RX ファミリ

GPIO モジュール Firmware Integration Technology

R01AN1721JJ0241
Rev.2.41
2018.11.16

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT) を使用した GPIO モジュールを使って、汎用入出力ドライバをシステムに組み込んでご使用いただけます。以降、本モジュールを GPIO FIT モジュールと称します。

対象デバイス

- RX110 グループ
- RX111 グループ
- RX113 グループ
- RX130 グループ
- RX210 グループ
- RX230 グループ
- RX231 グループ
- RX23T グループ
- RX24T グループ
- RX24U グループ
- RX63N グループ
- RX64M グループ
- RX651、RX65N グループ
- RX66T 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 e^2 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

1.1 GPIO 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 FIT モジュールの追加方法 を参照してください。

1.2 GPIO FIT モジュールの概要

GPIO FIT モジュールでは、抽象化レイヤを提供し、RX MCU の汎用入出力(GPIO)端子の読み込み、書き込み、および設定を行います。このモジュールの API 関数を使うことで、端子ごとに使用可能な GPIO レジスタを確認する必要がなくなります。RX では、以下の操作を行うためのレジスタが個別に用意されています。端子の方向制御、端子の読み込み、端子の書き込み、内部プルアップの有効設定、出力モードの設定、端子の周辺機能端子としての設定。

1.3 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 1.1API 関数一覧

関数	関数説明
R_GPIO_PortWrite()	1 つのポートに配置された全端子の出力レベルを書き込みます。.
R_GPIO_PortRead()	1 つのポートに配置された全端子の現在のレベルを読み込みます。
R_GPIO_PortDirectionSet()	1 つのポートに配置された全端子を入力または出力に設定します。
R_GPIO_PinWrite()	端子の出力レベルを設定します。
R_GPIO_PinRead()	その時点の端子のレベルを読み出します。
R_GPIO_PinDirectionSet()	端子の方向（入力／出力）を設定します。
R_GPIO_PinControl()	端子の設定を変更します（例: 内部プルアップ、オープンドレイン）
R_GPIO_GetVersion()	本モジュールのバージョン番号を返します。

1.4 制限事項

小ピンパッケージの MCU では、I/O ポートを多重化するポート切り替え機能を持つものがあり、これによって、端子を共用できます。本モジュールでは、ポートの切り替え機能はサポートしていませんが、ユーザアプリケーションで独自に切り替えることは可能です。

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。

- GPIO

2.2 ソフトウェアの要求

このドライバは以下の 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 (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「6.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4 使用する割り込みベクタ

なし

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_gpio_rx_if.h に記載しています。

2.6 整数型

このドライバは 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_gpio_rx_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。

コンフィギュレーションオプション（r_gpio_rx_config.h）	
GPIO_CFG_PARAM_CHECKING_ENABLE - デフォルト値 = “BSP_CFG_PARAM_CHECKING_ENABLE”	= 1：ビルド時にパラメータチェック処理をコードに含めます。 = 0：ビルド時にパラメータチェック処理をコードから省略します。 = BSP_CFG_PARAM_CHECKING_ENABLE（デフォルト）： システムのデフォルト設定を使用します。 注: ビルド時にパラメータチェックのコードを省略することで、コードサイズを小さくすることができます。

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。

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 および RAM のコードサイズ

デバイス	使用メモリ		備考
	パラメータチェックあり	パラメータチェックなし	
RX110	ROM: 511 バイト（コード）	ROM: 355 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX111、RX113	ROM: 514 バイト（コード）	ROM: 355 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX130、RX230	ROM: 613 バイト（コード）	ROM: 441 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX210	ROM: 621 バイト（コード）	ROM: 428 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX231 RX64M、RX71M	ROM: 613 バイト（コード）	ROM: 428 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX23T、 RX24T、RX24U	ROM: 597 バイト（コード）	ROM: 441 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX65N	ROM: 707 バイト（コード）	ROM: 495 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	
RX66T	ROM: 696 バイト（コード）	ROM: 462 バイト（コード）	
	RAM: 0 バイト	RAM: 0 バイト	

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_gpio_rx_if.h に記載されています。

2.9.1 ポート

MCU で使用できるポートを定義する列挙型です。この列挙型は、MCU グループとパッケージごとに用意されており、各 MCU グループのフォルダにあります。例えば、RX111 で使用できるポートを確認するには、*src/targets/rx111/r_gpio_rx111.h* を参照してください。以下に RX111 を使用した場合の例を示します。MCU が異なれば、ポートの列挙型の内容も異なります。

```
#if   (BSP_PACKAGE_PINS == 64)
typedef enum
{
    GPIO_PORT_0 = 0x0000,
    GPIO_PORT_1 = 0x0100,
    GPIO_PORT_2 = 0x0200,
    GPIO_PORT_3 = 0x0300,
    GPIO_PORT_4 = 0x0400,
    GPIO_PORT_5 = 0x0500,
    GPIO_PORT_A = 0x0A00,
    GPIO_PORT_B = 0x0B00,
    GPIO_PORT_C = 0x0C00,
    GPIO_PORT_E = 0x0E00,
```

```
    GPIO_PORT_H = 0x1100,
    GPIO_PORT_J = 0x1200,
} gpio_port_t;
```

2.9.2 端子

対象の MCU に対して、利用可能な GPIO 端子を定義する列挙型です。この列挙型は、MCU グループとパッケージごとに用意されており、各 MCU グループのフォルダにあります。例えば、RX111 で使用できる GPIO 端子を確認するには、*src/targets/rx111/r_gpio_rx111.h* を参照してください。以下は RX111 の例です。本列挙型の GPIO 端子は、r_bsp から自動的に取得される BSP_PACKAGE_PINS マクロによって制御されます。

```
#if   (BSP_PACKAGE_PINS == 64)
typedef enum
{
    GPIO_PORT_0_PIN_3 = 0x0003,
    GPIO_PORT_0_PIN_5 = 0x0005,
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_0 = 0x0300,
    GPIO_PORT_3_PIN_1 = 0x0301,
    GPIO_PORT_3_PIN_2 = 0x0302,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_0 = 0x0400,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_3 = 0x0403,
    GPIO_PORT_4_PIN_4 = 0x0404,
    GPIO_PORT_4_PIN_6 = 0x0406,
    GPIO_PORT_5_PIN_4 = 0x0504,
    GPIO_PORT_5_PIN_5 = 0x0505,
    GPIO_PORT_A_PIN_0 = 0x0A00,
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_1 = 0x0B01,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_B_PIN_5 = 0x0B05,
    GPIO_PORT_B_PIN_6 = 0x0B06,
    GPIO_PORT_B_PIN_7 = 0x0B07,
    GPIO_PORT_C_PIN_0 = 0x0C00,
    GPIO_PORT_C_PIN_1 = 0x0C01,
    GPIO_PORT_C_PIN_2 = 0x0C02,
    GPIO_PORT_C_PIN_3 = 0x0C03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_C_PIN_5 = 0x0C05,
    GPIO_PORT_C_PIN_6 = 0x0C06,
    GPIO_PORT_C_PIN_7 = 0x0C07,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_E_PIN_5 = 0x0E05,
    GPIO_PORT_E_PIN_6 = 0x0E06,
```

```
    GPIO_PORT_E_PIN_7 = 0x0E07,
    GPIO_PORT_H_PIN_7 = 0x1107,
    GPIO_PORT_J_PIN_6 = 0x1206,
    GPIO_PORT_J_PIN_7 = 0x1207,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 48)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_0 = 0x0400,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_6 = 0x0406,
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_1 = 0x0B01,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_B_PIN_5 = 0x0B05,
    GPIO_PORT_C_PIN_0 = 0x0C00,
    GPIO_PORT_C_PIN_1 = 0x0C01,
    GPIO_PORT_C_PIN_2 = 0x0C02,
    GPIO_PORT_C_PIN_3 = 0x0C03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_C_PIN_5 = 0x0C05,
    GPIO_PORT_C_PIN_6 = 0x0C06,
    GPIO_PORT_C_PIN_7 = 0x0C07,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_E_PIN_7 = 0x0E07,
    GPIO_PORT_H_PIN_7 = 0x1107,
    GPIO_PORT_J_PIN_6 = 0x1206,
    GPIO_PORT_J_PIN_7 = 0x1207,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 40)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_6 = 0x0206,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_2 = 0x0302,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_4_PIN_6 = 0x0406,
```

```
    GPIO_PORT_A_PIN_1 = 0x0A01,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_J_PIN_6 = 0x1306,
    GPIO_PORT_J_PIN_7 = 0x1307,
} gpio_port_pin_t;

#elif (BSP_PACKAGE_PINS == 36)
typedef enum
{
    GPIO_PORT_1_PIN_4 = 0x0104,
    GPIO_PORT_1_PIN_5 = 0x0105,
    GPIO_PORT_1_PIN_6 = 0x0106,
    GPIO_PORT_1_PIN_7 = 0x0107,
    GPIO_PORT_2_PIN_7 = 0x0207,
    GPIO_PORT_3_PIN_5 = 0x0305,
    GPIO_PORT_4_PIN_1 = 0x0401,
    GPIO_PORT_4_PIN_2 = 0x0402,
    GPIO_PORT_A_PIN_3 = 0x0A03,
    GPIO_PORT_A_PIN_4 = 0x0A04,
    GPIO_PORT_A_PIN_6 = 0x0A06,
    GPIO_PORT_B_PIN_0 = 0x0B00,
    GPIO_PORT_B_PIN_3 = 0x0B03,
    GPIO_PORT_C_PIN_4 = 0x0C04,
    GPIO_PORT_E_PIN_0 = 0x0E00,
    GPIO_PORT_E_PIN_1 = 0x0E01,
    GPIO_PORT_E_PIN_2 = 0x0E02,
    GPIO_PORT_E_PIN_3 = 0x0E03,
    GPIO_PORT_E_PIN_4 = 0x0E04,
    GPIO_PORT_J_PIN_6 = 0x1306,
    GPIO_PORT_J_PIN_7 = 0x1307
} gpio_port_pin_t;
#endif
```

2.9.3 ポート端子のマスク

この列挙型は MCU グループとパッケージごとに用意され、MCU のポート端子のマスクを定義します。ポート単位の書き込み、または読み込みを行う際に、ユーザが任意で適用できるビットマスクで、ポート単位で可能な動作を確認するために使用します。有効な I/O 端子に対応するポートの各ビットは‘1’に設定され、存在しない端子に対応するビットは‘0’に設定されます。ポート単位での書き込み関数呼び出し時、GPIO FIT モジュールでは、パフォーマンスを考慮して、有効な端子設定の確認は行いません。ユーザアプリケーションで、有効な端子のみに書き込みが実行されるようにしてください。また、本マスクの使用は任意です。以下に RX111 MCU を使用した場合の例を示します。

```
#if (BSP_PACKAGE_PINS == 64)
/* この列挙型には、MCU のポートに配置された各 GPIO 端子のビットマスクが記載されます。*/
typedef enum
{
GPIO_PORT0_PIN_MASK = 0x28,   /* Available pins:P03,P05                  */
GPIO_PORT1_PIN_MASK = 0xF0,   /* Available pins:P14, P15, P16,P17        */
GPIO_PORT2_PIN_MASK = 0xC0,   /* Available pins:P26,P27                  */
```

```
GPIO_PORT3_PIN_MASK = 0x27,   /* Available pins:P30 to P32, P35          */
GPIO_PORT4_PIN_MASK = 0x5F,   /* Available pins:P40 to P44, P46          */
GPIO_PORT5_PIN_MASK = 0x30,   /* Available pins:P54, P55                 */
GPIO_PORTA_PIN_MASK = 0x5B,   /* Available pins:PA0, PA1, PA3, PA4, PA6  */
GPIO_PORTB_PIN_MASK = 0xEB,   /* Available pins:PB0, PB1, PB3, PB5 to PB7*/
GPIO_PORTC_PIN_MASK = 0xFF,   /* Available pins:PC0 to PC7               */
GPIO_PORTE_PIN_MASK = 0xFF,   /* Available pins:PE0 to PE7               */
GPIO_PORTH_PIN_MASK = 0x80,   /* Available pins:PH7                      */
GPIO_PORTJ_PIN_MASK = 0xC0    /* Available pins:PJ6, PJ7                 */
} gpio_pin_bit_mask_t;
```

2.9.4 端子レベル

この列挙型は、GPIO 端子を読み込んだときに返される値を定義します。

```
/* 各端子のレベル */
typedef enum
{
    GPIO_LEVEL_LOW = 0,
    GPIO_LEVEL_HIGH
} gpio_level_t;
```

2.9.5 端子の方向

この列挙型は、GPIO 端子の方向設定に使用されるオプションを定義します。

```
/* R_GPIO_PortDirectionSet()、R_GPIO_PinDirectionSet()関数で
   使用できるオプション */
typedef enum
{
    GPIO_DIRECTION_INPUT = 0,
    GPIO_DIRECTION_OUTPUT
} gpio_dir_t;
```

2.9.6 制御コマンド

この列挙型は、R_GPIO_PinControl()関数に送信されるコマンドを定義します。

```
/* R_GPIO_PinControl()関数で使用できるコマンド。
   選択した MCU により、内容は異なります。 */
typedef enum
{
    GPIO_CMD_OUT_CMOS = 0,
    GPIO_CMD_OUT_OPEN_DRAIN_N_CHAN,
    GPIO_CMD_OUT_OPEN_DRAIN_P_CHAN,
    GPIO_CMD_IN_PULL_UP_DISABLE,
    GPIO_CMD_IN_PULL_UP_ENABLE,
    GPIO_CMD_ASSIGN_TO_PERIPHERAL,
    GPIO_CMD_ASSIGN_TO_GPIO,
    GPIO_CMD_DSCR_DISABLE,
    GPIO_CMD_DSCR_ENABLE,
    GPIO_CMD_DSCR2_DISABLE,
    GPIO_CMD_DSCR2_ENABLE
} gpio_cmd_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_gpio_rx_if.h で記載されています。

以下に R_GPIO_PinControl()関数で使用される戻り値を示します。

```
/* 関数の戻り値 */
typedef enum
{
    GPIO_SUCCESS = 0,
    GPIO_ERR_INVALID_MODE,  // 指定されたモードは、この端子に適用できません。
    GPIO_ERR_INVALID_CMD    // 入力されたコマンドはサポートされていません。
} gpio_err_t;
```

2.11 コールバック関数

なし

2.12 FIT 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+で開発中プロジェクトに FIT モジュールを追加
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。

2.13 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

```
while 文の例：
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized.*/
}

for 文の例：
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while 文の例：
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API 関数

R_GPIO_PortWrite

1 つのポートに配置された全端子のレベルを書き込みます。

Format

```
void R_GPIO_PortWrite (
    gpio_port_t     port,
    uint8_t         value
)
```

Parameters

gpio_port_t port

書き込みするポート。セクション「2.9.1 Ports」参照。

uint8_t value

ポートに書き込む値。この引数の各ビットはポートの端子に対応しています（例: ビット 0 の値は、指定されたポートの端子 0 に書き込まれます）。

Return Values

なし

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

入力値は、指定されたポートに書き込まれます。引数“value”の各ビットは、ポートの端子に対応しています。例えば、ビット 7 の書き込み値は端子 7、ビット 6 の書き込み値は端子 6 に対応します。

Reentrant

この関数は再入可能（リエントラント）です。

Example

```
/* ポート 5 に“0xAA”を書き込む */
R_GPIO_PortWrite(GPIO_PORT_5, 0xAA);
```

Special Notes:

本関数では、ポート単位での書き込み関数呼び出し時、パフォーマンスを考慮して、存在しない端子の確認は行いません。ユーザアプリケーションで、有効な端子のみに書き込みが実行されるようにしてください。

R_GPIO_PortRead

この関数は 1 つのポートに配置された全端子のレベルを読み出します。

Format

```
uint8_t     R_GPIO_PortRead (
            gpio_port_t     port
)
```

Parameters

gpio_port_t port

読み出すポート。セクション「2.9.1 Ports」参照。

Return Values

ポートの値

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

指定されたポートを読み込んで、すべての端子のレベルを返します。戻り値の各ビットはポートの端子に対応しています。例えば、読んだ値のビット 7 は端子 7 に、ビット 6 は端子 6 に対応しています。

Reentrant

この関数は異なるポートに対して再入可能（リエントラント）です。

Example

```
uint8_t    port_5_value;

/* ポート 5 を読み込む */
port_5_value = R_GPIO_PortRead(GPIO_PORT_5);
```

Special Notes:

なし

R_GPIO_PortDirectionSet

この関数は 1 つのポートに配置された全端子を入力または出力に設定します。

Format

```
Void    R_GPIO_PortDirectionSet (
        gpio_port_t port,
        gpio_dir_t  dir,
        uint8_t             mask
)
```

Parameters

gpio_port_t port
使用するポート。セクション「2.9.1 Ports」参照。

gpio_dir_t dir
使用する方向。セクション「2.9.5 Pin Direction」参照。

uint8_t mask
変更する端子のマスク。1 = 方向を設定、0 = 変更なし

Return Values

なし

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

1 つのポートの全端子の方向を 1 度で入力、または出力に設定できます。引数“mask”の各ビットはポートの端子に対応しています。例えば、“mask”のビット 7 は端子 7 に、ビット 6 は端子 6 に対応しています。ビットが‘1’に設定された場合、対応する端子の方向は、引数“dir”に従って、入力または出力に変更されます。ビットが‘0’に設定された場合、端子の方向は変更されません。

Reentrant

この関数は異なるポートに対して再入可能（リエントラント）です。

Example

```
/* ポート A の端子 0、1、5 を入力に設定 */
R_GPIO_PortDirectionSet(GPIO_PORT_A, GPIO_DIRECTION_INPUT, 0x23);

/* ポート A の端子 2、3、4、6、7 を出力に設定*/
R_GPIO_PortDirectionSet(GPIO_PORT_A, GPIO_DIRECTION_OUTPUT, 0xDC);
```

Special Notes:

本関数では、入力のプルアップ抵抗、またはオープンドレイン出力などの特殊なモードは設定できません。これらのモードは、R_GPIO_PinControl()関数を使って有効にできます。

R_GPIO_PinWrite

この関数は端子のレベルを設定します。

Format

```
void    R_GPIO_PinWrite (
            gpio_port_pin_t         pin,
            gpio_level_t            level
)
```

Parameters

gpio_port_pin_t pin

使用する端子。セクション「2.9.2 Pins」参照。

gpio_level_t level

端子に設定するレベル。

Return Values

なし

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

端子は High (‘1’)または Low (‘0’)に設定できます。

Reentrant

この関数は異なる端子に対して再入可能（リエントラント）です。

Example

```
/* ポート E の端子 0 を High に設定 */
R_GPIO_PinWrite(GPIO_PORT_E_PIN_0, GPIO_LEVEL_HIGH);

/* ポート 3 の端子 2 を Low に設定*/
R_GPIO_PinWrite(GPIO_PORT_3_PIN_2, GPIO_LEVEL_LOW);
```

Special Notes:

なし

R_GPIO_PinRead

この関数は端子のレベルを読み込みます。

Format

```
gpio_level_t      R_GPIO_PinRead (
        gpio_port_pin_t             pin
)
```

Parameters

gpio_port_pin_t pin

使用する端子。セクション「2.9.2 Pins」参照。

Return Values

指定された端子のレベル

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

指定された端子を読み込み、そのレベルを返します。

Reentrant

この関数は異なる端子に対して再入可能（リエントラント）です。

Example

```
/* ポート 5 の端子 4 のレベルを確認 */
if (R_GPIO_PinRead(GPIO_PORT_5_PIN_4) == GPIO_LEVEL_HIGH)
{
    …
}
else
{
    …
}
```

Special Notes:

なし

R_GPIO_PinDirectionSet

この関数は端子の方向（入力／出力）を設定します。

Format

```
void    R_GPIO_PinDirectionSet (
            gpio_port_pin_t     pin,
            gpio_dir_t          dir
)
```

Parameters

gpio_port_pin_t pin

使用する端子。セクション「2.9.2 Pins」参照。

gpio_dir_t dir

端子に設定する方向。セクション「2.9.5 Pin Direction」参照。

Return Values

なし

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

本関数は、端子を入力または出力に設定します。オープンドレイン出力や内部プルアップなどの設定を有効にする場合は、R_GPIO_PinControl()関数をご覧ください。

Reentrant

この関数は異なる端子に対して再入可能（リエントラント）です。

Example

```
/* ポート E の端子 0 を出力に設定 */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_0, GPIO_DIRECTION_OUTPUT);

/* ポート 3 の端子 2 を入力に設定*/
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_2, GPIO_DIRECTION_INPUT);
```

Special Notes:

なし

R_GPIO_PinControl

この関数は様々な端子の設定を制御します。

Format

```
gpio_err_t      R_GPIO_PinControl (
        gpio_port_pin_t         pin,
        gpio_cmd_t              cmd
)
```

Parameters

gpio_port_pin_t pin

使用する端子。セクション「2.9.2 Pins」参照。

gpio_cmd_t cmd

端子に対して実行するコマンド。 使用可能なコマンドについてはセクション「2.9.6 Control Commands」をご覧ください。

Return Values

```
[GPIO_SUCCESS]          /*成功; コマンドで指定されたとおりに端子が変更されました。      */
[GPIO_ERR_INVALID_MODE] /*エラー; 端子は指定されたオプションをサポートしていません。  */
[GPIO_ERR_INVALID_CMD]  /*エラー; 入力されたコマンドをサポートしていません。          */
```

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

方向や出力レベルの他に、MCU に応じて、端子の様々な設定が行えます。例えば、オープンドレイン出力や内部プルアップを有効にしたり、駆動能力のレベルを変更することができます。これらの機能は MCU ごとに異なりますので、本関数で選択できるオプションもそれに応じて異なります。

Reentrant

この関数は異なる端子に対して再入可能（リエントラント）です。

Example

```
gpio_err_t gpio_err;

/* ポート E の端子 0 を CMOS 出力に設定(デフォルト) */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_0, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_OUT_CMOS);

/* ポート E の端子 0 を High 出力に設定*/
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_DSCR_ENABLE);

/* ポート E の端子 0 を高速インタフェース用高駆動出力に設定*/
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_0, GPIO_CMD_DSCR2_ENABLE);
```

```
/* ポート E の端子 1 を P チャネル オープンドレイン出力に設定 */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_1, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_1,GPIO_CMD_OUT_OPEN_DRAIN_P_CHAN);

/* ポート E の端子 2 を N チャネル オープンドレイン出力に設定 */
R_GPIO_PinDirectionSet(GPIO_PORT_E_PIN_2, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_E_PIN_2,GPIO_CMD_OUT_OPEN_DRAIN_N_CHAN);

/* ポート 3 の端子 2 をプルアップなしの入力に設定（デフォルト）*/
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_2, GPIO_DIRECTION_INPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_3_PIN_2, GPIO_CMD_IN_PULL_UP_DISABLE);

/* ポート 3 の端子 3 をプルアップありの入力に設定*/
R_GPIO_PinDirectionSet(GPIO_PORT_3_PIN_3, GPIO_DIRECTION_INPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_3_PIN_3, GPIO_CMD_IN_PULL_UP_ENABLE);

/* ポート 2 の端子 6 を SCI の TXD1 として使用 */
R_GPIO_PinDirectionSet(GPIO_PORT_2_PIN_6, GPIO_DIRECTION_OUTPUT);
gpio_err |= R_GPIO_PinControl(GPIO_PORT_2_PIN_6, GPIO_CMD_ASSIGN_TO_PERIPHERAL);

/* ポート 5 の端子 4 を GPIO として使用*/
gpio_err |= R_GPIO_PinControl(GPIO_PORT_5_PIN_4, GPIO_CMD_ASSIGN_TO_GPIO);

/* GPIO_SUCCESS が 0 に設定されているので、gpio_err が 0 でなければ、いずれかの処理でエラーが
発生しています。必要に応じて、関数呼び出し後、毎回 gpio_err を確認できます。*/
if (GPIO_SUCCESS != gpio_err)
{ /* エラー処理 */}
```

R_GPIO_GetVersion

この関数は実行時に本モジュールのバージョンを返します。

Format

uint32_t R_GPIO_GetVersion (void)

Parameters

なし

Return Values

本モジュールのバージョン

Properties

ファイル r_gpio_rx_if.h にプロトタイプ宣言されています。

Description

この関数は本モジュールのバージョンを返します。バージョン番号は符号化され、最上位の 2 バイトがメジャーバージョン番号を、最下位の 2 バイトがマイナーバージョン番号を示しています。例えば、ver. 4.25 の場合、“0x00040019”が返されます。

Reentrant

この関数は再入可能（リエントラント）です。

Example

```
uint32_t cur_version;

/* 実行中の r_gpio_rx API のバージョンを取得 */
cur_version = R_GPIO_GetVersion();

/* 本アプリケーションを使用可能なバージョンであることを確認 */
if (MIN_VERSION > cur_version)
{
    /* お使いの r_gpio_rx バージョンは、本アプリケーションで必要な XXX 機能を
       備えていない旧バージョンです。警告 */
    ….
}
```

Special Notes:

本関数は r_gpio_rx.c のインライン関数となります。

4. 端子設定

GPIO FIT モジュールは端子設定を使用しません。

5. デモプロジェクト

デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。本 FIT モジュールには以下のデモプロジェクトが含まれます。

5.1 gpio_demo_rskrx113

gpio_demo_rskrx113 は、I/O ポートの方向（入力／出力）を設定する方法、および端子を読み込み／書き込みする方法をデモします。コードがコンパイルされ、対象のボードにダウンロードして実行されると、LED2 が 3 回点滅し、デモが実行中であることを示して、SW1 が押下されるのを待ちます。SW1 が押下されている間、LED2 は点灯し、SW1 を離すと、LED2 は消灯します。

5.2 gpio_demo_rskrx231、gpio_demo_rskrx64m、gpio_demo_rskrx71m、gpio_demo_rskrx65n、gpio_demo_rskrx65n_2m

gpio_demo_rskrx231、gpio_demo_rskrx64m、gpio_demo_rskrx71m、gpio_demo_rskrx65n および gpio_demo_rskrx65n_2mでは、gpio_demo_rskrx113と同様のデモに加え、端子の出力をHIGHに設定する方法も含まれます。

また、これらのデモではLED2の代わりにLED3が使用されます。

5.3 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」 >> 「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「終了」をクリックします。

5.4 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

6. 付録

6.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 6.1 動作確認環境 (Rev.2.41)

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 V.7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.01.00
	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.2.41
使用ボード	Renesas Starter Kit for RX66T（型名：RTK50566T0SxxxxxBE） Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2CxxxxxBR） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308CxxxxxBR）

表 6.2 動作確認環境 (Rev.2.40)

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 V.7.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.00.00
	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.2.40
使用ボード	Renesas Starter Kit for RX66T（型名：RTK50566T0SxxxxxBE） Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2CxxxxxBR） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308CxxxxxBR）

表 6.3 動作確認環境 (Rev.2.31)

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 Version 6.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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.2.31
使用ボード	Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2CxxxxxBR） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308CxxxxxBR）

表 6.4 動作確認環境 (Rev.2.30)

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 Version 6.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package 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.2.30
使用ボード	Renesas Starter Kit+ for RX 65N-2MB（型名：RTK50565N2CxxxxxBR） Renesas Starter Kit+ for RX130-512KB（型名：RTK5051308CxxxxxBR）

6.2 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)」
- e^2 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 e^2 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_gpio_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「コンフィグ設定が間違っている場合のエラーメッセージ」エラーが発生します。

A：“r_gpio_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_gpio_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。

7. 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 CC-RX コンパイラ 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。
なし

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	2013.11.15	—	初版
1.20	2014.04.23	1	サポート対象 MCU の一覧を追加。ドキュメント一覧を改訂。
		3	必要な BSP のバージョン一覧を更新。
		3	制限に関する注意を追加：ポートの切り替え機能をサポートしていない。
		4-7	PORT_J に関する説明で、割り当てる値を訂正。
		7	セクション「2.9.3 ポート端子のマスク」を追加。
		11	存在しない端子や有効な端子への書き込みに関する注意を追加。
1.30	2014.06.03	1	サポート対象 MCU の一覧で、RX64M に関する説明を追加。
		3	ツールチェーンのバージョンを更新。
		19	セクション 3.9 の書式を更新。
1.40	2014.11.28	— 5	RX113 グループのサポートを追加。 「コードサイズ」セクションを追加。
1.50	2015.03.06	— 5 10	RX71M グループのサポートを追加。 RX71M に対応するコードサイズを更新。 「制御コマンド」を更新し、DSCR に関する説明を追加。
1.60	2015.06.30	— 5	RX231 グループのサポートを追加。 RX231 に対応するコードサイズを更新。
1.70	2015.09.30	— 5	RX23T グループのサポートを追加。 RX23T に対応するコードサイズを更新。
1.80	2015.10.01	— 5	RX130 グループのサポートを追加。 RX130 に対応するコードサイズを更新。
1.90	2015.12.01	— 1, 10 4 5 20	RX24T グループのサポートを追加。 『ボードサポートパッケージモジュール Firmware Integration Technology』アプリケーションノートのドキュメント番号を変更。 セクション 2 の説明を変更。 RX24T に対応するコードサイズを更新。 「4. デモプロジェクト」を追加。
2.00	2016.01.01	— 5 21	RX230 グループのサポートを追加。 RX230 に対応するコードサイズを更新。 「テクニカルアップデートの対応について」を追加。
2.01	2016.06.15	20	「4. デモプロジェクト」に RSKRX64M を追加。
2.10	2016.10.01	— 1 4 5 10 19 22 23	RX65N グループのサポートを追加。 『ボードサポートパッケージモジュール Firmware Integration Technology』アプリケーションノートのドキュメント番号を変更。 対象デバイスに RX65N グループを追加。 「2.5 対応ツールチェーン」に RX65N を追加。 RX65N. に対応するコードサイズを更新。 「制御コマンド」を更新し、DSCR2 に関する説明を追加。 「3.8 R_GPIO_PinControl」に DSCR2 コマンドを追加。 「5. デモプロジェクト」を追加。 「6 参考ドキュメント」を追加。 「お問い合わせ先」を更新。

Rev.	発行日	改訂内容	
		ページ	ポイント
2.20	2017.02.28		RX24U グループのサポートを追加。 「2.5 対応ツールチェーン」に RXC v2.06.00 を追加。 RX24U. に対応するコードサイズを更新。 RX24T グループに対して以下対応 - P36、P37 の仕様公開に対応 - 64 ピンパッケージに対応
2.30	2017.07.21		RX130-512KB と RX65N-2MB のサポートを追加。 「2.5 対応ツールチェーン」に RXC v2.07.00 を追加。 「2.12 FIT モジュールをプロジェクトに追加する方法」を更新。
2.31	2017.10.31		「4. デモプロジェクト」に RSKRX65N と RSKRX65N-2MB を追加。 「4.4 デモのダウンロード方法」を追加。 「5. 付録」を追加。
2.40	2018.09.28	1	RX66T のサポートを追加。
		6	RX66T に対応するコードサイズを追加
		24	「6.1 動作確認環境」Rev.2.40 に対応する表を追加。
2.41	2018.11.16	—	XML 内にドキュメント番号を追加。
		24	Renesas Starter Kit+ for RX66T の型名を変更。 Rev.2.41 に対応する表を追加。

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、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

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/