

CodeWarrior for NINTENDO DSi **Technical FAQ**

第 1.2 版 2010 年 06 月 07 日

フリースケール・セミコンダクタ・ジャパン株式会社

目次

1	FAQ	4
1.1	セットアップ	4
1.1.1	アクセスパスが見つかりません.....	4
1.1.2	IDEやコマンドラインツールの動作が急に重くなった	4
1.2	C/C++コンパイラ	5
1.2.1	プラグマ「options align=」が働かない.....	5
1.2.2	プラグマ優先なのに、プラグマ「inline_depth」を使用していてもコンパイルオプションで「Don't Inline」を指定するとインラインされなくなる	5
1.2.3	逆アセンブルするとレジスタr3が無駄にスタックに退避されている場合がある	6
1.2.4	プロファイラ使用時の注意点	6
1.2.5	インライン展開されない場合がある	7
1.2.6	「Nintendo CodeGen」パネルの「Replace 8-bit memory access」を有効にすると動作がおかしくなる	7
1.2.7	「Use Instance Manager」オプションが機能しない.....	7
1.2.8	古い、または、不正なプリコンパイルヘッダが使用された時、コンパイラがクラッシュする場合がある	7
1.2.9	コンパイラはプログラム単位のIPAモードをサポートしていない.....	8
1.3	アセンブラ	9
1.3.1	Cソースと連携をとるルーチンをアセンブラで作るときに値を保護すべきレジスタは？	9
1.4	リンカ	10
1.5	ライブラリ	10
1.5.1	TWL SDK と malloc() の使用について	10
1.5.2	iostream.hの使用方法は？	11
1.5.3	サポートライブラリのリビルド.....	11
1.6	IDE	13
1.6.1	日本語を入力すると文字化けする	13
1.6.2	外部エディタを使っている時に一時的に内部エディタでソースファイルを開く方法は？	13
1.6.3	ツールバーにボタンを追加・削除する方法は？	13
1.7	デバッグ	15
1.7.1	オーバーレイ部分のデバッグが正しく行なえない。	15
1.7.2	IDEで.nefファイルを読み込ませてのデバッグ時にファイルをロックしないようにしたい.....	15
2	TIPS.....	16
2.1	C/C++プログラミングに関するTIPS	16
2.1.1	パックした構造体のメンバにアクセスする方法	16
2.2	Windows環境に関連するTIPS	17
2.2.1	フォルダのアイコンを変更するとプロジェクトが開けなくなる	17

3	その他	18
----------	------------------	-----------

改訂履歴

版数	日付	改訂概要
1.0	2008 年 06 月 25 日	CodeWarrior for NINTENDO DS Technical FAQ 第 1.2 版を元に作成
1.1	2009 年 09 月 30 日	1.2.8、古い、または、不正なプリコンパイルヘッダが使用された時、コンパイラがクラッシュする場合がある 1.2.9、コンパイラはプログラム単位のIPAモードをサポートしていない
1.2	2010 年 06 月 07 日	1.5.2、「iostream.h の使用方法は？」の追加 1.5.3、「サポートライブラリのリビルドについて」の追加

Freescall, the Freescall logo, and CodeWarrior are trademarks of Freescall Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2008–2010 Freescall Semiconductor, Inc. All rights reserved.

1 FAQ

1.1 セットアップ

1.1.1 アクセスパスが見つかりません

CodeWarrior IDE の編集メニューから「環境設定」を選択し、「ソースツリー」設定パネルを確認してください。

サンプルプロジェクト、およびステーションナリでは TWLSDK_ROOT および IS_TWL_DEBUGGER_DIR の設定が必要です。

名前	形式	値	備考
TWLSDK_ROOT	環境変数	CW_TWLSDK_ROOT	環境変数「CW_TWLSDK_ROOT」に TwlSDK へのフルパスが設定されていること。
IS_TWL_DEBUGGER_DIR	環境変数	IS_TWL_DEBUGGER_DIR	環境変数「IS_TWL_DEBUGGER_DIR」に IS-TWL-DEBUGGER がインストールされている場所へのフルパスが設定されていること。

1.1.2 IDEやコマンドラインツールの動作が急に重くなった

CodeWarrior ではライセンスの管理に FlexLM を使用していますが、FlexLM を使用している他社製品が環境変数「LM_LICENSE_FILE」を使用していると、ライセンスの検索順序が変更されてしまい、上記現象が発生することがあります。

環境変数「LM_LICENSE_FILE」に他社製品のライセンスパス（サーバ名やファイルパス）が指定されている場合、以下の手順をお試しください。

- 1) 「LM_LICENSE_FILE」環境変数で指定されている、他社製品のライセンスサーバが停止していないか確認する。
- 2) CodeWarrior の検索順序を他社製品のパスより前に置く（ライセンスパスは、セミコロン(';') で区切ります。）

例) LM_LICENSE_FILE = C:\Program Files\Freescale\CW for NINTENDO TWL Vx.x
 \license.dat; (他社製品のライセンスパス)

(注) x.xにはバージョン番号が入ります。SPxなどがついていることもあります。

- 3) 「LM_LICENSE_FILE」環境変数を削除する (内容をセーブしておいてください)

CodeWarrior IDE は、「LM_LICENSE_FILE」環境変数が設定されていない場合でも、インストールフォルダの license.dat ファイルを使用します。コマンドラインツールをご使用の場合は、コマンドラインツールと同一フォルダ内に license.dat をコピーしておいてください。

1.2 C/C++コンパイラ

1.2.1 プラグマ「options align=」が働かない

CodeWarrior for NINTENDO TWL では、「#pragma options align=」は数字指定を含め使用出来ません。

ソースの互換性の観点から「native」指定と「reset」指定の場合は警告を出さなくなっているのみです。

1.2.2 プラグマ優先なのに、プラグマ「inline_depth」を使用してもコンパイルオプションで「Don't Inline」を指定するとインラインされなくなる

全体的に、コンパイルオプションよりプラグマが優先となっています。

しかしながら、inline 関係のプラグマとコンパイルオプションは少々複雑な関係になっています。

コンパイルオプション「InlineDepth」は、プラグマ「dont_inline」とプラグマ「inline_depth」の両方の動作を持っています。

コンパイルオプション 「InlineDepth」での指定	同等のプラグマ表現
Don't Inline	"dont_inline on"
Smart	"dont_inline off" かつ "inline_depth(smart)"
1 ~ 8	"dont_inline off" かつ "inline_depth(n)" (n=1 to 8)

また、プラグマ「dont_inline」はプラグマ「inline_depth」より優先されます。

従いまして、優先順を不等号記号で表現しますと、

高

↑ `dont_inline` プラグマ

| > `InlineDepth` オプションで"Don't Inline"を指定

| > `inline_depth` プラグマ

↓ > `InlineDepth` オプションで"Don't Inline"以外を指定

低

となります。

コンパイルオプションで **Don't Inline** を指定した状態で、各ソースファイル単位でプラグマを用いて `inline` 展開の制御を行う場合には、

```
#pragma dont_inline off
```

```
#pragma inline_depth(smart)
```

という順序で記述して下さい。

1.2.3 逆アセンブルするとレジスタr3が無駄にスタックに退避されている場合がある

スタック領域を8バイトアライメントに保つ為に、CodeWarrior for NINTENDO TWL のコンパイラでは、奇数個のレジスタをスタックに退避する場合には R3 レジスタをダミーとして用いて偶数個のレジスタを **PUSH** しています。

この時、r3 レジスタはダミーという用途以外の意図はありません。

(CodeWarrior for NINTENDO DS 1.2 以前のコンパイラでは、『SP を減算／加算する』という方法を使っておりましたが、この方法では瞬間的にスタックのアラインが8バイト境界から外れてしまう上に、1命令余分にかかるので、変更になりました。)

1.2.4 プロファイラ使用時の注意点

- プロファイラは作業用の領域をターゲット (IS-TWL-DEBUGGER) のメモリ上に確保します。この為、`ProfilerInit()` の第3引数 (プロファイラが扱う関数の最大合計数) と第4引数 (プロファイラが解析する関数 **CALL** の最大深さ) であまり大きい値を指定しますとターゲットのメモリをオーバーしてしまいます。
- プロファイラは、スレッド及び割り込みに対応しておりません。プロファイラで解析を行なう範囲で、スレッドの呼び出しや割り込みの発生が頻繁に行なわれておきますと、正しく解析できなかったり、プロファイラの作業領域のデータが壊れたりする可能性があります。

1.2.5 インライン展開されない場合がある

下記の関数はインライン展開されません。

- 破棄が必要なクラスオブジェクトを戻す関数
- 破棄が必要なクラス引数を持つ関数
- 可変引数リストを持つ関数

1.2.6 「Nintendo CodeGen」パネルの「Replace 8-bit memory access」を有効にすると動作がおかしくなる

CodeWarrior for NINTENDO DS 2.0 からは「Replace 8-bit memory access」オプションはサポートされなくなりました。

このオプションは、以前の TEG ボード使用時の制限に対応する為のものでした。

このオプションが使えるのは CodeWarrior for NINTENDO DS 1.2 迄です。

CodeWarrior for NINTENDO TWL では、このオプションは決して有効にはしないで下さい。

1.2.7 「Use Instance Manager」オプションが機能しない

CodeWarrior for NINTENDO TWL では「Use Instance Manager」オプションはサポートしていません。

1.2.8 古い、または、不正なプリコンパイルヘッダが使用された時、コンパイラがクラッシュする場合がある

新しいバージョンのコンパイラを使用した時、最初に TWL SDK 中の古いプリコンパイルヘッダを削除しなければなりません。TWL SDK 中の古いプリコンパイルヘッダは {TWLSDK_ROOT}¥cache¥include フォルダに存在します。

1.2.9 コンパイラはプログラム単位のIPAモードをサポートしていない

警告！ プログラム単位の IPA モードは、ARM 開発では完全にテストされていません。各自の責任でご使用ください。

1.3 アセンブラ

1.3.1 Cソースと連携をとるルーチンをアセンブラで作るときに値を保護すべきレジスタは？

CodeWarrior for NINTENDO TWL のコンパイラでは ARM 社の ATPCS (Arm Thumb Procedure Call Standards)に基づいています。

- パラメータ値をルーチンに渡し、結果値を受け取るレジスタには、レジスタ r0 ～ r3 を使用します。サブルーチンコール間では、どのような目的にも r0 ～ r3 を使用することができます。呼び出されるルーチンは、復帰する前に r0 ～ r3 を復元する必要はありません。
- ルーチンのローカル変数の値を保持するレジスタには、レジスタ r4～ r11 を使用します。Thumb 状態の場合はほとんどの命令で、ローカル変数にはレジスタ r4 ～r7 しか使用できません。呼び出されるルーチンがこれらのレジスタを使用する場合は、そのルーチンが復帰前にこれらのレジスタの値を復元する必要があります。
- レジスタ r12 は、コール内スクラッチレジスタ、ip です。プロシージャコール間では、r12 はどのような目的にも使用することができます。呼び出されるルーチンは、復帰前に r12 を復元する必要はありません。
- レジスタ r13 は、スタックポインタ、sp です。このレジスタを他の目的で使用してはなりません。呼び出されるルーチンからのイグジットで sp に保持される値は、そのエントリで保持される値と同じでなければなりません。
- レジスタ r14 は、リンクレジスタ、lr です。復帰アドレスを保存する場合は、r14 をコール間の他の目的に使用することができます。
- レジスタ r15 は、プログラムカウンタ、pc です。このレジスタを他の目的に使用することはできません。

レジスタ	同義語	特殊	プロシージャコール標準における役割
r15	–	pc	プログラムカウンタ
r14	–	lr	リンクレジスタ
r13	–	sp	スタックポインタ
r12	–	ip	プロシージャコール内スクラッチレジスタ
r11	v8	–	ARM 状態変数レジスタ 8
r10	v7	sl	ARM 状態変数レジスタ 7。 スタックチェック対象バリエーション内のスタックリミットポインタ
r9	v6	sb	ARM 状態変数レジスタ 6。 RWPI バリエーションのスタティックベース
r8	v5	–	ARM 状態変数レジスタ 5

r7	v4	–	変数レジスタ 4
r6	v3	–	変数レジスタ 3
r5	v2	–	変数レジスタ 2
r4	v1	–	変数レジスタ 1
r3	a4	–	引数/結果/スクラッチレジスタ 4
r2	a3	–	引数/結果/スクラッチレジスタ 3
r1	a2	–	引数/結果/スクラッチレジスタ 2
r0	a1	–	引数/結果/スクラッチレジスタ 1

<注意>

レジスタの値をスタックに退避する時は、スタック領域が8バイトアライメントに保たれるようにして下さい。一度にストアするレジスタが奇数個の場合は、R3をダミーとして含めて偶数個にする方法をお勧めします。

1.4 リンカ

(現在、リンカに関する情報は有りません。)

1.5 ライブラリ

1.5.1 TWL SDK と malloc() の使用について

TWL SDK の crt0.o を使用するアプリケーションで MSL C ライブラリの malloc() 関数を使用するためには、あらかじめ TWL OS の API を使用してヒープの初期化を行ってください。

malloc() 関数は __sys_alloc() 関数を使用してカレントヒープからメモリを取得します。ランタイムライブラリソースにも TWL SDK の API を使用してヒープを作成するための関数 __init_hardware() が用意されています。

__init_hardware()は

```
{CodeWarriorインストールディレクトリ}¥ARM_EABI_Support¥Runtime¥Runtime_ARM¥
(Common_Source)¥__NITRO_eabi_init.c"
```

で定義されています。

1.5.2 iostream.hの使用方法は？

iostream ヘッダを使用するためには、malloc()が必要です。malloc()関数を使用するためには、事前にランタイム関数__init_hardware()を呼び出す必要があります。

__init_hardware()は

¥ARM_EABI_Support¥Runtime¥Runtime_ARM¥(Common_Source)¥__NITRO_eabi_init.c

で定義されています。

通常、__init_hardware()をカスタマイズする必要があるでしょう（例えば、ヒープメモリのサイズの変更）。そのカスタマイズコードは TwlStartup()関数に直接記述することができます。

例：

```
#include <twl.h>
```

```
#include <iostream.h>
```

```
extern "C" void __init_hardware(); // Setup heap memory
```

```
extern "C" void TwlStartup();      // this func is called before C++ init
                                   // (__call_static_initializers())
```

```
void TwlStartup()
```

```
{
    __init_hardware();
}
```

```
void TwlMain ()
```

```
{
    OS_Init();
    OS_Printf("Welcome to CodeWarrior for NINTENDO TWL¥n");
    OS_Terminate();
}
```

1.5.3 サポートライブラリのリビルド

DSi では、あらかじめビルドされた標準 C/C++とランタイムライブラリが提供されています。通常、これらのライブラリをリビルドする必要はありません。またリビルドすることは推奨されていません。しかし、必要な場合、提供されている **makefile** を使用してリビルドできます。コマンドラインで実行する場合、**cygwin** のコマンドシェル **bash** のみ利用可能です。ライブラリを

リビルドするためには、環境変数 `CWINSTALL` を設定する必要があります。bash では、以下の例のように設定します。

```
> export CWINSTALL="C:\Program Files\Freescale\CW for NINTENDO DSi V1.3"
```

パスにスペースが含まれる場合、ダブルクォーテーションで囲む必要があります。`CWINSTALL` には DSi のインストールルートディレクトリのパスを設定して下さい。

必要なライブラリのビルド用の **makefile** が 5 つあります。例えば、標準 **C** ライブラリをリビルドするためには、以下の通りに行います。

```
> cd $(CWINSTALL) \ARM_EABI_Support\msl\MSL_C\MSL_ARM\Project
> make -f MSL_C.NITRO.mak clean
> make -f MSL_C.NITRO.mak
```

残りのライブラリも上記と同様にリビルドできます。

C++ ライブラリをリビルドする場合、カレントディレクトリを

```
$(CWINSTALL) \ARM_EABI_Support\msl\MSL_C++\MSL_ARM\Project
```

に変更し、`MSL_C++.NITRO.mak` ファイルを使用します。通常、プリコンパイルヘッダは使用されません。もし必要な場合は、`MSL_C++_Header.NITRO.mak` を使ってリビルドできます。

```
> cd $(CWINSTALL) \ARM_EABI_Support\msl\MSL_C++\MSL_ARM\Project
> make -f MSL_C++.NITRO.mak clean
> make -f MSL_C++.NITRO.mak
```

追加の **MSL** ライブラリをリビルドする場合、カレントディレクトリを

```
$(CWINSTALL)\ARM_EABI_Support\msl\MSL_Extras\MSL_ARM\Project
```

に変更し、`MSL_Extras.NITRO.mak` ファイルを使用します。

```
> cd $(CWINSTALL)\ARM_EABI_Support\msl\MSL_Extras\MSL_ARM\Project
> make -f MSL_Extras.NITRO.mak clean
> make -f MSL_Extras.NITRO.mak
```

最後に、ランタイム・ライブラリをリビルドする場合、カレントディレクトリを

```
$(CWINSTALL)\ARM_EABI_Support\Runtime\Runtime_ARM\Runtime_NITRO\Projects)
```

に変更し、`NITRO_Runtime.mak` を使用します。

```
> cd
$(CWINSTALL)\ARM_EABI_Support\Runtime\Runtime_ARM\Runtime_NITRO\Projects)
> make -f NITRO_Runtime.mak clean
> make -f NITRO_Runtime.mak
```

1.6 IDE

1.6.1 日本語を入力すると文字化けする

CodeWarrior IDE の編集メニューから「環境設定」を選択し、「フォント&タブ」設定パネルを確認してください。日本語を表示できるフォント（日本語環境なら **System** など）を指定する必要があります。

詳細については、「IDE User Guide」を参照してください。

1.6.2 外部エディタを使っている時に一時的に内部エディタでソースファイルを開く方法は？

外部エディタを使うように設定されている場合に、ブレークポイントを設定する等の為に一時的に内部エディタでソースファイルを開くには、下記の方法があります。

- 「内／外部エディタトグル」を使う。

IDE のツールバーの一番右にあるアイコンが押し込まれている状態の時は外部エディタで、押し込まれていない状態の時は内部エディタでファイルを開くようになります。

この機能へのキーバインディングはデフォルトでは **Ctrl + J** です。

- Alt キーを使う

プロジェクトウィンドウでファイル名をダブルクリック又は **Enter** で開く時に、**Alt** キーを押しながらダブルクリック又は **Enter** で内部エディタで開きます。

また、既に内部エディタで開いているソースファイル上から定義や宣言の場所を内部エディタで開くには、下記の手順で行います。

- 1) 変数/型宣言/関数/マクロ/クラス等上で右クリック
- 2) ポップアップメニューの「○○定義へ移動」もしくは「○○宣言へ移動」の上で **Ctrl** キーを押しながら左クリック又は **Enter**。

1.6.3 ツールバーにボタンを追加・削除する方法は？

ツールバーにボタンを追加するには下記の手順で行います。

- 1) CodeWarrior IDE の編集メニューから「コマンドとキーバインディング」を選択し、「IDE コマンドのカスタマイズ」ウィンドウを開く。
- 2) 「コマンド」ペイン又は「ツールバー項目」ペインの中からツールバーに追加したい機能を選び、その機能名の左にあるアイコン部分をツールバーの任意の位置にドラッグ&ドロップする。
(アイコンが無い機能はツールバーには追加できません。)

ツールバーからボタンを削除するには下記の手順で行います。

- 1) 削除したいボタンの位置で右クリック。
- 2) ポップアップメニューの「ツールバー項目を削除」を選択。

1.7 デバッガ

1.7.1 オーバーレイ部分のデバッグが正しく行なえない。

CodeWarrior のデバッガは TWL-SDK の「ROM ファイルシステム(FS)API」のオーバーレイ関連の関数に内部的なブレークポイントを設定して監視することによって、実行中のオーバーレイの状況を把握し、正しいソースファイルを表示するなどを行なっています。

この為、オーバーレイの管理に TWL-SDK の API は使用しない作りになっている場合は、CodeWarrior のデバッガではオーバーレイ部分のデバッグは正しく行なえません。

そのような状態の場合、デバッグ開始時に下記のような警告ダイアログが表示されます。

- 『Unable to find needed symbol "FS_StartOverlay" in ARM9 Elf file.
Overlay debugging will not work correctly.』
- 『Unable to find needed symbol "FS_UnloadOverlayImage" in ARM9 Elf file.
Overlay debugging will not work correctly.』

1.7.2 IDEで.nefファイルを読み込ませる時のデバッグ時にファイルをロックしないようにしたい

デフォルトの状態では、IDE で.nef ファイルを読み込ませるとファイルをロックし続ける為、コマンドラインでの **make** が出来なくなります。

ファイルをロックしないようにするには、ターゲットの設定ウィンドウの「デバッガ設定」パネルの「実行後もシンボルのキャッシュを維持する」のチェックボックスをオフにして下さい。

尚、nef ファイルを読み込ませた直後は既に nef ファイルがロックされてしまっている為、この設定を行なっても **make** 出来ませんが、一度デバッガを実行させた後はロックが解除されます。

2 TIPS

2.1 C/C++プログラミングに関するTIPS

2.1.1 パックした構造体のメンバにアクセスする方法

int 型の変数にアクセスする場合は 4 バイトアクセス命令が使われる為、通常では int 型変数は 4 バイト境界にアラインされている必要があります。

構造体をパックした結果、前方に char 型や short 型の変数が有る為に int 型変数のアラインメントが 4 バイト境界から外れてしまう場合、アクセスする為には int 型変数のアラインメントが 4 バイト境界の位置に来るように構造体の先頭アドレスをずらす必要があります。

例)

```
typedef struct {
    u8  dstMac[6] __attribute__((aligned(1)));
    u8  srcMac[6] __attribute__((aligned(1)));
    u16 type      __attribute__((aligned(1)));
    u32 n2        __attribute__((aligned(1)));
} TEST_PACKET;

{
    TEST_PACKET packet_a;
    TEST_PACKET packet_b;
```

このような構造体の場合、変数"n2"に正しくアクセスするには構造体の先頭アドレスを 12 バイト境界に合わせる必要があります。

しかし、__attribute__((aligned(x))) で指定出来る値 x は 2 の階乗数 (1, 2, 4, 8, 16, 32, 64, etc...) のみです。

このような場合、下記のような方法があります。

```
char  packet_data_a[sizeof(u16) + sizeof(TEST_PACKET)] __attribute__((aligned(4)));
char  packet_data_b[sizeof(u16) + sizeof(TEST_PACKET)] __attribute__((aligned(4)));
```

```
TEST_PACKET* packet_a = (TEST_PACKET*)&packet_data_a[sizeof(u16)];  
TEST_PACKET* packet_b = (TEST_PACKET*)&packet_data_b[sizeof(u16)];
```

2.2 Windows環境に関連するTIPS

2.2.1 フォルダのアイコンを変更するとプロジェクトが開けなくなる

Windows XP からフォルダのアイコンを変更出来るようになっていますが、Windows XP 以降ではフォルダ自身の「読み取り専用属性」と「システム属性」をフォルダのカスタマイズフラグとして利用する動きになっています。

(Microsoft 社の公開情報 : <http://support.microsoft.com/kb/326549/ja>)

この為、プロジェクトファイルがあるフォルダのアイコンが変更されると、CodeWarrior はそのプロジェクトが置いてある場所を読み取り専用領域と判断し、プロジェクトを開けなくなってしまいます。 プロジェクトファイルがあるフォルダはアイコンの変更等のカスタマイズは行なわないで下さい。

もし、このような状態になってしまいましたら、コマンドプロンプトで

```
attrib -R -S [フォルダ名]
```

を行なって、フォルダの属性を戻して下さい。

3 その他

(現在、その他の情報は有りません。)