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。

RX ファミリ用 C/C++ コンパイラパッケージ

アプリケーションノート：＜サンプルプロジェクトＲＸ移行ガイド＞Ｈ８編

本ドキュメントでは、Ｈ８で作成したサンプルプロジェクトを RX へ移行する手順についての説明を行います。

目次

本編では、シミュレータデバッガで動作確認ができる H8 サンプルプロジェクトを、RX へ移行する手順をガイドします。

1. H8 サンプルプロジェクト概要

H8 サンプルプロジェクト'H8_Sample'は大きく分けて初期化などを行う前後処理と、主要な処理を行うメイン処理に分かれています。本編では主要な処理を行うメイン処理を RX プロジェクトに移行し、動作確認をする手順を示します。メイン処理を構成するファイルを以下に示します。

表 1-1 メイン処理ファイル一覧

No	処理内容	ファイル名	参照
1	char型の符号指定	H8_sign_char.c	2.4.1
2	ビットフィールドメンバの符号指定	H8_sign_bit_field.c	2.4.2
3	ビットフィールドメンバの割付指定	H8_bit_order.c	2.4.3
4	エンディアン指定	H8_endian.c	2.4.4
5	double型変数のサイズ指定	H8_double_size.c	2.4.5
6	int型変数のサイズ指定	H8_int_size.c	2.7.1
7	main 関数	H8_Sample_main.c	-

2. H8 サンプルプロジェクト RX 移行手順

2.1 RX プロジェクトの作成

H8 サンプルプロジェクトの移行先となる RX の新規プロジェクトワークスペースを作成します。
プロジェクトジェネレータ（HEW の[ファイル → 新規ワークスペース]メニューで起動）による、サンプルプロジェクトの生成手順について説明します。

(1) 新規ワークスペースの作成

プロジェクトタイプ“Application”を選択。

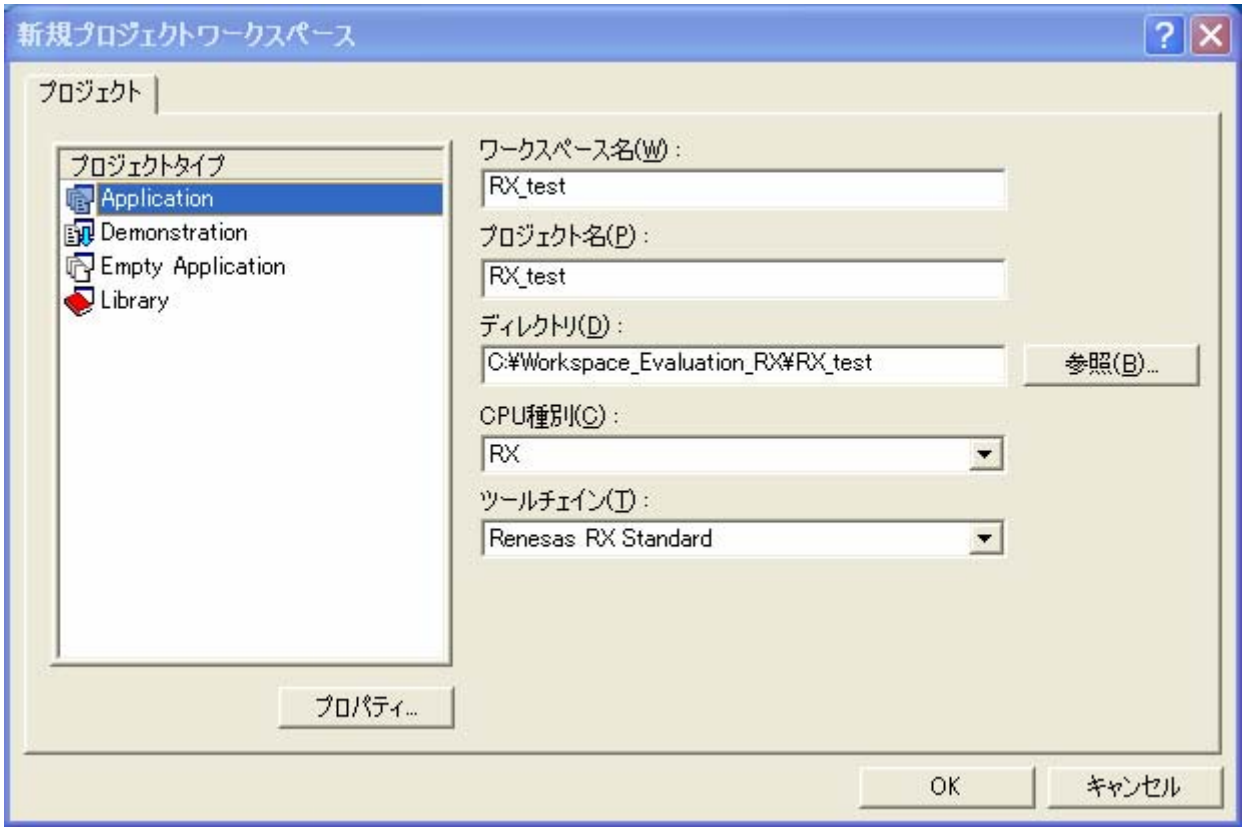


図 1-1

(2) CPU の選択

デフォルトのままで次に進む

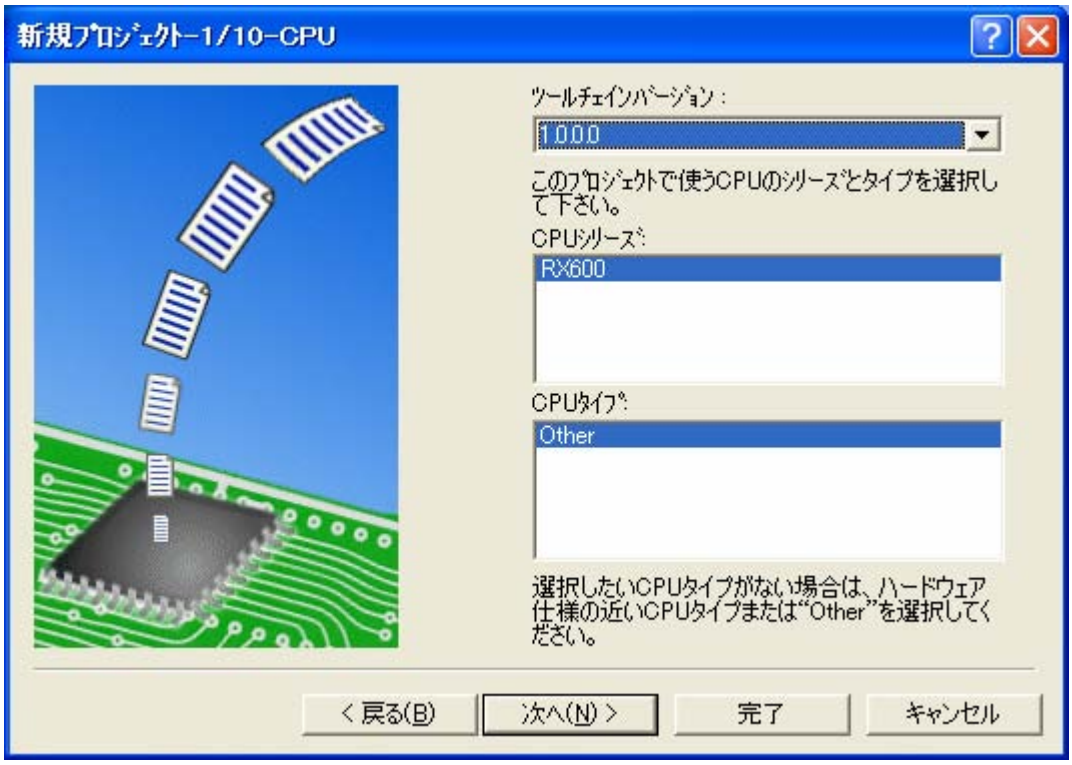


図 1-2

(3) オプション設定

デフォルトのままで次に進む

図 1-3

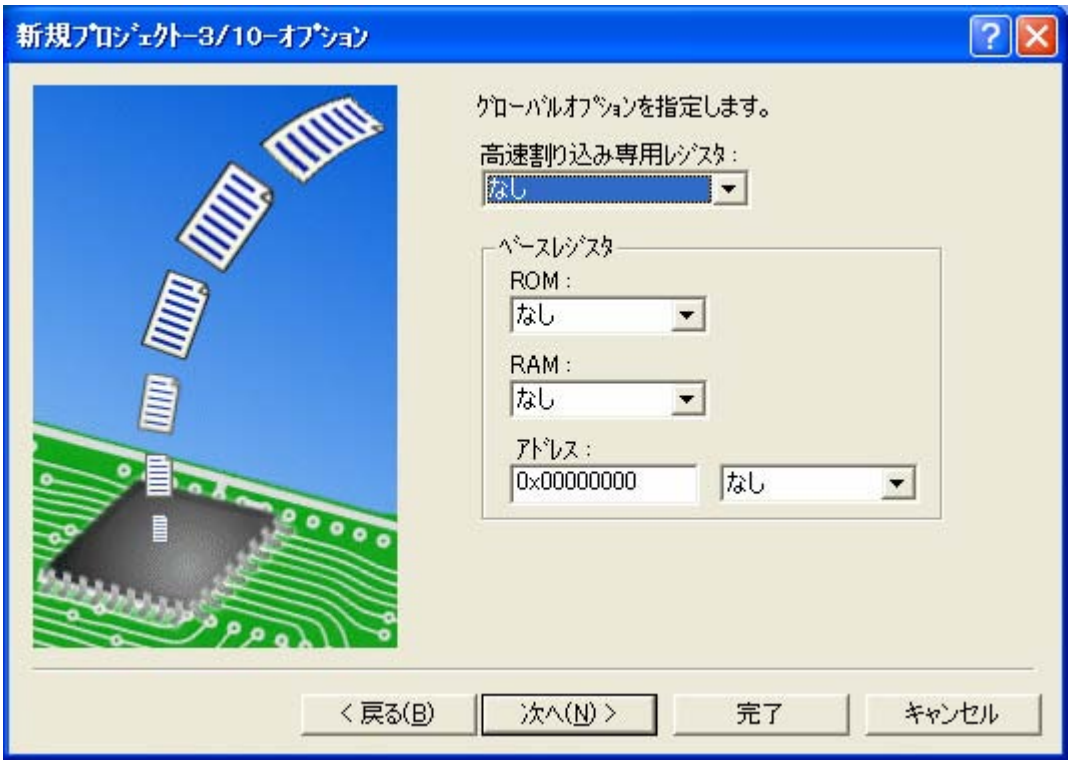


図 1-4

(4) 生成ファイルの設定

“I/O ライブラリ使用”をチェック
“I/O ストリーム数”に“20”を指定

図 1-5

(5) 標準ライブラリの設定

デフォルトのままで次に進む

図 1-6

(6) スタック領域の設定

デフォルトのままで次に進む

図 1-7

(7) ベクタの設定

デフォルトのままで次に進む

図 1-8

(8)デバッガの設定

“RX600 Simulator”をチェック

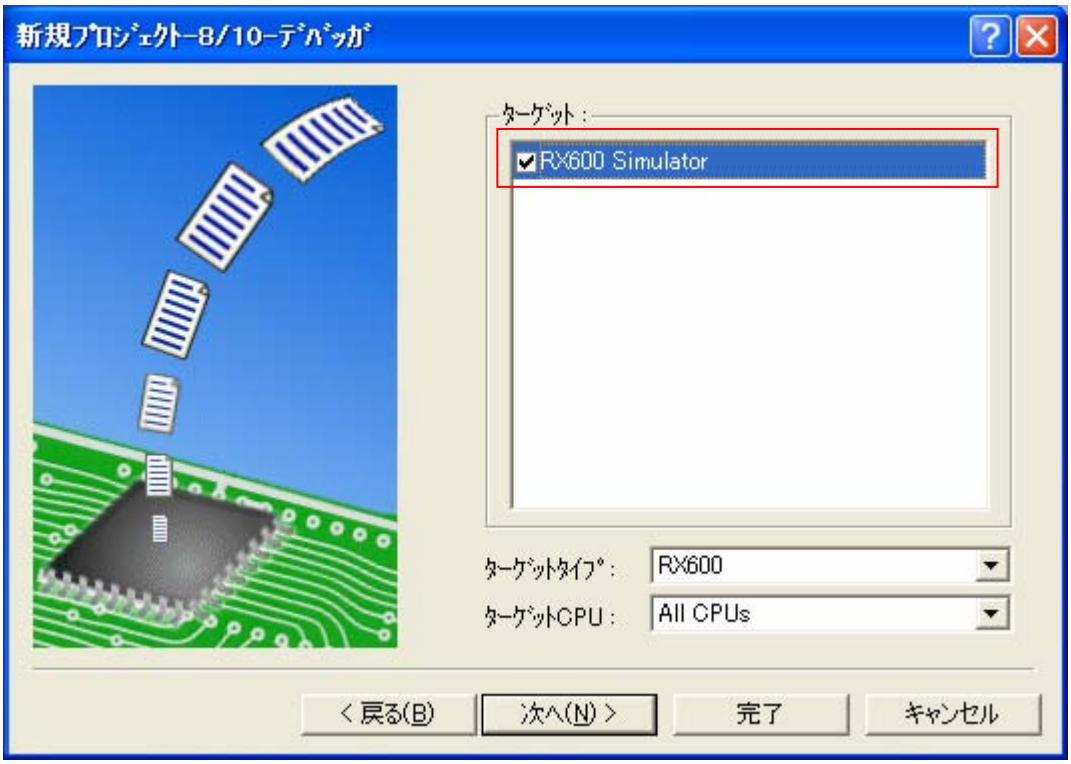


図 1-9

(9)デバッガオプションの設定

“初期セッション”をチェック

図 1-10

(10)生成ファイル名の確認

完了を選択

図 1-11

(11)シミュレータの設定

“OK”を選択

図 1-12

2.2 メイン処理ソースファイル移行

‘ 1.H8 サンプルプロジェクト概要 ’で説明したH8 サンプルプロジェクトのメイン処理を構成するファイルを、作成したRXプロジェクトにコピー・登録します。

(1)H8 サンプルプロジェクトフォルダからファイルをコピー

‘1.H8 サンプルプロジェクト概要’で説明した 7 つのファイルをコピーする。

[H8_Sample プロジェクト]

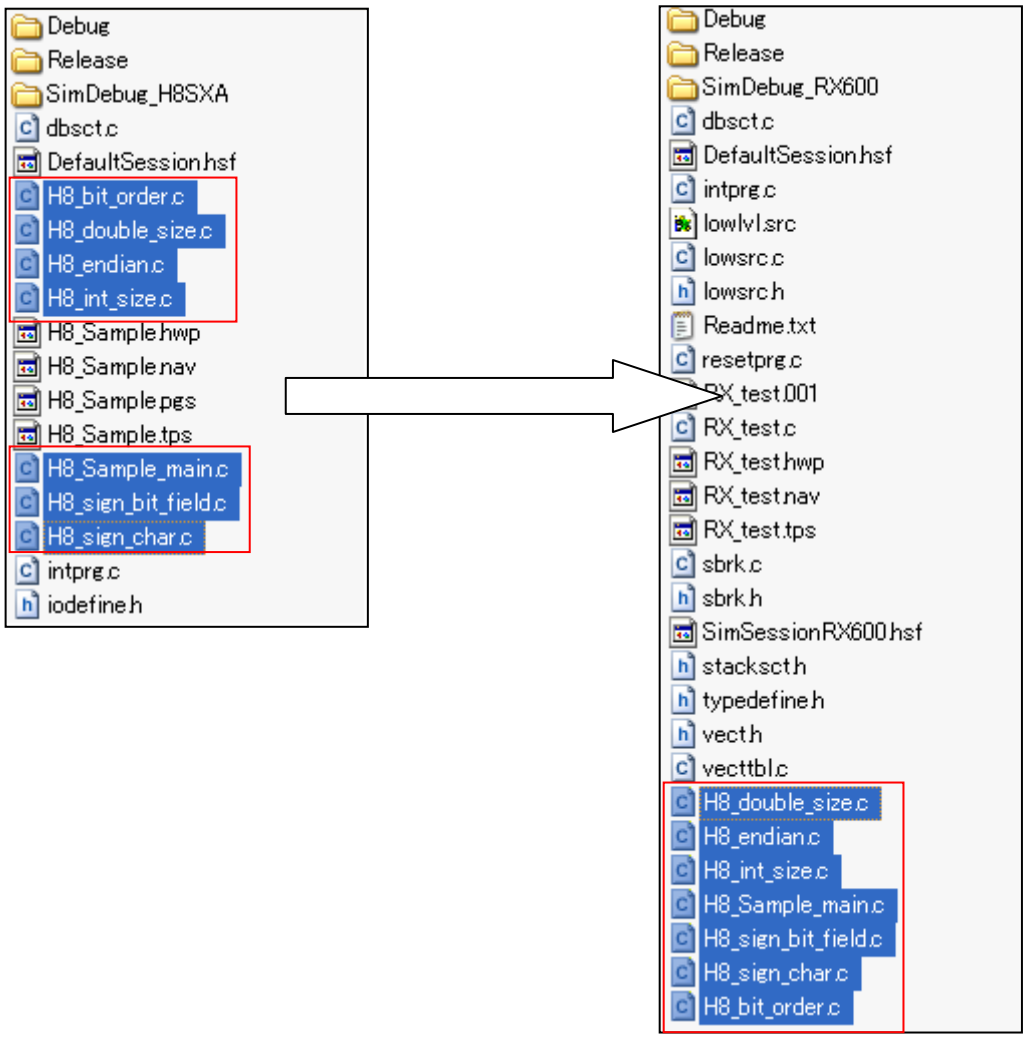


図 1-13

(2)貼り付けたファイルをプロジェクトに登録する

コピーしたファイルを、作成した RX プロジェクトに登録します。
HEW の[プロジェクト →ファイルの追加]を選択すると表示されるダイアログで次のように選択します。

図 1-14

(3)不要ファイルの登録削除

プロジェクトジェネレータが生成した main 関数のファイル’RX_test.c’は不要なので削除します。
(main 関数ファイルは H8 プロジェクトからコピーするため)
HEW の[プロジェクト →ファイルの削除]を選択すると表示されるダイアログで次のように選択します。

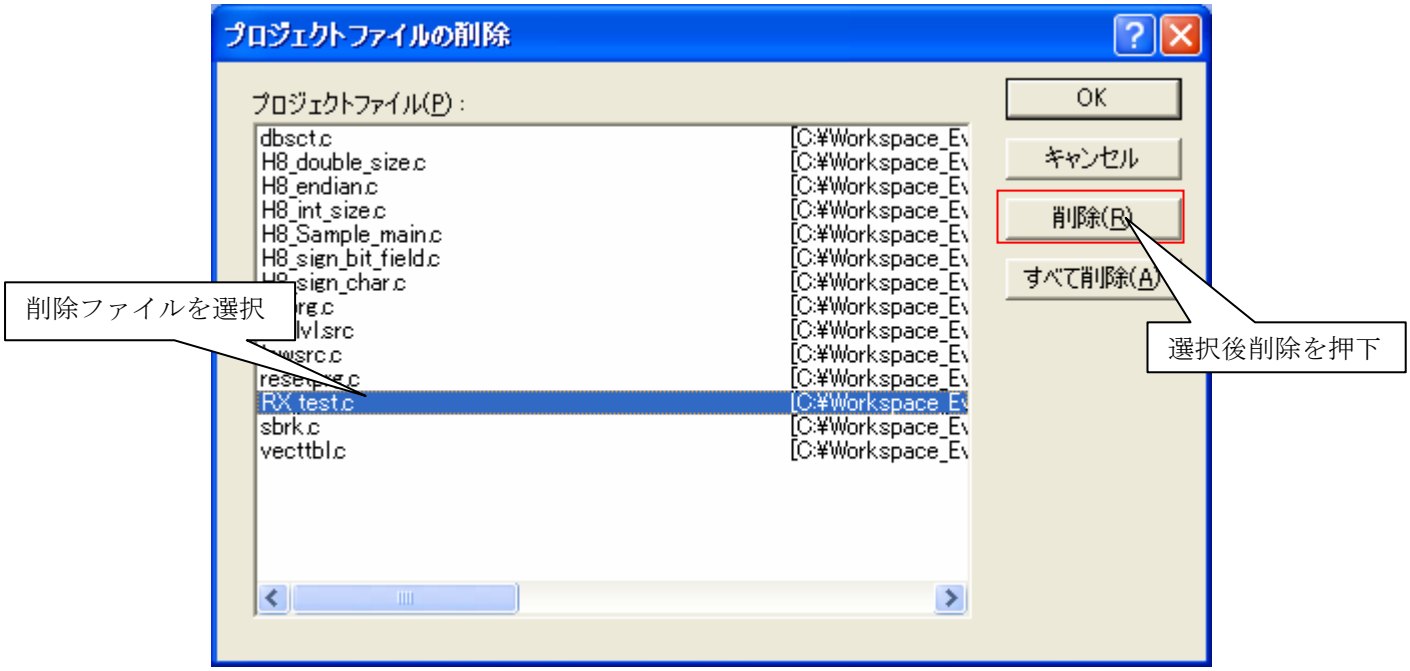


図 1-15

2.3 ビルド&H8 互換性チェック

メイン処理ファイルをコピー・登録した RX プロジェクトをビルドします。HEW ではビルド時に、H8 互換性チェック機能を有効にすることで、互換性に影響するオプション指定・ソース記述をチェックできます。本編では H8 互換性チェック機能を有効にしてビルドし、表示される互換性チェックメッセージを確認します。

(1)H8 互換性チェック機能設定

HEW の[ビルド →RX Standard Toolchain]を選択すると表示されるダイアログで次のように選択します。

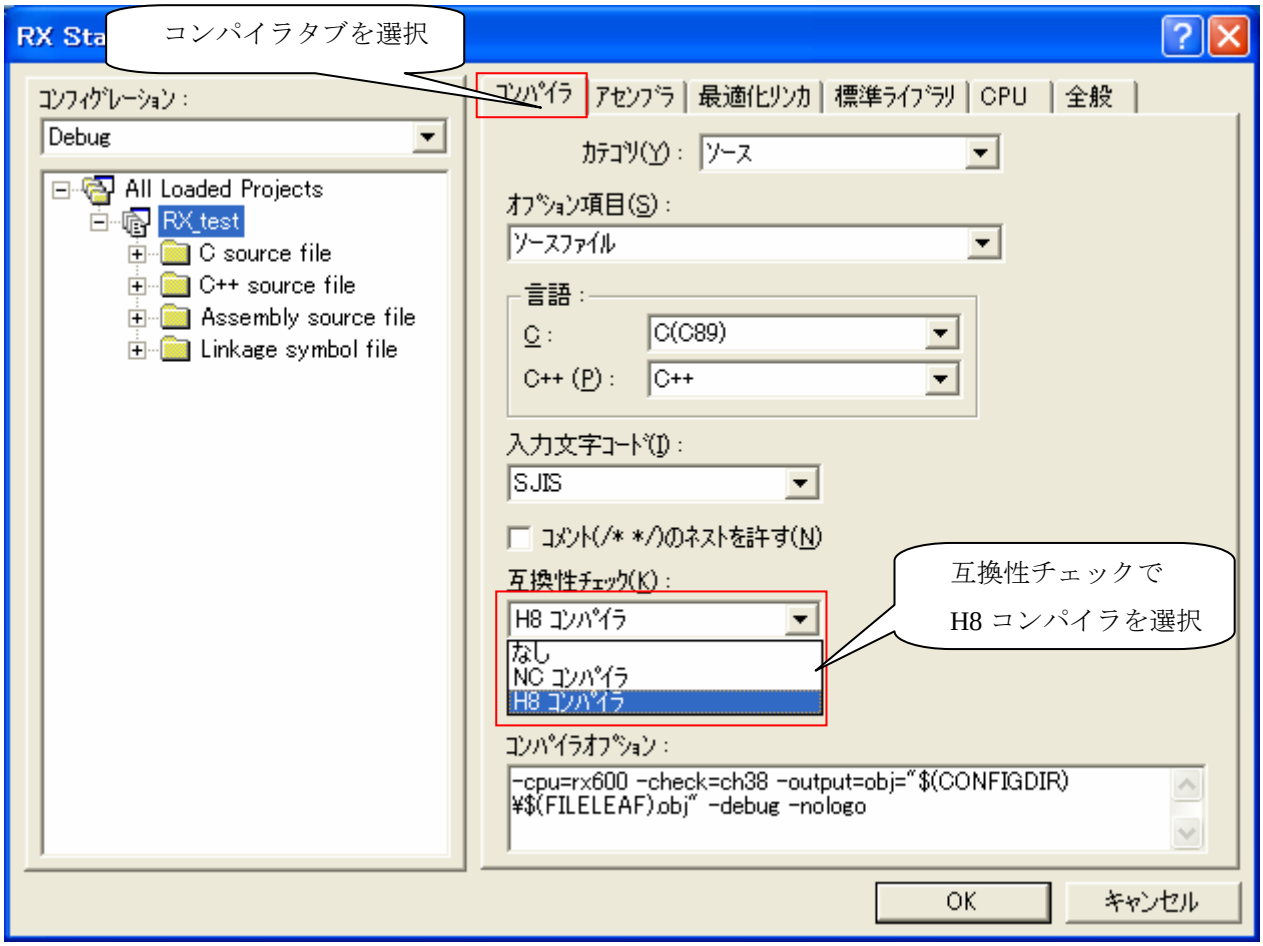


図 1-16

(2)ビルド

HEW の[ビルド →ビルド]を選択するとビルドを開始し、アウトプットウィンドウにビルド時のメッセージが表示されます。

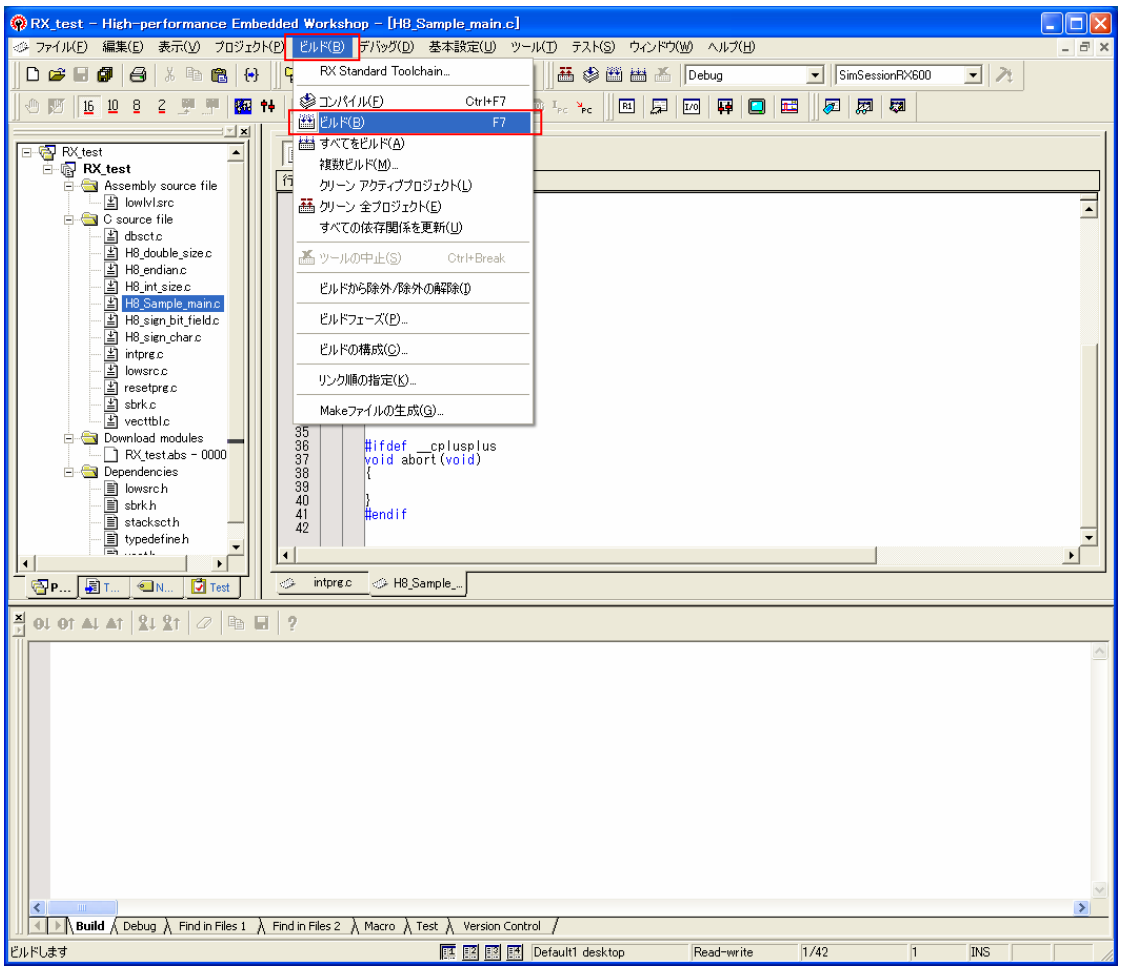


図 1-17

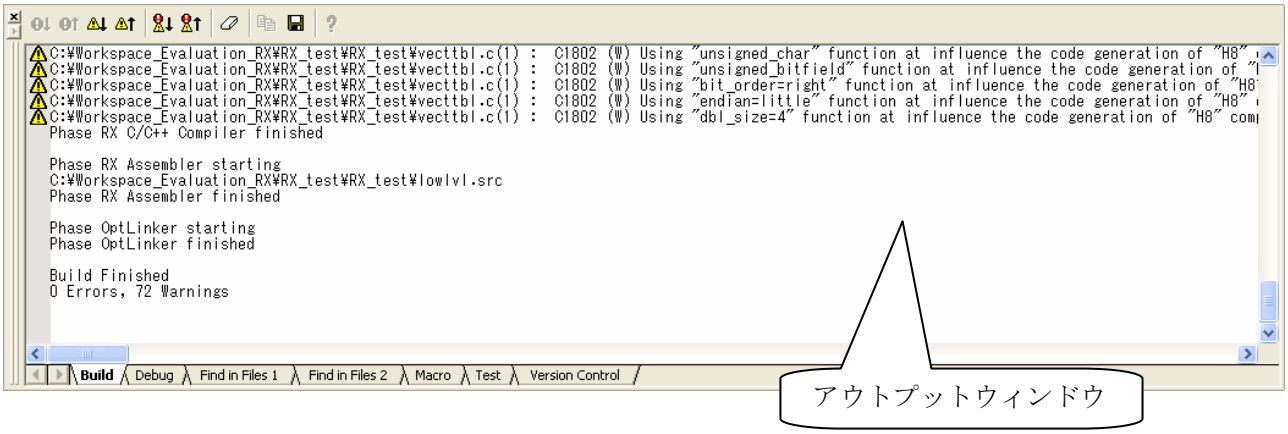


図 1-18

(3)互換性チェックメッセージ確認

アウトプットウィンドウに出力されたメッセージの中に「C1802」のウォーニングが出力されています。これが H8 との互換性に問題がある部分に対するメッセージになります。

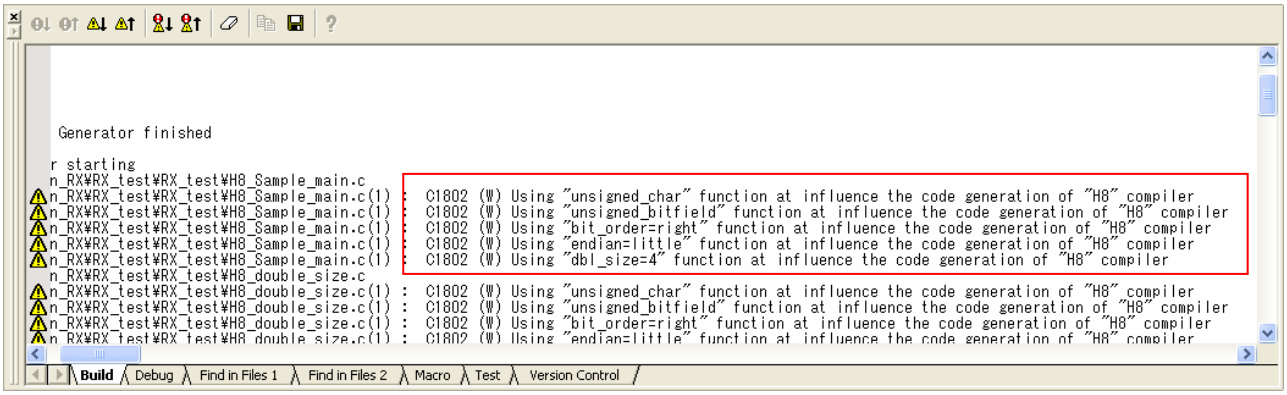


図 1-19

2.4 互換性チェック指摘対応

‘ 2.3ビルド＆H8 互換性チェック ’で指摘のあった、互換性に影響するC1802 メッセージの内容を確認、対処方法を示します。対象となるメッセージは以下のとおりです。

表 1-2 C1802 **メッセージ一覧**

No	**互換性に影響するメッセージ**	**参照**
1	Using "unsigned_char" function at influence the code generation of "H8" compiler	2.4.1
2	Using "unsigned_bitfield" function at influence the code generation of "H8" compiler	2.4.2
3	Using "bit_order=right" function at influence the code generation of "H8" compiler	2.4.3
4	Using "endian=little" function at influence the code generation of "H8" compiler	2.4.4
5	Using "dbl_size=4" function at influence the code generation of "H8" compiler	2.4.5

2.4.1 char 型の符号指定

メッセージ’Using "unsigned_char" function at influence the code generation of "H8" compiler’は、”unsigned_char”オプション指定が互換性に問題があることを指摘しています。H8 ファミリコンパイラでは符号指定のない char 型は、符号ありの signed char 型として扱います。対して RX ファミリコンパイラでは符号なしの unsigned char 型として扱います。

サンプルプログラムの”H8_sign_char.c”は符号なし char 型が signed char 型であることを前提として記述してあるため、”unsigned_char”オプションが指定されている場合、H8 とは異なった動作結果となります。

【サンプルプログラム”H8_sign_char.c” 】

ソースコード

```
struct S {
    char a;
} s = { -1 };

void sign_char(void)
{
    printf("(1) sign char : ");

    if (s.a < 0) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

char 型が符号ありの signed char 型であることを前提に作成したプログラムを RX に移行するには、”signed_char”オプションを指定します。オプション指定の詳細についてはコンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

2.4.2 ビットフィールドメンバの符号指定

メッセージ’ Using "unsigned_bitfield" function at influence the code generation of "H8" compiler’は、” unsigned_bitfield”オプション指定が互換性に問題があることを指摘しています。H8 ファミリコンパイラでは符号指定のないビットフィールドメンバは、符号ありの型として扱います。対して RX ファミリコンパイラでは符号なしの型として扱います。

サンプルプログラムの”H8_sign_bit_field.c”は、符号指定のないビットフィールドメンバが符号ありの型であることを前提として記述してあるため、” unsigned_bitfield”オプションが指定されている場合、H8 とは異なった動作結果となります。

【サンプルプログラム” H8_sign_bit_field.c”】

ソースコード

```
struct S {
    int a : 15;
} bit = { -1 };

void sign_bit_field(void)
{
    printf("(2) sign bit field : ");

    if (bit.a < 0) {
    printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

符号指定のないビットフィールドメンバが、符号ありであることを前提に作成したプログラムを RX に移行するには、”signed_bitfield”オプションを指定します。オプション指定の詳細についてはコンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

2.4.3 ビットフィールドメンバの割付指定

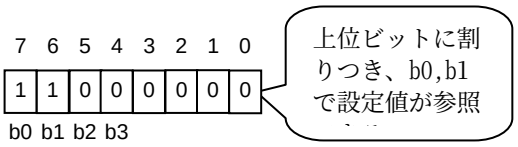
メッセージ’ Using "bit_order=right" function at influence the code generation of "H8" compiler’は、”bit_order=right”オプション指定が互換性に問題があることを指摘しています。H8 ファミリコンパイラではビットフィールドメンバを上位ビットから割り付けます。対して RX ファミリコンパイラでは下位ビットから割り付けます。サンプルプログラムの”H8_bit_order.c”は、ビットフィールドメンバを上位ビットから割り付けることを前提として記述してあるため、”bit_order=right”オプションが指定されている場合、H8 とは異なった動作結果となります。

【サンプルプログラム” H8_bit_order.c”】

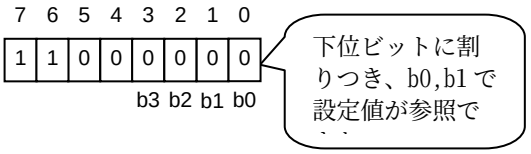
ソースコード

```
union {
    unsigned char c1;
    struct {
        unsigned char b0 : 1;
        unsigned char b1 : 1;
        unsigned char b2 : 1;
        unsigned char b3 : 1;
    } b;
} un;

(right)
void bit_order(void)
{
    printf("(3) bit field order : ");

    un.c1 = 0xc0;
    if ((un.b.b0 == 1) && (un.b.b1 == 1) &&
      (un.b.b2 == 0) && (un.b.b3 == 0)) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

H8のビット割付(left)



RXのビット割付



ビットフィールドメンバを上位から割り付けることを前提に作成したプログラムを RX に移行するには、”bit_order=left”オプションを指定します。オプション指定の詳細については、コンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

2.4.4 エンディアン指定

メッセージ’ Using "endian=little" function at influence the code generation of "H8" compiler’は、”endian=little”オプション指定が互換性に問題があることを指摘しています。H8 ファミリコンパイラではデータのバイト並びが big endian になります。対して RX ファミリコンパイラでは little endian になります。サンプルプログラムの”H8_endian.c”は、データのバイト並びが big endian であることを前提として記述してあるため、”endian=little”オプションが指定されている場合、H8 とは異なった動作結果となります。

【サンプルプログラム” H8_endian.c”】

ソースコード

```
typedef union{
    short data1;
    struct {
        unsigned char upper;
        unsigned char lower;
    } data2;
} UN;

UN u = { 0x7f6f };

void endian(void)
{
    printf("(3) endian : ");

    if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

データのバイト並びが big endian であることを前提に作成したプログラムを RX に移行するには、”endian=little”オプションを指定します。オプション指定の詳細については、コンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

2.4.5 double 型変数のサイズ指定

メッセージ’ Using "dbl_size=4" function at influence the code generation of "H8" compiler’は、”dbl_size=4”オプション指定が互換性に問題があることを指摘しています。H8 ファミリコンパイラでは double 型のサイズは 8byte です。対して RX ファミリコンパイラでは double 型のサイズは 4byte です。サンプルプログラムの”H8_double_size.c”は、double 型のサイズが 8byte であることを前提として記述してあるため、” dbl_size=4”オプションが指定されている場合、H8 とは異なった動作結果となります。

【サンプルプログラム” H8_double_size.c”】

ソースコード

```
double d1 = 1E30;
double d2 = 1E20;

void double_size(void)
{
    d1 = d1 * d1;
    d2 = d2 * d2;

    printf("(5) double type size : ");

    if (d1 > d2) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

double 型のサイズが 8byte であることを前提に作成したプログラムを RX に移行するには、”dbl_size=8”オプションを指定します。オプション指定の詳細については、コンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

2.5 再ビルド

‘2.4互換性チェック指摘対応’で互換性に問題があったオプション指定を変更後、プロジェクトを再ビルドします。ビルド方法については’2.3(3)ビルド’を参照してください。ビルドが成功すると以下のダイアログが表示されるので’はい’を選択し、ロードモジュールをダウンロードしてください。

図 1-20

2.6 シミュレータ実行

再ビルドしたロードモジュールをシミュレータで実行します。

(1)セッションのリフレッシュ設定

endian オプションで endian を little から big へ変更しているので、シミュレータについても endian を big に変更する必要があります。HEW の[ファイル →セッションのリフレッシュ]を選択すると、シミュレータ設定ダイアログが表示されるので endian を big に変更します。

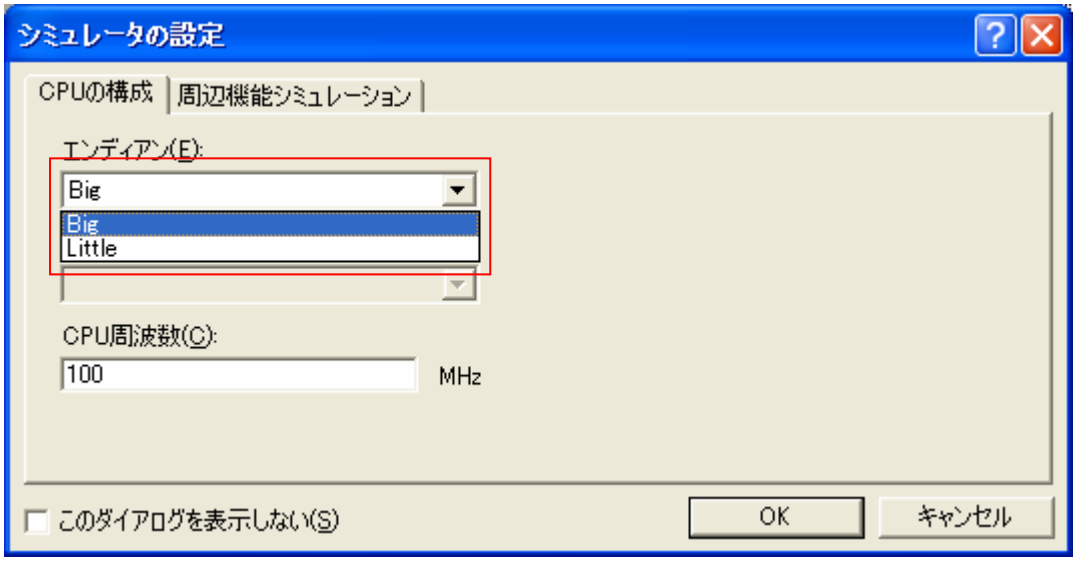


図 1-21

(2) I/0 シミュレーション設定

プログラムは実行結果を標準出力に表示します。標準出力を表示するには'I/O シミュレーション'ウィンドウを有効にする必要があります。HEW の[表示 →CPU →I/O シミュレーション]を選択すると'I/O シミュレーション'ウィンドウが表示されます。

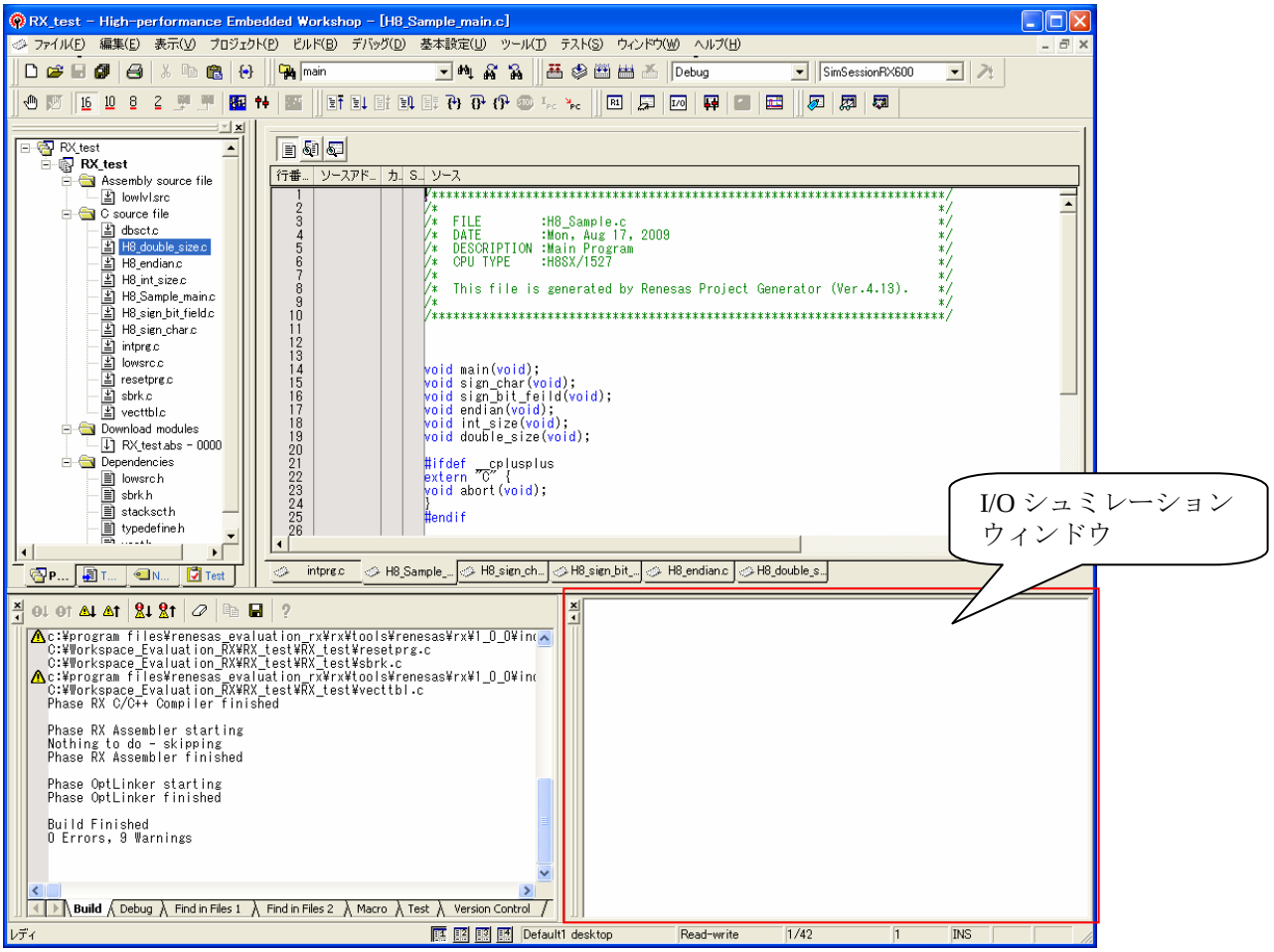
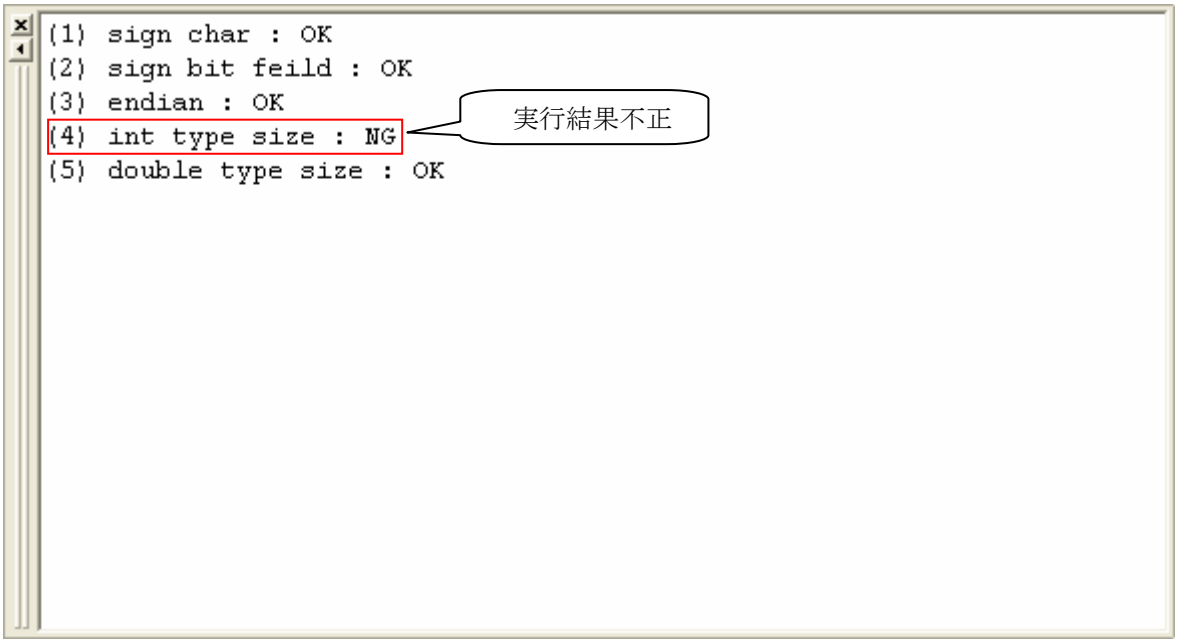


図 1-22

(3)シミュレータ実行

HEW の[デバッグ →リセット後実行]を選択するとプログラムがシミュレータにより実行され、’I/O シミュレーション’ウィンドウにプログラムの標準出力が表示されます。表示結果を見ると (4)の結果が不正であることがわかります。



2.7 実行結果不正対応

2.7.1 int 型変数のサイズ指定

H8 ファミリコンパイラでは int 型のサイズは 2byte です。対して RX ファミリコンパイラでは int 型のサイズは 4byte です。サンプルプログラムの”H8_int_size.c”は、int 型のサイズが 2byte あることを前提として記述してあるため H8 とは異なった動作結果となります。

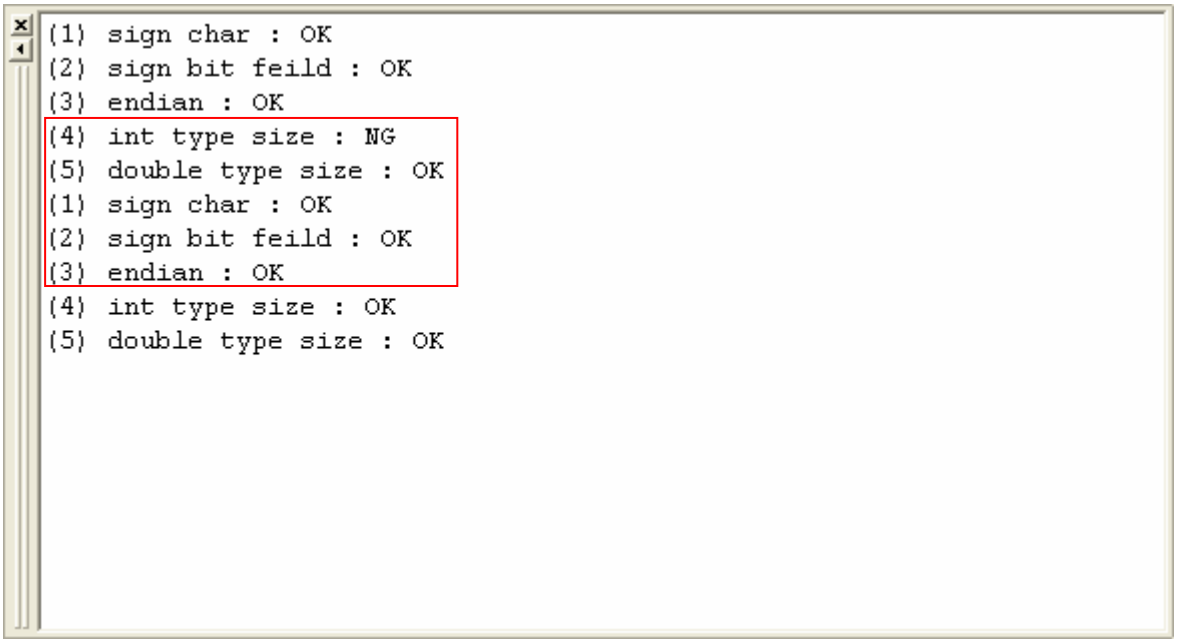
【サンプルプログラム” H8_int_size.c”】

ソースコード

```
typedef union{
    long data;
    struct {
        int dataH;
        int dataL;
    } s;
} UN;

void int_size(void)
{
    UN u;
    u.data = 0x7f6f5f4f;

    printf("(4) int type size : ");

    if (u.s.dataH == 0x7f6f && u.s.dataL == 0x5f4f) {
        printf("OK\n");
    } else {
        printf("NG\n");
    }
}
```

int 型のサイズが 2byte であることを前提に作成したプログラムを RX に移行するには、” **int_to_short**”オプションを指定します。オプション指定の詳細については、コンパイラ ユーザーズマニュアルを参照してください。また、作成した RX プロジェクトのオプション指定を変更してください。

オプションを変更後、再ビルド(’2.3(3)ビルド’を参照)、シミュレータ実行(2.6 シミュレータ実行を参照)を実施すると、以下のような実行結果となり、RX プロジェクトへの移行が完了となりました。



ホームページとサポート窓口<website and support>

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	2009.10.1	—	初版発行