RX600, RX200 シリーズ

RIIC を使った I²C シングルマスタ制御ソフトウェア

R01AN1254JJ0103
Rev.1.03
2015.01.30

要旨

本アプリケーションノートでは、RX ファミリ I²C バスインタフェース RIIC（以下、RIIC）を使用したシングルマスタ制御方法とサンプルコードの使用方法を説明します。

本サンプルコードでは、スレーブデバイスを制御するためのソフトウェアを上位層、I²C シングルマスタ基本プロトコル制御を実現するソフトウェアを下位層としています。上位層は下位層で用意されているプロトコルを組み合わせてスレーブデバイスを制御します。

本サンプルコードは、I²C シングルマスタ制御するための下位層に位置するソフトウェアです。本サンプルコードを使用して、上位層にあたるスレーブデバイスを制御するためのソフトウェアを作成してください。

なお、スレーブデバイス制御のためのソフトウェア例を別途用意していますので、必要に応じて入手してください。

対象デバイス

対応 MCU　　RX62N, RX63N, RX63T, RX210, RX21A

動作確認に使用したデバイス　　ルネサス エレクトロニクス製 R1EX24xxx シリーズ I²C Serial EEPROM

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。

目次

1. 仕様

RX ファミリの I²C バスインタフェースを使用し、I²C シングルマスタ制御を行います。スレーブデバイス制御のための上位層に位置するソフトウェアを入手、もしくは作成してください。

表 1-2に使用する周辺機器と用途を、図 1-1に使用例を示します。

以下に、機能概略を示します。

- マスタデバイスを RX ファミリとし、I²C バスインタフェースを使った I²C シングルマスタ用デバイスドライバです。
- I²C バス規格のプロトコルを実現します。本サンプルコードでは、マスタ送信、マスタ受信、マスタ複合（マスタ送信→マスタ受信）をサポートします。
- マスタ送信は 4 種類の送信パターンを設定可能です。動作パターンを表 1-1に示します。
- 複数のチャネルをサポートします。複数チャネルを使用した同時通信が可能です。
- １つのチャネル・バス上の複数かつ型名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。
- 各プロトコル制御を開始する関数（開始関数）と、通信を監視し処理を進める関数（アドバンス関数）により通信を実現します。アドバンス関数のリターン値により、通信状態を判断できます。
- 開始関数はスタートコンディションを生成します。それ以降は、ストップコンディションが生成されるまでは、アドバンス関数をコールし処理を進めます。
- スタートコンディション生成、スレーブアドレス送信、データ送信、データ受信、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割り込みが発生します。
- 通信速度レートは、ユーザ設定が可能です。[対応レート：～400KHz(max)]。ただし、同一チャネルに複数のデバイスが接続されている場合には、通信速度が最小のスレーブデバイスにあわせて設定してください。
- 通信中にノイズ等の影響により、通信が停止した場合（割り込みが発生しない等）、アドバンス関数によりエラーを返すことができます。アドバンス関数のコール回数が上限を超えた場合、異常による通信停止と判断し、“未応答エラー”を返します。上限値はユーザで設定できます。
- NACK エラーが発生した場合、ストップコンディションを生成します。
- SCL クロック生成処理があります。ノイズ等の影響によりマスタとスレーブ間で同期ズレが発生し、SDA=Low ホールドになった場合、SCL 疑似クロック生成関数をコールし、スレーブデバイス内部の状態を正常終了させることができます。
- 7bit アドレスデバイス間の通信のみをサポートします。特別なアドレス（ジェネラルコール等のアドレス）をサポートしていません。

表 1-1　マスタ送信の動作パターン

	ST 生成	スレーブアドレス送信	1st データ送信	2nd データ送信	SP 生成
パターン１	○	○	○	○	○
パターン２	○	○	—	○	○
パターン３	○	○	—	—	○
パターン４	○	—	—	—	○

備考

ST：スタートコンディション、SP：ストップコンディション

表1-2　使用する周辺機能と用途

周辺機能	用途
RIIC	I²C バスインタフェース 1ch（必須）

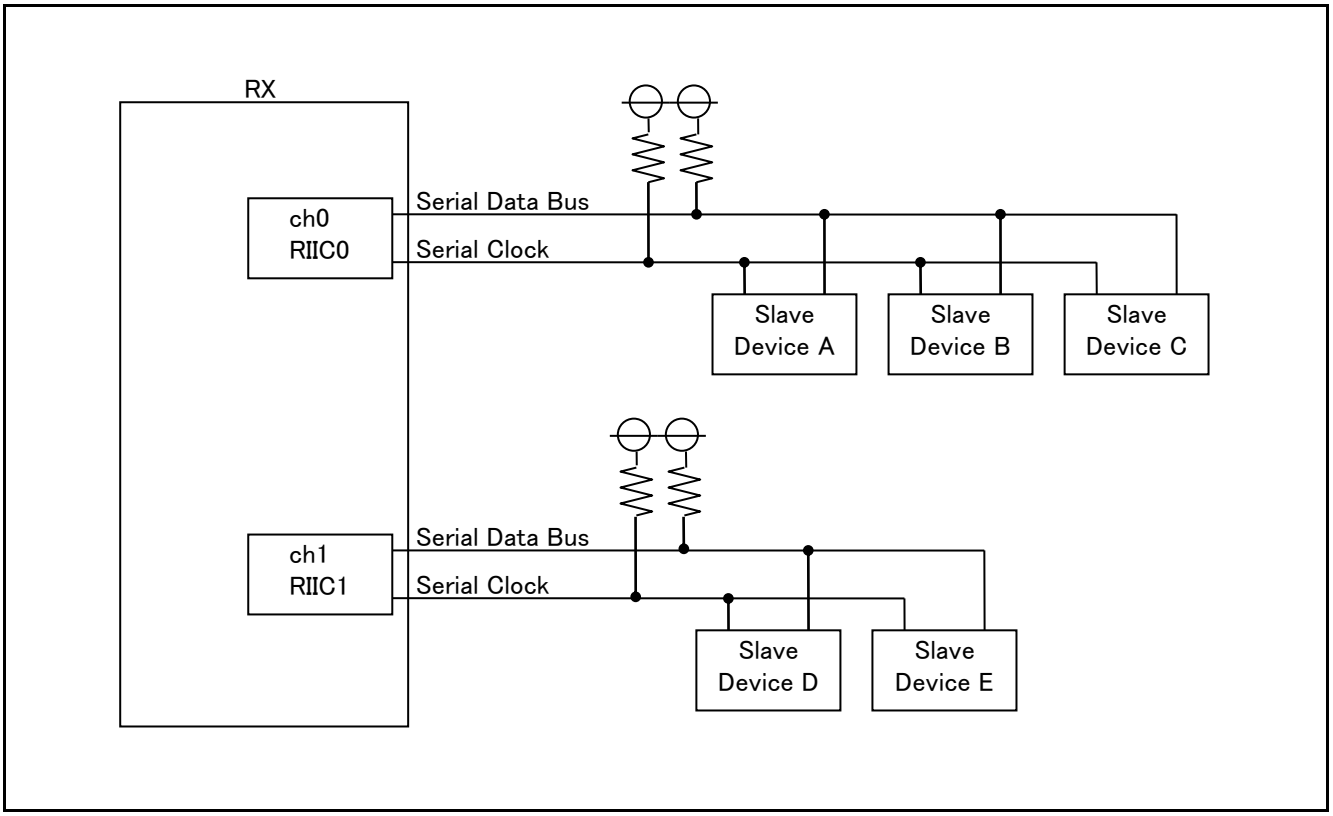


図1-1　使用例

2. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

(1) RX62N の場合

表2-1 動作確認条件

項目	内容
使用マイコン	RX62N グループ（プログラム ROM 512KB RAM 64KB）
使用メモリ	ルネサス エレクトロニクス製 R1EV24xxx/ R1EX24xxx/HN58X24xxx シリーズ I²C Serial EEPROM
動作周波数	ICLK：96MHz、PCLK：48MHz
動作電圧	3.3V
統合開発環境	ルネサス エレクトロニクス製 High-performance embedded Workshop Version 4.09.01.007
C コンパイラ	ルネサス エレクトロニクス製 RX ファミリ用 C/C++コンパイラケッケージ （ツールチェーン 1.2.1.0）
	コンパイルオプション 総合開発環境のデフォルト設定（※１）を使用しています。 ※１：最適化レベル”2”、最適化方法”サイズ優先”
エンディアン	ビッグエンディアン／リトルエンディアン
サンプルコードのバージョン	Ver.1.13
使用ソフトウェア	Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ) Ver1.01
使用ボード	Renesas Starter Kit for RX62N

(2) RX63N の場合

表2-2 動作確認条件

項目	内容
使用マイコン	RX63N グループ（プログラム ROM 1MB RAM 128KB）
使用メモリ	ルネサス エレクトロニクス製 R1EV24xxx/ R1EX24xxx/HN58X24xxx シリーズ I²C Serial EEPROM
動作周波数	ICLK：96MHz、PCLKB：48MHz
動作電圧	3.3V
統合開発環境	ルネサス エレクトロニクス製 High-performance embedded Workshop Version 4.09.01.007
C コンパイラ	ルネサス エレクトロニクス製 RX ファミリ用 C/C++コンパイラケッケージ （ツールチェーン 1.2.1.0）
	コンパイルオプション 総合開発環境のデフォルト設定（※１）を使用しています。 ※１：最適化レベル”2”、最適化方法”サイズ優先”
エンディアン	ビッグエンディアン／リトルエンディアン
サンプルコードのバージョン	Ver.1.13
使用ソフトウェア	Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ) Ver1.01
使用ボード	Renesas Starter Kit for RX63N

(3) RX63T の場合

表2-3 動作確認条件

項目	内容
使用マイコン	RX63T グループ（プログラム ROM 512KB RAM 64KB）
使用メモリ	ルネサス エレクトロニクス製 R1EV24xxx/ R1EX24xxx/HN58X24xxx シリーズ I²C Serial EEPROM
動作周波数	ICLK：96MHz、PCLKB：48MHz
動作電圧	3.3V
統合開発環境	ルネサス エレクトロニクス製 CubeSuite+ V2.00.00
C コンパイラ	ルネサス エレクトロニクス製 RX ファミリ用 C/C++コンパイラケッケージ （ツールチェーン 2.00.00）
	コンパイルオプション 総合開発環境のデフォルト設定（※１）を使用しています。 ※１：最適化レベル”2”、最適化方法”サイズ優先”
エンディアン	ビッグエンディアン／リトルエンディアン
サンプルコードのバージョン	Ver.1.13
使用ソフトウェア	Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ) Ver1.01
使用ボード	Renesas Starter Kit for RX63T

(4) RX210 の場合

表2-4 動作確認条件

項目	内容
使用マイコン	RX210 グループ（プログラム ROM 512KB RAM 64KB）
使用メモリ	ルネサス エレクトロニクス製 R1EV24xxx/ R1EX24xxx/HN58X24xxx シリーズ I²C Serial EEPROM
動作周波数	ICLK：50MHz、PCLKB：25MHz
動作電圧	3.3V
統合開発環境	ルネサス エレクトロニクス製 High-performance embedded Workshop Version 4.09.01.007
C コンパイラ	ルネサス エレクトロニクス製 RX ファミリ用 C/C++コンパイラケッケージ （ツールチェーン 1.2.1.0）
	コンパイルオプション 総合開発環境のデフォルト設定（※１）を使用しています。 ※１：最適化レベル”2”、最適化方法”サイズ優先”
エンディアン	ビッグエンディアン／リトルエンディアン
サンプルコードのバージョン	Ver.1.13
使用ソフトウェア	Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ) Ver1.01
使用ボード	Renesas Starter Kit for RX210

(5) RX21A の場合

表2-5 動作確認条件

項目	内容
使用マイコン	RX21A グループ（プログラム ROM 512KB RAM 64KB）
使用メモリ	ルネサス エレクトロニクス製 R1EV24xxx/ R1EX24xxx/HN58X24xxx シリーズ I²C Serial EEPROM
動作周波数	ICLK：50MHz、PCLKB：25MHz
動作電圧	3.3V
統合開発環境	ルネサス エレクトロニクス製 High-performance embedded Workshop Version 4.09.01.007
C コンパイラ	ルネサス エレクトロニクス製 RX ファミリ用 C/C++コンパイラケッケージ （ツールチェーン 1.2.1.0）
	コンパイルオプション 総合開発環境のデフォルト設定（※１）を使用しています。 ※１：最適化レベル”2”、最適化方法”サイズ優先”
エンディアン	ビッグエンディアン／リトルエンディアン
サンプルコードのバージョン	Ver.1.13
使用ソフトウェア	Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ) Ver1.01
使用ボード	株式会社北斗電子製 HSBRX21AP-B

3. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

Renesas R1EX24xxx シリーズ Serial EEPROM 制御ソフトウェア(R01AN1075JJ)

4. 周辺機能説明

RX ファミリの I²C 制御周辺機能として、I²C バスインタフェースとシリアルコミュニケーションインタフェースの簡易 I²C があります。

本アプリケーションノートでは、I²C バスインタフェースを使用します。

5. ハードウェア説明

5.1 使用端子一覧

表 5-1に使用端子と機能を示します。

表5-1 使用端子と機能

端子名	入出力	内容
SCL （図 5-1の SCL）	出力	シリアル・クロック出力
SDA （図 5-1の SDA）	入出力	シリアル・データ入出力

5.2 ハードウェア構成例

図 5-1にRX ファミリ I2C バスインタフェースと I2C スレーブデバイスの接続例を示します。シリアル・クロック・ラインおよびシリアル・データ・バス・ラインは、出力が N-ch オープン・ドレーンのため、外部にプルアップ抵抗が必要です。システムに応じて適正な抵抗を検討してください。また、各信号ラインの回路的マッチングを取るためのダンピング抵抗やコンデンサの付加を検討してください。

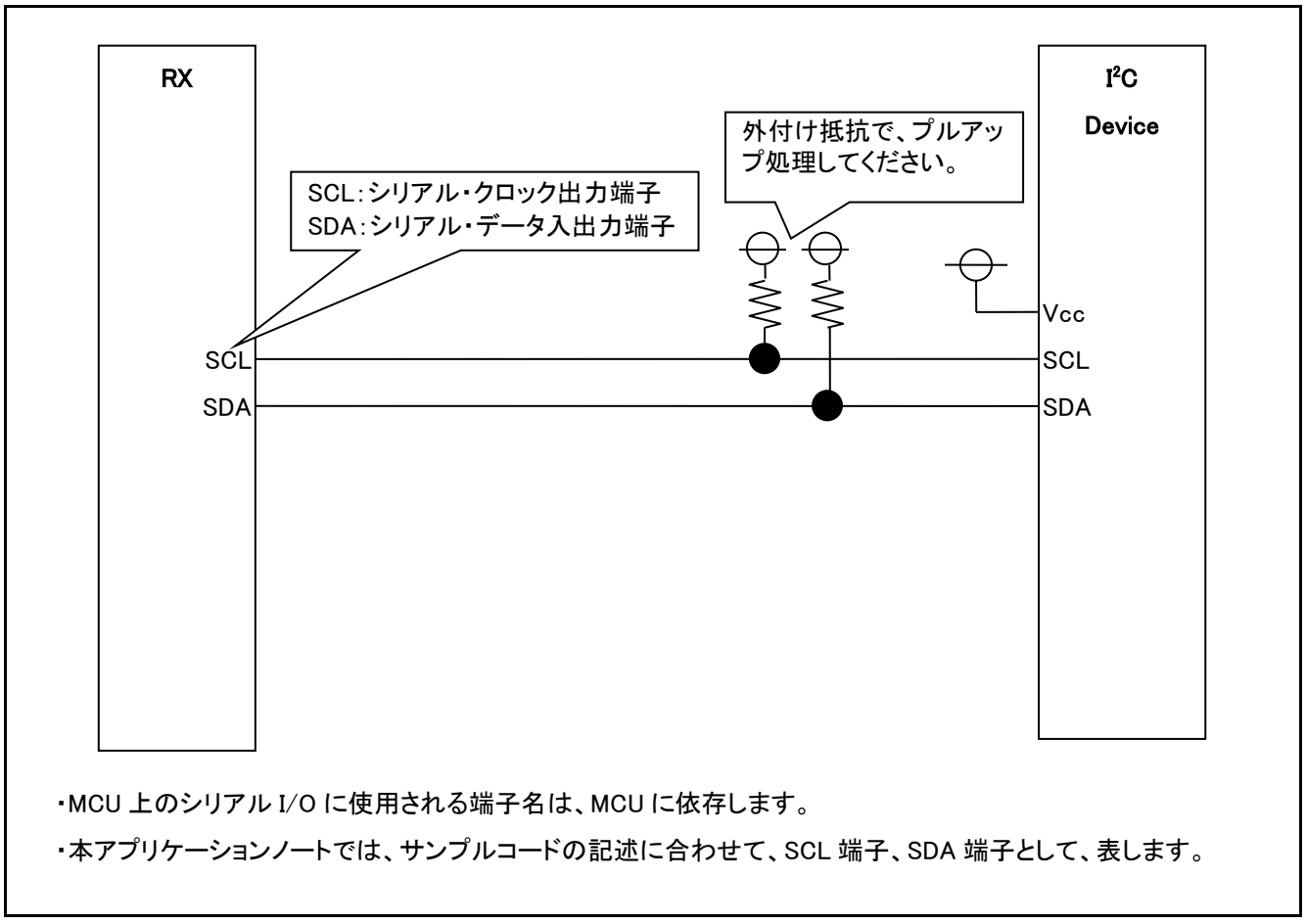


図5-1 RX ファミリ I²C バスインタフェースと I²C スレーブデバイスの接続例

5.3 複数スレーブデバイス制御

本サンプルコードでは複数のチャネルをサポートします。また、１つのチャネル・バス上に、複数かつ型名が異なるスレーブデバイスを搭載し、制御できます。ただし、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。

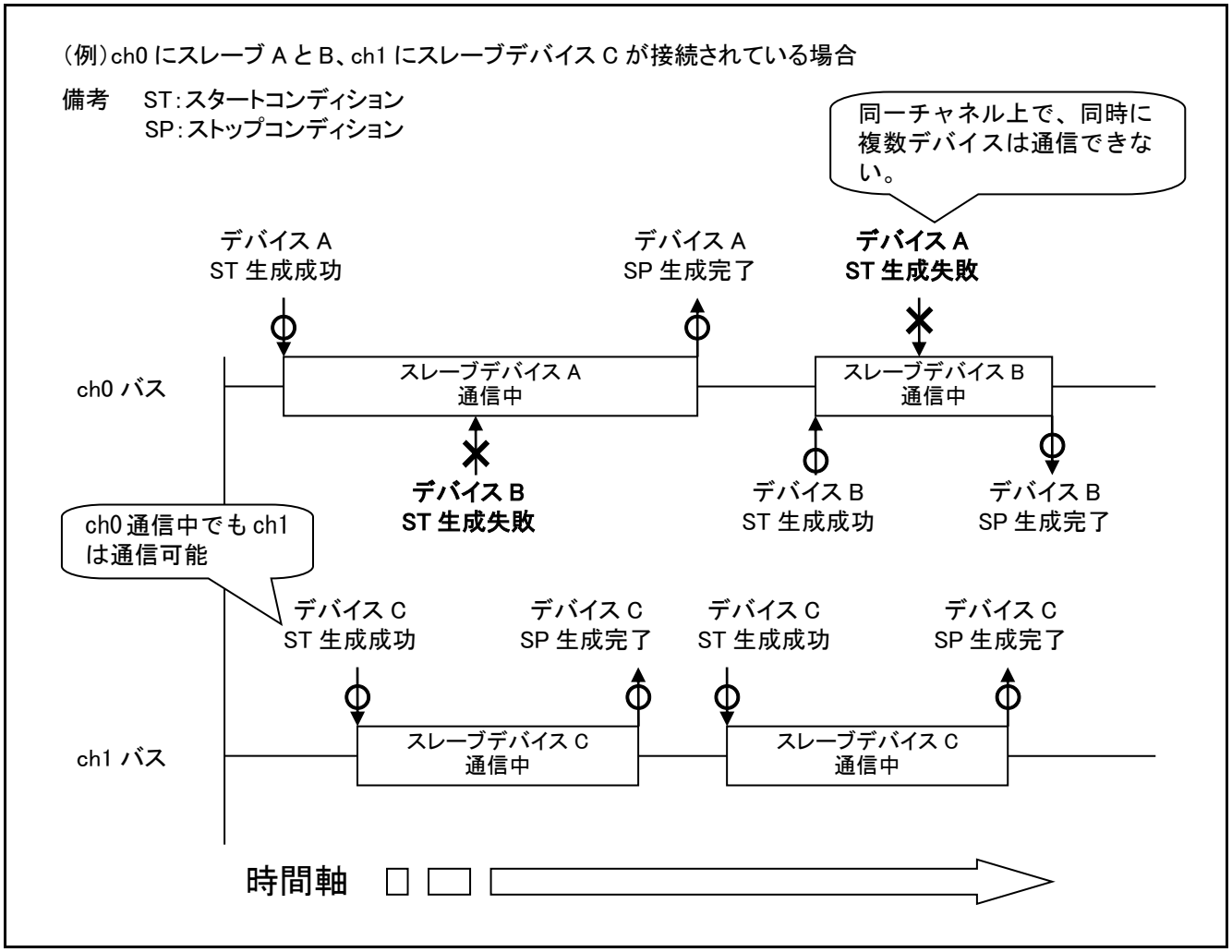


図 5-2　複数スレーブデバイス制御例

5.4 最大転送速度

最大 400KHz までの設定可能です。

ただし、標準モード・デバイスとファーストモード・デバイスが混載されるチャネルでは、標準モードの最大 100KHz に設定する必要があります。以下に、混載バス・システムでの最大転送速度を示します。

表 5-2　混載バス・システムでの最大転送速度

通信デバイス	混載デバイス	
	ファーストモード	標準モード
ファーストモード	0 – 400kHz	0 – 100kHz
標準モード	0 – 100kHz	0 – 100kHz

6. ソフトウェア説明

6.1 ソフトウェア構成

スレーブデバイスを制御するためのソフトウェアを上位層、I²C シングルマスタ基本プロトコル制御を実現するソフトウェアを下位層とします。上位層は下位層で用意されているプロトコルを組み合わせてスレーブデバイスを制御します。

本サンプルコードは、I²C シングルマスタ制御するための下位層に位置するソフトウェアです。

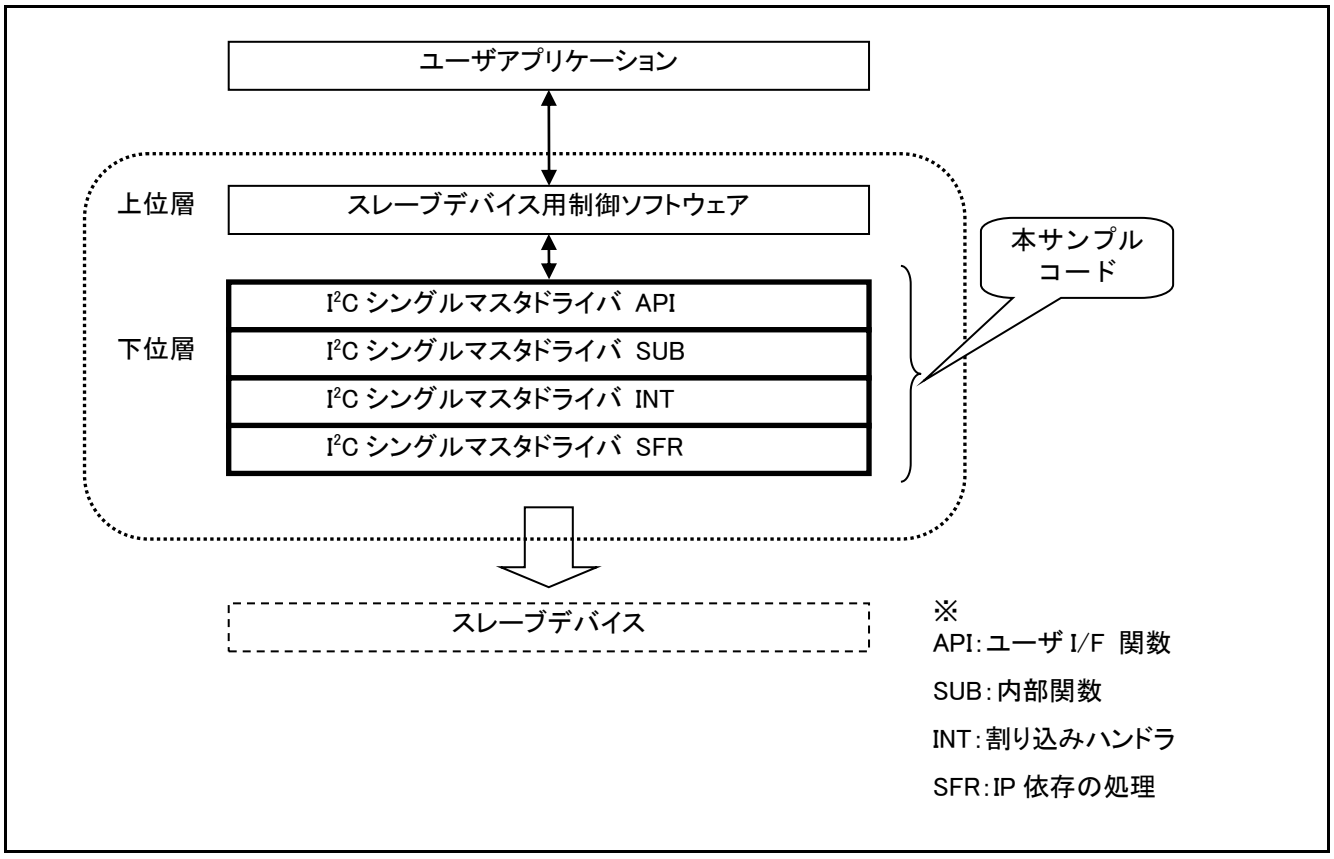


図 6-1 ソフトウェア構成

6.2 動作概要

RX ファミリ MCU I²C バスインタフェースを使用して、I²C シングルマスタ制御を実現します。

本サンプルコードは、シングルマスタとして以下のプロトコルを実現します。

表 6-1 制御プロトコル一覧

No	制御プロトコル	概要
1	マスタ送信	マスタ（MCU）からスレーブデバイスへデータを送信します。 4 種類の送信パターンを設定可能です。
2	マスタ受信	マスタ（MCU）はスレーブデバイスからデータを受信します。
3	マスタ複合	マスタ送信後、マスタ受信を行います。

6.2.1 マスタ送信

マスタ送信には、4 種類の送信パターンがあります。通信情報を管理する I²C 通信情報構造体の設定方法により機能を選択することができます。設定方法については、6.13.1 I2C 通信情報構造体を参照ください。

(1) パターン 1

マスタ（MCU）からスレーブデバイスへデータを送信します。

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。

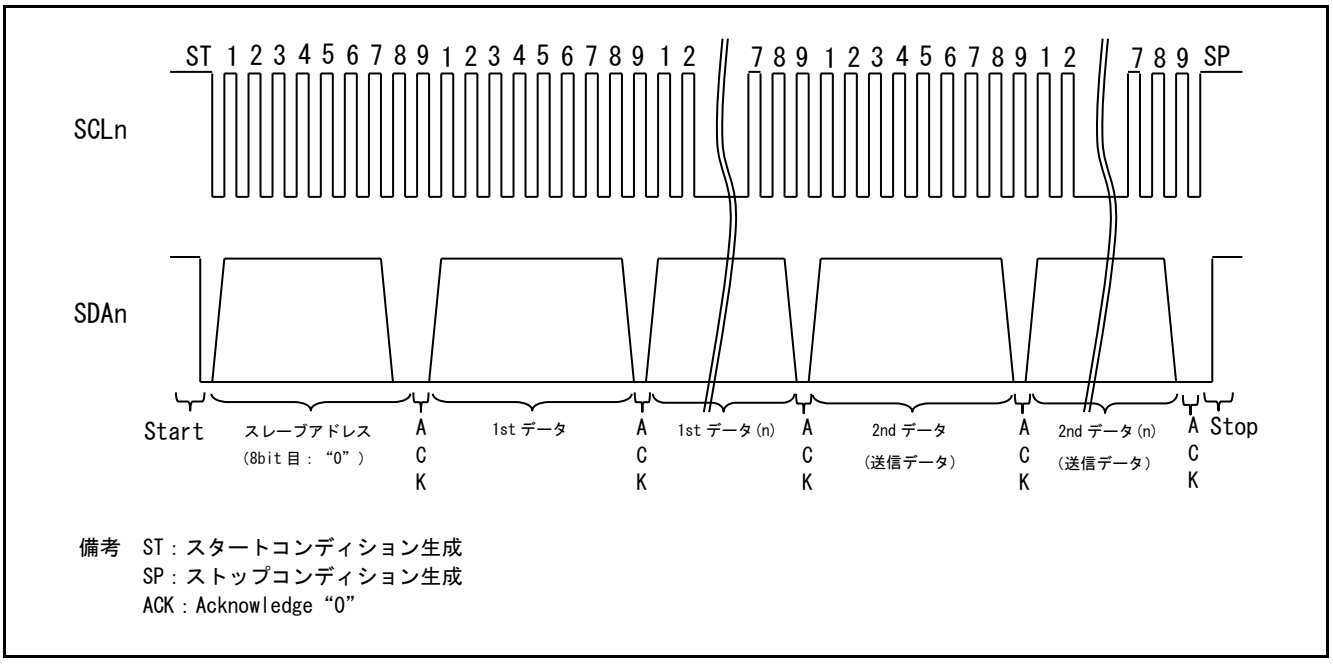


図 6-2 マスタ送信（パターン 1）信号図

(2) パターン 2

マスタ（MCU）からスレーブデバイスへデータを送信します。但し、1st データは送信しません。

スタートコンディション（ST）を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。

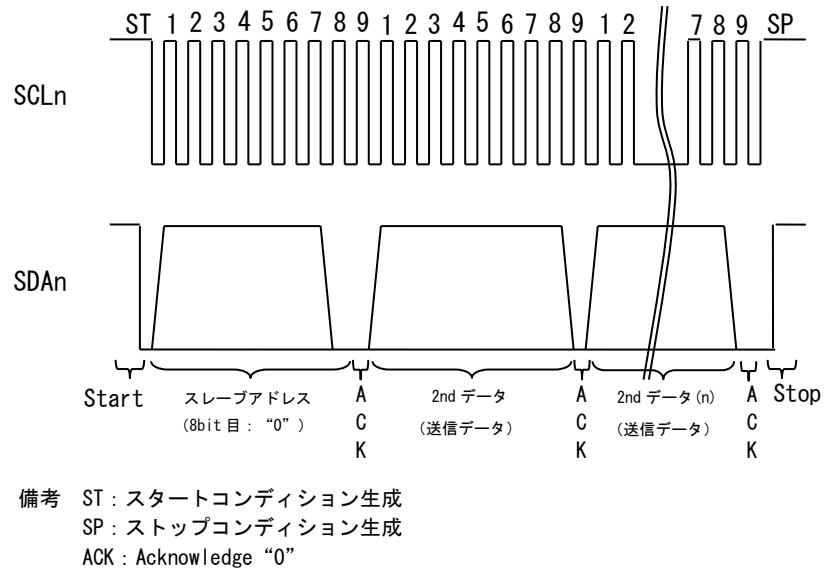


図 6-3 マスタ送信（パターン 2）信号図

(3) パターン 3

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通常時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。

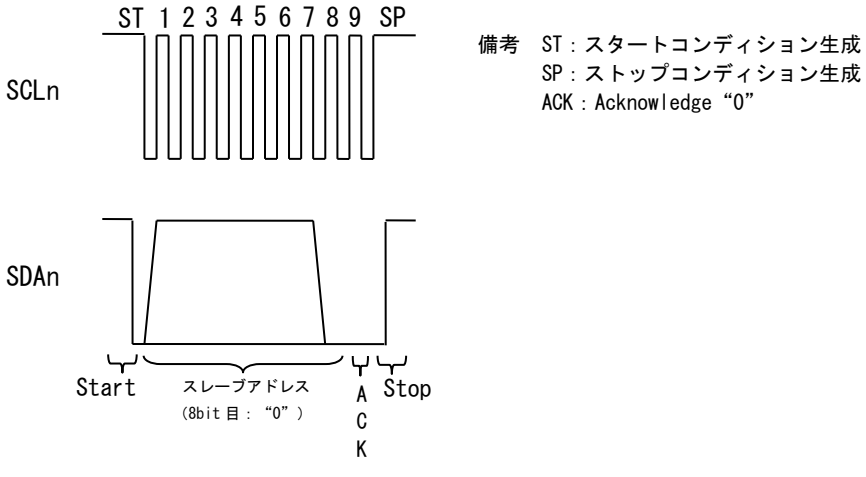


図 6-4 マスタ送信（パターン 3 動作）信号図

(4) パターン 4

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。

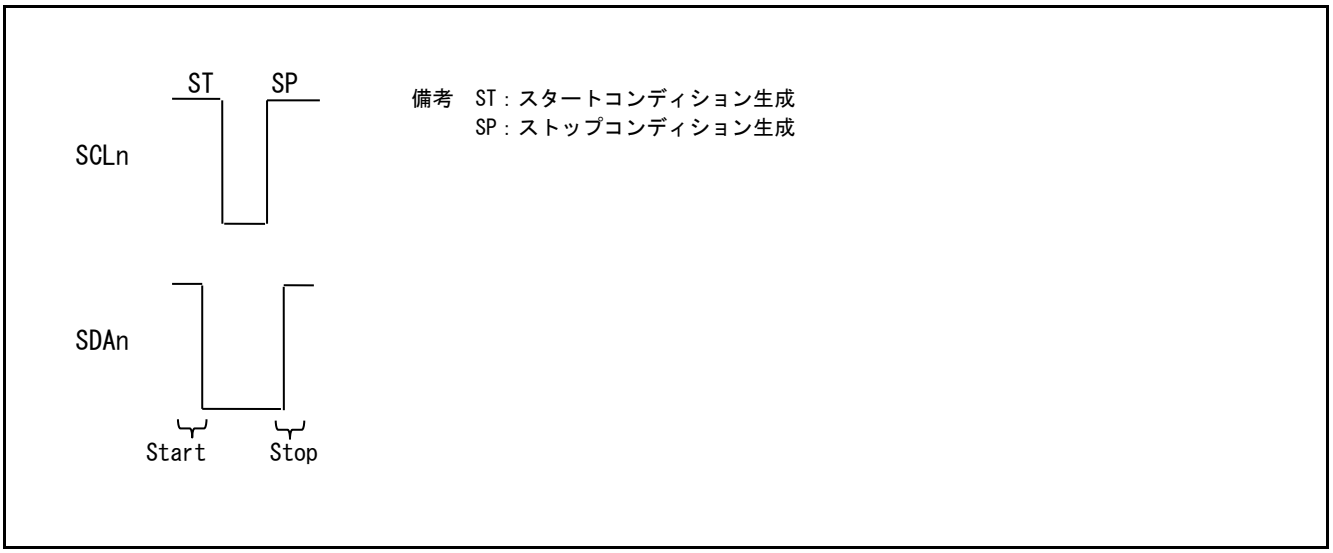


図 6-5 マスタ送信（パターン 4 動作）信号図

6.2.2 マスタ受信

マスタ（MCU）はスレーブデバイスからデータを受信します。

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“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。

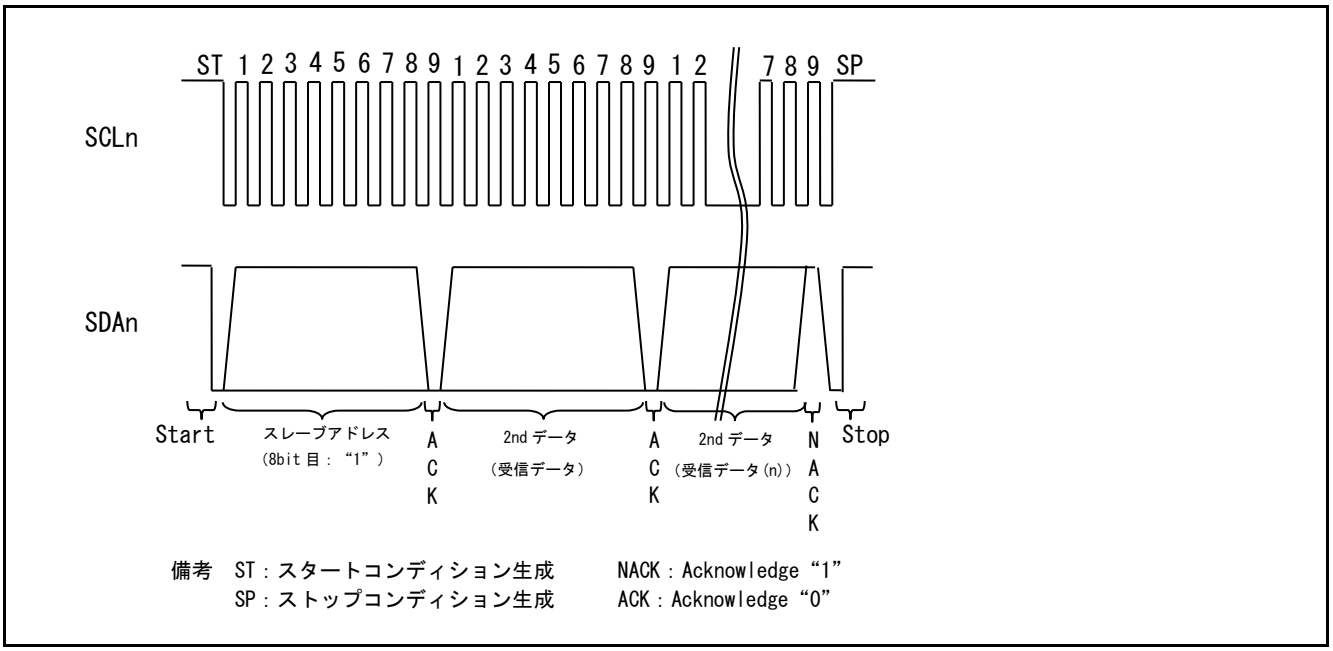


図 6-6 マスタ受信 信号図

6.2.3 マスタ複合

マスタ（MCU）からスレーブデバイスへデータを送信します（マスタ送信）。送信完了後、リスタートコンディションを生成し、転送方向を“1”（Read）に変更して、マスタはスレーブデバイスからデータを受信します（マスタ受信）。

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。

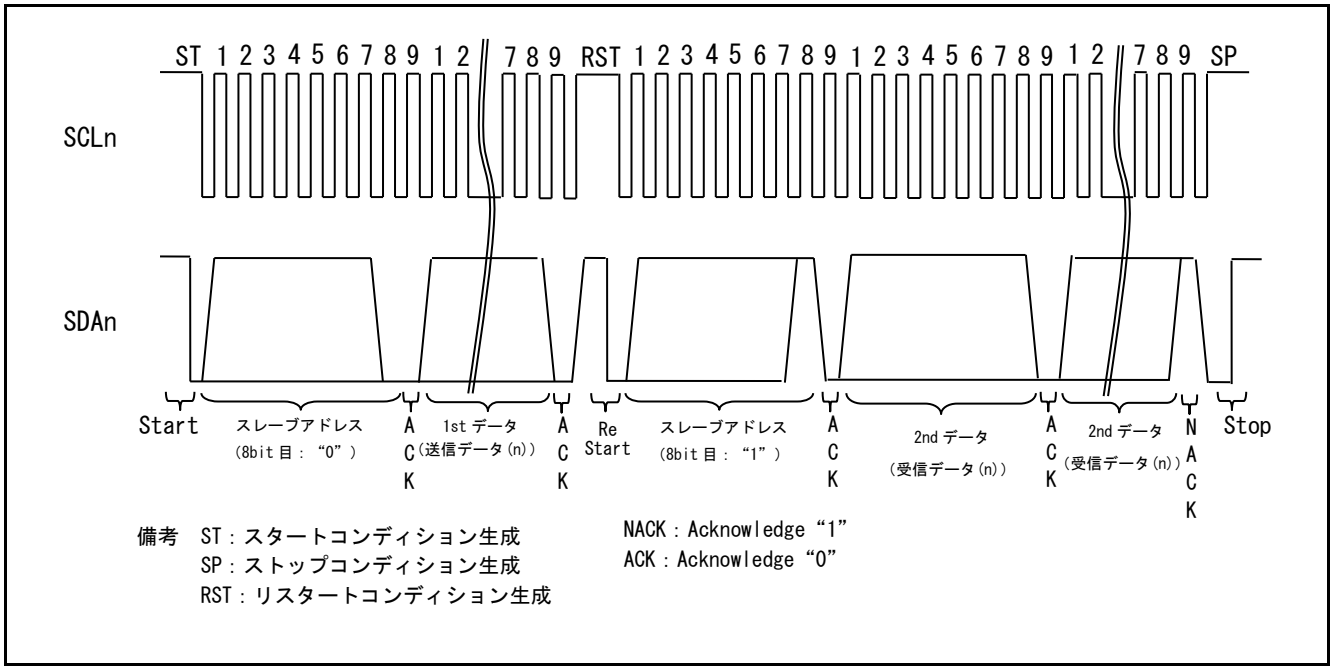


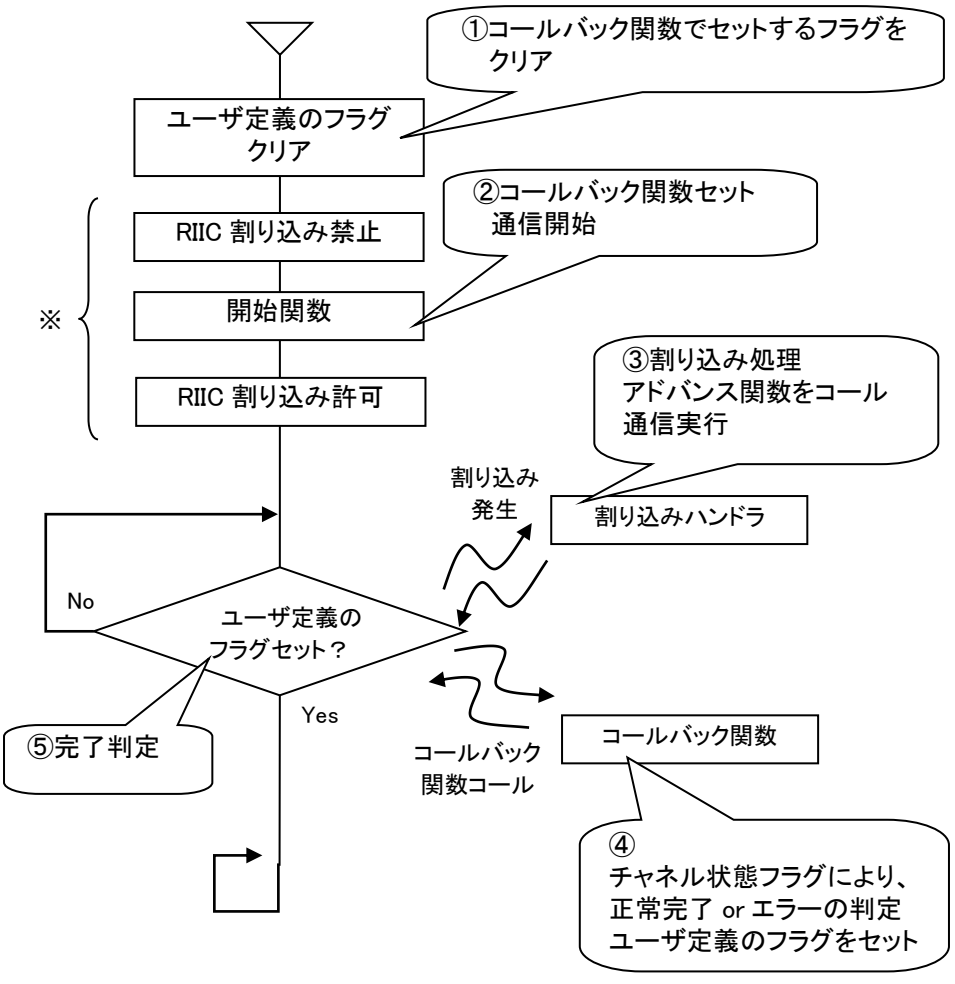
図 6-7 マスタ複合 信号図

6.3 ソフトウェア動作

開始関数をコールすることで通信を開始します。その後、ユーザがコールするアドバンス関数により、I²C 通信を進める処理を行います。アドバンス関数を RIIC 割り込みハンドラでコールする場合とメイン処理でコールする場合のソフトウェア動作を以下に示します。

(1) RIIC 割り込みハンドラでアドバンス関数をコールする場合

開始関数をコールすることで通信を開始します。通信完了はコールバック関数のコールによりフラグをセットする等して確認してください。コールバック関数は、正常終了またはエラー終了によりコールされます。正常またはエラーについては、チャネル状態フラグ（g_iic_ChStatus[]）を読み出すことで確認することができます。



※:
RIIC 割り込みハンドラでアドバンス関数をコールする場合、ユーザ側で各開始関数を呼び出す前後に RIIC 割り込み禁止と許可処理を追加してください。
この区間に RIIC 割り込みが発生し、RIIC 割り込みハンドラのアドバンス関数がコールされた場合、API の多重コールとなり、アドバンス関数は処理されずに終了します。これにより、その後の通信ができなくなります。

図 6-8 ソフトウェア動作例：RIIC 割り込みハンドラでアドバンス関数をコールする場合

(2) メイン処理でアドバンス関数をコールする場合

開始関数をコールすることで通信を開始します。通信が完了するまで、メイン処理でアドバンス関数コールを続けてください。通信中の状態は、アドバンス関数のリターン値により確認することができます。

割り込みが発生するとイベントフラグ（g_iic_Event[]）が セットされます。アドバンス関数はイベントフラグ（g_iic_Event[]）を監視しており、イベントフラグがセットされたことを確認すると通信を実行します。イベントフラグの詳細については、表 6-17を参照ください。

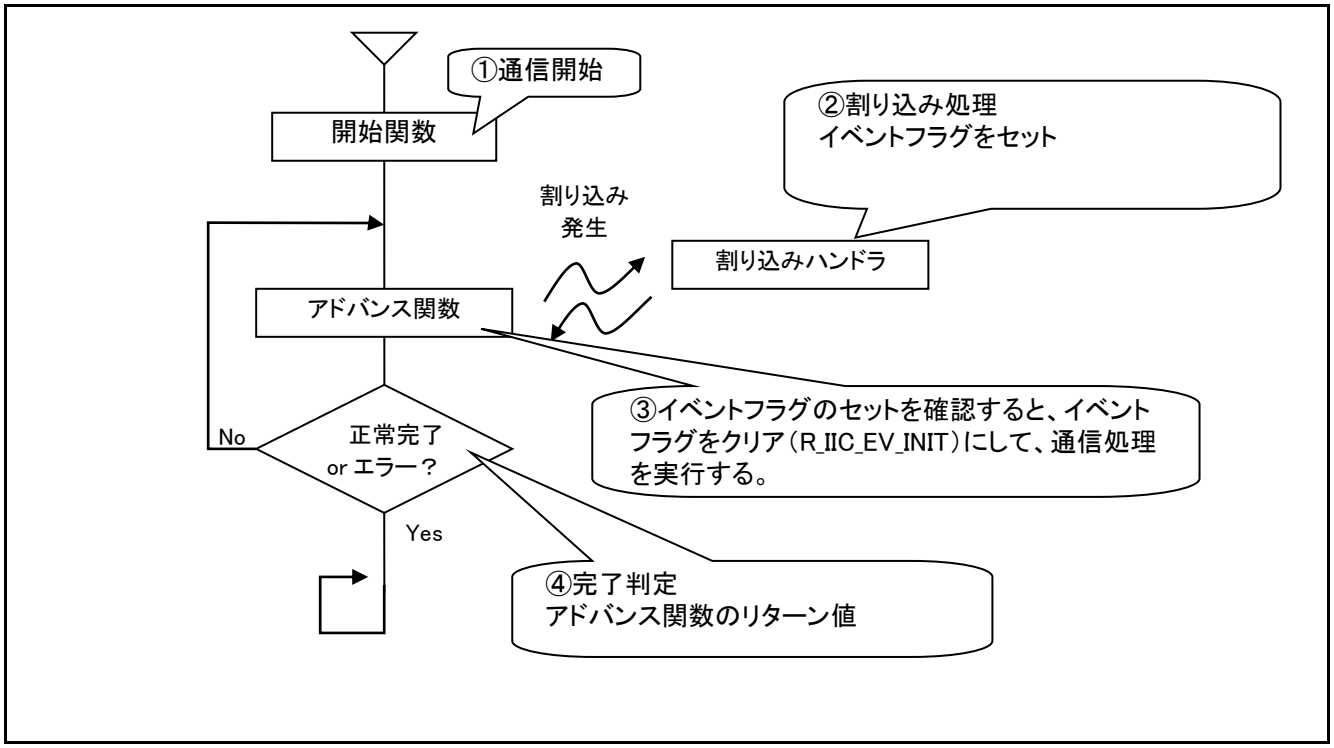


図 6-9 ソフトウェア動作例：メイン処理でアドバンス関数をコールする場合

6.4 ソフトウェア動作シーケンス

以下に、アドバンス関数を RIIC 割り込みハンドラでコールする場合とメイン処理でコールする場合の動作シーケンスを示します。

(1) RIIC 割り込みハンドラでアドバンス関数をコールする場合

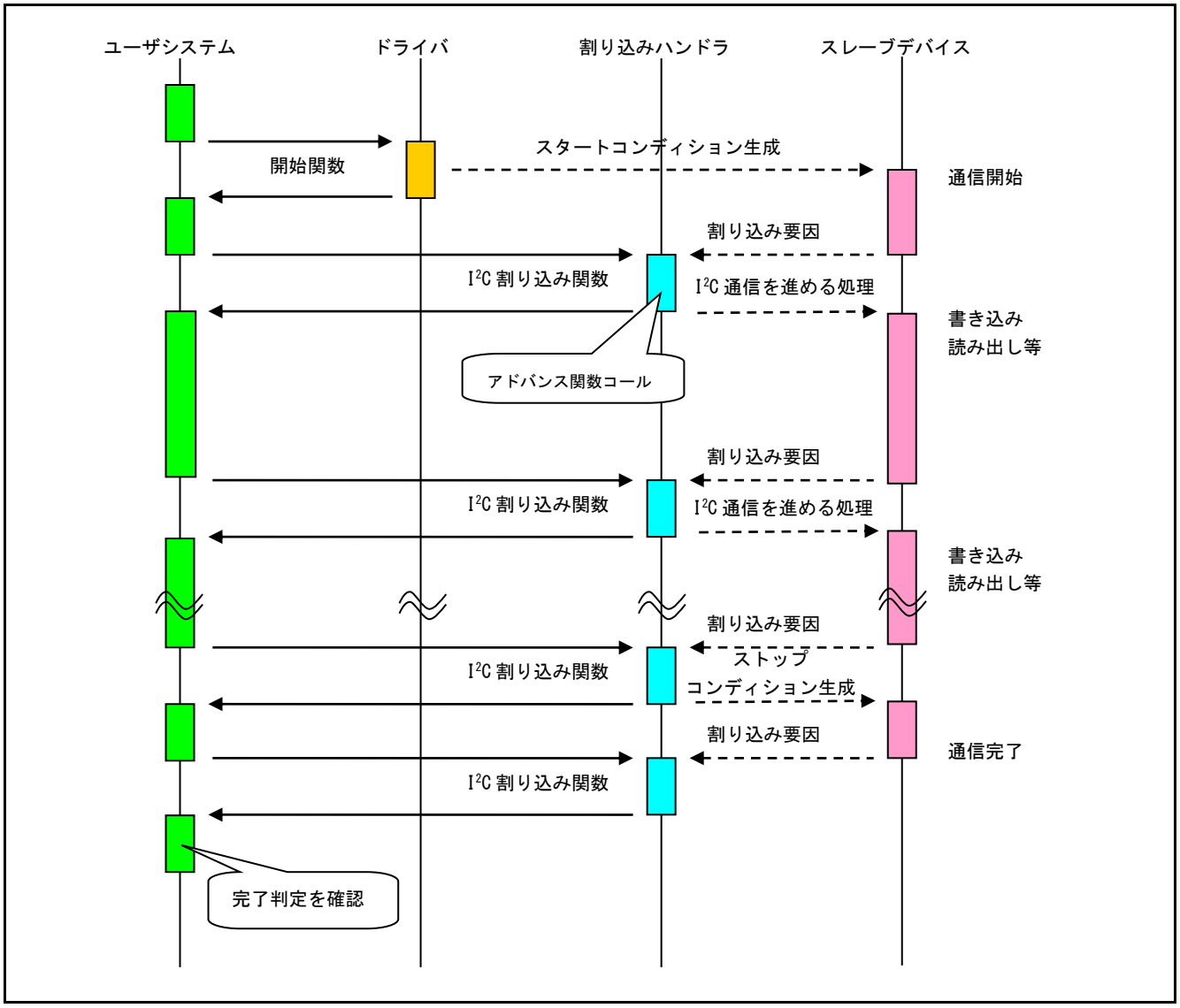


図 6-10 シーケンス図：RIIC 割り込みハンドラでアドバンス関数をコールする場合

(2) メイン処理でアドバンス関数をコールする場合

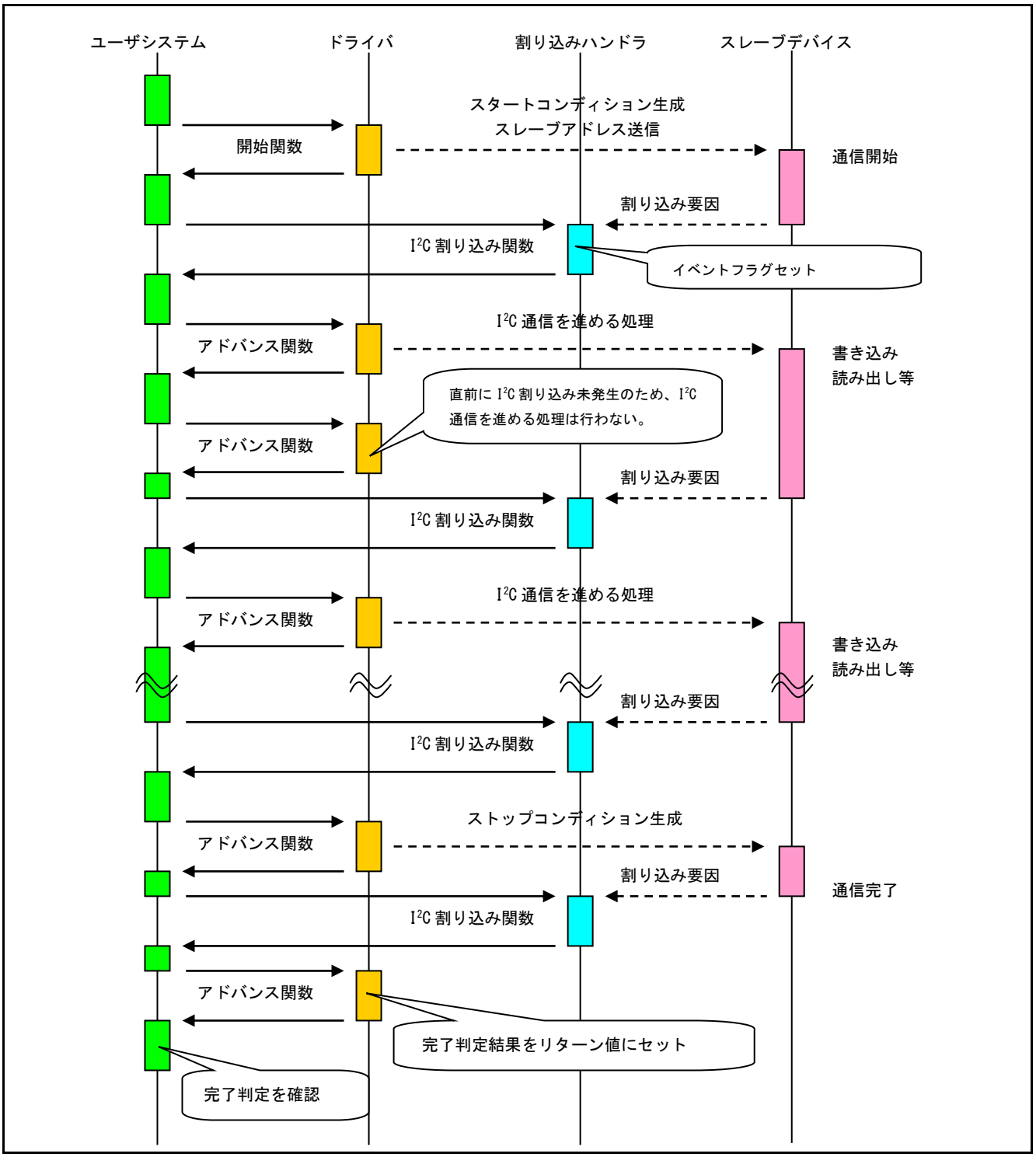


図 6-11 シーケンス図：メイン処理でアドバンス関数をコールする場合

6.5 スレーブデバイス制御の実現方法

(1) 各スレーブデバイス管理

使用するチャネルや通信するためのデータ等の情報は、構造体で管理します。制御するスレーブデバイス毎に構造体を設定することで、同一チャネル上の複数デバイスの通信を実現します。

構造体の詳細は、6.13.1 I2C 通信情報構造体を参照ください。

(2) 各チャネル状態管理

チャネル状態フラグ g_iic_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。チャネル状態フラグの詳細については、6.15 変数一覧の g_iic_ChStatus[]を参照ください。

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。I²C ドライバ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“R_IIC_IDLE/R_IIC_FINISH/R_IIC_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、“R_IIC_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毎に管理することで、複数チャネルの同時通信を実現します。

(3) 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*pDevStatus 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。使用方法については、8.6 同一チャネル上の複数のスレーブデバイスの制御方法 を参照ください。

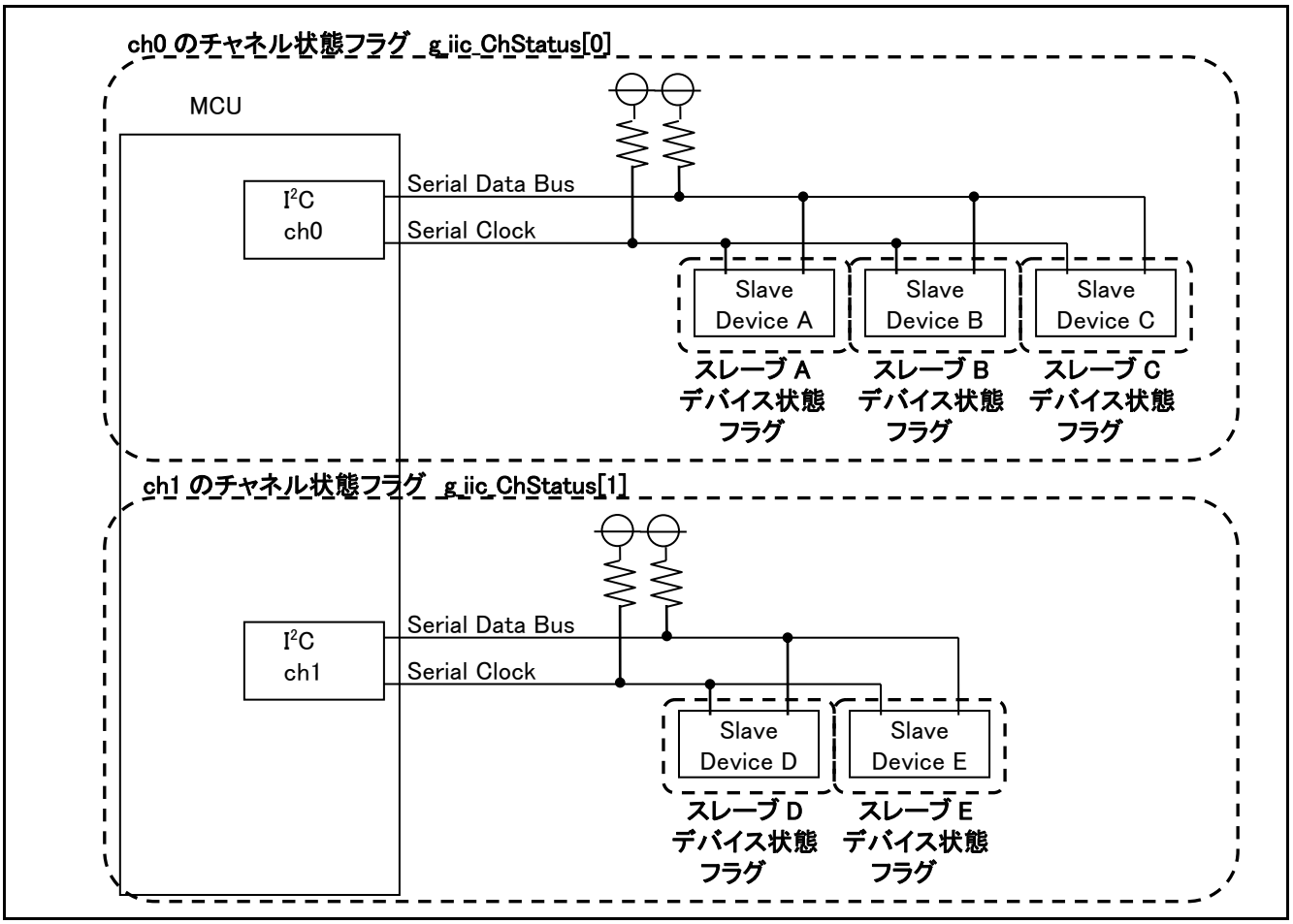


図 6-12 スレーブデバイス制御

6.6 通信の実現方法

本サンプルコードでは、スタートコンディションやスレーブアドレス送信等の処理を 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。

6.6.1 制御時の状態

プロトコル制御を実現するための状態を以下のように定義します。

表 6-2 プロトコル制御のための状態一覧（enum r_iic_drv_internal_status_t）

No	定数名	内容
STS0	R_IIC_STS_NO_INIT	未初期化状態
STS1	R_IIC_STS_IDLE	アイドル状態
STS2	R_IIC_STS_ST_COND_WAIT	スタートコンディション生成完了待ち状態
STS3	R_IIC_STS_SEND_SLVADR_W_WAIT	スレーブアドレス[Write]送信完了待ち状態
STS4	R_IIC_STS_SEND_SLVADR_R_WAIT	スレーブアドレス[Read]送信完了待ち状態
STS5	R_IIC_STS_SEND_DATA_WAIT	データ送信完了待ち状態
STS6	R_IIC_STS_RECEIVE_DATA_WAIT	データ受信完了待ち状態
STS7	R_IIC_STS_SP_COND_WAIT	ストップコンディション生成完了待ち状態

6.6.2 制御時のイベント

プロトコル制御時に発生するイベントを以下のように定義します。

割り込みだけでなく、本サンプルコードが提供するインタフェースがコールされた際も、イベントとして定義します。

表 6-3 プロトコル制御のためのイベント一覧（enum r_iic_drv_internal_event_t）

No	イベント	イベントの定義
EV0	R_IIC_EV_INIT	r_iic_drv_init_driver()コール
EV1	R_IIC_EV_GEN_START_COND	r_iic_drv_generate_start_cond()コール
EV2	R_IIC_EV_INT_START	ICEEI 割り込み発生（割り込みフラグ：START）
EV3	R_IIC_EV_INT_ADD	ICTEI 割り込み発生
EV4	R_IIC_EV_INT_SEND	ICTEI 割り込み発生
EV5	R_IIC_EV_INT_RECEIVE	ICRXI 割り込み発生
EV6	R_IIC_EV_INT_STOP	ICEEI 割り込み発生（割り込みフラグ：STOP）
EV7	R_IIC_EV_INT_AL	ICEEI 割り込み発生（割り込みフラグ：AL）
EV8	R_IIC_EV_INT_NACK	ICEEI 割り込み発生（割り込みフラグ：NACK）

6.6.3 プロトコル状態遷移

本サンプルコードでは、提供インタフェースのコール、または、I²C 割り込み発生をトリガに状態が遷移します。各プロトコルの状態遷移を以下に示します。

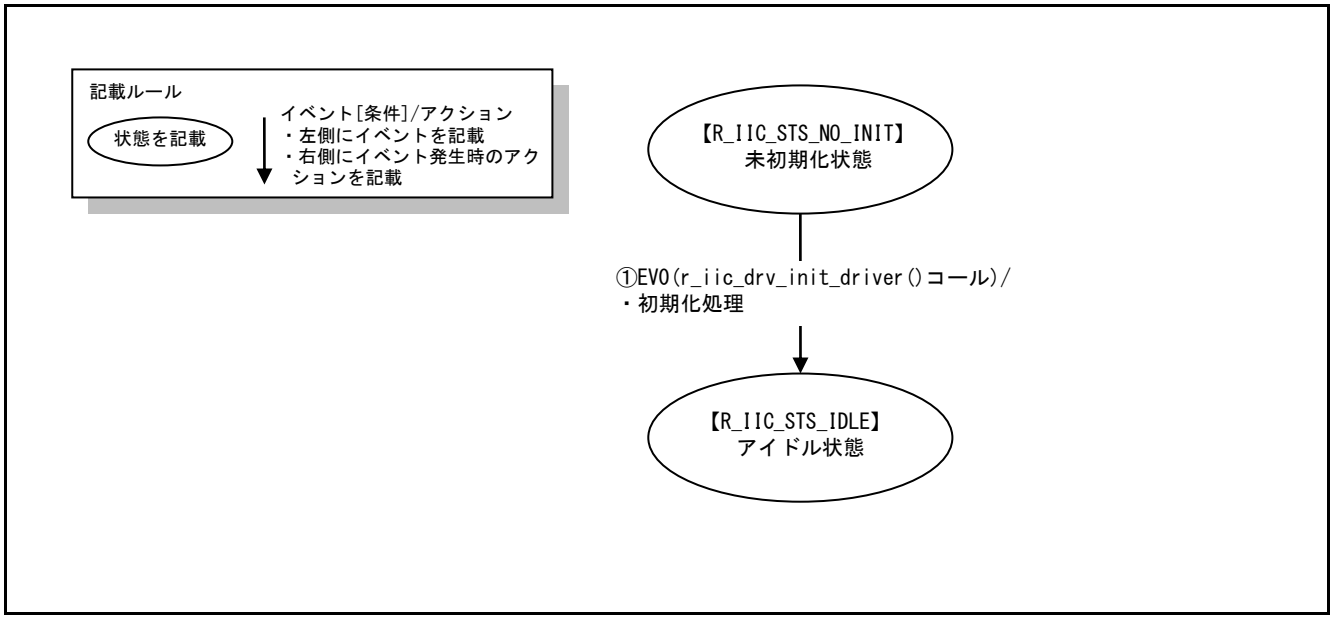


図 6-13 初期化 状態遷移図

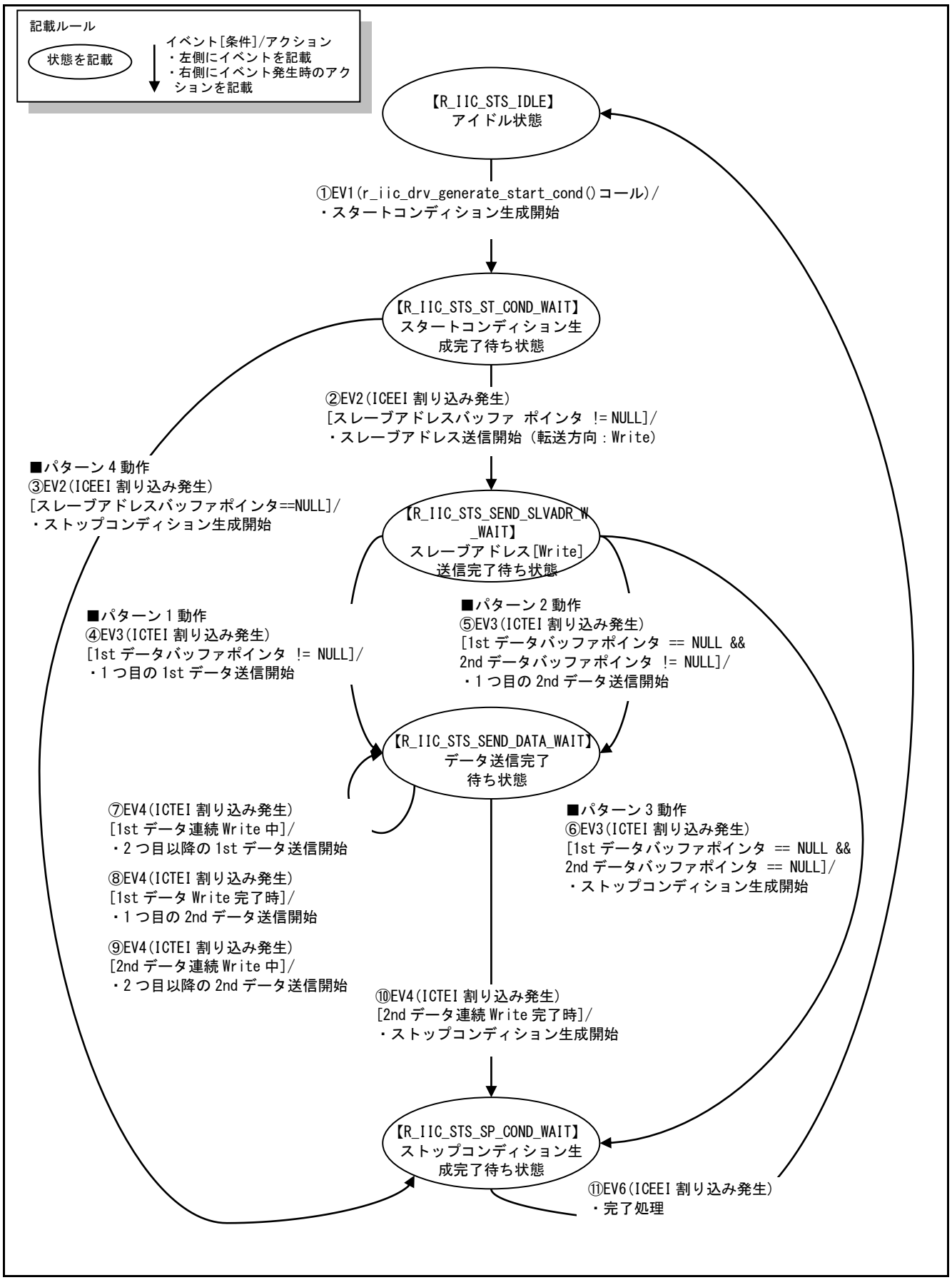


図 6-14　マスタ送信 状態遷移図

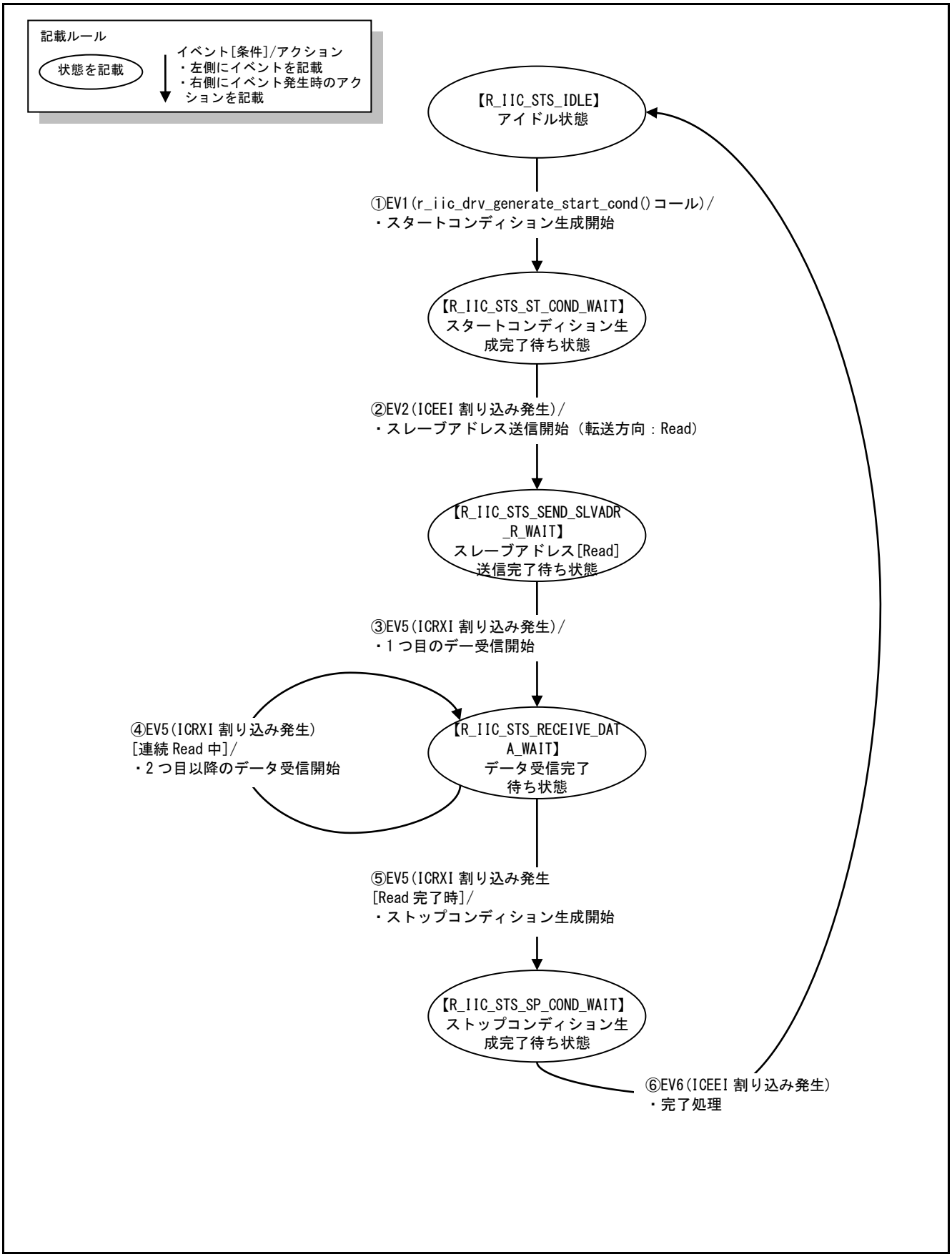


図 6-15　マスタ受信 状態遷移図

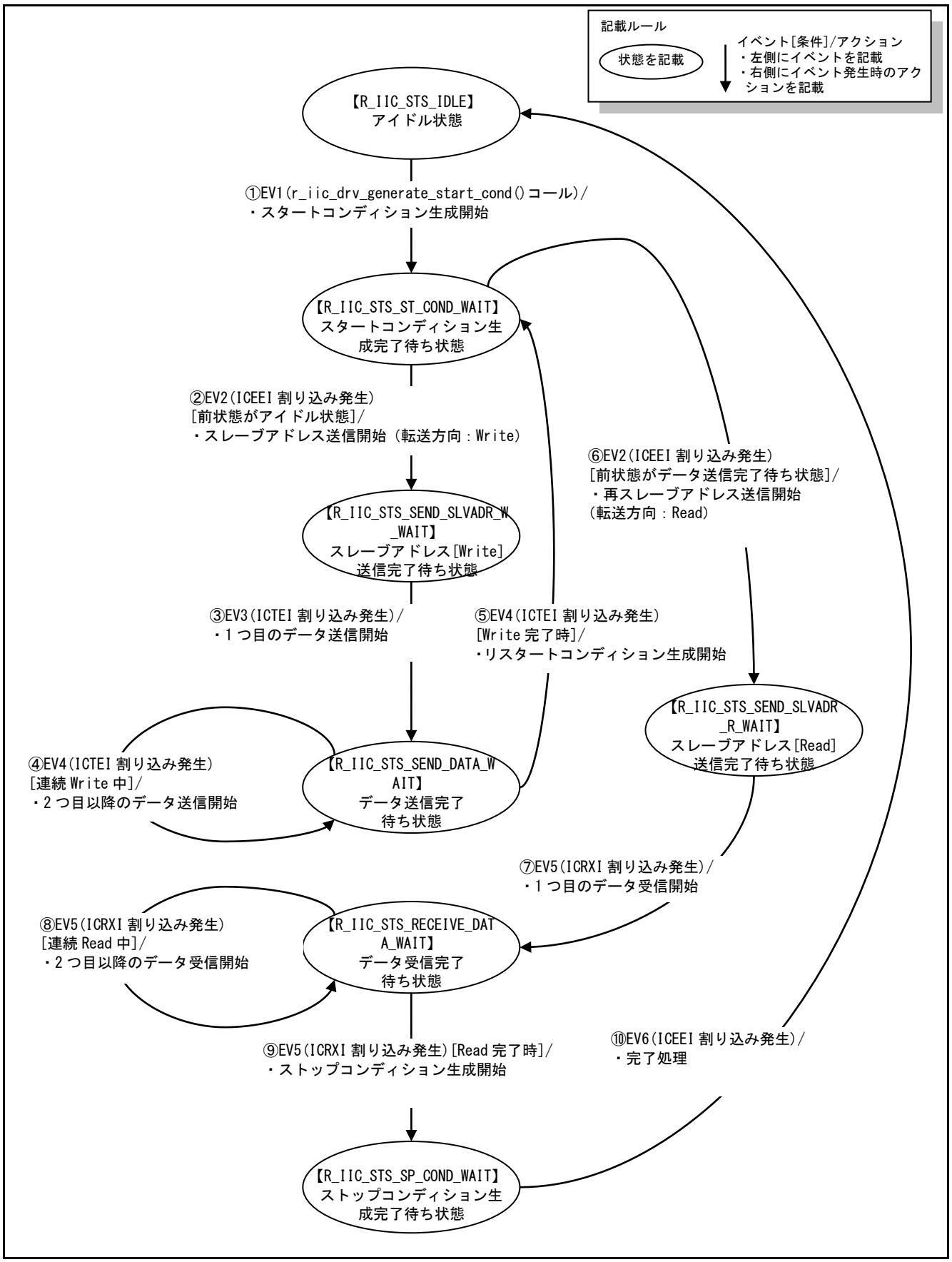


図 6-16　マスタ複合 状態遷移図

6.6.4 プロトコル状態遷移表

表 6-2の各状態で、表 6-3のイベントが発生した際に動作する処理を、以下の状態遷移表に定義します。

STS0 以降は表 6-2の No を参照ください。EV0 以降は表 6-3の No を参照ください。Func0 以降は表 6-5を参照してください。

表 6-4 プロトコル状態遷移表（gc_iic_mtx_tbl[][]）

状態＼イベント		EV0	EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8
STS0	未初期化状態 【R_IIC_STS_NO_INIT】	Func0	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS1	アイドル状態 【R_IIC_STS_IDLE】	ERR	Func1	ERR	ERR	ERR	ERR	ERR	ERR	ERR
STS2	スタートコンディション生成完了待ち状態 【R_IIC_STS_ST_COND_WAIT】	ERR	ERR	Func2	ERR	ERR	ERR	ERR	Func7	Func8
STS3	スレーブアドレス[Write]送信完了待ち状態 【R_IIC_STS_SEND_SLVADR_W_WAIT】	ERR	ERR	ERR	Func3	ERR	ERR	ERR	Func7	Func8
STS4	スレーブアドレス[Read]送信完了待ち状態 【R_IIC_STS_SEND_SLVADR_R_WAIT】	ERR	ERR	ERR	ERR	ERR	Func3	ERR	Func7	Func8
STS5	データ送信完了待ち状態 【R_IIC_STS_SEND_DATA_WAIT】	ERR	ERR	ERR	ERR	Func4	ERR	ERR	Func7	Func8
STS6	データ受信完了待ち状態 【R_IIC_STS_RECEIVE_DATA_WAIT】	ERR	ERR	ERR	ERR	ERR	Func5	ERR	Func7	Func8
STS7	ストップコンディション生成完了待ち状態 【R_IIC_STS_SP_COND_WAIT】	ERR	ERR	ERR	ERR	ERR	ERR	Func6	Func7	Func8

備考：

ERR は R_IIC_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。

6.6.5 プロトコル状態遷移登録関数

状態遷移表に登録されている関数を以下のように定義します。

表 6-5 プロトコル状態遷移登録関数一覧

処理	関数名	概要
Func0	r_iic_drv_init_driver()	初期設定処理
Func1	r_iic_drv_generate_start_cond()	スタートコンディション生成処理
Func2	r_iic_drv_arter_gen_start_cond()	スタートコンディション生成後処理
Func3	r_iic_drv_after_send_slvadr()	スレーブアドレス送信完了後処理
Func4	r_iic_drv_write_data_sending()	データ送信処理
Func5	r_iic_drv_read_data_receiving()	データ受信処理
Func6	r_iic_drv_release()	通信完了処理
Func7	r_iic_drv_arbitration_lost()	アービトレーションロストエラー処理
Func8	r_iic_drv_nack()	NACK エラー処理

6.6.6 プロトコル状態遷移時の処理

プロトコル状態遷移時の処理 r_iic_drv_func_table()（以降、通信を進める処理とする）について以下に説明します。

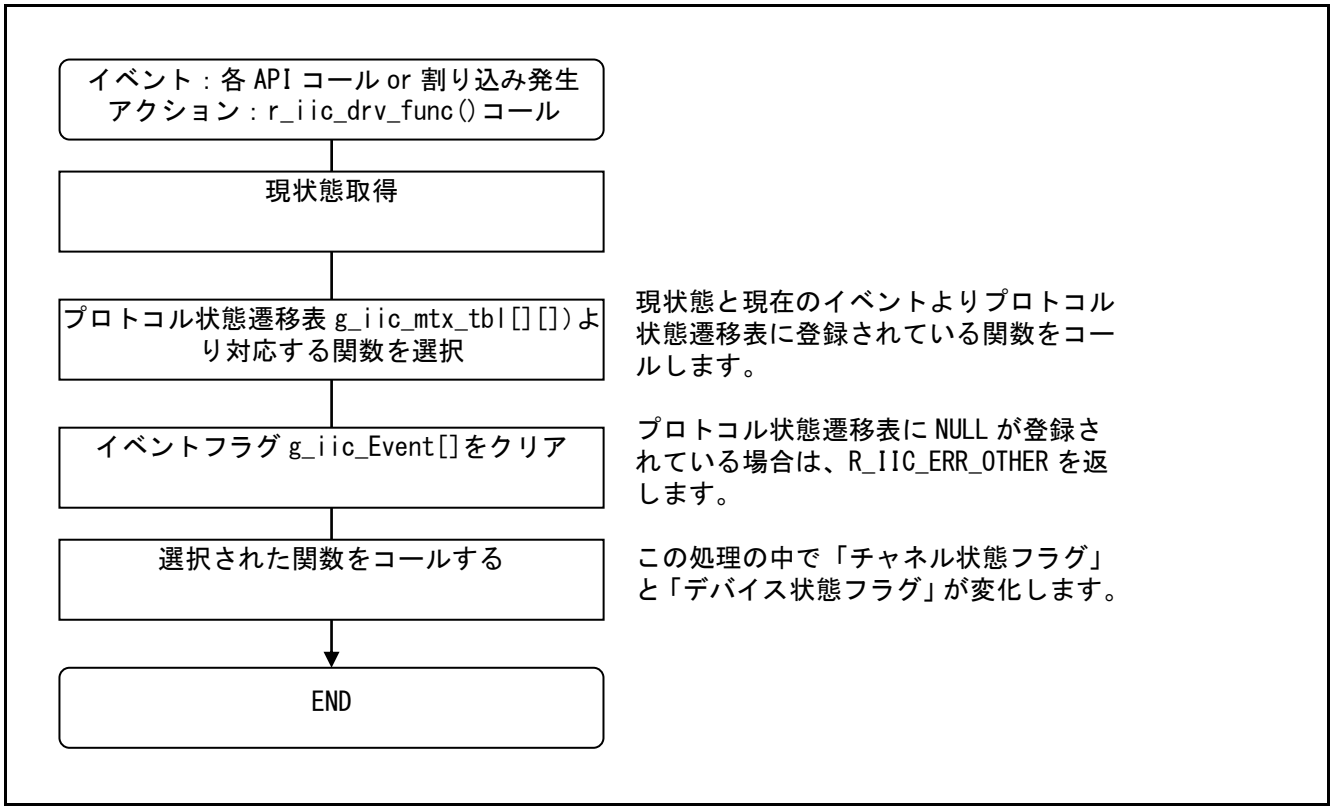


図 6-17 通信を進める処理がコールされる仕組み

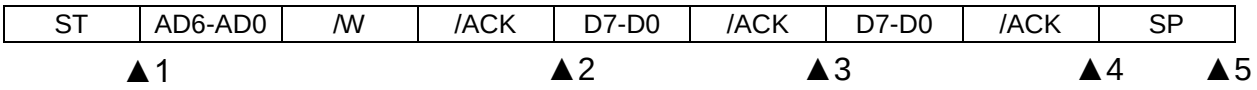
6.7 割り込み発生タイミング

以下に本ドライバ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

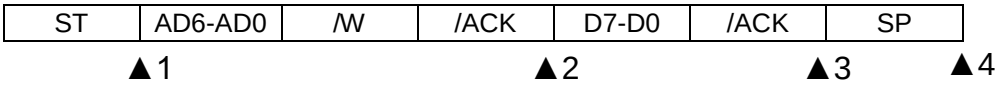
6.7.1 マスタ送信

(1) パターン 1



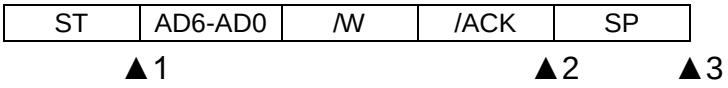
▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICTEI 割り込み・・・アドレス送信完了（転送方向ビット：Write） ※1
▲３：ICTEI 割り込み・・・データ送信完了（1st データ） ※1
▲４：ICTEI 割り込み・・・データ送信完了（2nd データ） ※1
▲５：ICEEI（STOP）割り込み・・・ストップコンディション検出

(2) パターン 2



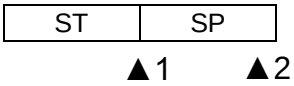
▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICTEI 割り込み・・・アドレス送信完了（転送方向ビット：Write） ※1
▲３：ICTEI 割り込み・・・データ送信完了（2nd データ） ※1
▲４：ICEEI（STOP）割り込み・・・ストップコンディション検出

(3) パターン 3



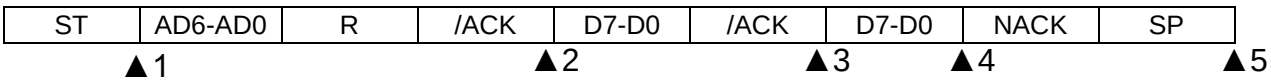
▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICTEI 割り込み・・・アドレス送信完了（転送方向ビット：Write） ※１
▲３：ICEEI（STOP）割り込み・・・ストップコンディション検出

(4) パターン 4



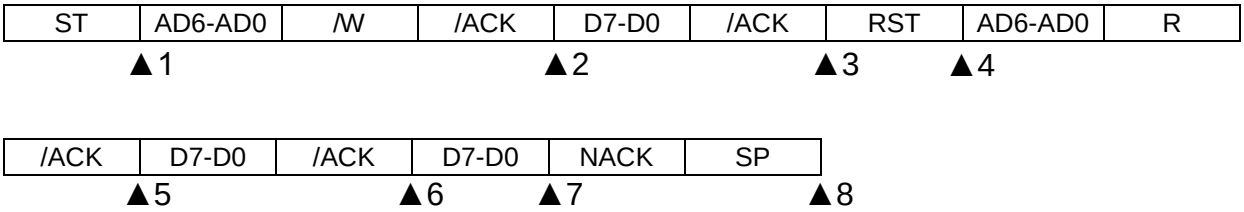
▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICEEI（STOP）割り込み・・・ストップコンディション検出

6.7.2 マスタ受信



▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICRXI 割り込み・・・アドレス送信完了（転送方向ビット：Read） ※１
▲３：ICRXI 割り込み・・・最終データ－１受信完了（2nd データ） ※１
▲４：ICRXI 割り込み・・・最終データ受信完了（2nd データ） ※２
▲５：ICEEI（STOP）割り込み・・・ストップコンディション検出

6.7.3 マスタ複合



▲１：ICEEI（START）割り込み・・・スタートコンディション検出
▲２：ICTEI 割り込み・・・アドレス送信完了（転送方向ビット：Write） ※１
▲３：ICTEI 割り込み・・・データ送信完了（1st データ） ※１
▲４：ICEEI（START）割り込み・・・リスタートコンディション検出
▲５：ICRXI 割り込み・・・アドレス送信完了（転送方向ビット：Read） ※１
▲６：ICRXI 割り込み・・・最終データー１受信完了（2nd データ） ※１
▲７：ICRXI 割り込み・・・最終データ受信完了（2nd データ） ※２
▲８：ICEEI（STOP）割り込み・・・ストップコンディション検出

※１：9 クロック目の立ち上がりで発生
※２：8 クロック目の立ち上がりで発生

6.8 コールバック関数

通信が正常に終了した場合、もしくはエラー終了した場合にコールされます。使用する場合は、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。構造体については、6.13.1 I2C 通信情報構造体を参照ください。

6.9 データバッファと送信／受信データの関係

本サンプルコードは、ブロック型デバイスドライバであり、送信／受信データポインタを引数として設定します。RAM 上のデータバッファのデータ並びと送信／受信順番の関係は、以下のとおりで、エンディアンや使用するシリアル通信機能に関係なく、送信データバッファの並びの順に送信し、また、受信の順に受信データバッファに書き込みます。

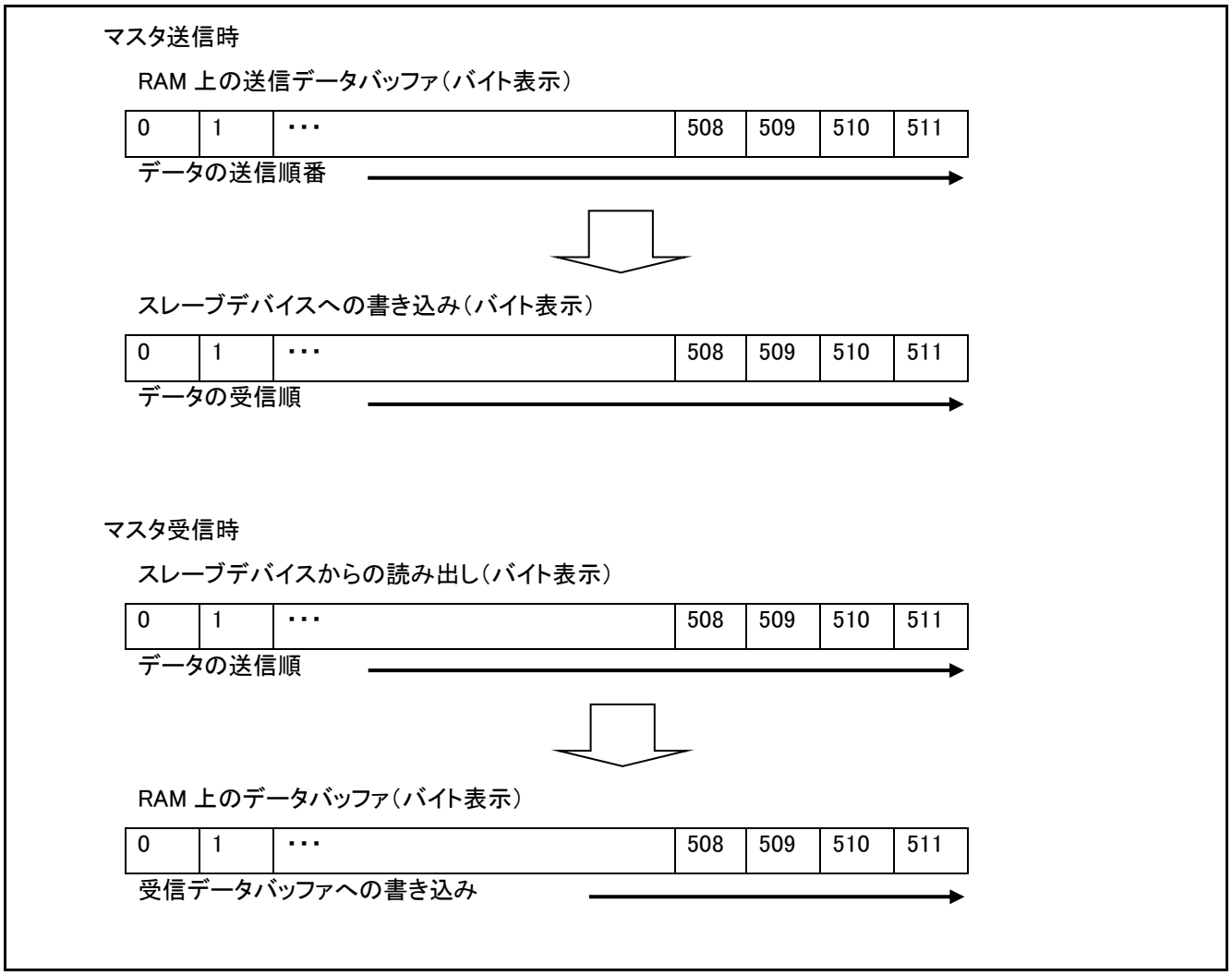


図 6-18 転送データの格納

6.10 必要メモリサイズ

以下に必要メモリサイズを示します。下記のメモリサイズは、1 チャネルを使用する場合の値です。使用するチャネル数により、メモリサイズは異なります。

(1) RX63N の場合

表6-6 必要メモリサイズ

使用メモリ	サイズ	備考
ROM	4,648 バイト （リトルエンディアン）	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	30 バイト （リトルエンディアン）	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_sfr.c r_iic_drv_sub.c
最大使用ユーザスタック	84 バイト	
最大使用割り込みスタック	4 バイト	

【注】 必要メモリサイズは 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。

(2) RX210 の場合

表6-7 必要メモリサイズ

使用メモリ	サイズ	備考
ROM	4,654 バイト （リトルエンディアン）	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_sfr.c r_iic_drv_sub.c
RAM	30 バイト （リトルエンディアン）	r_iic_drv_api.c r_iic_drv_int.c r_iic_drv_sfr.c r_iic_drv_sub.c
最大使用ユーザスタック	84 バイト	
最大使用割り込みスタック	4 バイト	

【注】 必要メモリサイズは 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。

6.11 ファイル構成

表 6-8にサンプルコードで使用するファイルを示します。なお、統合開発環境で自動生成されるファイルは除きます。

表6-8 ファイル構成

¥an_r01an1254jj0103_rx_iic <DIR>			サンプルコードのフォルダ
	r01an1254jj0103_rx.pdf		アプリケーションノート
	¥source <DIR>		プログラム格納用フォルダ
		¥r_iic_drv_rx <DIR>	I²C シングルマスタ制御ソフトウェアのフォルダ
		r_iic_drv_api.c	API ソースファイル
		r_iic_drv_api.h	API ヘッダファイル
		r_iic_drv_int.c	割り込みハンドラソースファイル
		r_iic_drv_int.h	割り込みハンドラヘッダファイル
		r_iic_drv_sfr.h.rx21a	レジスタ用共通定義ヘッダファイル（RX21A 用）
		r_iic_drv_sfr.h.rx62n	レジスタ用共通定義ヘッダファイル（RX62N 用）
		r_iic_drv_sfr.h.rx63n	レジスタ用共通定義ヘッダファイル（RX63N 用）
		r_iic_drv_sfr.h.rx63t	レジスタ用共通定義ヘッダファイル（RX63T 用）
		r_iic_drv_sfr.h.rx210	レジスタ用共通定義ヘッダファイル（RX210 用）
		r_iic_drv_sfr_rx21a.c	レジスタ用共通定義ソースファイル（RX21A 用）
		r_iic_drv_sfr_rx62n.c	レジスタ用共通定義ソースファイル（RX62N 用）
		r_iic_drv_sfr_rx63n.c	レジスタ用共通定義ソースファイル（RX63N 用）
		r_iic_drv_sfr_rx63t.c	レジスタ用共通定義ソースファイル（RX63T 用）
		r_iic_drv_sfr_rx210.c	レジスタ用共通定義ソースファイル（RX210 用）
		r_iic_drv_sub.c	内部関数ソースファイル
		r_iic_drv_sub.h	内部関数ヘッダファイル
		¥sample <DIR>	EEPROM 動作検証用プログラム格納用フォルダ
		sample_background.c	EEPROM 動作検証用のサンプルプログラム RIIC 割り込みハンドラでアドバンス関数をコールする場合
		sample_foreground.c	EEPROM 動作検証用のサンプルプログラム メイン処理でアドバンス関数をコールする場合

注）r_iic_drv_sfr.h.XXX は、MCU 毎に作成したものです。どれか一つを r_iic_drv_sfr.h にリネームして使用してください。対象 MCU のものが無い場合には、参照して、r_iic_drv_sfr.h を作成してください。

6.12 定数一覧

6.12.1 リターン値

以下に、サンプルコードで使用するリターン値／チャネル状態フラグ／デバイス状態フラグの管理値を示します。

表 6-9 リターン値／チャネル状態フラグ／デバイス状態フラグの管理値（r_iic_drv_api.h に定義）

定数名	設定値	内容
R_IIC_NO_INIT	(error_t)(0)	未初期化状態
R_IIC_IDLE	(error_t)(1)	アイドル状態：通信可能
R_IIC_FINISH	(error_t)(2)	アイドル状態：前通信終了、通信可能
R_IIC_NACK	(error_t)(3)	アイドル状態：前通信 NACK 終了、通信可能
R_IIC_COMMUNICATION	(error_t)(4)	通信中
R_IIC_LOCK_FUNC	(error_t)(5)	API 処理中 以下の場合に発生します。 ・API 処理中に他の API をコールした場合
R_IIC_BUS_BUSY	(error_t)(6)	バスビジー 以下の場合に発生します。 ・通信中に初期化関数または各開始関数をコールした場合 ・同一チャネル上の他のデバイスが通信中に、各開始関数またはアドバンス関数をコールした場合
R_IIC_ERR_PARAM	(error_t)(-1)	パラメータエラー
R_IIC_ERR_AL	(error_t)(-2)	アービトレーションロストエラー
R_IIC_ERR_NON_REPLY	(error_t)(-3)	未応答エラー
R_IIC_ERR_SDA_LOW_HOLD	(error_t)(-4)	SCL 疑似クロック生成関数コール時、SDA Low ホールドエラー
R_IIC_ERR_OTHER	(error_t)(-5)	その他エラー

6.12.2 各種定義

以下にサンプルコードで使用する各種定義値をファイル毎に示します。

表 6-10 r_iic_drv_api.h の各種定義値

定数名	設定値	内容
MAX_IIC_CH_NUM	(uint8_t)(1)	同時に使用する最大チャネル番号+1 本サンプルコードでは 1 を設定しています。
REPLY_CNT	(uinit32_t)(10000)	アドバンス関数カウンタ（※１）
STOP_COND_WAIT	(uint16_t)(1000)	ストップコンディション生成待ちカウンタ（※１）
BUSCHK_CNT	(uint16_t)(1000)	バスビジーチェックカウンタ（※１）
SDACHK_CNT	(uint16_t)(1000)	SDA レベルチェックカウンタ（※１）
GEN_SCLCLK_WAIT	(uint16_t)(1000)	疑似クロック生成待ちカウンタ（※１）
W_CODE	(uint8_t)(0x00)	スレーブアドレスの転送方向が Write の場合の設定値
R_CODE	(uint8_t)(0x01)	スレーブアドレスの転送方向が Read の場合の設定値
R_IIC_HI	(uint8_t)(0x01)	Port “H”
R_IIC_LOW	(uint8_t)(0x00)	Port “L”
R_IIC_OUT	(uint8_t)(0x01)	Port Output
R_IIC_IN	(uint8_t)(0x00)	Port Input
R_IIC_FALSE	(uint8_t)(0x00)	Flag "OFF"
R_IIC_TRUE	(uint8_t)(0x01)	Flag "ON"

表 6-11 r_iic_drv_sfr.h.rxXXX の各種定義値（XXX は MCU 名）

定数名	設定値	内容
R_IIC_CHx_LCLK	(uint8_t)(0xED)	チャネル x（x はチャネル番号）用 I²C バスビットレートローレベルレジスタ（ICBRL）設定値（※２）
R_IIC_CHx_HCLK	(uint8_t)(0xE6)	チャネル x（x はチャネル番号）用 I²C バスビットレートハイレベルレジスタ（ICBRL）設定値（※２）
R_IIC_CHx_ICMR1_INIT	(uint8_t)(0x28)	チャネル x（x はチャネル番号）用 I²C バスモードレジスタ１（ICMR1）設定値（※２）
R_IIC_IPR_CHx_EEI_INIT	(uint8_t)(0x02)	チャネル x（x はチャネル番号）用 ICEEIx 割り込みプライオリティ
R_IIC_IPR_CHx_RXI_INIT	(uint8_t)(0x02)	チャネル x（x はチャネル番号）用 ICRXIx 割り込みプライオリティ
R_IIC_IPR_CHx_TXI_INIT	(uint8_t)(0x02)	チャネル x（x はチャネル番号）用 ICTXIx 割り込みプライオリティ
R_IIC_IPR_CHx_TEI_INIT	(uint8_t)(0x02)	チャネル x（x はチャネル番号）用 ICTEIx 割り込みプライオリティ

※１：カウンタ値の設定について

ソフトウェアループ用のカウンタです。したがって使用するシステム・クロックによりループ時間が変化します。使用するシステム・クロックに合わせて設定してください。

※２：転送速度設定

転送クロックは、以下の設定により決定します。使用するチャネル毎に設定する必要があります。それぞれ定義してください。詳しい設定方法については、RX ファミリユーザーズマニュアル　ハードウェア編をご参照ください。

・I²C バスモードレジスタ１（ICMR1）の内部基準クロック選択ビット（CKS[2:0]）

・I²C バスビットレートローレベルレジスタ（ICBRL）

・I²C バスビットレートハイレベルレジスタ（ICBRH）

最大 400KHz までの設定可能です。ただし、標準モード・デバイスは、ファーストモード・デバイスが混載されるバスでは、標準モードの最大 100KHz に設定する必要があります。また、SDA，SCL 信号の立ち上がり時間（tR）と立ち下がり時間（tF）は，プルアップ抵抗と配線容量によって異なるため，必要に応じて設定値を見直してください。

6.13 構造体／共用体一覧

6.13.1 I²C 通信情報構造体

サンプルコードで使用する I²C 通信情報構造体を以下に示します。本構造体は、使用するスレーブデバイス毎に設定する必要があります。

```
typedef struct
{
  uint8_t          *pSlvAdr;          /* Pointer for Slave address buffer       */
  uint8_t          *pData1st;         /* Pointer for 1st Data buffer            */
  uint8_t          *pData2nd;         /* Pointer for 2nd Data buffer            */
  error_t          *pDevStatus;       /* Device status flag                     */
  uint32_t         Cnt1st;            /* 1st Data counter                       */
  uint32_t         Cnt2nd;            /* 2nd Data counter                       */
  r_iic_callback   CallBackFunc;      /* Callback function                      */
  uint8_t          ChNo;              /* Channel No.                            */
  uint8_t          rsv1;
  uint8_t          rsv2;
  uint8_t          rsv3;
} r_iic_drv_info_t;
```

図 6-19 I²C 通信情報構造体

(1) 構造体メンバ説明

構造体の説明を以下に示します。“r_iic_drv_info_t”メンバの設定方法は、表 6-13および表 6-14を参照してください。

表 6-12 構造体“r_iic_drv_info_t”のメンバー覧

構造体メンバ	内容
*pSlvAdr	スレーブアドレス格納バッファポインタ 1 バイト確保してください。
*pData1st	1st データ格納バッファポインタ
*pData2nd	2nd データ格納バッファポインタ
*pDevStatus	デバイス状態フラグポインタ 同一チャネル上に複数デバイスを接続している時に、通信中のデバイスを確認することができます。1 バイト確保してください。 使用例については、8.6項を参照ください。
Cnt1st	1st データカウンタ（バイト数）
Cnt2nd	2nd データカウンタ（バイト数）
CallBackFunc	コールバック関数
ChNo	使用デバイスのチャネル番号 使用するバスのチャネル番号を設定してください。
rsv1, rsv2, rsv3	アライメント調整用

(2) 設定方法

表 6-13にマスタ送信時、表 6-14にマスタ受信及びマスタ複合時の構造体“r_iic_drv_info_t”メンバのユーザ設定可能範囲を示します。

表 6-13 マスタ送信時、構造体“r_iic_drv_info_t”メンバのユーザ設定可能範囲

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
*pSlvAdr	スレーブアドレス格納元アドレス	スレーブアドレス格納元アドレス	スレーブアドレス格納元アドレス	NULL
*pData1st	1st データ格納元アドレス	NULL	NULL	NULL
*pData2nd	2nd データ（送信データ）格納元アドレス	2nd データ（送信データ）格納元アドレス	NULL	NULL
*pDevStatus	デバイス状態格納元アドレス	デバイス状態格納元アドレス	デバイス状態格納元アドレス	デバイス状態格納元アドレス
Cnt1st	0000 0001h（※）～FFFF FFFFh	（設定無効）	（設定無効）	（設定無効）
Cnt2nd	0000 0001h（※）～FFFF FFFFh	0000 0001h（※）～FFFF FFFFh	（設定無効）	（設定無効）
CallBackFunc	使用する場合は関数名を指定、使用しない場合は NULL。	使用する場合は関数名を指定、使用しない場合は NULL。	使用する場合は関数名を指定、使用しない場合は NULL。	使用する場合は関数名を指定、使用しない場合は NULL。
ChNo	00h～FFh	00h～FFh	00h～FFh	00h～FFh
rsv1,rsv2,rsv3	（設定無効）	（設定無効）	（設定無効）	（設定無効）

※：“0”は設定禁止です。

表 6-14　マスタ受信及びマスタ複合時、構造体“r_iic_drv_info_t”メンバのユーザ設定可能範囲

構造体メンバ	ユーザ設定可能範囲	
	マスタ受信	マスタ複合
*pSlvAdr	スレーブアドレス格納元アドレス	スレーブアドレス格納元アドレス
*pData1st	（設定無効）	1st データ格納元アドレス
*pData2nd	2nd（受信データ）格納先アドレス	2nd データ（受信データ）格納先アドレス
*pDevStatus	デバイス状態格納元アドレス	デバイス状態格納元アドレス
Cnt1st	（設定無効）	0000 0001h（※）～FFFF FFFFh
Cnt2nd	0000 0001h（※）～FFFF FFFFh	0000 0001h（※）～FFFF FFFFh
CallBackFunc	使用する場合は関数名を指定、使用しない場合は NULL。	使用する場合は関数名を指定、使用しない場合は NULL。
ChNo	00h～FFh	00h～FFh
rsv1,rsv2,rsv3	（設定無効）	（設定無効）

※：“0”は設定禁止です。

(3)　コールバック関数

通信が正常に終了した場合、もしくはエラー終了した場合にコールされます。使用する場合は、CallBackFunc に関数名を指定してください。

(4)　設定に関する注意事項

マスタ送信では本構造体で設定されたデータを参照し、どの動作を行うか決定します。表 6-13以外の設定を行うと、正常に動作できなくなりますので、設定の際には十分に注意してください。

6.13.2 内部情報管理構造体

サンプルコードで使用する内部情報管理構造体を以下に示します。本構造体はサンプルコードで制御するため、設定は不要です。

```
typedef struct
{
  r_iic_drv_internal_mode_t          Mode;       /* Mode of Control Protocol    */
  r_iic_drv_internal_status_t        N_status;   /* Internal Status of NOW      */
  r_iic_drv_internal_status_t        B_status;   /* Internal Status of BEFORE   */
} r_iic_drv_internal_info_t;
```

図 6-20 内部情報管理構造体

(1) 構造体メンバ説明

構造体の説明を以下に示します。

表 6-15 構造体“r_iic_drv_internal_info_t”のメンバー覧

構造体メンバ	内容
Mode	I²C プロトコル動作モード 格納されるデータの定義については、表 6-16を参照ください。
N_status	プロトコル制御のための現状態です。表 6-2に定義されている値が格納されます。
B_status	プロトコル制御のための前状態です。表 6-2に定義されている値が格納されます。

6.14 列挙型

本サンプルコードで使用する列挙型の定義を以下に示します。

表 6-16 I²C プロトコル動作モード一覧（enum r_iic_drv_internal_mode_t）

定義名	内容
R_IIC_MODE_NONE	未通信状態 未初期化状態またはアイドル状態のとき、このモードに遷移します。
R_IIC_MODE_WRITE	マスタ送信動作中 マスタ送信開始関数 R_IIC_Drv_MasterTx()により通信を開始すると、このモードに遷移します。
R_IIC_MODE_READ	マスタ受信動作中 マスタ受信開始関数 R_IIC_Drv_MasterRx()により通信を開始すると、このモードに遷移します。
R_IIC_MODE_COMBINED	マスタ複合動作中 マスタ複合開始関数 R_IIC_Drv_MasterTRx()により通信を開始すると、このモードに遷移します。

6.15 変数一覧

表 6-17にグローバル変数を示します。

表 6-17 グローバル変数

型	変数名	内容	使用関数
uint8_t	g_iic_ChStatus[MAX_IIC_CH_NUM]	チャネル状態フラグ 表 6-9に定義する通信状態を確認することができます。 初期化する場合、“R_IIC_NO_INIT”を設定してください。	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance R_IIC_Drv_GenClk R_IIC_Drv_Reset
r_iic_drv_internal_event_t	g_iic_Event[MAX_IIC_CH_NUM]	イベントフラグ 割り込みが発生した際にセットされ、アドバンス関数でクリアされます。クリアされると R_IIC_EV_INIT になります。設定値は表 6-3、セットとクリアの関係については、図 6-9を参照ください。	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
r_iic_drv_internal_info_t	g_iic_InternalInfo[MAX_IIC_CH_NUM]	内部情報管理 本サンプルコードで管理するため、設定は禁止です。	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
uint32_t	g_iic_ReplyCnt[MAX_IIC_CH_NUM]	アドバンス関数カウンタ アドバンス関数のコール回数の上限値です。ユーザがコールするアドバンス関数で減算されます。イベントが発生すると初期化されます。“0”になるとチャネル状態フラグとデバイス状態フラグに”R_IIC_ERR_NON_REPLY”がセットされます。 カウンタ値はマクロ定義“REPLY_CNT”で変更可能です。システムに合わせて設定してください。	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance
int32_t	g_iic_Api[MAX_IIC_CH_NUM]	API フラグ 本サンプルコードの API の多重コールを防止するために使用します。 API 処理開始時にセットされ、終了後にクリアされます。	R_IIC_Drv_Init R_IIC_Drv_MasterTx R_IIC_Drv_MasterRx R_IIC_Drv_MasterTRx R_IIC_Drv_Advance R_IIC_Drv_GenClk R_IIC_Drv_Reset

6.16 関数一覧

表 6-18に関数一覧を示します。

表 6-18 関数一覧

関数名	説明
R_IIC_Drv_Init()	I²C ドライバ初期化関数
R_IIC_Drv_MasterTx()	マスタ送信開始関数
R_IIC_Drv_MasterRx()	マスタ受信開始関数
R_IIC_Drv_MasterTRx()	マスタ複合開始関数
R_IIC_Drv_Advance()	アドバンス関数
R_IIC_Drv_GenClk()	SCL 疑似クロック生成関数
R_IIC_Drv_Reset()	I²C ドライバリセット関数

6.17 状態遷移図

図 6-15 に、各チャネルの状態遷移図を示します。

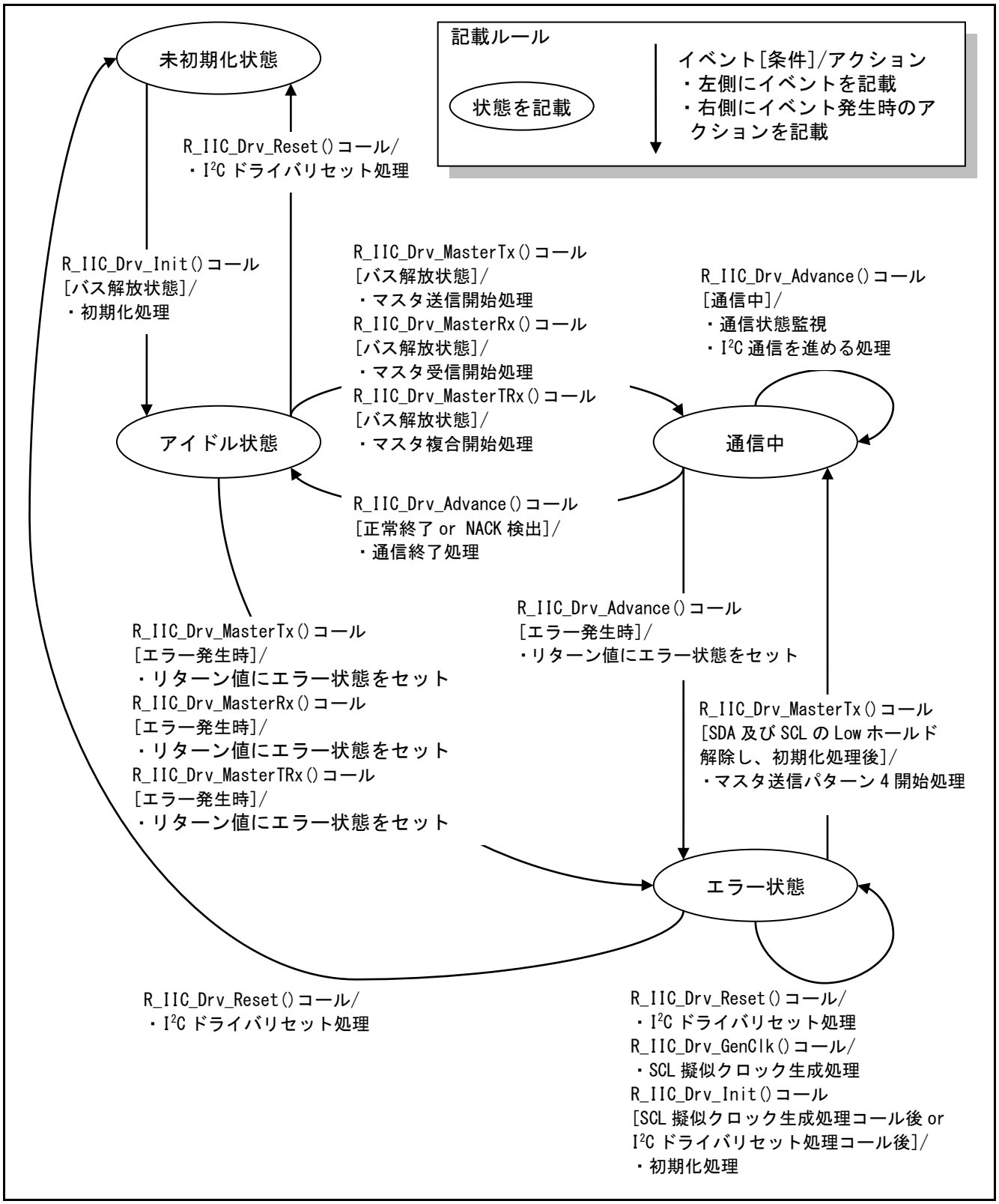


図 6-15 状態遷移図

6.17.1 エラー状態の定義

本サンプルコードでは、以降に示す現象が発生した場合をエラー状態と定義します。エラーの発生は、各 API コール後のリターン値により確認することができます。エラー時の対応方法については、6.18 関数仕様の各 API 項のリターン値を参照ください。

(1) パラメータエラー

リターン値：R_IIC_ERR_PARAM

各 API をコールする際、引数を設定していない場合。

(2) アービトレーションロスト

リターン値：R_IIC_ERR_AL

アービトレーションロストが発生した場合。発生条件は RX ファミリユーザーズマニュアル　ハードウェア編を参照ください。

(3) 未応答エラー

リターン値：R_IIC_ERR_NON_REPLY

以下の場合を未応答エラーとします。

- アドバンス関数のコール回数が上限を超えた場合。
- 各開始関数コール時、一定時間バスを監視したが、バスが解放されなかった場合。
- スタートコンディション生成処理を行ったが、一定時間が経過しても検出できなかった場合。
- アドバンス関数コール時、ストップコンディション生成処理を行ったが、一定時間が経過しても検出できなかった場合。

尚、バス監視とスタートコンディション生成の待ち時間はソフトウェアループで行います。カウンタ値はユーザで設定可能です。使用するシステム・クロックに合わせて設定してください。カウンタの定義については、表 6-10を参照ください。

(4) SDA Low ホールド（復帰不可）

リターン値：R_IIC_ERR_SDA_LOW_HOLD

SCL 疑似クロック生成を行ったが、SDA が Low ホールドのままの場合。

(5) その他エラー

リターン値：R_IIC_ERR_OTHER

上記（1）～（4）以外のエラーとします。

6.17.2 状態遷移時の各フラグの状態

状態遷移時の各フラグの状態を表 6-19に示します。

表 6-19 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_iic_ChStatus[]	I²C 通信情報構造体 *pDevStatus	内部通信情報構造体 Mode	内部通信情報構造体 N_status
未初期化状態	R_IIC_NO_INIT	R_IIC_NO_INIT	R_IIC_MODE_NONE	R_IIC_STS_NO_INIT
アイドル状態	R_IIC_IDLE R_IIC_FINISH R_IIC_NACK	R_IIC_IDLE R_IIC_FINISH R_IIC_NACK	R_IIC_MODE_NONE	R_IIC_STS_IDLE
通信中（マスタ送信）	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_WRITE	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
通信中（マスタ受信）	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_READ	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
通信中（マスタ複合）	R_IIC_COMMUNICATION	R_IIC_COMMUNICATION	R_IIC_MODE_COMBINED	R_IIC_STS_ST_COND_WAIT
				R_IIC_STS_SEND_SLVADR_W_WAIT
				R_IIC_STS_SEND_SLVADR_R_WAIT
				R_IIC_STS_SEND_DATA_WAIT
				R_IIC_STS_RECEIVE_DATA_WAIT
				R_IIC_STS_SP_COND_WAIT
エラー状態	R_IIC_ERR_PARAM	R_IIC_ERR_PARAM	-	-
	R_IIC_ERR_AL	R_IIC_ERR_AL	-	-
	R_IIC_ERR_NON_REPLY	R_IIC_ERR_NON_REPLY	-	-
	R_IIC_ERR_SCL_GENCLK	R_IIC_ERR_SCL_GENCLK	-	-
	R_IIC_ERR_OTHER	R_IIC_ERR_OTHER	-	-

6.18 関数仕様

6.18.1 関数共通処理

本サンプルコードでは、1 度に動作可能な API は 1 つであり、API 処理実行中に本サンプルコードの API がコールされた場合、処理を行わずに終了します。“R_IIC_LOCK_FUNC”がリターン値として返ります。

API の同時コールを防止するために API フラグを用意しています。このフラグは、API 処理が行われている間セットされます。各 API 処理の最初に必ずチェックされ、フラグがセットされていなければ、その処理を実行する仕組みです。図 6-21に概略フローを示します。

この処理は、6.16項で定義している関数に対して実施します。以降、6.18.2項以降では図 6-21に示す「ユーザ API 処理」の処理内容について記載します。

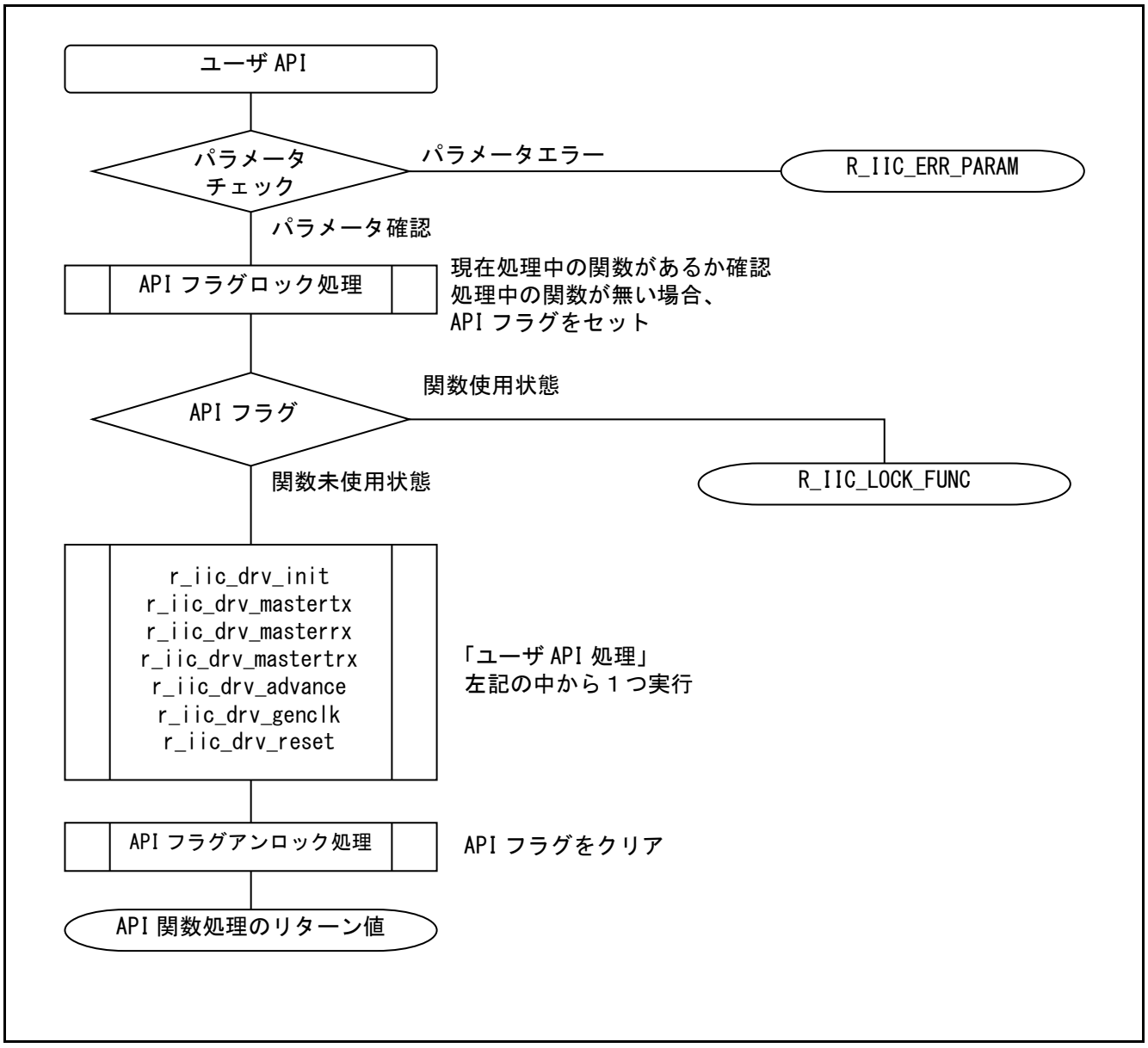


図 6-21 多重コール防止処理概略フロー

6.18.2 I²C ドライバ初期化関数

R_IIC_Drv_Init	
概要	I²C ドライバ初期化関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_Init(r_iic_drv_info_t *pRIic_Info)
説明	・対象チャネルの初期設定を行います。 ・本処理を行うためには、以下の設定が必要です。 構造体“r_iic_drv_info_t”メンバの ChNo ;使用チャネル番号 チャネル状態フラグ（g_iic_ChStatus[]） ;“R_IIC_NO_INIT”をセット（※１） デバイス状態フラグ（*(pRIic_Info.pDevStatus)）;“R_IIC_NO_INIT”をセット（※１）
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_IDLE 【チャネルが未初期化状態】の場合、初期化を行いアイドル状態に遷移しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_IDLE”を設定しました。 【すでに初期化済み】の場合、初期化は行わず、デバイス状態フラグに“R_IIC_IDLE”を設定しました。 →開始関数をコールすることで通信が可能です。 R_IIC_FINISH / R_IIC_NACK 前処理の実行結果です。開始関数が実行可能であるため、初期化は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。 →開始関数をコールすることで通信が可能です。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_BUS_BUSY 通信中です。初期化できませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_AL アービトレーションロストが発生しています。チャネル状態フラグとデバイス状態フラグは変更されません。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_NON_REPLY 未応答エラーが発生しています。チャネル状態フラグとデバイス状態フラグは変更されません。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_SDA_LOW_HOLD SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグと

デバイス状態フラグは変更されません。
→スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。

R_IIC_ERR_OTHER
その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。
すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。
→以下を確認してください。
・I²C 通信情報構造体の設定が正しく行われているか確認してください。

備考　※１：初期化関数をコールする場合、その前に“R_IIC_NO_INIT”をセットしてください。設定しないまま初期化関数をコールすると、初期化処理を行わない場合があります。

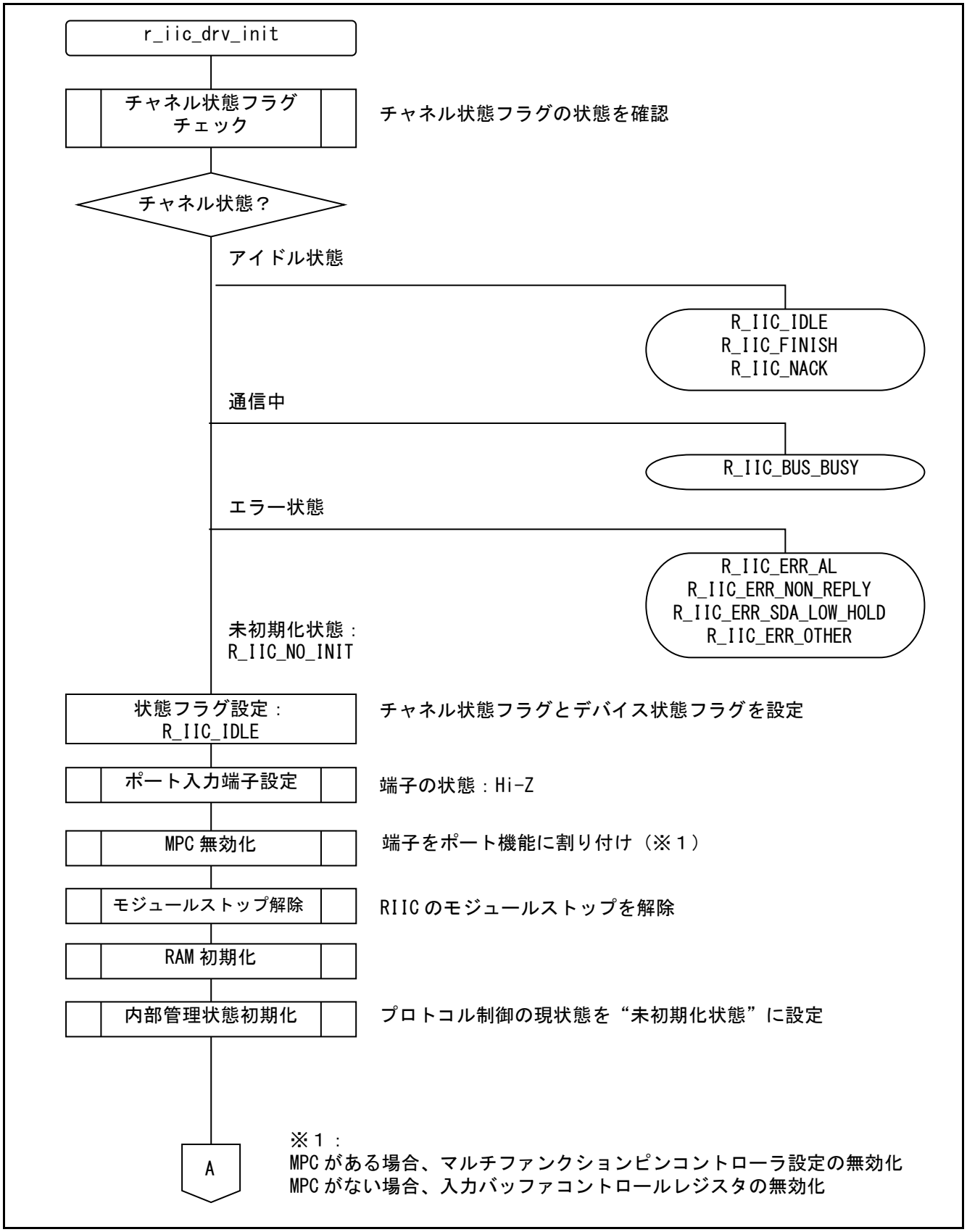


図 6-22　I²C ドライバ初期化関数概略フロー（1/2）

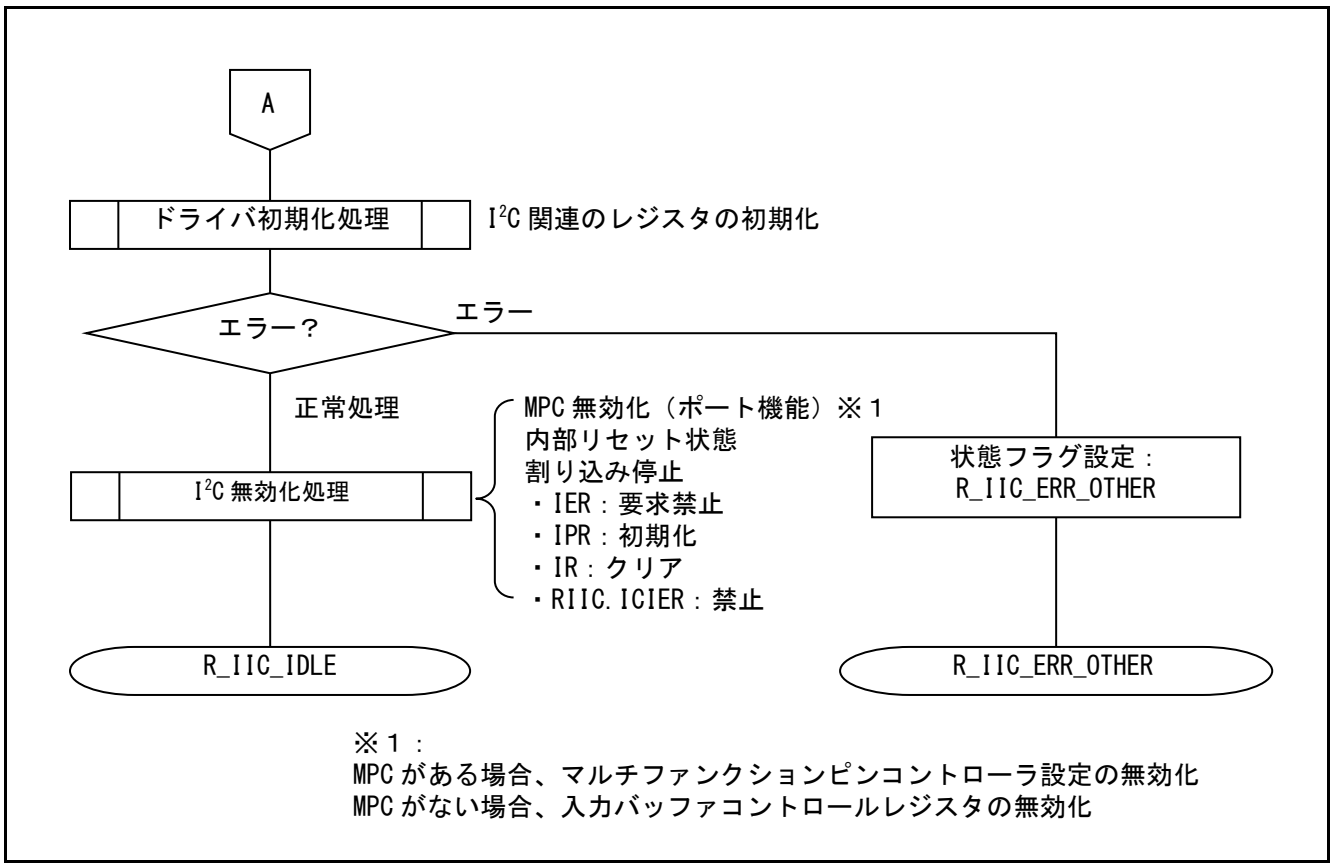


図 6-23　I²C ドライバ初期化関数概略フロー（2/2）

6.18.3 マスタ送信開始関数

R_IIC_Drv_MasterTx	
概要	マスタ送信開始関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_MasterTx (r_iic_drv_info_t *pRIic_Info)
説明	・マスタ送信を開始します。 ・本処理を行うためには、I²C 通信情報構造体“r_iic_drv_info_t”の設定が必要です。設定方法は表 6-13を参照してください。
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_COMMUNICATION マスタ送信を開始しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_COMMUNICATION”を設定しました。 →アドバンス関数をコールして通信を完了させてください。 R_IIC_NO_INIT 初期化されていません（※１）。チャネル状態フラグとデバイス状態フラグは変更されません。 →初期化関数をコールして初期化を終了させてください。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_BUS_BUSY 通信中です。マスタ送信を開始できませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_AL アービトレーションロストが発生しています。チャネル状態フラグとデバイス状態フラグは変更されません。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_NON_REPLY バスが開放されていないか、スタートコンディションを検出できませんでした。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_NON_REPLY”を設定しました。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_SDA_LOW_HOLD SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグとデバイス状態フラグは変更されません。 →スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。 R_IIC_ERR_OTHER その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。
→以下を確認してください。
・I²C 通信情報構造体の設定が正しく行われているか確認してください。

備考
・本関数から戻った時点では、I²C 通信は終了していません。I²C 通信を終了させるためには、アドバンス関数をコールする必要があります。
・開始関数コール後の通信状態は、アドバンス関数のリターン値で確認できます。

※１：一度初期化した場合でも、同一チャネル上の他デバイスが I²C ドライバリセット関数をコールしたことにより、未初期化状態になることがあります。この場合、再度 I²C ドライバ初期化関数をコールしてから、開始関数をコールしてください。

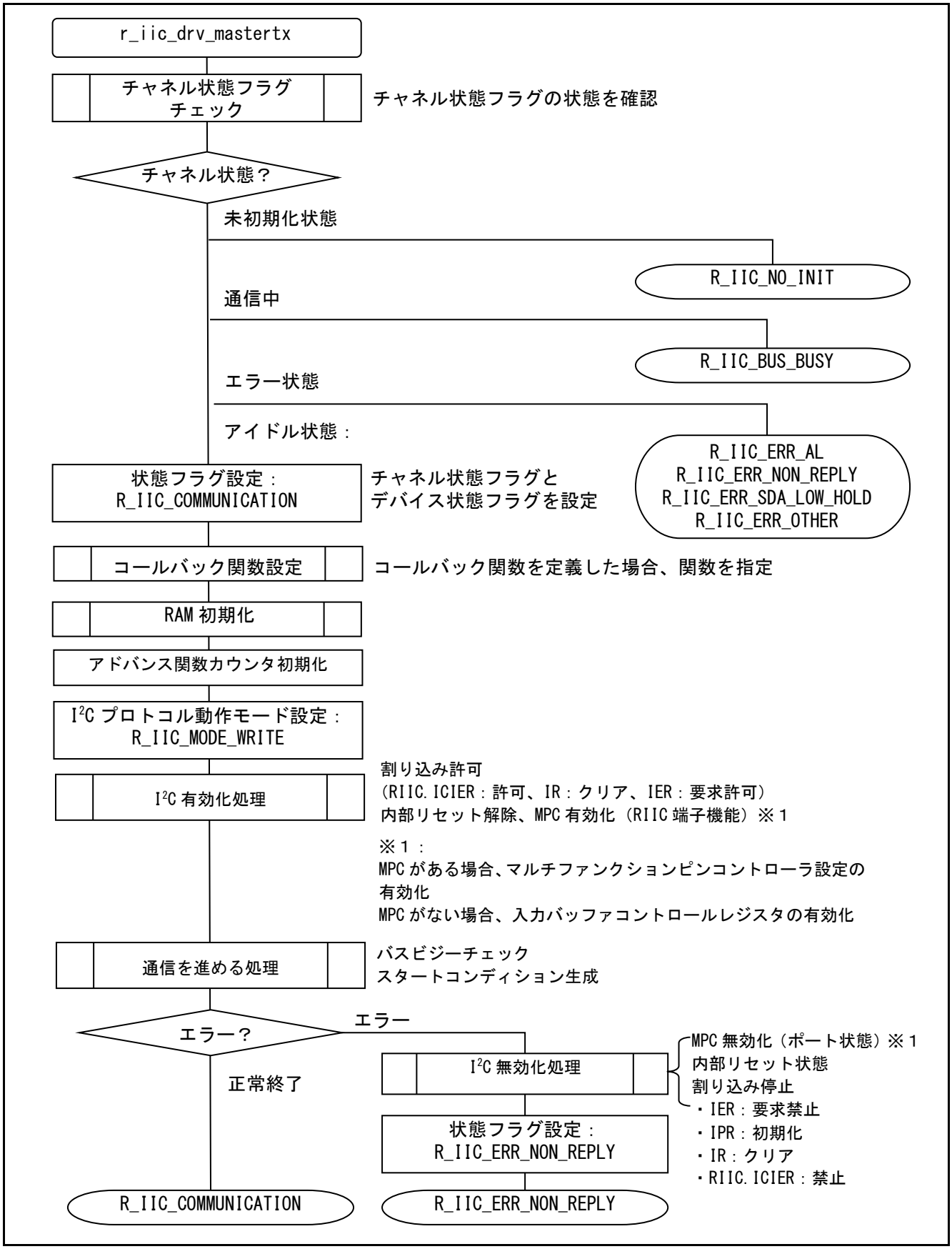


図 6-24　マスタ送信開始関数概略フロー

6.18.4 マスタ受信開始関数

R_IIC_Drv_MasterRx	
概要	マスタ受信開始関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_MasterRx (r_iic_drv_info_t *pRIic_Info)
説明	・マスタ受信を開始します。 ・本処理を行うためには、I²C 通信情報構造体“r_iic_drv_info_t”の設定が必要です。設定方法は表 6-14を参照してください。
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_COMMUNICATION マスタ受信を開始しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_COMMUNICATION”を設定しました。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_NO_INIT 初期化されていません（※１）。チャネル状態フラグとデバイス状態フラグは変更されません。 →初期化関数をコールして初期化を終了させてください。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_BUS_BUSY 通信中です。マスタ受信を開始できませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_AL アービトレーションロストが発生しています。チャネル状態フラグとデバイス状態フラグは変更されません。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_NON_REPLY バスが開放されていないか、スタートコンディションを検出できませんでした。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_NON_REPLY”を設定しました。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_SDA_LOW_HOLD SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグとデバイス状態フラグは変更されません。 →スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。 R_IIC_ERR_OTHER その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。
→以下を確認してください。
・I²C 通信情報構造体の設定が正しく行われているか確認してください。

備考
・本関数から戻った時点では、I²C 通信は終了していません。I²C 通信を終了させるためには、アドバンス関数をコールする必要があります。
・開始関数コール後の通信状態は、アドバンス関数のリターン値で確認できます。

※１：一度初期化した場合でも、同一チャネル上の他デバイスが I²C ドライバリセット関数をコールしたことにより、未初期化状態になることがあります。この場合、再度 I²C ドライバ初期化関数をコールしてから、開始関数をコールしてください。

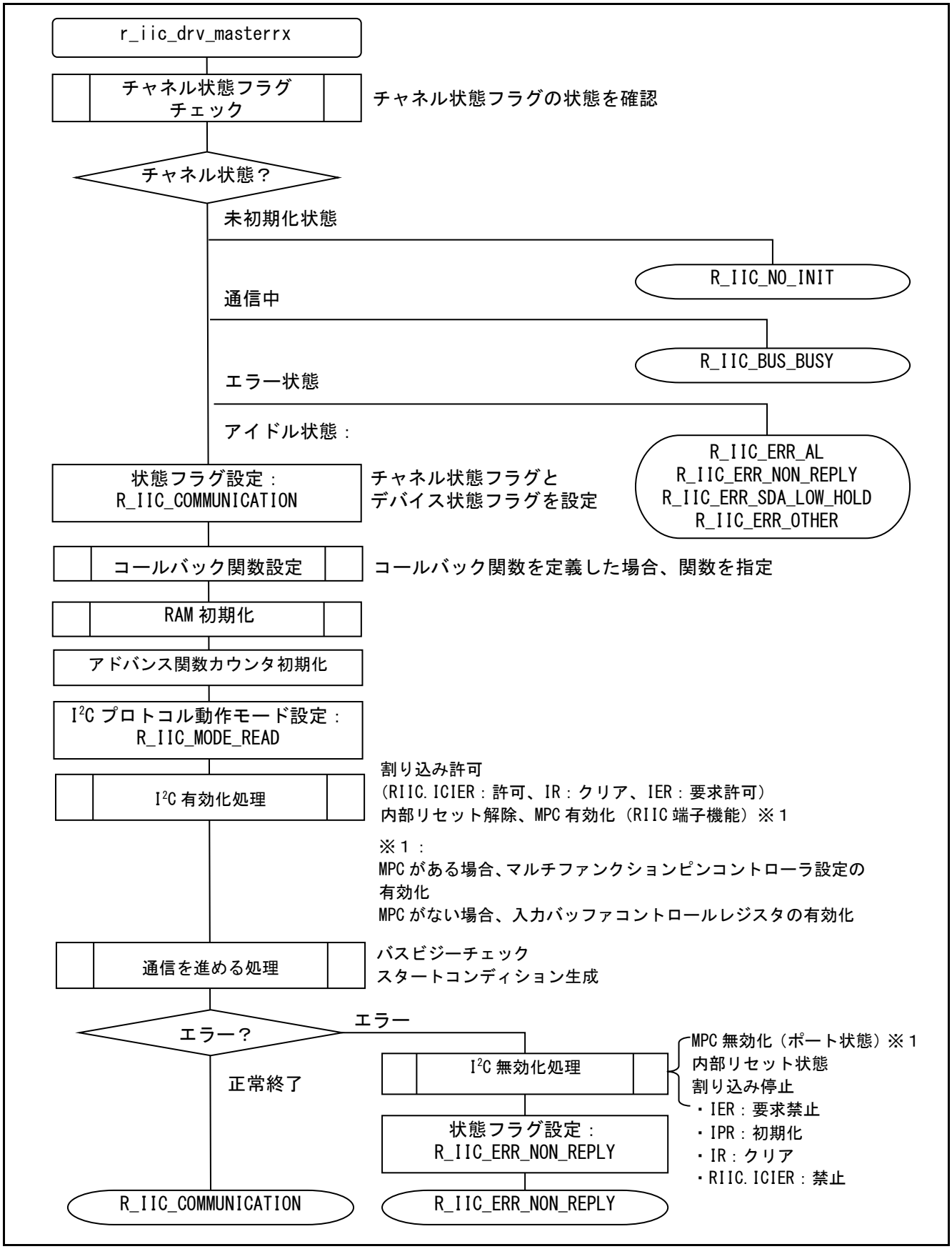


図 6-25　マスタ受信開始関数概略フロー

6.18.5 マスタ複合開始関数

R_IIC_Drv_MasterTRx	
概要	マスタ複合開始関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_MasterTRx (r_iic_drv_info_t *pRIic_Info)
説明	・マスタ複合を開始します。 ・本処理を行うためには、I²C 通信情報構造体“r_iic_drv_info_t”の設定が必要です。設定方法は表 6-14を参照してください。
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_COMMUNICATION マスタ複合を開始しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_COMMUNICATION”を設定しました。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_NO_INIT 初期化されていません（※１）。チャネル状態フラグとデバイス状態フラグは変更されません。 →初期化関数をコールして初期化を終了させてください。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_BUS_BUSY 通信中です。マスタ複合を開始できませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →アドバンス関数をコールして通信を終了させてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_AL アービトレーションロストが発生しています。チャネル状態フラグとデバイス状態フラグは変更されません。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_NON_REPLY バスが開放されていないか、スタートコンディションを検出できませんでした。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_NON_REPLY”を設定しました。 →7.3 復帰処理例　を参照し、復帰処理を行ってください。 R_IIC_ERR_SDA_LOW_HOLD SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグとデバイス状態フラグは変更されません。 →スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。 R_IIC_ERR_OTHER その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。
→以下を確認してください。
・I²C 通信情報構造体の設定が正しく行われているか確認してください。

備考
・本関数から戻った時点では、I²C 通信は終了していません。I²C 通信を終了させるためには、アドバンス関数をコールする必要があります。
・開始関数コール後の通信状態は、アドバンス関数のリターン値で確認できます。

※１：一度初期化した場合でも、同一チャネル上の他デバイスが I²C ドライバリセット関数をコールしたことにより、未初期化状態になることがあります。この場合、再度 I²C ドライバ初期化関数をコールしてから、開始関数をコールしてください。

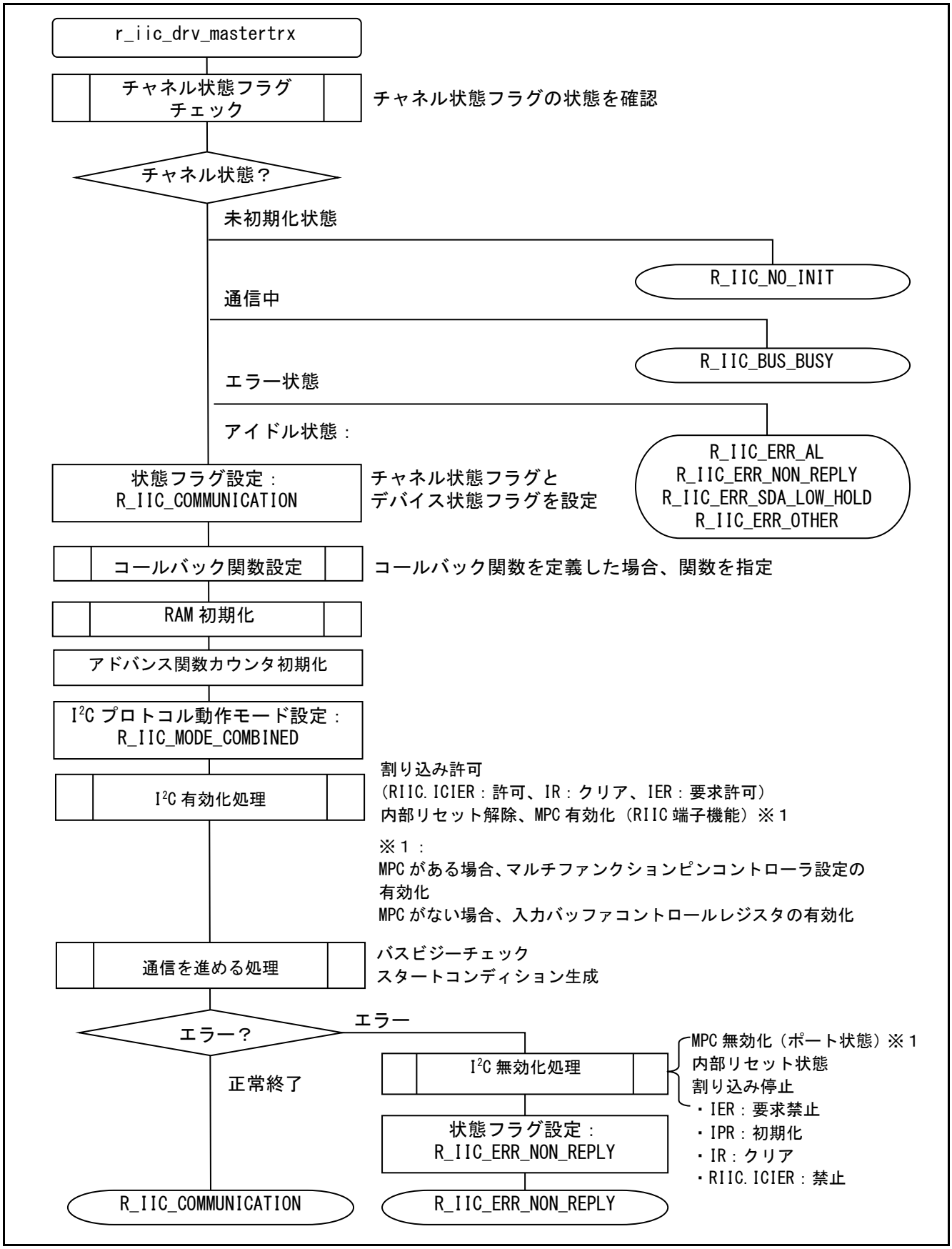


図 6-26　マスタ複合開始関数概略フロー

6.18.6 アドバンス関数

R_IIC_Drv_Advance	
概要	アドバンス関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_Advance (r_iic_drv_info_t *pRIic_Info)
説明	・通信を監視し、通信を進める処理を実行します。リターン値に通信の状態を返します。 ・次の通信を開始するには、アドバンス関数で通信を終了させる必要があります。
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_COMMUNICATION 通信中です。チャネル状態フラグとデバイス状態フラグは変更されません。 →アドバンス関数をコールして、通信を終了させてください。 R_IIC_FINISH 全通信は正常終了しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_FINISH”を設定しました。 すでに通信が終了している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。 →開始関数をコールすることで通信が可能です。 R_IIC_NACK NACK を検出しました。ストップコンディションを生成し、通信を終了しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_NACK”を設定しました。 すでに通信が終了している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。 →開始関数をコールすることで通信が可能です。 R_IIC_NO_INIT 初期化されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →初期化関数をコールして初期化を終了させてください。 R_IIC_IDLE アイドル状態です。チャネル状態フラグとデバイス状態フラグは変更されません。 →開始関数をコールすることで通信が可能です。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_BUS_BUSY 同一チャネル上の他のデバイスが通信中のため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他のデバイスの通信を終了させてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。

→引数を設定してください。

R_IIC_ERR_AL

アービトレーションロストが発生しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_AL”を設定しました。

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。

→7.3 復帰処理例　を参照し、復帰処理を行ってください。

R_IIC_ERR_NON_REPLY

下記が発生しました。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_NON_REPLY”を設定しました。

・アドバンス関数のコール回数が上限を超えた

・ストップコンディション生成処理を行ったが、一定時間が経過しても検出できなかった

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。

→ノイズ等の影響により SDA または SCL が Low ホールドした可能性があります。7.3 復帰処理例　を参照し、復帰処理を行ってください。

R_IIC_ERR_SDA_LOW_HOLD

SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグとデバイス状態フラグは変更されません。

→スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。

R_IIC_ERR_OTHER

その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。

すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。

→以下を確認してください。

・I²C 通信情報構造体の設定が正しく行われているか確認してください。

備考

・パラメータを確認します。

・イベントフラグ（g_iic_Event[]）がセットされている場合、以下の処理を行います。

アドバンス関数カウンタ（g_iic_ReplyCnt[]）を初期化します。

通信を進める処理を行います。

正常に処理された場合、全通信が終了したか確認します。全通信終了時、チャネル状態フラグに“R_IIC_FINISH”を設定します。

・イベントフラグ（g_iic_Event[]）がセットされていない場合、以下の処理を行います。

アドバンス関数カウンタ（g_iic_ReplyCnt[]）を減算します。

アドバンス関数カウンタが 0 の場合、リターン値に、”R_IIC_ERR_NON_REPLY”を設定します。

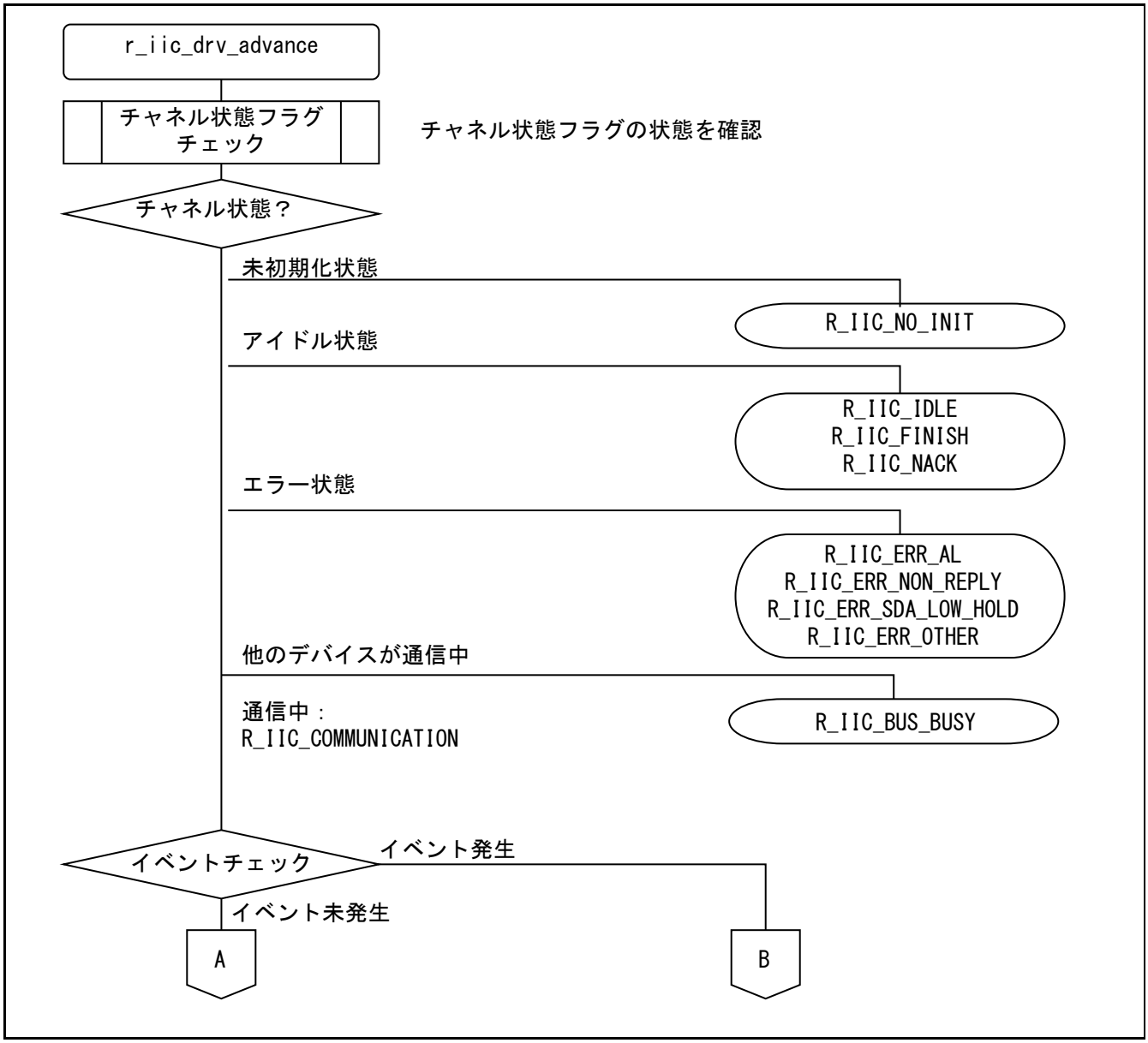


図 6-27　アドバンス関数概略フロー（1/3）

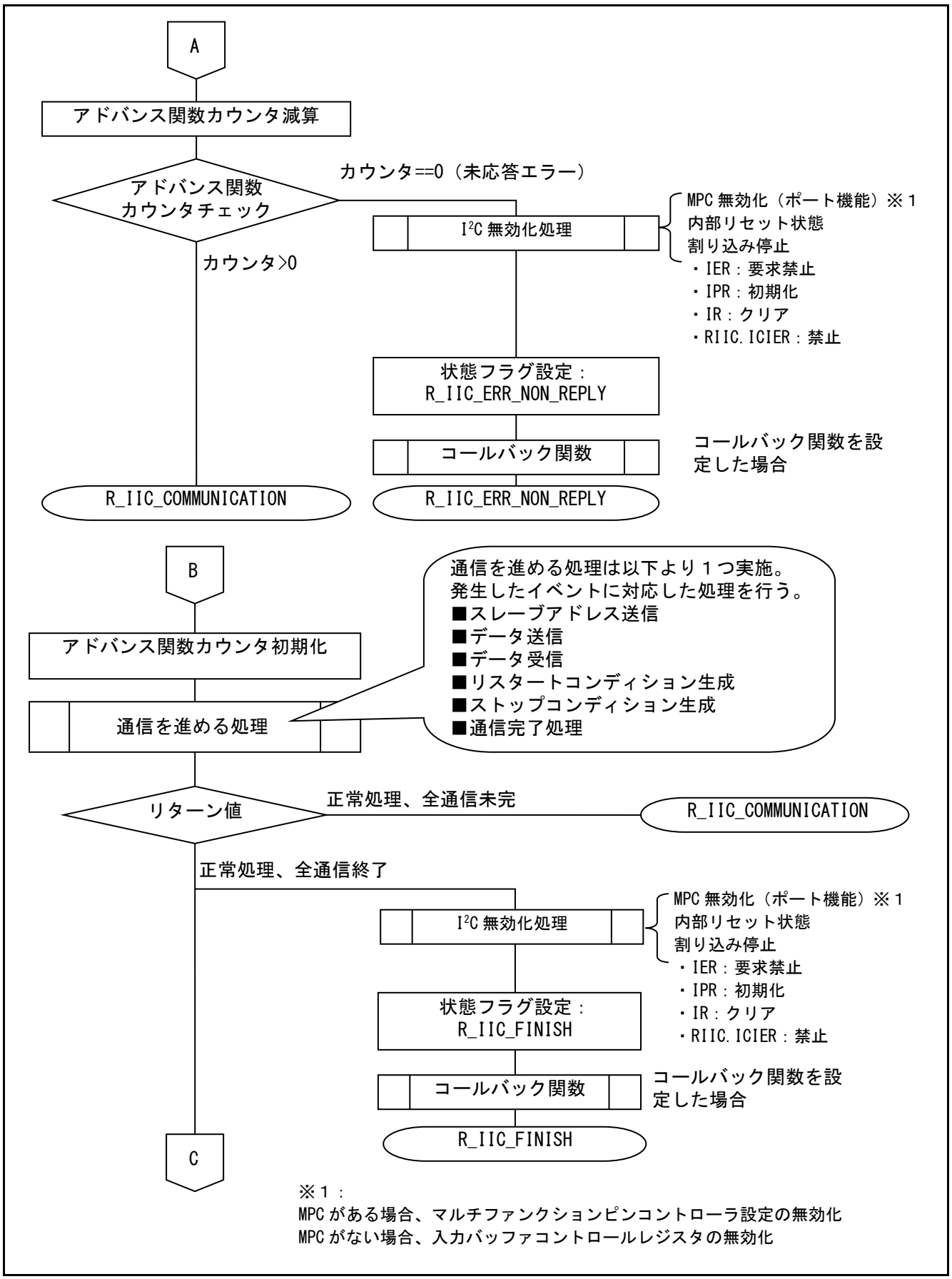


図 6-28　アドバンス関数概略フロー（2/3）

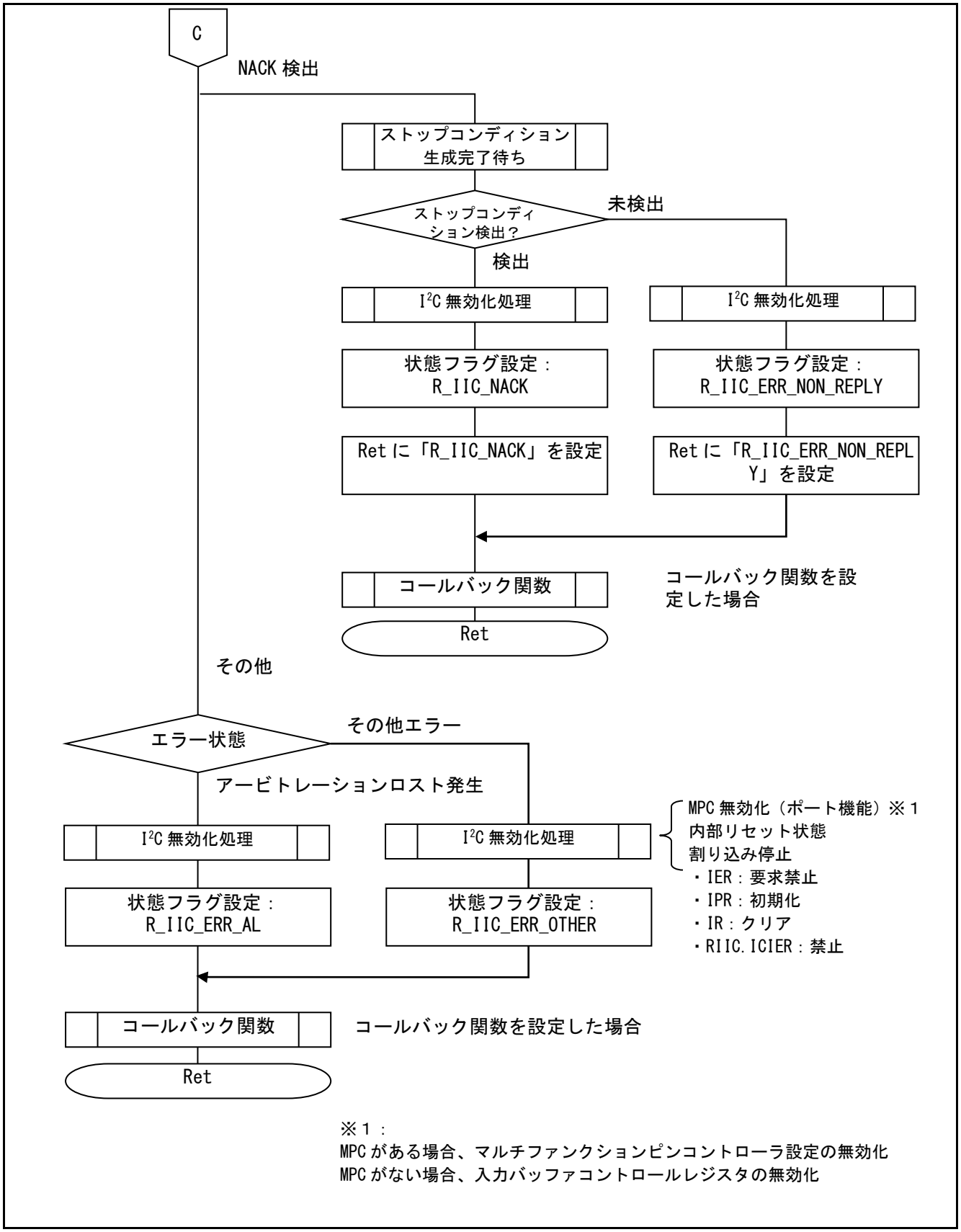


図 6-29　アドバンス関数概略フロー（3/3）

6.18.7 SCL 疑似クロック生成関数

R_IIC_Drv_GenClk	
概要	SCL 疑似クロック生成関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_GenClk (r_iic_drv_info_t *pRIic_Info, uint8_t ClkCnt)
説明	・SCL 疑似クロック生成を行います。ノイズ等の影響によりマスタとスレーブ間で同期ズレが発生し、SDA が Low ホールドした場合に、スレーブデバイス内部の状態を正常終了させることができます。 ・通常時は使用しないでください。正常な通信動作中に使用すると通信異常の原因になります。 ・本処理を行うためには、以下の設定が必要です。 構造体“r_iic_drv_info_t”メンバの ChNo ;使用チャネル番号 クロック回数“ClkCnt” ;01h～FFh
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ uint8_t ClkCnt ;SCL クロック回数
リターン値	R_IIC_NO_INIT SDA=High になり、スレーブデバイス内部の動作は正常終了し、未初期化状態になりました。チャネル状態フラグとデバイス状態フラグに“R_IIC_NO_INIT”を設定しました。 →通信を再開するには、以下の手順で処理を行ってください。 ①初期化関数コール ②マスタ送信のパターン 4 コール（※１） ③アドバンス関数をコールして通信を終了させる。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_SDA_LOW_HOLD SCL 疑似クロック生成を行いましたが、SDA は Low ホールドから復帰できていない状態です。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_SDA_LOW_HOLD”を設定しました。 →スレーブデバイスが Low ホールドしていないか、マスタデバイスから Low 信号が出力されていないか等、システムの状態を確認してください。 R_IIC_ERR_OTHER クロック生成ができませんでした。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。 →クロックは以下の条件時に出力できます。 ・バスフリー状態（ICCR2.BBSY フラグ=0）またはマスタモード（ICCR2.MST ビット=1、BBSY フラグ=1）のとき。 ・通信デバイスが SCL ラインを Low ホールドにしていない状態のとき。
備考	・SDA を Hi-Z 状態にした時、SDA=Low であれば、バスが解放されていないと判断します。

・SDA=Low の場合、SCL 端子をポート出力に切り替え、SDA=High になるまで Low->High クロックを入力します。
・設定回数クロックを生成しても SDA=Low の場合、エラーを返します。
・通信単位が 9 クロックの場合が多いため、クロック回数は 9 回以上設定することを推奨します。

※１：マスタ送信のパターン４はスタートコンディション生成後にストップコンディションを生成します。この処理はバスを確実に解放するために行います。

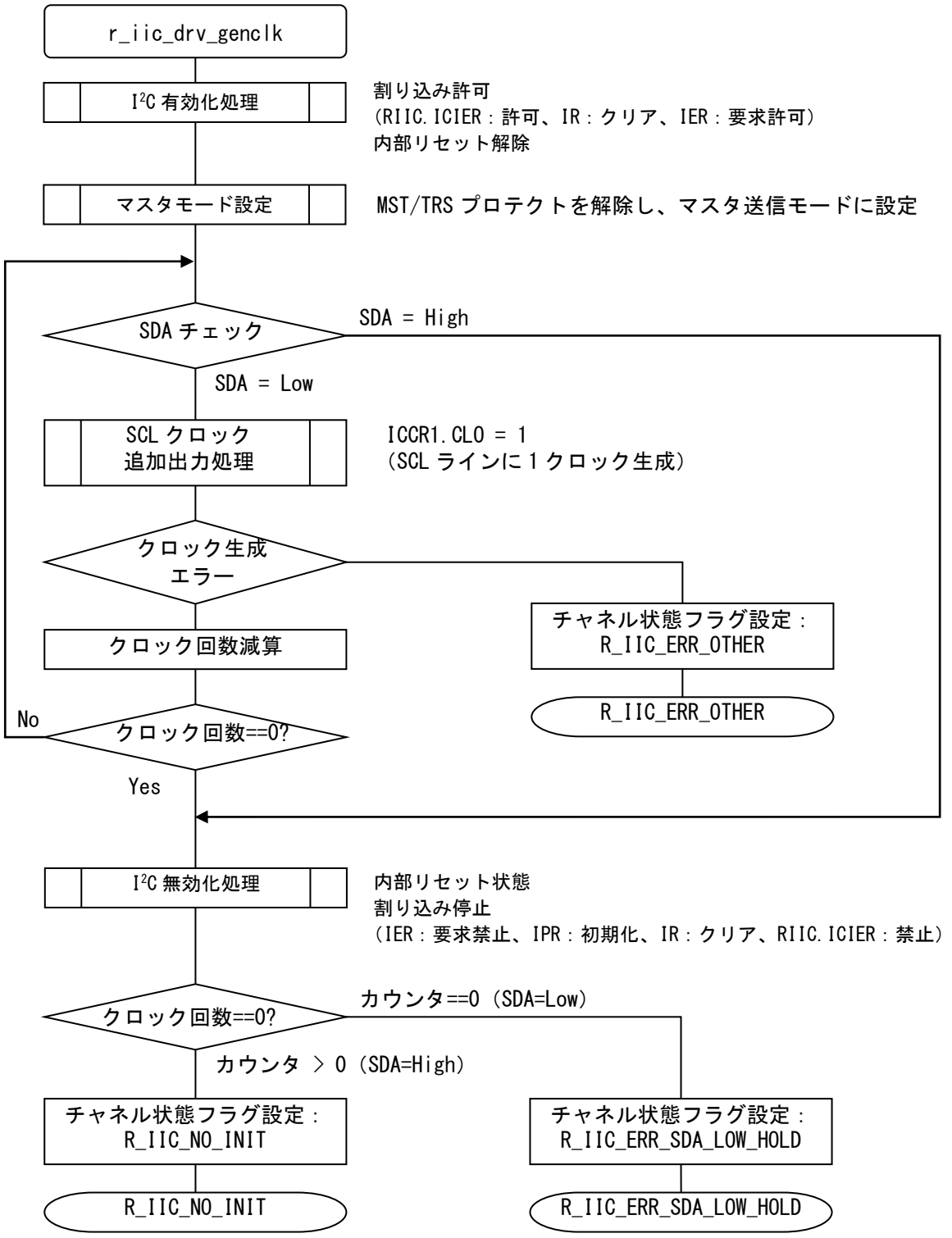


図 6-30　SCL 疑似クロック生成関数概略フロー

6.18.8 I²C ドライバリセット関数

R_IIC_Drv_Reset	
概要	I²C ドライバリセット関数
ヘッダ	r_iic_drv_api.h, r_iic_drv_sub.h, r_iic_drv_int.h, r_iic_drv_sfr.h
宣言	error_t R_IIC_Drv_Reset(r_iic_drv_info_t *pRIic_Info)
説明	・対象チャネルの I²C ドライバをリセットします。 ・ICCR1.ICE=1,IICRST=1 により RIIC を停止し、内部リセットを行います（※１）。 ・通信中に本関数をコールした場合でも、強制的に通信を停止します。 ・本処理を行うためには、以下の設定が必要です。 構造体“r_iic_drv_info_t”メンバの ChNo ;使用チャネル番号
引数	r_iic_drv_info_t *pRIic_Info ;I²C 通信情報構造体ポインタ
リターン値	R_IIC_NO_INIT 内部リセットを行い、未初期化状態になりました。チャネル状態フラグとデバイス状態フラグに“R_IIC_NO_INIT”を設定しました。 →通信を再開するには、以下の手順で処理を行ってください。 ①初期化関数コール ②マスタ送信のパターン 4 コール（※２） ③アドバンス関数をコールして通信を終了させる。 R_IIC_LOCK_FUNC 他の API 処理が行われているため、処理は行われませんでした。チャネル状態フラグとデバイス状態フラグは変更されません。 →他の API 処理終了後、本関数をコールしてください。 R_IIC_ERR_PARAM パラメータエラーです。引数が設定されていません。チャネル状態フラグとデバイス状態フラグは変更されません。 →引数を設定してください。 R_IIC_ERR_OTHER その他エラーが発生しています。チャネル状態フラグとデバイス状態フラグに“R_IIC_ERR_OTHER”を設定しました。 すでにエラーが発生している場合、何も処理は行いません。チャネル状態フラグとデバイス状態フラグは変更されません。 →以下を確認してください。 ・I²C 通信情報構造体の設定が正しく行われているか確認してください。
備考	・再度通信を開始するには、I²C ドライバ初期化関数をコールする必要があります。 ・通信中に強制的に RIIC を停止した場合、その通信は保証しません。 ※１：内部リセットするレジスタ、およびビット I²C バスコントロールレジスタ 1（ICCR1）の SCLO,SDAO ビット I²C バスコントロールレジスタ 2（ICCR2）の ST ビット I²C バスモードレジスタ 1（ICMR1）の BC[2:0]ビット I²C バスバスステータスレジスタ 1（ICSR1） I²C バスバスステータスレジスタ 2（ICSR2） I²C バスシフトレジスタ（ICDRS） ※２：マスタ送信のパターン４はスタートコンディション生成後にストップコンディションを生成します。この処理はバスを確実に解放するために行います。

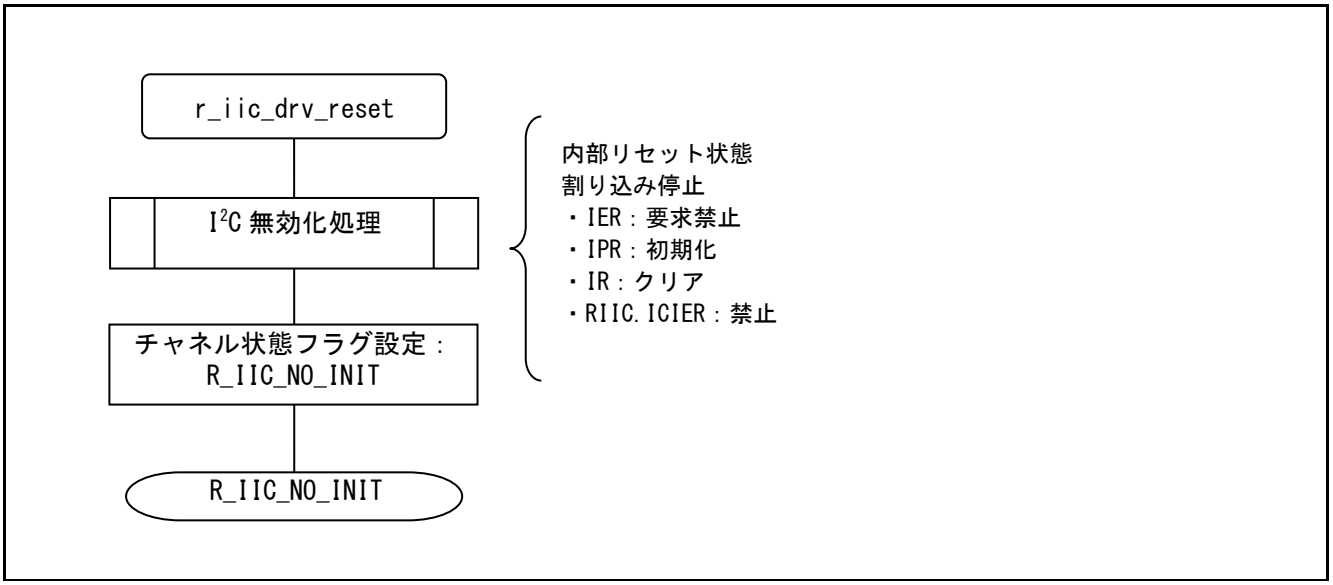


図 6-31　I²C ドライバリセット関数概略フロー

7. 応用例

7.1 r_iic_drv_api.h

使用する上での設定例を以下に示します。

設定箇所は、各ファイル中の「/** SET **/」というコメントの部分です。

(1) 使用する RIIC チャネルの選択

使用する I²C チャネルを指定してください。使用しないチャネルをコメントアウトすることで、使用 ROM 容量を縮小することができます。

下記の例は、チャネル 0 と 1 を使用する場合です。

```
/*-------------------------------------------------------------------------*/
/*   Select channels to enable.                                            */
/*-------------------------------------------------------------------------*/
#define RIIC0_ENABLE
#define RIIC1_ENABLE
```

(2) 使用する最大チャネル番号の定義

「使用する最大チャネル番号＋ 1 」の値を設定してください。

下記の例は、チャネル 0 とチャネル 1 使用する場合です。この場合、使用する最大チャネル番号は”1”であり、設定値は”2”になります。

```
/*-------------------------------------------------------------------------*/
/*   Define channel No.(max) + 1.                                          */
/*-------------------------------------------------------------------------*/
#define MAX_IIC_CH_NUM          (uint8_t)(2)
```

(3) アドバンス関数を RIIC 割り込みハンドラでコールする場合の定義

アドバンス関数を RIIC 割り込みハンドラでコールする場合、以下を定義してください。

```
/*-------------------------------------------------------------------------*/
/*   Define to use an advance processing by RIIC interrupt handler.        */
/*-------------------------------------------------------------------------*/
#define CALL_ADVANCE_INTERRUPT
```

(4) 各カウンタの定義

ソフトウェアループ用のカウンタです。したがって使用するシステム・クロックによりループ時間が変化します。必要に応じて設定値を見直してください。

```
/*------------------------------------------------------------------------------*/
/*   Define counter.                                                            */
/*------------------------------------------------------------------------------*/
#define REPLY_CNT        (uint32_t)(100000) /* Counter of non-reply errors   */
#define STOP_COND_WAIT   (uint16_t)(1000)
                          /* Counter of waiting stop condition generation    */
#define BUSCHK_CNT       (uint16_t)(1000) /* Counter of checking bus busy    */
#define SDACHK_CNT       (uint16_t)(1000) /* Counter of checking SDA level   */
#define GEN_SCLCLK_WAIT  (uint16_t)(1000) /* Counter of waiting SCL clock setting*/
```

7.2 r_iic_drv_sfr.h

r_iic_drv_sfr.h.XXX は、MCU 毎に作成したものです。どれか一つを r_iic_drv_sfr.h にリネームして使用してください。対象 MCU のものが無い場合には、参照して、r_iic_drv_sfr.h を作成してください。

使用する上での設定例を以下に示します。

設定箇所は、各ファイル中の「/** SET **/」というコメントの部分です。

(1) 転送クロックの定義

転送クロックは、以下の設定により決定します。使用するチャネル毎に設定する必要があります。それぞれ定義してください。詳しい設定方法については、RX ファミリユーザーズマニュアル　ハードウェア編をご参照ください。

・I²C バスモードレジスタ 1 （ICMR1）の内部基準クロック選択ビット（CKS[2:0]）

・I²C バスビットレートローレベルレジスタ（ICBRL）

・I²C バスビットレートハイレベルレジスタ（ICBRH）

最大 400KHz までの設定可能です。ただし、標準モード・デバイスは、ファーストモード・デバイスが混載されるバスでは、標準モードの最大 100KHz に設定する必要があります。また、SDA, SCL 信号の立ち上がり時間（tR）と立ち下がり時間（tF）は，プルアップ抵抗と配線容量によって異なるため，必要に応じて設定値を見直してください。

下記は、チャネル 0 と 1 の設定例です。

```
/*-----------------------------------------------------------------------------*/
/*   Define frequency as iic channel. (Please add a channel as needed.)       */
/*-----------------------------------------------------------------------------*/
/* The I2C transfer rate is calculated using the following expression.       */
/* Transfer rate = 1 / {[(ICBRH + 1) + (ICBRL + 1)] / (PCLK*Division ratio)
                + SCLn line rising time [tr] + SCLn line falling time [tf]} */
/* Note1:Division ratio sets it by ICMR1.CKS[2:0].                           */

/* Freq = 400KHz at main system clock = 48MHz */
#define R_IIC_CH0_LCLK  (uint8_t)(0xED) /* Channel 0 ICBRL register setting    */
#define R_IIC_CH0_HCLK  (uint8_t)(0xE6) /* Channel 0 ICBRH register setting    */
#define R_IIC_CH1_LCLK  (uint8_t)(0xED) /* Channel 1 ICBRL register setting    */
#define R_IIC_CH1_HCLK  (uint8_t)(0xE6) /* Channel 1 ICBRH register setting    */

/* Sets ICMR1 register.*/
#define R_IIC_CH0_ICMR1_INIT    (uint8_t)(0x28)
                               /* Channel 0 ICMR1 register setting          */
#define R_IIC_CH1_ICMR1_INIT    (uint8_t)(0x28)
                               /* Channel 1 ICMR1 register setting          */
```

(2) ポート番号の定義

RX210 と RX21A のチャネル 0 を使用する場合、使用する端子のポート番号を定義してください。

下記の例は、チャネル 0 のポート 12（SCL0）とポート 13（SDA0）を使用する場合です。

```
/*----------------------------------------------------------------------------*/
/*   Define channel register.                                                 */
/*----------------------------------------------------------------------------*/
#ifdef RIIC0_ENABLE

/* Define port registers */
#define R_IIC_PDR_SCL0     PORT1.PDR.BIT.B2  /* SCL0 Port direction register */
#define R_IIC_PDR_SDA0     PORT1.PDR.BIT.B3  /* SDA0 Port direction register */
#define R_IIC_PODR_SCL0    PORT1.PODR.BIT.B2 /* SCL0 Port output data register */
#define R_IIC_PODR_SDA0    PORT1.PIDR.BIT.B3 /* SDA0 Port output data register */
#define R_IIC_PIDR_SCL0    PORT1.PODR.BIT.B2 /* SCL0 Port input data register */
#define R_IIC_PIDR_SDA0    PORT1.PIDR.BIT.B3 /* SDA0 Port input data register */
#define R_IIC_PMR_SCL0     PORT1.PMR.BIT.B2  /* SCL0 Port mode register     */
#define R_IIC_PMR_SDA0     PORT1.PMR.BIT.B3  /* SDA0 Port mode register     */
#define R_IIC_DSCR_SCL0    PORT1.DSCR.BIT.B2
                                  /* SCL0 Drive capacity control register    */
#define R_IIC_DSCR_SDA0    PORT1.DSCR.BIT.B3
                                  /* SDA0 Drive capacity control register    */
#define R_IIC_PCR_SCL0     PORT1.PCR.BIT.B2
                                  /* SCL0 Pull-up resistor control register  */
#define R_IIC_PCR_SDA0     PORT1.PCR.BIT.B3
                                  /* SDA0 Pull-up resistor control register  */
```

(3) マルチファンクションピンコントローラ（MPC）の定義

RX210 と RX21A のチャネル 0 を使用する場合、使用する端子のマルチファンクションピンコントローラ（MPC）のレジスタ番号を定義してください。

下記の例は、チャネル 0 のポート 12（SCL0）とポート 13（SDA0）を使用する場合です。

```
/* Define Pin function control registers */
#define R_IIC_MPC_SCL0  MPC.P12PFS.BYTE  /* SCL0 Pin function control register */
#define R_IIC_MPC_SDA0  MPC.P13PFS.BYTE  /* SDA0 Pin function control register */
```

(4) RIIC 割り込み優先順位の定義

使用する RIIC チャネルの割り込み優先順位を決めるため、割り込み要因プライオリティレジスタ（IPR）に設定する値を定義してください。使用するチャネル毎に設定する必要があります。

下記の例は、チャネル 0 を使用し、各 RIIC 割り込みの優先順位をレベル 2 に設定する場合です。

```
/* Sets interrupt source priority initialization. */
#define R_IIC_IPR_CH0_EEI_INIT    (uint8_t)(0x02)
                                  /* EEIx interrupt source priority initialization*/
#define R_IIC_IPR_CH0_RXI_INIT    (uint8_t)(0x02)
                                  /* RXIx interrupt source priority initialization*/
#define R_IIC_IPR_CH0_TXI_INIT    (uint8_t)(0x02)
                                  /* TXIx interrupt source priority initialization*/
#define R_IIC_IPR_CH0_TEI_INIT    (uint8_t)(0x02)
                                  /* TEIx interrupt source priority initialization*/
```

7.2.2 割り込みハンドラの設定

本サンプルコードで使用する割り込みは、ICEEI 割り込み、ICTEI 割り込み、ICRXI 割り込みです。

総合開発環境で生成される vect.h（ベクタ関数のヘッダ）と intprg.c（ベクタ関数の定義）を使用する場合と使用しない場合の設定例を以下に示します。

(1) 使用する場合

intprg.c の RIIC 割り込み定義部分に、r_iic_drv_int.c の使用するチャネルの割り込みハンドラ関数を定義してください。

以下は、チャネル 0 を使用する場合の設定例です。

```
// RIIC0 EEI0
void Excep_RIIC0_EEI0(void){ r_iic_drv_intRIIC0_EEI_isr(); }

// RIIC0 RXI0
void Excep_RIIC0_RXI0(void){ r_iic_drv_intRIIC0_RXI_isr(); }

// RIIC0 TEI0
void Excep_RIIC0_TEI0(void){ r_iic_drv_intRIIC0_TEI_isr(); }
```

(2) 使用しない場合

r_iic_drv_int.c で使用するチャネルの割り込みハンドラ関数を#pragma interrupt 定義してください。
以下は、チャネル 0 を使用する場合の設定例です。

■ICEEI 割り込み定義

```
#pragma interrupt (r_iic_drv_intRIIC0_EEI_isr(vect=VECT_RIIC0_RXI0))
void r_iic_drv_intRIIC0_EEI_isr(void)
```

■ICRXI 割り込み定義

```
#pragma interrupt (r_iic_drv_intRIIC0_RXI_isr(vect=VECT_RIIC0_RXI0))
void r_iic_drv_intRIIC0_RXI_isr(void)
```

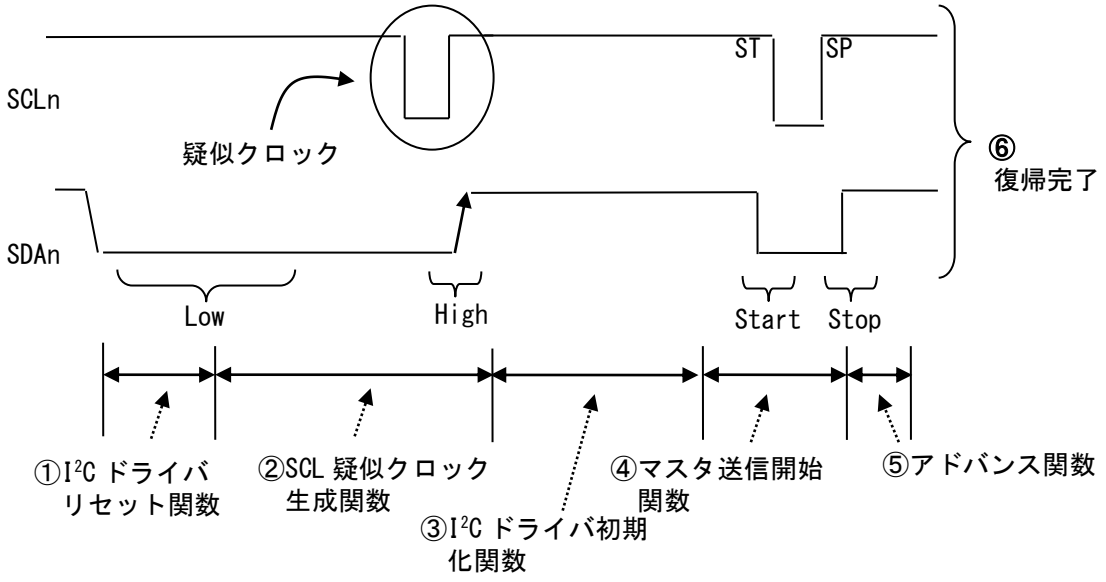
■ICTEI 割り込み定義

```
#pragma interrupt (r_iic_drv_intRIIC0_TEI_isr(vect=VECT_RIIC0_TEI0))
void r_iic_drv_intRIIC0_TEI_isr(void)
```

7.3 復帰処理例

SDA または SCL が Low ホールドした場合の通信への復帰処理を説明します。以下の手順で処理を行ってください。図 7-1に SCL 疑似クロック生成による復帰処理例を示します。

尚、本復帰処理後は、アイドル状態になります。開始関数をコールすることで通信が可能です。



①I²C 内部リセットを行います。
②SCL に疑似クロックを生成し、スレーブデバイスの Low ホールド状態から解放させます。
③マスタ送信（パターン 4）を行うため、初期化を行います。
④マスタ送信（パターン 4）を開始します。スタートコンディション生成後、ストップコンディションを生成します。
⑤アドバンス関数により通信を終了させます。
⑥アイドル状態になりました。以降、各開始関数をコールすることで通信が可能です。

図 7-1 SCL 疑似クロック生成による復帰処理例

①I²C ドライバリセット関数　R_IIC_Drv_Reset()

本処理は、内部リセットを行うことで、SDA 及び SCL のホールドを解除します。

本処理後に、SDA 及び SCL が High であることを確認してください。リセット後も SDA または SCL が Low ホールドしている場合、スレーブデバイスが Low ホールドしているか、マスタデバイスから Low 信号が出力されているなどの可能性があります。

②SCL 疑似クロック生成関数　R_IIC_Drv_GenClk()

本処理は、スレーブデバイスにより SDA を Low ホールドされた場合に、SCL に疑似クロックを生成することで、スレーブデバイスの内部処理を終了させて SDA を High にします。

リターン値が”R_IIC_SDA_HIGH”であれば SDA=High となり解除されました。”R_IIC_ERR_SDA_LOW_HOLD”なら SDA Low ホールドから復帰できていません。システムの状態を見直してください。

③I²C 初期化　R_IIC_Drv_Init()

上記処理で、SDA 及び SCL の Low ホールドを解除した場合、I²C ドライバ初期化関数をコールしてください。

④バス解放処理　R_IIC_Drv_MasterTx() パターン４

ストップコンディション生成により、バスを解放します。本サンプルコードでは、マスタ送信パターン４により『スタートコンディション生成→ストップコンディション生成』を行うことで実現します。

パターン４の設定を行い、マスタ送信開始関数をコールしてください。

⑤アドバンス関数　R_IIC_Drv_Advance()

④項で開始した処理をアドバンス関数コールにより終了させてください。

正常に終了するとアイドル状態になります。開始関数をコールすることで通信が可能です。

7.4 RIIC 割り込みハンドラでアドバンス関数をコールする場合の注意事項

RIIC 割り込みハンドラでアドバンス関数をコールする場合、以下の条件を守ってご使用ください。

(1) #define CALL_ADVANCE_INTERRUPT 有効化

7.1(3)に示すとおり、CALL_ADVANCE_INTERRUPT を定義してください。

(2) I²C 通信情報構造体の定義

r_iic_drv_int.h で定義されている以下のグローバルの I²C 通信情報構造体をメイン処理で定義し、設定してください。この定義はチャネル毎に用意されています。

```
#ifdef CALL_ADVANCE_INTERRUPT
    #ifdef RIIC0_ENABLE
extern r_iic_drv_info_t   g_iic_Info_ch0;   /* Channel 0 IIC driver information*/
    #endif /* #ifdef RIIC0_ENABLE */
    #ifdef RIIC1_ENABLE
extern r_iic_drv_info_t   g_iic_Info_ch1;   /* Channel 1 IIC driver information*/
    #endif /* #ifdef RIIC1_ENABLE */
    #ifdef RIIC2_ENABLE
extern r_iic_drv_info_t   g_iic_Info_ch2;   /* Channel 2 IIC driver information*/
    #endif /* #ifdef RIIC2_ENABLE */
    #ifdef RIIC3_ENABLE
extern r_iic_drv_info_t   g_iic_Info_ch3;   /* Channel 3 IIC driver information*/
    #endif /* #ifdef RIIC3_ENABLE */
#endif  /*#ifdef CALL_ADVANCE_INTERRUPT*/
```

(3) 開始関数コール時の RIIC 割り込み禁止／許可処理の追加

RIIC 割り込みハンドラでアドバンス関数をコールする場合、ユーザ側で各開始関数を呼び出す前後に RIIC 割り込み禁止と許可処理を追加してください。

この区間に RIIC 割り込みが発生し、RIIC 割り込みハンドラのアドバンス関数がコールされた場合、API の多重コールとなり、アドバンス関数は処理されずに終了します。これにより、その後の通信ができなくなります。

(4) 通信の完了判定方法

通信完了はコールバック関数のコールによりフラグをセットする等して確認してください。コールバック関数は、正常終了またはエラー終了によりコールされます。

コールバック関数はユーザが作成し、I²C 通信情報構造体の CallBackFunc に設定してください。

(5) 正常完了とエラーの判定方法

通信の完了を確認した後、チャネル状態フラグ（g_iic_ChStatus[]）を読み出すことで確認することができます。

(6) 通信中の RIIC 割り込みハンドラのアドバンス関数以外の API 関数コールの禁止

RIIC 割り込みハンドラでアドバンス関数をコールする場合、通信中に RIIC 割り込みハンドラのアドバンス関数以外の API 関数をコールしないでください。

通信中に API 関数がコールされ、その実行中に RIIC 割り込みが発生し、RIIC 割り込みハンドラのアドバンス関数がコールされた場合、API の多重コールとなり、割り込みハンドラのアドバンス関数は処理されずに終了します。これにより、その後の通信ができなくなります。

8. 使用上の注意事項

8.1 組み込み時の注意事項

8.1.1 インクルードファイル

本サンプルコードを組み込む場合は、以下をインクルードしてください。

・r_iic_drv_api.h
・r_iic_drv_sub.h
・r_iic_drv_sfr.h
・r_iic_drv_int.h

8.2 初期化時の注意事項

システム起動後、初めて初期化する場合、使用する全チャネルのチャネル状態フラグ g_iic_ChStatus[]に“R_IIC_NO_INIT”を設定してください。また、使用する全スレーブデバイスのデバイス状態フラグ*(pRIic_Info.pDevStatus)に“R_IIC_NO_INIT”を設定してください。

両フラグの設定を行った後、使用するスレーブデバイスの構造体情報を I²C ドライバ初期化関数にセットし、I²C ドライバ初期化関数をコールしてください。最初に全てのスレーブデバイスの初期化を完了させてください。

以降、チャネル状態フラグとデバイス状態フラグは、本サンプルコードにより管理するため、設定は禁止です。

8.3 チャネル状態フラグとデバイス状態フラグについて

本サンプルコードでは、チャネル状態フラグとデバイス状態フラグにより、通信状態の整合性を保っています。最初の初期化以外で変更された場合、その後の通信は保証しませんのでご注意ください。

8.4 動作検証用プログラム

本サンプルコードの動作検証用プログラムは、EEPROM の書き込みと読み出し動作を行うものです。

8.5 組み込み例

RIIC 割り込みハンドラでアドバンス関数をコールする場合の組み込み方法については、動作検証用プログラム sample_background.c をご参照ください。また、7.4項の注意事項を守ってご使用ください。

メイン処理でアドバンス関数をコールする場合の組み込み方法については、動作検証用プログラム sample_foreground.c をご参照ください。

8.6 同一チャネル上の複数のスレーブデバイスの制御方法

同一チャネル上の複数のスレーブデバイスを制御するためには、以下の手順で処理を行ってください。

下記①項の処理により、非通信状態のデバイスに対して通信が行われることを防止できます。

①アドバンス関数コール対象のスレーブデバイスの I²C 通信情報構造体のデバイス状態フラグが“R_IIC_COMMUNICATION”（通信中）であることを確認する。

②アドバンス関数をコールする。

③通信が終了するまで①～②を繰り返す。

④通信終了。以降、開始関数をコールすることで通信が可能です。

アドバンス関数は、スレーブデバイスを識別せず、通信中のスレーブデバイスの処理を進めるものです。したがって、デバイス状態フラグが“R_IIC_COMMUNICATION”（通信中）でない場合であっても、アドバンス関数コールにより通信中のスレーブデバイスの処理が進みます。

8.7 転送速度設定

対象チャネル毎に設定する必要があります。最大 400KHz までの設定可能です。

ただし、標準モード・デバイスとファーストモード・デバイスが混載されるバスでは、標準モードの最大 100KHz に設定する必要があります。転送レートは、表 6-11に定義する R_IIC_CHx_LCLK／R_IIC_CHx_LCLK／R_IIC_CHx_ICMR1_INIT（x はチャネル番号）で設定してください。

8.8 #define 定義「RIICx_ENABLE」と「MAX_IIC_CH_NUM」の設定について

例として、チャネル 2 のみを使用する場合の設定方法を示します。

#define 定義「RIICx_ENABLE」は「RIIC2_ENABLE」のみ有効にします。これにより、チャネル 0 とチャネル 1 のソースコードをマスクします。

```
/*------------------------------------------------------------------------*/
/*   Select channels to enable.                                           */
/*------------------------------------------------------------------------*/
/* #define RIIC0_ENABLE */
/* #define RIIC1_ENABLE */
#define RIIC2_ENABLE
```

#define 定義「MAX_IIC_CH_NUM」には“3”を設定します。使用するチャネル数は“1”ですが、ここで設定する値は「使用する最大チャネル番号＋１」になりますので、ご注意ください。

```
/*------------------------------------------------------------------------*/
/*   Define channel No.(max) + 1.                                         */
/*------------------------------------------------------------------------*/
#define MAX_IIC_CH_NUM        (uint8_t)(3)
```

8.9 RIIC 端子割り当てポート端子一覧

各 MCU の RIIC 端子が割り当てられているポート端子を下記に示します。

表 8-1 RIIC 端子割り当てポート一覧

MCU	ピン数	チャネル数	RIIC0		RIIC1		RIIC2		RIIC3	
			SCL0	SDA0	SCL1	SDA1	SCL2	SDA2	SCL3	SDA3
RX62N	176	2	P12	P13	P21	P20	–	–	–	–
	145	2	P12	P13	P21	P20	–	–	–	–
	100	1	P12	P13	–	–	–	–	–	–
	85	2	P12	P13	P21	P20	–	–	–	–
RX63N/ RX631	177,176	4	P12	P13	P21	P20	P16	P17	PC0	PC1
	145,144	4	P12	P13	P21	P20	P16	P17	PC0	PC1
	100	2	P12	P13	–	–	P16	P17	–	–
	64	1	–	–	–	–	P16	P17	–	–
	48	1	–	–	–	–	P16	P17	–	–
RX63T	144	2	PB1	PB2	P25	P26	–	–	–	–
	120	2	PB1	PB2	P25	P26	–	–	–	–
	112	1	PB1	PB2	–	–	–	–	–	–
	100	1	PB1	PB2	–	–	–	–	–	–
	64	1	PB1	PB2	–	–	–	–	–	–
	48	1	PB1	PB2	–	–	–	–	–	–
RX210	145,144	1	P12,P16	P13,P17	–	–	–	–	–	–
	100	1	P12,P16	P13,P17	–	–	–	–	–	–
	80	1	P12,P16	P13,P17	–	–	–	–	–	–
	64	1	P16	P17	–	–	–	–	–	–
	48	1	P16	P17	–	–	–	–	–	–
RX21A	100	2	P12,P16	P13,P17	P21	P20	–	–	–	–
	80	2	P12,P16	P13,P17	P21	P20	–	–	–	–
	64	1	P16	P17	–	–	–	–	–	–

8.10 ポート端子の指定が必要な MCU

8.9項に示すとおり、RX210 と RX21A のチャネル 0 は、SCL と SDA を 2 つのポートで使用することができます。使用する場合は、どちらか一方のポートを指定してください。

8.11 マスタ受信、マスタ複合時のスレーブアドレス送信直後の NACK 検出処理

マスタ受信またはマスタ複合時に、スレーブアドレス送信（転送方向ビット：“1”（Read））の際の 9 ビット目に NACK を受信した場合、ストップコンディション生成設定後に、I²C バス受信データレジスタ（ICDRR）のダミーリードを行います。

上記条件で NACK を検出した場合であっても、受信データフルフラグには“1”がセットされます。このフラグのクリアを行うために ICDRR のダミーリードを行います。

ホームページとサポート窓口

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

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

改訂記録	RX600, RX200 シリーズ　アプリケーションノート　RIIC を使った I2C シングルマスタ制御ソフトウェア

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2013.03.01	—	初版発行
1.01	2013.05.31	6-8	サンプルコードのバージョンを Ver.1.10 から Ver.1.10.R01 に改定した。
		7	RX63T の総合開発環境を変更した。
		7	RX63T の C コンパイラのツールチェーン Ver.を 1.2.1.0 から 2.00.00 に変更した。
		9	図 5-1 RX ファミリ I^2C バスインタフェースと I^2C スレーブデバイスの接続例　内容を修正した。
		33	RX63T の必要メモリサイズ　内容を修正した。
		34	表 6-11 ファイル構成　アプリケーションノートの改定にともない、Rev.を修正した。
1.02	2013.11.29	6-8	サンプルコードのバージョンを Ver.1.10.R01 から Ver.1.12 に改定した。
		24	図 6-15 図の③の内容を修正した。
		25	図 6-16 図の⑦の内容を修正した。
		29	6.7.2 マスタ受信 ▲2 の内容を修正した。
		30	6.7.3 マスタ複合▲5 の内容を修正した。
		32-33	RX62N, RX63N, RX63T, RX210, RX21A の必要メモリサイズ 内容を修正した。
		34	表 6-11 ファイル構成　アプリケーションノートの改定にともない、Rev.を修正した。
		-	製品ご使用上の注意事項とご注意書き更新した。
1.03	2015.01.30	6-8	2.動作確認条件　にて、動作確認表内 サンプルコードのバージョン　元は Ver.1.12 であった。 使用ソフトウェアのバージョン　元は Ver.1.00 であった。
		32	6.10 必要メモリサイズ　にて、 元は RX62N, RX63T, RX21A のメモリサイズを記載していた。 RX63N, RX210 の必要メモリサイズを修正した。
		33	表 6-8 ファイル構成　アプリケーションノートの改定にともない、Rev.を修正した。
		67	6.18.7 SCL 疑似クロック生成関数 図 6-30 : 元は MPC を設定していた。
		69	6.18.8 I^2C ドライバリセット関数 図 6-31 : 元は MPC を設定していた。

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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

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

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/