RXファミリ

DTCを使用したRSPI通信中に転送データ長を変更する方法

要旨

本アプリケーションノートは RX660 グループを例に、データトランスファコントローラ(以下、DTC)を使用したシリアルペリフェラルインタフェース(以下、RSPI)通信において、通信中に転送データ長を変更する方法について説明します。

対象デバイス

RX ファミリ

動作確認デバイス

RX660 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。

目次

1.　RSPI の転送データ長変更前後の SSL 信号について

1.1　ハードウェア制御による SSL 信号

RSPI 通信中に転送データ長を変更する場合、変更前の通信が終了している必要があります。

RSPI 通信が終了するとハードウェア制御により SSL 信号がネゲートするため、図 1.1に示すように、転送データ長の変更前後で SSL 信号のネゲート、アサートが発生します。

全データ通信中、SSL 信号をアサート状態に保持したい場合、SSL 信号を汎用ポートで制御する必要があります。

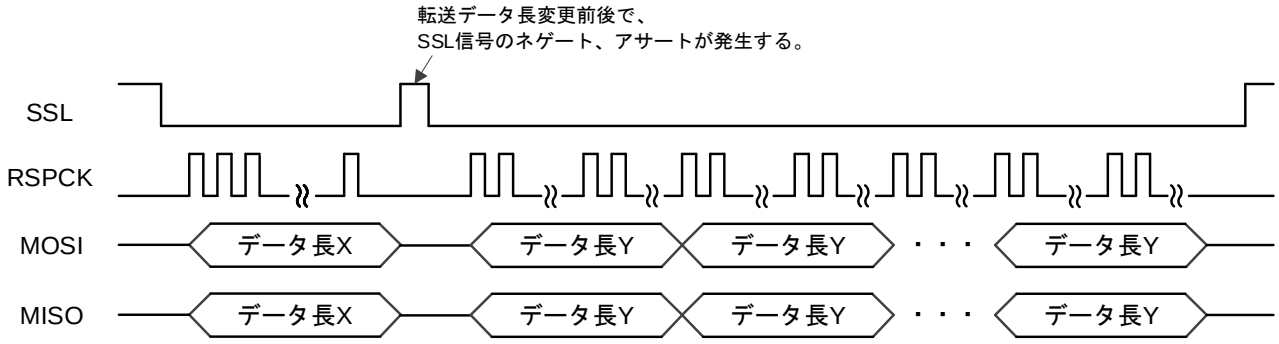


図 1.1　転送データ長変更前後のハードウェア制御による SSL 信号

1.2　汎用ポート制御による SSL 信号の生成

本アプリケーションノートでは、ハードウェア制御による SSL 信号の代わりに、汎用ポートを使用してソフトウェアで SSL 信号を制御します。

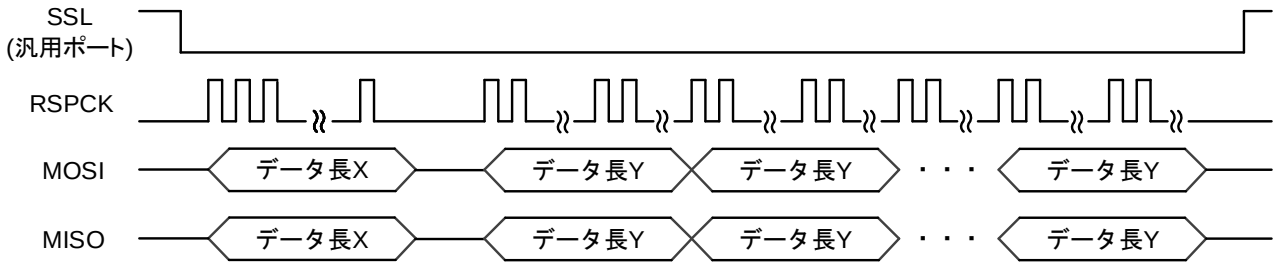


図 1.2　転送データ長変更前後の汎用ポート制御による SSL 信号

2. ハードウェア構成

図 2.1にハードウェア構成を示します。

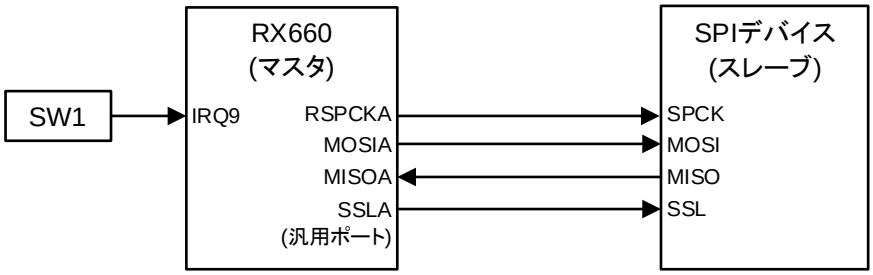


図 2.1 ハードウェア構成

表 2.1にRX660とSPIスレーブデバイスとの接続に使用するRSPI端子を示します。

表 2.1　RX660 と SPI スレーブデバイスとの接続に使用する RSPI 端子

端子名	入出力	使用ポート	機能
RSPCKA	出力	PA5	クロック入出力
MOSIA	出力	PA6	マスタ送出データ入出力
MISOA	入力	PA7	スレーブ送出データ入出力
SSLA	出力	PA2	汎用ポート制御によるスレーブセレクト出力

本アプリケーションノートでは、Renesas Starter Kit+ for RX660(以下、RSK)ボード搭載の SW1 を使用して RSPI 通信を開始します。

表 2.2にSW1入力に割り当てている外部端子割り込みを示します。

表 2.2　SW1 入力に割り当てている外部端子割り込み

端子名	使用ポート	機能
IRQ9	P91	SW1 押下を検出し、RSPI 通信を開始する

3. 動作確認条件

表 3.1　動作確認条件

項目	内容
使用マイコン	R5F56609HDFB (RX660 グループ)
動作周波数	• メインクロック: 24MHz • PLL: 240MHz (メインクロック 1 分周 10 逓倍) • システムクロック(ICLK): 120MHz (PLL 2 分周) • 周辺モジュールクロック A(PCLKA): 120MHz(PLL2 分周) • 周辺モジュールクロック B (PCLKB): 60MHz (PLL 4 分周)
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 e^2 studio Version 2024-01 (24.1.0)
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 V.3.06.00
	コンパイルオプション -lang = c99
iodefine.h のバージョン	Version 1.00
エンディアン	リトルエンディアン
動作モード	シングルチップモード
プロセッサモード	スーパバイザモード
サンプルプログラムのバージョン	Version 1.00
使用ボード	Renesas Starter Kit for RX660 (型名：RTK556609xxxxxxxxx)

4.　ソフトウェア説明

本アプリケーションノートでは、RSK 搭載の SW1 押下後、転送データ長 24 ビットで 1 フレームの通信を行い、その後転送データ長を 16 ビットに変更して 8 フレームの通信を行います。

SW1 は IRQ9 に接続されています。表 4.1にIRQ9(SW1押下検出用)の設定を示します。

表 4.1　IRQ9(SW1 押下検出用)の設定

項目	設定内容
検出タイプ	立ち下がりエッジ
デジタルフィルタ設定	PCLK/64

転送データ長 24 ビットと転送データ長 16 ビットの通信時の RSPI 設定を表 4.2と表 4.3に示します。

表 4.2　転送データ長 24 ビット通信時の RSPI 設定

項目	設定内容
RSPCK クロック	125k Hz
ビット長	24 ビット
フレーム数	1 フレーム
フォーマット	MSB ファースト
RSPCK 位相	奇数エッジでデータ変化、偶数エッジでデータサンプル
RSPCK 極性	アイドル時の RSPCK が Low
SSL 極性	アクティブ Low
SSL ネゲート動作	転送終了後から次アクセス開始まで SSL 信号レベルを保持
RSLCK 遅延	1RSPCK
SSL ネゲート遅延	1RSPCK
次アクセス遅延	1RSPCK+2PCLK

表 4.3　転送データ長 16 ビット通信時の RSPI 設定

項目	設定内容
RSPCK クロック	125k Hz
ビット長	16 ビット
フレーム数	1 フレーム
フォーマット	MSB ファースト
RSPCK 位相	奇数エッジでデータ変化、偶数エッジでデータサンプル
RSPCK 極性	アイドル時の RSPCK が Low
SSL 極性	アクティブ Low
SSL ネゲート動作	転送終了後から次アクセス開始まで SSL 信号レベルを保持
RSLCK 遅延	1RSPCK
SSL ネゲート遅延	1RSPCK
次アクセス遅延	1RSPCK+2PCLK

表 4.4に使用するRSPI割り込みを示します。

表 4.4　使用する RSPI 割り込み

割り込み	機能
SPTI0	送信バッファエンプティ割り込み
SPRI0	受信データフル割り込み
SPEI0	エラー割り込み
SPCI0	通信完了割り込み

本アプリケーションノートでは、データの送信/受信どちらにも DTC を使用しています。

転送データ長、送信または受信により DTC 設定が異なるため、以下のように DTC 設定を分類します。

DTC 転送 A：データ送信用 DTC 設定(転送データ長 24 ビット)

DTC 転送 B：データ受信用 DTC 設定(転送データ長 24 ビット)

DTC 転送 C：データ送信用 DTC 設定(転送データ長 16 ビット)

DTC 転送 D：データ受信用 DTC 設定(転送データ長 16 ビット)

表 4.5～表 4.8に各 DTC 設定を示します。

表 4.5　DTC 転送 A：データ送信用 DTC 設定(転送データ長 24 ビット)

項目	内容
起動要因	SPTI0 割り込み
転送モード	ノーマル転送
転送データサイズ	32 ビット
割り込み設定	指定した回数のデータ転送が終了したとき、CPU への割り込み要求が発生
転送元アドレス	アドレス固定
転送先アドレス	アドレス固定
転送元レジスタ(SAR)	0x3000(RAM 領域のアドレス)
転送先レジスタ(DAR)	SPDR レジスタのアドレス
転送カウントレジスタ A(CRA)	0x0001
転送カウントレジスタ B(CRB)	0x0000

表 4.6　DTC 転送 B：データ受信用 DTC 設定(転送データ長 24 ビット)

項目	内容
起動要因	SPRI0 割り込み
転送モード	ノーマル転送
転送データサイズ	32 ビット
割り込み設定	指定した回数のデータ転送が終了したとき、CPU への割り込み要求が発生
転送元アドレス	アドレス固定
転送先アドレス	アドレス固定
転送元レジスタ(SAR)	SPDR レジスタのアドレス
転送先レジスタ(DAR)	0x2000(RAM 領域のアドレス)
転送カウントレジスタ A(CRA)	0x0001
転送カウントレジスタ B(CRB)	0x0000

表 4.7　DTC 転送 C：データ送信用 DTC 設定(転送データ長 16 ビット)

項目	内容
起動要因	SPTI0 割り込み
転送モード	ノーマル転送
転送データサイズ	16 ビット
割り込み設定	指定した回数のデータ転送が終了したとき、CPU への割り込み要求が発生
転送元アドレス	アドレス固定
転送先アドレス	アドレス固定
転送元レジスタ(SAR)	g_w16_data のアドレス
転送先レジスタ(DAR)	SPDR レジスタのアドレス
転送カウントレジスタ A(CRA)	0x0008
転送カウントレジスタ B(CRB)	0x0000

表 4.8　DTC 転送 D：データ受信用 DTC 設定(転送データ長 16 ビット)

項目	内容
起動要因	SPRI0 割り込み
転送モード	ノーマル転送
転送データサイズ	16 ビット
割り込み設定	指定した回数のデータ転送が終了したとき、CPU への割り込み要求が発生
転送元アドレス	アドレス固定
転送先アドレス	インクリメント
転送元レジスタ(SAR)	SPDR レジスタのアドレス
転送先レジスタ(DAR)	g_r16_data[0]のアドレス
転送カウントレジスタ A(CRA)	0x0008
転送カウントレジスタ B(CRB)	0x0000

4.1　動作説明

4.1.1　転送データ長 24 ビットの通信

SW1 押下後、IRQ9 割り込み要求が発生します。IRQ9 割り込み処理では、汎用ポート(PA2)制御の SSL 信号を Low レベルに変更し、転送データ長 24 ビットの通信を開始します。

転送データ長 24 ビットの通信では 1 フレームの送受信を行います。

図 4.1に転送データ長24ビットの通信時のタイミング図を示します。

<データ送信>

SPTI0 割り込みを起動要因として表 4.5の DTC 転送 A が実行され、24 ビットのデータを SPDR レジスタへ書き込みます(DTC 転送は 32 ビットですが、下位 24 ビットが有効データとなります)。

DTC 転送 A の実行後に CPU に対して SPTI0 割り込み要求が発生します。

SPTI0 割り込み処理では SPTI0 割り込みを禁止します(SPCR.SPTIE=0)。

<データ受信>

SPRI0 割り込みを起動要因として表 4.6の DTC 転送 B が実行され、24 ビットのデータを SPDR レジスタから読み出します(DTC 転送は 32 ビットですが、下位 24 ビットが有効データとなります)。

DTC 転送 B の実行後に CPU に対して SPRI0 割り込み要求が発生します。

SPRI0 割り込み処理では SPCI0 割り込みを許可します(SPCR3.SPCIE=1)。

<通信完了処理>

SPCI0 割り込み処理では、転送データ長を 16 ビットに変更し、16 ビット送信用として表 4.7の DTC 転送 C、16 ビット受信用として表 4.8の DTC 転送 D の内容に DTC 設定を変更します。その後、転送データ長 16 ビットの通信を開始します。

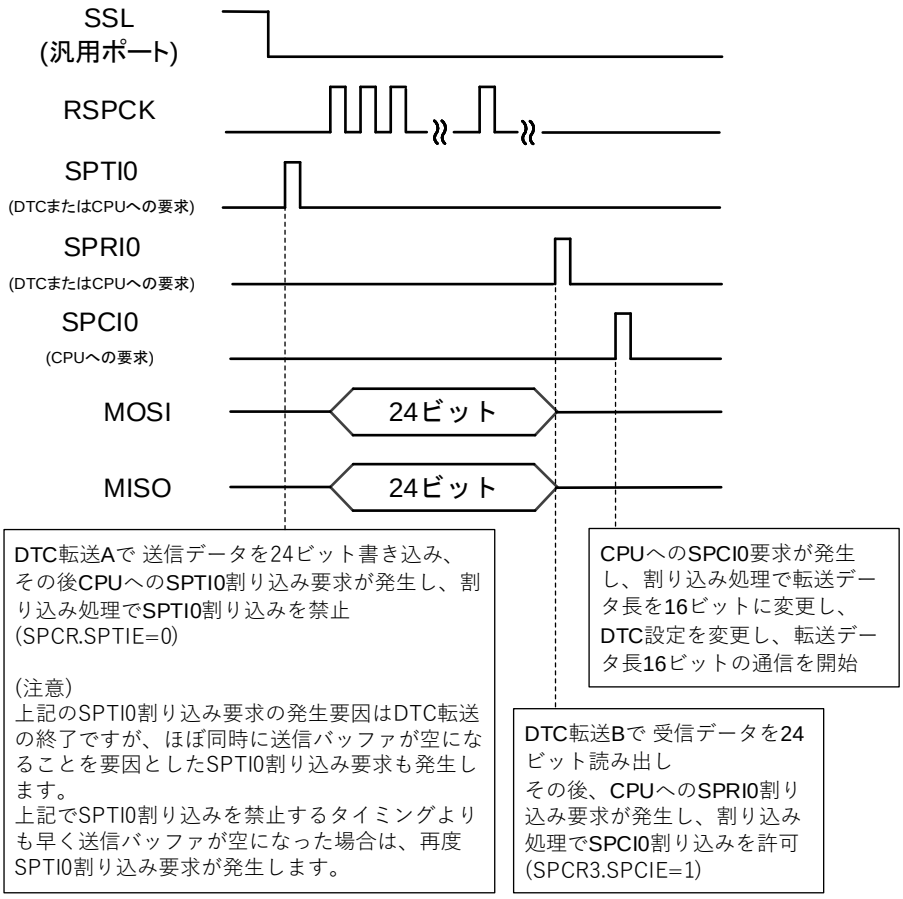


図 4.1　転送データ長 24 ビットの通信時のタイミング図

4.1.2　転送データ長 16 ビットの通信

転送データ長 24 ビットの通信に続いて転送データ長 16 ビットの通信を開始します。

転送データ長 16 ビットの通信では 8 フレームの送受信を行います。

図 4.2に転送データ長16ビットの送受信時のタイミング図を示します。

<データ送信>

SPTI0 割り込みを起動要因として表 4.7の DTC 転送 C が実行され、16 ビットデータを SPDR レジスタへ書き込みます。
SPTI0 割り込み要求に応じて 8 フレーム分の DTC 転送を実行すると、CPU に対して SPTI0 割り込み要求が発生します。

SPTI0 割り込み処理では SPTI0 割り込みを禁止します(SPCR.SPTIE=0)。

<データ受信>

SPRI0 割り込みを起動要因として表 4.8の DTC 転送 D が実行され、16 ビットデータを SPDR レジスタから読み出します。

SPRI0 割り込み要求に応じて 8 フレーム分の DTC 転送を実行すると、CPU に対して SPRI0 割り込み要求が発生します。

SPRI0 割り込み処理では SPCI0 割り込みを許可します(SPCR3.SPCIE=1)。

<通信完了処理>

SPCI0 割り込み処理では、汎用ポート(PA2)制御の SSL 信号を High レベルに変更し、転送データ長を 24 ビットに変更し、24 ビット送信用として表 4.5の DTC 転送 A、24 ビット受信用として表 4.6の DTC 転送 B の内容に DTC 設定を変更します。

その後、SW1 が押下されると、「4.1.1 転送データ長24ビットの通信」で説明した転送データ長 24 ビットの通信を開始します。

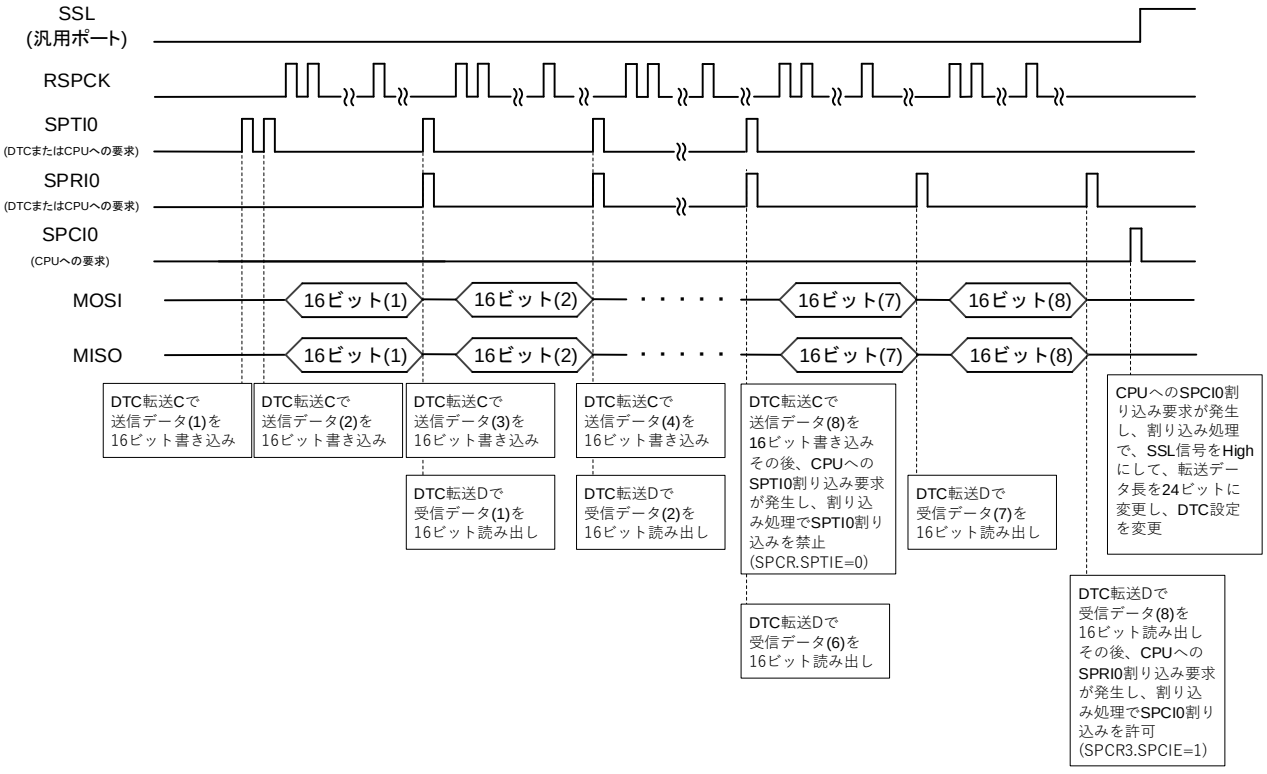


図 4.2　転送データ長 16 ビットの送受信時のタイミング図

4.2　使用する Firmware Integration Technology(以下、FIT)モジュールとコード生成のコンポーネント

表 4.9に使用するFITモジュールとコード生成のコンポーネントを示します。

表 4.9　使用する FIT モジュールとコード生成のコンポーネント

コンポーネント	分類	用途
Board Support Packages(以下、BSP)	FIT モジュール	リセットから main()関数までの全てのコードを提供
割り込みコントローラ	コード生成	ICU の設定
データトランスファコントローラ	コード生成	DTC の設定
SPI 動作モード(4 線式)	コード生成	RSPI の設定

4.2.1　FIT モジュールコンポーネントのスマート・コンフィグレータ(以下、SC)設定

本アプリケーションノートでは、新規プロジェクト作成時に自動生成される BSP モジュールを使用していますが、BSP の SC 設定は変更していません。

主要となるクロック設定のみを表 4.10に示します。

表 4.10　BSP モジュールのクロック設定

項目	設定
システムクロック設定	クロックソース：PLL 回路出力 240MHz システムクロック(ICLK)：x1/2　120 (MHz) 周辺モジュールクロック(PCLKA)：x1/2　120 (MHz) 周辺モジュールクロック(PCLKB)：x1/4　60 (MHz) 周辺モジュールクロック(PCLKD)：x1/4　60 (MHz) バスクロック(BCLK)：x1/4　60 (MHz) FlashIF クロック(FCLK)：x1/4　60 (MHz)
サブクロック発振器設定	動作 (デフォルト設定。サブクロックは使用しません。)
HOCO クロック設定	停止
LOCO クロック設定	停止
IWDT 専用クロック設定	停止

4.2.2　コード生成コンポーネントの SC 設定

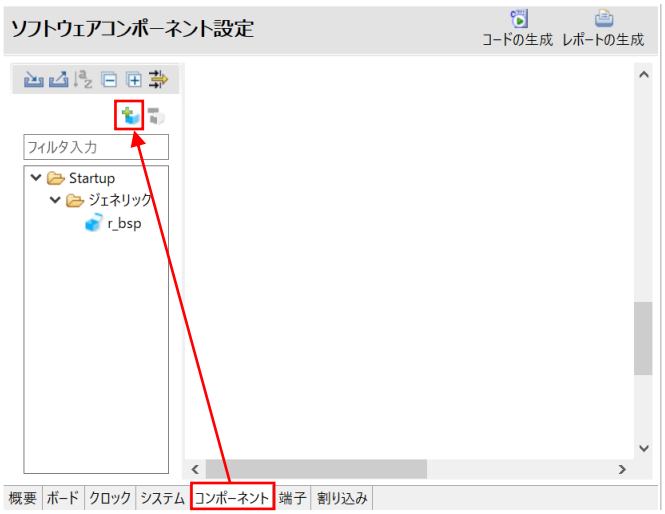
電源投入後、最初の通信は、転送データ長 24 ビットの通信です。

このため、本項目で説明する SC 設定は、転送データ長 24 ビットの通信用となります。

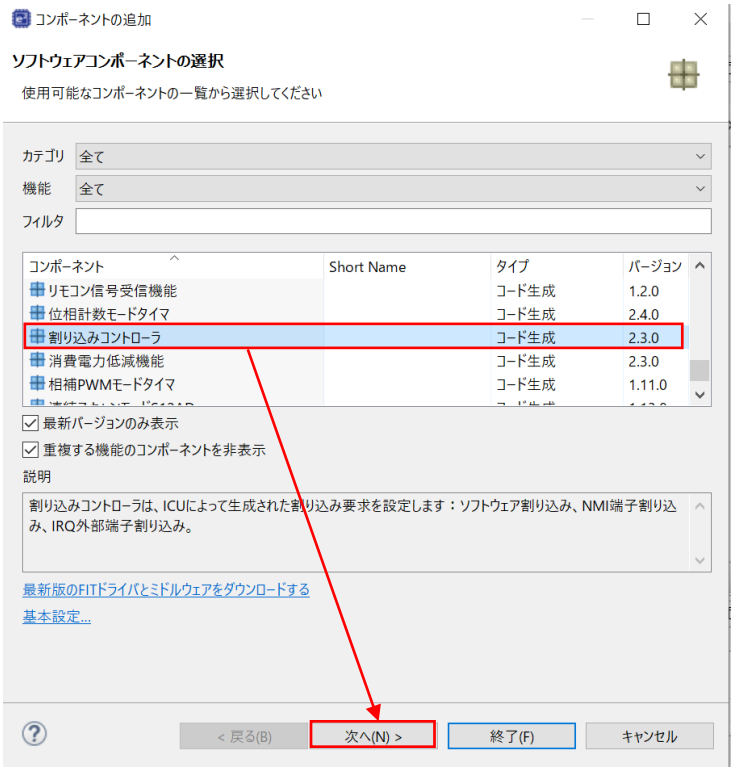
なお、転送データ長 16 ビットの通信で必要な設定は、SC の生成コードに対して手動で追記します。

4.2.2.1　割り込みコントローラ設定

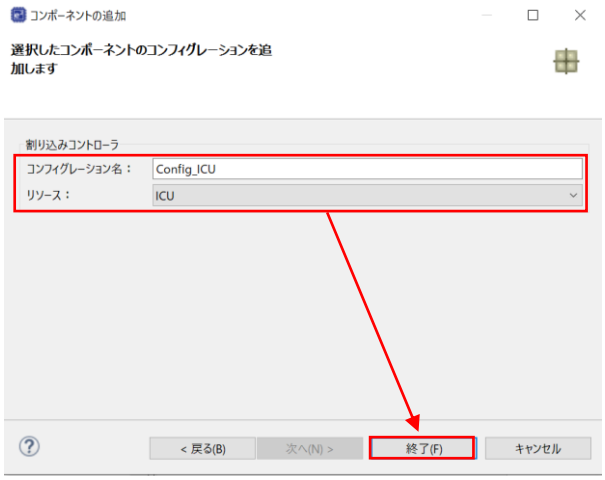
(1)　「コンポーネント」タブを開き、コンポーネントの追加を選択します。



(2)　ソフトウェアコンポーネントの選択画面で「割り込みコントローラ」を選択し、「次へ」ボタンをクリックします。



(3) コンポーネントの追加画面では「終了」ボタンをクリックします。

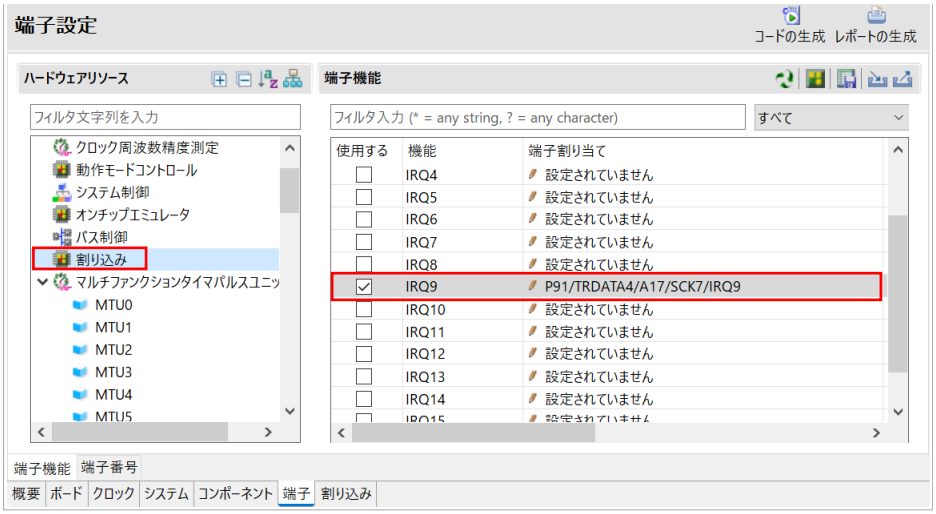


(4) ソフトウェアコンポーネント設定

RSK の SW1 に接続されている端子(P91)を IRQ9 に割り当てます。

(5) 端子設定

「端子」タブを開き、ハードウェアリソースから、「割り込み」を選択し、端子機能の IRQ9 に P91 を割り当てます。

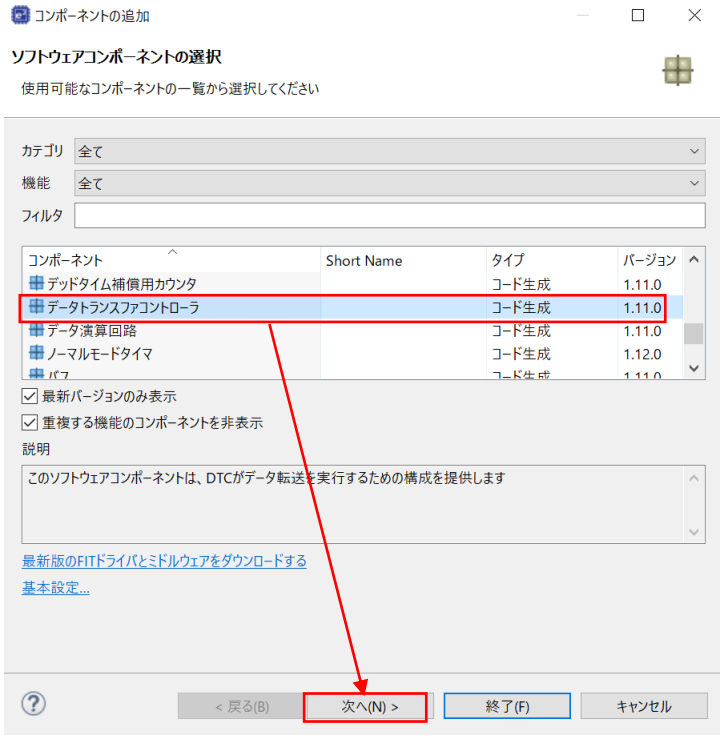


4.2.2.2 データトランスファコントローラ設定(データ送信用 DTC 設定(転送データ長 24 ビット))

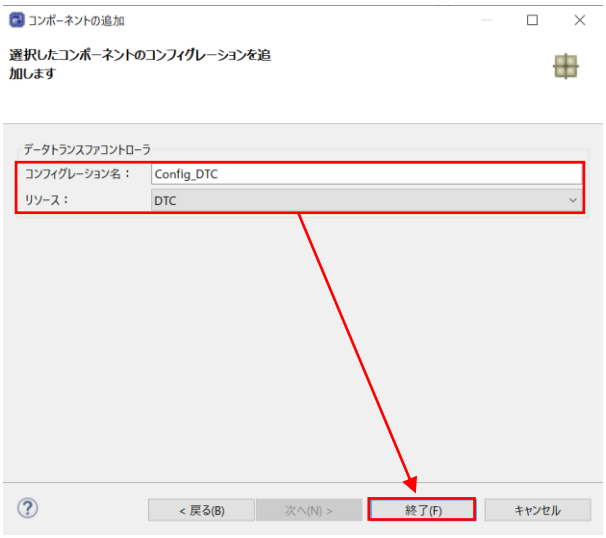
(1) 「コンポーネント」タブを開き、コンポーネントの追加を選択します。

選択画面は4.2.2.1(1)をご参照ください。

(2) ソフトウェアコンポーネントの選択画面で「データトランスファコントローラ」を選択し、「次へ」ボタンをクリックします。



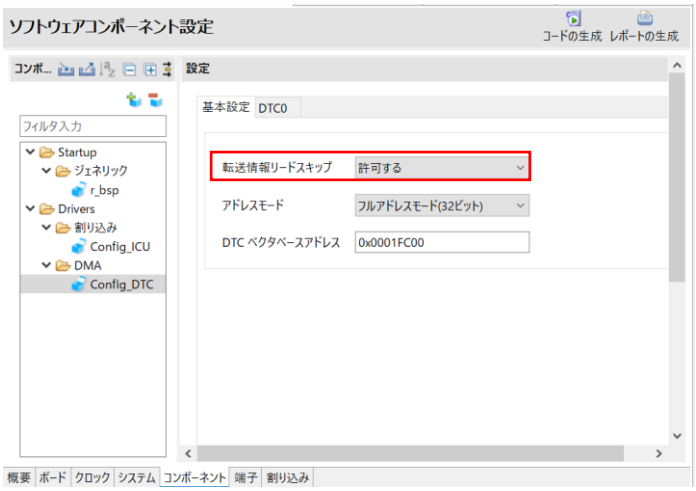
(3) コンポーネントの追加画面では、「終了」ボタンをクリックします。



(4)　ソフトウェアコンポーネント設定

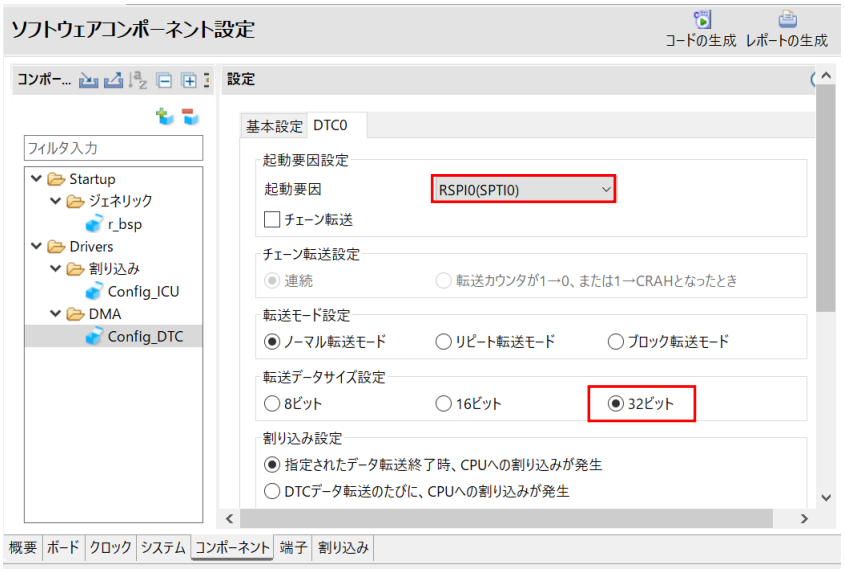
本アプリケーションノートでは、「基本設定」タブの項目を以下のように設定しています。

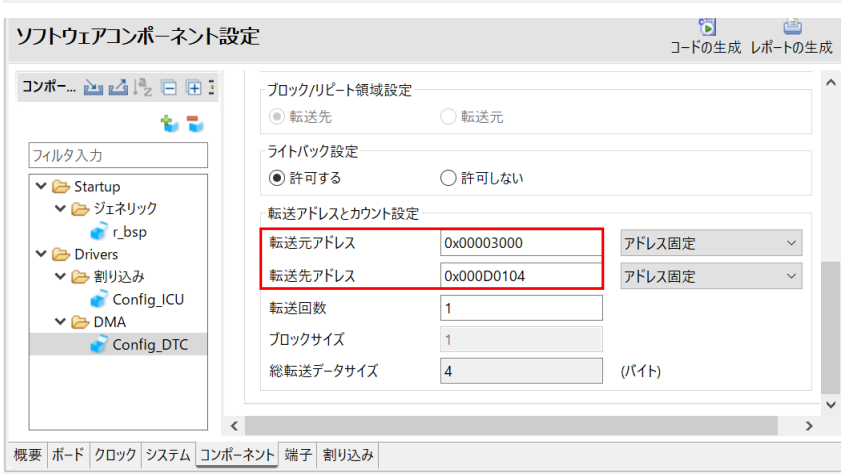
赤枠はデフォルトの状態から変更している設定です。



「DTC0」タブは以下のように設定しています。

赤枠はデフォルトの状態から変更している項目です。





4.2.2.3 データトランスファコントローラ設定(データ受信用 DTC 設定(転送データ長 24 ビット))

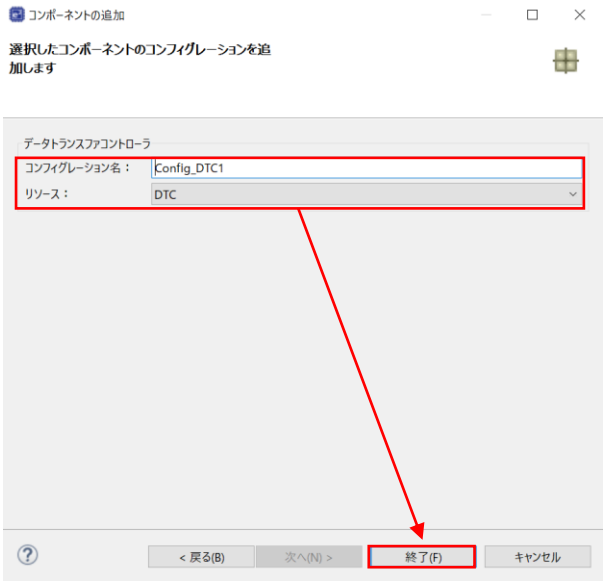
(1) 「コンポーネント」タブを開き、コンポーネントの追加を選択します。

選択画面は4.2.2.1(1)をご参照ください。

(2) ソフトウェアコンポーネントの選択画面で「データトランスファコントローラ」を選択し、「次へ」ボタンをクリックします。

選択画面は4.2.2.2(2)をご参照ください。

(3) コンポーネントの追加画面では、「終了」ボタンをクリックします。



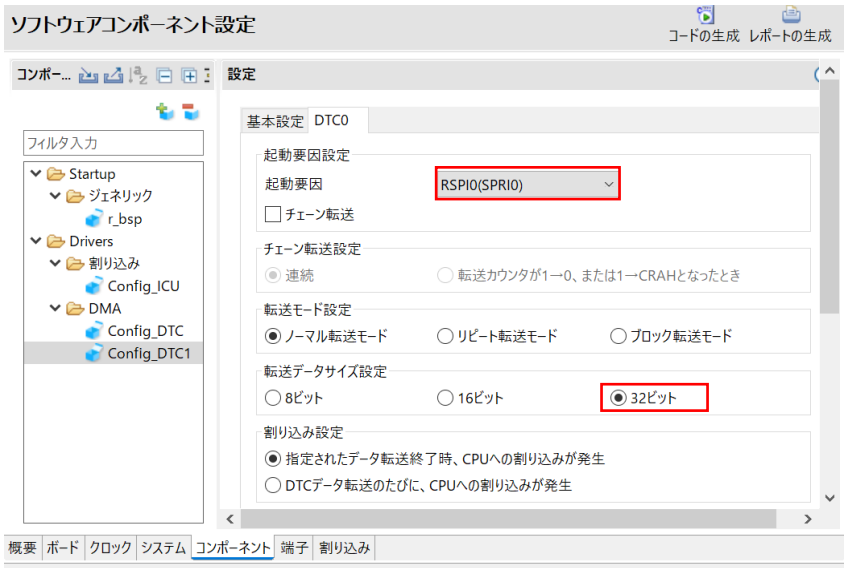
(4) ソフトウェアコンポーネント設定

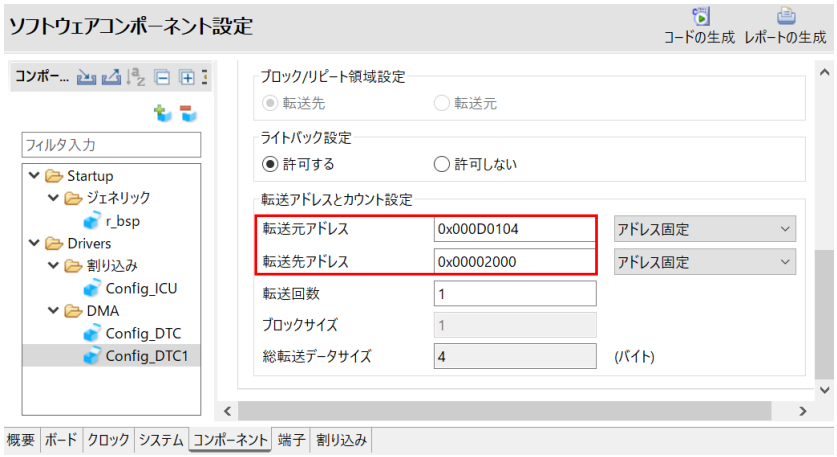
本アプリケーションノートでは「基本設定」タブの項目を以下のように設定しています。

赤枠はデフォルトの状態から変更している設定です。

「DTC0」タブは以下のように設定しています。

赤枠はデフォルトの状態から変更している項目です。



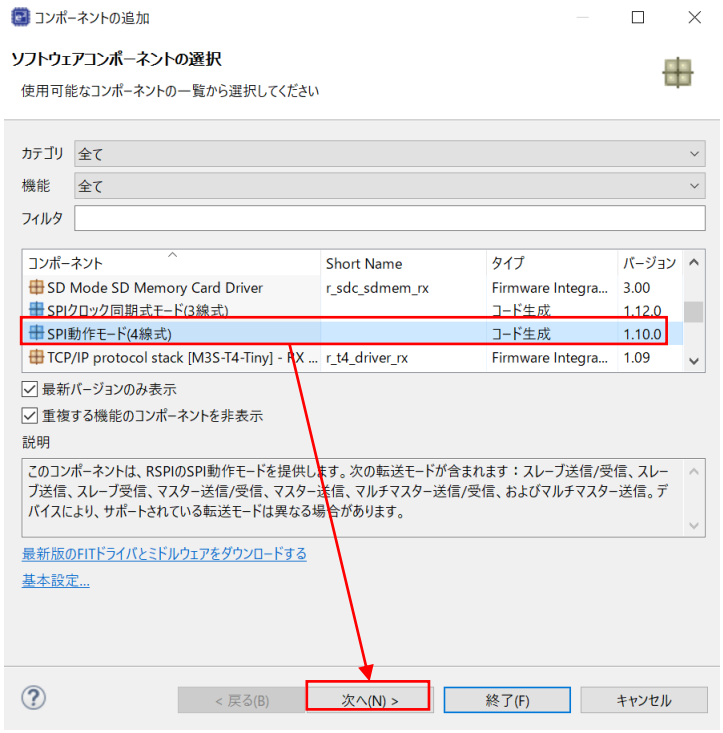


4.2.2.4　SPI 動作モード(4 線式)設定

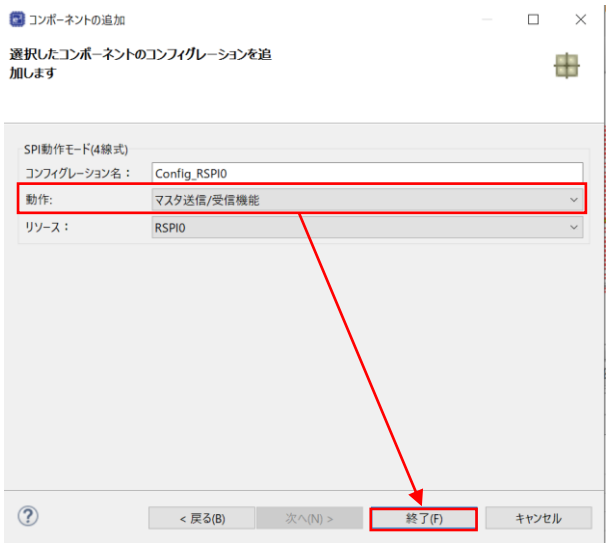
(1)　「コンポーネント」タブを開き、コンポーネントの追加を選択します。

選択画面は4.2.2.1(1)をご参照ください。

(2)　ソフトウェアコンポーネントの選択画面で「SPI 動作モード(4 線式)」を選択し、「次へ」ボタンをクリックします。



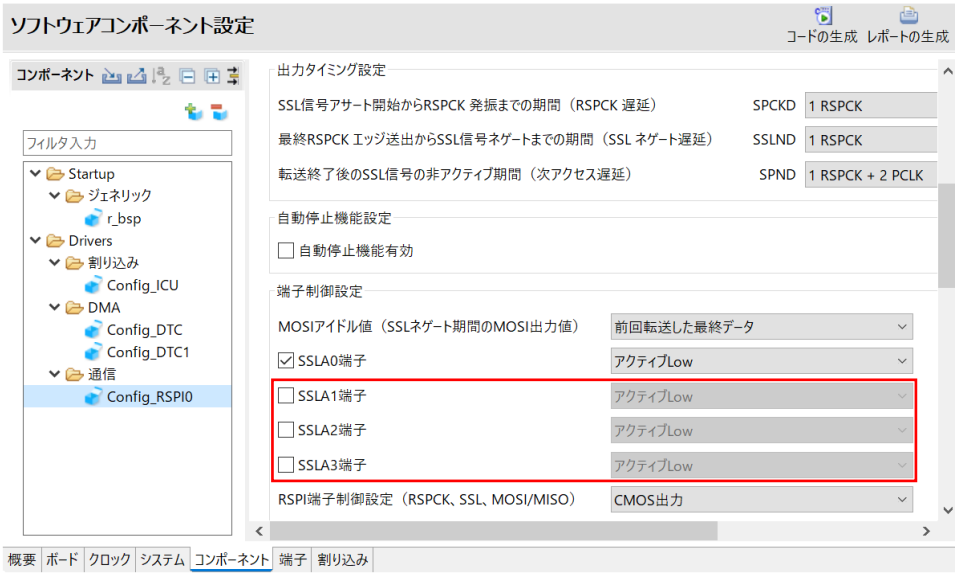
(3)　コンポーネントの追加画面の「動作」で「マスタ送信/受信機能」を選択し、「終了」ボタンをクリックします。

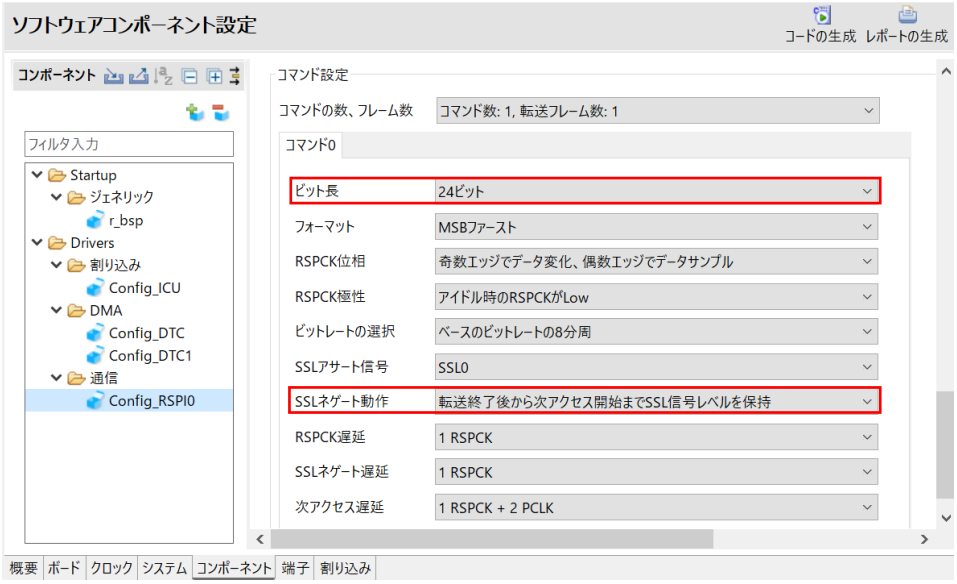


(4)　ソフトウェアコンポーネント設定

本アプリケーションノートでは以下のように設定しています。

赤枠はデフォルトの状態から変更している設定です。





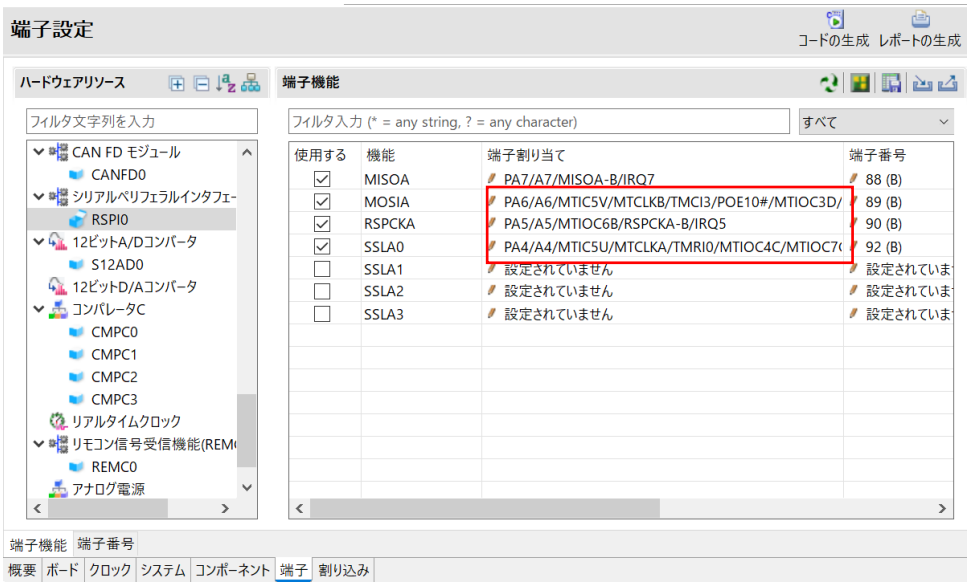
(5)　端子設定

「端子」タブを開き、ハードウェアリソースから「RSPI0」を選択し、RSPI で使用する端子を割り当てます。

本アプリケーションノートでは、RSK 上でオープンとなっている端子を割り当てています。

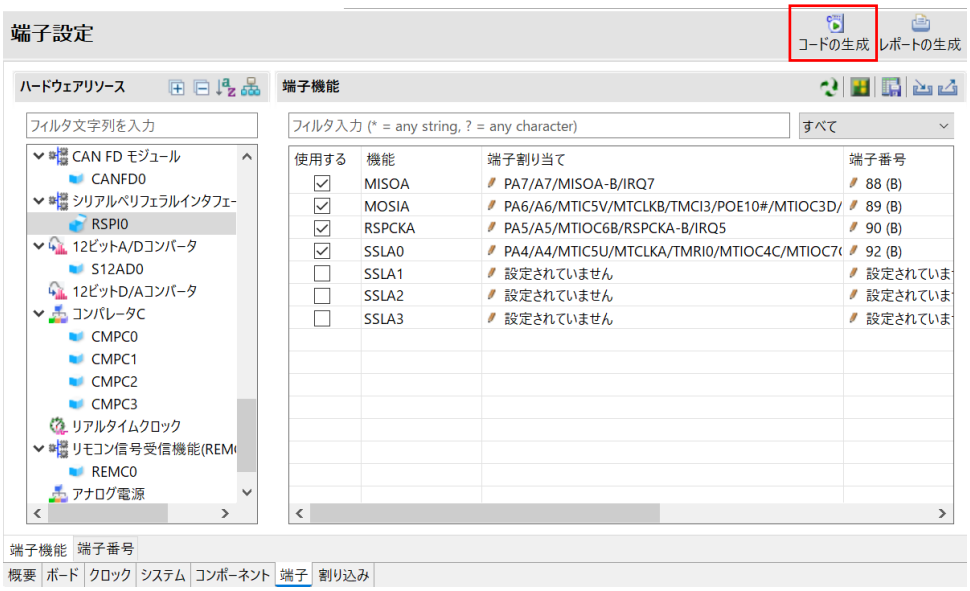
なお、SSLA0 端子は使用しません。

赤枠はデフォルトの状態から変更している設定です。

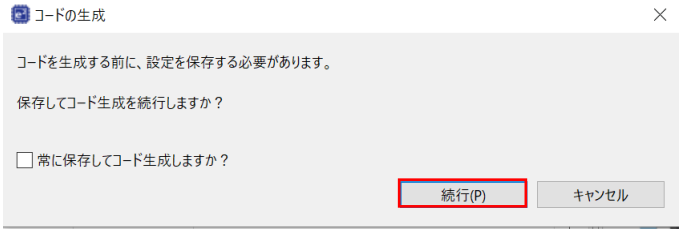


4.2.3　コードの生成

全ての SC 設定を終了後、「コード生成」ボタンをクリックしてコードを生成します。



以下のダイアログが表示されるため、「続行」ボタンをクリックします。



4.2.4 SC 生成コードへのコード追加

ユーザコードを追加する際は、ソースファイルやヘッダファイルの

/* Start user ・・・ */

/* End user ・・・ */

と記載されている行の間にコードを追加できます。

4.2.4.1 SC 生成コードへの追加処理

SC が生成したコードに対して、転送データ長 16 ビット通信のために必要な処理(転送データ長や DTC 設定の変更など)や SW1 押下後の IRQ9 割り込み処理を追加します。

表 4.11にSC生成コードへの追加内容一覧を示します。

表 4.11　SC 生成コードへの追加内容一覧

フォルダ	ファイル	変更または追加した関数	追加内容
src	data_length_change_sample_for_rspi_dtc.c	void main(void)	汎用ポート制御による SSL 信号を High、24 ビット通信完了チェックフラグ(g_length_check)をクリア、IRQ9 許可の処理を追加
src¥sm_gen¥Config_ICU	Config_ICU_user.c	static void r_Config_ICU_irq9_interrupt(void)	送信データ初期化、受信データ格納 RAM 初期化、DTC スタート、SSL 信号アサート、RSPI のスタートの処理を追加。
src¥sm_gen¥Config_DTC	Config_DTC.c	void set_16bit_data_transfer_mode(void)	void set_16bit_data_transfer_mode(void)関数を追加。本関数では、「表 4.7 DTC転送C：データ送信用DTC設定(転送データ長16ビット)」のための DTC 転送情報を設定する。
	Config_DTC.h	-	void set_16bit_data_transfer_mode(void)関数のプロトタイプ宣言追加。 上記関数で設定する転送カウントレジスタA(CRA)の値をマクロ定数として追加。
src¥sm_gen¥Config_DTC1	Config_DTC1.c	void set_16bit_data_receive_mode(void)	void set_16bit_data_receive_mode(void)関数を追加。本関数では、「表 4.8 DTC転送D：データ受信用DTC設定(転送データ長16ビット)」のための DTC 転送情報を設定する。
	Config_DTC1.h	-	void set_16bit_data_receive_mode(void)関数のプロトタイプ宣言追加。 上記関数で設定する転送カウントレジスタA(CRA)の値をマクロ定数として追加。
src¥sm_gen¥Config_RSPI0	Config_RSPI0_user.c	static void r_Config_RSPI0_communication_end_interrupt(void)	以下の処理を追加。 ・24 ビット通信完了時 転送データ長変更(16 ビットに変更)、DTC 設定変更、DTC スタート、RSPI スタートの処理。 ・16 ビット x 8 フレーム通信完了時 SSL 信号ネゲート、転送データ長変更(24 ビットに変更)、DTC 設定変更の処理。
		static void r_Config_RSPI0_callback_transmitend(void)	SPTI0 割り込みの禁止処理を追加。
		static void r_Config_RSPI0_callback_receiveend(void)	SPCI0 割り込みの許可処理を追加。
	Config_RSPI0.h	-	SSL 制御に使用する汎用ポートのマクロ定義を追加。

4.2.4.2　SC 生成コードへの追加定数

表 4.12にSC生成コードに追加した定数一覧を示します。

表 4.12　SC 生成コードに追加した定数一覧

定数名	設定値	内容
SSL_PORT_PODR_BIT	PORTA.PODR.BIT.B2	SSL 制御に使用する汎用ポートの PODR レジスタのビット
SSL_PORT_PDR_BIT	PORTA.PDR.BIT.B2	SSL 制御に使用する汎用ポートの PDR レジスタのビット
CRA_16BIT_TRANSFER	0x0008	16 ビット送信時の DTC 転送カウントレジスタ A(CRA)の設定値
CRA_16BIT_RECEIVE	0x0008	16 ビット受信時の DTC 転送カウントレジスタ A(CRA)の設定値

4.2.4.3　SC 生成コードへの追加変数

表 4.13にSC生成コードに追加した変数一覧を示します。

表 4.13　SC 生成コードに追加した変数一覧

型	変数名	内容	使用関数
volatile uint8_t	g_length_check	転送データ長 チェックフラグ 0：転送データ長 24 ビット 1：転送データ長 16 ビット	main() r_Config_ICU_irq9_interrupt() r_Config_RSPI0_communication_end_interrupt()
volatile uint32_t	g_w24_data	24 ビット送信データを格納	r_Config_ICU_irq9_interrupt()
volatile uint32_t	g_r24_data	24 ビット受信データを格納	r_Config_ICU_irq9_interrupt()
volatile uint16_t	g_w16_data	16 ビット送信データを格納	r_Config_ICU_irq9_interrupt() set_16bit_data_transfer_mode()
volatile uint16_t	g_r16_data[8]	16 ビット受信データを格納	r_Config_ICU_irq9_interrupt() set_16bit_data_receive_mode()

4.2.4.4　SC 生成コードへの追加関数

表 4.14にSC生成コードに追加した関数一覧を示します。

表 4.14　SC 生成コードに追加した関数一覧

型	変数名	引数	内容
void	set_16bit_data_transfer_mode	void	16 ビット送信時の DTC 転送情報を「表 4.7 DTC転送C：データ送信用DTC設定(転送データ長16ビット)」の内容に設定
void	set_16bit_data_receive_mode	void	16 ビット受信時の DTC 転送情報を「表 4.8 DTC転送D：データ受信用DTC設定(転送データ長16ビット)」の内容に設定

4.2.4.5　メインルーチンへのコード追加

data_length_change_sample_for_rspi_dtc.c ファイルのメインルーチン main()関数にコードを追加します。

図 4.3にmain()関数の概略フローを示します。

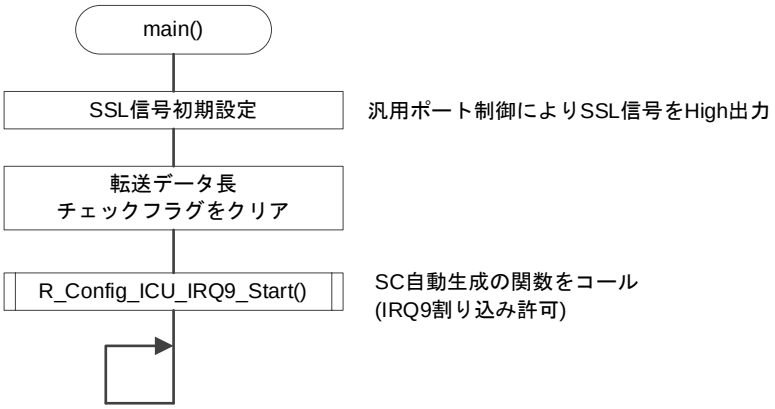


図 4.3 main()関数の概略フロー

Config_RSPI0.h ファイルにコードを追加します。

Config_RSPI0.h ファイルへのコード追加

```
/* Start user code for function. Do not edit comment generated here */
/**********************************************************************************************************************
Macro definitions
**********************************************************************************************************************/
#define SSL_PORT_PODR_BIT (PORTA.PODR.BIT.B2)   ← 汎用ポート PA2 で SSL 信号を制御
#define SSL_PORT_PDR_BIT   (PORTA.PDR.BIT.B2)
/* End user code. Do not edit comment generated here */
```

メインルーチンにコードを追加します。

main()関数へのコード追加

```
#include "r_smc_entry.h"          ← SC が自動生成した各ヘッダファイルをインクルード

volatile uint8_t g_length_check;  ← 転送データ長チェックフラグの定義

void main(void);

void main(void)
{
  /* Set General-purpose port for SSL control */
  SSL_PORT_PODR_BIT = 1U;         ← 汎用ポート制御により SSL 信号を High 出力
  SSL_PORT_PDR_BIT = 1U;

  g_length_check = 0U;            ← 転送データ長チェックフラグをクリア
  R_Config_ICU_IRQ9_Start();      ← IRQ9 を許可

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

4.2.4.6　IRQ9 割り込み処理へのコード追加

IRQ9 割り込み処理である r_Config_ICU_irq9_interrupt()関数にコードを追加しています。

図 4.4にr_Config_ICU_irq9_interrupt()関数の概略フローを示します。

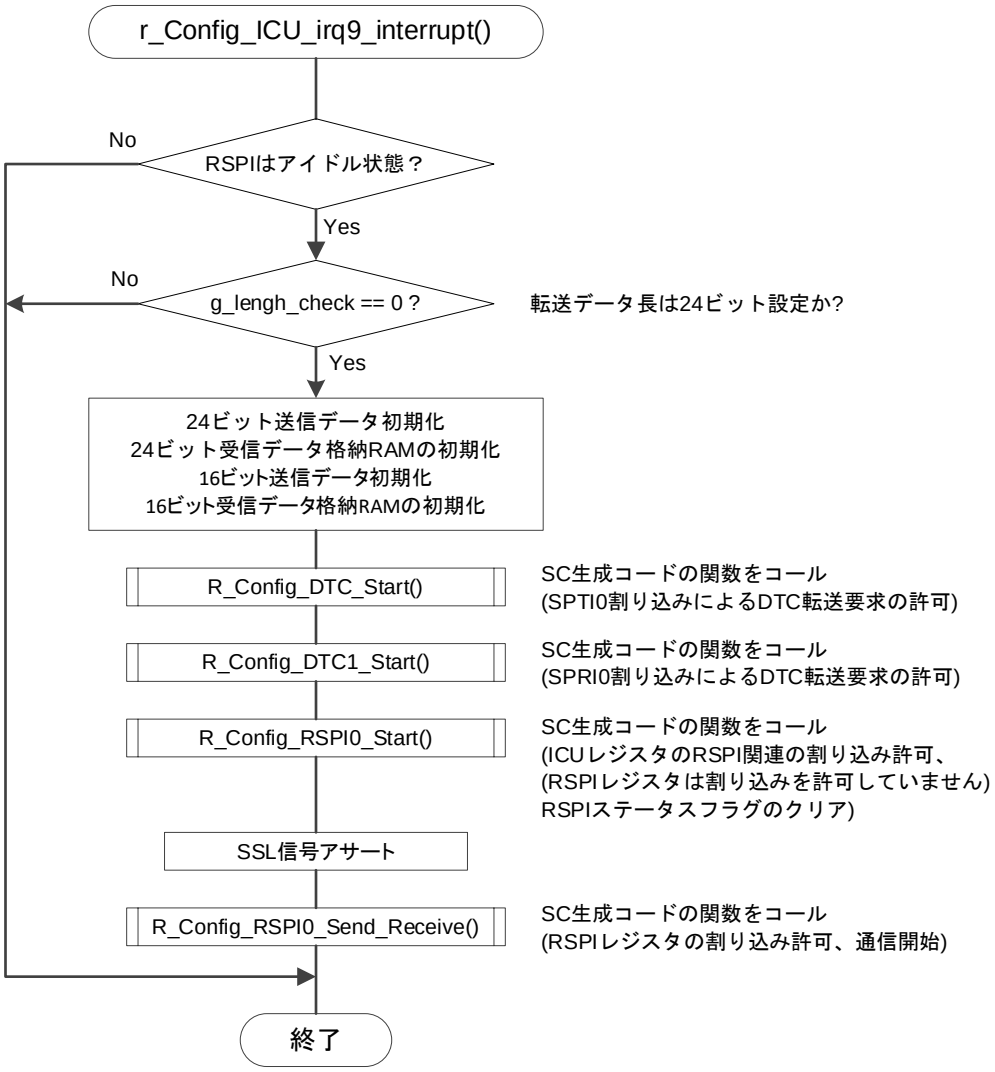


図 4.4 r_Config_ICU_irq9_interrupt()関数の概略フロー

Config_ICU_user.c ファイルの“Includes”と“Global variables and functions”にコードを追加します。

Config_ICU_user.c ファイルの“Includes”と“Global variables and functions”へのコード追加

```
/***********************************************************************************************************************
Includes
***********************************************************************************************************************/
#include "r_cg_macrodriver.h"
#include "Config_ICU.h"
/* Start user code for include. Do not edit comment generated here */
#include "r_smc_entry.h"                    ← SC が自動生成した各ヘッダファイルをインクルード
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_length_check;     ←転送データ長チェックフラグの extern 文

#pragma address (g_w24_data=0x03000U)       ← 「4.2.2.2や4.2.2.3で設定した転送元アドレス、
#pragma address (g_r24_data=0x02000U)           転送先アドレスに合わせてアドレスを定義

volatile uint32_t   g_w24_data;             ←24 ビット送信データ、受信データ格納 RAM
volatile uint32_t   g_r24_data;               16 ビット送信データ、受信データ格納 RAM
volatile uint16_t   g_w16_data;               の変数を定義
volatile uint16_t   g_r16_data[8];

/* End user code. Do not edit comment generated here */
```

r_Config_ICU_irq9_interrupt()関数にコードを追加します。

r_Config_ICU_irq9_interrupt ()関数へのコード追加

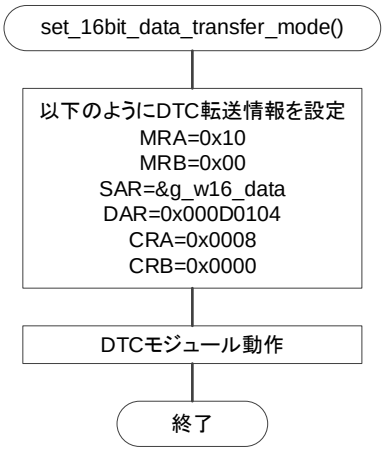
```
static void r_Config_ICU_irq9_interrupt(void)
{
/* Start user code for r_Config_ICU_irq9_interrupt. Do not edit comment generated here */
  uint32_t i;

  if (0U != RSPI0.SPSR.BIT.IDLNF)        ← RSPI がアイドル状態であることを確認します。
  {
    /* do nothing */
  }
  else
  {
    If (0U == g_length_check)            ← 転送データ長が最初の 24 ビット設定であることを確認
    {
        g_w24_data = 0x123456U;
        g_r24_data = 0xFFFFFFFFU;        ← 24 ビット送信データ、24 ビット受信データ格納 RAM、
                                           16 ビット送信データ、16 ビット受信データ格納 RAM
        g_w16_data = 0x789AU;              の初期化

        for (i=0U; i<8U; i++)
        {
            g_r16_data[i] = 0xFFFFU;
        }

        R_Config_DTC_Start();            ← SPTI0 割り込みや SPRI0 割り込みによる DTC 転送要求
        R_Config_DTC1_Start();             を許可
        R_Config_RSPI0_Start();          ICU レジスタの RSPI 関連の割り込みを許可
                                         (RSPI レジスタは割り込みを許可していません)
                                         RSPI ステータスフラグのクリア

        /* SSL assert */
        SSL_PORT_PODR_BIT = 0U;          ← SSL 信号をアサート

        R_Config_RSPI0_Send_Receive(NULL, 24U, NULL);   ← RSPI レジスタの割り込みを許可、
    }                                                     通信開始
  }
    /* End user code. Do not edit comment generated here */
}
```

4.2.4.7　Config_DTC.c ファイルへの set_16bit_data_transfer_mode()関数追加

Config_DTC.c ファイルに set_16bit_data_transfer_mode()関数を追加しています。

本関数では、16 ビット送信時の DTC 転送情報を「表 4.7 DTC転送C：データ送信用DTC設定(転送データ長16ビット)」の内容に設定します。

図 4.5にset_16bit_data_transfer_mode()関数の概略フローを示します。



設定内容は「表 4.7DTC転送C：データ送信用DTC設定(転送データ長16ビット)」

図 4.5 set_16bit_data_transfer_mode()関数の概略フロー

Config_DTC.h ファイルに set_16bit_data_transfer_mode()関数のプロトタイプ宣言と定数定義を追加します。

Config_DTC.h ファイルへのコード追加

```
/* Start user code for function. Do not edit comment generated here */
void set_16bit_data_transfer_mode(void);                ←プロトタイプ宣言
#define CRA_16BIT_TRANSFER          (0x0008U)      ←16 ビット送信時の CRA レジスタの設定値
/* End user code. Do not edit comment generated here */
```

Config_DTC.c ファイルの“Global variables and functions”に 16 ビット通信時の送信データ格納変数の extern 文を追加します。

Config_DTC.c ファイルの“Global variables and functions”へのコード追加

```
/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
#pragma address dtc_vector39=0x0001FC9CUL
volatile uint32_t dtc_vector39;
volatile st_dtc_data_t dtc_transferdata_vector39;
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t g_w16_data;        ←16 ビット通信時の送信データ格納変数の extern 文

/* End user code. Do not edit comment generated here */
```

Config_DTC.c ファイルに set_16bit_data_transfer_mode()関数を追加します。

Config_DTC.c ファイルに追加した set_16bit_data_transfer_mode()関数

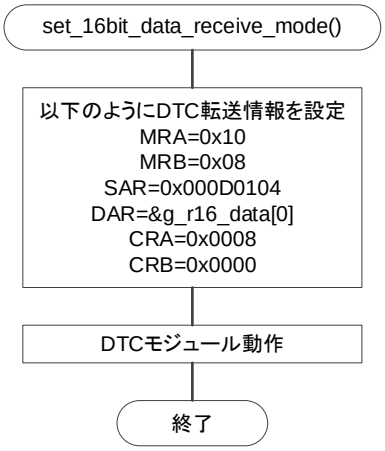
```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************************************************
* Function Name: set_16bit_data_transfer_mode
* Description   : This function initializes the DTC module for 16bit transmission.
* Arguments     : None
* Return Value : None
***********************************************************************************************************************/
void set_16bit_data_transfer_mode(void)
{
    /* Set DTC transfer data */
    dtc_transferdata_vector39.mra_mrb = ((uint32_t)(_00_DTC_WRITE_BACK_ENABLE |
                                        _00_DTC_SRC_ADDRESS_FIXED |
                                        _10_DTC_TRANSFER_SIZE_16BIT |
                                        _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |
                                        ((uint32_t)(_00_DTC_DST_ADDRESS_FIXED |
                                        _00_DTC_INTERRUPT_COMPLETED)<<16U);
    dtc_transferdata_vector39.sar = (uint32_t) &g_w16_data;
    dtc_transferdata_vector39.dar = _000D0104_DTC0_DST_ADDRESS;
    dtc_transferdata_vector39.cra_crb = (uint32_t)(CRA_16BIT_TRANSFER) << 16U;

    /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;
}
/* End user code. Do not edit comment generated here */
```

4.2.4.8　Config_DTC1.c ファイルへの set_16bit_data_receive_mode()関数追加

Config_DTC1.c ファイルに set_16bit_data_receive_mode()関数を追加しています。

本関数では、16 ビット受信時の DTC 転送情報を「表 4.8 DTC転送D：データ受信用DTC設定(転送データ長16ビット)」の内容に設定します。

図 4.6にset_16bit_data_receive_mode()関数の概略フローを示します。



設定内容は

「表 4.8 DTC転送D：データ受信用DTC設定(転送データ長16ビット)」

を参照

図 4.6 set_16bit_data_receive_mode()関数の概略フロー

Config_DTC1.h ファイルに set_16bit_data_receive_mode()関数のプロトタイプ宣言と定数定義を追加します。

Config_DTC1.h ファイルへのコード追加

```
/* Start user code for function. Do not edit comment generated here */
void set_16bit_data_receive_mode(void);                ←プロトタイプ宣言
#define CRA_16BIT_RECEIVE          (0x0008U)           ←16 ビット受信時の CRA レジスタの設定値
/* End user code. Do not edit comment generated here */
```

Config_DTC1.c ファイルの“Global variables and functions”に 16 ビット通信時の受信データ格納変数の extern 文を追加します。

Config_DTC1.c ファイルの“Global variables and functions”へのコード追加

```
/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
#pragma address dtc_vector38=0x0001FC98UL
volatile uint32_t dtc_vector38;
volatile st_dtc_data_t dtc_transferdata_vector38;
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t g_r16_data[8];                ←16 ビット通信時の受信データ格納変数の extern 文

/* End user code. Do not edit comment generated here */
```

Config_DTC1.c ファイルに set_16bit_data_receive_mode()関数を追加します。

Config_DTC1.c ファイルに追加した set_16bit_data_receive_mode()関数

```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************************************************
* Function Name: set_16bit_data_receive_mode
* Description   : This function initializes the DTC module for 16bit reception.
* Arguments     : None
* Return Value : None
***********************************************************************************************************************/
void set_16bit_data_receive_mode(void)
{
    /* Set DTC transfer data */
    dtc_transferdata_vector38.mra_mrb = ((uint32_t)(_00_DTC_WRITE_BACK_ENABLE |
                                        _00_DTC_SRC_ADDRESS_FIXED |
                                        _10_DTC_TRANSFER_SIZE_16BIT |
                                        _00_DTC_TRANSFER_MODE_NORMAL)<<24U) |
                                         ((uint32_t)(_08_DTC_DST_ADDRESS_INCREMENTED |
                                        _00_DTC_REPEAT_DST_SIDE |
                                        _00_DTC_INTERRUPT_COMPLETED)<<16U);
    dtc_transferdata_vector38.sar = _000D0104_DTC0_SRC_ADDRESS;
    dtc_transferdata_vector38.dar = (uint32_t) &g_r16_data[0];
    dtc_transferdata_vector38.cra_crb = (uint32_t)(CRA_16BIT_RECEIVE) << 16U;

    /* Enable DTC module start */
    DTC.DTCST.BYTE = _01_DTC_MODULE_START;
}
/* End user code. Do not edit comment generated here */
```

4.2.4.9　SPTI0 割り込み処理へのコード追加

SPTI0 割り込みのコールバック関数(r_Config_RSPI0_callback_transmitend())にコード追加しています。

図 4.7にr_Config_RSPI0_callback_transmitend()関数の概略フローを示します。

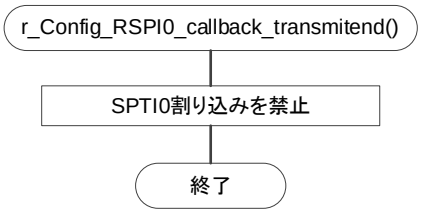


図 4.7 r_Config_RSPI0_callback_transmitend()関数の概略フロー

Config_RSPI0_user.c ファイルの r_Config_RSPI0_callback_transmitend()関数にコードを追加しています。

R_Config_RSPI0_callback_transmitend()関数へのコード追加

```
/* Start user code for r_Config_RSPI0_callback_transmitend. Do not edit comment generated here */
/* Disable Transmit buffer empty interrupt */
RSPI0.SPCR.BIT.SPTIE = 0U;                    ←SPTI0 割り込み禁止
/* End user code. Do not edit comment generated here */
```

4.2.4.10 SPRI0 割り込み処理へのコード追加

SPRI0 割り込みのコールバック関数(r_Config_RSPI0_callback_receiveend())にコードを追加しています。

図 4.8にr_Config_RSPI0_callback_reveiveend()関数の概略フローを示します。

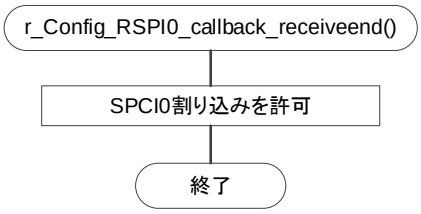


図 4.8　r_Config_RSPI0_callback_reveiveend()関数の概略フロー

Config_RSPI0_user.c ファイルの r_Config_RSPI0_callback_receiveend()関数にコードを追加しています。

R_Config_RSPI0_callback_receiveend()関数へのコード追加

```
/* Start user code for r_Config_RSPI0_callback_transmitend. Do not edit comment generated here */
/* Enable communication end interrupt */
RSPI0.SPCR3.BIT.SPCIE = 1U;                    ←SPCI0 割り込み許可
/* End user code. Do not edit comment generated here */
```

4.2.4.11 SPCI0 割り込み処理へのコード追加

SPCI0 割り込みのコールバック関数(r_Config_RSPI0_communication_end_interrupt())にコードを追加しています。

図 4.9にr_Config_RSPI0_communication_end_interrupt()関数の概略フローを示します。

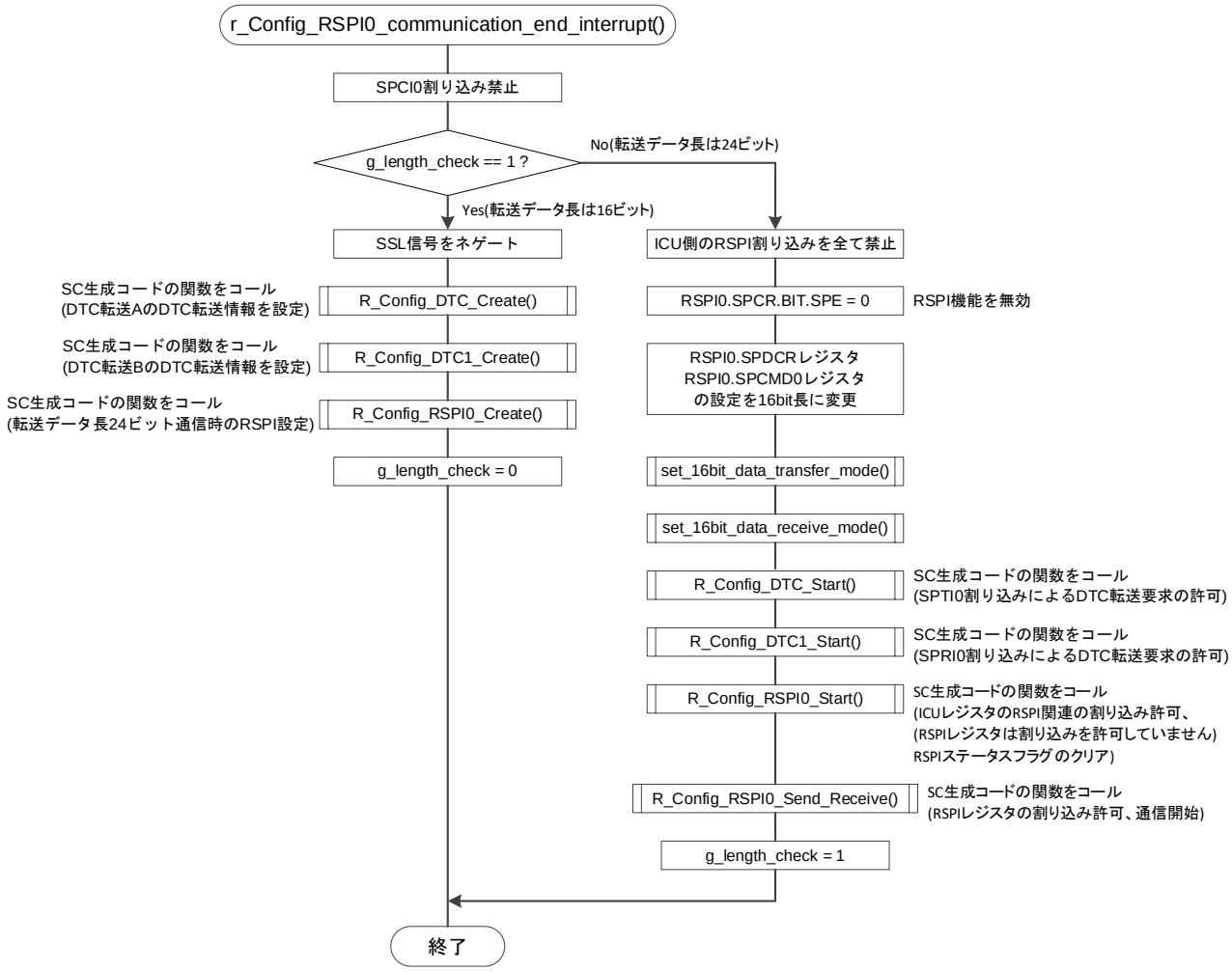


図 4.9　r_Config_RSPI0_communication_end_interrupt()関数の概略フロー

Config_RSPI0_user.c ファイルの“Includes”と“Global variables and functions”に定義文を追加します。

Config_RSPI0_user.c ファイルの“Includes”と“Global variables and functions”へのコード追加

```
/***********************************************************************************************************************
Includes
***********************************************************************************************************************/
#include “r_cg_macrodriver.h”
#include “Config_RSPI0.h”
/* Start user code for include. Do not edit comment generated here */
#include “r_smc_entry.h”                    ← SC が自動生成した各ヘッダファイルをインクルード
/* End user code. Do not edit comment generated here */
#include “r_cg_userdefine.h”

/***********************************************************************************************************************
Global variables and functions
***********************************************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_length_check;          ←転送データ長チェックフラグの extern 文
/* End user code. Do not edit comment generated here */
```

Config_RSPI0_user.c ファイルの r_Config_RSPI0_communication_end_interrupt()関数にコードを追加しています。

R_Config_RSPI0_communication_end_interrupt()関数へのコード追加

```
/* Start user code for r_Config_RSPI0_communication_end_interrupt. Do not edit comment generated here */
   /* Disable communication end interrupt */
   RSPI0.SPCR3.BIT.SPCIE = 0U;
   /* Processing when 16bit x 8frames is received */
      If (1U == g_length_check)
      {
         /*SSL negate */
         SSL_PORT_PODR_BIT = 1U;

         /* Return DTC and RSPI settings to default settings (for 24-bit settings) */
         R_Config_DTC_Create();
         R_Config_DTC1_Create();
         R_Config_RSPI0_Create();

         g_length_check = 0U;
      }
      else
      {
         /* Processing when 24bit x 1frame is received */
         /* Disable RSPI interrupts */
         IEN(RSPI0,SPTI0) = 0U;
         IEN(RSPI0,SPRI0) = 0U;
         EN(RSPI0,SPEI0) = 0U;
         EN(RSPI0,SPII0) = 0U;
         IEN(RSPI0, SPCI0) = 0U;

         /* Disable RSPI function */
         RSPI0.SPCR.BIT.SPE = 0U;

         /* Change RSPI settings for 16bit length */
         RSPI0.SPDCR.BIT.SPLW = 0U;
         RSPI0.SPDCR.BIT.SPBYT = 0U;
         RSPI0.SPDCR.BIT.SPFC = 0x00U;
         RSPI0.SPCMD0.WORD = _0001_RSPI_RSPCK_SAMPLING_EVEN |
                             _0000_RSPI_RSPCK_POLARITY_LOW |
                             _000C_RSPI_BASE_BITRATE_8 |
                             _0000_RSPI_SIGNAL_ASSERT_SSL0 |
                             _0080_RSPI_SSL_KEEP_ENABLE |
                             _0F00_RSPI_DATA_LENGTH_BITS_16 |
                             _0000_RSPI_MSB_FIRST |
                             _0000_RSPI_NEXT_ACCESS_DELAY_DISABLE |
                             _0000_RSPI_NEGATION_DELAY_DISABLE |
                             _0000_RSPI_RSPCK_DELAY_DISABLE;
```

←16 ビット通信の完了後、SSL 信号をネゲートし、転送データ長を 24 ビットに戻すための処理

24 ビット通信の完了後、転送データ長を 16 ビットに変更した通信を開始するための処理
↓

```
        /* Change DTC settings for 16bit x 8frames */
        set_16bit_data_transfer_mode();
        set_16bit_data_receive_mode();

        /* DTC, RSPI start */
        R_Config_DTC_Start();
        R_Config_DTC1_Start();
        R_Config_RSPI0_Start();
        R_Config_RSPI0_Send_Receive(NULL, 16U, NULL);

        g_length_check = 1U;
    }
    /* End user code. Do not edit comment generated here */
```

5.　プロジェクトをインポートする方法

サンプルプログラムは e² studio のプロジェクト形式で提供しています。本章では、 e² studio および CS+ へプロジェクトをインポートする方法を示します。インポート完了後、ビルドおよびデバッグの設定を確認してください。

5.1　e² studio での手順

e² studio でご使用になる際は、下記の手順で e² studio にインポートしてください。

なお、e² studio で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、 空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する e² studio のバージョンによっては画面が異なる場合があります。)

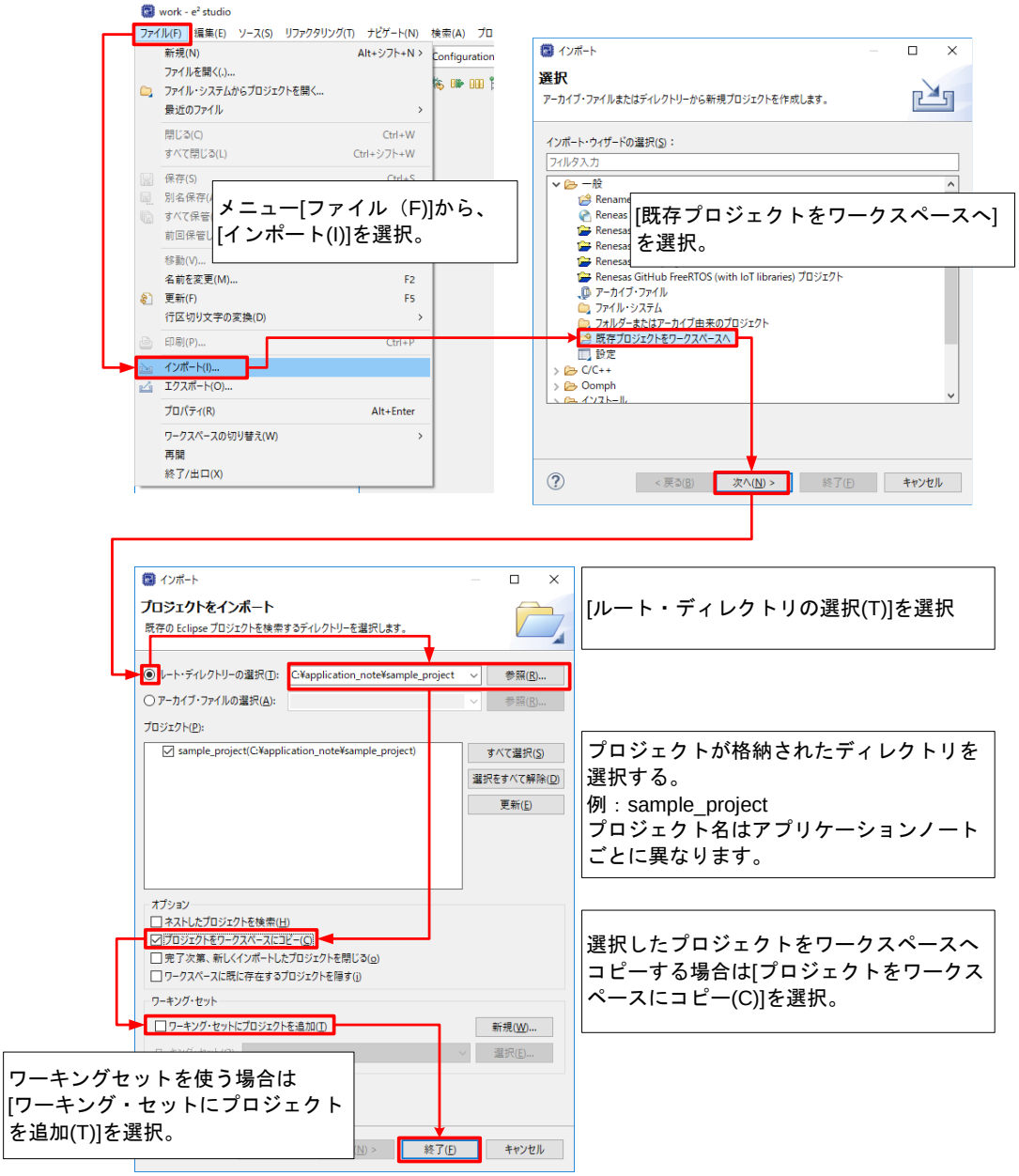


図 5.1　プロジェクトを e² studio にインポートする方法

5.2　CS+での手順

CS+でご使用になる際は、下記の手順で CS+にインポートしてください。

なお、CS+で管理するプロジェクトのフォルダ名、およびそのフォルダに至るファイルパスには、 空白文字の他、半角カナ文字、全角文字、半角記号(特に'$','#','%') が混じらないようにしてください。

(使用する CS+のバージョンによっては画面が異なる場合があります。)

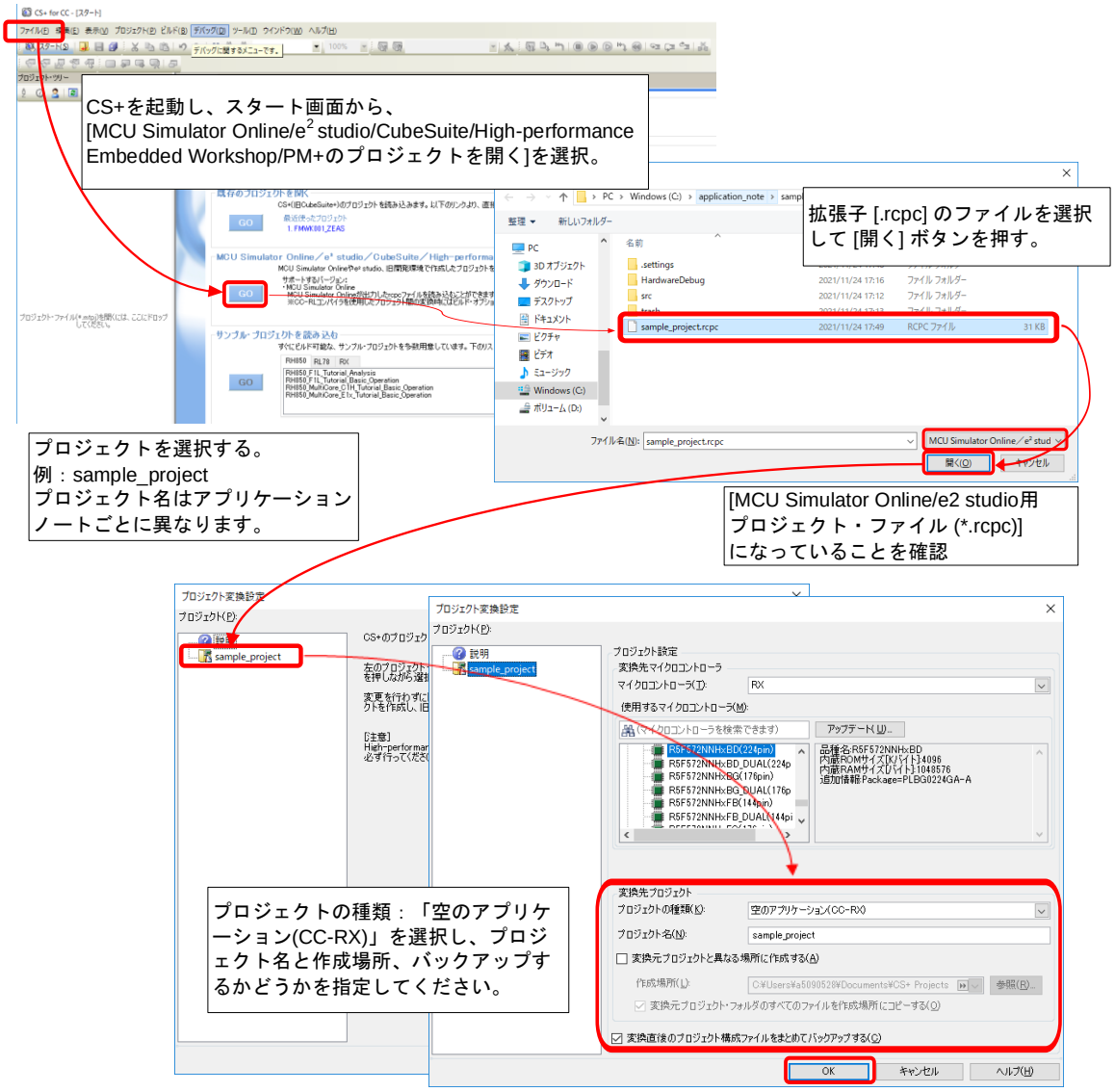


図5.2　プロジェクトを CS+にインポートする方法

6. 注意事項

6.1 ビット操作命令の注意事項

8 ビットの I/O レジスタに対してビット操作する場合、C 言語の記述方法によってはメモリアクセスを伴うビット操作命令が出力されない可能性があります。

例えば、下記の<C 言語記述例>のように、汎用ポートのビット操作の式の右辺を変数とした場合、コンパイル結果が<命令展開例>の命令に展開される可能性があります。

この場合、同じ I/O レジスタの他のビットを変更する割り込み処理が存在し、その割り込みが<命令展開例>の”A3”のタイミングの前後で発生すると、割り込み処理で変更した値が反映されません。

<C 言語記述例>

```
unsigned char i;
i=1;
PORTD.PODR.BIT.B6 = i;
```

<命令展開例>

```
A1 : mov.l      #0x8c02d, r14
A2 : mov.b      [r14], r15
A3 : bset       #6, r15
A4 : mov.b      r15, [r14]
```

対策として、以下のように式の右辺を即値とすることで、メモリアクセスを伴うビット操作命令が出力されます(CC-RX V2.06 以降)。

```
if( 0 == i)
{
    PORTD.PODR.BIT.B6 = 0;
}
else
{
    PORTD.PODR.BIT.B6 = 1;
}
```

また、CC-RX コンパイラでは組み込み関数を用意しています。

詳細は、「CC-RX コンパイラ ユーザーズマニュアル(R20UT3248)」をご参照ください。

7. 参考資料

- ユーザーズマニュアル：ハードウェア
- RX660 グループ ユーザーズマニュアル ハードウェア編 (R01UH0937)
- （最新版をルネサスエレクトロニクスホ―ムペ―ジから入手してください。）
-
- アプリケーションノート
- RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
- （最新版をルネサスエレクトロニクスホ―ムペ―ジから入手してください。）
-
- ユーザーガイド：スマート・コンフィグレータ
- Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
- （最新版をルネサスエレクトロニクスホ―ムペ―ジから入手してください。）
-
- ユーザーズマニュアル：コンパイラ
- 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―ムペ―ジから入手してください。）

- ユーザーズマニュアル：RSK
- Renesas Starter Kit for RX660 ユーザーズマニュアル (R20UT5017)
- （最新版をルネサスエレクトロニクスホ―ムペ―ジから入手してください。）
-
- 回路図：RSK
- Renesas Starter Kit for RX660 CPU ボード回路図 (R20UT5016)
- （最新版をルネサスエレクトロニクスホ―ムペ―ジから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jan.22.24	-	初版

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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