統合開発環境 e² studio

CMake を使用して CC-RX 用ソース・ファイルをビルドする

はじめに

CMake は、プラットフォームとコンパイラに依存しない設定ファイルを使用して、ソフトウェアのビルドプロセスを制御し自動化するためのツールです。

このドキュメントでは、CMake を使用して、RX ファミリ用 C/C++コンパイラパッケージ CC-RX に対応したプロジェクトのビルド環境（make によるビルド）を構築する方法について説明します。

目次

1. 概要

CMake は、プラットフォームとコンパイラに依存しない設定ファイルを使用して、ソフトウェアのビルドプロセスを制御し自動化するためのツールです。

コンパイラのオプション、つまりビルドに関する設定項目の仕様、および、それを格納する設定ファイルのフォーマットは統一されていません。そのため、使用する開発環境に合わせてプロジェクト・ファイルを使い分ける必要があります。

CMake は、CMakeLists.txt という設定ファイルを入力ファイルとして、CMake がサポートする任意のプロジェクト・ファイルを生成することができます。したがって、どの開発環境においても

1. CMakeLists.txt を作成し、CMake で任意のプロジェクト・ファイルを生成する。
2. 生成したプロジェクト・ファイルを使用してビルドを実行する。

という 2 ステップでビルドを実行することができます。

CMake がサポートする開発環境（プロジェクト・ファイル）の例を以下に示します。

- Xcode
- Visual Studio
- Ninja
- Make

このドキュメントでは、CMake を使用して、RX ファミリ用 C/C++コンパイラパッケージ CC-RX に対応したプロジェクトのビルド環境（make によるビルド）を構築する方法について説明します。

RL78 ファミリ用 C コンパイラパッケージ CC-RL に対応したプロジェクトの場合は、ビルドオプションに違いがありますが、同様の手順でビルド環境を構築することが可能です。

注意事項：

- クロスコンパイラ（PC 上の OS とは異なるプログラムをコンパイルする）を使用する場合は、コンパイラ、アセンブラ、リンカなどのオプション設定に関する情報をツールチェーン・ファイル（*.cmake）として作成する必要があります。
  CC-RX はクロスコンパイラですので、このファイルの作成が必要になります。
- クロスコンパイラをサポートする CMake は、V2.6.0 およびそれ以降になります。

2. 環境構築

下記ツールを WEB サイトからダウンロードしてインストールしてください。

- RX ファミリ用 C/C++コンパイラパッケージ CC-RX：
  https://www.renesas.com/software-tool/cc-compiler-package-rx-family#downloads
  （この例では、V3.03.00 を使用します）
- CMake：
  Download | CMake
  （この例では、V3.20.3 を使用します）
- Windows 版 GNU make（e² studio をインストールしている場合は、e² studio の make で代用することが可能です）：
  Make for Windows (sourceforge.net)
  （この例では、V3.81 を使用します）

次に、CC-RX、CMake および make をコマンドプロンプト上で実行できるように、下記手順に従って環境変数を設定してください。

1. 下記フォルダが環境変数 PATH にない場合は、追加してください。
   - <CC-RX のインストール・フォルダ>\bin
   - <CMake のインストール・フォルダ>\bin
   - <GNU make のインストール・フォルダ>
2. 次に CC-RX の実行に必要な以下の環境変数を追加してください。
   - BIN_RX
     変数名： BIN_RX
     値：　　<CC-RX のインストール・フォルダ>\bin
   - INC_RX
     変数名： INC_RX
     値：　　<CC-RX のインストール・フォルダ>\include

3. ファイル構成

CMake を使用する場合の CC-RX に対応したプロジェクトのビルド環境に必要なファイル構成です。この例を参考にプロジェクトを構築してください。

<project folder>	:
│ ccrx.cmake	: CC-RX コンパイラ用ツールチェーン・ファイル
│ CMakeLists.txt	: CMake 構成ファイル
│ Linker.tmp	: Linker 用サブコマンド・ファイル
├\generate	: ソース・ファイル・フォルダ
│ dbsct.c	: （ソース・ファイルのフォルダ構成は任意です）
│ hwsetup.c	:
│ intprg.c	:
│ iodefine.h	:
│ lowlvl.src	:
│ lowsrc.c	:
│ lowsrc.h	:
│ resetprg.c	:
│ sbrk.c	:
│ sbrk.h	:
│ stacksct.h	:
│ test.h	:
│ typedefine.h	:
│ vect.h	:
│ vecttbl.c	:
├\src	: ソース・ファイル・フォルダ
│ main.c	: （ソース・ファイルのフォルダ構成は任意です）
└_builds	: ビルドにより出力されるファイルの保存フォルダ

ビルド対象であるソース・ファイル以外に下記ファイルを作成してください。

- CC-RX 用ツールチェーン・ファイル
  ccrx.cmake
- CMake 構成ファイル
  CMakeLists.txt
- Linker 用サブコマンド・ファイル
  Linker.tmp

各ファイルの内容について、以降の章で具体的に説明します。

4. 作成が必要なファイル

4.1 CMake CC-RX 用ツールチェーン・ファイル

CC-RX を使用したクロス・コンパイルに必要なコンパイラおよびユーティリティに関する情報を定義したツールチェーン・ファイルです。この内容をベースにファイルを作成してください。

#	ccrx.cmake
1	### BEGIN CMAKE_TOOLCHAIN_FILE
2	# "Generic" is used when cross compiling
3	set(CMAKE_SYSTEM_NAME Generic)
4	# Optional, this one not so much
5	set(CMAKE_SYSTEM_VERSION 1)
6	# Set the environment root directory
7	# It can be used to specify the target environment location
8	# e.g compiler's location. This variable is most useful when cross-compiling.
9	set(CMAKE_FIND_ROOT_PATH "C:/Program Files (x86)/Renesas/RX/3_3_0/bin")
10	# CMake variables for compiler, assembler, native build system
11	# Specify the C compiler
12	set(CMAKE_C_COMPILER ccrx)
13	set(CMAKE_C_COMPILER_ID RXC)
14	set(CMAKE_C_COMPILER_ID_RUN TRUE)
15	set(CMAKE_C_COMPILER_FORCED TRUE)
16	# Specify the CPP compiler
17	set(CMAKE_CXX_COMPILER ccrx)
18	set(CMAKE_CXX_COMPILER_ID RXC)
19	set(CMAKE_CXX_COMPILER_ID_RUN TRUE)
20	set(CMAKE_CXX_COMPILER_FORCED TRUE)
21	# Specify the ASM compiler
22	set(CMAKE_ASM_COMPILER asrx)
23	set(CMAKE_ASM_COMPILER_ID RXA)
24	set(CMAKE_ASM_COMPILER_ID_RUN TRUE)
25	set(CMAKE_ASM_COMPILER_FORCED TRUE)
26	# Specify the linker
27	set(CMAKE_LINKER rlink)
28	# Specify options for command
29	set(COMMON_OPTIONS "-isa=rxv2 -fpu")
30	set(CMAKE_C_FLAGS "${COMMON_OPTIONS} -lang=c -include=\"../generate\" -define=DEBUG_CONSOLE -utf8 -outcode=utf8 -nomessage -debug -nologo")
31	set(CMAKE_CXX_FLAGS "${COMMON_OPTIONS} -lang=cpp -include=\"../generate\" -define=DEBUG_CONSOLE -utf8 -outcode=utf8 -nomessage -debug -nologo")
32	set(CMAKE_ASM_FLAGS "${COMMON_OPTIONS} -include=\"../generate\" -utf8 -debug -nologo")
33	set(CMAKE_EXE_LINKER_FLAGS "-subcommand=../Linker.tmp")
34	# Specify the make
35	set(CMAKE_MAKE_PROGRAM make)
36	### END CMAKE_TOOLCHAIN_FILE

- #9：<CC-RX のインストール・フォルダ>\bin フォルダを記載してください。
- #29：コンパイラ、およびアセンブラ双方に指定する共通オプションを記載してください。
- #30～31：コンパイラに指定するオプションを記載してください。
- #32：アセンブラに指定するオプションを記載してください。
- #33：リンカに指定するオプションは、「Linker.tmp」ファイルに記載してください。

オプションのパラメータにダブルクォーテーション（"）文字を指定する場合は、「\」文字をダブルクォーテーション文字の前に付加してください。

4.2　CMake 構成ファイル（CMakeLists.txt）

アセンブラ、コンパイラを実行して「.obj」ファイルを生成し、次にリンカで「.abs」ファイルを生成する場合の CMake 構成ファイルを以下に示します。この内容をベースにファイルを作成してください。

#	CMakeLists.txt
1	cmake_minimum_required(VERSION 3.16)
2	# set the project name
3	project(CCRX_Project)
4	enable_language(C ASM)
5	########## Specify the C/C++/ASM compiler and linker commands
6	set(CMAKE_C_COMPILE_OBJECT "\"${CMAKE_C_COMPILER}\" ${CMAKE_C_FLAGS} -output=obj=<OBJECT> <SOURCE>")
7	set(CMAKE_CXX_COMPILE_OBJECT "\"${CMAKE_CXX_COMPILER}\" ${CMAKE_CXX_FLAGS} -output=obj=<OBJECT> <SOURCE>")
8	set(CMAKE_ASM_COMPILE_OBJECT "\"${CMAKE_ASM_COMPILER}\" ${CMAKE_ASM_FLAGS} -output=<OBJECT> <SOURCE>")
9	set(CMAKE_C_LINK_EXECUTABLE "\"${CMAKE_LINKER}\" ${CMAKE_EXE_LINKER_FLAGS} <OBJECTS> -output=<TARGET>.abs")
10	set(CMAKE_CXX_LINK_EXECUTABLE "\"${CMAKE_LINKER}\" ${CMAKE_EXE_LINKER_FLAGS} <OBJECTS> -output=<TARGET>.abs")
11	########## Set C/C++ Source language
12	file(GLOB_RECURSE SOURCE ${CMAKE_CURRENT_SOURCE_DIR} "*.cpp" "*.c")
13	########## Set ASM Source language
14	file(GLOB_RECURSE src_asm ${CMAKE_CURRENT_SOURCE_DIR} "*.asm" "*.s" "*.src")
15	foreach(X IN ITEMS ${src_asm})
16	set_source_files_properties(${X} PROPERTIES LANGUAGE ASM)
17	endforeach()
18	########## Add function to scan and generate dependency files
19	function(dep_scan out_objects)
20	string(REPLACE " -" ";-" DEPENDENCY_C_OPTIONS ${CMAKE_C_FLAGS})
21	string(REPLACE " -" ";-" DEPENDENCY_CPP_OPTIONS ${CMAKE_CXX_FLAGS})
22	string(REPLACE " -" ";-" DEPENDENCY_ASM_OPTIONS ${CMAKE_ASM_FLAGS})
23	set(result)
24	foreach(in_f ${ARGN})
25	get_filename_component(ext ${in_f} LAST_EXT)
26	file(RELATIVE_PATH rel_f ${CMAKE_CURRENT_SOURCE_DIR} ${in_f})
27	set(depend_f "CMakeFiles/${PROJECT_NAME}.dir/${rel_f}.d")
28	set(obj_f "CMakeFiles/${PROJECT_NAME}.dir/${rel_f}.obj")
29	if(${ext} STREQUAL ".c")
30	add_custom_command (
31	COMMENT "Generate dependency: ${depend_f}"
32	OUTPUT ${depend_f}
33	COMMAND ${CMAKE_C_COMPILER} ${DEPENDENCY_C_OPTIONS} -MM -MP -output=dep=${depend_f} -MT=${obj_f} -MT=${depend_f} ${in_f}
34	DEPENDS "${in_f}"
35	VERBATIM
36	)
37	elseif(${ext} STREQUAL ".cpp")
38	add_custom_command (
39	COMMENT "Generate dependency: ${depend_f}"
40	OUTPUT ${depend_f}
41	COMMAND ${CMAKE_CXX_COMPILER} ${DEPENDENCY_CPP_OPTIONS} -MM -MP -output=dep=${depend_f} -MT=${obj_f} -MT=${depend_f} ${in_f}
42	DEPENDS "${in_f}"
43	VERBATIM
44	)
45	elseif((${ext} STREQUAL ".asm") OR (${ext} STREQUAL ".s") OR (${ext} STREQUAL ".src"))
46	add_custom_command (

```
47          COMMENT "Generate dependency: ${depend_f}"
48          OUTPUT ${depend_f}
49          COMMAND ${CMAKE_ASM_COMPILER} ${DEPENDENCY_ASM_OPTIONS} -MM -MP -output=${depend_f
    } -MF=${depend_f} -MT=${obj_f} -MT=${depend_f} ${in_f}
50          DEPENDS "${in_f}"
51          VERBATIM
52        )
53      endif()
54      list(APPEND result ${depend_f})
55    endforeach()
56    set(${out_objects} "${result}" PARENT_SCOPE)
57  endfunction()
58  ########## Call dep_scan
59  dep_scan(d_files ${SOURCE} ${src_asm})
60  ########## Add the executable
61  add_executable(${PROJECT_NAME} ${SOURCE} ${src_asm} ${d_files})
62  ########## Execute Library Generator
63  # Command rule for build .lib file
64  set(RENESAS_LIBRARY_GENERATOR "lbgrx.exe")
65  string(REPLACE " -" ";-" LIBRARY_GENERATOR_FLAGS "${COMMON_OPTIONS} -lang=c -head=runtime,
    stdio,stdlib,string,new -nologo")
66  add_custom_command(
67    TARGET ${PROJECT_NAME}
68    PRE_LINK
69    COMMAND ${RENESAS_LIBRARY_GENERATOR} ${LIBRARY_GENERATOR_FLAGS} -output=${PROJECT_NAME}.
    lib
70    COMMENT "Library Generator:"
71    VERBATIM
72  )
73  ########## Build finished
74  add_custom_command(
75    TARGET ${PROJECT_NAME}
76    COMMAND ${CMAKE_COMMAND} -E cmake_echo_color --cyan "Build Finished."
77  )
78  ########## Include dependencies files
79  # This process should be put at the end
80  foreach(in_f ${d_files})
81    set(include_command ${include_command}\n-include ${in_f})
82  endforeach()
83  add_custom_command(
84    TARGET ${PROJECT_NAME}
85    COMMAND ${include_command}
86    VERBATIM
87  )
```

- #3：プロジェクト名を指定する処理です。ここに指定したプロジェクト名は出力ファイル名になりますので、必要に応じて編集してください。
- #11～17：ビルド対象とするソース・ファイル（拡張子：.c、.cpp、.asm、.s、.src）を収集する処理です。ビルド対象とするソース・ファイルは、本ファイル（CMakeLists.txt）のあるフォルダおよびその階層フォルダに置いてください。
- #18～59：ソース・ファイルの依存関係を定義するファイルを生成する処理です。
- #62～72：ライブラリ・ジェネレータを実行する処理です。ライブラリ・ジェネレータに指定するオプションは、#65 に記載してください。

- #78～87：依存関係ファイルを makefile でインクルードする処理です。トリッキーな処理を行っているため、この処理は必ずファイルの最後に記載する必要があります。別の処理を追加する場合は、この処理より前に追加してください。

4.3　mot ファイルを生成する場合

作成された abs ファイルから mot ファイルを生成するには、CMake 構成ファイルの#73 以降の処理より前に以下の処理を追加してください。

#	CMakeLists.txt に追加する処理
1	########## Execute Converter
2	# Command rule for build .mot file from .abs file
3	set(CONVERTER_FLAGS -form=stype -nologo -nomessage)
4	add_custom_command(
5	TARGET ${PROJECT_NAME}
6	POST_BUILD
7	COMMAND ${CMAKE_LINKER} ${PROJECT_NAME}.abs ${CONVERTER_FLAGS}
8	COMMENT "Converter:"
9	VERBATIM
10	)

- #3：オプションを記載してください。.hex などの他のフォーマットのファイルを生成する場合は、「-form」オプションを編集してください。

4.4　x ファイルを生成する場合

e² studio 上でデバッグを行うには、abs ファイルから x ファイルを生成する必要があります。x ファイルを生成するには、CMake 構成ファイルの#73 以降の処理より前に以下の処理を追加してください。

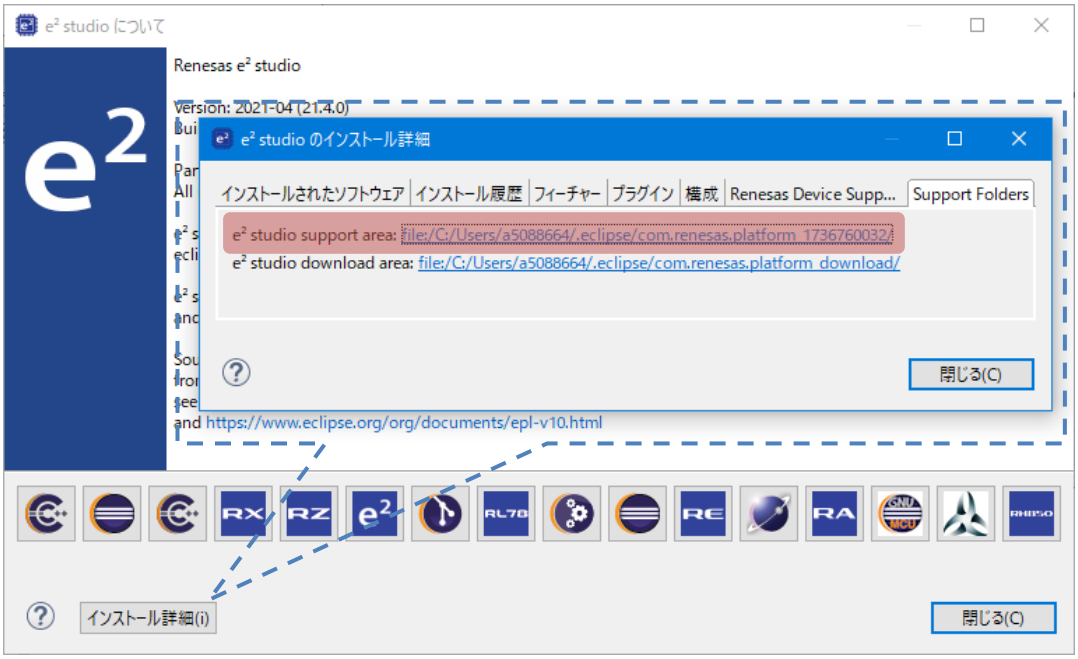
#	CMakeLists.txt に追加する処理
1	########## Execute X Converter
2	# Command rule for build .x file from .abs file
3	set(RENESAS_XCONVERTER renesas_cc_converter.exe)
	add_custom_command(
4	TARGET ${PROJECT_NAME}
5	POST_BUILD
6	COMMAND ${RENESAS_XCONVERTER} ${PROJECT_NAME}.abs ${PROJECT_NAME}.x
7	COMMENT "X Converter:"
8	VERBATIM
9	)

- #3：このプログラムは、e² studio のサポート・フォルダ内の\Utilities\ccrx フォルダにあります。そのフォルダを環境変数 PATH に追加してください。

サポート・フォルダの位置は、下記手順により確認することができます。

1. e² studio のメニュー［ヘルプ(H)］>［e² studio について(A)］をクリックします。
2. ［e² studio について］ダイアログが表示されます。［インストール詳細(i)］ボタンをクリックします。

3. ［e² studio のインストール詳細］ダイアログが表示されます。［Support Folders］タブをクリックします。



4.5　Linker 用サブコマンド・ファイル

リンカのオプションを記載したファイルです。この内容をベースにファイルを作成して、リンカのオプションを編集してください。

#	Linker.tmp
1	-noprelink
2	-start=SU,SI,B_1,R_1,B_2,R_2,B,R/04,PResetPRG,C_1,C_2,C,C$*,D*,W*,L,PIntPRG,P/0FFE00000,EXCEPTVECT/0FFFFFF80,RESETVECT/0FFFFFFFC
3	-form=absolute
4	-nomessage
5	-list=CCRX_Project.map
6	-nooptimize
7	-rom=D=R,D_1=R_1,D_2=R_2
8	-cpu=RAM=00000000-0003ffff,FIX=00080000-00083fff,FIX=00086000-00087fff,FIX=00088000-0009ffff,FIX=000a0000-000a3fff,RAM=000a4000-000a5fff,FIX=000a6000-000bffff,FIX=000c0000-000dffff,FIX=000e0000-000fffff,ROM=00100000-00107fff,FIX=007fc000-007fcfff,FIX=007fe000-007fffff,RAM=00800000-0085ffff,RAM=fe7f5d00-fe7f5d7f,RAM=fe7f7d70-fe7f7d9f,ROM=ffe00000-ffffffff
9	-nologo
10	-library=CCRX_Project.lib

5. ビルド実行

5.1 CMake を実行する

CMake を実行して makefile を生成します。

コマンドプロンプトを開き、前章で作成した CC-RX ツールチェーン・ファイル、および CMake 構成ファイルがあるフォルダに移動してください。次に、下記オプション・スイッチを指定して CMake を実行してください。CMake により_builds フォルダが生成され、そのフォルダ内に makefile が生成されます。

- -B：ビルド時の出力ディレクトリ
  ここでは、makefile および出力ファイルを生成するフォルダとして「_builds」を指定してください。
- -G：作成するプロジェクト・ファイル
  ここでは、標準的な makefile として「"Unix Makefiles"」を指定してください。
- -D：キャッシュエントリの定義
  ここでは、CC-RX ツールチェーン・ファイルの指定として、
  「CMAKE_TOOLCHAIN_FILE=ccrx.cmake」を指定してください。

```
cmake -H. -B_builds -G "Unix Makefiles" -DCMAKE_TOOLCHAIN_FILE=ccrx.cmake
```

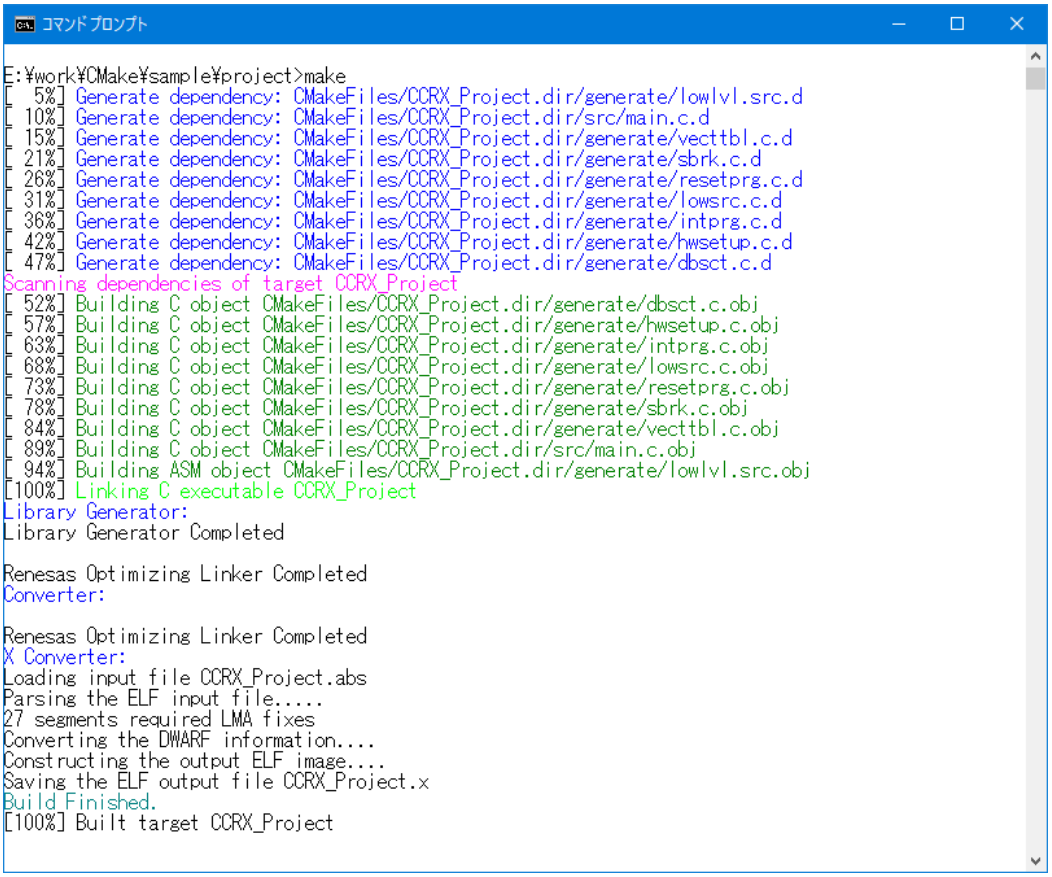
5.2 GNU Make を実行する

CMake が生成した makefile を使用して、make を実行し、オブジェクト・ファイルを生成します。

CMake に下記パラメータを指定して、make を実行します。

```
cd _builds
make
```

- 実行結果



6. サンプル・コード

このドキュメントで説明したサンプル・コードは、ルネサスエレクトロニクスホームページから入手してください。
サンプル・コードは RX65N をターゲットにしたプロジェクトです。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Apr.24.20	-	新規作成
2.00	Jun.30.21	すべて	全面見直しを実施 ・不要な情報を削除してより汎用的な設定内容に改善
2.01	Sep.15.21	11 ページ	サンプル・コードの章を追加

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. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   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. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「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。
5. クロックについて
   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。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）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。
8. 製品間の相違について
   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。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
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。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配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。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。