Rev 1.10.00

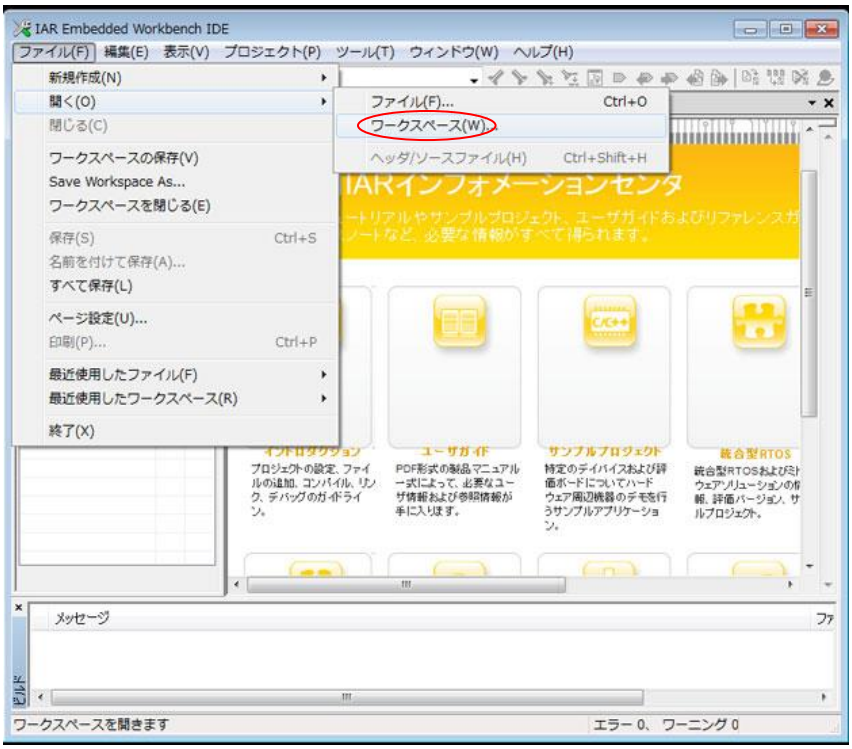
IAR（EWARM）ツールチェインの設定と必要事項の説明

（ルネサス　ＲＺ／Ａ１Ｈ用）

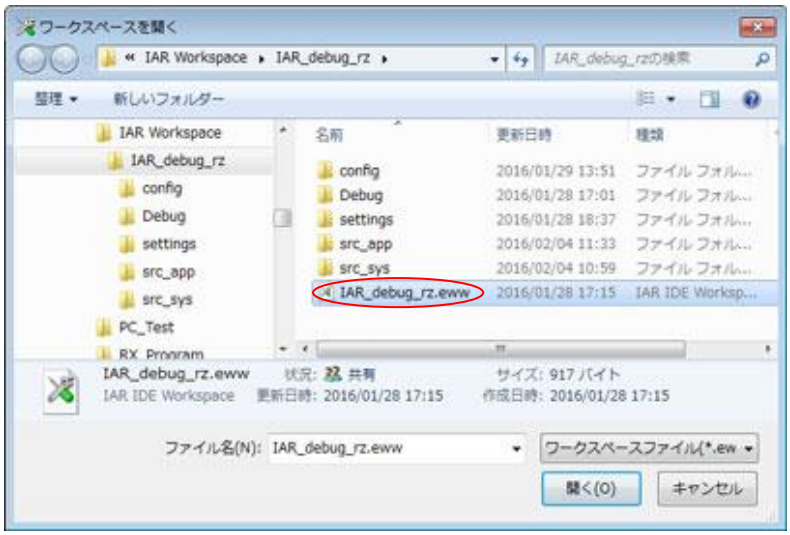
IAR(EWARM)ツールチェインの設定方法とサンプルプロジェクト「IAR_debug_rz」に必要な設定を説明します。

1．IAR Embedded Workbench IDE を起動する。

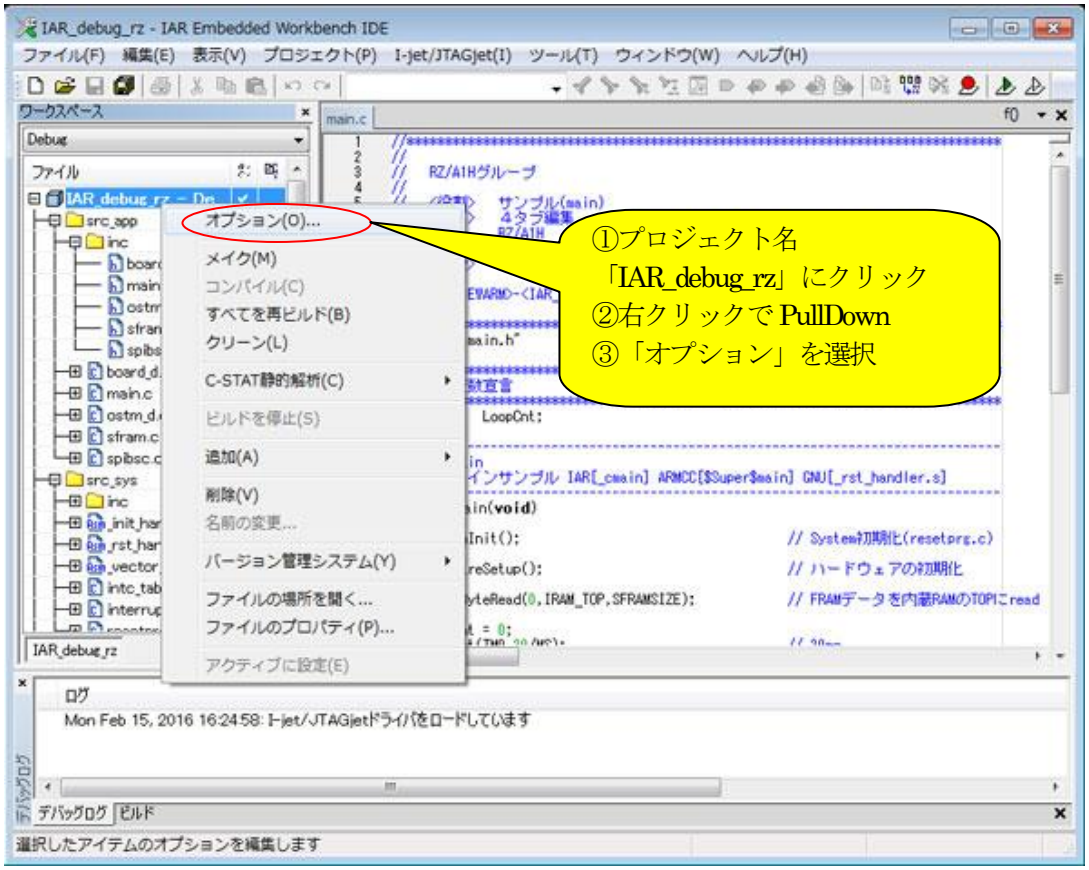
1）【ファイル】－【開く】－【ワークスペース】を選択する。



2）ワークスペースファイル「IAR_debug_rz.eww」を選択する。

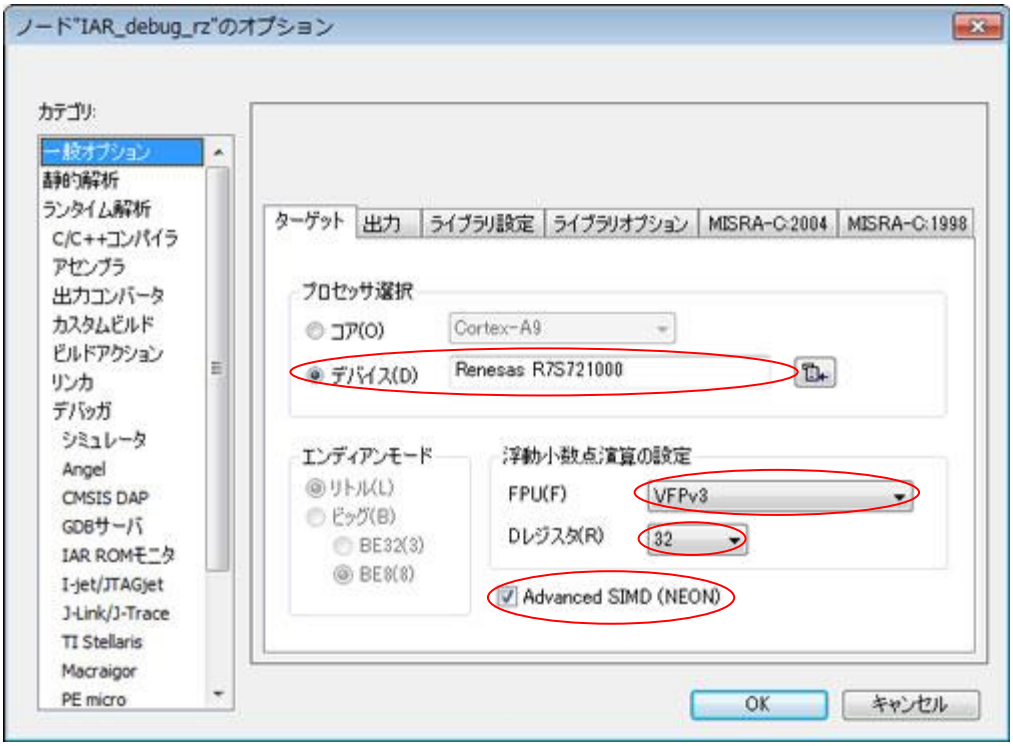


2．各ツールの設定内容を確認する。

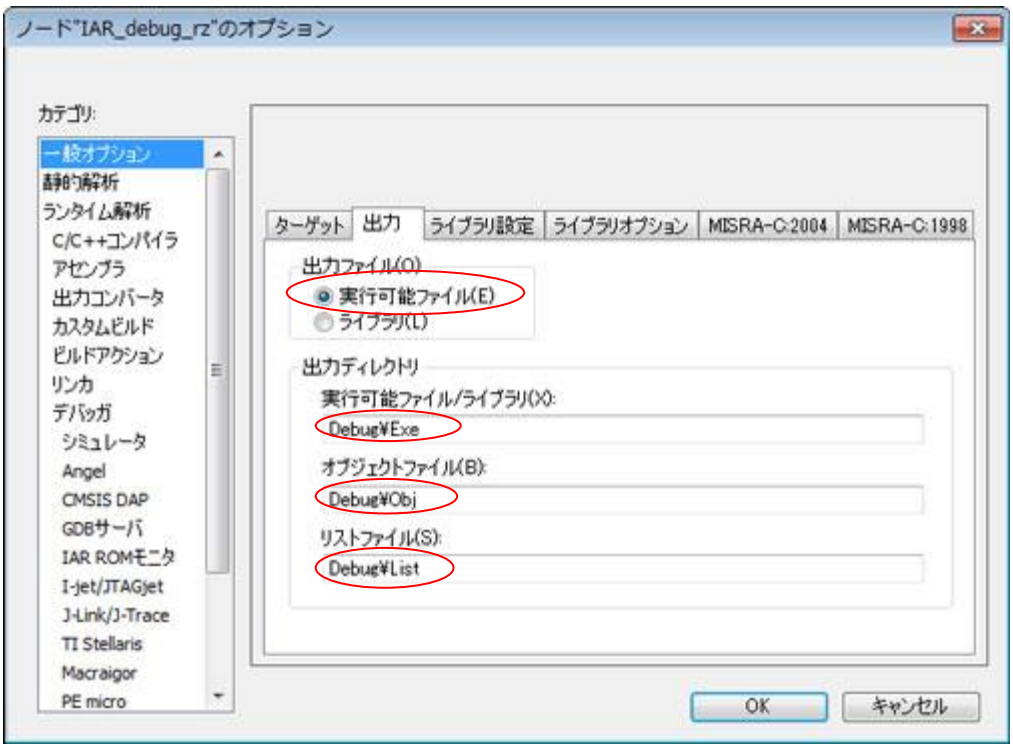


２－１．一般オプションの確認

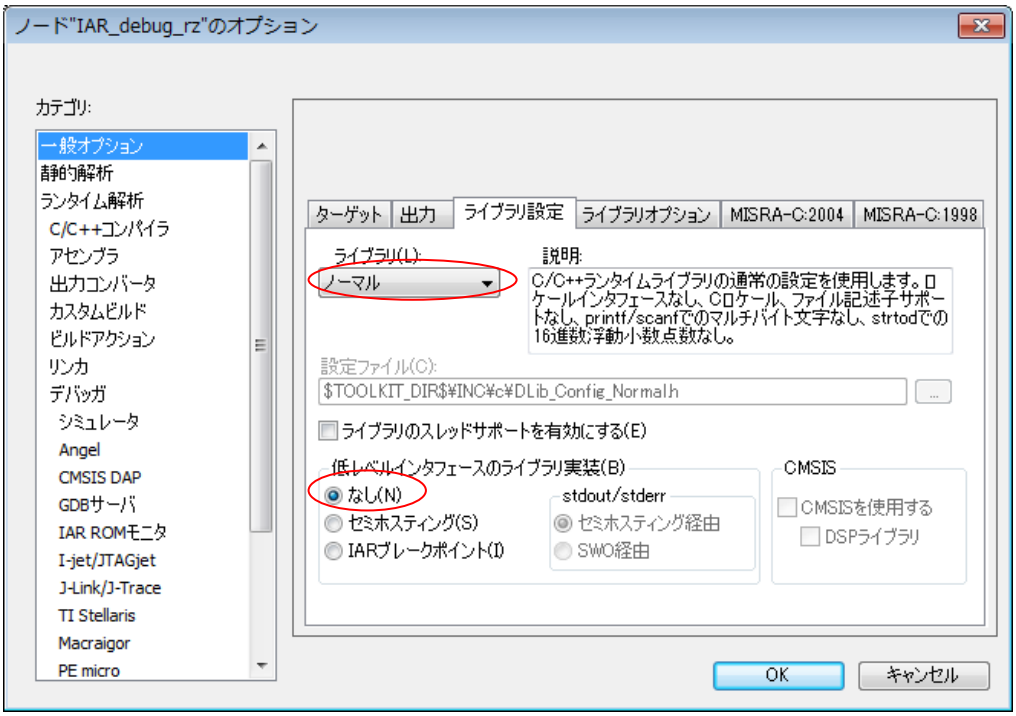
１）ターゲット



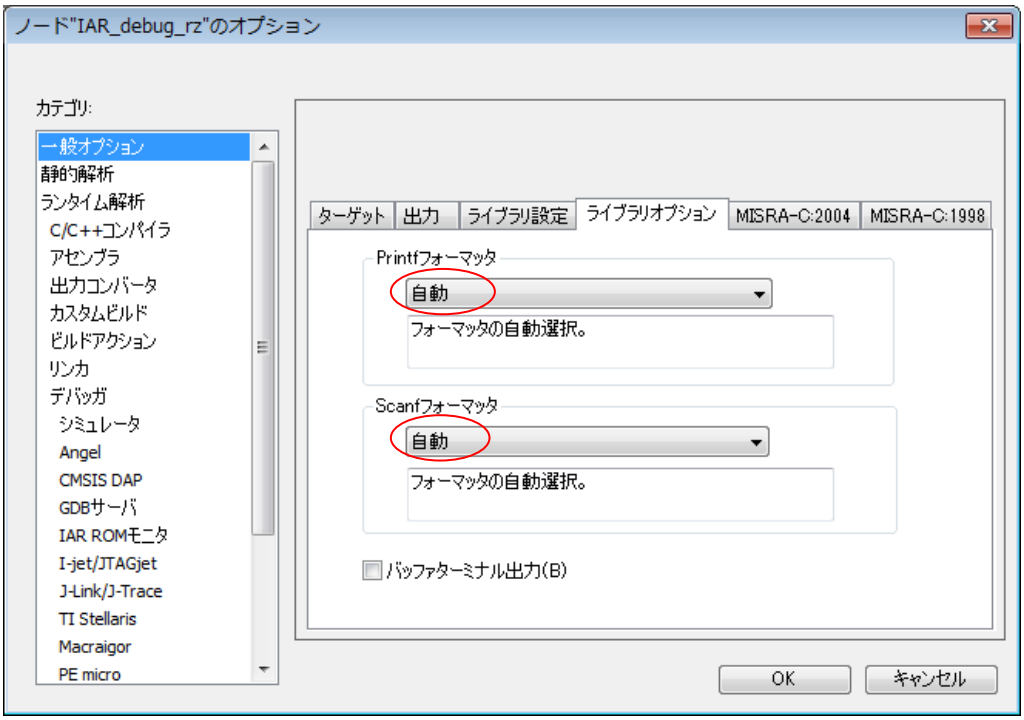
２）出力



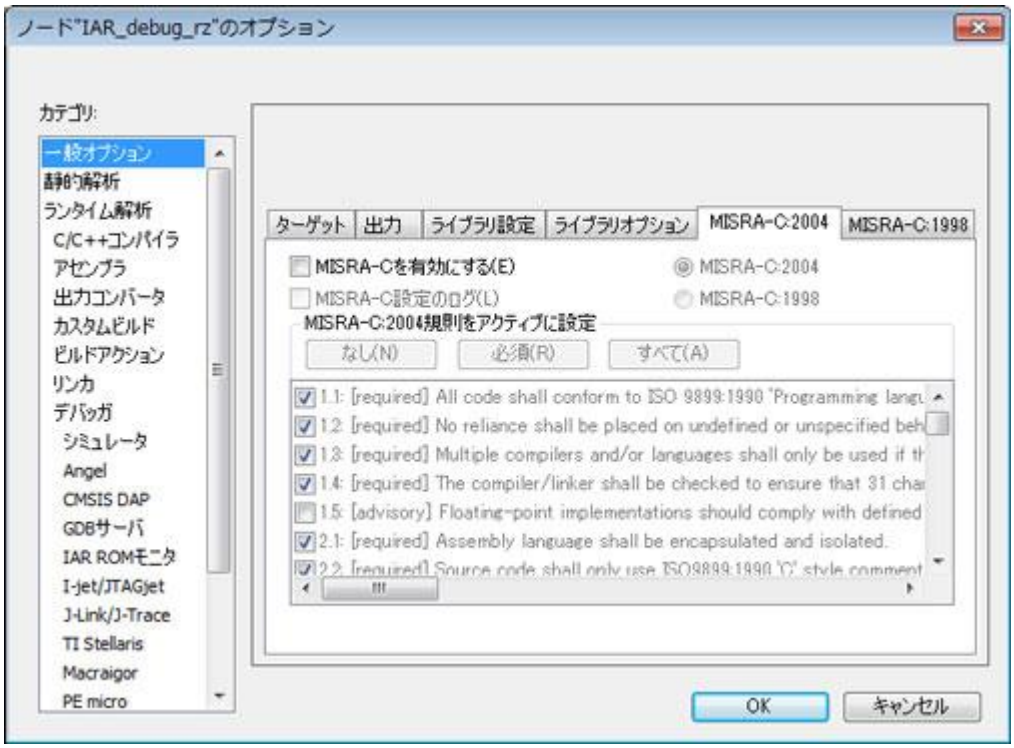
3）ライブラリ設定



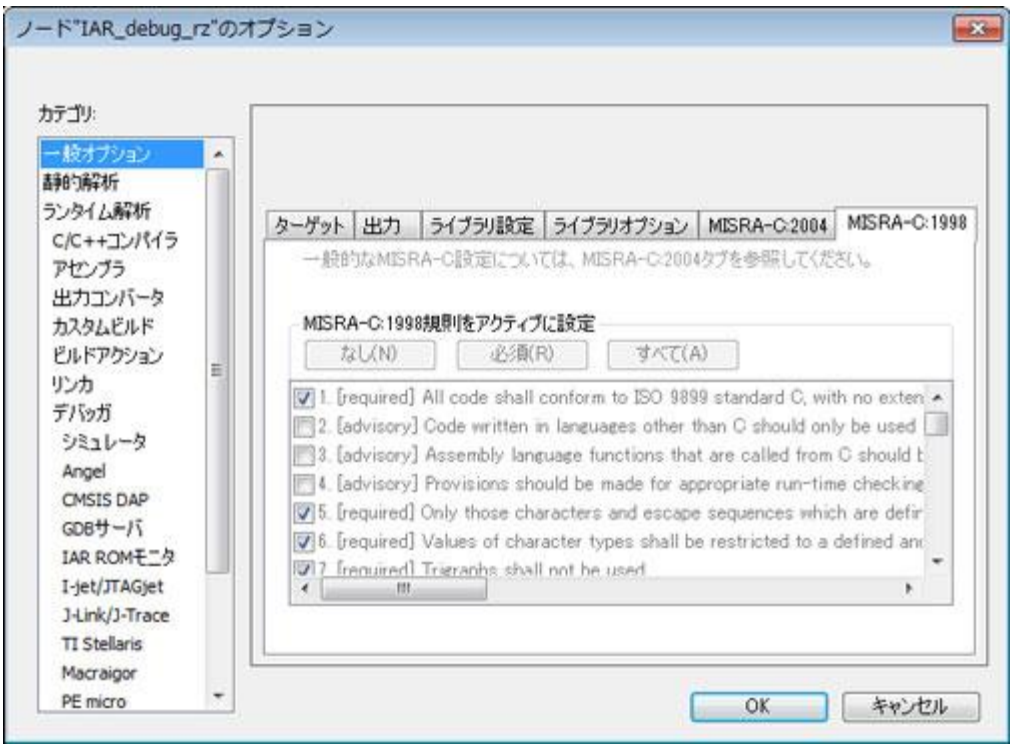
4）ライブラリオプション



5）MISRA-C:2004（設定なし）

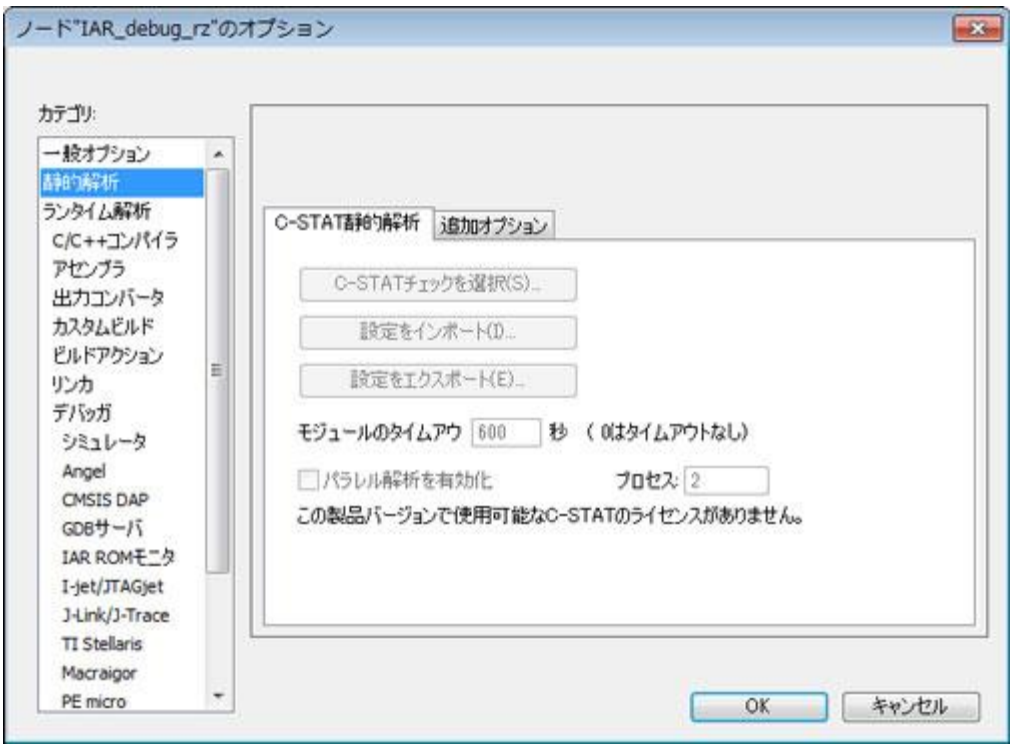


6）MISRA-C:1998（設定なし）

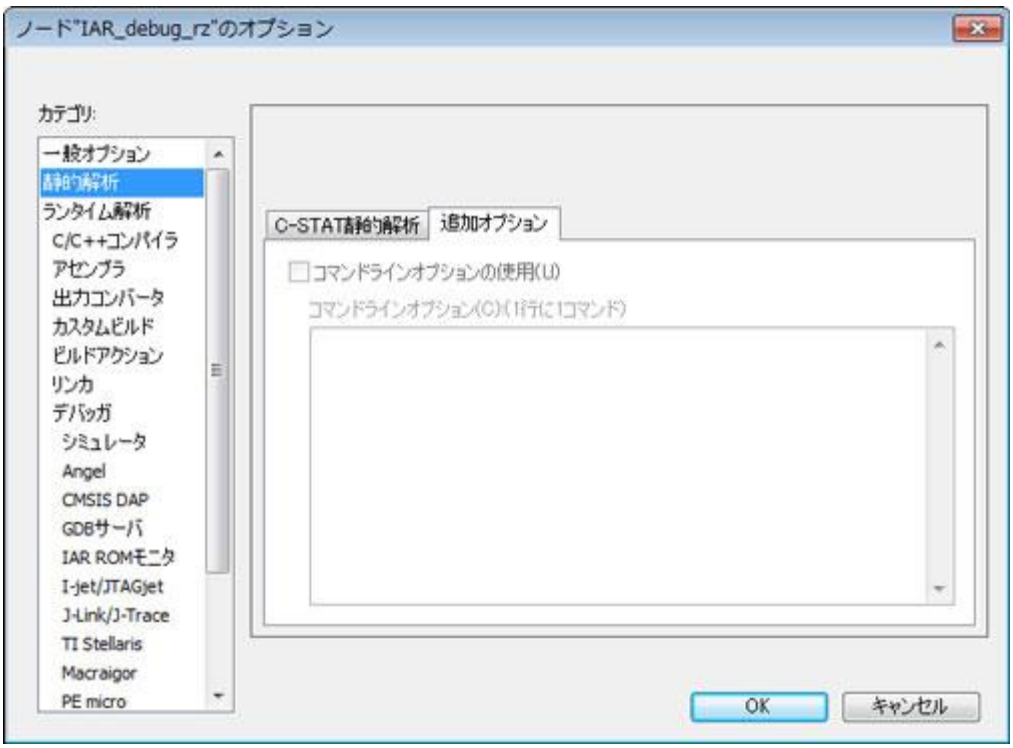


２－２．静的解析の確認

１）C-STAT 静的解析（設定なし）

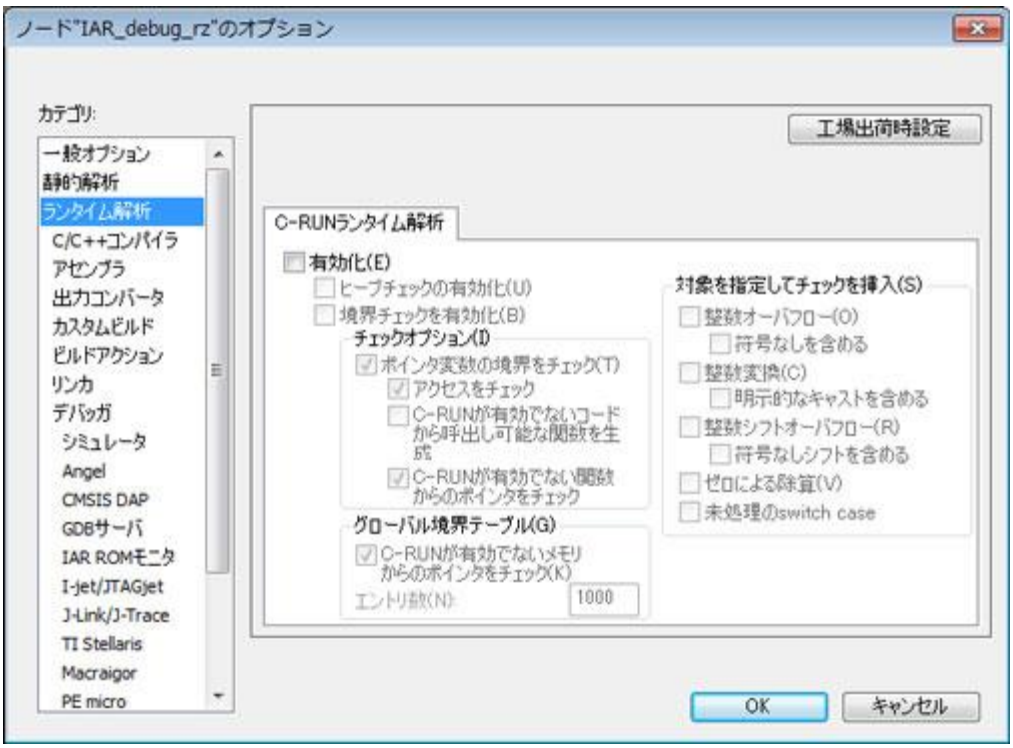


２）追加オプション（設定なし）



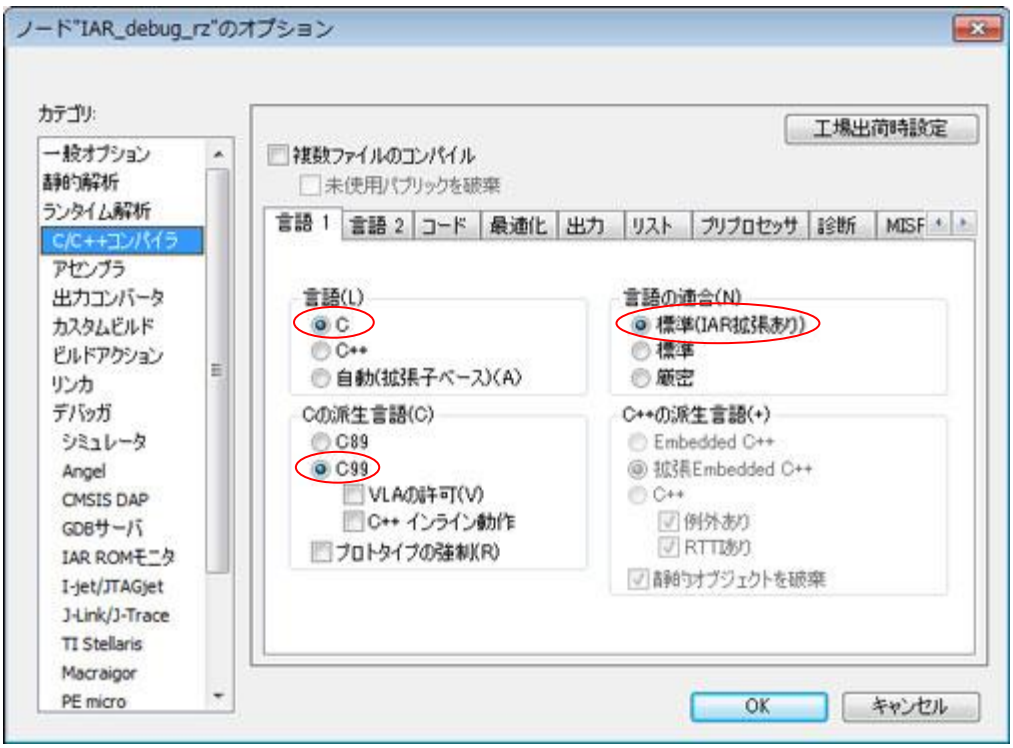
２－３．ランタイム解析の確認

１）C-RUN ランタイム解析（設定なし）

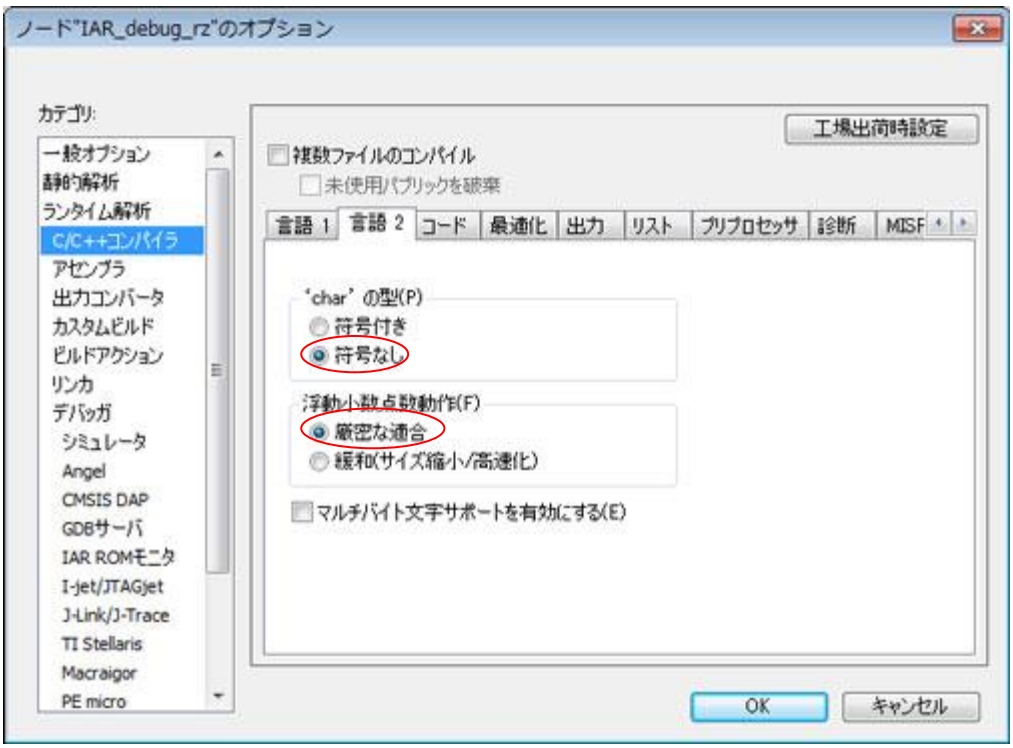


２－４．C/C++コンパイラの確認

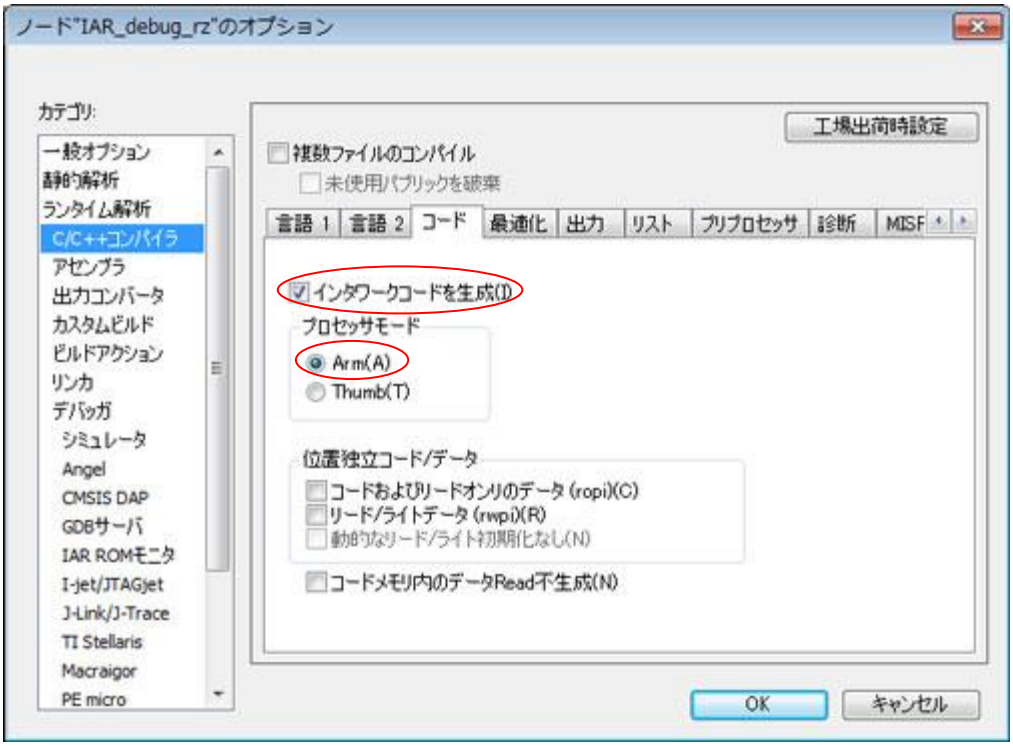
１）言語１



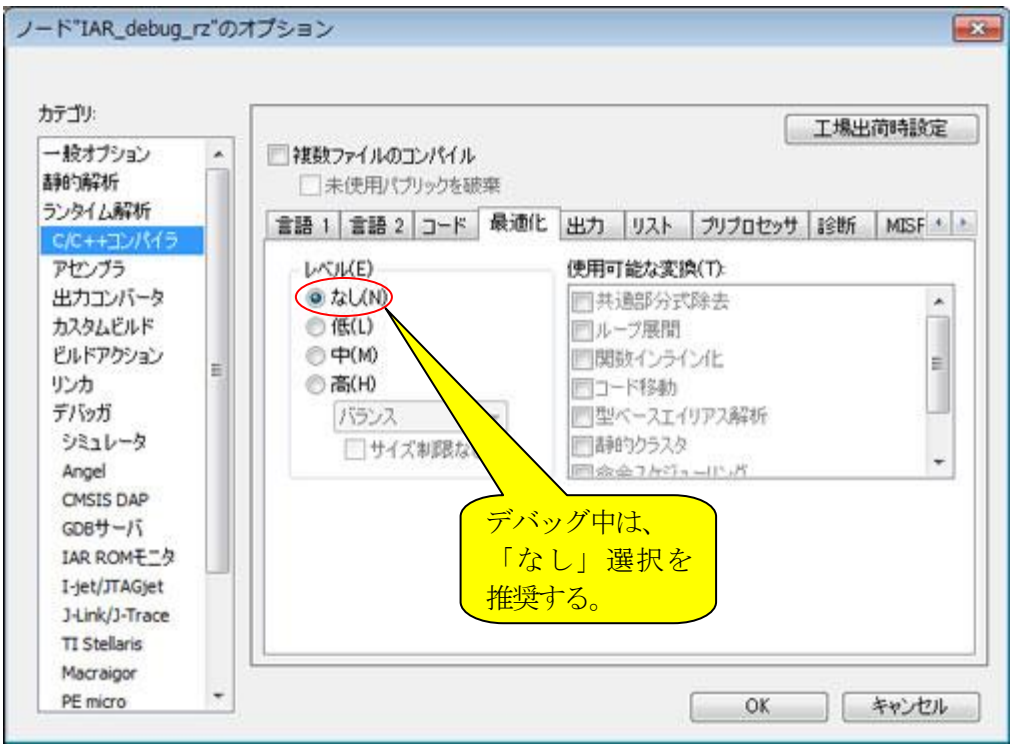
２）言語２



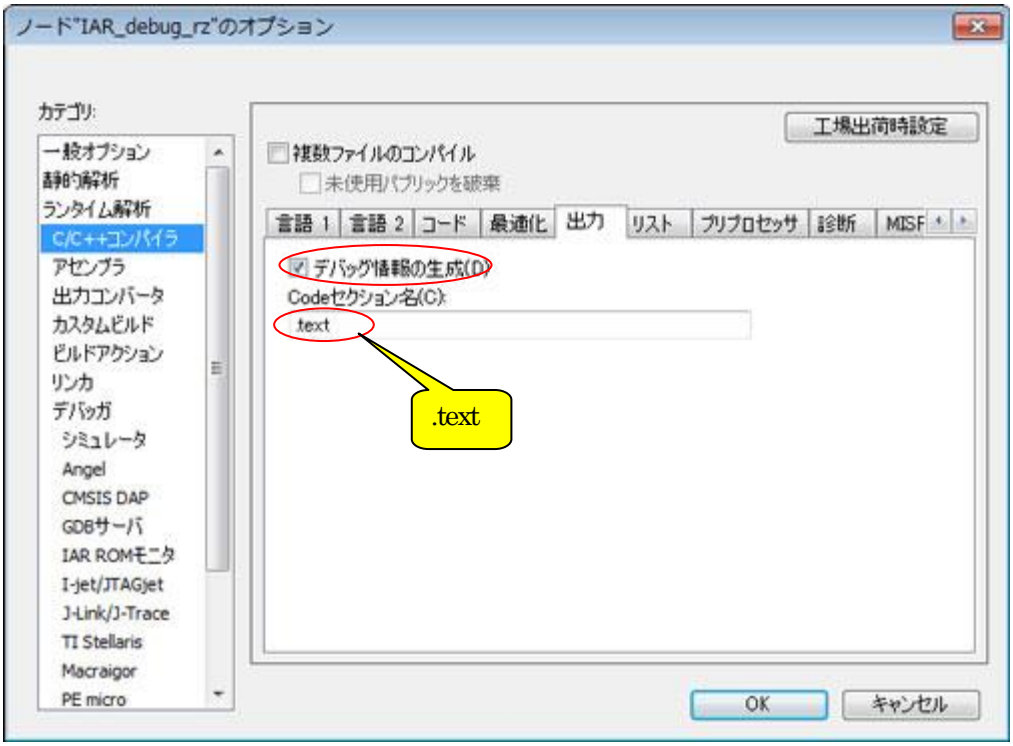
3）コード


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
複数ファイルのコンパイル
未使用パブリックを破棄
言語 1
言語 2
コード
最適化
出力
リスト
プリプロセッサ
診断
MISF
インタワークコードを生成(I)
プロセッサモード
Arm(A)
Thumb(T)
位置独立コード/データ
コードおよびリードオンリのデータ (ropi)(C)
リード/ライトデータ (rwpi)(R)
動的なリード/ライト初期化なし(N)
コードメモリ内のデータRead不生成(N)
OK
キャンセル

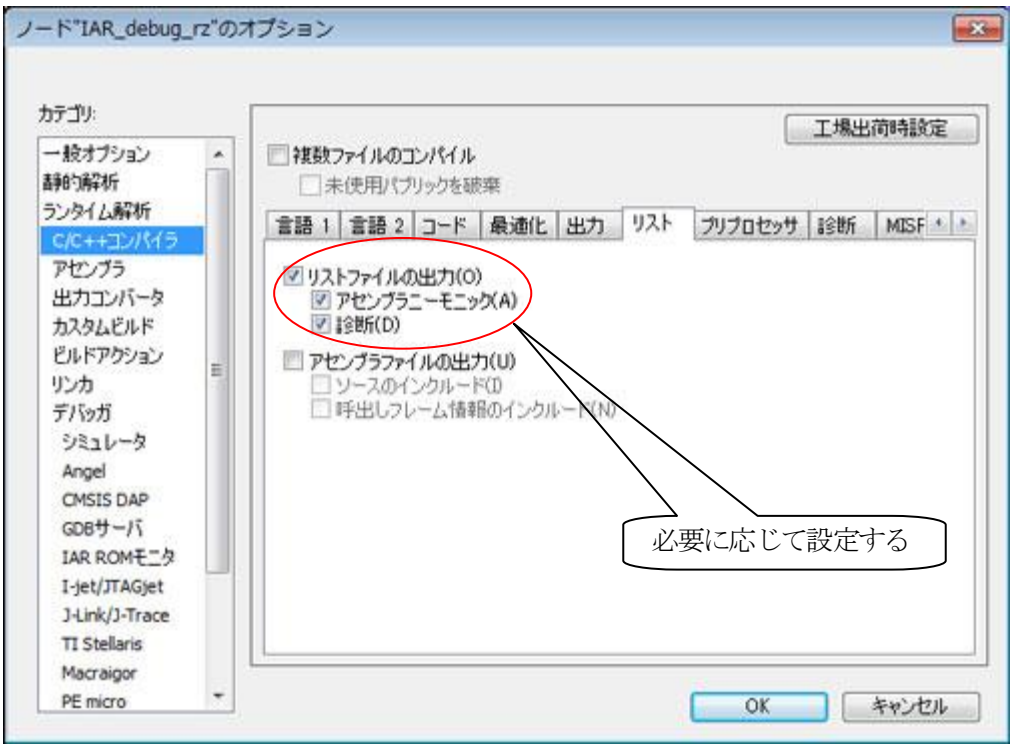
4）最適化


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
複数ファイルのコンパイル
未使用パブリックを破棄
言語 1
言語 2
コード
最適化
出力
リスト
プリプロセッサ
診断
MISF
レベル(E)
なし(N)
低(L)
中(M)
高(H)
バランス
使用可能な変換(T):
共通部分式除去
ループ展開
関数インライン化
コード移動
型ベースエイリアス解析
静的クラスタ
デバッグ中は、
「なし」選択を
推奨する。
OK
キャンセル

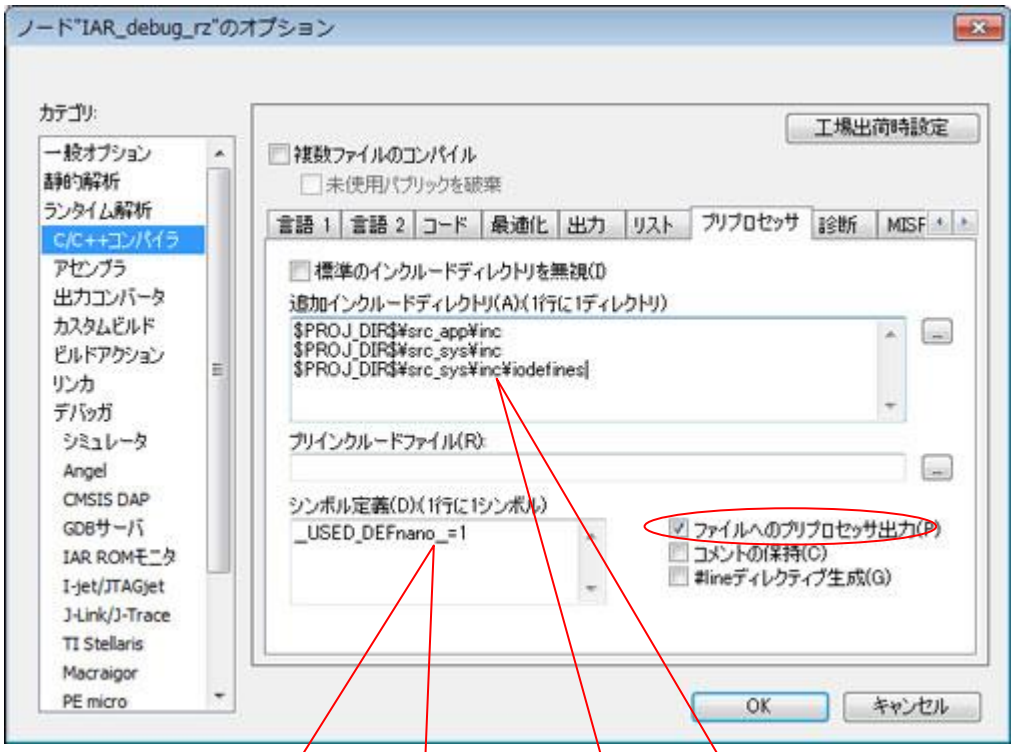
5）出力


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
複数ファイルのコンパイル
未使用パブリックを破棄
言語 1
言語 2
コード
最適化
出力
リスト
プリプロセッサ
診断
MISF
デバッグ情報の生成(D)
Codeセクション名(C):
.text
.text
OK
キャンセル

6）リスト


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
複数ファイルのコンパイル
未使用パブリックを破棄
言語 1
言語 2
コード
最適化
出力
リスト
プリプロセッサ
診断
MISF
リストファイルの出力(O)
アセンブラニーモニック(A)
診断(D)
アセンブラファイルの出力(U)
ソースのインクルード(I)
呼出しフレーム情報のインクルード(N)
必要に応じて設定する
OK
キャンセル

7）プリプロセッサ

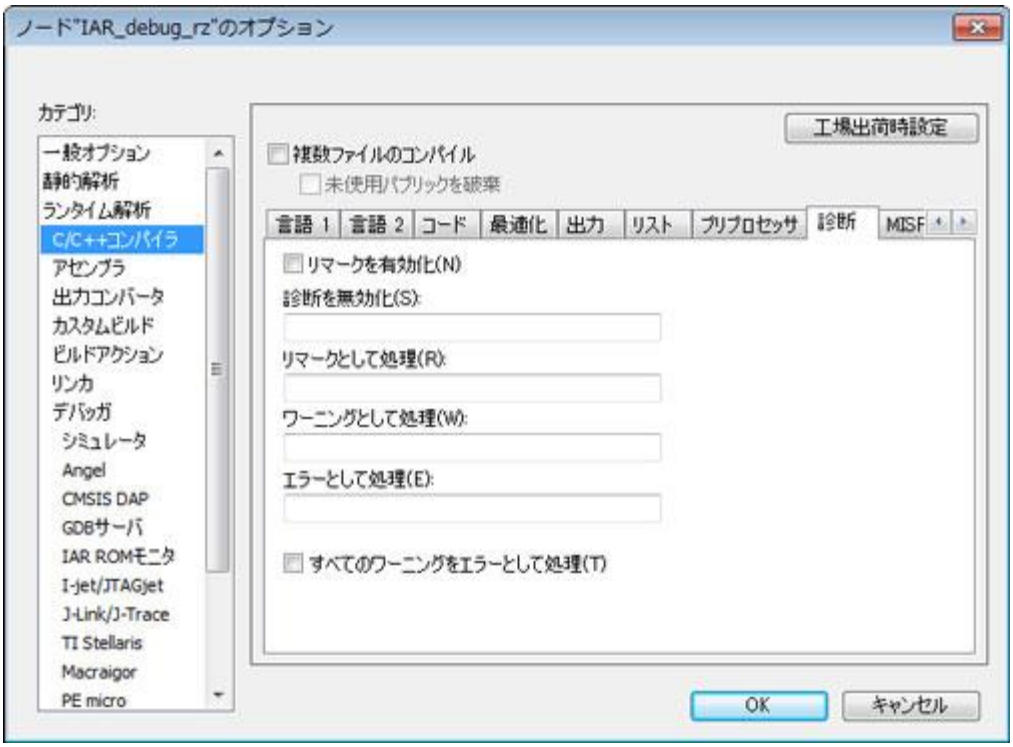


注＊１
DEFnano 使用／未使用の定義
「使用　」__USED_DEFnano__=1
「未使用」__USED_DEFnano__=0
デバッグ環境によって設定する。

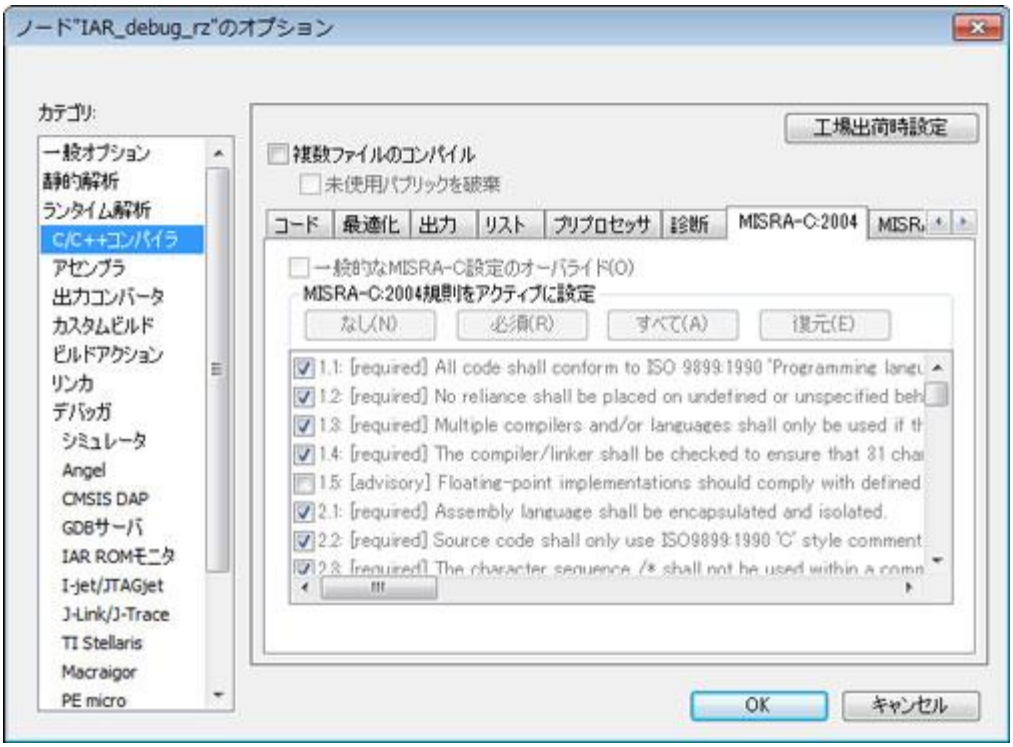
$PROJ_DIR$¥src_app¥inc
$PROJ_DIR$¥src_sys¥inc
$PROJ_DIR$¥src_sys¥inc¥iodefines

注＊１
「__USED_DEFnano__=0」と使用しない側に定義しても内蔵 RAM へのダウンロードとシリアルフラッシュ ROM への書き込み操作は可能です。ただし、再操作する場合はターゲット側のリセット操作が必要になります。

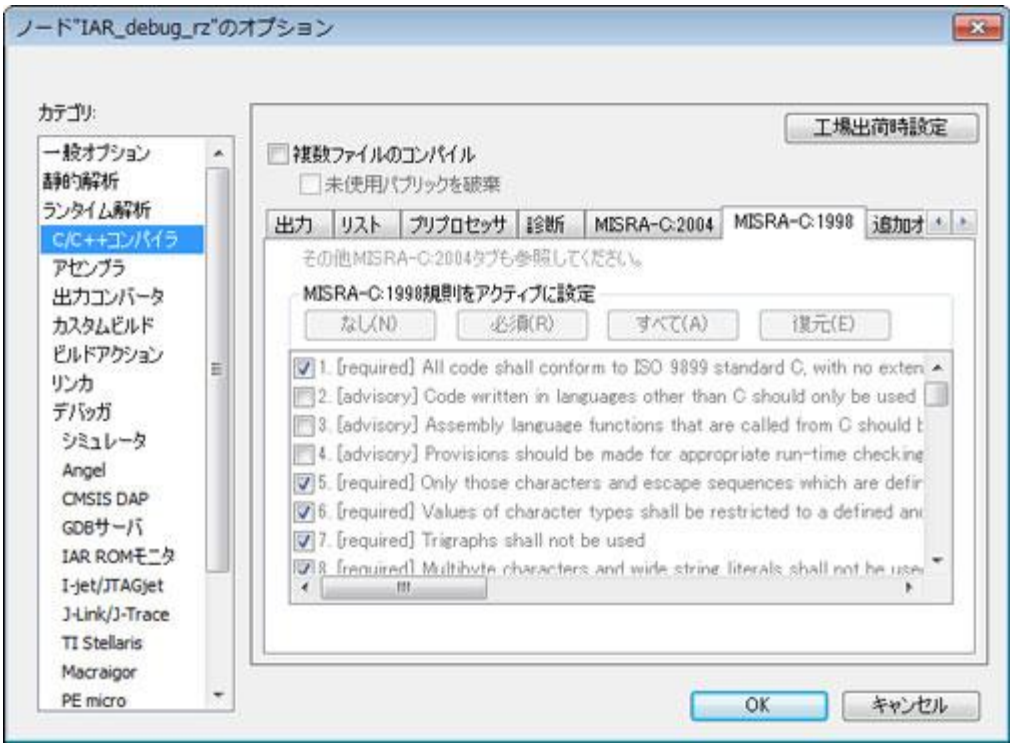
8）診断（設定なし）



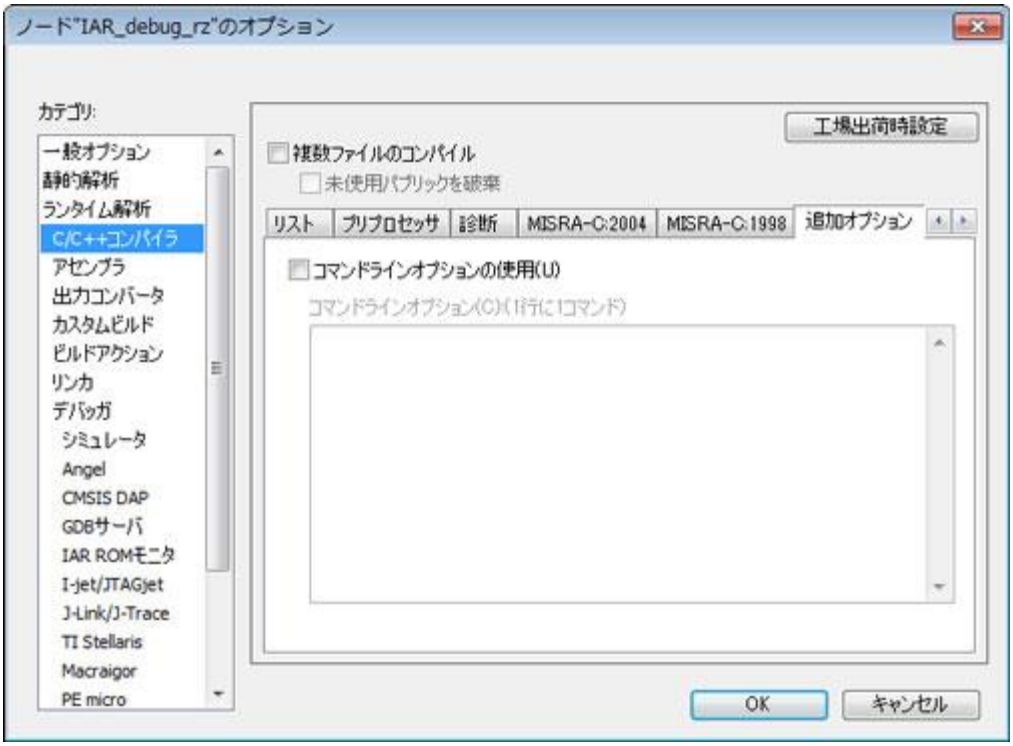
9）MISRA-C:2004（設定なし）



１０）MISRA-C:1998（設定なし）

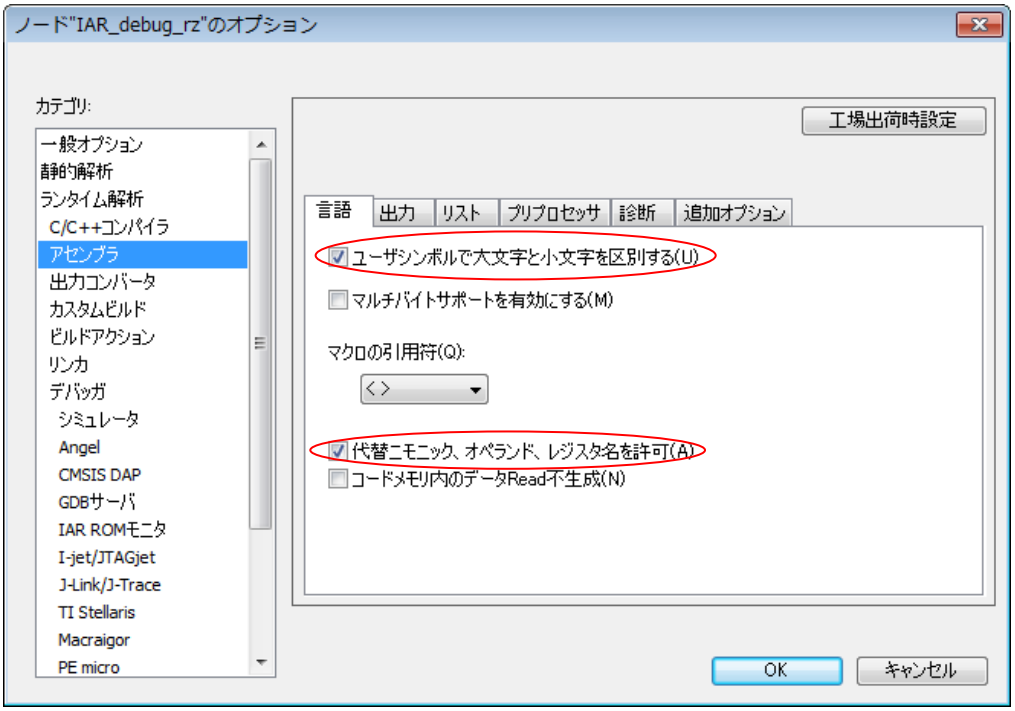


１１）追加オプション（設定なし）

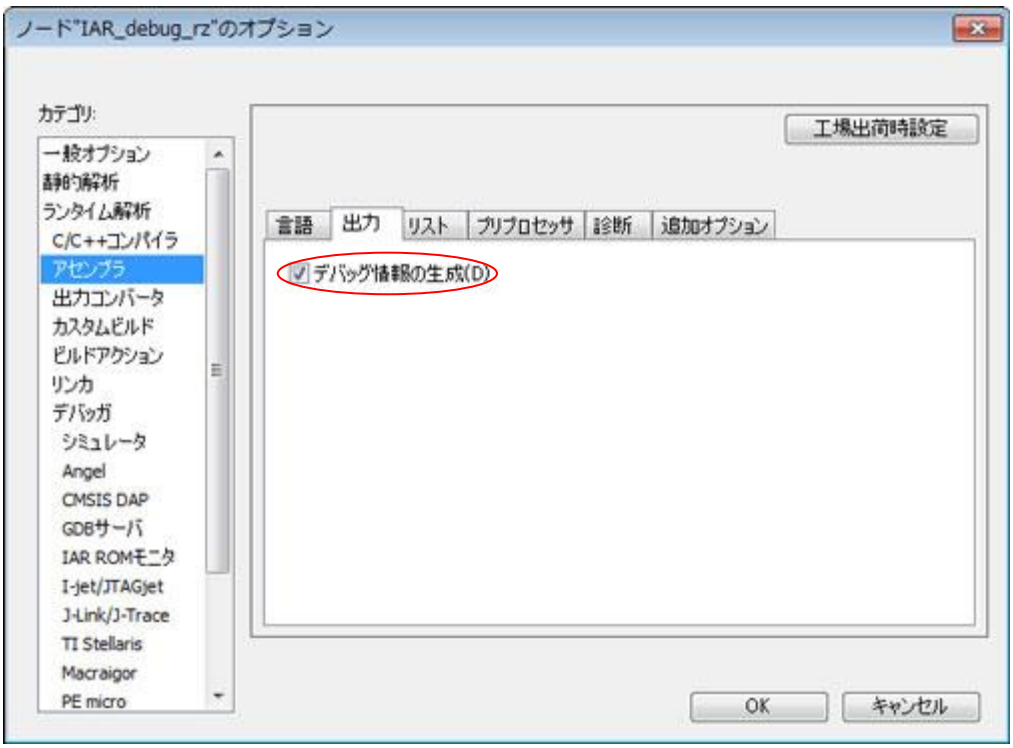


２－５．アセンブラの確認

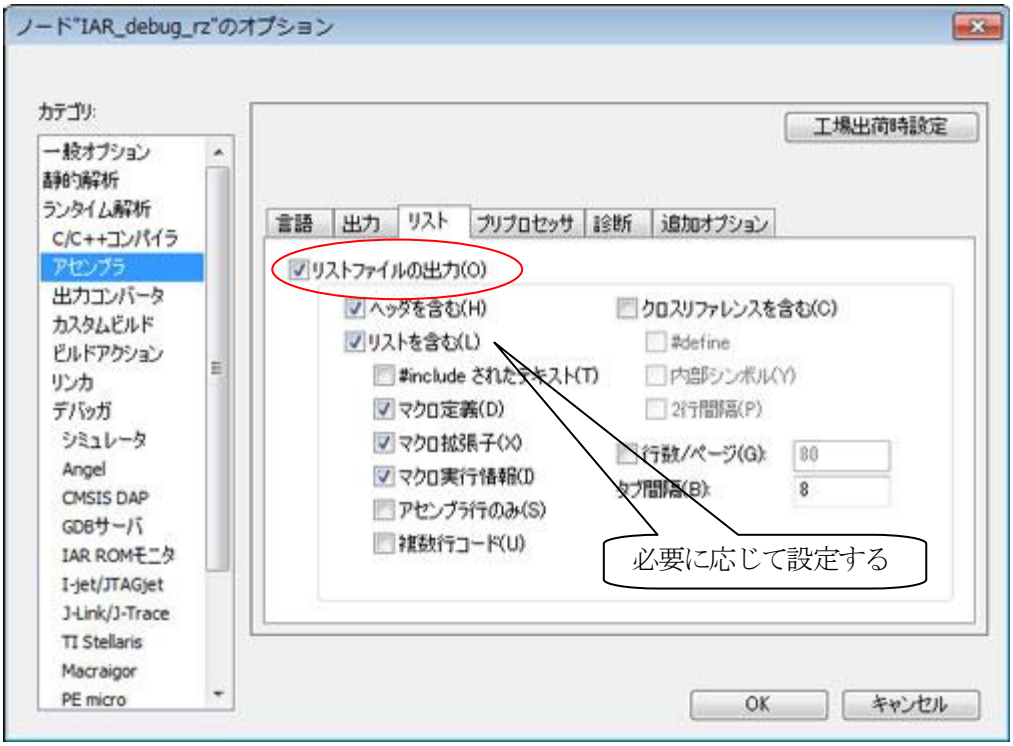
１）言語



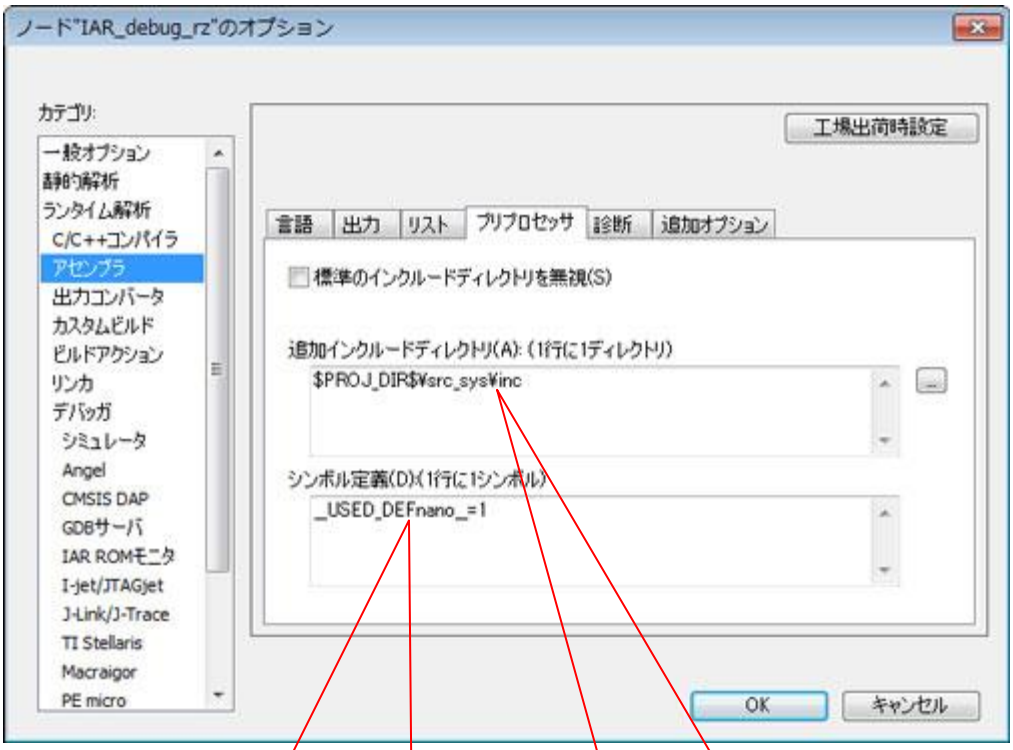
２）出力



3）リスト


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
言語
出力
リスト
プリプロセッサ
診断
追加オプション
リストファイルの出力(O)
ヘッダを含む(H)
リストを含む(L)
#include されたテキスト(T)
マクロ定義(D)
マクロ拡張子(X)
マクロ実行情報(I)
アセンブラ行のみ(S)
複数行コード(U)
クロスリファレンスを含む(C)
#define
内部シンボル(Y)
2行間隔(P)
行数/ページ(G):
80
タブ間隔(B):
8
必要に応じて設定する
OK
キャンセル

4）プリプロセッサ

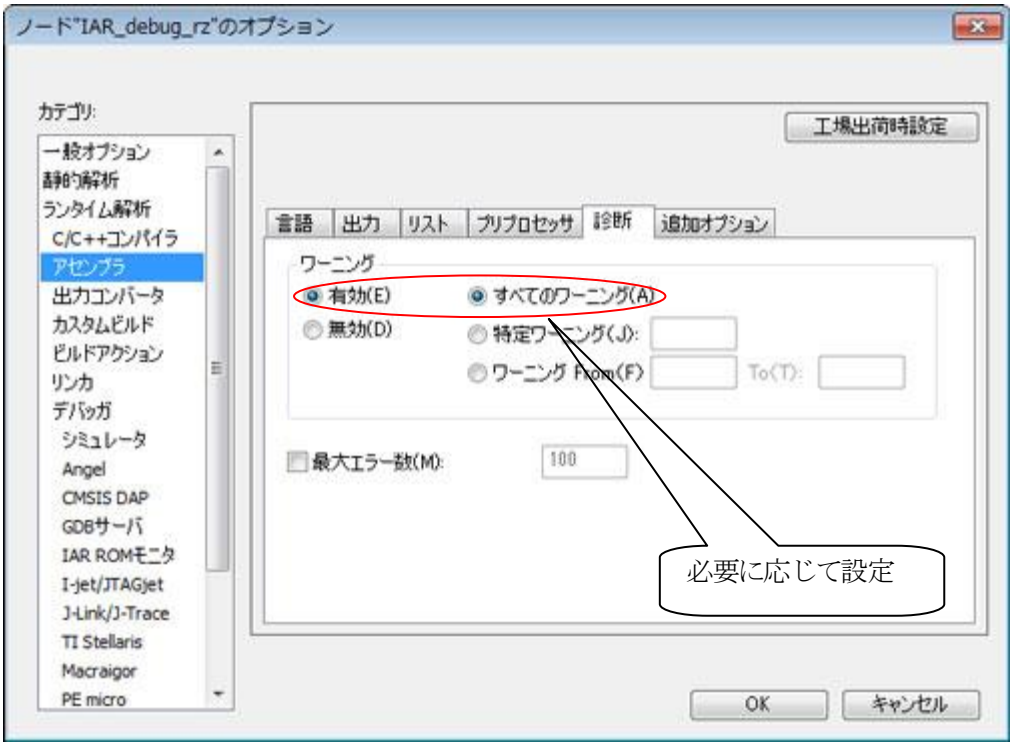


注＊1
DEFnano 使用／未使用の定義
「使用　」__USED_DEFnano__=1
「未使用」__USED_DEFnano__=0
デバッグ環境によって設定する。

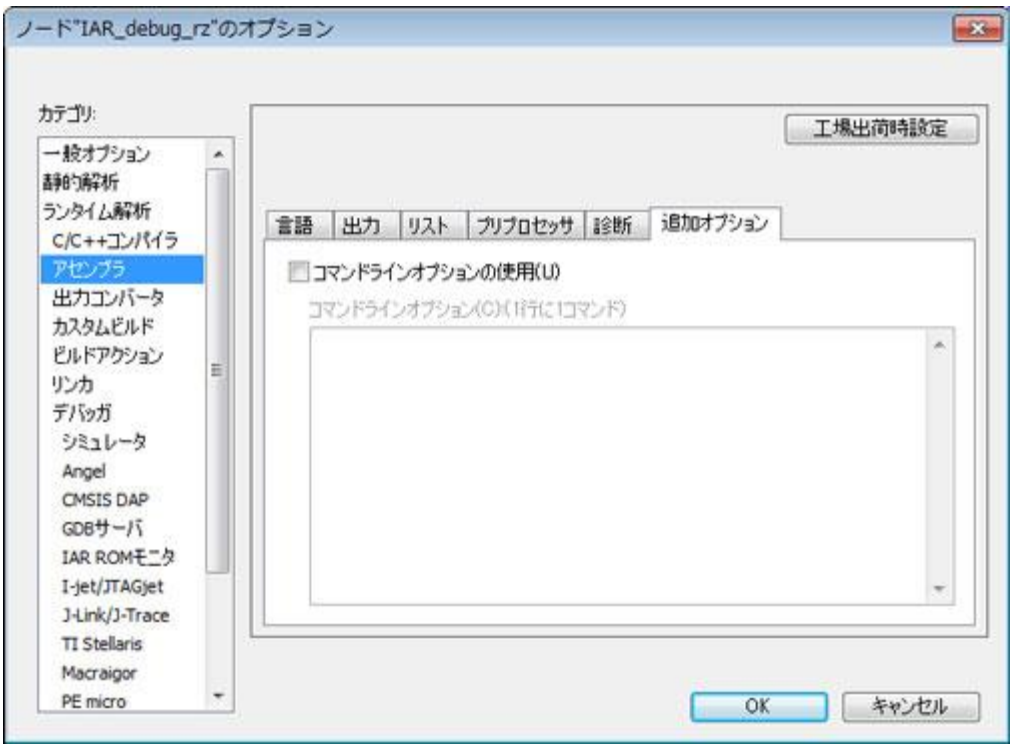
$PROJ_DIR$¥src_sys¥inc

注＊1
「__USED_DEFnano__=0」と使用しない側に定義しても内蔵 RAM へのダウンロードとシリアルフラッシュ ROM への書き込み操作は可能です。ただし、再操作する場合はターゲット側のリセット操作が必要になります。

5）診断

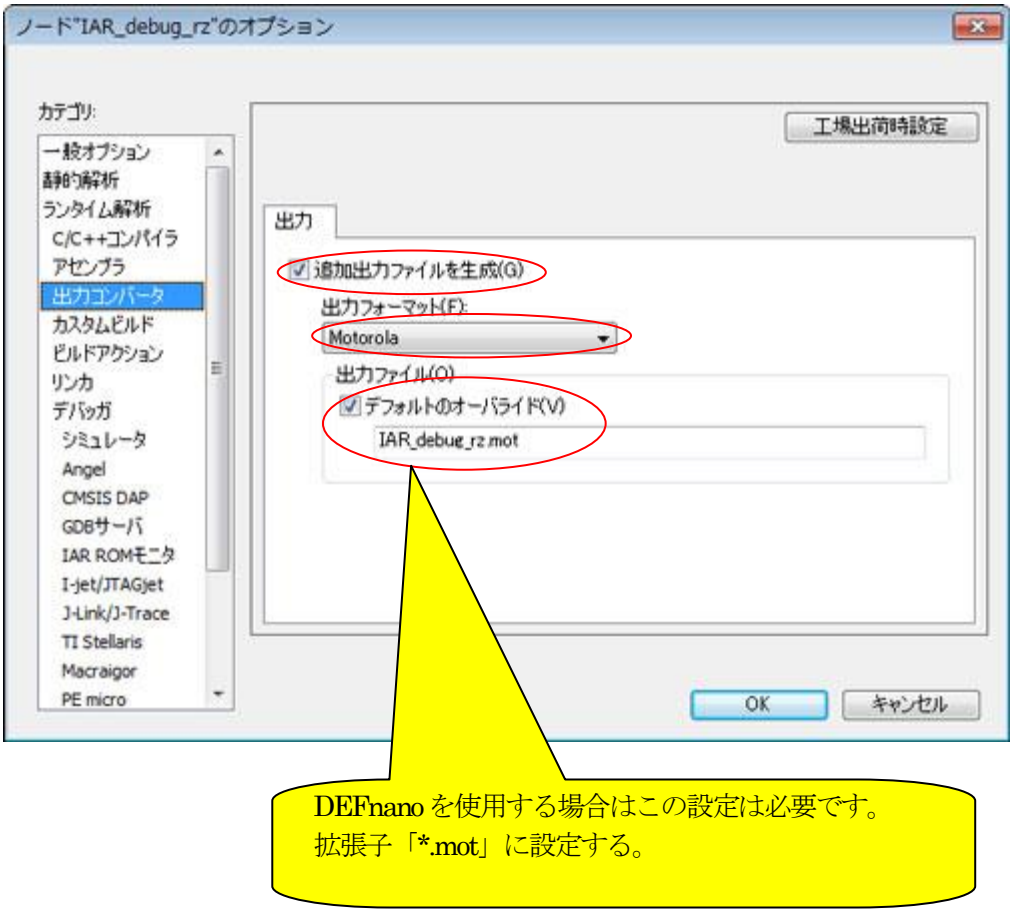


6）追加オプション（設定なし）



２－６．出力コンバータの確認

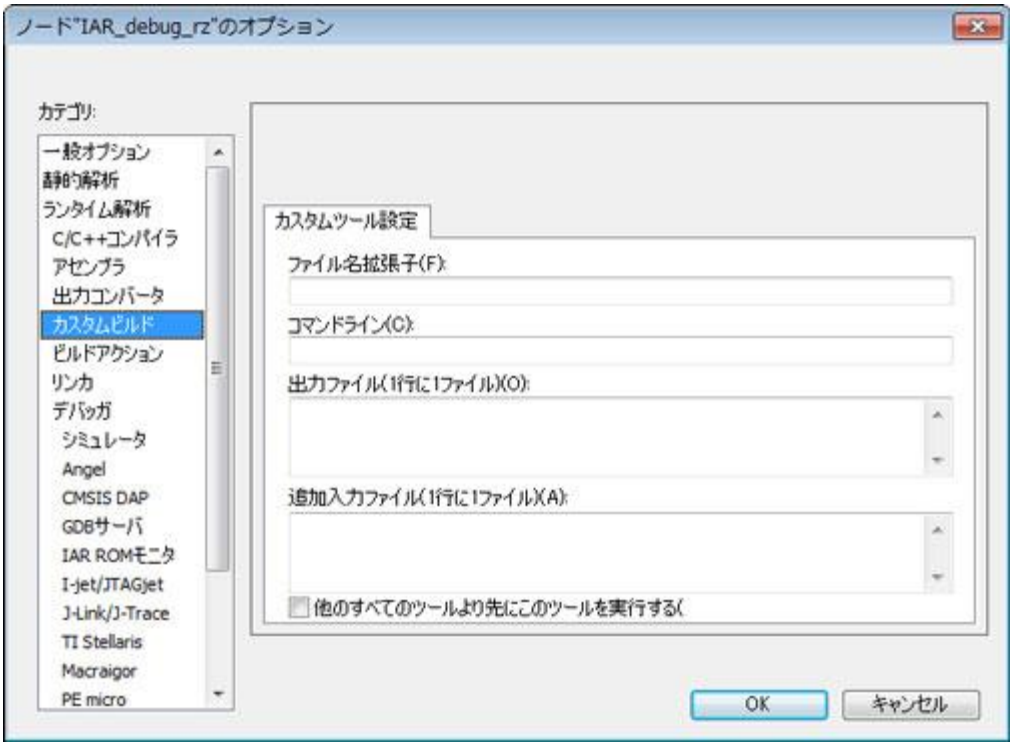
１）出力



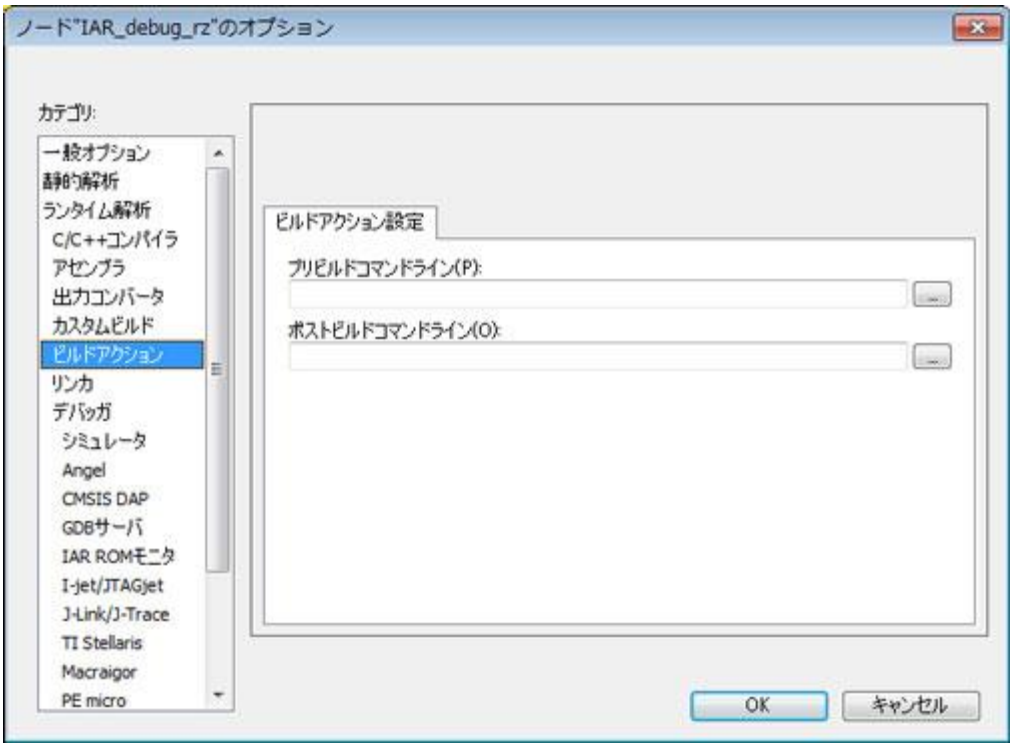
DEFnano を使用する場合はこの設定は必要です。
拡張子「*.mot」に設定する。

２－７．カスタムビルドの確認（設定なし）

１）カスタムツール設定

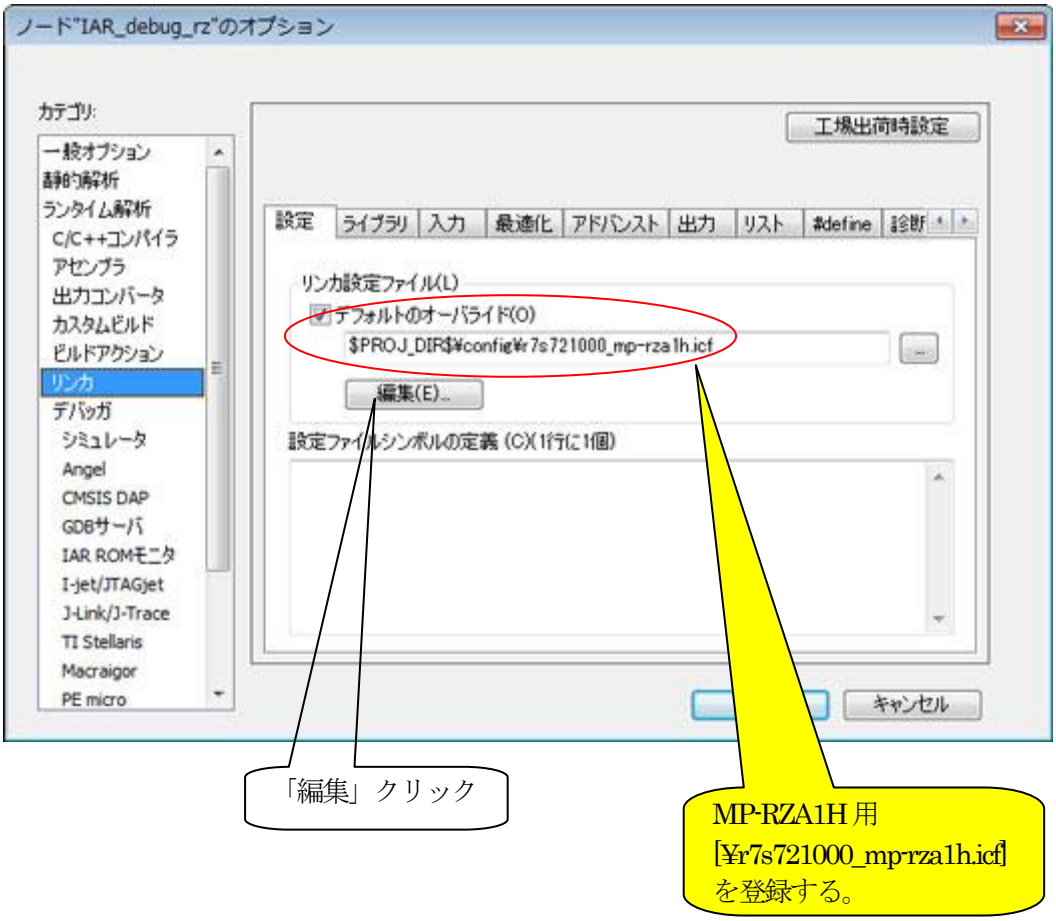


２－８．ビルドアクションの確認（設定なし）

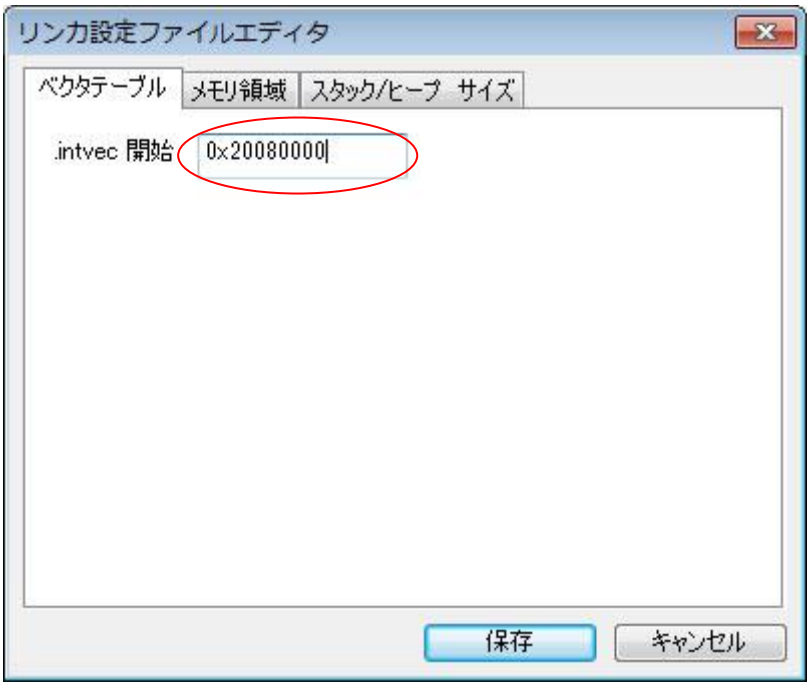


２－９．リンカの確認

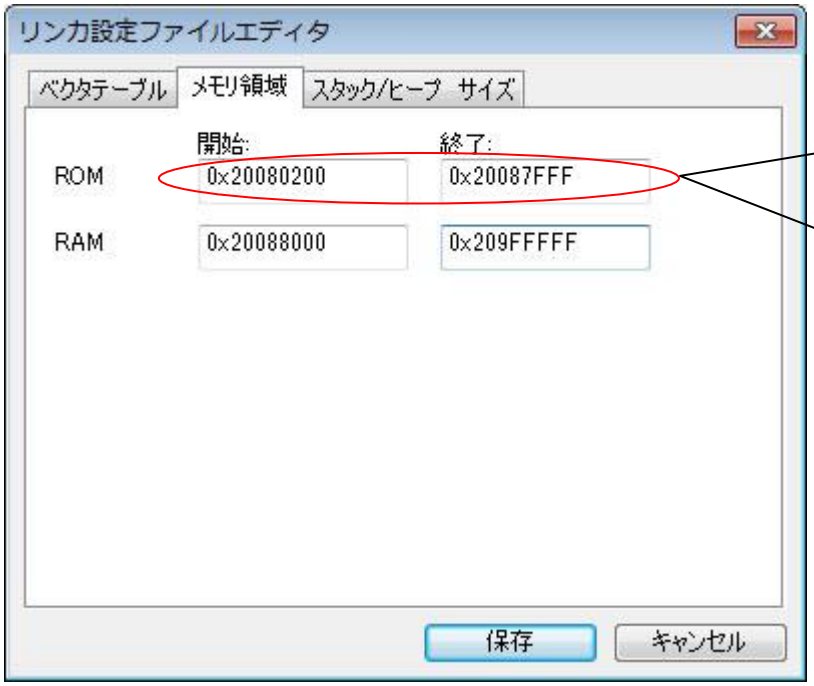
１）設定



１－１）ベクタテーブル

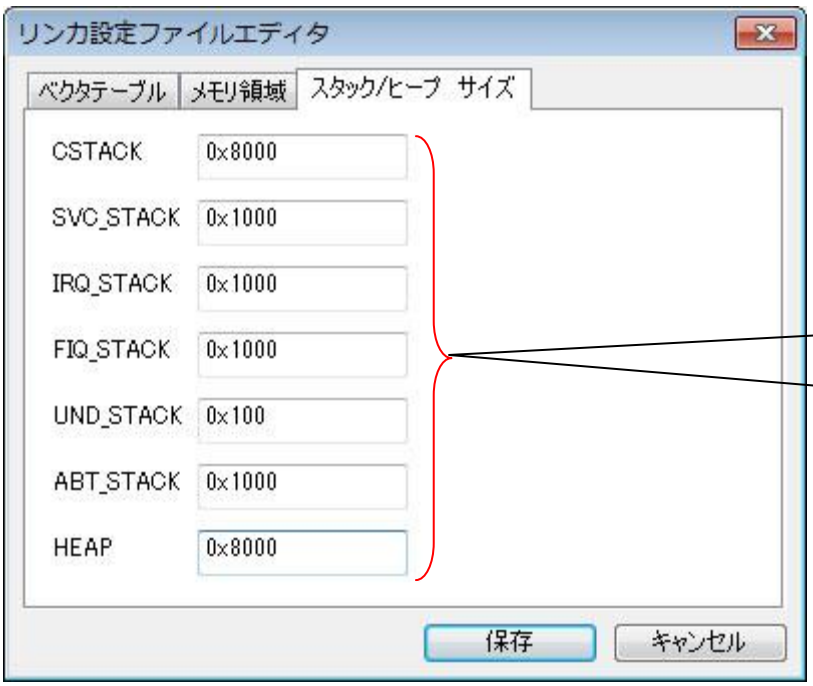


1－2）メモリ領域



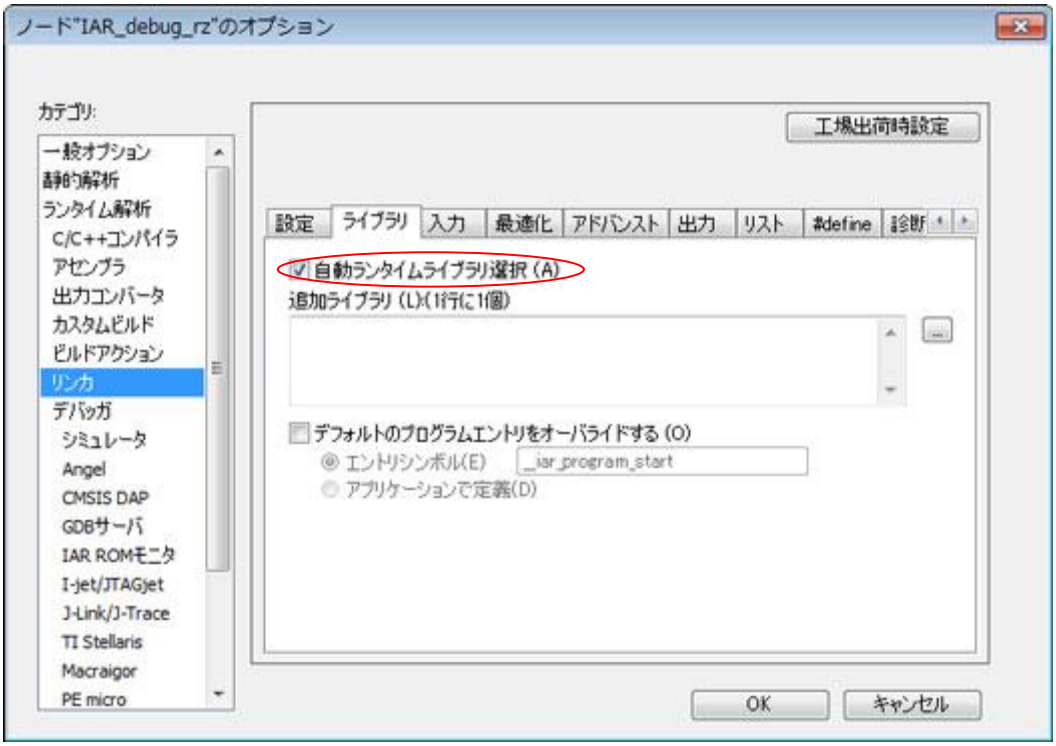
評価版のため ROM サイズ MAX(0x8000)までの設定とする。
残りエリア全てを RAM 側に割り付ける。

1－3）スタック／ヒープサイズ

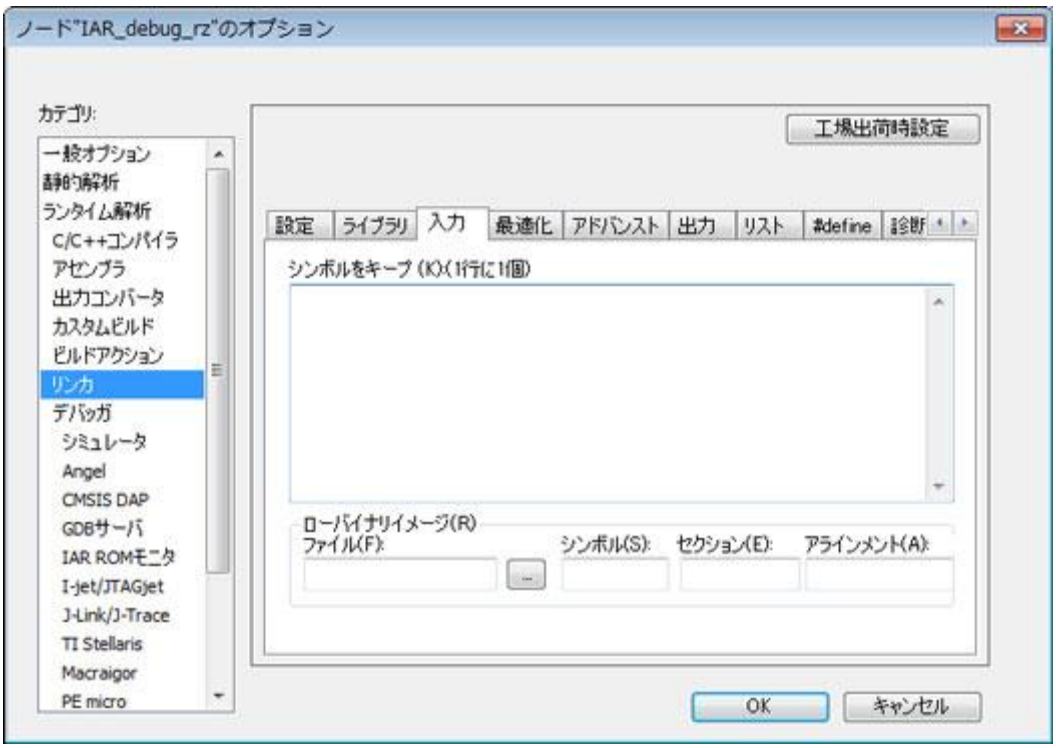


アプリケーションの規模に応じて設定する。

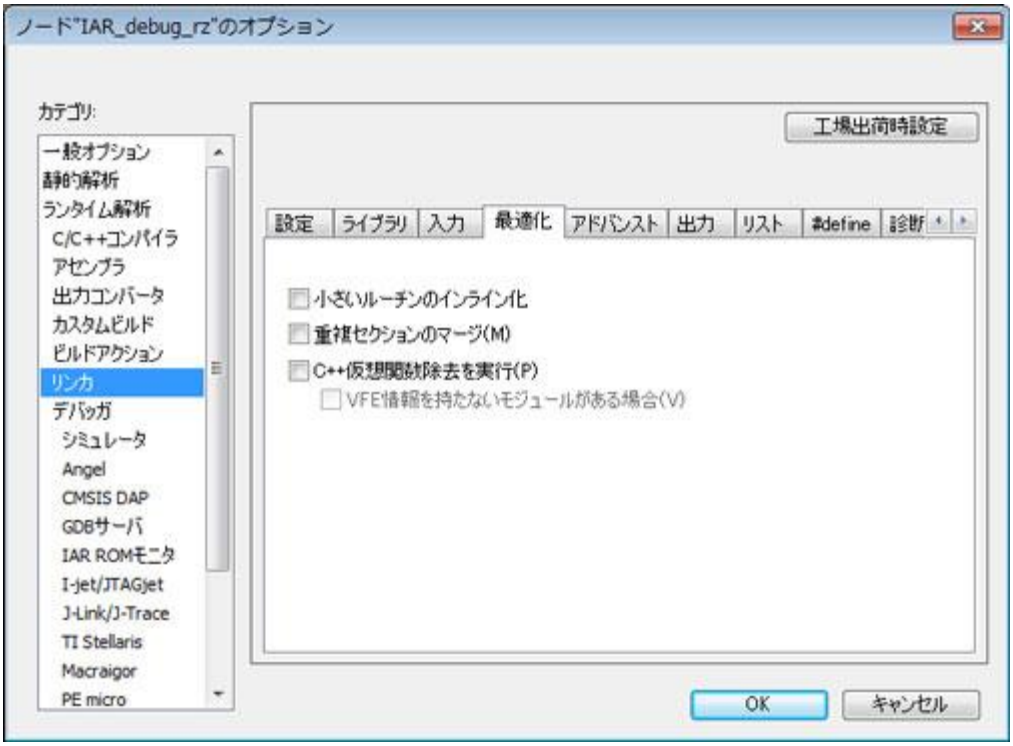
2）ライブラリ



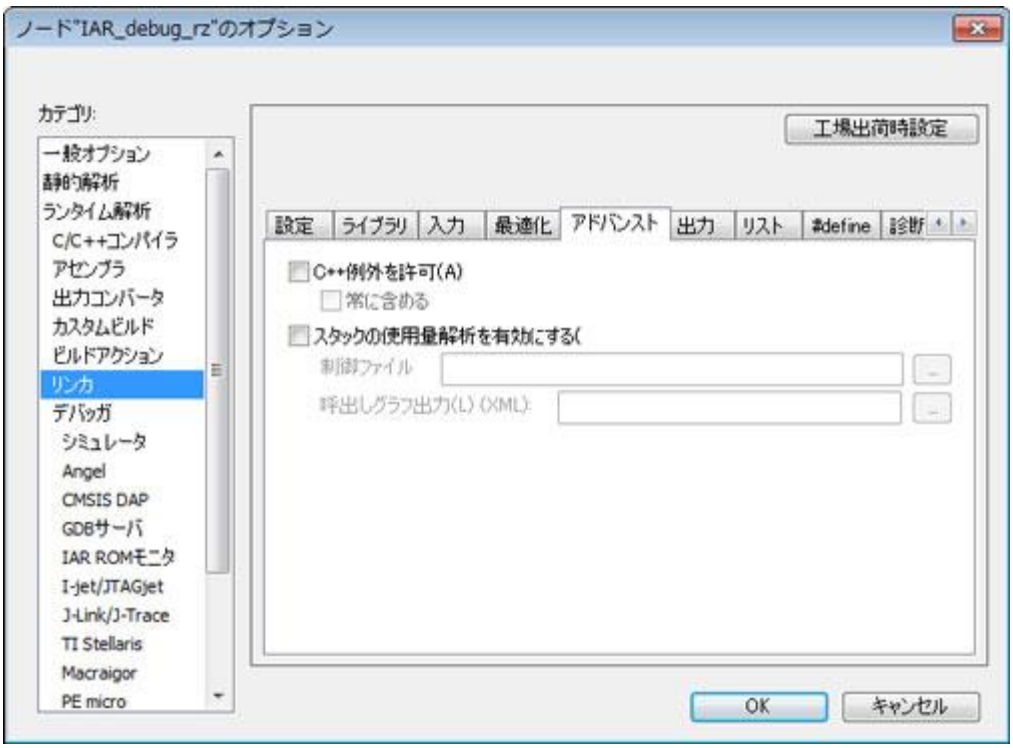
3）入力（設定なし）



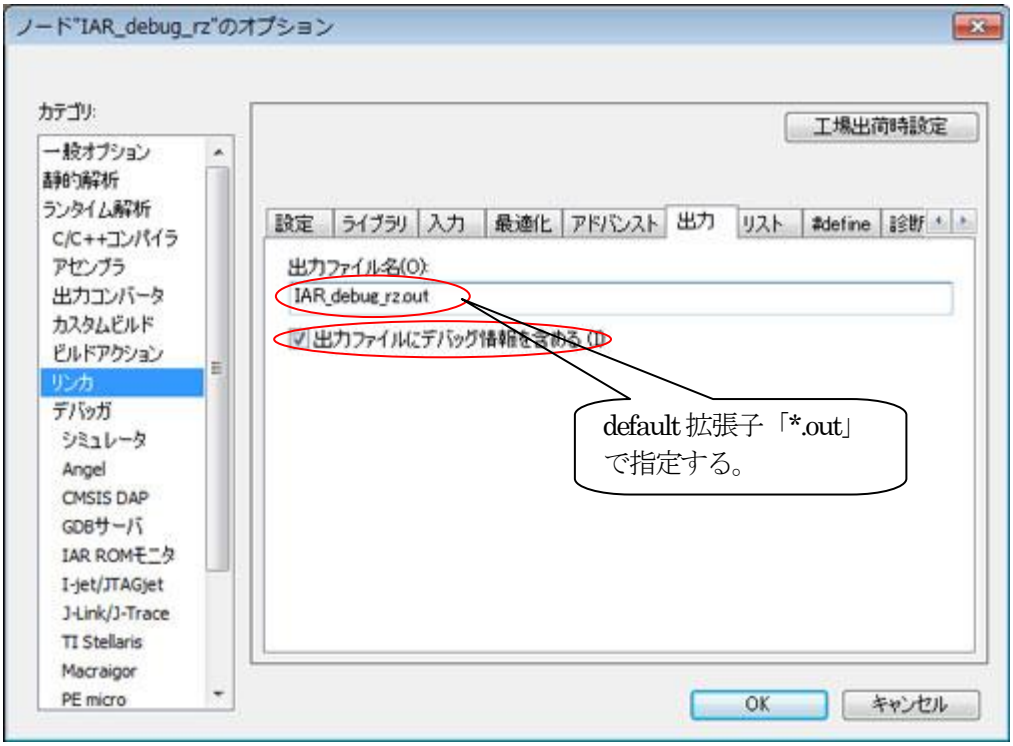
4）最適化（設定なし）



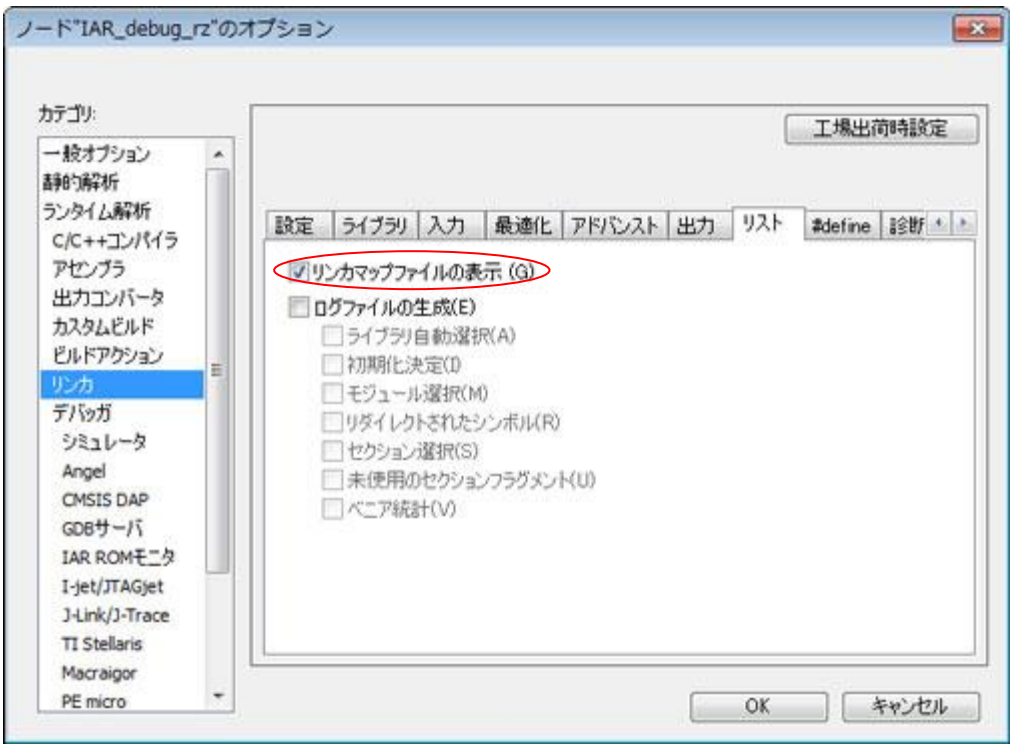
5）アドバンスト（設定なし）



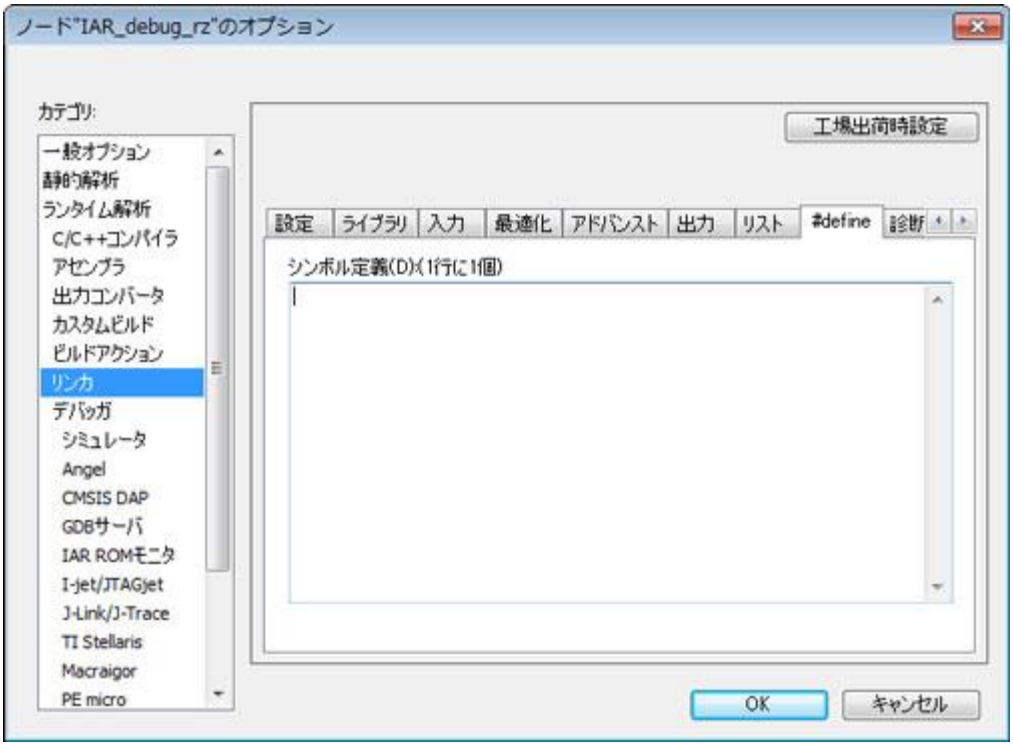
6）出力


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
設定
ライブラリ
入力
最適化
アドバンスト
出力
リスト
#define
診断
出力ファイル名(O):
IAR_debug_rz.out
出力ファイルにデバッグ情報を含める (I)
default 拡張子「*.out」
で指定する。
OK
キャンセル

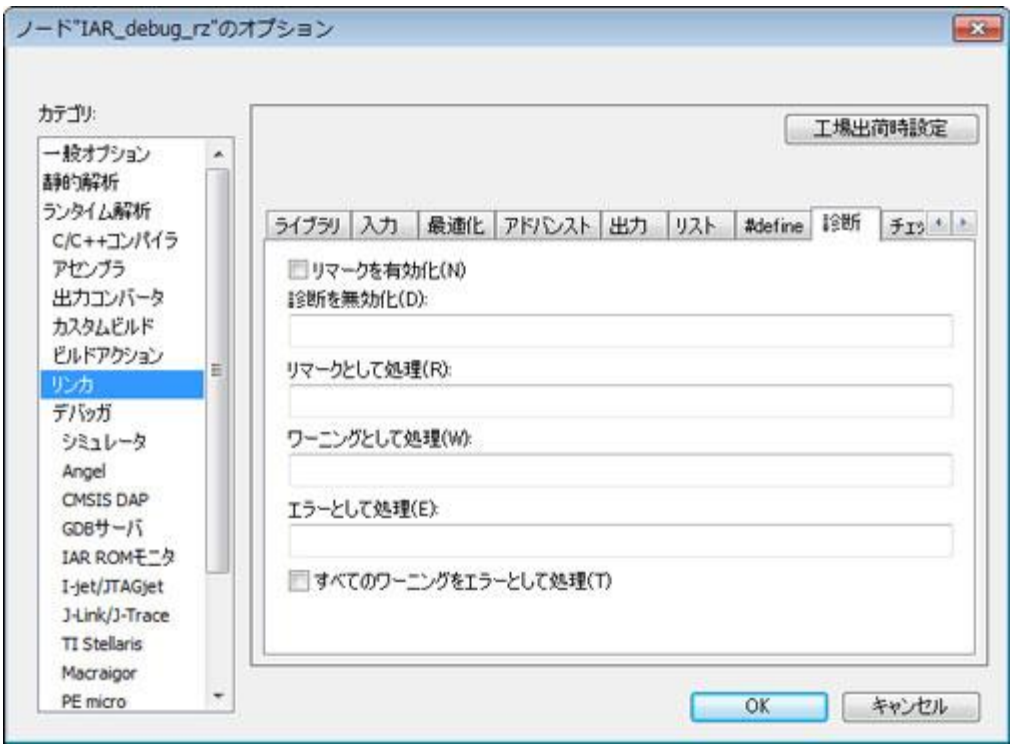
7）リスト


ノード"IAR_debug_rz"のオプション
カテゴリ:
一般オプション
静的解析
ランタイム解析
C/C++コンパイラ
アセンブラ
出力コンバータ
カスタムビルド
ビルドアクション
リンカ
デバッガ
シミュレータ
Angel
CMSIS DAP
GDBサーバ
IAR ROMモニタ
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Macraigor
PE micro
工場出荷時設定
設定
ライブラリ
入力
最適化
アドバンスト
出力
リスト
#define
診断
リンカマップファイルの表示 (G)
ログファイルの生成(E)
ライブラリ自動選択(A)
初期化決定(I)
モジュール選択(M)
リダイレクトされたシンボル(R)
セクション選択(S)
未使用のセクションフラグメント(U)
ベニア統計(V)
OK
キャンセル

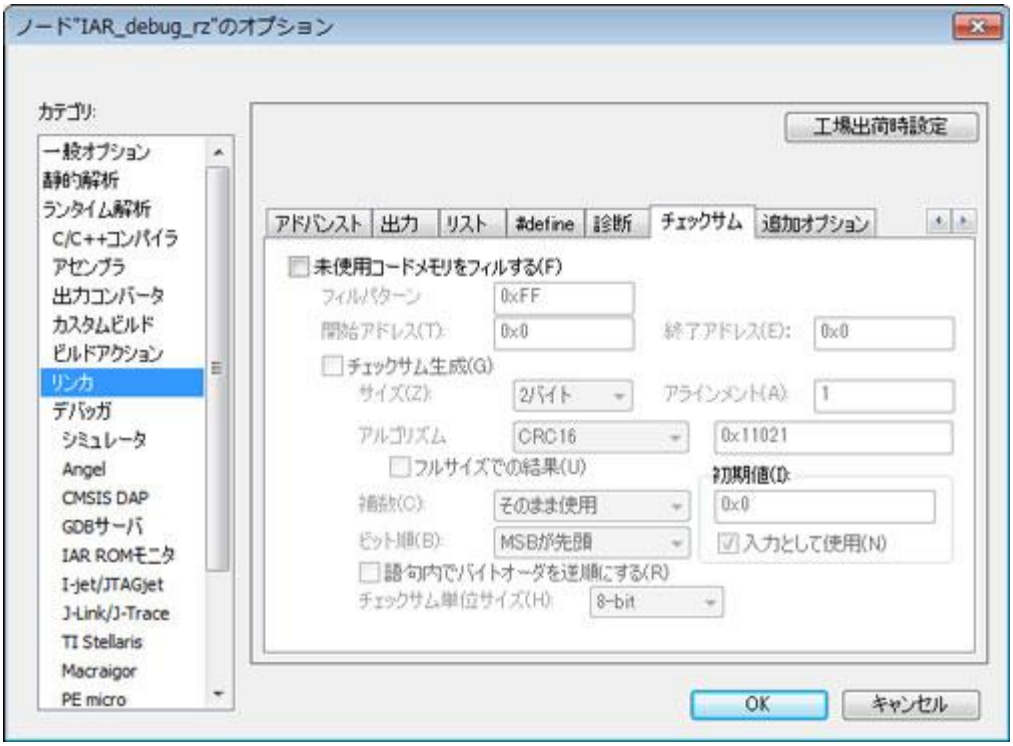
8）#define（設定なし）



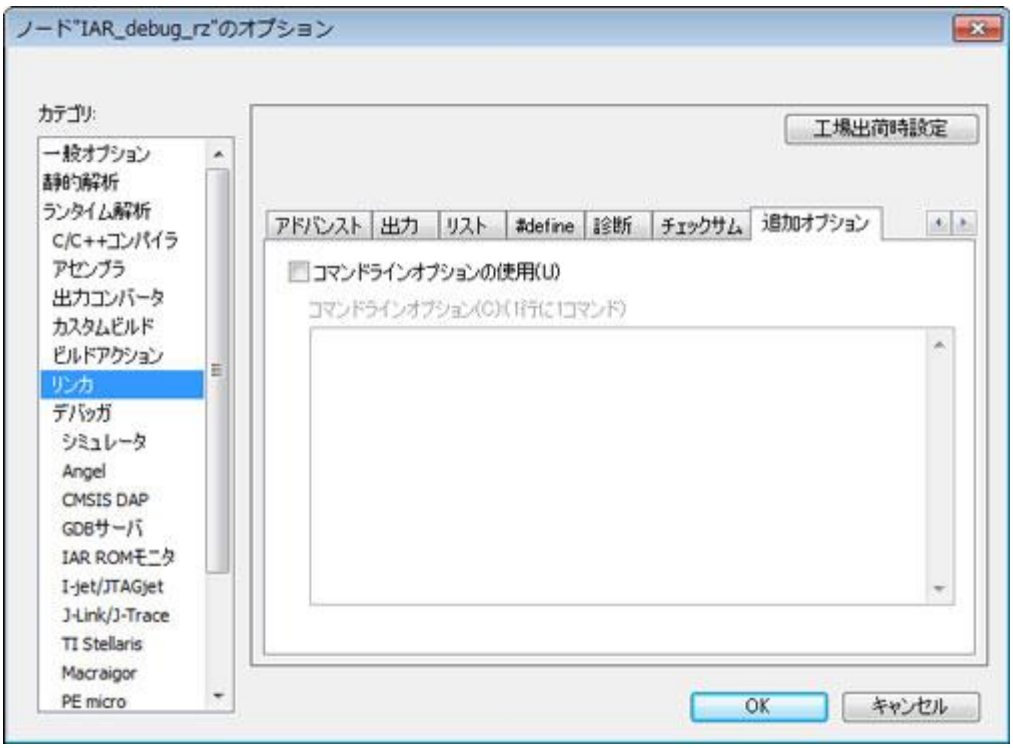
9）診断（設定なし）



１０）チェックサム（設定なし）

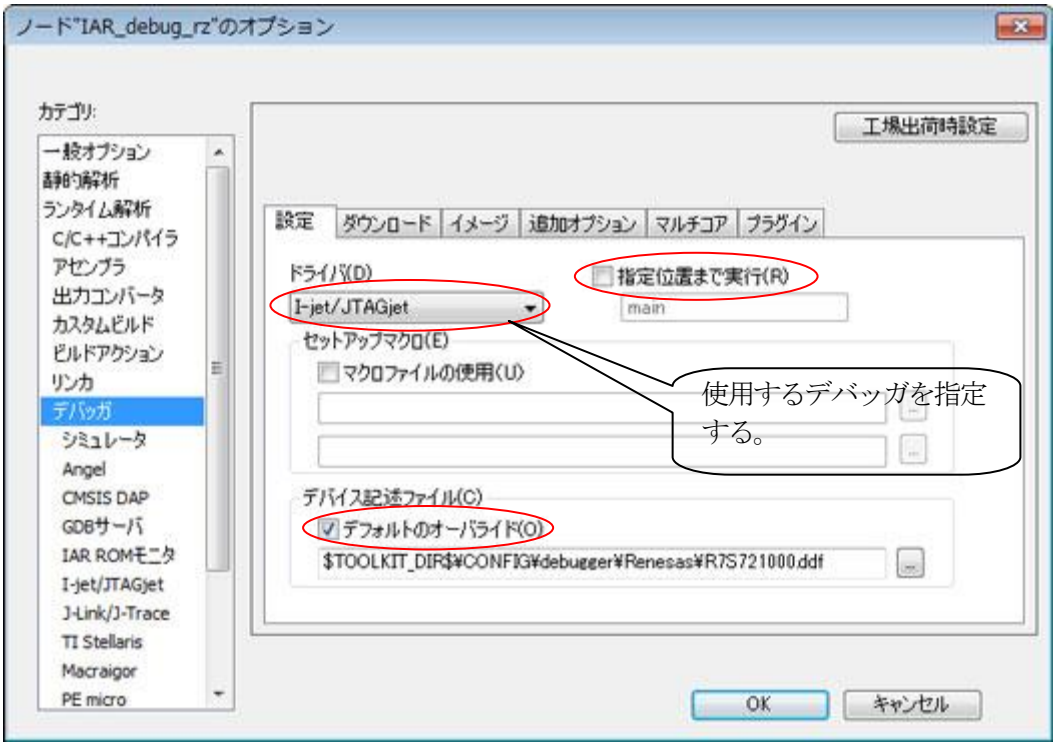


１１）追加オプション（設定なし）

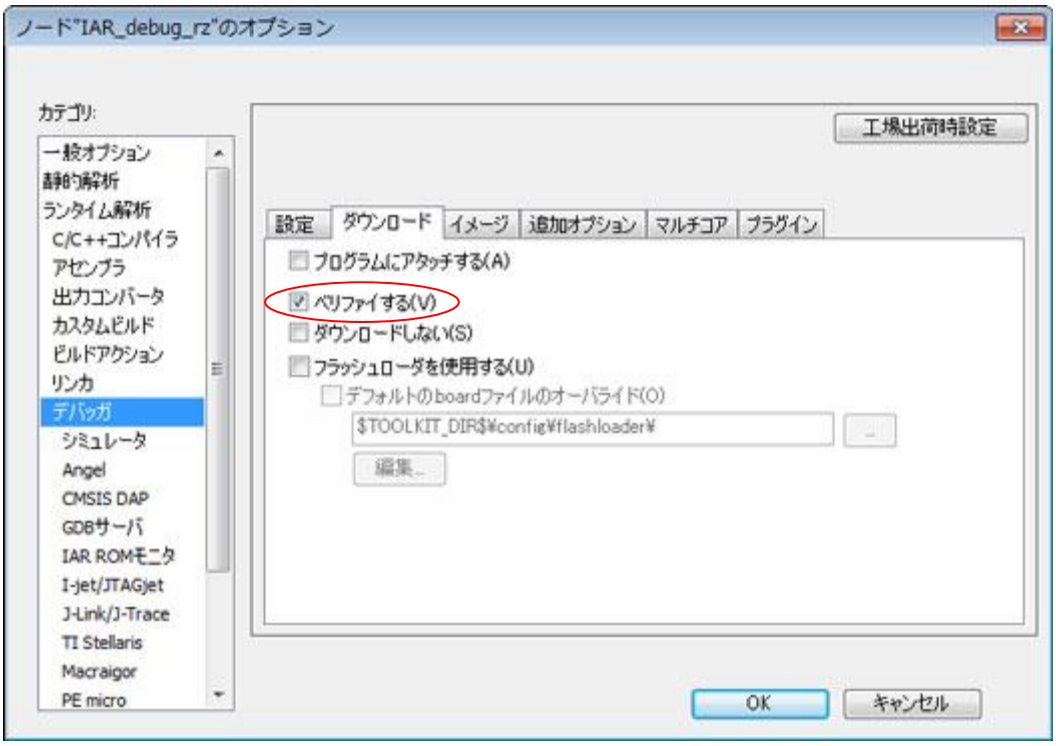


２－１０．デバッガの確認

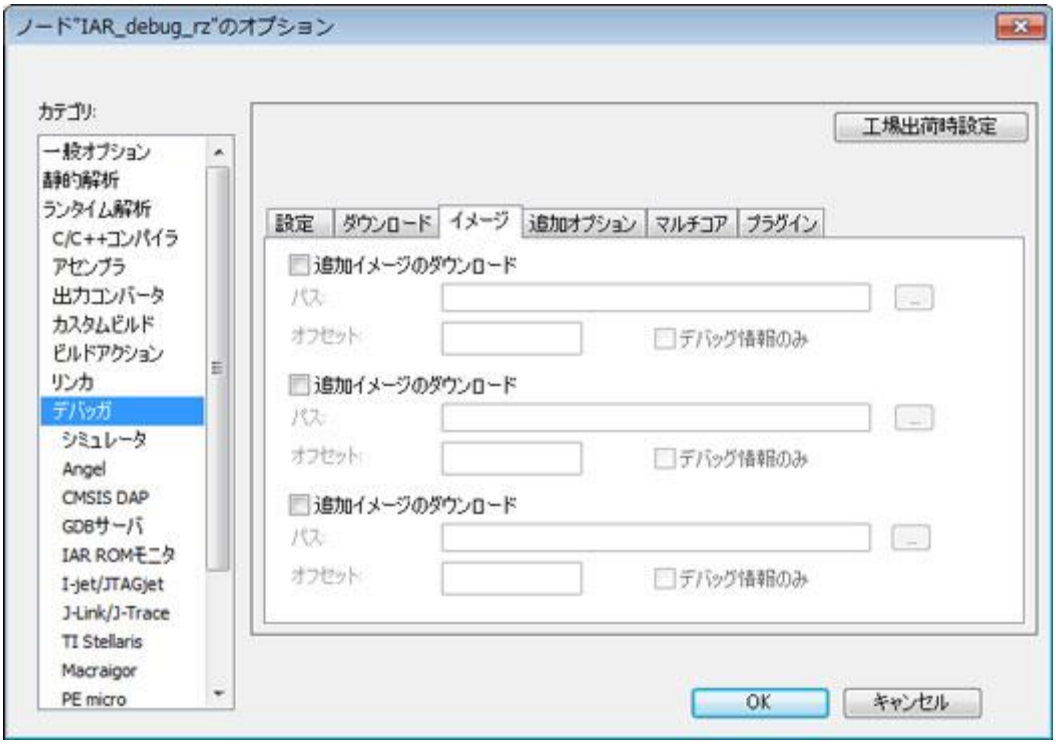
１）設定



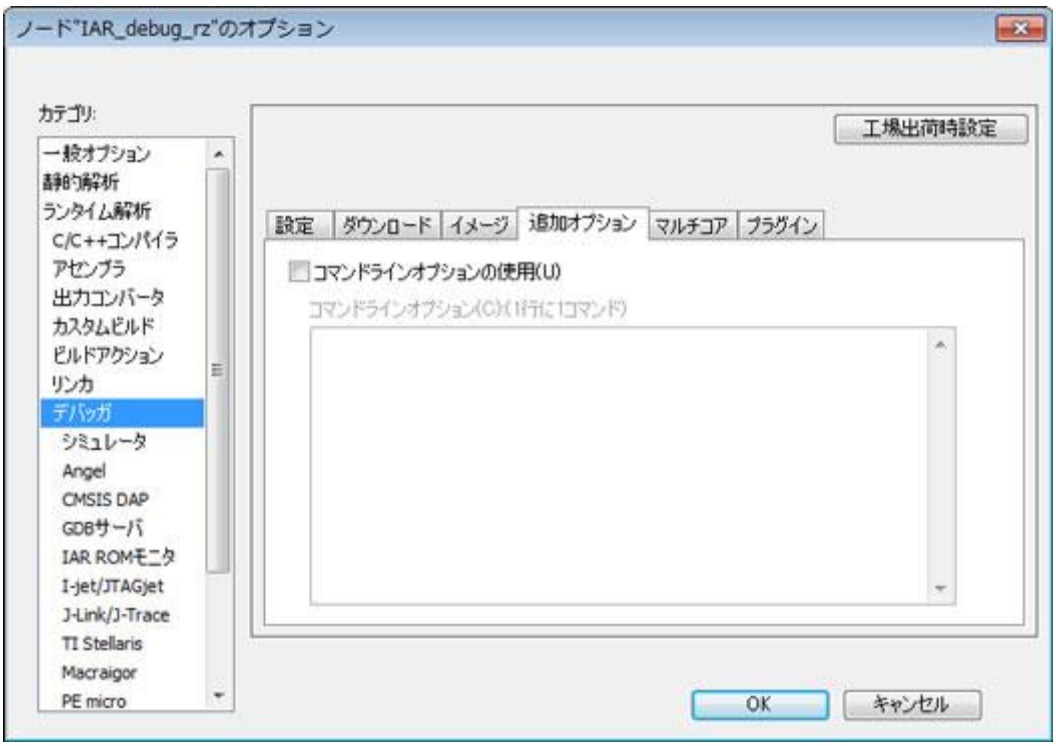
２）ダウンロード



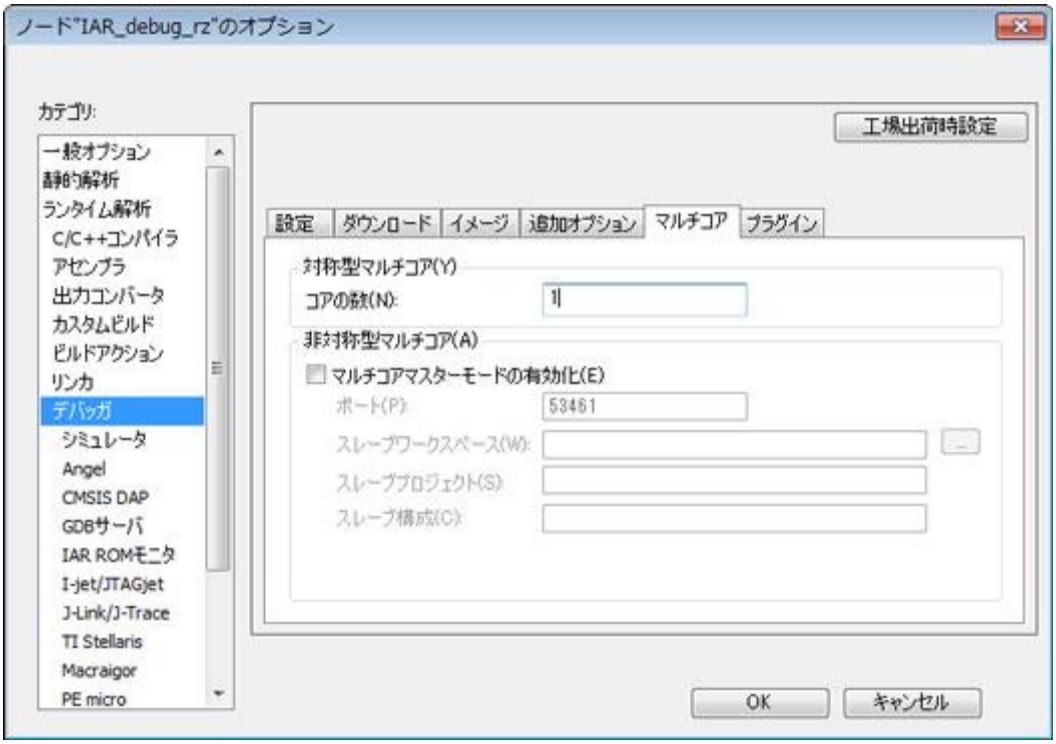
3）イメージ（設定なし）



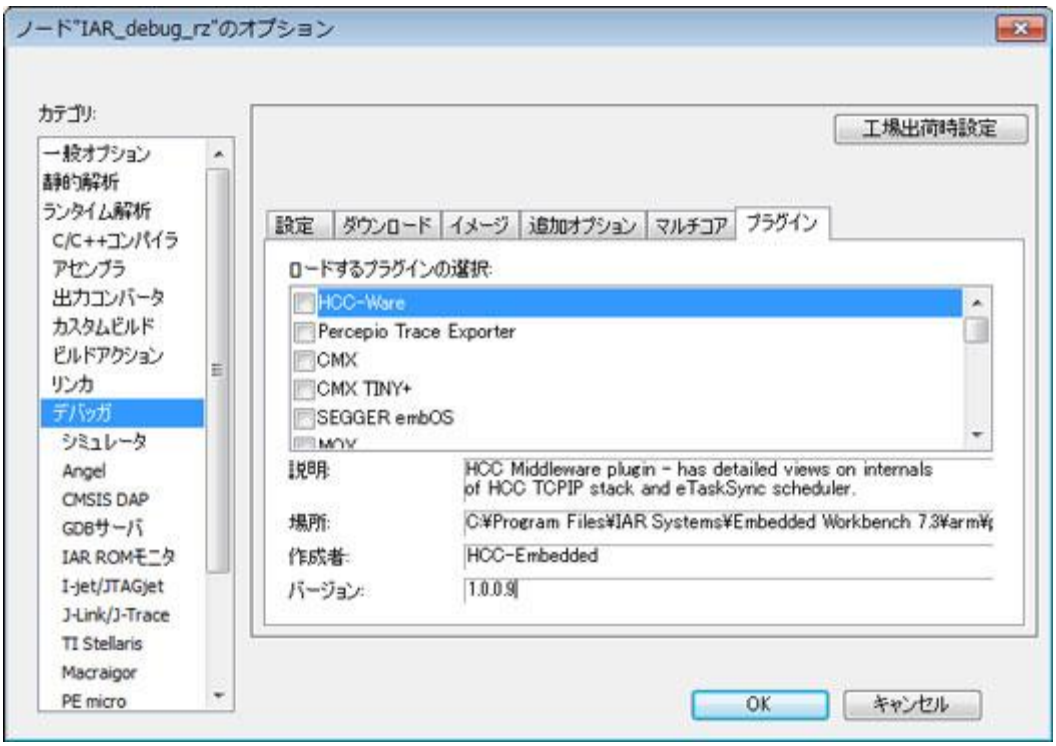
4）追加オプション（設定なし）



5）マルチコア（設定なし）

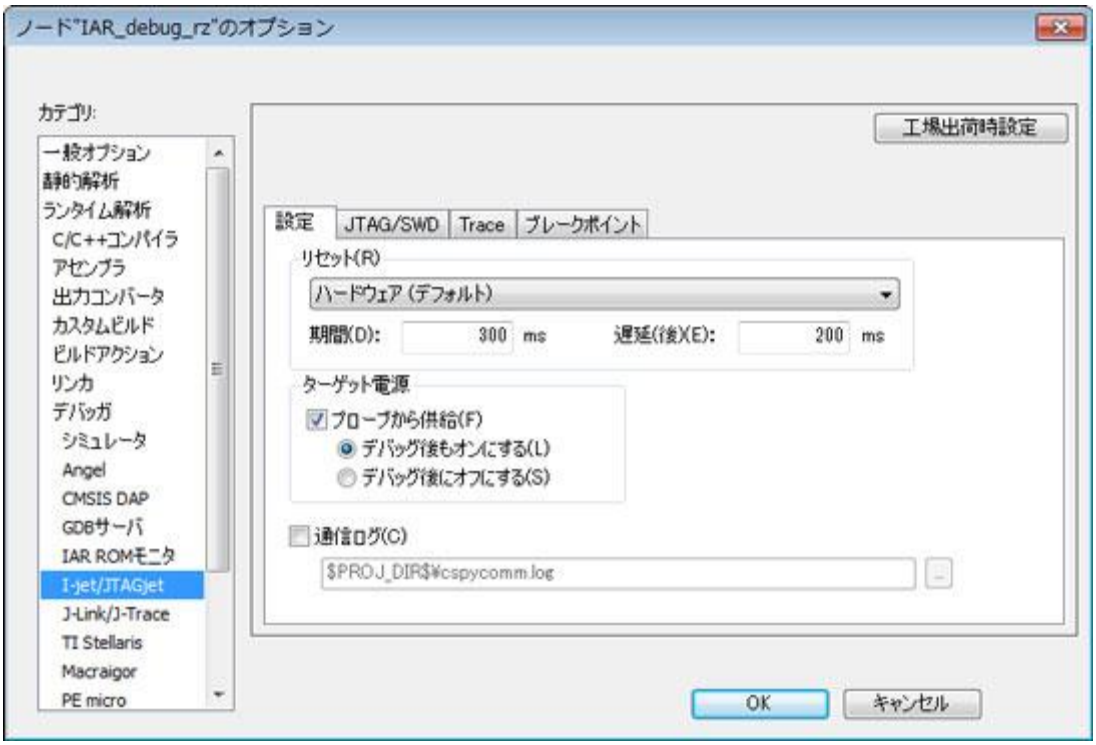


6）プラグイン（設定なし）

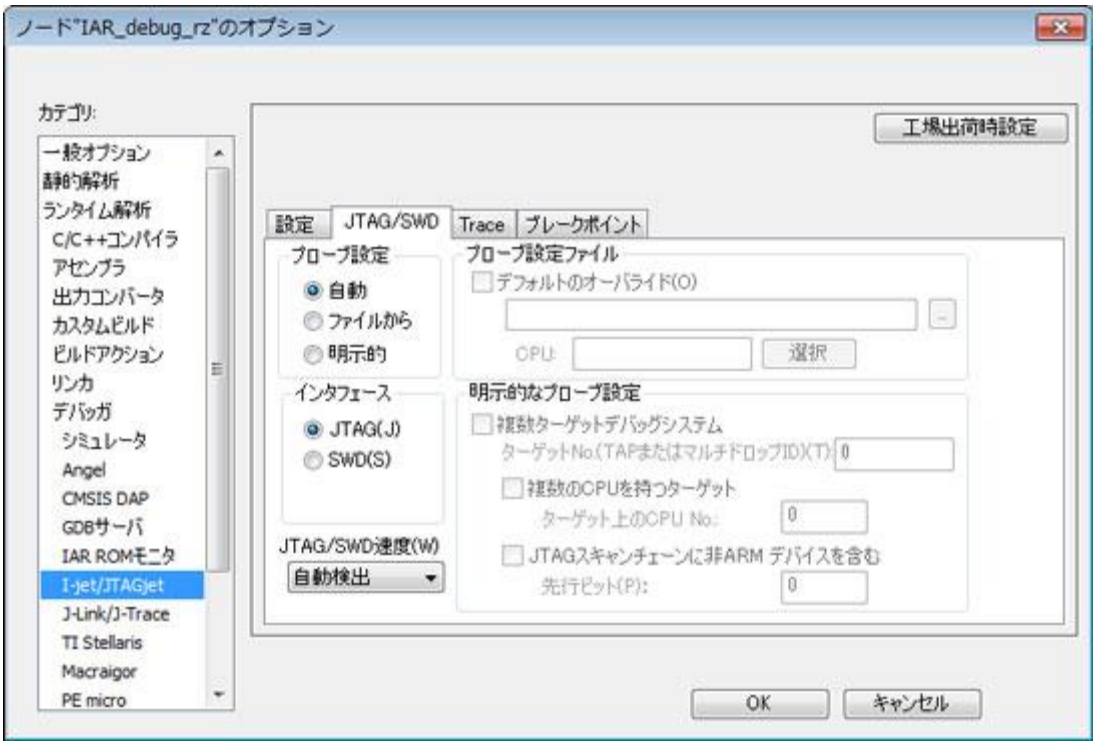


２－１１．I-jet/JTAGjet の確認

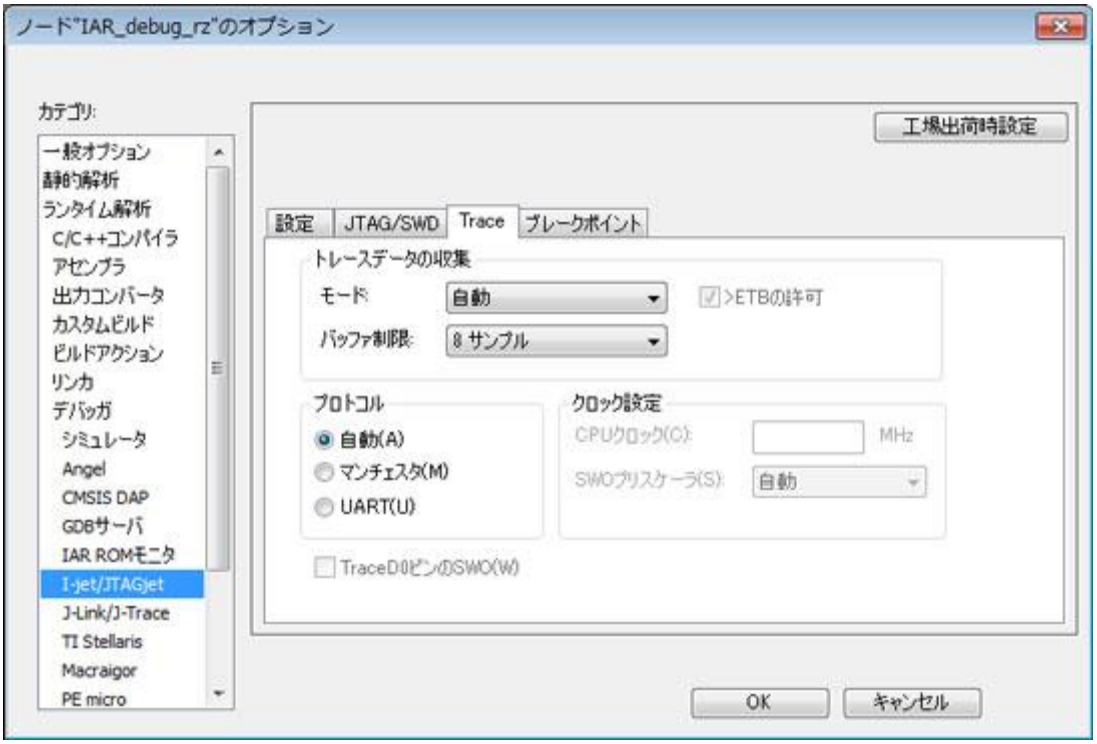
１）設定（デフォルト）



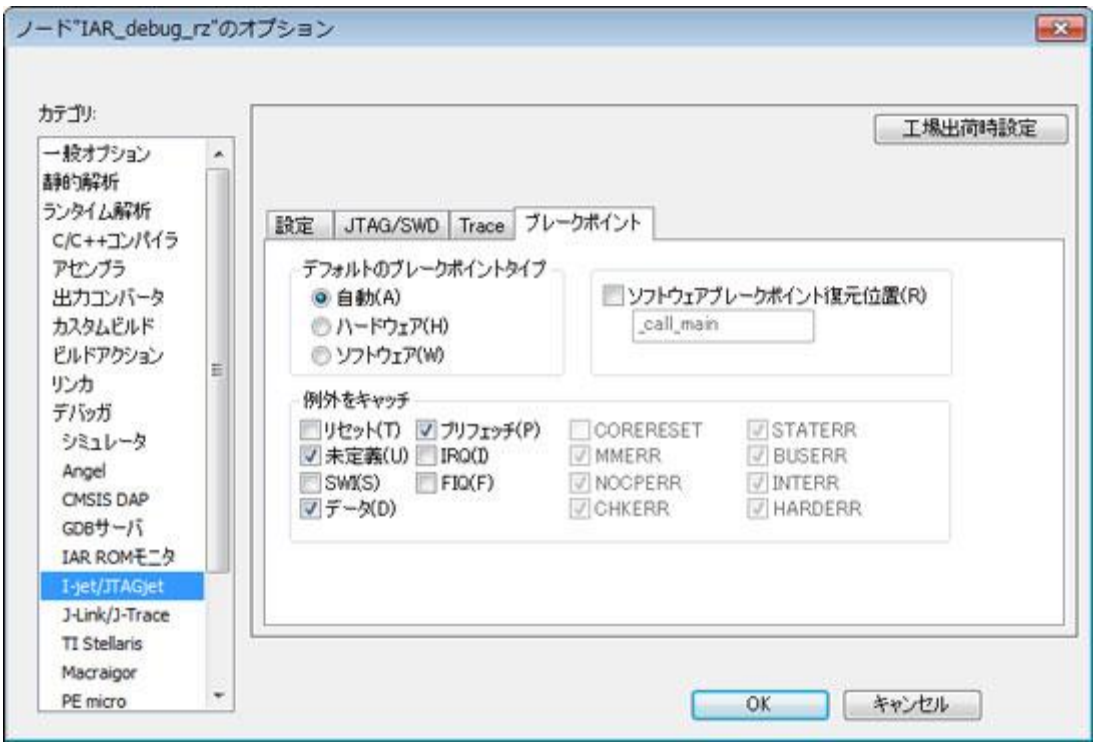
２）JTAG/SWD（デフォルト）



3）Trace（デフォルト）

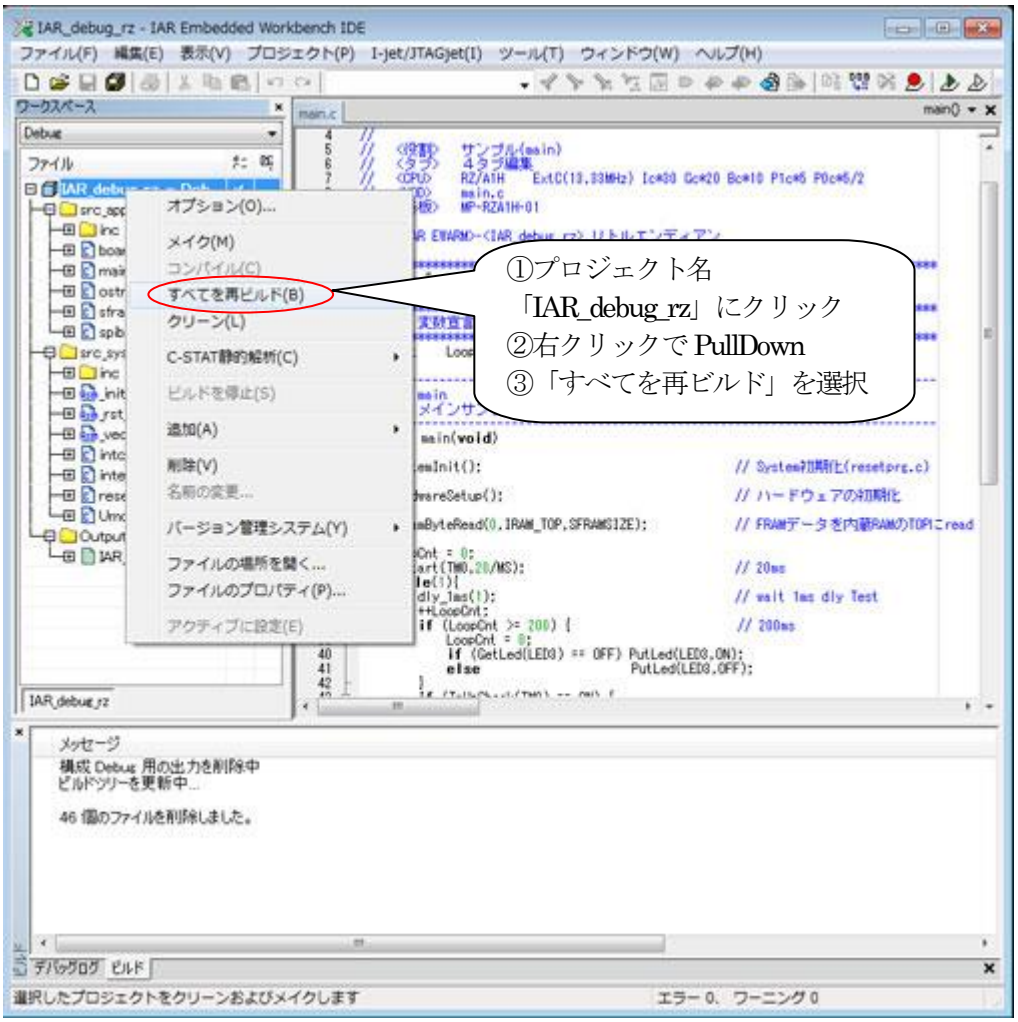


4）ブレークポイント（デフォルト）

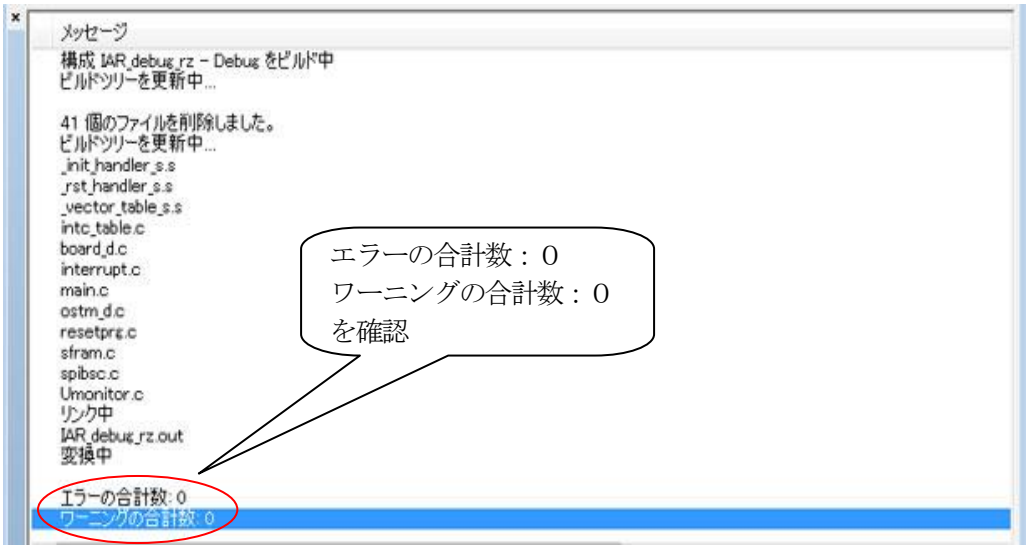


3．サンプルプロジェクト「IAR_debug_rz」をビルドする。

1）すべてをビルド

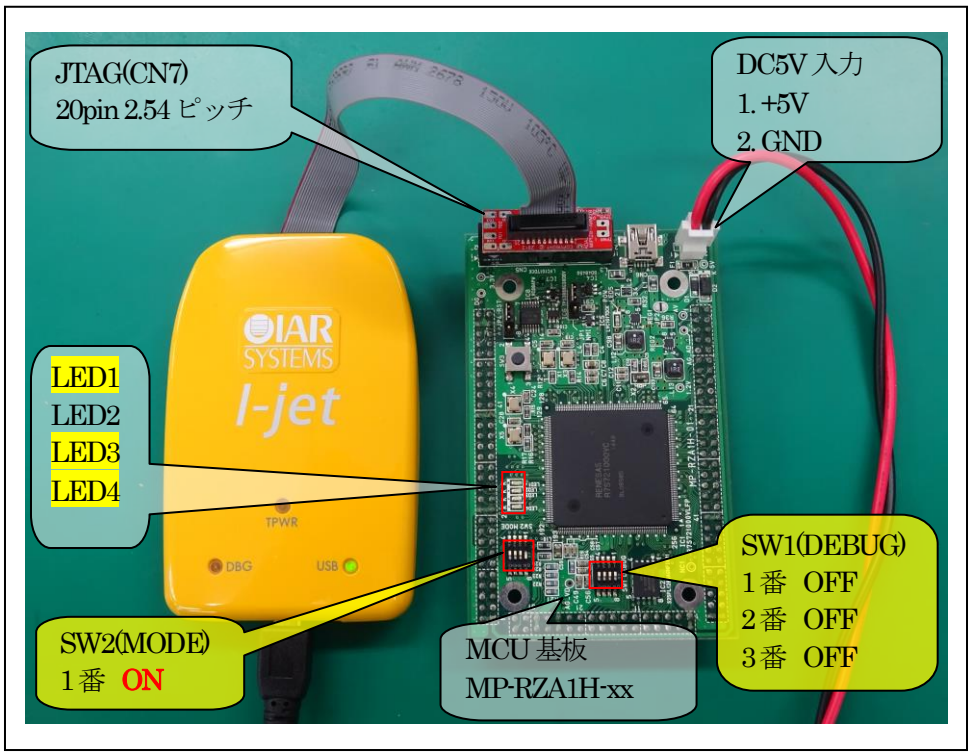


2）ビルド結果

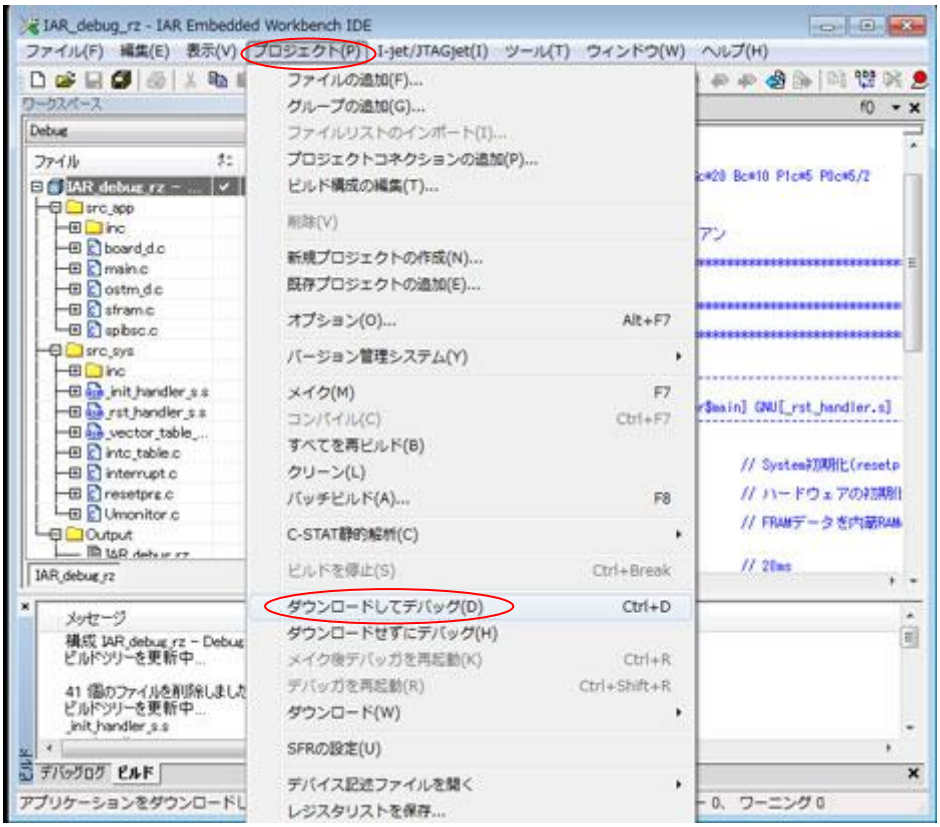


4．「I-jet/JTAG」でサンプルプロジェクト「IAR_debug_rz」をダウンロードする。

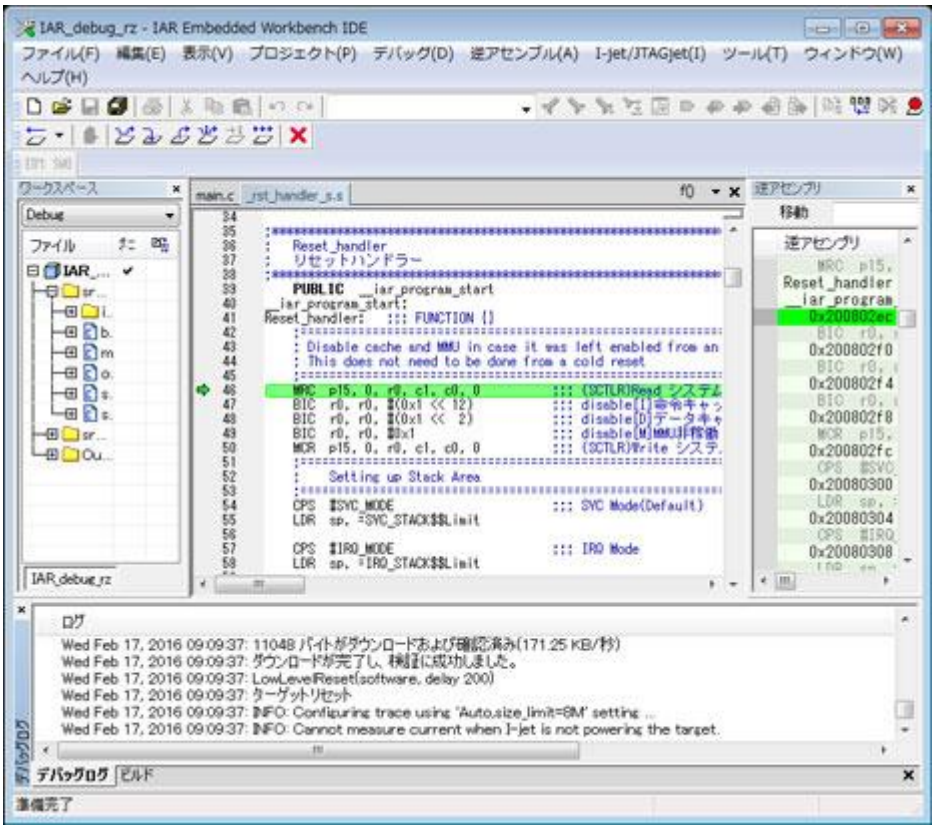
1）動作構成例（MP-RZA1H-01）



2）ダウンロード

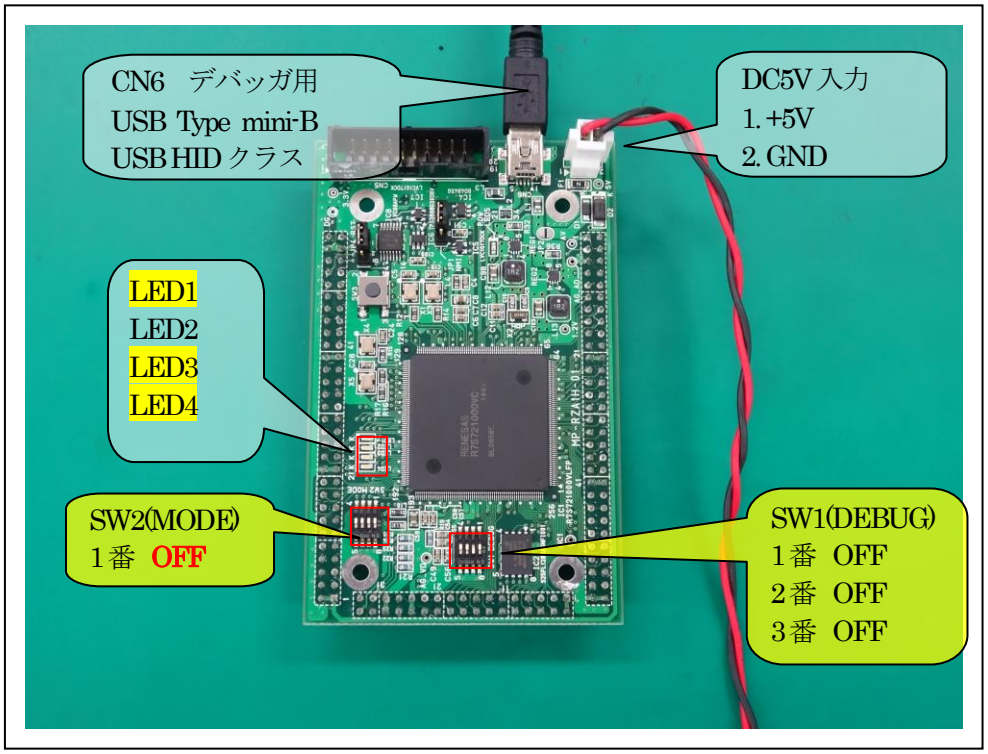


3）デバッガ起動画面

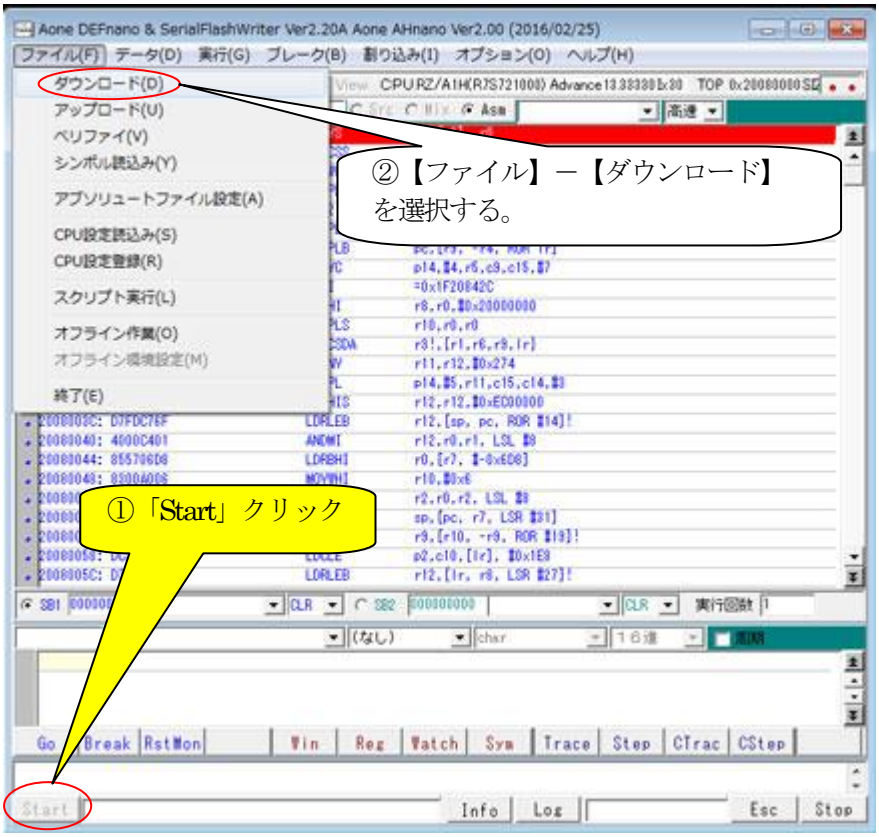

IAR_debug_rz - IAR Embedded Workbench IDE
ファイル(F) 編集(E) 表示(V) プロジェクト(P) デバッグ(D) 逆アセンブル(A) I-jet/JTAGjet(I) ツール(T) ウィンドウ(W)
ヘルプ(H)
ワークスペース
Debug
ファイル
IAR_debug_rz
main.c
rst_handler_s.s
逆アセンブリ
移動
Reset_handler
__iar_program_start:
Reset_handler: ::: FUNCTION []
; Reset_handler
; リセットハンドラー
PUBLIC __iar_program_start
; Disable cache and MMU in case it was left enabled from an
; This does not need to be done from a cold reset
MRC p15, 0, r0, c1, c0, 0 ::: (SCTLR)Read システム
BIC r0, r0, #(0x1 << 12) ::: disable[I]命令キャッ
BIC r0, r0, #(0x1 << 2) ::: disable[D]データキャ
BIC r0, r0, #0x1 ::: disable[M]MMU非有効
MCR p15, 0, r0, c1, c0, 0 ::: (SCTLR)Write システ
; Setting up Stack Area
CPS #SYC_MODE ::: SVC Mode(Default)
LDR sp, =SYC_STACK$$Limit
CPS #IRQ_MODE ::: IRQ Mode
LDR sp, =IRQ_STACK$$Limit
ログ
Wed Feb 17, 2016 09:09:37: 11048 バイトがダウンロードおよび確認済み(171.25 KB/秒)
Wed Feb 17, 2016 09:09:37: ダウンロードが完了し、検証に成功しました。
Wed Feb 17, 2016 09:09:37: LowLevelReset(software, delay 200)
Wed Feb 17, 2016 09:09:37: ターゲットリセット
Wed Feb 17, 2016 09:09:37: INFO: Configuring trace using 'Auto.size_limit=8M' setting ...
Wed Feb 17, 2016 09:09:37: INFO: Cannot measure current when I-jet is not powering the target.
デバッグログ
ビルド
準備完了

5．「DEFnano」でサンプルプロジェクト「IAR_debug_rz」をダウンロードする。

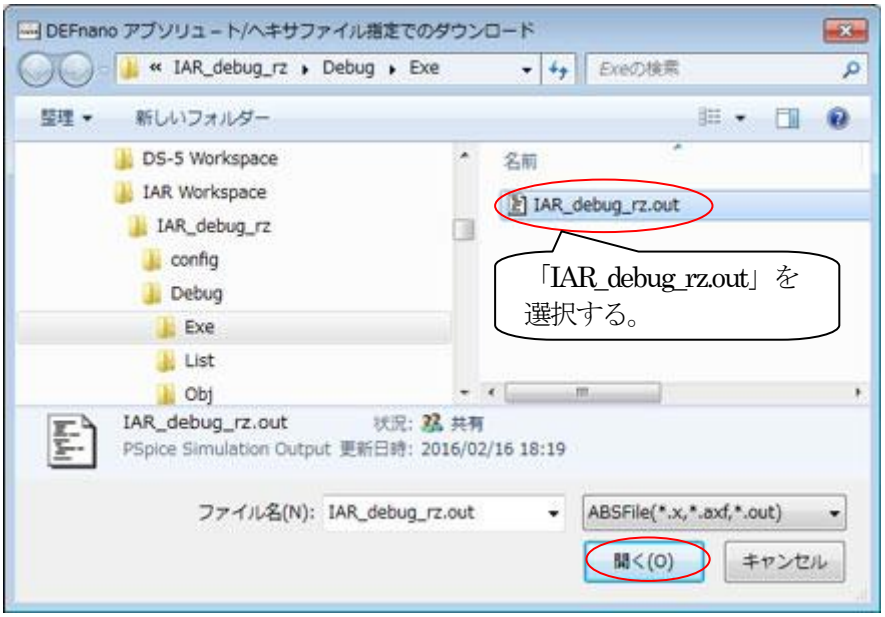
1）動作構成例（MP-RZA1H-01）



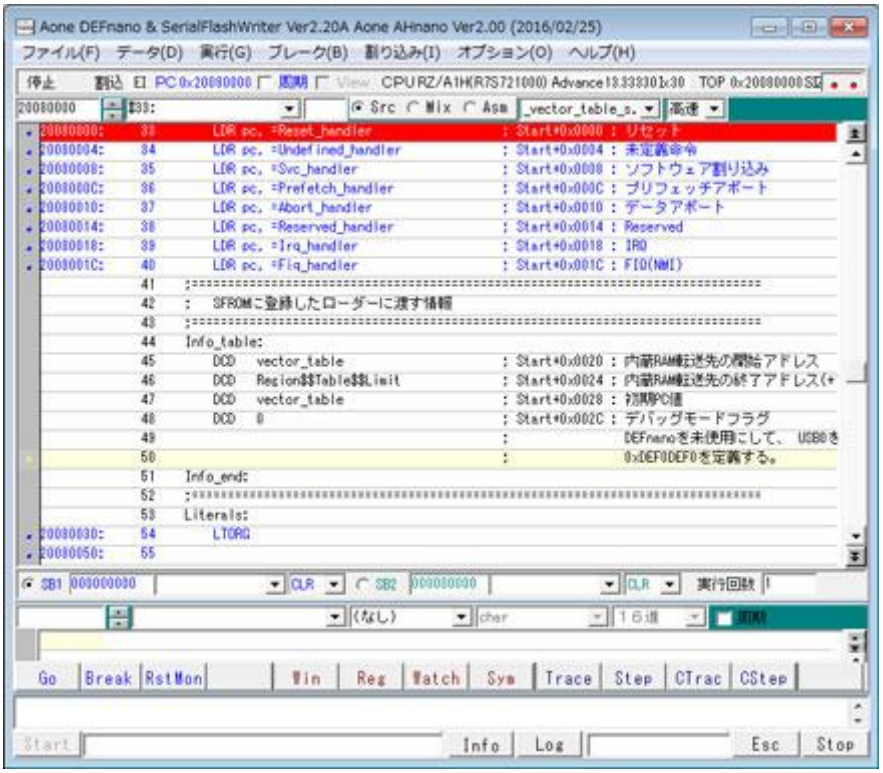
2）起動とダウンロード



3）ダウンロードファイルを選択する。



4）デバッガ起動画面



注＊１

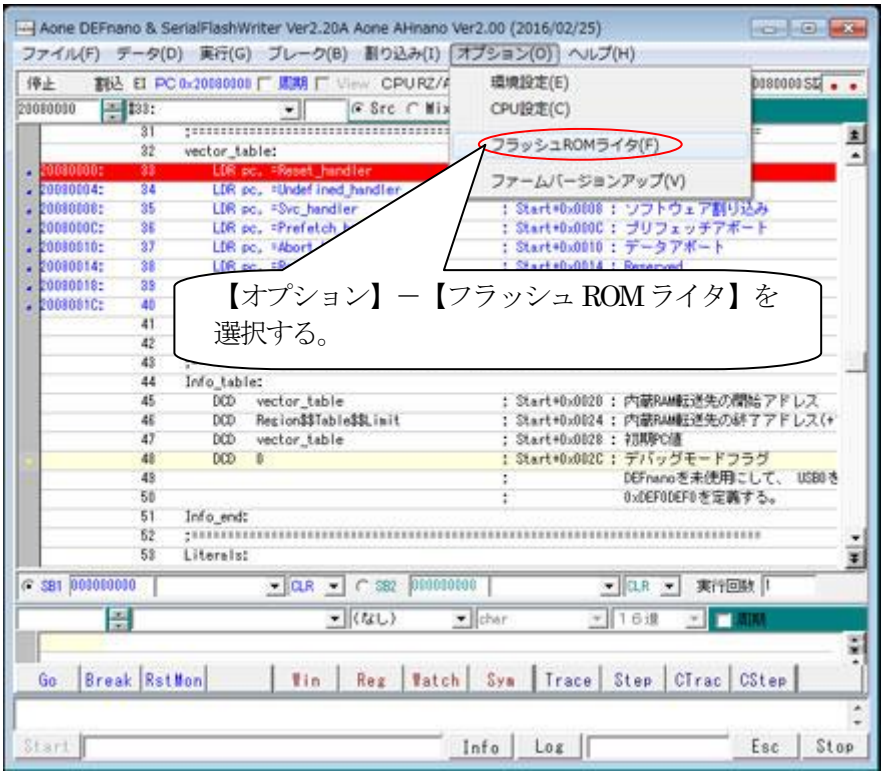
「__USED_DEFnano__=0」と使用しない側に定義してあっても内蔵 RAM へのダウンロード操作は可能です。ただし、再操作する場合はターゲット側のリセット操作が必要になります。

6．「DEFnano」でサンプルプロジェクト「IAR_debug_rz.mot」をシリアルフラッシュ ROM に書き込む。

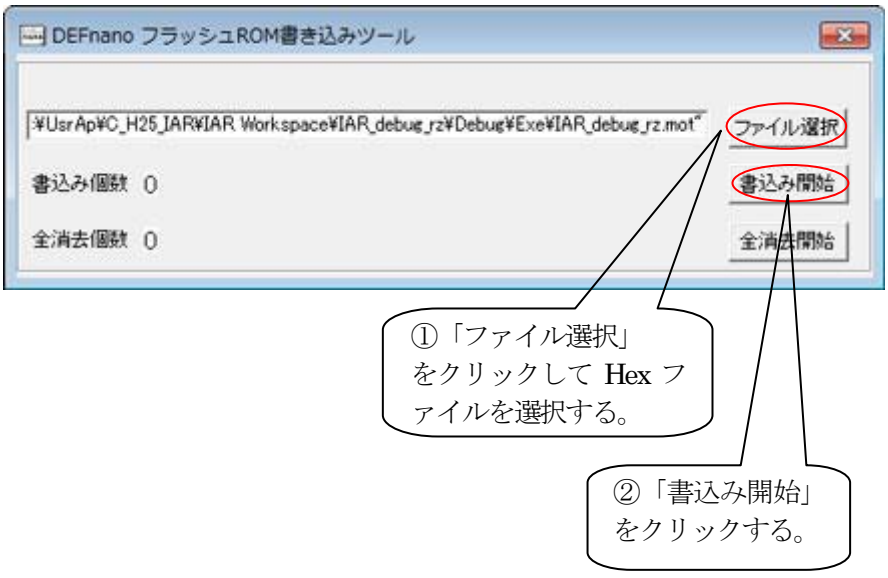
注＊１

「__USED_DEFnano__=0」と使用しない側に定義してあってもシリアルフラッシュ ROM への書き込み操作は可能です。ただし、再操作する場合はターゲット側のリセット操作が必要になります。

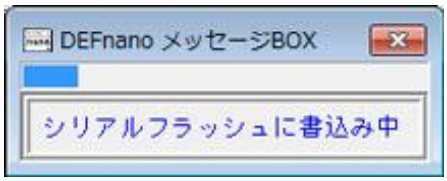
1）シリアルフラッシュ ROM ライタの起動



2）Hex ファイルの選択と書き込み開始



3）書き込み中



内部処理に応じてインジケータ表示
①RAM にダウンロード中
②セクターイレーズ
③シリアルフラッシュに書込み中
④ベリファイ中

4）書き込み終了

正常終了しますと、
「書込みは正常に終了しました。」
と表示され、個数がカウントアップします。
個数は、この窓を閉じますとゼロになります。

5）シリアルフラッシュ ROM の消去

「全消去開始」をクリックしますと、アプリケーションエリア(セクター２以降)のみを消去します。
ローダーとデバッガファームエリアは消去しません。

7．リンカ設定ファイル（*.icf）について

リンカ設定ファイルは、セグメントごとにアブソリュートアドレスを定義するためのファイルです。リンカファイル独自の予約語がありますので IAR 社が提供しているドキュメント「C/C++開発ガイド」の「パート２：リンカ設定ファイル」を参照して下さい。

```
/*####ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$¥config¥ide¥IcfEditor¥a_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x20080000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x20080200;
define symbol __ICFEDIT_region_ROM_end__   = 0x20087FFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20088000;
define symbol __ICFEDIT_region_RAM_end__   = 0x209FFFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0x8000;
define symbol __ICFEDIT_size_svcstack__ = 0x1000;
define symbol __ICFEDIT_size_irqstack__ = 0x1000;
define symbol __ICFEDIT_size_fiqstack__ = 0x1000;
define symbol __ICFEDIT_size_undstack__ = 0x100;
define symbol __ICFEDIT_size_abtstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__     = 0x8000;
/**** End of ICF editor section. ###ICF###*/

define symbol __ICFEDIT_region_RetRAM_start__     = 0x20000000;
define symbol __ICFEDIT_region_RetRAM_end__       = 0x2001FFFF;
define symbol __ICFEDIT_region_MirrorRetRAM_start__ = 0x60000000;
define symbol __ICFEDIT_region_MirrorRetRAM_end__   = 0x6001FFFF;
define symbol __ICFEDIT_region_MirrorRAM_start__    = 0x60090000;
define symbol __ICFEDIT_region_MirrorRAM_end__      = 0x604FFFFF;
define symbol __ICFEDIT_region_UncachedRAM_start__ = 0x60500000;
define symbol __ICFEDIT_region_UncachedRAM_end__   = 0x609FFFFF;

define memory mem with size = 4G;
define region RetRAM_region = mem:[from __ICFEDIT_region_RetRAM_start__ to __ICFEDIT_region_RetRAM_end__];
define region ROM_region = mem:[from __ICFEDIT_region_ROM_start__ to _ICFEDIT_region_ROM_end__];
define region RAM_region =mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__ ];
define region MirrorRAM_region=mem:[from __ICFEDIT_region_MirrorRAM_start__ to __ICFEDIT_region_MirrorRAM_end__];
define region MirrorRetRAM_region=mem:[from __ICFEDIT_region_MirrorRetRAM_start__ to __ICFEDIT_region_MirrorRetRAM_end__];
define region UncachedRAM_region = mem:[from __ICFEDIT_region_UncachedRAM_start__ to __ICFEDIT_region_UncachedRAM_end__];

/* stacks */
define symbol __ICFEDIT_size_TTB__  = 0x4000;
define block TTB       with alignment = 0x4000,size = __ICFEDIT_size_TTB__   { };
define block CSTACK    with alignment = 16,   size = __ICFEDIT_size_cstack__ { };
define block SVC_STACK with alignment = 16,   size = __ICFEDIT_size_svcstack__ { };
```

```
define block IRQ_STACK    with alignment = 16,  size = __ICFEDIT_size_irqstack__ { };
define block FIQ_STACK    with alignment = 16,  size = __ICFEDIT_size_fiqstack__ { };
define block ABT_STACK    with alignment = 16,  size = __ICFEDIT_size_abtstack__ { };
define block UND_STACK    with alignment = 16,  size = __ICFEDIT_size_undstack__ { };
define block HEAP         with alignment = 16,  size = __ICFEDIT_size_heap__   { };

initialize by copy with packing = none {
  readwrite,
};
do not initialize {
  section .noinit,
};
place at address mem: __ICFEDIT_intvec_start__ {
  section VECTOR_TABLE        /*asm*/
};
place at start of ROM_region {
  section RESET_INIT_HANDLER,   /*asm*/
  ro code,
  ro data,
};
place in RAM_region {
  block TTB,
  block IRQ_STACK,
  block FIQ_STACK,
  block SVC_STACK,
  block ABT_STACK,
  block UND_STACK,
  block CSTACK,
  block HEAP,
  readwrite,
  zi,
};
place at start of UncachedRAM_region {
};
```

リンカ設定ファイルを編集する場合の注意点

1） 1行（/*###ICF###）から１９行(###ICF###*/)は、エディタ等で直接変更しないで下さい。変更が必要な場合は、EWARM の【オプション】－【リンカ】－「設定」－「編集」で変更する。

8．ベクタテーブルとローダーの関係について

MP-RZA1H 基板は、シリアルフラッシュ ROM にローダーとアプリケーションプログラムを記憶させ、電源 ON 時にローダーが RZA1H の内蔵 RAM にアプリケーションプログラムをロードして実行させる仕組みになっています。ローダーは内蔵 RAM にロードする時にロード先の先頭アドレスと最終アドレスと実行開始アドレスを知る必要があります。この情報を得るため独自の定義が必要なため下記に説明します。

1）ローダーが必要な情報をベクタテーブルに登録する。「_vector_table_s.s」

```
Start
;=========================================================================
;   Entry point for the Reset handler
;=========================================================================
vector_table
    LDR pc, =Reset_handler          ; Start+0x0000 : リセット
    LDR pc, =Undefined_handler      ; Start+0x0004 : 未定義命令
    LDR pc, =Svc_handler            ; Start+0x0008 : ソフトウェア割り込み
    LDR pc, =Prefetch_handler       ; Start+0x000C : プリフェッチアボート
    LDR pc, =Abort_handler          ; Start+0x0010 : データアボート
    LDR pc, =Reserved_handler       ; Start+0x0014 : Reserved
    LDR pc, =Irq_handler            ; Start+0x0018 : IRQ
    LDR pc, =Fiq_handler            ; Start+0x001C : FIQ(NMI)
;=========================================================================
;   SFROMに登録してあるローダーに渡す情報
;=========================================================================
Info_table
DCD   vector_table              ; Start+0x0020 :①内蔵RAM転送先の開始アドレス
DCD   Region$$Table$$Limit      ; Start+0x0024 :②内蔵RAM転送先の終了アドレス(+1)
DCD   vector_table              ; Start+0x0028 :③初期PC値
DCD   0                         ; Start+0x002C :④デバッグモードフラグ
                                ;   DEFnanoを未使用にして、 USB0を開放する場合は、
                                ;   0xDEF0DEF0を定義する。
Info_end
;=========================================================================
```

【重要】
①②③④の情報は、ROM 化するためには必要な情報テーブルです。必ず、定義して下さい。

【注意事項】
④で「DEFnano 未使用」コード「0xDEF0DEF0」をセットし、シリアルフラッシュ ROM に登録した場合、二度と DEFnano を使用することが出来なくなります。復帰したい場合は、JTAG デバッガ等でシリアルフラッシュ ROM アドレス「0x2_002C」を未使用コード「0xDEF0DEF0」以外の数値を直接書き込んでください。

以上です。

９．注意事項

・本文書の著作権は、エーワン（株）が保有します。
・本文書を無断での転載は一切禁止します。
・本文書に記載されている内容についての質問やサポートはお受けすることが出来ません。
・本文章に関して、ARM 社およびルネサス　エレクトロニクス社への問い合わせは御遠慮願います。
・本文書の内容に従い、使用した結果、損害が発生しても、弊社では一切の責任は負わないものとします。
・本文書の内容に関して、万全を期して作成しましたが、ご不審な点、誤りなどの点がありましたら弊社までご連絡くだされば幸いです。
・本文書の内容は、予告なしに変更されることがあります。

１０．商標

・EWARM は、IAR 社の登録商標、または商品名称です。
・RZ および RZ/A1H は、ルネサス　エレクトロニクス株式会社の登録商標、または商品名です。
・その他の会社名、製品名は、各社の登録商標または商標です。

１１．参考文献

・「RZ/A1H グループ　ユーザーズマニュアル　ハードウェア編」
ルネサス　エレクトロニクス株式会社
・ルネサス　エレクトロニクス株式会社提供のサンプル集
・「IDE プロジェクト管理およびビルドガイド UIDEARM-9-j」　IAR 社
・「IAR C/C++開発ガイド コンパイラおよびリンク DARM-14-j」　IAR 社
・「IAR アセンブラリファレンスガイド AARM-9-j」　IAR 社
・「IAR デバッグプローブガイド IARprobes-2-j」　IAR 社
・その他

〒486-0852
愛知県春日井市下市場町 6-9-20
エーワン株式会社
http://www.robin-w.com