TRAVEO™ T2G ファミリのシリアル通信ブロック (SCB) 使用方法

本書について

適用範囲と目的

AN225401 では、TRAVEO™ T2G ファミリ MCU のシリアル通信ブロックの設定方法と使用方法を 3 つのシリアルインタフェースプロトコル (SPI, UART, および I2C) で説明します。

関連製品ファミリ

TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4 シリーズ

対象者

このドキュメントは、TRAVEO™ T2G ファミリを使用するすべての人を対象とします。

目次

目次

1 はじめに

このアプリケーションノートでは、TRAVEO™ T2G CYT2/CYT3/CYT4 シリーズ MCU のシリアル通信ブロック (SCB) の使い方を説明します。SCB は、他のデバイスとのシリアル通信に使用します。3 種類 (SPI, UART, および I2C) のシリアル通信プロトコルに対応しています。

このアプリケーションノートは、SCB の機能, 初期設定, およびユースケースでの通信制御手順を説明します。このアプリケーションノートにおいて、記述する機能, 使用される技術を理解するためには、Architecture technical reference manual (TRM)の Serial Communications Block (SCB) の章を参照してください。

1.1 機能

本 SCB は、以下の機能をサポートします。

- 標準の SPI マスタとスレーブ機能 (Motorola, Texas Instruments, および National Semiconductor のプロトコル)
- 標準の UART 機能 (Smart Card リーダ, Local Interconnect Network (LIN), および IrDA のプロトコル)
  - 標準 LIN スレーブ機能 (LIN v1.3, LIN v2.1/2.2 規格準拠)。TRAVEO™ T2G ファミリの SCB は、標準 LIN スレーブ機能のみサポートします。
- 標準 I2C マスタとスレーブ機能
- DeepSleep モード対応は、SCB[0]のみ
- EZ モード (SPI と I2C スレーブにおいて CPU の介在なく制御可能)
- CMD_RESP モード (SPI と I2C スレーブにおいて CPU の介在なく制御可能。DeepSleep 機能を有した SCB のみ有効)
- 低消費電力 (DeepSleep) モード (外部クロックを使用した SPI と I2C スレーブ時。DeepSleep 機能を有した SCB のみ有効)
- I2C スレーブアドレスの一致、または SPI スレーブ選択時の DeepSleep ウェイクアップ機能 (DeepSleep 機能を有した SCB のみ有効)
- DMA 接続用トリガ出力
- FIFO と転送の状態を示す複数の割込みソース

2 概要

本 SCB は、3 種類のシリアル通信プロトコルをサポートしています (SPI, UART, および I2C)。同時に 1 つのプロトコルだけが選択できます。

本 SCB は、標準 LIN のスレーブ機能のみサポートしています。そのため、SCB の UART-LIN は、LIN マスタとして使用できません。サポートされるハードウェアや LIN マスタ機能の詳細については、Architecture TRM の LIN ブロックを参照してください。

図 1 に、SCB のブロックダイヤグラムを示します。

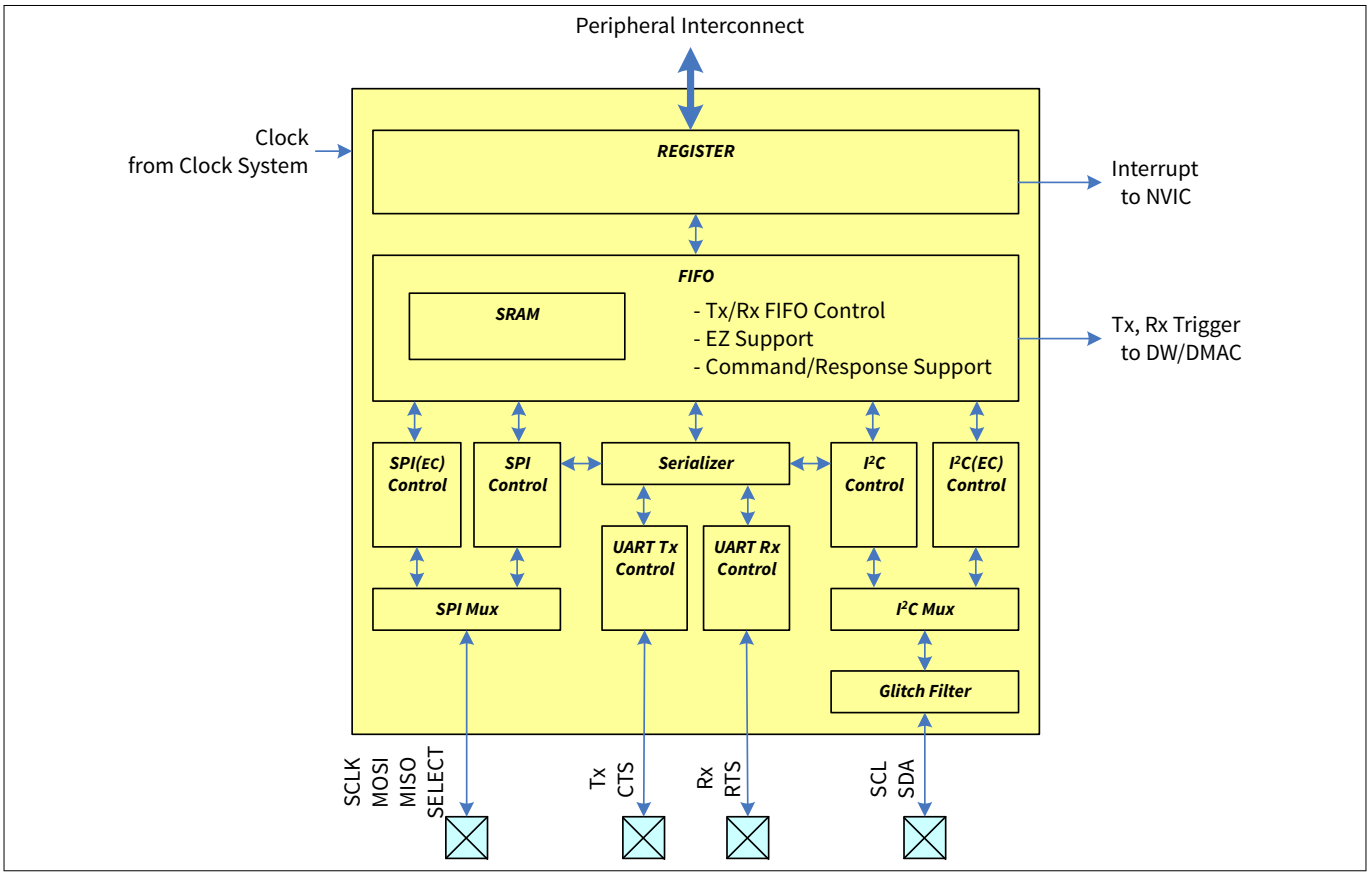

図 1 SCB ブロックダイヤグラム

本 SCB は、レジスタ, FIFO, およびそれぞれのプロトコル (SPI, UART, および I2C) のための制御ブロックから構成されます。レジスタは、SCB 設定や様々なイベントによる割込み生成のためのソフトウェアインタフェースとして使用されます。FIFO は、SRAM (256 バイト) で構成され、3 つのモード Tx/Rx FIFO (128x8 ビット/64x16 ビット/32x32 ビット), EZ (256x8 ビット), および Command/Response (256x8 ビット) を持ちます。それぞれのプロトコル制御ブロックは、送信と受信コントローラとして機能します。SPI (すべての SCB) と I2C (SCB[0]) は、スレーブモードにおいて、外部クロック (EC) モードをサポートします。

3 SPI 設定手順例

ここでは、サンプルドライバライブラリ (SDL) を使用した SPI の例を示します。SCB は、Motorola, Texas Instruments, および National Semiconductor プロトコルの SPI マスタモードと SPI スレーブモードをサポートします。各プロトコルの詳細については、Architecture TRM を参照してください。このアプリケーション ノートのプログラムコードは SDL の一部であり、CYT2B7 シリーズに基づいています。SDL については、その他の参考資料を参照してください。

SDL には、設定部とドライバ部があります。設定部では、主に目的の操作のパラメーター値を設定します。ドライバ部は、設定部のパラメーター値に基づいて各レジスタを設定します。システムに応じて設定部を設定できます。

3.1 マスタモード

この例では、SCB を Motorola の SPI マスタモードに設定し、2 ワード (ワード: 16 ビット) のデータを送信し、SPI スレーブから 2 ワードのオプションデータを受信します。

3.1.1 ユースケース

以下は、スケジューラ周期 (100 ms) でデータを送信し、Rx 割込みでデータを受信する例です。

- SCB モード = Motorola SPI マスタモード
- SCB チャネル = 1
- PCLK (周辺クロック) = 4 MHz
- ビットレート = 1 Mbps (OVS:オーバサンプリングされた倍数。Architecture TRM の Serial Communications Block (SCB) の章を参照してください。)

[ビットレート設定]

ビットレートの設定はマスタモードのみ有効です。ビット レートの計算式は次のとおりです: ビットレート[bps]＝入力クロック [Hz] / OVS。OVS : SCB_CTRL.OVS + 1。この場合、ビットレートは次のように計算されます。ビットレート = 入力クロック [Hz] / OVS = PCLK(4MHz) / (3+1) = 1 [Mbps]。詳細については、Architecture TRM を参照してください。

- Tx/Rx データ幅 = 16 ビット
- Tx/Rx FIFO = 使用 (16 ビット FIFO データエレメント)
- Rx 割込み = 有効
  - システム割込みソース : scb_0_interrupt_IRQn (IDX: 17)
  - CPU 割込みへのマップ : IRQ3
  - CPU 割込み優先度 : 3
- 使用ポート
  - SCLK : SCB0_CLK (P0.2)

– MOSI : SCB0_MOSI (P0.1)。MOSI データは、SCLK の立ち下りエッジで駆動されます。

– MISO : SCB0_MISO (P0.0)。MISO データは、立ち上りエッジから SPI SCLK で半周期後の SCLK の立ち下りエッジでキャプチャされます。

– SELECT : SCB0_SEL0 (P0.3)。図 2 は、SCB と他の SPI デバイス間の接続例を示します。

3 SPI 設定手順例

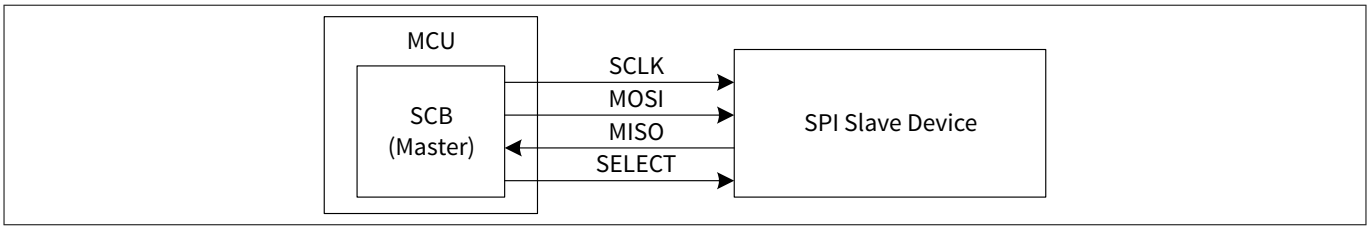

図 2　　SPI (マスタモード) 接続の例

SPI モードでは SCLK, MOSI, MISO, および SELECT 信号は、他のスレーブデバイスと接続されます。マスタモードでは、SCLK と MOSI は出力、MISO は入力です。SELECT は、スレーブデバイスに対する有効データ期間を示すために使用します。最大 4 本の SELECT 信号を割当て可能です。

図 3 にマスタモードの設定手順と動作の例を示します。

3 SPI 設定手順例

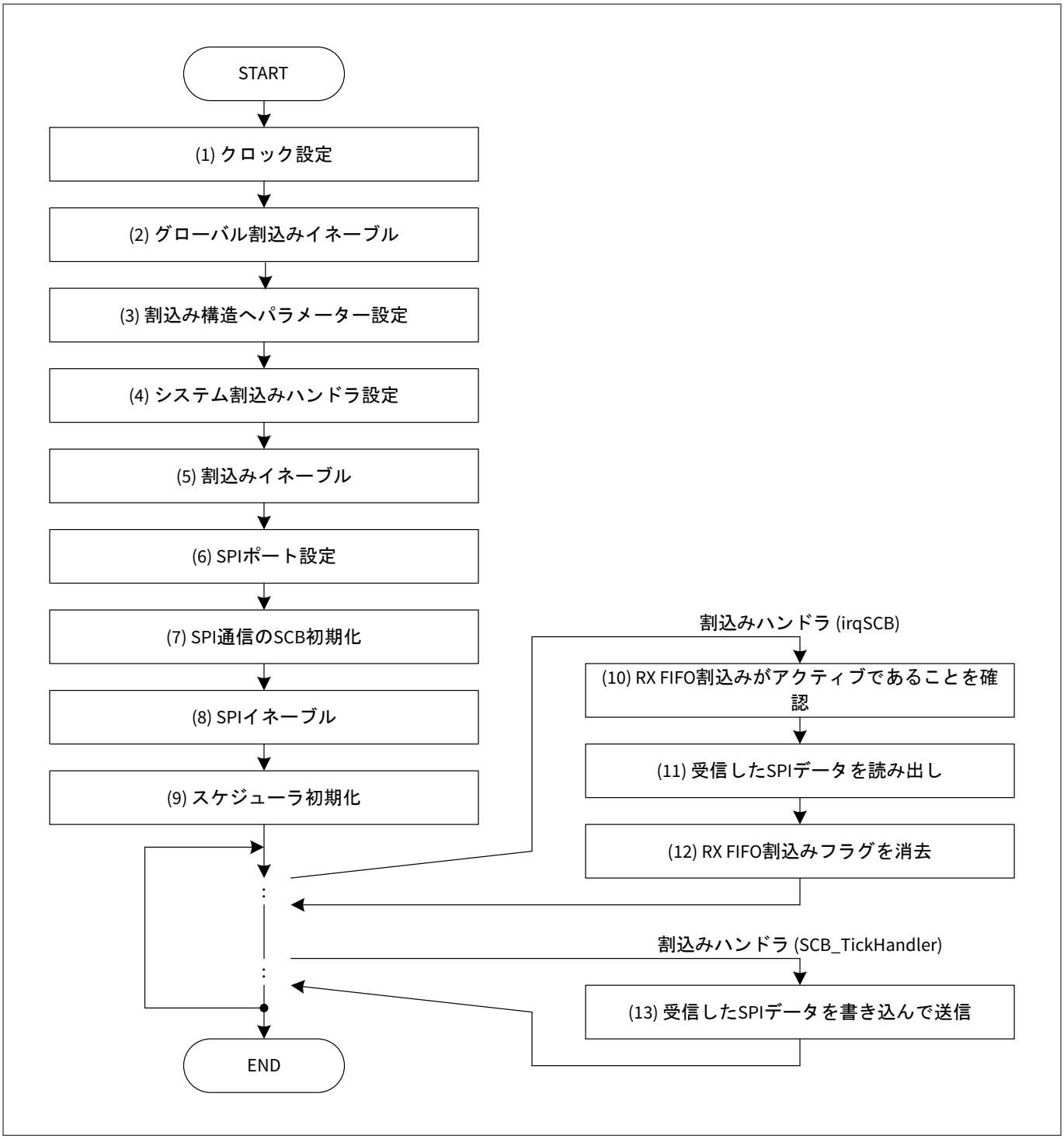

図 3　　SPI マスタモード動作

(1) クロックを設定

(2) グローバル割込みを有効にします (CPU 割込み有効)。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(3) 割込み構造を設定します。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(4) システム割込みハンドラを設定します。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(5) 割込みを有効にします。

(6) SPI ポートをマスタモードに設定します。SCLK, MOSI, および SELECT が出力。MISO が入力。
(7) SPI 通信用に SCB を初期化します。
(8) SPI を有効にします。
(9) SPI 送信データのスケジューラを初期化して起動します。
(10) マスタが任意のデータを受信すると、RX FIFO 割込みが発生します。
(11) ソフトウェアは、Rx FIFO から受信データを読み出します。
(12) RX FIFO 割込みをクリアします。
(13) スケジューラ割込みが発生したら、送信する 16 ビットのデータを SCB_TX_FIFO_WR レジスタに書き込みます。2 つ以上のバイトが FIFO に書き込まれるとすぐに、SCB は送信を開始します。(SCB_TX_FIFO_WR = transmit_data)

3.1.2 設定および例

表 1 に、SPI マスタ モードの SDL の設定部のパラメーターを示します。

表 1 SPI マスタ モードの設定パラメーター

パラメーター	説明	設定値
SOURCE_CLOCK_FRQ	入力分周クロック周波数	80000000ul (80MHz)
SCB_SPI_BAUDRATE	SPI ボーレート	125000ul
SCB_SPI_OVERSAMPLING	SPI のオーバサンプリング	16ul
SCB_SPI_CLOCK_FREQ	周辺クロック周波数	SCB_SPI_BAUDRATE*SCB_SPI_OVERSAMPLING (125000*16 = 2 MHz)
DIVIDER_NO_1	分周器番号	1ul
CY_SPI_SCB_PCLK	周辺クロック番号	PCLK_SCB0_CLOCK
CY_SPI_SCB_TYPE	SCB チャネル番号	SCB0
SCB_SPI_cfg.spiMode	動作モード	CY_SCB_SPI_MASTER (0x1)
SCB_SPI_cfg.subMode	SPI 動作サブモード	CY_SCB_SPI_MOTOROLA (0x0)
SCB_SPI_cfg.sclkMode	クロックの極性と位相	CY_SCB_SPI_CPHA0_CPOL0 (0x0)
SCB_SPI_cfg.oversample	オーバサンプリング係数	SCB_SPI_OVERSAMPLING (16ul)
SCB_SPI_cfg.rxDataWidth	Rx データ フレーム幅	16ul
SCB_SPI_cfg.txDataWidth	Tx データ フレーム幅	16ul
SCB_SPI_cfg.enableMsbFirst	最下位ビットまたは最上位ビットが最初	true (MSB)
SCB_SPI_cfg.enableFreeRunSclk	SCLK 生成モード	false
SCB_SPI_cfg.enableInputFilter	メディアン フィルタ	false
SCB_SPI_cfg.enableMisoLateSample	MISO がキャプチャされる SCLK エッジ	true (determined by CPOL and CPHA)
SCB_SPI_cfg.enableTransferSeperation	連続 SPI データ転送	true (disable)

(続く)

3 SPI 設定手順例

表 1　　(続き) SPI マスタ モードの設定パラメーター

パラメーター	説明	設定値
SCB_SPI_cfg.ssPolarity0	SELECT 1 の極性	false ('0' active, used)
SCB_SPI_cfg.ssPolarity1	SELECT 2 の極性	false ('0' active, unused)
SCB_SPI_cfg.ssPolarity2	SELECT 3 の極性	false ('0' active, unused)
SCB_SPI_cfg.ssPolarity3	SELECT 4 の極性	false ('0' active, unused)
SCB_SPI_cfg.enableWakeFromSleep	スレーブ選択検出ロジック	false
SCB_SPI_cfg.rxFifoTriggerLevel	Rx FIFO のトリガーレベル	1ul
SCB_SPI_cfg.rxFifoIntEnableMask	Rx FIFO の割込み	1ul
SCB_SPI_cfg.txFifoTriggerLevel	Tx FIFO のトリガーレベル	0ul
SCB_SPI_cfg.txFifoIntEnableMask	Tx FIFO の割込み	0ul
SCB_SPI_cfg.enableSpiDoneInterrupt	SPI マスタ転送完了イベント	false
SCB_SPI_cfg.enableSpiBusErrorInterrupt	SPI 転送で予期しない時間に SPI スレーブが選択解除された	false
CY_SPI_SCB_MISO_PORT	I/O ポート番号	GPIO_PRT0
CY_SPI_SCB_MISO_PIN	I/O ピン番号	0ul
CY_SPI_SCB_MISO_MUX	周辺機器接続	P0_0_SCB0_SPI_MISO (30ul)
CY_SPI_SCB_MOSI_PORT	I/O ポート番号	GPIO_PRT0
CY_SPI_SCB_MOSI_PIN	I/O ピン番号	1ul
CY_SPI_SCB_MOSI_MUX	周辺機器接続	P0_1_SCB0_SPI_MOSI
CY_SPI_SCB_CLK_PORT	I/O ポート番号	GPIO_PRT0
CY_SPI_SCB_CLK_PIN	I/O ピン番号	2ul
CY_SPI_SCB_CLK_MUX	周辺機器接続	P0_2_SCB0_SPI_CLK
CY_SPI_SCB_SEL0_PORT	I/O ポート番号	GPIO_PRT0
CY_SPI_SCB_SEL0_PIN	I/O ピン番号	3ul
CY_SPI_SCB_SEL0_MUX	周辺機器接続	P0_3_SCB0_SPI_SELECT0
SCB_MISO_DRIVE_MODE	MISO の DRIVE_MODE	CY_GPIO_DM_HIGHZ (0x08)
SCB_MOSI_DRIVE_MODE	MOSI の DRIVE_MODE	CY_GPIO_DM_STRONG_IN_OFF (0x06)
SCB_CLK_DRIVE_MODE	CLK の DRIVE_MODE	CY_GPIO_DM_STRONG_IN_OFF (0x06)
SCB_SEL0_DRIVE_MODE	SEL0 の DRIVE_MODE	CY_GPIO_DM_STRONG_IN_OFF (0x06)
SPI_port_pin_cfg.outVal	ピン出力状態	0ul
SPI_port_pin_cfg.driveMode	GPIO 駆動モード	0ul
SPI_port_pin_cfg.hsiom	I/O ピン配線の接続	HSIOM_SEL_GPIO (0x0)

(続く)

3 SPI 設定手順例

表 1 (続き) SPI マスタ モードの設定パラメーター

パラメーター	説明	設定値
SPI_port_pin_cfg.intEdge	IRQ をトリガするエッジ	0ul
SPI_port_pin_cfg.intMask	エッジ割込みをマスク	0ul
SPI_port_pin_cfg.vtrip	入力バッファ モード	0ul
SPI_port_pin_cfg.slewRate	スルーレート	0ul
SPI_port_pin_cfg.driveSel	GPIO 駆動強度	0ul
CY_SPI_SCB_IRQN	システム割込みインデックス番号	scb_0_interrupt_IRQn (IDX: 17)
irq_cfg.sysIntSrc	システム割込みインデックス番号	CY_SPI_SCB_IRQN
irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
.isEnabled	CPU 割込みイネーブル	true (Enable)

表 2 に、SDL のドライバ部の機能を示します。

表 2 機能一覧

機能	説明	備考
Cy_SysClk_PeriphAssignDivider (en_clk_dst_t ipBlock, cy_en_divider_types_t dividerType, uint32_t dividerNum)	選択した IP ブロックにプログラム可能な分周器を割り当てます	ipBlock: CY_SPI_SCB_PCLK dividerType: CY_SYSCLK_DIV_24_5_BIT dividerNum: DIVIDER_NO_1
Cy_SysClk_PeriphSetFracDivider (cy_en_divider_types_t dividerType, uint32_t dividerNum, uint32_t dividerIntValue, uint32_t dividerFracValue)	プログラム可能なクロック分周器の 1 つを設定します	dividerType: CY_SYSCLK_DIV_24_5_BIT dividerNum: DIVIDER_NO_1 dividerIntValue: ((divSetting >> 5ul) & 0x00000FFFul) - 1ul dividerFracValue: divSetting & 0x0000001Ful
Cy_SysClk_PeriphEnableDivider (cy_en_divider_types_t dividerType, uint32_t dividerNum)	選択した分周器を有効にします	dividerType: CY_SYSCLK_DIV_24_5_BIT dividerNum: 1ul

表 3 に、SDL のドライバ部の SPI 関数を示します。

表 3 SPI 関数

関数	説明	備考
Cy_SCB_SPI_DeInit (volatile stc_SCB_t *base)	SCB ブロックを初期化解除します	*base: CY_SPI_SCB_TYPE
Cy_SCB_SPI_Init (volatile stc_SCB_t *base, cy_stc_scb_spi_config_t const *config, cy_stc_scb_spi_context_t *context)	SPI 動作のために SCB を初期化します	*base: CY_SPI_SCB_TYPE *config: SCB_SPI_cfg context: NULL

(続く)

3 SPI 設定手順例

表 3 (続き) SPI 関数

関数	説明	備考
`Cy_SCB_SPI_SetActiveSlaveSelect (volatile stc_SCB_t *base, uint32_t slaveSelect)`	利用可能な 4 つのうちの 1 つからアクティブなスレーブ選択ラインを選択します	*base: CY_SPI_SCB_TYPE slaveSelect: 0ul
`Cy_SCB_SPI_Enable (volatile stc_SCB_t *base)`	SPI 動作用の SCB ブロックを有効にします	*base: CY_SPI_SCB_TYPE
`Cy_SCB_SPI_GetRxFifoStatus (volatile stc_SCB_t const *base)`	Rx FIFO の現在のステータスを返します	*base: CY_SPI_SCB_TYPE
`Cy_SCB_SPI_ReadArray (volatile stc_SCB_t const *base, void *rxBuf, uint32_t size)`	SPI Rx FIFO からデータアレイを読み出します	*base: CY_SPI_SCB_TYPE *rxBuf: (void*)readData size: 2ul
`Cy_SCB_SPI_ClearRxFifoStatus (volatile stc_SCB_t *base, uint32_t clearMask)`	Rx FIFO の選択されたステータスをクリアします	*base: CY_SPI_SCB_TYPE clearMask: CY_SCB_SPI_RX_TRIGGER
`Cy_SCB_SPI_WriteArray (volatile stc_SCB_t *base, void *txBuf, uint32_t size)`	データアレイを SPI Tx FIFO に配置します	*base: CY_SPI_SCB_TYPE *txBuf: (void*)readData size: 2ul

Code Listing 1 に、設定部で SPI マスタ モードを設定する例を示します。

3 SPI 設定手順例

Code Listing 1 設定部で SPI マスタモードを設定する例

```
/* Device Specific Settings */
#define CY_SPI_SCB_TYPE      SCB0  /* Define the SCB channel */

#define CY_SPI_SCB_MISO_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MISO_PIN  0ul  /* Define the port settings */
#define CY_SPI_SCB_MISO_MUX  P0_0_SCB0_SPI_MISO  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PIN  1ul  /* Define the port settings */
#define CY_SPI_SCB_MOSI_MUX  P0_1_SCB0_SPI_MOSI  /* Define the port settings */
#define CY_SPI_SCB_CLK_PORT  GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_CLK_PIN   2ul  /* Define the port settings */
#define CY_SPI_SCB_CLK_MUX   P0_2_SCB0_SPI_CLK  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PIN  3ul  /* Define the port settings */
#define CY_SPI_SCB_SEL0_MUX  P0_3_SCB0_SPI_SELECT0  /* Define the port settings */

#define CY_SPI_SCB_PCLK      PCLK_SCB0_CLOCK  /* Define the peripheral clock */

#define CY_SPI_SCB_IRQN      scb_0_interrupt_IRQn  /* Define the System interrupt index number
*/

/* Master Settings */
#define SCB_MISO_DRIVE_MODE CY_GPIO_DM_HIGHZ  /* Define the port settings */
#define SCB_MOSI_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */
#define SCB_CLK_DRIVE_MODE  CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */
#define SCB_SEL0_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */

/* User setting value */
#define SOURCE_CLOCK_FRQ 80000000ul
#define SCB_SPI_BAUDRATE     125000ul /* Please set baudrate value of SPI you want */
#define SCB_SPI_OVERSAMPLING 16ul     /* Please set oversampling of SPI you want */  /*Define
the clock parameters */
#define SCB_SPI_CLOCK_FREQ (SCB_SPI_BAUDRATE * SCB_SPI_OVERSAMPLING)

#define DIVIDER_NO_1 (1ul)

static cy_stc_gpio_pin_config_t SPI_port_pin_cfg =  /* Configure the port setting parameters */
{
    .outVal    = 0ul,
    .driveMode = 0ul,            /* Will be updated in runtime */
    .hsiom     = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

static cy_stc_sysint_irq_t irq_cfg =
```

3 SPI 設定手順例

```
{
    .sysIntSrc  = CY_SPI_SCB_IRQN,  /* Configure the interrupt structure parameters*1 */
    .intIdx     = CPUIntIdx3_IRQn,
    .isEnabled  = true,
};
uint16_t readData[2];

static const cy_stc_scb_spi_config_t SCB_SPI_cfg =  /* Configure the SCB parameters */
{
    .spiMode                    = CY_SCB_SPI_MASTER,      /*** Specifies the mode of
operation    ***/
    .subMode                    = CY_SCB_SPI_MOTOROLA,    /*** Specifies the sub mode of SPI
operation    ***/
    .sclkMode                   = CY_SCB_SPI_CPHA0_CPOL0, /*** Clock is active low, data is
changed on first edge ***/
    .oversample                 = SCB_SPI_OVERSAMPLING,   /*** SPI_CLOCK divided by
SCB_SPI_OVERSAMPLING should be baudrate  ***/
    .rxDataWidth                = 16ul,     /*** The width of RX data (valid range 4-16). It
must be the same as \ref txDataWidth except in National sub-mode. ***/
    .txDataWidth                = 16ul,     /*** The width of TX data (valid range 4-16). It
must be the same as \ref rxDataWidth except in National sub-mode. ***/
    .enableMsbFirst             = true,     /*** Enables the hardware to shift out the data
element MSB first, otherwise, LSB first ***/
    .enableFreeRunSclk          = false,    /*** Enables the master to generate a continuous
SCLK regardless of whether there is data to send  ***/
    .enableInputFilter          = false,    /*** Enables a digital 3-tap median filter to be
applied to the input of the RX FIFO to filter glitches on the line. ***/
    .enableMisoLateSample       = true,     /*** Enables the master to sample MISO line one
half clock later to allow better timings. ***/
    .enableTransferSeperation   = true,     /*** Enables the master to transmit each data
element separated by a de-assertion of the slave select line (only applicable for the master
mode) ***/
    .ssPolarity0                = false,    /*** SS0: active low ***/
    .ssPolarity1                = false,    /*** SS1: active low ***/
    .ssPolarity2                = false,    /*** SS2: active low ***/
    .ssPolarity3                = false,    /*** SS3: active low ***/
    .enableWakeFromSleep        = false,    /*** When set, the slave will wake the device when
the slave select line becomes active. Note that not all SCBs support this mode. Consult the
device datasheet to determine which SCBs support wake from deep sleep. ***/
    .rxFifoTriggerLevel         = 1ul,      /*** Interrupt occurs, when there are more entries
of 2 in the RX FIFO */
    .rxFifoIntEnableMask        = 1ul,      /*** Bits set in this mask will allow events to
cause an interrupt  */
    .txFifoTriggerLevel         = 0ul,      /*** When there are fewer entries in the TX FIFO,
then at this level the TX trigger output goes high. This output can be connected to a DMA
channel through a trigger mux. Also, it controls the \ref CY_SCB_SPI_TX_TRIGGER interrupt
source. */
    .txFifoIntEnableMask        = 0ul,      /*** Bits set in this mask allow events to cause an
interrupt  ***/
    .masterSlaveIntEnableMask   = 0ul,      /*** Bits set in this mask allow events to cause an
interrupt  ***/
    .enableSpiDoneInterrupt     = false,
    .enableSpiBusErrorInterrupt = false,
```

3 SPI 設定手順例

```
};

/* Master schedule handler */
static void SCB_TickHandler(void)
{
    Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE,(void*)readData, 2ul);
}

static void SchedulerInit(void)
{
    Cy_SysTick_Init(CY_SYSTICK_CLOCK_SOURCE_CLK_CPU, CORE_CLOCK_FRQ / 10ul); // 100[ms]
    Cy_SysTick_SetCallback(0ul, SCB_TickHandler);
    Cy_SysTick_Enable();
}

int main(void)
{
    SystemInit();

    /*****************************************************/
    /******* Calculate divider setting for the SCB ********/  /* (1) Configure the clock */
    /*****************************************************/
    Cy_SysClk_PeriphAssignDivider(CY_SPI_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);  /*
Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(SCB_SPI_CLOCK_FREQ, SOURCE_CLOCK_FRQ, DIVIDER_NO_1);  /* Configure
the divider (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 1ul);  /* Enable the divider (See
Code Listing 6) */



    __enable_irq(); /* Enable global interrupts. */  /* (2) Enable global interrupt*1 */



    /*****************************************/
    /*    De-initialization for peripherals     */
    /*****************************************/
    Cy_SCB_SPI_DeInit(CY_SPI_SCB_TYPE);  /* De-Initialize the SCB if necessary (See Code
Listing 7) */




    /*****************************************/
    /* Interrupt setting for SPI communication */
    /*****************************************/
    Cy_SysInt_InitIRQ(&irq_cfg);  /* (3) Set the parameters to interrupt structure*1 */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqSCB);  /* (4) Set the system interrupt
handler*1 (See Code Listing 3) */
    NVIC_EnableIRQ(irq_cfg.intIdx);  /* (5) Interrupt Enable*1 */
```

3 SPI 設定手順例

```
    /*************************************/
    /* Port Setting for SPI communication */  /* (6) Set the SPI port*2 */
    /*************************************/
    /* According to the HW environment to change SCB CH*/


    SPI_port_pin_cfg.driveMode = SCB_MISO_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MISO_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_MISO_PORT, CY_SPI_SCB_MISO_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_MOSI_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MOSI_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_MOSI_PORT, CY_SPI_SCB_MOSI_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_CLK_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_CLK_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_CLK_PORT,CY_SPI_SCB_CLK_PIN, &SPI_port_pin_cfg);  /* Change the
driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_SEL0_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_SEL0_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_SEL0_PORT, CY_SPI_SCB_SEL0_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */
]



    /*****************************************/
    /* SCB initialization for SPI communication */
    /*****************************************/
    Cy_SCB_SPI_Init(CY_SPI_SCB_TYPE, &SCB_SPI_cfg, NULL);  /* (7) Initialize SCB for SPI
communication (See Code Listing 8) */
    Cy_SCB_SPI_SetActiveSlaveSelect(CY_SPI_SCB_TYPE, 0ul);  /* Set the using cannel number (See
Code Listing 9) */
    Cy_SCB_SPI_Enable(CY_SPI_SCB_TYPE);  /* (8) Enable SPI (See Code Listing 10) */

    /*****************************************/
    /*      Write initial value to buffer       */
    /*****************************************/
```

3 SPI 設定手順例

```
    readData[0] = 0xAAAAul;  /* If necessary, Initialize the buffer values */
    readData[1] = 0xAAAAul;


    SchedulerInit();  /* (9) Initialize the scheduler */

    for(;;);
```

*1: 詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。
*2: 詳細については、Architecture TRM の I/O システムのセクションを参照してください。
Code Listing 2 に、分数クロック分周関数を示します。

Code Listing 2 SetPeripheFracDiv24_5() 関数

```
void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)  /* Create
the function to determine the divider division ratio */
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq);  /* Calculates the division ratio */
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
                                   (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
                                   (divSetting & 0x0000001Ful));  /* Set the division ratio
(See Code Listing 5) */
}
```

Code Listing 3 に、割込みハンドラの例を示します。

3 SPI 設定手順例

Code Listing 3 割込みハンドラの例

```
void irqSCB(void)
{
    uint32_t status;

    status = Cy_SCB_SPI_GetRxFifoStatus(CY_SPI_SCB_TYPE);  /* (10) Check the Interrupt is
Active (See Code Listing 11) */
    if(status & CY_SCB_SPI_RX_TRIGGER)
    {
        Cy_SCB_SPI_ReadArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul);  /* (11) Read the Received
SPI Data (See Code Listing 12) */
        Cy_SCB_SPI_ClearRxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_RX_TRIGGER);  /* (12) Clear
the RX TRIGGER Interrupt Flag (See Code Listing 13) */
    }
}




/* Master schedule handler */
static void SCB_TickHandler(void)  /* Interrupt handler for TX */
    Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE,(void*)readData, 2ul);  /* (13) Write and send the
Received SPI Data (See Code Listing 14) */
}
```

*1: 詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

Code Listing 4～Code Listing 6 に、ドライバ部で CLK を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- PERI->unCLOCK_CTL および unDIV は、Registers TRM に記載されている PERI_CLOCK_CTLx レジスタです。
- パフォーマンス改善策: レジスタ設定のパフォーマンスを向上させるために、SDL は完全な 32 ビット データをレジスタに書き込みます。各ビットフィールドは、ビット書込み可能なバッファで事前に生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
un_PERI_CLOCK_CTL_t tempCLOCK_CTL_RegValue;
tempCLOCK_CTL_RegValue.u32Register       = PERI->unCLOCK_CTL[ipBlock].u32Register;
tempCLOCK_CTL_RegValue.stcField.u2TYPE_SEL = dividerType;
tempCLOCK_CTL_RegValue.stcField.u8DIV_SEL = dividerNum;
PERI->unCLOCK_CTL[ipBlock].u32Register     = tempCLOCK_CTL_RegValue.u32Register;
```

- レジスタの共用体と構造表現の詳細については、*hdr/rev_x/ip* の下の *cy_sysclk.h* を参照してください。

3 SPI 設定手順例

Code Listing 4 Cy_SysClk_PeriphAssignDivider() 関数

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphAssignDivider(en_clk_dst_t ipBlock,
cy_en_divider_types_t dividerType, uint32_t dividerNum)
{
    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING)  /*
Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }

    un_PERI_CLOCK_CTL_t tempCLOCK_CTL_RegValue;
    tempCLOCK_CTL_RegValue.u32Register         = PERI->unCLOCK_CTL[ipBlock].u32Register;
    tempCLOCK_CTL_RegValue.stcField.u2TYPE_SEL = dividerType;
    tempCLOCK_CTL_RegValue.stcField.u8DIV_SEL  = dividerNum;
    PERI->unCLOCK_CTL[ipBlock].u32Register     = tempCLOCK_CTL_RegValue.u32Register;

    return CY_SYSCLK_SUCCESS;
}
```

3 SPI 設定手順例

Code Listing 5 Cy_SysClk_PeriphSetFracDivider() 関数

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphSetFracDivider(cy_en_divider_types_t
dividerType, uint32_t dividerNum, uint32_t dividerIntValue, uint32_t dividerFracValue)
{
    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING)  /*
Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }
    if (dividerType == CY_SYSCLK_DIV_16_5_BIT)
    {
        if ((dividerIntValue <= (PERI_DIV_16_5_CTL_INT16_DIV_Msk >>
PERI_DIV_16_5_CTL_INT16_DIV_Pos)) &&
            (dividerFracValue <= (PERI_DIV_16_5_CTL_FRAC5_DIV_Msk >>
PERI_DIV_16_5_CTL_FRAC5_DIV_Pos)))
    if (dividerType == CY_SYSCLK_DIV_16_5_BIT)
    {
        if ((dividerIntValue <= (PERI_DIV_16_5_CTL_INT16_DIV_Msk >>
PERI_DIV_16_5_CTL_INT16_DIV_Pos)) &&
            (dividerFracValue <= (PERI_DIV_16_5_CTL_FRAC5_DIV_Msk >>
PERI_DIV_16_5_CTL_FRAC5_DIV_Pos)))
        {
            PERI->unDIV_16_5_CTL[dividerNum].stcField.u16INT16_DIV = dividerIntValue;
            PERI->unDIV_16_5_CTL[dividerNum].stcField.u5FRAC5_DIV = dividerFracValue;
        }
        else
        {
            return CY_SYSCLK_BAD_PARAM;
        }
    }
    else if (dividerType == CY_SYSCLK_DIV_24_5_BIT)  /* Check the dividerType */
    {
        if ((dividerIntValue <= (PERI_DIV_24_5_CTL_INT24_DIV_Msk >>
PERI_DIV_24_5_CTL_INT24_DIV_Pos)) && (dividerFracValue <= (PERI_DIV_24_5_CTL_FRAC5_DIV_Msk >>
PERI_DIV_24_5_CTL_FRAC5_DIV_Pos)))
        {
            PERI->unDIV_24_5_CTL[dividerNum].stcField.u24INT24_DIV = dividerIntValue;  /*
Select INT24_DIV bits */
            PERI->unDIV_24_5_CTL[dividerNum].stcField.u5FRAC5_DIV = dividerFracValue;  /*
Select FRAC5_DIV bits */
        }
        else
        {
            return CY_SYSCLK_BAD_PARAM;
        }
    }
    else
    { /* return bad parameter */
        return CY_SYSCLK_BAD_PARAM;
    }
```

```
    return CY_SYSCLK_SUCCESS;
}
```

Code Listing 6 Cy_SysClk_PeriphEnableDivider() 関数

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphEnableDivider(cy_en_divider_types_t
dividerType, uint32_t dividerNum)
{

    if(Cy_SysClk_CheckDividerExisting(dividerType, dividerNum) == CY_DIVIDER_NOT_EXISTING)  /*
Check if configuration parameter values are valid */
    {
        return CY_SYSCLK_BAD_PARAM;
    }

    /* specify the divider, make the reference = clk_peri, and enable the divider */
    un_PERI_DIV_CMD_t tempDIV_CMD_RegValue;
    tempDIV_CMD_RegValue.u32Register            = PERI->unDIV_CMD.u32Register;
    tempDIV_CMD_RegValue.stcField.u1ENABLE      = 1ul;  /* Enable the Divider */
    tempDIV_CMD_RegValue.stcField.u2PA_TYPE_SEL = 3ul;
    tempDIV_CMD_RegValue.stcField.u8PA_DIV_SEL  = 0xFFul;
    tempDIV_CMD_RegValue.stcField.u2TYPE_SEL    = dividerType;
    tempDIV_CMD_RegValue.stcField.u8DIV_SEL     = dividerNum;
    PERI->unDIV_CMD.u32Register                 = tempDIV_CMD_RegValue.u32Register;

    (void)PERI->unDIV_CMD; /* dummy read to handle buffered writes */
    return CY_SYSCLK_SUCCESS;
}
```

Code Listing 7〜Code Listing 14 に、ドライバ部で SCB を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

3 SPI 設定手順例

Code Listing 7 Cy_SCB_SPI_DeInit() 関数

```
void Cy_SCB_SPI_DeInit(volatile stc_SCB_t *base)
{
    /* SPI interface */
    base->unCTRL.u32Register              = CY_SCB_CTRL_DEF_VAL;  /* Set the unCTRL reg */
    base->unSPI_CTRL.u32Register          = CY_SCB_SPI_CTRL_DEF_VAL;  /* Set the unSPI_CTRL Reg
*/

    /* RX direction */
    base->unRX_CTRL.u32Register           = CY_SCB_RX_CTRL_DEF_VAL;  /* Set the unRX_CTRL Reg */
    base->unRX_FIFO_CTRL.u32Register      = 0ul;  /* Set the unRX_FIFO_CTRL Reg to “0” */

    /* TX direction */
    base->unTX_CTRL.u32Register           = CY_SCB_TX_CTRL_DEF_VAL;  /* Set the unTX_CTRL Reg
*/
    base->unTX_FIFO_CTRL.u32Register      = 0ul;  /* Set the unTX_FIFO_CTRL Reg to “0” */

    /* Disable all interrupt sources */
    base->unINTR_SPI_EC_MASK.u32Register  = 0ul;  /* Disable the all interrupt */
    base->unINTR_I2C_EC_MASK.u32Register  = 0ul;  /* Disable the all interrupt */
    base->unINTR_RX_MASK.u32Register      = 0ul;  /* Disable the all interrupt */
    base->unINTR_TX_MASK.u32Register      = 0ul;  /* Disable the all interrupt */
    base->unINTR_M_MASK.u32Register       = 0ul;  /* Disable the all interrupt */
    base->unINTR_S_MASK.u32Register       = 0ul;  /* Disable the all interrupt */
}
```

3 SPI 設定手順例

Code Listing 8 Cy_SCB_SPI_Init() 関数

```
cy_en_scb_spi_status_t Cy_SCB_SPI_Init(volatile stc_SCB_t *base, cy_stc_scb_spi_config_t const
*config, cy_stc_scb_spi_context_t *context)
{
    cy_en_scb_spi_status_t retStatus = CY_SCB_SPI_BAD_PARAM;
    un_SCB_CTRL_t     tscbCtrl       = { 0ul };
    un_SCB_SPI_CTRL_t tscbSpiCtrl    = { 0ul };
    un_SCB_TX_CTRL_t  tscbTxCtrl     = { 0ul };
    un_SCB_RX_CTRL_t  tscbRxCtrl     = { 0ul };
    uint32_t          locSclkMode    = 0ul;
    uint32_t          maxOfDataWidth = 0ul;

    if ((NULL != base) && (NULL != config))  /* Check if configuration parameter values are
valid */
    {
        /* Set SCLK mode for TI - CY_SCB_SPI_CPHA1_CPOL0, NS - CY_SCB_SPI_CPHA0_CPOL0, Motorola
- take from config */
        if(CY_SCB_SPI_MOTOROLA == config->subMode)
        {
            locSclkMode = config->sclkMode;
        }
        else if(CY_SCB_SPI_NATIONAL == config->subMode)
        {
            locSclkMode = CY_SCB_SPI_CPHA0_CPOL0;
        }
        else
        {
            locSclkMode = CY_SCB_SPI_CPHA1_CPOL0;
        }
        maxOfDataWidth = (config->rxDataWidth >= config->txDataWidth) ? config->rxDataWidth :
config->txDataWidth;
        if ( maxOfDataWidth <= CY_SCB_BYTE_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_BYTE;
        }
        else if ( maxOfDataWidth <= CY_SCB_HALFWORD_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_HALFWORD;

        else if ( maxOfDataWidth <= CY_SCB_WORD_WIDTH )
        {
            tscbCtrl.stcField.u2MEM_WIDTH = CY_SCB_SPI_MEM_WIDTH_WORD;
        }
        else
        {
            return CY_SCB_SPI_BAD_PARAM;
        }

        if ( config->enableWakeFromSleep )  /* Set the SPI communication parameters */
        {
            tscbCtrl.stcField.u1EC_AM_MODE = true;
```

3 SPI 設定手順例

```
        }
        else
        {
            tscbCtrl.stcField.u1EC_AM_MODE = false;
        }

        tscbCtrl.stcField.u4OVS = (config->oversample - 1ul);  /* Set the SPI communication
parameters */
        tscbCtrl.stcField.u2MODE = CY_SCB_CTRL_MODE_SPI;  /* Set the SPI communication
parameters */
        base->unCTRL.u32Register = tscbCtrl.u32Register;  /* Set the SPI communication
parameters */

        tscbSpiCtrl.stcField.u1SSEL_CONTINUOUS = ~(config->enableTransferSeperation);  /* Set
the SPI communication parameters */
        tscbSpiCtrl.stcField.u1SELECT_PRECEDE = (0ul != (CY_SCB_SPI_TI_PRECEDE & config-
>subMode) ? 1ul : 0ul);  /* Set the SPI communication parameters */
        tscbSpiCtrl.stcField.u1LATE_MISO_SAMPLE = config->enableMisoLateSample;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1SCLK_CONTINUOUS = config->enableFreeRunSclk;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1MASTER_MODE = ((CY_SCB_SPI_MASTER == config->spiMode) ? 1ul :
0ul);  /* Set the SPI communication parameters */
        tscbSpiCtrl.stcField.u1CPHA = ((locSclkMode >> 1ul) & 0x01ul);  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1CPOL = (locSclkMode & 0x01ul);  /* Set the SPI communication
parameters */
        tscbSpiCtrl.stcField.u1SSEL_POLARITY0 = config->ssPolarity0;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1SSEL_POLARITY1 = config->ssPolarity1;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1SSEL_POLARITY2 = config->ssPolarity2;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u1SSEL_POLARITY3 = config->ssPolarity3;  /* Set the SPI
communication parameters */
        tscbSpiCtrl.stcField.u2MODE = config->subMode;  /* Set the SPI communication parameters
*/
        base->unSPI_CTRL.u32Register = tscbSpiCtrl.u32Register;  /* Set the SPI communication
parameters */

        tscbRxCtrl.stcField.u1MSB_FIRST = config->enableMsbFirst;  /* Set the SPI communication
parameters */
        tscbRxCtrl.stcField.u1MEDIAN = config->enableInputFilter;  /* Set the SPI communication
parameters */
        tscbRxCtrl.stcField.u5DATA_WIDTH = (config->rxDataWidth - 1ul);  /* Set the SPI
communication parameters */
        base->unRX_CTRL.u32Register = tscbRxCtrl.u32Register;  /* Set the SPI communication
parameters */
        base->unRX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->rxFifoTriggerLevel;  /* Set the
SPI communication parameters */
]
        tscbTxCtrl.stcField.u1MSB_FIRST = config->enableMsbFirst;  /* Set the SPI communication
parameters */
```

```
        tscbTxCtrl.stcField.u5DATA_WIDTH = (config->txDataWidth - 1ul);  /* Set the SPI
communication parameters */
        base->unTX_CTRL.u32Register = tscbTxCtrl.u32Register;  /* Set the SPI communication
parameters */
        base->unTX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->txFifoTriggerLevel;  /* Set the
SPI communication parameters */

        /* Set up interrupt sources */
        base->unINTR_TX_MASK.u32Register = config->txFifoIntEnableMask;  /* Set the SPI
interrupt  */
        base->unINTR_RX_MASK.u32Register = config->rxFifoIntEnableMask;  /* Set the SPI
interrupt  */
        base->unINTR_M.stcField.u1SPI_DONE = config->enableSpiDoneInterrupt;  /* Set the SPI
interrupt  */
        base->unINTR_S.stcField.u1SPI_BUS_ERROR = config->enableSpiBusErrorInterrupt;  /* Set
the SPI interrupt  */
        base->unINTR_SPI_EC_MASK.u32Register = 0ul;  /* Set the SPI interrupt  */

        /* Initialize the context */
        if (NULL != context)
        {
            context->status    = 0ul;  /* Set the TX/RX Buffer */
            context->txBufIdx  = 0ul;  /* Set the TX/RX Buffer */
            context->rxBufIdx  = 0ul;  /* Set the TX/RX Buffer */
            context->cbEvents = NULL;  /* Set the TX/RX Buffer */

        #if !defined(NDEBUG)
            /* Put an initialization key into the initKey variable to verify
            * context initialization in the transfer API.
            */
            context->initKey = CY_SCB_SPI_INIT_KEY;
        #endif /* !(NDEBUG) */
        }

        retStatus = CY_SCB_SPI_SUCCESS;
    }
    return (retStatus);
}
```

Code Listing 9 Cy_SCB_SPI_SetActiveSlaveSelect() 関数

```
__STATIC_INLINE void Cy_SCB_SPI_SetActiveSlaveSelect(volatile stc_SCB_t *base, uint32_t
slaveSelect)
{
    base->unSPI_CTRL.stcField.u2SSEL = slaveSelect;  /* Set the slave mode */
}
```

Code Listing 10 Cy_SCB_SPI_Enable() 関数

```
__STATIC_INLINE void Cy_SCB_SPI_Enable(volatile stc_SCB_t *base)
{
    base->unCTRL.stcField.u1ENABLED = true;  /* Set the slave mode */
}
```

Code Listing 11 Cy_SCB_SPI_GetRxFifoStatus() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_SPI_GetRxFifoStatus(volatile stc_SCB_t const *base)
{
    return (Cy_SCB_GetRxInterruptStatus(base) & CY_SCB_SPI_RX_INTR);  /* Read and check the Rx
Interrupt */
}
```

Code Listing 12 Cy_SCB_SPI_ReadArray() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_SPI_ReadArray(volatile stc_SCB_t const *base, void *rxBuf,
uint32_t size)
{
    return Cy_SCB_ReadArray(base, rxBuf, size);  /* Read the received data */
}
```

Code Listing 13 Cy_SCB_SPI_ClearRxFifoStatus() 関数

```
__STATIC_INLINE void Cy_SCB_SPI_ClearRxFifoStatus(volatile stc_SCB_t *base, uint32_t clearMask)
{
    Cy_SCB_ClearRxInterrupt(base, clearMask);  /* Clear the Rx Interrupt Factor */
}
```

Code Listing 14 Cy_SCB_SPI_WriteArray() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_SPI_WriteArray(volatile stc_SCB_t *base, void *txBuf, uint32_t
size)
{
    return Cy_SCB_WriteArray(base, txBuf, size);  /* Write and send the Received SPI Data */
}
```

3.2 スレーブモード

この例では、マスタがスレーブに対し 2 つのハーフワードデータを送信し、スレーブがマスタから 2 つのハーフワード受信するように、Motorola SPI スレーブモードを設定します。

3.2.1 ユースケース

- SCB モード = Motorola SPI スレーブモード
- SCB チャネル = 1
- PCLK = 4 MHz
- ビットレート = 1 Mbps
- Tx/Rx データ幅 = 16 ビット
- Tx/Rx FIFO = 使用 (16 ビット FIFO データエレメント)
- Tx/Rx 割込み = 有効
- 使用ポート
  - SCLK : SCB1_CLK (P18.2)
  - MOSI : SCB1_MOSI (P18.1) MOSI データは、SCLK の立ち下りエッジで駆動されます。
  - MISO : SCB1_MISO (P18.0) MISO データは、立ち上りエッジから SPI SCLK の半周期後に SCLK の立ち下りエッジでキャプチャされます。
  - SELECT : SCB1_SEL0 (P18.3)

図 4 は、SCB と他の SPI デバイスとの接続例を示します。

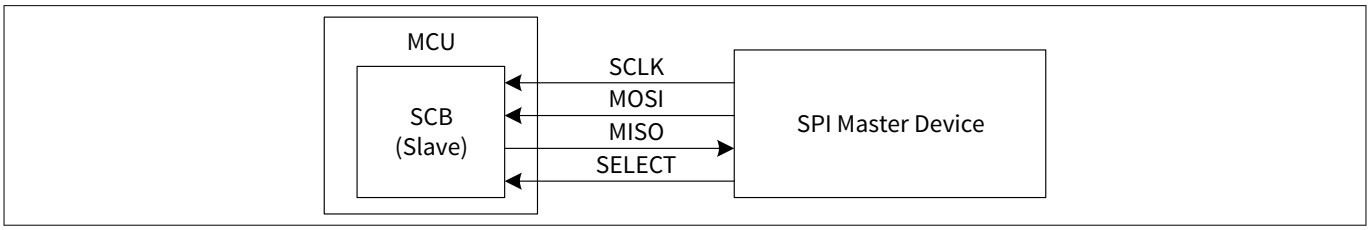

図 4 SPI (スレーブモード) 通信接続の例

SPI モードでは SCLK, MOSI, MISO, および SELECT 信号は、他のスレーブデバイスと接続されます。スレーブモードでは、SCLK, MOSI, および SELECT は入力、MISO は出力です。SELECT は、マスタデバイスもしくはスレーブデバイスから送信される有効データ期間を示すために使用されます。

図 5 にスレーブモードの設定手順と動作の例を示します。

3 SPI 設定手順例

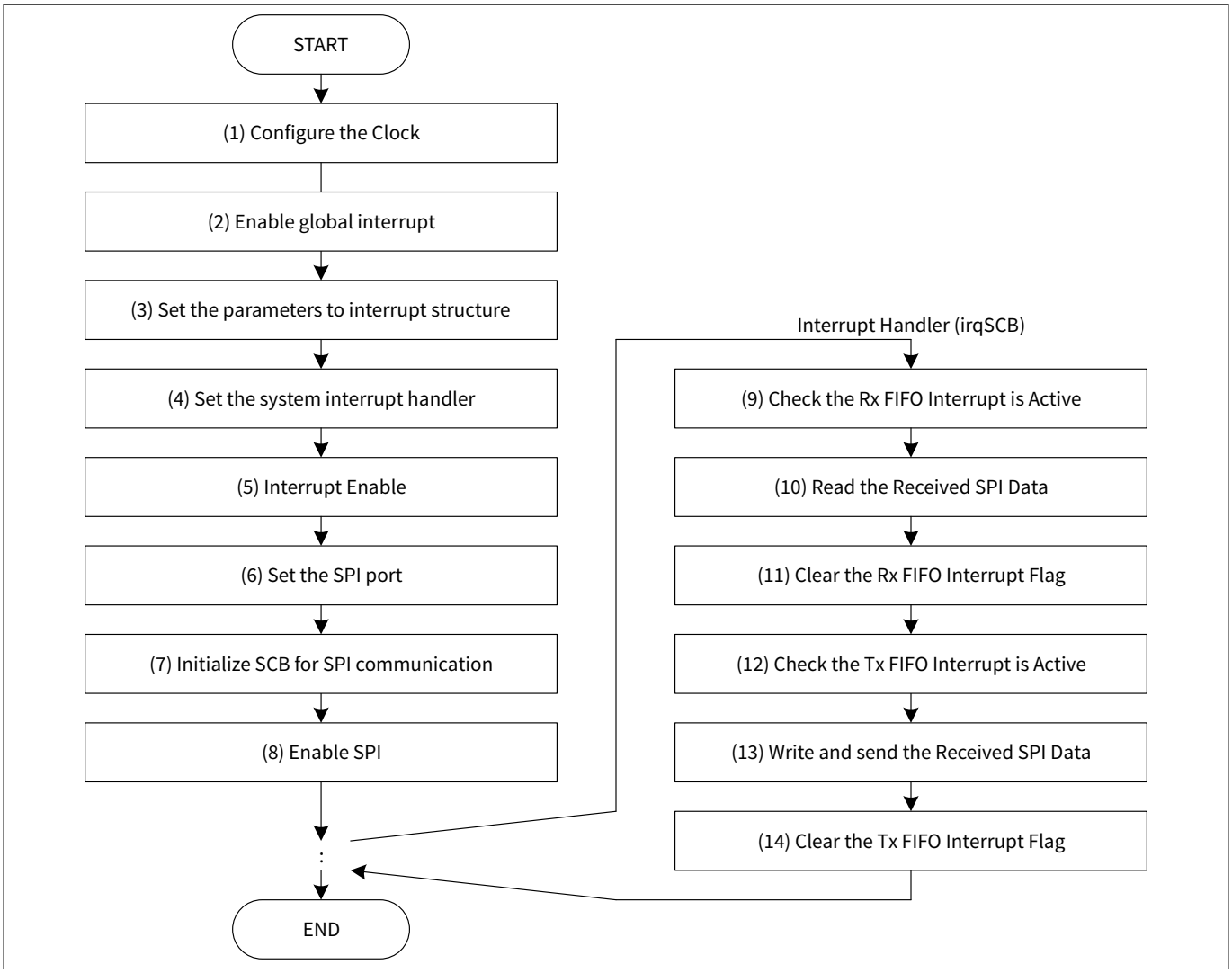

図 5　　SPI スレーブモード動作

(1) クロックを設定します。

(2) グローバル割込みを有効にします (CPU 割込み有効)。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(3) 割込み構造を設定します。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(4) システム割込みハンドラを設定します。詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

(5) 割込みを有効にします。

(6) SPI ポートをマスタモードに設定します。SCLK, MOSI, および SELECT が出力。MISO が入力。

(7) SPI 通信用に SCB を初期化します。

(8) SPI を有効にします。

(9) SCB が任意のデータを受信すると、Rx FIFO 割込みが発生します。

(10) ソフトウェアは、Rx FIFO から受信データを読み出します。

(11) Rx FIFO 割込みをクリアします。

(12) SCB が任意のデータを送信すると, Tx FIFO 割込みが発生します。

(13) ソフトウェアは、受信した SPI データを書き込んで送信します。

(14) Tx FIFO 割込みをクリアします。

3.2.2 設定および例

表 4 に、SPI スレーブ モードの SDL の設定部のパラメーターを示します。

表 4 SPI スレーブモード設定パラメーター

パラメーター	説明	設定値
SOURCE_CLOCK_FRQ	入力分周クロック周波数	80000000ul (80MHz)
SCB_SPI_BAUDRATE	SPI ボーレート	125000ul
SCB_SPI_OVERSAMPLING	SPI のオーバサンプリング	16ul
SCB_SPI_CLOCK_FREQ	周辺クロック周波数	SCB_SPI_BAUDRATE*SCB_SPI_OVERSAMPLING (125000*16 = 2 MHz)
DIVIDER_NO_1	分周器番号	1ul
CY_SPI_SCB_PCLK	周辺クロック番号	PCLK_SCB0_CLOCK
CY_SPI_SCB_TYPE	SCB チャネル番号	SCB0
SCB_SPI_cfg.spiMode	動作モード	CY_SCB_SPI_SLAVE (0x0)
SCB_SPI_cfg.subMode	SPI 動作サブモード	CY_SCB_SPI_MOTOROLA (0x0)
SCB_SPI_cfg.sclkMode	クロックの極性と位相	CY_SCB_SPI_CPHA0_CPOL0 (0x0)
SCB_SPI_cfg.oversample	オーバサンプリング係数	SCB_SPI_OVERSAMPLING (16ul)
SCB_SPI_cfg.rxDataWidth	Rx データ フレーム幅	16ul
SCB_SPI_cfg.txDataWidth	Tx データ フレーム幅	16ul
SCB_SPI_cfg.enableMsbFirst	最下位ビットまたは最上位ビットが最初	true (MSB)
SCB_SPI_cfg.enableFreeRunSclk	SCLK 生成モード	false
SCB_SPI_cfg.enableInputFilter	メディアン フィルタ	false
SCB_SPI_cfg.enableMisoLateSample	MISO がキャプチャされる SCLK エッジ	true (determined by CPOL and CPHA)
SCB_SPI_cfg.enableTransferSeperation	連続 SPI データ転送	true (enable)
SCB_SPI_cfg.ssPolarity0	SELECT 1 の極性	false ('0' active, used)
SCB_SPI_cfg.ssPolarity1	SELECT 2 の極性	false ('0' active, unused)
SCB_SPI_cfg.ssPolarity2	SELECT 3 の極性	false ('0' active, unused)
SCB_SPI_cfg.ssPolarity3	SELECT 4 の極性	false ('0' active, unused)
SCB_SPI_cfg.enableWakeFromSleep	スレーブ選択検出ロジック	false
SCB_SPI_cfg.rxFifoTriggerLevel	Rx FIFO のトリガーレベル	1ul
SCB_SPI_cfg.rxFifoIntEnableMask	Rx FIFO の割込み	1ul

(続く)

3 SPI 設定手順例

表 4 (続き) SPI スレーブモード設定パラメーター

パラメーター	説明	設定値
SCB_SPI_cfg.txFifoTriggerLevel	Tx FIFO のトリガーレベル	1ul
SCB_SPI_cfg.txFifoIntEnableMask	Tx FIFO の割込み	1ul
SCB_SPI_cfg.enableSpiDoneInterrupt	SPI マスタ転送完了イベント	false
SCB_SPI_cfg.enableSpiBusErrorInterrupt	SPI 転送で予期しない時間に SPI スレーブが選択解除された	false
CY_SPI_SCB_MISO_PORT	I/O 出力ポート番号	GPIO_PRT0
CY_SPI_SCB_MISO_PIN	I/O 出力ピン番号	0ul
CY_SPI_SCB_MISO_MUX	周辺機器接続	P0_0_SCB0_SPI_MISO (30ul)
CY_SPI_SCB_MOSI_PORT	I/O 出力ポート番号	GPIO_PRT0
CY_SPI_SCB_MOSI_PIN	I/O 出力ピン番号	1ul
CY_SPI_SCB_MOSI_MUX	周辺機器接続	P0_1_SCB0_SPI_MOSI (30ul)
CY_SPI_SCB_CLK_PORT	I/O 出力ポート番号	GPIO_PRT0
CY_SPI_SCB_CLK_PIN	I/O 出力ピン番号	2ul
CY_SPI_SCB_CLK_MUX	周辺機器接続	P0_2_SCB0_SPI_CLK
CY_SPI_SCB_SEL0_PORT	I/O 出力ポート番号	GPIO_PRT0
CY_SPI_SCB_SEL0_PIN	I/O 出力ピン番号	3ul
CY_SPI_SCB_SEL0_MUX	周辺機器接続	P0_3_SCB0_SPI_SELECT0
SCB_MISO_DRIVE_MODE	MISO の DRIVE_MODE	CY_GPIO_DM_STRONG_IN_OFF (0x06)
SCB_MOSI_DRIVE_MODE	MOSI の DRIVE_MODE	CY_GPIO_DM_HIGHZ (0x08)
SCB_CLK_DRIVE_MODE	CLK の DRIVE_MODE	CY_GPIO_DM_HIGHZ (0x08)
SCB_SEL0_DRIVE_MODE	SEL0 の DRIVE_MODE	CY_GPIO_DM_HIGHZ (0x08)
SPI_port_pin_cfg.outVal	ピン出力状態	0ul
SPI_port_pin_cfg.driveMode	GPIO 駆動モード	0ul
SPI_port_pin_cfg.hsiom	I/O ピン配線の接続	HSIOM_SEL_GPIO (0x0)
SPI_port_pin_cfg.intEdge	IRQ をトリガするエッジ	0ul
SPI_port_pin_cfg.intMask	エッジ割込みをマスク	0ul
SPI_port_pin_cfg.vtrip	入力バッファ モード	0ul
SPI_port_pin_cfg.slewRate	スルーレート	0ul
SPI_port_pin_cfg.driveSel	GPIO 駆動強度	0ul
CY_SPI_SCB_IRQN	システム割込みインデックス番号	scb_0_interrupt_IRQn (IDX: 17)
irq_cfg.sysIntSrc	システム割込みインデックス番号	CY_SPI_SCB_IRQN
irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
.isEnabled	CPU 割込みイネーブル	true (enable)

3 SPI 設定手順例

表 5 に、SDL のドライバ部の SPI パラメーターを示します。

表 5　　関数一覧

関数	説明	備考
`Cy_SCB_SPI_GetTxFifoStatus (volatile stc_SCB_t const *base)`	Tx FIFO の現在のステータスを返します	*base: CY_SPI_SCB_TYPE
`Cy_SCB_SPI_ClearTxFifoStatus (volatile stc_SCB_t *base, uint32_t clearMask)`	Tx FIFO の選択されたステータスをクリアします	*base: CY_SPI_SCB_TYPE clearMask: CY_SCB_SPI_TX_TRIGGER

「Code Listing 15 設定部で SPI スレーブモードを設定する例」に、設定部で SPI スレーブモードを設定する例を示します。

3 SPI 設定手順例

Code Listing 15 設定部で SPI スレーブモードを設定する例

```
/* Device Specific Settings */
#define CY_SPI_SCB_TYPE      SCB0  /* Define the SCB channel */

#define CY_SPI_SCB_MISO_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MISO_PIN  0ul  /* Define the port settings */
#define CY_SPI_SCB_MISO_MUX  P0_0_SCB0_SPI_MISO  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_MOSI_PIN  1ul  /* Define the port settings */
#define CY_SPI_SCB_MOSI_MUX  P0_1_SCB0_SPI_MOSI  /* Define the port settings */
#define CY_SPI_SCB_CLK_PORT  GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_CLK_PIN   2ul  /* Define the port settings */
#define CY_SPI_SCB_CLK_MUX   P0_2_SCB0_SPI_CLK  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PORT GPIO_PRT0  /* Define the port settings */
#define CY_SPI_SCB_SEL0_PIN  3ul  /* Define the port settings */
#define CY_SPI_SCB_SEL0_MUX  P0_3_SCB0_SPI_SELECT  /* Define the port settings */


#define CY_SPI_SCB_PCLK      PCLK_SCB0_CLOCK  /* Define the peripheral clock */
#define CY_SPI_SCB_IRQN      scb_0_interrupt_IRQn  /* Define the System interrupt index number
*/



/* Slave Settings */
#define SCB_MISO_DRIVE_MODE CY_GPIO_DM_STRONG_IN_OFF  /* Define the port settings */
#define SCB_MOSI_DRIVE_MODE CY_GPIO_DM_HIGHZ  /* Define the port settings */
#define SCB_CLK_DRIVE_MODE  CY_GPIO_DM_HIGHZ  /* Define the port settings */
#define SCB_SEL0_DRIVE_MODE CY_GPIO_DM_HIGHZ  /* Define the port settings */


/* User setting value */
#define SOURCE_CLOCK_FRQ 80000000ul  /*Define the clock parameters */
#define SCB_SPI_BAUDRATE     125000ul /* Please set baudrate value of SPI you want */  /*Define
the clock parameters */
#define SCB_SPI_OVERSAMPLING 16ul     /* Please set oversampling of SPI you want */  /*Define
the clock parameters */
#define SCB_SPI_CLOCK_FREQ   (SCB_SPI_BAUDRATE * SCB_SPI_OVERSAMPLING)  /*Define the clock
parameters */

#define DIVIDER_NO_1 (1ul)

static cy_stc_gpio_pin_config_t SPI_port_pin_cfg =  /* Configure the port setting parameters */
{
    .outVal    = 0ul,
    .driveMode = 0ul,            /* Will be updated in runtime */
    .hsiom     = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
```

3 SPI 設定手順例

```
    .driveSel  = 0ul,
};

static cy_stc_sysint_irq_t irq_cfg =  /* Configure the interrupt structure parameters*1 */
{
    .sysIntSrc  = CY_SPI_SCB_IRQN,
    .intIdx     = CPUIntIdx3_IRQn,
    .isEnabled  = true,
};
uint16_t readData[2];
uint16_t initialWriteData[4] = {0x1122u, 0x3344u, 0x5566u, 0x7788u};
#define SIZE_OF_INITIAL_DATA (sizeof(initialWriteData)/sizeof(uint16_t))

static const cy_stc_scb_spi_config_t SCB_SPI_cfg =  /* Configure the SCB parameters */
{
    .spiMode                    = CY_SCB_SPI_SLAVE,       /*** Specifies the mode of
operation    ***/
    .subMode                    = CY_SCB_SPI_MOTOROLA,    /*** Specifies the submode of SPI
operation    ***/
    .sclkMode                   = CY_SCB_SPI_CPHA0_CPOL0, /*** Clock is active low, data is
changed on first edge ***/
    .oversample                 = SCB_SPI_OVERSAMPLING,   /*** SPI_CLOCK divided by
SCB_SPI_OVERSAMPLING should be baudrate  ***/
    .rxDataWidth                = 16ul,                   /*** The width of RX data (valid
range 4-16). It must be the same as \ref txDataWidth except in National sub-mode. ***/
    .txDataWidth                = 16ul,                   /*** The width of TX data (valid
range 4-16). It must be the same as \ref rxDataWidth except in National sub-mode. ***/
    .enableMsbFirst             = true,                   /*** Enables the hardware to shift
out the data element MSB first, otherwise, LSB first ***/
    .enableFreeRunSclk          = false,                  /*** Enables the master to generate a
continuous SCLK regardless of whether there is data to send  ***/
    .enableInputFilter          = false,                  /*** Enables a digital 3-tap median
filter to be applied to the input of the RX FIFO to filter glitches on the line. ***/
    .enableMisoLateSample       = true,                   /*** Enables the master to sample
MISO line one half clock later to allow better timings. ***/
    .enableTransferSeperation   = true,                   /*** Enables the master to transmit
each data element separated by a de-assertion of the slave select line (only applicable for the
master mode) ***/
    .ssPolarity0                = false,                  /*** SS0: active low ***/
    .ssPolarity1                = false,                  /*** SS1: active low ***/
    .ssPolarity2                = false,                  /*** SS2: active low ***/
    .ssPolarity3                = false,                  /*** SS3: active low ***/
    .enableWakeFromSleep        = false,                  /*** When set, the slave will wake
the device when the slave select line becomes active. Note that not all SCBs support this mode.
Consult the device datasheet to determine which SCBs support wake from deep sleep. ***/
    .rxFifoTriggerLevel         = 1ul,                    /*** Interrupt occurs, when there are
more entries of 2 in the RX FIFO */
    .rxFifoIntEnableMask        = 1ul,                    /*** Bits set in this mask will allow
events to cause an interrupt  */
    .txFifoTriggerLevel         = 1ul,                    /*** When there are fewer entries in
the TX FIFO, then at this level the TX trigger output goes high. This output can be connected
to a DMA channel through a trigger mux. Also, it controls the \ref CY_SCB_SPI_TX_TRIGGER
interrupt source. */
```

```
    .txFifoIntEnableMask        = 1ul,                  /*** Bits set in this mask allow
events to cause an interrupt  */
    .masterSlaveIntEnableMask   = 0ul,                  /*** Bits set in this mask allow
events to cause an interrupt  */
    .enableSpiDoneInterrupt     = false,
    .enableSpiBusErrorInterrupt = false,
};

void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)  /* Create
the function to determine the divider division ratio */
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq);   /* Calculates the division ration */
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
                                   (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
                                   (divSetting & 0x0000001Ful));  /* Set the division ratio
(See Code Listing 5) */
}

int main(void)
{
    SystemInit();

    /*****************************************************/
    /******* Calculate divider setting for the SCB ********/  /* (1) Configure the clock */
    /*****************************************************/




    Cy_SysClk_PeriphAssignDivider(CY_SPI_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);/*
Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(SCB_SPI_CLOCK_FREQ, SOURCE_CLOCK_FRQ, DIVIDER_NO_1);  /* Configure
the divider (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 1ul);  /* Enable the divider (See
Code Listing 6) */







    __enable_irq(); /* Enable global interrupts. */  /* (2) Enable global interrupt*1 */


    /*****************************************/
    /*   De-initialization for peripherals     */
    /*****************************************/
    Cy_SCB_SPI_DeInit(CY_SPI_SCB_TYPE);  /* De-Initialize the SCB if necessary (See Code
Listing 7) */
```

3 SPI 設定手順例

```
    /****************************************/
    /* Interrupt setting for SPI communication */
    /****************************************/
    Cy_SysInt_InitIRQ(&irq_cfg);  /* (3) Set the parameters to interrupt structure*1 */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqSCB);  /* (4) Set the system interrupt
handler*1 (See Code Listing 3) */
    NVIC_EnableIRQ(irq_cfg.intIdx);  /* (5) Interrupt Enable*1 */









    /*************************************/
    /* Port Setting for SPI communication */  /* (6) Set the SPI port*2 */
    /*************************************/
    /* According to the HW environment to change SCB CH*/

    SPI_port_pin_cfg.driveMode = SCB_MISO_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MISO_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_MISO_PORT, CY_SPI_SCB_MISO_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_MOSI_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_MOSI_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_MOSI_PORT, CY_SPI_SCB_MOSI_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_CLK_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_CLK_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_CLK_PORT,CY_SPI_SCB_CLK_PIN, &SPI_port_pin_cfg);  /* Change the
driveMode and set the port setting parameters */

    SPI_port_pin_cfg.driveMode = SCB_SEL0_DRIVE_MODE;  /* Change the driveMode and set the port
setting parameters */
    SPI_port_pin_cfg.hsiom = CY_SPI_SCB_SEL0_MUX;  /* Change the driveMode and set the port
setting parameters */
    Cy_GPIO_Pin_Init(CY_SPI_SCB_SEL0_PORT, CY_SPI_SCB_SEL0_PIN, &SPI_port_pin_cfg);  /* Change
the driveMode and set the port setting parameters */




    /****************************************/
    /* SCB initialization for SPI communication */
```

```
    /*****************************************/
    Cy_SCB_SPI_Init(CY_SPI_SCB_TYPE, &SCB_SPI_cfg, NULL);  /* (7) Initialize SCB for SPI
communication (See Code Listing 8) */
    Cy_SCB_SPI_SetActiveSlaveSelect(CY_SPI_SCB_TYPE, 0ul);  /* Set the using cannel number (See
Code Listing 9) */
    Cy_SCB_SPI_Enable(CY_SPI_SCB_TYPE);  /* (8) Enable SPI (See Code Listing 10) */

    for(;;);
}
```

*1: 詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。
*2: 詳細については、Architecture TRM の I/O システムのセクションを参照してください。

Code Listing 16 割込みハンドラの例

```
void irqSCB(void)
{
    uint32_t status;

    status = Cy_SCB_SPI_GetRxFifoStatus(CY_SPI_SCB_TYPE);  /* (9) Check the Interrupt is Active
(See Code Listing 11) */
    if(status & CY_SCB_SPI_RX_TRIGGER)
    {
        /*** Read data from RX FIFO ***/
        Cy_SCB_SPI_ReadArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul);  /* (10) Read the Received
SPI Data (See Code Listing 12) */
        Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE, (void*)readData, 2ul);
        Cy_SCB_SPI_ClearRxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_RX_TRIGGER);  /* (11) Clear
the RX TRIGGER Interrupt Flag (See Code Listing 13) */
    }



    status = Cy_SCB_SPI_GetTxFifoStatus(CY_SPI_SCB_TYPE);  /* (12) Check the TX Interrupt is
Active (See Code Listing 17) */
    if(status & CY_SCB_SPI_TX_TRIGGER)
    {

        /*** Write back the data to TX FIFO ***/
        Cy_SCB_SPI_WriteArray(CY_SPI_SCB_TYPE, (void*)initialWriteData,
SIZE_OF_INITIAL_DATA);  /* (13) Write and send SPI Data (See Code Listing 14)
 */
        Cy_SCB_SPI_ClearTxFifoStatus(CY_SPI_SCB_TYPE, CY_SCB_SPI_TX_TRIGGER);  /* (14) Clear
the TX TRIGGER interrupt flag (See Code Listing 18) */
    }
}
```

*1: 詳細については、Architecture TRM の CPU 割込み処理のセクションを参照してください。

Code Listing 17 と Code Listing 18 に、ドライバ部で SCB を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

3 SPI 設定手順例

Code Listing 17 Cy_SCB_SPI_GetTxFifoStatus() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_SPI_GetTxFifoStatus(volatile stc_SCB_t const *base)
{
    return (Cy_SCB_GetTxInterruptStatus(base) & CY_SCB_SPI_TX_INTR);  /* Read and check the Tx
Interrupt */
}
```

Code Listing 18 Cy_SCB_SPI_ClearTxFifoStatus() 関数

```
__STATIC_INLINE void Cy_SCB_SPI_ClearTxFifoStatus(volatile stc_SCB_t *base, uint32_t clearMask)
{
    Cy_SCB_ClearTxInterrupt(base, clearMask);  /* Clear the Tx Interrupt Factor */
}
```

4 UART 設定手順例

ここでは、サンプルドライバライブラリ (SDL) を使用した SPI の例を示します。SCB は、Motorola, Texas Instruments, および National Semiconductor プロトコルで SPI マスタモードと SPI スレーブモードをサポートします。各プロトコルの詳細については、Architecture TRM を参照してください。このアプリケーションノートのプログラムコードは SDL の一部であり、CYT2B7 シリーズに基づいています。SDL については、その他の参考資料を参照してください。

SCB は、標準 UART, Multi-processor モード, Smart Card (ISO7816) リーダ, IrDA, および LIN (スレーブモード) の機能を持ちます。各プロトコルの詳細については、Architecture TRM を参照してください。ここでは、標準 UART の設定手順例を説明します。

4.1 UART モード

この例では、標準 UART モードの使用方法を示します。このユースケースでは、該当するレジスタ設定後、SCB は他のデバイスに 1 バイトのデータを送信し、他のデバイスからの Rx データを待ちます。

4.1.1 ユースケース

- SCB モード = 標準 UART
- SCB チャネル = 3
- PCLK = 80 MHz
- ボーレート = 115,200 bps

  [ボーレート設定]

  ボーレートの計算式は次のとおりです。

  ボーレート [bps] = 入力クロック [Hz] / OVS

  OVS: SCB_CTRL.OVS + 1

  例えば、以下は、理想的な UART ボーレートから実際の UART ボーレートを計算する方法を示します。

  - CLK_PERI 周波数 = 40 [MHz]
  - 対象の UART ボーレート (ビットレート) = 115,200 [bps]
  - OVS = 16 [oversamples]

  指定された CLK_PERI 周波数, ターゲット UART ボーレート, および OVS を使用して、実際のボーレートを計算できます。

  まず、SCB への理想的な入力クロックが計算されます。

  理想的な入力クロック = 対象のボーレート * OVS = 115,200 * 16 = 1,843,200 [Hz]

  次に、必要なクロック分周器制御レジスタ (DIV24.5) の理想値を計算できます。

  理想的な DIV24.5 = 40 [MHz] / 1,843,200 [Hz] = 21.7014

  ただし、DIV24.5 レジスタには整数部分用に 24 ビットがあり、小数部分用に 5 ビットに制限されています (1/32 に基づく)。したがって、実際の分周値と実際の UART ボーレートは次のように計算できます。

  実際の DIV24.5 = 21.6875 (整数部: 21, 小数部: 22/32)

  実際の UART ボーレート = 40 [MHz] / 21. 6875 / 16 = 115,274 [bps]

  詳細については、Architecture TRM を参照してください。
- データ幅 = 8 ビット
- パリティ= なし
- Stop ビット = 1
- フロー制御 = なし
- Tx/Rx FIFO = 使用

4 UART 設定手順例

- Rx 割込み = 無効
- 使用ポート
  - Tx : SCB3_TX (P13.1)
  - Rx : SCB3_RX (P13.0)
- MCU は UART デバイスから送信されたデータを受信します。MCU は受信したデータをそのまま送信します。MCU から送信された初期メッセージとデータが PC に表示されます。

図 6 に、SCB と外部 UART デバイス間の Tx-Rx 接続例を示します。この例では、フロー制御信号の RTS および CTS は未使用です。

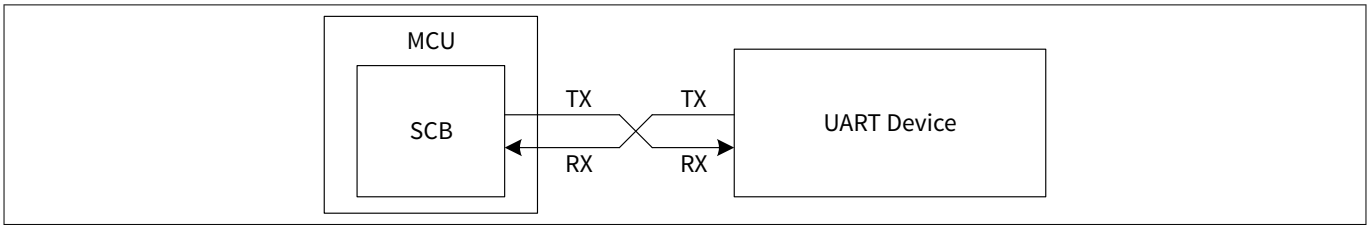

図 6　　UART 通信接続の例

図 7 に UART の設定手順と動作例を示します。

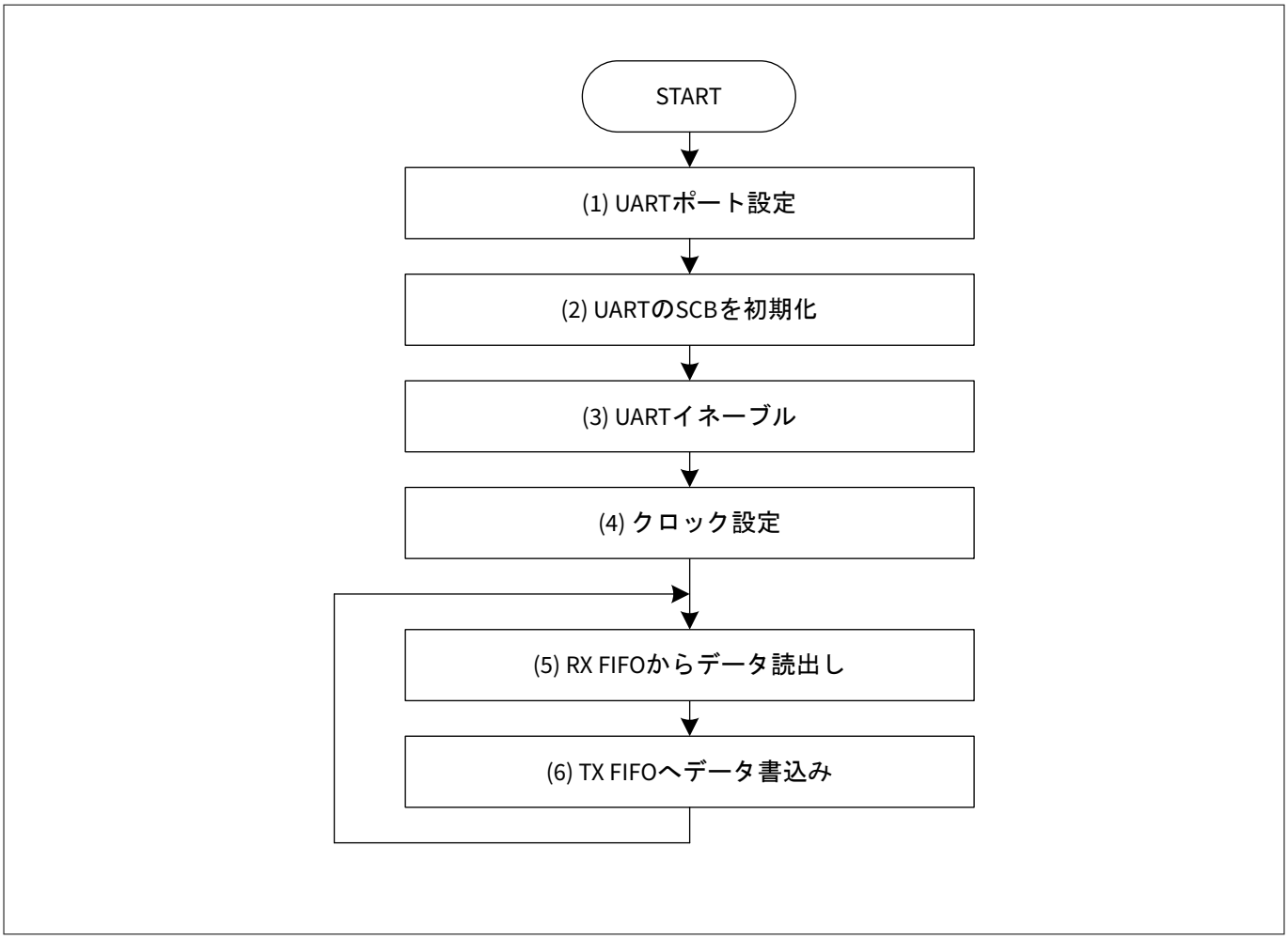

図 7　　UART 動作

(1) UART ポートを設定します。

(2) UART の SCB を初期化します。

(3) UART をイネーブルにします。
(4) クロックを設定します。
(5) Rx FIFO からデータを読み出します。
(6) Tx FIFO へデータを書き込みます。

4.1.2 設定および例

表 6 に、UART モードの SDL の設定部のパラメーターを示します。

表 6 UART モード設定パラメーター一覧

パラメーター	説明	設定値
UART_OVERSAMPLING	UART のオーバサンプリング	8ul
CY_USB_SCB_UART_PCLK	周辺クロック番号	PCLK_SCB3_CLOCK (17ul)
g_stc_uart_config.uartMode	UART 動作のサブモード	CY_SCB_UART_STANDARD (0ul)
g_stc_uart_config.oversample	UART のオーバサンプリング	UART_OVERSAMPLING (8ul)
g_stc_uart_config.dataWidth	データフレーム幅	8ul
g_stc_uart_config.enableMsbFirst	LSB が最初または MSB が最初	False (LSB)
g_stc_uart_config.stopBits	ストップビット	CY_SCB_UART_STOP_BITS_1 (2ul)
g_stc_uart_config.parity	パリティビット	CY_SCB_UART_PARITY_NONE (0ul)
g_stc_uart_config.enableInputFilter	メディアン フィルタ	False
g_stc_uart_config.dropOnParityError	パリティチェック失敗時の動作	False
g_stc_uart_config.dropOnFrameError	開始または停止期間にエラーが検出された場合の動作	False
g_stc_uart_config.enableMutliProcessorMode	マルチプロセッサモード	False (disable)
g_stc_uart_config.receiverAddress	スレーブ デバイス アドレス	0ul
g_stc_uart_config.receiverAddressMask	スレーブ デバイス アドレス マスク	0ul
g_stc_uart_config.acceptAddrInFifo	受信した一致するアドレスを RX FIFO で受け入れる	False (not accept)
g_stc_uart_config.irdaInvertRx	入力 RX ライン信号を反転	False
g_stc_uart_config.smartCardRetryOnNack	否定応答が受信されると、データフレームが再送信される	False
g_stc_uart_config.enableCts	UART 送信による CTS 入力信号の使用をイネーブル	False
g_stc_uart_config.ctsPolarity	CTS 入力信号の極性	CY_SCB_UART_ACTIVE_LOW (0ul)
g_stc_uart_config.rtsRxFifoLevel	トリガ レベル	0ul
g_stc_uart_config.rtsPolarity	RTS 出力信号の極性	CY_SCB_UART_ACTIVE_LOW (0ul)
g_stc_uart_config.breakWidth	ブレーク幅	0ul

(続く)

表 6 (続き) UART モード設定パラメーター一覧

パラメーター	説明	設定値
g_stc_uart_config.rxFifoTriggerLevel	Rx FIFO のトリガ レベル	0ul
g_stc_uart_config.rxFifoIntEnableMask	受信割込みマスク	0ul
g_stc_uart_config.txFifoTriggerLevel	Tx FIFO のトリガ レベル	0ul
g_stc_uart_config.txFifoIntEnableMask	送信割込みマスク	0ul
CY_USB_SCB_TYPE	SCB チャネル番号の使用	SCB3
stc_port_pin_cfg_uart.driveMode	GPIO 駆動モード	Rx: CY_GPIO_DM_HIGHZ (8ul) Tx: CY_GPIO_DM_STRONG_IN_OFF (6ul)
SPI_port_pin_cfg.hsiom	IO ピン配線の接続を指定	Rx: CY_USB_SCB_UART_RX_MUX (17ul) Tx: CY_USB_SCB_UART_TX_MUX (17ul)
CY_USB_SCB_UART_RX_PORT	Rx の GPIO ポート番号	GPIO_PRT13
CY_USB_SCB_UART_RX_PIN	Rx の GPIO ポートピン	0ul
CY_USB_SCB_UART_TX_PORT	Tx の GPIO ポート番号	GPIO_PRT13
CY_USB_SCB_UART_TX_PIN	Tx の GPIO ポートピン	1ul

表 7 に、SDL のドライバ部の UART 関数を示します。

表 7 関数一覧

関数	説明	備考
Cy_SCB_UART_GetNumInRxFifo (volatile stc_SCB_t const *base)	UART Rx FIFO 内のデータ要素の数を返します	*base: CY_USB_SCB_TYPE
Cy_SCB_UART_GetArray (volatile stc_SCB_t const *base, void *rxBuf, uint32_t size)	UART Rx FIFO からデータアレイを読み出します	*base: CY_USB_SCB_TYPE *rxBuf: uart_in_data size: rx_num
Cy_SCB_UART_PutArray (volatile stc_SCB_t *base, void *txBuf, uint32_t size)	データアレイを UART Tx FIFO に配置します	*base: CY_USB_SCB_TYPE *rxBuf: uart_in_data size: rx_num
Cy_SCB_UART_DeInit (volatile stc_SCB_t *base)	SCB ブロックを初期化解除します	*base: CY_USB_SCB_TYPE

(続く)

4 UART 設定手順例

表 7 (続き) 関数一覧

関数	説明	備考
`Cy_SCB_UART_Init (volatile stc_SCB_t *base, cy_stc_scb_uart_config_t const *config, cy_stc_scb_uart_context_t *context)`	UART 動作のために SCB を初期化します	*base: CY_USB_SCB_TYPE *config: g_stc_uart_config *context: g_stc_uart_context
`Cy_SCB_UART_Enable (volatile stc_SCB_t *base)`	UART 動作用の SCB ブロックを有効にします	*base: CY_USB_SCB_TYPE

Code Listing 19 に、設定部で UART モードを設定する例を示します。

4 UART 設定手順例

Code Listing 19 設定部で UART モードを設定する例

```
/* Select UART Echo Type                                                    */
/* Use Low-Level API. Polling & Receive by 1 byte unit                      */

/* Local Definition */
#define UART_OVERSAMPLING (8ul)

/* Local Functions Declaration */
void UART_Initialization(uint32_t boadrate, uint32_t sourceFreq);
void Term_Printf(void *fmt, ...);


/* SCB - UART Configuration */
cy_stc_scb_uart_context_t   g_stc_uart_context;
cy_stc_scb_uart_config_t    g_stc_uart_config =  /* Configure the UART parameters */
{
    .uartMode                   = CY_SCB_UART_STANDARD,
    .oversample                 = UART_OVERSAMPLING,
    .dataWidth                  = 8ul,
    .enableMsbFirst             = false,
    .stopBits                   = CY_SCB_UART_STOP_BITS_1,
    .parity                     = CY_SCB_UART_PARITY_NONE,
    .enableInputFilter          = false,
    .dropOnParityError          = false,
    .dropOnFrameError           = false,
    .enableMutliProcessorMode   = false,
    .receiverAddress            = 0ul,
    .receiverAddressMask        = 0ul,
    .acceptAddrInFifo           = false,
    .irdaInvertRx               = false,
    .irdaEnableLowPowerReceiver = false,
    .smartCardRetryOnNack       = false,
    .enableCts                  = false,
    .ctsPolarity                = CY_SCB_UART_ACTIVE_LOW,
    .rtsRxFifoLevel             = 0ul,
    .rtsPolarity                = CY_SCB_UART_ACTIVE_LOW,
    .breakWidth                 = 0ul,
    .rxFifoTriggerLevel         = 0ul,
    .rxFifoIntEnableMask        = 0ul,
    .txFifoTriggerLevel         = 0ul,
    .txFifoIntEnableMask        = 0ul
};

int main(void)
{
    SystemInit();

    /* UART Initialization */
    //                  boardrate, peri frequency
    UART_Initialization( 115200ul,     80000000ul);
```

4 UART 設定手順例

```
    /* Opening UART Comment */
    Term_Printf("\nUART TEST (driver ver=%d.%d)\n\r", CY_SCB_DRV_VERSION_MAJOR,
CY_SCB_DRV_VERSION_MINOR);
    Term_Printf("POLLING 1BYTE ECHO\n\r");

    for(;;)
    {
        uint32_t    rx_num;
        /* Check Receive Data */
        rx_num = Cy_SCB_UART_GetNumInRxFifo(CY_USB_SCB_TYPE);  /* Read the data from RX FIFO
STATUS (See Code Listing 20) */
        if (rx_num != 0ul)
        {
            uint8_t uart_in_data[128];
            Cy_SCB_UART_GetArray(CY_USB_SCB_TYPE, uart_in_data, rx_num);  /* (5) Read the data
from RX FIFO (See Code Listing 21) */
            Cy_SCB_UART_PutArray(CY_USB_SCB_TYPE, uart_in_data, rx_num);  /* (6) Write the data
to TX FIFO (See Code Listing 22) */
        }
    }
}

void UART_Initialization(uint32_t boadrate, uint32_t sourceFreq)
{
    /* Port Configuration for UART */
    cy_stc_gpio_pin_config_t    stc_port_pin_cfg_uart = {0ul};  /* (1) Set the UART port */
    stc_port_pin_cfg_uart.driveMode = CY_GPIO_DM_HIGHZ;  /* (1) Set the UART port */
    stc_port_pin_cfg_uart.hsiom     = CY_USB_SCB_UART_RX_MUX;  /* (1) Set the UART port */
    Cy_GPIO_Pin_Init(CY_USB_SCB_UART_RX_PORT, CY_USB_SCB_UART_RX_PIN,
&stc_port_pin_cfg_uart);  /* (1) Set the UART port */

    stc_port_pin_cfg_uart.driveMode = CY_GPIO_DM_STRONG_IN_OFF;  /* (1) Set the UART port */
    stc_port_pin_cfg_uart.hsiom     = CY_USB_SCB_UART_TX_MUX;  /* (1) Set the UART port */
    Cy_GPIO_Pin_Init(CY_USB_SCB_UART_TX_PORT, CY_USB_SCB_UART_TX_PIN,
&stc_port_pin_cfg_uart);  /* (1) Set the UART port */



    /* SCB-UART Initialization */
    Cy_SCB_UART_DeInit(CY_USB_SCB_TYPE);  /* If necessary, stop the UART operation (See Code
Listing 23) */
    Cy_SCB_UART_Init(CY_USB_SCB_TYPE, &g_stc_uart_config, &g_stc_uart_context);  /* (2)
Initialize SCB for UART (See Code Listing 24) */
    Cy_SCB_UART_Enable(CY_USB_SCB_TYPE);  /* (3) Enable the UART (See Code Listing 25) */
```

4 UART 設定手順例

```
    /* Clock Configuration for UART */  /* (4) Configure the clock */
    // Assign a programmable divider
    Cy_SysClk_PeriphAssignDivider(CY_USB_SCB_UART_PCLK, CY_SYSCLK_DIV_24_5_BIT, 0ul);  /*
Configure the Peripheral Clock (See Code Listing 4) */
    // Set divider value
    {

        uint64_t targetFreq     = UART_OVERSAMPLING * boadrate;
        uint64_t sourceFreq_fp5 = ((uint64_t)sourceFreq << 5ul);
        uint32_t divSetting_fp5 = (uint32_t)(sourceFreq_fp5 / targetFreq);
        Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT,  /* Configure the divider (See
Code Listing 5) */
                                       0ul,
                                       ((divSetting_fp5 & 0x1FFFFFE0ul) >> 5ul),
                                       (divSetting_fp5 & 0x0000001Ful));
    }

    // Enable peripheral divider
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, 0ul);/* Enable the divider (See Code
Listing 6) */
}

void Term_Printf(void *fmt, ...)
{
    uint8_t uart_out_data[128];
    va_list arg;

    /* UART Print */
    va_start(arg, fmt);
    vsprintf((char*)&uart_out_data[0], (char*)fmt, arg);
    while (Cy_SCB_UART_IsTxComplete(CY_USB_SCB_TYPE) != true) {};
    Cy_SCB_UART_PutArray(CY_USB_SCB_TYPE, uart_out_data, strlen((char *)uart_out_data));
    va_end(arg);
}
```

*1: 詳細については、Architecture TRM の I/O システムのセクションを参照してください。

Code Listing 20～Code Listing 25 に、ドライバ部で SCB を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

Code Listing 20 Cy_SCB_UART_GetNumInRxFifo() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_UART_GetNumInRxFifo(volatile stc_SCB_t const *base)
{
    return Cy_SCB_GetNumInRxFifo(base);  /* Read the data from RX FIFO STATUS */
}
```

4 UART 設定手順例

Code Listing 21 Cy_SCB_UART_GetArray() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_UART_GetArray(volatile stc_SCB_t const *base, void *rxBuf,
uint32_t size)
{
    return Cy_SCB_ReadArray(base, rxBuf, size);  /* Read the data from RX FIFO */
}
```

Code Listing 22 Cy_SCB_UART_PutArray() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_UART_PutArray(volatile stc_SCB_t *base, void *txBuf, uint32_t
size)
{
    return Cy_SCB_WriteArray(base, txBuf, size);  /* Write the data to TX FIFO */
}
```

4 UART 設定手順例

Code Listing 23 Cy_SCB_UART_DeInit() 関数

```
void Cy_SCB_UART_DeInit(volatile stc_SCB_t *base)
{
    /* De-initialize the UART interface */  /* Set the default value to unCTRL and unUART_CTRL
Reg */
    base->unCTRL.u32Register      = CY_SCB_CTRL_DEF_VAL;
    base->unUART_CTRL.u32Register = CY_SCB_UART_CTRL_DEF_VAL;

    /* De-initialize the RX direction */  /* Set the default value to unUART_RX_CTRL,
unRX_CTRL, unRX_FIFO_CTRL, and unRX_MATCH Reg */
    base->unUART_RX_CTRL.u32Register = 0ul;
    base->unRX_CTRL.u32Register      = CY_SCB_RX_CTRL_DEF_VAL;
    base->unRX_FIFO_CTRL.u32Register = 0ul;
    base->unRX_MATCH.u32Register     = 0ul;

    /* De-initialize the TX direction */  /* Set the default value to unUART_TX_CTRL,
unTX_CTRL, and unTX_FIFO_CTRL Reg */
    base->unUART_TX_CTRL.u32Register = 0ul;
    base->unTX_CTRL.u32Register      = CY_SCB_TX_CTRL_DEF_VAL;
    base->unTX_FIFO_CTRL.u32Register = 0ul;

    /* De-initialize the flow control */  /* Set the default value to unUART_FLOW_CTRL Reg */
    base->unUART_FLOW_CTRL.u32Register = 0ul;

    /* De-initialize the interrupt sources */  /* Set the default value to Reg for Interrupt */
    base->unINTR_SPI_EC_MASK.u32Register = 0ul;
    base->unINTR_I2C_EC_MASK.u32Register = 0ul;
    base->unINTR_RX_MASK.u32Register     = 0ul;
    base->unINTR_TX_MASK.u32Register     = 0ul;
    base->unINTR_M_MASK.u32Register      = 0ul;
    base->unINTR_S_MASK.u32Register      = 0ul;
}
```

4 UART 設定手順例

Code Listing 24 Cy_SCB_UART_Init() 関数

```
cy_en_scb_uart_status_t Cy_SCB_UART_Init(volatile stc_SCB_t *base, cy_stc_scb_uart_config_t
const *config, cy_stc_scb_uart_context_t *context)
{
    cy_en_scb_uart_status_t retStatus = CY_SCB_UART_BAD_PARAM;
    un_SCB_CTRL_t           temp_CTRL           = { 0ul };
    un_SCB_UART_CTRL_t      temp_UART_CTRL      = { 0ul };
    un_SCB_UART_RX_CTRL_t   temp_UART_RX_CTRL   = { 0ul };
    un_SCB_RX_CTRL_t        temp_RX_CTRL        = { 0ul };
    un_SCB_RX_MATCH_t       temp_RX_MATCH       = { 0ul };
    un_SCB_UART_TX_CTRL_t   temp_UART_TX_CTRL   = { 0ul };
    un_SCB_TX_CTRL_t        temp_TX_CTRL        = { 0ul };
    un_SCB_RX_FIFO_CTRL_t   temp_RX_FIFO_CTRL   = { 0ul };
    un_SCB_UART_FLOW_CTRL_t temp_UART_FLOW_CTRL = { 0ul };
    un_SCB_TX_FIFO_CTRL_t   temp_TX_FIFO_CTRL   = { 0ul };

    if ((NULL != base) && (NULL != config))  /* Check if configuration parameter values are
valid */
    {
        uint32_t ovs;

        if ((CY_SCB_UART_IRDA == config->uartMode) && (!config->irdaEnableLowPowerReceiver))
        {
            /* For Normal IrDA mode oversampling is always zero */
            ovs = 0ul;
}
        else
        {
            ovs = (config->oversample - 1ul);
        }

        /* Configure the UART interface */  /* Set the config value to unCTRL and unUART_CTRL
Reg */
        temp_CTRL.stcField.u1ADDR_ACCEPT = (config->acceptAddrInFifo ? 1ul : 0ul);
        temp_CTRL.stcField.u2MEM_WIDTH   = 0ul;
        temp_CTRL.stcField.u4OVS         = ovs;
        temp_CTRL.stcField.u2MODE        = CY_SCB_CTRL_MODE_UART;
        base->unCTRL.u32Register         = temp_CTRL.u32Register;
        temp_UART_CTRL.stcField.u2MODE   = config->uartMode;
        base->unUART_CTRL.u32Register    = temp_UART_CTRL.u32Register;


        /* Configure the RX direction */  /* Set the config value to unUART_RX_CTRL, unRX_CTRL,
and unRX_MATCH Reg */
        temp_UART_RX_CTRL.stcField.u1POLARITY             = (config->irdaInvertRx ? 1ul : 0ul);
        temp_UART_RX_CTRL.stcField.u1MP_MODE              = (config->enableMutliProcessorMode ?
1ul : 0ul);
        temp_UART_RX_CTRL.stcField.u1DROP_ON_PARITY_ERROR = (config->dropOnParityError ? 1ul :
0ul);
        temp_UART_RX_CTRL.stcField.u1DROP_ON_FRAME_ERROR  = (config->dropOnFrameError ? 1ul :
0ul);
```

4 UART 設定手順例

```
        temp_UART_RX_CTRL.stcField.u4BREAK_WIDTH          = (config->breakWidth - 1ul);
        temp_UART_RX_CTRL.stcField.u3STOP_BITS            = (config->stopBits - 1ul);
        temp_UART_RX_CTRL.stcField.u1PARITY               = (config->parity & 0x00000001ul);
        temp_UART_RX_CTRL.stcField.u1PARITY_ENABLED       = (config->parity & 0x00000002ul) >>
1;
        base->unUART_RX_CTRL.u32Register                  = temp_UART_RX_CTRL.u32Register;

        temp_RX_CTRL.stcField.u1MSB_FIRST  = (config->enableMsbFirst ? 1ul : 0ul);
        temp_RX_CTRL.stcField.u1MEDIAN     = (config->enableInputFilter ? 1ul : 0ul);
        temp_RX_CTRL.stcField.u5DATA_WIDTH = (config->dataWidth - 1ul);
        base->unRX_CTRL.u32Register        = temp_RX_CTRL.u32Register;

        temp_RX_MATCH.stcField.u8ADDR = config->receiverAddress;
        temp_RX_MATCH.stcField.u8MASK = config->receiverAddressMask;
        base->unRX_MATCH.u32Register  = temp_RX_MATCH.u32Register;

        /* Configure the TX direction */  /* Set the config value to unUART_TX_CTRL, unTX_CTRL,
and unRX_FIFO_CTRL Reg */
        temp_UART_TX_CTRL.stcField.u1RETRY_ON_NACK  = (config->smartCardRetryOnNack ? 1ul :
0ul);
        temp_UART_TX_CTRL.stcField.u3STOP_BITS      = (config->stopBits - 1ul);
        temp_UART_TX_CTRL.stcField.u1PARITY         = (config->parity & 0x00000001ul);
        temp_UART_TX_CTRL.stcField.u1PARITY_ENABLED = (config->parity & 0x00000002ul) >> 1;
        base->unUART_TX_CTRL.u32Register            = temp_UART_TX_CTRL.u32Register;

        temp_TX_CTRL.stcField.u1MSB_FIRST  = (config->enableMsbFirst ? 1ul : 0ul);
        temp_TX_CTRL.stcField.u5DATA_WIDTH = (config->dataWidth - 1ul);
        temp_TX_CTRL.stcField.u1OPEN_DRAIN = ((config->uartMode == CY_SCB_UART_SMARTCARD) ?
1ul : 0ul);
        base->unTX_CTRL.u32Register        = temp_TX_CTRL.u32Register;

        temp_RX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->rxFifoTriggerLevel;
        base->unRX_FIFO_CTRL.u32Register           = temp_RX_FIFO_CTRL.u32Register;









        /* Configure the flow control */  /* Set the config value to unUART_FLOW_CTRL,
unTX_FIFO_CTRL Reg */
        temp_UART_FLOW_CTRL.stcField.u1CTS_ENABLED   = (config->enableCts ? 1ul : 0ul);
        temp_UART_FLOW_CTRL.stcField.u1CTS_POLARITY  = ((CY_SCB_UART_ACTIVE_HIGH == config-
>ctsPolarity) ? 1ul : 0ul);
        temp_UART_FLOW_CTRL.stcField.u1RTS_POLARITY  = ((CY_SCB_UART_ACTIVE_HIGH == config-
>rtsPolarity) ? 1ul : 0ul);
        temp_UART_FLOW_CTRL.stcField.u8TRIGGER_LEVEL = config->rtsRxFifoLevel;
        base->unUART_FLOW_CTRL.u32Register           = temp_UART_FLOW_CTRL.u32Register;

        temp_TX_FIFO_CTRL.stcField.u8TRIGGER_LEVEL = config->txFifoTriggerLevel;
        base->unTX_FIFO_CTRL.u32Register           = temp_TX_FIFO_CTRL.u32Register;


        /* Set up interrupt sources */  /* Set the config value to unINTR_TX_MASK and
```

```
    unINTR_RX_MASK Reg */
            base->unINTR_TX_MASK.u32Register = config->txFifoIntEnableMask;
            base->unINTR_RX_MASK.u32Register = config->rxFifoIntEnableMask;

            /* Initialize context */
            if (NULL != context)
            {
                context->rxStatus  = 0ul;  /* Clear the context */
                context->txStatus  = 0ul;  /* Clear the context */

                context->rxRingBuf = NULL;  /* Clear the context */
                context->rxRingBufSize = 0ul;   /* Clear the context */

                context->rxBufIdx  = 0ul;  /* Clear the context */
                context->txLeftToTransmit = 0ul;  /* Clear the context */

                context->cbEvents = NULL;  /* Clear the context */

            #if !defined(NDEBUG)
                /* Put an initialization key into the initKey variable to verify
                * context initialization in the transfer API.
                */
                context->initKey = CY_SCB_UART_INIT_KEY;
            #endif /* !(NDEBUG) */
            }
            retStatus = CY_SCB_UART_SUCCESS;
        }
        return (retStatus);
    }
```

Code Lisitng 25 Cy_SCB_UART_Enable() 関数

```
__STATIC_INLINE void Cy_SCB_UART_Enable(volatile stc_SCB_t *base)
{
    base->unCTRL.stcField.u1ENABLED = 1ul;  /* Set the enable bit to “1” */
}
```

5 I²C 設定手順例

ここでは、サンプルドライバライブラリ (SDL) を使用した SPI の例を示します。SCB は、Motorola, Texas Instruments, および National Semiconductor プロトコルで SPI マスタモードと SPI スレーブモードをサポートします。各プロトコルの詳細については、Architecture TRM を参照してください。このアプリケーションノートのプログラムコードは SDL の一部です。このサンプルプログラムは CYT2B7 シリーズ用です。SDL については、その他の参考資料を参照してください。

この例では、I²C モードの使用方法を示します。SCB はマスタモード、スレーブモード、およびマルチマスタ モードをサポートします。各プロトコルの詳細については、Architecture TRM を参照してください。

5.1 マスタモード

この例では、SCB を I²C マスタとして設定します。スレーブデバイス (スレーブアドレス = 0x08) に 1 バイトのデータを送信した後、同じスレーブから 1 バイトのデータを受信します。この例ではシンプルになるよう、FIFO への書込み/読出しのために、割込みを使用せずポーリングしています。

5.1.1 ユースケース

- SCB モード = I²C マスタモード
- SCB チャネル = 0
- PCLK = 2 MHz
- ビットレート = 100 kbps
  [ビットレート設定]
  ビットレートの設定はマスタモードのみ有効です。ビットレートの計算式は次のとおりです。
  ビットレート [bps] = 入力クロック [Hz] / (Low_phase_ovs + High_phase_ovs)
  Low_phase_ovs : SCB_I2C_CTRL.LOW_PHASE_OVS + 1
  High_phase_ovs : SCB_I2C_CTRL.HIGH_PHASE_OVS + 1
  この場合、ビットレートは次のように計算されます。
  ビットレート = 入力クロック [Hz] / (High_phase_ovs + Low_phase_ovs)
  = PCLK(2MHz) / ((9+1) + (9+1)) = 100 [kbps]
  詳細については、Architecture TRM を参照してください。
- 7 ビットスレーブアドレス = 0x8 (別の I2C デバイス用)
- 最上位ビット (MSb) ファースト
- Tx/Rx FIFO = 使用
- Tx/Rx 割込み = 無効
- アナログフィルタ = 有効、デジタルフィルタ = 無効
- 使用ポート
  - SCL : SCB0_SCL (P1.0)
  - SDA : SCB0_SDA (P1.1)

図 8 に、SCB と別の I²C スレーブデバイス間の接続例を示します。

5 I²C 設定手順例

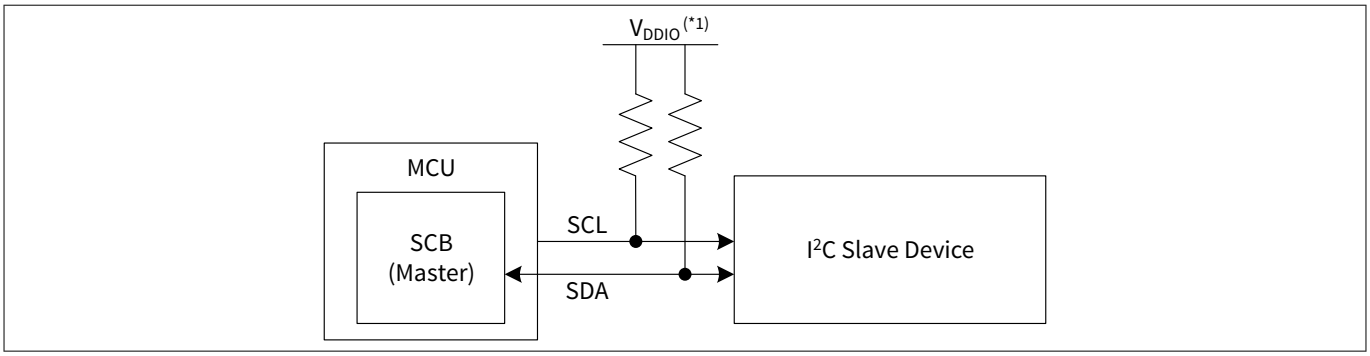

図 8　　I²C (マスタモード) 接続の例

*注:　　(*1) V_{DDIO} の値については、データシートを参照してください。(関連ドキュメントを参照)*

I²C マスタモードでは、SCL と SDA 信号が、他の I²C スレーブデバイスと接続されます。マスタデバイスは、スレーブデバイスにクロック (SCL) を出力します。データ信号 (SDA) は、双方向信号です。SCL と SDA はともに抵抗を介して V_{DDIO} に外部プルアップされます。

図 9 に I2C マスタモードの設定手順と動作例を示します。

5 I²C 設定手順例

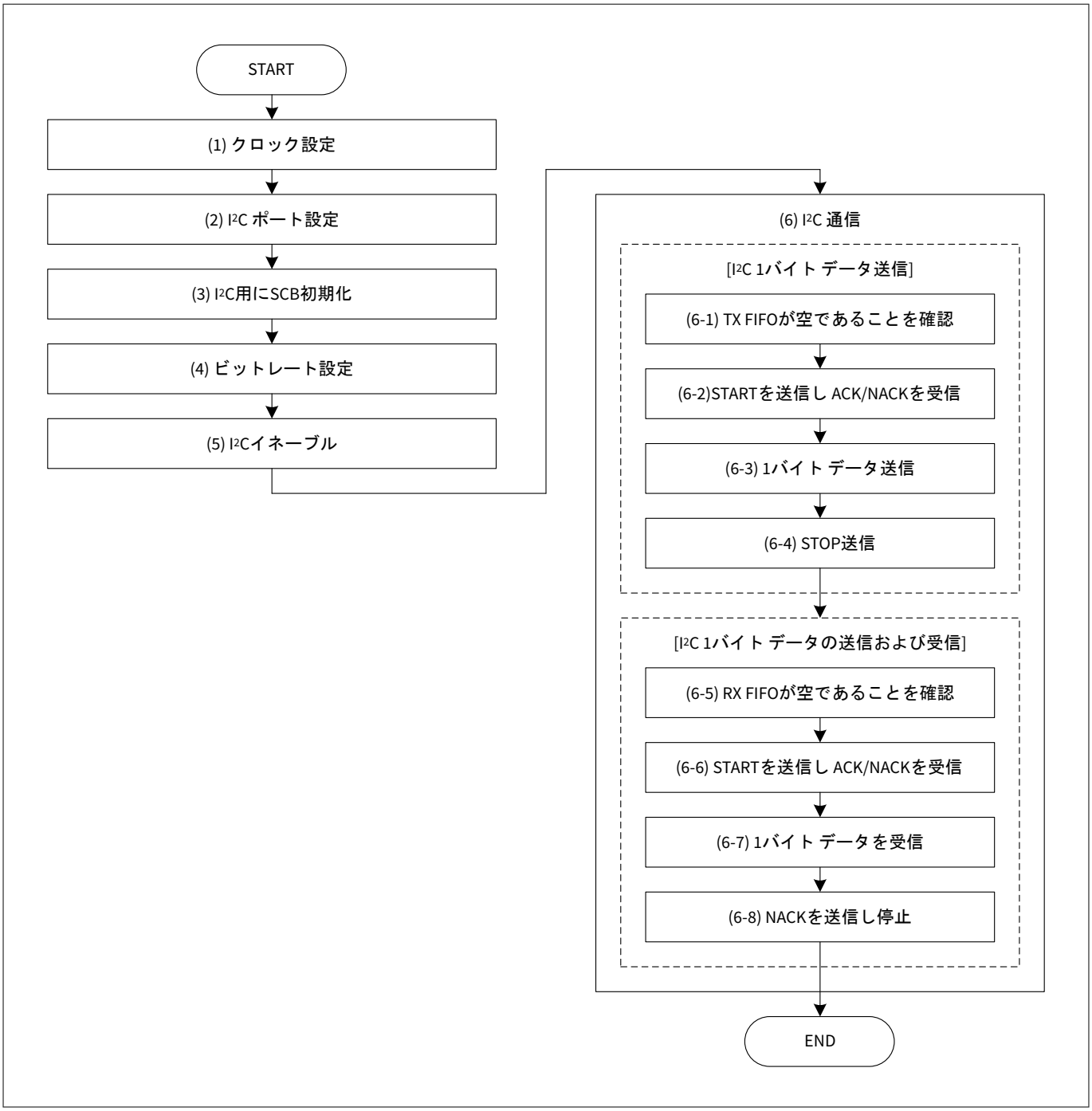

図 9　　I²C マスタモードの設定手順と動作例

1. クロックを設定します。
2. I²C ポートを設定します。
3. I²C 用に SCB を初期化します。
4. ビットレートを設定します。
5. I²C をイネーブルにします。
6. I²C 通信の例
   a. Tx FIFO が空であることを確認します。
   b. START を送信し ACK/NACK を受信します。
   c. 1 バイト データを送信します。

d. STOP を送信します。
e. Rx FIFO が空であることを確認します。
f. START を送信し ACK/NACK を受信します。
g. 1 バイト データを受信します。
h. NACK を送信し停止します。

5.1.2 設定および例

設定および例に、I²C マスタモードの SDL の設定部のパラメーターを示します。

表 8 I²C マスタモード設定パラメーター一覧

パラメーター	説明	設定値
E_SOURCE_CLK_FREQ	入力分周クロックの周波数	80000000ul (80MHz)
E_I2C_INCLK_TARGET_FREQ	周辺クロックの周波数	2000000ul (2MHz)
E_I2C_DATARATE	I2C のボーレート	100000ul
USER_I2C_SCB_PCLK	周辺クロック番号	PCLK_SCB0_CLOCK
DIVIDER_NO_1	分周器番号	1ul
E_I2C_SLAVE_ADDR	スレーブ デバイス アドレス	8ul
E_I2C_RECV_SIZE	マスタ バッファ サイズ	9ul
USER_I2C_SCB_TYPE	SCB チャネル番号	SCB0
I2C_SDA_PORT	I/O ポート番号	GPIO_PRT1
I2C_SDA_PORT_PIN	I/O ピン番号	1ul
I2C_SDA_PORT_MUX	周辺機器接続	P1_1_SCB0_I2C_SDA (14ul)
I2C_SCL_PORT	I/O ポート番号	GPIO_PRT1
I2C_SCL_PORT_PIN	I/O ピン番号	0ul
I2C_SCL_PORT_MUX	周辺機器接続	P1_0_SCB0_I2C_SCL (14ul)
g_stc_i2c_config.i2cMode	I2C マスタ/スレーブ モード	CY_SCB_I2C_MASTER (2ul)
g_stc_i2c_config.useRxFifo	受信 FIFO 制御	true
g_stc_i2c_config.useTxFifo	送信 FIFO 制御	true
g_stc_i2c_config.slaveAddress	スレーブ デバイス アドレス	E_I2C_SLAVE_ADDR (8ul)
g_stc_i2c_config.slaveAddressMask	スレーブ デバイス アドレス マスク	E_I2C_SLAVE_ADDR (8ul)
g_stc_i2c_config.acceptAddrInFifo	一致するアドレスを受信	false
g_stc_i2c_config.ackGeneralAddr	ゼネラルコールスレーブアドレスを受信	false
g_stc_i2c_config.enableWakeFromSleep	アドレス一致用クロック	false
g_stc_i2c_master_config.slaveAddress	スレーブ デバイス アドレス	E_I2C_SLAVE_ADDR (8ul)
g_stc_i2c_master_config.buffer	マスタ バッファへのポインタ	0ul
g_stc_i2c_master_config.bufferSize	現在のマスタ バッファ サイズ	0ul

(続く)

表 8 (続き) I²C マスタモード設定パラメーター一覧

パラメーター	説明	設定値
g_stc_i2c_master_config.xferPending	マスタが転送を終了する方法を格納	false
I2S_port_pin_cfg.outVal	ピン出力状態	0ul
I2S_port_pin_cfg.driveMode	GPIO 駆動モード	0ul
I2S_port_pin_cfg.hsiom	I/O ピン配線の接続	HSIOM_SEL_GPIO (0x0)
I2S_port_pin_cfg.intEdge	IRQ をトリガするエッジ	0ul
I2S_port_pin_cfg.intMask	エッジ割込みをマスク	0ul
I2S_port_pin_cfg.vtrip	入力バッファ モード	0ul
I2S_port_pin_cfg.slewRate	スルーレート	0ul
I2S_port_pin_cfg.driveSel	GPIO 駆動強度	0ul
USER_I2C_SCB_IRQN	システム割込みインデックス番号	scb_0_interrupt_IRQn
irq_cfg.sysIntSrc	システム割込みインデックス番号	USER_I2C_SCB_IRQN (IDX: 17)
irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
irq_cfg..isEnabled	CPU 割込みイネーブル	true (enable)

表 9 に、SDL のドライバ部の I2C パラメーターを示します。

表 9 関数一覧

関数	説明	備考
void Cy_SCB_I2C_DeInit (volatile stc_SCB_t *base)	SCB ブロックを初期化解除します	*base: USER_I2C_SCB_TYPE
Cy_SCB_I2C_Init (volatile stc_SCB_t *base, cy_stc_scb_i2c_config_t const *config, cy_stc_scb_i2c_context_t *context)	I2C 動作のために SCB を初期化します	*base: USER_I2C_SCB_TYPE *config: g_stc_i2c_config *context: g_stc_i2c_context
Cy_SCB_I2C_SetDataRate (volatile stc_SCB_t *base, uint32_t dataRateHz, uint32_t scbClockHz)	必要なデータレートで動作するように SCB を設定します	*base: USER_I2C_SCB_TYPE dataRateHz: E_I2C_INCLK_TARGET_FREQ scbClockHz: E_I2C_INCLK_TARGET_FREQ
Cy_SCB_I2C_Enable (volatile stc_SCB_t *base)	I2C 動作用の SCB ブロックを有効にします	*base: USER_I2C_SCB_TYPE
Cy_SCB_GetNumInTxFifo (volatile stc_SCB_t const *base)	現在の Tx FIFO にあるデータ要素の数を返します	*base: USER_I2C_SCB_TYPE

(続く)

表 9 (続き) 関数一覧

関数	説明	備考
`Cy_SCB_I2C_MasterSendStart (volatile stc_SCB_t *base, uint32_t address, uint32_t bitRnW, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)`	開始条件を生成し、読出し/書込みビットでスレーブアドレスを送信します	*base: USER_I2C_SCB_TYPE address: E_I2C_SLAVE_ADDR bitRnW: CY_SCB_I2C_WRITE_XFER timeoutMs: 2000ul *context: g_stc_i2c_context)
`Cy_SCB_I2C_MasterWriteByte (volatile stc_SCB_t *base, uint8_t theByte, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)`	スレーブに 1 バイトを送信します	*base: USER_I2C_SCB_TYPE theByte: g_send_byte timeoutMs: 2000ul *context: g_stc_i2c_context)
`Cy_SCB_I2C_MasterSendStop (volatile stc_SCB_t *base, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)`	現在のトランザクションを完了するための停止条件を生成します	*base: USER_I2C_SCB_TYPE timeoutMs: 2000ul *context: g_stc_i2c_context)
`Cy_SCB_GetNumInRxFifo (volatile stc_SCB_t const *base)`	現在の Rx FIFO にあるデータ要素の数を返します	*base: USER_I2C_SCB_TYPE
`Cy_SCB_I2C_MasterReadByte (volatile stc_SCB_t *base, guint32_t ackNack, uint8_t *byte, uint32_t timeoutMs, cy_stc_scb_i2c_context_t *context)`	スレーブから 1 バイトを読み出し、ACK を生成するか、NAK を生成する準備をします	*base: USER_I2C_SCB_TYPE ackNack: CY_SCB_I2C_NAK *byte: g_recv_byte timeoutMs: 2000ul *context: g_stc_i2c_context)

Code Listing 26 に、設定部で I²C マスタモードを設定する例を示します。

5 I²C 設定手順例

Code Listing 26 設定部で I²C マスタモードを設定する例

```
/* I2C Master Mode Test                                */
/*                                                     */
/* Partner Address(Slave): 0x08 (E_I2C_SLAVE_ADDR)    */

#define USER_I2C_SCB_TYPE       SCB0  /* Define the SCB parameters */
#define USER_I2C_SCB_PCLK       PCLK_SCB0_CLOCK  /* Define the SCB parameters */
#define USER_I2C_SCB_IRQN       scb_0_interrupt_IRQn  /* Define the SCB parameters */

#define I2C_SDA_PORT     GPIO_PRT1  /* Define the port parameters */
#define I2C_SDA_PORT_PIN (1ul)  /* Define the port parameters */
#define I2C_SDA_PORT_MUX P1_1_SCB0_I2C_SDA  /* Define the port parameters */

#define I2C_SCL_PORT     GPIO_PRT1  /* Define the port parameters */
#define I2C_SCL_PORT_PIN (0ul)  /* Define the port parameters */
#define I2C_SCL_PORT_MUX P1_0_SCB0_I2C_SCL  /* Define the port parameters */

#define DIVIDER_NO_1 (1ul)

/* Select Frequency */
#define E_SOURCE_CLK_FREQ     (80000000ul)  /* Define the clock parameters */

#define E_I2C_INCLK_TARGET_FREQ (2000000ul)  /* Define the clock parameters */
#define E_I2C_DATARATE          (100000ul)  /* Define the clock parameters */

#define E_I2C_SLAVE_ADDR        8ul
#define E_I2C_RECV_SIZE         9ul

static cy_stc_gpio_pin_config_t I2S_port_pin_cfg =  /* Configure the port parameters */
{
    .outVal    = 0ul,
    .driveMode = 0ul,            /* Will be updated in runtime */
    .hsiom     = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};


/* SCB - I2C Configuration */
static cy_stc_scb_i2c_context_t g_stc_i2c_context;
static const cy_stc_scb_i2c_config_t  g_stc_i2c_config =
{
    .i2cMode             = CY_SCB_I2C_MASTER,  /* Configure the I²C parameters */
    .useRxFifo           = true,  /* Configure the I²C parameters */
    .useTxFifo           = true,  /* Configure the I²C parameters */
    .slaveAddress        = E_I2C_SLAVE_ADDR,  /* Configure the I²C parameters */
    .slaveAddressMask    = E_I2C_SLAVE_ADDR,  /* Configure the I²C parameters */
    .acceptAddrInFifo    = false,  /* Configure the I²C parameters */
```

```
    .ackGeneralAddr      = false,  /* Configure the I²C parameters */
    .enableWakeFromSleep = false  /* Configure the I²C parameters */
};

static cy_stc_scb_i2c_master_xfer_config_t g_stc_i2c_master_config =  /* Configure the I²C
parameters */
{
    .slaveAddress = E_I2C_SLAVE_ADDR,  /* Configure the I²C parameters */
    .buffer       = 0ul,  /* Configure the I²C parameters */
    .bufferSize   = 0ul,  /* Configure the I²C parameters */
    .xferPending  = false  /* Configure the I²C parameters */
};




void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)  /* Create
the function to determine the divider division ratio */
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq);  /* Calculate the division ratio */
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,  /* Set the division ratio */
                                   (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
                                   (divSetting & 0x0000001Ful));
}


void Scb_I2C_Master_LowLevelAPI_Test(void)
{
    /*------------------------*/
    /* I2C Master Byte Write  */
    /*------------------------*/

    /* Make sure TX FIFO empty */
    while(Cy_SCB_GetNumInTxFifo(USER_I2C_SCB_TYPE) != 0ul);  /* (6-1) Make sure TX FIFO empty
(See Code Listing 31) */


    /* Send START and Receive ACK/NACK */
    CY_ASSERT(Cy_SCB_I2C_MasterSendStart(USER_I2C_SCB_TYPE, E_I2C_SLAVE_ADDR,
CY_SCB_I2C_WRITE_XFER, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS);  /* (6-2) Send START
and Receive ACK/NACK (See Code Listing 32) */


    /* Transmit One Byte Data */
    static uint8_t g_send_byte = 0xF1ul;
    CY_ASSERT(Cy_SCB_I2C_MasterWriteByte(USER_I2C_SCB_TYPE, g_send_byte, 2000ul,
&g_stc_i2c_context) == CY_SCB_I2C_SUCCESS);  /* (6-3) Transmit One Byte Data (See Code Listing
33) */
    /* Send STOP */
```

```
    CY_ASSERT(Cy_SCB_I2C_MasterSendWriteStop(USER_I2C_SCB_TYPE, 2000ul, &g_stc_i2c_context) ==
CY_SCB_I2C_SUCCESS);  /* (6-4) Send STOP (See Code Listing 34) */

    /*------------------------*/
    /* I2C Master Byte Read   */
    /*------------------------*/

    /* Make sure RX FIFO empty */
    while(Cy_SCB_GetNumInRxFifo(USER_I2C_SCB_TYPE) != 0ul);  /* (6-5) Make sure RX FIFO empty
(See Code Listing 35) */

    /* Send START and Receive ACK/NACK */
    CY_ASSERT(Cy_SCB_I2C_MasterSendStart(USER_I2C_SCB_TYPE, E_I2C_SLAVE_ADDR,
CY_SCB_I2C_READ_XFER, 2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS);  /* (6-6) Send START
and Receive ACK/NACK (See Code Listing 32) */



    /* Receive One Byte Data */
    static uint8_t g_recv_byte = 0x00ul;
    CY_ASSERT(Cy_SCB_I2C_MasterReadByte(USER_I2C_SCB_TYPE, CY_SCB_I2C_NAK, &g_recv_byte,
2000ul, &g_stc_i2c_context) == CY_SCB_I2C_SUCCESS);  /* (6-7) Receive One Byte Data (See Code
Listing 36) */
}

    /* Send NACK (and stop) */
    CY_ASSERT(Cy_SCB_I2C_MasterSendReadStop(USER_I2C_SCB_TYPE, 2000ul, &g_stc_i2c_context) ==
CY_SCB_I2C_SUCCESS);  /* (6-8) Send NACK and stop (See Code Listing 34) */
}




int main(void)
{
    SystemInit();

    /*----------------------*/
    /* Clock Configuration */  /* (1) Configure the clock */
    /*----------------------*/
    Cy_SysClk_PeriphAssignDivider(USER_I2C_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);  /*
Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(E_I2C_INCLK_TARGET_FREQ, E_SOURCE_CLK_FREQ, DIVIDER_NO_1);  /*
Configure the divider (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);  /* Enable the divider
(See Code Listing 6) */




    /*--------------------*/
    /* Port Configuration */
```

5 I²C 設定手順例

```
    /*--------------------*/
    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW;  /* (2) Configure the I²C port */
    I2S_port_pin_cfg.hsiom     = I2C_SDA_PORT_MUX;  /* (2) Configure the I²C port */
    Cy_GPIO_Pin_Init(I2C_SDA_PORT, I2C_SDA_PORT_PIN, &I2S_port_pin_cfg);  /* (2) Configure the
I²C port */

    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW;  /* (2) Configure the I²C port */
    I2S_port_pin_cfg.hsiom     = I2C_SCL_PORT_MUX;  /* (2) Configure the I²C port */
    Cy_GPIO_Pin_Init(I2C_SCL_PORT, I2C_SCL_PORT_PIN, &I2S_port_pin_cfg);  /* (2) Configure the
I²C port */




    /*-------------------------*/
    /*  Initialize & Enable I2C  */
    /*-------------------------*/
    Cy_SCB_I2C_DeInit(USER_I2C_SCB_TYPE);  /* If necessary, stop the I2C operation (See Code
Listing 27) */
    Cy_SCB_I2C_Init(USER_I2C_SCB_TYPE, &g_stc_i2c_config, &g_stc_i2c_context);  /* (3)
Initialize SCB for I2C (See Code Listing 28) */
    Cy_SCB_I2C_SetDataRate(USER_I2C_SCB_TYPE, E_I2C_DATARATE, E_I2C_INCLK_TARGET_FREQ);  /* (4)
Set the Bit Rate (See Code Listing 29) */
    Cy_SCB_I2C_Enable(USER_I2C_SCB_TYPE);  /* (5) Enable I²C (See Code Listing 30) */


    /* I2C Master Mode Test */
    Scb_I2C_Master_LowLevelAPI_Test();  /* (6) I²C Communication */

    for(;;);
}
```

*1: 詳細については、Architecture TRM の I/O システムのセクションを参照してください。

Code Listing 27～Code Listing 36 に、ドライバ部で SCB を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

5 I²C 設定手順例

Code Listing 27 Cy_SCB_I2C_DeInit() 関数

```
void Cy_SCB_I2C_DeInit(volatile stc_SCB_t *base)
{
    /* Returns block registers into the default state */
    base->unCTRL.u32Register             = CY_SCB_CTRL_DEF_VAL;  /* Set the default value to
I2C_CTRL Reg */
    base->unI2C_CTRL.u32Register         = CY_SCB_I2C_CTRL_DEF_VAL;  /* Set the default value
to I2C_CTRL Reg */
    base->unI2C_CFG.u32Register          = CY_SCB_I2C_CFG_DEF_VAL;  /* Set the default value to
I2C_CFG Reg */

    base->unRX_CTRL.u32Register          = CY_SCB_RX_CTRL_DEF_VAL;  /* Set the default value to
RX related Reg */
    base->unRX_FIFO_CTRL.u32Register     = 0ul;  /* Set the default value to RX related Reg */
    base->unRX_MATCH.u32Register         = 0ul;  /* Set the default value to RX related Reg */

    base->unTX_CTRL.u32Register          = CY_SCB_TX_CTRL_DEF_VAL;  /* Set the default value to
TX related Reg */
    base->unTX_FIFO_CTRL.u32Register     = 0ul;  /* Set the default value to TX related Reg */


    base->unINTR_SPI_EC_MASK.u32Register = 0ul;  /* Set the default value to interrupt  related
Reg */
    base->unINTR_I2C_EC_MASK.u32Register = 0ul;  /* Set the default value to interrupt  related
Reg */
    base->unINTR_RX_MASK.u32Register     = 0ul;  /* Set the default value to interrupt  related
Reg */
    base->unINTR_TX_MASK.u32Register     = 0ul;  /* Set the default value to interrupt  related
Reg */
    base->unINTR_M_MASK.u32Register      = 0ul;  /* Set the default value to interrupt  related
Reg */
    base->unINTR_S_MASK.u32Register      = 0ul;  /* Set the default value to interrupt  related
Reg */
}
```

Code Listing 28 Cy_SCB_I2C_Init() 関数

```
cy_en_scb_i2c_status_t Cy_SCB_I2C_Init(volatile stc_SCB_t *base, cy_stc_scb_i2c_config_t const
*config, cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_BAD_PARAM;  /* Check if configuration
parameter values are valid */
    un_SCB_CTRL_t         temp_CTRL         = {0ul};
    un_SCB_I2C_CTRL_t     temp_I2C_CTRL     = {0ul};
    un_SCB_RX_CTRL_t      temp_RX_CTRL      = {0ul};
    un_SCB_RX_MATCH_t     temp_RX_MATCH     = {0ul};
    un_SCB_TX_CTRL_t      temp_TX_CTRL      = {0ul};

    if ((NULL != base) && (NULL != config) && (NULL != context))
    {
        /* Configure the I2C interface */  /* Init CTRL Reg */
        temp_CTRL.stcField.u1ADDR_ACCEPT = (config->acceptAddrInFifo ? 1ul : 0ul);
        temp_CTRL.stcField.u1EC_AM_MODE  = (config->enableWakeFromSleep ? 1ul : 0ul);
        temp_CTRL.stcField.u2MEM_WIDTH   = 0ul;
        temp_CTRL.stcField.u2MODE        = CY_SCB_CTRL_MODE_I2C;
        base->unCTRL.u32Register         = temp_CTRL.u32Register;  /* Init I2C_CTRL  */

        temp_I2C_CTRL.stcField.u1S_GENERAL_IGNORE = (config->ackGeneralAddr ? 0ul : 1ul);
        temp_I2C_CTRL.stcField.u1SLAVE_MODE       = (config->i2cMode & 0x00000001ul);
        temp_I2C_CTRL.stcField.u1MASTER_MODE      = (config->i2cMode & 0x00000002ul) >> 1ul;
        base->unI2C_CTRL.u32Register              = temp_I2C_CTRL.u32Register;



        /* Configure the RX direction */
        temp_RX_CTRL.stcField.u5DATA_WIDTH = CY_SCB_I2C_DATA_WIDTH;
        temp_RX_CTRL.stcField.u1MSB_FIRST  = 1ul;
        base->unRX_CTRL.u32Register        = temp_RX_CTRL.u32Register;  /* Init RX_CTRL Reg */
        base->unRX_FIFO_CTRL.u32Register   = (config->useRxFifo ? (CY_SCB_I2C_FIFO_SIZE -
1ul) : 0ul);

        /* Set the default address and mask */
        temp_RX_MATCH.stcField.u8ADDR = ((uint32_t) config->slaveAddress << 1ul);
        temp_RX_MATCH.stcField.u8MASK = ((uint32_t) config->slaveAddressMask << 1ul);
        base->unRX_MATCH.u32Register  = temp_RX_MATCH.u32Register;  /* Init RX_MATCH Reg */

        /* Configure the TX direction */
        temp_TX_CTRL.stcField.u5DATA_WIDTH = CY_SCB_I2C_DATA_WIDTH;
        temp_TX_CTRL.stcField.u1MSB_FIRST  = 1ul;
        temp_TX_CTRL.stcField.u1OPEN_DRAIN = 1ul;
        base->unTX_CTRL.u32Register        = temp_TX_CTRL.u32Register;  /* Init TX_CTRL Reg */
        base->unTX_FIFO_CTRL.u32Register   = (config->useTxFifo ? CY_SCB_I2C_HALF_FIFO_SIZE :
1ul);  /* Init TX_FIFO Reg */


        /* Configure interrupt sources */  /* Init interrupt related Reg */
        base->unINTR_SPI_EC_MASK.u32Register = 0ul;
```

5 I²C 設定手順例

```
        base->unINTR_I2C_EC_MASK.u32Register = 0ul;
        base->unINTR_RX_MASK.u32Register     = 0ul;
        base->unINTR_TX_MASK.u32Register     = 0ul;
        base->unINTR_M_MASK.u32Register      = 0ul;

        base->unINTR_S_MASK.u32Register      = ((0ul != (CY_SCB_I2C_SLAVE & config->i2cMode)) ?
CY_SCB_I2C_SLAVE_INTR : 0ul);

        /* Initialize the context */
        context->useRxFifo = config->useRxFifo;
        context->useTxFifo = config->useTxFifo;

        context->state = CY_SCB_I2C_IDLE;

        /* Master-specific */
        context->masterStatus     = 0ul;
        context->masterBufferIdx  = 0ul;

        /* Slave-specific */
        context->slaveStatus       = 0ul;

        context->slaveRxBufferIdx  = 0ul;
        context->slaveRxBufferSize = 0ul;

        context->slaveTxBufferIdx  = 0ul;
        context->slaveTxBufferSize = 0ul;

        /* Un-register callbacks */
        context->cbEvents = NULL;
        context->cbAddr   = NULL;

        retStatus = CY_SCB_I2C_SUCCESS;
    }

    return (retStatus);
}
```

5 I²C 設定手順例

Code Listing 29 Cy_SCB_I2C_SetDataRate() 関数

```
uint32_t Cy_SCB_I2C_SetDataRate(volatile stc_SCB_t *base, uint32_t dataRateHz, uint32_t
scbClockHz)
{
    uint32_t actualDataRate = 0ul;

    if ((base->unI2C_CTRL.stcField.u1SLAVE_MODE == 1ul) && (base-
>unI2C_CTRL.stcField.u1MASTER_MODE == 0ul))
    {
        actualDataRate = Cy_SCB_I2C_GetDataRate(base, scbClockHz);

        /* Use an analog filter for the slave */
        base->unRX_CTRL.stcField.u1MEDIAN = 0ul;
        base->unI2C_CFG.u32Register       = CY_SCB_I2C_ENABLE_ANALOG_FITLER;
    }
    else
    {
        if ((scbClockHz > 0u) && (dataRateHz > 0u))
        {
            uint32_t sclLow;
            uint32_t sclHigh;
            uint32_t lowPhase;
            uint32_t highPhase;

            /* Convert scb clock and data rate in kHz */
            uint32_t scbClockKHz = scbClockHz / 1000ul;
            uint32_t dataRateKHz = dataRateHz / 1000ul;

            /* Get period of the scb clock in ns */
            uint32_t period = 1000000000ul / scbClockHz;

            /* Get duration of SCL low and high for the selected data rate */
            if (dataRateHz <= CY_SCB_I2C_STD_DATA_RATE)
            {
                sclLow  = CY_SCB_I2C_MASTER_STD_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_STD_SCL_HIGH;
            }
            else if (dataRateHz <= CY_SCB_I2C_FST_DATA_RATE)
            {
                sclLow  = CY_SCB_I2C_MASTER_FST_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_FST_SCL_HIGH;
            }
            else
            {
                sclLow  = CY_SCB_I2C_MASTER_FSTP_SCL_LOW;
                sclHigh = CY_SCB_I2C_MASTER_FSTP_SCL_HIGH;
            }

            /* Get low phase minimum value in scb clocks */
            lowPhase = sclLow / period;
            while (((period * lowPhase) < sclLow) && (lowPhase < CY_SCB_I2C_LOW_PHASE_MAX))
```

5 I²C 設定手順例

```
{
    ++lowPhase;
}

/* Get high phase minimum value in scb clocks */
highPhase = sclHigh / period;
while (((period * highPhase) < sclHigh) && (highPhase < CY_SCB_I2C_HIGH_PHASE_MAX))
{
    ++highPhase;
}

/* Get actual data rate */
actualDataRate = scbClockKHz / (lowPhase + highPhase);

uint32_t idx = 0ul;
while ((actualDataRate > dataRateKHz) &&
       ((lowPhase + highPhase) < CY_SCB_I2C_DUTY_CYCLE_MAX))
{
    /* Increase low and high phase to reach desired data rate */
    if (0ul != (idx & 0x1ul))
    {
        if (highPhase < CY_SCB_I2C_HIGH_PHASE_MAX)
        {
            highPhase++;
        }
    }
    else
    {
        if (lowPhase < CY_SCB_I2C_LOW_PHASE_MAX)
        {
            lowPhase++;
        }
    }

    idx++;

    /* Update actual data rate */
    actualDataRate = scbClockKHz / (lowPhase + highPhase);
}

/* Set filter configuration based on actual data rate */
if (actualDataRate > CY_SCB_I2C_FST_DATA_RATE)
{
    /* Use a digital filter */  /* Decide to use the Median filter */
    base->unRX_CTRL.stcField.u1MEDIAN = 1ul;
    base->unI2C_CFG.u32Register       = CY_SCB_I2C_DISABLE_ANALOG_FITLER;
}
else
{
    /* Use an analog filter */
    base->unRX_CTRL.stcField.u1MEDIAN = 0ul;
    base->unI2C_CFG.u32Register       = CY_SCB_I2C_ENABLE_ANALOG_FITLER;
}
```

```
            /* Set phase low and high */
            Cy_SCB_I2C_MasterSetLowPhaseDutyCycle (base, lowPhase);
            Cy_SCB_I2C_MasterSetHighPhaseDutyCycle(base, highPhase);

            /* Convert actual data rate in Hz */
            actualDataRate = scbClockHz / (lowPhase + highPhase);  /* Calculate the I2C Bit Rate
An example is shown below:
Bit rate     = Input Clock [Hz] / (High_phase_ovs + Low_phase_ovs)
       = PCLK(2MHz) / ((9+1) + (9+1)) = 100 [kbps] */
        }
    }

    return (actualDataRate);
}
```

Code Listing 30 Cy_SCB_I2C_Enable() 関数

```
__STATIC_INLINE void Cy_SCB_I2C_Enable(volatile stc_SCB_t *base)
{
    base->unCTRL.stcField.u1ENABLED = 1ul;  /* Set the enable bit to “1” */d
}
```

Code Listing 31 Cy_SCB_GetNumInTxFifo() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_GetNumInTxFifo(volatile stc_SCB_t const *base)
{
    return (base->unTX_FIFO_STATUS.stcField.u9USED);  /* Make sure TX FIFO empty */
}
```

Code Listing 32 Cy_SCB_I2C_MasterSendStart() 関数

```
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendStart(volatile stc_SCB_t *base, uint32_t address,
                                  uint32_t bitRnW, uint32_t timeoutMs,
                                  cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    /* Disable the I2C slave interrupt sources to protect the state */
    Cy_SCB_SetSlaveInterruptMask(base, CY_SCB_CLEAR_ALL_INTR_SRC);

    if (CY_SCB_I2C_IDLE == context->state)
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */  /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Set the read or write direction */
        context->state = CY_SCB_I2C_MASTER_ADDR;
        context->masterRdDir = (CY_SCB_I2C_READ_XFER == bitRnW);

        /* Clean up the hardware before a transfer. Note RX FIFO is empty at here */  /* Clear
the status */
        Cy_SCB_ClearMasterInterrupt(base, CY_SCB_I2C_MASTER_INTR_ALL);
        Cy_SCB_ClearRxInterrupt(base, CY_SCB_RX_INTR_NOT_EMPTY);
        Cy_SCB_ClearTxFifo(base);



        /* Generate a Start and send address byte */
        Cy_SCB_WriteTxFifo(base, (_VAL2FLD(CY_SCB_I2C_ADDRESS, address) | bitRnW));  /* Send
the address to slave */

        temp_I2C_M_CMD.u32Register               = 0ul;
        temp_I2C_M_CMD.stcField.u1M_START_ON_IDLE = 1ul;
        base->unI2C_M_CMD.u32Register            = temp_I2C_M_CMD.u32Register;

        /* Wait for a completion event from the master or slave */  /* Check the status] */
        do
        {
            locStatus = ((CY_SCB_I2C_MASTER_TX_BYTE_DONE &
Cy_SCB_GetMasterInterruptStatus(base)) |
                         (CY_SCB_I2C_SLAVE_ADDR_DONE & Cy_SCB_GetSlaveInterruptStatus(base)));

            locStatus |= WaitOneUnit(&timeout);

        } while (0ul == locStatus);

        retStatus = HandleStatus(base, locStatus, context);
    }
```

```
    /* Enable I2C slave interrupt sources */
    Cy_SCB_SetSlaveInterruptMask(base, CY_SCB_I2C_SLAVE_INTR);

    return (retStatus);
}
```

Code Listing 33 Cy_SCB_I2C_MasterWriteByte() 関数

```
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterWriteByte(volatile stc_SCB_t *base, uint8_t theByte,
                                  uint32_t timeoutMs,
                                  cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;

    if (CY_SCB_I2C_MASTER_TX == context->state)
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */  /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Send the data byte */
        Cy_SCB_WriteTxFifo(base, (uint32_t) theByte);  /* Transmit One Byte Data */

        /* Wait for a completion event from the master or slave */  /* Check the status */
        do
        {
            locStatus  = (CY_SCB_I2C_MASTER_TX_BYTE_DONE &
Cy_SCB_GetMasterInterruptStatus(base));
            locStatus |= WaitOneUnit(&timeout);

        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}
```

Code Listing 34 Cy_SCB_I2C_MasterSendStop() 関数

```
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendStop(volatile stc_SCB_t *base,uint32_t timeoutMs,
                                cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state))
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a stop (for Write direction) and NACK plus stop for the Read direction */
        temp_I2C_M_CMD.u32Register       = 0ul;
        temp_I2C_M_CMD.stcField.u1M_STOP = 1ul;
        temp_I2C_M_CMD.stcField.u1M_NACK = 1ul;
        base->unI2C_M_CMD.u32Register    = temp_I2C_M_CMD.u32Register;

        /* Wait for a completion event from the master or slave */
        do
        {
locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));

            locStatus |= WaitOneUnit(&timeout);

        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}

cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendWriteStop(volatile stc_SCB_t *base,uint32_t
timeoutMs,
                                cy_stc_scb_i2c_context_t *context)  /*
Cy_SCB_I2C_MasterSendWriteStop */
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;

    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state))
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */  /* Start the timer for status check */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a stop for Write direction */
```

```
        base->unI2C_M_CMD.stcField.u1M_STOP = 1ul;  /* Send to stop to slave device */

        /* Wait for a completion event from the master or slave */  /* Check the status */
        do
        {
            locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));

            locStatus |= WaitOneUnit(&timeout);

        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}

cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterSendReadStop(volatile stc_SCB_t *base,uint32_t
timeoutMs,
                                cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;

    if (0ul != (CY_SCB_I2C_MASTER_ACTIVE & context->state))
    {
        uint32_t locStatus;

        /* Convert the timeout to microseconds */
        uint32_t timeout = (timeoutMs * 1000ul);

        /* Generate a NACK plus for the Read direction */
        base->unI2C_M_CMD.stcField.u1M_NACK = 1ul;

        /* Wait for a completion event from the master or slave */
        Do
        {
            locStatus = (CY_SCB_I2C_MASTER_STOP_DONE & Cy_SCB_GetMasterInterruptStatus(base));

            locStatus |= WaitOneUnit(&timeout);

        } while (0ul == locStatus);

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);
    }

    return (retStatus);
}
```

Code Listing 35 Cy_SCB_GetNumInRxFifo() 関数

```
__STATIC_INLINE uint32_t Cy_SCB_GetNumInRxFifo(volatile stc_SCB_t const *base)
{
    return (base->unRX_FIFO_STATUS.stcField.u9USED);  /* Make sure RX FIFO empty */
}
```

Code Listing 36 Cy_SCB_I2C_MasterReadByte() 関数

```
cy_en_scb_i2c_status_t Cy_SCB_I2C_MasterReadByte(volatile stc_SCB_t *base, uint32_t ackNack,
                                uint8_t *byte, uint32_t timeoutMs,
                                cy_stc_scb_i2c_context_t *context)
{
    cy_en_scb_i2c_status_t retStatus = CY_SCB_I2C_MASTER_NOT_READY;
    un_SCB_I2C_M_CMD_t temp_I2C_M_CMD;

    if (CY_SCB_I2C_MASTER_RX0 == context->state)
    {
        bool rxEmpty;

        uint32_t locStatus;
        /* Convert the timeout to microseconds */
        uint32_t timeout = (timeoutMs * 1000ul);  /* Start the timer for status check */

        /* Wait for ACK/NAK transmission and data byte reception */  /* Check the status */
        do
        {
            locStatus = (CY_SCB_I2C_MASTER_RX_BYTE_DONE &
Cy_SCB_GetMasterInterruptStatus(base));
            rxEmpty   = (0ul == (CY_SCB_RX_INTR_NOT_EMPTY &
Cy_SCB_GetRxInterruptStatus(base)));

            locStatus |= WaitOneUnit(&timeout);

        } while ((rxEmpty) && (0ul == locStatus));

        /* The Read byte if available */
        if (!rxEmpty)
        {
            /* Get the received data byte */
            *byte = (uint8_t) Cy_SCB_ReadRxFifo(base);  /* Read the data */

            Cy_SCB_ClearRxInterrupt(base, CY_SCB_RX_INTR_NOT_EMPTY | CY_SCB_RX_INTR_LEVEL);
        }

        /* Convert the status from register plus timeout to the API status */
        retStatus = HandleStatus(base, locStatus, context);

        if (CY_SCB_I2C_SUCCESS == retStatus)
        {
            /* Generate ACK or wait for NAK generation */
            if (CY_SCB_I2C_ACK == ackNack)
            {
                temp_I2C_M_CMD.u32Register      = 0ul;
                temp_I2C_M_CMD.stcField.u1M_ACK = 1ul;
                base->unI2C_M_CMD.u32Register   = temp_I2C_M_CMD.u32Register;
            }
        }
    }
```

```
    return (retStatus);
}
```

5.2 スレーブモード

この例では、マスタからスレーブデバイスに write または read される、I²C スレーブを設定します。スレーブがデータ受信した場合、割込みが発生し、スレーブは read か write の手順を決定します。

5.2.1 ユースケース

- SCB モード = I²C スレーブモード
- SCB チャネル = 0
- PCLK = 2 MHz
- ビットレート = 100 kbps
- 7 ビットスレーブアドレス = 0x8
- Tx/Rx FIFO = 使用
- 最上位ビット (MSb) ファースト
- データ幅 = 8 ビット
- アナログフィルタ = 有効、デジタルフィルタ = 無効
- 有効な割込み
  - I²C_ARB_LOST (I²C スレーブ アービトレーション ロスト)
  - I²C_STOP (I²C STOP イベント検出)
  - I²C_ADDR_MATCH (I²C スレーブ アドレス一致)
  - I²C_GENERAL (I²C スレーブ ジェネラル コール アドレス受信)
  - I²C_BUS_ERROR (I²C スレーブ バスエラー検出)
- 使用ポート
  - SCL : SCB0_SCL (P1.0)
  - SDA : SCB0_SDA (P1.1)

図 10 は、SCB (スレーブ) と他の I²C マスタデバイス間の接続例を示します。

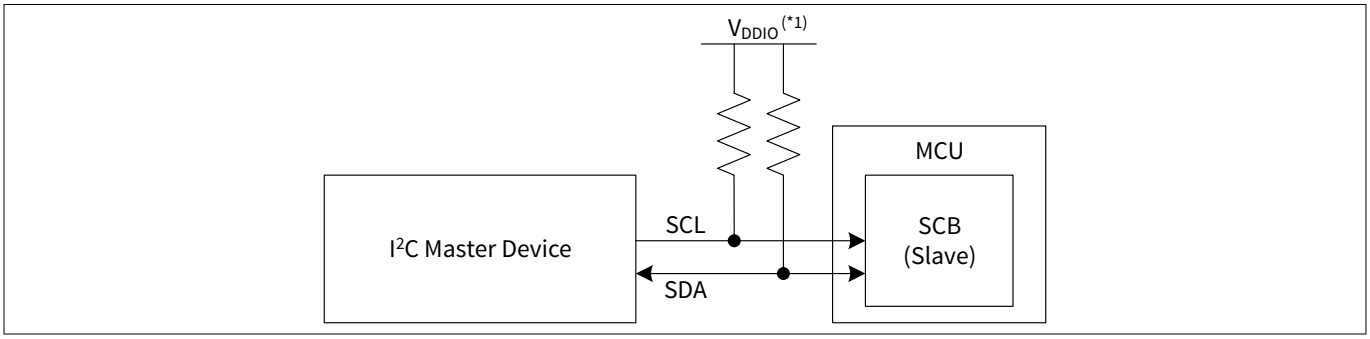

図 10 I²C (スレーブモード) 接続の例

*注: (*1) V_{DDIO} の値については、データシートを参照してください。(関連ドキュメントを参照)*

I²C スレーブモードでは、SCL と SDA 信号が、他の I²C スレーブデバイスと接続されます。マスタデバイスは、スレーブデバイスにクロック (SCL) を出力します。データ信号 (SDA) は、双方向信号です。SCL と SDA はともに抵抗を介して V_{DDIO} にプルアップされます。

5 I²C 設定手順例

図 11 に I²C スレーブモードの設定手順と動作の例を示します。

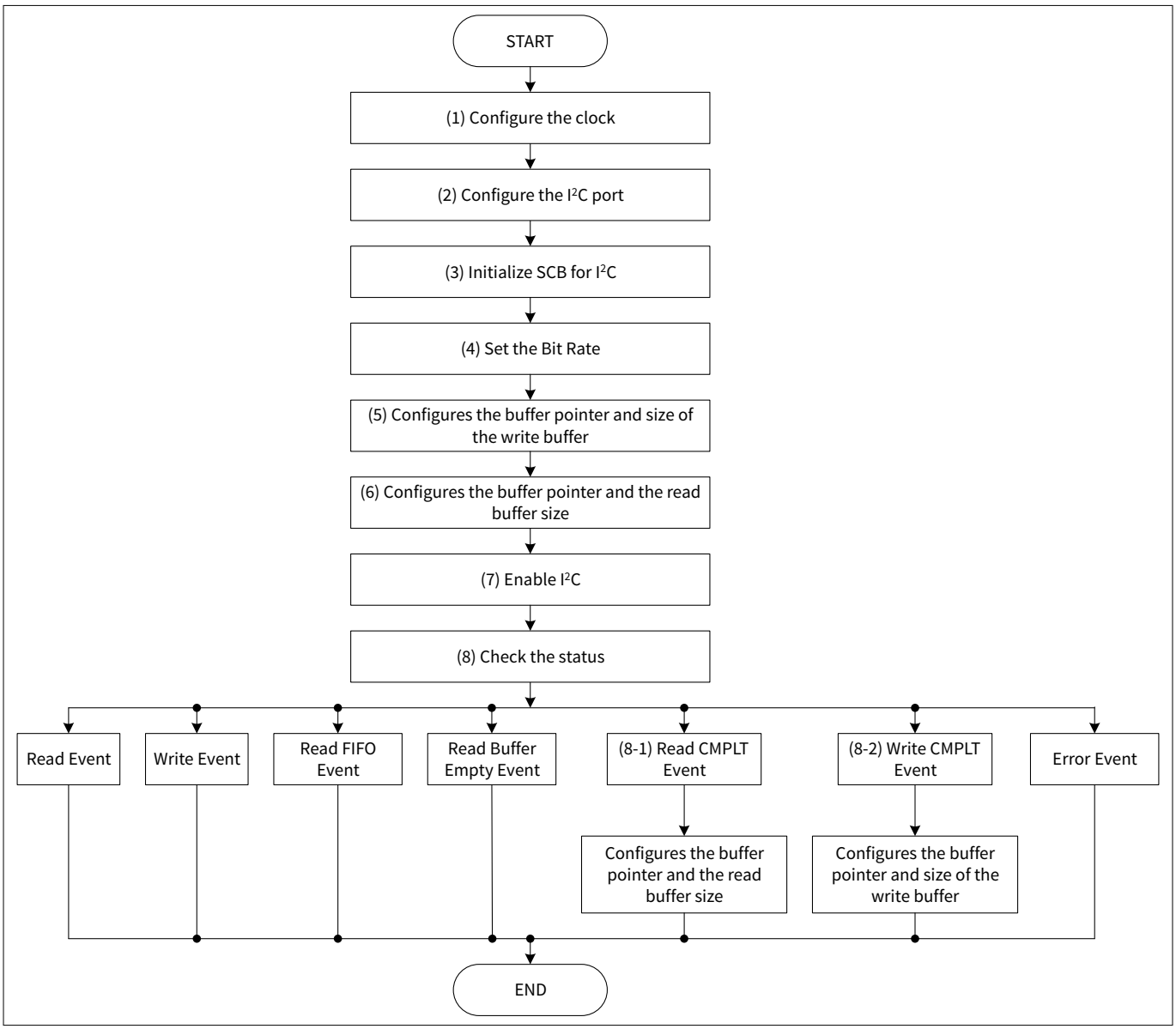

図 11　　I²C スレーブモード動作

1. クロックを設定します。
2. I2C ポートを設定します。
3. I2C 用に SCB を初期化します。
4. ビットレートを設定します。
5. バッファポインタと書込みバッファのサイズを設定します。
6. バッファポインタと読出しバッファサイズを設定します。
7. I2C を有効にします。
8. ステータスを確認します。
   a. CMPLT イベントの読出し: バッファポインタと読出しバッファサイズを設定します
   b. CMPLT イベントの書込み: バッファポインタと書込みバッファのサイズを設定します

5.2.2　　設定および例

表 10 に、I²C スレーブ モードの SDL の設定部のパラメーターを示します。

5 I²C 設定手順例

表 10 I²C スレーブモード設定パラメーター一覧

パラメーター	説明	設定値
E_SOURCE_CLK_FREQ	入力分周クロック周波数	80000000ul (80MHz)
E_I2C_INCLK_TARGET_FREQ	周辺クロック周波数	2000000ul (2MHz)
E_I2C_DATARATE	I2C ボーレート	100000ul
USER_I2C_SCB_PCLK	周辺クロック番号	PCLK_SCB0_CLOCK
DIVIDER_NO_1	分周器番号	1ul
E_I2C_SLAVE_ADDR	スレーブ デバイス アドレス	8ul
E_I2C_SLAVE_TXRX_BUF_SIZE	TXRX バッファサイズ	32ul
E_I2C_SLAVE_USER_BUF_SIZE	ユーザ バッファサイズ	32ul
USER_I2C_SCB_TYPE	SCB チャネル番号	SCB0
I2C_SDA_PORT	I/O ポート番号	GPIO_PRT1
I2C_SDA_PORT_PIN	I/O ピン番号	1ul
I2C_SDA_PORT_MUX	周辺機器接続	P1_1_SCB0_I2C_SDA (14ul)
I2C_SCL_PORT	I/O ポート番号	GPIO_PRT1
I2C_SCL_PORT_PIN	I/O ピン番号	0ul
I2C_SCL_PORT_MUX	周辺機器接続	P1_0_SCB0_I2C_SCL (14ul)
g_stc_i2c_config.i2cMode	I2C マスタ/スレーブ モード	CY_SCB_I2C_SLAVE (1ul)
g_stc_i2c_config.useRxFifo	受信 FIFO 制御	true
g_stc_i2c_config.useTxFifo	送信 FIFO 制御	true
g_stc_i2c_config.slaveAddress	スレーブ デバイス アドレス	E_I2C_SLAVE_ADDR (8ul)
g_stc_i2c_config.slaveAddressMask	スレーブ デバイス アドレス マスク	0x7Ful
g_stc_i2c_config.acceptAddrInFifo	一致するアドレスを受信	false
g_stc_i2c_config.ackGeneralAddr	ゼネラルコールスレーブアドレスを受信	true
g_stc_i2c_config.enableWakeFromSleep	アドレス一致用クロック	false
I2S_port_pin_cfg.outVal	ピン出力状態	0ul
I2S_port_pin_cfg.driveMode	GPIO 駆動モード	0ul
I2S_port_pin_cfg.hsiom	I/O ピン配線の接続	HSIOM_SEL_GPIO (0x0)
I2S_port_pin_cfg.intEdge	IRQ をトリガするエッジ	0ul
I2S_port_pin_cfg.intMask	エッジ割込みをマスク	0ul
I2S_port_pin_cfg.vtrip	入力バッファ モード	0ul
I2S_port_pin_cfg.slewRate	スルーレート	0ul
I2S_port_pin_cfg.driveSel	GPIO 駆動強度	0ul

(続く)

表 10 (続き) I²C スレーブモード設定パラメーター一覧

パラメーター	説明	設定値
USER_I2C_SCB_IRQN	システム割込みインデックス番号	scb_0_interrupt_IRQn
irq_cfg.sysIntSrc	システム割込みインデックス番号	USER_I2C_SCB_IRQN (IDX: 17)
irq_cfg.intIdx	CPU 割込み番号	CPUIntIdx3_IRQn
irq_cfg..isEnabled	CPU 割込みイネーブル	true (enable)

表 11 に、SDL のドライバ部の I²C パラメーターを示します。

表 11 関数一覧

関数	説明	備考
Cy_SCB_I2C_SlaveConfigWriteBuf (volatile stc_SCB_t const *base, uint8_t *wrBuf, uint32_t size, cy_stc_scb_i2c_context_t *context)	書込みバッファのバッファ ポインタとサイズを設定	*base: USER_I2C_SCB_TYPE *wrBuf: g_i2c_rx_buf[0] size: E_I2C_SLAVE_TXRX_BUF_SIZE *context: g_stc_i2c_context)
Cy_SCB_I2C_RegisterEventCallback (volatile stc_SCB_t const *base, scb_i2c_handle_events_t callback, cy_stc_scb_i2c_context_t *context)	通知するコールバック関数を登録	*base: USER_I2C_SCB_TYPE callback: Scb_I2C_Slave_Event *context: g_stc_i2c_context
Cy_SCB_I2C_SlaveConfigReadBuf (volatile stc_SCB_t const *base, uint8_t *rdBuf, uint32_t size, cy_stc_scb_i2c_context_t *context)	バッファ ポインタと読出しバッファ サイズを設定	*base: USER_I2C_SCB_TYPE *rdBuf: g_i2c_rx_buf[0] size: E_I2C_SLAVE_TXRX_BUF_SIZE *context: g_stc_i2c_context)

Code Listing 37 に、設定部で I²C マスタモードを設定する例を示します。

5 I²C 設定手順例

Code Listing 37 設定部での I²C スレーブモードの設定例

```
/* I2C Slave Mode Test                                */
/*                                                    */
/* Partner Address(Slave): 0x08 (E_I2C_SLAVE_ADDR)    */

#define USER_I2C_SCB_TYPE       SCB0  /* Define the SCB parameters */
#define USER_I2C_SCB_PCLK       PCLK_SCB0_CLOCK  /* Define the SCB parameters */
#define USER_I2C_SCB_IRQN       scb_0_interrupt_IRQn  /* Define the SCB parameters */

#define I2C_SDA_PORT     GPIO_PRT1  /* Define the port parameters */
#define I2C_SDA_PORT_PIN (1ul)  /* Define the port parameters */
#define I2C_SDA_PORT_MUX P1_1_SCB0_I2C_SDA  /* Define the port parameters */

#define I2C_SCL_PORT     GPIO_PRT1  /* Define the port parameters */
#define I2C_SCL_PORT_PIN (0ul)  /* Define the port parameters */
#define I2C_SCL_PORT_MUX P1_0_SCB0_I2C_SCL  /* Define the port parameters */

#define DIVIDER_NO_1 (1ul)

/* Select Frequency */
#define E_SOURCE_CLK_FREQ     (80000000ul)  /* Define the clock parameters */
#define E_I2C_INCLK_TARGET_FREQ (2000000ul)  /* Define the clock parameters */
#define E_I2C_DATARATE          (100000ul)  /* Define the clock parameters */

#define E_I2C_SLAVE_ADDR          8ul
#define E_I2C_SLAVE_TXRX_BUF_SIZE 32ul
#define E_I2C_SLAVE_USER_BUF_SIZE 32ul // should be 2^n

static cy_stc_gpio_pin_config_t I2S_port_pin_cfg =  /* Configure the port parameters */
{
    .outVal    = 0ul,
    .driveMode = 0ul,            /* Will be updated in runtime */
    .hsiom     = HSIOM_SEL_GPIO, /* Will be updated in runtime */
    .intEdge   = 0ul,
    .intMask   = 0ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
};

/* SCB - I2C Configuration */
static cy_stc_scb_i2c_context_t    g_stc_i2c_context;  /* Configure the I²C parameters */
static cy_stc_scb_i2c_config_t     g_stc_i2c_config =  /* Configure the I²C parameters */
{
    .i2cMode             = CY_SCB_I2C_SLAVE,  /* Configure the I²C parameters */
    .useRxFifo           = true,  /* Configure the I²C parameters */
    .useTxFifo           = true,  /* Configure the I²C parameters */
    .slaveAddress        = E_I2C_SLAVE_ADDR,  /* Configure the I²C parameters */
    .slaveAddressMask    = 0x7Ful,  /* Configure the I²C parameters */
    .acceptAddrInFifo    = false,  /* Configure the I²C parameters */
    .ackGeneralAddr      = true,  /* Configure the I²C parameters */
```

5 I²C 設定手順例

```
    .enableWakeFromSleep = false  /* Configure the I2C parameters */
};

/* Local Variables */
static uint8_t g_i2c_tx_buf[E_I2C_SLAVE_TXRX_BUF_SIZE] =
{
    0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x50,
    0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x60,
    0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x70,
    0x71, 0x72
};
static uint8_t          g_i2c_rx_buf[E_I2C_SLAVE_TXRX_BUF_SIZE];
static volatile uint8_t g_i2c_user_buf[E_I2C_SLAVE_USER_BUF_SIZE] = {0ul};
static uint8_t          g_i2c_user_buf_index = 0ul;


void SetPeripheFracDiv24_5(uint64_t targetFreq, uint64_t sourceFreq, uint8_t divNum)
{
    uint64_t temp = ((uint64_t)sourceFreq << 5ul);
    uint32_t divSetting;

    divSetting = (uint32_t)(temp / targetFreq);
    Cy_SysClk_PeriphSetFracDivider(CY_SYSCLK_DIV_24_5_BIT, divNum,
                                   (((divSetting >> 5ul) & 0x00000FFFul) - 1ul),
                                   (divSetting & 0x0000001Ful));
}


void Scb_I2C_Slave_Event(uint32_t locEvents)
{
    uint32_t recv_size;
    switch (locEvents)  /* (8) Check the status */
    {
    case CY_SCB_I2C_SLAVE_READ_EVENT:
        break;
    case CY_SCB_I2C_SLAVE_WRITE_EVENT:
        break;
    case CY_SCB_I2C_SLAVE_RD_IN_FIFO_EVENT:
        break;
    case CY_SCB_I2C_SLAVE_RD_BUF_EMPTY_EVENT:
        break;
    case CY_SCB_I2C_SLAVE_RD_CMPLT_EVENT:  /* (8-1) Read CMPLT Event */
        /* Clear Read Buffer (use same buffer) */
        Cy_SCB_I2C_SlaveConfigReadBuf(USER_I2C_SCB_TYPE, &g_i2c_tx_buf[0],
E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context  /* Configures the buffer pointer and the read
buffer size (See Code Listing 40) */);
        break;
    case CY_SCB_I2C_SLAVE_WR_CMPLT_EVENT:  /* (8-2) Write CMPLT Event */
        /* Copy Received Data to User Buffer(g_i2c_user_buf[32]) */
        recv_size  = Cy_SCB_I2C_SlaveGetWriteTransferCount(USER_I2C_SCB_TYPE,
&g_stc_i2c_context);
        for(uint32_t i = 0ul; i < recv_size; i++)
        {
```

```
            g_i2c_user_buf[g_i2c_user_buf_index] = g_i2c_rx_buf[i];
            g_i2c_user_buf_index = (g_i2c_user_buf_index + 1ul) & (E_I2C_SLAVE_USER_BUF_SIZE -
1ul);
        }
        /* Clear Write Buffer */
        Cy_SCB_I2C_SlaveConfigWriteBuf(USER_I2C_SCB_TYPE, &g_i2c_rx_buf[0],
E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context  /* Configures the buffer pointer and size of the
write buffer (See Code Listing 38) */);
        break;
    case CY_SCB_I2C_SLAVE_ERR_EVENT:
        break;
    default:
        break;
    }
}


int main(void)
{
    SystemInit();

    /*----------------------*/
    /* Clock Configuration */  /* (1) Configure the clock */
    /*----------------------*/
    Cy_SysClk_PeriphAssignDivider(USER_I2C_SCB_PCLK, CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);  /*
Configure the Peripheral Clock (See Code Listing 4) */
    SetPeripheFracDiv24_5(E_I2C_INCLK_TARGET_FREQ, E_SOURCE_CLK_FREQ, DIVIDER_NO_1);  /*
Configure the driver (See Code Listing 2) */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_24_5_BIT, DIVIDER_NO_1);  /* Enable the divider
(See Code Listing 5) */









    /*--------------------*/
    /* Port Configuration */
    /*--------------------*/
    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW;  /* Configure the I²C port */
    I2S_port_pin_cfg.hsiom     = I2C_SDA_PORT_MUX;  /* Configure the I²C port */
    Cy_GPIO_Pin_Init(I2C_SDA_PORT, I2C_SDA_PORT_PIN, &I2S_port_pin_cfg);  /* Configure the I²C
port */

    I2S_port_pin_cfg.driveMode = CY_GPIO_DM_OD_DRIVESLOW;  /* Configure the I²C port */

    I2S_port_pin_cfg.hsiom     = I2C_SCL_PORT_MUX;  /* Configure the I²C port */
    Cy_GPIO_Pin_Init(I2C_SCL_PORT, I2C_SCL_PORT_PIN, &I2S_port_pin_cfg);  /* Configure the I²C
port */
```

5 I²C 設定手順例

```
    /*--------------------------*/
    /*  Initialize & Enable I2C */
    /*--------------------------*/
    Cy_SCB_I2C_DeInit(USER_I2C_SCB_TYPE);  /* If necessary, stop the I2C operation (See Code
Listing 27) */
    Cy_SCB_I2C_Init(USER_I2C_SCB_TYPE, &g_stc_i2c_config, &g_stc_i2c_context);  /* (3)
Initialize SCB for I2C (See Code Listing 28) */


    Cy_SCB_I2C_SetDataRate(USER_I2C_SCB_TYPE, E_I2C_DATARATE, E_I2C_INCLK_TARGET_FREQ);  /* (4)
Set the Bit Rate (See Code Listing 29) */

    Cy_SCB_I2C_SlaveConfigWriteBuf(USER_I2C_SCB_TYPE, &g_i2c_rx_buf[0],
E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context);  /* (5) Configures the buffer pointer and size
of the write buffer (See Code Listing 38) */




    Cy_SCB_I2C_SlaveConfigReadBuf(USER_I2C_SCB_TYPE, &g_i2c_tx_buf[0],
E_I2C_SLAVE_TXRX_BUF_SIZE, &g_stc_i2c_context);  /* (6) Configures the buffer pointer and the
read buffer size (See Code Listing 40) */






    Cy_SCB_I2C_RegisterEventCallback(USER_I2C_SCB_TYPE,
(scb_i2c_handle_events_t)Scb_I2C_Slave_Event, &g_stc_i2c_context);  /* Jump the (8) (See Code
Listing 39) */


    Cy_SCB_I2C_Enable(USER_I2C_SCB_TYPE);  /* (7) Enable I2C (See Code Listing 30) */

    for(;;);
}
```

*1: 詳細については、Architecture TRM の I/O システムのセクションを参照してください。

Code Listing 38～Code Listing 40 に、ドライバ部で SCB を設定するサンプル プログラムを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

5 I²C 設定手順例

Code Listing 38 CY_SCB_I2C_SlaveConfigWriteBuf() 関数

```
void Cy_SCB_I2C_SlaveConfigWriteBuf(volatile stc_SCB_t const *base, uint8_t *wrBuf, uint32_t
size,
                                    cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;

    CY_ASSERT( ((NULL == wrBuf) && (0u == size)) || (NULL != wrBuf) );  /* Write the data to
buffer */

    context->slaveRxBuffer     = wrBuf;
    context->slaveRxBufferSize = size;
    context->slaveRxBufferIdx  = 0ul;
}
```

Code Listing 39 Cy_SCB_I2C_RegisterEventCallback() 関数

```
__STATIC_INLINE void Cy_SCB_I2C_RegisterEventCallback(volatile stc_SCB_t const *base,
            scb_i2c_handle_events_t callback, cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;

    context->cbEvents = callback;  /* Function (Scb_I2C_Slave_Event) callback */}
```

Code Listing 40 Cy_SCB_I2C_SlaveConfigReadBuf() 関数

```
void Cy_SCB_I2C_SlaveConfigReadBuf(volatile stc_SCB_t const *base, uint8_t *rdBuf, uint32_t
size,
                                   cy_stc_scb_i2c_context_t *context)
{
    /* Suppress a compiler warning about unused variables */
    (void) base;

    CY_ASSERT( ((NULL == rdBuf) && (0u == size)) || (NULL != rdBuf) );  /* Read the data from
buffer */

    context->slaveTxBuffer     = rdBuf;
    context->slaveTxBufferSize = size;
    context->slaveTxBufferIdx  = 0ul;
    context->slaveTxBufferCnt  = 0ul;
}
```

6 用語集

表 12 用語集

用語	説明
CMD_RESP mode	CMD_RESP (Command Response) モードは、EZ モードに類似です。 大きな違いは、スレーブのベースアドレスを CPU が設定するか、マスタデバイスが設定するかです。
DMA	Direct memory access
EZ mode	EZ (easy) モードは、SPI と I2C において、デバイス間の Write/Read を簡単に実現するインフィニオンのオリジナル通信プロトコルです。DeepSleep モード中、CPU の関与なしにマスタデバイスと通信可能です。
FIFO	First in First Out
I2C	Inter-Integrated Circuit。 I2C バスは、マルチマスタやマルチスレーブに対応したシリアル同期通信バスです。MCU とペリフェラルデバイス間の低速通信に使用されます。I2C バスはクロック (SCK) とデータ (SDA) の 2 線で、通常、抵抗でプルアップして使用されます。
IrDA	IrDA は、赤外線によるデータ通信の規格の一種です。
LIN	Local Interconnect Network。 LIN は、車載用途のシリアル通信ネットワークです。制御ユニットと様々なセンサ/アクチュエータ間のデータ通信に使用されます。LIN は CAN よりも低コストです。
Smart card	Smart card は、データを記録して制御する集積回路のカードです。
SPI	Serial Peripheral Interface。 SPI は、周辺デバイスとの短距離通信のための同期シリアル通信インタフェース仕様です。
UART	Universal asynchronous receiver-transmitter。 UART は、シリアル信号をパラレルに変換する、またはその逆の変換を行う受信送信回路です。MCU と外部装置間の低速通信に使用されます。

7 関連ドキュメント

以下は、TRAVEO™ T2G ファミリシリーズのデータシートとテクニカルリファレンスマニュアルです。これらのドキュメントの入手についてはテクニカルサポートに連絡してください。

- デバイスデータシート
  - CYT2B7 datasheet 32-Bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT2B9 datasheet 32-Bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
  - CYT4BF datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT4DN datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-24601)
  - CYT3BB/4BB datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
  - CYT3DL datasheet 32-Bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-27763)
- Body Controller Entry ファミリ
  - TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
  - TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
- Body Controller High ファミリ
  - TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
  - TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
- Cluster 2D ファミリ
  - TRAVEO™ T2G automotive cluster 2D family architecture technical reference manual (TRM) (Doc No. 002-25800)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN (Doc No. 002-25923)
  - TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL (Doc No. 002-29854)

8 その他の参考資料

インフィニオンは、さまざまな周辺機器にアクセスするためのサンプル ソフトウェアとして、スタートアップ コードを含むサンプル ドライバ ライブラリ (SDL) を提供しています。SDL は、公式の AUTOSAR 製品でカバーされていないドライバについて、お客様への参照としても機能します。SDL は、どの自動車規格にも適合しないため、製造目的で使用することはできません。このアプリケーション ノートのプログラムコードは SDL の一部です。SDL を取得するには、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2019-07-17	これは英語版 002-25401 Rev. **を翻訳した日本語版 Rev. **です。英語版の改訂内容: New application note.
*A	2019-12-19	これは英語版 002-25401 Rev. *A を翻訳した日本語版 Rev. *A です。英語版の改訂内容: Updated Associated Part Family as “TRAVEO™ T2G Family CYT2B/CYT4B/CYT4D Series”. Added target part numbers “CYT4D Series” related information in all instances across the document.
*B	2020-05-20	これは英語版 002-25401 Rev. *B を翻訳した日本語版 Rev. *B です。英語版の改訂内容: Updated Associated Part Family as “TRAVEO™ T2G Family CYT2/CYT3/CYT4 Series”. Changed target part numbers from “CYT2B/CYT4B/CYT4D Series” to “CYT2/CYT4 Series” in all instances across the document. Added target part numbers “CYT3 Series” in all instances across the document.
英語版(*C)	-	この版は英語版のみです。英語版の改訂内容: Updated to Infineon template. Completing Sunset Review.
*C	2022-09-08	これは英語版 002-25401 Rev. *D を翻訳した日本語版 Rev. *C です。英語版の改訂内容: Updated code examples using SDL
*D	2024-12-24	これは英語版 002-25401 Rev. *E を翻訳した日本語版 Rev. *D です。英語版の改訂内容: Template update; no content update

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-12-24

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-tsz1683609627211

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。