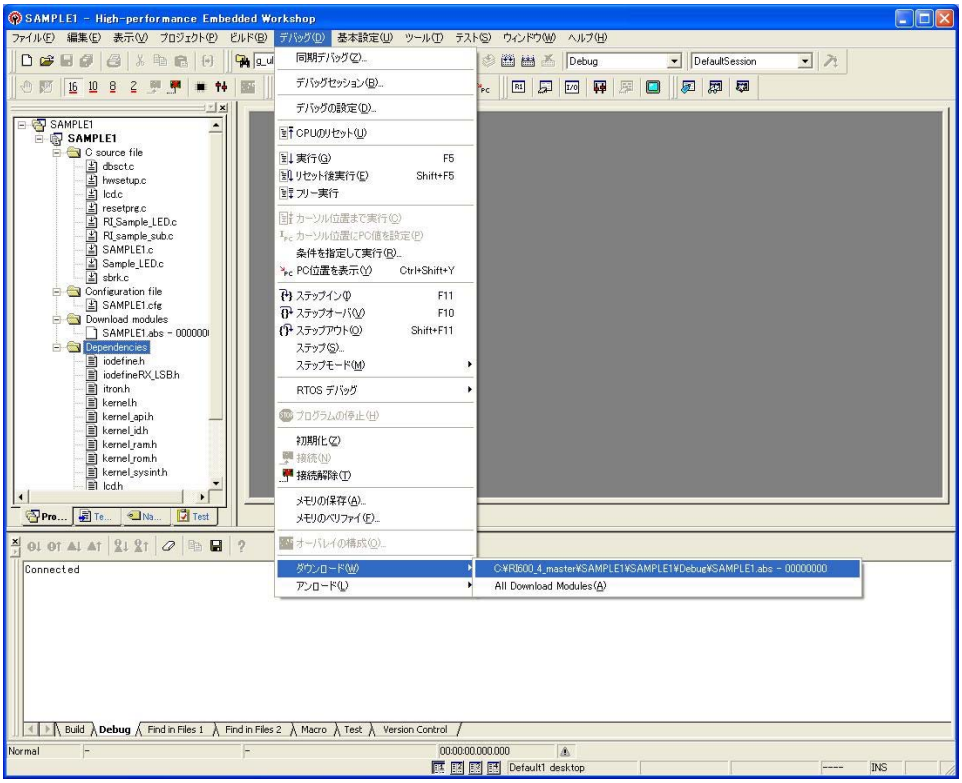


図 3-6 サンプルプログラムのダウンロード

ダウンロードが完了すると、OS 選択ダイアログボックスが表示されます。リストボックスに表示される「RI600/4」を選択し、OK ボタンを押します。

図 3-7 OS 定義ファイル選択ダイアログボックス

これで、デバッグ対象プログラムがユーザシステムにダウンロードされ、デバッグの環境が整いました。

(3) プログラムの実行と停止

HEW の[デバッグ]ー[実行]でプログラムを実行します。

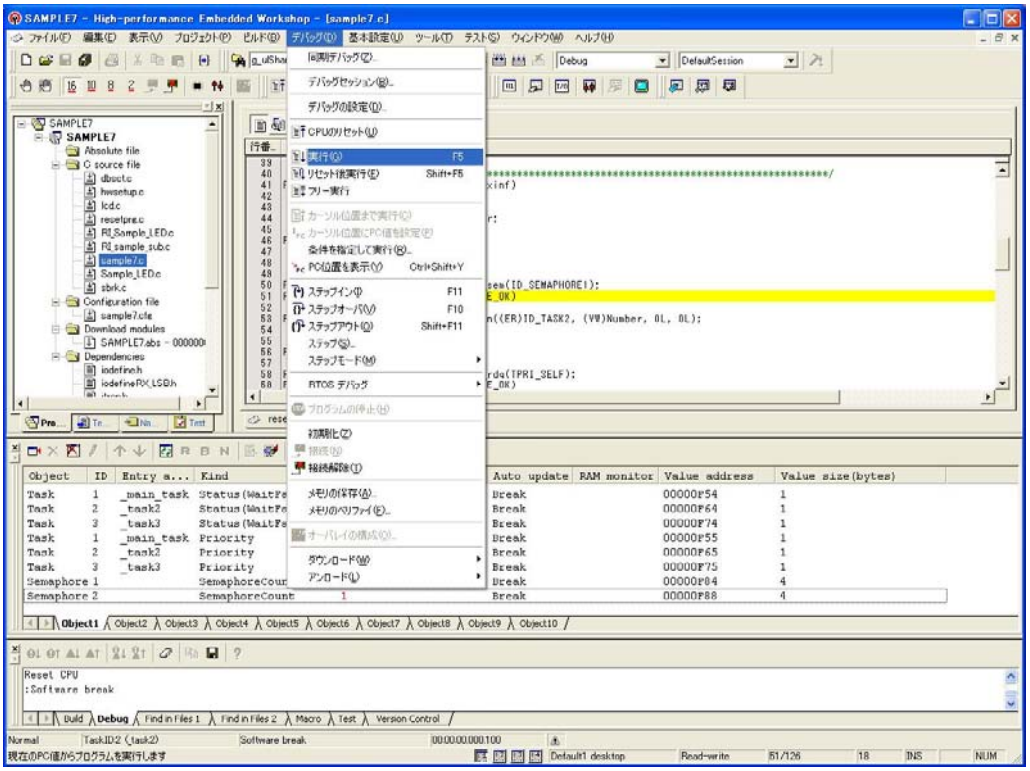


図 3-8 プログラムの実行

HEW の[デバッグ]ー[プログラムの停止]でプログラムを停止します。

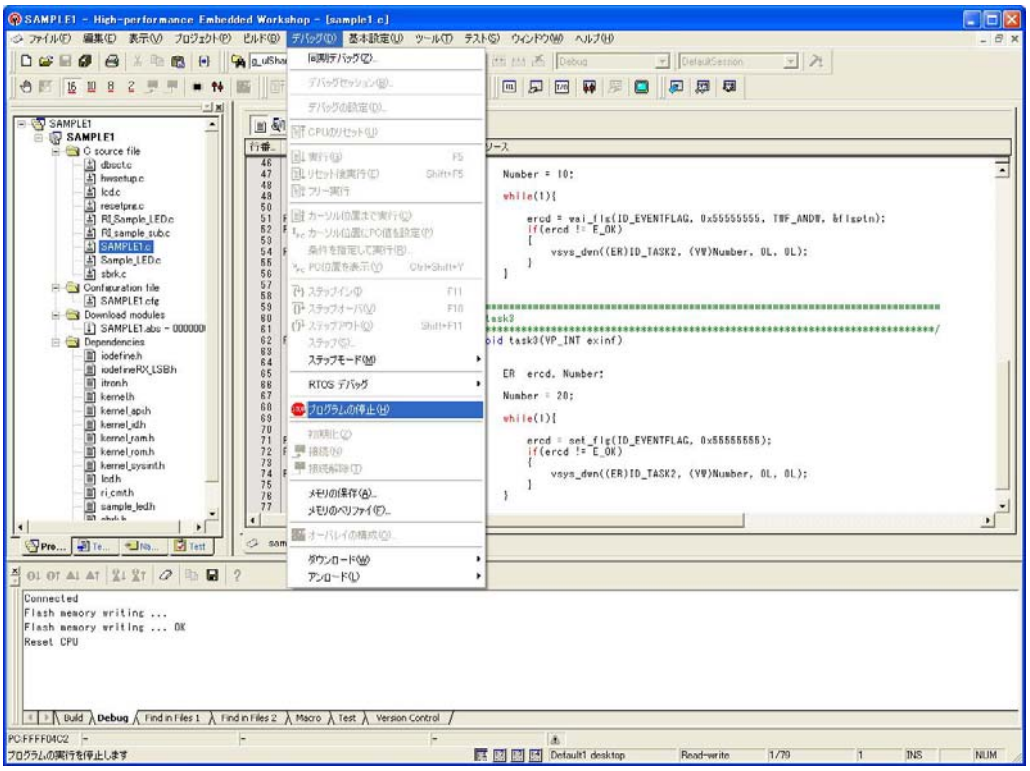


図 3-9 プログラムの停止

3.3 サンプルプログラムのデバッグ

サンプルプログラムを使用して、実際にプログラムをデバッグしてみましょう。

3.3.1 SAMPLE1：イベントが発生しない、イベント待ちが解除されない

SAMPLE1 は、イベントフラグを使用して下記のように動作するプログラムを作成しました。

(a) main_task で、task2、task3 を起動します。

(b) 優先度の高い task2 が起動されます。

(c) task2 は、task3 からのイベント発生通知が発行されるまで待ち状態に入ります。

(d) task3 が起動されます。

(e) task3 は、task2 に対してイベント発生を通知し、task2 の待ち状態を解除します。

■使用するオブジェクト

表 3-1 タスク（SAMPLE1）

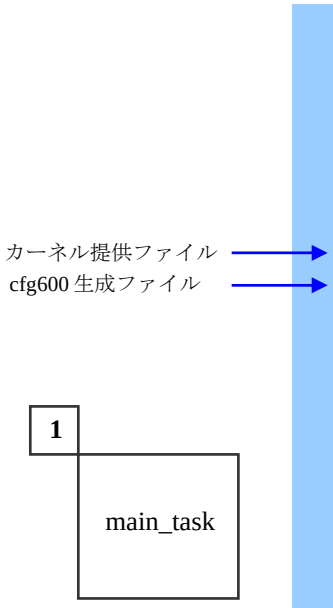
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	イベントフラグでイベント発生を待つ
task3	3	3	イベント発生をイベントフラグに通知

表 3-2 イベントフラグ（SAMPLE1）

名称	ID 番号	属性	初期ビットパターン
ID_EVENTFLAG	1	TA_CLR TA_TFIFO TA_WSGL	0x00000000

このサンプルプログラムを動作させたところ、task3 からのイベント発生を待っている task2 の待ちが解除されません。

ソースコードは、以下のように記述しています。



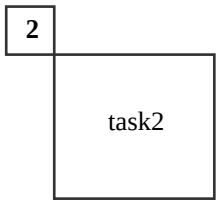
カーネル提供ファイル
cfg600 生成ファイル

```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

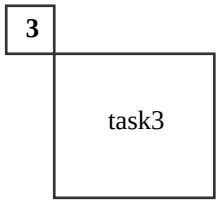
task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

図 3-10　サンプルプログラムソースコード『SAMPLE1.c』(1/2)



イベントフラグ待ち
（wai_flg）



イベント発生通知
（set_flg）

```
/*************************************************************************
  task2
*************************************************************************/
void task2(VP_INT exinf)
{

        FLGPTN flgptn;
        ER     ercd, Number;

        Number = 10;

        while(1){

                ercd = wai_flg((ID)ID_EVENTFLAG, (FLGPTN)0x55555555,
                                MODE)TWF_ANDW, (FLGPTN *)&flgptn);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }
        }
}

/*************************************************************************
  task3
*************************************************************************/
void task3(VP_INT exinf)
{

        ER     ercd, Number;

        Number = 20;

        while(1){

                ercd = set_flg((ID)ID_EVENTFLAG, (FLGPTN)0x33333333);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-10　サンプルプログラムソースコード『SAMPLE1.c』(2/2)

(2) デバッグ開始

wai_flg サービスコール発行タスク(task2)が待ち状態から解除されない、という問題の原因として、task3 で待ち解除のための set_flg サービスコールが発行されていない、または、task3 で発行されていても set_flg サービスコールのパラメータが、待ち解除条件を満たしていないということが考えられます。

これを確認するため、E1 エミュレータの S/W(ソフトウェア)ブレーク機能と OS オブジェクト機能を使用してデバッグします。

まず、各サービスコールのパラメータがどうなっているかを確認するために、set_flg と wai_flg サービスコールを実行した次の命令に S/W ブレークポイントを設定します。

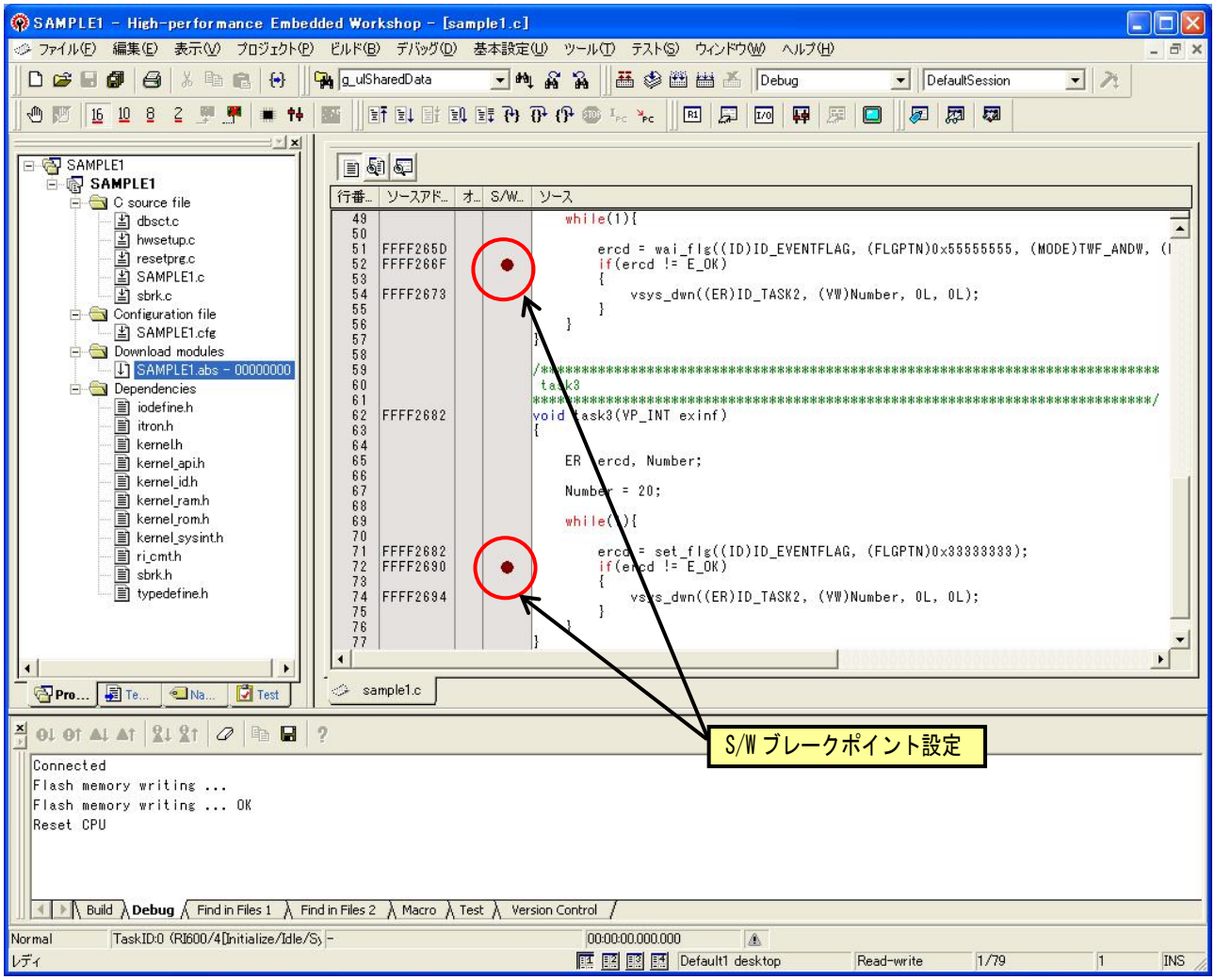


図 3-11 S/W ブレークポイント設定(SAMPLE1)

プログラムを実行すると、set_flg サービスコールの実行後に設定した S/W ブレークポイントで停止しました。ここで OS オブジェクトウィンドウを開き、オブジェクトの情報を確認します。

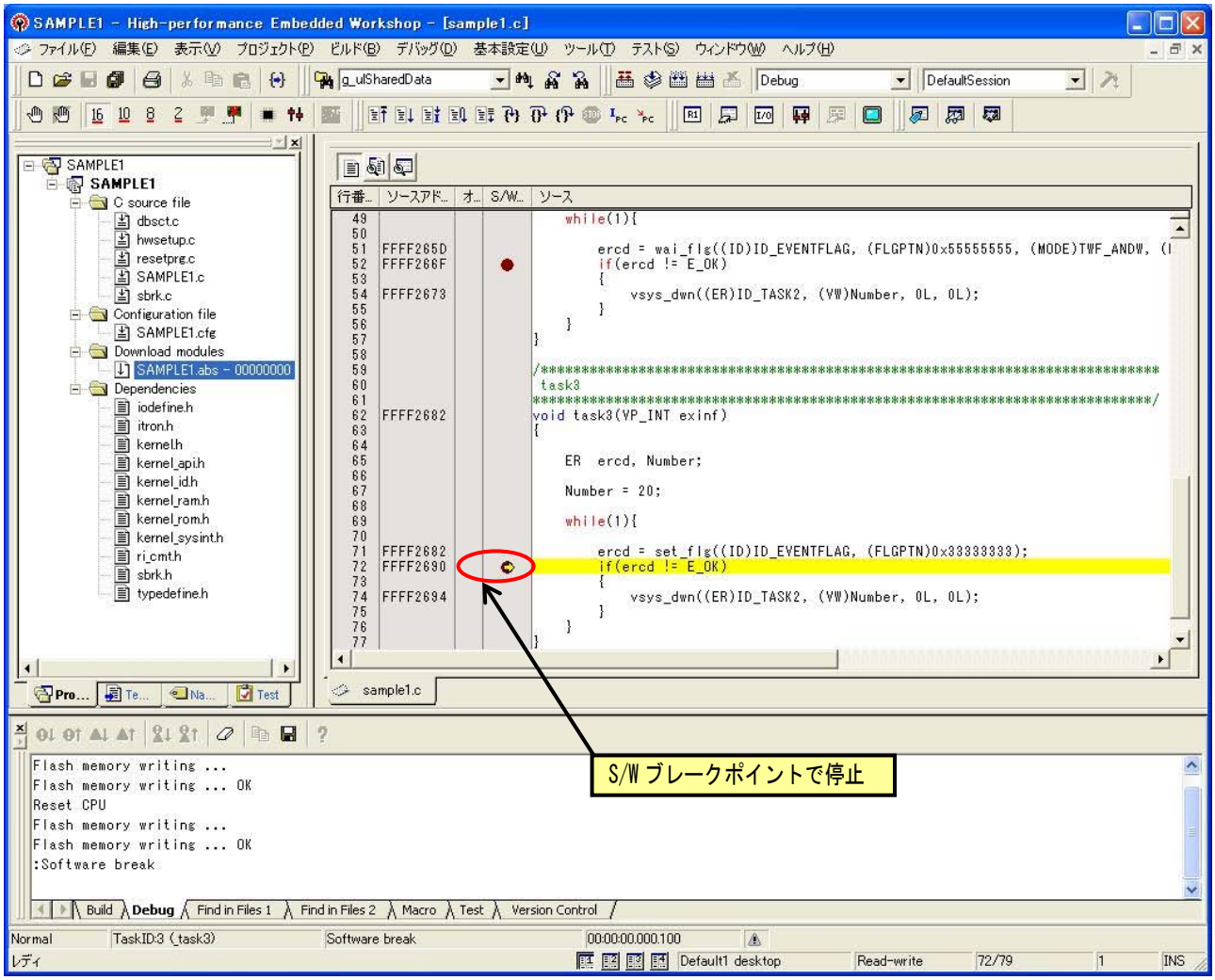


図 3-12　S/W ブレークポイント発生（SAMPLE1）

OSオブジェクトウィンドウを開いたら、オブジェクトデータを選択するウィンドウを開き、task2 の wai_flg サービスコールによる待ちビットパターンを追加します。

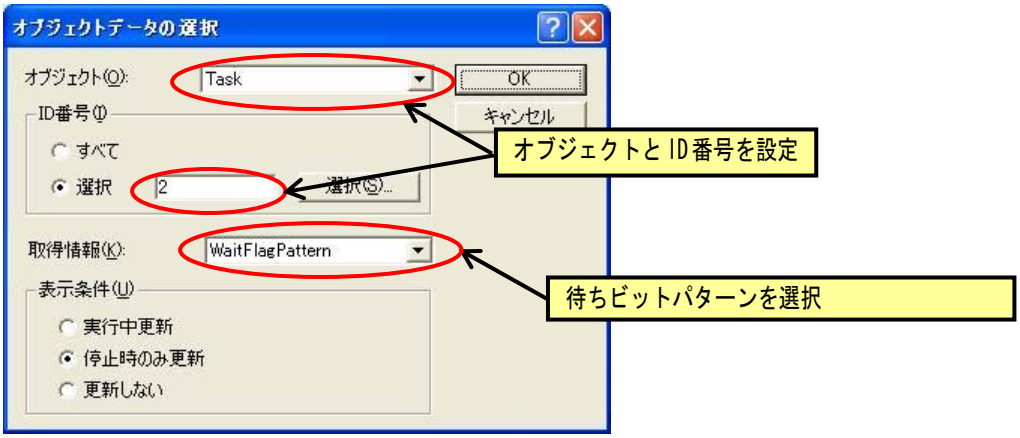


図 3-13　task2 の待ちビットパターン追加（SAMPLE1）

task3 の set_flg サービスコールでセットしたイベントフラグのビットパターン（ID_EVENTFLAG）を追加します。

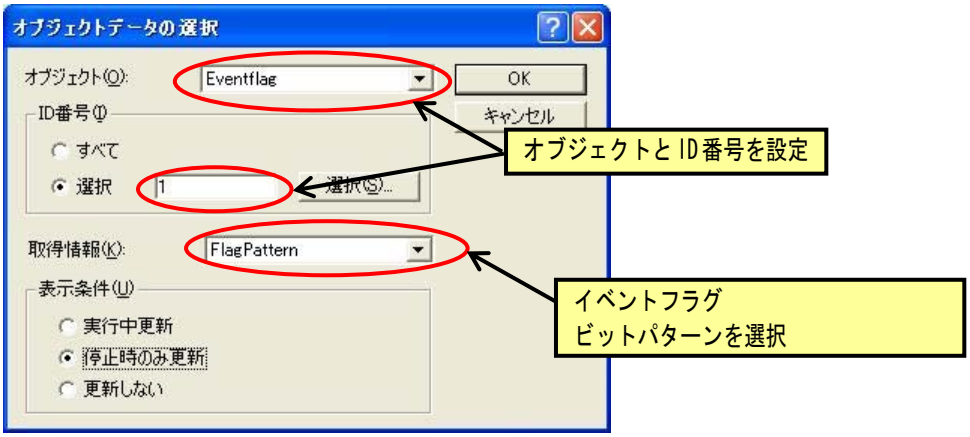


図 3-14　ID_EVENTFLAG のイベントフラグビットパターン追加（SAMPLE1）

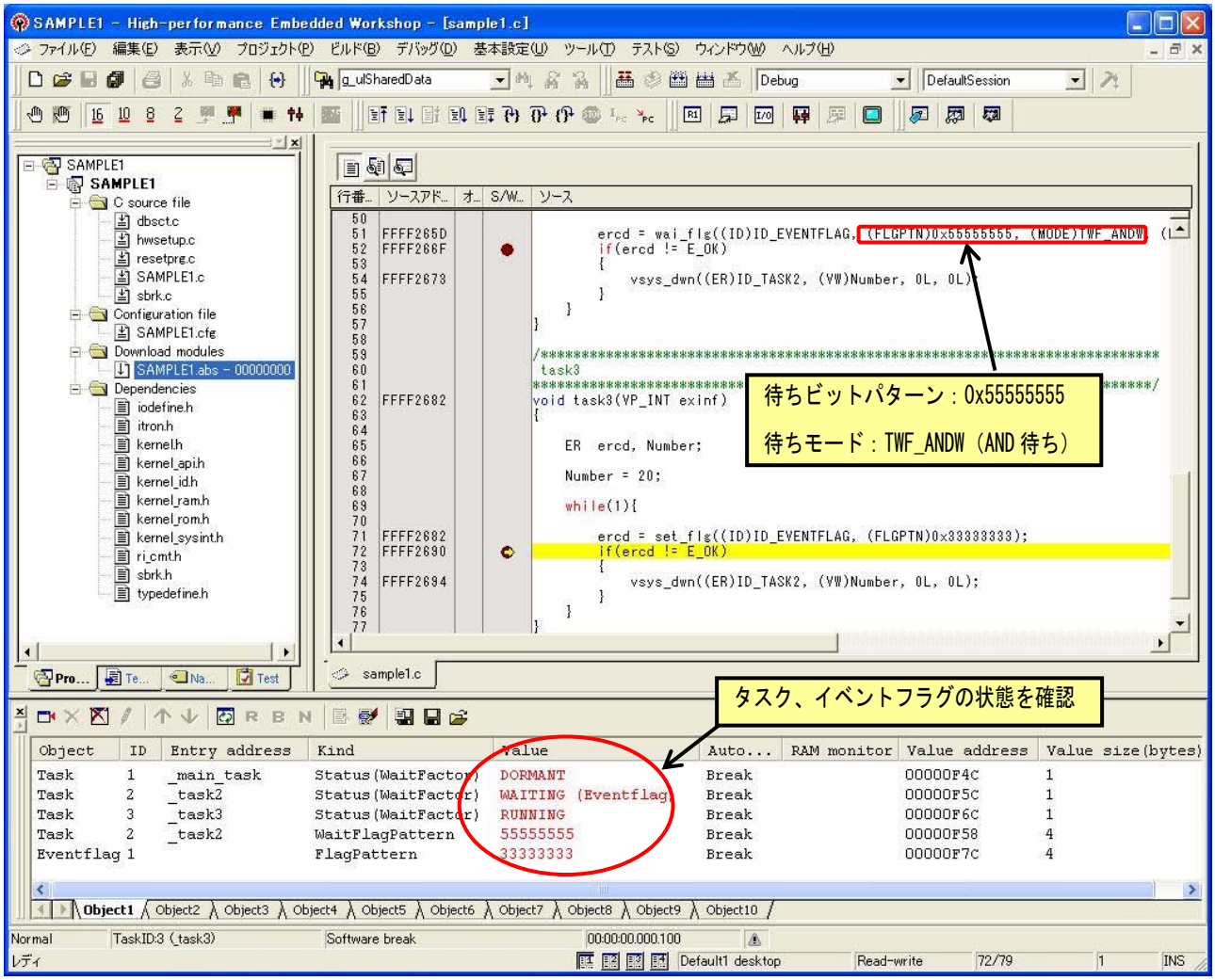


図 3-15　OS オブジェクトの状態確認（SAMPLE1）

OS オブジェクトウィンドウに表示されたタスク、イベントフラグの状態を見てみると、task2 の wai_flg サービスコールでの待ちビットパターンは 0x55555555、task3 の set_flg サービスコールでセットするビットパターンは 0x33333333 と表示されています。またソースコードから task2 の wai_flg サービスコールの待ちモードは TWF_ANDW(AND 待ち)を指定していることから、task2 の待ち状態解除条件を満たしていないことが分かりました。

したがって、このサンプルプログラムを正しく動作させるためには、task2 の待ち状態解除条件を満たすように、task3 の set_flg サービスコールでセットするビットパターンを 0x55555555 にすれば良いことが分かりました。

3.3.2 SAMPLE2：メールボックスのメッセージ内容を確認する

SAMPLE2 は、メールボックスを使用して下記のように動作するプログラムを作成しました。

(a) main_task で、task2、task3 を起動します。

(b) task2 が起動されます。

(c) task2 は、メールボックスに対してメッセージを送信し、その後実行権を task3 に渡します。

(d) task3 が起動されます。

(e) task3 は、task2 が送信したメッセージを受信し、その後実行権を task2 に戻します。

■使用するオブジェクト

表 3-3 タスク（SAMPLE2）

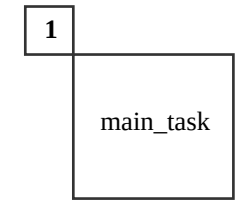
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	メッセージを送信する
task3	3	2	メッセージを受信する

表 3-4 メールボックス（SAMPLE2）

名称	ID 番号	属性
ID_MAILBOX1	1	TA_TFIFO TA_MFIFO

このサンプルプログラムでは、task2 でメッセージを生成してメールボックスに送信しますが、このメッセージが正しく生成・送信されたか確認する方法を説明します。

ソースコードは、以下のように記述しています。



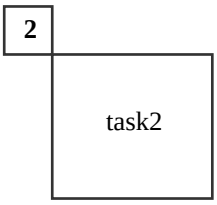
カーネル提供ファイル
cfg600 生成ファイル

task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

typedef struct {
                    T_MSG t_msg;      /* T_MSG  構造体  */
                    B data[8];        /*  ユーザメッセージデータ構造の例(任意の構造) */
} USER_MSG;

/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

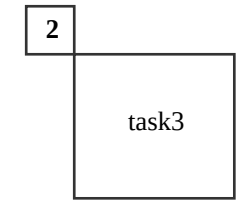
図 3-16　サンプルプログラムソースコード『SAMPLE2.c』(1/3)




```
/***********************************************************************
  task2
***********************************************************************/

USER_MSG user_msg;

void task2(VP_INT exinf)
{

        extern USER_MSG user_msg;
        T_MSG       *p_user_msg;

        ER    ercd, Number;

        Number = 10;

        p_user_msg = (T_MSG *)&user_msg;

        while(1){

                        user_msg.data[0] = 0;
                        user_msg.data[1] = 1;
                        user_msg.data[2] = 2;
                        user_msg.data[3] = 3;
                        user_msg.data[4] = 4;
                        user_msg.data[5] = 5;
                        user_msg.data[6] = 6;
                        user_msg.data[7] = 7;

            ercd = snd_mbx((ID)ID_MAILBOX, p_user_msg);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }
                Number++;

            ercd = rot_rdq((PRI)TPRI_SELF);
            if(ercd != E_OK)
            {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
            }

        }
}
```

メールボックスへ送信
（snd_mbx）

図 3-16　サンプルプログラムソースコード『SAMPLE2.c』(2/3)



メールボックスから受信
（rcv_mbx）

```
/*****************************************************************************
 task3
*****************************************************************************/
void task3(VP_INT exinf)
{
        T_MSG     **p_user_msg;

        ER    ercd, Number;

        Number = 20;

        while(1){

                ercd = rcv_mbx((ID)ID_MAILBOX, p_user_msg);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }

                ercd = rot_rdq((PRI)TPRI_SELF);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-16　サンプルプログラムソースコード『SAMPLE2.c』(3/3)

(2) デバッグ開始

task2 で snd_mbx サービスコールを使って送信したメッセージを、task3 の rcv_mbx サービスコールでメッセージのアドレスを正しく受け取ったかを確認するため、E1 エミュレータの S/W ブレーク機能、ラベル機能、メモリ機能と OS オブジェクト機能を使用します。

まず、task3 がメッセージを受信する前のメールボックスの状態を知るために、task3 の rcv_mbx サービスコールに S/W ブレークポイントを設定します。

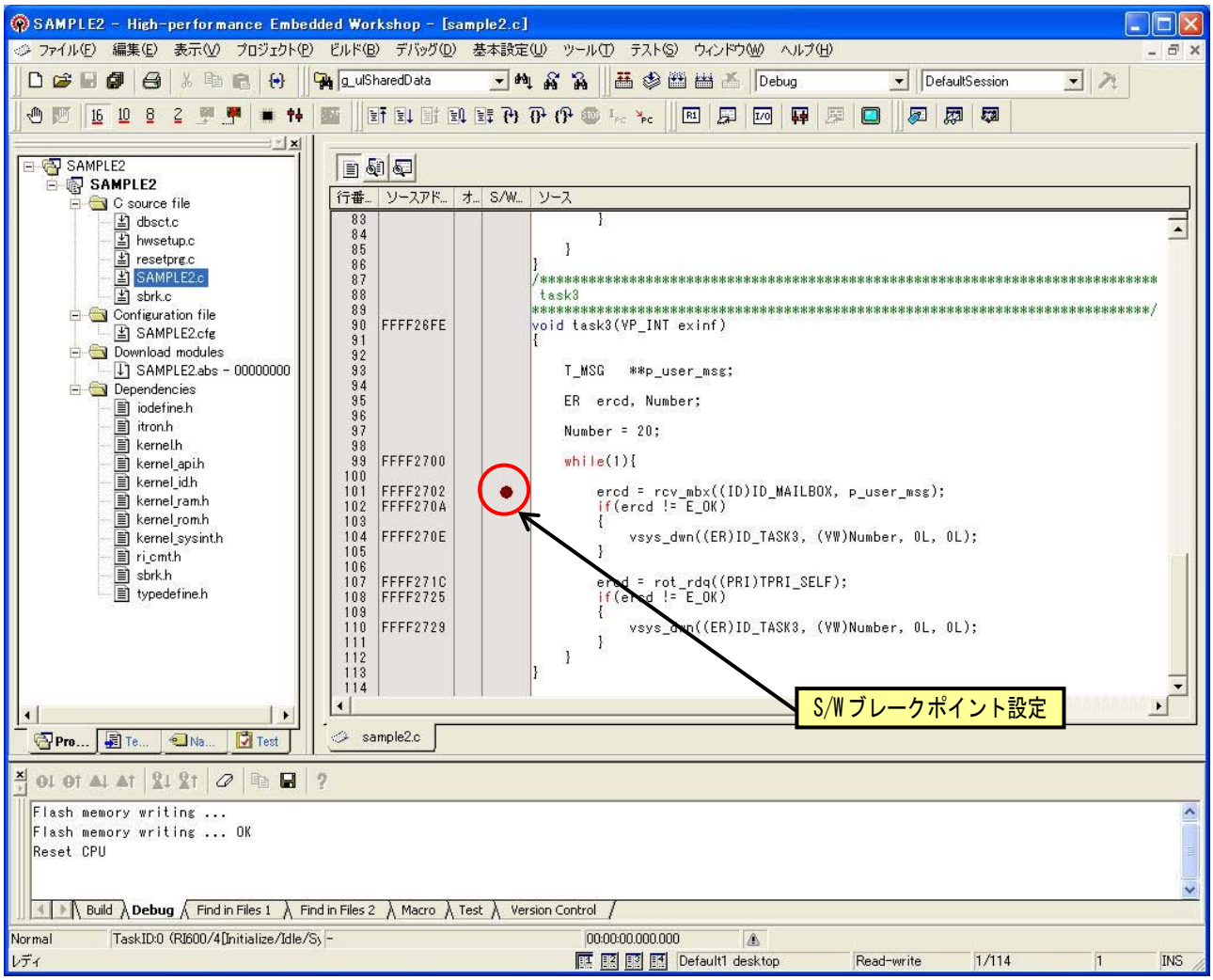


図 3-17 S/W ブレークポイント設定（SAMPLE2）

プログラムを実行すると、設定した S/W ブレークポイント(rcv_mbx サービスコール実行前)で停止しました。ここで OS オブジェクトウィンドウを開き、オブジェクトの情報を確認します。

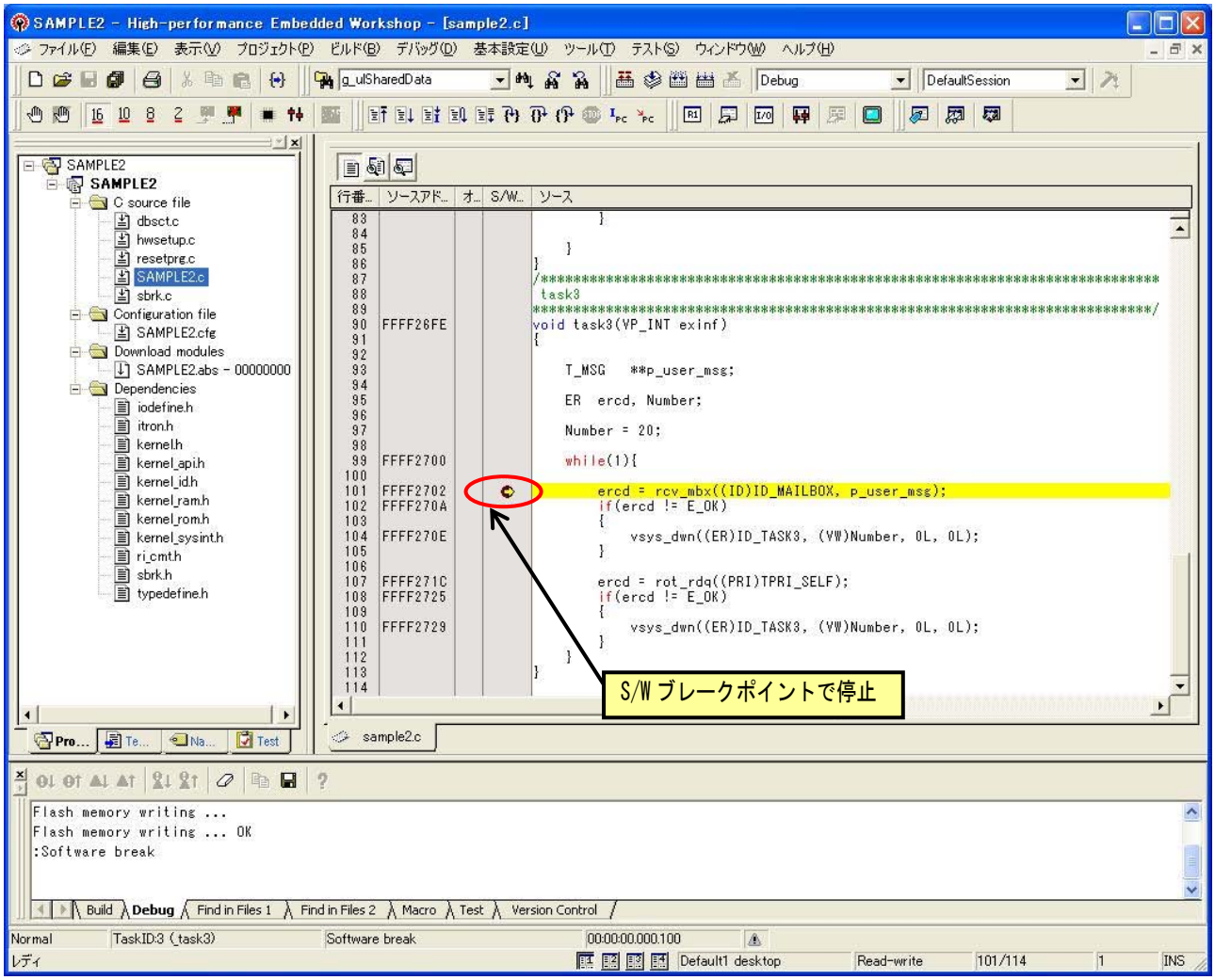


図 3-18　S/W ブレークポイント発生（SAMPLE2）

OS オブジェクトウィンドウを開いたら、指定したメールボックス（ID_MAILBOX）のメッセージアドレスが確認できるように設定します。

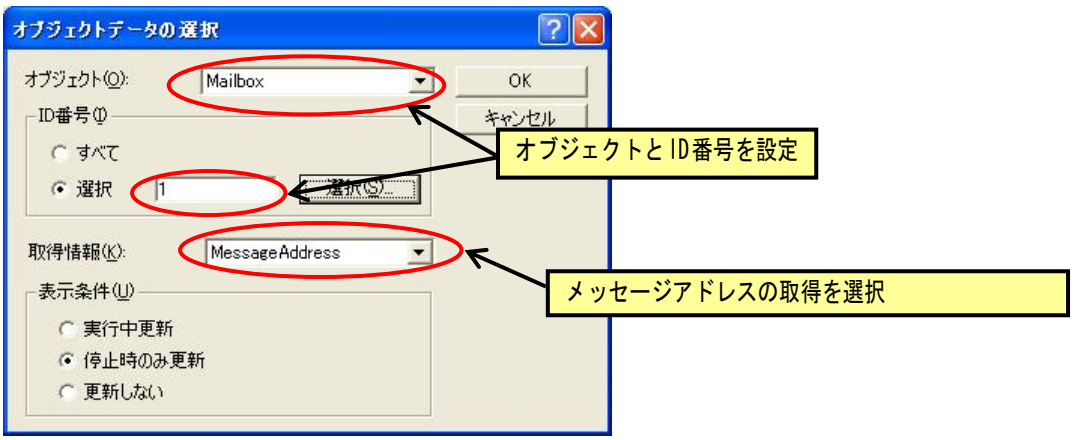


図 3-19　ID_MAILBOX のメッセージアドレス追加（SAMPLE2）

続いて、task2 の snd_msg サービスコールで送信したメッセージを確認します。送信したメッセージのアドレスを確認するためにラベルウィンドウを開きます。また、受け取ったメッセージの内容を確認するためにメモリウィンドウを開きます（ラベルウィンドウは、HEW の[表示]－[シンボル]－[ラベル]で開き、メモリウィンドウは、HEW の[表示]－[CPU]－[メモリ]で開きます）。

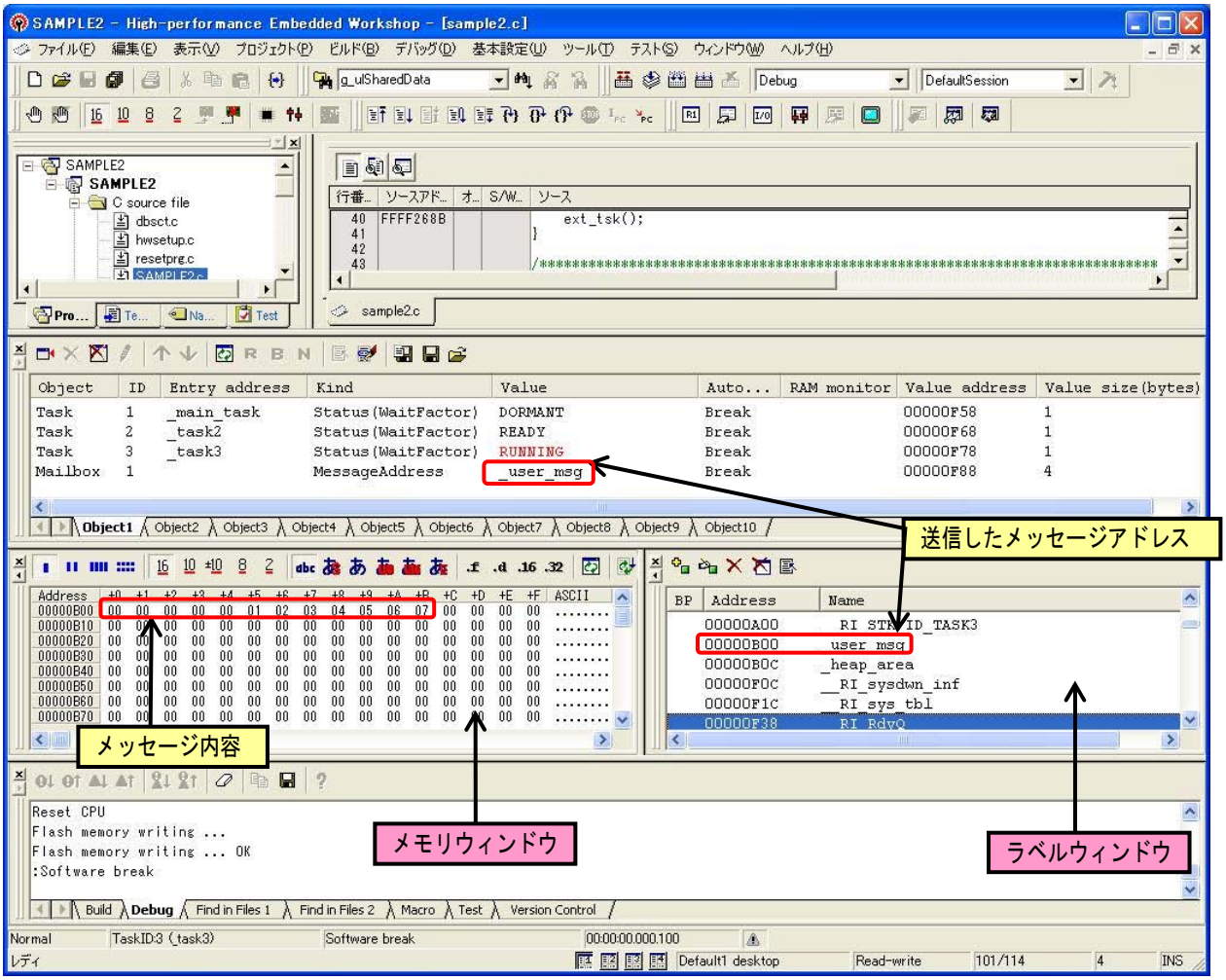


図 3-20　送信メッセージの確認（SAMPLE2）

OS オブジェクトウィンドウでは送信メッセージのアドレスのシンボル名を確認します。ラベルウィンドウでシンボル名 : _user_msg のアドレスが H’00000b00 であることが確認できました。メモリウィンドウでアドレスの内容を確認し、送信メッセージの内容が H’01234567 であることが分かりました。

3.3.3 SAMPLE3：固定長メモリプールの空きメモリブロック数を確認する

SAMPLE3 では、固定長メモリプールとメールボックスを使用して下記のように動作するプログラムを作成しました。。

(a) main_task で、task2、task3 を起動します。

(b) task2 よりタスク優先度が高い task3 が起動されます。

(c) task3 は、メッセージ受信待ち状態に遷移します。

(d) task2 は、固定長メモリプールから一つメモリブロックを獲得し、メッセージとしてメールボックスに送信します。

(e) task3 のメッセージ受信待ち状態が解除されます。メッセージを受信した後、メッセージ領域を固定長メモリプールに返却します。

■使用するオブジェクト

表 3-5 タスク（SAMPLE3）

アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	3	固定長メモリプールからメモリブロックを獲得し、メッセージとして送信
task3	3	2	メッセージを受信し、その領域を固定長メモリプールに返却

表 3-6 固定長メモリプール（SAMPLE3）

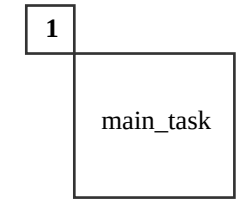
名称	ID 番号	ブロック数	ブロックサイズ	属性	セクション
ID_MPF1	1	2	256	TA_TFIFO	BRI_HEAP

表 3-7 メールボックス（SAMPLE3）

名称	ID 番号	属性
ID_MAILBOX1	1	TA_TFIFO TA_MFIFO

このサンプルプログラムでは、task2 が固定長メモリプールからメモリブロックを獲得して、メッセージとしてメールボックスに送信しますが、メモリブロック獲得後の空きのメモリブロック数を確認する方法を説明します。

ソースコードは、以下のように記述しています。



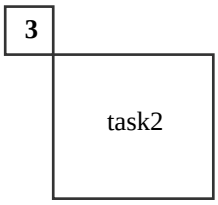
カーネル提供ファイル
cfg600 生成ファイル

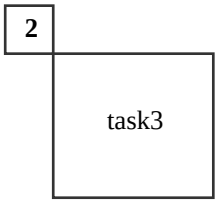
task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

```
/***********************************************************************
*
*     RI600/4 Sample program
*
***********************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/***********************************************************************
  main_task
***********************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

図 3-21　サンプルプログラムソースコード『SAMPLE3.c』(1/2)





メモリブロック獲得
（get_mpf）

メールボックスへ送信
（snd_mbx）

メールボックスから受信
（rcv_mbx）

固定長メモリブロック返却
（rel_mpf）

```
/************************************************************************
 task2
************************************************************************/
void task2(VP_INT exinf)
{
	ER	ercd, Number;
	VP	*p_blk;

	Number = 10;

	while(1){

		ercd = get_mpf((ID)ID_MPF1, (VP)&p_blk);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
		}

		ercd = snd_mbx((ID)ID_MAILBOX1, (T_MSG *)p_blk);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
		}

	}
}

/************************************************************************
 task3
************************************************************************/
void task3(VP_INT exinf)
{

	T_MSG	*p_user_msg;

	ER	ercd, Number;

	Number = 20;

	while(1){

		ercd = rcv_mbx((ID)ID_MAILBOX1, (T_MSG **)&p_user_msg);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
		}

		ercd = rel_mpf((ID)ID_MPF1, (VP)p_user_msg);
		if(ercd != E_OK)
		{
			vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
		}
	}
}
```

図 3-21　サンプルプログラムソースコード『SAMPLE3.c』(2/2)

(2) デバッグ開始

task2 で固定長メモリプールのメモリブロック獲得後の残りメモリブロック数を確認するため、E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能を使用します。

まず、task2 の get_mpf サービスコールでメモリブロックを獲得した後に S/W ブレークポイントを設定します。

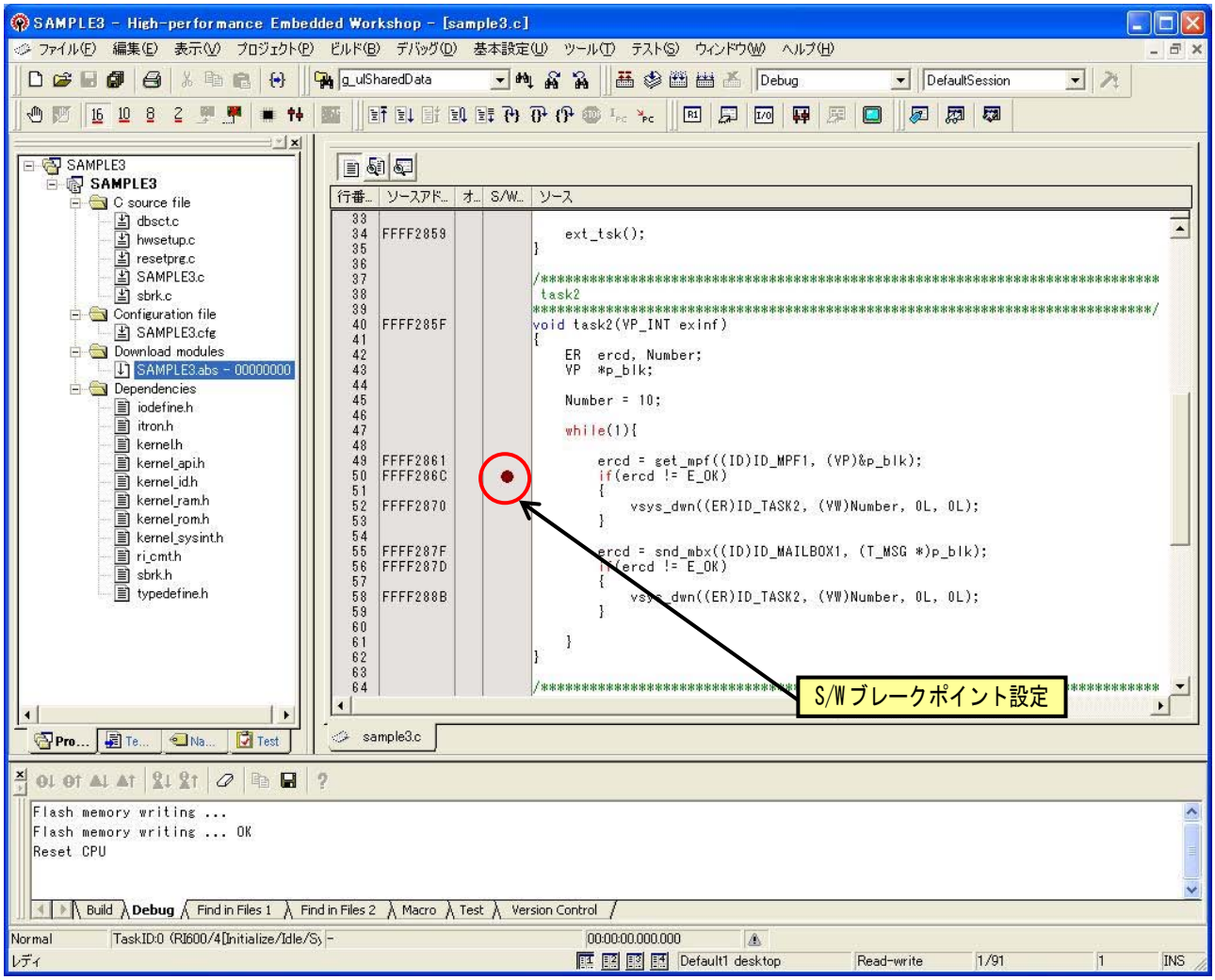


図 3-22 S/W ブレークポイント設定（SAMPLE3）

プログラムを実行すると、設定した S/W ブレークポイント（get_mpf サービスコールの実行後）で停止しました。ここで OS オブジェクトウィンドウを開き、オブジェクトの情報を確認します。

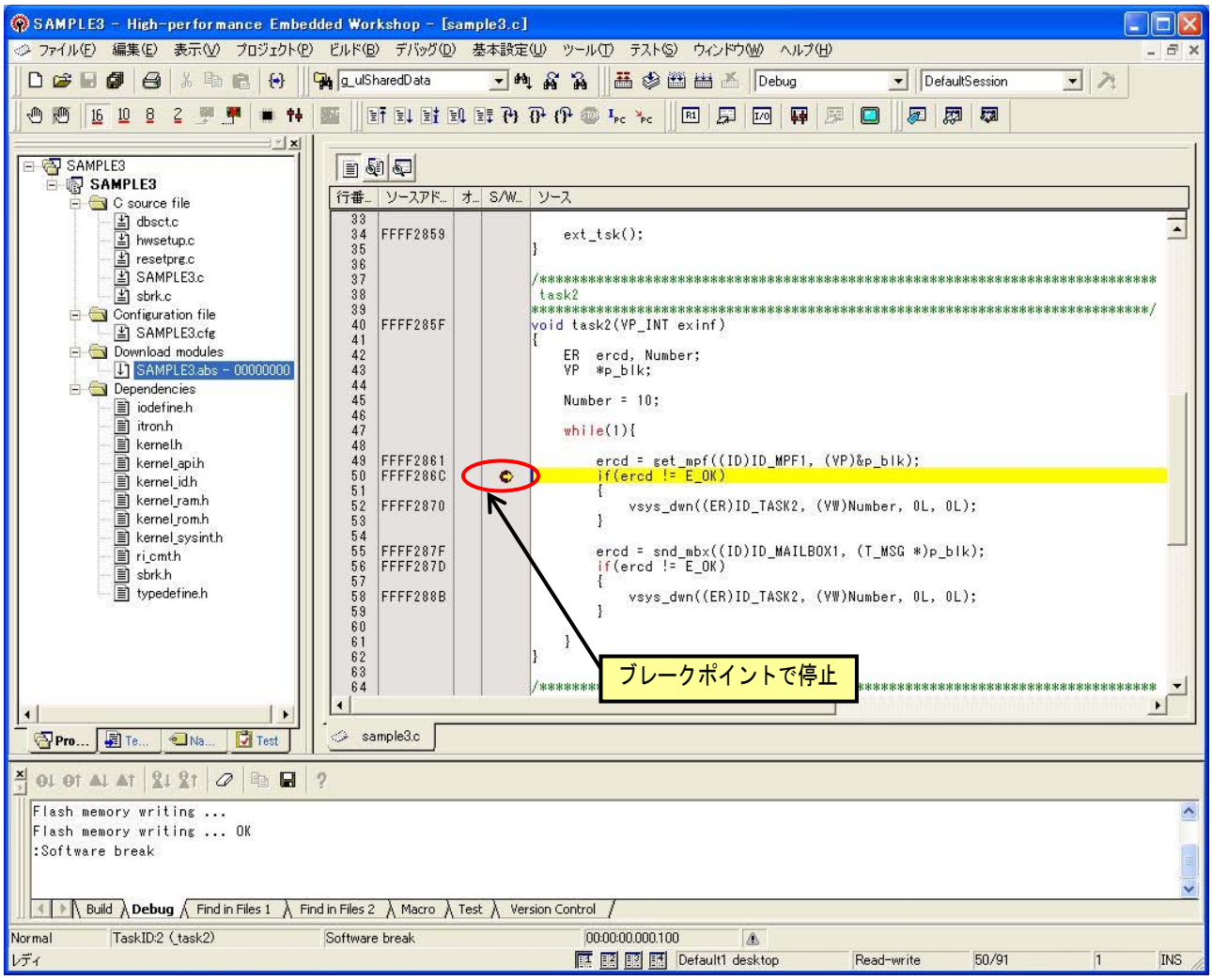


図 3-23　S/W ブレークポイント発生（SAMPLE3）

OS オブジェクトウィンドウを開いたら、指定した固定長メモリプール（ID_MPF1）の空きブロック数が確認できるように設定します。

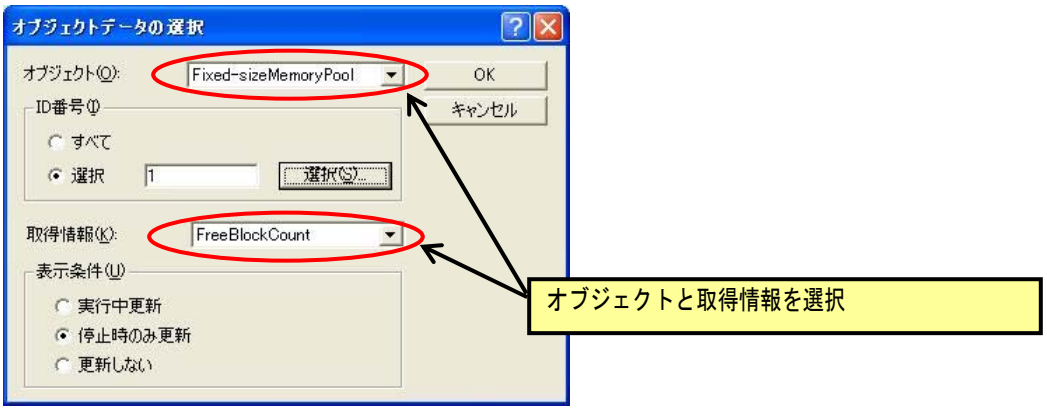


図 3-24　ID_MPF1 の空きブロック数追加（SAMPLE3）

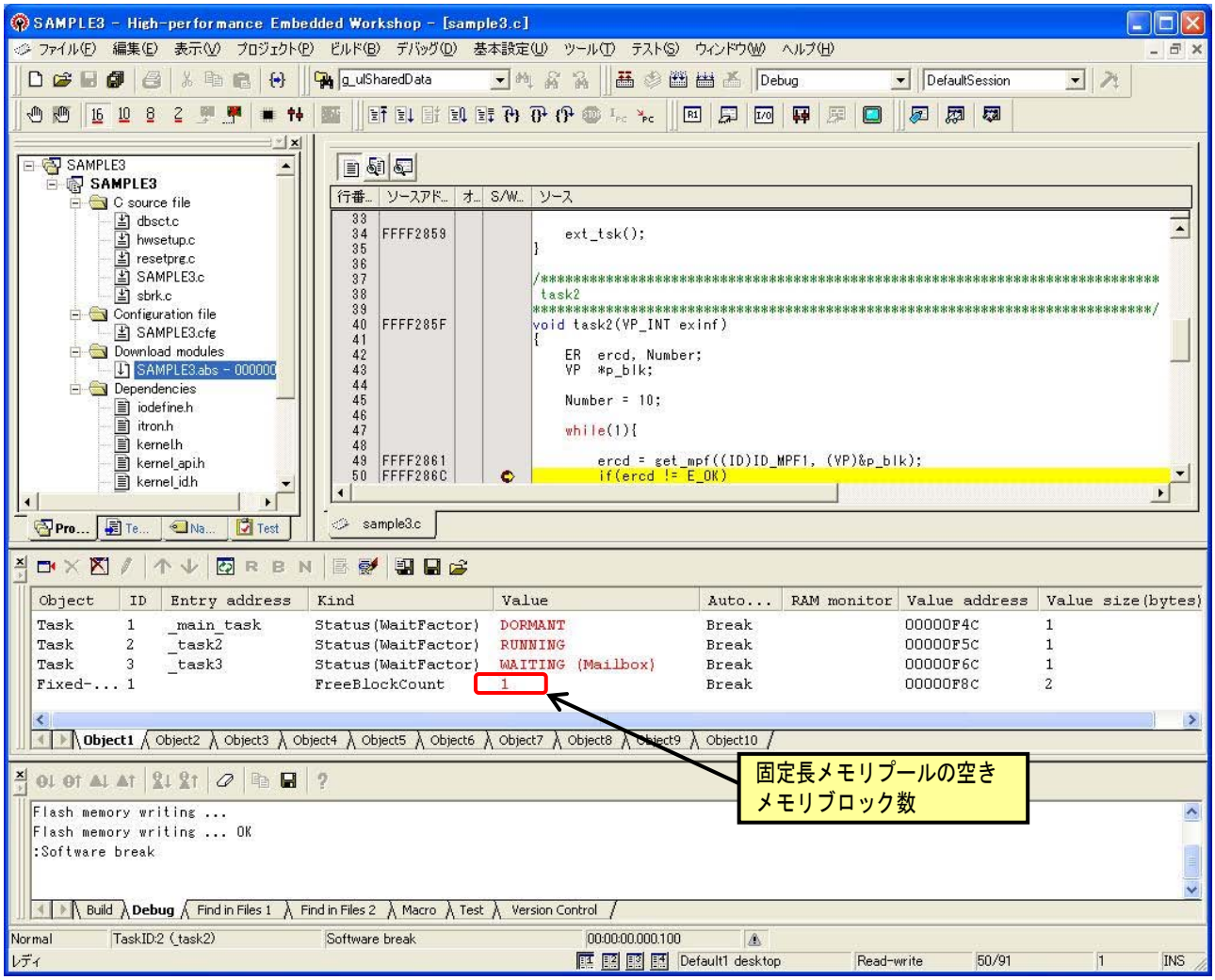


図 3-25　固定長メモリプールの空きメモリブロック数確認（SAMPLE3）

空きメモリブロックは 1 つあることが分かりました。

3.3.4 SAMPLE4：タスクが想定しない順序で実行した時のデバッグ

SAMPLE4 では、タスクのスケジューリングが優先度順に行われない場合のデバッグ方法を説明します。サンプルプログラムは、下記のような動作を想定して作成しました。

(a) main_task で、task2、task3 を起動します。

(b) task2 よりタスク優先度が高い task3 が起動されます。

(c) task3 は、前に実行されたタスク ID を確認し、task2 に実行権を渡します。

(d) task2 が起動されます。

(e) task2 は、前に実行されたタスク ID を確認し、task3 に実行権を渡します。

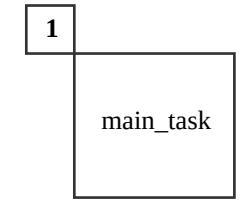
■使用するオブジェクト

表 3-8 タスク（SAMPLE4）

アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	3	前に実行中だったタスク ID の確認
task3	3	2	前に実行中だったタスク ID の確認

このサンプルプログラムでは、優先度の高い task3 が最初に実行状態に遷移するはずであるが、task2 が実行状態に遷移する、という問題の解決方法を説明します。

ソースコードは、以下のように記述しています。



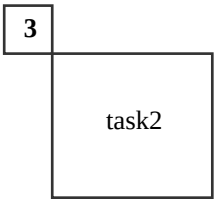
カーネル提供ファイル
cfg600 生成ファイル

```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

ID pre_tskid;
/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      extern ID pre_tskid;
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      pre_tskid = TSK_NONE;
      ext_tsk();
}
```

task2 起動
（sta_tsk）

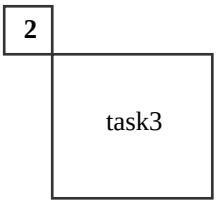
task3 起動
（sta_tsk）

図 3-26　サンプルプログラムソースコード『SAMPLE4.c』(1/3)




```
/**************************************************************************
  task2
**************************************************************************/
void task2(VP_INT exinf)
{

        extern ID pre_tskid;
        ER    ercd, Number;

        Number = 10;

        while(1){

                if(pre_tskid != (ID)ID_TASK3)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }

                pre_tskid = (ID)ID_TASK2;

                ercd = wup_tsk((ID)ID_TASK3);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }

                ercd = slp_tsk();
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-26　サンプルプログラムソースコード『SAMPLE4.c』(2/3)




```
/**********************************************************************************
  task3
**********************************************************************************/
void task3(VP_INT exinf)
{

        extern ID pre_tskid;
        ER    ercd, Number;

        Number = 20;

        if(pre_tskid != (ID)TSK_NONE)
        {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }

        Number++;
        while(1){

                pre_tskid = (ID)ID_TASK3;
                ercd = slp_tsk();
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
                if(pre_tskid != (ID)ID_TASK2)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
                ercd = wup_tsk((ID)ID_TASK2);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-26　サンプルプログラムソースコード『SAMPLE4.c』(3/3)

(2) デバッグ開始

task3 よりも優先度が低い task2 が先に実行状態が遷移されてしまう原因として、タスク優先度の設定に誤りがあることが考えられます。これを確認するため、E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能を使用してデバッグします。

まず、どのタスクが最初に実行状態になるのかを確認するため、task2 と task3 の先頭に S/W ブレークポイントを設定します。

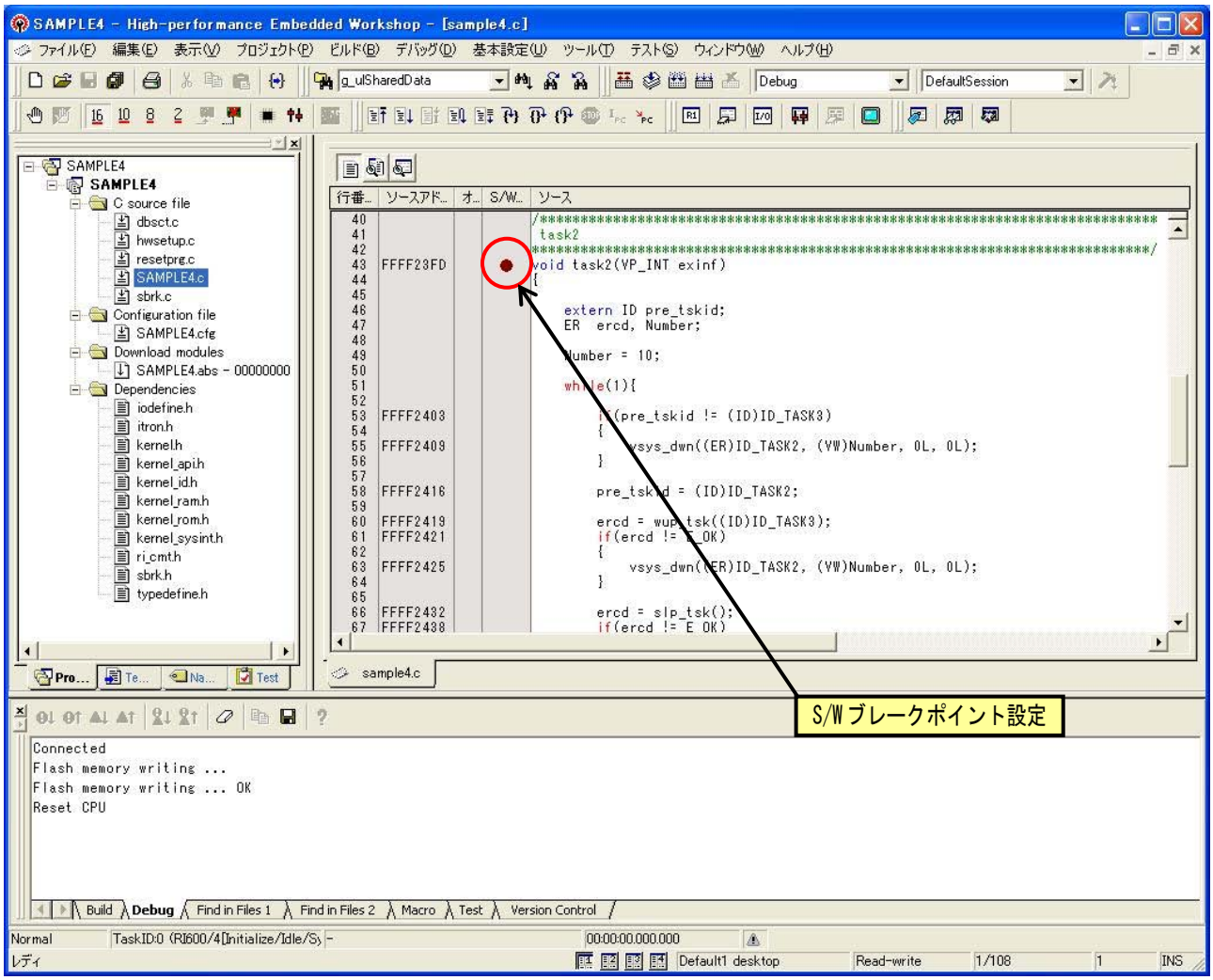


図 3-27 task2 への S/W ブレークポイント設定（SAMPLE4）

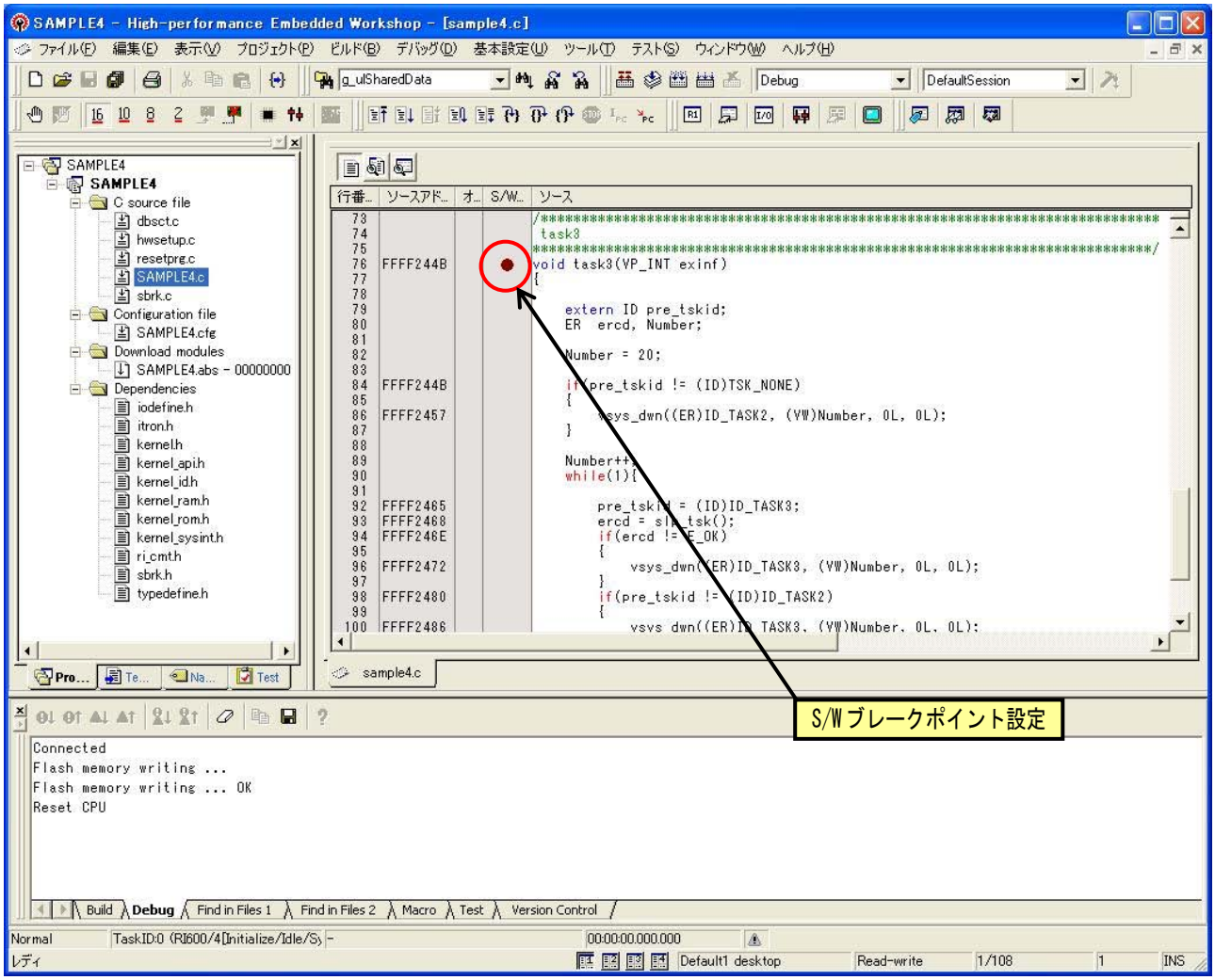


図 3-28　task3 への S/W ブレークポイント設定（SAMPLE4）

プログラムを実行してみると、設定したブレークポイント(task2　先頭行)で停止しました。ここで OS オブジェクトウィンドウを開き、オブジェクトの情報を確認します。

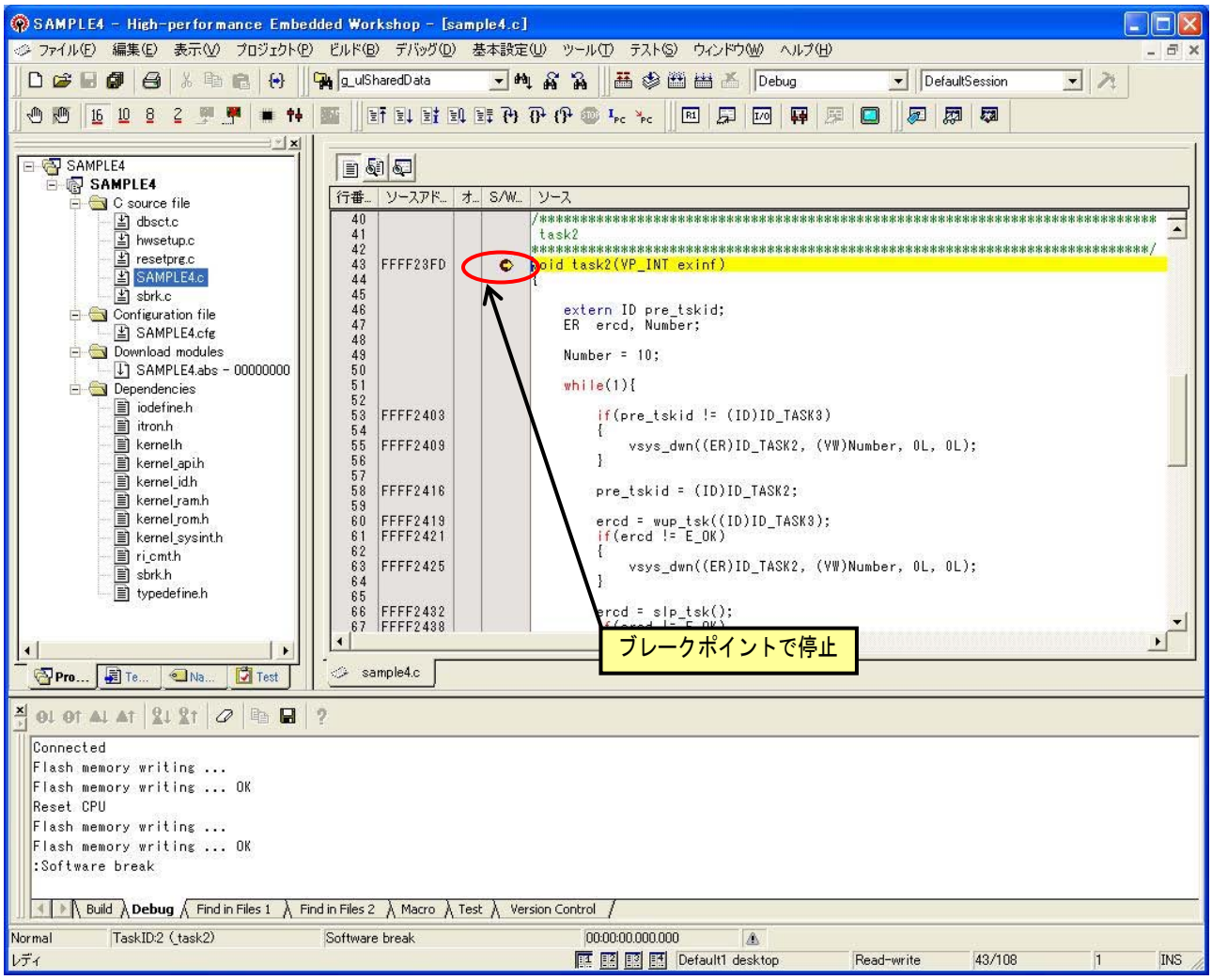


図 3-29　S/W ブレークポイント発生（SAMPLE4）

OS オブジェクトウィンドウを開き、オブジェクトデータの選択ウィンドウで全タスクの優先度を表示させるように設定します。

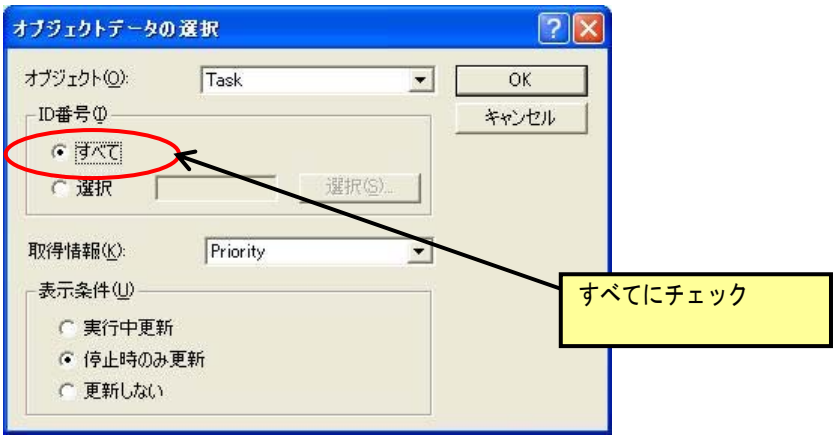


図 3-30　全タスクの優先度追加（SAMPLE4）

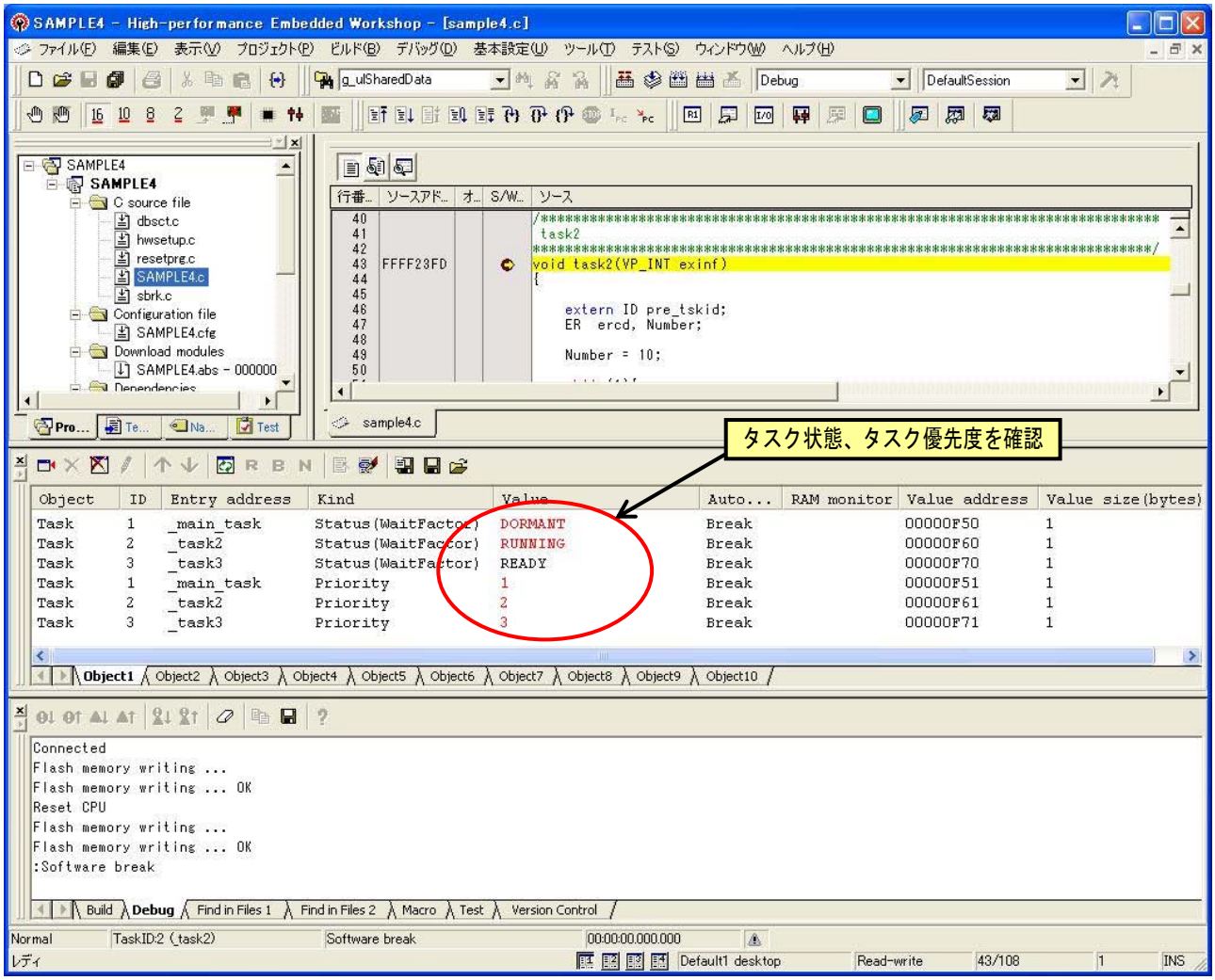


図 3-31　タスク状態とタスク優先度の確認（SAMPLE4）

タスク状態とタスク優先度を確認すると、task3 が最初に実行状態になるためには、task2 と task3 のタスク優先度を表 3-8 で定めた値に変更すれば良いことが分かります。コンフィギュレーションファイルを見直し、修正する必要があります。

3.3.5 SAMPLE5：イベント発生の周期時間確認

SAMPLE5 は、周期ハンドラを使用してタスクが周期的に起床するプログラムです。

(a) main_task で、task2 を起動します。

(b) task2 は周期ハンドラからの起床要求が発行されるまで、待ち状態に遷移します。

(c) 周期ハンドラの起動時間(50ms)が経過し、cychdr1 が起動されます。

(d) cychdr1 は、task2 に対して起床要求を発行し、task2 の待ち状態を解除します。

■使用するオブジェクト

表 3-9 タスク（SAMPLE5）

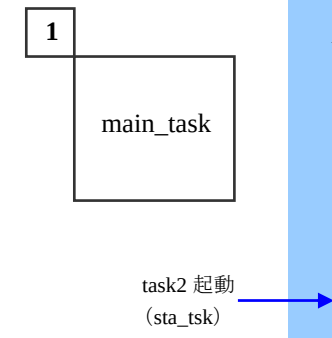
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 を起動
task2	2	2	周期ハンドラに起床されるまで起床待ち状態に遷移

表 3-10 周期ハンドラ（SAMPLE5）

名称	ID 番号	周期時間	起動位相	属性
ID_CYCHDR1	1	100msec	50msec	TA_STA TA_PHS

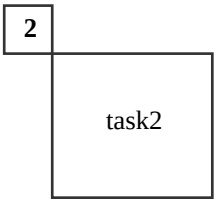
このサンプルプログラムでは、cychdr1（周期ハンドラ）で task2 を起床してから、次に cychdr1 で task2 が起床されるまでの時間を確認する方法を説明します。

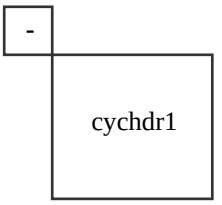
ソースコードは、以下のように記述しています。




```
/*************************************************************************
*
*     RI600/4 Sample program
*
*************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/*************************************************************************
  main_task
*************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2, (VP_INT)stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

図 3-32　サンプルプログラムソースコード『SAMPLE5.c』(1/2)





起床待ち
（slp_tsk）

task2 起床
（iwup_tsk）

```
/*****************************************************************************
 task2
*****************************************************************************/
void task2(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 10;

    while(1){

        ercd = slp_tsk();
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
    }
}

/*****************************************************************************
 cychdr1
*****************************************************************************/
void cychdr1(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 20;

    ercd = iwup_tsk((ID)ID_TASK2);
    if(ercd != E_OK)
    {
        vsys_dwn((ER)ID_CYCHDR1, (VW)Number, 0L, 0L);
    }
}
```

図 3-32　サンプルプログラムソースコード『SAMPLE5.c』(2/2)

(2) デバッグ開始

cychdr1 に task2 の起床待ち状態が解除されてから次の task2 の起床待ち状態解除までの残り時間を確認するため、E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能を使用します。

まず、task2 の slp_tsk サービスコール実行後の次の命令に S/W ブレークポイントを設定します。

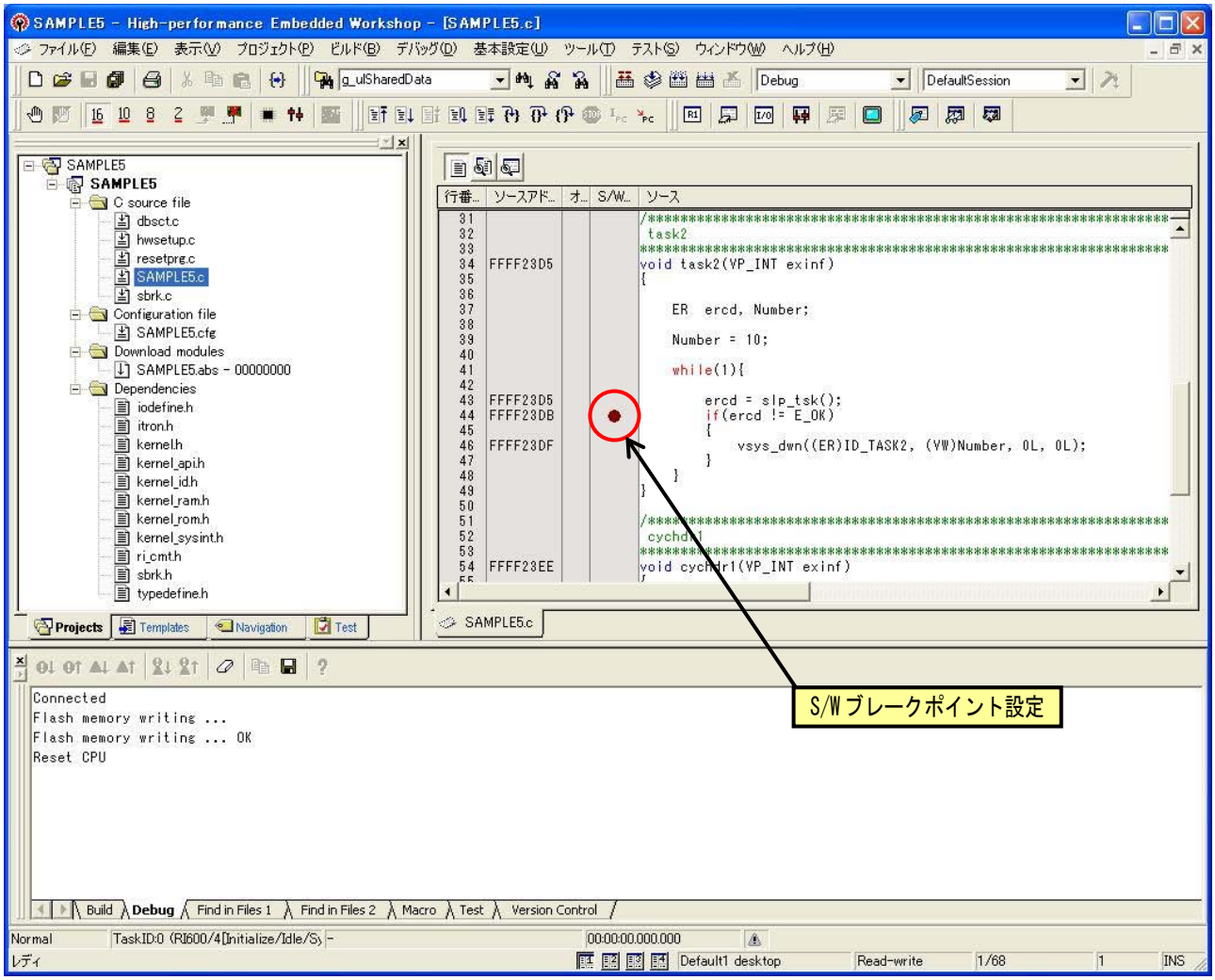


図 3-33 S/W ブレークポイント設定（SAMPLE5）

プログラムを実行すると、設定した S/W ブレークポイント(slp_tsk サービスコール実行後)で停止しました。ここで OS オブジェクトウィンドウを開き、オブジェクトの情報を確認します。

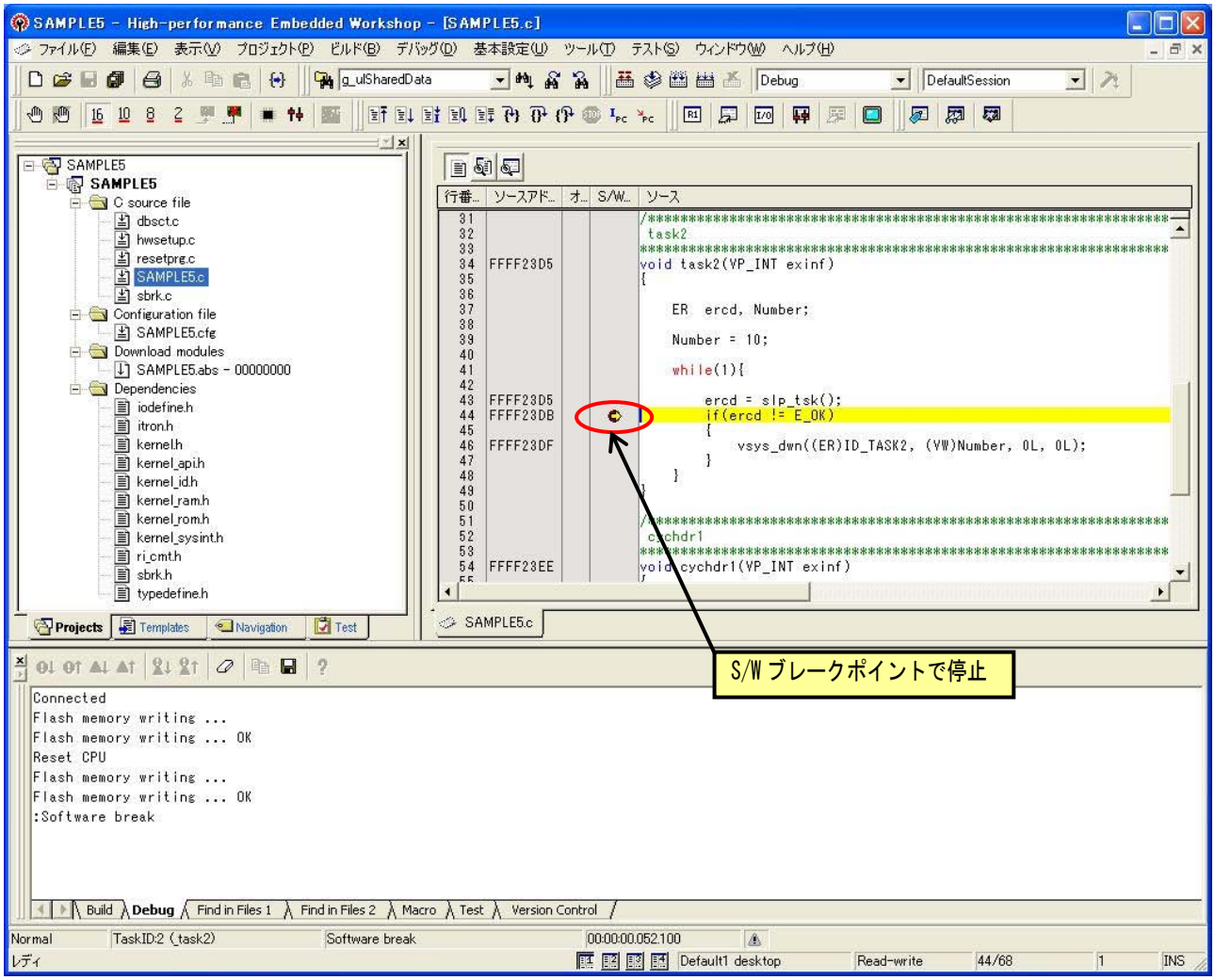


図 3-34　S/W ブレークポイント発生（SAMPLE5）

OS オブジェクトウィンドウを開き、オブジェクトデータの選択ウィンドウで周期ハンドラ ID_CYCHDR1 の起動までの残り時間が取得できるように設定します。

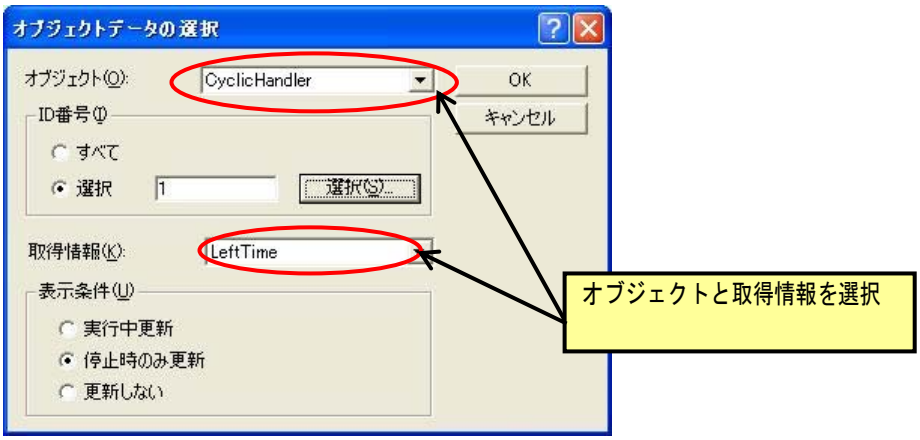


図 3-35　ID_CYCHDR1 の起動までの残時間追加（SAMPLE5）

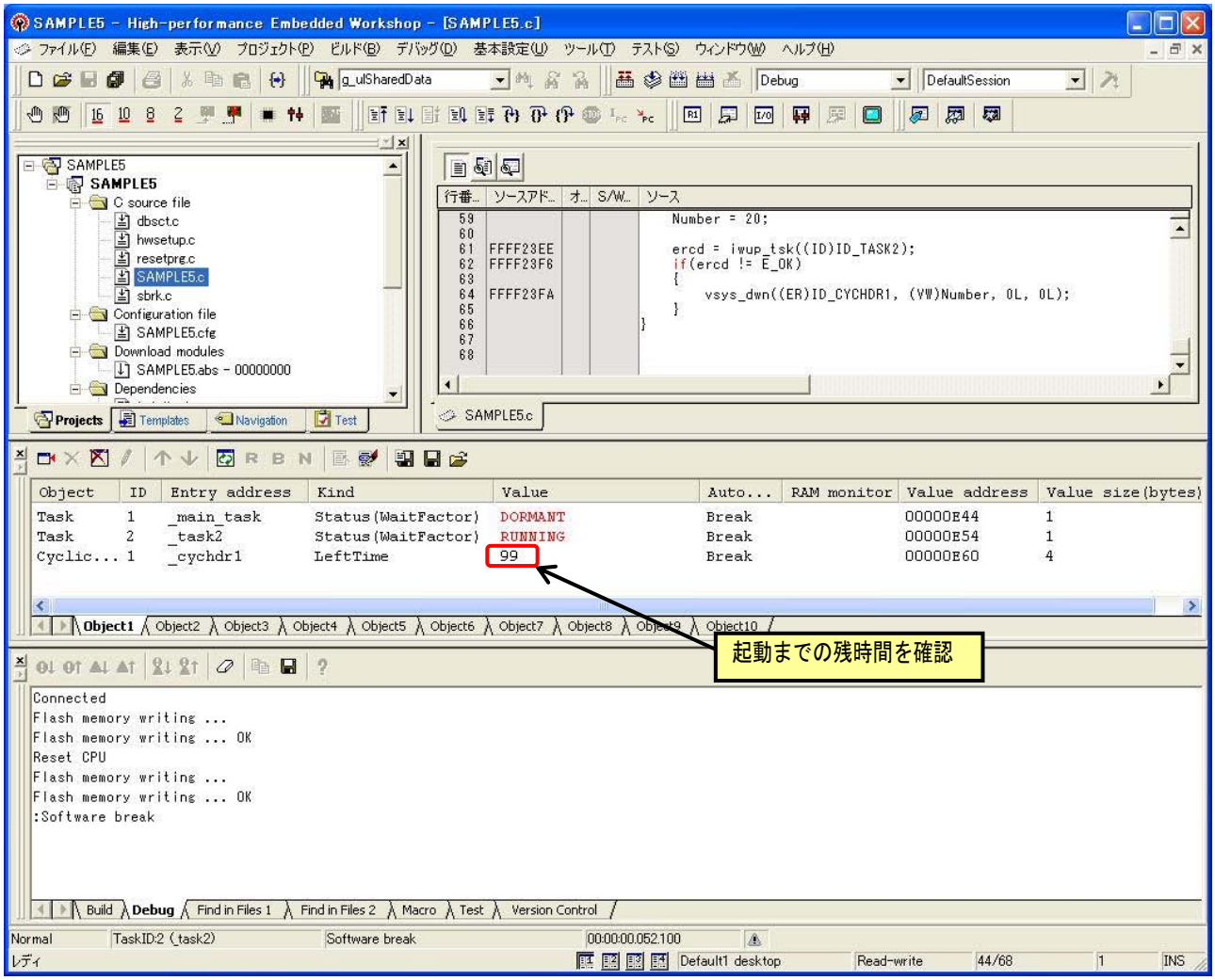


図 3-36　OS オブジェクトの状態確認（SAMPLE5）

周期ハンドラが起動されるまでの残時間は 99msec であることが分かりました。ID_CYCHDR1 の周期時間は 100ms と設定しましたが、結果が 99ms になっているのは、OS で以下の処理を行っているために 1ms 経過しています。

(a)　周期ハンドラを実行する

(b)　周期ハンドラから発行される iwup_tsk サービスコールの処理

(c)　周期ハンドラを終了する

(d)　task2 の起床（タスク切り替え処理）

ただし、周期時間がミリ秒単位での設定しかできないため、実際にはこの処理が 1ms かかっていない場合でも、1ms 経過したとして、このサンプルプログラムのように 99msec に変化することがあるので注意してください。

3.3.6 SAMPLE6：セマフォ資源獲得に関する不正動作のデバッグ

SAMPLE6 は、セマフォを使用して同期・通信を行うプログラムです。

(a) main_task で、task2、task3 を起動します。

(b) task2 が起動されます。

(c) task2 は wai_sem サービスコールでセマフォ資源をひとつ獲得しようとしますが、対象セマフォ資源数が 0 のため、待ち状態に遷移します。

(d) task3 が起動されます。

(e) task3 の sig_sem サービスコールで対象セマフォの資源数に 1 を加えます。これによって task2 は資源を獲得して待ち状態を解除します。

■使用するオブジェクト

表 3-11 タスク（SAMPLE6）

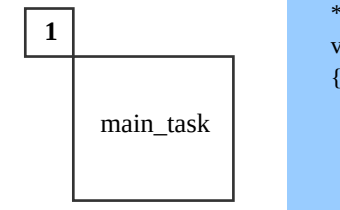
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	セマフォ獲得
task3	3	3	セマフォ返却

表 3-12 セマフォ（SAMPLE6）

名称	ID 番号	最大資源数	初期資源数	属性
ID_SEMAPHORE1	1	1	0	TA_TFIFO

このサンプルプログラムを動作させたところ、task2 の wai_sem サービスコールでセマフォ資源待ち状態に遷移するはずが、待ち状態にならず、セマフォ資源の獲得待ちが解除されてしまいました。

ソースコードは、以下のように記述しています。



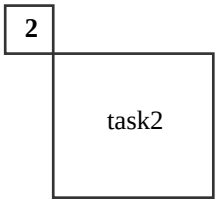
カーネル提供ファイル
cfg600 生成ファイル

task2 起動
（sta_tsk）

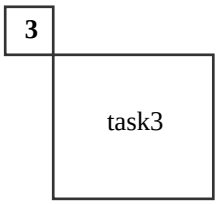
task3 起動
（sta_tsk）

```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/*****************************************************************************
  main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

図 3-37　サンプルプログラムソースコード『SAMPLE6.c』(1/2)



セマフォ資源獲得
（wai_sem）



セマフォ資源返却
（sig_sem）

```
/**************************************************************************
 task2
**************************************************************************/
void task2(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 10;

    while(1){

        ercd = wai_sem((ID)ID_SEMAPHORE1);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
    }
}

/**************************************************************************
 task3
**************************************************************************/
void task3(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 20;

    while(1){

        ercd = sig_sem((ID)ID_SEMAPHORE1);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
    }
}
```

図 3-37　サンプルプログラムソースコード『SAMPLE6.c』(2/2)

(2) デバッグ開始

task2 が wai_sem サービスコールを発行してセマフォ資源待ち状態に遷移しない原因として、すでにセマフォ資源を獲得していることが考えられます。これを確認するため、E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能を使用してデバッグします。

まず、図 3-38 に示すように、E1 エミュレータの S/W ブレークを task2 の wai_sem サービスコールと task3 の sig_sem サービスコールの実行前に設定します。

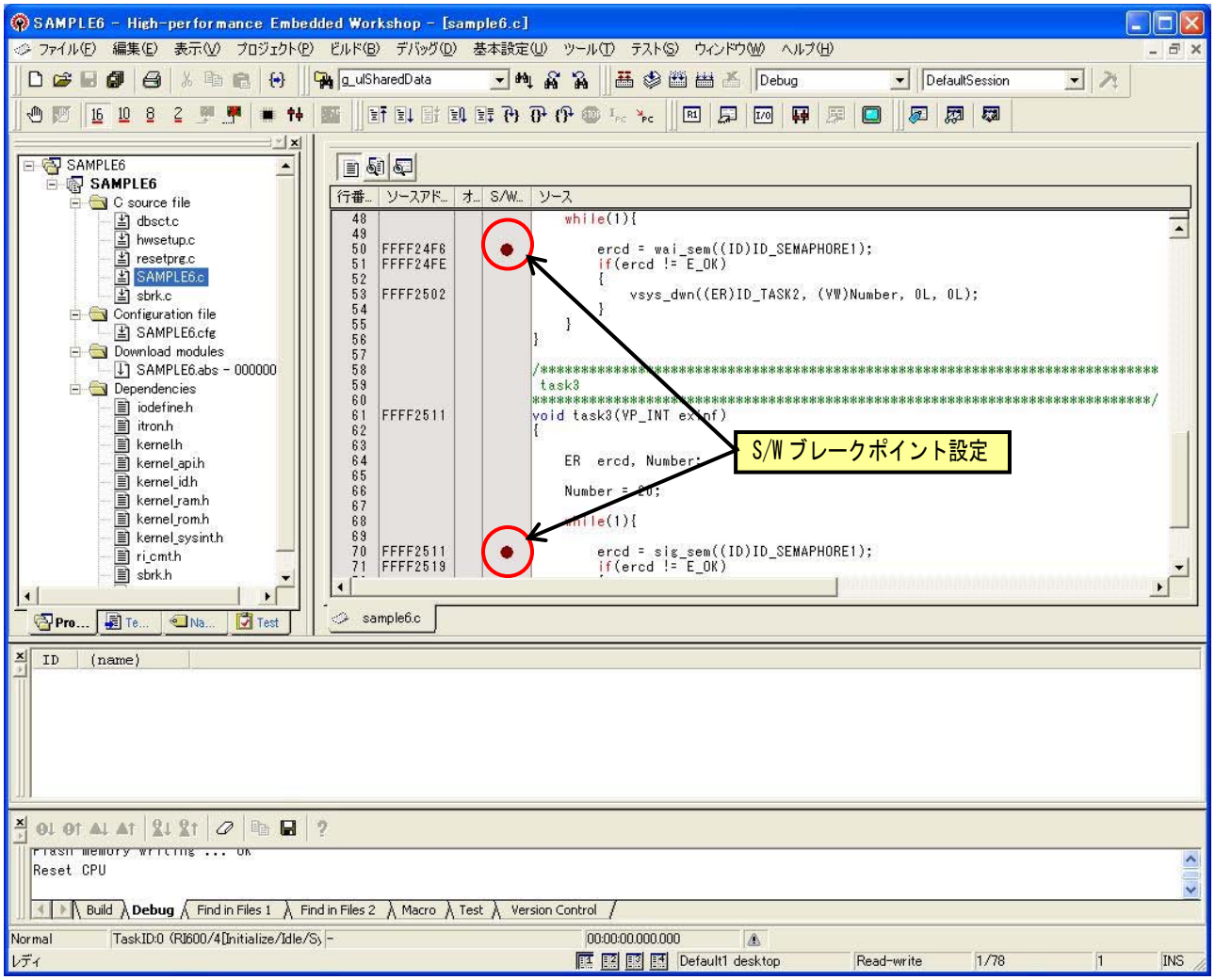


図 3-38 S/W ブレークポイント設定（SAMPLE6）

また、図 3-39 に示すように、ブレークした時の対象セマフォの状態を確認するため、OS オブジェクトウィンドウで対象セマフォの状態を取得できるように設定します。

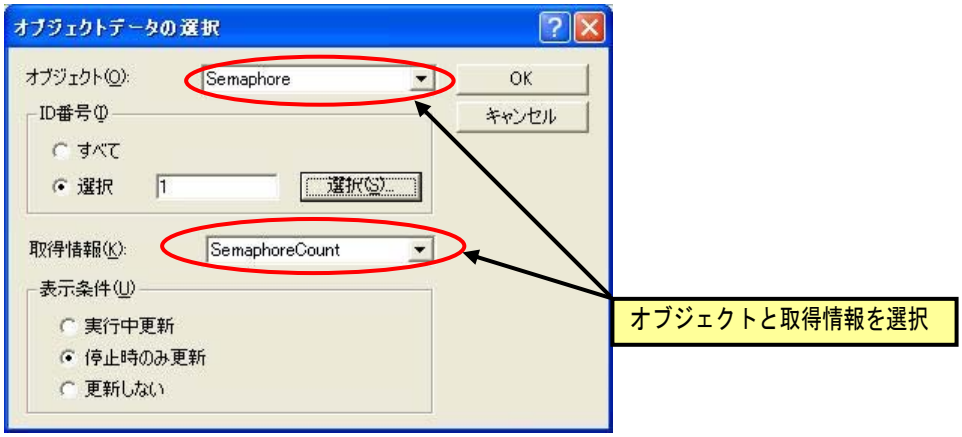


図 3-39　セマフォの追加（SAMPLE6）

これでプログラムを実行させると、task2 の wai_sem サービスコールのところに設定した S/W ブレークポイントで停止しました。

図 3-40 でオブジェクトウィンドウの情報を見てみると、対象セマフォ資源数が、task2 の wai_sem サービスコール実行前で、すでに“1”になっていました。つまり想定していた動作概要では、セマフォの初期資源数を 0 としていましたが、実際には 1 に設定されていたため、task2 はセマフォを獲得し、待ち状態にならなかったことが分かりました。

task2 の wai_sem サービスコールで待ち状態に遷移させるには、コンフィギュレータでセマフォを生成する際に、設定した初期資源数が間違っていたことが分かりました。

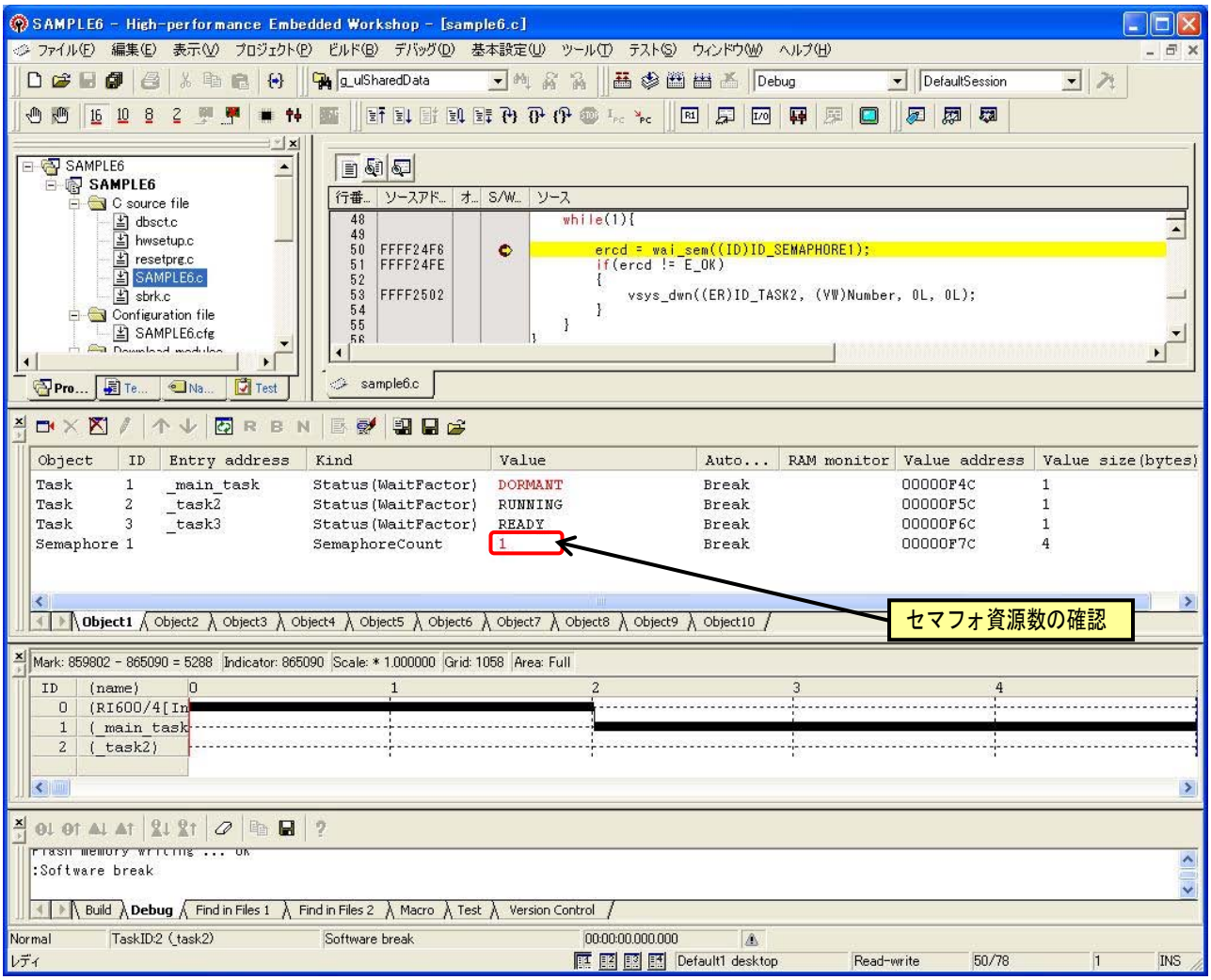


図 3-40　OS オブジェクトの状態確認（SAMPLE6）

3.3.7 SAMPLE7:デッドロック状態のデバッグ

SAMPLE7 は、セマフォを使用したプログラムです。

このサンプルプログラムは、task2、task3 共にセマフォの待ち状態となり、どちらのタスクも実行状態へ遷移できない状態です。このような状態をデッドロックと言います。

(a) main_task で、task2、task3 を起動します。

(b) task2 が起動されます。

(c) task2 は、ID_SEMAPHORE1 を獲得し、実行権を task3 に渡します。

(d) task3 が起動されます。

(e) task3 は、ID_SEMAPHORE2 を獲得し、実行権を task2 に戻します。

(f) task2 は、ID_SEMAPHORE2 を獲得し、実行権を task3 に戻します。

(g) task3 は、ID_SEMAPHORE1 を獲得し、実行権を task2 に戻します。

■使用するオブジェクト

表 3-13 タスク(SAMPLE7)

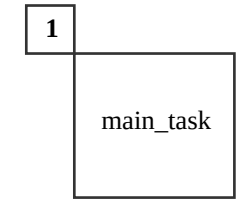
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	ID_SEMAPHORE1、2 を獲得する
task3	3	3	ID_SEMAPHORE2、1 を獲得する

表 3-14 セマフォ(SAMPLE7)

名称	ID 番号	最大資源数	初期資源数	属性
ID_SEMAPHORE1	1	1	1	TA_TFIFO
ID_SEMAPHORE2	2	1	1	TA_TFIFO

なぜ、このサンプルプログラムでデッドロック状態に陥ってしまうのかについて説明します。

ソースコードは、以下のように記述しています。



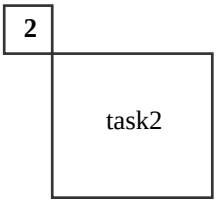
カーネル提供ファイル
cfg600 生成ファイル

task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

```
/************************************************************************
*
*     RI600/4 Sample program
*
************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/************************************************************************
  main_task
************************************************************************/
void main_task(VP_INT exinf)
{
        ER    ercd, Number;
        VP_INT    stacd;

        stacd = 0;
        Number = 0;

        ercd = sta_tsk((ID)ID_TASK2,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
        Number++;

        ercd = sta_tsk((ID)ID_TASK3,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }

        ext_tsk();
}
```

図 3-41　サンプルプログラムソースコード『SAMPLE7.c』(1/3)

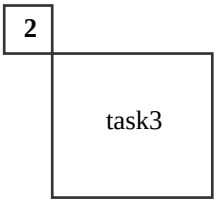


セマフォ資源獲得
（wai_sem）

セマフォ資源獲得
（wai_sem）

```
/*****************************************************************************
 task2
*****************************************************************************/
void task2(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 10;

    while(1){

        ercd = wai_sem((ID)ID_SEMAPHORE1);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);

        }
        Number++;

        ercd = rot_rdq((PRI)TPRI_SELF);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);

        }

        Number++;
        ercd = wai_sem((ID)ID_SEMAPHORE2);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }

        Number++;
        ercd = rot_rdq((PRI)TPRI_SELF);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }

    }
}
```

図 3-41　サンプルプログラムソースコード『SAMPLE7.c』(2/3)



セマフォ資源獲得
（wai_sem）

セマフォ資源獲得
（wai_sem）

```
/*****************************************************************************
 task3
*****************************************************************************/
void task3(VP_INT exinf)
{

    ER    ercd, Number;

    Number = 20;

    while(1){

        ercd = wai_sem((ID)ID_SEMAPHORE2);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);

        }
        Number++;

        ercd = rot_rdq((PRI)TPRI_SELF);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);

        }

        Number++;
        ercd = wai_sem((ID)ID_SEMAPHORE1);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }

        Number++;
        ercd = rot_rdq((PRI)TPRI_SELF);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }

    }
}
```

図 3-41　サンプルプログラムソースコード『SAMPLE7.c』(3/3)

(2) デバッグ開始

E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能を使用してデバッグします。

まず、セマフォの獲得状況を確認するため、task2 と task3 で wai_sem サービスコールを実行した直後で S/W ブレークポイントを設定します。

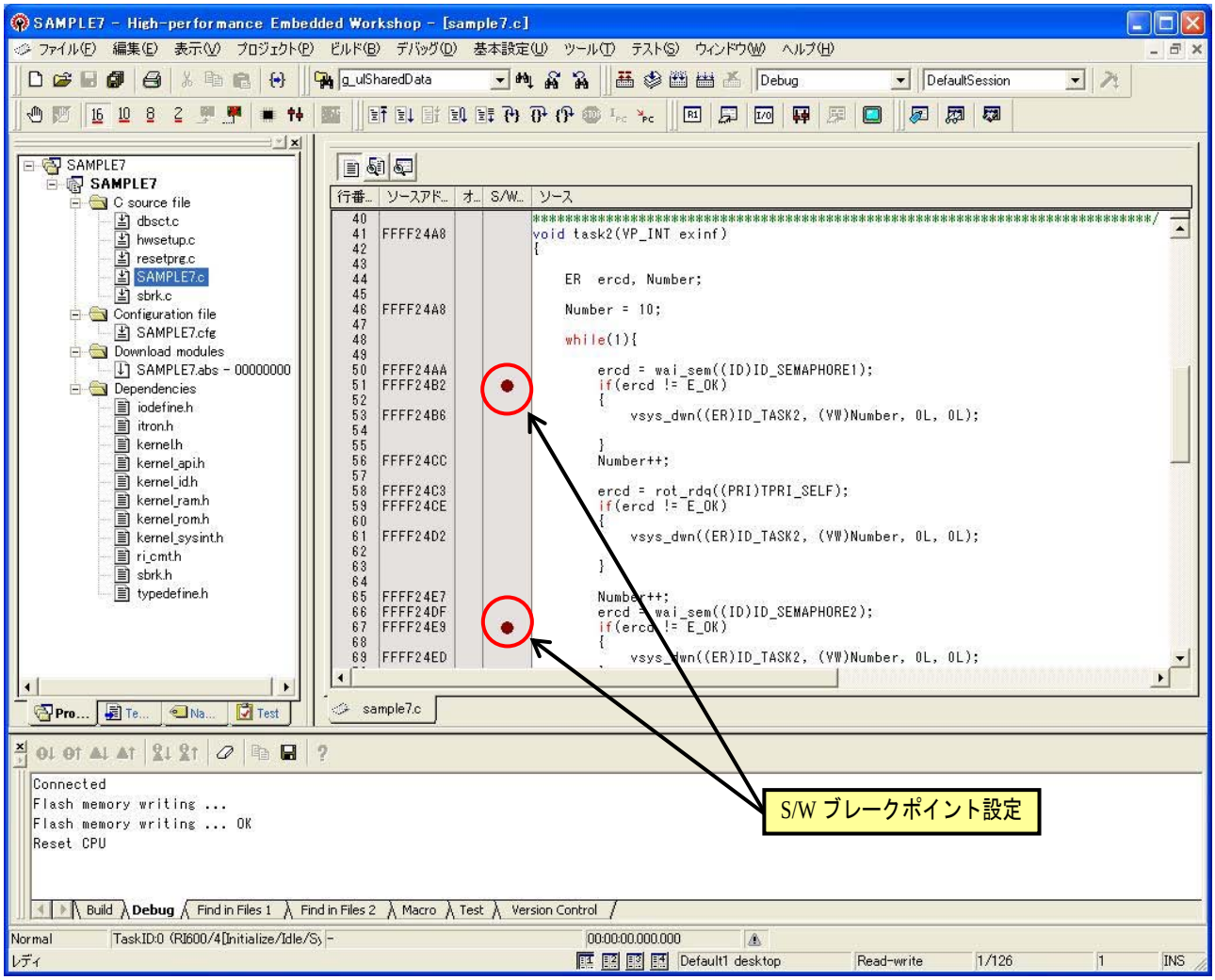


図 3-42 task2 S/W ブレークポイント設定（SAMPLE7）

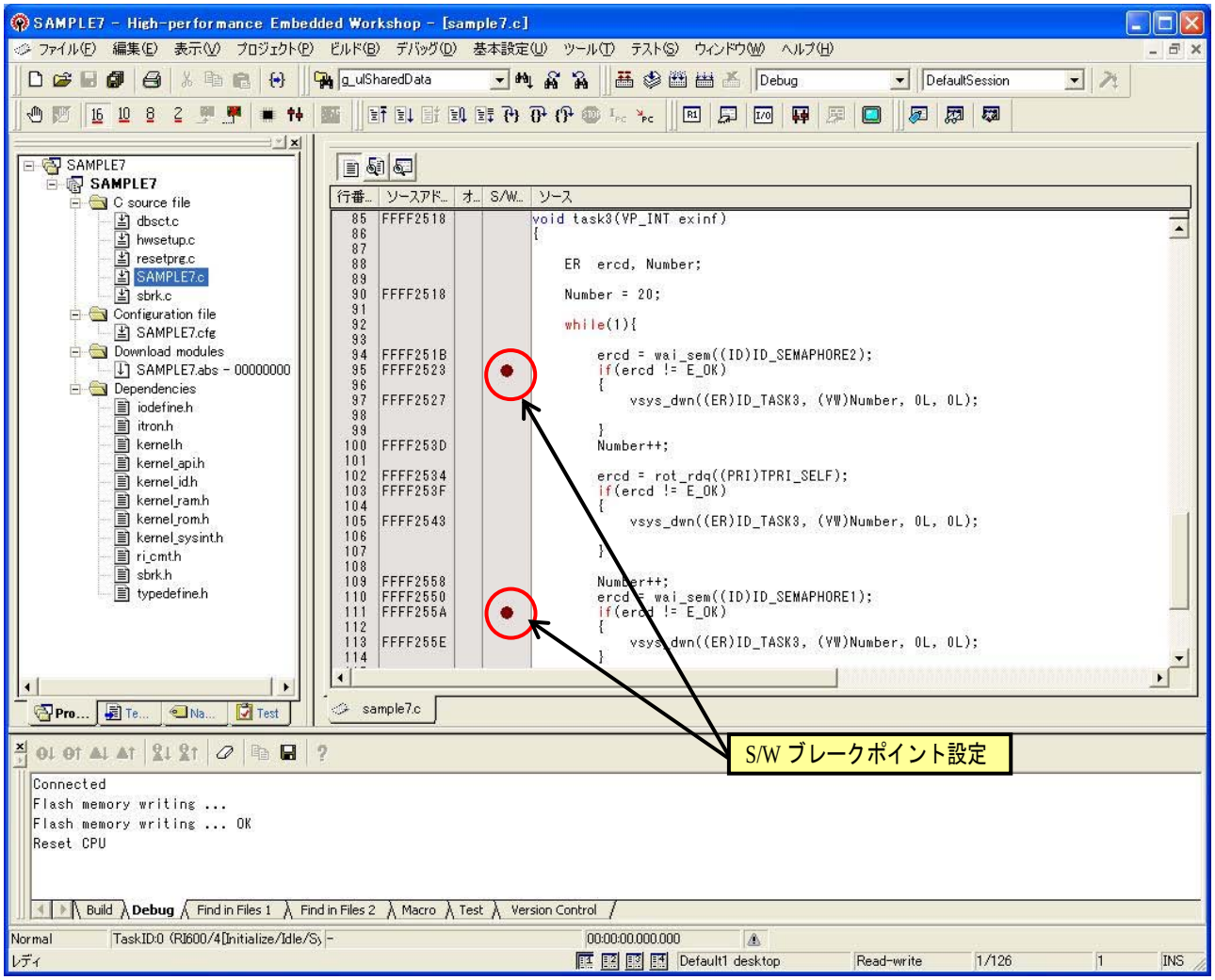


図 3-43　task3　S/W ブレークポイント設定（SAMPLE7）

S/W ブレークポイントを設定後、プログラムを実行します。

task2　wai_sem サービスコールの実行後の S/W ブレークポイントで停止しました。ここで、task2 が ID_SEMAPHORE1 を獲得した状態です。

引き続きプログラムを実行します。

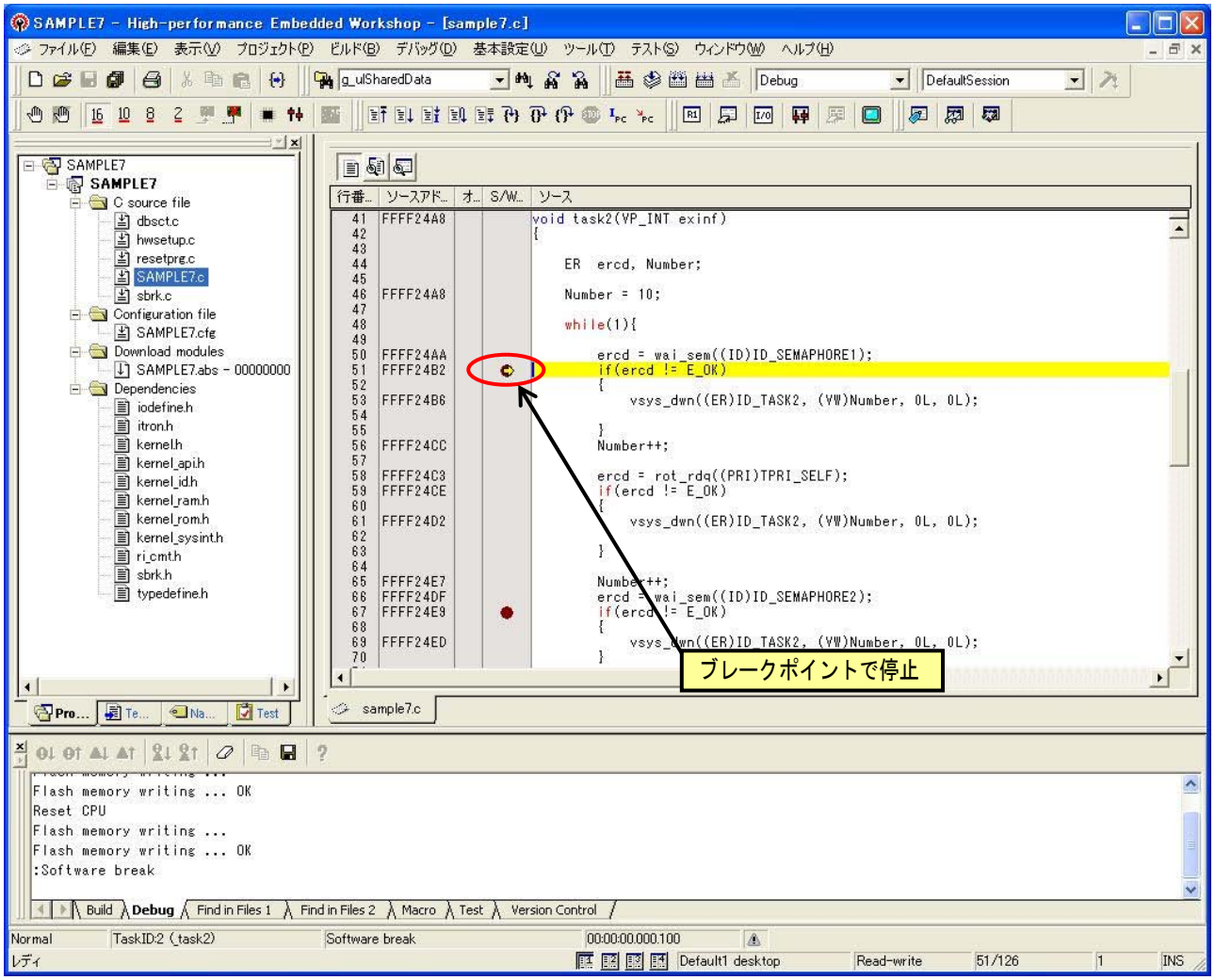


図 3-44　task2　S/W ブレークポイント発生（SAMPLE7）

task3　wai_sem サービスコール実行後の S/W ブレークポイントで停止しました。ここで task3 が ID_SEMAPHORE2 を獲得した状態です。

引き続きプログラムを実行します。

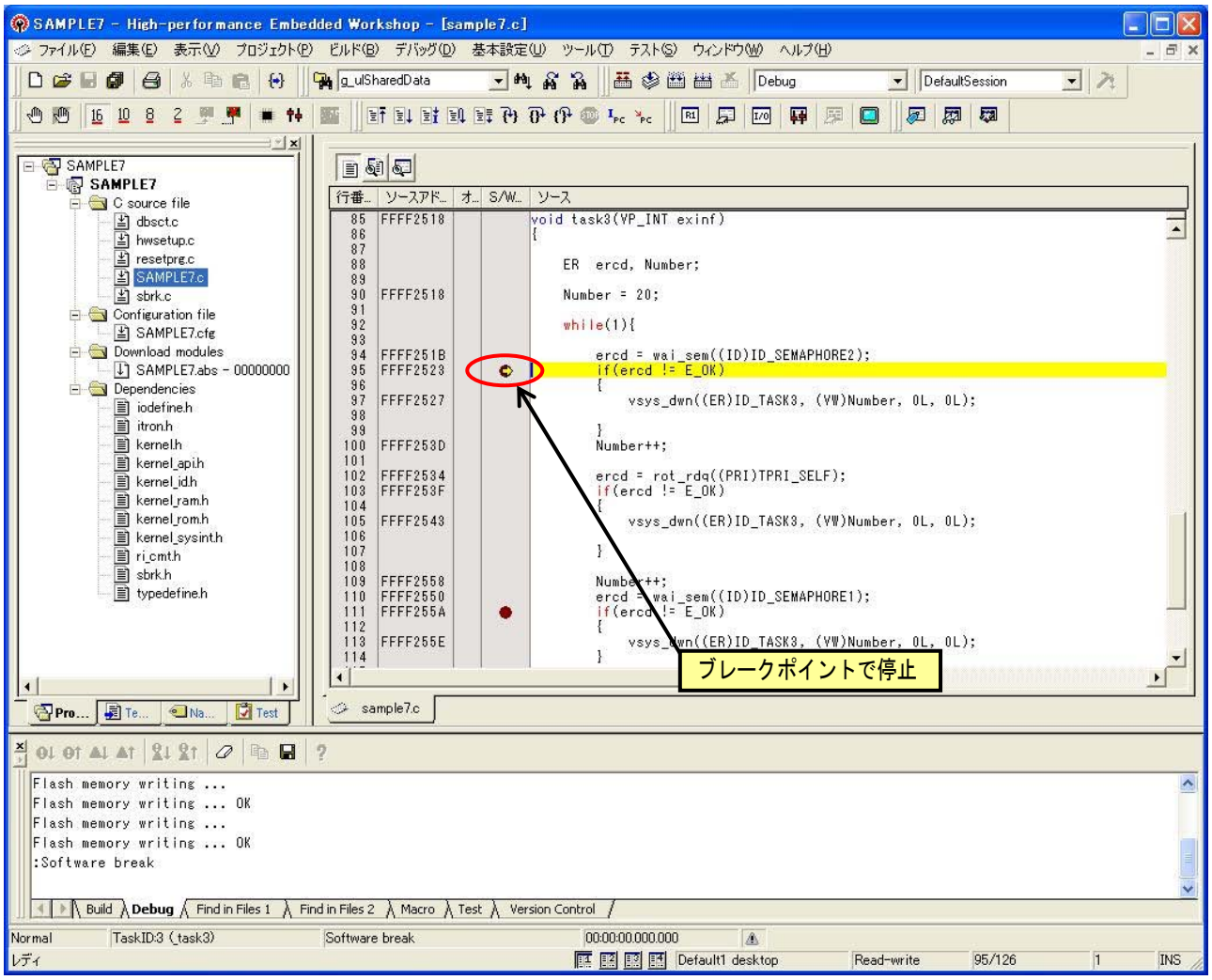


図 3-45　task3　S/W ブレークポイント発生（SAMPLE7）

ところがこれ以降は、設定しているいずれの S/W ブレークポイントで停止することもないため、HEW の機能を使って手動でプログラム停止させます［デバッグ -> プログラムの停止］。

ここで OS オブジェクトウィンドウを開きオブジェクト情報を確認します。

OS オブジェクトウィンドウを開いたら、ID_SEMAPHORE1 のセマフォ資源数を追加します。

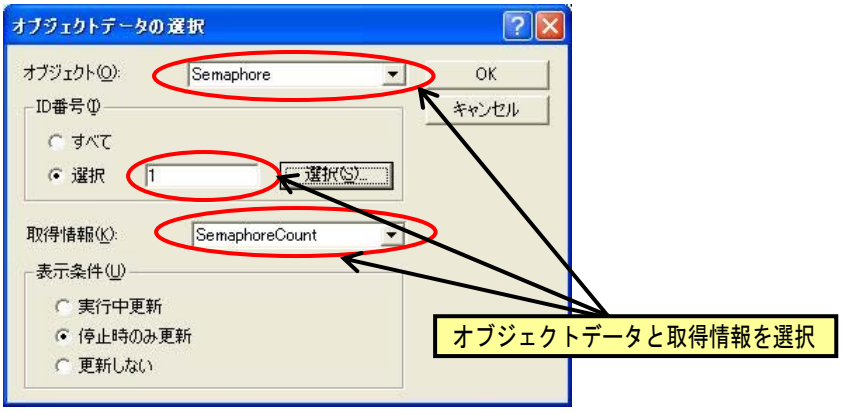


図 3-46　ID_SEMAPHORE1 のセマフォ資源数追加（SAMPLE7）

ID_SEMAPHORE2 のセマフォ資源数を追加します。

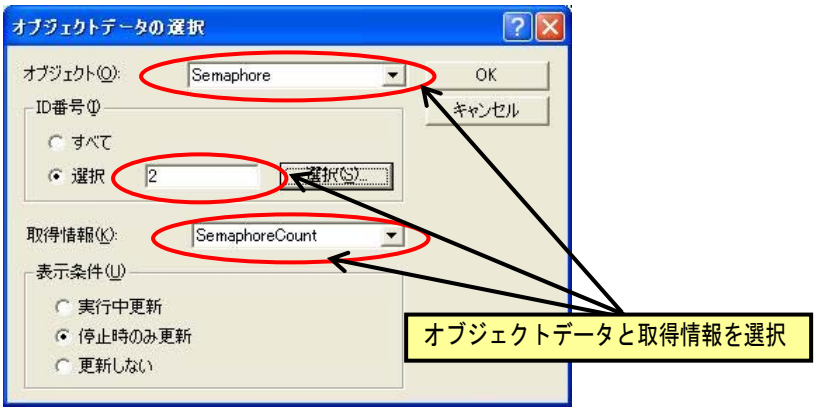


図 3-47　ID_SEMAPHORE2 のセマフォ資源数追加（SAMPLE7）

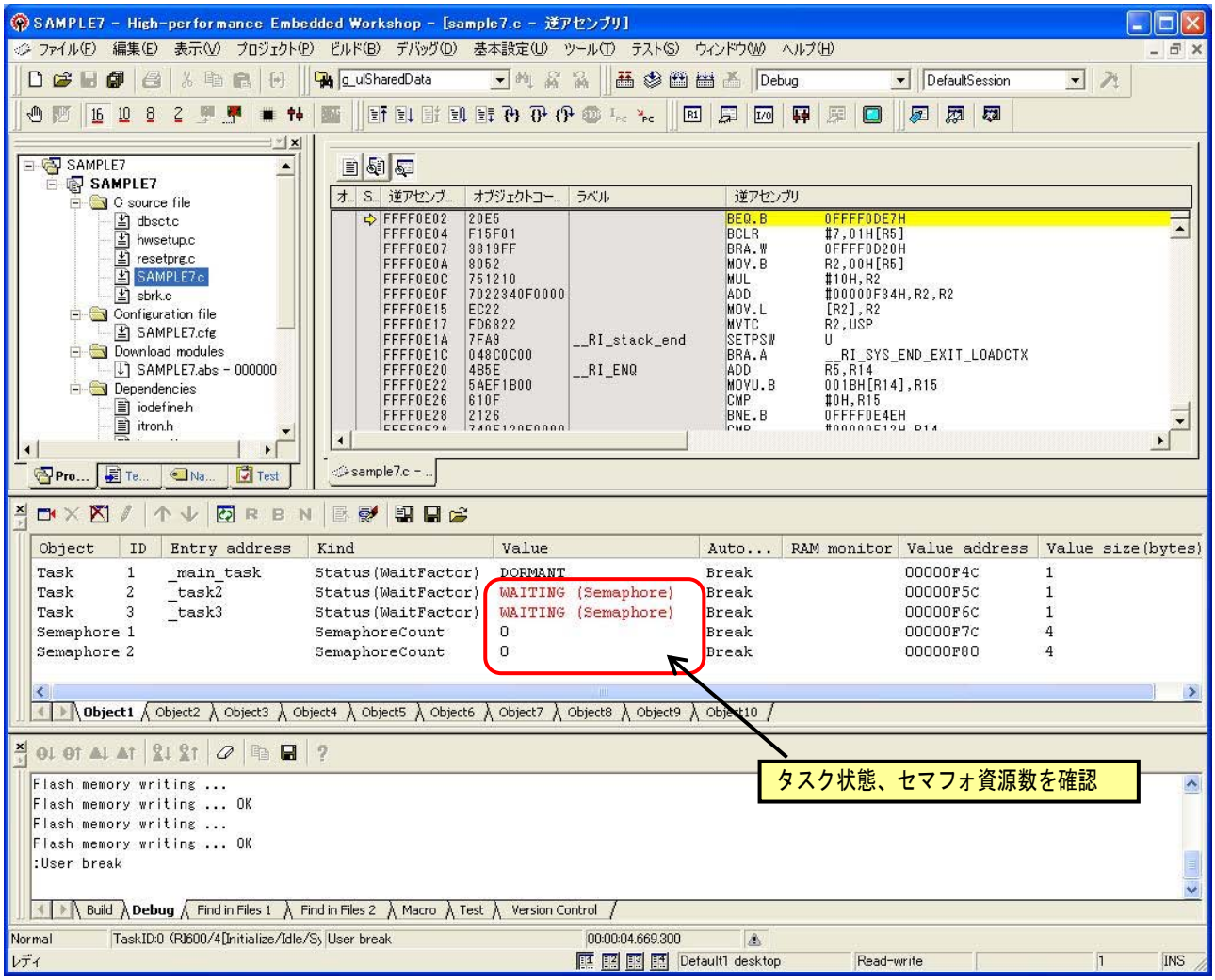


図 3-48　OS オブジェクトの状態確認（SAMPLE7）

図 3-48 オブジェクトウィンドウの情報から、task2 と task3 の状態がセマフォの獲得待ちであること、ID_SEMAPHORE1 と ID_SEMAPHORE2 のセマフォ資源数が“ 0 ”であることが分かります。

task2 と task3 は、

(a)　task2 は ID_SEMAPHORE1 のセマフォを獲得。

(b)　task3 は ID_SEMAPHORE2 のセマフォを獲得。

(c)　task2 は ID_SEMAPHORE2 のセマフォを獲得しようとしますが、task3 が獲得しているため待ち状態

(d)　task3 は ID_SEMAPHORE1 のセマフォを獲得しようとしますが、task2 が獲得しているため待ち状態

という動作をして、明らかにデッドロック状態となっています。

このようにデッドロック状態にならないためには、task2 と task3 の間でセマフォ獲得の順番を同じにしておく必要があります。

3.3.8 SAMPLE8：共通関数でのタスク ID 確認

SAMPLE8 では、複数のタスクで共通の関数をコールしてタスク管理を行います。

(a) main_task で、task2、task3 を起動します。

(b) task2 が起動されます。

(c) task2 は、共通関数（common_module(void)）をコールします。

(d) task3 が起動されます。

(e) task3 も、共通関数（common_module(void)）をコールします。

(f) 共通関数では、カレントのタスク優先度を回転（rot_rdq）します。

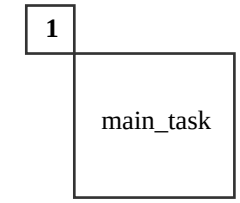
■使用するオブジェクト

表 3-15 使用するタスク（SAMPLE8）

アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	common_module をコール
task3	3	2	common_module をコール

このサンプルプログラムでは、task2 と task3 で共に同じ関数をコールしますが、この共通関数内でプログラムを停止させたとき、関数をコールしていたタスクのタスク ID を確認する方法を説明します。

ソースコードは、以下のように記述しています。



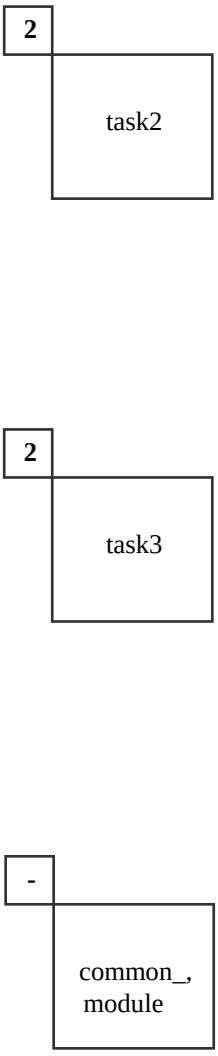
カーネル提供ファイル
cfg600 生成ファイル

task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

```
/************************************************************************
*
*     RI600/4 Sample program
*
************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

extern void common_module();
/************************************************************************
  main_task
************************************************************************/
void main_task(VP_INT exinf)
{
        ER    ercd, Number;
        VP_INT    stacd;

        stacd = 0;
        Number = 0;

        ercd = sta_tsk((ID)ID_TASK2,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
        Number++;

        ercd = sta_tsk((ID)ID_TASK3,stacd);
        if(ercd != E_OK)
        {
                vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }

        ext_tsk();
}
```

図 3-49　サンプルプログラムソースコード『SAMPLE8.c』(1/2)




```
/************************************************************************
  task2
************************************************************************/
void task2(VP_INT exinf)
{

        while(1){

                common_module();

        }
}

/************************************************************************
  task3
************************************************************************/
void task3(VP_INT exinf)
{

        while(1){

                common_module();

        }
}

/************************************************************************
  common module
************************************************************************/
void common_module(void)
{

        ER     ercd, Number;

        Number = 30;

        while(1){
                ercd = rot_rdq((PRI)TPRI_SELF);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)E_CTX, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-49　サンプルプログラムソースコード『SAMPLE8.c』(2/2)

(3) デバッグ開始

E1 エミュレータの S/W ブレーク機能と OS オブジェクト機能、現在タスク ID 表示機能を使用します。

まず、共通関数の common_module の先頭行に E1 エミュレータの S/W ブレークポイントを設定します。

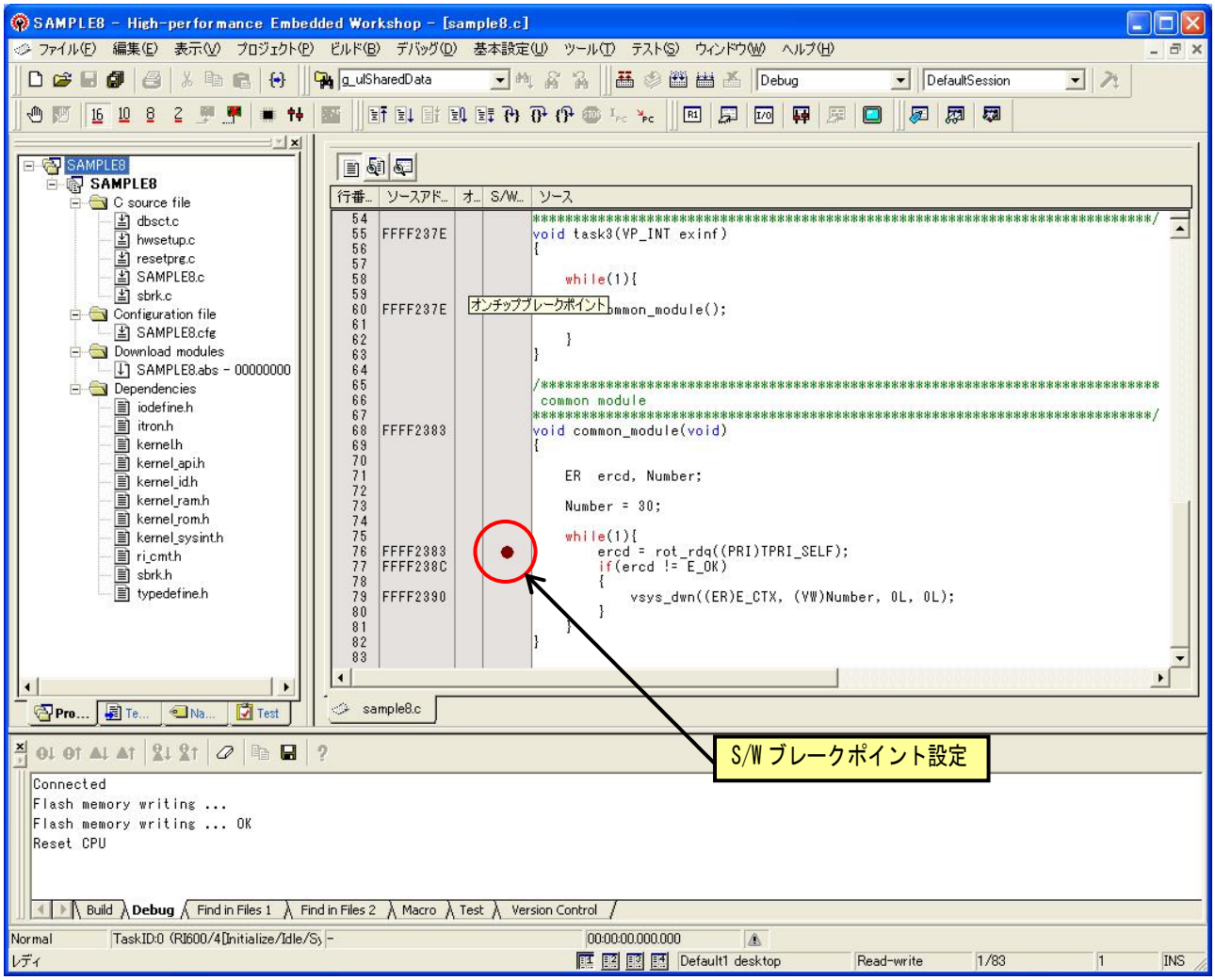


図 3-50 S/W ブレークポイント設定（SAMPLE8）

プログラムを実行すると、設定したブレークポイント(common_module　先頭行)で停止しました。ここで OS オブジェクトウィンドウを開きます。

コール元のタスクの状態は“RUNNING”と表示されかつ、HEW ウィンドウの左下のステータスバーには実行状態のタスク ID が表示されます。コール元のタスクは task2 であることが分かります。

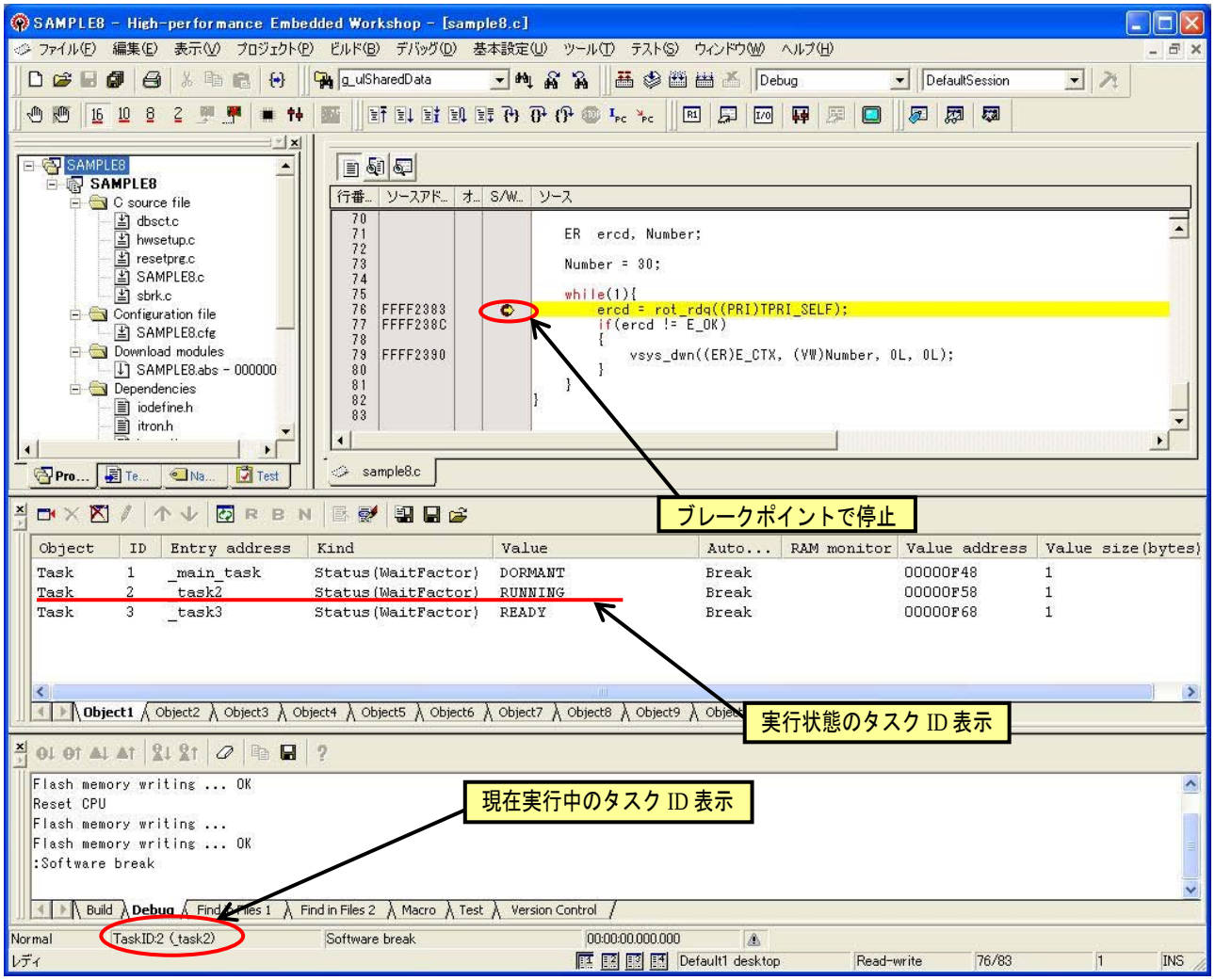


図 3-51　S/W ブレークポイント発生（SAMPLE8）

3.3.9 SAMPLE9：システム異常終了のデバッグ

SAMPLE9 は、メッセージバッファを使用してデータ通信を行うプログラムです。

このサンプルプログラムでは、task3 で何らかの異常が発生し、vsys_dwn サービスコールを発行してシステムダウンしてしまいます。vsys_dwn サービスコールを発行してシステムダウンしてしまう原因を見つける方法を説明します。

(a) main_task で、task2、task3 を起動します。

(b) task2 が起動され、task2 はメッセージバッファにメッセージを送信します。メッセージバッファサイズが 0 のため、メッセージ送信の待ち状態に遷移します。

(c) task3 が起動されます。

(d) task3 は、メッセージバッファからメッセージを受信し、その後、rot_rdq サービスコールを発行してタスク優先度を回転し、実行権を task2 に渡します。

■使用するオブジェクト

表 3-16 タスク（SAMPLE9）

アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	メッセージを送信する
task3	3	2	メッセージを受信する

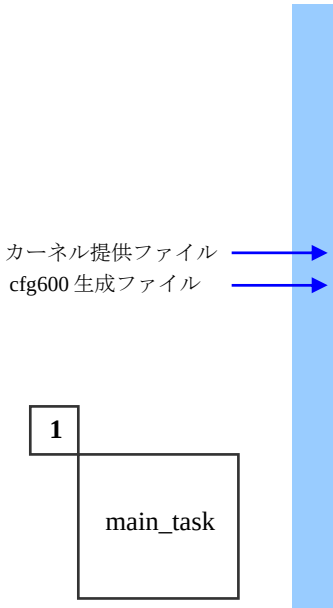
表 3-17 メッセージバッファ（SAMPLE9）

名称	ID 番号	バッファサイズ	メッセージ最大サイズ	属性
ID_MBF1	1	0 *1	10	TA_TFIFO

*1: メッセージバッファのバッファサイズを 0 に設定すると、以下の状態が満たされるまで、先にサービスコールを発行したタスクは待ち状態に遷移します。

- メッセージ送信のサービスコールを発行したタスクは、メッセージ受信のサービスコールが発行されるまで。
- メッセージ受信のサービスコールを発行したタスクは、メッセージ送信のサービスコールが発行されまで。

タスクのソースコードは、以下のように記述しています。



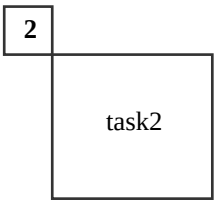
カーネル提供ファイル
cfg600 生成ファイル

```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ext_tsk();
}
```

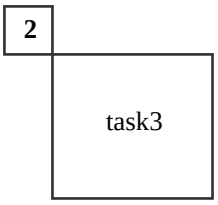
task2 起動
（sta_tsk）

task3 起動
（sta_tsk）

図 3-52　サンプルプログラムソースコード『SAMPLE9.c』(1/2)



メッセージ送信
（snd_mbf）



メッセージ受信
（prcv_mbf）

```
/************************************************************************
 task2
************************************************************************/
void task2(VP_INT exinf)
{

    ER    ercd, Number;
    B     msg[10];

    Number = 10;

    while(1){

        ercd = snd_mbf((ID)ID_MBF1, (VP)msg, (UINT)10);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
        }
    }
}

/************************************************************************
 task3
************************************************************************/
void task3(VP_INT exinf)
{

    ER    ercd, Number;
    B     msg[10];

    Number = 20;

    while(1){

        ercd = prcv_mbf((ID)ID_MBF2, (VP)msg);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }
        ercd = rot_rdq((PRI)TPRI_SELF);
        if(ercd != E_OK)
        {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
        }
    }
}
```

図 3-52　サンプルプログラムソースコード『SAMPLE9.c』(2/2)

(2) デバッグ開始

task3 で異常発生が起こった原因は特定できないため、どうして vsys_dwn サービスコールが発行する条件が満たされたかを確認します。この確認では、E1 エミュレータの S/W ブレーク機能、インスタントウォッチ機能とタスクステップ機能を使用します。

まず、vsys_dwn サービスコールが発行される箇所を特定するために、task3 の prcv_mbf サービスコールおよび rot_rdq サービスコールを実行した直後の 2 箇所に S/W ブレークポイントを設定します。

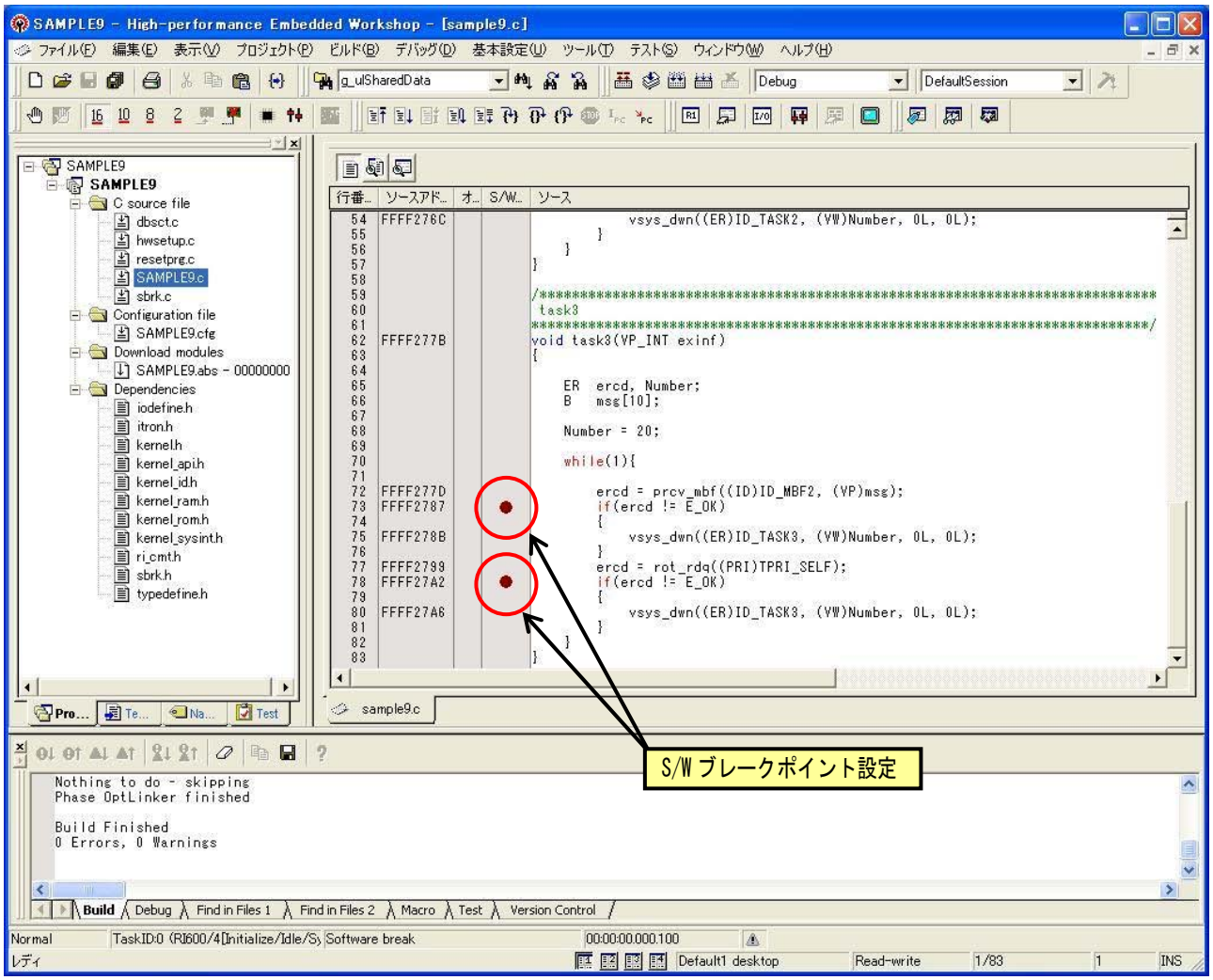


図 3-53 S/W ブレークポイント設定（SAMPLE9）

プログラムを実行すると、prcv_mbf サービスコールの実行後に設定した S/W ブレークポイントで停止しました。ここで、prcv_mbf からのリターン値を確認します。

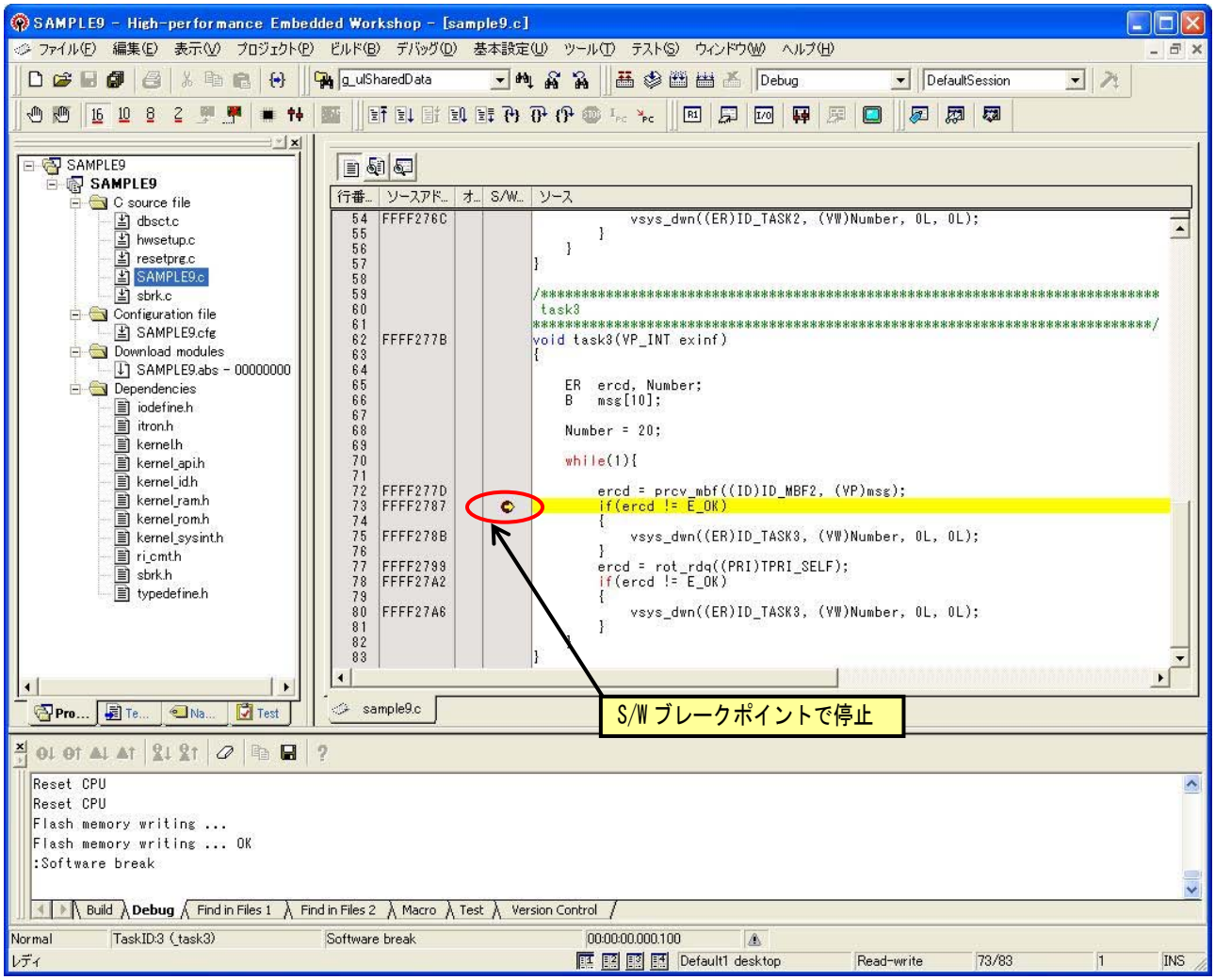


図 3-54　S/W ブレークポイント発生（SAMPLE9）

prcv_mbf サービスコールのリターン値を確認するため、タスクをステップオーバでステップ実行します。すると、vsys_dwn 発行に進みました。

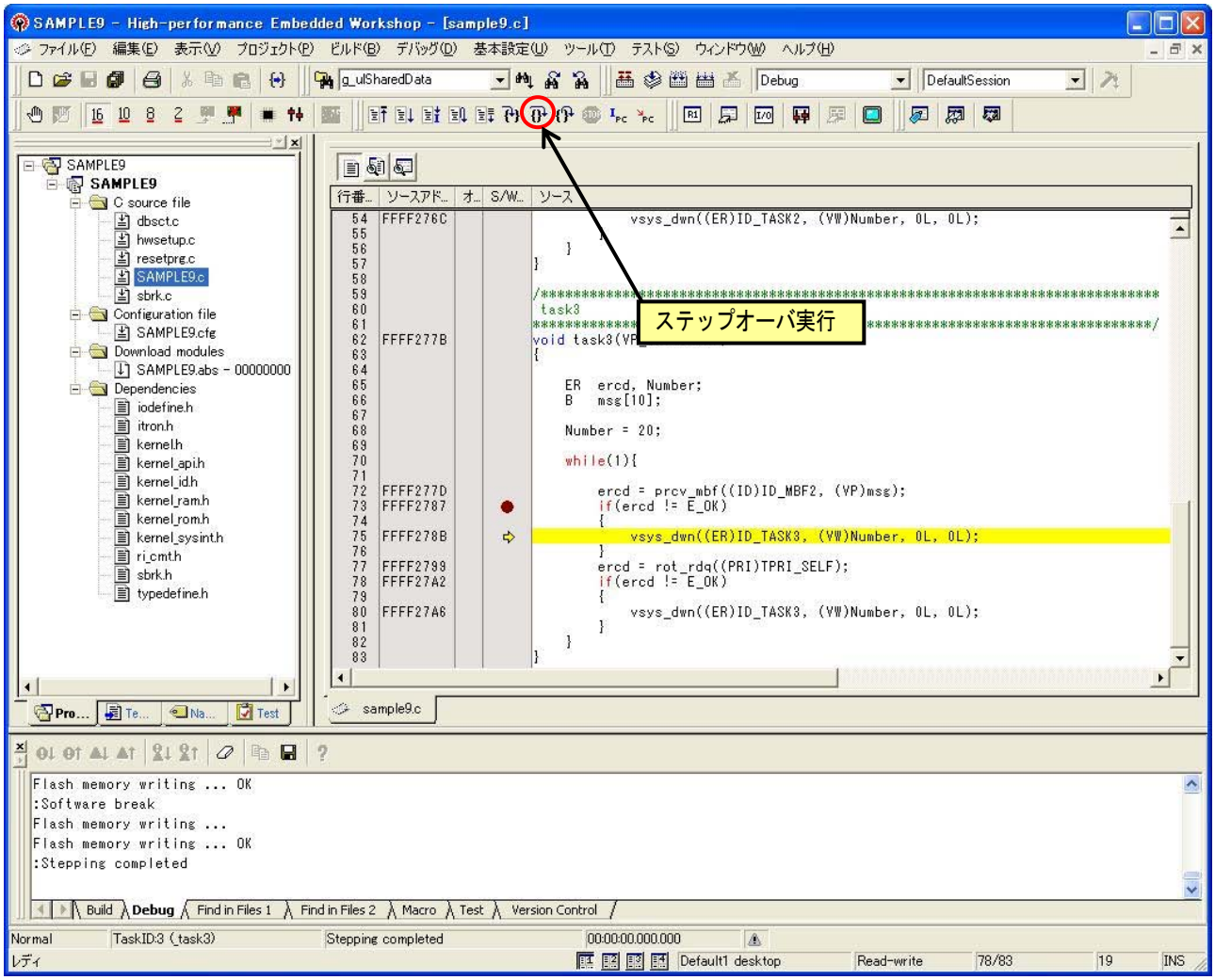


図 3-55 タスクステップ実行（SAMPLE9）

prcv_mbf サービスコールからのリターン値として E_OK 以外のエラーコードが返却されたことが分かりました。返ってきたエラーコードを確認するためには、HEW の[デバッグ]→[リセット後実行]で再度実行させ、S/W ブレークポイントが発生したら、変数 ercd をインスタントウォッチで確認します。

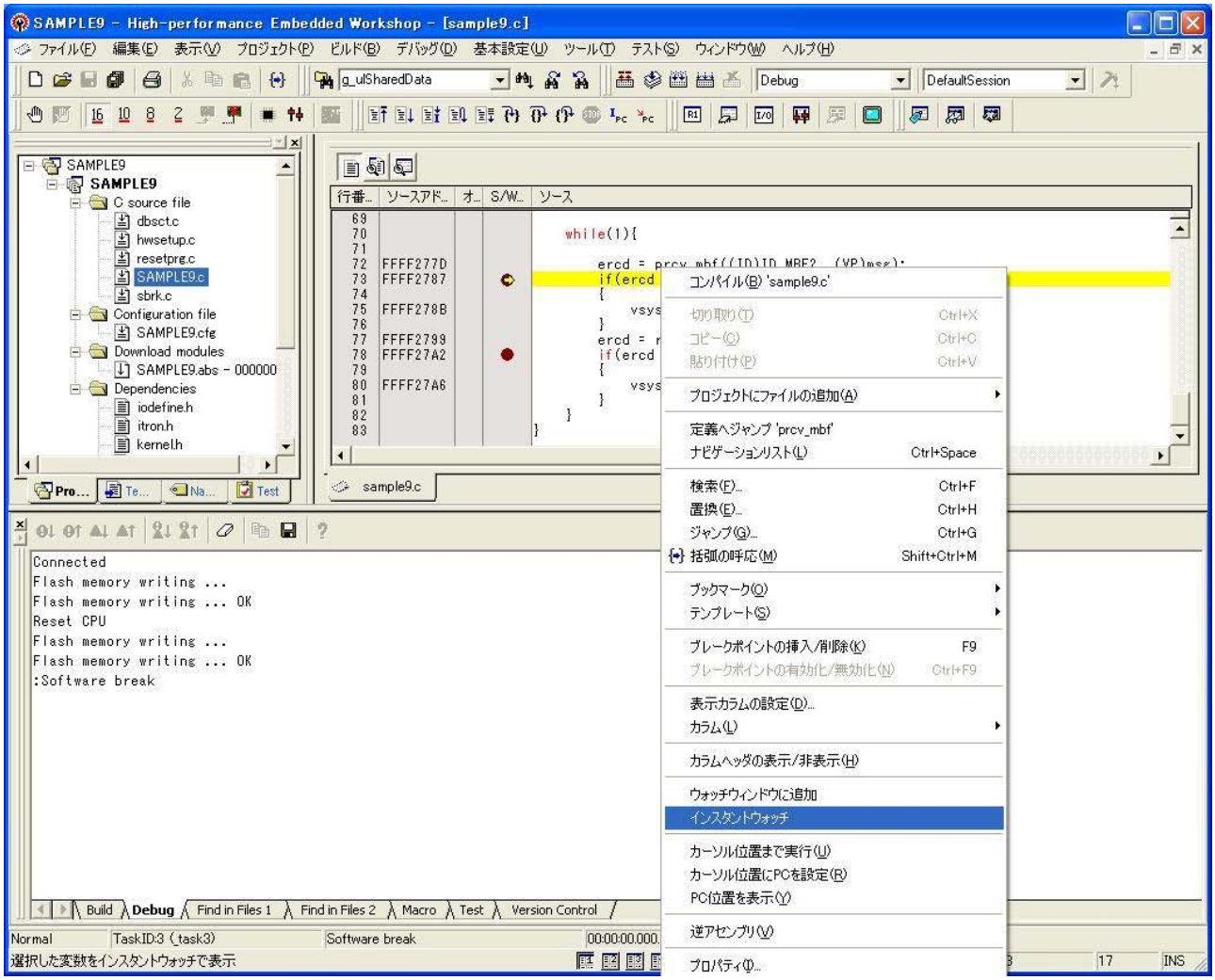


図 3-56　インスタントウォッチの選択（SAMPLE9）

図 3-57　インスタントウォッチの表示（SAMPLE9）

図 3-57 に示すように prcv_mbf サービスコールのリターン値は正常終了（E_OK）ではなくエラーコード（=H’ ffffffce　：　E_TMOUT）が返りました。このエラーコードはポーリング失敗した時に返すエラーコードなので、メッセージを受信できなかったことを意味します。

図 3-52 で task2 の snd_mbf サービスコールと task3 の prcv_mbf サービスコールを比較すると、task2、task3 それぞれで対象メッセージバッファが異なることが分かりました（task2 では ID_MBF1、task3 では ID_MBF2）。したがって、task3 の対象メッセージバッファを ID_MBF1 にすれば良いことが分かりました。

3.3.10 SAMPLE10：システムのスループット向上策

SAMPLE10 は、メッセージバッファを使用してデータ通信を行うプログラムです。

このサンプルプログラムでは、task2 と task3 で行う処理が正常に行われているかを確認するために、task2 と task3 の処理の中にパトロールタスクである task4 を実行させるタスク切り替え処理を組み込みました。

(a) main_task で、task2、task3、task4 を起動します。

(b) task2 が起動されます。(task2 と task3 と task4 のタスク優先度は同じですが、先に実行可能状態になったタスクの方を優先します(First-Come First-Service)。)

(c) task2 は、メッセージバッファにメッセージを送信します。その後実行権を task3 に渡します。

(d) task3 が起動されます。

(e) task3 は、メッセージバッファからメッセージを受信します。その後実行権を task4 に渡します。

(f) task4 が起動されます。

(g) task4 は、task2 と task3 の状態を確認します。問題なければ、実行権を task2 に渡します。

■使用するオブジェクト

表 3-18 タスク(SAMPLE10)

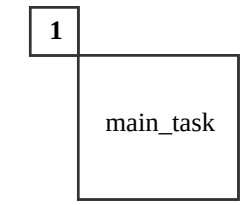
アドレス	ID 番号	優先度	動作
main_task	1	1	初期起動 task2 と task3 を起動
task2	2	2	メッセージを送信する
task3	3	2	メッセージを受信する
task4	4	2	task2 と task3 の状態監視

表 3-19 メッセージバッファ(SAMPLE10)

名称	ID 番号	バッファサイズ	メッセージ最大サイズ	属性
ID_MBF1	1	16	10	TA_TFIFO

このサンプルプログラムでは、パトロールタスク(task4)により task2 と task3 の CPU 使用時間が短くなり、スループットの低下が発生するため、スループットを向上の対策について説明します。

タスクのソースコードは、以下のように記述しています。



カーネル提供ファイル
cfg600 生成ファイル

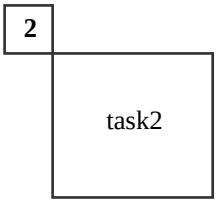
```
/*****************************************************************************
*
*     RI600/4 Sample program
*
*****************************************************************************/
#include <machine.h>
#include <stdio.h>
#include "kernel.h"
#include "kernel_id.h"

/*****************************************************************************
 main_task
*****************************************************************************/
void main_task(VP_INT exinf)
{
      ER    ercd, Number;
      VP_INT    stacd;

      stacd = 0;
      Number = 0;

      ercd = sta_tsk((ID)ID_TASK2,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
      }
      Number++;

      ercd = sta_tsk((ID)ID_TASK3,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
      }

      ercd = sta_tsk((ID)ID_TASK4,stacd);
      if(ercd != E_OK)
      {
            vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
      }
      ext_tsk();
}
```

task2 起動（sta_tsk）

task3 起動（sta_tsk）

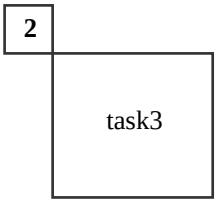
task4 起動（sta_tsk）

図 3-58　サンプルプログラムソースコード『SAMPLE10.c』(1/3)



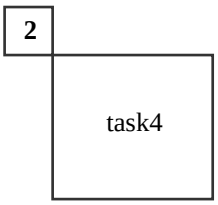
メッセージ送信
（snd_mbf）

```
/************************************************************************
  task2
************************************************************************/
void task2(VP_INT exinf)
{

        ER    ercd, Number;
        B     msg[10];

        Number = 10;

        while(1){

                ercd = snd_mbf((ID)ID_MBF1, (VP)msg, (UINT)10);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }
                ercd = rot_rdq((PRI)TPRI_SELF);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK2, (VW)Number, 0L, 0L);
                }
        }
}

/************************************************************************
  task3
************************************************************************/
void task3(VP_INT exinf)
{

        ER    ercd, Number;
        B     msg[10];

        Number = 20;

        while(1){

                ercd = prcv_mbf((ID)ID_MBF1, (VP)msg);
                if(ercd < (ER)0)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
                ercd = rot_rdq((PRI)TPRI_SELF);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK3, (VW)Number, 0L, 0L);
                }
        }
}
```



メッセージ受信
（prcv_mbf）

図 3-58　サンプルプログラムソースコード『SAMPLE10.c』(2/3)



task2 タスク状態参照
（ref_tsk）

task3 タスク状態参照
（ref_tsk）

```
/*****************************************************************************
  task4
*****************************************************************************/
void task4(VP_INT exinf)
{

        ER    ercd, Number;
        T_RTSK    sts_tsk;

        Number = 30;

        while(1){

                ercd = ref_tsk((ID)ID_TASK2, (T_RTSK *)&sts_tsk);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
                }
                if(sts_tsk.tskstat != (STAT)TTS_RDY)
                {
                        vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
                }

                ercd = ref_tsk((ID)ID_TASK3, (T_RTSK *)&sts_tsk);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
                          }
                if(sts_tsk.tskstat != (STAT)TTS_RDY)
                {
                        vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
                }

                ercd = rot_rdq((PRI)TPRI_SELF);
                if(ercd != E_OK)
                {
                        vsys_dwn((ER)ID_TASK4, (VW)Number, 0L, 0L);
                }
        }
}
```

図 3-58　サンプルプログラムソースコード『SAMPLE10.c』(3/3)

(2) デバッグ開始

task2 と task3 の CPU 使用時間の実態を確認するため、タスクトレース機能、タスクアナライズ機能を使用します。

サンプルプログラムを実行する前に、タスクトレースウィンドウとタスクアナライズウィンドウを開きます。

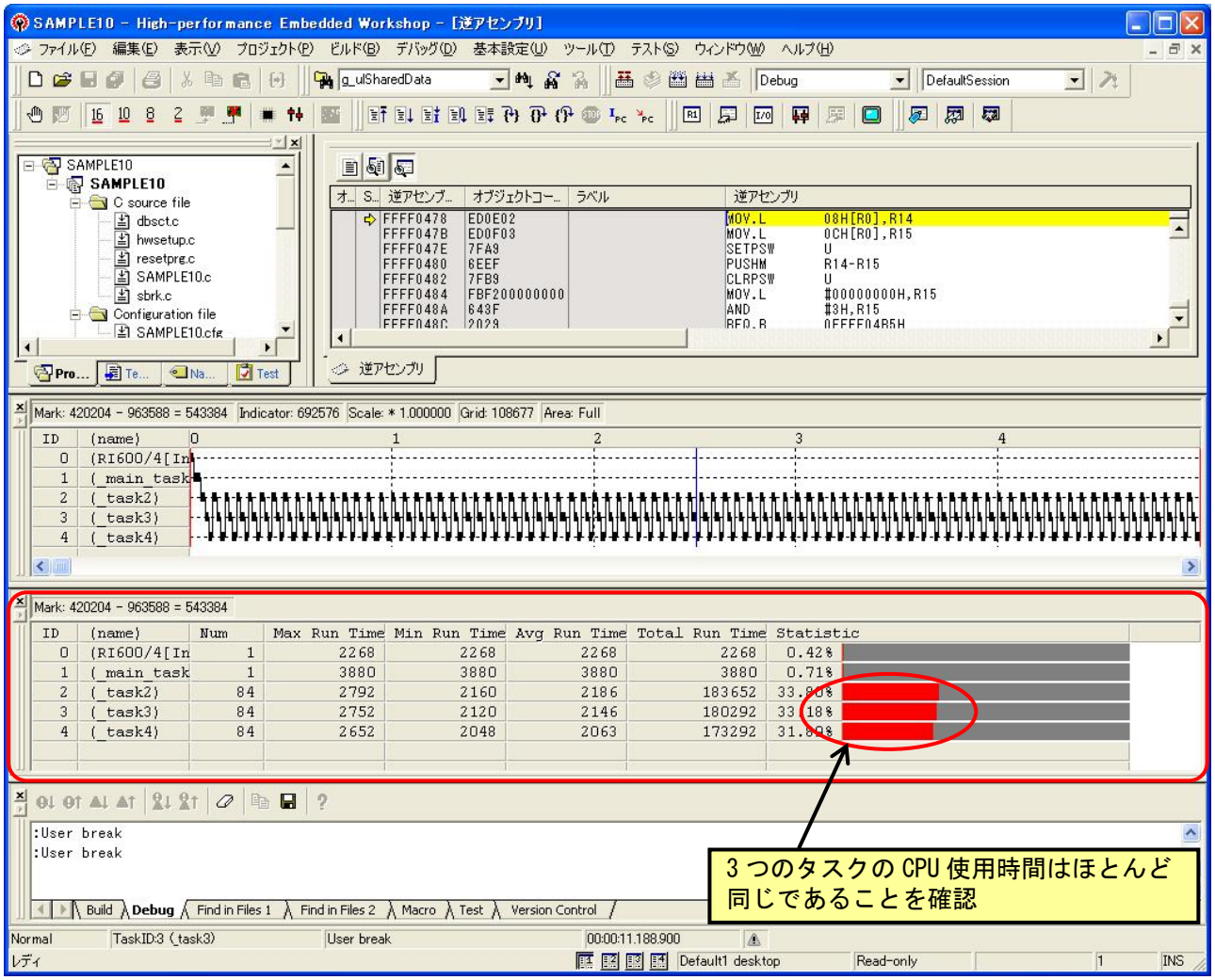


図 3-59 CPU 使用時間確認（SAMPLE10）

図 3-59 から task2、task3、task4 の CPU 使用時間は、ほとんど同じであることが確認できます。

スループットを向上するには、task2 と task3 の CPU 使用時間を長くして、task4 の CPU 使用時間を短くしなければなりません。これを実現する 1 つの方法として、task4 の通常状態は起床待ち状態として、この待ち状態は周期ハンドラを使用してミリセカンド単位で解除するようにします。この方法であれば、task2 と task3 の CPU 使用時間に対して、変更前に task4 に与えていた CPU 使用時間を与えることができるため、その分スループットが向上します。

(3) スループット向上策

その他、スループットを向上させる方法として、以下のことが考えられますので参考にしてください。

(a) タスク一本における処理量の最適化を図ります。例えば一本のタスクで複数の事象に対する処理を行うと、プログラムが複雑化して、その結果処理時間が長くかかることがあります。一本のタスクで処理する事象の最適化を図ることによって処理時間が早くなるため、スループットが向上します。

(b) 1つの事象に対する処理において、タスクが切り替わる回数を少なくする処理方式を採用します。これによって、OS のタスク切り替え処理が減り処理時間が早くなるため、スループットが向上します。

(c) 割込みハンドラの処理時間を早くする（割込みハンドラの処理方式の最適化）と、その分の CPU をタスクが使用できることになるため、スループットが向上します。

(d) 割込みの発生要因をもとに、スループットを向上させるのに必要な要因の割込みの優先度は高くします。これによって他の割込みが発生しても処理時間に影響を受けないため、スループットが向上します。

ホームページとサポート窓口

- ルネサス エレクトロニクスホームページ
  http://japan.renesas.com/
- お問合せ先
  http://japan.renesas.com/inquiry

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

Colophon 1.0