AN99218

FM MCU のマルチファンクション シリアル インターフェース

著者: Edison Zhang

関連製品ファミリ: FM0+、FM3、FM4

関連サンプル コード: なし

関連アプリケーション ノート: なし

本アプリケーション ノート (AN99218) では、マルチファンクション シリアル (MFS) インターフェースの様々なモードについて説明します。ご提案するこの情報により、あらゆる FM MCU において MFS モジュールを迅速に立ち上げて動作させることをお手伝いします。

目次

1 はじめに

シリアル通信インターフェース (SCI) は最も一般的な通信インターフェースです。これには、UART (Universal Asynchronous Receiver Transmitter)、SPI (Serial Peripheral Interface)、I^2C (Inter-Integrated Circuit)、LIN (Local Interconnect Network) が含まれています。ほとんどのマイクロコントローラー メーカーは、これらのインターフェースに独立した内蔵ペリフェラルを提供しています。FM ファミリのマイクロコントローラーは内蔵 MFS インターフェースを備えています。これはユーザー設定で UART やクロック同期シリアル インターフェース CSIO (SPI)、I^2C、LIN に構成することができ、アプリケーションに柔軟性と利便性を提供しています。

本アプリケーション ノートは MFS モジュールをご紹介し、PDL (Peripheral Driver Library) による MFS 使用方法をご説明します。また、各通信プロトコルのデータ フォーマットをお見せし、割り込みタイミングをご説明します。この基本的な情報は、各 MFS モードがどのように動作するかをご理解をいただくために役立ちます。最後に、PDL を使用するいくつかの例は、各モードのデータ送受信を実現する方法を示します。

2 UART

UART は、外部デバイスとの非同期通信 (スタート／ストップ ビットによる同期) を実装する汎用シリアル データ通信インターフェースです。

SMR レジスタの MD ビットを b'000 にセットした時、UART モードは設定されます。

bit7	bit6	bit5	説明
0	0	0	動作モード 0 (非同期ノーマル モード)
0	0	1	動作モード 1 (非同期マルチプロセッサ モード)
0	1	0	動作モード 2 (クロック同期モード)
0	1	1	動作モード 3 (LIN 通信モード)
1	0	0	動作モード 4 (I^2C モード)
上記以外			設定禁止

2.1 特長

- 全二重動作
- 15 ビット ボーレート選択 [1]
- 5～9 ビット データ長選択
- Non Return to Zero (NRZ) と反転 NRZ データ フォーマットのサポート
- MSB／LSB 転送方向のサポート
- ハードウェア フロー制御 [2] のサポート
- 受信エラー検出
  - フレーミング エラー
  - オーバーラン エラー
  - パリティ エラー
- 割り込み要求
  - 受信完了、フレーミング エラー、オーバーラン エラーまたはパリティ エラーの要因による受信割り込み要求
  - 送信データが空、送信バス アイドルの要因による送信割り込み要求
  - 送信 FIFO が空の要因による送信 FIFO 割り込み要求
- DMA 転送のサポート
- 送信／受信 FIFO 搭載 [3]

[1] ボーレート ジェネレータは外部クロック供給ができます。

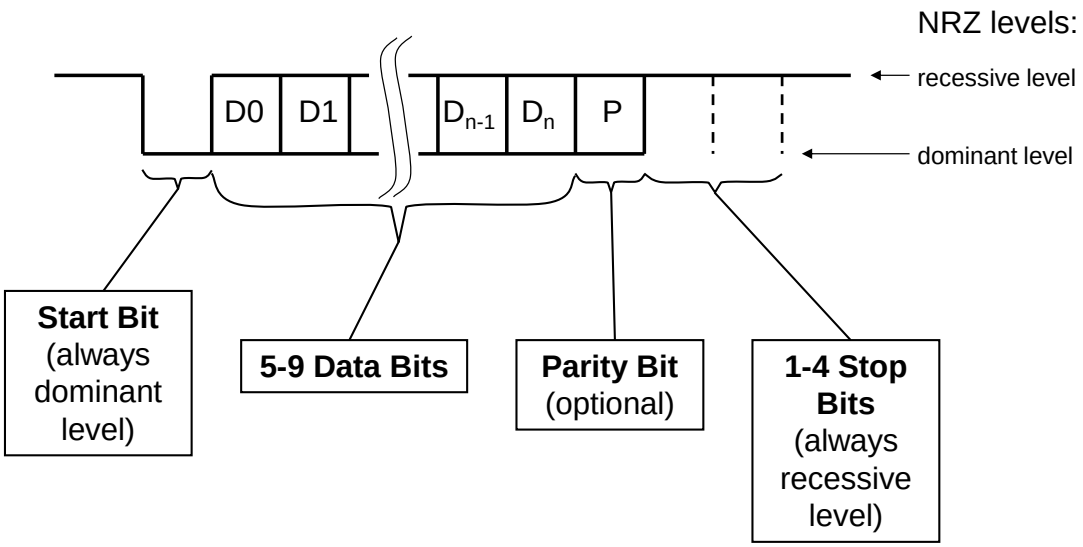
[2] 一部の MFS チャネルだけがハードウェア フロー制御機能を備えます。ご使用製品のデータシートをご参照ください。

[3] FIFO 容量は製品種類によって異なります。ご使用製品のデータシートをご参照ください。

2.2 データ フォーマット

図 1 に示すように、UART フレームはスタート ビットで始まり、その後にデータ ビットが続き、ストップ ビットで終わります。データ長は、ESCR レジスタの L2、L1、L0 ビットを設定して 5～9 ビットに選択できます。パリティ チェックは任意であり、PEN ビットの設定に依存します。パリティ チェックが有効になっている場合、偶数または奇数パリティを ESCR レジスタの P ビットで選択できます。ストップ ビット長は SMR レジスタの SBL ビットおよび ESCR レジスタの ESBL ビットで選択できます。

図 1. UART データ フォーマット



ESCR レジスタでは、ほとんどのデータ フォーマットの設定を行います。図 2 に ESCR レジスタのビットを示します。

図 2. ESCR レジスタ説明

bit	15 ... 8	7	6	5	4	3	2	1	0
フィールド	(SSR)	FLWEN	ESBL	INV	PEN	P	L2	L1	L0
属性		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初期値		0	0	0	0	0	0	0	0

[bit6] ESBL: 拡張ストップ ビット長選択ビット

Bit	説明	
0	SMR:SBL=0	1 ビット
	SMR:SBL=1	2 ビット
1	SMR:SBL=0	3 ビット
	SMR:SBL=1	4 ビット

[bit2:0] L2、L1、L0: データ長選択ビット

bit2	bit1	bit0	説明
0	0	0	8 ビット長
0	0	1	5 ビット長
0	1	0	6 ビット長
0	1	1	7 ビット長
1	0	0	9 ビット長

[bit3] P: パリティ選択ビット

Bit	説明
0	偶数パリティ
1	奇数パリティ

[bit4] PEN: パリティ 許可 ビット

Bit	説明
0	パリティ無効
1	パリティ有効

2.3 割り込み要因とタイミング

本節のタイミング図は次のビットとレジスタを参照します。

- TDRE (送信データ レジスタ エンプティ) ビットは送信データ レジスタのデータが空かどうかを示します。
- TBI (送信バス アイドル) ビットは SOT ラインがアイドル状態であるかどうかを示します。
- TIE (送信割り込み許可) ビットは送信割り込みを有効／無効にするために使用されます。
- TBIE (送信バス割り込み許可) ビットは送信バス割り込みを有効／無効にするために使用されます。
- TDR (送信データ レジスタ) はシリアル データ送信用のバッファ レジスタです。
- RDRF (受信データ レジスタ フル) ビットは受信データ レジスタが満杯であるかどうかを示します。
- FRE (フレーム エラー) ビットは、受信データのストップ ビットが 0 である場合に発生するフレーム エラーを示します。
- ORE (オーバーラン エラー) ビットは、受信データが読み出される前に次のデータが受信される場合に発生するオーバーラン エラーを示します。
- REC (受信エラー クリア) ビットは受信エラーをクリアするために使用されます。
- RIE (受信割り込み許可) ビットは送信割り込みを有効／無効にするために使用されます。
- RDR (受信データ レジスタ) はシリアルデータ受信用のバッファ レジスタです。

図 3 は FIFO を使用しない場合の UART データ送信タイミング図です。

- TRD が空の場合 (TDRE = 1)、送信割り込みを有効にすると (TIE = 1)、送信割り込み要求が発行されます。その後、転送データは TDR に書き込まれ、TDRE ビットが 0 になります。
- TDR の内容は最初に送信シフト レジスタにロードされ、その後データは SOT ピンにビットずつシフトアウトされます。転送データの最初のビットが SOT ピンにシフトアウトされた後、TDRE ビットは 1 になります。TIE = 1 になると、送信割り込み要求が発行されます。その後、次のデータを再び TDR に書き込むことができます。
- 最初のデータの全ビットが SOT ピンにシフトアウトされ、第 2 のデータの最初のビットが SOT ピンにシフトアウトされた後、TDRE ビットは 1 になり送信シフト レジスタが空であることを示します。
- 第 2 のデータの全ビットが SOT ピンにシフトアウトされた後、2 バイトの送信が完了し、TBI ビットが 1 になります。送信バス割り込みが有効になると (TBIE = 1)、送信バス割り込み要求が発行されます。

図 3. UART 送信タイミング図

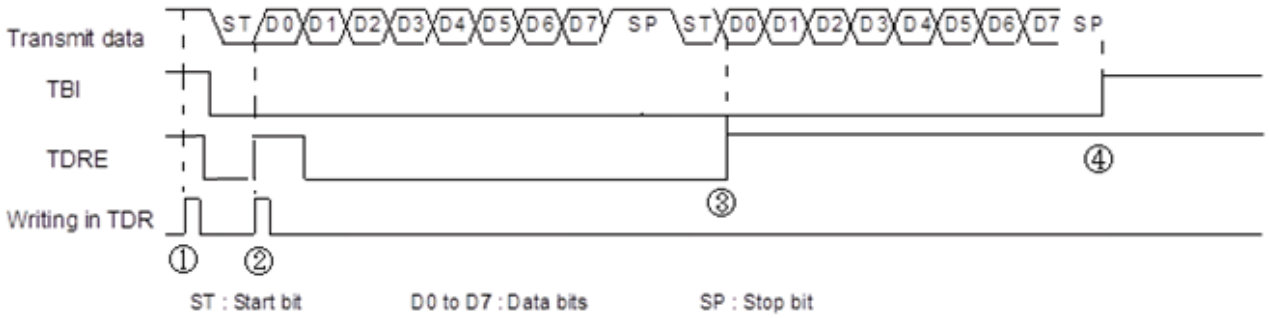


図 4 は FIFO を使用しない場合の UART データ受信タイミング図です。受信データが RDR に格納されている場合、RDRF ビットが 1 にセットされます。受信割り込みが有効になると (RIE = 1)、受信割り込み要求が発行されます。データが RDR から読み出された後、RDRF ビットは自動的にクリアされます。ただし、RDRF ビットがセットされている時、フレーム エラーが発生 (FRE = 1) 、またはオーバーラン エラーが発生した (ORE = 1) 場合、受信データは無効となり、REC ビットを 1 にセットすることでエラーをクリアする必要があります。

図 4. UART 受信タイミング図

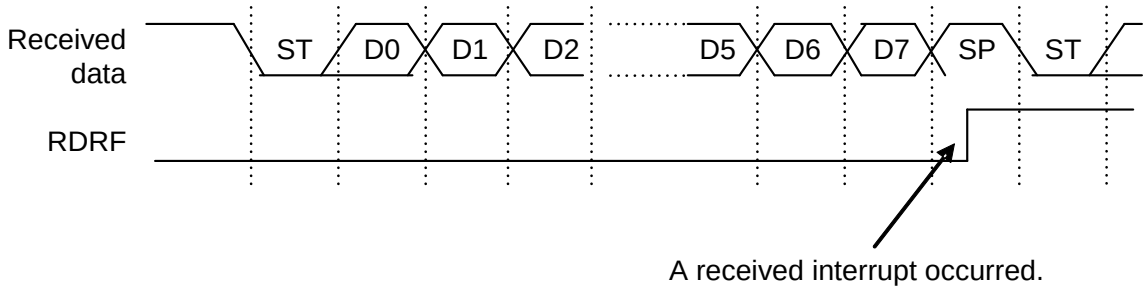


図 5 は FIFO が有効になっている場合のデータ送信タイミング図です。これは、送信 FIFO を使用して 5 バイト データを送信するためです。

1. FIFO 送信データ要求ビットが 1 にセットされている (FDRQ = 1) 場合、FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。
2. 3 バイトが送信 FIFO に書き込まれます。TDR が空であるため、最初のバイトは TDR に転送されます。その後、FDRQ ビットを手動で 0 にクリアする必要があります。この時点で、FBYTE レジスタによって表示されるように、FIFO のデータ カウントは 2 です。
3. FIFO が空になり、データの最初のビットがシフト レジスタからシフトアウトされた後、FDRQ ビットは 1 になり、FIFO が空であることを示します。FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。
4. 2 バイトが送信 FIFO に書き込まれます。その後、FDRQ ビットを手動で 0 にクリアする必要があります。この時点で、FBYTE レジスタによって表示されるように、FIFO のデータ カウントは再び 2 になります。
5. FIFO が空になり、データの最初のビットがシフト レジスタからシフトアウトされた後、FDRQ ビットは 1 になり、FIFO が空であることを示します。FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。
6. TDR が空になり、データの最初のビットがシフト レジスタからシフトアウトされた後、TDRE ビットは 1 になります。

   シフト レジスタ内のデータ ビットがすべてシフトアウトされた時、TBI ビットは 1 にセットされ、すべてのデータ送信が完了したことを示します。

図 5. FIFO が有効の場合の UART 送信タイミング図

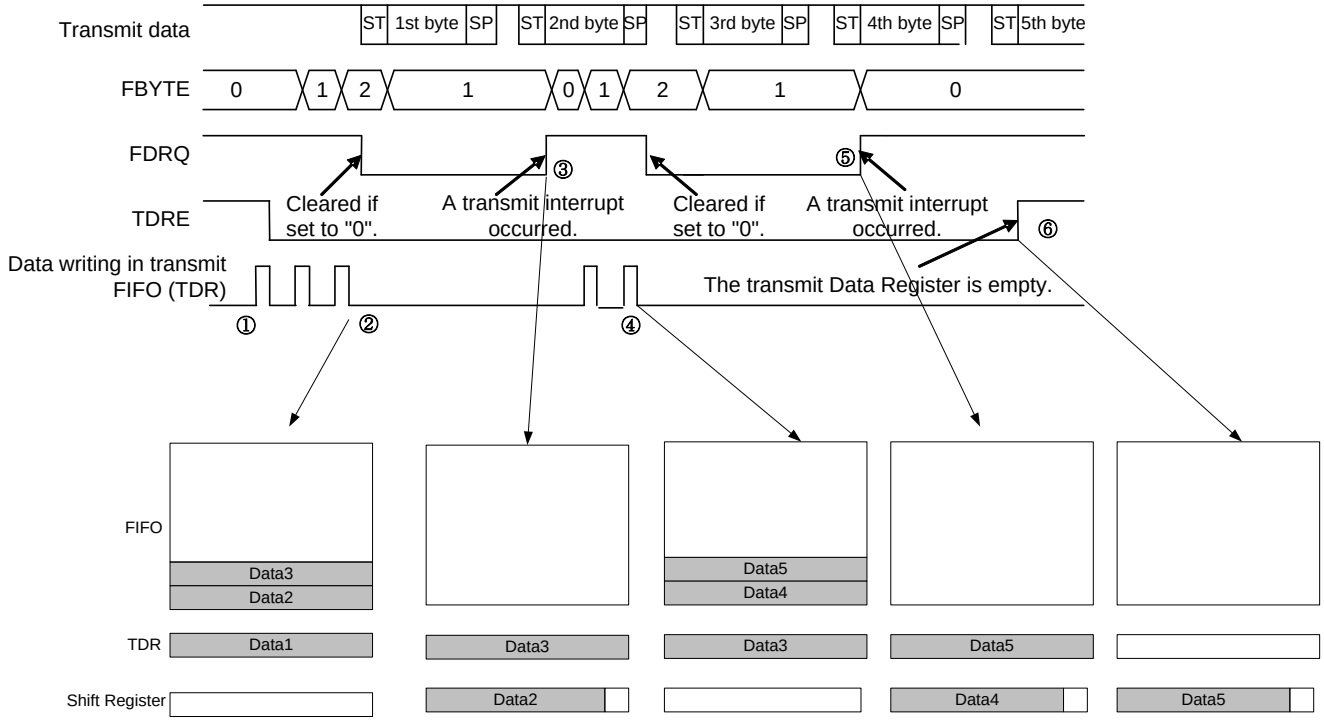
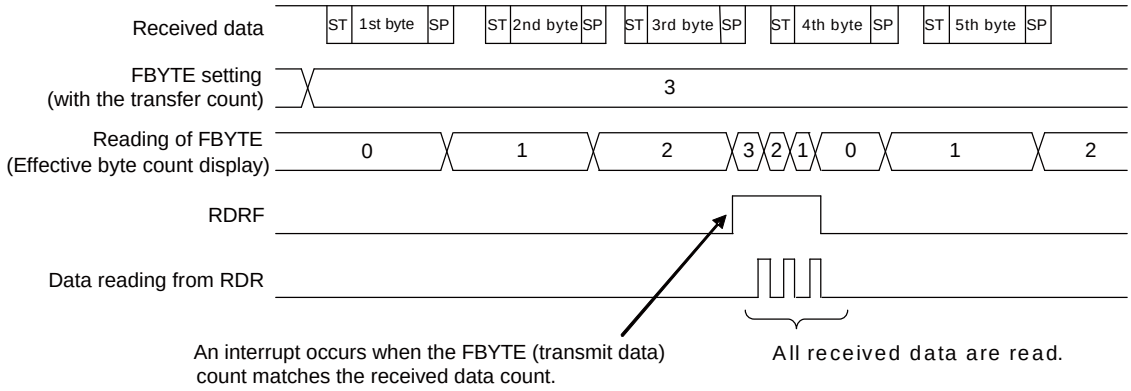


図 6 は FIFO が有効になっている場合のデータ受信タイミング図です。

1. FIFO データ数を FBYTE で設定する必要があります。
2. データ受信を開始すると、受信データは FIFO に順次格納されます。FIFO のデータ数が FBYTE と一致すると、RDRF ビットは 1 にセットされます。RIE ビットが 1 にセットされると、受信割り込みが発生します。
3. 次のデータがなく 1 バイトだけ受信した時、受信 FIFO アイドル検出が有効になり (FRIIE = 1)、受信アイドル状態が 8 ボーレート クロック以上継続すると、RDRF ビットは 1 にセットされます。
4. FIFO のデータがすべて読み出されると、RDRF ビットは自動的に 0 にクリアされます。

   受信データ数が受信 FIFO の最大容量を超えると、オーバーラン エラーが発生します。

図 6. FIFO が有効の場合の UART 受信タイミング図



2.4 ロー レベルの API

以下は、PDL の mfs.c/h ファイルにある UART ドライバーの API です。

- Mfs_Uart_Init()
- Mfs_Uart_DeInit()
- Mfs_Uart_EnableIrq()
- Mfs_Uart_DisableIrq()
- Mfs_Uart_SetBaudRate()
- Mfs_Uart_EnableFunc()
- Mfs_Uart_DisableFunc()
- Mfs_Uart_GetStatus()
- Mfs_Uart_ClrStatus()
- Mfs_Uart_SendData()
- Mfs_Uart_ReceiveData()
- Mfs_Uart_ResetFifo()
- Mfs_Uart_SetBaudRate()
- Mfs_Uart_SetFifoCount()
- Mfs_Uart_GetFifoCount()

`Mfs_Uart_Init()` は、`#stc_mfs_uart_config_t` 型の `Mfs_Uart_Init #pstcConfig` パラメーターがある UART モードの MFS インスタンスを初期化するために使用されます。この関数は基本的な UART 環境のみを設定します。`Mfs_Uart_DeInit()` はすべての MFS の UART 関連レジスタをリセットするために使用されます。

`Mfs_Uart_EnableIrq()` は`#en_uart_irq_sel_t` 列挙型で選択された UART 割り込みソースを有効にします。`Mfs_Uart_DisableIrq()` は`#en_uart_irq_sel_t` 列挙型で選択された UART 割り込みソースを無効にします。

`Mfs_Uart_SetBaudRate()` は、UART が初期化された後、UART ボーレートを変更できます。

`Mfs_Uart_EnableFunc()` は `Mfs_Uart_EnableFunc#enFunc` パラメーターで選択された UART 機能を有効にし、`Mfs_Uart_DisableFunc()` は UART 機能を無効にします。

`Mfs_Uart_GetStatus()` は `Mfs_Uart_GetStatus#enStatus` で選択された UART 状態を取得し、`Mfs_Uart_ClrStatus()` は選択された UART 状態をクリアします。いくつかの状態はハードウェアにより自動的にクリアできます。

`Mfs_Uart_SendData()` は UART 送信バッファにデータを書き込み、`Mfs_Uart_ReceiveData()` は UART 受信バッファからデータを読み出します。

`Mfs_Uart_ResetFifo()` は UART ハードウェア FIFO をリセットします。

`Mfs_Uart_SetFifoCount()` は、UART が初期化された後、FIFO サイズを変更できます。

`Mfs_Uart_GetFifoCount()` は現時点の FIFO のデータ数を取得します。

2.5 サンプル コード

ロー レベルの API ドライバーに基づいて、この例は割り込みフラグのポーリング方法を使用し、UART を介してデータを転送する方法を示します。これは UART チャネル 0 を使用し、10 バイトを転送してから 10 バイトを受信します。

UART を使用する前に、次のように UART のピン機能を設定します。

```
/* Initialize UART function I/O */
SetPinFunc_SIN0_0();
SetPinFunc_SOT0_0();
```

次に、UART 環境構造を設定し、UART チャネルを初期化します。

- ボーレート: 115200
- データ長: 8 ビット
- パリティ チェック: 無
- ストップ ビット長: 1 ビット
- ハードウェア フロー制御: 無

```
stc_mfs_uart_config_t stcUartConfig;

/* Initialize UART TX and RX channel  */
stcUartConfig.enMode = UartNormal;
stcUartConfig.u32BaudRate = 115200;
stcUartConfig.enDataLength = UartEightBits;
stcUartConfig.enParity = UartParityNone;
stcUartConfig.enStopBit = UartOneStopBit;
stcUartConfig.enBitDirection = UartDataLsbFirst;
stcUartConfig.bInvertData = FALSE;
stcUartConfig.bHwFlow = FALSE;
stcUartConfig.bUseExtClk = FALSE;
stcUartConfig.pstcFifoConfig = NULL;

if (Ok != Mfs_Uart_Init(&UART0, &stcUartConfig))
{
    while(1); // Initialization error
}
```

次のコードでは、SOT0_0 を介して 10 バイトを送信します。

```
uint8_t u8Cnt = 0;
uint8_t au8UartTxBuf[10] = {0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF};

/* Enable TX function of UART0   */
Mfs_Uart_EnableFunc(&UART0, UartTx);

while(u8Cnt < 10)
{
    /* wait until TX buffer empty */
    while (TRUE != Mfs_Uart_GetStatus((&UART0, UartTxEmpty));
    /* Write data to transmit data register */
    Mfs_Uart_SendData(UartCh0, au8UartTxBuf[u8Cnt]);

    u8Cnt++;
}

/* wait until TX idle */
while (TRUE != Mfs_Uart_GetStatus((&UART0, UartTxIdle));


/* Disable TX function of UART0   */
Mfs_Uart_DisableFunc(&UART0, UartTx);
```

次のコードでは、SIN0_0 から 10 バイトを受信します。

```
uint8_t u8Cnt = 0;
uint8_t au8UartRxBuf[10] = {0};

/* Enable RX function of UART0   */
Mfs_Uart_EnableFunc(&UART0, UartRx);

while(u8Cnt < 10)
{
    /* wait until RX buffer full */
    while(TRUE != Mfs_Uart_GetStatus(&UART0, UartRxFull));
    /* Read data from receive data register */
    au8UartRxBuf[u8Cnt] = Mfs_Uart_ReceiveData(&UART0);
    u8Cnt++;
}

/* Disable TX function of UART0   */
Mfs_Uart_DisableFunc(&UART0, UartRx);
```

注: PDL プロジェクトでは、MFS チャネルを使用する前に「PDL_PERIPHERAL_ENABLE_MFSx」の定義が有効になっていることをご確認ください。

3 CSIO (SPI)

CSIO は、外部デバイスとの同期通信用の汎用シリアル データ通信インターフェース (SPI サポート) です。これにも送信／受信 FIFO があります。

SMR レジスタの MD ビットを b'010 にセットした時、CSIO モードは設定されます。

bit7	bit6	bit5	説明
0	0	0	動作モード 0 (非同期ノーマル モード)
0	0	1	動作モード 1 (非同期マルチプロセッサ モード)
0	1	0	動作モード 2 (クロック同期モード)
0	1	1	動作モード 3 (LIN 通信モード)
1	0	0	動作モード 4 (I^2C モード)
上記以外			設定禁止

3.1 特長

- 全二重動作
- クロック同期化 (スタート／ストップ ビット未使用)
- マスター／スレーブ機能
- SPI サポート (マスターとスレーブの両モード)
- 15 ビット ボーレート選択 [1]
- 5～16 ビット データ長選択
- MSB／LSB 転送方向のサポート
- チップ セレクト機能 [2] のサポート
  - 4 チャネル制御 (シングル制御、ラウンドロビン制御)
  - 設定／ホールド／選択解除時間は変更可能
  - アクティブ レベルはチャネルごとに設定可能
- シリアル タイマーによりトリガーされるシリアル データ転送のサポート [2]
- 受信エラー検出
  - オーバーラン エラー
- 割り込み要求
  - 受信完了、オーバーラン エラーの要因による割り込み要求
  - 送信データが空、送信バス アイドルの要因による送信割り込み要求
  - 送信 FIFO が空の要因による送信 FIFO 割り込み要求
  - シリアル タイマー レジスタ (STMR) とシリアル タイマーの要因によるステータス割り込み要求
  - シリアル タイマー比較レジスタ (STMCR) 一致
- 送信／受信 FIFO 搭載 [3]

[1] ボーレート ジェネレータは外部クロック供給ができます。

[2] 一部の FM 製品のみチップ セレクトとシリアル タイマー機能を備えています。ご使用製品のデータシートをご参照ください。

[3] FIFO 容量は製品種類によって異なります。ご使用製品のデータシートをご参照ください。

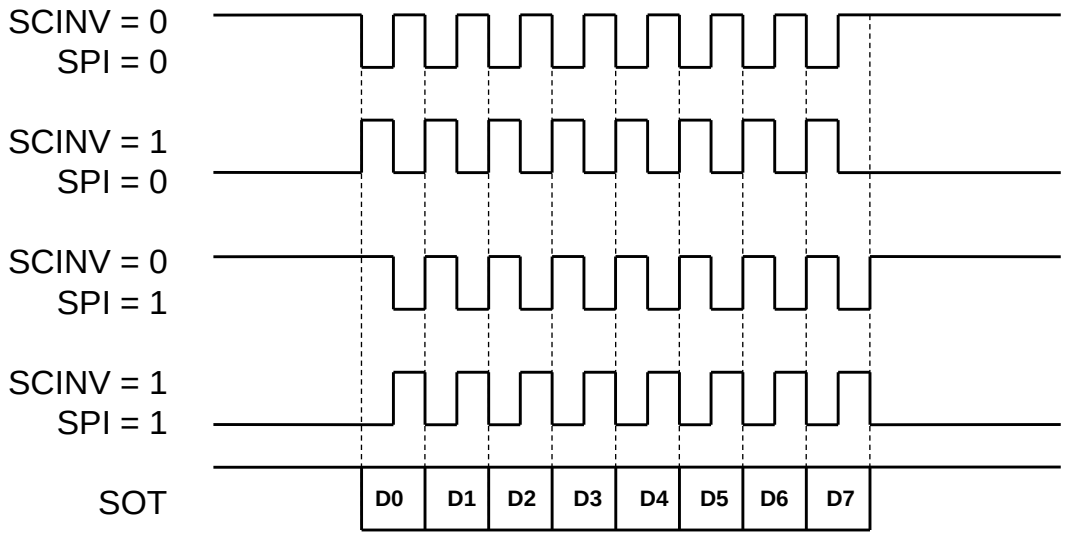
3.2 転送モード

CSIO 通信には、SCINV および SPI ビットで設定できる 4 つの転送モードがあります。これらのモードはアプリケーションのすべての同期通信タイミングを含んでいます。

項目	モード 0 (SCINV = 0、SPI = 0)	モード 1 (SCINV = 1、SPI = 0)	モード 2 (SCINV = 0、SPI = 1)	モード 3 (SCINV = 1、SPI = 1)
シリアル クロック (SCK) 信号マーク レベル	HIGH	LOW	HIGH	LOW
送信データ出力タイミング	SCK 信号の立ち下がりエッジ	SCK 信号の立ち上りエッジ	SCK 信号の立ち上りエッジ	SCK 信号の立ち下がりエッジ
受信データ サンプリング	SCK 信号の立ち上りエッジ	SCK 信号の立ち下がりエッジ	SCK 信号の立ち下がりエッジ	SCK 信号の立ち上りエッジ
データ長	5～16 ビット	5～16 ビット	5～16 ビット	5～16 ビット

モード 0 と 1 は「CSIO モード」、モード 2 と 3 は「SPI モード」と呼びます。図 7 はこれらの転送モードのタイミング図を示します。

図 7. CSIO および SPI の転送モード

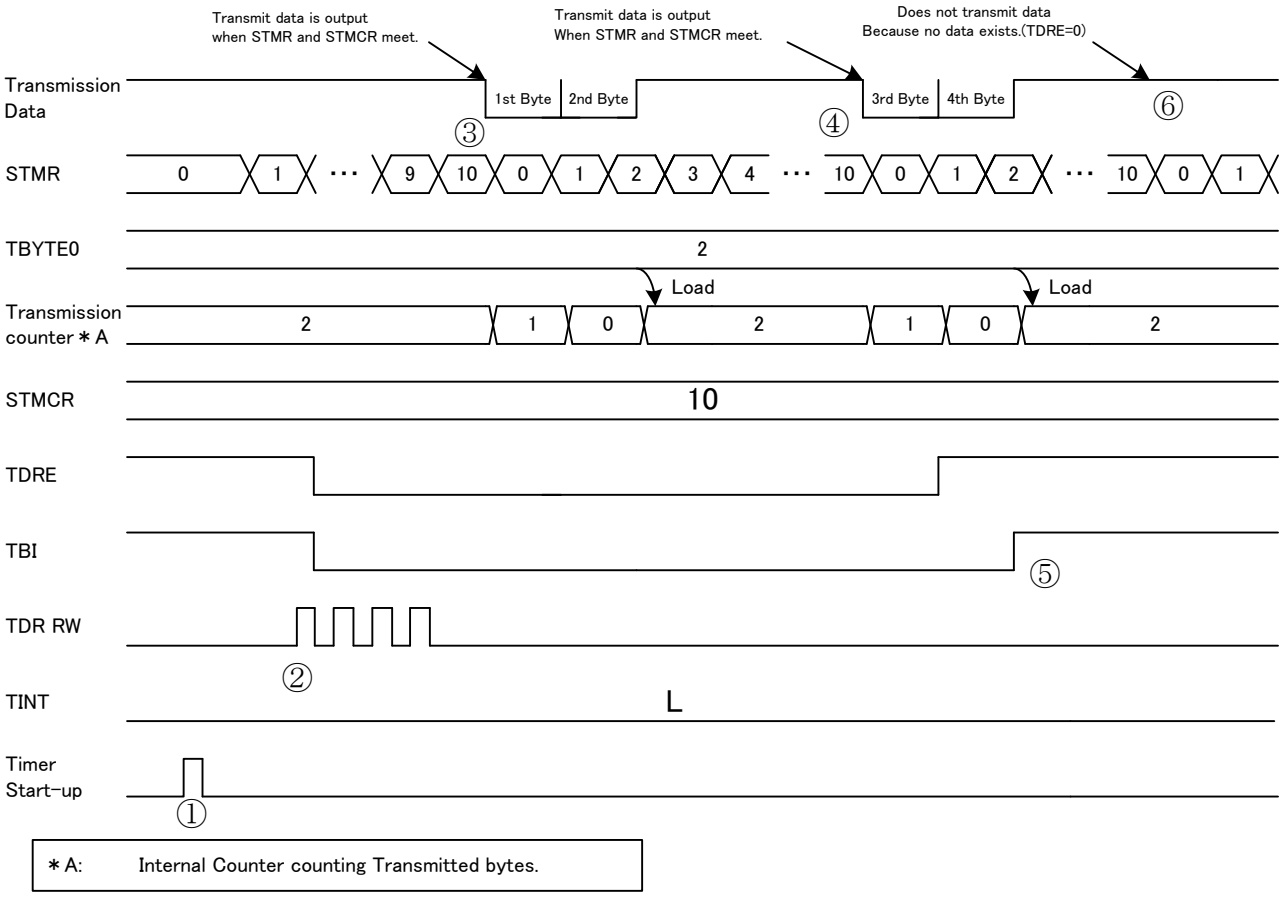


3.3 シリアル タイマー

CSIO モジュールはシリアル タイマーを内蔵しており、一定間隔に従って CSIO データ転送をトリガーすることができます。図 8 は CSIO データ転送をトリガーするシリアル タイマーの使用例を示します。

1. シリアル タイマーを使用する前に、STMCR のカウント比較値と FBYTE0 レジスタの転送バイト数を設定します。その後、シリアル タイマーを開始します。
2. 4 バイト データが送信 FIFO に書き込まれますが、データはすぐに転送されません。
3. タイマーは、タイマー クロックに応じて 0 からカウントします。STMR により反映される現在のタイマー カウントが STMCR の値と一致すると、2 バイトが転送され (TBYTE = 2)、タイマー カウントは 0 にリセットします。
4. タイマーは、タイマ クロックに応じて 0 からカウントします。STMR により反映される現在のタイマー カウントが再び STMCR の値と一致すると、更なる 2 バイトが転送され (TBYTE = 2)、タイマー カウントは 0 にリセットします。
5. すべての 4 つのデータ送信が完了した後、TBI ビットが 1 にセットされます。
6. TDR と送信 FIFO にデータがないため、STMR により反映される現在のタイマー カウントが再び STMCR の値と一致すると、データ転送は行われません。

図 8. シリアル タイマー動作タイミング



注:

1. FM ファミリ製品の一部だけがシリアル タイマー機能を持っています。ご使用製品のデータシートをご参照ください。
2. CSIO シリアル タイマー付きの製品では CSIO シリアル タイマーは CSIO マスター モードだけで動作できます。

3.4 チップ セレクト機能

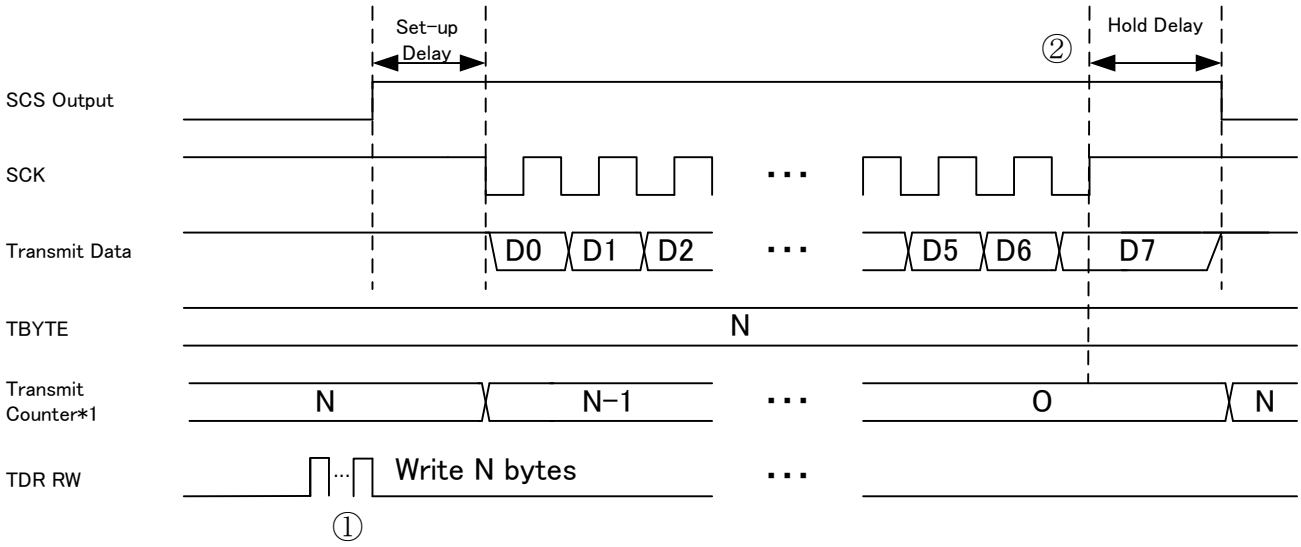
CSIO モジュールにはデータ転送が利用可能であるかどうかを制御するためのチップ セレクト ピン (SCS ピン) があります。マスター モードとスレーブ モードの両方でチップ セレクト機能をサポートしていますが、SCS0 ピンだけがスレーブ モードでチップ セレクト ピンとして使用できます。

図 9 はマスター転送モード 0 (MS = 0、SPI = 0、SCINV = 0) でチップ セレクト ピンを使用して N バイトを転送する際のタイミング図です。

1. データ送信を開始する前に、データ転送数を TBYTE で指定する必要があります。そして、シリアル チップ セレクト制御ステータス レジスタ (SCSCR) の SST と SED のビットを設定することによりチップ セレクト ピンを選択します。CSEN ビットを 1 に設定することによりチップ セレクト機能を有効にし、TEX ビットを 1 に設定することによりデータ送信を可能にします。最後に、N バイトが送信 FIFO に書き込まれ、セットアップ ディレイ値の経過後にデータ送信を開始します。
2. N バイトの転送が完了すると、SCK は HIGH になり、SCS はホールド ディレイ値後に HIGH になります。

   SCS ピンの非アクティブ レベルは SCSCR の CSLVL ビットによって設定されますが、SCS0 だけがこの機能を持っています。セットアップ ディレイ値とホールド ディレイ時間はシリアル チップ セレクト タイミング レジスタ (SCSTR) で設定されます。

図 9. チップ セレクト機能を使用した CSIO データ転送



*1:Internal Counter counting transmit bytes.

注: FM ファミリ製品の一部だけがチップ セレクト機能を持っています。ご使用製品のデータシートをご参照ください。

3.5 割り込み要因とタイミング

本節のタイミング図は次のビットとレジスタを参照します。

- TDRE (送信データ レジスタ エンプティ) ビットは送信データ レジスタのデータが空かどうかを示します。
- TBI (送信バス アイドル) ビットは SOT ラインがアイドル状態であるかどうかを示します。
- TIE (送信割り込み許可) ビットは送信割り込みを有効／無効にするために使用されます。
- TBIE (送信バス割り込み許可) ビットは送信バス割り込みを有効／無効にするために使用されます。
- TDR (送信データ レジスタ) はシリアル データ送信用のバッファ レジスタです。
- RDRF (受信データ レジスタ フル) ビットは受信データ レジスタが満杯であるかどうかを示します。
- ORE (オーバーラン エラー) ビットは、受信データが読み出される前に次のデータが受信される場合に発生するオーバーラン エラーを示します。
- REC (受信エラー クリア) ビットは受信エラーをクリアするために使用されます。
- RIE (受信割り込み許可) ビットは送信割り込みを有効／無効にするために使用されます。
- RDR (受信データ レジスタ) はシリアル データ受信用のバッファ レジスタです。

図 10 は FIFO を使用しない場合の CSIO データ送信タイミング図です。

1. TRD が空になる時 (TDRE = 1)、送信割り込みを有効にすると (TIE = 1)、送信割り込み要求が発行されます。その後、転送データは TDR に書き込むことができ、TDRE ビットが 0 になります。
2. TDR の内容は最初に送信シフト レジスタにロードされ、その後データは SOT ピンにビットずつシフトアウトされます。送信シフト レジスタに TDR のデータがロードされた後、TDRE ビットは 1 になります。TIE = 1 になると、送信割り込み要求が発行されます。その後、次のデータを再び TDR に書き込むことができます。
3. 最初のデータのすべてのビットは SOT ピンにシフトアウトされ、次のデータは再び TDR から送信シフト レジスタにロードされます。その後、TDRE ビットは 1 になり、送信シフト レジスタが空であることを示します。
4. 2 番目のデータの全ビットが SOT ピンにシフトアウトされた後、2 バイトの送信が完了し、TBI ビットが 1 になります。送信バス割り込みが有効 (TBIE = 1) になると、送信バス割り込み要求が発行されます。

図 10. CSIO データ送信タイミング図

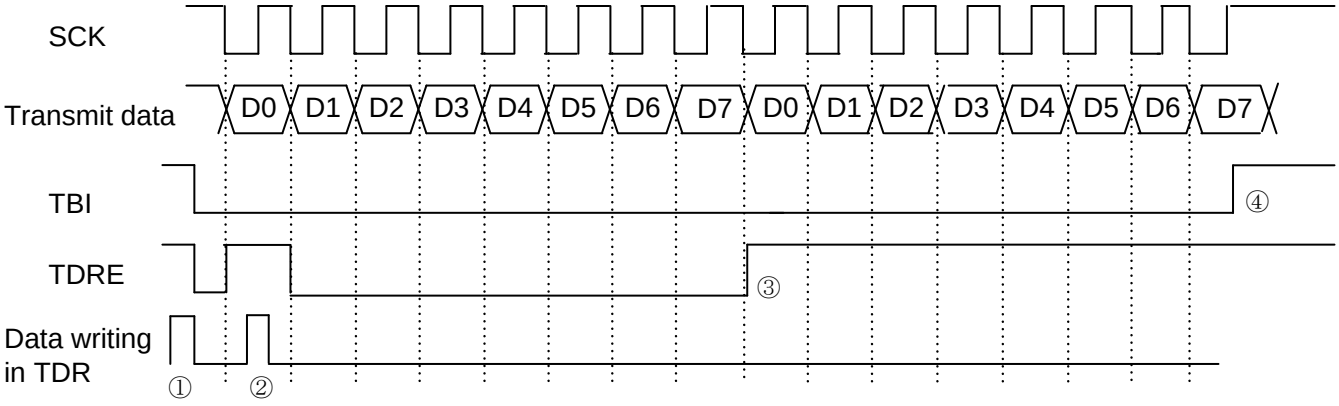
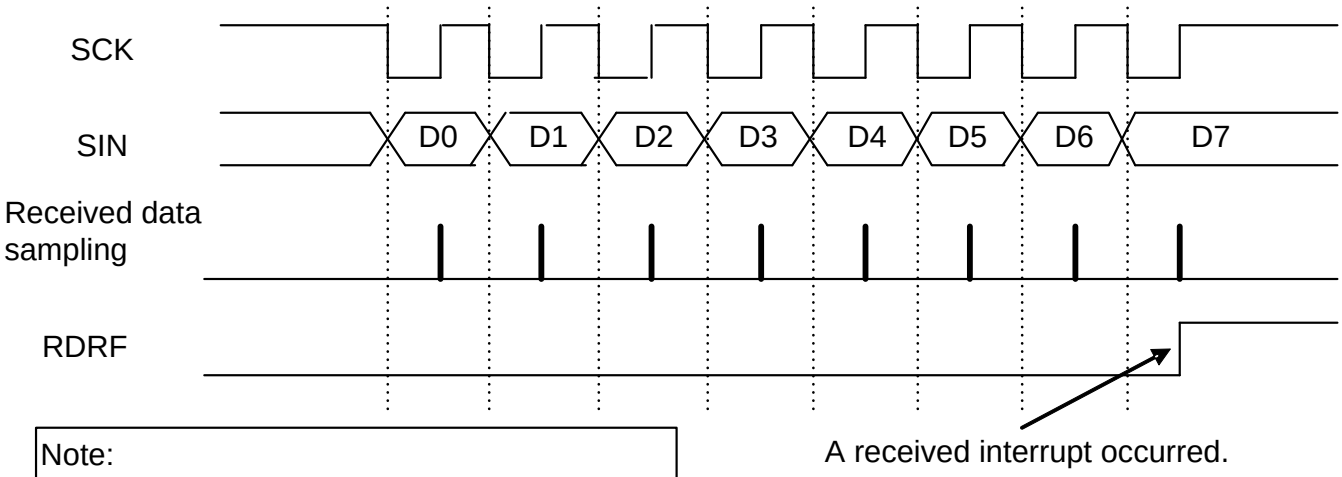


図 11 は FIFO を使用しない場合の CSIO データ受信タイミング図です。受信データが RDR に格納されている時、RDRF ビットが 1 にセットされます。受信割り込みが有効になると (RIE = 1)、受信割り込み要求が発行されます。データが RDR から読み出された後、RDRF ビットは自動的にクリアされます。ただし、RDRF ビットがセットされている時、オーバーラン エラーが発生した (ORE = 1) 場合、受信データは無効になり、REC ビットを 1 にセットすることでエラーをクリアする必要があります。

図 11. CSIO データ受信タイミング図



Note:
This figure shows the signal timing under the following conditions.
SCR: MS=1, SPI=0
ESCR: L2 to L0=0b000
SMR:SCINV=0, BDS=0, SCKE=0, SOE=0

図 12 は FIFO が有効になっている場合のデータ送信タイミング図です。これは、送信 FIFO を使用して 4 バイト データを送信するためです。

1. FDRQ = 1 の時、FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。
2. 3 バイトが送信 FIFO に書き込まれます。TDR が空であるため、最初のバイトは TDR に転送されます。その後、FDRQ ビットを手動で 0 にクリアする必要があります。この時点で、FBYTE レジスタによって表示されるように、FIFO のデータ カウントは 2 です。
3. FIFO が空になった後、FDRQ ビットは 1 になり、FIFO が空であることを示します。FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。

4. 1 バイトが送信 FIFO に書き込まれます。その後、FDRQ ビットを手動で 0 にクリアする必要があります。この時点で、FBYTE レジスタによって表示されるように、FIFO のデータ カウントは 1 です。
5. FIFO が空になった後、FDRQ ビットは 1 になり、FIFO が空であることを示します。FIFO 送信割り込みが有効になると (FTIE = 1)、送信割り込みが発生します。
6. TDR が空になった後、TDRE ビットは 1 になります。

シフト レジスタ内のデータ ビットがすべてシフトアウトされた時、TBI ビットは 1 にセットされ、すべてのデータ送信が完了したことを示します。

図 12. FIFO が有効の場合の CSIO データ送信タイミング図

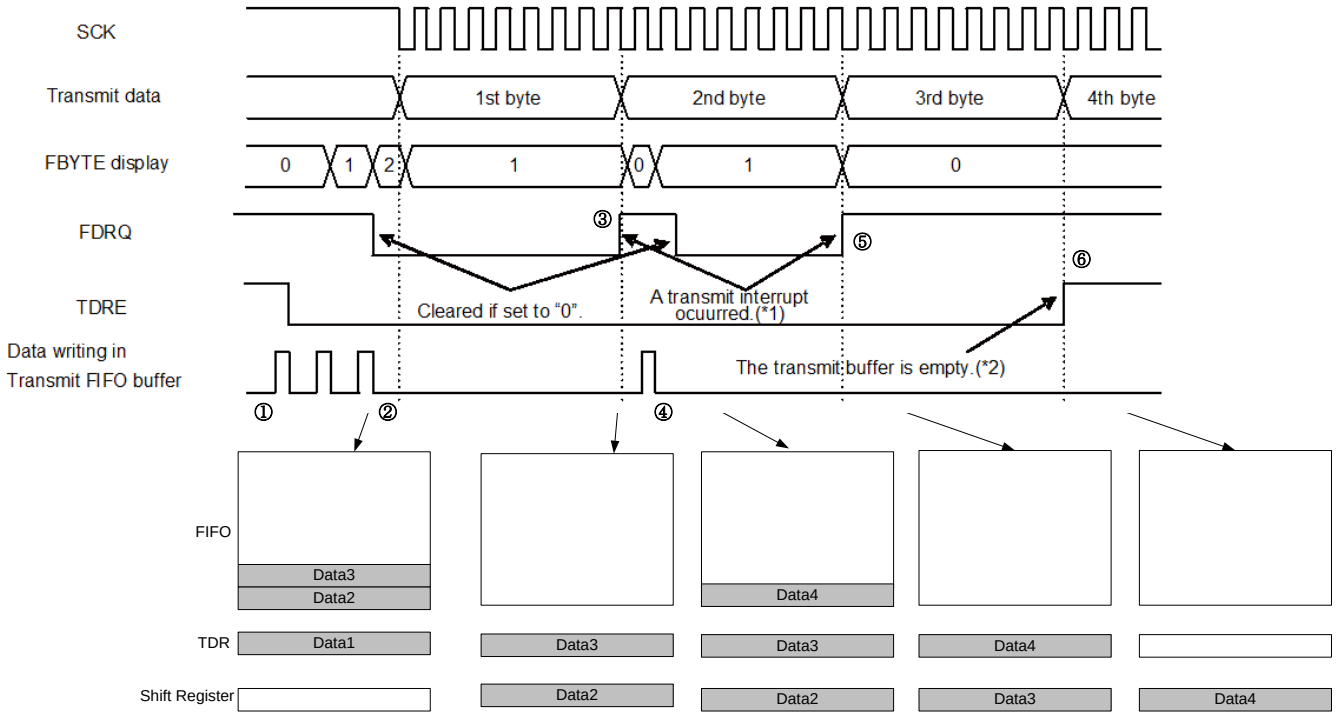
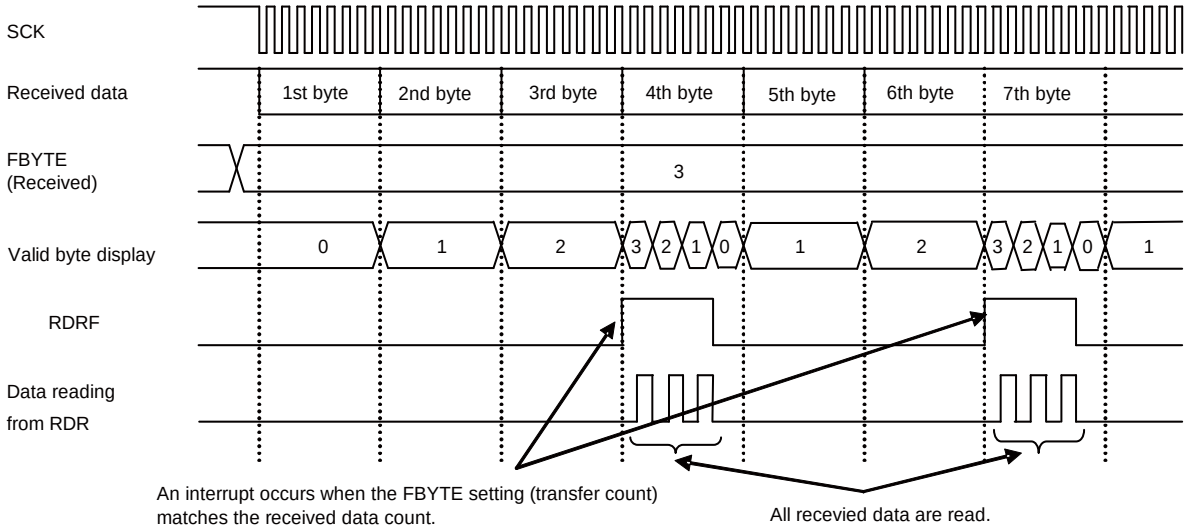


図 13 は FIFO が有効になっている場合のデータ受信タイミング図です。

1. FIFO 一致数を FBYTE で設定する必要があります。
2. データ受信を開始すると、受信データは FIFO に順次格納されます。FIFO のデータ数が FBYTE と一致すると、RDRF ビットは 1 にセットされます。RIE ビットが 1 にセットされると、受信割り込みが発生します。
3. 次のデータがなく 1 バイトだけ受信した時、受信 FIFO アイドル検出が有効になり (FRIIE = 1)、受信アイドル状態が 8 ボーレート クロック以上継続すると、RDRF ビットは 1 にセットされます。
4. FIFO のデータがすべて読み出されると、RDRF ビットは自動的に 0 にクリアされます。

   受信データ数が受信 FIFO の最大容量を超えると、オーバーラン エラーが発生します。

図 13. FIFO が有効の場合の CSIO データ受信タイミング図



3.6 ロー レベルの API

以下は、PDL の mfs.c/h ファイルにある CSIO ドライバーの API です。

- Mfs_Csio_Init()
- Mfs_Csio_DeInit()
- Mfs_Csio_EnableIrq()
- Mfs_Csio_DisableIrq()
- Mfs_Csio_SetBaudRate()
- Mfs_Csio_SetTimerCompareValue()
- Mfs_Csio_SetCsTransferByteCount()
- Mfs_Csio_SetCsHoldStatus()
- Mfs_Csio_SetTimerTransferByteCount()
- Mfs_Csio_EnableFunc()
- Mfs_Csio_DisableFunc()
- Mfs_Csio_GetStatus()
- Mfs_Csio_ClrStatus()
- Mfs_Csio_SendData()
- Mfs_Csio_ReceiveData()
- Mfs_Csio_ResetFifo()
- Mfs_Csio_SetFifoCount()
- Mfs_Csio_GetFifoCount()

`Mfs_Csio_Init()` は`#stc_mfs_csio_config_t` 型の pstcConfig パラメーターがある CSIO モードの MFS インスタンスを初期化するために使用されます。Mfs_Csio_DeInit() はすべての MFS の CSIO 関連レジスタをリセットするために使用されます。

`Mfs_Csio_EnableIrq()` は`#en_csio_irq_sel_t` 列挙型で選択された CSIO 割り込みソースを有効にします。`Mfs_Csio_DisableIrq()` は`#en_csio_irq_sel_t` 列挙型で選択された CSIO 割り込みソースを無効にします。

`Mfs_Csio_SetBaudRate()` は、CSIO が初期化された後、CSIO ボー レートを変更できるようにします。

`Mfs_Csio_SetTimerCompareValue()` は CSIO シリアル タイマーの比較値を変更します。

`Mfs_Csio_SetCsTransferByteCount()` は選択されたチップ セレクト ピンの転送バイト数を変更します。

`Mfs_Csio_SetTimerTransferByteCount()` はシリアル タイマーによってトリガーされる転送プロセスの転送バイト数を設定します。

`Mfs_Csio_SetCsHoldStatus()` は 1 回の転送が終了した後の CS ピンのホールド状態を設定します。

`Mfs_Csio_EnableFunc()` は `Mfs_Csio_EnableFunc#enFunc` パラメーターで選択された CSIO 機能を有効にし、`Mfs_Csio_DisableFunc()` は CSIO 機能を無効にします。

`Mfs_Csio_GetStatus()` は `Mfs_Csio_GetStatus#enStatus` で選択された状態を取得し、`Mfs_Csio_ClrStatus()` は選択された CSIO 状態をクリアします。いくつかの状態はハードウェアにより自動的にクリアされます。

`Mfs_Csio_SendData()` は CSIO 送信バッファに 1 バイト データを書き込み、`Mfs_Csio_ReceiveData()` は CSIO 受信バッファから 1 バイト データを読み出します。データのビット長は `Mfs_Csio_Init()` で設定されます。

`Mfs_Csio_ResetFifo()` は CSIO ハードウェア FIFO をリセットします。

`Mfs_Csio_SetFifoCount()` は、CSIO が初期化された後、FIFO のサイズを変更できるようにします。`Mfs_Csio_GetFifoCount()` は現時点の FIFO のデータ数を取得します。

3.7 サンプル コード

ロー レベルの API ドライバーに基づいて、この例は割り込みフラグのポーリング方法を使用し、CSIO を介してデータ転送する方法を示します。これは CSIO チャネル 0 を使用し、10 バイトを転送してから 10 バイトを受信します。

CSIO を使用する前に、次のように CSIO のピン機能を設定します。

```
/* Initialize CSIO function I/O */
  SetPinFunc_SIN0_0();
  SetPinFunc_SOT0_0();
  SetPinFunc_SCK0_0();
```

次に、CSIO 環境構造を設定し、CSIO チャネルを初期化します。

- モード: マスター モード
- ボーレート: 100kbps
- データ長: 8 ビット
- 方向: MSB ファースト

```
stc_mfs_csio_config_t stcCsioConfig;

 /* Clear configuration structure */
 PDL_ZERO_STRUCT(stcCsioConfig);

 /* Initialize CSIO master  */
 stcCsioConfig.enMsMode = CsioMaster;
 stcCsioConfig.enActMode = CsioActNormalMode;
 stcCsioConfig.bInvertClk = FALSE;
 stcCsioConfig.u32BaudRate = 100000;
 stcCsioConfig.enDataLength = CsioEightBits;
 stcCsioConfig.enBitDirection = CsioDataMsbFirst;
 stcCsioConfig.enSyncWaitTime = CsioSyncWaitZero;
 stcCsioConfig.pstcFifoConfig = NULL;

 if (Ok != Mfs_Csio_Init(&CSIO0, &stcCsio0Config))
 {
     While(1);
 }
```

次のコードでは、SOT ピンを介して CSIO の 10 バイトを送信します。

```
 uint8_t u8Cnt = 0;
 uint8_t au8TxBuf[10] = {0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF};

 /* Enable TX function of CSIO0   */
 Mfs_Csio_EnableFunc(&CSIO0, CsioTx);

 while(u8Cnt < 10)
 {
     /* Wait until transmit data register empty */
     while (TRUE != Mfs_Csio_GetStatus(&CSIO0,CsioTxEmpty));
     /* Write data to transmit data register */
     Mfs_Csio_SendData(&CSIO0, au8TxBuf[u8Cnt], TRUE);
     u8Cnt++;
 }

 /* Wait until master TX bus idle */
 while (TRUE != Mfs_Csio_GetStatus(&CSIO0, CsioTxIdle));

 /* Disable TX function of CSIO0   */
 Mfs_Csio_DisableFunc(&CSIO0, CsioTx);
```

次のコードでは、SIN ピンを介して CSIO の 10 バイトを受信します。同期通信では、マスターがスレーブからデータを受信していても、クロック ラインは常にマスターにより制御されるため、ご注意ください。マスターがスレーブからデータを受信する必要がある場合、ダミー データをクロックを生成するために送信する必要があります。

```
uint8_t u8Cnt = 0;
uint8_t au8RxBuf[10];

/* Enable TX and RX function of CSIO0   */
Mfs_Csio_EnableFunc(&CSIO0, CsioTx);
Mfs_Csio_EnableFunc(&CSIO0, CsioRx);

while(u8Cnt < 10)
{
    /* write a dummy data */
    Mfs_Csio_SendData(&CSIO0, 0x00u, FALSE);

    /* wait until receive data register full */
    while(TRUE != Mfs_Csio_GetStatus(&CSIO0, CsioRxFull));
    /* Read data from receive data register */
    au8RxBuf[u8Cnt] = Mfs_Csio_ReceiveData(&CSIO0);
    u8Cnt++;
}

/* Wait until master TX bus idle */
while (TRUE != Mfs_Csio_GetStatus(&CSIO0, CsioTxIdle));

/* Disable RX function of CSIO0   */
Mfs_Csio_DisableFunc(&CSIO0, CsioTx);
Mfs_Csio_DisableFunc(&CSIO0, CsioRx);
```

4 I²C

I²C インターフェース (I²C 通信制御インターフェース) は I²C バスをサポートし、I²C バス上のマスター／スレーブ デバイスとして動作します。

SMR レジスタの MD ビットを b'100 にセットした時、I²C モードは設定されます。

bit7	bit6	bit5	説明
0	0	0	動作モード 0 (非同期ノーマル モード)
0	0	1	動作モード 1 (非同期マルチプロセッサ モード)
0	1	0	動作モード 2 (クロック同期モード)
0	1	1	動作モード 3 (LIN 通信モード)
1	0	0	動作モード 4 (I²C モード)
上記以外			設定禁止

4.1 特長

- マスター／スレーブ機能
- 15 ビット ボーレート選択
- 8 ビット データ長
- バス アービトレーション機能
- スレーブ モードでの送信方向検出機能

- 反復スタート条件の生成と検出機能
- バス エラー検出機能
- スレーブでの 7 ビット アドレス指定
- データ送信またはバス エラー時に割り込み生成可能
- シリアル クロック／シリアル データ入力のため、バス クロックの 2～32 クロックでノイズの除去 [1]
- DMA 転送のサポート
- 割り込み要求
  - 受信完了、オーバーラン エラーの要因による受信割り込み要求
  - 送信データが空、送信バス アイドルの要因による送信割り込み要求
  - 送信 FIFO が空の要因による送信 FIFO 割り込み要求
  - データの送信／受信の完了、バス エラーおよび NACK 検出の要因によるステータス割り込み要求
  - ストップ条件と反復スタート条件の検出
- 送信／受信 FIFO 搭載 [2]

[1] FM ファミリ製品の一部だけがノイズ フィルターを持っています。ご使用製品のデータシートをご参照ください。

[2] FIFO 容量は製品種類によって異なります。ご使用製品のデータシートをご参照ください。

4.2 プロトコル

図 14 に示すとおり I²C フレームは常にスタート条件、7 ビットのスレーブ アドレス+1 ビットの R/W、データ、およびストップ条件で構成されています。

I²C マスターは I²C スレーブにデータを送信すると、R/W ビットを 0 にセットします。スタート条件信号と最初のバイトを送信した後、マスターはスレーブから ACK ビットを受信します。マスターはスレーブにデータを送信し続け、正常な状態では各バイトの最後にスレーブから ACK ビットを受信します。I²C マスターはすべてのデータを送信した後、データ転送が完了したことを示すためにスレーブにストップ条件信号を送信します。

I²C マスターは I²C スレーブからデータを受信すると、R/W ビットを 1 にセットします。スタート条件信号と最初のバイトを送信した後、マスターはスレーブから ACK ビットを受信します。マスターは I²C スレーブからデータを読み出し、各バイトを受信した後、スレーブに 1 ビットの ACK を送信する必要があります。マスターは最後のデータを受信した後、スレーブに 1 ビット NACK を送信する必要があります。これはマスターが受信したい最後のデータであることをスレーブに通知するためです。その後、マスターは I²C データ受信処理を完了するためにストップ条件信号を送信します。

図 14. I²C データ フォーマット

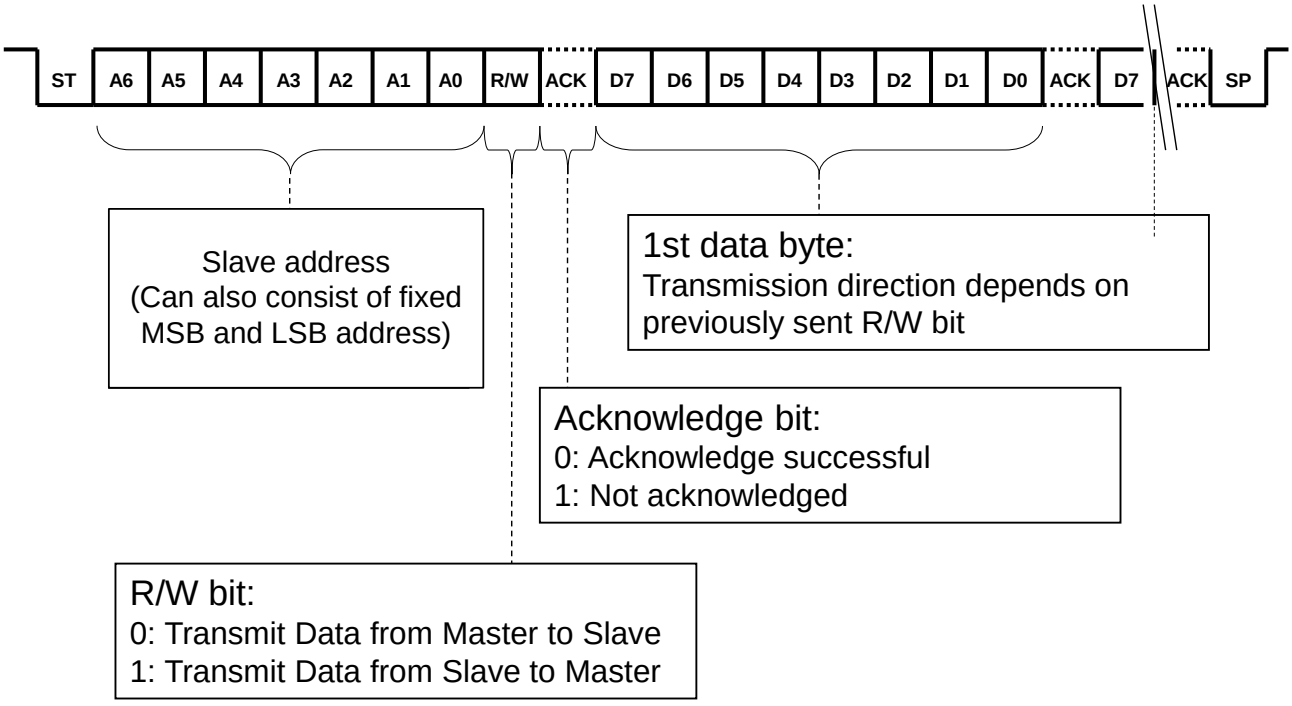


図 15 は I²C のスタートとストップ条件のタイミング図です。

スタート条件

- SCL (SCK) ラインが HIGH で、SDA (SOT) ライン上の立ち下がりエッジは I²C フレームの開始を示します。
- FM ファミリ MCU の I²C モジュールでは、MSS = 0 および ACT = 0 の時、MSS を 1 に設定すると I²C のスタート条件を生成します。次に、ACT ビットも自動的に 1 に設定され、マスター モード動作に入ったことを示します。

ストップ条件

- SCL (SCK) ラインが HIGH であり、SDA (SOT) ライン上の立ち上がりエッジになっている場合、I²C フレームの停止を示します。
- FM ファミリ MCU の I²C モジュールでは、MSS = 1 および ACT = 1 の時、MSS を 0 に設定すると I²C のストップ条件を生成します。次に、ACT ビットも自動的に 0 にセットされ、ストップ モードに入ったことを示します。

図 15. I²C スタートおよびストップ条件のタイミング図

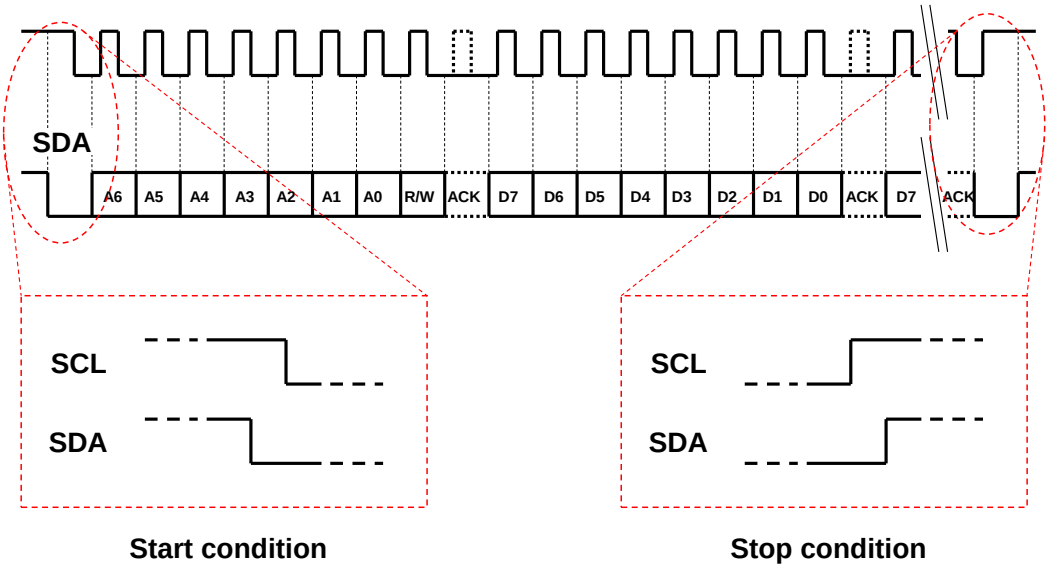


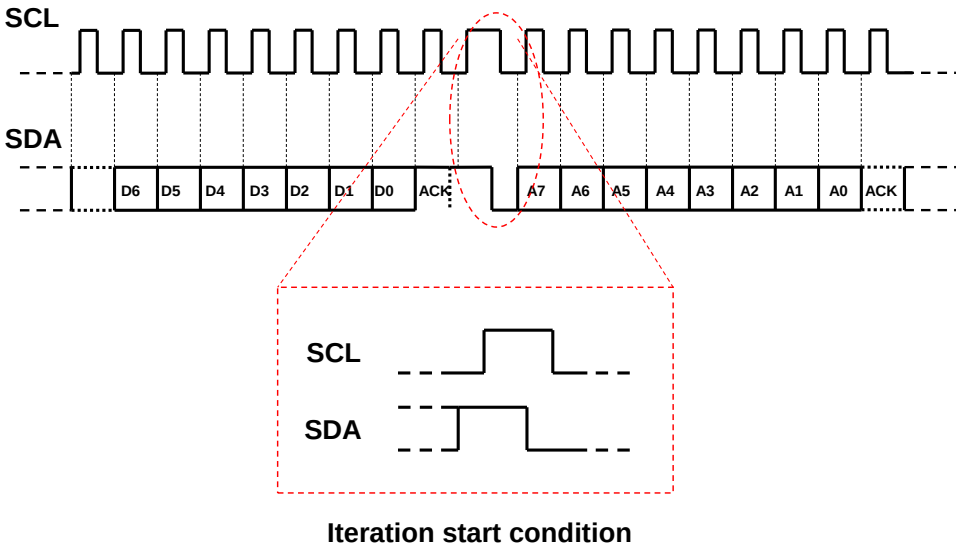
図 16 は I²C の反復 (繰り返し) スタート条件のタイミング図です。

反復 (繰り返し) スタート条件:

I²C 通信が開始されたとき、マスターが再びスタート条件を生成することは「反復スタート条件」または「繰り返しスタート条件」と呼ばれます。I²C の EEPROM からデータを読み出す時、この信号を使用できます。

FM ファミリ MCU の I²C モジュールでは、MSS = 1 および ACT = 1 の時、MSS を 1 に設定すると反復スタート条件を生成します。

図 16. I²C 反復スタート条件のタイミング図



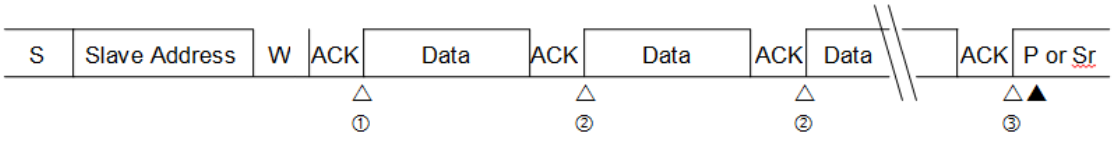
以下は IBCR レジスタの MSS と ACT ビットの説明です:

MSS Bit	ACT Bit	状態
0	0	アイドル
0	1	スレーブ アドレスが一致している、または ACK は予約アドレスに応答し、スレーブ モードが動作中
1	0	マスター モードの動作は待機中
1	1	マスター モードの動作中

4.3 データ転送タイミング

図 17 は FIFO が無効の場合、I²C 経由でいくつかのデータを書き込む際のタイミング図です。

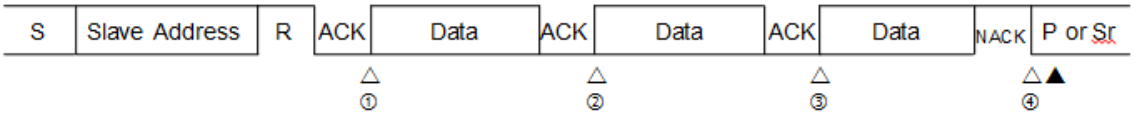
図 17. I²C によるデータ転送図



S: Start condition
W: Data direction bit (write direction)
P: Stop condition
Sr: Iteration start condition
△: Interrupt by INTE="1"
▲: Interrupt by CNDE="1"
① An interrupt occurs when the slave address is sent, the direction bit is sent, and an ACK is received.
- The send data is written in the TDR register, and the INT bit is set to "0".
② An interrupt occurs when a single byte is sent and an ACK is received.
- The send data is written in the TDR register, and the INT bit is set to "0".
③ An interrupt occurs when a single byte is sent and an ACK is received.
- MSS bit is set to "0", or MSS and SCC bits are set to "1".
*) If an interrupt flag (INT) is set, the TDRE bit is set to "1".

図 18 は FIFO が無効の場合、I²C 経由でいくつかのデータを読み出す際のタイミング図です。

図 18. I²C によるデータ受信図



△: Interrupt by INTE="1"
▲: Interrupt by CNDE="1"
① An interrupt occurs when the slave address is sent, the direction bit is sent, and an ACK is received.
- If the INT bit is set to "0", the interrupt flag is cleared to "0".
② An interrupt occurs when a single byte is received and an ACK is sent.
- After the received data has been read, the INT bit is set to "0".
③ An interrupt occurs when a single byte is received and an ACK is sent.
- After the received data has been read, both ACKE and INT bits are set to "0".
④ An interrupt occurs when a single byte is received and an ACK is sent.
- MSS bit is set to "0", or both MSS and SCC bits are set to "1".
*) If an interrupt flag (INT) is set, the TDRE bit is set to "1".

4.4 ロー レベルの API

以下は、PDL の `mfs.c/h` ファイルにある I²C ドライバーの API です。

- Mfs_I2c_Init()
- Mfs_I2c_DeInit()
- Mfs_I2c_EnableIrq()
- MfI2c_DisableIrq()
- Mfs_I2c_GenerateStart()
- Mfs_I2c_GenerateRestart()
- Mfs_I2c_GenerateStop()
- Mfs_I2c_SetBaudRate()
- Mfs_I2c_SendData()
- Mfs_I2c_ReceiveData()
- Mfs_I2c_ConfigAck()
- Mfs_I2c_GetAck()
- Mfs_I2c_GetStatus()
- Mfs_I2c_ClrStatus()
- Mfs_I2c_GetDataDir()
- Mfs_I2c_ResetFifo()
- Mfs_I2c_SetFifoCount()

- Mfs_I2c_GetFifoCount()

`Mfs_I2c_Init()` は`#stc_mfs_i2c_config_t` 型の `pstcConfig` パラメーターがある I²C モードの MFS インスタンスを初期化するために使用されます。この関数は基本的な I²C 環境のみを設定します。`Mfs_I2c_DeInit()` はすべての MFS の I²C 関連レジスタをリセットするために使用されます。

`Mfs_I2c_EnableIrq()` は `#en_i2c_irq_sel_t` 列挙型で選択された I²C 割り込みソースを有効にし、`Mfs_I2c_DisableIrq()` は`#en_i2c_irq_sel_t` 列挙型で選択された I²C 割り込みソースを無効にします。

`Mfs_I2c_SetBaudRate()` は、I²C が初期化された後、I²C ボーレートを変更できるようにします。

`Mfs_I2c_SendData()` は I²C 送信バッファに 1 バイト データを書き込み、`Mfs_I2c_ReceiveData()` は I²C 受信バッファから 1 バイト データを読み出します。

`Mfs_I2c_GenerateStart()` は I²C スタート信号を生成します。`Mfs_I2c_GenerateRestart()` は I²C リスタート信号を生成します。`Mfs_I2c_GenerateStop()` は I²C ストップ信号を生成します。

`Mfs_I2c_ConfigAck()` はデータを受信している時に ACK 信号を設定します。`Mfs_I2c_GetAck()` は ACK を受信した後に ACK 信号の状態を取得します。

`Mfs_I2c_GetStatus()` は `Mfs_I2c_GetStatus#enStatus` にて選択した状態を取得し、`Mfs_I2c_ClrStatus()` は選択された I²C 状態をクリアします。いくつかの状態はハードウェアだけによって自動的にクリアされます。

`Mfs_I2c_GetDataDir()` はスレーブ モードで I²C のデータ方向を取得します。

`Mfs_I2c_ResetFifo()` は I²C ハードウェア FIFO をリセットします。`Mfs_Csio_SetFifoCount()` は、I²C が初期化された後、FIFO のサイズを変更できるようにします。

`Mfs_I2c_GetFifoCount()` は現時点の FIFO のデータ カウントを取得します。

4.5 サンプル コード

ロー レベルの API ドライバーに基づいて、この例は割り込みフラグのポーリング方法を使用し、I²C を介してデータ転送する方法を示します。これは I²C チャネル 0 を使用し、10 バイトを転送してから 10 バイトを受信します。

I²C を使用する前に、次のように I²C のピン機能を設定します。

```
/* Initialize I2C function I/O */
    SetPinFunc_SOT0_0();
    SetPinFunc_SCK0_0();
```

次に、I²C 環境構造を設定し、I²C チャネルを初期化します。

- モード: マスター モード
- ボーレート: 100kbps

```
    stc_mfs_i2c_config_t stcI2c0Config;

    /* Configure I2C structure */
    stcI2c0Config.enMsMode = I2cMaster;
    stcI2c0Config.u32BaudRate = 100000u;
    stcI2c0Config.bWaitSelection = FALSE;
    stcI2c0Config.bDmaEnable = FALSE;
    stcI2c0Config.pstcFifoConfig = NULL;

    if (Ok != Mfs_I2c_Init(&I2C0, &stcI2c0Config))
    {
        While(1);
    }
```

次のコードでは、I²C を介して 10 バイト データを送信します。デバイス アドレスは 0x50 であることを前提にします。

```
/*****************************************************************/
/*      Generate start condition                                 */
/*****************************************************************/
/* Prepare I2C device address */
Mfs_I2c_SendData(&I2C0, (0x50<<1));

/* Generate I2C start signal */
if(Ok != Mfs_I2c_GenerateStart(&I2C0))
{
    while(1); /* Timeout or other error */
}

while(1)
{
    if(TRUE != Mfs_I2c_GetStatus(&I2C0, I2cRxTxIrq))
    {
        break;
    }
}

if(I2cNAck == Mfs_I2c_GetAck((&I2C0))
{
    while(1);   /* NACK */
}

if(TRUE == Mfs_I2c_GetStatus((&I2C0, I2cBusErr))
{
    while(1); /* Bus error occurs? */
}

/*****************************************************************/
/*      Send data                                                */
/*****************************************************************/

for(uint8_t i=0;i<10;i++)
{
    /* Transmit the data */
    Mfs_I2c_SendData(&I2C0, pTxData[i]);
    Mfs_I2c_ClrStatus(&I2C0, I2cRxTxIrq);
    /* Wait for end of transmission */
    while(1)
    {
        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cRxTxIrq))
        {
            break;
        }
    }

    while(1)
    {
        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cTxEmpty))
        {
            break;
        }
    }

    if(I2cNAck == Mfs_I2c_GetAck(&I2C0))
    {
```

```
            while(1);   /* NACK */
        }

        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cBusErr))
        {
            while(1); /* Bus error occurs? */
        }
    }
    /*****************************************************************/
    /*      Generate stop condition                                  */
    /*****************************************************************/
    /* Generate I2C start signal */
    if(Ok != Mfs_I2c_GenerateStop(&I2C0))
    {
        while(1); /* Timeout or other error */
    }
    /* Clear Stop condition flag */
    while(1)
    {
        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cStopDetect))
        {
            break;
        }
    }
    Mfs_I2c_ClrStatus(&I2C0, I2cStopDetect);
    Mfs_I2c_ClrStatus(&I2C0, I2cRxTxIrq);
```

次のコードでは、I²C を介して 10 バイト データを読み出します。デバイス アドレスは 0x50 であることを前提にします。

```
    /*****************************************************************/
    /*      Generate start condition                                 */
    /*****************************************************************/
    /* Prepare I2C device address */
    Mfs_I2c_SendData(&I2C0, (0x50<<1));

    /* Generate I2C start signal */
    if(Ok != Mfs_I2c_GenerateStart(&I2C0))
    {
        while(1); /* Timeout or other error */
    }

    while(1)
    {
        if(TRUE != Mfs_I2c_GetStatus(&I2C0, I2cRxTxIrq))
        {
            break;
        }
    }

    if(I2cNAck == Mfs_I2c_GetAck((&I2C0))
    {
        while(1);   /* NACK */
    }

    if(TRUE == Mfs_I2c_GetStatus((&I2C0, I2cBusErr))
    {
        while(1); /* Bus error occurs? */
    }
```

```
/*****************************************************************/
/*      Read data                                                */
/*****************************************************************/
uint8_t i;

/* Clear interrupt flag generated by device address send */
Mfs_I2c_ClrStatus(&I2C0, I2cRxTxIrq);

if(I2cNAck == Mfs_I2c_GetAck(&I2C0))
{
    while(1);   /* NACK */
}

while(i < u8Size)
{
    /* Wait for the receive data */
    while(1)
    {
        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cRxTxIrq))
        {
            break;
        }
    }

    if(i == u8Size-1)
    {
        Mfs_I2c_ConfigAck(&I2C0, I2cNAck); /* Last byte send a NACK */
    }
    else
    {
        Mfs_I2c_ConfigAck(&I2C0, I2cAck);
    }

    /* Clear interrupt flag and receive data to RX buffer */
    Mfs_I2c_ClrStatus(&I2C0, I2cRxTxIrq);

    /* Wait for the receive data */
    while(1)
    {
        if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cRxFull))
        {
            break;
        }
    }

    if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cBusErr))
    {
        while(1);   /* Bus error occurs? */
    }

    if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cOverrunError))
    {
        while(1;  /* Overrun error occurs? */
    }

    pRxData[i++] = Mfs_I2c_ReceiveData(&I2C0);
}
/*****************************************************************/
/*      Generate stop condition                                  */
/*****************************************************************/
```

```
        /* Generate I2C start signal */
        if(Ok != Mfs_I2c_GenerateStop(&I2C0))
        {
            while(1); /* Timeout or other error */
        }
        /* Clear Stop condition flag */
        while(1)
        {
            if(TRUE == Mfs_I2c_GetStatus(&I2C0, I2cStopDetect))
            {
                break;
            }
        }
        Mfs_I2c_ClrStatus(&I2C0, I2cStopDetect);
        Mfs_I2c_ClrStatus(&I2C0, I2cRxTxIrq);
```

5 LIN

LIN インターフェース (LIN 通信制御インターフェース Ver. 2.1) は、LIN バスに準拠した機能をサポートします。

SMR レジスタの MD ビットを b'011 にセットした時、LIN モードは設定されます。

bit7	bit6	bit5	説明
0	0	0	動作モード 0 (非同期ノーマル モード)
0	0	1	動作モード 1 (非同期マルチプロセッサ モード)
0	1	0	動作モード 2 (クロック同期式モード)
0	1	1	動作モード 3 (LIN 通信モード)
1	0	0	動作モード 4 (I^2C モード)
上記以外			設定禁止

5.1 特長

- LIN プロトコル Rev 2.1 をサポート
- マスター／スレーブ デバイス動作
- 全二重動作
- LIN break field の生成 (13～16 ビットの可変ビット長)[1]
- LIN break デリミタの生成 (1～4 ビットの可変データ長)[1]
- LIN break field 検出 [1]
- スタート／ストップ エッジの入力キャプチャによる LIN Sync field 検出 [1]
- 15 ビット ボーレート選択 [2]
- 8 ビット データ長
- 受信エラー検出
  - フレーミング エラー
  - オーバーラン エラー
- 割り込み要求

- 受信完了、フレーミング エラー、オーバーラン エラー、またはパリティ エラーの要因による割り込み要求
- 送信データが空、送信バス アイドルの要因による送信割り込み要求
- 送信 FIFO が空の要因による送信 FIFO 割り込み要求

- DMA 転送のサポート
- 送信／受信 FIFO 搭載 [3]

[1] 一部の製品のみが LIN モジュールを搭載しています。ご使用製品のデータシートをご参照ください。

[2] ボーレート ジェネレータは外部クロック供給できます。

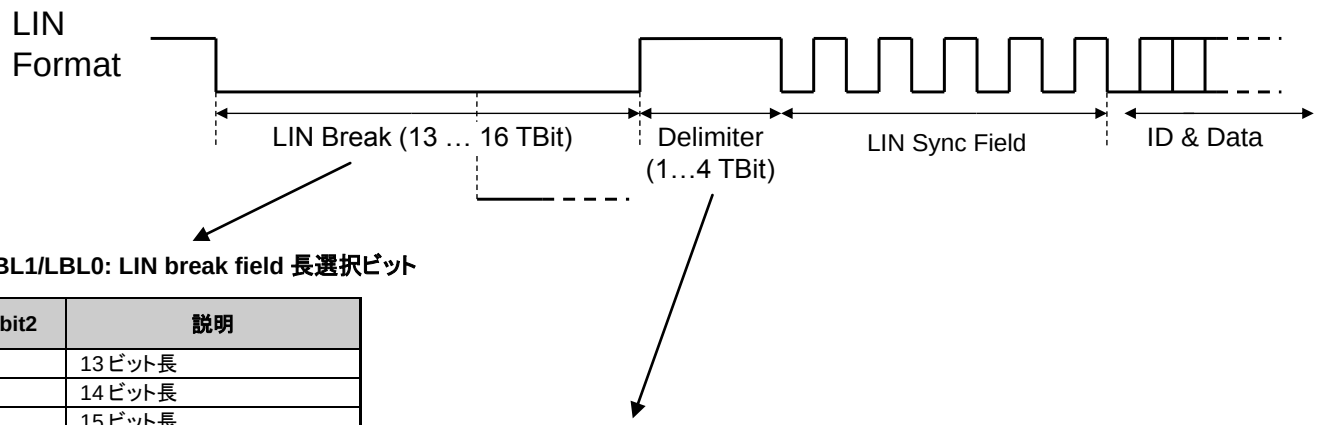
[3] FIFO 容量サイズは製品種類によって異なります。ご使用製品のデータシートをご参照ください。

5.2 データ フォーマット

図 19 に示すように、LIN フレームは LIN break、デリミタ、LIN Sync field、および ID とデータ field で構成されます。

LIN マスター モードでは、LIN break 長は LBL0 と LBL1 ビットでセットすることができ、LIN デリミタ長は ESCR レジスタの DEL0 と DEL1 ビットでセットすることができます。

図 19. LIN データ フォーマット



[bit3:2] LBL1/LBL0: LIN break field 長選択ビット

bit3	bit2	説明
0	0	13 ビット長
0	1	14 ビット長
1	0	15 ビット長
1	1	16 ビット長

[bit1:0] DEL1/DEL0: LIN break デリミタ長選択ビット

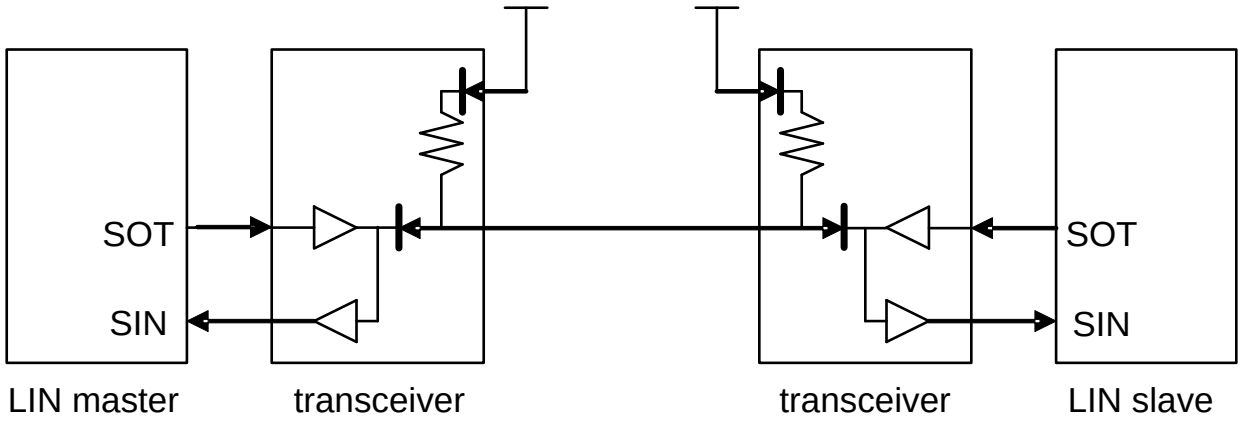
bit1	bit0	説明
0	0	1 ビット長
0	1	2 ビット長
1	0	3 ビット長
1	1	4 ビット長

LIN スレーブ モードでは、LIN break は 11 ビット続く LOW レベルの後に検出することができます。ボーレートは内部入力キャプチャ ユニット (ICU) と LIN Sync field のキャプチャにより調整することができます。

5.3 通信システム

図 20 は LIN トランシーバーを介して 1 つの LIN マスターと 1 つの LIN スレーブからなる通信システムを示します。

図 20. LIN 通信システム



5.4 動作タイミング

図 21 は LIN マスター モードでのデータ送信のタイミング図です。

1. LIN break field (LBR) ビットを 1 にセットすると、マスターにより送信される break が発生し、その後 0x55 (LIN Sync field) は TDR に書き込むことができます。これは LIN break の後に送信することができます。
2. LIN Sync field (0x55) の最初のビットは SOT ピンにシフトアウトされます。TDRE ビットが 1 にセットされ、TIE ビットが 1 にセットされている場合、送信割り込みが発生します。その後、ID バイトは TDR に書き込むことができます。
3. LIN マスターは送信するものは何でも受け取ることができます。この時点で 0x55 を受信し、元データとの比較はデータ送信が正常であるかどうかを確認することができます。
4. ID の最初のビットが送信されると、TDRE ビットが 1 になり、TIE ビットが 1 にセットされている場合、送信割り込みが発生します。その後、最初のデータを送信することができます。

データ送信プロセスは ID field と同じです。

図 21. LIN マスター データ転送タイミング図

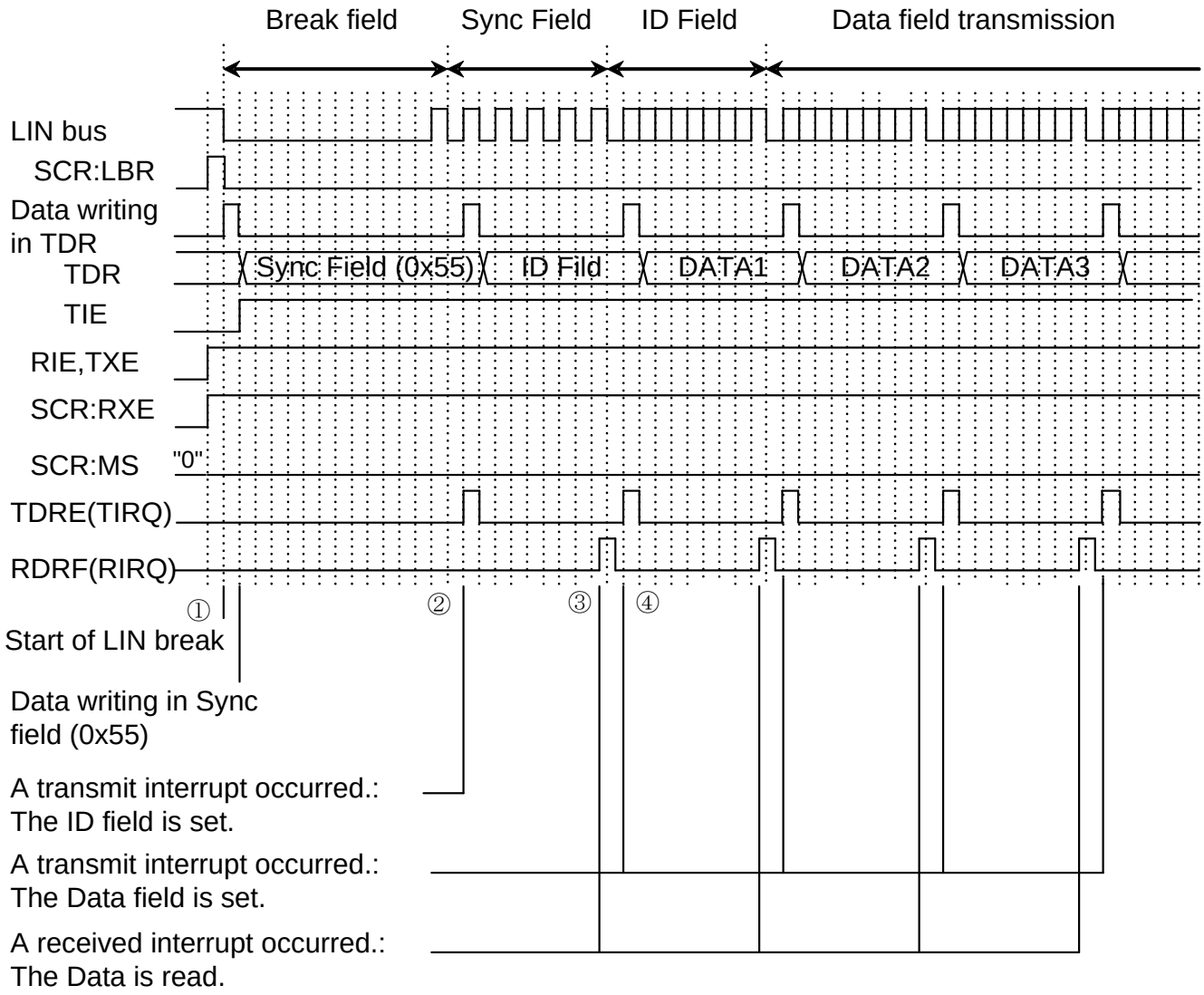


図 22 は LIN スレーブ モードでのデータ送信タイミング図です。

1. LIN break は 11 ビット続く LOW レベルの後に検出することができます。LIN break 割り込み許可 (LBIE) ビットが 1 にセットされていると、LIN break 割り込みが発生します。その後、SYNC 信号を接続する ICU は初期化され、有効になります。特定の LIN チャネルに対応する ICU チャンネルを確認するため、ペリフェラル マニュアルの「GPIO」の章をご参照ください。
2. Sync フィールドの最初の立ち下がりエッジを検出すると、ICU 割り込みがトリガされ、ICU データ レジスタ値を *a* 変数に格納することができます。
3. ICU 割り込みが再度発生すると、ICU データ レジスタ値を *b* 変数に格納することができます。

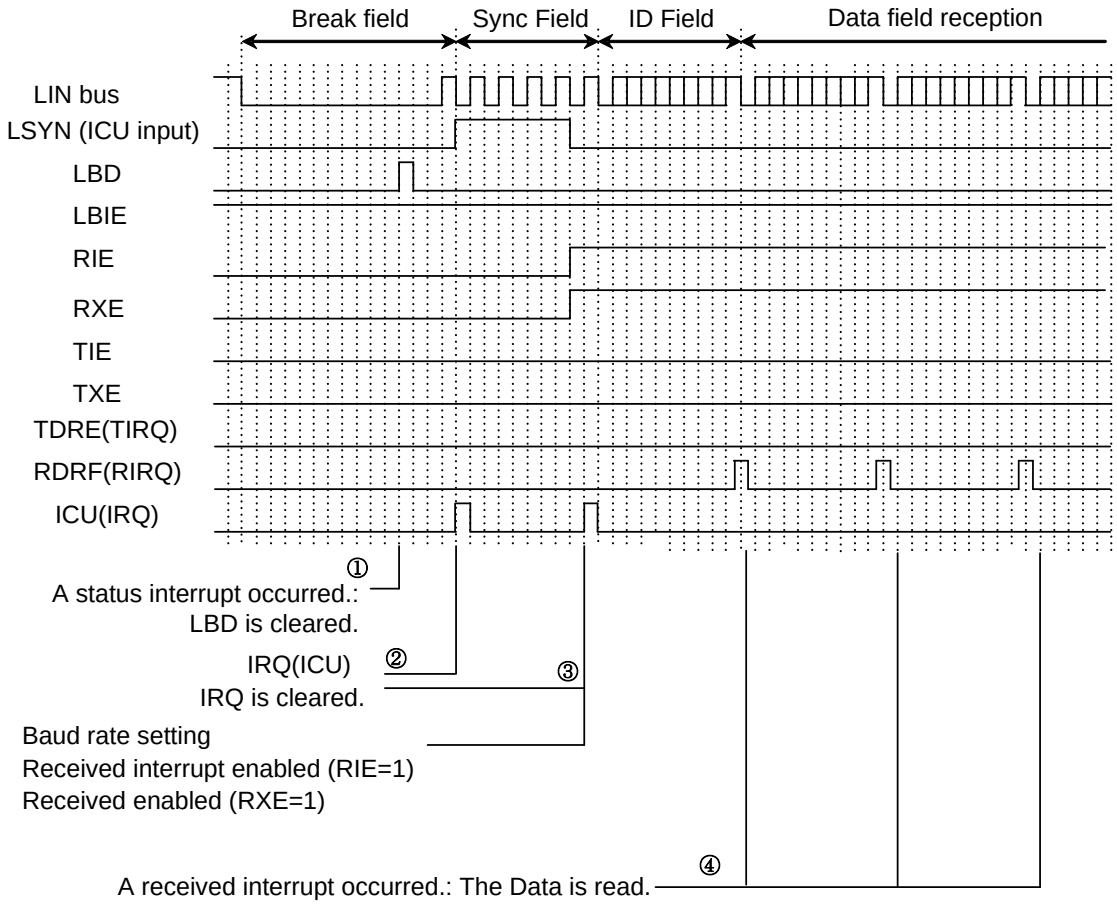
   LIN マスターの正確なボーレートは以下の式で計算することができます (FRT がオーバー フローせず、MFS と FRT のために同じクロックである場合)。新しいボーレートを BGR にセットする必要があり、ICU 機能を無効にする必要があります。その後、受信を有効にすることができ (RXE = 1) 、受信割り込みを有効にすることができます (RIE = 1)。

```
BGR value = (b - a)/8 - 1
a:    The ICU data register value after the first interrupt
b:    The ICU data register value after the second interrupt
```

4. 受信データ レジスタが満杯になり (RDRF = 1)、受信割り込みが有効になっている場合、受信割り込みが発生し、ID フィールドは RDR から読み出すことができます。

   データ受信プロセスは ID field と同じです。

図 22. LIN スレーブ データ転送タイミング図



5.5 ロー レベルの API

以下は、PDL の `mfs.c/h` ファイルにある LIN ドライバーの API です。

- Mfs_Lin_Init()
- Mfs_Lin_DeInit()
- Mfs_Lin_EnableIrq()
- Mfs_Lin_DisableIrq()
- MfsLinIrqHandlerStatus()
- Mfs_Lin_SetBaudRate()
- Mfs_Lin_GenerateBreakField()
- Mfs_Lin_EnableFunc()
- Mfs_Lin_DisableFunc()
- Mfs_Lin_GetStatus()

- Mfs_Lin_ClrStatus()
- Mfs_Lin_SendData()
- Mfs_Lin_ReceiveData()
- Mfs_Lin_ResetFifo()
- Mfs_Lin_SetFifoCount()
- Mfs_Lin_GetFifoCount()

`Mfs_Lin_Init()` は専用の LIN 設定`#stc_mfs_lin_config_t` で LIN モードの MFS インスタンスを初期化するために使用されます。この関数は基本的な LIN 環境のみを設定します。`Mfs_Lin_DeInit()` はすべての MFS の LIN 関連レジスタをリセットするために使用します。

`Mfs_Lin_EnableIrq()` は`#en_lin_irq_sel_t` 割り込みタイプで選択された LIN 割り込みソースを有効にし、`Mfs_Lin_DisableIrq()` は`#en_lin_irq_sel_t` 割り込みタイプで選択された LIN 割り込みソースを無効にします。

`Mfs_Lin_SetBaudRate()` は、LIN が初期化された後、LIN ボーレートを変更できるようにします。

`Mfs_Lin_GenerateBreakField()` は `LIN break field` を生成し、これは自身によっても検出できます。

`Mfs_Lin_EnableFunc()` は `Mfs_Lin_EnableFunc#enFunc` パラメーターで選択された LIN 機能を有効にし、`Mfs_Lin_DisableFunc()` は LIN 機能を無効にします。

`Mfs_Lin_GetStatus()` は`Mfs_Lin_GetStatus#enStatus` で選択された状態を取得し、`Mfs_Lin_ClrStatus()` は選択された LIN 状態をクリアします。いくつかの状態はハードウェアだけによって自動的にクリアされます。

`Mfs_Lin_SendData()` は LIN 送信バッファに 1 バイト データを書き込み、`Mfs_Lin_ReceiveData()` は LIN 受信バッファから 1 バイト データを読み出します。

`Mfs_Lin_ResetFifo()` は LIN のハードウェア FIFO をリセットします。

`Mfs_Lin_SetFifoCount()` は、LIN が初期化された後、FIFO サイズを変更できるようにします。

`Mfs_Lin_GetFifoCount()` は現時点の FIFO のデータ カウントを取得します。

5.6 サンプル コード

ロー レベルの API ドライバーに基づいて、この例は割り込みフラグのポーリング方法を使用し、LIN を介してデータ転送する方法を示します。これは LIN チャネル 0 を LIN マスターとして使用し、10 バイトを転送します。

LIN を使用する前に、次のように LIN のピン機能を設定します。

```
/* Initialize LIN function I/O */
SetPinFunc_SOT0_0();
SetPinFunc_SIN0_0();
```

次に、LIN 環境構造を設定し、LIN チャネルを初期化します。

- モード: マスター モード
- ボーレート: 9600bps
- ブレーク長: 13 ビット
- デリミタ長: 1 ビット
- ストップ ビット長: 1 ビット

```
stc_mfs_lin_config_t stcLinConfig;
uint32_t u32i;
uint8_t u8RdData;

/* Initialize LIN  */
stcLinConfig.enMsMode = LinMasterMode;
stcLinConfig.u32BaudRate = 9600;
stcLinConfig.enBreakLength = LinBreakLength13;
stcLinConfig.enDelimiterLength = LinDelimiterLength1;
stcLinConfig.enStopBits = LinOneStopBit;
stcLinConfig.pstcFifoConfig = NULL;

if (Ok != Mfs_Lin_Init(&LIN0, &stcLinConfig))
{
    while(1); /* Initialization error */
}
```

以下のコードで ID フィールドが 0x3A に設定された 10 バイトを送信します。

```
/*****************************************************************/
/*   Send LIN break                                              */
/*****************************************************************/
/* Generate LIN break field */
Mfs_Lin_GenerateBreakField(&LIN0);
while(Mfs_Lin_GetStatus(&LIN0, LinBreakFlag) != TRUE);
Mfs_Lin_ClrStatus(&LIN0, LinBreakFlag);

/* Enable TX and RX function of LIN   */
Mfs_Lin_EnableFunc(&LIN0, LinTx);
Mfs_Lin_EnableFunc(&LIN0, LinRx);



/*****************************************************************/
/*   Send Sync filed                                             */
/*****************************************************************/
while(Mfs_Lin_GetStatus(&LIN0, LinTxEmpty) != TRUE); // Wait until TDR empty
Mfs_Lin_SendData(&LIN0, 0x55);
while(Mfs_Lin_GetStatus(&LIN0, LinRxFull) != TRUE);  // Wait until RDR full
u8RdData = Mfs_Lin_ReceiveData(&LIN0);
if(u8RdData != 0x55)
{
    while(1); /* Send data error */
}

/*****************************************************************/
/*   Send ID filed                                               */
/*****************************************************************/
while(Mfs_Lin_GetStatus(&LIN0, LinTxEmpty) != TRUE); // Wait until TDR empty
Mfs_Lin_SendData(&LIN0, 0x3A);
while(Mfs_Lin_GetStatus(&LIN0, LinRxFull) != TRUE);  // Wait until RDR full
u8RdData = Mfs_Lin_ReceiveData(&LIN0);
if(u8RdData != 0x3A)
{
    while(1); /* Send data error */
}
```

```
/*****************************************************************/
/*   Send Data filed                                             */
/*****************************************************************/
for(u32i=0; u32i<10; u32i++)
{
    while(Mfs_Lin_GetStatus(&LIN0, LinTxEmpty) != TRUE); // Wait until TDR empty
    Mfs_Lin_SendData(&LIN0, pData[u32i]);
    while(Mfs_Lin_GetStatus(&LIN0, LinRxFull) != TRUE); // Wait until RDR full
    u8RdData = Mfs_Lin_ReceiveData(&LIN0);
    if(u8RdData != pData[u32i])
    {
        While(1);
    }
}

while(Mfs_Lin_GetStatus(LinCh1, LinTxIdle) != TRUE); // Wait until TX bus idle
```

6 まとめ

MFS インターフェースは非常に柔軟で、シリアル通信の様々なタイプに適用することができます。

改訂履歴

文書名: AN99218 - FM MCU のマルチファンクション シリアル インターフェース

文書番号: 002-09821

版	ECN	変更者	発行日	変更内容
**	5026162	HIDH	12/03/2015	これは英語版 001-99218 Rev. **を翻訳した日本語版 002-09821 Rev. **です。
*A	5821521	AESATMP8	07/19/2017	ロゴと著作権を更新しました。

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店、および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

製品

ARM® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
タッチ センシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス	cypress.com/wireless

PSoC® ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

サイプレス開発者コミュニティ

フォーラム | WICED IOT Forums | Projects | ビデオ | ブログ | トレーニング | Components

テクニカルサポート

cypress.com/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709