FUJITSU

FM3 Family
32-BIT MICROCONTROLLER
MB9Axxx / MB9Bxxx Series

FM3 Family Drivers Software V01
MultiFunciton Serial Driver
ユーザーズマニュアル

FUJITSU

・本資料の記載内容は、予告なしに変更することがありますので、ご用命の際は営業部門にご確認ください。
・本資料に記載された動作概要や応用回路例は、半導体デバイスの標準的な動作や使い方を示したもので、実際に使用する機器での動作を保証するものではありません。したがいまして、これらを使用するにあたってはお客様の責任において機器の設計を行ってください。これらの使用に起因する損害などについては、当社はその責任を負いません。・本資料に記載された動作概要・回路図を含む技術情報は、当社もしくは第三者の特許権、著作権等の知的財産権やその他の権利の使用権または実施権の許諾を意味するものではありません。また、これらの使用について、第三者の知的財産権やその他の権利の実施ができることの保証を行うものではありません。したがって、これらの使用に起因する第三者の知的財産権やその他の権利の侵害について、当社はその責任を負いません。
・本資料に記載された製品は、通常の産業用、一般事務用、パーソナル用、家庭用などの一般的用途に使用されることを意図して設計・製造されています。極めて高度な安全性が要求され、仮に当該安全性が確保されない場合、社会的に重大な影響を与えかつ直接生命・身体に対する重大な危険性を伴う用途（原子力施設における核反応制御、航空機自動飛行制御、航空交通管制、大量輸送システムにおける運行制御、生命維持のための医療機器、兵器システムにおけるミサイル発射制御をいう）、ならびに極めて高い信頼性が要求される用途（海底中継器、宇宙衛星をいう）に使用されるよう設計・製造されたものではありません。したがって、これらの用途にご使用をお考えのお客様は、必ず事前に営業部門までご相談ください。ご相談なく使用されたことにより発生した損害などについては、責任を負いかねますのでご了承ください。・半導体デバイスはある確率で故障が発生します。当社半導体デバイスが故障しても、結果的に人身事故、火災事故、社会的な損害を生じさせないよう、お客様は、装置の冗長設計、延焼対策設計、過電流防止対策設計、誤動作防止設計などの安全設計をお願いします。
・本資料に記載された製品を輸出または提供する場合は、外国為替及び外国貿易法および米国輸出管理関連法規等の規制をご確認の上、必要な手続きをおとりください。
・本書に記載されている社名および製品名などの固有名詞は、各社の商標または登録商標です。

改 版 履 歴 票		
日付	版数	改版内容
2010/11/11	1.00	初版

（履歴票のページ：1/1）

はじめに

本書の目的と対象読者

本書は、ソフトウェア技術者の方を対象に FM3 Family Drivers Software V01 の MultiFunciton Serial Driver について解説したものです。

読者に必要な知識

本書は、ご使用のマイクロコントローラに内蔵されたマルチファンクションシリアルインタフェースの仕様を、読者が理解されていることを前提に説明しています。必要に応じてご使用のマイクロコントローラの仕様書を参照してください。

また、ANSI(American National Standards Institute)準拠の C 言語に関する知識が必要です。

マイクロコントローラの仕様に関する資料

マイクロコントローラの仕様については、**表 a** に示す資料を参照してください。

表 a マイクロコントローラ資料一覧

資料名	本書内での呼称
FM3 ファミリ MB9Axxx / MB9Bxxx Series ペリフェラルマニュアル	マイクロコントローラの仕様書

本書の表記について

本書の表記は、次の規則に従っています。ただし、ソース記述は除きます。

太字　関数名、構造体名および定数ラベル(C 言語マクロ名)

斜体　パラメータ名(仮引数名)

ソース記述は、等間隔フォント(Courier New)でテキストと区別します。

目次

1 概要

本章では、MultiFunciton Serial Driver の概要について説明します。

1.1 マルチファンクションシリアルインタフェースとは

マルチファンクションシリアルインタフェースは、動作モードとして以下のインタフェースモードをもち、チャネルごとにインタフェースモードを設定し、データの送受信を行うことが可能です。

- UART0（非同期ノーマルシリアルインタフェース）
- UART1（非同期マルチプロセッサシリアルインタフェース）
- CSIO（クロック同期式シリアルインタフェース）（SPI に対応可能）
- LIN（LIN インタフェース）
- I²C（I²Cバスインタフェース）

1.2 MultiFunciton Serial Driverの構成

MultiFunciton Serial Driver は、マイクロコントローラに内蔵されたマルチファンクションシリアルインタフェースを制御するデバイスドライバです。MultiFunction Serial Driver は、以下の 4 つのドライバで構成されています。使用するインタフェースモードにより、チャネルごとに４つのドライバから１つを選択して使用してください。

- UART Driver
- I²C Driver
- SIO Driver
- SPI Driver

ドライバとインタフェースモードとの関係を**表 1-1**に示します。

表 1-1 ドライバとインタフェースモードの関係

ドライバ	インタフェースモード
UART Driver	UART0
I²C Driver	I²C
SIO Driver	CSIO
SPI Driver	

1.3 制限事項

制限事項については、README_J.txt を参照してください。

2 UART Driver

本章では UART Driver について説明します。

2.1 概要

UART Driver は 8 チャネル（ch0 から ch7）に対応しています。ch4 から ch7 はマクロ定義されたバイト数（0 バイトから 16 バイト）のハードウェア FIFO を使用します。ch0 から ch3 はハードウェア FIFO がありません。DMA 転送を使用した送受信には、全チャネル対応していません。

2.1.1 ファイル構成

UART Driverのファイル構成を**表 2-1**に示します。

表 2-1 UART Driver ファイル構成

ファイル名	説明
UartDev.h	UART Driver を使用する場合は本ファイルをインクルードしてください。本ファイルには構造体定義、初期設定用マクロ定義を記述しています。
UartDev_FM3.h	UartDev_FM3.c からのみインクルードされます。本ファイルにはマイクロコントローラに依存した定義を記述しています。
UartDev_FM3.c	UART Driver を使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルには API 関数と内部関数を記述しています。
MfsDev_FM3.h	UartDev_FM3.c からインクルードされます。本ファイルにはドライバ内部で使用するチャネル管理用の関数のプロトタイプ宣言を記述しています。
MfsDev_FM3.c	UART Driver を使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルにはチャネル管理用の関数を記述しています。

2.2 マクロ定義

UART Driver のマクロ定義について説明します。

2.2.1 FIFO バイト数

FIFOバイト数の指定用マクロを表 2-2に示します。これらの値によりチャネルのFIFOバイト数を変更できます。デフォルト値以外を使用する場合のみ変更してください。

表 2-2 FIFO バイト数の定義（UartDev_FM3.h）

マクロ名	説明	デフォルト値
UartDev_CH04_FIFO_DEPTH	Ch4 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
UartDev_CH05_FIFO_DEPTH	Ch5 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
UartDev_CH06_FIFO_DEPTH	Ch6 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
UartDev_CH07_FIFO_DEPTH	Ch7 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16

2.3 データ型

UART Driver では、標準 C ライブラリの stdint.h で定義されたデータ型を使用しています。stdint.h については、ご使用のコンパイラのマニュアルを参照してください。

2.4 構造体

UART Driverで使用する構造体について説明します。ドライバ構成情報構造体は、ドライバの初期化時に値を設定してください。初期化の手順については、「**2.5使用手順**」をご覧ください。

2.4.1 UartDev_CFG 《ドライバ構成情報》

【宣言】

```
typedef struct {
  int32_t BaudRate;
  int32_t DataBits:5;
  int32_t StopBits:3;
  int32_t Parity:3;
  int32_t BitOrder:2;
  int32_t Reserved:19;
  int32_t BufSize;
} UartDev_CFG;
```

【メンバ】

BaudRate	ボーレート（bps）	
	ご使用のシステムで有効な値（デフォルト：115200）	
DataBits	データビット	
	UartDev_DATABITS_6	6 データビット
	UartDev_DATABITS_7	7 データビット
	UartDev_DATABITS_8	8 データビット（デフォルト）
StopBits	ストップビット	
	UartDev_STOPBITS_1	1 ストップビット（デフォルト）
	UartDev_STOPBITS_2	2 ストップビット
Parity	パリティ	
	UartDev_PARITY_NONE	パリティなし（デフォルト）
	UartDev_PARITY_ODD	偶数パリティ
	UartDev_PARITY_EVEN	奇数パリティ
BitOrder	ビットオーダー	
	UartDev_BITORDER_MSB	MSB ファースト（デフォルト）
	UartDev_BITORDER_LSB	LSB ファースト

BufSize	送受信バッファサイズ（バイト） 2 のべき乗でご使用のシステムで有効な値（デフォルト：128）
Reserved	予約領域（値を設定しないでください）

2.4.2 UartDev_IOB 《I/O ブロック》

【宣言】

```
typedef struct {
  UartDev_CFG  Cfg;
  int32_t  (*Init)(void);
  int32_t  (*UnInit)(void);
  int32_t  (*BufTx)(void *pData, int32_t *pSize, uint32_t flags);
  int32_t  (*BufRx)(void *pData, int32_t *pSize, uint32_t flags);
  int32_t  (*BufFlush)(void);
} UartDev_IOB;
```

【メンバ】

Cfg	**UartDev_CFG 構造体**
Init	初期化関数の関数ポインタ
UnInit	初期化解除関数の関数ポインタ
BufTx	データ送信関数の関数ポインタ
BufRx	データ受信関数の関数ポインタ
BufFlush	バッファフラッシュ関数の関数ポインタ

2.5 使用手順

UART Driver の使用手順を説明します。

2.5.1 初期化から初期化解除までの手順

初期化から初期化解除までの手順を**図 2-1**に示します。

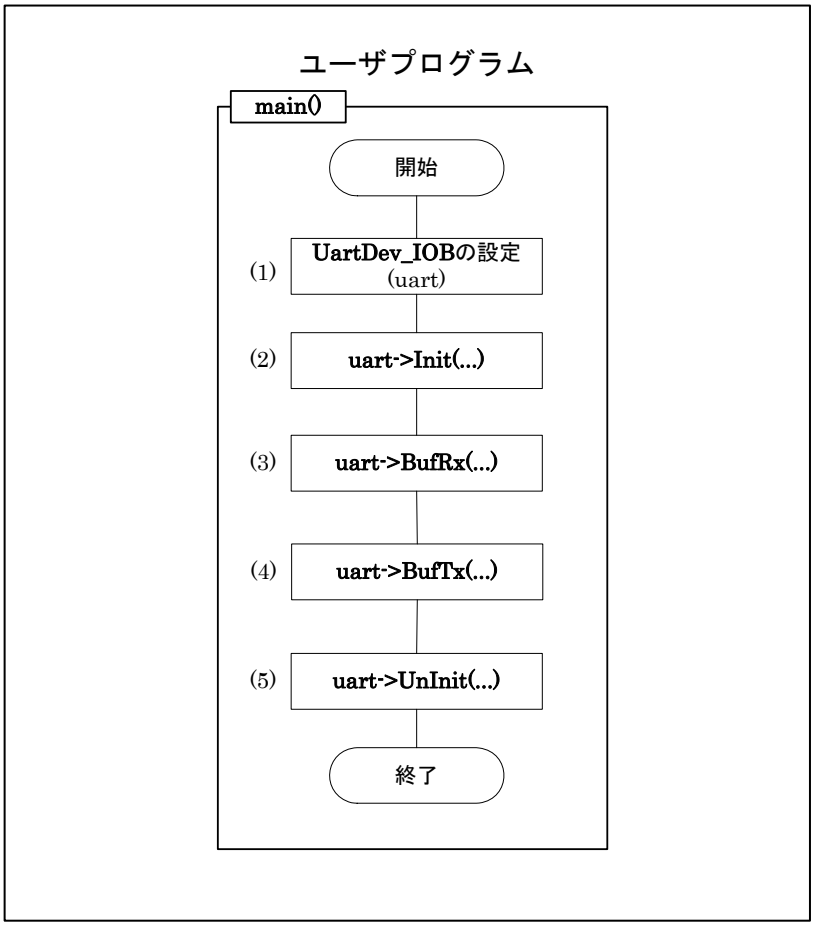

図 2-1 UART Driver の初期化から初期化解除までの手順

(1) UART Driver は、ドライバ内でチャネル数分の **UartDev_IOB** 構造体配列（UartDev[ ]）を定義し、デフォルト値を設定します。したがって、UART Driver を初期化する前に、以下の処理を行ってください。

(1-1) UartDev[ ]を外部参照(extern 宣言)してください。

(1-2) チャネル番号を要素番号に指定して UartDev[ ]を参照してください。

(1-3) **UartDev_IOB** 構造体のメンバ Cfg のデフォルト値以外を使用する場合は値を変更してください。

(2) **uart->Init** 関数を呼び出し、UART Driver の初期化を行ってください。

(3) **uart->BufRx** 関数を呼び出し、データを受信してください。

(4) **uart->BufTx** 関数を呼び出し、データを送信してください。

(5) **uart->UnInit** 関数を呼び出し、UART Driver の初期化解除を行ってください。

以下に初期化処理までのユーザプログラム例を示します。

```
extern UartDev_IOB  UartDev[];                                  /* (1-1) extern declaration */

void main(void)
{
    UartDev_IOB  *uart;

    uart = &UartDev[3];                                         /* (1-2) refer to UartDev[ ]*/

    uart->Cfg.BaudRate = 57600;               /* baudrate:57600bps */     /* (1-3) set Cfg */
    uart->Cfg.DataBits = UartDev_DATABITS_7;  /* databit:7bit      */     /* (1-3) set Cfg */

    uart->Init();                                                 /* (2) call init function */

    /* snip */
}
```

2.6 API関数リファレンス

本節では API 関数の呼出しインタフェースを説明します。

以降の記述形式は、以下のようになっています。

【形式】

関数のプロトタイプ宣言

【引数】

引数名、入出力方向[in, out, in/out]、説明、範囲を記述しています。

データ型については「**2.3データ型**」を、構造体については「**2.4構造体**」を参照してください。

【戻り値】

戻り値とその説明を記述しています。

【説明】

関数の説明を記述しています。

【注意】

注意事項を記述しています。

注意事項がない場合は、本項目はありません。

2.6.1 Init 《初期化》

【形式】

int32_t **Init**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

UART Driver の初期化を行います。本関数を呼び出すことで該当チャネルを使用できます。

該当チャネルに対応した **UartDev_IOB** 構造体のメンバ Cfg の値で初期化を行いますので、Cfg のデフォルト値以外を使用する場合は、本関数を呼び出す前に値を変更してください。

UART Driver、SIO Driver、SPI Driver、I²C Driverで使用中のチャネルに対して本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

2.6.2 UnInit 《初期化解除》

【形式】

int32_t **UnInit**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

UART Driver の初期化解除を行います。本関数を呼び出すことで該当チャネルを終了できます。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

2.6.3 BufTx 《データ送信》

【形式】

int32_t **BufTx**(void **pData*, int32_t **pSize*, uint32_t *flags*)

【引数】

pData	[in] 送信するデータの格納場所の先頭アドレス	
pSize	[in] 送信するバイト数	
	[out] 実際に送信したバイト数	
flags	[in] 送信方法	
	UartDev_FLAG_BLOCKING	ブロッキング
	UartDev_FLAG_NONBLOCKING	ノンブロッキング

【戻り値】

0	成功
-1	失敗

【説明】

データを送信します。本関数の送信方法はブロッキングとノンブロッキングの 2 種類あります。送信方法でブロッキングを選択し本関数を呼び出すことで、引数 *pSize* に指定したバイト数のデータを送信できます。その際、本関数はデータの送信が完了するまで処理をブロックします。送信方法でノンブロッキングを選択し本関数を呼び出すことで、送信バッファの空きサイズ数のデータを送信できます。その際、本関数は実際に送信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、送信方法にブロッキングを選択して本関数を呼び出さないでください。

2.6.4 BufRx 《データ受信》

【形式】

int32_t **BufRx**(void **pData*, int32_t **pSize*, uint32_t *flags*)

【引数】

pData	[out] 受信するデータの格納場所の先頭アドレス	
pSize	[in] 受信するバイト数	
	[out] 実際に受信したバイト数	
flags	[in] 受信方法	
	UartDev_FLAG_BLOCKING	ブロッキング
	UartDev_FLAG_NONBLOCKING	ノンブロッキング

【戻り値】

0	成功
-1	失敗

【説明】

データを受信します。本関数の受信方法はブロッキングとノンブロッキングの 2 種類あります。受信方法でブロッキングを選択し本関数を呼び出すことで、引数 *pSize* に指定したバイト数のデータを受信できます。その際、本関数はデータの受信が完了するまで処理をブロックします。受信方法でノンブロッキングを選択し本関数を呼び出すことで、受信バッファに格納されているデータを受信できます。その際、本関数は実際に受信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、送信方法にブロッキングを選択して本関数を呼び出さないでください。

2.6.5 BufFlush 《バッファフラッシュ》

【形式】

int32_t **BufFlush**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

送受信バッファをフラッシュします。本関数を呼び出すことで、送受信バッファに格納されているデータを破棄できます。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

3 I²C Driver

本章ではI²C Driverについて説明します。

3.1 概要

I²C Driverは 7 チャネル（ch1 からch7）に対応しています（ch0 はマイクロコントローラの仕様で使用できません）。ch4 からch7 はマクロ定義されたバイト数（0 バイトから 16 バイト）のハードウェアFIFOを使用します。。ch0 からch3 はハードウェアFIFOがありません。マスタモードとスレーブモードの両方に対応しています。DMA転送を使用した送受信には、全チャネル対応していません。

3.1.1 ファイル構成

I²C Driverのファイル構成を**表 3-1**に示します。

表 3-1 I²C Driverファイル構成

ファイル名	説明
I2cDev.h	I²C Driverを使用する場合は本ファイルをインクルードしてください。本ファイルには構造体定義、初期設定用マクロ定義を記述しています。
I2cDev_FM3.h	I2cDev_FM3.c からのみインクルードされます。本ファイルにはマイクロコントローラに依存した定義を記述しています。
I2cDev_FM3.c	I²C Driverを使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルにはAPI関数と内部関数を記述しています。
MfsDev_FM3.h	I2cDev_FM3.c からインクルードされます。本ファイルにはドライバ内部で使用するチャネル管理用の関数のプロトタイプ宣言を記述しています。
MfsDev_FM3.c	I²C Driverを使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルにはチャネル管理用の関数を記述しています。

3.2 マクロ定義

I²C Driverのマクロ定義について説明します。

3.2.1 FIFO バイト数

FIFOバイト数の指定用マクロを表 3-2に示します。これらの値によりチャネルのFIFOバイト数を変更できます。デフォルト値以外を使用する場合のみ変更してください。

表 3-2 FIFO バイト数の定義（I2cDev_FM3.h）

マクロ名	説明	デフォルト値
I2cDev_CH04_FIFO_DEPTH	Ch4 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
I2cDev_CH05_FIFO_DEPTH	Ch5 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
I2cDev_CH06_FIFO_DEPTH	Ch6 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
I2cDev_CH07_FIFO_DEPTH	Ch7 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16

3.3 データ型

I²C Driverでは、標準Cライブラリのstdint.hで定義されたデータ型を使用しています。stdint.hについては、ご使用のコンパイラのマニュアルを参照してください。

3.4 構造体

I²C Driverで使用する構造体について説明します。ドライバ構成情報構造体とI/Oブロック構造体のメンバ**CallBackIrq**は、ドライバの初期化時に値を設定してください。初期化の手順については、「**3.5使用手順**」をご覧ください。

3.4.1 I2cDev_CFG 《ドライバ構成情報》

【宣言】

```
typedef struct {
  int32_t BaudRate:20;
  int32_t SlaveAddr:10;
  int32_t AddrMode:2;
  int32_t Mode:2;
  int32_t Reserved:30;
} I2cDev_CFG;
```

【メンバ】

BaudRate	ボーレート（bps）	
	ご使用のシステムで有効な値（デフォルト：400000）	
SlaveAddr	スレーブアドレス	
	ご使用のシステムで有効な値（デフォルト：0x00）	
AddrMode	スレーブアドレスモード	
	I2cDev_ADDRMODE_7	7 ビットアドレスモード（デフォルト）
Mode	マスタ／スレーブ選択	
	I2cDev_MODE_MASTER	マスタモード（デフォルト）
	I2cDev_MODE_SLAVE	スレーブモード
Reserved	予約領域（値を設定しないでください）	

3.4.2 I2cDev_IOB 《I/O ブロック》

【宣言】

```
typedef struct {
   I2cDev_CFG   Cfg;
   int32_t  (*Init)(void);
   int32_t  (*UnInit)(void);
   int32_t  (*DataTxRx)(void *pDataTx, void *pDataRx, uint32_t Size);
   int32_t  (*DataTx)(void *pData, int32_t *pSize);
   int32_t  (*DataRx)(void *pData, int32_t *pSize);
   void     (*CallBackIrq)(int32_t Status);
} I2cDev_IOB;
```

【メンバ】

Cfg	**I2cDev_CFG 構造体**
Init	**初期化関数の関数ポインタ**
UnInit	**初期化解除関数の関数ポインタ**
DataTxRx	**データ送受信関数の関数ポインタ**
DataTx	**データ送信関数の関数ポインタ**
DataRx	**データ受信関数の関数ポインタ**
CallBackIrq	**割込み状況通知コールバック関数の関数ポインタ（デフォルト：NULL）** **（I²C Driver使用者が作成）**

3.5 使用手順

I²C Driverの使用手順を説明します。

3.5.1 初期化から初期化解除までの手順

初期化から初期化解除までの手順を**図 3-1**に示します。

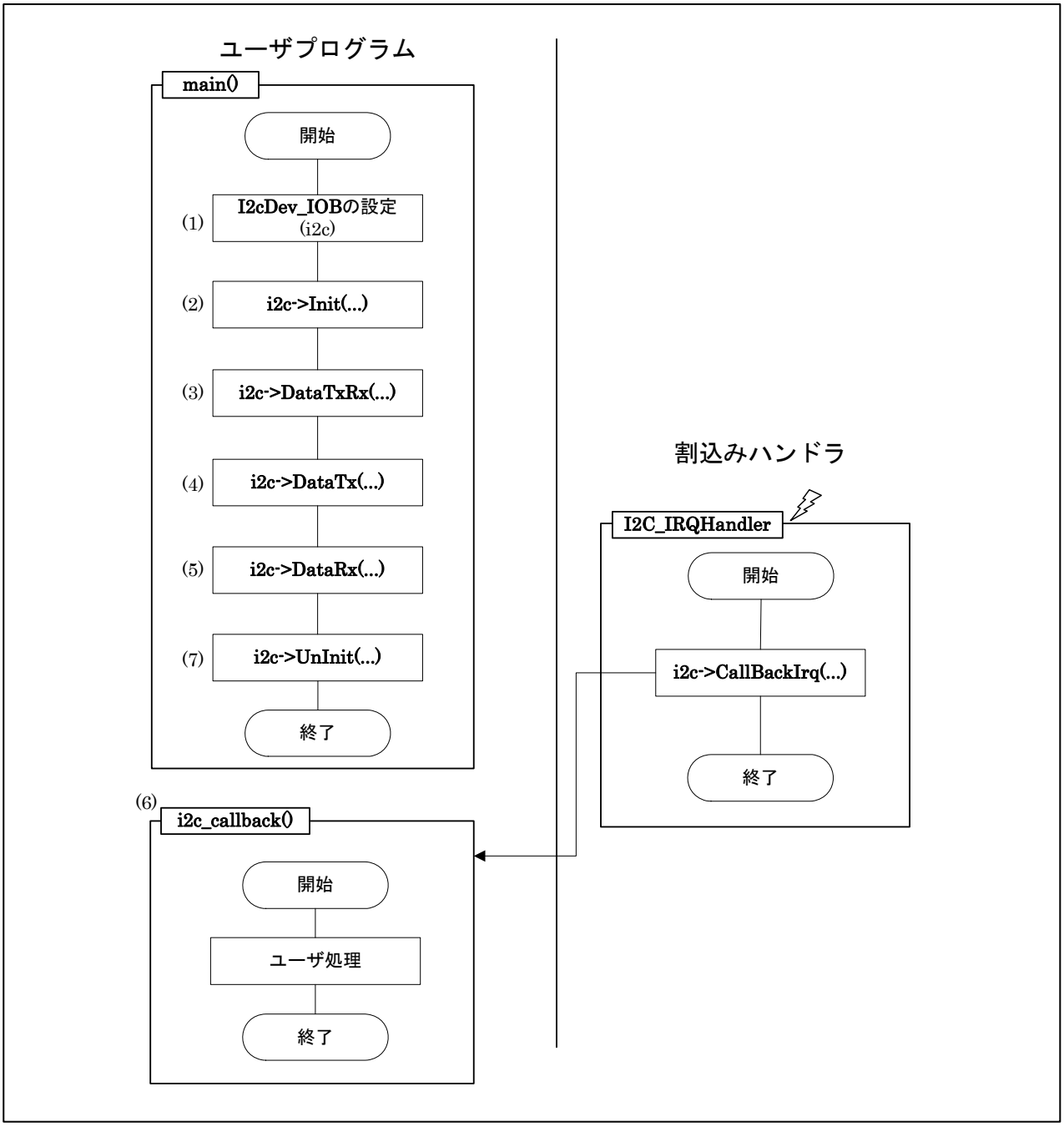

図 3-1 I²C Driverの初期化から初期化解除までの手順

(1) I²C Driverは、ドライバ内でチャネル数分のI2cDev_IOB構造体配列（I2cDev[ ]）を定義し、デフォルト値を設定します。したがって、I²C Driverを初期化する前に、以下の処理を行ってください。

(1-1) I2cDev[ ]を外部参照(extern 宣言)してください。

(1-2) チャネル番号を要素番号に指定して I2cDev[ ]を参照してください。

(1-3) **I2cDev_IOB** 構造体のメンバ Cfg のデフォルト値以外を使用する場合は値を変更してください。（Cfg の SlaveAddr は、初期化後の変更も有効です。）

(1-4) 割込み状況通知コールバック関数を使用する場合は、**I2cDev_IOB** 構造体のメンバ **CallBackIrq** に割込み状況通知コールバック関数を設定してください。

(2) **i2c->Init**関数を呼び出し、I²C Driverの初期化を行ってください。

(3) **i2c->DataTxRx** 関数を呼び出し、データを送受信してください。データの送信のみを行う場合は(4)を行ってください。データの受信のみを行う場合は(6)を行ってください。

(4) **i2c->DataTx** 関数を呼び出し、データを送信してください。スレーブアドレスを変更したい場合は、Cfg の SlaveAddr を再設定してください。

(5) **i2c->DataRx** 関数を呼び出し、データを受信してください。スレーブアドレスを変更したい場合は、Cfg の SlaveAddr を再設定してください。

(6) 割込みをトリガに割込みハンドラから割込み状況通知コールバック関数が呼び出されます。割込み発生タイミングは不定であるため、割込み状況通知コールバック関数が呼び出されるタイミングも不定です。

(7) **i2c->UnInit**関数を呼び出し、I²C Driverの初期化解除を行ってください。

以下に初期化処理までのユーザプログラム例を示します。

```
extern I2cDev_IOB  I2cDev[];                                    /* (1-1) extern declaration */

void main(void)
{
    I2cDev_IOB  *i2c;

    i2c = &I2cDev[4];                                            /* (1-2) refer to I2cDev[ ]*/

    i2c->Cfg.SlaveAddr    = 0xac;         /* slave address     */         /* (1-3) set Cfg */
    i2c->CallBackIrq      = i2c_callback;                          /* (1-4) set CallBackIrq */

    i2c->Init();                                                  /* (2) call init function */

    /* snip */
}

void i2c_callback(int32_t Status)
{
    /* I2C Driver user have to implement this function. */
}
```

3.6 API関数リファレンス

本節では API 関数の呼出しインタフェースを説明します。

以降の記述形式は、以下のようになっています。

【形式】

関数のプロトタイプ宣言

【引数】

引数名、入出力方向[in, out, in/out]、説明、範囲を記述しています。

データ型については「**3.3データ型**」を、構造体については「**3.4構造体**」を参照してください。

【戻り値】

戻り値とその説明を記述しています。

【説明】

関数の説明を記述しています。

【注意】

注意事項を記述しています。

注意事項がない場合は、本項目はありません。

3.6.1 Init 《初期化》

【形式】

int32_t **Init**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

I²C Driverの初期化を行います。本関数を呼び出すことで該当チャネルを使用できます。

該当チャネルに対応した **I2cDev_IOB** 構造体のメンバ Cfg の値で初期化を行いますので、Cfg のデフォルト値以外を使用する場合は、本関数を呼び出す前に値を変更してください。

I²C Driver、UART Driver、SIO Driver、SPI Driverで使用中のチャネルに対して本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

3.6.2 UnInit 《初期化解除》

【形式】

int32_t **UnInit**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

I²C Driverの初期化解除を行います。本関数を呼び出すことで該当チャネルを終了できます。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

3.6.3 DataTxRx 《データ送受信》

【形式】

int32_t **DataTxRx**(void **pDataTx*, void **pDataRx*, uint32_t *Size*)

【引数】

pDataTx	[in] 送信するデータの格納場所の先頭アドレス
pDataRx	[out] 受信するデータの格納場所の先頭アドレス
Size	[in] 送信するバイト数（受信するバイト数は送信するバイト数と同じ）

【戻り値】

0	成功
-1	失敗

【説明】

データを送受信します。本関数を呼び出すことで、スタートコンディションとストップコンディションの間でデータを送受信できます。例えば、スレーブデバイスに対してデバイスアドレスを送信して、データを受信する場合などに本関数を呼び出してください。マスタモードの場合に呼び出し可能です。本関数では、引数 *Size* に指定したバイト数分のデータを送信後、同じバイト数分のデータを受信します。

スレーブモードで呼び出した場合はエラー終了し、戻り値-1 を返します。Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

3.6.4 DataTx 《データ送信》

【形式】

int32_t **DataTx**(void **pData*, int32_t **pSize*)

【引数】

pData	[in] 送信するデータの格納場所の先頭アドレス
pSize	[in] 送信するバイト数
	[out] 実際に送信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを送信します。本関数はマスタモードとスレーブモードで処理に違いがあります。マスタモードの場合に本関数を呼び出すことで、**I2cDev_CFG** 構造体のメンバ SlaveAddr のスレーブデバイスへ、指定したデータを送信できます。スレーブモードの場合は、受信したアドレスが I2cDev_CFG 構造体のメンバ SlaveAddr と一致し、且つデータ方向が送信のときに、本関数を呼び出すことで、指定したデータを送信できます。本関数は実際に送信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

3.6.5 DataRx 《データ受信》

【形式】

int32_t **DataRx**(void **pData*, int32_t **pSize*)

【引数】

pData	[out] 受信するデータの格納場所の先頭アドレス
pSize	[in] 受信するバイト数
	[out] 実際に受信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを受信します。本関数はマスタモードとスレーブモードで処理に違いがあります。マスタモードの場合に本関数を呼び出すことで、**I2cDev_CFG** 構造体のメンバ SlaveAddr のスレーブデバイスから、指定したデータを受信できます。スレーブモードの場合に本関数を呼び出すことで、受信したアドレスが **I2cDev_CFG** 構造体のメンバ SlaveAddr と一致し、且つデータ方向がスレーブ受信を示すときに、指定したデータを受信できます。本関数は実際に受信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

3.6.6 CallBackIrq 《割込み状況通知コールバック関数》

【形式】

void **CallBackIrq**(int32_t *Status*)

【引数】

Status	[in] 割込み状況	
	I2cDev_IRQSTATUS_SLAVE_TX	スレーブ送信要求あり
	I2cDev_IRQSTATUS_SLAVE_RX	スレーブ受信要求あり

【戻り値】

なし

【説明】

割込み状況を知るためにI²C Driver使用者が作成する関数です。割込み状況を知る必要がないI²C Driver使用者は作成する必要はありません。本関数は、I²C Driver使用者が**I2cDev_IOB**構造体のメンバ**CallBackIrq**に設定した場合、I²C Driverの割込みハンドラから呼び出されます。本関数には、一意な関数名を付与してください。

スレーブモードでマスタから送信要求があった場合、引数 *Status* が **I2cDev_IRQSTATUS_SLAVE_TX** になります。スレーブモードでマスタから受信要求があった場合、引数 *Status* が **I2cDev_IRQSTATUS_SLAVE_RX** になります。

4 SIO Driver

本章では SIO Driver について説明します。

4.1 概要

SIO Driver は 8 チャネル（ch0 から ch7）に対応しています。ch4 から ch7 はマクロ定義されたバイト数（0 バイトから 16 バイト）のハードウェア FIFO を使用します。ch0 から ch3 はハードウェア FIFO がありません。マスタモードとスレーブモードの両方に対応しています。DMA 転送を使用した送受信と、4 チャネル同時通信には、全チャネル対応していません。

4.1.1 ファイル構成

SIO Driverのファイル構成を**表 4-1**に示します。

表 4-1 SIO Driver ファイル構成

ファイル名	説明
SioDev.h	SIO Driver を使用する場合は本ファイルをインクルードしてください。本ファイルには構造体定義、初期設定用マクロ定義を記述しています。
SioDev_FM3.h	SioDev_FM3.c からのみインクルードされます。本ファイルにはマイクロコントローラに依存した定義を記述しています。
SioDev_FM3.c	SIO Driver を使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルには API 関数と内部関数を記述しています。
MfsDev_FM3.h	SioDev_FM3.c からインクルードされます。本ファイルにはドライバ内部で使用するチャネル管理用の関数のプロトタイプ宣言を記述しています。
MfsDev_FM3.c	SIO Driver を使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルにはチャネル管理用の関数を記述しています。

4.2 マクロ定義

SIO Driver のマクロ定義について説明します。

4.2.1 FIFO バイト数

FIFOバイト数の指定用マクロを表 4-2に示します。これらの値によりチャネルのFIFOバイト数を変更できます。デフォルト値以外を使用する場合のみ変更してください。

表 4-2 FIFO バイト数の定義（SioDev_FM3.h）

マクロ名	説明	デフォルト値
SioDev_CH04_FIFO_DEPTH	Ch4 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
SioDev_CH05_FIFO_DEPTH	Ch5 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
SioDev_CH06_FIFO_DEPTH	Ch6 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16
SioDev_CH07_FIFO_DEPTH	Ch7 で使用する FIFO バイト数を指定します。0 から 16 を指定してください。FIFO を使用しない場合は 0 を指定してください。	16

4.3 データ型

SIO Driver では、標準 C ライブラリの stdint.h で定義されたデータ型を使用しています。stdint.h については、ご使用のコンパイラのマニュアルを参照してください。

4.4 構造体

SIO Driverで使用する構造体について説明します。ドライバ構成情報構造体とI/Oブロック構造体のメンバ**CallBackIrq**は、ドライバの初期化時に値を設定してください。初期化の手順については、「**4.5使用手順**」をご覧ください。

4.4.1 SioDev_CFG 《ドライバ構成情報》

【宣言】

```
typedef struct {
  int32_t BaudRate:20;
  int32_t DataBits:5;
  int32_t Mode:2;
  int32_t SckPolarity:2;
  int32_t BitOrder:2;
  int32_t Reserved:1;
} SioDev_CFG;
```

【メンバ】

BaudRate	ボーレート（bps）	
	ご使用のシステムで有効な値（デフォルト：115200）	
DataBits	データビット	
	SioDev_DATABITS_6	6 データビット
	SioDev_DATABITS_7	7 データビット
	SioDev_DATABITS_8	8 データビット（デフォルト）
Mode	マスタ／スレーブ選択	
	SioDev_MODE_MASTER	マスタモード（デフォルト）
	SioDev_MODE_SLAVE	スレーブモード
SckPolarity	シリアルクロック極性	
	SioDev_SCKPOLARITY_HIGH	High（デフォルト）
	SioDev_SCKPOLARITY_LOW	Low
BitOrder	ビットオーダー	
	SioDev_BITORDER_MSB	MSB ファースト（デフォルト）
	SioDev_BITORDER_LSB	LSB ファースト
Reserved	予約領域（値を設定しないでください）	

4.4.2 SioDev_IOB 《I/O ブロック》

【宣言】

```
typedef struct {
  SioDev_CFG   Cfg;
  int32_t  (*Init)(void);
  int32_t  (*UnInit)(void);
  int32_t  (*DataTx)(void *pData, int32_t *pSize);
  int32_t  (*DataRx)(void *pData, int32_t *pSize);
  void     (*CallBackIrq)(int32_t Status);
} SioDev_IOB;
```

【メンバ】

Cfg	**SioDev_CFG 構造体**
Init	**初期化関数の関数ポインタ**
UnInit	**初期化解除関数の関数ポインタ**
DataTx	**データ送信関数の関数ポインタ**
DataRx	**データ受信関数の関数ポインタ**
CallBackIrq	**割込み状況通知コールバック関数の関数ポインタ（デフォルト：NULL）** **（SIO Driver 使用者が作成）**

4.5 使用手順

SIO Driver の使用手順を説明します。

4.5.1 初期化から初期化解除までの手順

初期化から初期化解除までの手順を**図 4-1**に示します。

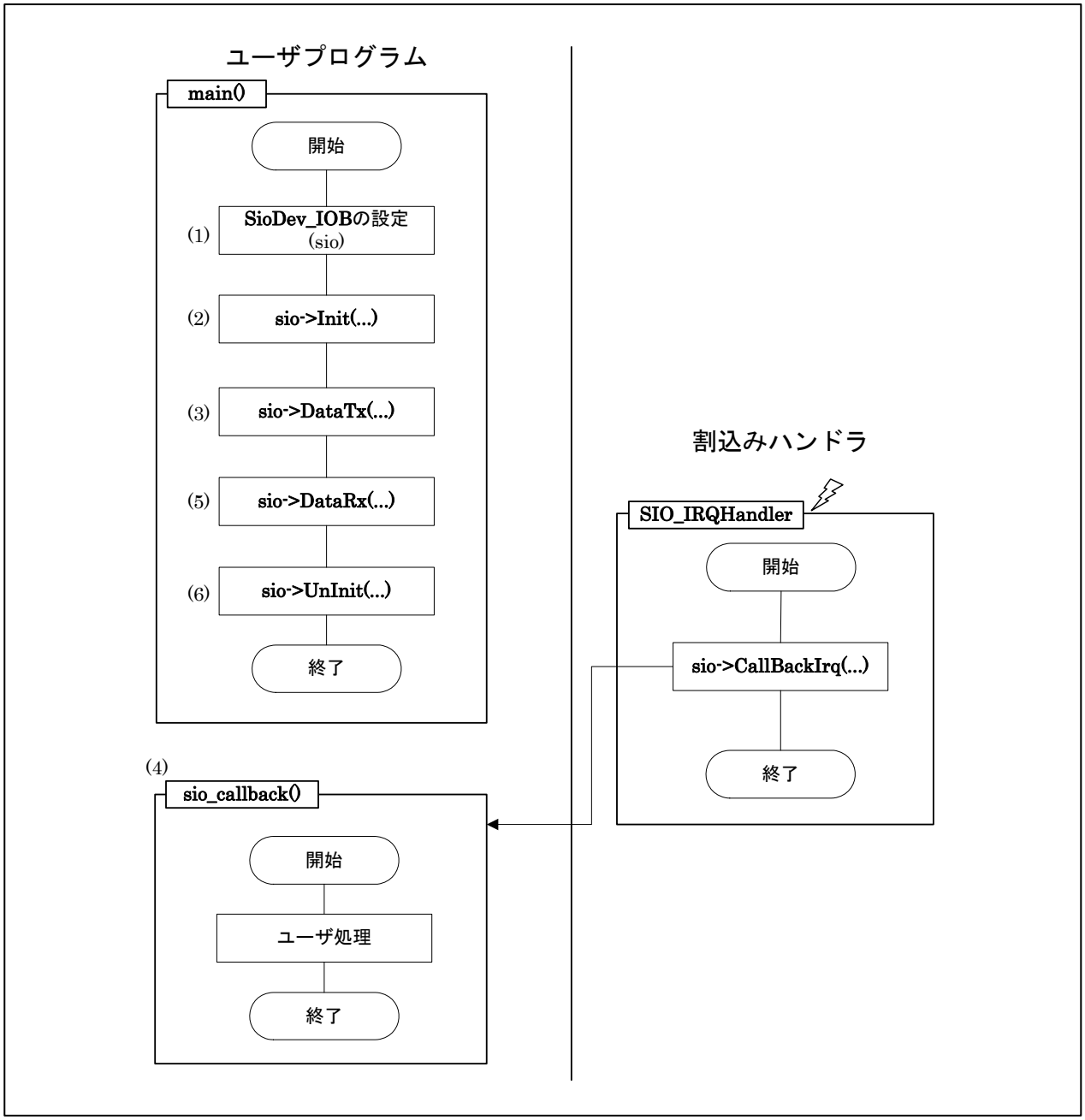

図 4-1 SIO Driver の初期化から初期化解除までの手順

(1) SIO Driver は、ドライバ内でチャネル数分の **SioDev_IOB** 構造体配列（SioDev[ ]）を定義し、デフォルト値を設定します。したがって、SIO Driver を初期化する前に、以下の処理を行ってください。

(1-1) SioDev[ ]を外部参照(extern 宣言)してください。

(1-2) チャネル番号を要素番号に指定して SioDev[ ]を参照してください。

(1-3) **SioDev_IOB** 構造体のメンバ Cfg のデフォルト値以外を使用する場合は値を変更してください。

(1-4) 割込み状況通知コールバック関数を使用する場合は、**SioDev_IOB** 構造体のメンバ **CallBackIrq** に割込み状況通知コールバック関数を設定してください。

(2) **sio->Init** 関数を呼び出し、SIO Driver の初期化を行ってください。

(3) **sio->DataTx** 関数を呼び出し、データを送信してください。

(4) 割込みをトリガに割込みハンドラからコールバック関数が呼び出されます。割込み発生タイミングは不定であるため、コールバック関数が呼び出されるタイミングも不定です。

(5) **sio->DataRx** 関数を呼び出し、データを受信してください。

(6) **sio->UnInit** 関数を呼び出し、SIO Driver の初期化解除を行ってください。

以下に初期化処理までのユーザプログラム例を示します。

```
extern SioDev_IOB  SioDev[];                                    /* (1-1) extern declaration */

void main(void)
{
    SioDev_IOB  *sio;

    sio = &SioDev[5];                                          /* (1-2) refer to SioDev[ ]*/

    sio->Cfg.SckPolarity = SioDev_SCKPOLARTY_LOW;  /* low */              /* (1-3) set Cfg */
    sio->CallBackIrq = sio_callback;                             /* (1-4) set CallBackIrq */

    sio->Init();                                                /* (2) call init function */

    /* snip */
}

void sio_callback(int32_t Status)
{
    /* SIO Driver user have to implement this function. */
}
```

4.6 API関数リファレンス

本節では API 関数の呼出しインタフェースを説明します。

以降の記述形式は、以下のようになっています。

【形式】

関数のプロトタイプ宣言

【引数】

引数名、入出力方向[in, out, in/out]、説明、範囲を記述しています。

データ型については「**4.3データ型**」を、構造体については「**4.4構造体**」を参照してください。

【戻り値】

戻り値とその説明を記述しています。

【説明】

関数の説明を記述しています。

【注意】

注意事項を記述しています。

注意事項がない場合は、本項目はありません。

4.6.1 Init 《初期化》

【形式】

int32_t **Init**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

SIO Driver の初期化を行います。本関数を呼び出すことで該当チャネルを使用できます。

該当チャネルに対応した **SioDev_IOB** 構造体のメンバ Cfg の値で初期化を行いますので、Cfg のデフォルト値以外を使用する場合は、本関数を呼び出す前に値を変更してください。

SIO Driver、I²C Driver、UART Driver、SPI Driverで使用中のチャネルに対して本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

4.6.2 UnInit 《初期化解除》

【形式】

int32_t **UnInit**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

SIO Driver の初期化解除を行います。本関数を呼び出すことで該当チャネルを終了できます。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

4.6.3 DataTx 《データ送信》

【形式】

int32_t **DataTx**(void **pData*, int32_t **pSize*)

【引数】

pData	[in] 送信するデータの格納場所の先頭アドレス
pSize	[in] 送信するバイト数
	[out] 実際に送信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを送信します。本関数を呼び出すことで指定したデータを送信できます。本関数は実際に送信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

4.6.4 DataRx 《データ受信》

【形式】

int32_t **DataRx**(void **pData*, int32_t **pSize*)

【引数】

pData	[out] 受信するデータの格納場所の先頭アドレス
pSize	[in] 受信するバイト数
	[out] 実際に受信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを受信します。本関数を呼び出すことで指定したデータを受信できます。本関数は実際に受信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

4.6.5 CallBackIrq 《割込み状況通知コールバック》

【形式】

void **CallBackIrq**(int32_t *Status*)

【引数】

Status	[in] 割込み状況	
	SioDev_IRQSTATUS_RX	受信データあり

【戻り値】

なし

【説明】

割込み状況を知るために SIO Driver 使用者が作成する関数です。割込み状況を知る必要がない SIO Driver 使用者は作成する必要はありません。本関数は、SIO Driver 使用者が **SioDev_IOB** 構造体のメンバ **CallBackIrq** に設定した場合、SIO Driver の割込みハンドラから呼び出されます。本関数には、一意な関数名を付与してください。

スレーブモードでマスタからデータの受信が発生した場合、引数 *Status* が **SioDev_IRQSTATUS_RX** になります。

5 SPI Driver

本章では SPI Driver について説明します。

5.1 概要

SPI Driver は最大 8 チャネル（ch0 から ch7）に対応しています。マスタモードとスレーブモードの両方に対応しています。また、スレーブセレクトはご使用のシステム構成に従ってユーザプログラムで操作してください。

SPI Driver は内部で SIO Driver を使用しています。

5.1.1 ファイル構成

SPI Driverのファイル構成を**表 5-1**に示します。また、SPI Driverを使用する場合は、SIO Driverのファイル（**表 4-1**）が必要です。

表 5-1 SPI Driver ファイル構成

ファイル名	説明
SpiDev.h	SPI Driver を使用する場合は本ファイルをインクルードしてください。本ファイルには構造体定義、初期設定用マクロ定義を記述しています。
SpiDev_FM3.h	SpiDev_FM3.c からのみインクルードされます。本ファイルにはマイクロコントローラに依存した定義を記述しています。
SpiDev_FM3.c	SPI Driver を使用する場合は本ファイルをコンパイルしリンクしてください。本ファイルには API 関数と内部関数を記述しています。

5.2 データ型

SPI Driver では、標準 C ライブラリの stdint.h で定義されたデータ型を使用しています。stdint.h については、ご使用のコンパイラのマニュアルを参照してください。

5.3 構造体

SPI Driverで使用する構造体について説明します。ドライバ構成情報構造体は、ドライバの初期化時に値を設定してください。初期化の手順については、「**5.4使用手順**」をご覧ください。

5.3.1 SpiDev_CFG 《ドライバ構成情報》

【宣言】

```
typedef struct {
  int32_t Baudrate:20;
  int32_t Polarity:2;
  int32_t Phase:2;
  int32_t SlaveSelect:2;
  int32_t Mode:2;
  int32_t BitOrder:2;
  int32_t Reserved:2;
} SpiDev_CFG;
```

【メンバ】

Baudrate	ボーレート（bps）	
	ご使用のシステムで有効な値（デフォルト：500000）	
Polarity	シリアルクロック極性	
	SpiDev_CLOCK_POLARITY_IDLELOW	Low
	SpiDev_CLOCK_POLARITY_IDLEHIGH	High（デフォルト）
Phase	未使用	
SlaveSelect	未使用	
Mode	マスタ／スレーブ選択	
	SpiDev_MODE_MASTER	マスタモード（デフォルト）
	SpiDev_MODE_SLAVE	スレーブモード
BitOrder	ビットオーダー	
	SpiDev_BITORDER_MSB	MSB ファースト（デフォルト）
	SpiDev_BITORDER_LSB	LSB ファースト
Reserved	予約領域（値を設定しないでください）	

5.3.2 SpiDev_IOB 《I/O ブロック》

【宣言】

```
typedef struct {
  SpiDev_CFG  Cfg;
  int32_t  (*Init)(void);
  int32_t  (*UnInit)(void);
  int32_t  (*BufTxRx)(void *pDataTx, void *pDataRx, uint32_t Size);
  int32_t  (*SetBaudrate)(int32_t Baudrate);
  int32_t  (*DataTx)(void *pData, int32_t *pSize);
  int32_t  (*DataRx)(void *pData, int32_t *pSize);
} SpiDev_IOB;
```

【メンバ】

Cfg	**SpiDev_CFG 構造体**
Init	初期化関数の関数ポインタ
UnInit	初期化解除関数の関数ポインタ
BufTxRx	データ送受信関数の関数ポインタ
SetBaudrate	ボーレート設定関数の関数ポインタ
DataTx	データ送信関数の関数ポインタ
DataRx	データ受信関数の関数ポインタ

5.4 使用手順

SPI Driver の使用手順をに示します。

5.4.1 初期化から初期化解除までの手順

初期化から初期化解除までの手順を図 5-1に示します。

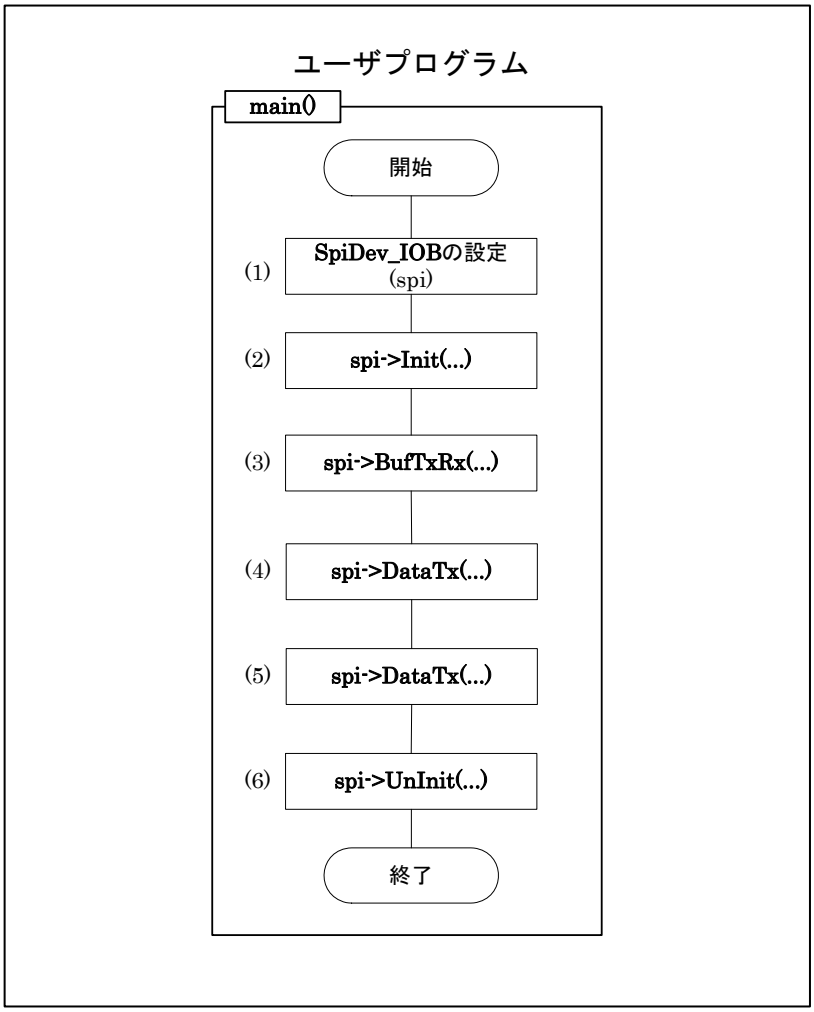

図 5-1 SPI Driver の初期化から初期化解除までの手順

(1) SPI Driver は、ドライバ内でチャネル数分の **SpiDev_IOB** 構造体配列（SpiDev[ ]）を定義し、デフォルト値を設定します。したがって、SPI Driver を初期化する前に、以下の処理を行ってください。

(1-1) SpiDev[ ]を外部参照(extern 宣言)してください。

(1-2) チャネル番号を要素番号に指定して SpiDev[ ]を参照してください。

(1-3) **SpiDev_IOB** 構造体のメンバ Cfg のデフォルト値以外を使用する場合は値を変更してください。

(2) **spi->Init** 関数を呼び出し、SPI Driver の初期化を行ってください。

(3) **spi->BufTxRx** 関数を呼び出し、データを送受信してください。データの送信のみを行う場合は(4)を行ってください。データの受信のみを行う場合は(5)を行ってください。

(4) **spi->DataTx** 関数を呼び出し、データを送信してください。

(5) **spi->DataRx** 関数を呼び出し、データを受信してください。

(6) **spi->UnInit** 関数を呼び出し、SPI Driver の初期化解除を行ってください。

以下に初期化処理までのユーザプログラム例を示します。

```
extern SpiDev_IOB  SpiDev[];                                    /* (1-1) extern declaration */

void main(void)
{
    SpiDev_IOB  *spi;

    spi = &SpiDev[6];                                            /* (1-2) refer to SpiDev[ ]*/

    spi->Cfg.Polarity = SpiDev_CLOCK_POLARITY_IDLELOW;  /* low */         /* (1-3) set Cfg */

    spi->Init();                                                  /* (2) call init function */

    /* snip */
}
```

5.5 API関数リファレンス

本節では API 関数の呼出しインタフェースを説明します。

以降の記述形式は、以下のようになっています。

【形式】

関数のプロトタイプ宣言

【引数】

引数名、入出力方向[in, out, in/out]、説明、範囲を記述しています。

データ型については「**5.2データ型**」を、構造体については「**5.3構造体**」を参照してください。

【戻り値】

戻り値とその説明を記述しています。

【説明】

関数の説明を記述しています。

【注意】

注意事項を記述しています。

注意事項がない場合は、本項目はありません。

5.5.1 Init 《初期化》

【形式】

int32_t **Init**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

SPI Driver の初期化を行います。本関数を呼び出すことで該当チャネルを使用できます。

該当チャネルに対応した **SpiDev_IOB** 構造体のメンバ Cfg の値で初期化を行いますので、Cfg のデフォルト値以外を使用する場合は、本関数を呼び出す前に値を変更してください。

SPI Driver 、SIO Driver、I²C Driver、UART Driverで使用中のチャネルに対して本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

5.5.2 UnInit 《初期化解除》

【形式】

int32_t **UnInit**(void)

【引数】

なし

【戻り値】

0	成功
-1	失敗

【説明】

SPI Driver の初期化解除を行います。本関数を呼び出すことで該当チャネルを終了できます。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

5.5.3 BufTxRx 《データ送受信》

【形式】

int32_t **BufTxRx**(void **pDataTx*, void **pDataRx*, uint32_t *Size*)

【引数】

pDataTx	[in] 送信するデータの格納場所の先頭アドレス
pDataRx	[out] 受信するデータの格納場所の先頭アドレス
Size	[in] 送受信するバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを送受信します。本関数を呼び出すことで指定したデータを送受信できます。本関数は、引数 *Size* に指定したバイト数分のデータ送受信を交互に繰り返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

5.5.4 SetBaudrate 《ボーレート設定》

【形式】

int32_t **SetBaudrate**(int32_t *Baudrate*)

【引数】

Baudrate	[in]	ボーレート（bps） ご使用のシステムで有効な値

【戻り値】

0	成功
-1	失敗

【説明】

ボーレートを変更します。初期化後にボーレートを変更したい場合に、本関数を呼び出してください。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

5.5.5 DataTx 《データ送信》

【形式】

int32_t **DataTx**(void **pData*, int32_t **pSize*)

【引数】

pData	[in] 送信するデータの格納場所の先頭アドレス
pSize	[in] 送信するバイト数
	[out] 実際に送信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを送信します。本関数を呼び出すことで指定したデータを送信できます。本関数は実際に送信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

5.5.6 DataRx 《データ受信》

【形式】

int32_t **DataRx**(void **pData*, int32_t **pSize*)

【引数】

pData	[out] 受信するデータの格納場所の先頭アドレス
pSize	[in] 受信するバイト数
	[out] 実際に受信したバイト数

【戻り値】

0	成功
-1	失敗

【説明】

データを受信します。本関数を呼び出すことで指定したデータを受信できます。本関数は実際に受信したバイト数を *pSize* に返します。

Init 関数を呼び出す前に本関数を呼び出した場合はエラー終了し、戻り値-1 を返します。

【注意】

割込みハンドラ内では、本関数を呼び出さないでください。

FM3 Family Drivers Software V01
MultiFunciton Serial Driver
ユーザーズマニュアル

2010 年 11 月 第 1.00 版発行

発 行 **富士通セミコンダクター株式会社**

編 集 マイコンソリューション事業本部ソフトウェア技術統括部ミドルウェア開発部