RXファミリ
I²C バスインタフェース (RIIC) モジュール
Firmware Integration Technology

R01AN1692JJ0220
Rev.2.20
2017.08.31

要旨

本アプリケーションノートでは、Firmware Integration Technology (FIT)を使用した I²C バスンタフェースモジュール (RIIC) について説明します。本モジュールは RIIC を使用して、デバイス間で通信を行います。以降、本モジュールを RIIC FIT モジュールと称します。

対象デバイス

- RX110、RX111、RX113 グループ
- RX130 グループ
- RX230、RX231、RX23T グループ
- RX24T、RX24U グループ
- RX64M グループ
- RX65N、RX651 グループ
- RX71M グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

関連ドキュメント

Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)
CS+に組み込む方法 Firmware Integration Technology (R01AN1826)
Renesas e² studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)

目次

1. 概要

RIIC FIT モジュールは、RIIC を使用し、マスタデバイスとスレーブデバイスが送受信を行うための手段を提供します。RIIC は NXP 社が提唱する I²C バス(Inter-IC-Bus)インタフェース方式に準拠しています。以下に本モジュールがサポートしている機能を列挙します。

- マスタ送信、マスタ受信、スレーブ送信、スレーブ受信に対応
- 複数のマスタがひとつのスレーブと調停を行いながら通信するマルチマスタ構成
- 通信モードはスタンダードモードとファストモードに対応し、最大転送速度は 400kbps
  ただし、RX64M、RX71M、RX65N のチャネル 0 はファストモードプラスに対応し、最大転送速度は 1Mbps

制限事項

本モジュールには以下の制限事項があります。

（1）DMAC、DTC と組み合わせて使用することはできません。

（2）RIIC の NACK アービトレーションロスト機能に対応していません。

（3）10 ビットアドレスの送信に対応していません。

（4）スレーブデバイス時、リスタートコンディションの受け付けに対応していません。

リスタートコンディション直後のアドレスで本モジュールを組み込んだデバイスのアドレスを指定しないでください。

（5）本モジュールは多重割り込みには対応していません。

（6）コールバック関数内では R_RIIC_GetStatus 関数以外の API 関数の呼び出しは禁止です。

（7）割り込みを使用するため、I フラグは“1”で使用してください。

1.1 RIIC FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.12 モジュールの追加方法」を参照してください。

1.2 API の概要

表 1.1に本モジュールに含まれる API 関数を示します。

表 1.1　API 関数一覧

関数	関数説明
R_RIIC_Open()	この関数は RIIC FIT モジュールを初期化する関数です。この関数は他の API 関数を使用する前に呼び出される必要があります。
R_RIIC_MasterSend()	マスタ送信を開始します。引数に合わせてマスタのデータ送信パターンを変更します。 ストップコンディション生成まで一括で実施します。
R_RIIC_MasterReceive()	マスタ受信を開始します。引数に合わせてマスタのデータ受信パターンを変更します。 ストップコンディション生成まで一括で実施します。
R_RIIC_SlaveTransfer()	スレーブ送受信を行う関数。 引数のパターンに合わせてデータ送受信パターンを変更します。
R_RIIC_GetStatus()	本モジュールの状態を返します。
R_RIIC_Control()	各コンディション出力、SDA 端子のハイインピーダンス出力、SCL クロックのワンショット出力、および RIIC のモジュールリセットを行う関数です。 主に通信エラー時に使用してください。
R_RIIC_Close()	RIIC の通信を終了し、使用していた RIIC の対象チャネルを解放します。
R_RIIC_GetVersion()	本モジュールのバージョンを返します。

1.3 RIIC FIT モジュールの概要

1.3.1 RIIC FIT モジュールの仕様

1) 本モジュールは、マスタ送信、マスタ受信、スレーブ送信、スレーブ受信 をサポートします。

— マスタ送信では 4 種類の送信パターンが設定可能です。マスタ送信の詳細は1.3.2に示します。

— マスタ受信では、マスタ受信とマスタ送受信の 2 種類の受信パターンが設定可能です。マスタ受信の詳細は1.3.3に示します。

— スレーブ受信とスレーブ送信は、マスタから送信されるデータの内容によって、その後の動作を行います。スレーブ受信の詳細は、「1.3.4スレーブ送受信の処理」の「(1) スレーブ受信」に、スレーブ送信の詳細は、「1.3.4スレーブ送受信の処理」の「(2) スレーブ送信」に示します。

2) 割り込みは、スタートコンディション生成、スレーブアドレス送信／受信、データ送信／受信、NACK 検出、アービトレーションロスト検出、ストップコンディション生成のいずれかの処理が完了すると発生します。RIIC の割り込み内で本モジュールの通信制御関数を呼び出し、処理を進めます。

3) RIIC のチャネルが複数存在する場合、本モジュールは、複数のチャネルを制御することができます。また、複数のチャネルを持つデバイスでは、複数のチャネルを使用して同時に通信することができます。

4) 1 つのチャネル・バス上の複数かつアドレスが異なるスレーブデバイスを制御できます。ただし、通信中(スタートコンディション生成から、ストップコンディション生成完了までの期間)は、そのデバイス以外の通信はできません。図 1.1に複数スレーブデバイスの制御例を示します。

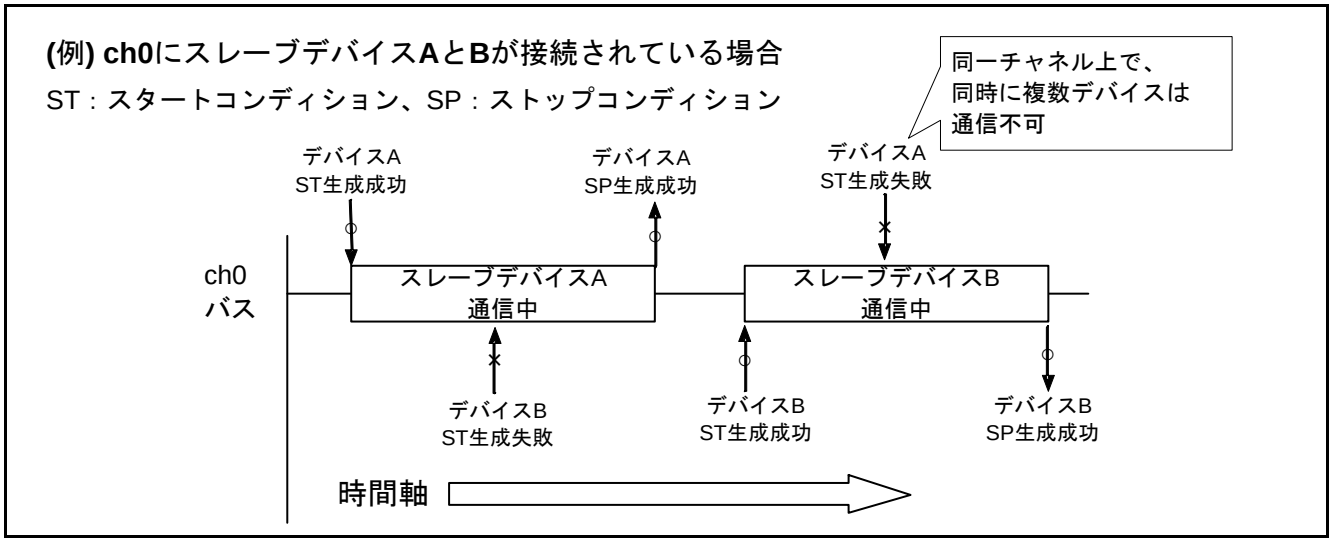


図 1.1 複数スレーブデバイスの制御例

1.3.2 マスタ送信の処理

マスタデバイスとして、スレーブデバイスへデータを送信します。

本モジュールでは、マスタ送信は 4 種類の波形を生成できます。マスタ送信の際、引数とする I²C 通信情報構造体の設定値によってパターンを選択します。図 1.2～図 1.5に 4 種類の送信パターンを示します。また、I²C 通信情報構造体の詳細は、2.9を参照してください。

(1) パターン 1

マスタデバイスとして、2 つのバッファのデータ(1st データと 2nd データ)をスレーブデバイスへ送信する機能です。

初めにスタートコンディション(ST)を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目は転送方向指定ビットになりますので、データ送信時には“0”(Write)を送信します。次に 1st データを送信します。1st データとは、データ送信を行う前に、事前に送信したいデータがある場合に使用します。例えばスレーブデバイスが EEPROM の場合、EEPROM 内部のアドレスを送信することができます。次に 2nd データを送信します。2nd データがスレーブデバイスへ書き込むデータになります。データ送信を開始し、全データの送信が完了すると、ストップコンディション(SP)を生成してバスを解放します。

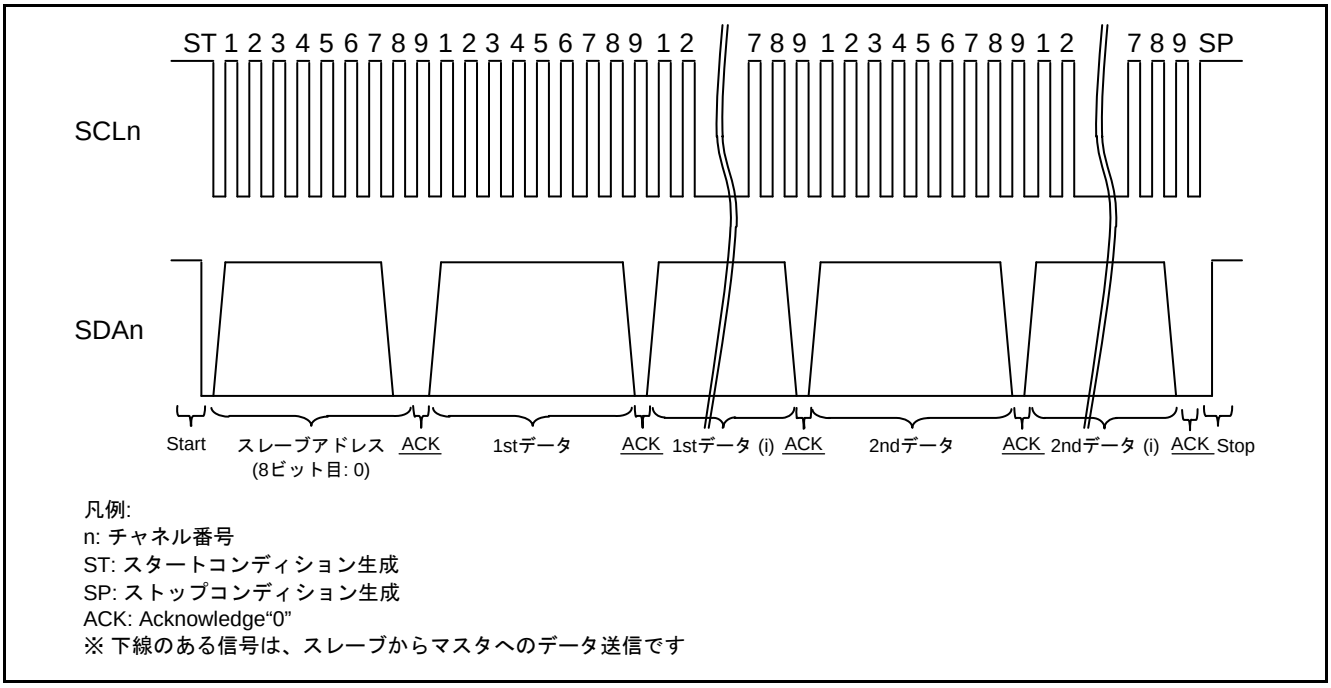
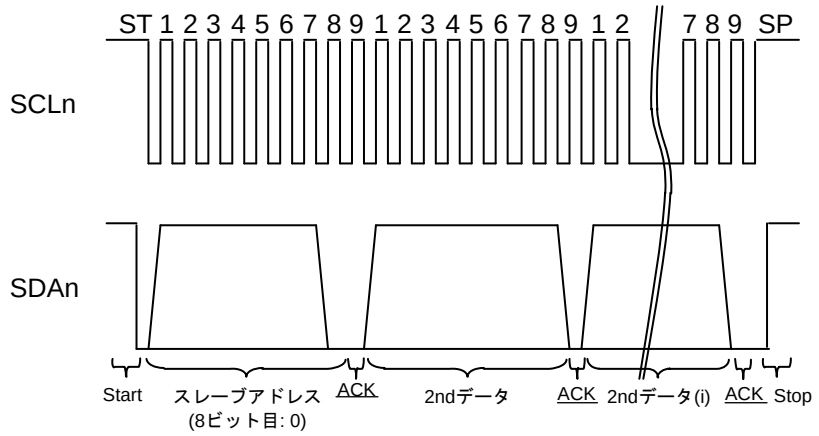


図 1.2 マスタ送信(パターン 1)信号図

(2) パターン 2

マスタデバイスとして、1 つのバッファのデータ(2nd データ)をスレーブデバイスへ送信する機能です。

スタートコンディション(ST)の生成からスレーブデバイスのアドレスを送信まではパターン 1 と同様に動作します。次に 1st データを送信せず、2nd データを送信します。全データの送信が完了すると、ストップコンディション(SP)を生成してバスを解放します。



凡例:
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
ACK: Acknowledge“0”
※ 下線のある信号は、スレーブからマスタへのデータ送信です

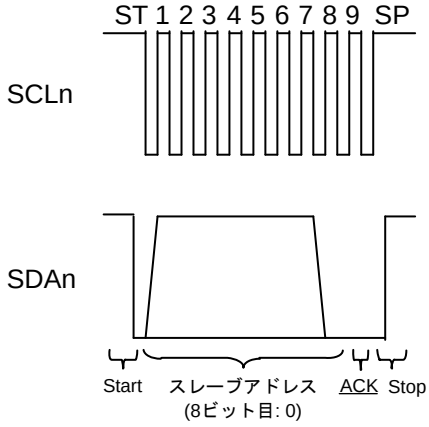
図 1.3 マスタ送信(パターン 2)信号図

(3) パターン 3

マスタデバイスとして、スレーブアドレスのみをスレーブデバイスへ送信する機能です。

スタートコンディション(ST)を生成から、スレーブアドレス送信まではパターン 1 と同様に動作します。スレーブアドレス送信後、1st データ/2nd データを設定していない場合、データ送信は行わず、ストップコンディション(SP)を生成してバスを解放します。

接続されているデバイスを検索する場合や、EEPROM 書き換え状態を確認する Acknowledge Polling を行う際に有効な処理です。



凡例:
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
ACK: Acknowledge“0”
※ 下線のある信号は、スレーブからマスタへのデータ送信です

図 1.4 マスタ送信(パターン 3)信号図

(4) パターン 4

マスタデバイスとして、スタートコンディションとストップコンディションのみをスレーブデバイスへ送信する機能です。

スタートコンディション(ST)を生成後、スレーブアドレスと 1st データ/2nd データを設定していない場合、スレーブアドレス送信とデータの送信は行わず、ストップコンディション(SP)を生成してバスを解放します。バス解放のみを行いたい場合に有効な処理です。

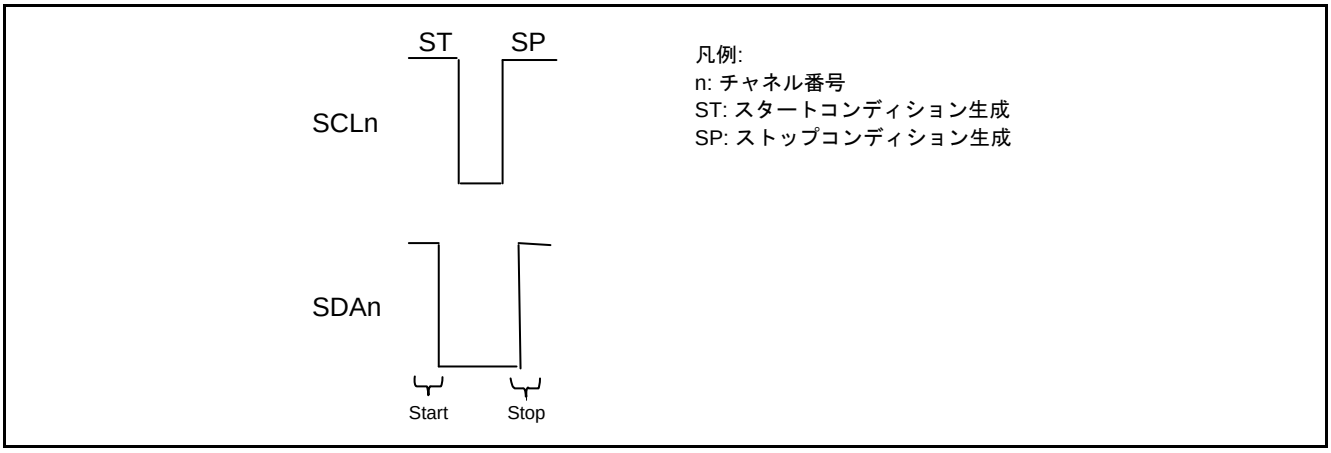


図 1.5　マスタ送信(パターン 4)信号図

図 1.6にマスタ受信を行う際の手順を示します。コールバック関数は、ストップコンディション生成後に呼ばれます。I²C 通信情報構造体メンバの CallBackFunc に関数名を指定してください。

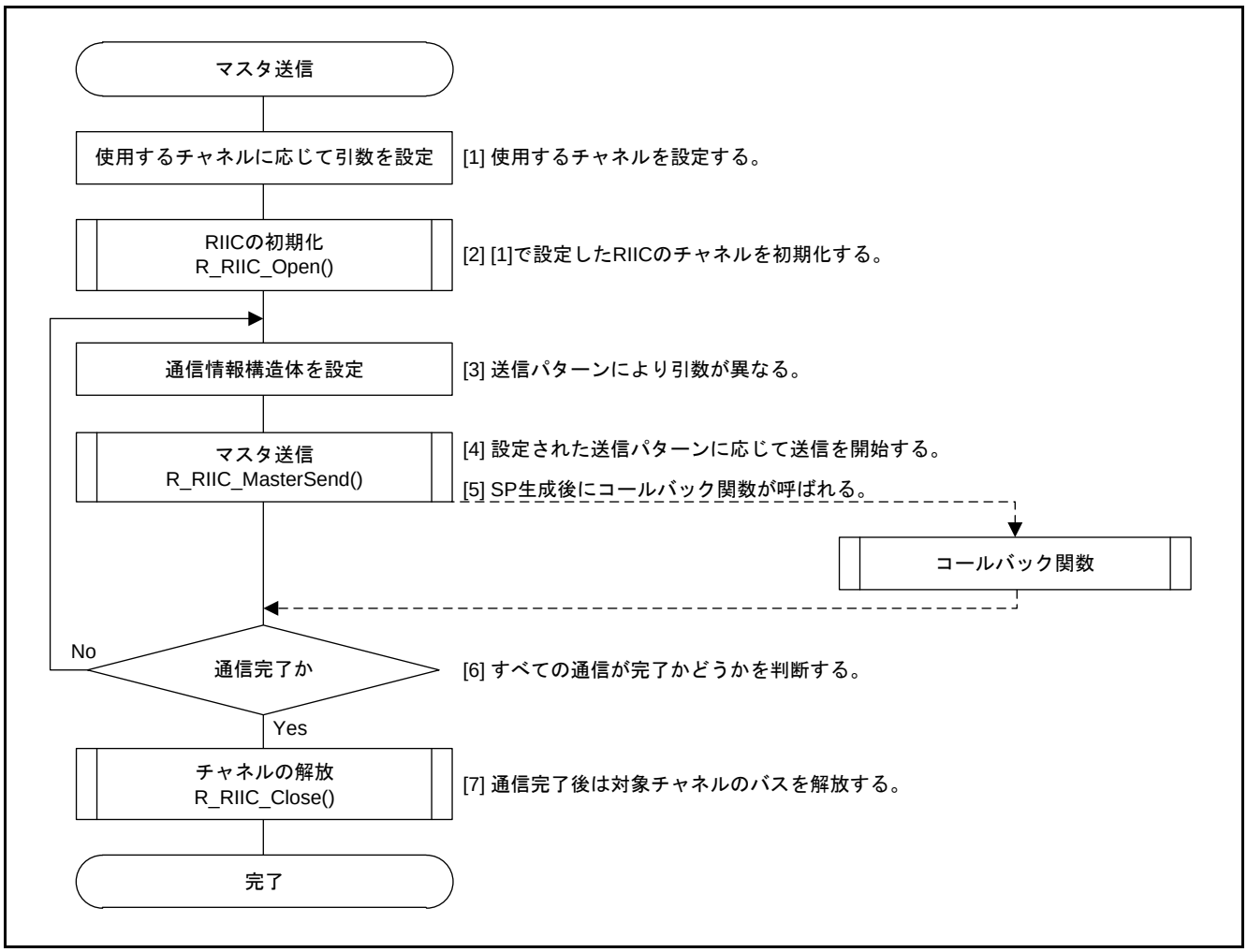


図 1.6　マスタ送信の処理例

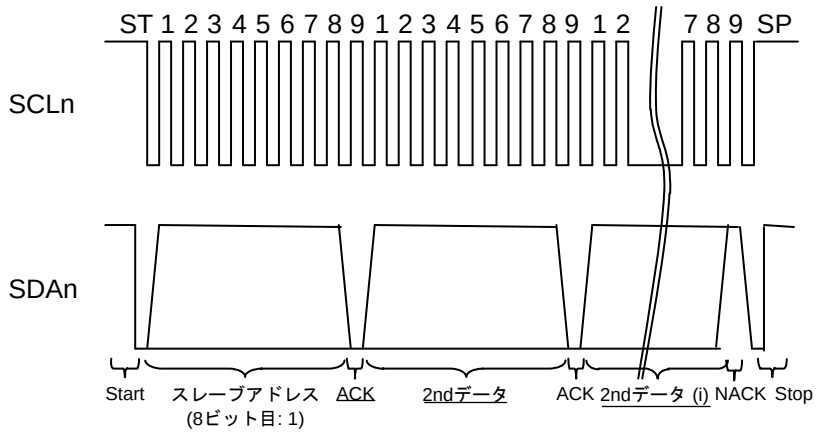
1.3.3 マスタ受信の処理

マスタデバイスとして、スレーブデバイスからデータを受信します。本モジュールでは、マスタ受信とマスタ送受信に対応しています。マスタ受信の際の引数とする I²C 通信情報構造体の設定値によってパターンを選択します。図 1.7～図 1.8に受信パターンを示します。また、I²C 通信情報構造体の詳細は、2.9を参照してください。

(1) マスタ受信

マスタデバイスとして、スレーブデバイスからデータを受信する機能です。

初めにスタートコンディション(ST)を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目は転送方向指定ビットになりますので、データ受信時には“1”(Read)を送信します。次にデータ受信を開始します。受信中は、1 バイト受信するごとに ACK を送信しますが、最終データ時のみ NACK を送信し、スレーブデバイスへ受信処理が完了したことを通知します。全データの受信が完了すると、ストップコンディション(SP)を生成してバスを解放します。



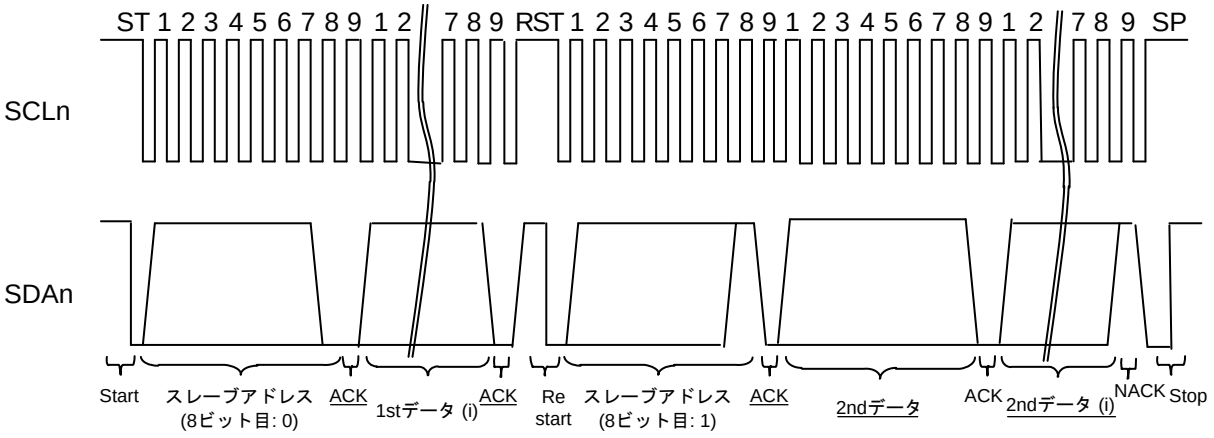
凡例:
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
NACK: Acknowledge “1”
ACK: Acknowledge “0”
※ 下線のある信号は、スレーブからマスタへのデータ送信です

図 1.7 マスタ受信 信号図

(2) マスタ送受信

マスタデバイスとして、スレーブデバイスへデータを送信します。送信完了後、リスタートコンディションを生成し、スレーブデバイスからデータを受信する機能です。

初めにスタートコンディション(ST)を生成し、次にスレーブデバイスのアドレスを送信します。このとき、8 ビット目の転送方向指定ビットには、“0”(Write)を送信します。次に 1st データを送信します。データの送信が完了すると、リスタートコンディション(RST)を生成し、スレーブアドレスを送信します。このとき、転送方向指定ビットには、“1”(Read)を送信します。次にデータ受信を開始します。受信中は、1 バイト受信するごとに ACK を送信しますが、最終データ時のみ NACK を送信し、スレーブデバイスへ受信処理が完了したことを通知します。全データの受信が完了すると、ストップコンディション(SP)を生成してバスを解放します。



凡例:
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
RST: リスタートコンディション生成
※ 下線のある信号は、スレーブからマスタへのデータ送信です

NACK: Acknowledge “1”
ACK: Acknowledge “0”

図 1.8 マスタ送受信 信号図

図 1.9にマスタ受信を行う際の手順を示します。コールバック関数は、ストップコンディション生成後に呼ばれます。I²C 通信情報構造体メンバの CallBackFunc に関数名を指定してください。

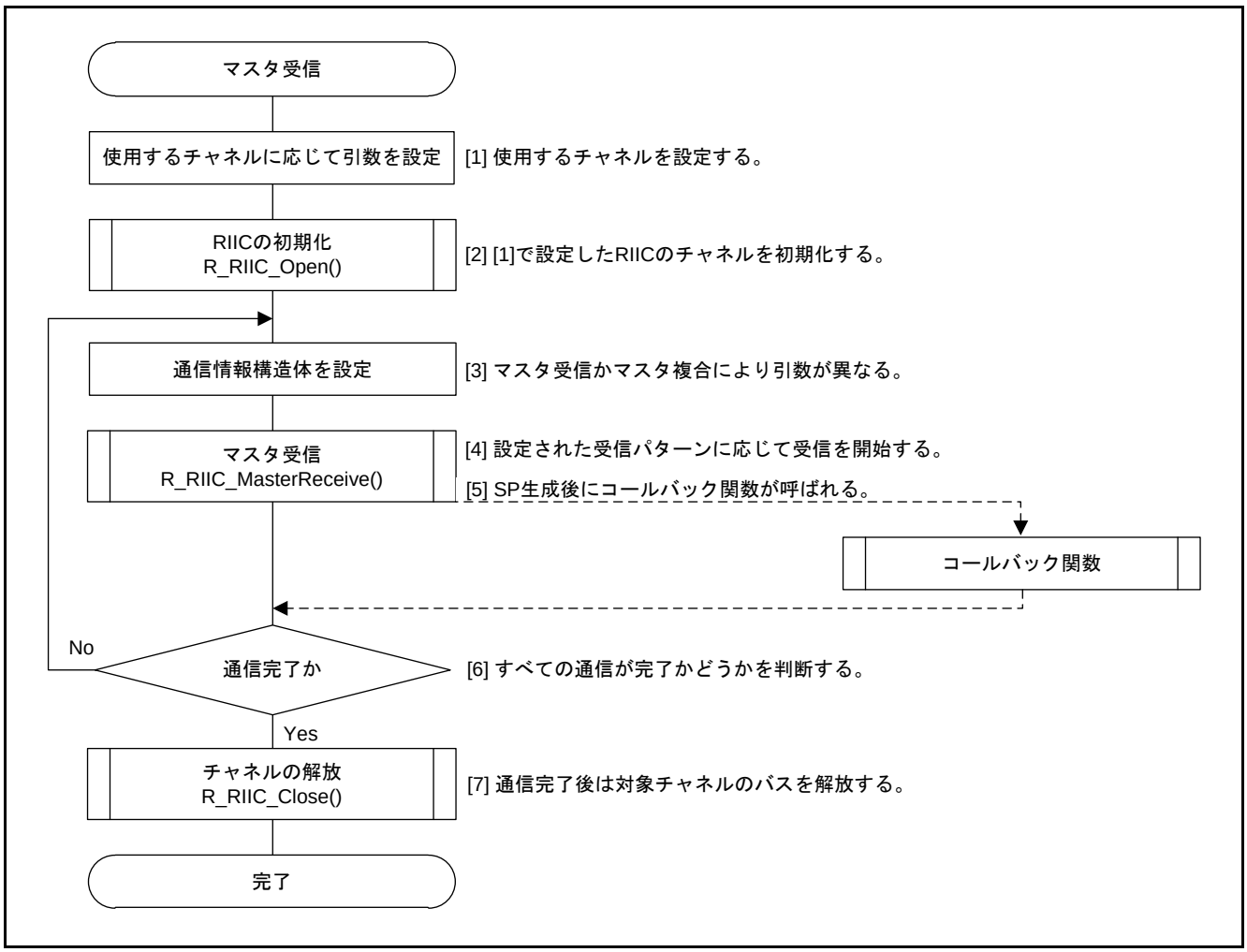


図 1.9　マスタ受信の処理例

1.3.4 スレーブ送受信の処理

マスタデバイスから送信されるデータを、スレーブデバイスとして受信します。

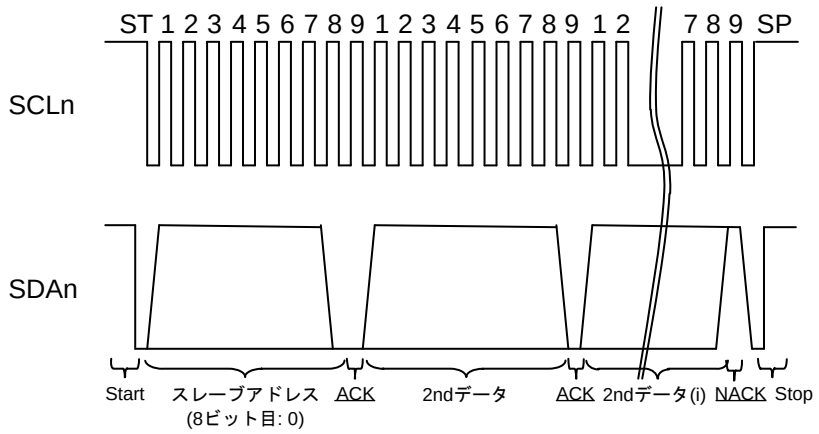
マスタデバイスからの送信要求により、スレーブデバイスとしてデータを送信します。

マスタデバイスが指定するスレーブアドレスが、「r_riic_config.h」で設定したスレーブデバイスのスレーブアドレスと一致したとき、スレーブ送受信を開始します。スレーブアドレスの 8 ビット目(転送方向指定ビット）によって、本モジュールが自動的にスレーブ受信かスレーブ送信かを判断して処理を行います。

(1) スレーブ受信

スレーブデバイスとして、マスタデバイスからのデータを受信する機能です。

マスタデバイスが生成したスタートコンディション(ST)を検出した後に、受信したスレーブアドレスが、自アドレスと一致し、かつスレーブアドレスの 8 ビット目(転送方向指定ビット）が“0”(Write)のとき、スレーブデバイスとして受信動作を開始します。最終データ(I²C 通信情報構造体に設定された受信データ数)を受信時は、NACK を返すことでマスタデバイスに必要なデータをすべて受信したことを通知します。図 1.10にスレーブ受信の信号図を示します。



凡例:
n: チャネル番号
ST: スタートコンディション生成
SP: ストップコンディション生成
NACK: Acknowledge “1”
ACK: Acknowledge “0”
※ 下線のある信号は、スレーブからマスタへのデータ送信です

図 1.10 スレーブ受信 信号図

図 1.11にスレーブ受信を行う際の手順を示します。コールバック関数は、ストップコンディション検出後に呼ばれます。I²C 通信情報構造体メンバの CallBackFunc に関数名を指定してください。

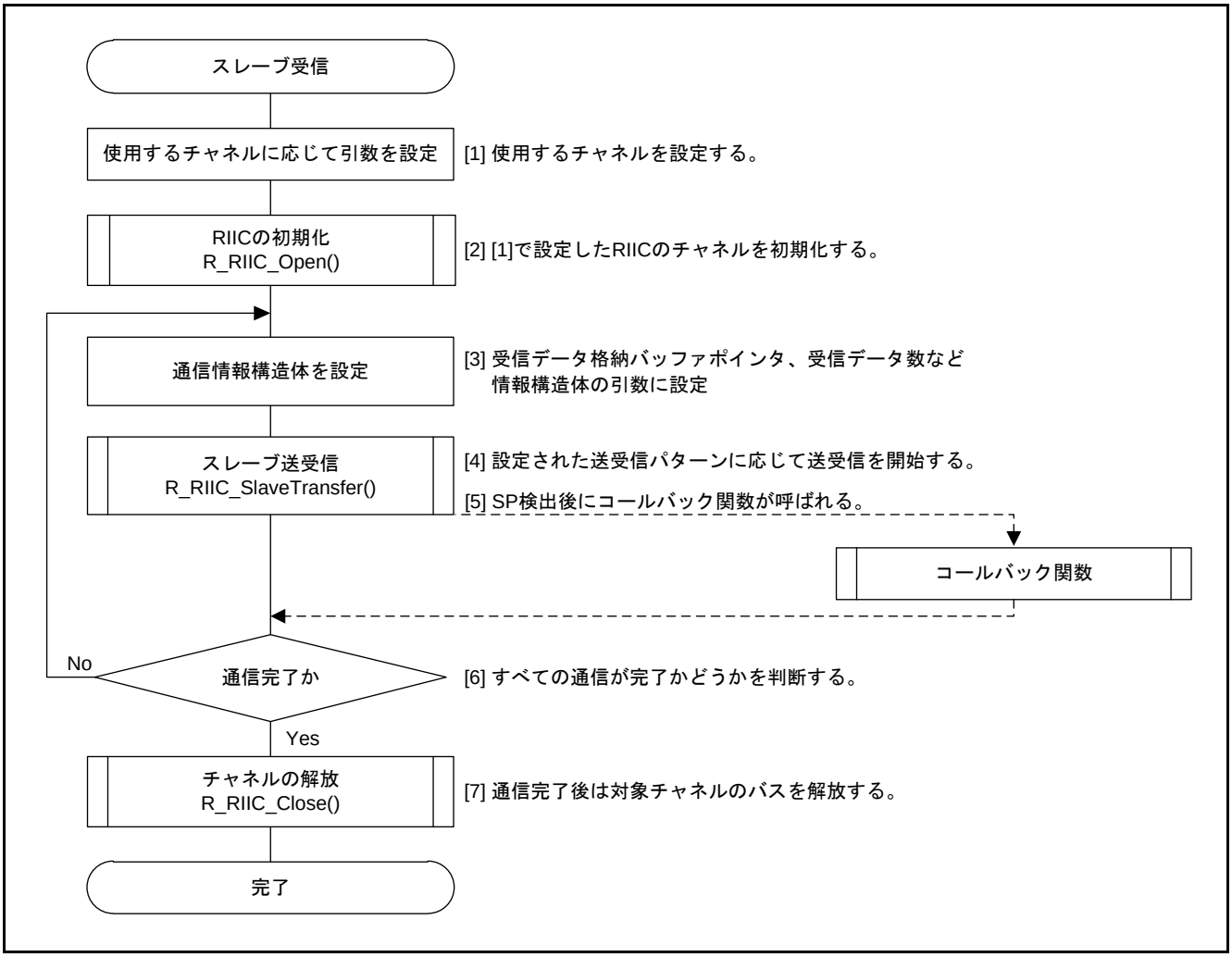


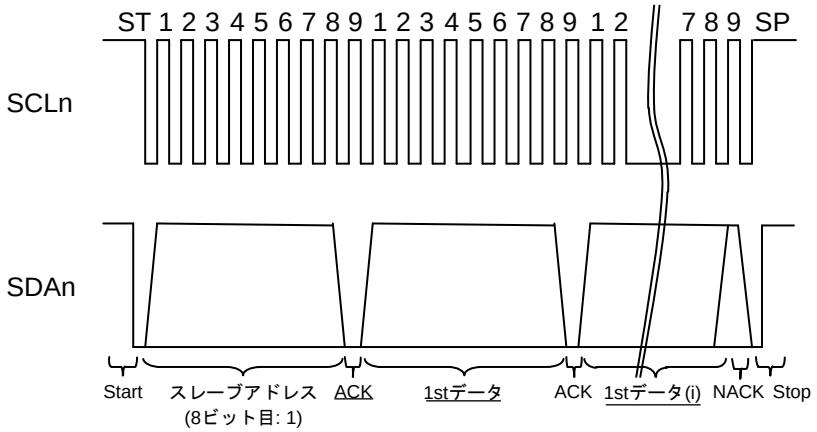
図 1.11　スレーブ受信の処理例

(2) スレーブ送信

スレーブデバイスとして、マスタデバイスへデータを送信する機能です。

マスタデバイスからのスタートコンディション(ST)検出後に、受信したスレーブアドレスが、自アドレスと一致し、かつスレーブアドレスの 8 ビット目(転送方向指定ビット）が“1”(Read)のとき、スレーブデバイスとして送信動作を開始します。設定したデータ数(IIC 通信情報構造体に設定した送信データ数)を超える送信データの要求があった場合、“0xFF”をデータとして送信します。ストップコンディション(SP)を検出するまでデータを送信します。

図 1.12にスレーブ送信の信号図を示します。



凡例:
n: チャネル番号
ST: スタートコンディション生成　　NACK: Acknowledge “1”
SP: ストップコンディション生成　　ACK: Acknowledge “0”
※ 下線のある信号は、スレーブからマスタへのデータ送信です

図 1.12　スレーブ送信 信号図

図 1.13にスレーブ送信を行う際の手順を示します。コールバック関数は、ストップコンディション検出後に呼ばれます。I²C 通信情報構造体メンバの CallBackFunc に関数名を指定してください。

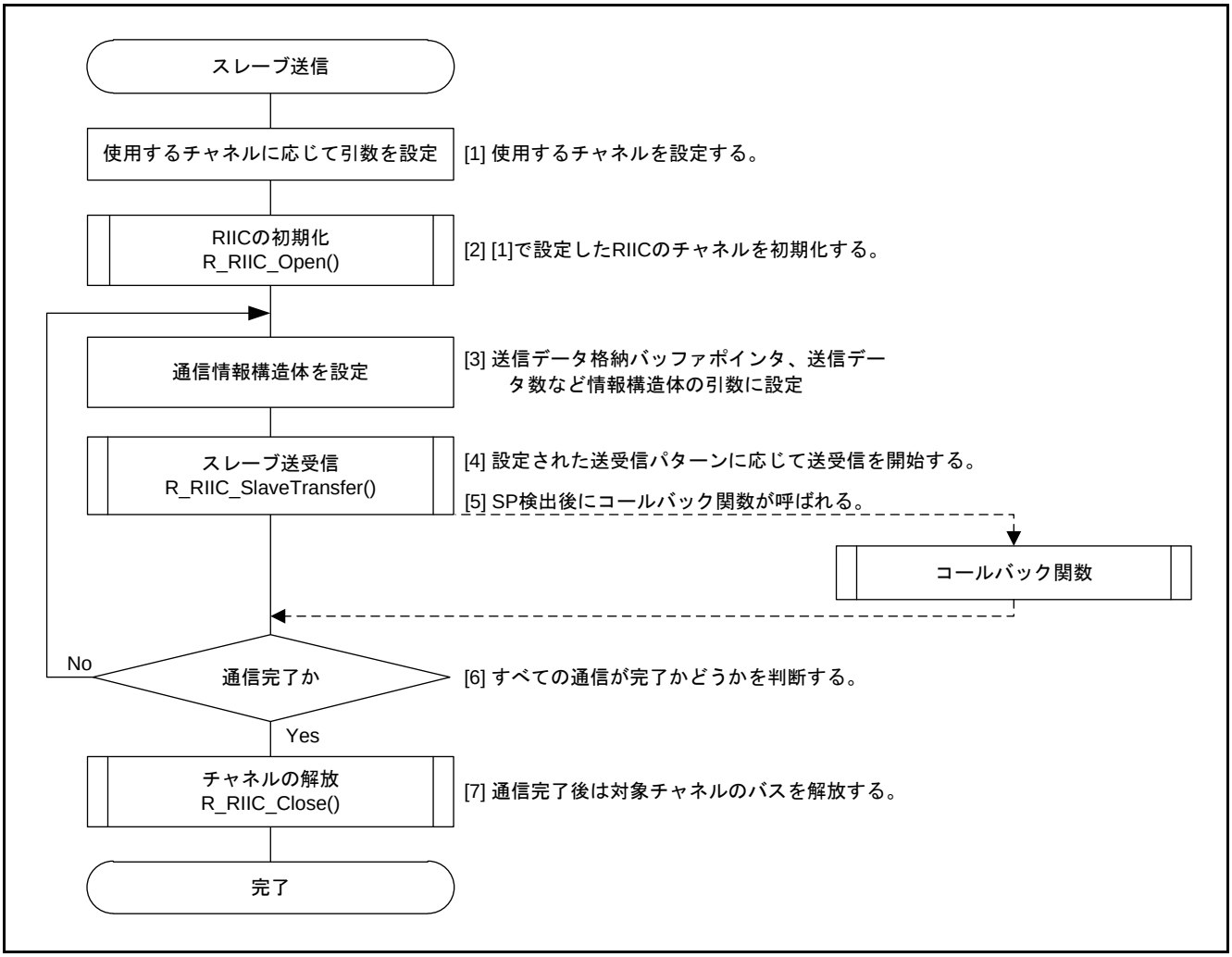


図 1.13　スレーブ送信の処理例

1.3.5 状態遷移図

本モジュールの状態遷移図を図 1.14に示します。

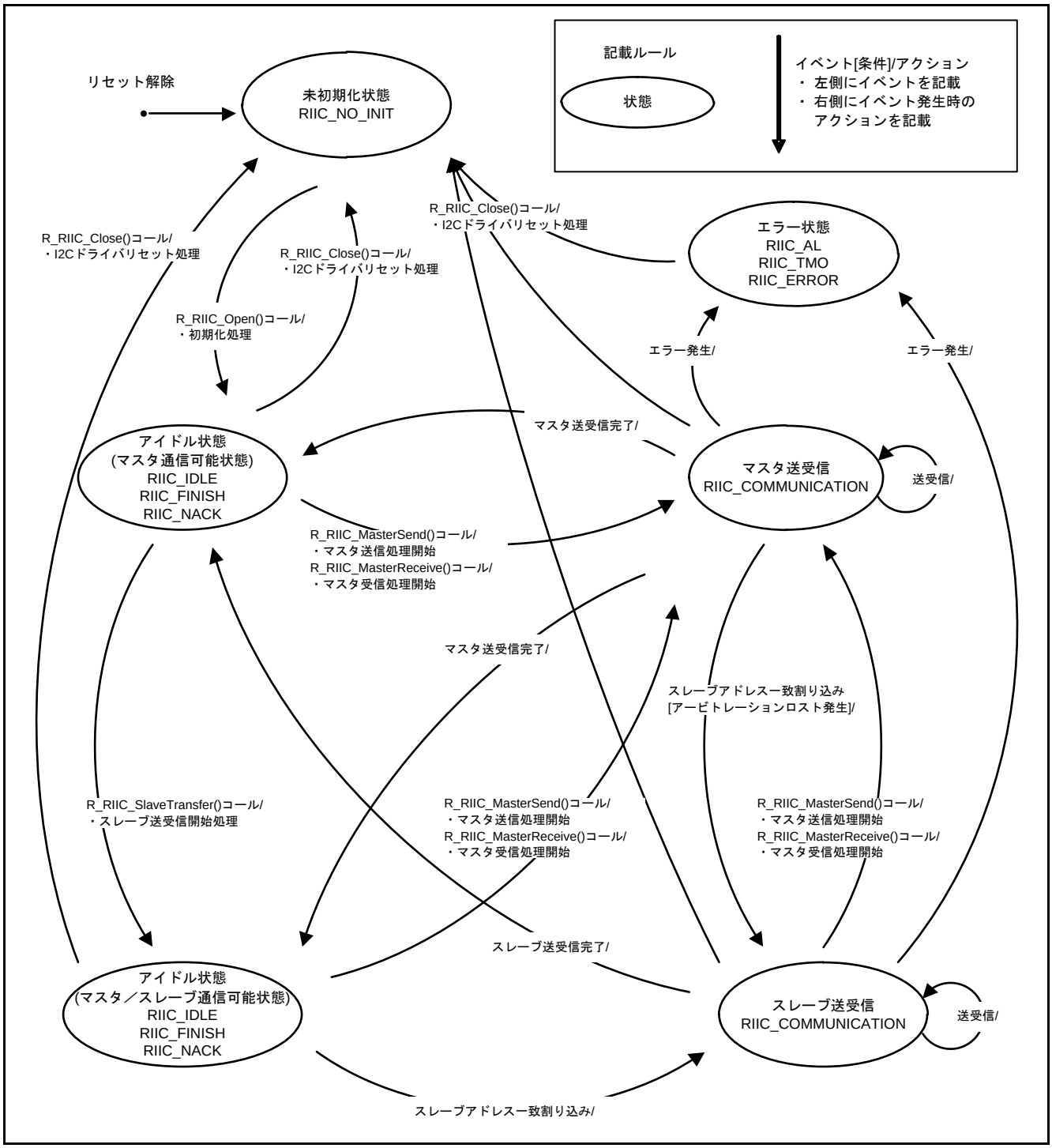


図 1.14　RIIC FIT モジュールの状態遷移図

1.3.6 状態遷移時の各フラグ

I²C 通信情報構造体メンバには、デバイス状態フラグ(dev_sts)があります。デバイス状態フラグには、そのデバイスの通信状態が格納されます。また、このフラグにより、同一チャネル上の複数のスレーブデバイスの制御を行うことができます。表 1.2に状態遷移時のデバイス状態フラグの一覧を示します。

表 1.2　状態遷移時のデバイス状態フラグの一覧

状態	デバイス状態フラグ(dev_sts)
未初期化状態	RIIC_NO_INIT
アイドル状態	RIIC_IDLE RIIC_FINISH RIIC_NACK
通信中 (マスタ送信、マスタ受信、 スレーブ送信、スレーブ受信)	RIIC_COMMUNICATION
アービトレーションロスト検出状態	RIIC_ AL
タイムアウト検出状態	RIIC_TMO
エラー	RIIC_ERROR

1.3.7 アービトレーションロスト検出機能

本モジュールは、以下に示すアービトレーションロストを検出することができます。なお、RIIC は、以下に加えて、スレーブ送信時におけるアービトレーションロストの検出にも対応していますが、本モジュールは対応していません。

(1) バスビジー状態で、スタートコンディションを発行したとき

既に他のマスタデバイスがスタートコンディションを発行して、バスを占有している状態(バスビジー状態)でスタートコンディションを発行すると、本モジュールはアービトレーションロストを検出します。

(2) バスビジー状態ではないが、他のマスタより遅れてスタートコンディションを発行したとき

本モジュールは、スタートコンディションを発行するとき、SDA ラインを Low にしようとします。しかし、他のマスタデバイスがこれよりも早くスタートコンディションを発行した場合、SDA ライン上の信号レベルは、本モジュールが出力したレベルと一致しなくなります。このとき、本モジュールはアービトレーションロストを検出します。

(3) スタートコンディションが同時に発行されたとき

複数のマスタデバイスが、同時にスタートコンディションを発行すると、それぞれのマスタデバイス上でスタートコンディションの発行が正常に終了したと判断されることがあります。その後、それぞれのマスタデバイスは通信を開始しますが、以下に示す条件が成立すると、本モジュールはアービトレーションロストを検出します。

a. それぞれのマスタデバイスが送信するデータが異なる場合

データ通信中、本モジュールは SDA ライン上の信号レベルと、本モジュールが出力したレベルを比較しています。そのため、スレーブアドレス送信を含むデータ送信中に、SDA ライン上と本モジュールが出力したレベルが一致しなくなると、本モジュールはその時点でアービトレーションロストを検出します。

b. それぞれのマスタデバイスが送信するデータは同じだが、データの送信回数が異なる場合

上記 a.に合致しない場合(スレーブアドレスおよび送信データが同じ)、本モジュールはアービトレーションを検出しませんが、それぞれのマスタデバイスがデータを送信する回数が異なる場合であれば、本モジュールはアービトレーションロストを検出します。

1.3.8 タイムアウト検出機能

本モジュールは、タイムアウト検出機能を有効にすることができます(デフォルト有効)。タイムアウト検出機能では SCL ラインが Low または High に固定されたまま一定時間以上経過したことを検知し、バスの異常状態を検出します。

タイムアウト検出機能は以下の期間で SCL ラインの Low 固定または High 固定のバスハングアップを検出します。

(1) マスタモードで、バスビジー

(2) スレーブモードで、自スレーブアドレス一致かつバスビジー

(3) スタートコンディション発行要求中で、バスフリー

タイムアウト検出機能の有効/無効の設定方法については、「2.7 コンパイル時の設定」の

RIIC_CFG_CH0_TMO_ENABLE、RIIC_CFG_CH2_TMO_ENABLE、
RIIC_CFG_CH0_TMO_DET_TIME、RIIC_CFG_CH2_TMO_DET_TIME、
RIIC_CFG_CH0_TMO_LCNT、RIIC_CFG_CH2_TMO_LCNT、

RIIC_CFG_CH0_TMO_HCNT、RIIC_CFG_CH2_TMO_HCNT
を参照ください。
タイムアウト検出時の対応方法については、「5.3 タイムアウトの検出、および検出後の処理」を参照ください。

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- RIIC

2.2 ソフトウェアの要求

FIT モジュールは以下の FIT モジュールに依存しています。

- ボードサポートパッケージモジュール (r_bsp)

2.3 サポートされているツールチェーン

本 FIT モジュールは下記ツールチェーンで動作確認を行っています。詳細は、「5.4 動作確認環境」を参照ください。

- Renesas RX Toolchain v.2.01.01
- Renesas RX Toolchain v.2.03.00
- Renesas RX Toolchain v.2.05.00
- Renesas RX Toolchain v.2.06.00
- Renesas RX Toolchain v.2.07.00

2.4 使用する割り込みベクタ

(マクロ定義 RIIC_CFG_CHi_INCLUDED(i = 0～2)が 1 の時)、R_RIIC_MasterSend 関数、R_RIIC_MasterReceive 関数、R_RIIC_SlaveTransfer 関数を呼び出したとき、(引数のパラメータで指定したチャネル番号のチャネルに対応した)EEI 割り込み、RXI 割り込み、TXI 割り込み、TEI 割り込みが有効になります。表 2.1に RIIC FIT モジュールが使用する割り込みベクタを示します。

表 2.1　使用する割り込みベクター覧

デバイス	割り込みベクタ
RX110 RX111 RX113 RX130 RX230 RX231 RX23T RX24T RX24U	EEI0 割り込み[チャネル 0]（ベクタ番号: 246） RXI0 割り込み[チャネル 0]（ベクタ番号: 247） TXI0 割り込み[チャネル 0]（ベクタ番号: 248） TEI0 割り込み[チャネル 0]（ベクタ番号: 249）
RX64M RX71M	RXI0 割り込み[チャネル 0]（ベクタ番号: 52） TXI0 割り込み[チャネル 0]（ベクタ番号: 53） RXI2 割り込み[チャネル 2]（ベクタ番号: 54） TXI2 割り込み[チャネル 2]（ベクタ番号: 55） GROUPBL1 割り込み（ベクタ番号: 111） • TEI0 割り込み[チャネル 0]（グループ割り込み要因番号: 13） • EEI0 割り込み[チャネル 0]（グループ割り込み要因番号: 14） • TEI2 割り込み[チャネル 2]（グループ割り込み要因番号: 15） • EEI2 割り込み[チャネル 2]（グループ割り込み要因番号: 16）
RX65N RX651	RXI0 割り込み[チャネル 0]（ベクタ番号: 52） TXI0 割り込み[チャネル 0]（ベクタ番号: 53） RXI1 割り込み[チャネル 1]（ベクタ番号: 50） TXI1 割り込み[チャネル 1]（ベクタ番号: 51） RXI2 割り込み[チャネル 2]（ベクタ番号: 54） TXI2 割り込み[チャネル 2]（ベクタ番号: 55） GROUPBL1 割り込み（ベクタ番号: 111） • TEI0 割り込み[チャネル 0]（グループ割り込み要因番号: 13） • EEI0 割り込み[チャネル 0]（グループ割り込み要因番号: 14） • TEI1 割り込み[チャネル 1]（グループ割り込み要因番号: 28） • EEI1 割り込み[チャネル 1]（グループ割り込み要因番号: 29） • TEI2 割り込み[チャネル 2]（グループ割り込み要因番号: 15） • EEI2 割り込み[チャネル 2]（グループ割り込み要因番号: 16）

2.5 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_riic_rx_if.h に記載しています。

2.6 整数型

このプロジェクトは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.7 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_riic_rx_config.h、r_riic_rx_pin_config.h で行います。

オプション名および設定値に関する説明を、下表に示します。

Configuration options in *r_riic_rx_config.h*	
RIIC_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は“1”	パラメータチェック処理をコードに含めるか選択できます。 “0”を選択すると、パラメータチェック処理をコードから省略できるため、コードサイズが削減できます。 “0”の場合、パラメータチェック処理をコードから省略します。 “1”の場合、パラメータチェック処理をコードに含めます。
RIIC_CFG_CHi_INCLUDED (注 1) i=0～2 ※i = 0 のデフォルト値は“1” ※i = 1,2 のデフォルト値は“0”	該当チャネルを使用するかを選択できます。 該当チャネルを使用しない場合は“0”に設定してください。 “0”の場合、該当チャネルに関する処理をコードから省略します。 “1”の場合、該当チャネルに関する処理をコードに含めます。
RIIC_CFG_CH0_kBPS ※デフォルト値は“400”	RIIC0 の通信速度を設定できます。 “RIIC_CFG_CH0_kBPS”と周辺クロックからビットレートレジスタおよび内部基準クロック選択ビットへの設定値が算出されます。 [転送速度がファストモードプラスに対応しない対象デバイス] “400”以下の値を設定してください。 [RX64M、RX71M、RX65N] “1000”以下の値を設定してください。
RIIC_CFG_CH1_kBPS (注 1) ※デフォルト値は“400” ※チャネル 2 をサポートしない対象デバイスでは本設定は無効です。	RIIC1 の通信速度を設定できます。 “RIIC_CFG_CH1_kBPS”と周辺クロックからビットレートレジスタおよび内部基準クロック選択ビットへの設定値が算出されます。 “400”以下の値を設定してください。
RIIC_CFG_CH2_kBPS (注 1) ※デフォルト値は“400” ※チャネル 2 をサポートしない対象デバイスでは本設定は無効です。	RIIC2 の通信速度を設定できます。 “RIIC_CFG_CH2_kBPS”と周辺クロックからビットレートレジスタおよび内部基準クロック選択ビットへの設定値が算出されます。 “400”以下の値を設定してください。
RIIC_CFG_CHi_DIGITAL_FILTER i=0～2 ※i = 0～2 のデフォルト値は“2”	指定した RIIC のノイズフィルタの段数を選択できます。 “0”の場合、ノイズフィルタは無効となります。 “1”～“4”の場合、選択した段数のフィルタが有効になるようノイズフィルタ段数選択ビットおよびデジタルノイズフィルタ回路有効ビットの設定値が選択されます。
RIIC_CFG_PORT_SET_PROCESSING ※デフォルト値は“1”	R_RIIC_CFG_RIICi_SCLi_PORT、R_RIIC_CFG_RIICi_SCLi_BIT、R_RIIC_CFG_RIICi_SDAi_PORT、R_RIIC_CFG_RIICi_SDAi_BIT で選択したポートを SCL、SDA 端子として使用するための設定処理をコードに含めるかを選択します。 “0”の場合、ポートの設定処理をコードから省略します。 “1”の場合、ポートの設定処理をコードに含めます。

注1. 該当チャネルをサポートしない対象デバイスでは本設定は無効です。

Configuration options in *r_riic_rx_config.h*	
RIIC_CFG_CHi_MASTER_MODE (注 2) i=0～2 ※i = 0～2 のデフォルト値は“1”	指定した RIIC のマスタアービトレーションロスト検出機能の有効/無効を選択できます。 マルチマスタで使用する場合は、“1”(有効)にしてください。 “0”の場合、マスタアービトレーションロスト検出を無効にします。 “1”の場合、マスタアービトレーションロスト検出を有効にします。
RIIC_CFG_CHi_SLV_ADDR0_FORMAT(※1) (注 2) RIIC_CFG_CHi_SLV_ADDR1_FORMAT(※2) (注 2) RIIC_CFG_CHi_SLV_ADDR2_FORMAT(※2) (注 2) i=0～2 ※1 i = 0～2 のデフォルト値は“1” ※2 i = 0～2 のデフォルト値は“0”	指定した RIIC のスレーブアドレスのフォーマットを 7 ビット/10 ビットから選択できます。 “0”の場合、スレーブアドレスを設定しません。 “1”の場合、7 ビットアドレスフォーマットに設定します。 “2”の場合、10 ビットアドレスフォーマットに設定します。
RIIC_CFG_CHi_SLV_ADDR0(※1) (注 2) RIIC_CFG_CHi_SLV_ADDR1(※2) (注 2) RIIC_CFG_CHi_SLV_ADDR2(※2) (注 2) i=0～2 ※1 i = 0～2 のデフォルト値は“0x0025” ※2 i = 0～2 のデフォルト値は“0x0000”	指定した RIIC のスレーブアドレスを設定します。 “RIIC_CFG_CHj_SLV_ADDRi_FORMAT”の設定値によって、設定可能範囲が変わります。“RIIC_CFG_CHj_SLV_ADDRi_FORMAT” (j=0~2)が “0”の場合は、設定値は無効となります。 “1”の場合は、設定値の下位 7 ビットが有効となります。 “2”の場合は、設定値の下位 10 ビットが有効となります。
RIIC_CFG_CHi_SLV_GCA_ENABLE (注 2) i=0～2 ※ i = 0～2 のデフォルト値は“0”	指定した RIIC のゼネラルコールアドレスの有効/無効が選択できます。 “0”の場合、ゼネラルコールアドレスを無効にします。 “1”の場合、ゼネラルコールアドレスを有効にします。
RIIC_CFG_CHi_RXI_INT_PRIORITY (注 2) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIIC の受信データフル割り込み(RXI0)の優先レベルを選択できます。 “1”～”15”の範囲で設定してください。
RIIC_CFG_CHi_TXI_INT_PRIORITY (注 2) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIICの送信データエンプティ割り込み(TXI0)の優先レベルを選択できます。 “1”～”15”の範囲で設定してください。
RIIC_CFG_CHi_EEI_INT_PRIORITY(注 1) (注 2) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIICの通信エラー/イベント発生割り込み(EEIi)の優先レベルを選択できます。 “1”～”15”の範囲で設定してください。 RIIC_CFG_CHi_RXI_INT_PRIORITY、RIIC_CFG_CHi_TXI_INT_PRIORITY で指定した優先レベルの値より低い値を設定しないでください。
RIIC_CFG_CHi_TEI_INT_PRIORITY(注 1) (注 2) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIIC の送信終了割り込み(TEIi)の優先レベルを選択できます。 “1”～”15”の範囲で設定してください。 RIIC_CFG_CHi_RXI_INT_PRIORITY、RIIC_CFG_CHi_TXI_INT_PRIORITY で指定した優先レベルの値より低い値を設定しないでください。

注1. EEIi、TEIi (i = 0～2)がグループ BL1 割り込みとしてグループ化されているデバイスでは、優先レベルを個別に設定することはできません。その場合の EEIi、TEIi (i = 0～2)の優先レベルは、r_riic_confg.h で設定された各優先レベルの中で最大の値に統一されます。ただし、RIIC 以外のモジュールで既にグループ BL1 の割り込み優先レベルが設定されていた場合は、より大きい値に統一されます。また、EEIi、TEIi (i = 0～2)は RXIi、TXIi (i = 0～2)の優先レベル未満には設定しないでください。

注2. 該当チャネルをサポートしない対象デバイスでは本設定は無効です。

Configuration options in r_riic_rx_config.h	
RIIC_CFG_CHi_TMO_ENABLE (注 1) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIIC のタイムアウト検出機能を有効にできます。 “0”の場合、RIICi のタイムアウト検出機能無効 “1”の場合、RIICi のタイムアウト検出機能有効
RIIC_CFG_CHi_TMO_DET_TIME (注 1) i=0～2 ※ i = 0～2 のデフォルト値は“0”	指定した RIIC のタイムアウト検出時間を選択できます。 “0”の場合、ロングモードを選択。 “1”の場合、ショートモードを選択。
RIIC_CFG_CHi_TMO_LCNT (注 1) i=0～2 ※ i = 0～2 のデフォルト値は“1”	指定した RIIC のタイムアウト検出機能有効時、SCLi ラインが Low 期間中にタイムアウト検出機能の内部カウンタのカウントアップを有効にできます。 “0”の場合、SCLi ラインが Low 期間中のカウントアップ禁止。 “1”の場合、SCLi ラインが Low 期間中のカウントアップ有効。
RIIC_CFG_CHi_TMO_HCNT (注 1) i=0～2 ※ i = 0～2 のデフォルト値は“1”	RIIC0 のタイムアウト検出機能有効時、SCLi ラインが High 期間中にタイムアウト検出機能の内部カウンタのカウントアップを有効にできます。 “0”の場合、SCLi ラインが High 期間中のカウントアップ禁止。 “1”の場合、SCLi ラインが High 期間中のカウントアップ有効。
RIIC_CFG_BUS_CHECK_COUNTER ※デフォルト値は“1000”	RIIC の API 関数のバスチェック処理時に、ソフトウェアによりタイムアウトカウンタ(バス確認回数)を設定できます。 “0xFFFFFFFF”以下の値を設定してください。 バスチェック処理は、 ・スタートコンディション生成前 ・ストップコンディション検出後 ・RIIC 制御機能(R_RIIC_Control 関数)を使用した 　各コンディションおよび SCL ワンショットパルスの生成後 に行います。 バスチェック処理では、バスビジー時、バスフリーになるまでソフトウェアによりタイムアウトカウンタをデクリメントします。“0”になるとタイムアウトと判断し、戻り値でエラー（Busy）を返します。 **※バスがロックされないようにするためのカウンタであるため、相手デバイスが SCL 端子を“L”ホールドする時間以上になるよう値を設定してください。** 「タイムアウト時間(ns) ≒ (1 / ICLK(Hz)) * カウンタ値 * 10」

注1. 該当チャネルをサポートしない対象デバイスでは本設定は無効です。

Configuration options in *r_riic_rx_pin_config.h*	
R_RIIC_CFG_RIICi_SCLi_PORT i=0～2 ※i = 0 のデフォルト値は'1' ※i = 1 のデフォルト値は'2' ※i = 2 のデフォルト値は'1'	SCL端子として使用するポートグループを選択します。 '0'～'J' (ASCIIコード)の範囲で設定してください。
R_RIIC_CFG_RIICi_SCLi_BIT i=0～2 ※i = 0 のデフォルト値は'2' ※i = 1 のデフォルト値は'1' ※i = 2 のデフォルト値は'6'	SCL端子として使用する端子を選択します。 '0'～'7' (ASCIIコード)の範囲で設定してください。
R_RIIC_CFG_RIICi_SDAi_PORT i=0～2 ※i = 0 のデフォルト値は'1' ※i = 1 のデフォルト値は'2' ※i = 2 のデフォルト値は'1'	SDA端子として使用するポートグループを選択します。 '0'～'J' (ASCIIコード)の範囲で設定してください。
R_RIIC_CFG_RIICi_SDAi_BIT i=0～2 ※i = 0 のデフォルト値は'3' ※i = 1 のデフォルト値は'0' ※i = 2 のデフォルト値は'7'	SDA端子として使用する端子を選択します。 '0'～'7' (ASCIIコード)の範囲で設定してください。

2.8 コードサイズ

本モジュールのコードサイズを下表に示します。RX100 シリーズ、RX200 シリーズ、RX600 シリーズから代表して 1 デバイスずつ掲載しています。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。掲載した値は、「2.3 サポートされているツールチェーン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル：2、最適化のタイプ：サイズ優先、データ・エンディアン：リトルエンディアンです。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

ROM、RAM およびスタックのコードサイズ

デバイス	分類		使用メモリ		備考
			パラメータチェック処理あり	パラメータチェック処理なし	
RX130	ROM	1 チャネル使用	9170 バイト	8887 バイト	
	RAM	1 チャネル使用	37 バイト		
	最大使用スタックサイズ		396 バイト		多重割り込み禁止のため 1 チャネル使用時の最大値のみを記載しています。
RX231	ROM	1 チャネル使用	9105 バイト	8822 バイト	
	RAM	1 チャネル使用	37 バイト		
	最大使用スタックサイズ		372 バイト		多重割り込み禁止のため 1 チャネル使用時の最大値のみを記載しています。
RX64M	ROM	1 チャネル使用	9195 バイト	8912 バイト	
		2 チャネル使用	10057 バイト	9774 バイト	
	RAM	1 チャネル使用	111 バイト		
		2 チャネル使用	111 バイト		
	最大使用スタックサイズ		360 バイト		多重割り込み禁止のため 1 チャネル使用時の最大値のみを記載しています。

2.9 引数

API 関数の引数である構造体を示します。この構造体は、API 関数のプロトタイプ宣言とともに r_riic_rx_if.h に記載されています。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

```
typedef volatile struct
{
    uint8_t                 rsv2;           /* 予約領域 */
    uint8_t                 rsv1;           /* 予約領域 */
    riic_ch_dev_status_t    dev_sts;        /* デバイス状態フラグ */
    uint8_t                 ch_no;          /*使用するデバイスのチャネル番号 */
    riic_callback           callbackfunc;   /* コールバック関数 */
    uint32_t                cnt2nd;         /* 2nd データカウンタ(バイト数) */
    uint32_t                cnt1st;         /* 1st データカウンタ(バイト数) */
    uint8_t *               p_data2nd;      /* 2nd データ格納バッファポインタ */
    uint8_t *               p_data1st;      /* 1st データ格納バッファポインタ */
    uint8_t *               p_slv_adr;      /* スレーブアドレスのバッファポインタ */
} riic_info_t;
```

2.10 戻り値

API 関数の戻り値を示します。この列挙型は、API 関数のプロトタイプ宣言とともに r_riic_rx_if.h で記載されています。

```
typedef enum
{
    RIIC_SUCCESS = 0U,          /* 関数の処理が正常に終了した場合 */
    RIIC_ERR_LOCK_FUNC,         /* 他のモジュールで RIIC が使用されている場合 */
    RIIC_ERR_INVALID_CHAN,      /* 存在しないチャネルを指定した場合 */
    RIIC_ERR_INVALID_ARG,       /* 不正な引数を設定した場合 */
    RIIC_ERR_NO_INIT,           /* 未初期化状態の場合 */
    RIIC_ERR_BUS_BUSY,          /* バスビジーの場合 */
    RIIC_ERR_AL,                /* アービトレーションロスト検出状態で関数を呼び出した場合 */
    RIIC_ERR_TMO,               /* タイムアウトを検出した場合 */
    RIIC_ERR_OTHER              /* その他エラー */
} riic_return_t;
```

2.11 コールバック関数

本モジュールでは、以下のいずれかの条件を満たし EEI 割り込み要求が発生したときに、ユーザが設定したコールバック関数を呼び出します。

(1) 通信動作（マスタ送信、マスタ受信、マスタ送受信、スレーブ送信、スレーブ受信）が完了し、ストップコンディションを発行した。

(2) 通信動作（マスタ送信、マスタ受信、マスタ送受信、スレーブ送信、スレーブ受信）中にタイムアウトを検出した。(注 1)

注1. タイムアウト検出機能(2.7 コンパイル時の設定 RIIC_CFG_CHi_TMO_ENABLE(i=0～2))を有効にしている場合。

コールバック関数は、「2.9 引数」に記載された構造体メンバ“callbackfunc”に、コールバック関数のアドレスを格納し、R_RIIC_MasterSend 関数、R_RIIC_MasterReceive 関数、R_RIIC_SlaveTransfer 関数を呼び出したときに設定されます。

コールバック関数内では R_RIIC_GetStatus 関数以外の API 関数の呼び出しは禁止です。

2.12 モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては、(2)、(4)の方法を使用してください。

(1) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e² studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

3. API 関数

3.1 R_RIIC_Open()

この関数は RIIC FIT モジュールを初期化する関数です。この関数は他の API 関数を使用する前に呼び出される必要があります。

Format

```
riic_return_t R_RIIC_Open(
        riic_info_t *       p_riic_info         /* 構造体データ */
)
```

Parameters

p_riic_info

I²C 通信情報構造体のポインタ。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t        dev_sts;        /* デバイス状態フラグポインタ (更新あり) */
uint8_t                     ch_no;          /* チャネル番号 */
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_LOCK_FUNC      /* 他のタスクが API をロックしている場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
RIIC_ERR_OTHER          /* 現在の状態に該当しない不正なイベントが発生した場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC の通信を開始するための初期設定をします。引数で指定した RIIC のチャネルを設定します。チャネルの状態が“未初期化状態 (RIIC_NO_INIT)”の場合、次の処理を行います。

- 状態フラグの設定
- ポートの入出力設定
- I²C 出力ポートの割り当て
- RIIC のモジュールストップ状態の解除
- API で使用する変数の初期化
- RIIC 通信で使用する RIIC レジスタの初期化
- RIIC 割り込みの禁止

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
volatile riic_return_t  ret;
riic_info_t             iic_info_m;

iic_info_m.dev_sts = RIIC_NO_INIT;
iic_info_m.ch_no   = 0;

ret = R_RIIC_Open(&iic_info_m);
```

Special Notes:

なし

3.2 R_RIIC_MasterSend()

マスタ送信を開始します。引数に合わせてマスタのデータ送信パターンを変更します。ストップコンディション生成まで一括で実施します。

Format

```
riic_return_t R_RIIC_MasterSend(
        riic_info_t *       p_riic_info         /* 構造体データ */
)
```

Parameters

**p_riic_info*

I²C 通信情報構造体のポインタ。引数によって、送信パターン(4 パターンあります)を変更できます。各送信パターンの指定方法および引数の設定可能範囲は、「Special Notes」を参照ください。また、送信パターンの波形のイメージは「1.3.2 マスタ送信の処理」を参照ください。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

スレーブアドレスを設定する際、1 ビット左シフトせずに格納してください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t    dev_sts;        /* デバイス状態フラグ(更新あり) */
uint8_t                 ch_no;          /* チャネル番号 */
riic_callback           callbackfunc;   /* コールバック関数 */
uint32_t                cnt2nd;         /* 2nd データカウンタ(バイト数)
                                           (パターン 1、2 のみ更新あり) */
uint32_t                cnt1st;         /* 1st データカウンタ(バイト数)
                                           (パターン 1 のみ更新あり) */
uint8_t *               p_data2nd;      /* 2nd データ格納バッファポインタ */
uint8_t *               p_data1st;      /* 1st データ格納バッファポインタ */
uint8_t *               p_slv_adr;      /* スレーブアドレスのバッファポインタ */
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
RIIC_ERR_NO_INIT        /* 初期設定ができていない場合 (未初期化状態) */
RIIC_ERR_BUS_BUSY       /* バスビジーの場合 */
RIIC_ERR_AL             /* アービトレーションエラーが発生した場合 */
RIIC_ERR_TMO            /* タイムアウトを検出した場合 */
RIIC_ERR_OTHER          /* 現在の状態に該当しない不正なイベントが発生した場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC のマスタ送信を開始します。引数で指定した RIIC のチャネル、送信パターンで送信します。チャネルの状態が“アイドル状態”(RIIC_IDLE、RIIC_FINISH、RIIC_NACK)の場合、次の処理を行います。

- 状態フラグの設定
- API で使用する変数の初期化
- RIIC 割り込みの許可
- スタートコンディションの生成

スタートコンディションの生成処理までが正常に終了した時、本関数は戻り値として RIIC_SUCCESS を返します。

スタートコンディションの生成時に下記条件に該当した時､本関数は戻り値として RIIC_ERR_BUS_BUSY を返します。(注 1)

- 内部のステータスビットが BUSY 状態である
- SCL、SDA ラインのいずれかが Low の状態である

送信の処理は、本関数が RIIC_SUCCESS を返した後発生する割り込み処理の中で順次行われます。
使用する割り込みは、「2.4 使用する割り込みベクタ」を参照ください。
マスタ送信の割り込みの発生タイミングは、「5.2.1 マスタ送信」を参照ください。

送信終了でストップコンディションを発行した後、引数で指定したコールバック関数が呼び出されます。

送信が正常に完了したかどうかは、引数で指定したデバイス状態フラグ、またはチャネル状態フラグ g_riic_ChStatus[ ]が”RIIC_FINISH”になっているかどうかで確認することができます。

注1. SCL と SDA 端子が外部回路でプルアップされていない場合、SCL、SDA ラインのいずれかを Low の状態として検出し、RIIC_ERR_BUS_BUSY を返すことがあります。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
/* for MasterSend(Pattern 1) */
#include <stddef.h>
#include "platform.h"
#include "r_riic_rx_if.h"

riic_info_t iic_info_m;

void CallbackMaster(void);
void main(void);

void main(void)
{
    volatile riic_return_t ret;

    uint8_t addr_eeprom[1]   = {0x50};
    uint8_t access_addr1[1]  = {0x00};
    uint8_t mst_send_data[5] = {0x81,0x82,0x83,0x84,0x85};

    /* Sets IIC Information for sending pattern 1. */
    iic_info_m.dev_sts = RIIC_NO_INIT;
    iic_info_m.ch_no = 0;
    iic_info_m.callbackfunc = &CallbackMaster;
    iic_info_m.cnt2nd = 3;
    iic_info_m.cnt1st = 1;
    iic_info_m.p_data2nd = mst_send_data;
    iic_info_m.p_data1st = access_addr1;
```

```
    iic_info_m.p_slv_adr = addr_eeprom;

    /* RIIC open */
    ret = R_RIIC_Open(&iic_info_m);

    /* RIIC send start */
    ret = R_RIIC_MasterSend(&iic_info_m);

    if (RIIC_SUCCESS == ret)
    {
        while(RIIC_FINISH != iic_info_m.dev_sts);
    }
    else
    {
        /* error */
    }

    /* RIIC send complete */
    while(1);
}

void CallbackMaster(void)
{
    volatile riic_return_t ret;
    riic_mcu_status_t      iic_status;

    ret = R_RIIC_GetStatus(&iic_info_m, &iic_status);
    if(RIIC_SUCCESS != ret)
    {
        /* R_RIIC_GetStatus 関数のエラー処理 */
    }
    else
    {
        /* iic_status のステータスフラグを確認して
           タイムアウト、アービトレーションロスト、NACK
           などが検出されていた場合の処理を記述 */
    }
}
```

Special Notes:
送信パターンごとの引数の設定可能範囲は、下表を参照してください。

構造体メンバ	ユーザ設定可能範囲			
	マスタ送信 パターン 1	マスタ送信 パターン 2	マスタ送信 パターン 3	マスタ送信 パターン 4
*p_slv_adr	スレーブアドレス バッファポインタ	スレーブアドレス バッファポインタ	スレーブアドレス バッファポインタ	FIT_NO_PTR (注 1)
*p_data1st	[送信用]1st データ バッファポインタ	FIT_NO_PTR (注 1)	FIT_NO_PTR (注 1)	FIT_NO_PTR (注 1)
*p_data2nd	[送信用]2nd データ バッファポインタ	[送信用]2nd データ バッファポインタ	FIT_NO_PTR (注 1)	FIT_NO_PTR (注 1)
cnt1st	0000 0001h～ FFFF FFFFh (注 2)	0	0	0
cnt2nd	0000 0001h～ FFFF FFFFh (注 2)	0000 0001h～ FFFF FFFFh (注 2)	0	0
callbackfunc	使用する関数名を 指定してください。	使用する関数名を 指定してください。	使用する関数名を 指定してください。	使用する関数名を 指定してください。
ch_no	00h～FFh	00h～FFh	00h～FFh	00h～FFh
dev_sts	デバイス状態 フラグ	デバイス状態 フラグ	デバイス状態 フラグ	デバイス状態 フラグ
rsv1,rsv2	予約領域 (設定無効)	予約領域 (設定無効)	予約領域 (設定無効)	予約領域 (設定無効)

注 1：パターン 2、パターン 3、パターン 4 を使用する場合は、上表のとおり該当の構造体メンバに“FIT_NO_PTR”を入れてください。

注 2：“0”は設定しないでください。

3.3 R_RIIC_MasterReceive()

マスタ受信を開始します。引数に合わせてマスタのデータ受信パターンを変更します。ストップコンディション生成まで一括で実施します。

Format

```
riic_return_t R_RIIC_MasterReceive(
        riic_info_t *       p_riic_info         /* 構造体データ */
)
```

Parameters

**p_riic_info*

I²C 通信情報構造体のポインタ。引数の設定によって、マスタ受信かマスタ送受信を選択できます。マスタ受信およびマスタ送受信の指定方法と引数の設定可能範囲は「Special Notes」を参照ください。また、受信パターンの波形イメージは「1.3.3マスタ受信の処理」を参照ください。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

スレーブアドレスを設定する際、1 ビット左シフトせずに格納してください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t    dev_sts;        /* デバイス状態フラグ(更新あり) */
uint8_t                 ch_no;          /* チャネル番号 */
riic_callback           callbackfunc;   /* コールバック関数 */
uint32_t                cnt2nd;         /* 2nd データカウンタ(バイト数) (更新あり) */
uint32_t                cnt1st;         /* 1st データカウンタ(バイト数)
                                           (マスタ送受信のみ更新あり) */
uint8_t *               p_data2nd;      /* 2nd データ格納バッファポインタ */
uint8_t *               p_data1st;      /* 1st データ格納バッファポインタ */
uint8_t *               p_slv_adr;      /* スレーブアドレスのバッファポインタ */
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
RIIC_ERR_NO_INIT        /* 初期設定ができていない場合 (未初期化状態) */
RIIC_ERR_BUS_BUSY       /* バスビジーの場合 */
RIIC_ERR_AL             /* アービトレーションエラーが発生した場合 */
RIIC_ERR_TMO            /* タイムアウトを検出した場合 */
RIIC_ERR_OTHER          /* 現在の状態に該当しない不正なイベントが発生した場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC のマスタ受信を開始します。引数で指定した RIIC のチャネル、受信パターンで受信します。チャネルの状態が“アイドル状態”(RIIC_IDLE、RIIC_FINISH、RIIC_NACK) の場合、次の処理を行います。

－ 状態フラグの設定
－ API で使用する変数の初期化
－ RIIC 割り込みの許可
－ スタートコンディションの生成

スタートコンディションの生成処理までが正常に終了した時、本関数は戻り値として RIIC_SUCCESS を返します。

スタートコンディションの生成時に下記条件に該当した時､本関数は戻り値として RIIC_ERR_BUS_BUSY を返します。(注 1)

- 内部のステータスビットが BUSY 状態である
- SCL、SDA ラインのいずれかが Low の状態である

受信の処理は、本関数が RIIC_SUCCESS を返した後発生する割り込み処理の中で順次行われます。
使用する割り込みは、「2.4 使用する割り込みベクタ」を参照ください。
マスタ受信の割り込みの発生タイミングは、「5.2.2 マスタ受信」を参照ください。

受信終了でストップコンディションを発行した後、引数で指定したコールバック関数が呼び出されます。

受信が正常に完了したかどうかは、引数で指定したデバイス状態フラグ、またはチャネル状態フラグ g_riic_ChStatus[ ]が”RIIC_FINISH”になっているかどうかで確認することができます。

注1. SCL と SDA 端子が外部回路でプルアップされていない場合、SCL、SDA ラインのいずれかを Low の状態として検出し、RIIC_ERR_BUS_BUSY を返すことがあります。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
#include <stddef.h>
#include "platform.h"
#include "r_riic_rx_if.h"

riic_info_t    iic_info_m;

void CallbackMaster(void);
void main(void);

void main(void)
{
    volatile riic_return_t ret;

    uint8_t addr_eeprom[1]    = {0x50};
    uint8_t access_addr1[1]   = {0x00};
    uint8_t mst_store_area[5] = {0xFF,0xFF,0xFF,0xFF,0xFF};

    /* Sets IIC Information. */
    iic_info_m.dev_sts = RIIC_NO_INIT;
    iic_info_m.ch_no = 0;
    iic_info_m.callbackfunc = &CallbackMaster;
    iic_info_m.cnt2nd = 3;
    iic_info_m.cnt1st = 1;
    iic_info_m.p_data2nd = mst_store_area;
```

```
    iic_info_m.p_data1st = access_addr1;
    iic_info_m.p_slv_adr = addr_eeprom;

    /* RIIC open */
    ret = R_RIIC_Open(&iic_info_m);

    /* RIIC receive start */
    ret = R_RIIC_MasterReceive(&iic_info_m);

    if (RIIC_SUCCESS == ret)
    {
        while(RIIC_FINISH != iic_info_m.dev_sts);
    }
    else
    {
        /* error */
    }

    /* RIIC receive complete */
    while(1);
}

void CallbackMaster(void)
{
    volatile riic_return_t ret;
    riic_mcu_status_t      iic_status;

    ret = R_RIIC_GetStatus(&iic_info_m, &iic_status);
    if(RIIC_SUCCESS != ret)
    {
        /* R_RIIC_GetStatus 関数のエラー処理 */
    }
    else
    {
        /* iic_status のステータスフラグを確認して
           タイムアウト、アービトレーションロスト、NACK
           などが検出されていた場合の処理を記述 */
    }
}
```

Special Notes:
受信パターンごとの引数の設定可能範囲は、下表を参照してください。

構造体メンバ	ユーザ設定可能範囲	
	マスタ受信	マスタ送受信
*p_slv_adr	スレーブアドレス バッファポインタ	スレーブアドレス バッファポインタ
*p_data1st	未使用 (設定無効)	[送信用]1st データ バッファポインタ
*p_data2nd	[受信用]2nd データ バッファポインタ	[受信用]2nd データ バッファポインタ
dev_sts	デバイス状態 フラグ	デバイス状態 フラグ
cnt1st(注 1)	0	0000 0001h～ FFFF FFFFh
cnt2nd	0000 0001h～ FFFF FFFFh (注 2)	0000 0001h～ FFFF FFFFh (注 2)
callbackfunc	使用する関数名を 指定してください。	使用する関数名を 指定してください。
ch_no	00h～FFh	00h～FFh
rsv1,rsv2,rsv3	予約領域 (設定無効)	予約領域 (設定無効)

注 1：1st データが“0”か“0 以外”かで受信パターンが決まります。

注 2：“0”は設定しないでください。

3.4 R_RIIC_SlaveTransfer()

スレーブ送受信を行います。引数のパターンに合わせてデータ送受信パターンを変更します。

Format

```
riic_return_t R_RIIC_SlaveTransfer (
        riic_info_t *       p_riic_info         /* 構造体データ */
)
```

Parameters

**p_riic_info*

I²C 通信情報構造体のポインタ。引数の設定によって、スレーブ受信許可状態かスレーブ送信許可状態、またはその両方を選択できます。引数の設定可能範囲は「Special Notes」を参照ください。また、受信パターンの波形イメージは「図 1.10スレーブ受信 信号図」を、送信パターンの波形イメージは「図 1.12 スレーブ送信 信号図」を参照ください。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t    dev_sts;        /* デバイス状態フラグ(更新あり) */
uint8_t                 ch_no;          /* チャネル番号 */
riic_callback           callbackfunc;   /* コールバック関数 */
uint32_t                cnt2nd;         /* 2nd データカウンタ(バイト数)
                                           (スレーブ受信時のみ更新あり) */
uint32_t                cnt1st;         /* 1st データカウンタ(バイト数)
                                           (スレーブ送信時のみ更新あり) */
uint8_t *               p_data2nd;      /* 2nd データ格納バッファポインタ */
uint8_t *               p_data1st;      /* 1st データ格納バッファポインタ*/
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
RIIC_ERR_NO_INIT        /* 初期設定ができていない場合 (未初期化状態) */
RIIC_ERR_BUS_BUSY       /* バスビジーの場合 */
RIIC_ERR_AL             /* アービトレーションエラーが発生した場合 */
RIIC_ERR_TMO            /* タイムアウトを検出した場合 */
RIIC_ERR_OTHER          /* 現在の状態に該当しない不正なイベントが発生した場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC のスレーブ送信、またはスレーブ受信できる状態にします。マスタ通信中に本関数を呼び出した場合は、エラーとなります。引数で指定した RIIC のチャネルを設定します。チャネルの状態が“アイドル状態(RIIC_IDLE、RIIC_FINISH、RIIC_NACK)”の場合、次の処理を行います。

- 状態フラグの設定
- API で使用する変数の初期化
- RIIC 通信で使用する RIIC レジスタの初期化
- RIIC 割り込みの許可
- スレーブアドレスの設定、スレーブアドレス一致割り込みの許可

スレーブアドレスの設定、スレーブアドレス一致割り込みの許可までが正常に終了した時、本関数は戻り値として RIIC_SUCCESS を返します。
スレーブ送信、またはスレーブ受信の処理は、その後発生する割り込み処理の中で順次行われます。
使用する割り込みは、「2.4 使用する割り込みベクタ」を参照ください。
スレーブ送信の割り込みの発生タイミングは、「5.2.4 スレーブ送信」を参照ください。スレーブ受信の割り込みの発生タイミングは、「5.2.5 スレーブ受信」を参照ください。

スレーブ送信、またはスレーブ受信終了のストップコンディションを検出した後、引数で指定したコールバック関数が呼び出されます。

スレーブ受信が正常に完了したかどうかは、引数で指定したデバイス状態フラグ、またはチャネル状態フラグ g_riic_ChStatus[ ]が”RIIC_FINISH”になっているかどうかで確認することができます。スレーブ送信が正常に完了したかどうかは、引数で指定したデバイス状態フラグ、またはチャネル状態フラグ g_riic_ChStatus[ ]が”RIIC_FINISH”もしくは”RIIC_NACK”になっているかどうかで確認することができます。マスタデバイスが最後の受信完了を NACK で通知する場合、”RIIC_NACK”になります。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
#include <stddef.h>
#include "platform.h"
#include "r_riic_rx_if.h"

riic_info_t    iic_info_m;

void CallbackMaster(void);
void CallbackSlave(void);
void main(void);

void main(void)
{
    volatile    riic_return_t ret;
    riic_info_t iic_info_s;

    uint8_t addr_eeprom[1]    = {0x50};
    uint8_t access_addr1[1]   = {0x00};
    uint8_t mst_send_data[5]  = {0x81,0x82,0x83,0x84,0x85};
    uint8_t slv_send_data[5]  = {0x71,0x72,0x73,0x74,0x75};
    uint8_t mst_store_area[5] = {0xFF,0xFF,0xFF,0xFF,0xFF};
    uint8_t slv_store_area[5] = {0xFF,0xFF,0xFF,0xFF,0xFF};

    /* Sets IIC Information for Master Send. */
    iic_info_m.dev_sts = RIIC_NO_INIT;
```

```
    iic_info_m.ch_no = 0;
    iic_info_m.callbackfunc = &CallbackMaster;
    iic_info_m.cnt2nd = 3;
    iic_info_m.cnt1st = 1;
    iic_info_m.p_data2nd = mst_store_area;
    iic_info_m.p_data1st = access_addr1;
    iic_info_m.p_slv_adr = addr_eeprom;

    /* Sets IIC Information for Slave Transfer. */
    iic_info_s.dev_sts = RIIC_NO_INIT;
    iic_info_s.ch_no = 0;
    iic_info_s.callbackfunc = &CallbackSlave;
    iic_info_s.cnt2nd = 3;
    iic_info_s.cnt1st = 3;
    iic_info_s.p_data2nd = slv_store_area;
    iic_info_s.p_data1st = slv_send_data;
    iic_info_s.p_slv_adr = (uint8_t*)FIT_NO_PTR;

    /* RIIC open */
    ret = R_RIIC_Open(&iic_info_m);

    /* RIIC slave transfer enable */
    ret = R_RIIC_SlaveTransfer(&iic_info_s);

    /* RIIC master send start */
    ret = R_RIIC_MasterSend(&iic_info_m);

    while(1);
}

void CallbackMaster(void)
{
    volatile riic_return_t ret;
    riic_mcu_status_t      iic_status;

    ret = R_RIIC_GetStatus(&iic_info_m, &iic_status);
    if(RIIC_SUCCESS != ret)
    {
        /* R_RIIC_GetStatus 関数のエラー処理 */
    }
    else
    {
        /* iic_status のステータスフラグを確認して
           タイムアウト、アービトレーションロスト、NACK
           などが検出されていた場合の処理を記述 */
    }
}

void CallbackSlave(void)
{
    /* スレーブモードでのイベント発生時に必要な処理があれば記述 */
}
```

Special Notes:
受信パターンごとの引数の設定可能範囲は、下表を参照してください。

構造体メンバ	ユーザ設定可能範囲	
	スレーブ受信	スレーブ送信
*p_slv_adr	未使用 (設定無効)	未使用 (設定無効)
*p_data1st	(スレーブ送信用)	[送信用]1st データ バッファポインタ(注 1)
*p_data2nd	[受信用]2nd データ バッファポインタ(注 2)	(スレーブ受信用)
dev_sts	デバイス状態 バッファフラグ	デバイス状態 バッファフラグ
cnt1st	(スレーブ送信用)	0000 0001h～ FFFF FFFFh
cnt2nd	0000 0001h～ FFFF FFFFh	(スレーブ受信用)
callbackfunc	使用する関数名を 指定してください。	使用する関数名を 指定してください。
ch_no	00h～FFh	00h～FFh
rsv1,rsv2,rsv3	予約領域 (設定無効)	予約領域 (設定無効)

注 1：スレーブ送信を使用する場合、設定してください。
　　システムとして、スレーブ送信を使用しない場合、“FIT_NO_PTR”を設定してください。
注 2：スレーブ受信を使用する場合、設定してください。
　　システムとして、スレーブ受信を使用しない場合、“FIT_NO_PTR”を設定してください。

3.5 R_RIIC_GetStatus()

本モジュールの状態を返します。

Format

```
riic_sts_flg_t R_RIIC_GetStatus(
        riic_info_t *           p_riic_info     /* 構造体データ */
        riic_mcu_status_t *     p_riic_status   /* RIIC のステータス */
)
```

Parameters

p_riic_info

I²C 通信情報構造体のポインタ。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t        dev_sts;        /* デバイス状態フラグ
                                               (ステータスが“RIIC_AL”時、更新あり) */
uint8_t                     ch_no;          /* チャネル番号 */
```

p_riic_status

RIIC のステータスを格納する変数のポインタ。

下記構造体で定義しているメンバで指定します。

```
typedef union
{
    uint32_t            LONG;
    struct
    {
        uint32_t        rsv:19;         /* reserve */
        uint32_t        TMO:1;          /* Time out flag */
        uint32_t        AL:1;           /* Arbitration lost detection flag */
        uint32_t        rsv:4;          /* reserve */
        uint32_t        SCLO:1;         /* SCL pin output control status */
        uint32_t        SDAO:1;         /* SDA pin output control status */
        uint32_t        SCLI:1;         /* SCL pin level */
        uint32_t        SDAI:1;         /* SDA pin level */
        uint32_t        NACK:1;         /* NACK detection flag */
        uint32_t        rsv:1;          /* reserve */
        uint32_t        BSY:1;          /* Bus status flag */
    }BIT;
} riic_mcu_status_t;
```

Return Values

```
RIIC_SUCCESS                /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN       /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG        /* 不正な引数の場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

本モジュールの状態を返します。
引数で指定した RIIC のチャネルの状態を、レジスタの読み出し、端子レベルの読み出し、変数の読み出しなどにより取得し、32 ビットの構造体で戻り値として返します。
本関数では、RIIC のアービトレーションロストフラグ、および NACK フラグを“0”にクリアします。ステータスが“RIIC_ AL”の場合、“RIIC_FINISH”に更新します。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
volatile riic_return_t  ret;
riic_info_t             iic_info_m;
riic_mcu_status_t       riic_status;

iic_info_m.ch_no = 0;

ret = R_RIIC_GetStatus(&iic_info_m, &riic_status);
```

Special Notes:

以下にステータスフラグの配置を示します。

b31 – b16
Reserve
Reserve
Rsv
Undefined

b15 – b13	b12	b11	b10 – b8
Reserve	Event detection		Reserve
Reserve	Time out detection	Arbitration lost detection	Reserve
Rsv	TMO	AL	Rsv
Undefined	0:Not detected 1:Detected		Undefined

b7	b6	b5	b4	b3	b2	b1	b0
Reserve	Pin status		Pin level		Event detection	Reserve	Bus state
Reserve	SCL Pin control	SDA Pin control	SCL Pin level	SDA Pin level	NACK detection	Reserve	Bus busy/ready
Rsv	SCLO	SDAO	SCLI	SDAI	NACK	Rsv	BSY
Undefined	0:Output Low level 1:Output Hi-z		0:Low level 1:High level		0:Not detected 1:Detected	Undefined	0:Idle 1:Busy

3.6 R_RIIC_Control()

各コンディション出力、SDA 端子のハイインピーダンス出力、SCL クロックのワンショット出力、および RIIC のモジュールリセットを行う関数です。主に通信エラー時に使用してください。

Format

```
riic_return_t R_RIIC_Control(
        r_riic_info_t *   p_riic_info       /* 構造体データ */
        uint8_t           ctrl_ptn          /* 出力パターン */
)
```

Parameters

**p_riic_info*

I²C 通信情報構造体のポインタ。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t    dev_sts;    /* デバイス状態フラグ (出力パターンに
                                       “RIIC_GEN_RESET”指定時、更新あり) */
uint8_t                 ch_no;      /* チャネル番号 */
```

ctrl_ptn

出力パターンを設定します。

- 次の出力パターンは、同時指定が可能です。同時指定する場合は、“|”(OR)を用いてください。
  - “RIIC_GEN_START_CON”と“RIIC_GEN_STOP_CON”と“RIIC_GEN_RESTART_CON”の、3 つ、または、いずれか 2 つの組み合わせで同時指定可能です。
  - “RIIC_GEN_SDA_HI_Z”と“RIIC_GEN_SCL_ONESHOT”の 2 つを同時指定可能です。

```
#define RIIC_GEN_START_CON          (uint8_t)(0x01)  /* スタートコンディションの生成*/
#define RIIC_GEN_STOP_CON           (uint8_t)(0x02)  /* ストップコンディションの生成*/
#define RIIC_GEN_RESTART_CON        (uint8_t)(0x04)  /* リスタートコンディションの生成*/
#define RIIC_GEN_SDA_HI_Z           (uint8_t)(0x08)  /* SDA 端子をハイインピーダンス出力 */
#define RIIC_GEN_SCL_ONESHOT        (uint8_t)(0x10)  /* SCL クロックのワンショット出力 */
#define RIIC_GEN_RESET              (uint8_t)(0x20)  /* RIIC のモジュールリセット */
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
RIIC_ERR_BUS_BUSY       /* バスビジーの場合 */
RIIC_ERR_AL             /* アービトレーションエラーが発生した場合 */
RIIC_ERR_OTHER          /* 現在の状態に該当しない不正なイベントが発生した場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC の制御信号を出力します。引数で指定した各コンディション出力、SDA 端子のハイインピーダンス出力、SCL クロックのワンショット出力、および RIIC のモジュールリセットを行います。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
/* Outputs an extra SCL clock cycle after the SDA pin state is changed to a
high-impedance state. */
volatile riic_return_t  ret;
riic_info_t             iic_info_m;

iic_info_m.ch_no = 0;

ret = R_RIIC_Control(&iic_info_m, RIIC_GEN_SDA_HI_Z | RIIC_GEN_SCL_ONESHOT);
```

Special Notes:

【出力パターンの SCL クロックのワンショット出力について】

マスタモード時、ノイズ等の影響でスレーブデバイスとの同期ズレが発生するとスレーブデバイスが SDA ラインを Low 固定状態にする場合があります(バスハングアップ)。この場合、SCL を 1 クロックずつ出力することでスレーブデバイスによる SDA ラインの Low 固定状態を解放させ、バス状態を復帰させることができます。

本モジュールでは、出力パターンに RIIC_GEN_SCL_ONESHOT(SCL クロックのワンショット出力)を設定して R_RIIC_Control 関数を呼び出すことにより、SCL を 1 クロック出力することができます。

3.7 R_RIIC_Close()

RIIC の通信を終了し、使用していた RIIC の対象チャネルを解放します。

Format

```
riic_return_t R_RIIC_Close(
        riic_info_t *       p_riic_info        /* 構造体データ */
)
```

Parameters

**p_riic_info*

I²C 通信情報構造体のポインタ。

この構造体のうち、本関数で使用するメンバのみを以下に示します。この構造体の詳細については2.9を参照してください。

構造体の内容は、通信中に参照、更新されます。このため、通信中(RIIC_COMMUNICATION)に構造体の内容を書き換えないでください。

下記のうち、API 実行中に値が更新される引数には、“更新あり”と記載しています。

```
riic_ch_dev_status_t        dev_sts;        /* デバイス状態フラグ (更新あり) */
uint8_t                     ch_no;          /* チャネル番号 */
```

Return Values

```
RIIC_SUCCESS            /* 問題なく処理が完了した場合 */
RIIC_ERR_INVALID_CHAN   /* 存在しないチャネルの場合 */
RIIC_ERR_INVALID_ARG    /* 不正な引数の場合 */
```

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

RIIC 通信を終了するための設定をします。引数で指定した RIIC のチャネルを無効にします。本関数では次の処理を行います。

- RIIC のモジュールストップ状態への遷移
- I²C 出力ポートの開放
- RIIC 割り込みの禁止

再度通信を開始するには、R_RIIC_Open (初期化関数)を呼び出す必要があります。通信中に強制的に停止した場合、その通信は保証しません。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
volatile riic_return_t  ret;
riic_info_t             iic_info_m;

iic_info_m.ch_no = 0;

ret = R_RIIC_Close(&iic_info_m);
```

Special Notes:

なし

3.8 R_RIIC_GetVersion()

本モジュールのバージョンを返します。

Format

uint32_t R_RIIC_GetVersion(void)

Parameters

なし

Return Values

バージョン番号

Properties

r_riic_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、現在インストールされている RIIC FIT モジュールのバージョンを返します。バージョン番号はコード化されています。最初の 2 バイトがメジャーバージョン番号、後の 2 バイトがマイナーバージョン番号です。例えば、バージョンが 4.25 の場合、戻り値は‘0x00040019’となります。

Reentrant

- 異なるチャネルからリエントラントは可能です。

Example

```
uint32_t    version;

version = R_RIIC_GetVersion();
```

Special Notes:

この関数は“#pragma inline”を使用してインライン化されています。

4. 端子設定

RIIC FIT モジュールを使用するためには、マルチファンクションピンコントローラ(MPC)で周辺機能の入出力信号を端子に割り付ける（以下、端子設定と称す）必要があります。
RIIC FIT モジュールは、コンフィグレーションオプションの RIIC_CFG_PORT_SET_PROCESSING の設定により、R_RIIC_Open 関数の中で端子設定するかを選択できます。
コンフィグレーションオプションの詳細は、「2.7 コンパイル時の設定」を参照ください。

e² studio の場合は「FIT Configurator」または「Smart Configurator」の端子設定機能を使用することができます。FIT Configurator、Smart Configurator の端子機能を使用すると、端子設定画面で選択した端子を使用することができます。選択した端子情報は r_riic_pin_config.h に反映され、表 4.1に示すマクロ定義の値が選択した端子に応じた値に上書きされます。RIIC FIT モジュールでは、FIT Configurator を使用する場合、端子設定機能を有効にする関数が記述されたソースファイル（および“r_pincfg”フォルダ）は生成されません。

表 4.1 端子設定マクロ定義

選択したチャネル	選択した端子	マクロ定義
チャネル 0	SCL0 端子	R_RIIC_CFG_RIIC0_SCL0_PORT R_RIIC_CFG_RIIC0_SCL0_BIT
	SDA0 端子	R_RIIC_CFG_RIIC0_SDA0_PORT R_RIIC_CFG_RIIC0_SDA0_BIT
チャネル 1	SCL1 端子	R_RIIC_CFG_RIIC1_SCL1_PORT R_RIIC_CFG_RIIC1_SCL1_BIT
	SDA1 端子	R_RIIC_CFG_RIIC1_SDA1_PORT R_RIIC_CFG_RIIC1_SDA1_BIT
チャネル 2	SCL2 端子	R_RIIC_CFG_RIIC2_SCL2_PORT R_RIIC_CFG_RIIC2_SCL2_BIT
	SDA2 端子	R_RIIC_CFG_RIIC2_SDA2_PORT R_RIIC_CFG_RIIC2_SDA2_BIT

r_riic_pin_config.h で選択した端子は、R_RIIC_Open 関数呼び出し後、周辺機能端子として SCL 端子、SDA 端子となります。
周辺機能端子の割り付けは R_RIIC_Close 関数が呼び出されると解除され、端子は汎用入出力端子（入力状態）になります。
なお、SCL 端子、SDA 端子は外付け抵抗でプルアップ処理を行ってください。

もし RIIC_CFG_PORT_SET_PROCESSING の設定で本モジュール内の端子設定処理を使用しない場合は、R_RIIC_Open 関数を呼び出した後、その他の API を呼び出す前にユーザ処理で使用する端子の端子設定を行ってください。

5. 付録

5.1 通信方法の実現

本モジュールでは、スタートコンディション生成やスレーブアドレス送信などの処理を 1 つのプロトコルとして管理しており、このプロトコルを組み合わせることで通信を実現します。

5.1.1 制御時の状態

表 5.1に、プロトコル制御を実現するための状態を定義します。

表 5.1　プロトコル制御のための状態一覧(enum r_riic_api_status_t)

No	状態名	状態の定義
STS0	RIIC_STS_NO_INIT	未初期化状態
STS1	RIIC_STS_IDLE	アイドル状態(マスタ通信可能状態)
STS2	RIIC_STS_IDLE_EN_SLV	アイドル状態(マスタ／スレーブ通信可能状態)
STS3	RIIC_STS_ST_COND_WAIT	スタートコンディション検出待ち状態
STS4	RIIC_STS_SEND_SLVADR_W_WAIT	スレーブアドレス[Write]送信完了待ち状態
STS5	RIIC_STS_SEND_SLVADR_R_WAIT	スレーブアドレス[Read]送信完了待ち状態
STS6	RIIC_STS_SEND_DATA_WAIT	データ送信完了待ち状態
STS7	RIIC_STS_RECEIVE_DATA_WAIT	データ受信完了待ち状態
STS8	RIIC_STS_SP_COND_WAIT	ストップコンディション検出待ち状態
STS9	RIIC_STS_AL	アービトレーションロスト状態
STS10	RIIC_STS_TMO	タイムアウト検出状態

5.1.2 制御時のイベント

表 5.2にプロトコル制御時に発生するイベントを定義します。割り込みだけでなく、本モジュールが提供する API 関数が呼び出された際も、イベントとして定義します。

表 5.2　プロトコル制御のためのイベント一覧(enum r_riic_api_event_t)

No	イベント名	イベントの定義
EV0	RIIC_EV_INIT	R_RIIC_Open()呼び出し
EV1	RIIC_EV_EN_SLV_TRANSFER	R_RIIC_SlaveTransfer()呼び出し
EV2	RIIC_EV_GEN_START_COND	R_RIIC_MasterSend() または R_RIIC_MasterReceive()呼び出し
EV3	RIIC_EV_INT_START	EEI 割り込み発生（割り込みフラグ：START）
EV4	RIIC_EV_INT_ADD	TEI 割り込み発生、TXI 割り込み発生
EV5	RIIC_EV_INT_SEND	TEI 割り込み発生、TXI 割り込み発生
EV6	RIIC_EV_INT_RECEIVE	RXI 割り込み発生
EV7	RIIC_EV_INT_STOP	EEI 割り込み発生（割り込みフラグ：STOP）
EV8	RIIC_EV_INT_AL	EEI 割り込み発生（割り込みフラグ：AL）
EV9	RIIC_EV_INT_NACK	EEI 割り込み発生（割り込みフラグ：NACK）
EV10	RIIC_EV_INT_TMO	EEI 割り込み発生(割り込みフラグ：TMO)

5.1.3 プロトコル状態遷移

本モジュールでは、提供する API 関数の呼び出し、または、I²C 割り込み発生をトリガに状態が遷移します。図 5.1～図 5.4に各プロトコルの状態遷移を示します。

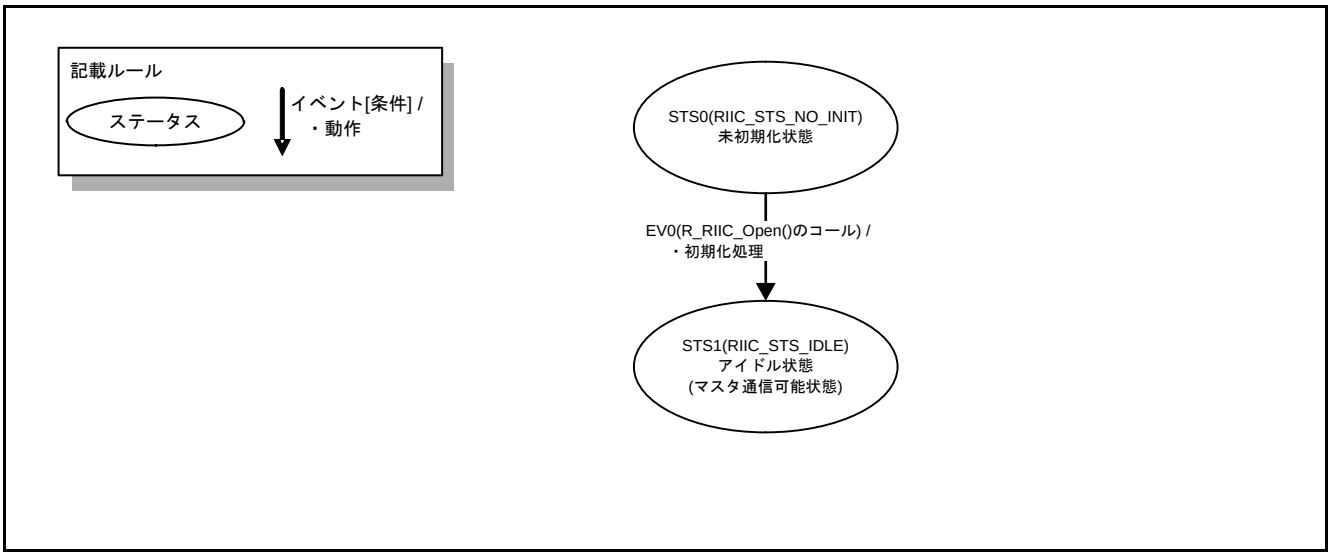


図 5.1　初期化処理（R_RIIC_Open()呼び出し）時の状態遷移図

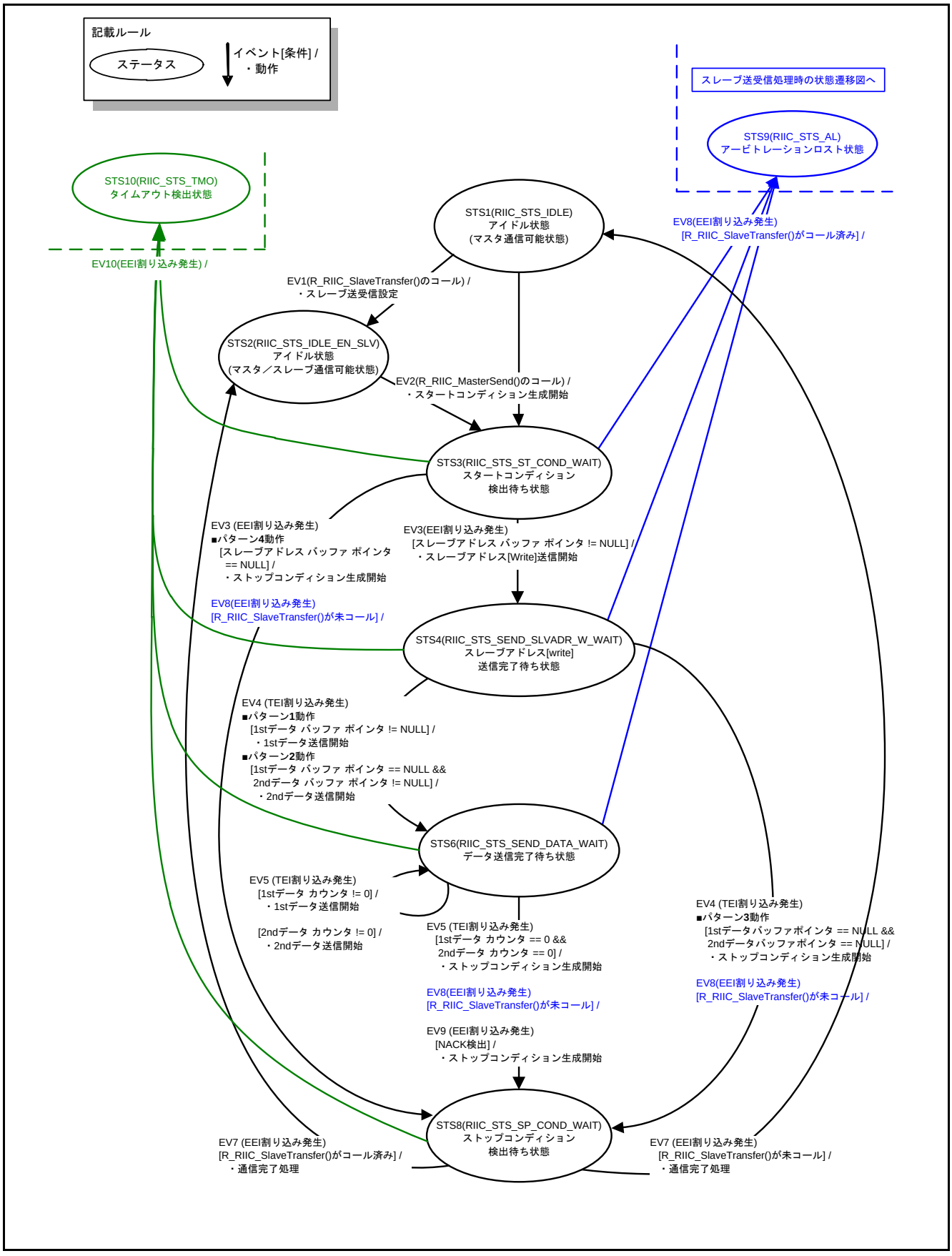


図 5.2　マスタ送信処理（R_RIIC_MasterSend()呼び出し）時の状態遷移図

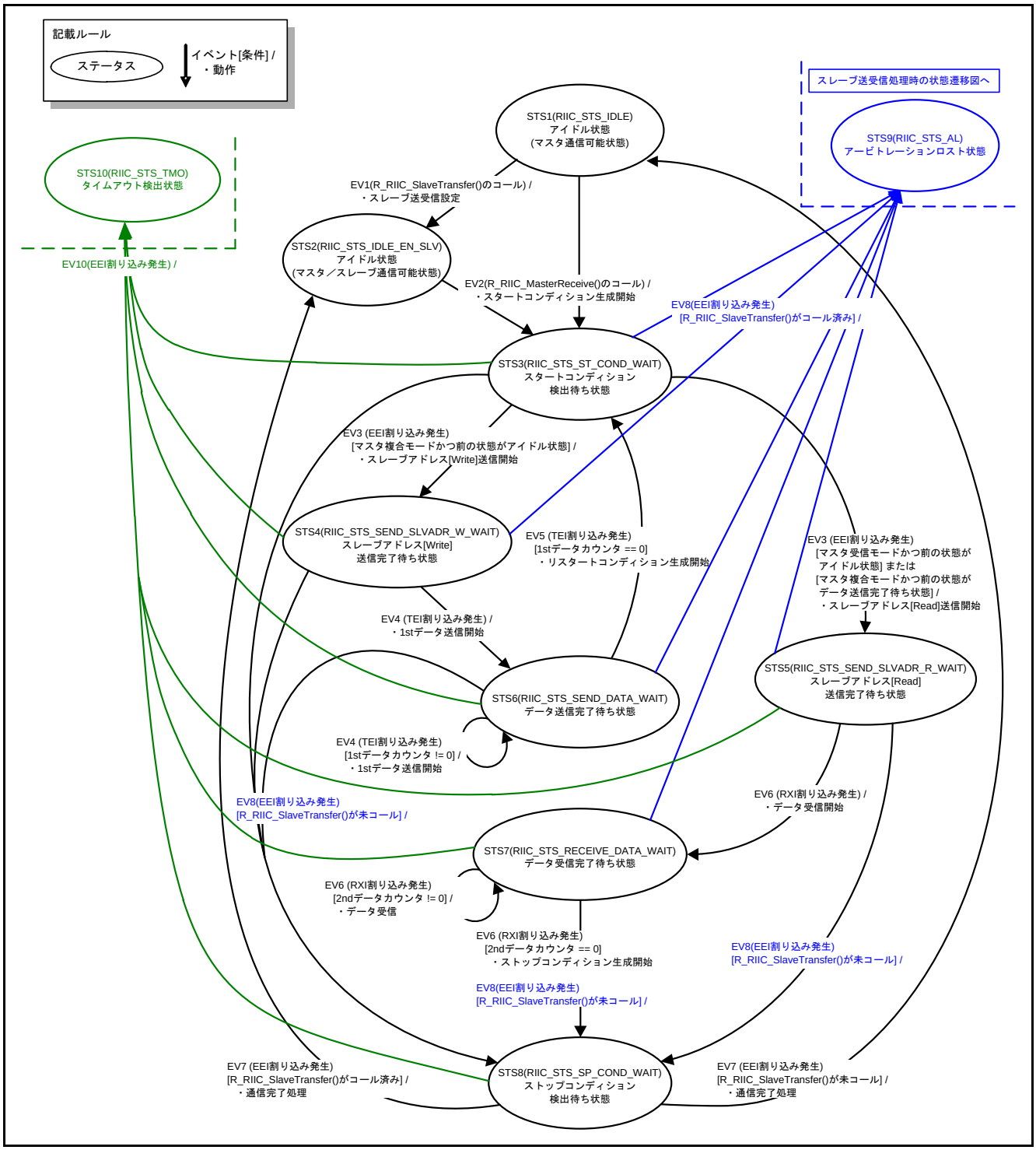


図 5.3　マスタ受信処理（R_RIIC_MasterReceive()呼び出し）時の状態遷移図

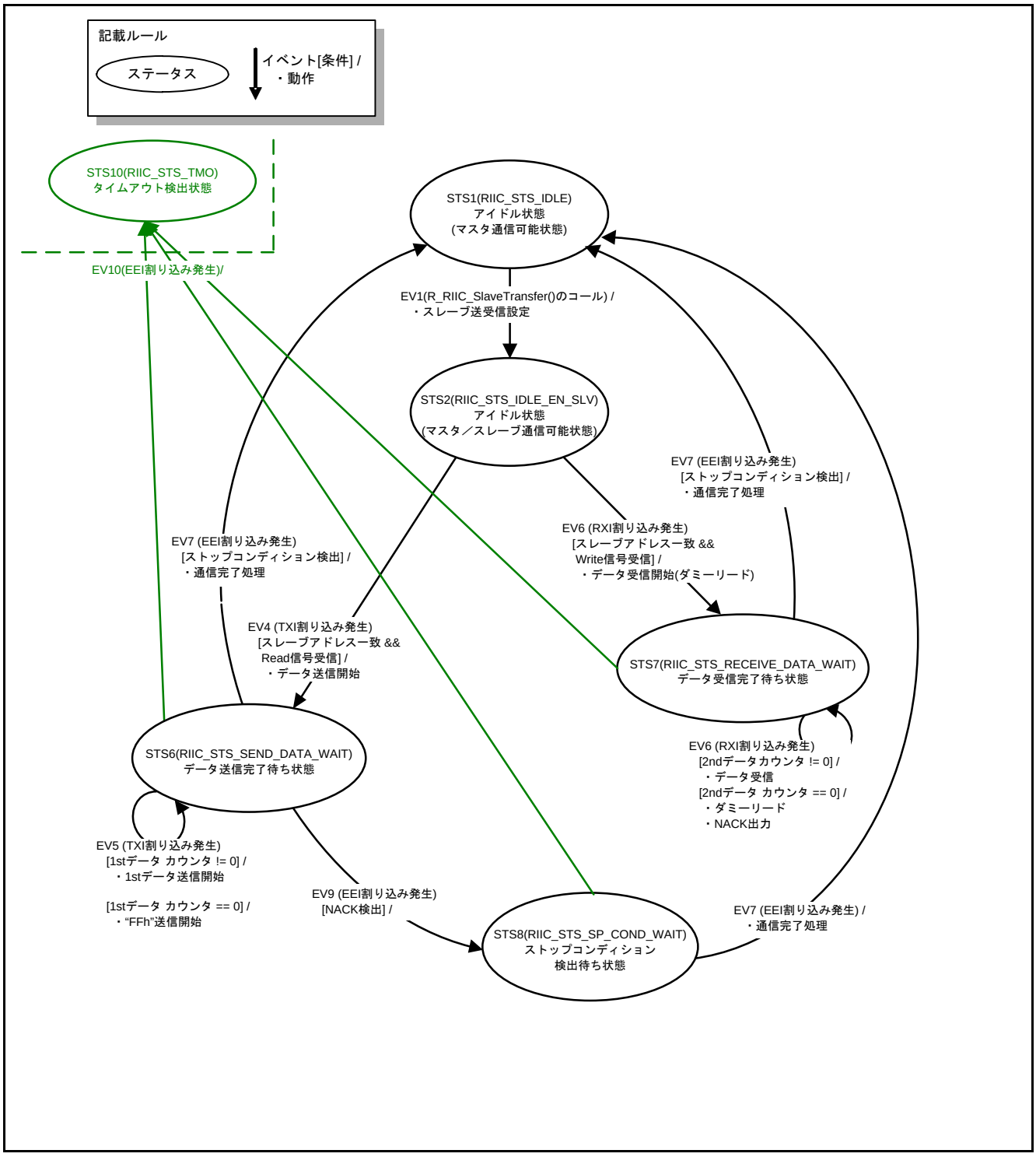


図 5.4　スレーブ送受信処理（R_RIIC_SlaveTransfer()呼び出し）時の状態遷移図

5.1.4 プロトコル状態遷移表

表 5.1の各状態で、表 5.2のイベントが発生した際に動作する処理を、表 5.3の状態遷移表に定義します。

Func0～Func11 については、表 5.4を参照してください。

表 5.3 プロトコル状態遷移表(gc_riic_mtx_tbl [ ] [ ])

イベント / 状態		EV0	EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8	EV9	EV10
STS0	未初期化状態 【RIIC_STS_NO_INIT】	**Func0**	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS1	アイドル状態 (マスタ通信可能状態) 【RIIC_STS_IDLE】	ERR	**Func10**	**Func1**	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS2	アイドル状態 (マスタ／スレーブ通信可能状態) 【RIIC_STS_IDLE_EN_SLV】	ERR	ERR	**Func1**	ERR	**Func4**	ERR	**Func4**	ERR	ERR	ERR	ERR
STS3	スタートコンディション生成完了待ち状態 【RIIC_STS_ST_COND_WAIT】	ERR	ERR	ERR	**Func2**	ERR	ERR	ERR	ERR	**Func8**	**Func9**	**Func11**
STS4	スレーブアドレス[Write]送信完了待ち状態 【RIIC_STS_SEND_SLVADR_W_WAIT】	ERR	ERR	ERR	ERR	**Func3**	ERR	ERR	ERR	**Func8**	**Func9**	**Func11**
STS5	スレーブアドレス[Read]送信完了待ち状態 【RIIC_STS_SEND_SLVADR_R_WAIT】	ERR	ERR	ERR	ERR	ERR	ERR	**Func3**	ERR	**Func8**	**Func9**	**Func11**
STS6	データ送信完了待ち状態 【RIIC_STS_SEND_DATA_WAIT】	ERR	ERR	ERR	ERR	ERR	**Func5**	ERR	ERR	**Func8**	**Func9**	**Func11**
STS7	データ受信完了待ち状態 【RIIC_STS_RECEIVE_DATA_WAIT】	ERR	ERR	ERR	ERR	ERR	ERR	**Func6**	ERR	ERR	**Func9**	**Func11**
STS8	ストップコンディション生成完了待ち状態 【RIIC_STS_SP_COND_WAIT】	ERR	ERR	ERR	ERR	ERR	ERR	ERR	**Func7**	ERR	**Func9**	**Func11**
STS9	アービトレーションロスト状態 【RIIC_STS_AL】	ERR	ERR	ERR	ERR	ERR	**Func5**	**Func6**	**Func7**	ERR	ERR	ERR
STS10	タイムアウト検出状態 【RIIC_STS_TMO】	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR

備考：ERR は RIIC_ERR_OTHER を表します。ある状態で意図しないイベントが通知された場合には、すべてエラー処理を行います。

5.1.5 プロトコル状態遷移登録関数

表 5.4に状態遷移表に登録されている関数を定義します。

表 5.4 プロトコル状態遷移登録関数一覧

処理	関数名	概要
Func0	riic_init_driver()	初期設定処理
Func1	riic_generate_start_cond()	スタートコンディション生成処理(マスタ送信用)
Func2	riic_after_gen_start_cond()	スタートコンディション生成後処理
Func3	riic_after_send_slvadr()	スレーブアドレスが送信完了した後の処理
Func4	riic_after_receive_slvadr()	受信したスレーブアドレスが一致した後の処理
Func5	riic_write_data_sending()	データ送信処理
Func6	riic_read_data_receiving()	データ受信処理
Func7	riic_after_dtct_stop_cond ()	通信完了処理
Func8	riic_arbitration_lost()	アービトレーションロスト検出した時の処理
Func9	riic_nack()	NACK 検出した時の処理
Func10	riic_enable_slave_transfer()	スレーブ送受信有効
Func11	riic_time_out()	タイムアウト検出時の処理

5.1.6 状態遷移時の各フラグの状態

<各チャネル状態管理>

チャネル状態フラグ g_riic_ChStatus[ ]により、1 つのバス上に接続された複数スレーブデバイスの排他制御を行います。

本フラグは、各チャネルに対して 1 つ存在し、グローバル変数で管理します。本モジュールの初期化処理を完了し、対象バスで通信が行われていない場合、本フラグは“RIIC_IDLE/RIIC_FINISH/RIIC_NACK”(アイドル状態(通信可能))となり、通信が可能です。通信中の本フラグの状態は、“RIIC_COMMUNICATION”(通信中)になります。通信開始時、必ず本フラグの確認を行うため、通信中に同一チャネル上の他デバイスの通信を開始しません。本フラグをチャネルごとに管理することで、複数チャネルの同時通信を実現します。

<各デバイス状態管理>

I²C 通信情報構造体メンバのデバイス状態フラグ dev_sts により、同一チャネル上の複数のスレーブデバイスの制御を行うことができます。デバイス状態フラグには、そのデバイスの通信状態が格納されます。

表 5.5に状態遷移時の各フラグの状態を示します。

表 5.5　状態遷移時の各フラグの状態一覧

状態	チャネル状態フラグ	デバイス状態フラグ（通信のデバイス）	I²C プロトコルの動作モード	プロトコル制御の現状態
	g_riic_ChStatus[]	I²C 通信情報構造体 dev_sts	内部通信情報構造体 N_Mode	内部通信情報構造体 N_status
未初期化状態	RIIC_NO_INIT	RIIC_NO_INIT	RIIC_MODE_NONE	RIIC_STS_NO_INIT
アイドル状態 (マスタ通信可能状態)	RIIC_IDLE RIIC_FINISH RIIC_NACK	RIIC_IDLE RIIC_FINISH RIIC_NACK	RIIC_MODE_NONE	RIIC_STS_IDLE
アイドル状態 (マスタ/スレーブ通信可能状態)	RIIC_IDLE	RIIC_IDLE	RIIC_MODE_S_READY	RIIC_STS_IDLE_EN_SLV
通信中（マスタ送信）	RIIC_COMMUNICATION	RIIC_COMMUNICATION	RIIC_MODE_M_SEND	RIIC_STS_ST_COND_WAIT RIIC_STS_SEND_SLVADR_W_WAIT RIIC_STS_SEND_DATA_WAIT RIIC_STS_SP_COND_WAIT RIIC_STS_AL RIIC_STS_TMO
通信中（マスタ受信）	RIIC_COMMUNICATION	RIIC_COMMUNICATION	RIIC_MODE_M_RECEIVE	RIIC_STS_ST_COND_WAIT RIIC_STS_SEND_SLVADR_R_WAIT RIIC_STS_RECEIVE_DATA_WAIT RIIC_STS_SP_COND_WAIT RIIC_STS_AL RIIC_STS_TMO
通信中（マスタ送受信）	RIIC_COMMUNICATION	RIIC_COMMUNICATION	RIIC_MODE_M_SEND_RECEIVE	RIIC_STS_ST_COND_WAIT RIIC_STS_SEND_SLVADR_W_WAIT RIIC_STS_SEND_SLVADR_R_WAIT RIIC_STS_SEND_DATA_WAIT RIIC_STS_RECEIVE_DATA_WAIT RIIC_STS_SP_COND_WAIT RIIC_STS_AL RIIC_STS_TMO
通信中(スレーブ送信)	RIIC_COMMUNICATION	RIIC_COMMUNICATION	RIIC_MODE_S_SEND	RIIC_STS_SEND_DATA_WAIT RIIC_STS_SP_COND_WAIT RIIC_STS_TMO
通信中(スレーブ受信)	RIIC_COMMUNICATION	RIIC_COMMUNICATION	RIIC_MODE_S_RECEIVE	RIIC_STS_RECEIVE_DATA_WAIT RIIC_STS_SP_COND_WAIT RIIC_STS_TMO
アービトレーションロスト検出状態	RIIC _AL	RIIC _AL		
タイムアウト検出状態	RIIC_TMO	RIIC_TMO		
エラー状態	RIIC_ERROR	RIIC_ERROR	-	-

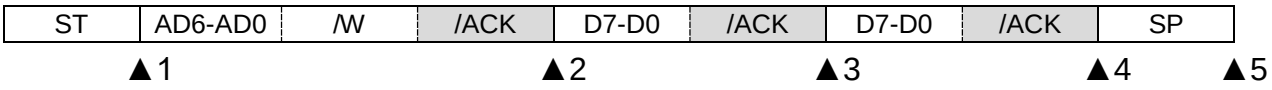
5.2 割り込み発生タイミング

以下に本モジュールの割り込みタイミングを示します。

備考 ST : スタートコンディション
AD6-AD0 : スレーブアドレス
/W : 転送方向ビット“0”(Write)
R : 転送方向ビット“1”(Read)
/ACK : Acknowledge“0”
NACK : Acknowledge“1”
D7-D0 : データ
RST : リスタートコンディション
SP : ストップコンディション

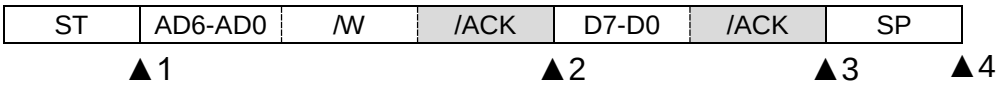
5.2.1 マスタ送信

1. パターン 1



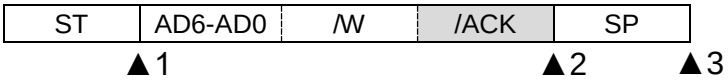
▲1：EEI（START）割り込み・・・スタートコンディション検出
▲2：TEI 割り込み・・・アドレス送信完了（転送方向ビット：Write）
▲3：TEI 割り込み・・・データ送信完了（1st データ）
▲4：TEI 割り込み・・・データ送信完了（2nd データ）
▲5：EEI（STOP）割り込み・・・ストップコンディション検出

2. パターン 2



▲1：EEI（START）割り込み・・・スタートコンディション検出
▲2：TEI 割り込み・・・アドレス送信完了（転送方向ビット：Write）
▲3：TEI 割り込み・・・データ送信完了（2nd データ）
▲4：EEI（STOP）割り込み・・・ストップコンディション検出

3. パターン 3

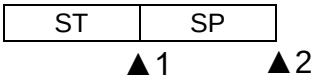


▲1：EEI（START）割り込み・・・スタートコンディション検出

▲2：TEI 割り込み・・・アドレス送信完了（転送方向ビット：Write）

▲3：EEI（STOP）割り込み・・・ストップコンディション検出

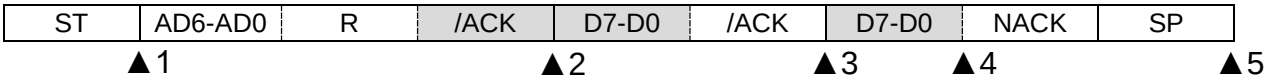
4. パターン 4



▲1：EEI（START）割り込み・・・スタートコンディション検出

▲2：EEI（STOP）割り込み・・・ストップコンディション検出

5.2.2 マスタ受信



▲1：EEI（START）割り込み・・・スタートコンディション検出

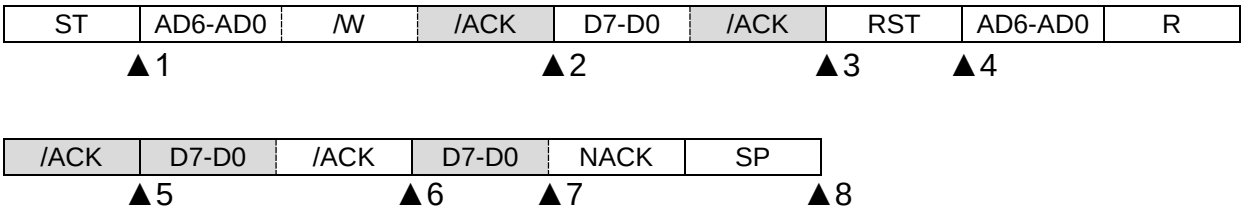
▲2：RXI 割り込み・・・アドレス送信完了（転送方向ビット：Read）

▲3：RXI 割り込み・・・最終データ－1 受信完了（2nd データ）

▲4：RXI 割り込み・・・最終データ受信完了（2nd データ）

▲5：EEI（STOP）割り込み・・・ストップコンディション検出

5.2.3 マスタ送受信



▲1：EEI（START）割り込み・・・スタートコンディション検出

▲2：TEI 割り込み・・・アドレス送信完了（転送方向ビット：Write）

▲3：TEI 割り込み・・・データ送信完了（1st データ）

▲4：EEI（START）割り込み・・・リスタートコンディション検出

▲5：RXI 割り込み・・・アドレス送信完了（転送方向ビット：Read）

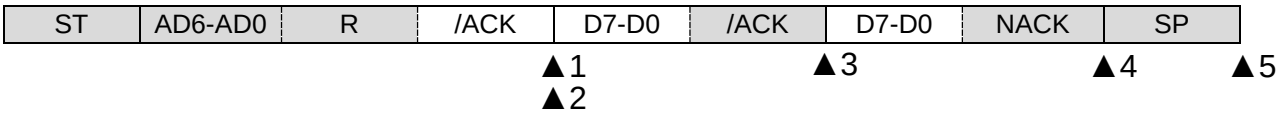
▲6：RXI 割り込み・・・最終データ－1 受信完了（2nd データ）

▲7：RXI 割り込み・・・最終データ受信完了（2nd データ）

▲8：EEI（STOP）割り込み・・・ストップコンディション検出

5.2.4 スレーブ送信

2 バイト送信時



▲1：TXI 割り込み・・・アドレス受信一致（転送方向ビット：Read）

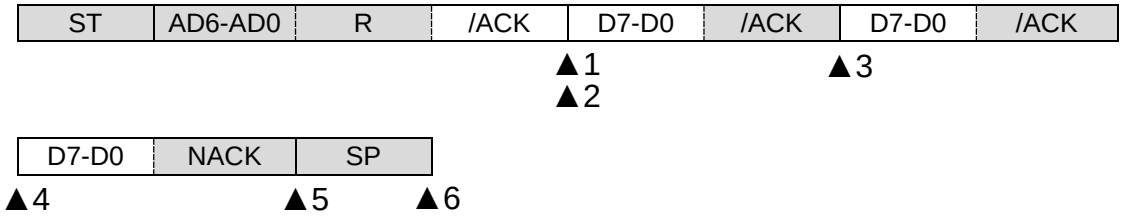
▲2：TXI 割り込み・・・送信バッファ空

▲3：TXI 割り込み・・・送信バッファ空

▲4：EEI（NACK）割り込み・・・NACK 検出

▲5：EEI（STOP）割り込み・・・ストップコンディション検出

3 バイト送信時



▲1：TXI 割り込み・・・アドレス受信一致（転送方向ビット：Read）

▲2：TXI 割り込み・・・送信バッファ空

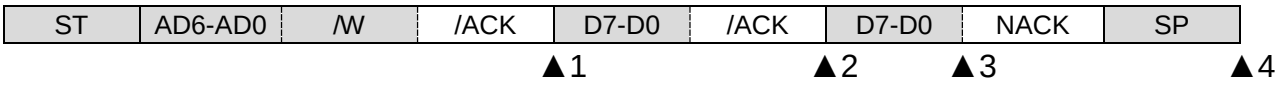
▲3：TXI 割り込み・・・送信バッファ空

▲4：TXI 割り込み・・・送信バッファ空

▲5：EEI（NACK）割り込み・・・NACK 検出

▲6：EEI（STOP）割り込み・・・ストップコンディション検出

5.2.5　スレーブ受信



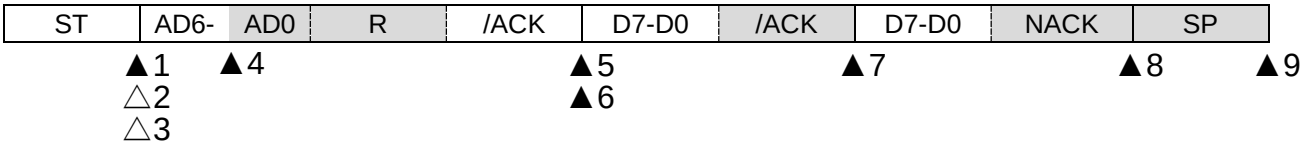
▲1：RXI 割り込み・・・アドレス受信一致（転送方向ビット：Write）

▲2：RXI 割り込み・・・最終データ－1 受信完了（2nd データ）

▲3：RXI 割り込み・・・最終データ受信完了（2nd データ）

▲4：EEI（STOP）割り込み・・・ストップコンディション検出

5.2.6　マルチマスタ通信（マスタ送信中の AL 検出後、スレーブ送信）



▲1：EEI（START）割り込み・・・スタートコンディション検出

△2：TXI 割り込み・・・スタートコンディション検出　※処理なし

△3：TXI 割り込み・・・送信バッファ空　※処理なし

▲4：EEI（AL）割り込み・・・アービトレーションロスト検出

▲5：TXI 割り込み・・・アドレス受信一致（転送方向ビット：Read）

▲6：TXI 割り込み・・・送信バッファ空

▲7：TXI 割り込み・・・送信バッファ空

▲8：EEI（NACK）割り込み・・・NACK 検出

▲9：EEI（STOP）割り込み・・・ストップコンディション検出

5.3　タイムアウトの検出、および検出後の処理

5.3.1　タイムアウト検出機能によるタイムアウト検出

「r_riic_config.h」の設定でタイムアウト検出機能を有効にした場合、コールバック関数内で R_RIIC_GetStatus 関数を呼び出してください。

タイムアウト検出情報は R_RIIC_GetStatus 関数の第 2 引数に設定した riic_mcu_status_t 構造体変数の TMO ビットにより確認できます。

TMO ビットが“1”：タイムアウトを検出

TMO ビットが“0”：タイムアウト未検出

5.3.2　タイムアウト検出後の対応方法

タイムアウトが検出された場合は、いったん R_RIIC_Close 関数を呼び出し、初期化処理の R_RIIC_Open 関数から通信を再開する必要があります。

また、バスハングアップによりタイムアウトが検出される場合もあります。マスタモード時、ノイズ等の影響でスレーブデバイスとの同期ズレが発生するとスレーブデバイスが SDA ラインを Low 固定状態にする場合があります(バスハングアップ)。この状態ではストップコンディションは発行できないため、タイムアウトが検出されます。

バスハングアップから復帰するためには、SCL クロック追加出力機能を使用します。追加クロックを 1 クロックずつ出力することでスレーブデバイスによる SDA ラインの Low 固定状態を解放させ、バス状態を復帰させることができます。

追加クロックを 1 クロック出力するためには、R_RIIC_Control 関数の第 2 引数に RIIC_GEN_SCL_ONESHOT (SCL クロックのワンショット出力)を設定して R_RIIC_Control()を呼び出してください。

また、SCL の端子状態は R_RIIC_GetStatus 関数で確認できます。

SCL が High になるまで SCL クロックのワンショット出力を繰り返してください。

図 5.5にタイムアウト検出と対応方法例(マスタ送信)を示します。

SCL クロック追加出力機能についての詳細は、各マイコンのユーザーズマニュアル　ハードウェア編の RIIC 章に記載されている「SCL クロック追加出力機能」を参照ください。

例）RX111 グループユーザーズマニュアル　ハードウェア編の場合は「27.11.2 SCL クロック追加出力機能」

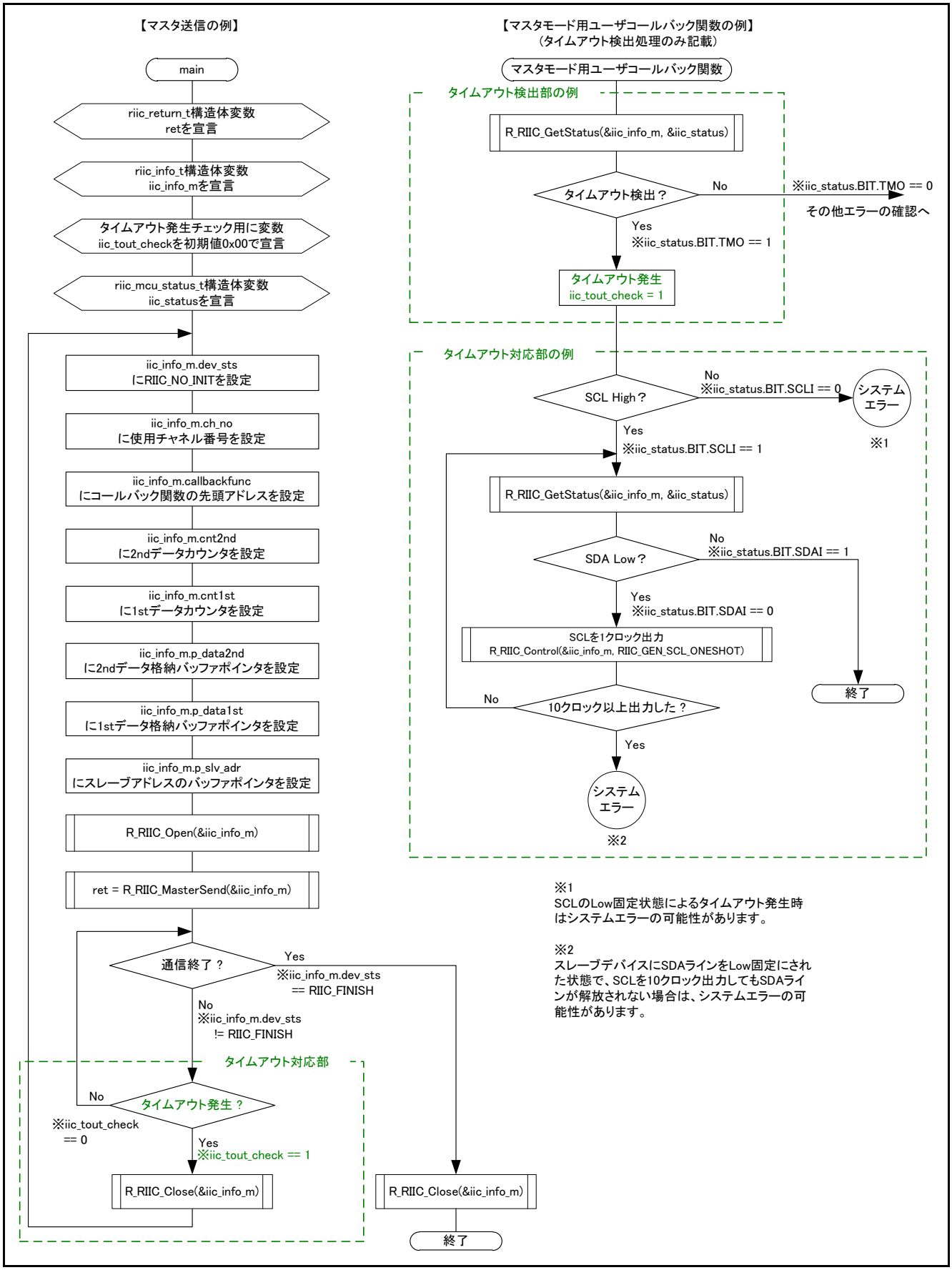


図 5.5　タイムアウト検出と対応方法例(マスタ送信)

5.4 動作確認環境

本モジュールの動作確認環境を以下に示します。

表 5.6 動作確認環境 (Rev.1.60、Rev.1.70)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V3.1.0.024
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.01.01
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.1.60、Rev.1.70
使用ボード	Renesas Starter Kit for RX111（型名：R0K505111SxxxBE） Renesas Starter Kit for RX231（型名：R0K505231SxxxBE） Renesas Starter Kit+ for RX64M（型名：R0K50564MSxxxBE） Renesas Starter Kit+ for RX71M（型名：R0K50571MSxxxBE）

表 5.7 動作確認環境 (Rev.1.80)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V4.0.2.008
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.03.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.1.80
使用ボード	Renesas Starter Kit for RX130（型名：RTK5005130SxxxxxBE） Renesas Starter Kit for RX23T（型名：RTK500523TSxxxxxBE）

表 5.8 動作確認環境 (Rev.1.90)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V4.1.0.018
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.03.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.1.90
使用ボード	Renesas Starter Kit for RX111（型名：R0K505111SxxxBE） Renesas Starter Kit for RX113（型名：R0K505113SxxxBE） Renesas Starter Kit for RX130（型名：RTK5005130SxxxxxBE） Renesas Starter Kit for RX231（型名：R0K505231SxxxBE） Renesas Starter Kit for RX23T（型名：RTK500523TSxxxxxBE） Renesas Starter Kit for RX24T（型名：RTK500524TSxxxxxBE） Renesas Starter Kit+ for RX64M（型名：R0K50564MSxxxBE） Renesas Starter Kit+ for RX71M（型名：R0K50571MSxxxBE）

表 5.9　動作確認環境 (Rev.2.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V5.0.1.005
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.05.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.2.00
使用ボード	Renesas Starter Kit for RX231（型名：R0K505231SxxxBE） Renesas Starter Kit+ for RX65N（型名：RTK500565NSxxxxxBE）

表 5.10　動作確認環境 (Rev.2.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V5.3.0.023
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.06.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.2.10
使用ボード	Renesas Starter Kit for RX24T（型名：RTK500524TSxxxxxBE） Renesas Starter Kit for RX24U（型名：RTK500524USxxxxxBE）

表 5.11　動作確認環境 (Rev.2.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V6.0.0.001
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.06.00 ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V2.07.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.2.20
使用ボード	Renesas Starter Kit for RX130-512KB（型名：RTK5051308SxxxxxBE） Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565N2SxxxxxBE）

5.5 トラブルシューティング

(1) Q : 本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A : FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e² studio を使用している場合
  アプリケーションノート RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r_riic_rx module.」エラーが発生します。

A：追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

(3) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「ERROR - RIIC_CFG_xxx_xxx - ...」エラーが発生します。

A：“r_riic_rx_config.h”ファイルの設定値が間違っている可能性があります。“r_riic_rx_config.h”ファイルを確認して正しい値を設定してください。詳細は「2.7 コンパイル時の設定」を参照してください。

5.6 サンプルコード

5.6.1 1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする場合の例

RIIC の 1 つのチャネルを使用し、1 つのスレーブデバイスに対して、連続アクセスする場合のサンプルコードを示します。

次の(1)～(6)の順に動作します。

(1) RIIC の ch0 を使用可能にするため、R_RIIC_Open 関数を実行する。
(2) EEPROM に 16 バイトのデータを書き込むため、R_RIIC_MasterSend 関数を実行する。
(3) EEPROM 書き込み完了を確認するために、R_RIIC_MasterSend 関数を使用し、Acknowledge Polling を行う。
(4) EEPROM から 16 バイトのデータを読み出すため、R_RIIC_MasterReceive 関数を実行する。
(5) 書き込みデータと読み出しデータを比較する。
(6) RIIC の ch0 を RIIC FIT モジュールから解放するため、R_RIIC_Close 関数を実行する。

このサンプルコードは 対象デバイスの Renesas Starter Kit で動作確認をしています。スレーブデバイスのアドレスは使用する EEPROM によって異なりますのでご注意ください。

```
#include <stddef.h>
#include "platform.h"
#include "r_riic_rx_if.h"

/* EEPROM device code (fixed) */
#define    EEPROM_DEVICE_CODE       (0xA0)

/* Device address code(under 4 bit is A2(Vss=0), A1(Vcc=1), A0(Vcc=1), and RW code)
   for hardware connection with EEPROM on RSK of the supported target device.
   Please change the following settings as necessary. */
#define    EEPROM_DEVICE_ADDRESS_CODE    (0x06)

/* E2PROM device address */
#define    EEPROM_DEVICE_ADDRESS    ((EEPROM_DEVICE_CODE | EEPROM_DEVICE_ADDRESS_CODE) >> 1)

/* variables */
static volatile riic_return_t ret;   /* Return value */
static riic_info_t iic_info_m;       /* Structure data */

static uint8_t addr_eeprom[1] = { EEPROM_DEVICE_ADDRESS };
static uint8_t access_addr1[1] = { 0x00 };

/* This data is sent to the EEPROM when target device is the master device. */
static uint8_t master_send_data[16] =
{ 0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f };

/* This buffer stores data received from the slave device. */
static uint8_t master_store_area[16] =
{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

/* private functions */
static void callback_master (void);
static void eeprom_write (void);
static void acknowledge_polling (void);
static void eeprom_read (void);
```

図 5.6　1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする例(1)

```
/***********************************************************************************************
* Function Name: main
* Description  : The main loop
* Arguments    : none
* Return Value : none
***********************************************************************************************/
void main (void)
{
    uint8_t i = 0;

    /* Initialize */
    for (i = 0; i < 16; i++)
    {
        master_store_area[i] = 0xFF;
    }

    /* Set arguments for R_RIIC_Open. */
    iic_info_m.ch_no = 0;               /* Channel number */
    iic_info_m.dev_sts = RIIC_NO_INIT; /* Device state flag (to be updated) */

    ret = R_RIIC_Open(&iic_info_m);
    if (RIIC_SUCCESS != ret)
    {
        /* This software is for single master.
           Therefore, return value should be always 'RIIC_SUCCESS'. */
        while (1)
        {
            nop();    /* error */
        }
    }

    /* EEPROM Write (Master transfer) */
    eeprom_write();

    /* Acknowledge polling (Master transfer) */
    acknowledge_polling();

    /* EEPROM Read (Master transfer and Master receive) */
    eeprom_read();

    /* Compare */
    for (i = 0; i < 16; i++)
    {
        if (master_store_area[i] != master_send_data[i])
        {
            /* Detected mismatch. */
            LED3 = LED_ON;
        }
        else
        {
            LED0 = LED_ON;
        }
    }

    ret = R_RIIC_Close(&iic_info_m);
    if (RIIC_SUCCESS != ret)
    {
        /* This software is for single master.
           Therefore, return value should be always 'RIIC_SUCCESS'. */
        while (1)
        {
            nop();    /* error */
        }
    }

    while (1)
    {
        /* do nothing */
    }

} /* End of function main() */
```

図 5.7　1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする例(2)

```
/***********************************************************************************************
* Function Name: callback_master
* Description  : This function is sample of Master Mode callback function.
* Arguments    : none
* Return Value : none
***********************************************************************************************/
static void callback_master (void)
{
    riic_mcu_status_t      iic_status;

    ret = R_RIIC_GetStatus(&iic_info_m, &iic_status);
    if (RIIC_SUCCESS != ret)
    {
        /* This software is for single master.
           Therefore, return value should be always 'RIIC_SUCCESS'. */
        while (1)
        {
            nop();    /* error */
        }
    }
    else
    {
        /* Processing when a timeout, arbitration-lost, NACK,
           or others is detected by verifying the iic_status flag. */
    }

} /* End of function callback_master() */


/***********************************************************************************************
* Function Name: eeprom_write
* Description  : This function is sample of EEPROM write function using R_RIIC_MasterSend.
* Arguments    : none
* Return Value : none
***********************************************************************************************/
static void eeprom_write (void)
{
    /* Set arguments for R_RIIC_MasterSend. */
    iic_info_m.p_slv_adr = addr_eeprom;  /* Pointer to the slave address storage buffer */
    iic_info_m.p_data1st = access_addr1; /* Pointer to the first data storage buffer */
    iic_info_m.cnt1st = 1;               /* First data counter (number of bytes)(to be updated) */
    iic_info_m.p_data2nd = master_send_data;    /* Pointer to the second data storage buffer */
    iic_info_m.cnt2nd = 16;              /* Second data counter (number of bytes)(to be updated) */
    iic_info_m.callbackfunc = &callback_master; /* Callback function */

    /* Master send start. */
    ret = R_RIIC_MasterSend(&iic_info_m);
    if (RIIC_SUCCESS == ret)
    {
        /* Waitting for R_RIIC_MasterSend completed. */
        while (RIIC_COMMUNICATION == iic_info_m.dev_sts)
        {
            /* do nothing */
        }

        if (RIIC_NACK == iic_info_m.dev_sts)
        {
            /* Slave returns NACK. The slave address may not correct.
               Please check the macro definition value or hardware connection etc. */
            while (1)
            {
                nop();    /* error */
            }
        }
    }
    else
    {
        /* This software is for single master.
           Therefore, return value should be always 'RIIC_SUCCESS'. */
        while (1)
        {
                nop();    /* error */
```

図 5.8　1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする例(3)

```
        }
    }

} /* End of function eeprom_write() */


/***********************************************************************************************
* Function Name: acknowledge_polling
* Description  : This function is sample of Acknowledge Polling using R_RIIC_MasterSend with
*                master send pattern 3.
* Arguments    : none
* Return Value : none
***********************************************************************************************/
static void acknowledge_polling (void)
{
    do
    {
        /* Set arguments for R_RIIC_MasterSend. */
        iic_info_m.p_slv_adr = addr_eeprom;        /* Pointer to the slave address storage buffer */
        iic_info_m.p_data1st = (uint8_t*) FIT_NO_PTR; /* Pointer to the first data storage buffer */
        iic_info_m.cnt1st = 0;                     /* First data counter (number of bytes) */
        iic_info_m.p_data2nd = (uint8_t*) FIT_NO_PTR; /* Pointer to the second data storage buffer */
        iic_info_m.cnt2nd = 0;                     /* Second data counter (number of bytes) */
        iic_info_m.callbackfunc = &callback_master;   /* Callback function */

        /* Master send start. */
        ret = R_RIIC_MasterSend(&iic_info_m);
        if (RIIC_SUCCESS == ret)
        {
            /* Waitting for R_RIIC_MasterSend completed. */
            while (RIIC_COMMUNICATION == iic_info_m.dev_sts)
            {
                /* do nothing */
            }

            /* Slave returns NACK. Set retry interval. */
            if (RIIC_NACK == iic_info_m.dev_sts)
            {
                /* Waitting for retry interval 100us. */
                R_BSP_SoftwareDelay(100, BSP_DELAY_MICROSECS);
            }
        }
        else
        {
            /* This software is for single master.
               Therefore, return value should be always 'RIIC_SUCCESS'. */
            while (1)
            {
                nop();    /* error */
            }
        }
    } while (RIIC_FINISH != iic_info_m.dev_sts);

} /* End of function acknowledge_polling() */
```

図 5.9　1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする例(4)

```
/***********************************************************************************************
* Function Name: eeprom_read
* Description  : This function is sample of EEPROM read function using R_RIIC_MasterReceive.
* Arguments    : none
* Return Value : none
***********************************************************************************************/
static void eeprom_read (void)
{
    /* Set arguments for R_RIIC_MasterReceive. */
    iic_info_m.p_slv_adr = addr_eeprom;         /* Pointer to the slave address storage buffer */
    iic_info_m.p_data1st = access_addr1;        /* Pointer to the first data storage buffer */
    iic_info_m.cnt1st = 1;               /* First data counter (number of bytes)(to be updated) */
    iic_info_m.p_data2nd = master_store_area;   /* Pointer to the second data storage buffer */
    iic_info_m.cnt2nd = 16;              /* Second data counter (number of bytes)(to be updated) */
    iic_info_m.callbackfunc = &callback_master; /* Callback function */

    /* Master send receive start. */
    ret = R_RIIC_MasterReceive(&iic_info_m);
    if (RIIC_SUCCESS == ret)
    {
        /* Waitting for R_RIIC_MasterSend completed. */
        while (RIIC_COMMUNICATION == iic_info_m.dev_sts)
        {
            /* do nothing */
        }

        if (RIIC_NACK == iic_info_m.dev_sts)
        {
            /* Slave returns NACK. The slave address may not correct.
               Please check the macro definition value or hardware connection etc. */
            while (1)
            {
                nop();    /* error */
            }
        }
    }
    else
    {
        /* This software is for single master.
           Therefore, return value should be always 'RIIC_SUCCESS'. */
        while (1)
        {
                nop();    /* error */
        }
    }

} /* End of function eeprom_read() */
```

図 5.10　1 つのチャネルで 1 つのスレーブデバイスに連続アクセスする例(5)

6. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
(最新版をルネサス エレクトロニクスホームページから入手してください。)

テクニカルアップデート／テクニカルニュース
(最新の情報をルネサス エレクトロニクスホームページから入手してください。)

ユーザーズマニュアル：開発環境
RX ファミリ C/C++コンパイラ CC-RX ユーザーズマニュアル (R20UT3248)
(最新版をルネサス エレクトロニクスホームページから入手してください。)

テクニカルアップデートの対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- TN-RX*-A012A/J

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	2013.07.01	—	初版発行
1.10	2013.09.30	—	戻り値の変更
1.20	2013.11.15	3	FIT モジュール改修のため、割り込みスタックサイズを変更 「116 バイト」→「120 バイト」
		4	RIIC FIT モジュール改修のため、ROM サイズを変更 「7125 バイト」→「7340 バイト」 RIIC FIT モジュール改修のため、最大使用割り込みスタックサイズを変更 「116 バイト」→「120 バイト」
		44	図 5.2 マスタ送信処理（R_RIIC_MasterSend()呼び出し）時の状態遷移図 一部変更
		45	図 5.3　マスタ受信処理（R_RIIC_MasterReceive()呼び出し）時の状態遷移図　一部変更
1.30	2014.04.01	—	FIT モジュールの RX100 シリーズ対応
1.40	2014.10.01	—	FIT モジュールの RX64M グループ対応
		—	タイムアウト機能対応
		4	RIIC FIT モジュール改修のため、必要メモリサイズの変更 ・ROM サイズ 「7340 バイト」→「9144 バイト」 ・RAM サイズを変更 「47 バイト」→「37 バイト」 ・最大使用ユーザスタック 「208 バイト」→「232 バイト」 ・最大使用割り込みスタック 「120 バイト」→「160 バイト」
		17	図 1.14 RIIC FIT モジュールの状態遷移図　一部変更
		18	表 1.2 状態遷移時のデバイス状態フラグの一覧　一部変更
		19	1.3.8 タイムアウト検出機能　追加
		21	2.3 サポートされているツールチェーン　一部変更
		22 ～ 25	2.7 コンパイル時の設定　オプション追加、および一部削除
		26	2.10 戻り値　一部変更
		29 ～ 41	3 API 関数 「Return Value」に RIIC_ERR_TMO を追加 「Example」変更
		43	3.6 R_RIIC_Control() 「Specal Notes」追加。
		46	表 5.1 プロトコル制御のための状態一覧(enum r_riic_api_status_t) 一部変更 表 5.2 プロトコル制御のためのイベント一覧(enum r_riic_api_event_t)　一部変更
		48	図 5.2　マスタ送信処理（R_RIIC_MasterSend()呼び出し）時の状態遷移図　一部変更
		49	図 5.3　マスタ受信処理（R_RIIC_MasterReceive()呼び出し）時の状態遷移図　一部変更

Rev.	発行日	改訂内容	
		ページ	ポイント
1.40	2014.10.01	50	図 5.4　スレーブ送受信処理（R_RIIC_SlaveTransfer()呼び出し）時の状態遷移図　一部変更
		51	表 5.3 プロトコル状態遷移表(gc_riic_mtx_tbl [ ] [ ])　一部変更
		52	表 5.4 プロトコル状態遷移登録関数一覧　一部変更
		53	表 5.5 状態遷移時の各フラグの状態一覧　一部変更
		58 ～ 59	5.3 タイムアウトの検出、および検出後の処理 追加
		ﾌﾟﾛｸﾞﾗﾑ	ソフトウェア不具合のため、RIIC FIT モジュールを改修 ■内容 アービトレーションロストが発生後、スレーブの 通信ができずバスロックが発生する場合があります。 ■発生条件 次の 4 つの条件に該当したとき ・RIIC FIT モジュール Rev.1.30 以前のバージョンをご使用されている ・マルチマスタ環境で、自デバイスがマスタ、スレーブ通信の両方を行う。 ・マスタ通信中にアービトレーションロストを検出する ・マスタ受信とスレーブ受信以外の通信を行う ■対策 RIIC FIT モジュール Rev1.40 をご使用ください。
1.50	2014.11.14	—	FIT モジュールの RX113 グループ対応
1.60	2014.12.15	—	FIT モジュールの RX71M グループ対応
1.70	2014.12.15	—	FIT モジュールの RX231 グループ対応
1.80	2015.10.31	—	FIT モジュールの RX130 グループ、RX230 グループ、RX23T グループ対応
		32	「3.2 R_RIIC_MasterSend()」「Example」を変更
		35	「3.3 R_RIIC_MasterReceive()」「Example」を変更
		38、39	「3.4 R_RIIC_SlaveTransfer()」「Example」を変更
1.90	2016.03.04	—	FIT モジュールの RX24T グループ対応
		4	「表 1.2 必要メモリサイズ」の説明を変更
		22、26	「2.7 コンパイル時の設定」に r_riic_rx_pin_config.h について説明を追記
		—	「マスタ複合」の表記を「マスタ送受信」に変更
2.00	2016.10.01	—	FIT モジュールの RX65N グループ対応
		27	コードサイズの説明「表 1.2 必要メモリサイズ」から「2.8 コードサイズ」に変更。
		ﾌﾟﾛｸﾞﾗﾑ	チャネル 2 の RXI、TXI の割り込みステータスフラグを参照する定義(RIIC_IR_RXI2, RIIC_IR_TXI2)の誤りを修正。

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2016.10.01	ﾌﾟﾛｸﾞﾗﾑ	ソフトウェア不具合のため、RIIC FIT モジュールを改修 ■内容 Rev.1.90 の RX110 の端子機能設定処理に誤りがあるため、RX110 を使用した場合、ビルドエラーが発生します。 ■発生条件 「2.10 モジュールの追加方法」を参考に、RX110 の新規プロジェクトを作成し、本モジュールの Rev.1.90 を組み込んだ後、プロジェクトをビルドした時。 ■対策 RX110 の riic_mcu_mpc_enable 関数および riic_mcu_mpc_disable 関数の端子機能設定処理を修正しました。 RIIC FIT モジュール Rev2.00 をご使用ください。
2.10	2017.06.02	—	「対象デバイス」に RX24U グループを追加
		—	RX24T-512KB 版に対応
		22	「2.4 使用する割り込みベクタ」を追加
		31	「2.10 コールバック関数」を追加
		31	「2.12 モジュールの追加方法」を変更
		51	「4 端子設定」を追加
		66、67	「5.4 動作確認環境」を追加
		69	「5.5 トラブルシューティング」を追加
2.20	2017.08.31	—	RX65N-2MB 版に対応
		—	RX130-512KB 版に対応
		1	「関連ドキュメント」にスマート・コンフィグレータ ユーザーガイドを追加
		22	「2.4 使用する割り込みベクタ」を変更 表 2.1 使用する割り込みベクター覧に RX65N-2MB で使用する割り込みベクタを追加
		25	「2.7 コンパイル時の設定」 RIIC_CFG_PORT_SET_PROCESSING の説明を変更
		25 ～ 27	「2.7 コンパイル時の設定」チャネル 1 に関する定義を追加
		31	「2.12 モジュールの追加方法」を変更
		52	「4 端子設定」を変更
		68	「5.4 動作確認環境」表 5.11 動作確認環境 (Rev.2.20)を追加
		70 ～ 74	「5.6 サンプルコード」追加
		75	「6. 提供するモジュール」削除
		ﾌﾟﾛｸﾞﾗﾑ	チャネル 1 に関する定義を追加

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. **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（予約領域）のアクセス禁止**
   【注意】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、将来の機能拡張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。
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、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、内部ROM、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、ノイズ輻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、その他の不適切に使用しないでください。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、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、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。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   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. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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。

(Rev.3.0-1　2016.11)

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　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■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：https://www.renesas.com/contact/