RX100 シリーズ

RX100 上で CoreMark を実行する方法

R01AN1543JU0100
Rev.1.00
2014.02.28

要旨

本アプリケーションノートでは、EEMBC の CoreMark ベンチマークの入手、設定、および実行に必要な手順を説明しています。例として、本アプリケーションノートを通してボード RSKRX111 がフラットフォームとして使われており、そのセットアップの方法が示されています。また、IAR Embedded Workbench IDE が IAR RX Toolchain とともに使用されています。

EEMBC の CoreMark ベンチマークについての詳細については、次のリンクを参照してください。

http://www.coremark.org/

動作確認デバイス

このドキュメントでは RX111 が例として使用されていますが、他の MCU に対しても同じ方法で CoreMark を適切に設定できます。

目次

1. 概要

特定のアプリケーションに適した MCU を探す際には、MCU が要件を満たす十分な処理能力を持っていることを確認する必要があります。利用できる多くのベンチマークがありますが、最も広く知られているのはおそらく Dhrystone と言えます。Dhrystone には問題が内在しており、その内容に関しては CoreMark Software に付属するドキュメント Coremark-requirements.doc の Background セクションに記載されています。この問題点に対処し、シンプルでオープンソースのベンチマークを提供するために、EEMBC は CoreMark を開発しました。

2. CoreMarkベンチマークの入手方法

CoreMark Software は CoreMark のウェブサイトから無償でダウンロードすることができます。次のウェブサイトからスタートします。
http://www.coremark.org/download.

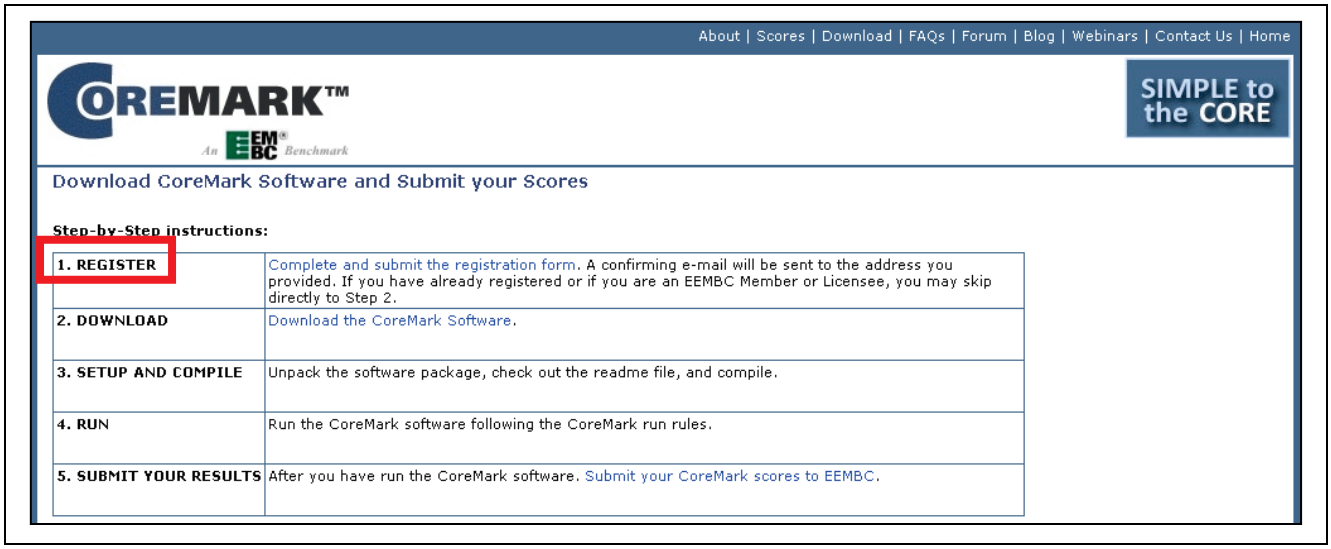

図 1　CoreMark の登録とダウンロード

ソフトウェアをダウンロードするために、ユーザは最初に EEMBC に登録する必要があります。これは上記で参照したウェブページの REGISTER セクションの指示に従って実行します。登録の手順が終了すると、DOWNLOAD セクションの指示に従って、Download CoreMark Software ウェブページを表示します。このウェブページから図 2 に示されているように CoreMark Software をダウンロードします。このダウンロードには CoreMark のドキュメントも含まれています。

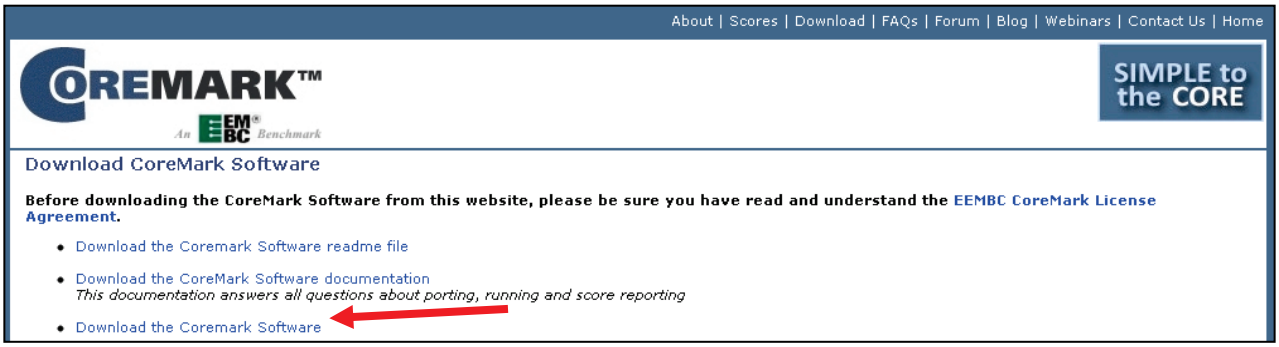

図 2　CoreMark Software とドキュメントのダウンロード

3. CoreMark IARプロジェクトの作成

このセクションでは、IAR IDE が既にインストールされていると想定しています。インストールされていない場合には、IAR ウェブサイトから最新の IAR RX Toolchain をダウンロードしてインストールしてください。RX100 は v2.14 以降でサポートされています。

CoreMark のソースファイルと IAR RX Toolchain が用意できたら、IAR ワークスペースを作成できます。Embedded Workbench を開き、Windows にワークスペースのための新しいフォルダを作成することから始めます。

Embedded Workbench で、Project メニューから新しいプロジェクトを作成します。以下の図のように RX Toolchain を選び、Empty Project テンプレートを使用します。

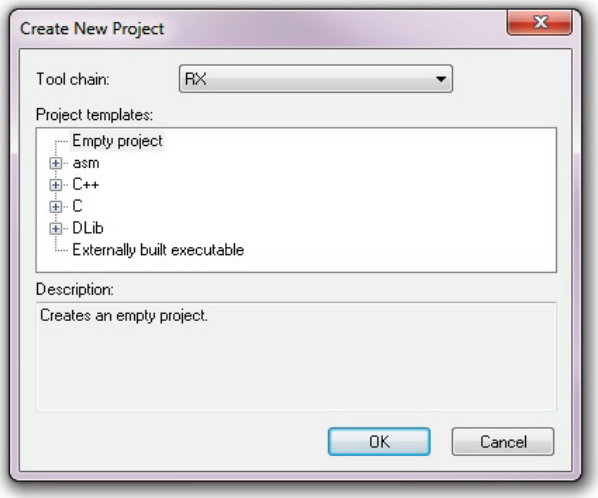

図 3　新しい IAR プロジェクトの作成

OK をクリックした後、プロジェクト名を入力し、前に作成したワークスペースフォルダに.ewp ファイルとして保存します。

Embedded Workbench でプロジェクトを右クリックし、オプションを選択します。Category で General Options を選択し、次に Target タブの下で RX111 group >> R5F1115 を選びます。これが RX111 RSK の MCU です。

Category の下で Debugger を選択し、Setup タブの下で Driver 項目を Simulator から E1/E20 emulator に変更して OK をクリックします。

4. CoreMarkソースファイルの追加

IAR プロジェクトのセットアップが終わると、CoreMark ソースファイルを追加することができます。Windows で新しいフォルダをワークスペースフォルダ内に作成し、その名前を CoreMark にします。前にダウンロードした CoreMark Software のパッケージを表示し、次のファイルを先ほど作成した CoreMark フォルダにコピーします。

- core_list_join.c
- core_main.c
- core_matrix.c
- core_state.c
- core_util.c
- coremark.h
- simple/core_portme.c
- simple/core_portme.h

IAR プロジェクトへのファイルの追加は Project >> Add Files から行えます。CoreMark フォルダからプロジェクトにファイルを追加した後、これらをグループ化することもできます。このためには Embedded Workbench で左側ペインのどこかを右クリックし、Add >> Add Group を選択して、その名前を CoreMark と入力します。フォルダの作成後、CoreMark ファイルをこのフォルダの中にドラッグします。

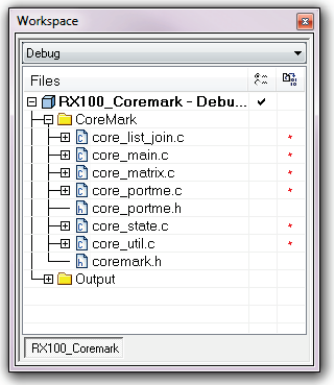

図 4　CoreMark ファイルの追加とグループ化

5.　ボードサポートファイルの追加

CoreMark Software とともに、ユーザは使用する開発システムをサポートするファイルを追加する必要があります。このアプリケーションノートでは RX111 RSK を使用しており、このボードサポートファイルは、このアプリケーションノートに付属するボードサポートパッケージの bsp フォルダ内に含まれています。Windows で bsp フォルダをワークスペースフォルダにコピーし、Embedded Workbench 内のプロジェクトにこれらを追加します。Project >> Add Files をクリックしてプロジェクトにファイルを追加します。

- rx111_mtu.c と rx111_mtu.h
  — MTU タイマ周辺モジュールを使用するためのコードが含まれます。これはパフォーマンス測定に使われます。
- serial_printf.c と serial_printf.h
  — printf()をシリアルポートにリダイレクトする低水準関数に置き換わります。SCI 周辺モジュールの初期化も行います。
- hardware_setup.c
  — RX111 MCU のクロックのセットアップを行います。これはプロジェクトジェネレータで作成するデフォルトの hardware_setup.c ファイルに置き換わります。このため、ユーザは最初に元のファイルを削除し、新しいファイルを追加する必要があります。単に新しいファイルを古いファイルに上書きすることもできます。
- low_level_init.c
  — IAR が必要とするリセット処理関数。
- iorx111.h
  — IAR の RX111 ヘッダファイル。

以上により、プロジェクトウィンドウは次のようになります。

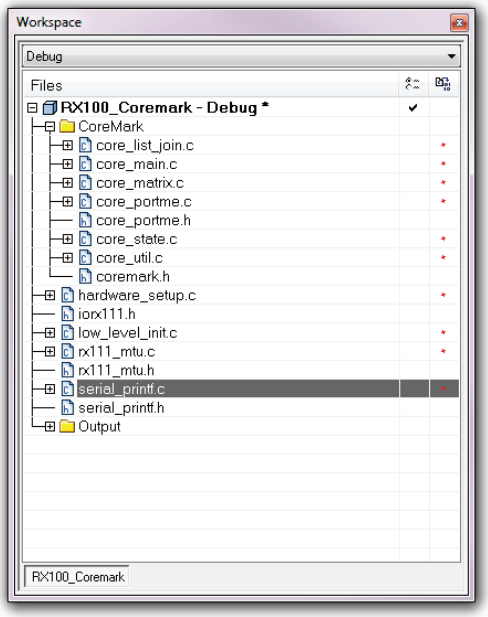

図 5　bsp ファイルの追加

File >> Save Workspace を選択し、名前を付けてワークスペースを Windows のワークスペースフォルダに保存します。

6.　CoreMarkの設定

CoreMark ソースの追加後、CoreMark の設定が必要になります。設定は core_portme.c と core_portme.h ファイルで行います。これらのファイルには十分なコメントが含まれていますが、より詳細な情報が必要な場合は coremark.org でダウンロードしたパッケージに含まれる CoreMark の資料を参照してください。

6.1　システム定義とデータ型

以下の情報は用意されているサンプルプロジェクトで使用する設定の詳細です。core_portme.h ファイルを開きます。このファイルには、開発するシステムの要件に合わせて変更が必要な定義のリストが含まれます。これらの定義は RX111 の設定例として指定する値とともに以下に示されています。

- HAS_FLOAT
  — RX111 は FPU を内蔵していませんので 0 を設定します。ただし、これを 1 に設定した場合でも結果を出力する際の分解能が向上するに過ぎず、最終的な結果に顕著な相違をもたらすものではありません。
- HAS_TIME_H
  — 時刻機能が実装されていませんので、0 に設定します。
- USE_CLOCK
  — 時刻機能が実装されていませんので、0 に設定します。
- HAS_STDIO
  — IAR Toolchain は stdio.h をサポートしていますので 1 に設定します。
- HAS_PRINTF
  — IAR Toolchain は printf()をサポートしていますので 1 に設定します。
- MAIN_HAS_NOARGC
  — main()の引数は使用されませんので 1 に設定します。

core_portme.h で定義されるデータ型はデフォルトのまま使用できます。ユーザは size_t 型の定義（typedef）が定義されるよう、stddef.h をファイルの先頭でインクルードする必要があります。

```
#include <stddef.h>
```

time.h がサポートされていないので、clock_t 型は unsigned long に変更する必要があります。変更が必要な個所は 2 個所です。core_portme.h 内の以下のコードを以下の表のコードのように変更します。

元のコード	変更後のコード
#include <time.h> typedef clock_t CORE_TICKS;	**#if 0** #include <time.h> typedef clock_t CORE_TICKS; **#else** **typedef unsigned long CORE_TICKS;** **#endif**

core_portme.c 内の以下のコードを以下の表のコードのように変更します。

元のコード	変更後のコード
#define CORETIMETYPE clock_t	#define CORETIMETYPE **unsigned long**

また CLOCKS_PER_SEC の定義が抜けています。これを修正するため core_portme.c で以下のように変更します。

元のコード	変更後のコード
#define NSECS_PER_SEC CLOCKS_PER_SEC	**/* Timer clock frequency: PCLK=32MHz */** #define NSECS_PER_SEC 32000000

6.2　実行時設定の定義

CoreMark の実行に関して設定する必要のある定義が 2 つあります。1 つ目の定義は ITERATIONS で、CoreMark が実行を繰り返す回数を定めています。2 つ目の定義は、FLAGS_STR で、長さ 0 の文字列が設定されています。これらの定義は core_portme.h 内に配置することができ、またツールチェインのオプションとして定義することもできます。今回の例では、これらの定義はツールチェインのオプションとして設定します。

また、プロジェクトのインクルードパスを設定する必要があります。

これは次の手順で行います。

1. Embedded Workbench でプロジェクトを右クリックし、options を選択します。
2. Category の下で C/C++ Compiler を選択します。
3. Preprocessor タブが選択されていることを確認します。
4. Additional Include Directories ボックスで$PROJ_DIR$¥bsp と$PROJ_DIR$¥CoreMark をそれぞれ別の行に入力します。
5. Defined Symbols ボックスで次の内容をそれぞれ別の行に入力します。
    a. ITERATIONS=2000
    b. FLAGS_STR=""
    c. COMPILER_VERSION="EWRX V.2.41.1"
    新しいバージョンのコンパイラを使用している場合は、これにあわせて文字列を変更します。
6. 入力後の画面は次のようになります。

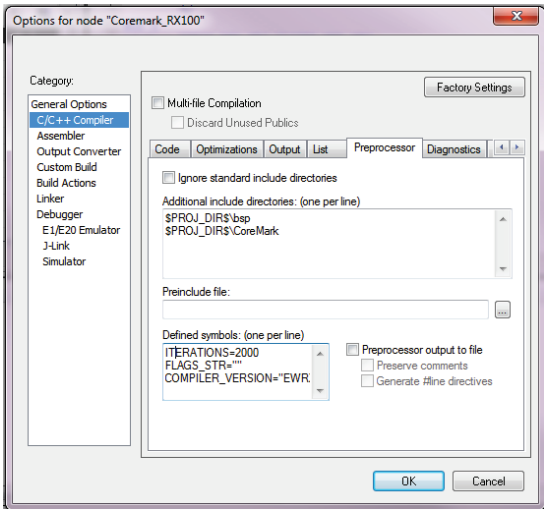

図 6　定義の設定

7. OK をクリックします。

6.3　スタックサイズの拡張

CoreMark では、デフォルトのメモリ配置としてスタックを使用しています。IAR RX プロジェクトジェネレータを使った場合、デフォルトでユーザスタックと割り込みスタックに割り当てられるバイト数はそれぞれ 256 バイトです。スタックオーバフローを避けるためにユーザスタックのサイズを増やす必要があります。スタックサイズの拡張は次の手順で行います。

1. Embedded Workbench でプロジェクトを右クリックし、options を選択します。
2. Category の下で General Options を選択します。
3. Stack/Heap タブが選択されていることを確認します。
4. Supervisor mode stack を 0x1000 に設定します。
5. Heap size に 0 をセットします。
6. OK をクリックします。

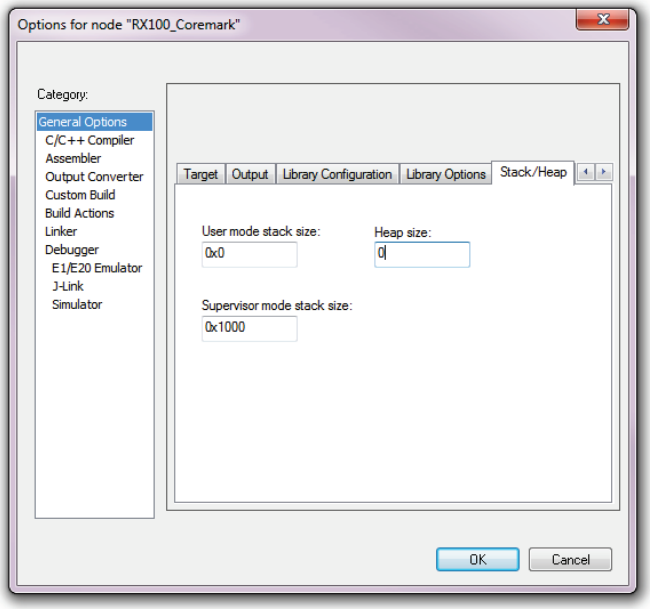

図 7　スタックサイズの拡張

6.4　実行サイクル数の取得と結果の出力

CoreMark ベンチマークからパフォーマンス結果を得るためには、CoreMark プログラムを実行するために必要となったサイクル数を確認する必要があります。確認は core_portme.c 内で start_time()と stop_time()関数を使用して行います。今回の例では、サイクル数をカウントするために RX111 の MTU タイマを使用しました。MTU の各チャネルは 16 ビットで、2 つのチャネルをカスケード接続して 32 ビットタイマを作ることができます。MTU が最高速度の 32 MHz で動作する場合は、最大時間は約 134 秒となります。CoreMark の資料では、ベンチマークは少なくとも 10 秒は実行する必要があると記載されています。つまり、更なるクロックの分周は必要ないということになります。分解能の高いタイマと多数の CoreMark の繰り返しで結果のサイクルカウントをより高精度なものとすることができます。MTU の初期化、開始、および停止を行う関数は rx111_mtu.c で提供されており、このファイルはこのアプリケーションノートとともに提供されています。

CoreMark の計測セクションが終了した後、結果が計算され printf()関数を使用して出力されます。今回の例では、printf()の出力は開発ボードのシリアルポートに接続されている SCI1 に送られます。printf()の出力先はユーザが独自の putchar()関数と getchar()関数を記述することでリダイレクトされます。これらの関数と SCI 周辺モジュールの初期化のための関数はこのアプリケーションノートとともに serial_printf.c として含まれています。

表 1 は MTU タイマの使用と SCI 周辺モジュールの初期化のために core_portme.c で必要となる変更です。

表 1　core_portme.c の変更

元のコード	変更後のコード
#include <stdlib.h> #include "coremark.h"	#include <stdlib.h> **#include “rx111_mtu.h”** **#include “serial_printf.h”** #include “coremark.h”
#define CORETIMETYPE unsigned long #define GETMYTIME(_t) (*_t=clock())	**#if 0** #define GETMYTIME(_t) (*_t=clock()) **#endif**

void start_time(void) {
　　GETMYTIME(&start_time_val);
}
	void start_time(void) {
	　　start_time_val = timer_start();
	}

void stop_time(void) {
　　GETMYTIME(&stop_time_val);
}
	void stop_time(void) {
	　　stop_time_val = timer_stop();
	}

void portable_init(core_portable *p, int *argc, char *argv[])
{
　　...
　　p->portable_id=1;
}
	void portable_init(core_portable *p, int *argc, char *argv[])
	{
	　　...
	　　p->portable_id=1;
	　　timer_init();
	　　sci_init();
	}

7.　CoreMarkの最適化

最適なベンチマークの結果を得るためには、最適化を有効にします。このセクションでは、セクション 9 にある CoreMark 値を得るために使われたのと同じ最適化を行う方法を示しています。

7.1　コンパイラの最適化

1. Embedded Workbench でプロジェクトを右クリックし options を選びます。
2. Category の下で C/C++ Compiler を選択します。
3. Code タブが選択されていることを確認します。
4. Align functions に 4 を設定します。
5. Optimizations タブを選択します。
6. Level を High に設定し、Balanced から Speed に変更します。
7. No Size Constraints オプションをクリックします。
8. OK をクリックします。

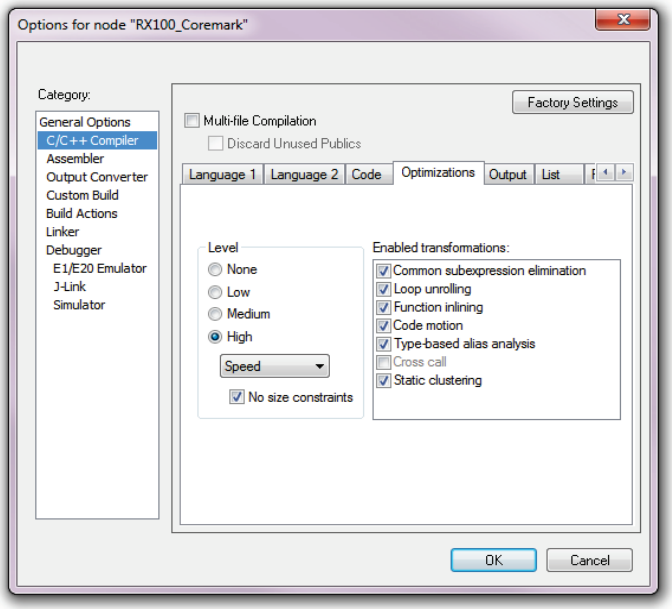

図 8　コンパイラで最適化を有効に設定

7.2　リンカの最適化

1. Embedded Workbench でプロジェクトを右クリックし options を選択します。
2. Category の下で Linker を選択します。
3. Optimizations タブが選択されていること確認選択します。
4. Inline small routines オプションをチェックします。

8. CoreMark Benchmarkの実行

これまでのセクションで説明されている変更を行った後に、Project >> Make を選択してプロジェクトをビルドします。この処理の終了後に開発ボードを接続し、コードを実行します。この手順は次の通りです。

1. 付属のケーブルを使って PC に E1 デバッガを接続します。
2. 開発ボードを E1 デバッガに接続します。
3. E1/E20 Emulator >> Hardware Setup を選択してエミュレータセットアップウィンドウを表示します。
    a. Input clock を 16.0000 MHz に変更します。
    b. 開発ボードに外部電源が接続されていない場合は Power target from emulator オプションにチェックし、外部電源が接続されていれば、このオプションのチェックを外します。開発ボードで内部電源またはエミュレータ電源を使用する設定の詳細については、開発ボードの回路図をご覧ください。

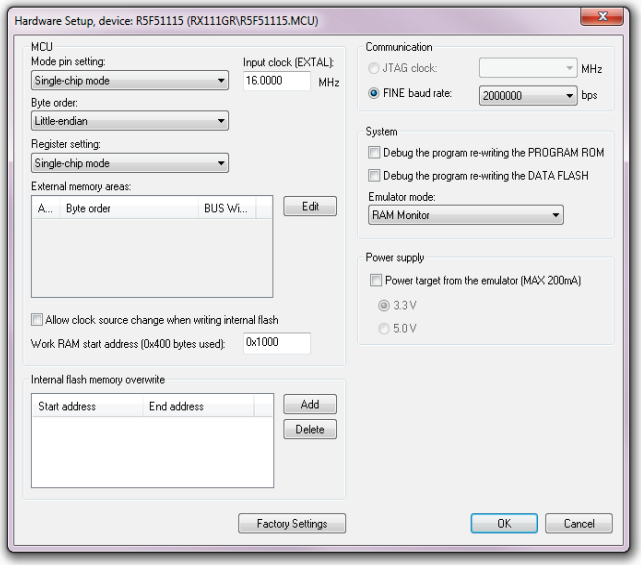

図 9　E1 デバッグセッションの設定

    c. OK をクリックします。
4. F7 を押してプロジェクトをビルドします。
5. Ctrl+D を押してデバッグを開始します。
6. 開発ボードと PC をシリアルケーブルで接続します。
7. 端末プログラムを次の設定で起動します。
    a. 115200 ボー
    b. 8 ビットデータ
    c. 1 ストップビット
    d. パリティなし
    e. フロー制御なし
8. Debug >> Go をクリックします。
9. 約 20 秒経過すると結果が端末のウィンドウに表示されます。出力の例はセクション 9 に示されています。

9. RX111 におけるベンチマークの結果

このセクションでは、このドキュメントに従ってビルドされたプロジェクトの例のベンチマーク結果を示します。使用したシステムは RSK RX111 です。すべての RX100 デバイスでは同じコアが共通に使用されているため、これらの結果は同じになります。

CoreMark 出力
2K performance run parameters for coremark. CoreMark Size : 666 Total ticks : 650799876 Total time (secs): 20.337496 Iterations/Sec : 98.340520 Iterations : 2000 Compiler version : EWRX V.2.41.1 Compiler flags : Memory location : STACK seedcrc : 0xe9f5 [0]crclist : 0xe714 [0]crcmatrix : 0x1fd7 [0]crcstate : 0x8e3a [0]crcfinal : 0x4983 Correct operation validated. See readme.txt for run and reporting rules. CoreMark 1.0 : 98.340520 / EWRX V.2.41.1 / STACK

複数の MCU を検討する際に、多くのユーザが関心を持つのは MCU のメガヘルツあたりの CoreMark 値です。CoreMark 値が実際の処理能力を示すのに対して、CoreMark/MHz はコアの効率を示しています。もし 100MHz、80mA で動作する MCU-A が 50MHz、40mA で動作する MCU-B の処理能力に相当するのであれば、ほとんどのユーザは MCU-B を選びます。

CoreMark/MHz を算出するには、ベンチマーク実行時に使われたクロック速度で CoreMark 値を割ります。この例では RX111 は 32MHz で動作しています。

CoreMark/MHz = CoreMark スコア／クロック速度

CoreMark/MHz = 98.340520／32MHz

CoreMark/MHz = 3.073

【注】 セクション 6.1 にあるように HAS_FLOAT を 0 に設定している場合は、上記と若干異なる結果（CoreMark/MHz = 3.125）が得られます。これは最終結果を出力する際の分解能の低下によるものです。

10. まとめ

ルネサスが提供する CoreMark/MHz 値をユーザが再現するための詳細な方法の説明を行いました。どのようなベンチマークであっても、その結果だけに基づいて MCU を選ぶのは望ましくありません。むしろ、CoreMark は MCU のコア処理能力に対するひとつの一般的な指標として使用してください。例えば、MCU-A の CoreMark 値が 95 で MCU-B の CoreMark 値が 90 のとき、これから得られる結論は、両方の MCU は同程度のコア能力を有しており、一方が要求を満たすのに十分な能力を持っていれば他方も同様の能力を持っているということになります。この例を引き続き使うと、MCU-A では動作するけれども MCU-B では性能が不十分なため動作しないという場合では、選択すべき MCU は明らかであると言えます。この場合には、より大きな処理能力を持つ一クラス上の MCU に移行することが適切です。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.02.28	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理

【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。

CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

【注意】電源投入時は，製品の状態は不定です。

電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。

外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。

同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. リザーブアドレスのアクセス禁止

【注意】リザーブアドレスのアクセスを禁止します。

アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

4. クロックについて

【注意】リセット時は、クロックが安定した後、リセットを解除してください。

プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。

リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

5. 製品間の相違について

【注意】型名の異なる製品に変更する場合は、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。

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 エレクトロニクス株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

■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/