CubeSuite+ V1.02.01

リリースノート

R20UT2108JJ0100
Rev. 1.00
2012.06.18

この度は，統合開発環境 CubeSuite+をご使用いただきまして，誠にありがとうございます。

この添付資料では，本製品をお使いいただく上での制限事項および注意事項等を記載しております。ご使用の前に，必ずお読みくださいますようお願い申し上げます。

第1章　対象デバイスについて

統合開発環境 CubeSuite+がサポートする対象デバイスに関しては，WEB サイトに掲載しています。
こちらをご覧ください。
CubeSuite+製品ページ：
http://japan.renesas.com/cubesuite+

第2章　ユーザーズ・マニュアルについて

本製品に対応したユーザーズ･マニュアルは，次のようになります。本文書と合わせてお読みください。

マニュアル名	資料番号
CubeSuite+ V1.02.00 起動編	R20UT0975JJ0100
CubeSuite+ V1.00.00 78K0 設計編	R20UT0546JJ0100
CubeSuite+ V1.00.00 78K0R 設計編	R20UT0547JJ0100
CubeSuite+ V1.02.00 RL78 設計編	R20UT0976JJ0100
CubeSuite+ V1.00.00 V850 設計編	R20UT0549JJ0100
CubeSuite+ V1.01.00 78K0 デバッグ編	R20UT0731JJ0100
CubeSuite+ V1.01.00 78K0R デバッグ編	R20UT0732JJ0100
CubeSuite+ V1.02.00 RL78 デバッグ編	R20UT0978JJ0100
CubeSuite+ V1.01.00 V850 デバッグ編	R20UT0734JJ0100
CubeSuite+ V1.02.00 RX デバッグ編	R20UT1143JJ0100
CubeSuite+ V1.02.00 解析編	R20UT0979JJ0100
CubeSuite+ V1.02.00 メッセージ編	R20UT0980JJ0100

第3章　アンインストール時の選択キーワード

本製品をアンインストールする場合は，2 つの方法があります。

・統合アンインストーラを使用する(CubeSuite+自体をアンインストールする)

・個別にアンインストールする(本製品のみをアンインストールする)

個別にアンインストールを行なう場合，コントロールパネルの

・「プログラムの追加と削除」(WindowsXP の場合)

・「プログラムと機能」(Windows Vista, Windows 7 の場合)

から，「CubeSuite+」を選択してください。

第4章　変更点

本章では，CubeSuite+の V1.02.00 から V1.02.01 の変更点について説明します。

4.1　デバッグ・ツールの機能追加

4.1.1　ウォッチパネルで EQU 定義のビット表示機能追加

78K0 使用時 EQU 定義したビットをウォッチパネルに登録/参照出来るようになりました。

4.1.2　78K0R/RL78 用 AZ の測定可能時間拡張（IECUBE のトレース機能使用時）

78K0R/RL78 用 IECUBE のトレース機能を用いた AZ でのデバッグで測定可能な時間を拡張しました。

4.2　注意事項解除

4.2.1　統合開発環境 High-performance Embedded Workshop のプロジェクトを CubeSuite+のプロジェクトへ変換する場合の注意事項

High-performance Embedded Workshop の SH、R8C、M16C、H8SX、H8S または H8 ファミリ用のプロジェクトを V850、RX、RL78 用プロジェクトとして CubeSuite+に読み込んだ場合、以下のエラーが発生して CubeSuite+のプロジェクトに変換できません。

エラー(E0202002)
プロジェクトの読み込みに失敗しました。

[エラーの直接原因]
プロジェクトはサポートされていないツールチェインを使用しています：
*** Standard Toolchain(E0292005)

注： *** に入る文字はマイコンファミリによって異なります。

4.2.2　RX ファミリ用 CubeSuite+パッケージの使用上の注意事項

(1) ライブラリ・ジェネレータに関する注意事項

CubeSuite+のプロパティパネルで、以下のとおりいずれかのオプションを変更した後、ビルドを実行してもライブラリ・ジェネレータが実行されません。

- 「共通オプション」タブ中の「アドレス値を設定するベースレジスタのアドレス」ボックス値を変更
- 「ライブラリ・ジェネレート・オプション」タブ中の「string(C89/C99)を有効にする」プロパティ設定で「はい」から「いいえ」に、または「いいえ」から「はい」に変更

(2)コンパイル・オプションおよびリンク・オプション（ビルドオプション）を 使用する場合の注意事項

外部変数アクセスの最適化に関するオプションを使用して、ビルドを実行すると、誤ったコードを出力する場合があります。

4.2.3 CubeSuite+の RL78 ファミリ用デバッグ機能の使用上の注意事項

RL78 ファミリ MCU 搭載システムを、CubeSuite+ 共通部分 V1.01.01 以前で、オンチップ・デバッグした後、該当製品でデバッグする場合、リアルタイム RAM モニタ機能を ON にしてデバッグ・ツールとの接続機能を実行すると、以下のエラーが出て接続できない場合があります。

ダウンロードに失敗しました。

[エラーの直接原因]
拡張モニタ領域が使用中のためモニタ・コードを書き込めません。(E1203128)。

第5章　注意事項

本章では，注意事項について説明します。

5.1　CubeSuite+全体の注意事項

5.1.1　ファイル名に関する注意事項

フォルダ名，ファイル名に関しては次の注意事項があります。

・フォルダ名，ファイル名

Windows のエクスプローラで作成することのできないフォルダ名とファイル名は，使用しないでください。

・ソース・ファイル名とロード・モジュール・ファイル名とプロジェクト・ファイル名

ファイル名は，a-z，A-Z，0-9，.(ピリオド)，_(アンダスコア)，+，- のいずれかの文字で構成されます。

ファイル名の先頭と最後に，.(ピリオド)の文字は使えません。

ファイル名の先頭に「+」（プラス）／「-」（マイナス）は使えません。

英大文字(A - Z)，英小文字(a - z)は区別されません。

ファイル名は，パスを含めて最大 259 文字です。

・上記以外のファイル名

Windows のファイル名規約に準拠します。

なお，ファイル名には次の文字は使えません。

¥ / : * ? " < > | ;

ファイル名の先頭と最後に.(ピリオド) とスペースは使えません。

英大文字(A - Z)，英小文字(a - z)は区別されません。

ファイル名は，パスを含めて最大 259 文字です。

・フォルダ名

Windows のファイル名規約に準拠します。

なお，ファイル名には次の文字は使えません（RL78，78K0，78K0R，V850 のプロジェクトを除く）。

( ) , =

5.1.2 パネル表示に関する注意事項

使用するハードウエア環境が CubeSuite+ の推奨サポート環境を下回るスペックである場合，［プロパティ］パネルのサイズを小さくすると表示内容が乱れることがあります。

その場合には，分割パネル領域から［プロパティ］パネルを外に出してください。

・ドッキング可能を ON にして，ドッキング・パネル化する

・フローティングを ON にして，フローティング・パネル化する

5.1.3 ユーザーアカウント制御(UAC)機能に関する注意事項

Windows Vista / Windows 7 において UAC 機能を無効にした場合, 管理者権限をもたないユーザでプロジェクトを作成や開いた場合で，かつ，デバイス依存情報をインストールしていない場合，デバイス依存情報のインストールが開始されますがインストールに失敗します。UAC 機能を無効にする場合は，管理者権限でログインしてプロジェクトを作成してください。

5.1.4 分割パネル・カテゴリに含まれるコマンドのアクセラレータに関する注意事項

分割パネル・カテゴリに含まれるコマンドのメニューにアクセラレータが表示されているが，キーを押しても反応しません。メニューを使用する場合には，マウスで選択してください。

5.1.5 Windows の更新プログラムに関する注意事項

マイクロソフト株式会社より公開された、Windows 用の更新プログラム（KB2393802）を適用している場合，パソコンがブルースクリーンになる障害に該当することがあります。この障害に対しては，パソコン等の各メーカより提供される修正プログラムを適用してください。

5.1.6 弊社製リアルタイム OS に関する注意事項

弊社製の RX ファミリ用のリアルタイム OS を使用する場合には、Cubesuite+のインストール・フォルダを括弧がないフォルダに変更してインストールしてください。64bit 版の Windows にインストールする場合には、¥Program Files (x86) がデフォルトのインストール・フォルダになり、フォルダ名に括弧がある場合エラーになります。

5.1.7 エディタ・パネルに関する注意事項

- ファイルのタブを使用してアクティブなファイルを切り替えたときに、「ジャンプ先の位置へ進む」、「ジャンプ前の位置へ戻る」、機能が動作しないときがあります。
- 「空白記号を表示する」オプションで、日本語の空白文字が表示されません。
- ページ設定ダイアログが使用できません。
- 印刷プレビューのツールバーにコピーボタンがありますが、使用できません。
- 印刷、および、印刷プレビューで、行番号は印刷／表示されません。また、カバレッジ行、アドレス行、イベント行、メイン行は印刷／表示されません。アウトライン機能が有効な場合、折りたたみ表示と展開表示の両方が印刷／表示されます。
- 「ジャンプ先の位置へ進む」、「ジャンプ前の位置へ戻る」メニューのショートカットキーがデフォルトで割り当てられていませんので、必要であればショートカットキーを割り当ててください。

- アウトライン機能は、条件コンパイル（#if , #else 等）に対応していません。条件コンパイル式がないものとして、アウトライン処理を実施します。

```
例）
#if AA
void main(void) {
        int test=0;
#else
void main(int argc, char *argv[]) {
        int test=1;
#endif
        test++;
}
sub()
{
}
```

  このようなソースの場合、main 関数の終端を sub()の終端と認識します。

- エディタのクライアントエリアにファイルをドラッグ&ドロップしても，ファイルがオープンされません。プロジェクト・ツリー パネルにて，オープンしたいファイルを選択し，ダブルクリックでファイルをオープンしてください。
- 関数へジャンプ機能で、別ファイルに定義されている static 関数には移動できません。
- メインプロジェクトとサブプロジェクトに、パスの違う同名のソースファイルが登録されていて、メインプロジェクトとサブプロジェクトのロードモジュールを両方ダウンロードしたとき、次のようになります。
  - 当該ファイルでは、メインプロジェクトのアドレスが表示される
  - 当該ファイルの逆アセンブルから「ソースへジャンプ」を行うと、メインプロジェクトに登録されているファイルが開く
  - どちらのプロジェクトから当該ファイルを開いても 1 つのファイルしか開けない

5.1.8 PM+から CubeSuite+プロジェクトへの移行に関する注意事項

PM+ V6.00/V6.10/V6.11 で作成した CA850 のプロジェクトに対して、ビルド・モードを新規追加した場合、そのプロジェクトを CubeSuite+で読み込むと以下のようになります。

1)Debug Build または Release Build が選択されている場合：
  新規追加したビルド・モードの情報が変換されません。
2)新規追加したビルド・モードが選択されている場合：
  エラーとなります。

回避策として、PM+ V6.20 以上でプロジェクトを開いて保存し、保存後のプロジェクトを CubeSuite+で読み込んで下さい。

5.1.9 プロジェクト流用時のデバッグ・ツールの設定に関する注意事項

プロジェクトを流用作成する時、作成するプロジェクトにてデフォルトで選択されているデバッグ・ツールに対してのみ、流用した設定を反映します。

ただし、RX ファミリについては、内部処理がエミュレータ、シミュレータで共通となっている為、デバッグ・ツールの選択状態に関わらず流用した設定を反映します。

5.1.10 オンライン・ヘルプに関する注意事項

オンライン・ヘルプにおいて、検索タブ(S)を表示した状態で閉じ、再度オンライン・ヘルプを表示し、目次(C)タブを表示した場合、コーディング編とビルド編が表示されない場合があります。

このようになった場合には、目次(C)タブを表示したままオンライン・ヘルプを閉じてから、再度オンライン・ヘルプを表示しなおしてください。

5.1.11 プロジェクト変換時の注意事項

High-performance Embedded Workshop/PM+/旧 CubeSuite を開いた時の〔プロジェクト変換設定〕ダイアログで、プロジェクトの変換先デバイスを切り替えた時、〔プロジェクトの種類〕で選択されていた値を初期値であるコンボボックスの先頭の値へ戻ります。

例えば、デバイスを選択し直すとプロジェクトの種類が先頭の（例えば〔アプリケーション〕）に切り替わります。

5.1.12 High-performance Embedded Workshop プロジェクト変換時の注意事項

High-performance Embedded Workshop のプロジェクトを CubeSuite+環境で読み込んだ場合、プロジェクト変換ができずエラーとなったり、ビルド実行時にエラーが発生する場合があります。

(1) CubeSuite+用のプロジェクトへ変換ができない

- ルネサス エレクトロニクス社製ツールチェインが使用されていないプロジェクト
- High-performance Embedded Workshop 環境の設定ファイル（tps ファイル）が存在していないプロジェクト（tps ファイルは、High-performance Embedded Workshop 環境で一度開くと自動生成されます。）
  プロジェクト変換前に一度プロジェクトを High-performance Embedded Workshop 環境で開くことで解決できます
- ルネサス エレクトロニクス社製リアルタイム OS の設定ファイル（CFG ファイル）が複数存在しているプロジェクト

(2) CubeSuite+用のプロジェクトへ変換はできるが、ビルド実行でエラーが発生

- プレースホルダ（$(TCINSTALL)）を使用しているプロジェクト
  $(TCINSTALL)は、変換後のプロジェクトにそのまま残ります。

  CubeSuite+は、$(TCINSTALL)を解釈できません。オプションのパラメータに$(TCINSTALL)を使用していた場合は、そのままオプションに渡されますので意図したビルド結果を得られない可能性があります。（ビルドでエラーが発生するなど）
  $(TCINSTALL)をプロジェクト変換後に、お客様自身で変更してください。

- プレースホルダ（$(WORKSPDIR)）を使用しているプロジェクト
  プロジェクトファイル（拡張子 hwp）を指定して変換した場合、「%ProjectDir%¥..」

（プロジェクトフォルダの 1 つ上のフォルダ）に固定で変換します。

プロジェクトフォルダの 1 つ上のフォルダにワークスペースがない場合は、正しいフォルダを示さなくなりますので、ビルドでエラーが発生することがあります。
その場合、プロジェクト変換後に「%ProjectDir%¥..」を、お客様自身で変更してください。

・ カスタムビルドフェーズを使用しているプロジェクト
カスタムビルドフェーズは、削除されます。

カスタムビルドフェーズは、ビルド時に実行されなくなります。
よって、カスタムビルドフェーズで生成されたファイル出力を使用している場合はビルドエラーとなる可能性があります。
プロジェクト変換後に、カスタムビルドフェーズのコマンドを、各フェーズの前後実行コマンドに必要に応じて登録してください。

・ カスタムプレースホルダを使用しているプロジェクト
カスタムプレースホルダは変換しません。

CubeSuite+は、カスタムプレースホルダを解釈できません。オプションのパラメータにカスタムプレースホルダを使用していた場合は、そのままオプションに渡されますので意図したビルド結果を得られない可能性があります。（ビルドでエラーが発生するなど）
プロジェクト変換後に、カスタムプレースホルダを、お客様自身で変更してください。

5.1.13 パック機能に関する注意事項

CubeSuite+のラピッドスタートが有効な状態で、CubeSuite+ V1.02.00、V1.02.01 でパックされた環境を起動するとラピッドスタートしている CubeSuite+が起動してしまう。

回避策

パックされた環境を起動する時、通知領域（タスクトレイ）内にラピッドスタートしている CubeSuite+を終了させてからパックされた環境を起動してください。

5.2 設計ツールの注意事項

5.2.1 パッケージの変更に関する注意事項

端子配置のプロパティでパッケージ名を変更した場合，端子配置図および端子配置表の入力データはクリアされます。

5.2.2 プロジェクト保存に関する注意事項

サブプロジェクトが存在するプロジェクトにて，端子配置図または端子配置表パネルが開いた状態でプロ

ジェクトの保存を行った場合に，プロジェクト・ツリー上の最後のサブプロジェクトの端子配置図，端子配置表が必ず表示されます。

5.2.3 プロジェクトの移行に関する注意事項

【対　象】78K0 / 78K0R / RL78

リンク・オプションの「オンチップ・デバッグを設定する」および「ユーザ・オプション・バイトを設定する」の設定が、プロジェクト保存時とプロジェクト読み込み時で異なる場合があります。

[発生条件]

1) プロジェクト・ファイルを読み込むログイン者の.mtud ファイルがないとき

例 1) ログイン者 A がプロジェクト・ファイルを保存後、ログイン者 A とは別のログイン者 B がプロジェクト・ファイルを読み込む場合

例 2) ログイン者 A がプロジェクト・ファイルを保存し、故意に.mtud ファイルを削除したのちにプロジェクト・ファイルを読み込む場合

2) プロジェクト・ファイルを読み込むログイン者の.mtud ファイルがあるときで、プロジェクト・ファイルを読み込み後にコード生成部のパネルが最前面にある場合

[処置]

プロジェクト・ファイルを読み込み後、またはビルド前に該当オプションが正しい設定値になっているか確認してください。

5.3 ビルド・ツールの注意事項

5.3.1 スタートアップ・ノードに関する注意事項

CX 用プロジェクトである場合、スタートアップ・ノードにオブジェクト・モジュール・ファイル(.obj)を登録すると、以下の警告が出力します。この警告は無視してください。

W0560111:同じファイルが入力ファイルとして複数回指定されています。

マルチコア用プロジェクトである場合、スタートアップ・ノードにオブジェクト・モジュール・ファイル(.obj)を登録すると、以下のエラーが出力します。スタートアップ・ノードにはアセンブル・ファイル(.asm)を登録してください。

F0560208:シンボル"xxx"は多重に定義されています。

5.4 デバッグ･ツールの注意事項

文中において以下の略称を使用しています。

OCD(シリアル) ：MINICUBE2，E1 エミュレータ(シリアル)，E20 エミュレータ(シリアル)

OCD(JTAG) ：MINICUBE，E1 エミュレータ(JTAG)，E20 エミュレータ(JTAG)

5.4.1 サブプロジェクトの追加に関する注意事項

【対 象】 全デバッグ・ツール，全デバイス共通

メインプロジェクトと異なるデバイスを扱うサブプロジェクトを追加する場合，デバッグ・ツールを切断してから行ってください。

5.4.2 ブートスワップ実行時の注意事項

【対 象】シミュレータ/OCD(JTAG)/OCD(シリアル)，V850 / 78K0 / 78K0R / RL78

ブートスワップ領域にソフトウエア・ブレークを設定した場合，フラッシュ ROM にブレーク用の命令が書き込まれるため，ブートスワップ後もブレーク用の命令が残ってしまいます。

・OCD(JTAG)/OCD(シリアル)の場合：ブレークを設定する場合は，ハードウエア・ブレークを使用してください。

・シミュレータの場合：ブートスワップ領域にブレークを設定しないでください。

5.4.3 内蔵 RAM でのプログラム実行に関する注意事項

【対 象】シミュレータ，V850

V850E/MA3 などの V850E マイコンでは，内蔵 RAM でのプログラム実行(0x0fff0000 番地～0x0fffefff 番地でのプログラム実行)に関して以下の注意事項があります。

・内蔵 RAM 内でブレークした際，逆アセンブル･パネル上では(0x03ff0000 番地～0x03ffefff 番地)が表示されます。

・内蔵 RAM 内の関数へステップ･インした場合，ステップ･オーバーの動作となります。

5.4.4 ストップ･モードの注意事項

【対 象】全デバッグ・ツール，V850 / 78K0 / 78K0R / RL78

STOP モードや HALT モードなどのスタンバイ･モード中に強制ブレークを行った場合や，ステップ実行でスタンバイ・モードに移行する命令を実行した場合，シミュレータとエミュレータ(IECUBE，OCD(JTAG)，OCD(シリアル))では以下のような動作の差があります。

・エミュレータ：強制ブレークによりスタンバイ･モードは解除されます。また，ステップ実行ではスタンバイ・モードに移行しません。
・シミュレータ：強制ブレークによりスタンバイ･モードは解除されません。また，ステップ実行ではスタンバイ・モードに移行します。

どちらの場合とも，強制ブレーク時に PC(プログラム･カウンタ)行は，HALT などのスタンバイ･モード以降命令の次命令でブレークします。このためシミュレータの場合，スタンバイ･モードが解除されているようにも見えます。スタンバイ･モードが解除されているかどうかの確認はステータス･バー行なってください。スタンバイ･モード中の場合，ステータス･バーに"Halt"や"Standby"の表示が出ます。

5.4.5 低消費電力モードに関する注意事項

【対 象】全デバッグ・ツール，RX

スリープモード、ストップモードおよびスタンバイモードなどの低消費電力モード中に強制ブレークを行った場合や，ステップ実行で低消費電力モードに移行する命令を実行した場合，シミュレータとエミュレータでは以下のような動作の差があります。

・エミュレータ：強制ブレークにより低消費電力モードは解除されます。また，ステップ実行では低消費電力モードに移行します。
・シミュレータ：レジスタなどによる低消費電力モードへの移行はサポートしていません。WAIT命令実行時にはブレークし、PCは次の命令のアドレスとなります。また，ステップ実行では低消費電力モードに移行せず、PCは次の命令のアドレスとなります。

5.4.6 乗除算器に関する注意事項

【対 象】シミュレータ，78K0

78K0 の命令シミュレーションを行なう場合，乗除算器に対応していません。このため，プログラム内で乗算や除算を行なう場合は，ビルド･ツールのプロパティ･パネルを開き，[コンパイル･オプション]タブで[[乗除算器を使用する]ドロップダウン･リストの「いいえ」を選択してください。

5.4.7 メモリ･バンクに関する注意事項

【対 象】シミュレータ，78K0

78K0 の命令シミュレーションを行なう場合，メモリ･バンク機能に対応していません。

5.4.8 CPU 動作クロックに関する注意事項

【対 象】シミュレータ，78K0R，RL78

・78K0R の命令シミュレーションを行う場合，高速内蔵発振器の周波数は 8MHz 固定です。

・RL78 の命令シミュレーションを行う場合，CPU 動作クロックは RL78/G13 の仕様で動作します。

5.4.9 乗除算器，積和演算器に関する注意事項

【対 象】シミュレータ，78K0R / RL78

78K0R，RL78 の命令シミュレーションを行う場合，乗除算器や積和演算器の使用に関して以下の注意事項があります。

(1) 乗除算器や積和演算器を除算モードで使用した場合，除算処理は 1 クロックで終了します。

(2) 乗除算器や積和演算器を除算モードで使用した場合，除算演算完了割り込みは発生しません。ただし，除算完了を示す SFR は変化します。（乗除算コントロール・レジスタ"MDUC"の DIVST ビットが 0 になります。）

5.4.10 任意区間のトレースに関する注意事項

【対 象】シミュレータ，全デバイス共通

トレース開始イベントからトレース終了イベントまでをトレースする場合，シミュレータではトレース終了イベントがトレース結果として表示されません。このため，シミュレータを使用する場合はトレース終了イベントをトレース･データとして表示させる範囲の 1 行下に設定してください。

5.4.11 任意区間の実行時間測定に関する注意事項

【対 象】シミュレータ，V850 / 78K0 / 78K0R / RL78

タイマ開始イベントからタイマ終了イベントまでを実行時間測定する場合，シミュレータではタイマ終了イベントが時間測定結果に含まれません。このため，シミュレータを使用する場合はタイマ終了イベントを時間測定する区間の 1 行下に設定してください。

5.4.12 CPU 動作クロックに関する注意事項

【対 象】シミュレータ，V850

V850 の命令シミュレーション･モードでは，クロック･ジェネレータのシミュレーションは行われません。このため，CPU の動作クロックは常にプロパティ･パネルで設定したメインクロック周波数となります(クロック･ジェネレータが持つ内蔵周辺 I/O レジスタを操作しても，CPU の動作クロックは変化しません)。

5.4.13 メモリ表示パネルでの最大アドレス空間表示について

【対 象】OCD(シリアル)/IECUBE，78K0

メモリパネル等でデバイス最大サイズの内部 ROM，内部高速 RAM，内部拡張 RAM にアクセスするには，メモリ・サイズ切り替えレジスタ(IMS)と内部拡張 RAM サイズ切り替えレジスタ(IXS)をフック処理に設定してください。

5.4.14 リターン実行，コール・スタック表示について

【対 象】OCD(JTAG)／OCD(シリアル)／IECUBE，78K0R,RL78

エディタパネルで(ソース・モードで)ステップ実行した場合，デバッグ・ツールは PSW レジスタの NP，EP，ID フラグをもとに割り込み処理中かどうかを判断しています。そのため，多重割り込みを使用している場合など，上記フラグやレジスタを変更した場合は，リターン実行や，コール・スタックの表示が正常に行なわれない場合があります。

5.4.15 ROM 化を行ったときのソフトウェアブレークについて

【対 象】全デバッグ・ツール，RL78 / V850

ROM 化の対象がコードの場合，そのコードに対してソフトウェアブレークを設定しても，RAM へのコピー時にブレーク用の命令が削除されるため，ブレークしません。OCD(JTAG)，OCD(シリアル)，IECUBE を使用している場合は，ハードウェアブレークを使用してください。なお，シミュレータを使用している場合，ハードウェアブレークを使用してもブレークしませんが，トレーサ，またはタイマーを ON にすることでブレークするようになります。

5.4.16 サブプロジェクトの追加について

【対 象】全デバッグ・ツール，全デバイス

デバッグ・ツール接続中にサブプロジェクトを追加すると，ダウンロード等に失敗することがあります。サブプロジェクトの追加は，デバッグ・ツール切断中にしてください。

5.4.17 フラッシュ・オプションの設定について

【対 象】OCD(JTAG)，V850E2M

下記フラッシュ・オプションで以下に示すビットは 1 固定になります。0 を書き込みたい場合は，フラッシュ・プログラマをお使いください。

・オンチップ・デバッグ・セキュリティ ID のビット 95(セキュリティ・ロック信号解除)

・オプション・バイト 0 のビット 31(デバッグ・インタフェース接続禁止ビット)

5.4.18 スタック・トレース表示についての注意

【対 象】全デバッグ・ツール，78K0

スタック・トレース表示機能は，スタックにフレーム・ポインタ(HL)を Push しない関数(noauto，norec 関数等)がある場合やメモリ・バンクを使っている場合には，main 関数まで正しく表示されないことがあります。

また，スタックにフレーム・ポインタ(HL)を Push しない関数(noauto，norec 関数等)や，メモリ・バンク関数からリターン実行した場合，フリーラン状態になることがあります。

5.4.19 メモリ・バンク内でステップ・インした際の注意

【対 象】全デバッグ・ツール，78K0

メモリ・バンク内のユーザ定義ライブラリ関数またはメモリ・バンク内のデバッグ情報なし関数にソース・レベルでステップ・インした場合，バンク切り替えライブラリ内でブレークします。

5.4.20 ローカル変数の表示に関する注意

【対 象】全デバッグ・ツール，78K0

スタック・トレース・パネルで，カレント PC のスコープ外のローカル変数は，正しく表示できません。

5.4.21 逆アセンブル・ウインドウについての注意

【対 象】全デバッグ・ツール，78K0

コモン領域内の命令を逆アセンブル・ウインドウで表示する際，表示される命令にメモリ・バンク領域内のシンボルが使用されていると，異なるバンクのシンボルを表示してしまう場合があります。

5.4.22 ブレークポイントの設定等が不正になる注意

【対 象】全デバッグ・ツール，全デバイス

関数名や変数名を，先頭のアンダー・バーの有無などで使い分けている場合，デバッガが誤認識してしまい，シンボル変換や，ブレークポイントの設定が不正になる場合があります。

例えば_reset と__reset という 2 つの関数が存在していた場合などが該当します。

5.4.23 ブレークの競合に関する注意

【対 象】IECUBE/OCD(JTAG)/OCD(シリアル)，V850

関数名や変数名を，先頭のアンダー・バーの有無などで使い分けている場合，デバッガが誤認識してしまい，シンボル変換や，ブレークポイントの設定が不正になる場合があります。

ソフトウェアブレークと以下のハードウェアブレークが競合した場合，PC の値を不正に補正する場合があります。ソフトウェアブレークではなく，ハードウェアブレークを使用してください。

(1) トレース・フル・ブレーク
(2) ノンマップ・ブレーク
(3) ライト・プロテクト・ブレーク
(4) IO イリーガル・アクセス・ブレーク
(5) 停止ボタンによる強制ブレーク
(6) イベント・ブレーク(ハードウェアブレーク)
(7) タイムアウト・ブレーク

5.4.24 V850E2 のシミュレーションについて

【対 象】シミュレータ，V850E2

V850E2 用シミュレータ(命令)は，以下に示す機能をシミュレーションします。それ以外の機能はシミュレーションしません。

・CPU 命令
・例外
・システム・レジスタ保護
・メモリ保護
・タイミング監視機能
・浮動小数点演算機能

また，以下に示す点に注意してください。

(1) 外部メモリ領域へのアクセスはできません。
(2) 浮動小数点ユニット(FPU)のシミュレーションの結果は，実デバイスと誤差が生じます。シミュレータは，Visual C++の浮動小数点ライブラリを用いて，80ビットで計算した結果をレジスタに格納します。
(3) 以下の例外要因はサポートしていません。
システム･エラー例外，メモリ･エラー例外
(4) キャッシュ・メモリのシミュレーションはサポートしていません。
(5) SYNCE/SYNCM/SYNCPの3命令はサポートしていません。実行した場合は，NOPと同様の動作になります。
(6) CPUの動作クロックは4MHz固定となります。プロパティ･パネルでメイン･クロック周波数の設定を変更してもCPUの動作クロックに反映されません。
(7) データ･フラッシュの領域にアクセスできません。アクセスした場合，エラーが発生してブレークします。
(8) オプション･バイト格納レジスタ"OPBT0"の値は常に0となります。
(9) EH_RESET レジスタの機能はサポートしていません。CPU リセットが発生した時のリセット・アドレスは 0x0 に固定しています。
(10) 各命令の実行クロック数は，命令実行直後に他の命令を実行する場合の実行クロック数となります。

5.4.25 同名の変数の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

異なるソースファイルに無名名前空間を記述し、その中に同名の変数を定義した場合、ウォッチパネルでは、最初に見つかる変数の情報を表示します。

5.4.26 メンバ変数ポインタの取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

下記のプログラムに定義されたメンバ変数ポインタ"mp1"をウォッチパネルおよびローカル変数パネルに登録した場合、型名に"int Foo::*"ではなく"int *"と表示されます。

```
class Foo {
    int  m1;
};
int Foo::*mp1 = &Foo::m1;
```

5.4.27 レジスタ割付された共用体の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

共用体がレジスタに割り付いている場合、共用体のメンバはレジスタの下位バイトから割り付いているとみなします。このため、ビッグエンディアンの場合はメンバの値を正しく表示できません。

5.4.28 char型の引数を持つ同名の関数の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

下記のように char 型を使用した 3 つの関数を定義した場合、"Func(signed char)"のアドレスを正しく表示できません。（"Func(char)"のアドレスを表示します。）

```
void Func(char);
void Func(signed char);
void Func(unsigned char);
```

5.4.29 char型の一次元配列の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

下記のような char 型の一次元配列がレジスタやメモリの複数個所に割り付いていた場合は、ウォッチパネルおよびローカル変数パネルに配列"array"を登録しても値のカラムに文字列を表示できません。（"" が値のカラムに表示されます。）

```
char array[5] = "ABCD";
```

5.4.30 オーバーレイ・セクションの優先セクションの変更に関する注意事項

【対 象】全デバッグ・ツール，RX

オーバーレイ・セクションの優先セクションを変更しても、デバッガの機能には直ぐには反映されません。

例えば、エディタ上のアドレス表示については、ファイルを一旦閉じ、再度開くことにより反映されます。また、ウォッチパネル上の変数表示については、1 回ステップを実行することにより反映されます。

5.4.31 レジスタ割付された変数の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

ローカル変数パネルの[スコープ]にて"カレント"以外を選択中は、レジスタに割りついた変数の値は正しく表示できません。また、その変数の値を編集することも出来ません。

5.4.32 変数の割り付き位置表示の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

以下の条件を全て満たす変数を定義した場合、ウォッチパネル、ローカル変数パネルでは、対象のメンバ変数の割り付き位置文字列が変数全体の割り付き位置文字列で表示されます。

<条件>

(1)定義した変数が複数のアドレスやレジスタに割りついている。

（アドレスカラムに 2 つ以上のアドレスやレジスタ名が表示される場合）

(2)変数に以下の型のメンバが定義されている。

- 構造体、クラス、配列、共用体のいずれか

<例>

```
struct Mem {
     long m_base;
};
struct Sample {
    long m_a;
    struct Mem m_b;    <-条件(2)に該当
};

main () {
    struct Sample obj;
}
```

表示結果：

```
"obj"                          -       { R1:REG, R2:REG }      (struct Sample)
    L m_a              0x00000000      { R1:REG }              (long)
    L m_b                          -       { R1:REG, R2:REG }      (struct Base)
        L m_base       0x00000000      { R2:REG }              (long)
```

5.4.33 変数をキャストする際の取り扱いに関する注意事項

【対 象】全デバッグ・ツール，RX

ウォッチパネルにて、キャストする変数の型がクラスの場合、基底クラスおよび派生クラスへのキャストはできません。また、キャストする変数が、構造体または共用体の場合、その変数の型以外へのキャストはできません。

```
class AAA [
    int m_aaa;
} objA;
class BBB : public AAA {   //BBB は AAA を継承している
    int m_bbb;
} objB;
class CCC {   //CCC は AAA を継承していない
    int m_ccc;
} objC

class AAA* pa = objA;
class BBB* pb = objB;
class CCC* pc = objC;
```

"(AAA*)pa" ・・・使用可能

"(BBB*)pb" ・・・使用可能

"(AAA*)pb" ・・・pb の指すクラス”BBB”は AAA を継承しているが、制限事項により使用不可

"(CCC*)pc" ・・・使用可能

"(AAA*)pc" ・・・pc の指すクラス”CCC”が AAA から継承されていないため使用不可

5.4.34 RRM機能使用時のメモリ・パネルの取り扱いに関する注意事項

【対 象】E20 エミュレータ(JTAG)，RX

プログラム実行中、RRM 機能使用時、ウォッチ・パネルに設定した変数の値をメモリ・パネルで表示すると、"**"でなく"00"を表示する場合があります。

RRM 機能使用時はメモリ・パネルでなくウォッチ・パネルを使用してください。

5.4.35 プログラム実行中のハードウェアブレーク設定に関する注意事項

【対 象】OCD(JTAG)，OCD(シリアル)，RX

「カーソル位置まで実行」、「リセット時に main 関数の先頭まで実行」を実行中にハードウェアブレーク設定を行うと、実行がブレークで停止しなくなります。

実行中は、ハードウェアブレークの設定を変更しないでください。

5.4.36 PCスリープ状態からの復帰に関する注意事項

【対 象】OCD(JTAG), OCD(シリアル), RX

Windows Vista または Windows 7 でデバッグ中に PC がスリープ状態または休止状態に移行した場合、復帰後にデバッグを継続できません。

Windows Vista または Windows 7 で使用する場合は、PC がスリープ状態および休止状態に移行しない設定でご使用ください。

5.4.37 プログラム実行中のトレース停止、再開の注意事項

【対 象】全デバッグ・ツール, RX

トレース開始イベント、あるいはトレース終了イベントを設定している場合、プログラム実行中のトレース停止・再開はできません。

5.4.38 トレースのタイムスタンプについての注意事項

【対 象】OCD(JTAG), OCD(シリアル), RX

トレース情報に付加されるタイムスタンプは、フレーム間の経過時間がトレースクロックの 20 ビット分を超える場合、および、トレース出力でロストが発生した場合、正しい時間となりません。

5.4.39 フラッシュ・セルフ・エミュレーションについて

【対 象】IECUBE，V850

IECUBE でフラッシュ・セルフ・プログラミングのエミュレーションを行う場合，次に示すフラッシュ関数のエミュレーション可否，および注意事項を確認してください。

表 フラッシュ・セルフ・プログラミングType01使用時のフラッシュ関数エミュレーション可否

フラッシュ関数名	機能概要および制限	エミュレーション可否
FlashEnv	フラッシュ環境初期化／終了関数	○
FlashBlockErase	1ブロック消去関数	○
FlashWordWrite	1ワード分の書き込み関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスでフェイル・セーフ・ブレークが発生します。	△
FlashBlockIVerify	1ブロックの内部ベリファイ処理関数	○
FlashBlockBlankCheck	1ブロックのブランク・チェック関数	○
FlashGetInfo	フラッシュ情報取得関数	
	Option = 2：CPU番号，およびCPUで持っているブロック総数 制限：CPU番号はコンフィギュレーションで設定したデバイス名称(4桁番号)が返却されます。	△
	Option = 3：セキュリティ情報	○
	Option = 4：ブート領域入れ替え情報取得 制限：ブート領域入れ替え情報が反映されません。	△
	Option = 5＋ブロック番号： ブロックの最終アドレス取得	○
FlashSetInfo	フラッシュ情報設定関数 制限：ブート領域入れ替え設定が無視されます。	△
FlashStatusCheck	直前に実行したフラッシュ関数の動作状況確認関数 制限：FlashBlockEraseおよびFlashBlockBlankCheckでFE_BUSYからFE_OKになるタイミングが実デバイスと異なります。	△
FlashBootSwap	ブート領域のブロック入れ替え関数	×
FlashSetUserHandler	ユーザ割り込みハンドラ登録関数	○
FlashFLMDCheck	FLMD0端子の状態チェック関数	○
FlashSetInfoEx	フラッシュ情報設定関数 制限：ブート領域入れ替え設定が無視されます。	△
FlashNWordRead	Nワード分の読み出し関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスでフェイル・セーフ・ブレークが発生します。	△

○：エミュレーション可能， △：制限付きでエミュレーション可能， ×：エミュレーション不可

表　フラッシュ・セルフ・プログラミングType02c使用時のフラッシュ関数エミュレーション可否

フラッシュ関数名	機能概要および制限	エミュレーション可否
FlashEnv	フラッシュ環境初期化／終了関数	○
FlashBlockErase	1ブロック消去関数	○
FlashWordWrite	1ワード分の書き込み関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスで フェイル・セーフ・ブレークが発生します。	△
FlashBlockIVerify	1ブロックの内部ベリファイ処理関数	○
FlashBlockBlankCheck	1ブロックのブランク・チェック関数	○
FlashGetInfo	フラッシュ情報取得関数	
	Option = 2：CPU番号，および，CPUで持っているブロック総数 制限：CPU番号はDFの名称(4桁番号)が返却されます。	△
	Option = 3：セキュリティ情報	○
	Option = 4：ブート領域入れ替え情報取得 制限：ブート領域入れ替え情報が反映されません。	△
	Option = 5＋ブロック番号： ブロックの最終アドレス取得	○
FlashSetInfo	フラッシュ情報設定関数 制限：ブート領域入れ替え設定が無視されます。	△
FlashBootSwap	ブート領域のブロック入れ替え関数	×
FlashFLMDCheck	FLMD0端子の状態チェック関数	○
FlashWordRead	データ読み出し関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスで フェイル・セーフ・ブレークが発生します。	△

○：エミュレーション可能，　△：制限付きでエミュレーション可能，　×：エミュレーション不可

表　フラッシュ・セルフ・プログラミングType03使用時のフラッシュ関数エミュレーション可否

フラッシュ関数名	機能概要および制限	エミュレーション可否
FlashEnv	フラッシュ環境初期化／終了関数	○
FlashBlockErase	1ブロック消去関数	○
FlashWordWrite	1ワード分の書き込み関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスで フェイル・セーフ・ブレークが発生します。	△
FlashBlockIVerify	1ブロックの内部ベリファイ処理関数	○
FlashBlockBlankCheck	1ブロックのブランク・チェック関数	○
FlashGetInfo	フラッシュ情報取得関数	
	Option = 2：CPU番号，および，CPUで持っているブロック総数 制限：CPU番号はDFの名称(4桁番号)が返却されます。	△
	Option = 3：セキュリティ情報	○
	Option = 4：ブート領域入れ替え情報取得 制限：ブート領域入れ替え情報が反映されません。	△
	Option = 5＋ブロック番号： ブロックの最終アドレス取得	○
FlashSetInfo	フラッシュ情報設定関数 制限：ブート領域入れ替え設定が無視されます。	△
FlashBootSwap	ブート領域のブロック入れ替え関数	×
FlashFLMDCheck	FLMD0端子の状態チェック関数	○
FlashWordRead	データ読み出し関数 制限：第三引数にガード領域が指定されていた場合，意図しないアドレスで フェイル・セーフ・ブレークが発生します。	△
FlashIVerify	内部ベリファイ関数(EEPROM専用)	×
FlashBlankCheck	ブランク・チェック関数(EEPROM専用)	×
EEPROM_Init	EEPROM 領域初期化関数(EEPROM専用)	×
EEPROM_Write	EEPROM 書き込み関数(EEPROM専用)	×
EEPROM_Read	EEPROM 読み出し関数(EEPROM専用)	×
EEPROM_Copy	EEPROM コピー関数(EEPROM専用)	×
EEPROM_VChK	EEPROM 有効領域チェック関数(EEPROM専用)	×
EEPROM_Erase	EEPROM 消去関数(EEPROM専用)	×

○：エミュレーション可能，　△：制限付きでエミュレーション可能，　×：エミュレーション不可

表　フラッシュ・セルフ・プログラミングType04使用時のフラッシュ関数エミュレーション可否

フラッシュ関数名	機能概要および制限		エミュレーション可否
FlashInit	セルフ・ライブラリの初期化関数		○
FlashEnv	フラッシュ環境初期化／終了関数		○
FlashBlockErase	1ブロック消去関数		○
FlashWordWrite	1ワード分の書き込み関数		○
FlashBlockIVerify	1ブロックの内部ベリファイ処理関数		○
FlashBlockBlankCheck	1ブロックのブランク・チェック関数		○
FlashGetInfo	フラッシュ情報取得関数		
	Option = 2	デバイス情報(ブロック総数，デバイス番号)	○
	Option = 3	セキュリティ・フラグ，ブート・ブロックの末尾ブロック番号	○
	Option = 4	デバイス情報	○
	Option = 5	リセット・ベクタ・アドレス	○
	Option=6 + ブロック番号n	ブロック番号nの末尾アドレス	○
FlashSetInfo	フラッシュ情報設定関数 制限：ブート領域入れ替え設定が無視されます。		△
FlashStatusCheck	直前に実行したフラッシュ関数の動作状況確認 制限：SELFLIB_BUSYは返却されません。		△
FlashBootSwap	ブート領域のブロック入れ替え関数 制限：関数の呼び出しはできますが，ブート・スワップは実行されません。		×
FlashFLMDCheck	FLMD0端子の状態チェック関数		○

○：エミュレーション可能，　△：制限付きでエミュレーション可能，　×：エミュレーション不可

No	注意事項
1	以下の場合，フラッシュ・メモリ・セルフ・プログラミング・エミュレーションを有効にできません。 (A)内蔵ROMサイズがデフォルトサイズ以外に設定した場合 (B)実行前ブレークを2個使用している場合 回避策としては，実行前ブレーク1個を無効にするか，削除してください。
2	フラッシュ・メモリ・セルフ・プログラミング・エミュレーションを有効にすると，以下のデバッグ機能に制限がつきます。 (A)内蔵ROMと内蔵RAMサイズの変更はできません (B)プログラムDMM/擬似RRMの機能は使用できません。 (C)SPレジスタが適当な位置(内蔵RAM等)に初期化される前にイベント等のブレークが発生した場合は，スタックエリアに対する不正ブレーク要因となってしまいます。この間にブレークが発生する可能性がある場合は，プログラム実行前にSPを適当な値にしてください。 (D)ご使用のIECUBEで，下記制限事項が存在する場合，不正ブレークが発生する可能性があります。フェイル・セーフ・ブレーク設定ダイアログで内蔵RAMのNon Mapのチェックをはずしてください。 「内蔵RAMでプログラム実行時のイリーガル・ブレーク制限事項」
3	フラッシュ・メモリ・セルフ・プログラミング・エミュレーションを有効に設定すると，0番地から4バイト分は予約領域になり，0番地にjr 0xfffd6の4バイト命令が書き込まれます。 そのため，リセットベクタ0番地で使用の際には，スタート・アップルーチンは4番地から配置してください。 また，フラッシュ・メモリ・セルフ・プログラミングを有効→無効に設定すると，0番地から4バイト分は0が書き込まれます。リセットベクタ0番地以外でご使用の際も0番地に分岐させるコードは記述しないでください。 FlashNWordReadまたはFlashWordReadで予約領域を読み出したときは，0の値が読めます。 実デバイスでも同一のプログラムを動作させるためには，以下に記述するコードを作成してください。 # RESET handler （0番地の場合） .section “RESET”, text jr __start --jr 0xfffd6に書き換えられる jr __start
4	リセット・ベクタ・ハンドリング指定アドレスに，0番地を指定するとリセットベクタは4番地になります。 0番地以外を指定すると+4されずに指定アドレスがリセットベクタになります。
5	FlashBlockErase()とFlashBlockBlankCheck()後のFlashStatusCheck()の動作について，エミュレーション時は，FlashStatusCheck()の返却値がFE_BUSYからFE_OKになるまでのタイミングが実デバイスと異なりますので注意してください。
6	FlashWordWrite，FlashWordRead，FlashNWordReadの第3引数に指定したアドレスがガード領域であった場合，不正なメモリアクセスとなり，意図しないアドレスでフェイル・セーフ・ブレークが発生します。 FlashWordWrite，FlashWordRead，FlashNWordReadで指定するアドレスを適切なアドレスに修正してください。
7	フラッシュ・オプション設定ダイアログの各設定を有効にするためには，設定後必ずCPUリセットして，再実行するようにしてください。CPUリセットを行わず再実行した場合，設定が反映されない場合があります。

No	注意事項
8	デバッガのワーク分として，最低84(54H)バイトのスタックを確保してください。ブレーク時やフラッシュ書き換えエミュレーション処理時に最低84(54H)バイトのスタックを消費します。 割り込みを許可する場合は，さらに，デバッガのワーク分として，84(54H)バイトのスタックが必要です。また，多重割り込みの場合は，1段ごとに84(54H)バイトのスタックを確保する必要があります。
9	CPUリセット時に内蔵RAMの内容が破壊されます。通常，実デバイスにおいて，リセット後の内蔵RAMデータは保証していませんが，動作が異なる場合がありますので注意してください。
10	フラッシュ関数を仕様範囲外の方法で使用したり，サポートしていないフラッシュ関数を呼び出した場合，戻り値として"1" が返ります。
11	Type04 のエミュレーション時は，以下の制限があります。 （A）内蔵RAM の最終アドレスから48 バイト分は予約領域としてデバッガが使用します。 （B）内蔵フラッシュ・メモリが1M バイトのデバイスの場合，0xFF300 以降の内蔵フラッシュ領域は，デバッグ・ツールが使用します。 （C）フラッシュ関数をアセンブル・モードでステップ実行した場合，デバッグ・ツールのエミュレーション用コードが実行されるため，実際のデバイスが実行するコードと異なります。このため，デバッグ時はソース・モードでステップ実行してください。

5.5 解析ツールの注意事項

5.5.1 関数一覧パネルに関する注意事項（CC-RX(C++言語)）

- テンプレート関数／テンプレート・クラス中に定義されているメンバ関数の注意事項は次のようになります。

  [ファイル名]列には、「（定義箇所なし）」と表示されます。

  [引数]列には、引数の型のみが表示され、引数名が表示されません。

  テンプレート・クラス中に定義されたメンバ関数の[開始アドレス]／[終了アドレス]列には、「-(ハイフン)」が表示されます。

  [開始アドレス]列に「-(ハイフン)」が表示されている場合は、[エディタ]パネルへのジャンプ／[逆アセンブル]パネルへのジャンプ／[メモリ]パネルへのジャンプができません。

  [全ての参照の検索]メニューにて、定義箇所が表示されません。また、参照している関数／変数の情報が表示されません。

  テンプレート関数／テンプレート・クラス中に定義されているメンバ関数内で参照している関数の参照回数がカウントされません。同様に、[全ての参照の検索]メニューにて、参照情報が表示されません。

  テンプレート・クラス中に定義されているメンバ関数の場合、[関数の先頭にブレークを設定]メニューにて、関数の先頭にブレーク・ポイントを設定できません。
- クラス宣言にて定義されているメンバ関数が、宣言のみで使用されていない場合は、ファイル名が表示されません。定義箇所がない関数として扱います。
- 関数の引数にクラス型を指定すると、[開始アドレス]／[終了アドレス]／[コード・サイズ]列の値を「-(ハイフン)」と表示します。
- 関数の引数にsigned char型を指定している関数とchar型を指定しているオーバーロード関数が定義されている場合、 [開始アドレス]／[終了アドレス]／[コード・サイズ]列の値を「-(ハイフン)」と表示します。

5.5.2 変数一覧パネルに関する注意事項（CC-RX(C++言語)）

- テンプレート関数／テンプレート・クラス中に定義されているメンバ関数にて定義されている関数内 static 変数が表示されません。
- テンプレート関数／テンプレート・クラス中に定義されているメンバ関数内で参照している変数の参照回数がカウントされません。
- extern/volatile 宣言されていない const 変数は、コンパイラによって定数値に置換される。このため、変数として[変数一覧]パネルに表示されません。
- ファイルが異なる無名名前空間にて定義されている同名のグローバル変数の型は同じ型として扱います。

5.5.3 コール・グラフパネルに関する注意事項（CC-RX(C++言語)）

- デフォルトの設定では、テンプレート関数／テンプレート・クラス中に定義されているメンバ関数は、[コール・グラフ]パネルに表示されません。[定義箇所がない関数／変数をコール・グラフの表示対象とする]プロパティを「はい」に設定して表示させてください。
- テンプレート関数／テンプレート・クラス中に定義されているメンバ関数から呼び出している関数／参照している変数は、[コール・グラフ]パネルに表示されません。

5.5.4 クラス／メンバパネルに関する注意事項（CC-RX(C++言語)）

- テンプレート関数／テンプレート・クラス中に定義されているメンバ関数の注意事項は次のようになります。

  [ソースへジャンプ]メニューにて、定義位置にジャンプできません。

  [ソースの宣言へジャンプ]メニューにて、ソースの宣言位置にジャンプできません。
- 名前空間の別名は表示しません。

5.5.5 変数パネルに関する注意事項

- 無名構造体／無名共用体のアドレスとサイズは表示できません。
- 定義のみで使用されていない変数は、コンパイラの最適化によりサイズ情報が削除され、[サイズ]列の値を 0 と表示します。【CC-RX】

5.5.6 クラス／メンバパネルに関する注意事項

- 「マクロと定数」のノードは表示しません【CA850】
- 構造体／共用体／列挙体のノードから型の定義位置にソース・ジャンプできません。構造体／共用体の場合は、メンバのノードからメンバの定義位置にソース・ジャンプできません。列挙体の場合は、メンバを表示しません。【CX】
- 列挙型のメンバを選択して、ソース・ジャンプを実施した場合は、列挙型の定義位置にジャンプします。【CC-RX】

5.6 Python コンソールの注意事項

5.6.1 日本語入力に関する注意事項

Python コンソールでは日本語入力機能を有効にする事ができません。日本語を入力する場合は，外部エディタ等で作成しコピーし貼り付けてください。

5.6.2 プロンプト表示に関する注意事項

Python コンソールのプロンプトが>>>であるところが>>>>>>というように複数表示される場合や>>>の後に結果が表示され，キャレットの前に>>>がない場合があります。このような状態でも継続して関数を入力することが可能です。

5.6.3 フォルダやファイルへのパスに関する注意事項

IronPython では，¥(バックスラッシュ)を制御文字として認識します。例えば，先頭が t で始まるフォルダ名やファイル名の場合¥t で TAB 文字と認識してしまいます。これを回避するには次のように，""(パス指定)の前に r を記載してください。IronPython は""の中がパスと認識します。

(例) r"c:¥test¥test.py"

なお，パスの指定には¥(バックスラッシュ)ではなく/(スラッシュ)も使用可能ですが，一部関数ではスラッシュの使用によりエラーが発生することがあります。このため，スラッシュは使用しないでください。

5.6.4 ロードモジュールがないプロジェクトのスクリプト実行に関する注意事項

ロードモジュール・ファイルがないプロジェクトを使用して起動オプションでスクリプト指定した場合，もしくはプロジェクトファイル名.py をプロジェクト・ファイルと同じフォルダにおいてある場合は，通常プロジェクト読み込み後に自動的にスクリプトを実行しますが，ロードモジュール・ファイルがない場合は実行しません。

5.6.5 強制終了に関する注意事項

無限ループしているようなスクリプトを実行中に以下の操作を行うと，強制的に関数の実行を終了させるため，関数の実行結果がエラーになる場合があります。

1. Python コンソールのコンテキストメニューの「強制終了」や Ctrl+D で強制終了
2. 複数のプロジェクトをもつプロジェクトでアクティブプロジェクトを変更した場合

5.6.6 強制停止に関する注意事項

コンテキストメニューの[強制停止]を実行した場合、実行中のスクリプトや関数を強制停止しますが、[強制停止]した時点で実行が開始していないHook関数やCallback関数がある場合は、[強制停止]後順次実行します。

5.6.7 RX(シミュレータ、E1/E20 エミュレータ)使用時の注意事項

debugger.Assemble.LineAssemble 関数は、ビッグエンディアンに対応していません。このため、ビッグエンディアン領域の逆アセンブルは正しく行われません。

5.6.8 ラピッドスタート機能有効時の CubeSuiteExit()関数に関する注意事項

CubeSuiteExit()関数を使用して CubeSuite+を終了する場合、ラピッドスタート機能を有効にしていても CubeSuite+は常駐しません。

第6章　制限事項

本章では，制限事項について説明します。

6.1　デバッグ・ツールの制限事項

文中において，以下の略称を使用しています。

OCD(シリアル)　：MINICUBE2，E1 エミュレータ(シリアル)，E20 エミュレータ(シリアル)

OCD(JTAG)　　：MINICUBE，E1 エミュレータ(JTAG)，E20 エミュレータ(JTAG)

6.1.1　デバッグ･ツールの制限事項一覧

No.	対象ツール	対象デバイス	制限事項
1	OCD(JTAG)	V850E2M	フラッシュ・オプション設定に関する制限事項
2	IECUBE	78K0R/Kx3	フラッシュ・セルフ・エミュレーション設定に関する制限事項
3	OCD(JTAG) OCD(シリアル)	RX	デバッガ接続に関する制限事項

6.1.2　デバッグ・ツールの制限事項詳細

No.1　フラッシュ・オプション設定に関する制限事項

【対　象】OCD(JTAG)，OCD(シリアル)　V850E2M

【内　容】フラッシュ・オプション設定プロパティのセキュリティ設定とブート・ブロック・クラスタ設定にどのような値を設定しても無効になります。

【回避策】回避策はございません。

No.3　デバッガ接続に関する制限事項

【対　象】OCD(JTAG)，OCD(シリアル)　RX

【内　容】以下の操作を行うと、デバッガを接続できません。

(1)デバッガ RX E1(Serial)または RX E20(Serial)に接続します。

(2)デバッガの接続を解除します。

(3)プロジェクトを閉じずにデバッガを RX E1(JTAG)または RX E20(JTAG)に変更し接続すると、エラーが表示され接続できません。

【回避策】プロジェクトを一旦閉じてから再接続してください。

第7章　ドキュメント訂正

本章では，CubeSuite+のドキュメントの訂正について説明します。

7.1　エディタに関するドキュメント訂正事項

エディタに関するドキュメントの訂正について説明します。エディタの説明は，各種コーディング編，デバッグ編に記載があります。

7.1.1　ツールバーの説明追加

【追　加】

	デバッグ・ツール接続時に，通常表示モードと混合表示モードを切り替えます。 通常表示ではソース・プログラムを表示します。 混合表示ではソース・プログラムとアセンブル・コードを表示します。
	デバッグ・ツール接続時の混合表示モードにおいて、ステップ実行を行った際の動作について、ソース・レベルか、アセンブラ・レベルかを切り替えます。 ソース・レベルの場合には、プログラムカウンタ（PC）を示す表示が、ソース・プログラム行を示します。 アセンブラ・レベルの場合には、プログラムカウンタ（PC）を示す表示が、アセンブル・コード行を示します。
	デバッグ・ツール接続時の混合表示モードにおいて、現在のプログラムカウンタ（PC）位置を表示します。
	［ジャンプ前の位置へ戻る］を実行する前の位置へ進みます。
	［関数へジャンプ］を実行する前の位置へ戻ります。
	カラムの表示／非表示を切り替えます。クリックすることにより切り替えできるエリアの項目が表示されます。

7.1.2　ドラッグ＆ドロップの説明追加

【追　加】

シンボルをドラッグ＆ドロップすることにより、ウオッチ・パネルや解析グラフ（値の遷移）へのシンボル登録が可能です。

7.1.3 混合表示の説明追加

【追　加】

混合表示とは、デバッグ作業時、通常であればソースプログラムのみの表示であるが、アセンブラ・コードも合わせて表示する機能です。
混合表示の設定は、ツールバーで行います。

混合表示モードを使用する際には、次の点のご注意ください。
・編集することはできません。
・ファイルを保存することができません。
・切り取り、貼り付け、削除、やり直し、置換、アウトライン、インデント操作等、内容を変更する機能は使用できません。
・すべて選択機能を使用することができません。

7.1.4 リサイクル・モードの説明追加

【追　加】

リサイクル･モードとは、デバッグ作業時、ステップ実行等で、プログラム･カウンタ（PC）が複数のソース・ファイルをまたいで移動する場合、複数のソース・ファイルのエディタ パネルを表示するところを、1 枚のエディタ パネルで順に複数のソース・ファイルを表示するモードです。
リサイクル・モードの設定は［ツール］－［オプション］の［テキスト・エディタ］パネルにて行います。

7.1.5 分割バーの説明変更

【変更前】

縦と横の分割バーを使うことにより，エディタ パネルを分割して表示することができます。分割の上限は，**4 分割**までです。

【変更後】

縦と横の分割バーを使うことにより，エディタ パネルを分割して表示することができます。分割の上限は，**縦 2 分割、横 2 分割**までです。

7.2 起動編のドキュメント訂正事項

起動編(資料番号：R20UT0975JJ0100)のドキュメントの訂正について説明します。

7.2.1 起動高速化ユーティリティの説明追加

【追　加】

起動時間高速化ユーティリティは，ラピッド・スタート機能を未使用時に CubeSuite+の起動を高速化するユーティリティです。

CubeSuite+の実行ファイルと同じフォルダの AccelerationUtility.exe を実行し、“起動の高速化”ボタンを選択してください。デフォルトのインストール・フォルダは次の通りです。

C:¥Program Files¥Renesas Electronics¥CubeSuite+

本ユーティリティの効果は、パソコンにより違いがありますのでご注意ください。

7.2.2 Python クラスの一覧追加

【場　所】455 ページ　表 G-5 に追加

【変更前】

クラス名	機能概要
BreakCondition	ブレーク条件を作成します。
BuildCompletedEventArgs	ビルド完了時のパラメータを保持します。

【変更後】

クラス名	機能概要
BreakCondition	ブレーク条件を作成します。
BuildCompletedEventArgs	ビルド完了時のパラメータを保持します。
BreakpointInfo	ブレークポイント情報
DisassembleInfo	逆アセンブル情報
DownloadInfo	ダウンロード情報
MapInfo	マップ情報
StackInfo	スタック情報
TraceInfo	トレース情報
XRunBreakInfo	XRunBreak情報
XTimeInfo	タイマ情報

7.2.3 Python クラスの説明追加

【場　所】459 ページ

【変更前】

なし

【変更後】

BreakpointInfo

ブレークポイント情報(debugger.Breakpoint.Information の戻り値)

【型】

class BreakpointInfo:

Number = 0

Name = None

Enable = True

BreakType = BreakType.Hardware

Address1 = None

Address2 = None

Address3 = None

Address4 = None

【変数】

変数	説明
Number	イベント番号が格納されます。
Name	ブレークポイント名が格納されます。
Enable	有効の場合は True。無効な場合は False が格納されます。
BreakType	BreakType が格納されます。BreakType については CubeSuite+ Python クラス"BreakCondition"を参照してください。
Address1	アドレス情報 1 が文字列として格納されます。
Address2	アドレス情報 2 が文字列として格納されます。(組み合わせブレーク時のみ)
Address3	アドレス情報 3 が文字列として格納されます。(組み合わせブレーク時のみ)
Address4	アドレス情報 4 が文字列として格納されます。(組み合わせブレーク時のみ)

【機能】

BreakpointInfo は class の形式になっており debugger.Breakpoint.Information 関数を実行した場合戻り値として渡されます。

【例】

```
>>>info = debugger.Breakpoint.Information()
    1 ブレーク 0001 Enable    test1.c#_main+2
    2 ブレーク 0002 Disable test2.c#_sub4+10
>>>print info[0].Number
1
>>>print info[0].Name
ブレーク 0001
>>>print info[0].BreakType
Hardware
>>>print info[0].Enable
True
>>>print info[0].Address1
test1.c#_main+2
>>>print info[0].Address2
None
>>>print info[1].Number
2
>>>print info[1].Name
ブレーク 0002
>>>print info[1].BreakType
Hardware
>>>print info[1].Enable
False
>>>print info[1].Address1
test2.c#_sub4+10
>>>print info[1].Address2
None
>>>
```

DisassembleInfo

逆アセンブル情報(debugger.Assemble.Disassemble の戻り値)

【型】

class DisassembleInfo:

Address = 0

Code = None

Mnemonic = None

【変数】

変数	説明
Address	アドレスが格納されます。
Code	コード情報がバイト単位のコレクションとして格納されます。
Mnemonic	ニーモニック情報が格納されます。

【機能】

DisassembleInfo は class の形式になっており debugger.Assemble.Disassemble 関数の戻り値の構造です。

【例】

```
>>>info = debugger.Assemble.Disassemble("main", 4)          ※逆アセンブルの実行
0x000002DC      B51D        br _main+0x36
0x000002DE      0132        mov0x1、r6
0x000002E0      60FF3800    jarl _func_static1, lp
0x000002E4      63570100    st.w r10, 0x0[sp]
>>>print info[0].Address
732
>>>print info[0].Code[0]
181
>>>print info[0].Code[1]
29
>>>print Mnemonic
br _main+0x36
>>>print info[3].Address
740
>>>print info[3].Code[0]
99
>>>print info[3].Code[1]
87
>>>print info[3].Code[2]
1
>>>print info[3].Code[3]
0
>>>print info[3].Mnemonic
st.w r10, 0x0[sp]
>>>
```

DownloadInfo

ダウンロード情報(debugger.Download.Information の戻り値)

【型】

class DownloadInfo:

Number = None

Name = None

ObjectDownload = True

SymbolDownload = False

【変数】

変数	説明
Number	ダウンロード番号が格納されます。
Name	ファイル名が格納されます。
ObjectDownload	オブジェクト情報をダウンロードしている場合は True。ダウンロードしていない場合は False が格納されます。
SymbolDownload	シンボル情報をダウンロードしている場合は True。ダウンロードしていない場合は False が格納されます。

【機能】

DownloadInfo は class の形式になっており debugger.Download.Information 関数の戻り値の構造です。

【例】

```
>>>info = debugger.Download.Information()
    1: DefaultBuild¥sample.out
>>>print info[0].Number
1
>>>print info[0].Name
DefaultBuild¥sample.out
>>>print info[0].ObjectDownload
True
>>>print info[0].SymbolDownload
True
>>>
```

MapInfo

マップ情報(debugger.Map.Information の戻り値)

【型】

class MapInfo:

Number = 0

StartAddress = 0

EndAddress = 0

AccessSize = 0

MapTypeName = None

【変数】

変数	説明
Number	番号が格納されます。
StartAddress	マップ領域の開始アドレスが格納されます。
EndAddress	マップ領域の終了アドレスが格納されます。
AccessSize	マップ領域のアクセスサイズが格納されます。
MapTypeName	マップ領域の型名が格納されます。

【機能】

MapInfo は class の形式になっており debugger.Map.Information 関数の戻り値の構造です。

【例】

```
>>>info = debugger.Map.Information()                ※Map.Information 関数の実行
    1: 0x00000000 0x0003FFFF 32 (内蔵 ROM 領域)
    2: 0x00040000 0x00048FFF  8 (ノン・マップ領域)
    3: 0x00049000 0x001003FF  8 (エミュレーション ROM 領域)
    4: 0x00100400 0x03FF8FFF  8 (ノン・マップ領域)
    5: 0x03FF9000 0x03FFEFFF 32 (内蔵 RAM 領域)
    6: 0x03FFF000 0x03FFFFFF  8 (I/O レジスタ領域)
>>>print info[0].StartAddress
0
>>>print info[0].EndAddress
262143
>>>print info[0].AccessSize
32
>>>print info[0].MapTypeName
内蔵 ROM 領域
>>>print info[5].StartAddress
67104768
>>>print info[5].EndAddress
67108863
>>>print info[5].AccessSize
8
>>>print info[5].MapTypeName
I/O レジスタ領域
>>>
```

StackInfo

スタック情報(debugger.Where の戻り値)

【型】

class StackInfo:

Number = None

AddressInfoText = None

【変数】

変数	説明
Number	スタック番号が格納されます。
AddressInfoText	スタックのアドレス情報が文字列で格納されます。

【機能】

StackInfo は class の形式になっており debugger.Where 関数の戻り値の構造です。

【例】

```
>>>info = debugger.Where()
    1: test2.c#
    2: test1.c#main#41
>>>print info[0].Number
1
>>>print info[0].AddressInfoText
test2.c#
>>>info = debugger.Where()
1: test2.c#
--- Information below might be inaccurate.
2: test1.c#main#41
>>>print info[1].Number
None
>>>print info[1].AddressInfoText
--- Information below might be inaccurate.
>>>
```

TraceInfo

トレース情報(debugger.XTrace.Dump の戻り値)

【型】

class TraceInfo:

FrameNumber = None

Timestamp = None

FetchAddress = None

Mnemonic = None

ReadAddress = None

ReadData = None

WriteAddress = None

WriteData = None

VectorAddress = None

VectorData = None

IsDma = True

【変数】

変数	説明
FremNumber	フレーム番号情報が格納されます。
Timestamp	タイムスタンプ情報が格納されます。
FetchAddress	フェッチアドレス情報が格納されます。
Mnemonic	ニーモニック情報が格納されます。
ReadAddress	リードアドレス情報が格納されます。
ReadData	リードデータ情報が格納されます。
WriteAddress	ライトアドレス情報が格納されます。
WriteData	ライドデータ情報が格納されます。
VectorAddress	ベクターアドレス情報が格納されます。
Vect0rData	ベクターデータが格納されます。
IsDma	データが DMA の場合は True。DMA 以外の場合は False が格納されます。

【機能】

TraceInfo は class の形式になっており debugger.XTrace.Dump 関数の戻り値の構造です。

【例】

```
>>>info = debugger.XTrace.Dump(10)
    853  00h00min00s001ms704us000ns  0x000002c2  movhi 0xffff, gp, r1
    854  00h00min00s001ms706us000ns  0x000002c6  id.w 0x7ff4[r1], r6
    855  00h00min00s001ms706us000ns                    0x03ff9000 R 0x00000000
    856  00h00min00s001ms706us000ns  0x000002ca  movhi 0xffff, gp, r1
    857  00h00min00s001ms710us000ns  0x000002ce  movea 0x7ff8, r1, r7
    858  00h00min00s001ms712us000ns  0x000002d2  jarl _main+0x36
    859  00h00min00s001ms716us000ns  0x000002dc  br _main+0x36
    860  00h00min00s001ms720us000ns  0x00000312  prepare lp, 0x4
    861  00h00min00s001ms720us000ns                    0x03ff9308 W 0x000002d6
    862  00h00min00s001ms724us000ns  0x00000316  br _main+0x2
>>>print info[0].FrameNumber
853
>>>print info[0].Timestamp
1704000
>>>print info[0].FetchAddress
706
>>>print info[0].Mnemonic
movhi 0xffff, gp, r1
>>>print info[0].ReadAddress
None
>>>print info[0].ReadData
None
>>>print info[0].IsDma
False
>>>
>>>print info[2].FrameNumber
855
>>> print info[2].Timestamp
1706000
>>>print info[2].FetchAddress
None
>>>print info[2].Mnemonic
None
>>>print info[2].ReadAddress
        67080192
```

XRunBreakInfo

XRunBreak 情報(debugger.XRunBreak.Refer の戻り値)

【型】

class XRunBreakInfo:

Value = 0

TimeType = Timetype.Min

IsPeriodic = True

【変数】

変数	説明
Value	イベントの発生間隔値が格納されます。
TimeType	発生間隔値の単位が格納されます。詳細は CubeSuite+ Python 関数（デバッグ・ツール用）の"debugger.XRunBreak.Set"を参照してください。
IsPeriodic	指定時間毎にコールバックされるかどうかが格納されます。

【機能】

XRunBreakInfo は class の形式になっており debugger.XRunBreak.Refer 関数の戻り値の構造です。

【例】

```
>>>debugger.XRunBreak.Set(10, TimeType.S, True)
   >>>info = debugger.XRunBreak.Refer()
10Second Periodic
>>>print info.Value
10
>>>print info.TimeType
S
>>>print info.IsPeriodic
True
>>>
```

XTimeInfo

タイマ情報(debugger.XTime の戻り値)

【型】

class XTimeInfo:

Value = 0

IsCpuClock = False

IsOverFlow = False

【変数】

変数	説明
Value	タイマの計測値が格納されます。
IsCpuClock	CPU クロックの計測の場合は True。それい外の場合は False が格納されます。
IsOverFlow	オーバーフローが発生した場合 True。発生していない場合は False が格納されます。

【機能】

XTimeInfo は class の形式になっており debugger.XTime 関数の戻り値の構造です。

【例】

```
>>>info = debugger.XTime()
9820214200nsec
>>>print info.Value
9820214200
>>>print info.IsCpuClock
False
>>>print info.IsOverFlow
False
>>>
```

7.2.4 Python プロパティ(共通)の一覧追加と変更

【場　所】460 ページ　表 G-6 に追加と変更

【変更前】

プロパティ名	機能概要
common.ConsoleClear	アクティブ・プロジェクト変更時にPython コンソールの表示をクリアする／しないを設定／参照します。
common.Output	CubeSuite+ 用Python関数の戻り値，またはエラー内容を参照します。
common.ThrowExcept	Python関数の実行時に例外を発生させる／させないを設定／参照します。
common.UsedRemoting	Pythonコンソールの起動時に外部ツールと連携する機能を有効にする／しないを設定／参照します。
common.Version	CubeSuite+ のバージョンを参照します。
common.ViewLine	Pythonコンソールの表示桁数を設定／参照します。
common.ViewOutput	CubeSuite+ 用Python関数の実行結果，またはエラー内容をPythonコンソールに表示する／しないを設定／参照します。

【変更後】

プロパティ名	機能概要
common.ConsoleClear	アクティブ・プロジェクト変更時にPython コンソールの表示をクリアする／しないを設定／参照します。
common.EnableRemotingStartup	CubeSuite+の起動時に外部ツール連携機能を有効にする／しないの設定／参照を行います
common.Output	CubeSuite+ 用Python関数の戻り値，またはエラー内容を参照します。
common.ThrowExcept	Python関数の実行時に例外を発生させる／させないを設定／参照します。
common.UseRemoting	Pythonコンソールの起動中に外部ツールと連携する機能を有効にする／しないを設定／参照します。
common.Version	CubeSuite+ のバージョンを参照します。
common.ViewLine	Pythonコンソールの表示桁数を設定／参照します。
common.ViewOutput	CubeSuite+ 用Python関数の実行結果，またはエラー内容をPythonコンソールに表示する／しないを設定／参照します。

7.2.5 Python プロパティ(共通)の説明追加

【場　所】461 ページ　"common.ConsoleClear"の後ろに追加

【変更前】

なし

【変更後】

common.EnableRemotingStartup

CubeSuite+の起動時に外部ツール連携機能を有効にする/しないの設定/参照を行います。

【形式】

common.EnableRemotingStartup = *bool*

【設定】

設定	説明
bool	CubeSuite+の起動時に外部ツール連携機能を有効にする場合は True。無効にする場合は False を指定します。(デフォルト True)。 起動中に外部ツール連携を有効/無効にする場合は common.UseRemoting を使用してください。

【参照】

現在の設定値

【例】

```
>>>print common.EnableRemotingStartup
False
>>>common. EnableRemotingStartup = True
```

7.2.6 Python プロパティ(共通)の"common.UsedRemoting" 名称変更

【場 所】464 ページ "common.UsedRemoting"の記述全て

【変更前】

common.UsedRemoting

【変更後】

common.UseRemoting

7.2.7 Python プロパティ(共通)の"common.UsedRemoting" 説明変更

【場 所】464 ページ 「Python コンソールの起動時に」の記述全て

【変更前】

Python コンソールの起動時に

【変更後】

Pythonコンソールの起動中に

7.2.8 Python プロパティ(共通)の"common.UseRemoting"説明追加

【場 所】464 ページ "common.UseRemoting"の[設定]に追加

【変更前】

設定	説明
bool	Python コンソールの起動中に外部ツールと連携する機能を有効にするかどうかを設定します。 True ：外部ツールと連携する機能を有効にします（デフォルト）。 False ：外部ツールと連携する機能を無効にします。

【変更後】

設定	説明
bool	Python コンソールの起動中に外部ツールと連携する機能を有効にするかどうかを設定します。 True ：外部ツールと連携する機能を有効にします（デフォルト）。 False ：外部ツールと連携する機能を無効にします。 起動時に common.EnableRemotingStartup が True の場合は True が、False の場合は False が設定されます。

7.3 ビルド編のドキュメント訂正事項

ビルド編(資料番号：R20UT0730JJ0100, R20UT0783JJ0100)のドキュメントの訂正について説明します。

7.3.1 スタック見積もりツールの注意事項の説明追加

【場　所】 351 ページ　→解析対象関数

【追加後】 したがって，ユーザが記述したアセンブラ・ソース・ファイル，およびユーザが作成したライブラリ・ファイルに内包されている関数については，解析対象外となるため，スタックサイズ変更ダイアログを用いて該当情報を設定する必要があります。
また、割り込み関数も解析対処外となるため、スタックサイズ変更ダイアログを用いて該当情報を設定する必要があります。

【場　所】 362 ページ　→解析対象関数

【追加後】 したがって，ユーザが記述したアセンブラ・ソース・ファイル，およびユーザが作成したライブラリ・ファイルに内包されている関数については，解析対象外となるため，スタックサイズ変更ダイアログを用いて該当情報を設定する必要があります。
また、割り込み関数も解析対処外となるため、スタックサイズ変更ダイアログを用いて該当情報を設定する必要があります。

7.4 RX デバッグ編のドキュメント訂正事項

解析編(資料番号：R20UT1143JJ0100)のドキュメントの訂正について説明します。

7.4.1 関数へジャンプするの説明追加

【場　所】74 ページ

【変更前】（a）CC-RX の場合
- 対象が，C 言語の関数，変数，またはラベルである
- エディタパネルにフォーカスがある

【変更後】（a）CC-RX の場合
- 対象が，C 言語の関数，変数，またはラベルである
- エディタパネルにフォーカスがある

・デバッグ・ツールに接続していない場合
- アクティブプロジェクトの種類が“アプリケーション”である
- ［ダウンロードするファイル］の 1 番目に指定されたファイルに対象の関数が定義されている
- 上記ファイルにシンボル情報が存在する
- 対象の関数がグローバル関数である

・デバッグ・ツールに接続している場合
- ダウンロードしたロード・モジュール内にシンボル情報が存在する
- プログラムカウンタ(PC)の指すアドレスから呼び出し可能な関数である
  ※例えば､PC の指すアドレスのファイル以外で定義した static 関数へはジャンプできません。

なお、C++言語プログラムで関数へジャンプする機能を使う場合は、関数を特定するための以下の注意事項があります。選択された関数名の文字列で関数が特定できない場合、ジャンプができないか、別の同名関数へジャンプする可能性があります。

(1)クラスのメンバ関数
　対象の関数の所属するクラス名を含む必要があります。また、同名で引数の異なる関数が存在する場合は、引数の型名も含めてください。
　例："memfunc" : NG
　　"Class::memfunc(short)" : OK

(2)名前空間内に定義した関数
　対象の関数の所属する名前空間を全て含む必要があります。また、同名で引数の異なる関数が存在する場合は、引数の型名も含めてください。
　例："func" : NG
　　"Namespace1::Namespace2::func(int)" : OK

(3)テンプレート関数
　コンパイラが生成した関数の引数の型名を含めてください。
　例："template" : NG
　　"template(int, short)" : OK

7.5 78K0 デバッグ編ドキュメント訂正

78K0 デバッグ編(資料番号：R20UT0731JJ0100)のドキュメントの訂正について説明します。

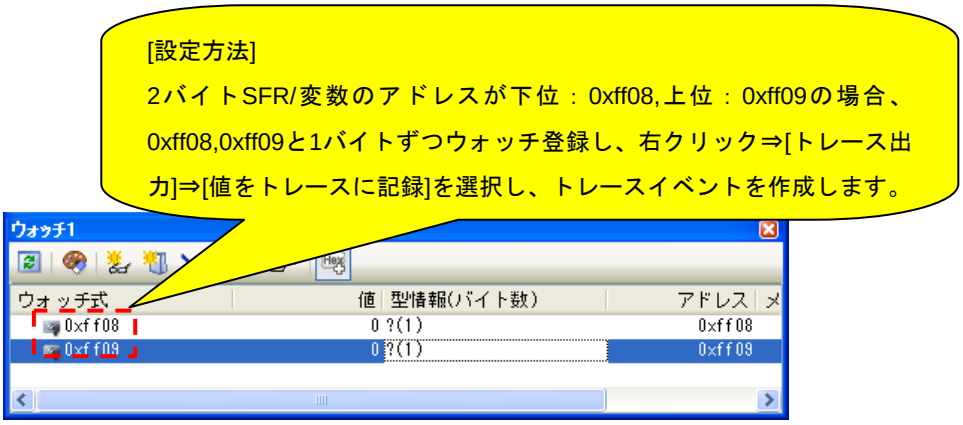
7.5.1 2 バイト SFR/変数のポイントトレースの説明追加

[場所] 138 ページ 「2.11.5 実行履歴を表示する」の前に追加

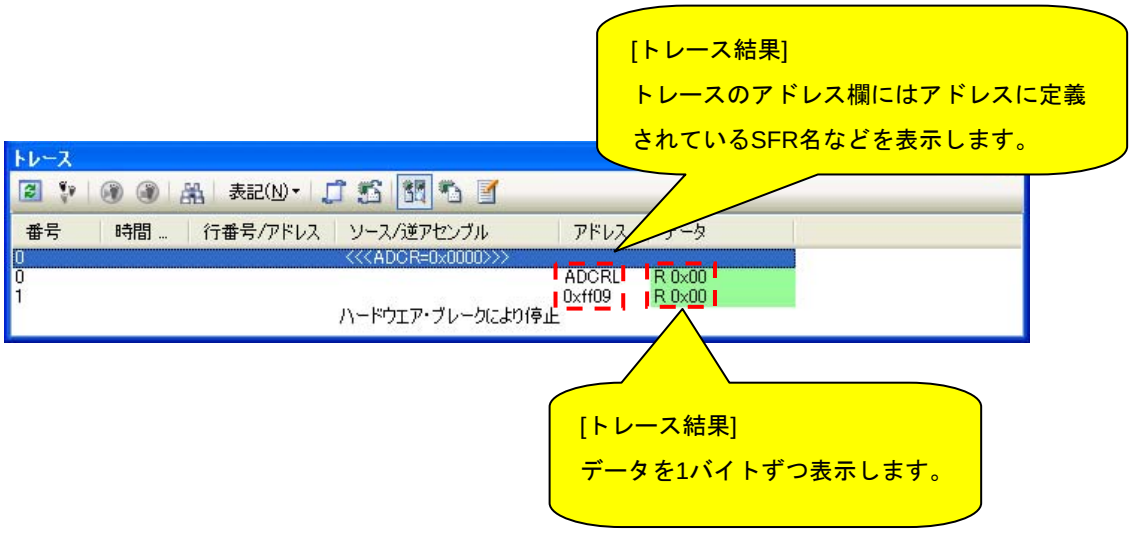
[追加後]

2.11.4(2) 2 バイト変数/SFR へのアクセスが発生したとき [IECUBE]

2 バイト SFR/変数のポイントトレースを行うには、上位 8 ビット、下位 8 ビットのアドレスを直接ウォッチパネルに登録し、トレースイベントを作成します。(図参照)



この設定を行った場合のトレース結果は以下のようになります。



ご注意書き

1. 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・システムの設計において、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て、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   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、産業用ロボット等
   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、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図し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損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。
9. 本資料に記載されている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・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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禁じます。

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/contact/