お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

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。

SuperH RISC engine C/C++ コンパイラパッケージ

アプリケーションノート：＜統合開発環境活用ガイド＞

シミュレータ活用ガイド

本ドキュメントでは、シミュレータの便利な機能について説明します。

目次

1. はじめに

本ドキュメントでは、High-performance Embedded Workshop (以下、ルネサス統合開発環境と呼ぶ)のシミュレータ機能のうち、表 1-1で示す機能について説明します。

表 1-1 本ドキュメントで説明する機能一覧

機能	説明している章
I/O シミュレーション	2章
画像表示	3章
プロファイル	4章
パフォーマンス解析	4章
擬似割り込み	5章
タイマシミュレーション	6章
イベントポイント	7章
仮想入出力パネル	8章

各機能のサンプルプロジェクトを用意しています。サンプルプロジェクトは本ドキュメントと共にダウンロードサイトより提供しています。擬似割り込みに対してSH-2AおよびSH-4 用のサンプルプロジェクトを用意しています。擬似割り込み以外の機能に対してはSH-2A用のみのサンプルプロジェクトを用意しています。それぞれのサンプルプロジェクトのプロジェクト名は 表 1-2の通りです。なお、サンプルプロジェクトのワークスペースはC:¥WorkSpace¥sampleに置いてください。

表 1-2 本ドキュメントで説明するサンプルプロジェクト

内容	プロジェクト名
I/O シミュレーション	sample_file_io
画像表示	sample_img
プロファイル／パフォーマンス解析	sample_profile
擬似割り込み(SH-2A 用)	sample_trigger_sh2a
擬似割り込み(SH-4 用)	sample_trigger_sh4
タイマシミュレーション	sample_timer
イベントポイント	sample_ep
仮想入出力パネル	sample_panel

サンプルプロジェクトの詳細は各章のサンプルプログラムの説明を参照してください。サンプルプロジェクトは、以下の環境で作成されています。この環境より古いバージョンではサンプルプロジェクトを開けません。なお、用意したサンプルプロジェクトはシミュレータ用です。I/O シミュレーション用のサンプルプログラムはエミュレータでは動作しません。

- Renesas SuperH ファミリ用 C/C++コンパイラパッケージ ・・・・V.9.01 Release 01
- High-performance Embedded Workshop ・・・・V.4.03.00
- ツールチェイン ・・・・V.9.1.1.0
- シミュレータ ・・・・V.9.0.7

2. I/O シミュレーション

2.1 概要

シミュレータには、ファイル／標準入出力をシミュレータ上で仮想的に確認するために、I/O シミュレーション機能が備わっています。本章では、I/O シミュレーション機能の概要と、本機能を使用してファイル／標準入出力を確認するためのサンプルプログラムについて説明します。

(1) 低水準インタフェースルーチン

C/C++言語を使用してプログラムを開発する際、標準入出力ライブラリ(fopen()、printf()、scanf()などの関数)を使用してファイル／標準入出力を行うことがあります。このとき、標準入出力ライブラリは、ユーザ側で実装した実際に入出力を行う関数を呼び出してファイル／標準入出力を行います。この実際に入出力を行う関数のことを「低水準インタフェースルーチン」と呼びます。この低水準インタフェースルーチンが必要になる理由は次の通りです。組み込みアプリケーションでは、標準入出力の入出力先は LCD、HDD、プリンタ、CD-R/RW、DIP スイッチ、キーボード、マウス、携帯電話のボタン、タッチパネルなど多岐に渡ります。これらは、ユーザシステムに適した入出力処理が必要となります。そのため、標準入出力ライブラリからは、ユーザ定義の低水準インタフェースルーチンと呼ばれる関数群を呼ぶ構成にしています。

低水準インタフェースルーチンは、「SuperH™ RISC engine C/C++コンパイラ、アセンブラ、最適化リンケージエディタユーザズマニュアル 9.2.2 実行環境の設定 (6) 低水準インタフェースルーチン」の仕様に則って実装して下さい。本章では、シミュレータ上で仮想的にファイル／標準入出力を行うための低水準インタフェースルーチンのサンプルプログラムを紹介します。サンプルプログラムは、標準入力(stdin)、標準出力(stdout)、標準エラー出力(stderr)、ファイル入出力に対応しています。

(2) I/O シミュレーション機能

シミュレータ上で仮想的にファイル／標準入出力を行う機能を、I/Oシミュレーション機能と呼びます。I/Oシミュレーション機能は、シミュレータ上で設定した特定のアドレス(I/Oシミュレーションアドレス)への分岐命令により実行されます。I/Oシミュレーションアドレスへの分岐命令は、I/Oシミュレーション機能が呼び出されるのみで、分岐自体は行われません。I/Oシミュレーション機能のパラメータはR0 レジスタとR1 レジスタよりシミュレータに渡されます。R0 レジスタには、使用するI/Oシミュレーションの機能(機能コード)を設定します。シミュレータは、R0 レジスタに設定された機能コードにしたがって、ファイル入出力やシミュレータのウィンドウへの文字列出力を行います。R1 レジスタには、ファイル名や出力文字など、I/Oシミュレーション機能固有のパラメータが設定されたメモリ領域(パラメータブロック)の先頭アドレスを設定します。I/Oシミュレーションの機能コードおよびパラメータブロックの設定方法は、「2.2機能」をご覧ください。

上記の通り、I/O シミュレーション機能を呼び出すとき(分岐命令を実行するとき)には、R0 と R1 にパラメータが設定されている必要があります。このため、I/O シミュレーション機能の呼び出し部分(I/O シミュレーション関数)は、アセンブリ言語で記述する必要があります。サンプルプログラムでは、シミュレータ上でファイル／標準入出力を行えるよう、低水準インタフェースルーチンから I/O シミュレーション関数を呼び出しています。

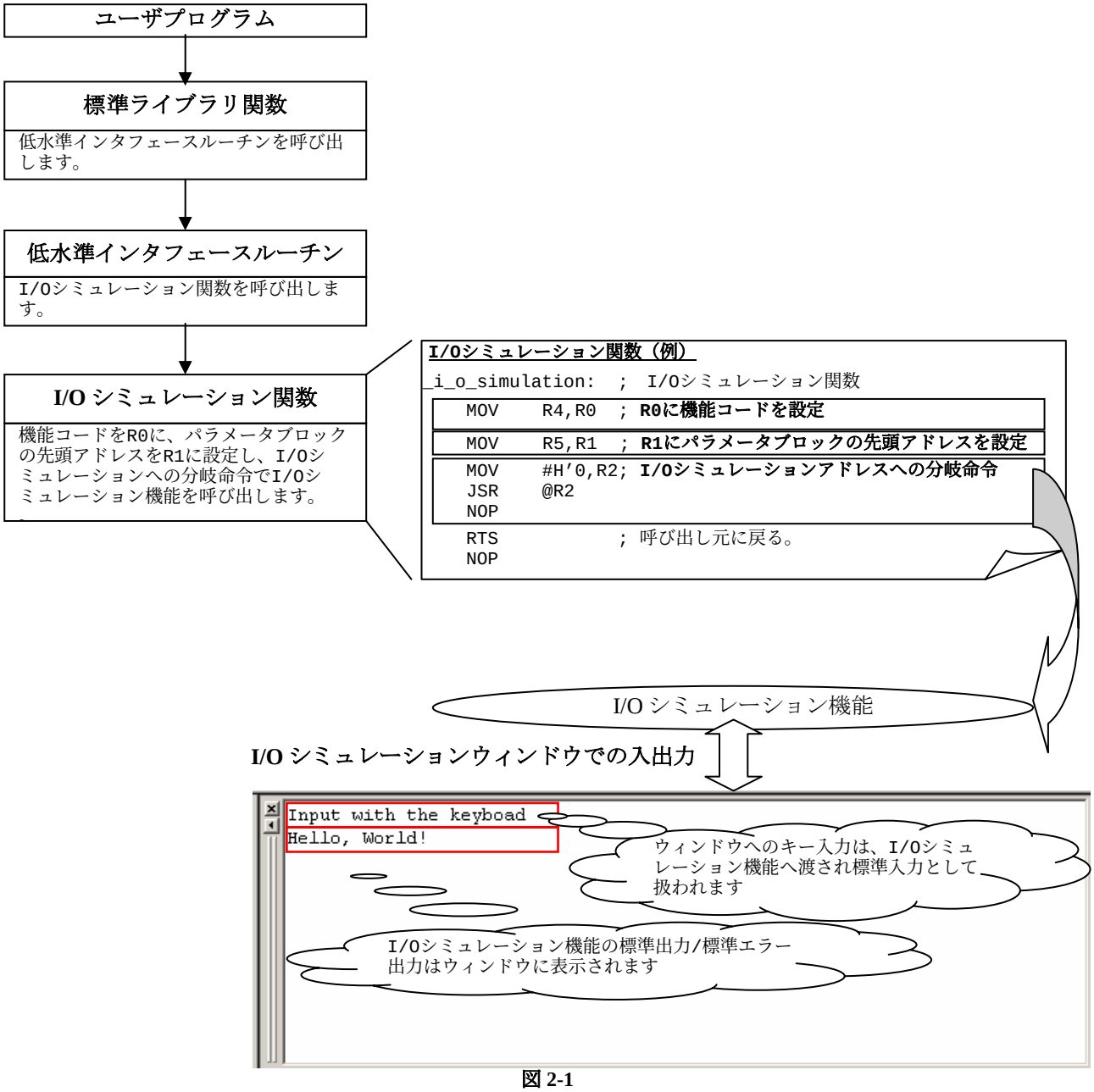

ユーザプログラム
標準ライブラリ関数
低水準インタフェースルーチンを呼び出します。
低水準インタフェースルーチン
I/Oシミュレーション関数を呼び出します。
I/O シミュレーション関数
機能コードをR0に、パラメータブロックの先頭アドレスをR1に設定し、I/Oシミュレーションへの分岐命令でI/Oシミュレーション機能を呼び出します。
I/Oシミュレーション関数（例）
_i_o_simulation: ; I/Oシミュレーション関数
MOV R4,R0 ; R0に機能コードを設定
MOV R5,R1 ; R1にパラメータブロックの先頭アドレスを設定
MOV #H'0,R2; I/Oシミュレーションアドレスへの分岐命令
JSR @R2
NOP
RTS ; 呼び出し元に戻る。
NOP
I/O シミュレーション機能
I/O シミュレーションウィンドウでの入出力
Input with the keyboad
Hello, World!
ウィンドウへのキー入力は、I/Oシミュレーション機能へ渡され標準入力として扱われます
I/Oシミュレーション機能の標準出力/標準エラー出力はウィンドウに表示されます

図 2-1

2.2 機能

I/Oシミュレーション機能を呼び出す前に、機能コードをR0 レジスタに、パラメータブロックの先頭アドレスをR1 レジスタに設定する必要があります。R0 レジスタおよびR1 レジスタの設定内容を 図 2-2に示します。入出力処理が終了して、I/Oシミュレーションアドレスへの分岐命令の次の命令からシミュレーションを再開します。

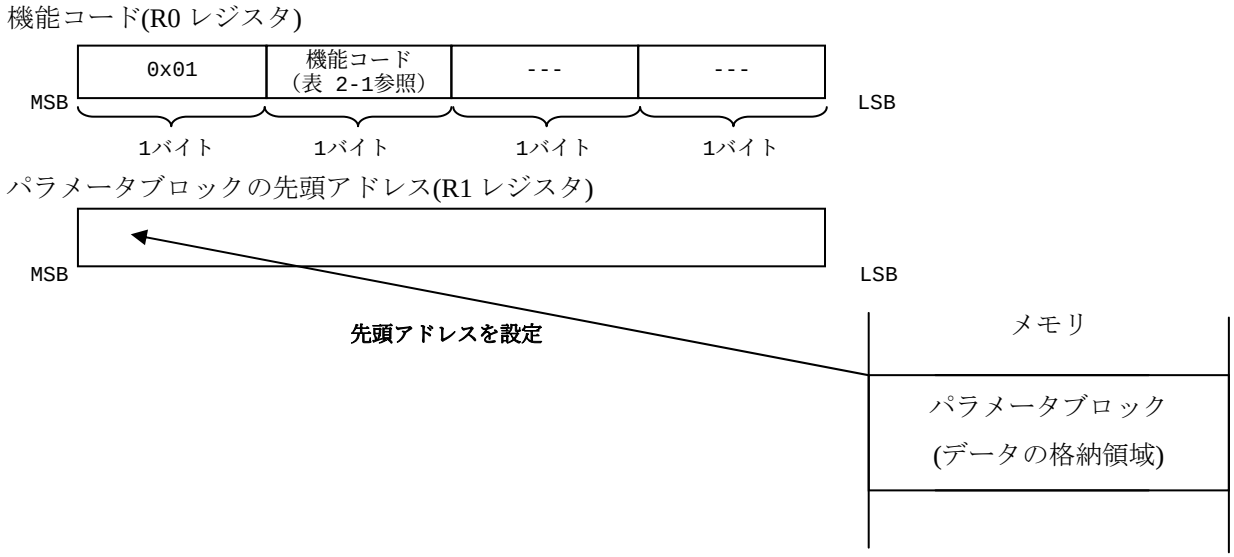


図 2-2

I/O シミュレーションでサポートしている機能は 表 2-1の通りです。

表 2-1 機能一覧

番号	機能コード	機能名	内容
1	H'21	GETC	**標準入力からの１バイト入力**
2	H'22	PUTC	**標準出力への１バイト出力**
3	H'23	GETS	**標準入力からの 1 行入力**
4	H'24	PUTS	**標準出力への１行出力**
5	H'25	FOPEN	**ファイルのオープン**
6	H'06	FCLOSE	**ファイルのクローズ**
7	H'27	FGETC	**ファイルからの１バイト入力**
8	H'28	FPUTC	**ファイルへの１バイト出力**
9	H'29	FGETS	**ファイルからの１行入力**
10	H'2A	FPUTS	**ファイルへの１行出力**
11	H'0B	FEOF	**エンドオブファイルのチェック**
12	H'0C	FSEEK	**ファイルポインタの移動**
13	H'0D	FTELL	**ファイルポインタの現在位置を得る**

各入出力機能を以下の記述で説明します。

<table>
<tr><td rowspan="2">(1) 表 2-1に対応する番号</td><td>(2) 機能名</td><td rowspan="2">(4) 入出力の機能</td></tr>
<tr><td>(3) 機能コード</td></tr>
<tr><td colspan="2">(5) 入出力のパラメータブロック</td><td>(6) 入出力のパラメータ</td></tr>
</table>

GETC：標準入力からの 1 バイト入力

1	GETC H’21	標準入力からの 1 バイト入力
1バイト 1バイト +0 +2 入力バッファ先頭アドレス		• **入力バッファ先頭アドレス(入力)** **入力データを書き込むバッファの先頭アドレス**

PUTC：標準出力への 1 バイト出力

2	PUTC H’22	標準出力への 1 バイト出力
1バイト 1バイト +0 +2 出力バッファ先頭アドレス		• **出力バッファ先頭アドレス(出力)** **出力データを格納しているバッファの先頭アドレス**

GETS：標準入力からの 1 行入力

3	GETS H’23	標準入力からの 1 行入力
1バイト 1バイト +0 +2 入力バッファ先頭アドレス		• **入力バッファ先頭アドレス(入力)** **入力データを書き込むバッファの先頭アドレス**

PUTS：標準出力への 1 行出力

4	PUTS H’24	標準出力への 1 行出力
1バイト 1バイト +0 +2 出力バッファ先頭アドレス		• **出力バッファ先頭アドレス(出力)** **出力データを格納しているバッファの先頭アドレス**

FOPEN：ファイルのオープン

5	FOPEN H’25	ファイルのオープン

[FOPEN]によってファイルをオープンして、ファイル番号を返します。
以後のファイル入出力、ファイルクローズ等では、このファイル番号を用います。
同時にオープンできる最大ファイル数は 256 **です。**

	1バイト	1バイト
+0	実行結果	ファイル番号
+2	オープンモード	未使用
+4 +6	ファイル名先頭アドレス	

- **実行結果(出力)**
  0 **正常終了**
  -1 **エラー**
- **ファイル番号(出力)**
  オープン処理以降のファイルアクセスで使用する番号
- **オープンモード(入力)**
  H'00 "r"
  H'01 "w"
  H'02 "a"
  H'03 "r+"
  H'04 "w+"
  H'05 "a+"
  H'10 "rb"
  H'11 "wb"
  H'12 "ab"
  H'13 "r+b"
  H'14 "w+b"
  H'15 "a+b"
  各モードの内容は以下の通りです。
  "r" **読み出し用にオープンする。**
  "w" **空ファイルを書き込み用にオープンする。**
  "a" **ファイルの最後から書き込み用にオープンする。**
  "r+" **読み出し，書き込み用にオープンする。**
  "w+" **空ファイルを読み出し，書き込み用にオープンする。**
  "a+" **読み出し追加用にオープンする。**
  "b" **バイナリモードでオープンする。**
- **ファイル名先頭アドレス（入力）**
  ファイル名を格納している領域の先頭アドレス

FCLOSE：ファイルのクローズ

6	FCLOSE H’06	ファイルのクローズ

	1バイト	1バイト
+0	実行結果	ファイル番号

- **実行結果(出力)**
  0 **正常終了**
  -1 **エラー**
- **ファイル番号(入力)**
  ファイルオープン時に返す番号

FGETC：ファイルからの 1 バイトデータ読み出し

7	FGETC H’27	ファイルからの 1 バイトデータ読み出し

	1バイト	1バイト
+0	実行結果	ファイル番号
+2	未使用	
+4 +6	入力バッファ先頭アドレス	

- **実行結果(出力)**
  0 **正常終了**
  -1 EOF **検出**
- **ファイル番号(入力)**
  ファイルオープン時に返す番号
- **入力バッファ先頭アドレス(入力)**
  入力データを書き込むバッファの先頭アドレス

FPUTC：ファイルへの 1 バイトデータ書き込み

8	FPUTC H’28	ファイルへの 1 バイトデータ書き込み
1バイト　1バイト +0 実行結果 | ファイル番号 +2 未使用 +4 +6 出力バッファ先頭アドレス		• **実行結果(出力)** 0 **正常終了** -1 **エラー** • **ファイル番号(入力)** **ファイルオープン時に返す番号** • **出力バッファ先頭アドレス(入力)** **出力データを格納しているバッファの先頭アドレス**

FGETS：ファイルからの文字列データ読み出し

9	FGETS H’29	ファイルからの文字列データ読み出し
改行コードまたは ” NULL ” コードを検出するまで, またはバッファサイズに達するまでファイルから文字列データを読み出します。 1バイト　1バイト +0 実行結果 | ファイル番号 +2 バッファサイズ +4 +6 入力バッファ先頭アドレス		• **実行結果(出力)** 0 **正常終了** -1 EOF **検出** • **ファイル番号(入力)** **ファイルオープン時に返す番号** • **バッファサイズ (入力)** **データを格納する領域のサイズ** **(バイト単位で最大** 256 **バイトまで)** • **入力バッファ先頭アドレス(入力)** **入力データを書き込むバッファの先頭アドレス**

FPUTS：ファイルへの文字列データ書き込み

10	FPUTS H’2A	ファイルへの文字列データ書き込み
ファイルへ文字列データ書き込みます。 文字列終端記号の ” NULL ” コードはファイルには書き込みません。 1バイト　1バイト +0 実行結果 | ファイル番号 +2 未使用 +4 +6 出力バッファ先頭アドレス		• **実行結果(出力)** 0 **正常終了** -1 **エラー** • **ファイル番号(入力)** **ファイルオープン時に返す番号** • **出力バッファ先頭アドレス(入力)** **出力データを格納しているバッファの先頭アドレス**

FEOF：エンドオブファイルのチェック

11	FEOF H’0B	エンドオブファイルのチェック
1バイト　1バイト +0 実行結果 | ファイル番号		• **実行結果(出力)** 0 EOF **でない** -1 EOF **検出** • **ファイル番号(入力)** **ファイルオープン時に返す番号**

FSEEK：指定位置にファイルポインタを移動

12	FSEEK H'0C	指定位置にファイルポインタを移動
1バイト　1バイト +0 実行結果 | ファイル番号 +2 ディレクション | 未使用 +4 +6 オフセット		• **実行結果(出力)** 0 **正常終了** -1 **エラー** • **ファイル番号(入力)** **ファイルオープン時に返す番号** • **ディレクション (入力)** 0 **オフセットはファイルの先頭からのバイト数** 1 **オフセットは現在のファイルポインタからのバイト数** 2 **オフセットはファイルの最後尾からのバイト数** • **オフセット (入力)** **ディレクションで指定した位置からのバイト数**

FTELL：ファイルポインタの現在位置を調査

13	FTELL H'0D	ファイルポインタの現在位置を調査
1バイト　1バイト +0 実行結果 | ファイル番号 +2 未使用 +4 +6 オフセット		• **実行結果(出力)** 0 **正常終了** -1 **エラー** • **ファイル番号(入力)** **ファイルオープン時に返す番号** • **オフセット (入力)** **ディレクションで指定した位置からのバイト数**

2.3　使用方法

I/O シミュレーションを使用するには I/O シミュレーションの機能を有効にしてください。また、標準入出力を確認するには、[I/O シミュレーション]ウィンドウを表示する必要があります。

(1) I/O シミュレーションの有効化

I/O シミュレーションを有効にするには、[基本設定]メニューの[シミュレータ]の[システム]を選択して、[シミュレータシステム]ダイアログボックスを表示してください。[シミュレータシステム]ダイアログボックスで、[有効]チェックボックスをオンにし、I/O シミュレーションアドレスを設定してください。

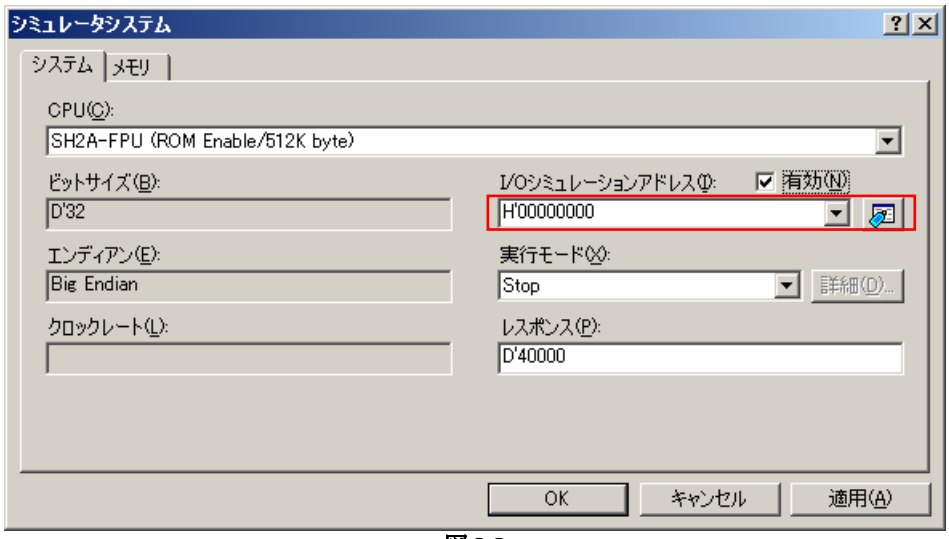


図 2-3

[シミュレータシステム]ダイアログボックスの[I/O シミュレーションアドレス]とプログラム上の I/O シミュレーションアドレスが一致していなければ I/O シミュレーション機能は動作しません。例えば添付したサンプルプログラムは I/O シミュレーションアドレスを H'00000000 としています。CPU が SH-4/SH-4A の場合は、I/O シミュレーションアドレスのデフォルトは H'00000004 に設定されます。したがって、SH-4/SH-4A でサンプルプログラムを使用する場合は、[シミュレータシステム]ダイアログボックスの[I/O シミュレーションアドレス]を変更するか、もしくは、プログラム上の I/O シミュレーションアドレスを修正する必要があります。

(2) I/O シミュレーションウィンドウを表示

I/O シミュレーションを使用するには、[I/O シミュレーション]ウィンドウを表示する必要もあります。[表示]メニュー[CPU]で[I/O シミュレーション]を選択し、I/O シミュレーションを表示してください。なお、ファイル入出力を使用する場合も、[I/O シミュレーション]ウィンドウを表示してください。

図 2-4

(3) I/O シミュレーションウィンドウの動作

低水準インタフェースルーチンが I/O シミュレーション用に実装されていると、アプリケーションプログラムからの標準出力が本ウィンドウに出力されます。また、本ウィンドウ上でのキー入力がアプリケーションプログラムへの標準入力となります。

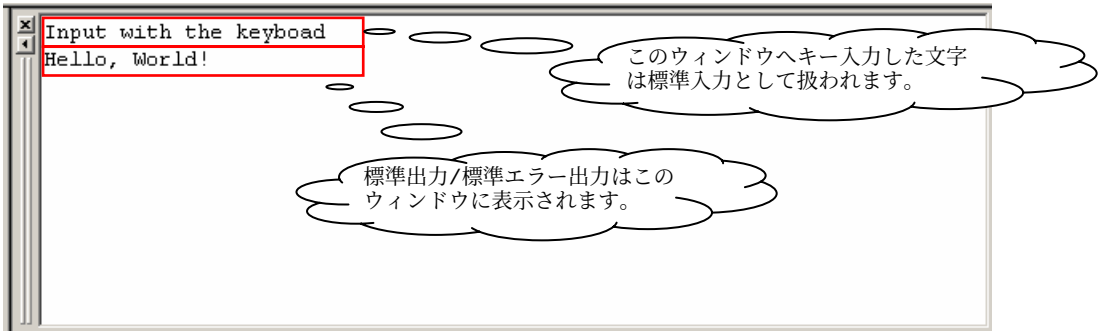


図 2-5

2.4 サンプルプログラム

サンプルプログラムでは、I/O シミュレーションでファイル／標準入出力を確認できるように、lowsrc.src、lowsrc.c を実装しています。lowsrc.c では、標準ライブラリ関数で使用する低水準インタフェースルーチンの open()、close()、read()、write()、lseek()を実装しています。また、lowsrc.src では低水準インタフェースルーチンから呼び出される I/O シミュレーション関数を実装しています。

2.4.1 ソースファイル

(1) lowsrc.src

I/O シミュレーション関数はアセンブラで記述する必要があります。lowsrc.src で、I/O シミュレーション関数 i_o_simulation を定義しています。この I/O シミュレーション関数を低水準インタフェースルーチン open()、close()、read()、write()、lseek()で使用し、I/O シミュレーション機能を呼び出しています。

i_o_simulation は、I/O シミュレーション機能を呼び出す前に次の処理を行います。

- 第 1 引数に指定された機能コードを R0 に設定
- 第 2 引数に指定されたパラメータブロックのアドレスを R1 に設定

(2) lowsrc.c

lowsrc.c で、低水準インタフェースルーチンの open()、close()、read()、write()、lseek()および標準ライブラリの初期化プログラム_INIT_IOLIB()、_CLOSEALL()を実装しています。

表 2-2 lowsrc.src で定義した関数

関数名	機能	処理
open	ファイルオープン	FOPEN の機能コードを指定して i_o_simulation 関数を呼び出します。stdin が指定された場合は指定されたオープンモードを無視し、リードオンリーでファイルオープンします。stdin 以外が指定された場合は、オープンモードにしたがってファイルオープンします。
close	ファイルクローズ	FCLOSE の機能コードを指定して i_o_simulation 関数を呼び出し、ファイルをクローズします。
read	データの入力	stdin が指定された場合は、GETC の機能コードを指定して i_o_simulation 関数を呼び出し、標準入力からの入力を読み込みます。stdin 以外が指定された場合は、FGETC の機能コードを指定して i_o_simulation 関数を呼び出し、指定されたファイルから文字列を読み込みます。
write	データの出力	stdout/stderr が指定された場合、PUTC の機能コードを指定して i_o_simulation 関数を呼び出し、指定された文字列を標準出力に出力します。stdout/stderr 以外を指定した場合は、FPUTC の機能コードを指定して i_o_simulation 関数を呼び出し、指定されたファイルに文字列を出力します。
lseek	指定位置にファイルポインタを移動	FSEEK の機能コードを指定して i_o_simulation 関数を呼び出し、ファイルポインタを指定位置に移動します。その後、FTELL の機能コードを指定して i_o_simulation 関数を呼び出し、現在のオフセット値を取得します。
_INIT_IOLIB	標準入出力の初期設定	stdin/stdout/stderr をオープンします。
_CLOSEALL	全てのファイルのクローズ	オープンしている全てのファイル(stdin/stdout/stderr を含む)をクローズします。

【注意】

サンプルプログラムでは、main()終了後に_CLOSEALL()関数を呼び出し、全てのファイルのクローズ処理をしています。プログラム実行時に、クローズ処理を呼び出される前に、リセット後実行をすると、_CLOSEALL()関数が呼ばれないため、意図した動作にならない場合があります。_INIT_IOLIB()、open()関数を使用した場合は、_CLOSEALL()関数を呼び出してからリセット後実行をしてください。もし、_CLOSEALL()関数を呼ばずにデバッガをリセットした場合は、[ファイル]メニューの[セッションのリフレッシュ]を選択してください。

2.4.2 メイン処理の説明

サンプルプログラムのメイン関数は、標準入力へ入力された文字列を scanf()関数で読み込み、その読み込んだ文字列を printf()関数で標準出力へ出力しています。その後、ファイル(C:¥WorkSpace¥sample¥file.txt) を fopen()関数でオープンし、fscanf()関数でファイルから文字列を読み込み、その読み込んだ文字列を printf()関数で標準出力に出力します。

```
#include <stdio.h>

unsigned char buf[100];

void main(void)
{
    FILE* fp;

    scanf("%s", buf);
    printf("Input=%s¥n", buf);

    fp = fopen("C:¥¥WorkSpace¥¥sample¥¥file.txt", "r");
    if ( fp != NULL ) {
        fscanf(fp, "%s", buf);
        printf("File=%s¥n", buf);
        fclose(fp);
    }
}
```

標準ライブラリの scanf()、printf()、fopen()、fscanf()、fclose()関数ではそれぞれ以下のように低水準インタフェースルーチンを呼び出します。

- scanf()： 標準入力(stdin)に対し read()関数を呼び出します。
- printf()： 標準出力(stdout)に対し write()関数を呼び出します。
- fopen()： fopen()に指定されたファイルに対し open()関数を呼び出します。
- fscanf()： fscanf()に指定されたファイルに対し read()関数を呼び出します。
- fclose()： fclose()に指定されたファイルポインタに対し close()関数を呼び出します。

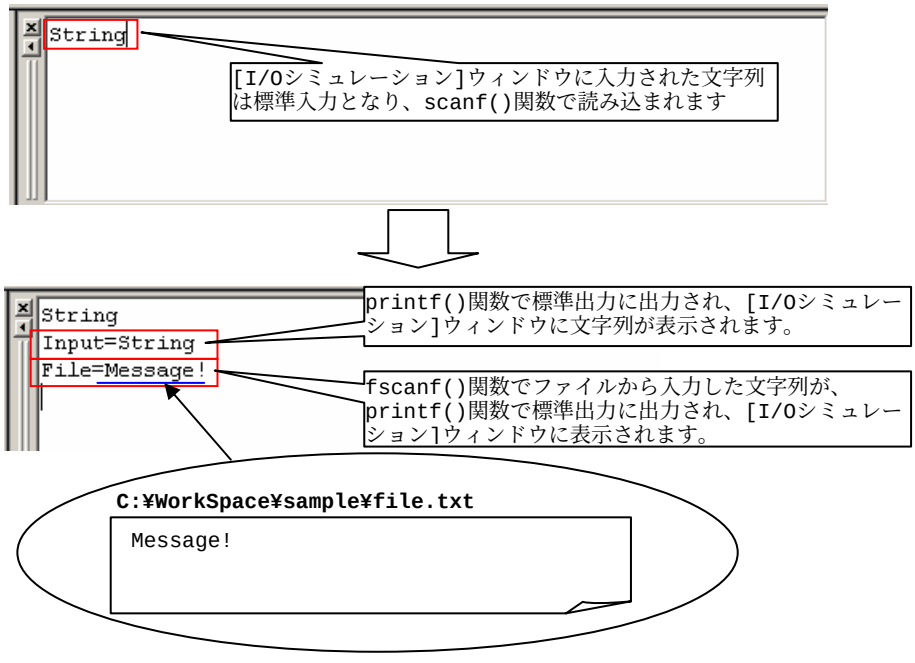


図 2-6

【注意】

I/O シミュレーションでファイル入出力を確認する場合は、ファイルオープン(fopen())で指定するファイル名は絶対パスで指定してください。なお、絶対パスに指定されるディレクトリの区切り文字は”¥”ではなく”¥¥”で記述してください。

3. 画像表示

3.1 概要

画像表示機能を使用して、プログラム実行中の画像データの内容を視覚的に確認するこができます。[表示]メニューの[グラフィック]の[画像]を選択し、[画像プロパティ]ダイアログボックスを表示してください。

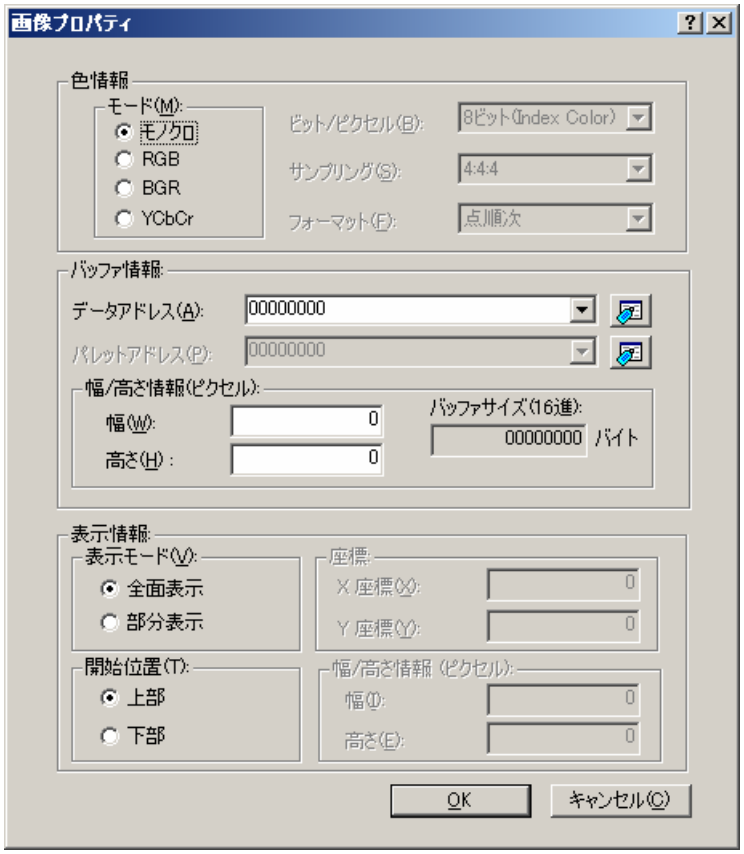


図 3-1

[画像プロパティ]ダイアログボックスで、 [バッファ情報]にバッファのアドレスおよびサイズを指定し、[色情報] にバッファが持つ画像データの形式を指定すると、[画像]ウィンドウにプログラム実行中の画像データが表示されます。表示することができる画像の形式は次節で説明します。

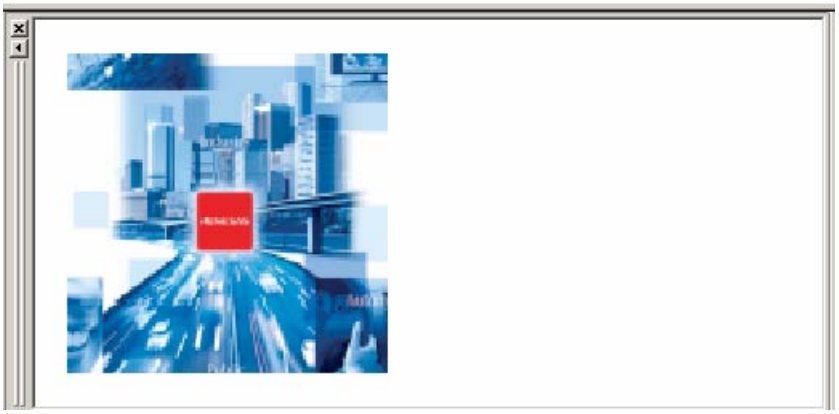


図 3-2

プログラム実行中の画像をリアルタイムに更新したい場合は、[画像]ウィンドウで右クリックし、ポップアップメニューを表示して[自動更新]の[リアルタイム]を選択してください。

図 3-3

3.2 対応している画像の形式

[画像プロパティ]ダイアログボックスでは次の画像形式を選択できます。

表 3-1 画像形式一覧 - モノクロ/RGB/BGR

色情報		データ	データイメージ
モード	ビット/ピクセル		
モノクロ	-	1 ピクセルに対して 1 ビット ビットデータ 1:白 0:黒	ビットデータ:B'10100000 □■□■■■■■
RGB/BGR	8 ビット(Index Color)	256 色(RGB/BGR32 ビットで表現)のパレットテーブルを用意し、画像データはパレットテーブルを参照する番号(0～255)を格納します。 パレットテーブルのピクセルデータは青:8 緑:8 赤:8 α:8 ビットで表現します。 画像表示機能では、透明度情報の α 値を無視します。	ピクセルデータ: H'05 パレットテーブルから 6 番目の色を参照 パレットテーブルの 6 番目のピクセルデータ H'E7 H'C3 H'81 H'00 青:231 緑:195 赤:129 α:0 231 195 129 ---
	16 ビット(5:5:5)	RGB: 赤:5 緑:5 青:5 BGR: 青 5 緑:5 赤:5 最上位ビットはパディング	ピクセルデータ H'4677(=B'0100011001110111) RGB: 赤:17 緑 19 青:23 17 19 23 BGR: 青:17 緑 19 赤:23 17 19 23
	16 ビット(5:6:5)	RGB: 赤:5 緑:6 青:5 ビット BGR: 青 5 緑:6 赤:5 ビット 緑の最上位ビットはパディング	ピクセルデータ H'8A77(=B'1000101001110111) RGB: 赤:17 緑 19 青:23 17 19 23 BGR: 青:17 緑 19 赤:23 17 19 23
	24 ビット	RGB: 赤:8 緑:8 青:8 ビット BGR: 青 8 緑:8 赤:8 ビット	ピクセルデータ H'81 H'C3 H'E7 RGB: 赤:129 緑 195 青:231 129 195 231 BGR: 青:129 緑 195 赤:231 129 195 231

	32 ビット	RGB: α:8 赤:8 緑:8 青:8 ビット BGR: α:8 青 8 緑:8 赤:8 ビット 画像表示機能では、透明度情報の α 値を無視します。	ピクセルデータ H'00 H'81 H'C3 H'E7 RGB: α:0 赤:129 緑 195 青:231  BGR: α:0 青:129 緑 195 赤:231 

表 3-2 画像形式一覧(2) - YCbCr

色情報		データ		データイメージ
モード	サンプリング比率	元のデータ	サンプリングデータ	
YCbCr	4:4:4	11,12,13,14 21,22,23,24 31,32,33,34 41,42,43,44	11,12,13,14 21,22,23,24 31,32,33,34 41,42,43,44	点順次 $Y_{11},Cb_{11},Cr_{11},Y_{12},Cb_{12},Cr_{12},Y_{13},Cb_{13},Cr_{13},$ $...,Y_{nm},Cb_{nm},Cr_{nm}$ 面順次 $Y_{11},Y_{12},Y_{13},...,Y_{nm},Cb_{11},Cb_{12},Cb_{13},...,Cb_{nm},$ $Cr_{11},Cr_{12},Cr_{13},...,Cr_{nm}$ 面順次 2 $Y_{11},Y_{12},Y_{13},...,Y_{nm},$ $Cb_{11},Cr_{11},Cb_{12},Cr_{12},Cb_{13},Cr_{13},...,Cb_{nm},Cr_{nm}$
	4:2:2	11,12,13,14 21,22,23,24 31,32,33,34 41,42,43,44	11,11,13,13 21,21,23,23 31,31,33,33 41,41,43,43	点順次 $Y_{11},Y_{12},Cb_{11},Cr_{11},Y_{13},Y_{14},Cb_{13},Cr_{13},$ $...,Y_{n(m-1)},Y_{nm},Cb_{n(m-1)},Cr_{n(m-1)}$ 面順次 $Y_{11},Y_{12},Y_{13},...,Y_{nm},Cb_{11},Cb_{13},Cb_{15},...,Cb_{n(m-1)},$ $Cr_{11},Cr_{13},Cr_{15},...,Cr_{n(m-1)}$ 面順次 2 $Y_{11},Y_{12},Y_{13},...,Y_{nm},$ $Cb_{11},Cr_{11},Cb_{13},Cr_{13},Cb_{15},Cr_{15},...,Cb_{n(m-1)},Cr_{n(m-1)}$
	4:1:1	11,12,13,14 21,22,23,24 31,32,33,34 41,42,43,44	11,11,11,11 21,21,21,21 31,31,31,31 41,41,41,41	点順次 $Y_{11},Y_{12},Y_{13},Y_{14},Cb_{11},Cr_{11},$ $Y_{21},Y_{22},Y_{23},Y_{24},Cb_{23},Cr_{21},$ $...,Y_{n(m-3)},Y_{n(m-2)},Y_{n(m-1)},Y_{nm},Cb_{n(m-3)},Cr_{n(m-3)}$ 面順次 $Y_{11},Y_{12},Y_{13},...,Y_{nm},Cb_{11},Cb_{13},Cb_{15},...,Cb_{n(m-1)},$ $Cr_{11},Cr_{13},Cr_{15},...,Cr_{n(m-1)}$ 面順次 2 $Y_{11},Y_{12},Y_{13},...,Y_{nm},$ $Cb_{11},Cr_{11},Cb_{13},Cr_{13},Cb_{15},Cr_{15},...,Cb_{n(m-1)},Cr_{n(m-1)}$
	4:2:0	11,12,13,14 21,22,23,24 31,32,33,34 41,42,43,44	11,11,13,13 11,11,13,13 31,31,33,33 31,31,33,33	面順次 $Y_{11},Y_{12},Y_{13},...,Y_{nm},$ $Cb_{11},Cb_{13},Cb_{15},...,Cb_{n(m-1)},$ $Cr_{11},Cr_{13},Cr_{15},...,Cr_{n(m-1)}$ 面順次 2 $Y_{11},Y_{12},Y_{13},...,Y_{nm},$ $Cb_{11},Cr_{11},Cb_{13},Cr_{13},Cb_{15},Cr_{15},...,Cb_{n(m-1)},Cr_{n(m-1)}$

3.3 サンプルプログラムの説明

サンプルプログラムでは、2 つの画像データ data0、data1 を交互にバッファ buf にコピーしています。使用している画像データは、サイズ 200 ピクセル×200 ピクセルで 24 ビット RGB 形式です。

```
#define SIZE (120000)

extern const unsigned char data0[SIZE]; /* Image Data0 */
extern const unsigned char data1[SIZE]; /* Image Data1 */
unsigned char buf[SIZE]; /* Buffer */

void main(void)
{
    int i;

    for(;;)
    {
        for(i=0; i < SIZE; i++ ) {
            buf[i] = data0[i]; /* Image Data0 */
        }
        for(i=0; i < SIZE; i++ ) {
            buf[i] = data1[i]; /* Image Data1 */
        }
    }
}
```

[表示]メニューの[グラフィック]の[画像]を選択し、[画像プロパティ]ダイアログボックスを表示してください。[画像プロパティ]ダイアログボックスで次のように設定します。

- [色情報]
  - [モード]に RGB を選択
  - [ビット/ピクセル]ドロップリストから”24 ビット”を選択
- [バッファ情報]
  - [データアドレス]フィールドの右にあるボタンをクリックして、[ラベル選択]ダイアログボックスを表示し、そこで “_buf” を選択
  - [幅/高さ情報(ピクセル)]にそれぞれ 10 進数で[幅]を “200”、[高さ]を “200”に設定

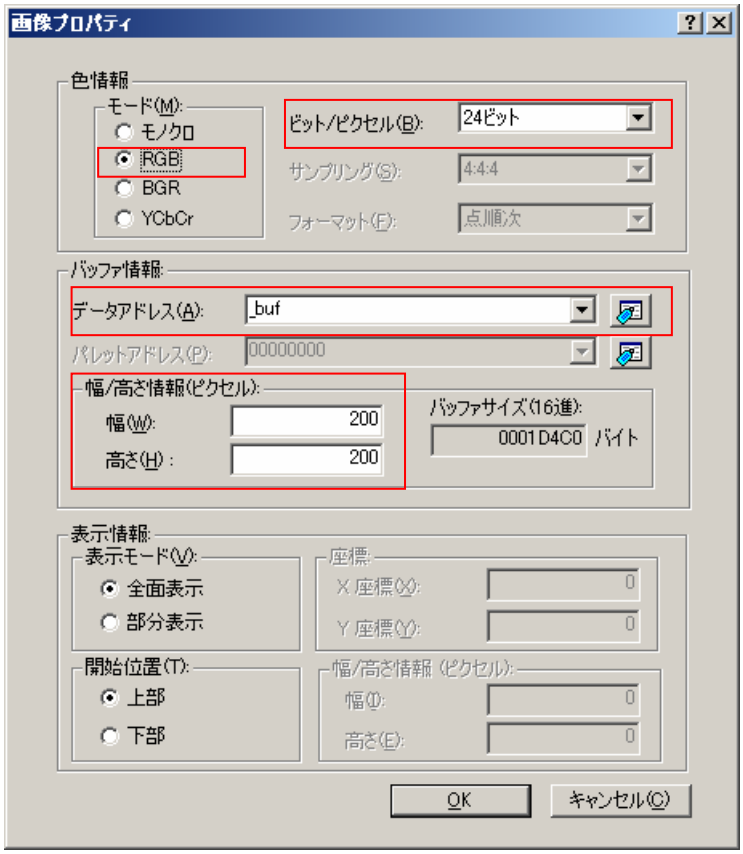


図 3-4

[画像]ウィンドウに画像が表示されます。なお、画像をリアルタイムに更新する場合は、[画像]ウィンドウで右クリックし、ポップアップメニューを表示して[自動更新]の[リアルタイム]を選択してください。2つの画像データが交互に切り替わる様子がわかります。

図 3-5

4. プロファイル

4.1 概要

シミュレータにはアプリケーションプログラムの実行性能を測定するためにプロファイル機能およびパフォーマンス解析機能があります。プロファイル機能とパフォーマンス解析機能のそれぞれの特徴を下記に示します。

表 4-1 プロファイル機能とパフォーマンス解析機能の特徴

項目	パフォーマンス解析機能	プロファイル機能
測定範囲	特定関数	全関数
取得可能な実行性能項目	少ない	多い
子関数の測定	不可能	可能
シミュレーション速度	速い	遅い

(1) パフォーマンス解析機能

パフォーマンス解析機能はアプリケーションプログラム内の特定関数に対して、次の項目の測定を行います。

- 実行サイクル数
- 呼び出し回数
- プログラム全体の実行サイクル数に占める当該関数の実行サイクル数の割合
- 上記割合のヒストグラム

特定関数のみ測定を行うため、プロファイルよりも高速なシミュレーションが可能です。シミュレーションに時間の掛かる大規模なアプリケーションを測定したい場合などに使用してください。

なお、子関数については測定をしません。子関数の測定をしたい場合は、子関数もパフォーマンス解析の測定対象に含めてください。

(2) プロファイル機能

プロファイル機能は、アプリケーションプログラム内の全関数に対し、様々な実行性能項目を測定します。アプリケーションプログラム中の性能劣化の原因となっている場所および要因を詳細に調査することができます。また、子関数の実行結果を含んで表示することもできます。

取得可能な実行性能項目は「SuperH^{TM} RISC engine シミュレータ／デバッガ ユーザーズマニュアル 3.6.10 表示データの種類および用途」を参照してください。

4.2 使用方法

パフォーマンス解析機能およびプロファイル機能のそれぞれの使用方法を説明します。

(1) パフォーマンス解析機能

[表示]メニューの[パフォーマンス]の[パフォーマンス解析]を選択し、[パフォーマンス解析]ウィンドウを表示してください。[パフォーマンス解析]ウィンドウで右クリックし、ポップアップメニューを表示して、[範囲の追加]を選択してください。[パフォーマンスオプション]ダイアログボックスが表示されます。[パフォーマンスオプション]ダイアログボックスで測定対象とする関数名を指定してください。さらに、ポップアップメニューで[有効]を選択します(ポップアップメニューにチェックマークが付きます)。

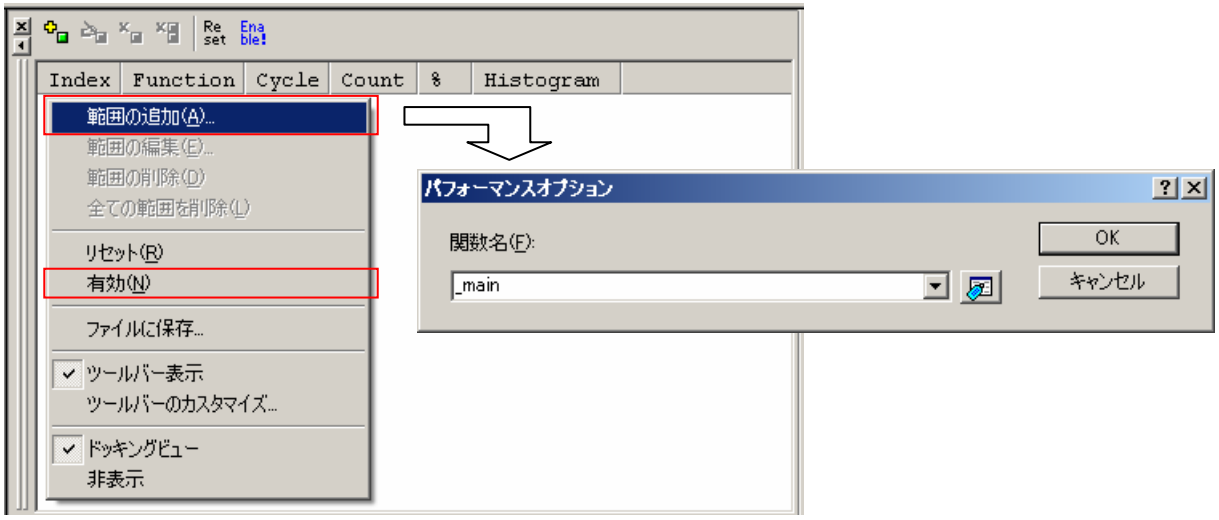


図 4-1

プログラムを実行すると指定関数の測定を行います。測定した結果は下記の図のように表示されます。

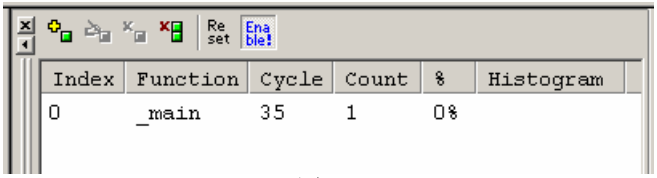


図 4-2

表示される項目は次の通りです。

[Index]	設定条件のインデックス番号
[Function]	測定対象の関数名（または関数の開始アドレス）
[Cycle]	当該関数の累計実行サイクル数
[Count]	当該関数の累計呼び出し回数
[%]	プログラム全体の実行サイクル数に占める当該関数の実行サイクル数の割合
[Histogram]	上記割合のヒストグラム表示

(2) プロファイル機能

[表示]メニューの[パフォーマンス]の[プロファイル]を選択し、[プロファイル]ウィンドウを表示してください。[プロファイル]ウィンドウで右クリックし、ポップアップメニューを表示して、[有効]を選択します(ポップアップメニューにチェックマークが付きます)。

なお、子関数の実行結果を含めて表示する場合は、ポップアップメニューの[表示設定]の[子関数の実行結果を含んで表示]を選択してください(ポップアップメニューにチェックマークが付きます)。

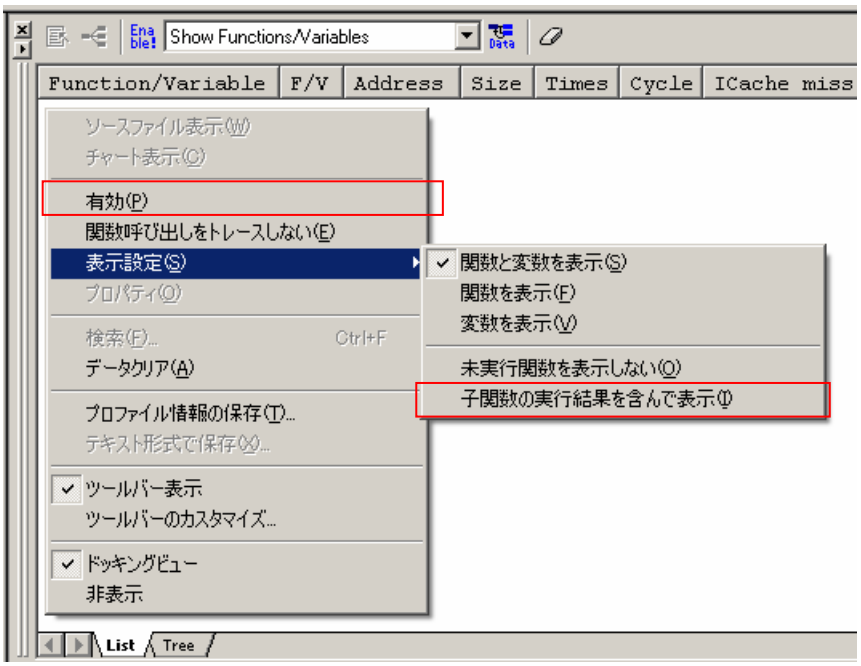


図 4-3

プログラムを実行して関数および変数の情報を取得します。[Times]に関数の実行回数もしくは変数のアクセス回数が、[Cycle]に実行サイクル数が表示されます。その他の表示項目については、「SuperH™ RISC engine シミュレータ／デバッガ ユーザーズマニュアル　3.6.10 表示データの種類および用途」を参照してください。

[プロファイル]ウィンドウには、[List]シートと[Tree]シートがあります。それぞれ下記の図のように表示されます。

- List
  - 測定結果をリスト表示します。

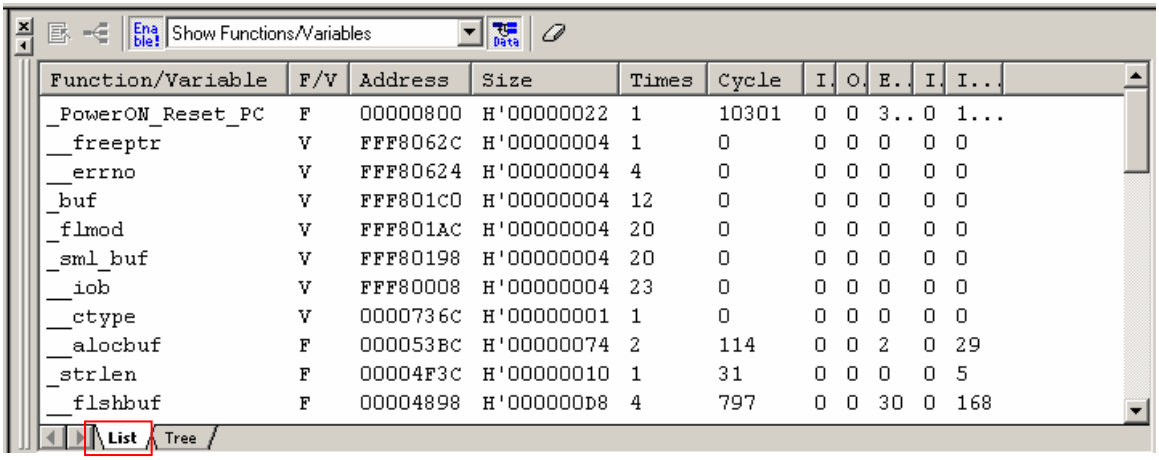


図 4-4

- Tree
  – [Function]列をツリー形式にして、測定結果を表示します。
    [Function]列の関数をダブルクリックして、ツリー構造を拡張または収縮表示します。

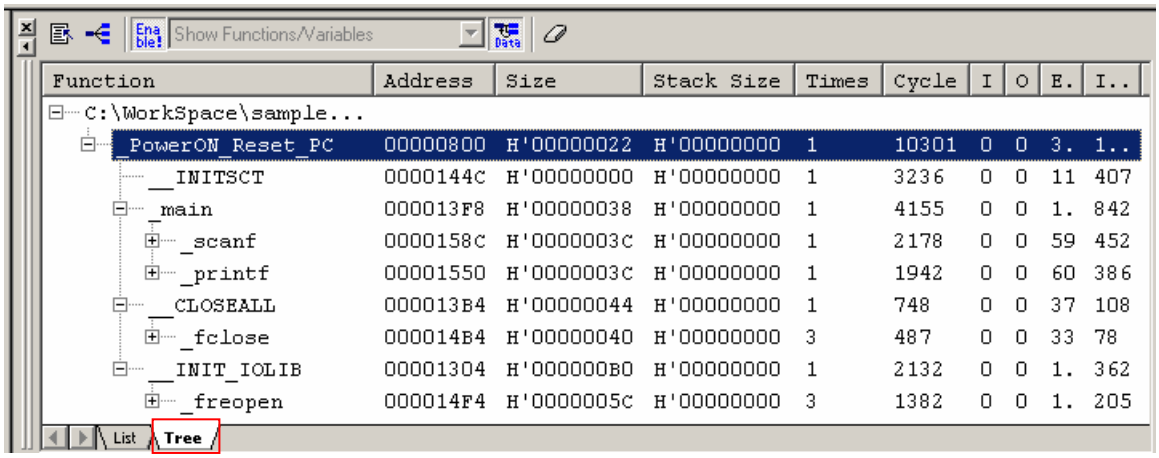


図 4-5

[プロファイル]ウィンドウの測定結果より、特定の関数に着目した関数の呼び出し関係を表示することができます。
図 4-4、もしくは 図 4-5で関数を選択し、右クリックでポップアップメニューを表示してください。 ポップアップメニューで[チャート表示]を選択すると、[プロファイルチャート]ウィンドウが表示されます。
[プロファイルチャート]ウィンドウには、中央に着目する関数を表示し、その左側には着目した関数を呼び出した関数、右側には、着目している関数が呼び出した関数を、それぞれ表示します。また、各関数の呼び出し回数も表示します。

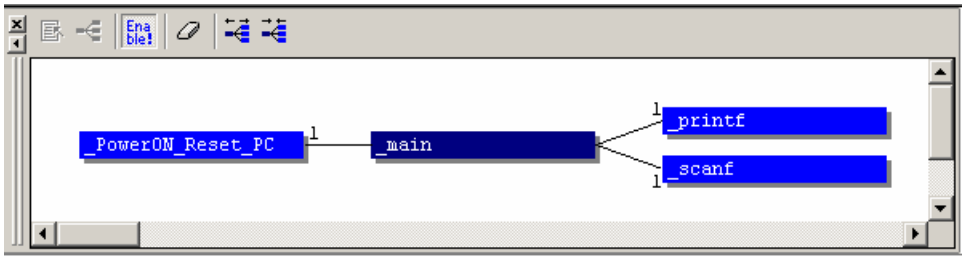


図 4-6

4.3 サンプルプログラムの説明

サンプルプログラムとして用意した 3 種類のソートプログラム(クイックソート:quicksort()、ヒープソート:heapsort()およびバブルソート:bubblesort()) をパフォーマンス解析機能およびプロファイル機能を使用して性能測定する例を示します。

なお、ソートするデータに偏りがない場合の効率は、クイックソート > ヒープソート > バブルソート となります。

- メイン処理

```
#include <string.h>

extern void quicksort(int n, int a[]);
extern void bubblesort(int n, int a[]);
extern void heapsort(int n, int a[]);

#define N (80)
const int array[N] = {
    0, 690, 505, 591, 554, 378, 257, 207, 626, 340,
  843,  68, 409, 879, 319, 980,  85, 907, 102, 921,
  507, 872, 333, 692, 556, 361,  31, 858,  98, 877,
  449, 432, 606, 927, 664, 395, 438, 652, 928, 949,
  307, 596, 783, 338, 805, 942,  66, 857, 977, 889,
  545, 864, 457, 800, 873, 821, 185,  86, 638, 233,
  462,   7, 635, 421, 953, 210, 970, 261, 857, 581,
  707, 285, 318, 643, 858, 668, 443,  55, 777, 594,
};

void main(void)
{
    int dist1[N];
    int dist2[N];
    int dist3[N];

    memcpy(dist1, array, sizeof(int) * N);
    quicksort(N, dist1);

    memcpy(dist2, array, sizeof(int) * N);
    bubblesort(N, dist2);

    memcpy(dist3, array, sizeof(int) * N);
    heapsort(N, dist3);
}
```

- クイックソート: quicksort()

```
/* Quick Sort */
void qsort(int a[], int first, int last);

void quicksort(int n, int a[])
{
    qsort(a, 0, n-1);
}

void qsort(int a[], int first, int last)
{
    int i, j;
    int pivot;
    int tmp;

    pivot = a[(first + last) / 2];
    i = first;  j = last;
    for (;;) {
        while (a[i] < pivot) i++;
        while (pivot < a[j]) j--;
        if (i >= j) break;
        tmp = a[i]; a[i] = a[j]; a[j] = tmp;
        i++;  j--;
    }
    if (first < i - 1) qsort(a, first , i - 1);
    if (j + 1 < last) qsort(a, j + 1, last);
}
```

- ヒープソート: heapsort()

```
/* Heap Sort */
void heapsort(int n, int a[])
{
    int i, j, k;
    int x;

    for (k=n/2; k >= 1; k--) {
        i = k;  x = a[i];
        while ((j=2*i) <= n) {
            if (j < n && a[j] < a[j + 1]) j++;
            if (x >= a[j]) break;
            a[i] = a[j];  i = j;
        }
        a[i] = x;
    }
    while (n > 1) {
        x = a[n];  a[n] = a[1];  n--;
        i = 1;
        while ((j=2*i) <= n) {
            if (j < n && a[j] < a[j + 1]) j++;
            if (x >= a[j]) break;
            a[i] = a[j];  i = j;
        }
        a[i] = x;
    }
}
```

- バブルソート: bubblesort()

```
/* Bubble Sort */
void bubblesort(int n, int a[])
{
    int i, j, k;
    int tmp;

    k = n - 1;
    while (k >= 0) {
        j = -1;
        for (i=1; i <= k; i++)
            if (a[i - 1] > a[i]) {
                j = i - 1;
                tmp = a[j]; a[j] = a[i]; a[i] = tmp;
            }
        k = j;
    }
}
```

(1) パフォーマンス解析機能

[パフォーマンス解析]ウィンドウで測定する関数を設定してください、測定する関数は、クイックソート:quicksort()、ヒープソート:heapsort()、バブルソート:bubblesort()およびquicksort()から呼び出すqsort()です。設定方法は「4.2使用方法(1) パフォーマンス解析機能」を参照してください。

プログラムを実行すると[パフォーマンス解析]ウィンドウに測定結果が出力されます。

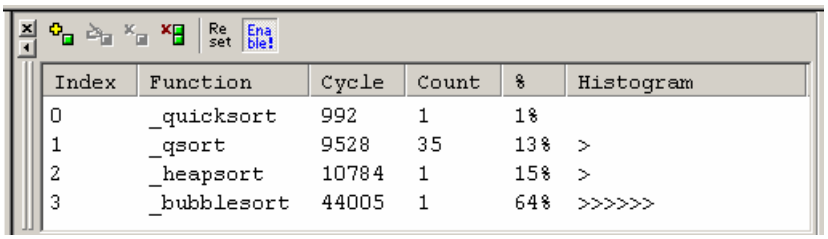


図 4-7

図 4-7より、各ソートのサイクル数が 表 4-2のようになることがわかります。

表 4-2 パフォーマンス解析機能測定結果

ソート名	関数	サイクル数
クイックソート	quicksort() + qsort()	10520
ヒープソート	heapsort()	10784
バブルソート	bubblesort()	44005

(2) プロファイル機能

[プロファイル]ウィンドウで、[子関数の実行結果も含んで表示]を有効にしてください。設定方法は「4.2使用方法(2) プロファイル機能」を参照してください。

プログラムを実行すると、[プロファイル]ウィンドウに測定結果が次のように出力されます。

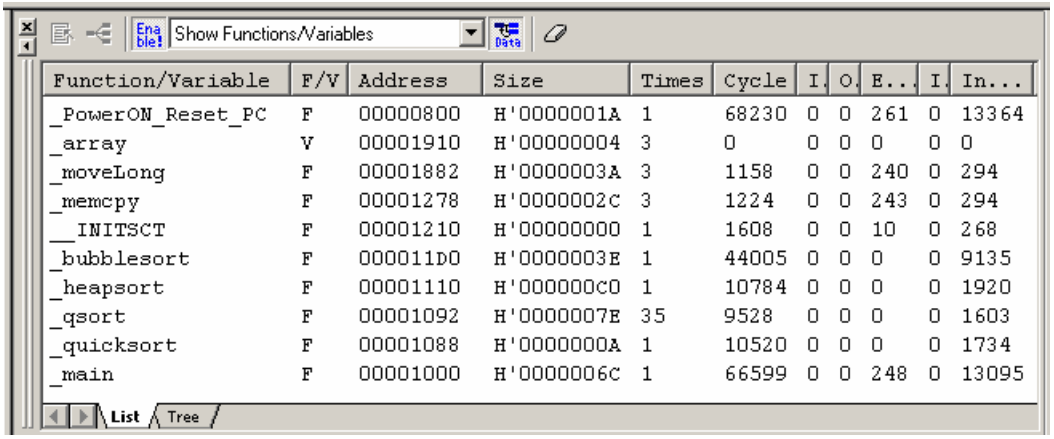


図 4-8

図 4-8より、各ソートのサイクル数が 表 4-3のようになることがわかります。

表 4-3 プロファイル機能測定結果

ソート名	関数	サイクル数
クイックソート	quicksort()	10520
ヒープソート	heapsort()	10784
バブルソート	bubblesort()	44005

5. 擬似割り込み

シミュレータのウィンドウ上での操作により、手動で擬似的に割り込みを発生させることができます。これにより、外部機器による割り込みなどシミュレータでは発生させることができない割り込みに対してもシミュレーションできます。本章では、擬似割り込みの使用方法を説明します。

5.1 使用方法

擬似割り込みを使用するためには、[トリガ]ウィンドウに配置されているボタン(トリガボタン)を使用します。ここでは、トリガボタンを使用して擬似割り込みを発生させる方法を説明します。まず、[表示]メニューの[CPU]の[トリガ]を選択 して[トリガ]ウィンドウを表示し、[トリガ]ウィンドウで右クリックしポップアップメニューを表示します。次に、そのポップアップメニューで[設定]を選択し、[トリガ設定]ダイアログボックスを表示してトリガボタンを設定します。

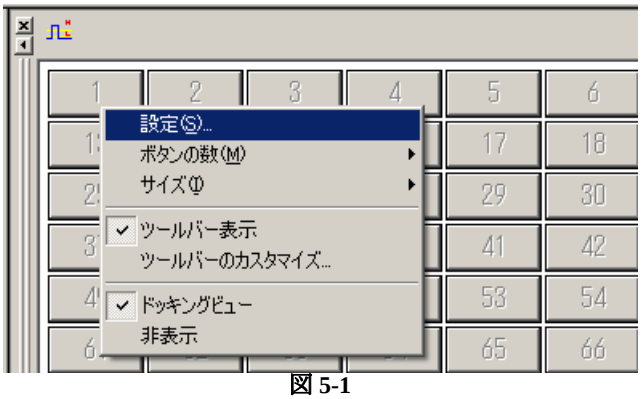


図 5-1

[トリガ設定]ダイアログボックスで、トリガボタンクリック時に発生する割り込みの内容を設定します。なお、ボタンは最大 256 個設定することができます。

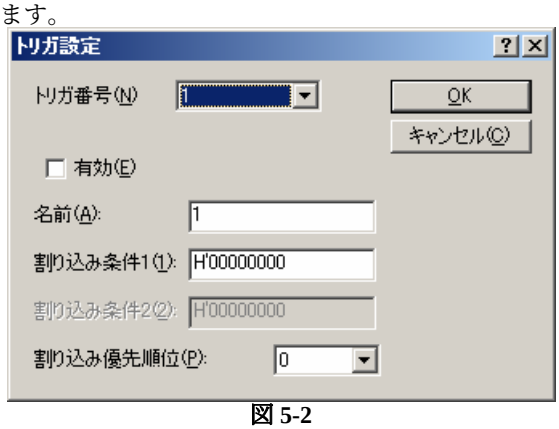


図 5-2

[トリガ設定]ダイアログボックスでの設定内容は以下の通りです。

[トリガ番号] 設定するトリガボタンを選択します。

[名前] [トリガ]ウィンドウに表示するトリガボタンの名前を指定します。

[有効] チェックするとトリガボタンが有効になります。

[割り込み条件 1] CPU ごとに下記を指定します。

・ SH-1、SH-2、SH2-DSP、SH2A-FPU シリーズ
割り込みベクタ番号

・ SH-3、SH-4、SH3-DSP シリーズ
INTEVT 値（0~H'FFF）

・ SH-4A シリーズ
INTEVT 値（0~H'3FFF）

[割り込み条件 2] SH3-DSP シリーズのみ下記を指定可能です。
INTEVT2 値（0~H'FFF）

[割り込み優先順位] 割り込み優先順位を指定します。（0～17）
16 指定時は、SR の I ビットによらず割り込みが発生しますが、SR の BL によってマスクします。
17 指定時は、SR の I ビット、BL ビットによらず、割り込みが発生します。

割り込み発生時にシミュレーションを停止するか、あるいはシミュレーションを実行し続けるか、[シミュレータシステム]ダイアログボックスで設定できます。 [基本設定]メニューの[シミュレータ]の[システム]を選択して、[シミュレータシステム]ダイアログボックスを表示してください。シミュレーションを実行し続ける場合は、[実行モード]ダウンリストを “Continue” に設定してください。シミュレーションを停止したい場合は、[実行モード]ダウンリストを “Stop” に設定した上で、[詳細]ボタンをクリックして表示される[停止設定]ダイアログボックスで、[停止要因]の “Interrupt Exception” チェックボックスをオンにしてください。

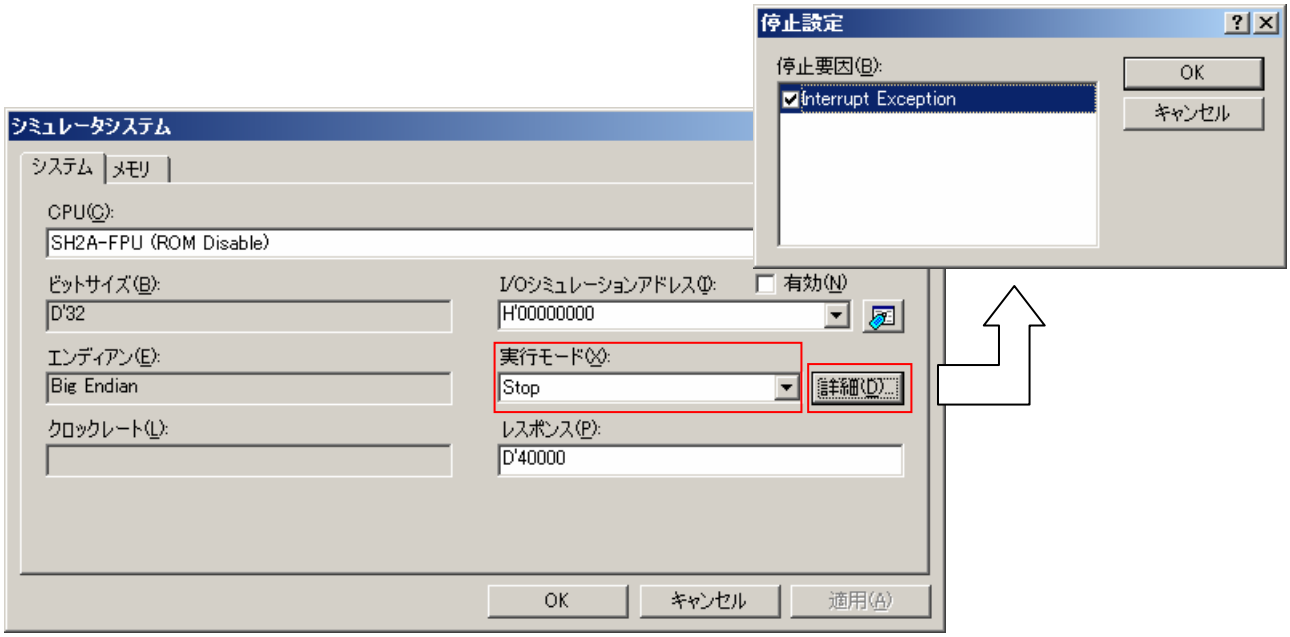


図 5-3

5.2 サンプルプログラム

SH-1、SH-2、SH-2A シリーズと SH-3、SH-4、SH-4A シリーズとでは、割り込みの仕様が異なります。また、 [トリガ設定] ダイアログボックスの設定も両者で異なります。SH-2A および SH-4 用のサンプルプログラムを用いて、それぞれの擬似割り込みを使用する方法を説明します。

5.2.1 SH-2A 用

SH-2A シミュレータで擬似割り込みを使用する方法を説明します。サンプルプログラムでは割り込みを受け付けるために、メイン関数で set_imask()を使用して、割り込みマスクを 0 に設定します。これにより、割り込み優先順位が 1 以上の割り込みを受け付けるようになります。そして、割り込みマスクの設定後は、無限ループのみの処理としています。割り込みが発生すると、割り込みベクタテーブルに登録されているベクタ番号に対応する関数が呼び出されます。

```
#include <machine.h>

void main(void)
{
    set_imask(0);

    for(;;) {
        nop();
    }
}
```

[トリガ設定] ダイアログボックスの [割り込み条件 1]には、割り込みベクタテーブルのベクタ番号を指定します。図 5-4 ではベクタ番号に H’00000040 を、割り込み優先順位に 5 を指定する例を示しています。サンプルプログラムでは、ベクタ番号H’00000040 に対応する関数は、INT_IRQ0()です。

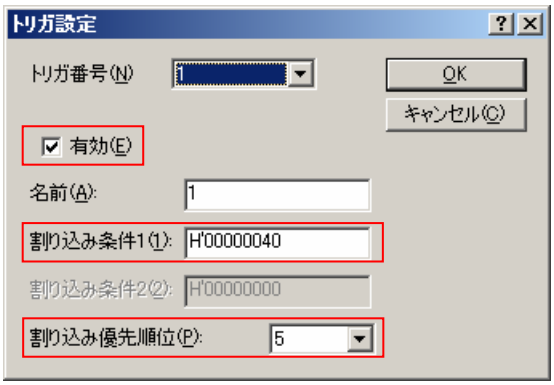


図 5-4

割り込みが発生したときにシミュレーションを停止するように、[シミュレータシステム]ダイアログボックスで設定します。設定方法は、「5.1使用方法」を参照してください。プログラム実行後、[トリガ]ウィンドウに設定したトリガボタンをクリックしてください。

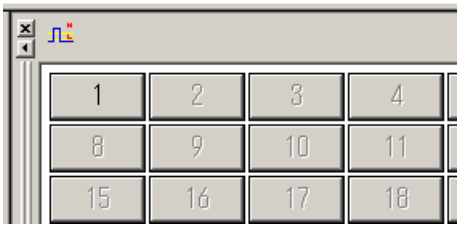


図 5-5

トリガボタンをクリックすると、擬似的に割り込みが発生し、ベクタ番号 H’00000040 に対応する INT_IRQ0()が呼び出されます。

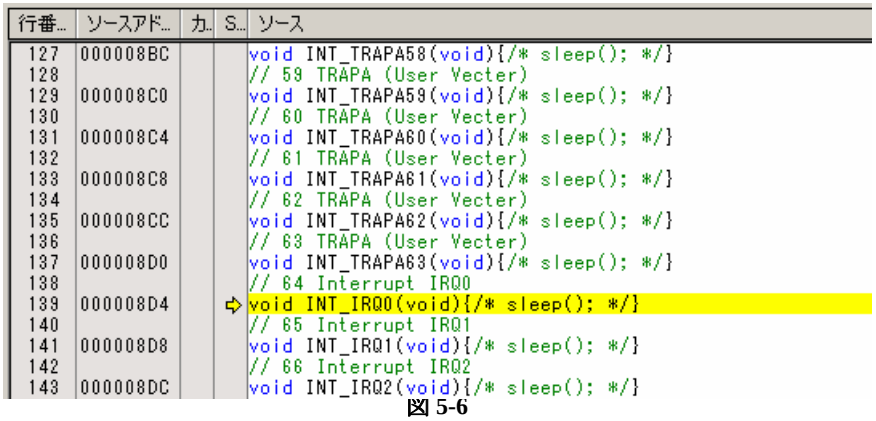


図 5-6

5.2.2 SH-4 用

SH-4 シミュレータで擬似割り込みを使用する方法を説明します。サンプルプログラムでは割り込みを受け付けるために、パワーオンリセット関数 PowerON_Reset()で set_cr ()を使用して、割り込みマスクを 0 に設定しています。これにより、割り込み優先順位が 1 以上の割り込みを受け付けるようになります。そして、割り込みマスクの設定後は、無限ループのみの処理としています。割り込みが発生すると、例外処理ハンドラ_IRQHandler が呼び出され、_ IRQHandler で INTEVT 値に対応する割り込み処理関数を呼び出すようにしています。

```
#include    <machine.h>
        .
        .
        .
#define SR_Init    0x00000000
        .
        .
        .
void PowerON_Reset(void)
{
        .
        .
        .
    set_cr(SR_Init);

    for(;;) {
        nop();
    }
        .
        .
        .
```

[トリガ設定] ダイアログボックスの [割り込み条件 1]には、INTEVT値を指定します。図 5-7では、INTEVT値にH’00000200 を、割り込み優先順位に 4 を指定する例を示しています。サンプルプログラムでは、INTEVT値がH’00000200 のときに、_ IRQHandler で割り込み処理関数INT_Extern_0000()を呼び出すようにしています。

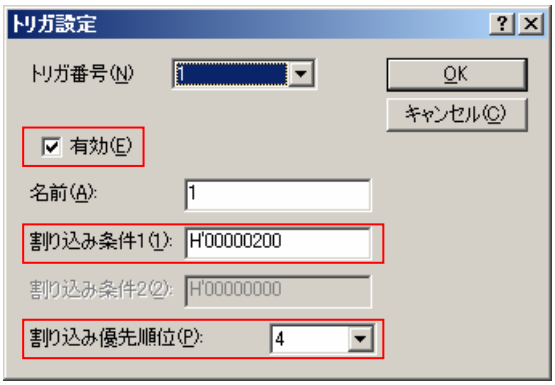


図 5-7

割り込みが発生したときにシミュレーションを停止するように、[シミュレータシステム]ダイアログボックスで設定します。設定方法は、「5.1使用方法」を参照してください。プログラム実行後、[トリガ]ウィンドウに設定したトリガボタンをクリックしてください。

図 5-8

トリガボタンをクリックすると割り込みが発生し、無限ループ中にシミュレーションが停止します。ここで、ステップイン(F11)をすると、例外処理ハンドラ_ IRQHandler が呼び出されることがわかります。さらにステップインを繰り返すと、INTEVT 値の H'00000200 に対応する割り込み処理関数 INT_Extern_0000()が呼び出されることがわかります。

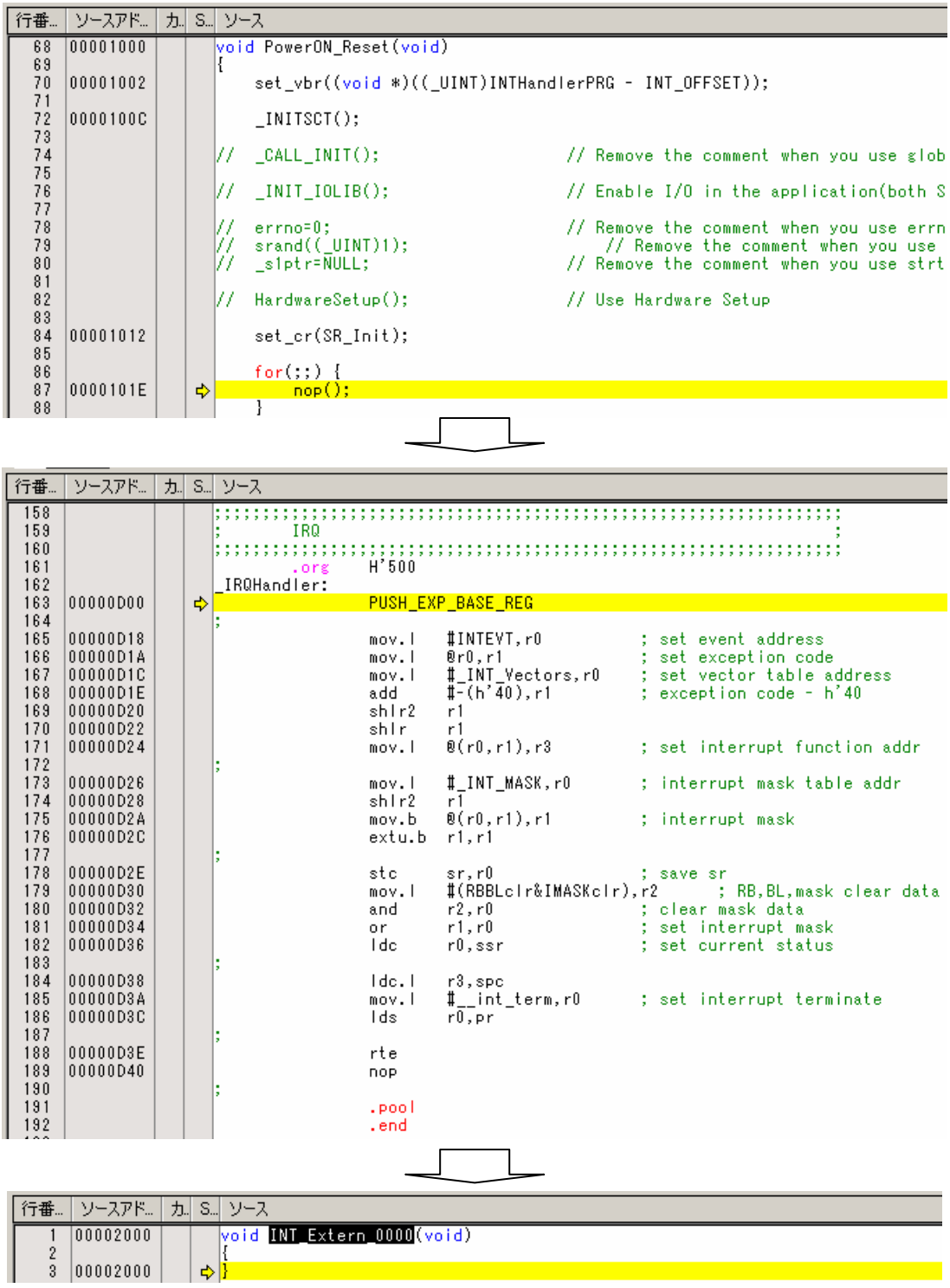


図 5-9

6. タイマシミュレーション

多くの組み込みアプリケーションではタイマ制御が使用されています。シミュレータではタイマ機能を一部シミュレーションしており、タイマを使用したアプリケーションのデバッグも可能です。

なお、シミュレータでサポートしているタイマはCPUにより異なります。シミュレータでサポートしているタイマは「SuperH™ RISC engine シミュレータ／デバッガ ユーザーズマニュアル 2.9.2 制御レジスタ」を参照してください。ただし、インプットキャプチャ等の端子入出力をともなう機能のシミュレーションは擬似割り込み(「5擬似割り込み」参照)を使用してください。また、デバイスが持つタイマの種類や、タイマの使用方法は使用するデバイスによって異なります。タイマの詳細は使用するデバイスのハードウェアマニュアルを参照してください。

6.1 使用方法

シミュレータでタイマ制御をシミュレーションするには、使用するタイマの周辺機能シミュレーションモジュールを登録する必要があります。また、使用するデバイスのレジスタアドレスがシミュレータのデフォルトの設定と異なる場合は、周辺機能シミュレーションモジュールのレジスタアドレスの変更が必要です。以下に、周辺機能シミュレーションモジュールの登録方法、および周辺機能シミュレーションモジュールのレジスタアドレス変更方法を説明します。

(1) 周辺機能シミュレーションモジュールの登録

周辺機能シミュレーションモジュールの登録は、シミュレータ起動時に表示される[周辺機能シミュレーションの設定]ダイアログボックスより行います。この[周辺機能シミュレーションの設定]ダイアログボックスの[周辺機能]一覧で、使用するタイマのチェックボックスをオンにしてください。

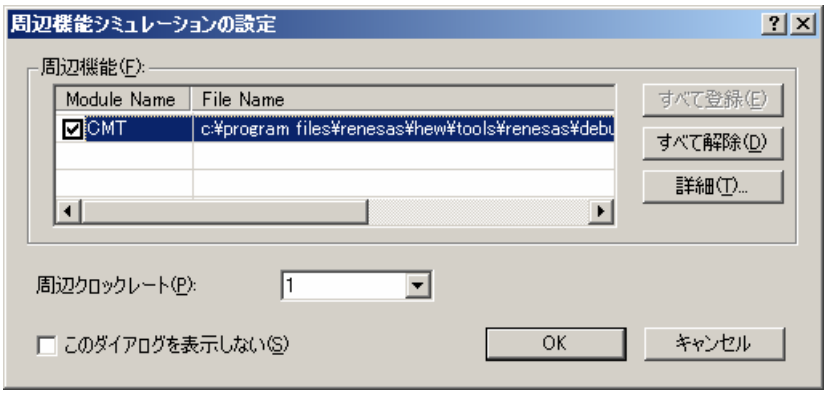


図 6-1

なお、[周辺機能シミュレーションの設定]ダイアログボックスの各項目の内容は以下の通りです。

[周辺機能] 周辺機能シミュレーションモジュールの情報を表示します。
- [Module Name] シミュレーションする周辺機能名
- [File Name] 周辺機能シミュレーションモジュールファイル名
- [Module name] 欄のチェックボックスをチェックした周辺機能シミュレーションモジュールが登録されて、利用可能となります。

[すべて登録] すべての周辺機能を有効にします。
[すべて無効] すべての周辺機能を無効にします。
[詳細…] 周辺機能情報の表示、周辺機能の開始アドレス、および割り込み要因情報の変更を行うための[周辺モジュールの構成]ダイアログボックスを表示します。
[周辺クロックレート] 周辺クロックと内部クロックの比(周辺 1 クロックが内部クロックいくつに相当するか)を指定します(1,2,3,4,6,8,12,16,24,32 から選択)。

[周辺機能シミュレーションの設定]ダイアログボックスの[このダイアログを表示しない]チェックボックスをオンにすると、シミュレータ起動時に[周辺機能シミュレーションの設定]ダイアログボックスが表示されなくなります。再度表示する場合は、[基本設定]メニューの[オプション]を選択し[オプション]ダイアログボックスを表示して、[確認]タブを選択してください。[確認]タブの[確認ダイアログボックスの表示]一覧で“起動時に周辺機能シミュレーションの設定ダイアログを表示”チェックボックスをオンすると、シミュレータ起動時に[周辺機能シミュレーションの設定]ダイアログボックスが表示されます。

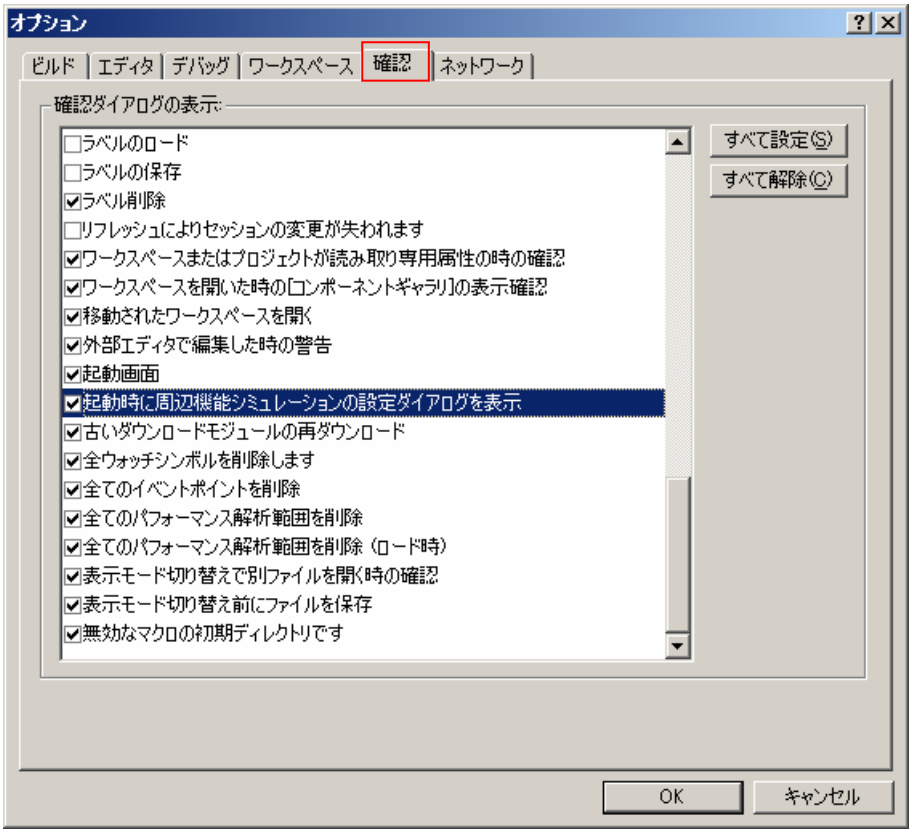


図 6-2

(2) 周辺機能シミュレーションモジュールのアドレス変更

周辺機能シミュレーションモジュールのレジスタアドレスの変更は、[周辺モジュールの構成]ダイアログボックスより行います。[周辺機能シミュレーションの設定]ダイアログボックスの[周辺機能]欄でレジスタアドレスを変更する周辺機能を選択し、[詳細]ボタンをクリックして[周辺モジュールの構成]ダイアログボックスを表示してください。[周辺モジュールの構成]ダイアログボックスで、使用するデバイスのレジスタアドレスを[開始アドレス]に設定してください。

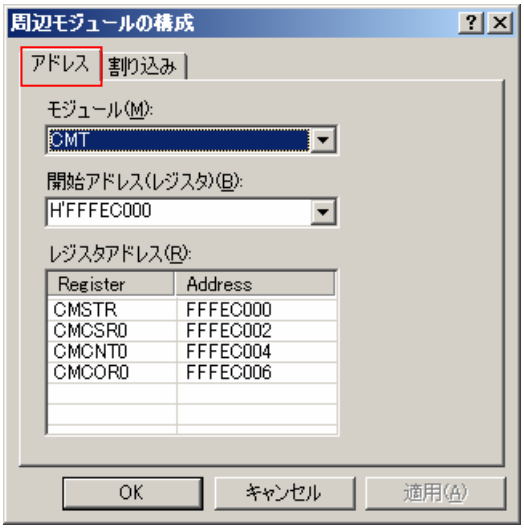
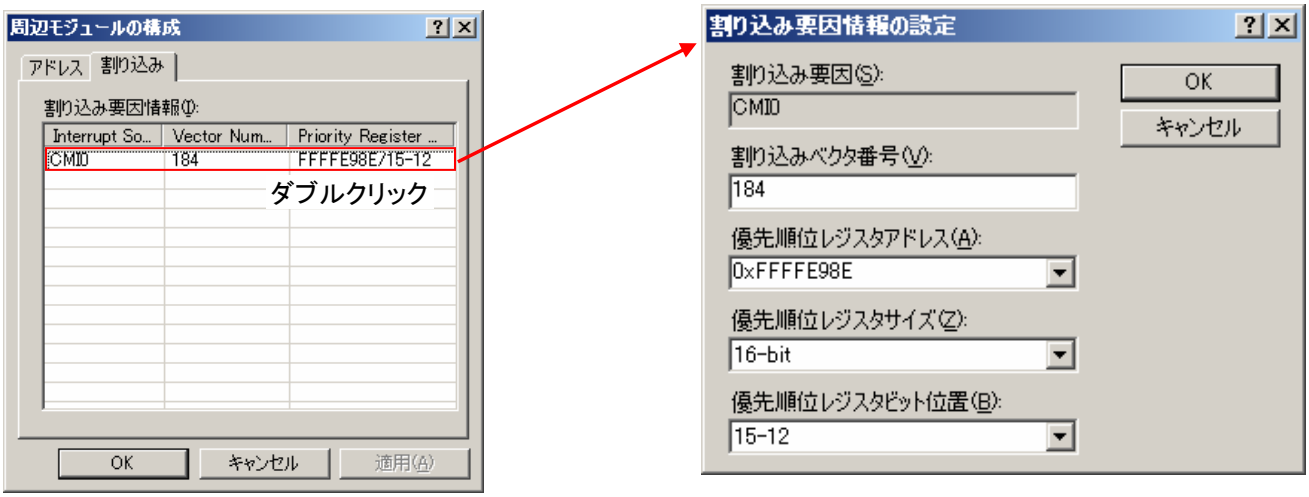


図 6-3

[周辺モジュールの構成]ダイアログボックスの[アドレス]タブでは以下の項目を表示、設定します。

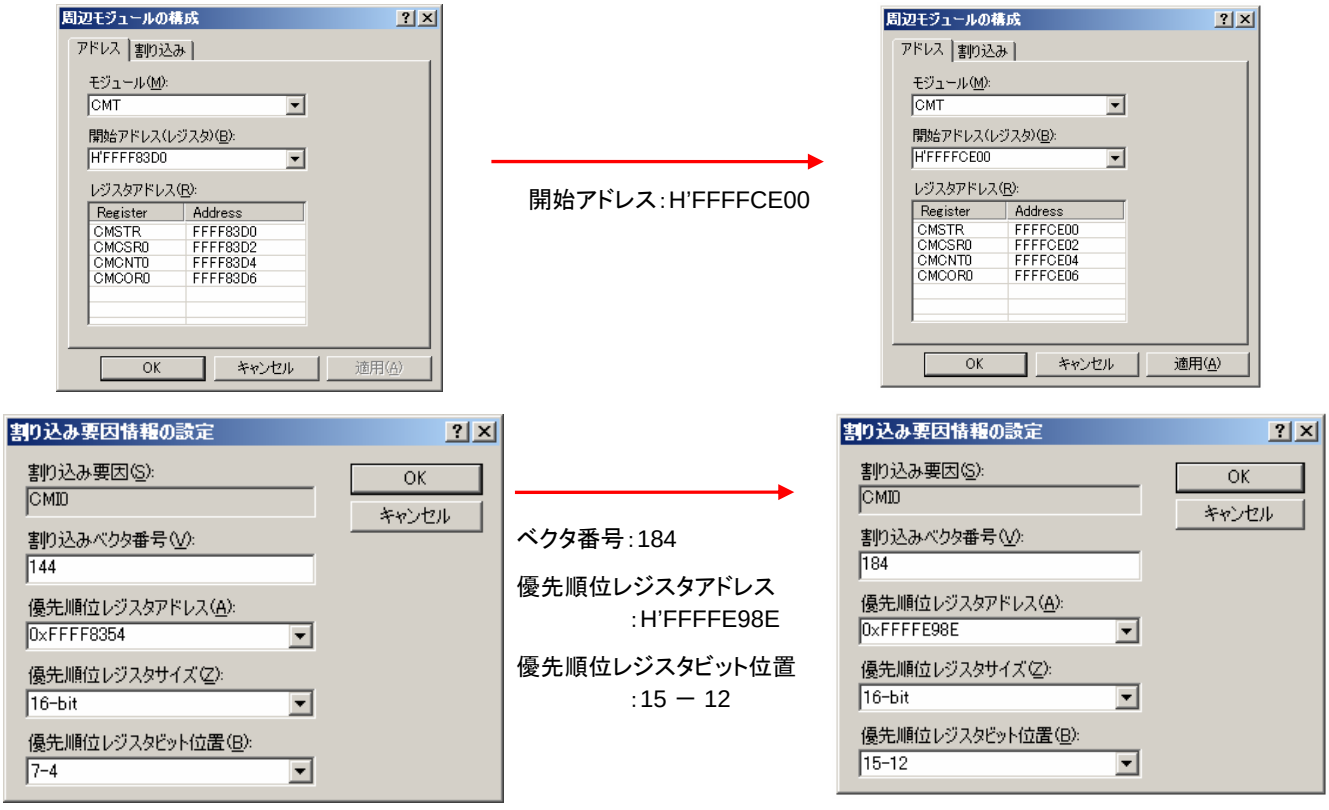
[モジュール]	選択した周辺機能シミュレーションモジュールでサポートしている周辺機能名
[開始アドレス]	[モジュール]で選択した周辺機能の開始アドレス
[レジスタアドレス]	[モジュール]で指定した周辺機能のレジスタ名、レジスタアドレスを表示します。 個々のレジスタアドレスは変更できません。

[周辺モジュールの構成]ダイアログボックスの[割り込み]タブでは、周辺機能の割り込み要因情報の参照と変更ができます。[割り込み要因情報]から変更したい割り込み要因を選択してダブルクリックして下さい。なお、SH-4 シリーズでは参照のみ可能です。



[割り込みベクタ番号]	割り込みベクタ番号
[優先順位レジスタアドレス]	優先順位レジスタのアドレス
[優先順位レジスタサイズ]	優先順位レジスタのサイズ
[優先順位レジスタビット位置]	優先順位レジスタ内の割り込み要因のビット位置

SH7085 を例に[周辺モジュールの構成]の設定例を示します。



6.2 サンプルプログラム

サンプルプログラムを用いてタイマシミュレーションの使用方法を説明します。サンプルプログラムでは、コンペアマッチタイマ(CMT)による割り込みを使用して一定間隔で変数をカウントアップしています。CMT はタイマカウントとコンペアマッチ値が一致したときに割り込みが発生するタイマです。

以下にサンプルプログラムの詳細を説明します。なお、割り込みコントローラ(INTC)および CMT の周辺機能モジュールのレジスタに対応する構造体は iodefine.h に定義しています。

- メイン関数
  (1) カウント用変数 count を初期化します。
  (2) 割り込みを受け付けるように、set_imask()で割り込みマスクを 0 に設定します。
  (3) CMT の割り込み優先順位を設定します。
  (4) CMT のコンペアマッチ値を設定します。
  (5) CMT の割り込み発生を許可します。
  (6) CMT のタイマカウントをスタートします。
  (7) 無限ループし、CMT の割り込みを受け続けます。

```
#include <machine.h>
#include "iodefine.h"

unsigned int count;

void main(void)
{
    count = 0;                /* (1) */

    set_imask(0);             /* (2) */
    INTC.IPR08.WORD = 0xF000; /* (3) */
    CMT0.CMCOR      = 0x0FFF; /* (4) */
    CMT0.CMCSR.WORD = 0x0040; /* (5) */
    CMT.CMSTR.WORD  = 0x01;   /* (6) */

    for(;;) {                 /* (7) */
        nop();
    }
}
```

- タイマ割り込み関数
  (1) カウント用変数 count をインクリメントします。
  (2) CMT のタイマカウントがコンペアマッチ値と一致すると、タイマの割り込み発生が禁止されるため、再度割り込みを許可しま
  す。なお、CMT のタイマカウントはコンペアマッチ値と一致した時点で 0 に戻っており、カウントアップが再開されます。

```
// 140 CMT CMI0
#include "iodefine.h"

extern unsigned int count;

void INT_CMT_CMI0(void){
    count++;                  /* (1) */
    CMT0.CMCSR.WORD = 0x0040; /* (2) */
}
```

シミュレータを起動するときは、[周辺機能シミュレーションの設定]ダイアログボックスで、周辺機能シミュレーションモジュールを登録してください。登録方法は、「6.1使用方法」を参照してください。

また、プログラム実行中の変数countの値は、[ウォッチ]ウィンドウで確認してください。[表示]メニューの[シンボル]の[ウォッチ]を選択し[ウォッチ]ウィンドウを表示してください。[ウォッチ]ウィンドウで、変数 count のシンボルを登録し、登録したシンボル count に対して自動更新有効化を設定します。サンプルプログラムを実行すると、[ウォッチ]ウィンドウで変数countがカウントアップされます (図 6-4)。このため、定期的にタイマ割り込みが発生している様子がわかります。

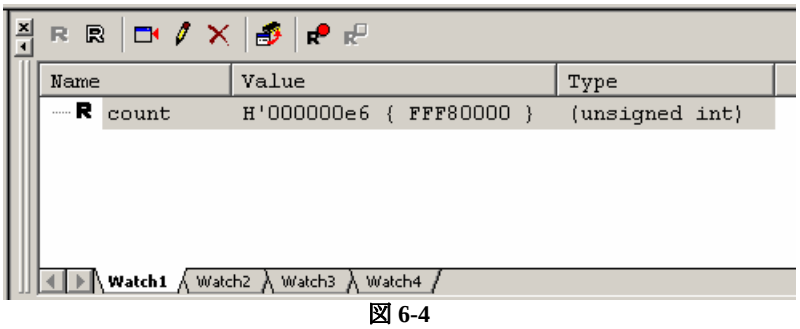


図 6-4

7. イベントポイント

エディターウィンドウから設定するブレークポイント(ソフトウェアブレークポイント)は、設定されたアドレスの命令を実行する直前にユーザプログラムの実行を停止します。それに対しイベントポイントは、ソフトウェアブレークポイントより高度なプログラム停止条件を設定できます。例えば、以下に記載したそれぞれの条件を満たしたときにプログラムを停止するブレークポイントを設定できます。

(1) あるレジスタ/データの値が指定した値になった場合
(2) 指定したデータ領域にアクセスした場合
(3) 実行サイクル数が指定したサイクル数になった場合

またイベントポイントでは、設定した条件が成立した場合にプログラムを停止する代わりに、ファイルからメモリへの書き込み、メモリからファイルへの書き込み、あるいは割り込みの発生などの動作を指定できます。

7.1 使用方法

イベントポイントは、 [表示]メニューの[コード]の[イベントポイント]を選択して表示される [イベントポイント]ウィンドウで設定できます。この[イベントポイント]ウィンドウで右クリックしポップアップメニューを表示して、そのポップアップメニューから[追加]を選択し [ブレーク種別の選択]ダイアログボックスを表示します。この[ブレーク種別の選択]ダイアログボックスで、ユーザプログラムを停止する条件や条件成立時の動作を設定します。[動作種別]の設定で “停止”を指定した場合、設定したイベントポイントの内容が[イベントポイント]ウィンドウの[Software Break]タブに表示されます。また、[動作種別]の設定で“停止”以外を指定した場合、設定したイベントポイントの内容が[イベントポイント]ウィンドウの[Software Event]タブに表示されます。イベントポイントに設定可能な項目は「SuperH™ RISC engine シミュレータ／デバッガ ユーザーズマニュアル 3.4 シミュレータ・デバッガのブレークポイントを使用する」を参照してください。

図 7-1

[イベントポイント]ウィンドウには以下の情報が表示されます。

[Type] ブレーク種別を表示します。
- [BP] ： PC ブレーク
- [BA] ： ブレークアクセス
- [BD] ： ブレークデータ
- [BR] ： ブレークレジスタ
- [BS] ： ブレークシーケンス
- [BCY] ： ブレークサイクル

[State] 該当ブレークポイントの有効/無効を示します。
- [Enable] ： 有効
- [Disable] ： 無効

[Condition] ブレークが成立する条件を表示します。表示内容はブレーク種別により異なります。ブレーク種別が BR のときはレジスタ名を、BCY のときはサイクル数を表示します。
BP 時 ： PC=プログラムカウンタ（対応するファイル名／行、シンボル名）
BA 時 ： Address=アドレス（シンボル名）
BD 時 ： Address=アドレス（シンボル名）
BR 時 ： Register=レジスタ名
BS 時 ： PC=プログラムカウンタ（対応するファイル名／行、シンボル名）
BCY 時 ： Cycle=サイクル数（16 進表示）
[Action]ブレーク条件成立時の動作を表示します。
[Stop] ： 実行停止
[File Input] ： （ファイル名）［ファイルの状態］ ：ファイルからのメモリデータ読みこみ
[File Output] ：（ファイル名）［ファイルの状態］ ：ファイルへメモリデータ書きこみ
[Interrupt] ： （割り込み種別／優先順位） ：割り込み処理
SH3-DSP のみ（割り込み種別１、割リ込み種別２／優先順位）と表示

7.2 サンプルプログラム

サンプルプログラムを用いてイベントポイントのブレークデータの使い方を説明します。サンプルプログラムではループでグローバル変数 x をインクリメントしています。このプログラムを用いて、グローバル変数 x が特定の値になるときにプログラムを停止する例を示します。
サンプルプログラムは以下の通りです。

```
volatile long x;

void main(void)
{
    int i;

    x = 0;
    for(i=0; i < 10000; i++ ) {
        x++;
    }
}
```

グローバル変数 x の値が 5000 になるときに、プログラムを停止する方法を説明します。まず、[ブレーク種別の選択]ダイアログボックスで、[ブレーク種別]に “ブレークデータ” を選択し、[詳細]ボタンをクリックしてください。[ブレークデータ条件の設定]ダイアログボックスを表示します。次に、[ブレークデータ条件の設定]ダイアログボックスで以下のように各項目を設定します。

- [アドレス]: “_x”
- [オプション]: “一致”
- [データ]: “D’5000”
- [データマスク]: チェックボックスをオフ
- [サイズ] : “Long word”

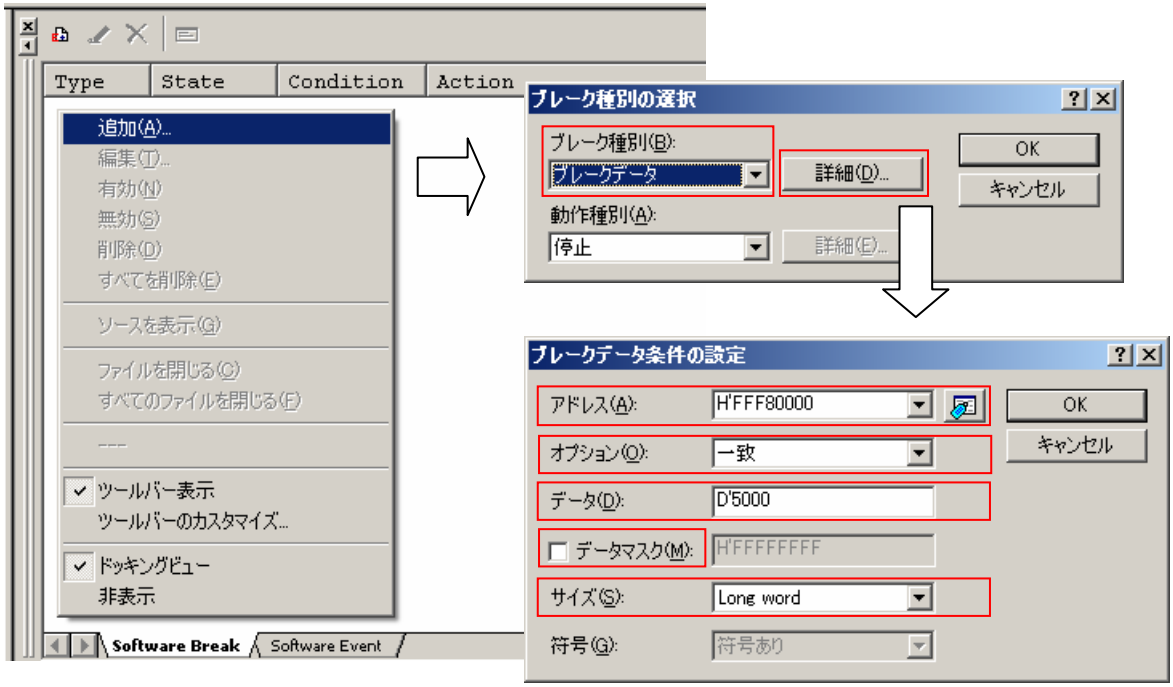


図 7-2

設定したイベントポイントが[イベントポイント]ウィンドウに表示されます。

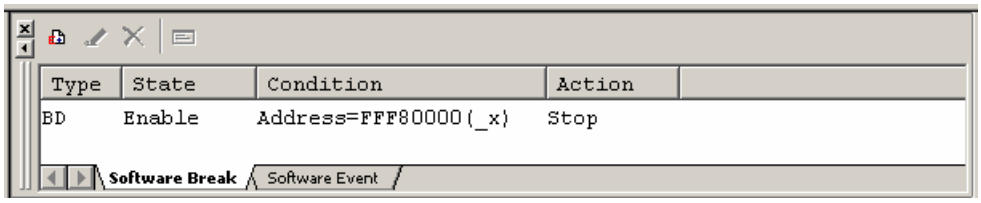


図 7-3

サンプルプログラムを実行すると、変数 x の値が 5000 になった直後にプログラムが停止します。

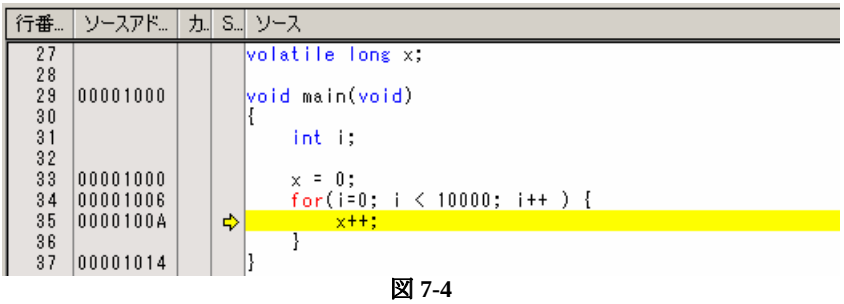


図 7-4

8. 仮想入出力パネル

仮想入出力パネルを使用することで、シミュレータのウィンドウ上に仮想的なボタンや LED を配置して、データを視覚的に確認できます。仮想入出力パネルでは以下の GUI 部品を使用できます。

- ボタン
  ボタンをクリックすることにより、指定したアドレスへのデータ入力もしくは仮想割り込みを発生させます。
- ラベル
  指定したアドレス(もしくはビット)に指定した値が書き込まれた際に、特定の文字列を表示/消去します。
- LED
  指定したアドレス(もしくはビット)に指定した値が書き込まれた際に、指定した色で表示します(LED 点灯の代用)。
- テキスト
  指定した文字列を常に表示します。

8.1 使用方法

仮想化出力パネルは、[GUI I/O]ウィンドウにパネルを配置して使用します。以下にパネルの設定方法を説明します。

[表示]メニューの[グラフィック]の[GUI I/O]を選択して[GUI I/O]ウィンドウを表示します。パネルを配置するには、[GUI I/O]ウィンドウで右クリックしポップアップメニューを表示して、作成するパネルの項目を選択してください。マウスカーソルが”+”に変化します。この状態で、作成するパネルの左上位置から右下位置までドラッグし、出力枠を作成します。出力枠が作成された直後は、出力枠を選択している状態であるため、作成した出力枠をダブルクリックして、パネルの表示内容を設定するためのダイアログボックスを表示します。

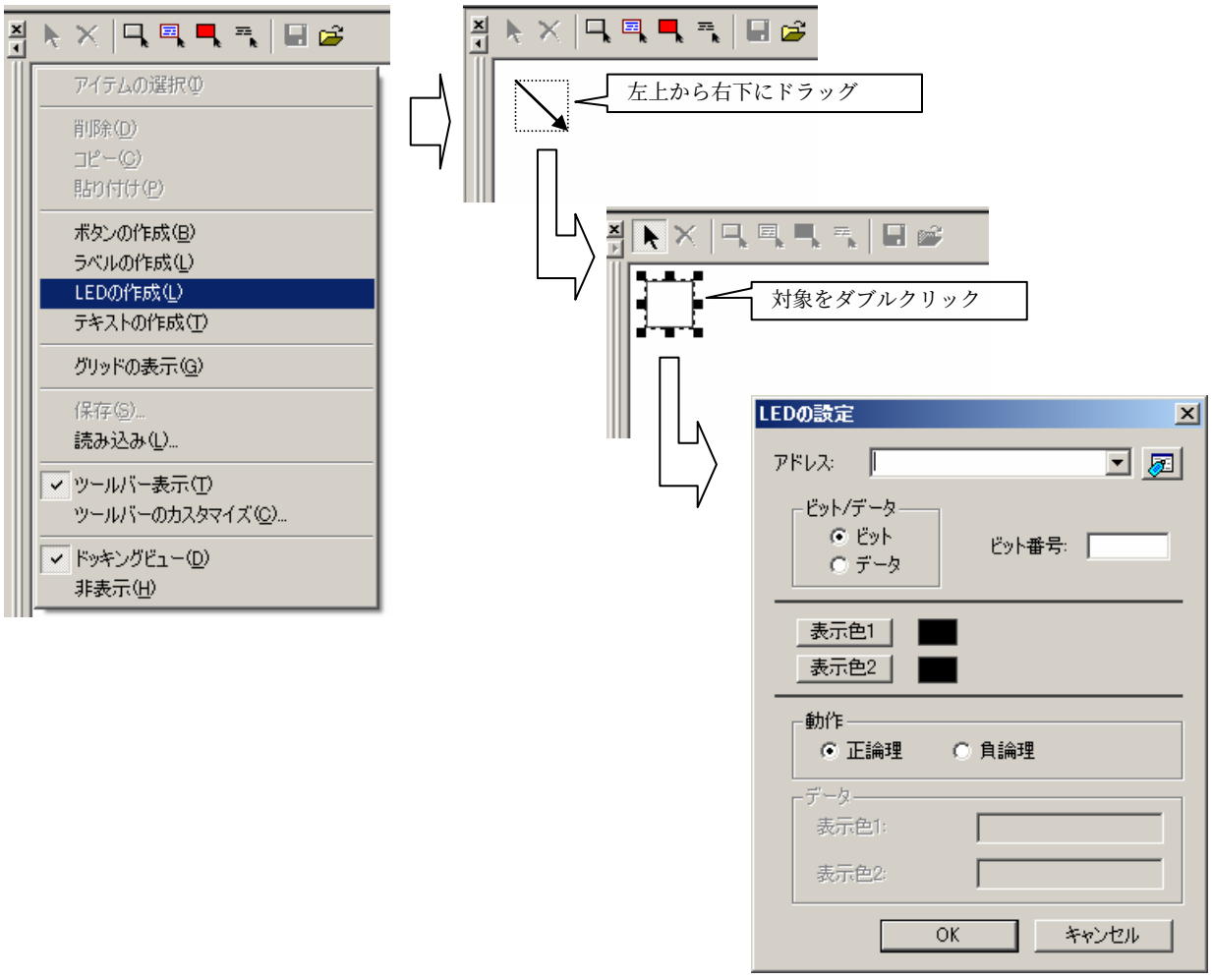


図 8-1

[GUI I/O]ウィンドウのパネルの設定はワークスペースやプロジェクトの設定と独立して保存されます。そのため、ワークスペースやプロジェクトとは別にパネルの設定をファイルに保存しておく必要があります。パネルの設定を保存するには、ポップアップメニューの[保存]を選択し[GUI I/O パネルファイルの保存]ダイアログボックスを表示してください。その[GUI I/O パネルファイルの保存]ダイアログボックスで、保存先ファイル名を指定します(デフォルトの拡張子は".pnl")。パネルの設定を保存したファイルを読み込むには、 [GUI I/O]ウィンドウでポップアップメニューより[読み込み]を選択して、保存したファイルを選択してください。

8.1.1 ボタン表示

ボタンパネルを作成したい場合は、ポップアップメニューで[ボタンの作成]を選択してください。ボタン出力枠作成後、出力枠を選択して、ダブルクリックすると[ボタンの設定]ダイアログボックスが表示されます。

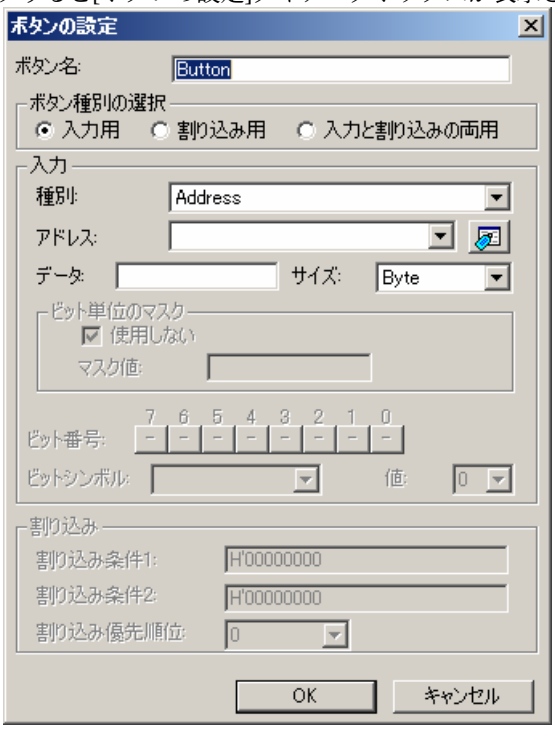


図 8-2

[ボタンの設定]ダイアログボックスで次の設定を行います。

[ボタン名] ボタンパネルに表示する名前を指定します。

[ボタン種別] [入力用]、[割り込み用]もしくは[入力と割り込みの両用]のいずれかを選択します。

[入力] [ボタン種別] で[入力用]もしくは[入力と割り込みの両用]を選択した場合に指定してください。

[種別] ボタンパネルのクリック時に、指定アドレスを特定の値に変更したい場合は "Address" を選択します。ビット位置を指定して、指定アドレスの値を変更したい場合は"Address & Bit No."を選択します。

[アドレス] ボタンパネルのクリック時に、値を変更するアドレスを指定します。

[データ] ボタンパネルのクリック時に、指定アドレスに対し置き換える値を指定します。[種別]に"Address"を選択した場合、もしくは"Address & Bit No."を選択しビット単位のマスクを使用する場合に指定してください。

[サイズ] データのサイズを指定します。

[ビット単位のマスク] [種別]で "Address & Bit No."選択時に以下の設定をしてください。ビット単位のマスクを使用しない場合は、[使用しない]チェックボックスをオフに設定し、[ビット番号]を設定します。ビット単位のマスクを使用する場合は[マスク値]を設定します。

[マスク値] 指定したマスク値で 1 になっているビットのみ、指定アドレスの値をデータに指定した値のビットに変更します。

[ビット番号] 7～0 のビットに対し、 "0"、 "1"、 "-" を指定してください。
値を 0 にしたいビットに対しては "0"を指定してください。
値を 1 にしたいビットに対しては "1"を指定してください。
値を変更したくないビットに対しては"-"を指定してください。

[ビットシンボル]および[値]は、SH マイコン用のシミュレータでは入力できません。

[割り込み] [ボタン種別] で[割り込み用]もしくは[入力と割り込みの両用]を選択した場合に指定してください。

[割り込み条件 1] CPU ごとに下記を指定します。
- SH-1、SH-2、SH2-DSP、SH2A-FPU シリーズ
  割り込みベクタ番号
- SH-3、SH-4、SH3-DSP シリーズ
  INTEVT 値（0~H'FFF）
- SH-4A シリーズ
  INTEVT 値（0~H'3FFF）

[割り込み条件 2] SH3-DSP シリーズのみ下記を指定可能です。
INTEVT2 値（0~H'FFF）

[割り込み優先順位] 割り込み優先順位を指定します。
16 指定時は、SR の I ビットによらず割り込みが発生しますが、SR の BL によってマスクします。17 指定時は、SR の I ビット、BL ビットによらず、割り込みが発生します。

8.1.2 ラベル表示

ラベルパネルを作成したい場合は、ポップアップメニューで[ラベル の作成]を選択してください。ラベル出力枠の作成後、出力枠を選択してダブルクリックすると、[ラベルの設定]ダイアログボックスが表示されます。

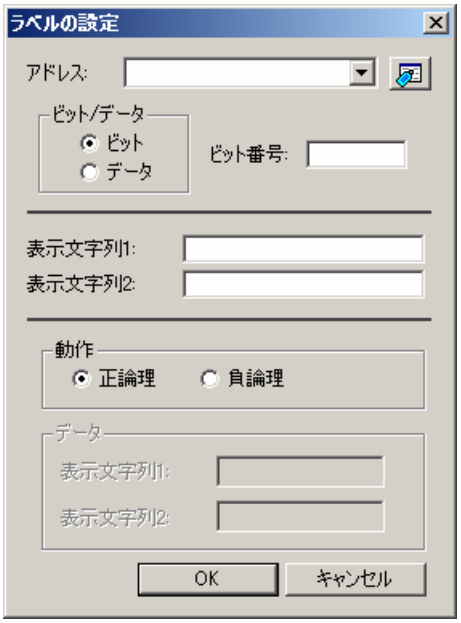


図 8-3

[ラベルの設定]ダイアログボックスで次の設定を行います。

(1) [ビット/データ]で[ビット]ラジオボタンを選択した場合

[アドレス] バイトデータの先頭アドレスを設定します。

[ビット番号] バイトデータの中の LSB から何ビット目(0～7)かを設定

[表示文字列 1] 表示する文字列を設定します。

[表示文字列 2] 表示する文字列を設定します。

[動作] ラベルパネルの表示を以下のようにします。
[正論理]ラジオボタンを選択した場合、設定したビットが 1 であれば表示文字列 1 を表示、設定したビットが 0 であれば表示文字列 2 を表示します。
[負論理]ラジオボタンを選択した場合、設定したビットが 1 であれば表示文字列 2 を表示、設定したビットが 0 であれば表示文字列 1 を表示します。

(2) [ビット/データ]で[データ]ラジオボタンを選択した場合

[アドレス] バイトデータの先頭アドレスを設定

[表示文字列 1] 表示する文字列を設定します。

[表示文字列 2] 表示する文字列を設定します。

[データ] ラベルパネルの表示を以下のようにします。
[アドレス]に指定したデータが[表示文字列 1]に設定した値であれば表示文字列 1 を表示、[表示文字列 2]に設定した値であれば表示文字列 2 を表示します。なお、 [表示文字列 1]、[表示文字列 2]に設定した値以外であれば文字列を表示しません。

8.1.3 LED 表示

LED パネルを作成したい場合は、ポップアップメニューで[LED の作成]を選択してください。LED 出力枠の作成後、出力枠を選択してダブルクリックすると、[LED の設定]ダイアログボックスが表示されます。

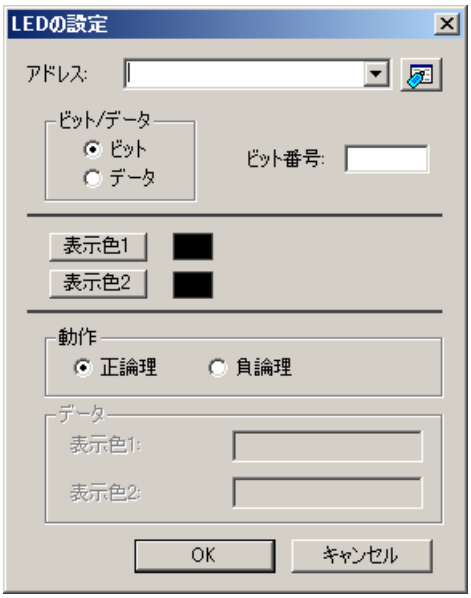


図 8-4

[LED の設定]ダイアログボックスで次の設定を行います。

(1) [ビット/データ]で[ビット]ラジオボタンを選択した場合

[アドレス]	バイトデータの先頭アドレスを設定
[ビット番号]	バイトデータの中の LSB から何ビット目(0～7)かを設定
[表示色 1]	表示する色を選択
[表示色 2]	表示する色を選択
[動作]	LED パネルの表示を以下のようにします。 [正論理]ラジオボタンを選択した場合、設定したビットが 1 であれば表示色 1 で表示、設定した ビットが 0 であれば表示色 2 で表示します。 [負論理]ラジオボタンを選択した場合、設定したビットが 1 であれば表示色 2 で表示、設定したビットが 0 であれば表示色 1 で表示します。

(2) [ビット/データ]で[データ]ラジオボタンを選択した場合

[アドレス]	バイトデータの先頭アドレスを設定
[表示色 1]	表示する色を選択
[表示色 2]	表示する色を選択
[データ]	LED パネルの表示を以下のようにします。 [アドレス]に指定したデータが[表示色 1]に設定した値であれば表示色 1 で表示、[表示色 2]に設定した値であれば表示色 2 で表示します。なお、 [表示色 1]、[表示色 2]に設定した値以外であれば白で表示します。

8.1.4 テキスト表示

テキストパネルを作成したい場合は、ポップアップメニューで[テキストの作成]を選択してください。テキスト出力枠の作成後、出力枠を選択してダブルクリックすると、[テキストの設定]ダイアログボックスが表示されます。

図 8-5

[テキストの設定]ダイアログボックスで次の設定を行います。

[テキスト]　テキストパネルに表示する文字列を設定します。

[フォント]　[フォント]ボタンをクリックして、表示する文字列のフォントおよびフォントサイズを設定します。

[色]　[テキスト]ボタンをクリックして、表示する文字列の色を設定します。[背景]ボタンをクリックして、テキストパネルの背景色を設定します。

8.2 サンプルプログラム

サンプルプログラムを用いて仮想入出力パネルの使用方法を説明します。

サンプルプログラムの仮想入出力パネルには、ある変数の値に応じて点灯/消灯する LED パネルと、その変数の値を変更するボタンパネルを配置しています。サンプルプログラムでは、ボタンパネルによって変数の値を変化させ、その結果、LED パネルが点灯/消灯する例を示します。なお、サンプルプログラムは、変数のアクセスを LED の点灯/消灯などに使われる I/O ポートのアクセスと仮定しています。実機で LED を表示する場合やボタンからの入力をする場合など、I/O ポートの初期化処理などが必要となります。

サンプルプログラムは以下の通りです。メイン関数で無限ループのみを行っています。

```
#include <machine.h>

volatile unsigned char port;

void main(void)
{
    port = 0;
    for(;;) {
        nop();
    }
}
```

[GUI I/O]ウィンドウに、仮想入出力パネルの出力枠を以下のように配置します。変数 port の 2 ビット目の値に応じて、表示が変化する LED パネルおよびラベルパネルを配置します。また、変数 port の 2 ビット目を 0 にするボタンパネル、および変数 port の 2 ビット目を 1 にするボタンパネルを配置します。

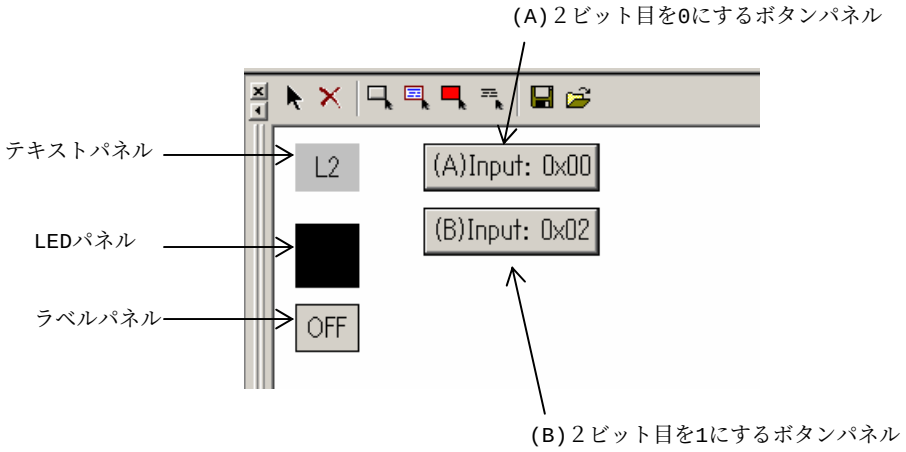


図 8-6

各パネルの設定内容を次のようにします。

テキストパネル

文字列 “L2”を表示するテキストパネルを用意します。

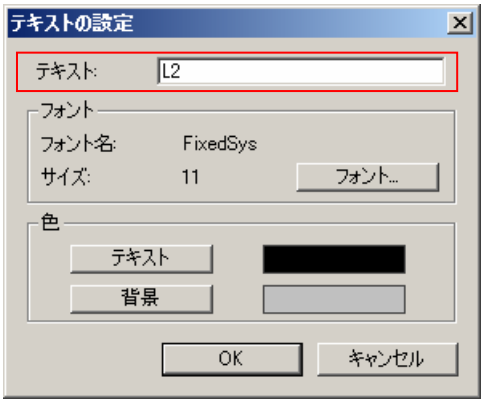


図 8-7

LED パネル

変数 port の 2 ビット目の値に応じて、色が変化する LED パネルを用意します。なお、[ビット番号]には“1” を指定してください。そして、[動作]を正論理に設定してください。

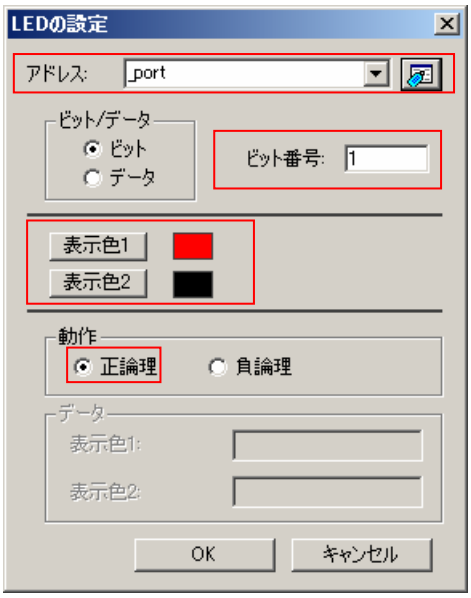


図 8-8

ラベルパネル

変数 port の 2 ビット目の値に応じて、表示文字列が変化するラベルパネルを用意します。なお、[ビット番号]には“1” を指定してください。そして、[動作]を正論理に設定してください。

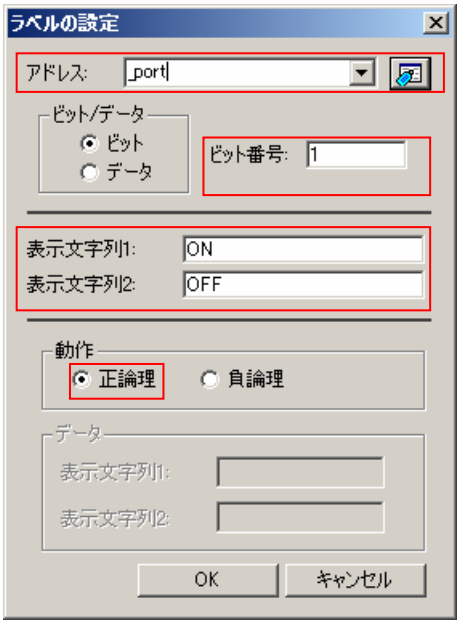
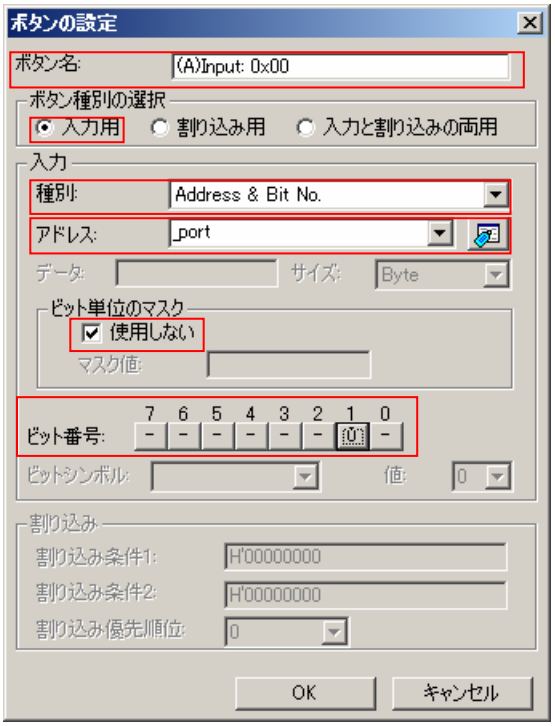


図 8-9

ボタンパネル

変数 port にデータを設定するための次の 2 つのボタンパネルを用意します。
(A) 変数 port の 2 ビット目を 0 にするボタンパネル。
(B) 変数 port の 2 ビット目を 1 にするボタンパネル。

(A)



(B)

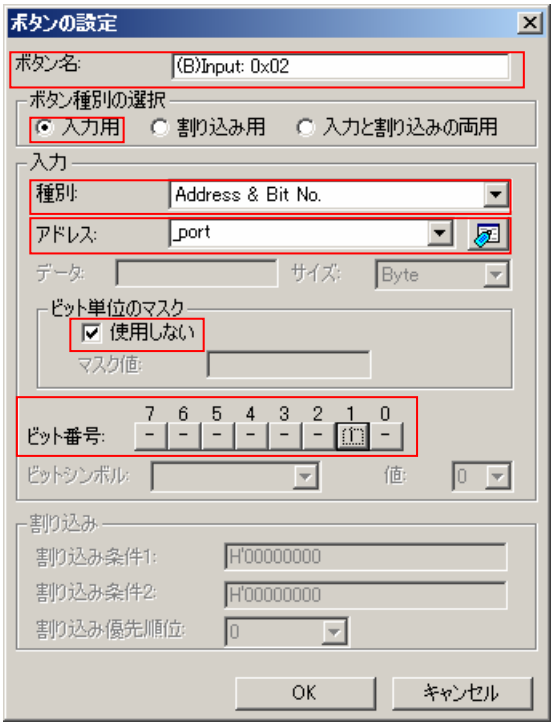


図 8-10

プログラム実行中に[GUI I/O]ウィンドウが更新されない場合は、コマンドウィンドウで ”cache off” を実行してください(図 8-11)。コマンドウィンドウは[表示]メニューの[コマンド]を選択して表示してください。

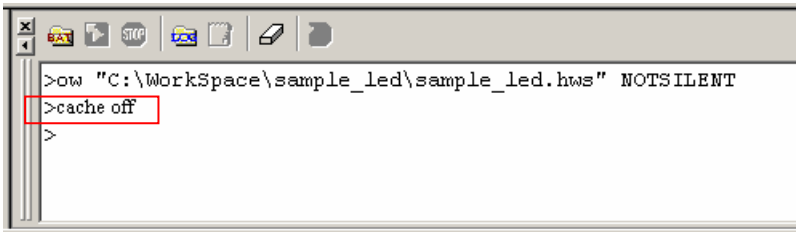


図 8-11

プログラム実行中に(B)のボタンパネルをクリックすると、変数 port の 2 ビット目が 1 になります。そのため、LED パネルの色が黒から赤に変化します。同様に、ラベルパネルの文字列が”OFF”から”ON”になります。

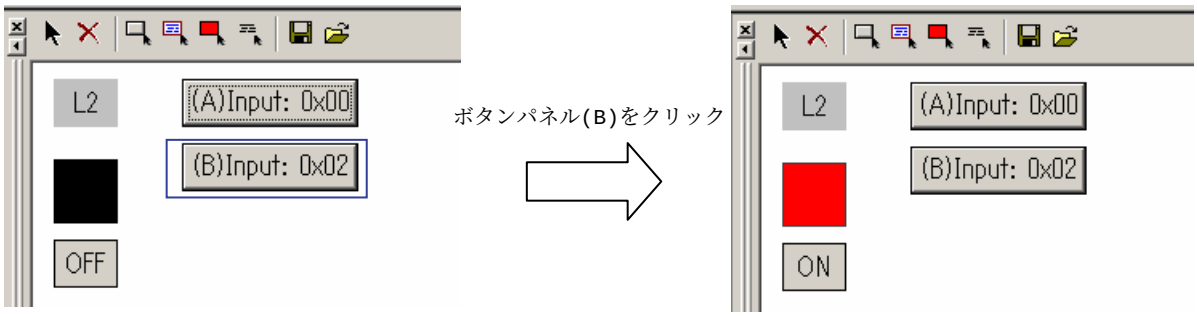


図 8-12

次に、(A)のボタンパネルをクリックすると、変数 port の 2 ビット目が 0 になります。そのため、LED パネルの色が赤から黒に変化します。同様に、ラベルパネルの文字列が”ON”から”OFF”になります。

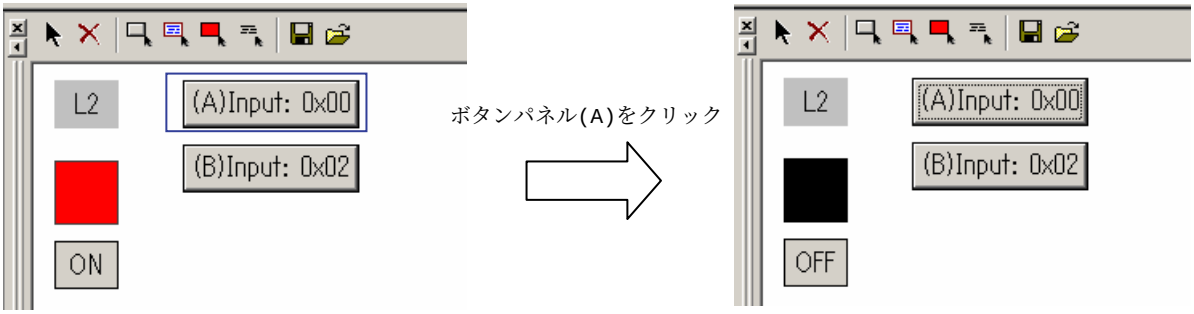


図 8-13

ホームページとサポート窓口<website and support,ws>

ルネサステクノロジホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry
csc@renesas.com

改訂記録<revision history,rh>

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2008.4.1	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。