R-Car Series, 3rd Generation

Capacity Aware Migration Strategy

要旨

[Purpose]

車載情報システムのような、ナビゲーション、音楽再生、動画再生、メーター表示等、複数のアプリケーションを同時に動作させるユースケースでは、マルチコアプロセッシング環境を使用し、複数のCPUを並列に動作させることで高い計算能力により、アプリケーションの快適な動作が実現できます。

R-Car H3/M3-W/M3-W+はCortex-A57とCortex-A53の2種類で計8個(H3)/6個(M3-W/M3-W+)のCPUを搭載しており、Capacity Aware migration Strategy(CAS)の利用により効果的なマルチコアプロセッシングが可能です。本ドキュメントではCASの機能説明とデバッグ方法を紹介します。

[Readers]

本ドキュメント読者は、下記の一般的な知識を持っていることを想定しています。

* Engineering, logic circuits, microcontrollers, and Linux.
* The functionality of the multiple processor cores of R-Car H3, R-Car M3-W and R-Car M3-W+ products.
* The electrical specifications of the multiple processor cores of R-Car H3, R-Car M3-W and R-Car M3-W+ products.
* The functions of the BSP drivers for R-Car H3, R-Car M3-W and R-Car M3-W+ products.

[Note]

* 本ドキュメントは弊社が提供するYocto v3.7.0を対象として記述しております。

ターゲットデバイス

・R-Car H3

・R-Car M3-W/R-Car M3-W+

目次

[1. 概要 3](#_Toc515010962)

[1.1 CASとは 3](#_Toc515010963)

[1.2 CASの機能概要 3](#_Toc515010964)

[1.3 CASの動作概要 4](#_Toc515010965)

[2. デバッグガイド 5](#_Toc515010966)

[2.1 タスクのモニター方法(htop) 5](#_Toc515010967)

[2.1.1 事前準備 5](#_Toc515010968)

[2.1.2 操作手順 5](#_Toc515010969)

[2.2 タスクのモニター方法(ftrace) 6](#_Toc515010970)

[2.2.1 事前準備 6](#_Toc515010971)

[2.2.2 操作手順 6](#_Toc515010972)

[3. CAS機能の使用上の注意点 7](#_Toc515010979)

[3.1 スケジューラの負荷分散について 7](#_Toc515010980)

[3.2 タスクの高レスポンス化 8](#_Toc515010983)

[3.2.1 Processor Affinityによる対策 8](#_Toc515010984)

[3.2.2 Realtime process化による対策 13](#_Toc515010998)

[4. CAS機能の無効化 14](#_Toc515011010)

[Appendix 15](#_Toc515011011)

[A1. Cortex-A57/A53間のタスクスイッチの処理時間 15](#_Toc515011012)

[A2. htopの環境構築 16](#_Toc515011013)

[A3. ftraceの環境構築 17](#_Toc515011014)

[A4. tasksetの環境構築 18](#_Toc515011015)

# 概要

## CASとは

CASとは弊社が非対称CPUシステム向けに提案する機能の名称で、Linux Kernelの非対称CPU容量サポートにSoC依存部のパラメータを追加することで実現しております。CPU容量はCPUの計算能力のことを示しており、この非対称CPU容量サポートとは、ARM社が考案したEAS(Energy Aware Scheduler)の機能の一部となります。

ARM has refactored EAS into several parts.

Acceptance by the Linux kernel community of some parts of EAS.

EAS

Capacity

Awareness

Frequency-invariance support

Core of

EAS

Upstream

Linux

Kernel 4.14

SoC

dependent

parameters

(device tree)

CAS

Renesas

Linux

BSP

Incorporation by Renesas of a facility for enabling big.LITTLE on our relevant BSP

図 1‑1 CASの構成と経緯

## CASの機能概要

CASは Cortex-A57とCortex-A53が同時に動作するデバイスにおいて、性能を重視したタスク配置を行います。

Linuxの標準スケジューラであるCompletely Fair Scheduling(CFS)は、CPU間の負荷を均等に維持することでマルチコアプロセッシングを効果的に利用することができます(図 1‑2 [CFS])。

一方、Cortex-A57とCortex-A53では性能差があるため、CPU間の負荷を均等に維持するより計算能力の大きいCortex-A57により多くのタスクを処理させる方がシステム全体の性能が向上します。

CASはCortex-A57とCortex-A53のCPU容量をスケジューラに意識させることで、Cortex-A57に優先的にタスクを割り付けるようにしています（図1-2[CAS]）。

CPU 0

CPU 1

CPU 2

CPU 3

Cortex-A57

Cortex-A57

Cortex-A53

Cortex-A53

CPU 0

CPU 1

CPU 2

CPU 3

[CFS]

[CAS]

タスク

Cortex-A57

Cortex-A57

Cortex-A53

Cortex-A53

図 1‑2　CASのタスク割り付け

## CASの動作概要

CASの特徴はCPU容量を意識したタスクのスケジューリングを行うことです。具体的な振る舞いについて後述に説明します。

Linux Kernel内ではCPU毎にCPU容量が定義されています。このCPU容量の中は使用中の容量と空き容量に区分されており、CPUにタスクが割り付けられると、タスクの負荷に応じてそのCPUの使用中の容量が増え、空き容量が減少する仕組みとなっております。

Cortex-A53とCortex-A57では計算能力が異なるため、それに合わせて最大CPU容量も異なります。CAS機能を有効化することで、スケジューラは最も空き容量が大きいCPUからタスクを割り付けるようになるため、結果として最大CPU容量が大きなCortex-A57にタスクが優先的に割り付けられる振る舞いとなります。

具体的に負荷の異なる例でのタスク割り付け状態を図で示します(図1-3)。なお、CPUの構成は説明のため単純化しており、実際のデバイスとは異なります。

低負荷の例 : Cortex-A57の空き容量(a) > Cortex-A53の空き容量(b)

スケジューラはCortex-A57にタスクを割り付けます。

中負荷の例 : Cortex-A57の空き容量(a) > Cortex-A53の空き容量(b)

スケジューラはCortex-A57にタスクを割り付けます。

高負荷の例 : Cortex-A57の空き容量(a) < Cortex-A53の空き容量(b)

スケジューラはCortex-A53にタスクを割り付けます。

以上のように、CASではCPU容量の差異によりCortex-A57へ優先的にタスクが割り付けられ、CPU容量を意識したタスクスケジューリングを実現できます。

Cortex-A57

Cortex-A57

Cortex-A53

Cortex-A53

CPU 0

CPU 1

CPU 2

CPU 3

**低負荷**

Cortex-A57

Cortex-A57

Cortex-A53

Cortex-A53

CPU 0

CPU 1

CPU 2

CPU 3

Cortex-A57

Cortex-A57

Cortex-A53

Cortex-A53

CPU 0

CPU 1

CPU 2

CPU 3

**中負荷**

**高負荷**

空き容量

(a)

(b)

(a)

(b)

(a)

(b)

タスク

図 1‑3　負荷別タスク割り付け例

# デバッグガイド

CASによるパフォーマンス改善はお客様のユースケースに強く依存します。お客様のユースケースにおいて、CASを最大限に活かすためには、タスクの割り付け状況や、タスクの遷移、経過をモニターすることが非常に重要となります。ここでは、デバッグに有効なタスクのモニター方法(htopとftrace)を紹介します。

## タスクのモニター方法(htop)

タスクの割り付け状態をモニターする簡易な手段として、htopを紹介します。

htopは、CPU負荷や実行中タスクのCPU割り付け状況などを、コンソール上でモニターできるツールです。事前にファイルシステムにhtopを追加するだけで利用することができます。

ただしhtopはスナップショットで情報を表示しているため、タスクの遷移や経過を連続的に捉えることはできません。タスクの連続的な変化を観測するには、Linux Kernelのトレース機能(ftrace)を利用したタスクのモニター方法(2.2章)を提案します。

### 事前準備

Appendix 「A2.htopの環境構築」を参照してください。

### 操作手順

ホストPC(コンソール)から、次のコマンド入力を行います。

$ htop

コマンド実行により、タスクの状況が次のようにコンソールに表示されます。

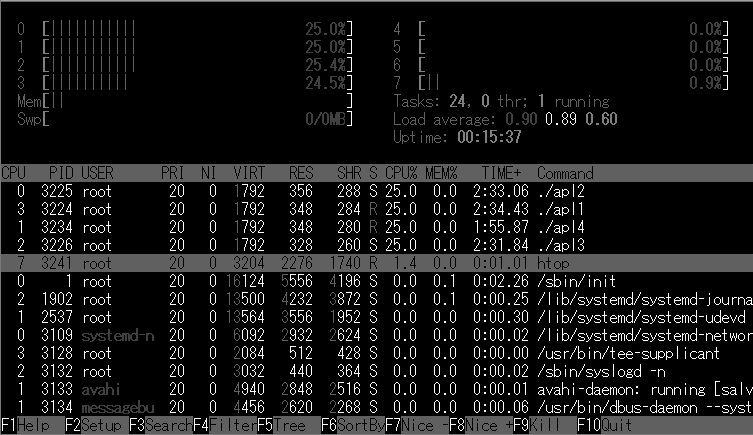


図 2‑1　htopの表示例(R-Car H3)

詳しい使用方法については、htopのHelpを参照してください。

## タスクのモニター方法(ftrace)

タスクの遷移や経過など連続的なタスク割り付け状況を詳細に観測する手段として、Linux Kernelのトレース機能(ftrace)を使用します。

ftraceは、Linux Kernelに組み込まれたトレース機能で、実行したカーネルのイベントをログとして出力することができます。

出力されたログを解析してGUI表示するために、trace-cmdとKernelSharkを使用します。

### 事前準備

Appendix 「A3.ftraceの環境構築」を参照してください。

### 操作手順

1. トレースの実施(記録)

観測したいタスクが動作している状況で、次のコマンドをコンソールから入力してトレースを開始します。

$ trace-cmd record -e sched -b 4000

トレースを終了するには、コマンドを入力したコンソールから Ctrl^C(‘Ctrl’ + ‘C’) キーを押します。

トレースされたデータは、カレントフォルダに”trace.dat”のファイル名で保存されます。

1. トレース内容の表示

KernelSharkを使って取得したトレース結果を表示します。

KernelSharkがインストールされたLinuxホストPC上で、取得した”trace.dat”を次のコマンドで解析します。

$ kernelshark {ファイルの位置/}trace.dat

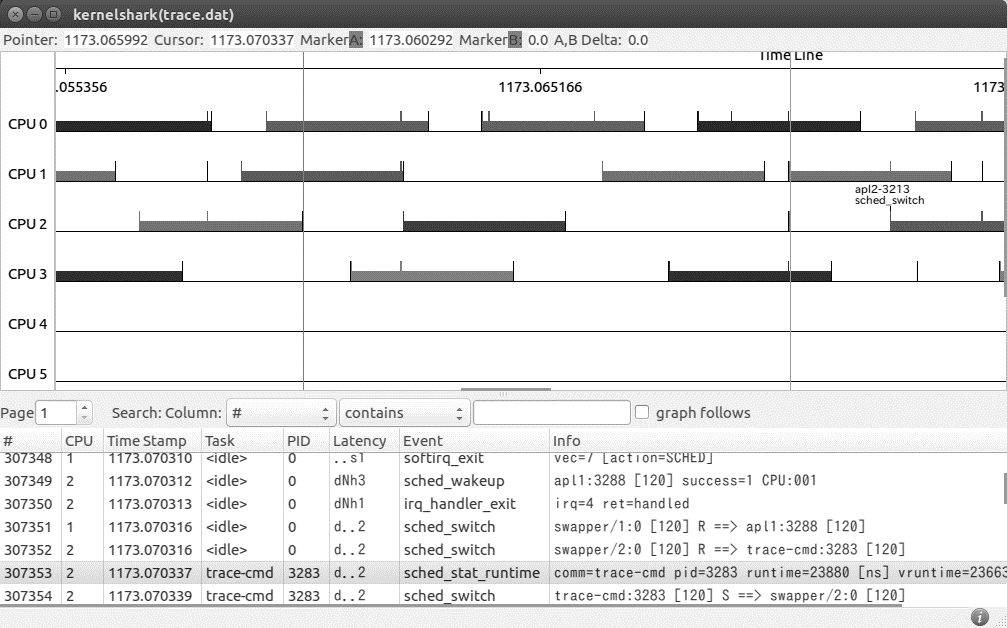


図 2‑2　KernelSharkによるftrace結果の表示例(R-Car H3)

# CAS機能の使用上の注意点

## スケジューラの負荷分散について

CASによってスケジューラはCortax-A57へタスクを優先的に割り付けますが、スケジューラの標準動作である負荷分散プロセスにより瞬間的にCortex-A53にタスクが割り付けられ、その後再度Cortex-A57に割り付けられることがあります。

この振る舞いにより、該当タスクがCortex-A57上で動作し続けた場合に比べて、CPUの性能差だけ該当タスクのレスポンスなどが低下することになります。

タスク

CPU 0

CPU 1

Cortex-A57

Cortex-A53

Cortex-A57

Cortex-A53

CPU 0

CPU 1

CPU 0

CPU 1

Cortex-A57

Cortex-A53

定常状態

瞬間的にタスクが

Cortex-A53へ割り付け

タスクを割り付け

CFSによるタスク割り付け

再度タスクが

Cortex-A57へ割り付け

タスクを割り付け

図 3‑1　瞬間的なタスク割り付けの発生

## タスクの高レスポンス化

### Processor Affinityによる対策

Processor affinityは、ユーザが指定したアプリケーションをCPU又はCPUグループに固定で割り付け動作させる手法です。Cortex-A57とCortex-A53が同時起動した環境では、ユーザがアプリケーションをCortex-A57に固定的に割り付けることで、高負荷プロセスが意図せずCortex-A53で動作することがなくなります。

Processor affinityを実現するための手法は表 3‑1に示すようにいくつかありますが、プロセスをグループで管理できる観点から、Cgroupを推奨します。

表 3‑1　 Processor Affinityの実現手法

|  |  |
| --- | --- |
| Processor affinity | 概要 |
| Cgroup | CgroupはLinuxの標準機能で、プロセスをグループ化して、CPUやメモリなどのリソースをグループ単位に割り当て管理することができます。Cgroupはsysfsからコントロールすることができます。 |
| taskset | tasksetは、コマンドラインからプロセスID(PID)と動作させるCPUを指定し実行することで、Processor affinityを実現することができます。tasksetではプロセス単位でCPUを割り当てる必要があります。 |

　なお、本ドキュメントでは上記の手法を推奨しておりますが、Linuxコミュニティではユーザースペースのデーモン(bLsched)がCPUへタスクを配置する手法がアイディアの1つとして提案されております。bLschedの詳細情報は下記のURLで公開済みですので、必要に応じてご参照ください。

https://github.com/BayLibre/bLsched

#### Cgroupの設定手順

CgroupはLinuxの標準機能で、プロセスのグループを作成して、CPUやメモリなどのリソースをグループ単位に割り当て管理することができる機能です。グループは複数作成することができ、グループに属するプロセスは割り当てられたリソース内で動作します。Cortex-A57とCortex-A53を同時起動させた環境の場合は、図 3‑2に示したように、Cortex-A57又はCortex-A53のみで動作させるアプリケーションごとにグループ化し、各CPUリソースを割り当てることができます。

ここでは、Cortex-A57で固定的に割り付けることで、Cortex-A53への意図しないタスクスイッチを防ぐ設定を例として説明します。

App. 1

App. 2

App. 3

Cortex-A53

Cortex-A53

Cortex-A57

Cortex-A57

Cortex-A57

Cortex-A57

CPUリソース

App. 1

アプリケーション

Cortex-A53

Cortex-A53

CPUリソース

App. 2

アプリケーション

App. 3

Cortex-A53 group

Cortex-A57 group

図 3‑2 Cgroupを用いたアプリケーションの割り付け例

##### Cgroupのグループの作成

CgroupとしてCortex-A57(big group)で動作させるグループの作成方法について説明します。

Step 1：起動CPUの確認

下記のコマンドを実行し、CPU番号とCPUの種類(Cortex-A57 or Cortex-A53)を確認してください。

$ cat /proc/cpuinfo

/\* CPUの種類は、実行結果のCPU partで確認できます。　Cortex-A57→0xd07、Cortex-A53→0xd03 \*/

Step 2：big groupの作成

下記のコマンドでbig groupを作成します。

$ mkdir /sys/fs/cgroup/cpuset/big

Step 3：memory nodeの設定

memory nodeは、Cgroupのメモリの割り当てを設定するパラメータです。メモリの割り当てを指定しない場合は"0"を設定してください。

$ echo 0 > /sys/fs/cgroup/cpuset/big/cpuset.mems

Step 4：各グループへのCPUリソースの割り当て

STEP1で確認したCortex-A57のCPU番号をbig groupに登録します。

* R-Car H3の登録

$ echo 0-3 > /sys/fs/cgroup/cpuset/big/cpuset.cpus

/\* R-Car H3の場合、 Cortex-A57のCPU番号は0～3\*/

* R-Car M3-W/R-Car M3-W+の登録

$ echo 0-1 > /sys/fs/cgroup/cpuset/big/cpuset.cpus

/\* R-Car M3-W, R-Car M3-W+の場合、 Cortex-A57のCPU番号は0～1\*/

##### Cgroupのアプリケーションの割り付け

アプリケーションを作成したグループに割り付ける方法について説明します。

Step 1：アプリケーションのPIDの確認

$ ps

/\* 実行結果のPID列を確認してください \*/

下記のコマンドでCgroupにて割り付けるアプリケーションのPIDを確認してください。

Step 2：アプリケーションの割り付け

下記のコマンドでアプリケーションをbig groupに割り付けます。コマンドの [PID] は割り付けるアプリケーションのPIDに置き換えて実行してください。

* big group割り付けの登録

$ echo [PID] > /sys/fs/cgroup/cpuset/big/tasks

また、下記のコマンドで、big groupの割り付けを解除することができます。

* big group割り付けの解除

$ echo [PID] > /sys/fs/cgroup/cpuset/tasks

#### tasksetの設定手順

tasksetは、実行中のプロセスのCPU affinityを設定するために使用されます。

CPU affinityは、プロセスをシステム上の特定のCPUに「結合」するスケジューラのプロパティです。

実行中のプロセス毎にCPUを指定する場合に利用します。

tasksetを使用するためには、ファイルシステムの構築時にtasksetの追加が必要です。

詳細は Appendix A4を参照してください。

下記のコマンドで、指定したプロセスが指定したCPUで動作します。

$ taskset -pc X [プロセスID]

/\* XはCPUの番号(0,1,2...)を指定する \*/

### Realtime process化による対策

3.2.1 Processor Affinityによる対策にて、プロセスをCortex-A57に固定的に割り付けることで意図しないCortex-A53へのタスクスイッチを防ぎ、さらにRealtime process化することでプロセスを優先的に動作させることができます。

　Realtime process化する方法として、chrtを使用した事例を表 3‑2に示します。

表 3‑2　Realtime優先度設定事例

|  |  |  |  |
| --- | --- | --- | --- |
|  | 概要 | 詳細 | コマンド |
| 1 | リアルタイム優先度を指定してプロセス実行 | リアルタイムの優先度を指定してプロセスを実行します。\*1 | chrt -r X [プロセス名]  ※Xは優先度(0-99)、[プロセス名]は動作させるプロセス名を指定する。  例）優先度99で/usr/local/bin/sampleプロセスを起動する際のコマンド例は、以下となります。  chrt -r 99 /usr/local/bin/sample |

Note: 1. root権限で実施してください

# CAS機能の無効化

CASの効果を確認するため、あるいはCASを必要としない場合、CASの機能を無効にすることができます。

CASの機能を無効にするには、big.LITTLEアーキテクチャを無効にします。

具体的な方法は、RENESAS\_RCH3M3M3NE3\_YoctoStartupGuide\_UME\_v3.7.0(Linux Interface Specification Yocto recipe Start-Up Guide)の5. Confirm starting of U-Boot and Linuxに記載されているStep 6 change the bootargs by U-Bootを参照してください。

# Appendix

# Cortex-A57/A53間のタスクスイッチの処理時間

タスクスイッチに掛かる時間を表 A1‑1に示します。

表 A1‑1　タスクスイッチの処理時間

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cortex-A57  ->Cortex-A57 | Cortex-A57  ->Cortex-A53 | Cortex-A53  ->Cortex-A57 |
| クラスタ跨ぎ | 無 | 有 | 有 |
| 切り替え時間 | 14us | 19us | 21us |

[環境]

* SW : Yocto v3.7.0 (Linux BSP 3.6.2)
* SoC : R-Car H3 Ver.3.0 (Salvator-XS)
* 測定方法：

1. 任意の無限ループする実行プロセス(A)をCPU1で起動

2. tasksetコマンド(\*)でプロセスAをCPU2にスイッチ

(\*) tasksetはプロセスのCPUアサインを変更設定するコマンド

図 A1‑1　タスクスイッチの処理フロー

１．CPU1でプロセスAが実行

２．プロセスAを停止

３．プロセスＡをCPU2へスイッチ

４．プロセスAをCPU2上でWakeup

A

CPU1

Run queue

CPU2

Run queue

A

A

Run queue

Run queue

CPU2

Run queue

CPU1

Run queue

CPU2

A

Run queue

CPU1

CPU1

CPU2

Run queue

測定区間

：停止状態

：動作中

# htopの環境構築

htopを利用するには、以下に記載したファイルを編集し、再度ファイルシステムを構築してください。

図 A2‑1 htopの追加

[build/conf/local.confに次の行を追加します。]

CORE\_IMAGE\_EXTRA\_INSTALL += "htop"

# ftraceの環境構築

* + - * 1. ftraceを有効にする。

ftrace機能を有効にするには、Kernel Configurationで次の設定を行います。

図 A3‑1 Kernel Configuration

Kernel hacking --->

[\*] Tracers --->　 **\***1

[\*] Kernel Function Tracer　 **\***1

[\*] Kernel Function Graph Tracer　 **\***1

**\***1　 ftraceを有効にするためにチェックしてください。

* + - * 1. trace-cmdをファイルシステムにインストールする。

trace-cmdを利用するには、以下に記述したファイルを編集し、再度ファイルシステムを構築してください。

図 A3‑2 trace-cmdの追加

[build/conf/local.confに次の行を追加します。]

CORE\_IMAGE\_EXTRA\_INSTALL += "trace-cmd"

* + - * 1. KernelSharkをインストールする。

KernelSharkはLinuxホストPCで実行します。

LinuxホストPCにapt-getコマンドなどでKernelSharkをインストールしてください。

# tasksetの環境構築

tasksetを利用するには、以下に記述したファイルを編集し、再度ファイルシステムを構築してください。

図 A4‑1 tasksetの追加

[build/conf/local.confに次の行を追加します。]

CORE\_IMAGE\_EXTRA\_INSTALL += "util-linux"

改訂記録

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | 発行日 | 改訂内容 | |
| ページ | ポイント |
| 1.00 | 2018.06 | － | 新規作成 |
| 1.01 | 2019.03 | P1 | Purpose, Target Readers and Target DeviceにR-Car M3-W+を追加 |
| P11 | Assigning CPU resources to the groups updated. (“ R-Car M3-W+”) |

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、VIL（Max.）からVIH（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、VIL（Max.）からVIH（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－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](https://www.renesas.com/) |  |  |
| 商標について |  |  |
| Arm、CortexはArm Limitedの登録商標または商標です。  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。 |  |  |



変更内容〔ルネサス内部向け〕

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | 発行日 | 改訂内容 | |
| ページ | ポイント |
| 1.00 | June, 2018 | All | 新規作成 |