µNet3/ModBus ユーザーズガイド

第 3 版

イー・フォース株式会社

はじめに

Modbus プロトコルは、産業用自動化システムで広く使用されるオープンな通信プロトコルで、シリアル通信（RTU）および TCP/IP（TCP）を介した通信をサポートします。主にマスタ・スレーブ方式で動作し、マスタがリクエストを送信し、スレーブが応答します。

μNet3/ModBus(マイクロ・ネット・キューブ・モドバス) は、弊社　リアルタイムオペレーティングシステム **μC3 (マイクロ･シー･キューブ)** とその TCP/IP プロトコル スタック **μNet3 (マイクロ・ネット・キューブ)**上で動作する ModBus マスタ・スレーブで構成されています。

本書の位置づけ

本書は、μNet3/ModBus のマニュアルとしており、リアルタイム OS の使用方法などについては「μC3/Compact ユーザーズガイド」および「μC3/Standard ユーザーズガイド」を、TCP/IP プロトコルスタックの使用方法などについては「μNet3 ユーザーズガイド」を参照してください。

μC3 は、イー･フォース株式会社の登録商標です。
μNet3 は、イー･フォース株式会社の登録商標です。
本書で記載されている内容は、予告無く変更される場合があります。

Copyright © eForce Co.,Ltd. All Rights Reserved

改訂履歴

第 2 版で訂正された項目

ページ／章	内容
第 2 章 2.2 2.3	機能追加による変更。 ・mbs_get_map(), mbs_get_map_next(), mbs_get_map_req() の追加。 ・mbs_is_valid_addr()の引数追加。 ・mbu_write_4xword(), mbu_read_4xword(),mbu_swap_4xword()の追加。
第 4 章 4.2	機能追加によるサンプル内容の変更。

第 3 版で訂正された項目

ページ／章	内容
第 1 章 1.4 1.5	ModBus マスタ対応によるドキュメントの変更。 ・ModBus マスタ用の構成ファイルの追加。 ・ModBus マスタ用の概要、動作要件の追加。
第 2 章 2.1 2.4 2.5	ModBus マスタ対応によるドキュメントの変更。 ・ModBus マスタ API 用の章追加。 ・ModBus マスタ用の設定マクロ／構造体の章追加 ・ModBus スレーブ用の章を変更。(2.4 → 2.5)
第 3 章 3.1	ModBus マスタ対応によるドキュメントの変更。 ・ModBus マスタ用の章追加。
第 4 章 4.1	ModBus マスタ対応によるドキュメントの変更。 ・ModBus マスタ用の章追加。
全体	ModBus マスタ・スレーブのどちらの章かが分かるように [ma] … マスタ、[sl] … スレーブ とプレフィクスを追加した。

Copyright © eForce Co., Ltd. All Rights Reserved

目次

はじめに　2
目次　4
第 1 章　μNet3/ModBus とは 8
1.1　序説 8
1.2　ModBus の概要 8
1.3　μNet3-ModBus 適用範囲 9
1.4　μNet3-ModBus 構成ファイル 10
1.5　μNet3-ModBus マスタ [ma] 12
1.5.1　概要 12
1.5.2　動作要件 12
1.6　μNet3-ModBus スレーブ [sl] 13
1.6.1　概要 13
1.6.2　動作要件 13
1.6.3　スレーブタスク動作フロー 14
1.6.4　ファンクション処理実施フロー 16
第 2 章　μNet3/ModBus API 18
2.1　マスタ API [ma] 18
modbus_master（マスタ処理開始） 18
modbus_master_stop（停止） 20
mbm_exec_func（ファンクションコード実行） 21
mbm_F01（Read Coils） 22
mbm_F02（Read Discrete Inputs） 22
mbm_F03（Read Holding Registers） 23
mbm_F04（Read Input Registers） 23
mbm_F05（Write Single Coil） 24
mbm_F06（Write Single Register） 24
mbm_F15（Write Multiple Coils） 25
mbm_F16（Write Multiple registers） 25
mbm_set_slaveid（スレーブ ID の指定） 26
mbm_set_rsptmo（応答タイムアウト時間の設定） 27
2.2　スレーブ API [sl] 28
modbus_slave（スレーブ処理開始） 28
modbus_slave_stop（停止） 30
mbs_reg_func_cbk（コールバック関数の登録） 31
mbs_del_func_cbk（コールバック関数の削除） 32
mbs_ena_func_cbk（コールバック関数の有効・無効化） 32

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_reg_map（各テーブルのバッファ登録） **33**
mbs_reg_map_logi（各テーブルのバッファ登録／論理アドレス指定） **33**
mbs_del_map（各テーブルのバッファ登録解除） **35**
mbs_del_map_logi（各テーブルのバッファ登録解除／論理アドレス指定） **35**
mbs_get_map（各テーブルのバッファ取得） **36**
mbs_get_map_next（隣接後続バッファ取得） **36**
mbs_get_map_req（登録バッファに対する要求メッセージの相対値取得） **37**
cbk_mbs_chkreq（チェック用コールバック関数） **39**
cbk_mbs_func（ファンクション用コールバック関数） **40**
mbs_is_valid_quant（数量の妥当性チェック） **41**
mbs_is_valid_addr（アドレスの妥当性チェック） **42**
mbs_upd_buflen（通信バッファサイズの更新） **43**
2.3　ユーティリティ API [ma][sl] 44
mbu_addr_to_logi（ModBus アドレス変換 内部→論理） **44**
mbu_logi_to_addr（ModBus アドレス変換 論理→内部） **45**
mbu_copy_bits（ビット単位のコピー処理） **46**
mbu_copy_words（ワード単位のコピー処理） **46**
mbu_write_bit（1 ビット書込） **47**
mbu_read_bit（1 ビット読込） **47**
mbu_write_word（1 ワード書込） **48**
mbu_read_word（1 ワード読込） **48**
mbu_write_2xword（2 ワード/32 ビット書込） **49**
mbu_read_2xword（2 ワード/32 ビット読込） **49**
mbu_swap_2xword（2 ワード/32 ビット スワップ処理） **50**
mbu_write_4xword（4 ワード/64 ビット書込） **51**
mbu_read_4xword（4 ワード/64 ビット読込） **51**
mbu_swap_4xword（4 ワード/64 ビット スワップ処理） **52**
2.4　マスタ 設定マクロ／構造体 [ma] 53
T_MODBUS_MASTER (ModBus マスタ制御情報) **53**
T_MODBUS_MASTER_COM (ModBus マスタ制御情報 – COM 部) **53**
T_MODBUS_MASTER_TCP (ModBus マスタ制御情報 – TCP 部) **54**
ModBus マスタ設定ファイル (modbus_master_cfg.h) **54**
ファンクションコード実行 API 別名 **55**
2.5　スレーブ 設定マクロ／構造体 [sl] 56
T_MODBUS_SLAVE (ModBus スレーブ制御情報) **56**
T_MODBUS_SLAVE_COM (ModBus スレーブ制御情報 – COM 部) **56**
T_MODBUS_SLAVE_MAP (ModBus スレーブ テーブルマップ情報) **56**
ModBus スレーブ設定ファイル (modbus_slave_cfg.h) **58**

Copyright © eForce Co., Ltd. All Rights Reserved

第 3 章　コンフィグレーション **59**
3.1　μNet3-ModBus マスタ [ma] 59
3.1.1　プロジェクト導入手順 **59**
3.1.2　マスタ用リソースの準備 **60**
3.1.3　マスタ処理開始までの流れ **60**
3.2　μNet3-ModBus スレーブ [sl] 61
3.2.1　プロジェクト導入手順 **61**
3.2.2　スレーブ用リソースの準備 **62**
3.2.3　スレーブ処理開始までの流れ **63**
第 4 章　サンプルソースコード **64**
4.1　μNet3-ModBus マスタ [ma] 64
modbus_master_user.c 処理概要 **64**
ソース解説: サンプル動作設定 **65**
コマンド解説: ModBus マスタ開始 - mb, mbc **66**
コマンド解説: ModBus マスタ停止 - mb_exit **66**
コマンド解説: ファンクションコード実行 - mb_fxx **67**
コマンド解説: データ 32/64bit 表示 - mb_read32/64 ..... **68**
4.2　μNet3-ModBus スレーブ [sl] 69
modbus_slave_user.c 処理概要 **69**
ソース解説: サンプル動作設定 **70**
ソース解説: modbus スレーブ初期化処理 - コールバック関数登録 **71**
ソース解説: modbus スレーブ初期化処理 - 各テーブルのバッファ割当 **71**
ソース解説: modbus スレーブタスク(TCP 動作) **73**
ソース解説: modbus スレーブタスク(RTU 動作) **74**
ソース解説: 共通チェック用コールバック関数 **75**
ソース解説: 共通ファンクションコード用コールバック関数 **76**
ソース解説: HoldingRegisters チェック用コールバック関数 **81**
ソース解説: HoldingRegisters ファンクション用コールバック関数 **83**
第 5 章　付録 **86**
5.1　パケット形式 86
5.1.1　modbus メッセージアクセス用構造体 **86**
5.2　定数とマクロ 89
5.2.1　ModBus ファンクションコード **89**
5.2.2　ModBus 例外コード **89**
5.2.3　PDU データサイズ **90**
5.3　ModBus 用語 91
マスタ・スレーブ方式 **91**
PDU、ADU **91**

Copyright © eForce Co.,Ltd. All Rights Reserved

ファンクションコード..... 91
データフィールド..... 91
アドレスフィールド..... 91
要求メッセージ (Request PDU, mb_req_pdu)..... 92
応答メッセージ (Response PDU, mb_rsp_pdu)..... 92
例外応答メッセージ (Exception Response PDU, mb_excep_rsp_pdu)..... 92

Copyright © eForce Co., Ltd. All Rights Reserved

第1章　µNet3/ModBus とは

1. 1 序説

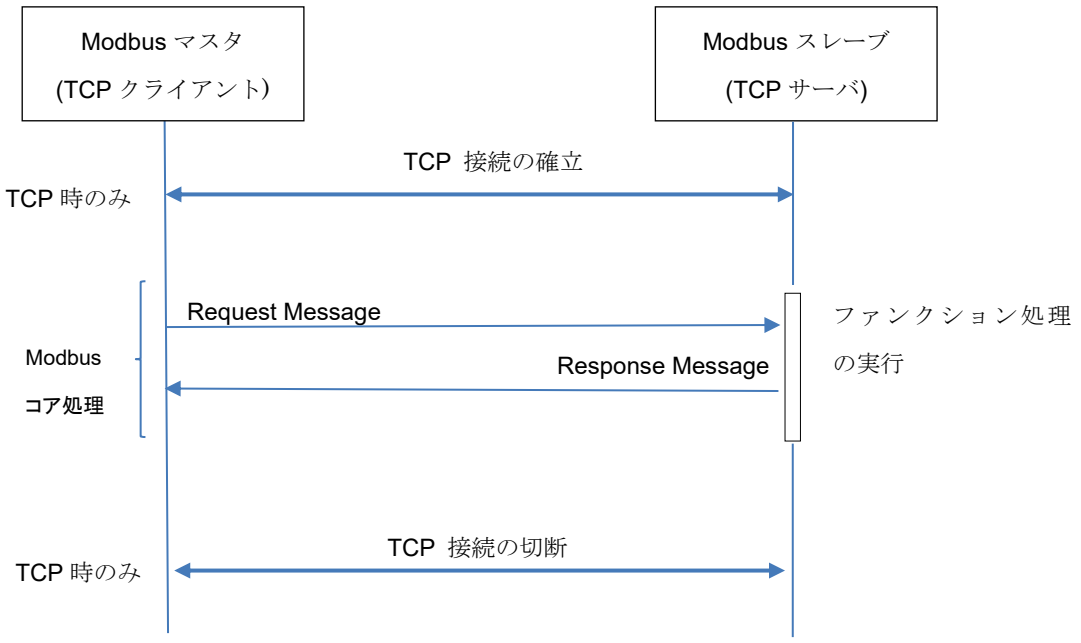
µNet3/ModBus は µC3(RTOS) + µNet3(TCP/IP プロトコルスタック)上で動作するコンパクトな ModBus プロトコルです。

以下 ModBus 公式ドキュメントを元に作成しています。ModBus プロトコルの詳細はこれらのドキュメントをご確認下さい。

・MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3
・MODBUS over serial line specification and implementation guide V1.02

1. 2 ModBus の概要

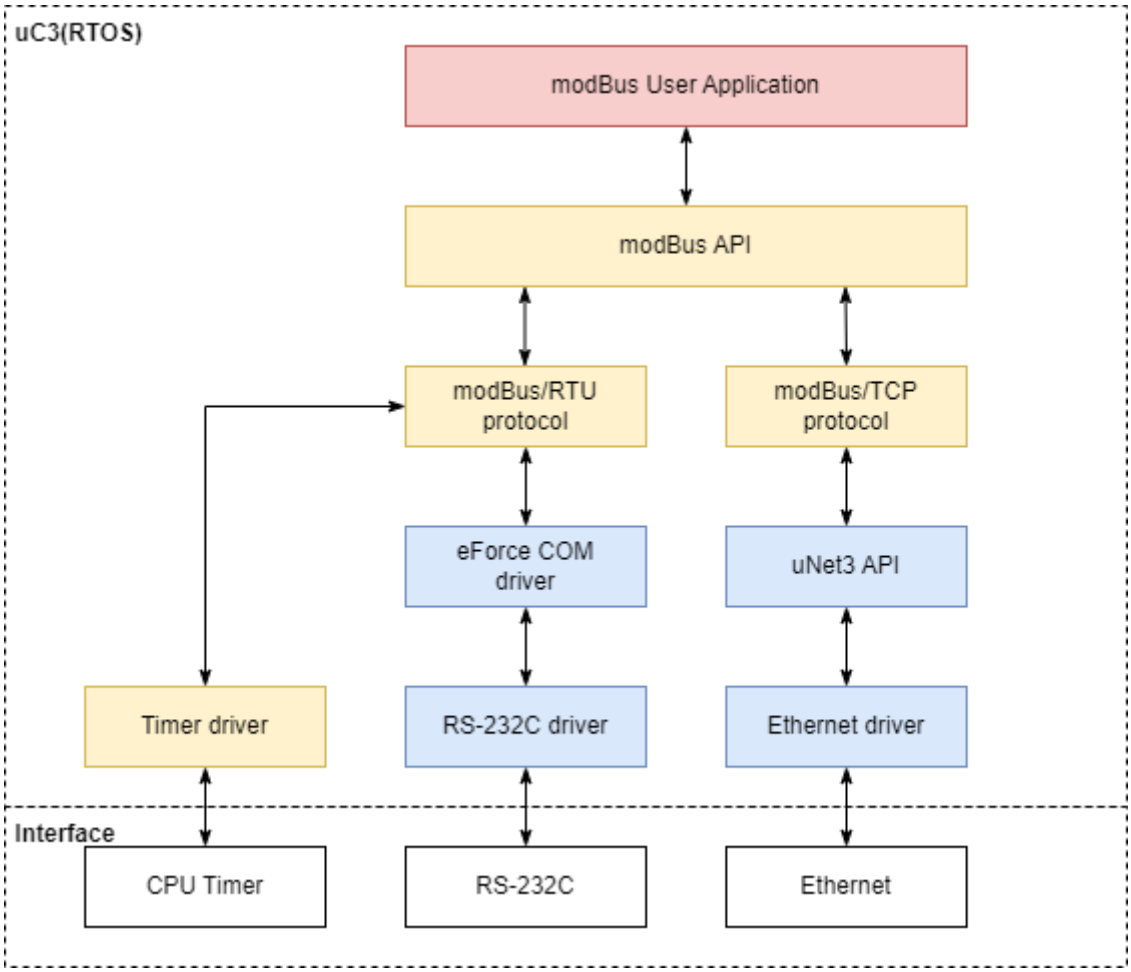
Modbus プロトコルは、産業用自動化システムで広く使用されるオープンな通信プロトコルで、シリアル通信（RTU）および TCP/IP（TCP）を介した通信をサポートします。主にマスタ・スレーブ方式で動作し、マスタがリクエストを送信し、スレーブが応答します。



Copyright © eForce Co.,Ltd. All Rights Reserved

1. 3 μNet3-ModBus 適用範囲

μNet3-ModBus はμC3+μNet3 上で動作する ModBus プロトコルになります。



ModBus プロトコルのコア処理部・黄色箇所の API を提供しています。ModBus 動作に当たっては赤色箇所の ModBus アプリケーション部分をユーザ側で作成する必要があります。

Copyright © eForce Co., Ltd. All Rights Reserved

1. 4 µNet3-ModBus 構成ファイル

µNet3-ModBus のファイルは uC3/Network/ModBus/フォルダ配下に格納されます。

µNet3-ModBus の構成ファイルは以下となります。

No.	ファイル名	処理概要	ma	sl
1	/src/modbus_master.c	Modbus マスタ動作	●	
2	/inc/modbus_master.h	Modbus マスタ動作 ヘッダ	●	
3	/cfg/modbus_master_cfg.h	Modbus マスタ 設定ファイル	●	
4	/src/modbus_master_tcp.c	Modbus マスタ TCP 動作	●	
5	/src/modbus_master_com.c	Modbus マスタ RTU 動作	●	
6	/cfg/modbus_master_user.c	Modbus マスタ サンプルソース(※)	●	
7	/src/modbus_slave.c	Modbus スレーブ動作		●
8	/inc/modbus_slave.h	Modbus スレーブ動作 ヘッダ		●
9	/cfg/modbus_slave_cfg.h	Modbus スレーブ 設定ファイル		●
10	/src/modbus_slave_tcp.c	Modbus スレーブ TCP 動作		●
11	/src/modbus_slave_com.c	Modbus スレーブ RTU 動作		●
12	/cfg/modbus_slave_user.c	Modbus スレーブ サンプルソース(※)		●
13	CPU 依存タイマドライバ	RTU 動作時のフレーム判定に使用	●	●
14	/inc/modbus_def.h	Modbus 定義(マスタ・スレーブ共通)	●	●
15	/src/modbus_slave_util.c	Modbus ユーティリティ API	●	●
16	/inc/modbus_slave_util.h	Modbus ユーティリティ API ヘッダ	●	●
列説明) [ma]ModBus マスタ動作に必要、**[sl]**ModBus スレーブ動作に必要				

µNet3-ModBus を TCP 動作する場合、別途以下が必要となります。

・uNet3 (TCP/IP プロトコルスタック)

・uNet3 デバイスドライバ (Ethernet ドライバ等)

µNet3-ModBus を RTU 動作する場合、別途以下が必要となります。

・eForce COM ドライバ + CPU 依存シリアルドライバ (µC3 パッケージ同梱)

・CPU 依存 Timer ドライバ (µNet3-ModBus に同梱、もしくはユーザ作成)

Copyright © eForce Co.,Ltd. All Rights Reserved

CPU 依存タイマドライバ

ModBus/RTU 動作時、フレームの判定処理に時間待ちを利用します。時間待ちは us 単位となるため、ModBus/RTU から利用可能なタイマドライバが別途必要となります。μNet3-ModBus に同梱するタイマドライバは CPU に搭載のタイマペリフェラルを利用しています。

以下はμNet3-ModBus に同梱されるタイマドライバです。

No.	ファイル名	概要
1	/hal/RX/DDR_RX_CMT_modbus.c	ルネサス RX 用タイマドライバ(CMT 使用)

提供パッケージにより上記ドライバが同梱しない場合があります、予めご了承下さい。

タイマドライバ インタフェース仕様

ModBus/RTU よりタイマドライバを呼び出すインタフェースは以下を利用します。ユーザ側でタイマドライバを用意する場合は同名 API を作成して下さい。

ER set_notify_timer(UB ch, UINT tmo_us, VP notify_cbk)

CPU のペリフェラルタイマのチャンネル ch を利用して、マイクロ秒 tmo_us 後に登録コールバック関数 notify_cbk をコールする用にタイマ割り込みを設定します。

void stop_notify_timer(UB ch)

CPU のペリフェラルタイマのチャンネル ch のタイマ割り込みを停止します。

eForce 提供タイマドライバは、タイマ 1ch=シリアル 1ch 利用の想定となります。そのため、複数シリアル ch を利用する場合、同数タイマ ch を用意して下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

1. 5 μNet3-ModBus マスタ [ma]

1. 5. 1 概要

μNet3-ModBus マスタ動作の機能概要を示します。

- TCP 動作をサポート (uNet3 使用)
- シリアル動作をサポート (eForce COM ドライバ、CPU 依存タイマドライバ使用 ※1)
  動作ボーレート例) 9600/19200/38400/57600/115200/230400 bps ※2
- データ転送はバイナリモードをサポート (ASCII モードは未サポート)
- 以下ファンクションコードをサポートしています。
  1: ReadCoils、2:ReadDiscreteInputs、3:ReadHoldingRegisters、4:ReadInputRegisters、
  5:WriteSingleCoil、6:WriteSingleRegister、15:WriteMultipleCoils、16:WriteMultipleRegisters

※1: シリアル動作時、API 処理中は指定デバイス・タイマを占有します。
他の用途に利用することは出来ません。(利用した場合の動作は不定となります)
※2: eForce COM ドライバの動作 CPU に依存して、ボーレート上限は制限されます。

1. 5. 2 動作要件

μNet3-ModBus マスタ動作に必要なリソースを示します。

μNet3-ModBus マスタ TCP 動作

・API 動作
・μNet3 TCP ソケット
・μNet3 デバイスドライバ (Ethernet ドライバ、無線 LAN ドライバなど)

μNet3-ModBus マスタ RTU 動作

・API 動作
・eForce COM ドライバ
・CPU 依存タイマドライバ (μNet3-ModBus に同梱)

Copyright © eForce Co.,Ltd. All Rights Reserved

1. 6 μNet3-ModBus スレーブ [sl]

1. 6. 1 概要

μNet3-ModBus スレーブ動作の機能概要を示します。

- TCP 動作をサポート (uNet3 使用)
- シリアル動作をサポート (eForce COM ドライバ、CPU 依存タイマドライバ使用 ※1)
  動作ボーレート例) 9600/19200/38400/57600/115200/230400 bps ※2
- データ転送はバイナリモードをサポート (ASCII モードは未サポート)
- ModBus メッセージのチェック処理、ファンクションコード実施・応答メッセージ作成処理はユーザ定義コールバック関数で行います。(サンプルコード同梱)
- サンプルコードは以下ファンクションコードをサポートしています。
  1: ReadCoils、2:ReadDiscreteInputs、3:ReadHoldingRegisters、4:ReadInputRegisters、5:WriteSingleCoil、6:WriteSingleRegister、15:WriteMultipleCoils、16:WriteMultipleRegisters

※1: シリアル動作時、指定デバイス・タイマを占有します。
他の用途に利用することは出来ません。(利用した場合の動作は不定となります)
※2: eForce COM ドライバの動作 CPU に依存して、ボーレート上限は制限されます。

1. 6. 2 動作要件

μNet3-ModBus スレーブ動作に必要なリソースを示します。

μNet3-ModBus スレーブ TCP 動作

- μC3 タスク
- μNet3 TCP ソケット
- μNet3 デバイスドライバ (Ethernet ドライバ、無線 LAN ドライバなど)

μNet3-ModBus スレーブ RTU 動作

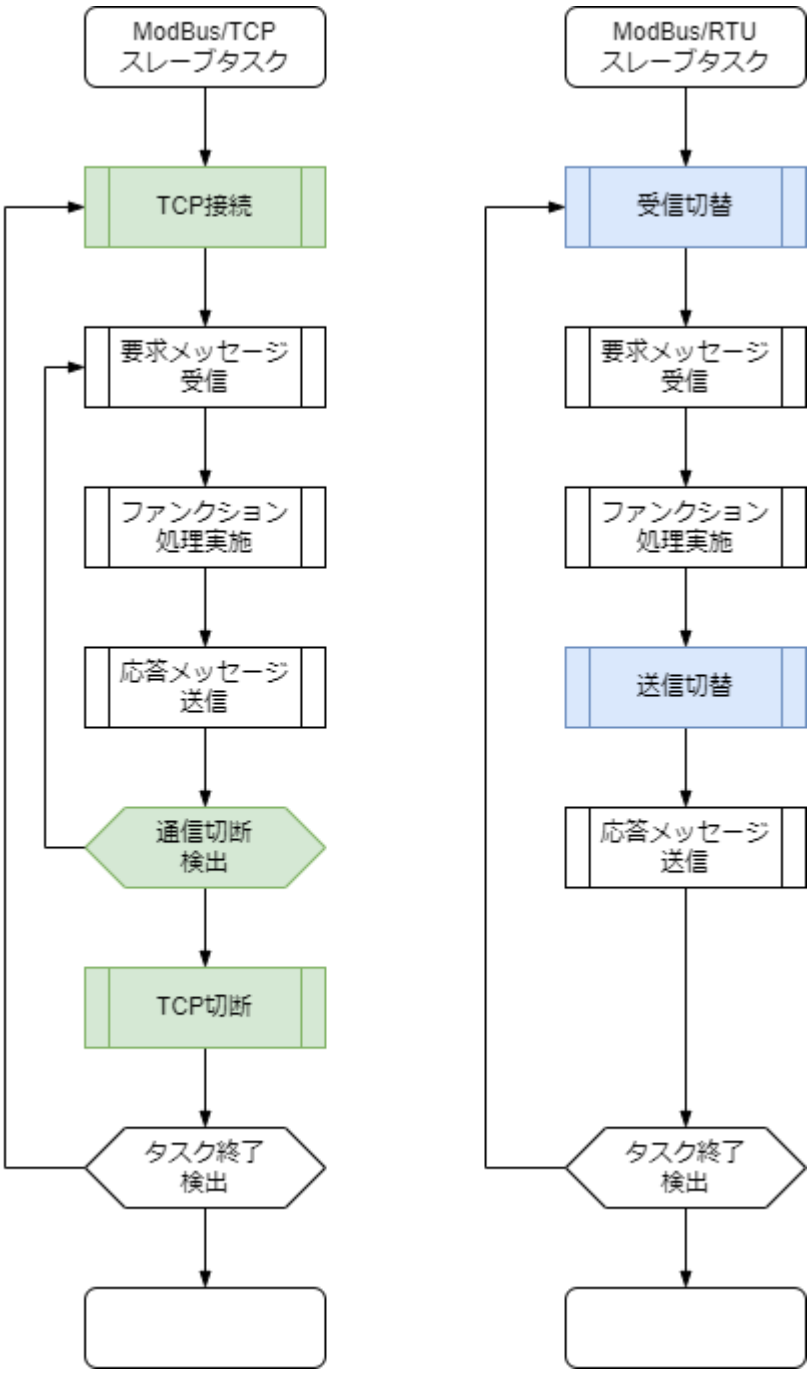
- μC3 タスク
- eForce COM ドライバ
- CPU 依存タイマドライバ (μNet3-ModBus に同梱)

1 タスク 1 チャンネル動作となります。例えば modbus スレーブを TCP ソケット 1 つ、UART チャネル 1 つで同時に利用したい場合、2 タスク必要となります。μNet3-ModBus スレーブで登録する要素(コールバック関数、バッファ)はタスク間で共通です。

Copyright © eForce Co., Ltd. All Rights Reserved

1. 6. 3　スレーブタスク動作フロー

μNet3-ModBus スレーブタスク(modbus_slave())の動作フローを示します。



Modbus TCP,RTU の動作フローで大きく異なるのは TCP 接続の有無となっています。

Copyright © eForce Co.,Ltd. All Rights Reserved

TCP 接続　(TCP 動作のみ)

modbus マスタからの接続待ちを行います。

要求メッセージ受信

modbus マスタからの要求メッセージを受信します。RTU 接続時は他スレーブ宛ての要求メッセージを受信する場合があるため、この場合は破棄します。

ファンクション処理実施

要求メッセージのファンクションコードに応じた処理を実施します。ここの実行処理はユーザアプリケーションから予めファンクションコードに対応したコールバック関数を登録しておき、そのコールバック関数内にて処理を実施します。

コールバック関数内では要求メッセージの解析、ファンクション処理の実施、応答メッセージの作成を行います。

応答メッセージ送信

ファンクション処理実施にて作成した応答メッセージを modbus マスタに送信します。

通信切断検出　(TCP 動作のみ)

通信処理中に切断が検出された場合、TCP 切断に移行します。

TCP 切断　(TCP 動作のみ)

通信処理が継続できない場合、TCP 接続を切断します。

受信切替／送信切替　(RTU 動作のみ)

シリアル通信の受信切替・送信切替を実施します。μNet3-ModBus では本処理はコールバック関数でユーザ定義します、ボード依存した処理(RS-485 の RTS 信号の切替など)を行う際に利用出来ます。後述するコールバック関数(sc.cbk_sw_tr)で詳細説明します。

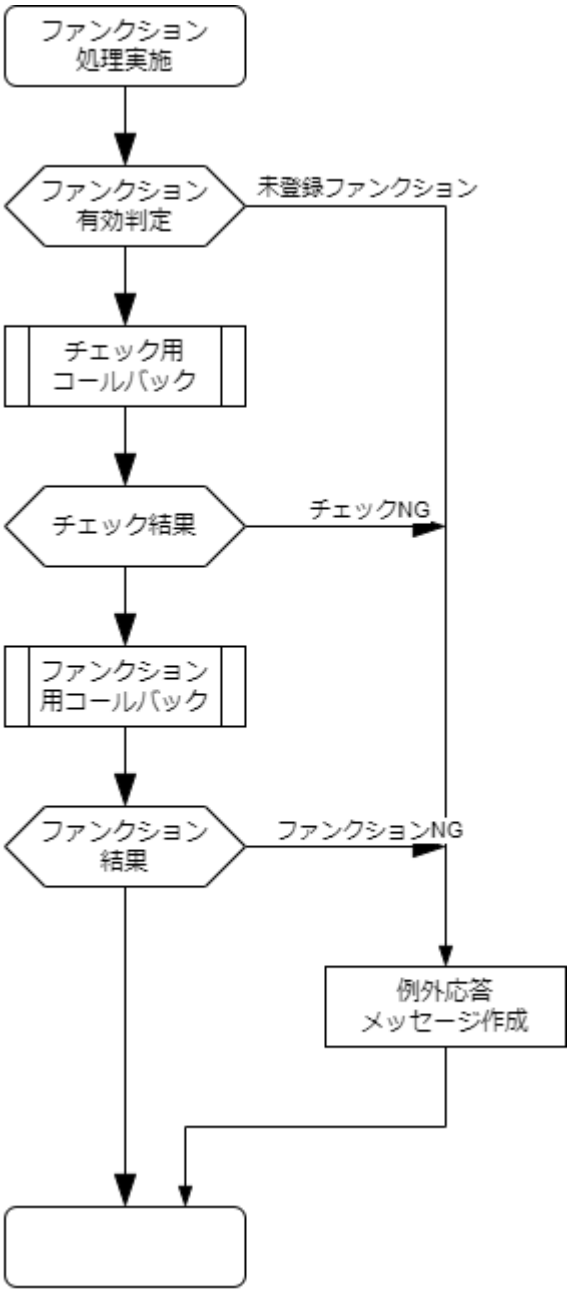
タスク終了検出

modbus スレーブタスクの終了 API が呼ばれた場合、タスクを終了します。

Copyright © eForce Co., Ltd. All Rights Reserved

1. 6. 4　ファンクション処理実施フロー

ファンクション処理実施の動作フローを示します。



ファンクション処理実施では、予めユーザ登録されたコールバック関数を呼び、ファンクションの実施と応答メッセージの作成を行います。ファンクションコードに応じたコールバック関数はチェック用とファンクション用の 2 種類設定できます。(チェック用コールバックは不要であれば未登録も可能です。)

Copyright © eForce Co.,Ltd. All Rights Reserved

ファンクション有効判定

ファンクションコードに応じたコールバック関数が未登録の場合、エラーとなります。(EC01_ILLEGAL_FUNCTION)

チェック用コールバック、チェック結果

ファンクションコードに応じたチェック用コールバック関数を呼ばれます。
要求メッセージのチェック処理を実施して下さい。(未登録時は何もしません)

チェック結果が NG の場合、例外コードをコールバック関数の戻り値として下さい。(EC02_ILLEGAL_DATA_ADDRESS、EC03_ILLEGAL_DATA_VALUE)

ファンクション用コールバック、ファンクション結果

ファンクションコードに応じたファンクション用コールバック関数を呼ばれます。
要求メッセージに応じたファンクション処理の実施、応答メッセージの作成を行って下さい。
ファンクション結果が NG の場合、例外コードをコールバック関数の戻り値として下さい。

例外応答メッセージ作成

2 バイトの例外応答メッセージを作成します。1 バイト目が Function code + 0x80、2 バイト目が例外コード(コールバック関数の戻り値)になります。

Copyright © eForce Co., Ltd. All Rights Reserved

第2章　μNet3/ModBus API

2. 1 マスタ API [ma]

modbus_master（マスタ処理開始）

【書式】

ER ercd = modbus_master(T_MODBUS_MASTER *mm);

【パラメータ】

T_MODBUS_MASTER	* mm	ModBus マスタ制御情報

【戻り値】

ER	ercd	次のエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*mn が NULL。mm->id が不正、mm->com が未指定など）
E_CLS	TCP 接続に失敗した。(対向が切断)
E_TMOUT	TCP 接続に失敗した。(タイムアウト)

【解説】

このAPIはModBusマスタを動作させ、ModBusスレーブに要求を投げる準備を行います。

接続タイプ mm->type に MODBUS_TYPE_TCP を指定すると ModBus/TCP 動作、MODBUS_TYPE_RTU を指定すると ModBus/RTU 動作します。

mm->id には MODBUS_TYPE_TCP 指定時は TCP ソケット ID、MODBUS_TYPE_RTU 指定時は COM のデバイス ID を指定して下さい。
TCP ソケット ID は予め生成(cre_soc)済みのソケット ID を指定して下さい。
COM デバイス ID は予め初期化(ini_com)済みのデバイス ID を指定して下さい。

T_MODBUS_MASTER::com には TCP 動作時・RTU 動作時で異なる構造体を割り当てます。
TCP 動作時は T_MODBUS_MASTER_TCP 構造体変数を割り当てます。
RTU 動作時は T_MODBUS_MASTER_COM 構造体変数を割り当てます。

Copyright © eForce Co.,Ltd. All Rights Reserved

(T_MODBUS_MASTER_TCP 変数 mt が定義されている仮定で以下説明)
変数 mt には以下を設定して下さい。(これ以外の構造体メンバは制御用のため設定不要です)
・mt.svr　　接続先 ModBus スレーブのリモート情報 (T_NODE)

mt.svr に設定した ModBus スレーブに TCP 接続します。接続失敗するとエラーとなります。

(T_MODBUS_MASTER_COM 変数 mc が定義されている仮定で以下説明)
変数 mc には以下を設定して下さい。(これ以外の構造体メンバは制御用のため設定不要です)
・mc.slave_id　ModBus スレーブの宛先スレーブ ID
・mc.baud_rate　シリアル通信のボーレート速度
・mc.tmr_ch　タイマドライバの使用チャンネル番号
・mc.cbk_sw_tr　シリアル送信・受信切替コールバック関数

シリアル送信・受信切替コールバック関数(sc.cbk_sw_tr)は送信・受信前のタイミングで呼ばれるコールバック処理となります。

RTU 動作時、シリアル通信は eForce COM ドライバが実施します。
RS-485 通信時には送信・受信許可信号(DE/RE 信号など)を制御する必要が出てきますが、COM ドライバにはこれらの信号の制御処理はありません。そのため、上記コールバック関数を利用してユーザ関数での信号の制御処理を実装するようにして下さい。

以下はRX72M評価ボードでのRS-485送信･受信許可信号の制御サンプルコードになります。
(このボードは P80 ピンが RS-485 IC の DE、RE# 信号に接続されており、
P80 ピンを High、Low とすることで送信・受信を切り替えます。)

sc.cbk_sw_tr コールバック関数サンプル

```
/* RS-485信号制御 (P80=DE=RE#)  */
void cbk_rs485_tx_enable(UB ena_tx)
{
    if (ena_tx) {   /* ena_tx=1 で送信許可 */
        PORT8.PODR.BIT.B0 = 1U;   /* Driver output enable */
    }
    else {    /* ena_tx=0 で受信許可 */
        PORT8.PODR.BIT.B0 = 0U;   /* Receiver output enable */
    }
}
```


Copyright © eForce Co., Ltd. All Rights Reserved

modbus_master_stop（停止）

【書式】

ER ercd = modbus_master_stop(T_MODBUS_MASTER *mm)

【パラメータ】

T_MODBUS_MASTER	*mm	ModBus マスタ制御情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*mm が NULL）
E_OBJ	実行不可の状態。(modbus_master()未実施)

【解説】

この API は Modbus マスタの処理を停止、スレーブとの接続を終了します。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbm_exec_func（ファンクションコード実行）

【書式】

ER mbm_exec_func(T_MODBUS_MASTER *mm,
T_MODBUS_DATA **rsp, T_MODBUS_MASTER_PARAM *mp)

【パラメータ】

T_MODBUS_MASTER	*mm	ModBus マスタ制御情報
T_MODBUS_DATA	**rsp	応答データ
T_MODBUS_MASTER_PARAM	*mp	ファンクション用パラメータ

【戻り値】

ER	ercd	受信メッセージサイズ（>0）またはエラーコード

【エラーコード】

0	ブロードキャスト完了（受信データなし）
E_PAR	不正なパラメータが指定された。 （*mm や *mp が NULL）
E_OBJ	実行不可の状態。(modbus_master()未実施、通信切断など)
E_TMOUT	応答パケットが時間内に来なかった。

【解説】

Modbus スレーブに対してファンクションコードを発行します。mp の値でファンクションコード、アドレス、個数などを制御します。

戻り値が正常値の場合、応答データ rsp が有効となります。これは ModBus マスタ制御情報構造体のバッファを間接的に指すため、rsp にはポインタ変数を渡して下さい。
応答データのファンクションコード rsp->pdu->func_code に応じた応答データの構造体の型は以下となります。

func_code	rsp のキャスト先の構造体	備考
0x01, 0x02	T_MODBUS_DATA_RSP1	
0x03, 0x04	T_MODBUS_DATA_RSP2	
0x05, 0x06, 0x0F, 0x10	T_MODBUS_DATA_RSP3	
0x81 ～ 0x90	T_MODBUS_DATA_ERR	

後述する mbm_Fxx_yyyy()と命名された API は、この API をファンクションコード毎に分けた API になります。必要に応じて使い分けして下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbm_F01（Read Coils）

【書式】

ER mbm_F01(T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP1 **rsp, UH addr, UH coils)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP1 **rsp	応答データ
UH addr	開始アドレス
UH coils	コイル数

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 01 Read Coils を発行します。

mbm_F02（Read Discrete Inputs）

【書式】

ER mbm_F02(T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP1 **rsp, UH addr, UH inputs)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP1 **rsp	応答データ
UH addr	開始アドレス
UH inputs	ディスクリート入力数

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 02 Read Discrete Inputs を発行します。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbm_F03（Read Holding Registers）

【書式】

ER mbm_F03 (T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP2 **rsp, UH addr, UH regs)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP2 **rsp	応答データ
UH addr	開始アドレス
UH regs	レジスタ数

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 03 Read Holding Registers を発行します。

mbm_F04（Read Input Registers）

【書式】

ER mbm_F04 (T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP2 **rsp, UH addr, UH regs)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP2 **rsp	応答データ
UH addr	開始アドレス
UH regs	レジスタ数

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 04 Read Input Registers を発行します。

Copyright © eForce Co., Ltd. All Rights Reserved

mbm_F05（Write Single Coil）

【書式】

ER mbm_F05 (T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP3 **rsp, UH addr, UB val)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP3 **rsp	応答データ
UH addr	開始アドレス
UH val	データ(ビット)

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 05 Write Single Coil を発行します。指定データ val は 0 以外→0xFF00、0→0x0000 と Modbus の ON/OFF 値に変換して送付します。

mbm_F06（Write Single Register）

【書式】

ER mbm_F06(T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP3 **rsp, UH addr, UH val)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP3 **rsp	応答データ
UH addr	開始アドレス
UH val	データ(レジスタ)

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 06 Write Single Register を発行します。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbm_F15（Write Multiple Coils）

【書式】

ER mbm_F15(T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP3 **rsp, UH addr, UH outputs, UB *val)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP3 **rsp	応答データ
UH addr	開始アドレス
UH outputs	ビット数
UB *val	ビットデータ(ビット単位の 0,1) * outputs

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 15 Write Multiple Coils を発行します。
指定データ val は val[0]:b0 から b15、val[1]:b16～b31、…～b[outputs-1]の順番で指定します。

mbm_F16（Write Multiple registers）

【書式】

ER mbm_F16(T_MODBUS_MASTER *mm,
T_MODBUS_DATA_RSP3 **rsp, UH addr, UH regs, UH *val)

【パラメータ】

T_MODBUS_MASTER *mm	ModBus マスタ制御情報
T_MODBUS_DATA_RSP3 **rsp	応答データ
UH addr	開始アドレス
UH regs	レジストリ数
UH *val	レジストリデータ * regs

※ 戻り値、エラーコードは mbm_exec_func()と同じです。

【解説】

Modbus スレーブに対して、ファンクション: 16 Write Multiple registers を発行します。

Copyright © eForce Co., Ltd. All Rights Reserved

mbm_set_slaveid（スレーブ ID の指定）

【書式】

ER mbm_set_slaveid(T_MODBUS_MASTER *mm, UB slaveid);

【パラメータ】

T_MODBUS_MASTER	*mm	ModBus マスタ制御情報
UB slaveid		宛先スレーブ ID

【戻り値】

ER	ercd	現在の設定値（>=0）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*mm が NULL。slaveid が有効値ではない）
E_OBJ	実行不可の状態。(modbus_master()未実施)

【解説】

宛先スレーブ ID を指定します。ModBus/RTU 時、ブロードキャスト送信をする場合は slaveid に MODBUS_SLAVE_BC(0)を指定して下さい。ModBus/TCP 時、unit_id フィールドに入る値を指定できます。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbm_set_rsptmo（応答タイムアウト時間の設定）

【書式】

ER mbm_set_rsptmo(T_MODBUS_MASTER *mm, UH tmo);

【パラメータ】

T_MODBUS_MASTER	*mm	ModBus マスタ制御情報
UH tmo		応答タイムアウト時間(ms)

【戻り値】

ER	ercd	現在の設定時間（>0）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*mm が NULL）
E_OBJ	実行不可の状態。(modbus_master()未実施)

【解説】

ファンクションコード実行時の応答待ち時間を設定します。

tmo に 0 指定すると応答待ち時間に初期値にします。初期値は以下を定義しています。

・ModBus/RTU： (1T * ModBus ペイロード最大(253) * 2 + 3.5T) / 1000

・ModBus/TCP： 定義値 MM_TCP_RCV_TMO(2000)

Copyright © eForce Co., Ltd. All Rights Reserved

2. 2 スレーブ API [sl]

modbus_slave（スレーブ処理開始）

【書式】

ER ercd = modbus_slave(T_MODBUS_SLAVE *ms);

【パラメータ】

T_MODBUS_SLAVE	* ms	ModBus スレーブ制御情報

【戻り値】

ER	ercd	次のエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*ms が NULL。ms->id が不正、RTU 動作時に ms->com が未指定など）
E_RLWAI	mobus_slave_stop()がコールされ停止した。

【解説】

この API は ModBus スレーブを動作させ、ModBus マスタからの要求を受け付けます。専用タスクを用意して API を発行するようにして下さい。
接続タイプ ms->type に MODBUS_TYPE_TCP を指定すると TCP 動作のスレーブとして、MODBUS_TYPE_RTU を指定すると RTU 動作のスレーブとして動作します。

ms->id には MODBUS_TYPE_TCP 指定時は TCP ソケット ID、MODBUS_TYPE_RTU 指定時は COM のデバイス ID を指定して下さい。
TCP ソケット ID は予め生成(cre_soc)済みのソケット ID を指定して下さい。
COM デバイス ID は予め初期化(ini_com)済みのデバイス ID を指定して下さい。

RTU 動作時には T_MODBUS_SLAVE::com に、T_MODBUS_SLAVE_COM 構造体変数を割り当てます。(T_MODBUS_SLAVE_COM 変数 sc が定義されている仮定で以下説明)

Copyright © eForce Co.,Ltd. All Rights Reserved

変数 sc には以下を設定して下さい。(これ以外の構造体メンバは制御用のため設定不要です)

・sc.slave_id　　ModBus スレーブの自スレーブ ID
・sc.baud_rate　シリアル通信のボーレート速度
・sc.tmr_ch　　タイマドライバの使用チャンネル番号
・sc.cbk_sw_tr　シリアル送信・受信切替コールバック関数

シリアル送信・受信切替コールバック関数(sc.cbk_sw_tr)は送信・受信前のタイミングで呼ばれるコールバック処理となります。

RTU 動作時、シリアル通信は eForce COM ドライバが実施します。
RS-485 通信時には送信・受信許可信号(DE/RE 信号など)を制御する必要が出てきますが、COM ドライバにはこれらの信号の制御処理はありません。そのため、上記コールバック関数を利用してユーザ関数での信号の制御処理を実装するようにして下さい。

以下はRX72M評価ボードでのRS-485送信･受信許可信号の制御サンプルコードになります。
(このボードは P80 ピンが RS-485 IC の DE、RE# 信号に接続されており、
P80 ピンを High、Low とすることで送信・受信を切り替えます。)

sc.cbk_sw_tr コールバック関数サンプル

```
/* RS-485信号制御 (P80=DE=RE#)  */
void cbk_rs485_tx_enable(UB ena_tx)
{
    if (ena_tx) {   /* ena_tx=1 で送信許可 */
        PORT8.PODR.BIT.B0 = 1U;   /* Driver output enable */
    }
    else {   /* ena_tx=0 で受信許可 */
        PORT8.PODR.BIT.B0 = 0U;   /* Receiver output enable */
    }
}
```


Copyright © eForce Co., Ltd. All Rights Reserved

modbus_slave_stop（停止）

【書式】

ER ercd = modbus_slave_stop(T_MODBUS_SLAVE *ms)

【パラメータ】

T_MODBUS_SLAVE	*ms	ModBus スレーブ制御情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*ms が NULL。ms->type が不正など）

【解説】

この API は起動している Modbus サーバに停止要求を出します。modbus_slave()が戻り値 E_RLWAI で処理終了しますが、通信中は即座に停止しない場合があります。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_reg_func_cbk（コールバック関数の登録）

【書式】

ER mbs_reg_func_cbk(UB func_code, cbk_mbs_chkreq chkreq, cbk_mbs_func func);

【パラメータ】

UB	func_code	ファンクションコード
cbk_mbs_chkreq	chkreq	チェック用コールバック関数
cbk_mbs_func	func	ファンクション用コールバック関数

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （func_code が有効値ではない、func が NULL）
E_NOMEM	登録可能なファンクションコードの空きがない

【解説】

ファンクションコードに対応したコールバック関数を登録します。コールバック関数はチェック用とファンクション用の 2 種類設定できます。ファンクション用コールバック関数内でチェック処理も行う合、チェック用コールバック関数には NULL 指定が可能です。
コールバック関数が未登録のファンクションコードの要求メッセージを受信した場合、μNet3-ModBus スレーブは例外応答メッセージの 01: ILLEGAL_FUNCTION を返します。

エラーコード E_NOMEM が返却される場合、設定ファイルの定義「USE_FUNCCODE_NUM」の値を適宜増やして下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbs_del_func_cbk（コールバック関数の削除）

【書式】

ER mbs_del_func_cbk(UB func_code);

【パラメータ】

UB	func_code	ファンクションコード

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なファンクションコード
E_ID	未登録のファンクションコード

【解説】

登録したコールバック関数を削除します。

mbs_ena_func_cbk（コールバック関数の有効・無効化）

【書式】

ER mbs_ena_func_cbk(UB func_code, UB enable);

【パラメータ】

UB	func_code	ファンクションコード
UB	enable	1=有効化、0=無効化

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なファンクションコード
E_ID	未登録のファンクションコード

【解説】

登録したコールバック関数の有効・無効化を行います。無効化したものは未登録と同じ状態として扱われます。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_reg_map（各テーブルのバッファ登録）
mbs_reg_map_logi（各テーブルのバッファ登録／論理アドレス指定）

【書式】

ER mbs_reg_map(UB mbmap, UH sta_adr, UH adr_len, VP dat, VP_INT ext);
ER mbu_reg_map_logi(UW sta_logi, UH adr_len, VP dat, VP_INT ext);

【パラメータ】

UB　mbmap	テーブル識別子
UH　sta_adr	紐付け開始アドレス
UH　sta_logi	紐付け開始アドレス（論理アドレス）
VP　dat	指定バッファ
UH　adr_len	紐付けサイズ(ビット・ワード指定)
VP_INT ext	拡張情報

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（mbmap, sta_logi が有効値ではない、dat が NULL、adr_len が 0 など）
E_OBJ	登録アドレス・サイズが既存登録アドレス・サイズと被っているため、登録ができない。
E_NOMEM	紐付け可能なバッファ登録管理情報の空きがない

【解説】

Modbus の各テーブルを指定バッファと紐付けします。各テーブルには複数バッファを紐付け可能です。

テーブル識別子 mbmap には各テーブルの識別子を示す以下が指定できます。

mbmap	テーブル名	備考
MBMAP_INPUT_DISCRETE	Discretes Input	Single bit, ReadOnly
MBMAP_COILS	Coils	Single bit, ReadWrite
MBMAP_INPUT_REGISTERS	Input Registers	16-bit word, ReadOnly
MBMAP_HOLDING_REGISTERS	Holding Registers	16-bit word, ReadWrite

指定バッファ dat の先頭アドレスを各テーブルの開始アドレス sta_adr を指定します。有効な値の範囲は 0x0000-0xFFFF です。有効サイズを adr_len で指定します。ビットを扱うテーブ

Copyright © eForce Co., Ltd. All Rights Reserved

ルの場合はビットサイズを、ワードを扱うテーブルの場合はワードサイズを指定して下さい。

mbs_reg_map_logi()は同じ機能の API ですが、紐付け開始アドレスに論理アドレスが利用出来ます。各テーブルと内部アドレス、論理アドレスの関係は以下表のとおりです。

mbmap	addr (内部アドレス)	logi (論理アドレス)
MBMAP_INPUT_DISCRETE	0x0000-0x270E	00001-09999
MBMAP_COILS	0x0000-0x270E	10001-19999
MBMAP_INPUT_REGISTERS	0x0000-0x270E	30001-39999
MBMAP_HOLDING_REGISTERS	0x0000-0x270E	40001-49999

拡張情報 ext はユーザアプリで自由に利用が出来ます。

本 API をコールすると、mbs_is_valid_addr()、mbs_get_map() API がコールバック関数内で利用可能になります。本 API をコールしても自動的にファンクションコードに応じて指定バッファへの読み書きが発生するわけではないので注意して下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_del_map（各テーブルのバッファ登録解除）
mbs_del_map_logi（各テーブルのバッファ登録解除／論理アドレス指定）

【書式】

ER mbs_del_map(UB mbmap, UH del_adr);
ER mbs_del_map_logi(UH del_logi);

【パラメータ】

UB	mbmap	テーブル識別子
UH	del_adr	削除対象アドレス
UH	del_logi	削除対象アドレス（論理アドレス）

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（mbmap, del_logi が有効値ではない、dat が NULL、adr_len が 0 など）
E_OBJ	削除対象アドレスに登録情報がない

【解説】

ModBus の各テーブルに紐付けられたバッファの登録を解除します。削除対象アドレス del_adr は mbs_reg_map()で登録したアドレス範囲がマッチすれば対象となるバッファ部を紐付けから削除します。

論理アドレス指定で実施する場合、mbs_del_map_logi()を使用して下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbs_get_map（各テーブルのバッファ取得）

【書式】

T_MODBUS_SLAVE_MAP* mbs_get_map(UB func_code, UH addr);

【パラメータ】

UB func_code	ファンクションコード
UH addr	開始アドレス

【戻り値】

T_MODBUS_SLAVE_MAP*	登録バッファのポインタ

【エラーコード】

NULL	指定アドレスに登録バッファがない

【解説】

コールバック関数内で利用出来ます。指定ファンクションコード(func_code)、開始アドレス(addr)に応じた mbs_reg_map()で登録したバッファのポインタを返します。

mbs_get_map_next（隣接後続バッファ取得）

【書式】

T_MODBUS_SLAVE_MAP* mbs_get_map_next(T_MODBUS_SLAVE_MAP *sm)

【パラメータ】

T_MODBUS_SLAVE_MAP *sm	登録バッファのポインタ

【戻り値】

T_MODBUS_SLAVE_MAP*	隣接後続バッファのポインタ

【エラーコード】

NULL	指定登録バッファに隣接後続バッファがない

【解説】

コールバック関数内で利用出来ます。指定登録バッファ(sm)の登録アドレスが隣接する後続バッファのポインタを取得します。要求メッセージの開始アドレス・数量の組み合わせが複数の登録バッファに跨る場合に利用出来ます。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_get_map_req（登録バッファに対する要求メッセージの相対値取得）

【書式】

ER mbs_get_map_req(T_MODBUS_DATA_REQ1*rel,
T_MODBUS_DATA_REQ1 *abs, T_MODBUS_SLAVE_MAP *sm)

【パラメータ】

T_MODBUS_DATA_REQ1*rel	要求メッセージの登録バッファ相対値
T_MODBUS_DATA_REQ1 *abs	要求メッセージ(アドレス情報使用)
T_MODBUS_SLAVE_MAP *sm	登録バッファのポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。（指定引数が NULL）
E_OBJ	要求メッセージのアドレス・数量は登録バッファを指さない。

【解説】

要求メッセージのアドレス・数量(abs->addr, abs->quant)を、指定登録バッファの開始アドレス・サイズ(sm->sta_adr, sm->adr_len)に応じた相対値に変換します。

例を記します。要求メッセージの アドレス=1、数量=6 (abs->addr=1, abs->quant=6) の場合、開始アドレス=1 から 6 個分のデータを取得することになります。登録バッファが A (sm->sta_adr=0, sm->adr_len=3)と B (sm->sta_adr=3, sm->adr_len=4) と 2 つあると定義します。。

ercd = mbs_get_map_req(&rel, &abs, &sm); // API 例

上記条件で登録バッファ(sm)に A、B を指定するとそれぞれ以下となります。

A) 相対アドレス=1、相対数量=2 (rel->addr=1, rel->quant=2)

B) 相対アドレス=0、相対数量=4 (rel->addr=0, rel->quant=4)

取得した相対値は登録バッファへのアクセスするアドレス・数量として利用出来ます。また、要求メッセージのアドレス・数量が登録バッファに該当しない場合(例では登録バッファ C を指定した場合)、E_OBJ を返却します。

Copyright © eForce Co., Ltd. All Rights Reserved

例外として、要求メッセージのファンクションコード(abs->func_code)が 5:Write Single Coil、6:Write Single Register の場合、要求メッセージの数量を 1 で計算します。(abs->quant の位置の値はコイル値・レジスタ値を示すため、そのまま rel->quant に代入します。)

Copyright © eForce Co.,Ltd. All Rights Reserved

cbk_mbs_chkreq（チェック用コールバック関数）

【書式】

ER (*cbk_mbs_chkreq)(T_MODBUS_SLAVE *ms, UB *req);

【パラメータ】

T_MODBUS_SLAVE *ms		ModBus スレーブ制御情報
UB *req		要求メッセージ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

0	チェック正常
0<	例外コード
0>	内部エラーとして判定される。(04:SERVER_DEVICE_FAILURE)

【解説】

要求メッセージ受信後、ファンクション処理の前に行われるチェック処理のコールバック関数です。

要求メッセージ req は PDU の先頭アドレスが入ります。アクセス簡便用に構造体を用意しています。(T_MODBUS_DATA_REQ, T_MODBUS_DATA_REQ2, T_MODBUS_DATA_REQ3) 構造体利用時は req をこれらの構造体ポインタにキャストしてご利用下さい。

関数戻り値に 0 を返すと、チェック正常と見なされファンクション用コールバックの処理に遷移します。0 より上の値を指定すると例外コードとして判定されます。通常は範囲チェックで異常の場合は EC03_ILLEGAL_DATA_VALUE を、値チェックで異常の場合は EC02_ILLEGAL_DATA_ADDRESS を返すようにして下さい。

マイナスの値を返すと内部エラーと見なされ、コール元で例外コードに EC04_SERVER_DEVICE_FAILURE が設定されます。

Copyright © eForce Co., Ltd. All Rights Reserved

cbk_mbs_func（ファンクション用コールバック関数）

【書式】

ER (*cbk_mbs_func)(T_MODBUS_SLAVE *ms, UB *buf);

【パラメータ】

T_MODBUS_SLAVE *ms	ModBus スレーブ制御情報
UB *buf	要求メッセージ、応答メッセージ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

0	チェック正常
0<	例外コード
0>	内部エラーとして判定される。(04:SERVER_DEVICE_FAILURE)

【解説】

要求メッセージ受信後、チェック処理が正常終了した際に呼ばれるファンクション処理のコールバック関数です。

引数 buf には要求メッセージと応答メッセージの共通バッファとして機能します。buf は PDU の先頭アドレスが入ります。
コールバック関数が呼ばれた際、引数 buf を要求メッセージとして解釈してファンクション処理を実行して下さい。その後、引数 buf を応答メッセージとして処理結果を書き込むようにして下さい。
アクセス簡便用に構造体を用意しています。(T_MODBUS_DATA_xxx) 構造体利用時は buf をこれらの構造体ポインタにキャストしてご利用下さい。

関数戻り値に 0 を返すと、正常と見なされコールバック関数内で作成された応答メッセージ buf を送信します。0 より上の値を指定すると例外コードとして判定されます。
マイナスの値を返すとと内部エラーと見なされ、コール元で例外コードに EC04_SERVER_DEVICE_FAILURE が設定されます。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_is_valid_quant（数量の妥当性チェック）

【書式】

ER mbs_is_valid_quant(UB func_code, UH quant)

【パラメータ】

UB	func_code	指定ファンクションコード
UH	quant	数量要素

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	範囲外の値
E_NOSPT	指定ファンクションコードは未対応

【解説】

チェック用コールバック関数内で利用出来ます。要求メッセージの数量要素(Quantity of coils、Quantity of Inputs、Quantity of xxx ..)が有効な範囲内にあるかをチェックします。
範囲内の場合は E_OK、範囲外は E_PAR、指定ファンクションコードが未対応のものを指定した場合は E_NOSPT を返します。

判定処理は以下となります。リスト外のファンクションコードは未対応です。

func_code	quant 判定処理	備考
0x01	0x0001 <= quant <= 0x07D0	
0x02	0x0001 <= quant <= 0x07D0	
0x03	0x0001 <= quant <= 0x007D	
0x04	0x0001 <= quant <= 0x007D	
0x05	(0x0000 == quant) || (0xFF00 == quant)	quant に”Output Value”の値を入れて下さい。
0x06	0x0000 <= quant <= 0xFFFF	quant に”Register Value”の値を入れて下さい。
0x07,0x0B,0x0C	判定なし	無条件で E_OK
0x0F	0x0001 <= quant <= 0x07B0	
0x10	0x0001 <= quant <= 0x007B	

Copyright © eForce Co., Ltd. All Rights Reserved

mbs_is_valid_addr（アドレスの妥当性チェック）

【書式】

ER mbs_is_valid_addr(UB func_code, UH addr, UH quant, UB chk_next);

【パラメータ】

UB	func_code	指定ファンクションコード
UH	addr	開始アドレス
UH	quant	数量要素
UB	chk_next	隣接後続バッファのアドレスチェック有無

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	対応テーブルのアドレスは登録されているが範囲外
E_NOSPT	指定ファンクションコードは未対応
E_OBJ	対応テーブルのアドレスが未登録

【解説】

チェック用コールバック関数内で利用出来ます。要求メッセージの開始アドレス(addr)、数量要素(Quantity of coils、Quantity of Inputs、Quantity of xxx ..)から有効なアドレスへのアクセスかを判断します。ファンクションコードに 5:Write Single Coil、6:Write Single Register を指定した場合、引数 quant は無視されます。(quant=1 として処理します)

引数 chk_next は後続隣接した登録バッファのアドレスを跨いでチェックするかを指定します。0 はチェック無効(バッファを跨ぐ指定は E_PAR になる)、0 以外はチェック有効になります。チェックを有効とした場合、ファンクション用コールバック関数は複数バッファに跨る処理をデザインして下さい。mbs_get_map_next()、mbs_get_map_req()が利用出来ます。

本 API を利用するには予め mbs_reg_map() でバッファ登録をしておく必要があります。mbs_req_map()の引数 sta_adr, adr_len の値が動作に影響します。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbs_upd_buflen（通信バッファサイズの更新）

【書式】

ER mbs_upd_buflen(T_MODBUS_SLAVE *ms, UB pdu_len)

【パラメータ】

T_MODBUS_SLAVE	*ms	ModBus スレーブ制御情報
UB	pdu_len	更新 PDU サイズ（応答メッセージサイズ）

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （*ms が NULL。pdu_len のサイズが不正など）

【解説】

ファンクション用コールバック関数内で利用出来ます。

通信データサイズを更新します。応答メッセージを作成後に呼び出して下さい。

以下主要なファンクションコードでの設定例です。

func_code	pdu_len 設定例	備考
0x01	MBL_PDULEN_RSP_F01 + {Byte Count}	固定長 + 可変長
0x02	MBL_PDULEN_RSP_F02 + {Byte Count}	固定長 + 可変長
0x03	MBL_PDULEN_RSP_F03 + {Byte Count}	固定長 + 可変長
0x04	MBL_PDULEN_RSP_F04 + {Byte Count}	固定長 + 可変長
0x05	MBL_PDULEN_RSP_F05	固定長
0x06	MBL_PDULEN_RSP_F06	固定長
0x0F	MBL_PDULEN_RSP_F15	固定長
0x10	MBL_PDULEN_RSP_F16	固定長

Copyright © eForce Co., Ltd. All Rights Reserved

2. 3 ユーティリティ API [ma][sl]

mbu_addr_to_logi（ModBus アドレス変換 内部→論理）

【書式】

ER mbu_addr_to_logi(UB mbmap, UH addr);

【パラメータ】

UB	mbmap	テーブル識別子
UH	addr	内部アドレス

【戻り値】

ER	ercd	論理アドレスまたはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （mbmap が不正、addr が変換可能範囲にない）

【解説】

ModBus の内部アドレスを論理アドレスに変換します。

各テーブルと内部アドレス、論理アドレスの関係は以下表のとおりです。

mbmap	addr (内部アドレス)	logi (論理アドレス)
MBMAP_INPUT_DISCRETE	0x0000-0x270E	00001-09999
MBMAP_COILS	0x0000-0x270E	10001-19999
MBMAP_INPUT_REGISTERS	0x0000-0x270E	30001-39999
MBMAP_HOLDING_REGISTERS	0x0000-0x270E	40001-49999

Copyright © eForce Co.,Ltd. All Rights Reserved

mbu_logi_to_addr（ModBus アドレス変換 論理→内部）

【書式】

ER mbu_logi_to_addr(UB *mbmap, UH logi);

【パラメータ】

UB	*mbmap	テーブル識別子
UH	logi	論理アドレス

【戻り値】

ER	ercd	内部アドレスまたはエラーコード

【エラーコード】

E_PAR	不正なパラメータが指定された。 （mbmap が NULL、logi が不正）

【解説】

ModBus の論理アドレスを内部アドレスに変換します。

Copyright © eForce Co., Ltd. All Rights Reserved

mbu_copy_bits（ビット単位のコピー処理）

【書式】

UB mbu_copy_bits(UB *dst, UB *src, UH da, UH sa, UH quants);

【パラメータ】

UB *dst	転送先バッファ
UB *src	転送元バッファ
UH da	転送先バッファの開始アドレス
UH sa	転送元バッファの開始アドレス
UH quants	コピービット数

【戻り値】

UB	転送先バッファの終了バイト位置

【解説】

指定バッファ間でビット単位でのコピー処理を行います。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

mbu_copy_words（ワード単位のコピー処理）

【書式】

UB mbu_copy_words(UB *dst, UB *src, UH da, UH sa, UH quants);

【パラメータ】

UB *dst	転送先バッファ
UB *src	転送元バッファ
UH da	転送先バッファの開始アドレス
UH sa	転送元バッファの開始アドレス
UH quants	コピーワード数

【戻り値】

UB	転送先バッファの終了バイト位置

【解説】

指定バッファ間でワード単位でのコピー処理を行います。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbu_write_bit（1 ビット書込）

【書式】

void mbu_write_bit(UB *dst, UH addr, UH val);

【パラメータ】

UB *dst	転送先バッファ
UH addr	書込アドレス
UH val	書込ビット（0 = 0, 0 以外=1）

【解説】

指定バッファ・アドレスのビット値に 0/1 を書き込みます。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

mbu_read_bit（1 ビット読込）

【書式】

UB mbu_read_bit(UB *src, UH addr);

【パラメータ】

UB *src	転送元バッファ
UH addr	読込アドレス

【戻り値】

UB	読込アドレスのビット値（0 or 1）

【解説】

指定バッファ・アドレスのビット値を読み込みます。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbu_write_word（1 ワード書込）

【書式】

void mbu_write_word(UB *dst, UH addr, UH val);

【パラメータ】

UB *dst	転送先バッファ
UH addr	書込アドレス
UH val	書込ワード

【解説】

指定バッファ・アドレスにワード値 val を書き込みます。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

mbu_read_word（1 ワード読込）

【書式】

UH mbu_read_word(UB *src, UH addr);

【パラメータ】

UB *src	転送元バッファ
UH addr	読込アドレス

【戻り値】

UH	読込アドレスのワード値

【解説】

指定バッファ・アドレスのワード値を読み込みます。

本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbu_write_2xword（2 ワード/32 ビット書込）

【書式】

void mbu_write_2xword(UB *dst, UH addr, UW val);

【パラメータ】

UB *dst	転送先バッファ
UH addr	書込アドレス
UW val	書込 2 ワード

【解説】

指定バッファ・アドレスに 2 ワード/32 ビット値 val を書き込みます。
本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

mbu_read_2xword（2 ワード/32 ビット読込）

【書式】

UW mbu_read_2xword(UB *src, UH addr);

【パラメータ】

UB *src	転送元バッファ
UH addr	読込アドレス

【戻り値】

UW	読込アドレスの 2 ワード値

【解説】

指定バッファ・アドレスの 2 ワード/32 ビット値を読み込みます。
本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbu_swap_2xword（2 ワード/32 ビット スワップ処理）

【書式】

UW mbu_swap_2xword(UW s, UB end_swp);

【パラメータ】

UW　s	スワップ元データ
UB　end_swp	スワップ用フラグ

【戻り値】

UW	スワップ後データ

【解説】

指定 2 ワード/32 ビット値 s をスワップ変換して戻り値に返します。

スワップ元データにビッグエンディアン環境で 0x12345678（もしくはリトルエンディアン環境で 0x78563412）の値を入れた場合、フラグ値により以下のようになります。

end_swp	フラグ概要	バイト列（戻り値）
ES_BIGE_NOSWP	ビッグエンディアン	0x12 0x34 0x56 0x78
ES_BIGE_SWP	ビッグエンディアン+スワップあり	0x34 0x12 0x78 0x56
ES_LITE_NOSWP	リトルエンディアン	0x78 0x56 0x34 0x12
ES_LITE_SWP	リトルエンディアン+スワップあり	0x56 0x78 0x12 0x34

未サポートのフラグ値を指定した場合、戻り値は引数 s と同じになります。

μNet3-ModBus マスタのファンクションコード API のメッセージ、もしくは
μNet3-ModBus スレーブのコールバック関数内のメッセージの 2 ワード値(※)に対して本 API を利用する場合、引数 end_swp の値に論理和で ES_DONE_BO を追加して下さい。
※コールバック関数が呼ばれた時点でワード値は既にバイトオーダー変換
(NetworkByteOrder→HostByteOrder)されており、スワップ調整を行います。

Copyright © eForce Co.,Ltd. All Rights Reserved

mbu_write_4xword（4 ワード/64 ビット書込）

【書式】

void mbu_write_4xword(UB *dst, UH addr, UD val);

【パラメータ】

UB *dst	転送先バッファ
UH addr	書込アドレス
UD val	書込 4 ワード

【解説】

指定バッファ・アドレスに 4 ワード/64 ビット値 val を書き込みます。
本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

mbu_read_4xword（4 ワード/64 ビット読込）

【書式】

UD mbu_read_4xword(UB *src, UH addr);

【パラメータ】

UB *src	転送元バッファ
UH addr	読込アドレス

【戻り値】

UD	読込アドレスの 4 ワード値

【解説】

指定バッファ・アドレスの 2 ワード/32 ビット値を読み込みます。
本 API は引数の NULL チェックをしません、必要があればコール元で実施して下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

mbu_swap_4xword（4 ワード/64 ビット スワップ処理）

【書式】

UD mbu_swap_4xword(UD s, UB end_swp);

【パラメータ】

UD　s	スワップ元データ
UB　end_swp	スワップ用フラグ

【戻り値】

UD	スワップ後データ

【解説】

指定 4 ワード/64 ビット値 s をスワップ変換して戻り値に返します。

スワップ元データにビッグエンディアン環境で 0x1122334455667788（もしくはリトルエンディアン環境で 0x8877665544332211） の値を入れた場合、フラグ値により以下のようになります。

end_swp	フラグ概要	バイト列（戻り値）
ES_BIGE_NOSWP	ビッグエンディアン	0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
ES_BIGE_SWP	ビッグエンディアン+スワップあり	0x22 0x11 0x44 0x33 0x66 0x55 0x88 0x77
ES_LITE_NOSWP	リトルエンディアン	0x88 0x77 0x66 0x55 0x44 0x33 0x22 0x11
ES_LITE_SWP	リトルエンディアン+スワップあり	0x77 0x88 0x55 0x66 0x33 0x44 0x11 0x22

未サポートのフラグ値を指定した場合、戻り値は引数 s と同じになります。

μNet3-ModBus マスタのファンクションコード API のメッセージ、もしくは
μNet3-ModBus スレーブのコールバック関数内のメッセージの 2 ワード値(※)に対して本 API を利用する場合、引数 end_swp の値に論理和で ES_DONE_BO を追加して下さい。
※コールバック関数が呼ばれた時点でワード値は既にバイトオーダー変換
(NetworkByteOrder→HostByteOrder)されており、スワップ調整を行います。

Copyright © eForce Co.,Ltd. All Rights Reserved

2. 4 マスタ 設定マクロ／構造体 [ma]

T_MODBUS_MASTER (ModBus マスタ制御情報)

```
typedef struct t_modbus_master {
    UH type;                    [in] 動作タイプ
    UH id;                      [in] ソケット ID or デバイス ID
    UH flag;                    制御フラグ
    UH buf_len;                 作業バッファサイズ
    UB buf[260];                作業バッファ
    enum modbus_state step;     動作状態
    VP com;                     [in] マスタ制御情報–拡張部(COM or TCP)
} T_MODBUS_MASTER;
```

ModBus スレーブの制御用構造体です。

T_MODBUS_MASTER_COM (ModBus マスタ制御情報 – COM 部)

```
typedef struct t_modbus_master_com {
    UW baud_rate;               [in] COM ボーレート
    UH chr_tmo;                 1.5T タイムアウト時間(us)
    UH msg_tmo;                 3.5T タイムアウト時間(us)
    UH byt_tmo;                 1T タイムアウト時間(us)
    UB tmr_ch;                  [in] タイマドライバチャンネル
    UB slave_id;                [in] 宛先スレーブ ID
    void (*cbk_sw_tr)(UB ena_tx);   [in] 送受信切替処理コールバック
} T_MODBUS_MASTER_COM;
```

ModBus スレーブの RTU 部の制御用構造体です。RS485 動作時は送受信切替処理コールバック(cbk_sw_tr)で送受信の切替ピンの制御を実施して下さい。RS232 動作時は本コールバックは NULL で問題ありません。

Copyright © eForce Co., Ltd. All Rights Reserved

T_MODBUS_MASTER_TCP (ModBus マスタ制御情報 – TCP 部)

```
typedef struct t_modbus_master_tcp {
    T_NODE svr;                 [in] 接続先 ModBus スレーブ情報
    UH trans_id;                トランザクション ID
    UB unit_id;                 ユニット ID
} T_MODBUS_MASTER_TCP;
```

ModBus スレーブの TCP 部の制御用構造体です。

ModBus マスタ設定ファイル (modbus_master_cfg.h)

Modbus_master_cfg.h		
#define CFG_MM_USE_TCP	1	/* use ModBus/TCP */
#define CFG_MM_USE_COM	1	/* use ModBus/RTU */

CFG_MM_USE_TCP

ModBus マスタの TCP 処理部の有効・無効化設定です。(0:無効、0 以外:有効)

ModBus/TCP 使用時は有効にして下さい。

CFG_MM_USE_COM

ModBus マスタの COM 処理部の有効・無効化設定です。(0:無効、0 以外:有効)

ModBus/RTU 使用時は有効にして下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

ファンクションコード実行 API 別名

ファンクションコード実行 API には別名マクロを用意しています。

```
#define mbm_F01_read_coils                  mbm_F01
#define mbm_F02_read_discrete_inputs        mbm_F02
#define mbm_F03_read_holding_registers      mbm_F03
#define mbm_F04_read_inputs_registers       mbm_F04
#define mbm_F05_write_single_coil           mbm_F05
#define mbm_F06_write_single_register       mbm_F06
#define mbm_F15_write_multiple_coils        mbm_F15
#define mbm_F16_write_multiple_registers    mbm_F16
```


Copyright © eForce Co., Ltd. All Rights Reserved

2. 5 スレーブ 設定マクロ／構造体 [sl]

T_MODBUS_SLAVE (ModBus スレーブ制御情報)

```
typedef struct t_modbus_slave {
    UH type;                        [in] 動作タイプ
    UH id;                          [in] ソケット ID or デバイス ID
    UH flag;                        制御フラグ
    UH buf_len;                     作業バッファサイズ
    UB buf[260];                    作業バッファ
    enum modbus_state step;         動作状態
    VP com;                         [in] スレーブ制御情報 – COM 部
} T_MODBUS_SLAVE;
```

ModBus スレーブの制御用構造体です。

T_MODBUS_SLAVE_COM (ModBus スレーブ制御情報 – COM 部)

```
typedef struct t_modbus_slave_com {
    UW baud_rate;                   [in] COM ボーレート
    UH chr_tmo;                     1.5T タイムアウト時間(us)
    UH msg_tmo;                     3.5T タイムアウト時間(us)
    UH byt_tmo;                     1T タイムアウト時間(us)
    UB tmr_ch;                      [in] タイマドライバチャンネル
    UB slave_id;                    [in] 自スレーブ ID
    void (*cbk_sw_tr)(UB ena_tx);   [in] 送受信切替処理コールバック
} T_MODBUS_SLAVE_COM;
```

ModBus スレーブの RTU 部の制御用構造体です。RS485 動作時は送受信切替処理コールバック(cbk_sw_tr)で送受信の切替ピンの制御を実施して下さい。RS232 動作時は本コールバックは NULL で問題ありません。

T_MODBUS_SLAVE_MAP (ModBus スレーブ テーブルマップ情報)

```
typedef struct t_modbus_slave_map {
    UB *dat;                        バッファ
```


Copyright © eForce Co.,Ltd. All Rights Reserved

```
    VP_INT ext;                 拡張情報
    UH sta_adr;                 開始アドレス
    UH adr_len;                 開始アドレスからの有効サイズ
    UB mbmap;                   テーブル識別子
} T_MODBUS_SLAVE_MAP;
```

mbs_get_map()で取得される構造体、mbs_reg_map()で登録したバッファ情報が入ります。
adr_len はテーブル識別子によってビット単位サイズ、ワード単位サイズと変わります。

Copyright © eForce Co., Ltd. All Rights Reserved

ModBus スレーブ設定ファイル (modbus_slave_cfg.h)

```
Modbus_slave_cfg.h
#define CFG_MS_USE_TCP          1       /* use ModBus/TCP */
#define CFG_MS_USE_COM          1       /* use ModBus/RTU */

#define CFG_MS_NUM_FUNCCODE     10
#define CFG_MS_NUM_SLAVEMAP     10
```

CFG_MS_USE_TCP

ModBus スレーブの TCP 処理部の有効・無効化設定です。(0:無効、0 以外:有効)

ModBus/TCP 使用時は有効にして下さい。

CFG_MS_USE_COM

ModBus スレーブの COM 処理部の有効・無効化設定です。(0:無効、0 以外:有効)

ModBus/RTU 使用時は有効にして下さい。

CFG_MS_NUM_FUNCCODE

ユーザアプリが mbs_reg_func_cbk()で登録可能なファンクションコードの数です。

mbs_reg_func_cbk ()が E_NOMEM を返す場合、この値を適宜増やして下さい。

CFG_MS_NUM_SLAVEMAP

ユーザアプリが mbs_reg_map()で登録可能な紐付けバッファ数です。

mbs_reg_map()が E_NOMEM を返す場合、この値を適宜増やして下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

第3章　コンフィグレーション

μNet3/ModBus はコンフィグレータでのソース生成に対応していないため、プロジェクトへのソース追加やリソース追加の作業が別途必要となります。本章で解説します。

3. 1　μNet3-ModBus マスタ [ma]

3. 1. 1　プロジェクト導入手順

① 「uC3/Network/ModBus/」フォルダを使用プロジェクトにコピー
μNet3-ModBus ファイルが含まれるフォルダを利用プロジェクトのソース格納先にコピーします。

② プロジェクト追加：「{コピー先}/ModBus/src/*.c」
利用プロジェクトに「{コピー先}/ModBus/src/*.c」のソースファイルを追加します。
(「1. 4　μNet3-ModBus 構成ファイル」の **ma** 列に●がある項目を追加する。)

③ プロジェクト追加：「{コピー先}/ModBus/hal/{CPU}/*.c」
利用プロジェクトに CPU が合致する CPU 依存タイマドライバを追加します。

④ プロジェクトのインクルード設定追加
以下 ModBus フォルダ配下にプロジェクトのインクルードパスを通します
・{コピー先}/ModBus/inc/
・{コピー先}/ModBus/cfg/

⑤ ModBus マスタ設定ファイルの内容変更
以下パスの設定ファイルをユーザアプリに合ったものに変更します。
・{コピー先}/ModBus/cfg/modbus_master_cfg.h

⑥ マスタ用リソースの準備
動作させるため、「3. 1. 2 マスタ用リソースの準備」を行います。

⑦ ユーザアプリで ModBus マスタ API を利用する。
以下パスの ModBus サンプルソースが API 利用の参考となります。
・{コピー先}/ModBus/cfg/modbus_master_user.c

項番①～④まで進めるとビルド可能となります。ビルドが問題なければ項番⑤～⑦まで進めて下さい。項番⑦後、設定内容に問題がなければ ModBus マスタとして動作します。

Copyright © eForce Co., Ltd. All Rights Reserved

3.1.2 マスタ用リソースの準備

① TCP ソケット追加（TCP 動作のみ）
ModBus/TCP 動作時に使う TCP ソケットを用意します。コンフィグレータもしくは API(cre_soc)でソケット生成を行います。タイムアウト値は環境に応じて変更出来ます。サンプルでは以下のソケットパラメータを使用します。

ソケット ID	デバイス番号	IP Ver	プロトコル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_MBM	0	v4	TCP	ANY	-1	-1	-1	-1	1024	1024

② シリアルデバイス追加（RTU 動作のみ）
ModBus/RTU 動作時に使うシリアルデバイスを用意します。コンフィグレータよりシリアルデバイスの選択もしくは API で対象デバイス ID を使用可能にします。

③ タイマドライバの ISR 追加(uC3/Compact 利用時、RTU 動作のみ)
ModBus/RTU 動作時に使うタイマドライバの割り込みサービスルーチン(ISR)をコンフィグレータに登録します。uC3/Standard 利用時はタイマドライバの初期化処理で API(cre_isr)で生成するので本処理は不要です。

3.1.3 マスタ処理開始までの流れ

マスタ処理の実装の流れを以下に説明します。

① ModBus/TCP 動作初期化（TCP 動作のみ）
ModBus マスタ制御情報(T_MODBUS_MASTER *mm)に以下を指定する
- mm->type: MODBUS_TYPE_TCP
- mm->id: 生成済みの TCP ソケット ID(3.1.2-①)
- mm->com: TCP 部制御情報（modbus_master()解説を参照）

② ModBus/RTU 動作初期化（RTU 動作のみ）
ModBus スレーブ制御情報(T_MODBUS_SLAVE *ms)に以下を指定する
- ms->type: MODBUS_TYPE_RTU
- ms->id: ini_com()で初期化済みのデバイス ID(3.1.2-②)
- ms->com: COM 部制御情報（modbus_master()解説を参照）

③ マスタ処理開始 実行
modbus_master()を実行すると、その他マスタ用 API が利用可能となります。

実装詳細は「第 4 章 **サンプルソースコード**」をご参照下さい。

Copyright © eForce Co.,Ltd. All Rights Reserved

3. 2 μNet3-ModBus スレーブ [sl]

3. 2. 1 プロジェクト導入手順

① 「uC3/Network/ModBus/」フォルダを使用プロジェクトにコピー
μNet3-ModBus ファイルが含まれるフォルダを利用プロジェクトのソース格納先にコピーします。

② プロジェクト追加：「{コピー先}/ModBus/src/*.c」
利用プロジェクトに「{コピー先}/ModBus/src/*.c」のソースファイルを追加します。
(「1. 4 μNet3-ModBus 構成ファイル」の **sl** 列に●がある項目を追加する。)

③ プロジェクト追加：「{コピー先}/ModBus/hal/{CPU}/*.c」
利用プロジェクトに CPU が合致する CPU 依存タイマドライバを追加します。

④ プロジェクトのインクルード設定追加
以下 ModBus フォルダ配下にプロジェクトのインクルードパスを通します
・{コピー先}/ModBus/inc/
・{コピー先}/ModBus/cfg/

⑤ ModBus スレーブ設定ファイルの内容変更
以下パスの設定ファイルをユーザアプリに合ったものに変更します。
・{コピー先}/ModBus/cfg/modbus_slave_cfg.h

⑥ スレーブ用リソースの準備
動作させるため、「3. 2. 2 スレーブ用リソースの準備」を行います。

⑦ ユーザアプリに ModBus スレーブタスクを実装する。
以下パスの ModBus サンプルソースが実装の参考となります。
・{コピー先}/ModBus/cfg/modbus_slave_user.c

項番①～④まで進めるとビルド可能となります。ビルドが問題なければ項番⑤～⑦まで進めて下さい。項番⑦後、設定内容に問題がなければ ModBus スレーブとして動作します。

Copyright © eForce Co., Ltd. All Rights Reserved

3. 2. 2　スレーブ用リソースの準備

① タスク追加

modbus_slave() API をコールするタスクを用意します。コンフィグレータもしくは API(cre_tsk)でタスク生成を行います。タスク ID／関数名は任意で設定します。modbus_slave()コールタスクから登録コールバック関数は呼ばれるため、スタックサイズはそちらを考慮したサイズを適宜設定して下さい。

1 タスク 1 チャンネル動作のため、複数チャンネルを使用する場合は複数定義して下さい。サンプルでは以下 ID／関数名を使用します。

No	ID	関数名	ｽﾀｯｸｻｲｽﾞ	備考
1	ID_MBS_TCP_TSK	modbus_slave_tcp_task	1024	TCP 動作
2	ID_MBS_COM_TSK	modbus_slave_com_task	1024	RTU 動作

② TCP ソケット追加 (TCP 動作のみ)

ModBus/TCP 動作時に使う TCP ソケットを用意します。コンフィグレータもしくは API(cre_soc)でソケット生成を行います。ポート番号 502 は ModBus スレーブの既定値です。タイムアウト値は環境に応じて変更出来ます。サンプルでは以下のソケットパラメータを使用します。

ソケット ID	デバイス番号	IP Ver	プロトコル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_MBS	0	v4	TCP	502	-1	-1	-1	-1	1024	1024

③ シリアルデバイス追加 (RTU 動作のみ)

ModBus/RTU 動作時に使うシリアルデバイスを用意します。コンフィグレータよりシリアルデバイスの選択もしくは API で対象デバイス ID を使用可能にします。

④ タイマドライバの ISR 追加(uC3/Compact 利用時、RTU 動作のみ)

ModBus/RTU 動作時に使うタイマドライバの割り込みサービスルーチン(ISR)をコンフィグレータに登録します。uC3/Standard 利用時はタイマドライバの初期化処理で API(cre_isr)で生成するので本処理は不要です。

Copyright © eForce Co.,Ltd. All Rights Reserved

3. 2. 3　スレーブ処理開始までの流れ

スレーブ処理開始(modbus_slave)の実行前までの流れを以下に説明します。

① コールバック関数登録（mbs_reg_func_cbk）
要求メッセージのファンクションコード毎に応じたユーザコールバック関数を登録します。コールバック関数はメッセージチェック用、ファンクション実行用の 2 種類登録が可能です。mbs_reg_func_cbk()を利用します。

② 各テーブルモデルのバッファ登録（mbs_reg_map）
ModBus 各テーブル(Discretes Input, Coils, Input Registers, Holding Registers)のアドレスに対してバッファ割当を行います。mbs_reg_map()を利用します。

③ ModBus/TCP 動作初期化（TCP 動作のみ）
ModBus スレーブ制御情報(T_MODBUS_SLAVE *ms)に以下を指定する
- ms->type:　MODBUS_TYPE_TCP
- ms->id:　生成済みの TCP ソケット ID(3.2.2-②)

④ ModBus/RTU 動作初期化（RTU 動作のみ）
ModBus スレーブ制御情報(T_MODBUS_SLAVE *ms)に以下を指定する
- ms->type:　MODBUS_TYPE_RTU
- ms->id:　ini_com()で初期化済みのデバイス ID(3.2.2-③)
- ms->com:　COM 部制御情報（modbus_slave()解説を参照）

⑤ スレーブ処理開始 実行
modbus_slave()を実行すると、設定に応じた動作をします。

実装詳細は「第 4 章 **サンプルソースコード**」をご参照下さい。

Copyright © eForce Co., Ltd. All Rights Reserved

第4章　サンプルソースコード

4.1 µNet3-ModBus マスタ [ma]

サンプルソースコード modbus_master_user.c は API 使用方法を理解するのに役立ちます。

modbus_master_user.c 処理概要

- サンプル動作

コンソール画面のコマンドベースで実行できる ModBus マスタが動作します。ファンクションコード 1,2,3,4,5,6,15,16 に対応しています。

サンプル ModBus マスタ コマンド一覧

```
Shell> ?
 mb           -> start ModBus/TCP
 mbc          -> start ModBus/RTU
 mb_exit      -> stop ModBus
 mb_f01       -> mb_func 01:ReadCoilse
 mb_f02       -> mb_func 02:Read Discrete Inputs
 mb_f03       -> mb_func 03:Read Holding Registers
 mb_f04       -> mb_func 04:Read Input Registers
 mb_f05       -> mb_func 05:Write Single Coil
 mb_f06       -> mb_func 06:Write Single Register
 mb_f15       -> mb_func 15:Write Multiple Coils
 mb_f16       -> mb_func 16:Write Multiple registers
 mb_read32    -> Display last response as 32 bit data
 mb_read64    -> Display last response as 64 bit data
 E_OK (0)
```

- 本ファイルの関数とコマンドの対応は以下となります。

a. shell_usr_cmd_mb()　“mb”　ModBus/TCP マスタの開始
　“mbc”　ModBus/RTU マスタの開始
b. shell_usr_cmd_mb_exit()　“mb_exit”　ModBus マスタの停止
c. shell_usr_cmd_mb_req()　“mb_f01”, “mb_f02” … “mb_f16”
　各ファンクションコード実行
d. shell_usr_cmd_mb_read()　“mb_read32”, “mb_read64”
　応答メッセージのデータを 32/64bit 表示

Copyright © eForce Co.,Ltd. All Rights Reserved

ソース解説: サンプル動作設定

modbus_master_user.c – 動作設定抜粋

```
/* ModBus sample configuration */
#define USE_MODBUS_TCP          1
#define USE_MODBUS_RTU          1

#if (USE_MODBUS_RTU)       /* ModBus/RTU settings */
#include "device_id.h"
#include "DDR_COM.h"

/* settings */
#define SPCFG_COM_BAUDRATE    115200U    /* COM baudrate */
#define SPCFG_TIMER_CH          1U                /* Use timer ch */

/* Serial settings */
const T_COM_SMOD rtu_com_ini = {SPCFG_COM_BAUDRATE, BLEN8, PAR_NONE, SBIT1,
FLW_NONE};


/* evaluation board defines */
#define MBS_BOARD_TS_RX72M_COM    1           /* RX72M: TS-RX72M-COM */

/* Use board settings */
#define MBS_BOARD     MBS_BOARD_TS_RX72M_COM
```

サンプルファイルの先頭部に動作設定があり、動作変更が行えます。(提供サンプルは既に動作する設定値となっており、変更不要で動作確認できます)

USE_MODBUS_TCP (USE_MODBUS_RTU)

ModBus マスタの TCP 動作(RTU 動作)を有効にします。

SPCFG_COM_BAUDRATE

RTU 動作時のボーレート設定です。ボーレート以外は rtu_com_ini 変数で設定します。

SPCFG_TIMER_CH

RTU 動作時のタイマドライバのチャンネル設定です。

MBS_BOARD

プリセットされた評価ボード用の処理を有効にします。提供サンプルで値が変わります。

Copyright © eForce Co., Ltd. All Rights Reserved

コマンド解説: ModBus マスタ開始 - mb, mbc

サンプル ModBus マスタ コマンド例
書式) mb <server_ipa> // ModBus/TCP スレーブ IP 指定 Shell> mb 192.168.68.89 E_OK (0) 書式) mbc <slave_id> // ModBus/RTU スレーブ ID 指定 Shell> mbc 1 E_OK (0)

ModBus マスタを TCP or RTU 動作を開始するコマンドです。TCP 動作時は対向スレーブが存在しないとコマンドは失敗します。サンプルでは TCP/RTU 動作は排他となり、片方の動作中はもう片方は動作しません。各 ModBus コマンドを実行する前に呼び出す必要があります。
対応する shell_usr_cmd_mb()関数内では、modbus_master()を呼び出しています。

コマンド解説: ModBus マスタ停止 - mb_exit

サンプル ModBus マスタ コマンド例
書式) mb_exit // ModBus マスタ停止 Shell> mb_exit E_OK (0)

ModBus マスタを停止するコマンドです。
対応する shell_usr_cmd_mb_exit()関数内では、modbus_master_stop()を呼び出しています。

Copyright © eForce Co.,Ltd. All Rights Reserved

コマンド解説: ファンクションコード実行 - mb_fxx

サンプル **ModBus** マスタ コマンド例

```
書式)    mb_f01 <addr> <num of coils>          // ReadCoils
         mb_f02 <addr> <num of inputs>         // ReadDiscreteInputs
         mb_f03 <addr> <num of regs>           // ReadHoldingRegisters
         mb_f04 <addr> <num of regs>           // ReadInputRegisters
         mb_f05 <addr> <value>                 // WriteSingleCoil
         mb_f06 <addr> <value>                 // WriteSingleRegister
         mb_f15 <addr> <num of outputs> <val1> ... <valN>  // WriteMultipleCoils
         mb_f16 <addr> <num of regss> <val1> ... <valN>    // WriteMultipleRegisters


Shell> mb_f01 1 5           // addr=1, coils=5
func_code: 129 (0x81)       // 対向側で利用出来ない場合、例外となる
exp_code : 0x01, illegal function

Shell> mb_f03 0 4           // addr=0, regs=4
func_code: 03 (0x03)        // 対向側で利用可能な場合、正常な応答
byte count : 8
register value:
 0x0064 0x00c8 0x012c 0x0190
```

対向 ModBus スレーブにファンクションコードを発行するコマンドです。
対応する shell_usr_cmd_mb_req()関数内では、mbm_F01(), mbm_F02(), …, mbm_F16()を呼び出しています。
この関数は API へのデータ入力方法および、応答データ表示関数 shell_printf_modbus_rsp()は応答データのアクセス方法の参考になります。

Copyright © eForce Co., Ltd. All Rights Reserved

コマンド解説: データ 32/64bit 表示 - mb_read32/64

サンプル **ModBus** マスタ コマンド例

```
Shell> mb_f03 0 4            // コマンド前に予め応答データを取得する
func_code: 03 (0x03)
byte count : 8
register value:
 0x0064 0x00c8 0x012c 0x0190

Shell> mb_read32             // 応答データを 32bit データとして表示する
func_code: 03 (0x03)
byte count : 8
register (32bit):
Addr    ES_BIGE_NOSWP   ES_BIGE_SWP     ES_LITE_NOSWP   ES_LITE_SWP
0           6553800(0x6400C8)           1677772800(0x6400C800)    3355468800(0xC8006400)
13107300(0xC80064)
2           19661200(0x12C0190)         738299905(0x2C019001)     2415995905(0x90012C01)
26214700(0x190012C)

Shell> mb_read64             // 応答データを 64bit データとして表示する
func_code: 03 (0x03)
byte count : 8
register (64bit):
Addr    ES_BIGE_NOSWP   ES_BIGE_SWP     ES_LITE_NOSWP   ES_LITE_SWP
0     28148356684186000(0x6400C8012C0190)       7205979306856648705(0x6400C8002C019001)
10376623402600391680(0x90012C01C8006400)  112591279187558500(0x190012C00C80064)
```

mb_f03,mb_f04 コマンドで取得した応答メッセージ内のレジストリデータを 32bit もしくは 64bit で表示するコマンドです。(32bit = mb_read32、64bit = mb_read64)
この関数はレジストリデータを 32bit もしくは 64bit でアクセスする方法の参考になります。

Copyright © eForce Co.,Ltd. All Rights Reserved

4. 2 μNet3-ModBus スレーブ [sl]

サンプルソースコード modbus_slave_user.c は API 使用方法を理解するのに役立ちます。

modbus_slave_user.c 処理概要

- サンプル動作

ファンクションコード 1,2,3,4,5,6,15,16 に対応した ModBus スレーブが動作します。本サンプルは ModBus 各テーブルのバッファをいくつかのパターンで登録しています。

- 本ファイルは次の主要関数で構成されています。

e. modbus_slave_tcp_task() modbus スレーブタスク(TCP 動作)
f. modbus_slave_com_task() modbus スレーブタスク(RTU 動作)
g. user_ini_modbus_slave() modbus スレーブ初期化処理
h. cbk_chk_req() 共通チェック用コールバック関数
i. cbk_func_req() 共通ファンクションコード用コールバック関数

j. 各ファンクションコード用処理関数
   - F01_read_coils() (0x01) Read Coils
   - F02_read_discrete_inputs() (0x02) Read Discrete Inputs
   - F03_read_holding_registers() (0x03) Read Holding Registers
   - F04_read_inputs_registers() (0x04) Read Input Registers
   - F05_write_single_coil() (0x05) Write Single Coil
   - F06_write_single_register() (0x06) Write Single Register
   - F15_write_multiple_coils() (0x0F) Write Multiple Coils
   - F16_write_multiple_registers () (0x10) Write Multiple registers

k. cbk_chk_req_hr() HoldingRegisters チェック用コールバック関数
l. cbk_func_req_hr() HoldingRegisters ファンクション用コールバック関数
   32bit、64bit ワードアクセス用サンプル処理

Copyright © eForce Co., Ltd. All Rights Reserved

ソース解説: サンプル動作設定

modbus_slave_user.c – 動作設定抜粋

```
/* ModBus sample configuration */
#define USE_MODBUS_TCP          1
#define USE_MODBUS_RTU          1

#if (USE_MODBUS_RTU)       /* ModBus/RTU settings */
#include "device_id.h"
#include "DDR_COM.h"

/* settings */
#define SPCFG_SLAVE_ID          1U              /* slave id */
#define SPCFG_COM_BAUDRATE    115200U   /* COM baudrate */
#define SPCFG_TIMER_CH          1U               /* Use timer ch */

/* Serial settings */
const T_COM_SMOD rtu_com_ini = {SPCFG_COM_BAUDRATE, BLEN8, PAR_NONE, SBIT1,
FLW_NONE};


/* evaluation board defines */
#define MBS_BOARD_TS_RX72M_COM    1          /* RX72M: TS-RX72M-COM */

/* Use board settings */
#define MBS_BOARD     MBS_BOARD_TS_RX72M_COM
```

サンプルファイルの先頭部に動作設定があり、動作変更が行えます。(提供サンプルは既に動作する設定値となっており、変更不要で動作確認できます)

USE_MODBUS_TCP (USE_MODBUS_RTU)

ModBus スレーブの TCP 動作タスク(RTU 動作タスク)を有効にします。

SPCFG_SLAVE_ID

RTU 動作時の自スレーブ ID の設定です。

SPCFG_COM_BAUDRATE

RTU 動作時のボーレート設定です。ボーレート以外は rtu_com_ini 変数で設定します。

SPCFG_TIMER_CH

RTU 動作時のタイマドライバのチャンネル設定です。

MBS_BOARD

プリセットされた評価ボード用の処理を有効にします。提供サンプルで値が変わります。

Copyright © eForce Co.,Ltd. All Rights Reserved

ソース解説: modbus スレーブ初期化処理 - コールバック関数登録

ファンクションコードに対応したコールバック関数の登録を行います。

modbus_slave_user.c – 抜粋

```
void user_ini_modbus_slave()
{
…中略…
    /* コールバック登録 (チェック用、ファンクション実行用) */
    ercd = mbs_reg_func_cbk((UB)F01_READ_COILS,                cbk_chk_req, cbk_func_req);
    ercd = mbs_reg_func_cbk((UB)F02_READ_DISCRETE_INPUTS,      cbk_chk_req, cbk_func_req);
    ercd = mbs_reg_func_cbk((UB)F03_READ_HOLDING_REGISTERS,
            cbk_chk_req_hr, cbk_func_req_hr);
    ercd = mbs_reg_func_cbk((UB)F04_READ_INPUT_REGISTERS,      cbk_chk_req, cbk_func_req);
    ercd = mbs_reg_func_cbk((UB)F05_WRITE_SINGLE_COIL,         cbk_chk_req, cbk_func_req);
    ercd = mbs_reg_func_cbk((UB)F06_WRITE_SINGLE_REGISTER,
            cbk_chk_req_hr, cbk_func_req_hr);
    ercd = mbs_reg_func_cbk((UB)F15_WRITE_MULTIPLE_COILS,      cbk_chk_req, cbk_func_req);
    ercd = mbs_reg_func_cbk((UB)F16_WRITE_MULTIPLE_REGISTERS,
            cbk_chk_req_hr, cbk_func_req_hr);
    dbg_assert(E_OK == ercd);
…中略…
```

mbs_reg_func_cbk()を使い、ファンクションコードとコールバック関数を紐づけます。サンプルコードでは Holding Registers テーブルが対象のファンクションコード(3,6,16)以外は共通にしており、同じ関数を登録しています。

ソース解説: modbus スレーブ初期化処理 - 各テーブルのバッファ割当

各テーブルアドレスに対応したバッファの登録を行っています。

modbus_slave_user.c – 抜粋

```
UB gMBUF_common[16] = {0};
UB gMBUF_di[4] = {0x12, 0x34, 0x56, 0x78};
UB gMBUF_co[8] = {0};
UH gMBUF_ir[4] = {0x1234, 0x5678, 0x9abc, 0xdef0};
UH gMBUF_hr[8] = {0};
…中略…

void user_ini_modbus_slave()
{
…中略…
/** データモデルの登録 **/
/* 各テーブルに同じバッファの割当 */
bylen = (UH)sizeof(gMBUF_common);          /* di,co,ir: 0x0000 - 0x007F */
ercd = mbs_reg_map((UB)MBMAP_INPUT_DISCRETE,    0U, CNVL_BT(bylen), gMBUF_common, 0);
ercd = mbs_reg_map((UB)MBMAP_COILS,             0U, CNVL_BT(bylen), gMBUF_common, 0);
ercd = mbs_reg_map((UB)MBMAP_INPUT_REGISTERS, 0U, CNVL_WD(bylen), gMBUF_common, 0);
```


Copyright © eForce Co., Ltd. All Rights Reserved

```
/*  各テーブルに異なるバッファの割当  */
bylen = sizeof(gMBUF_di);                   /* di: 0x0080 - 0x009F */
ercd = mbs_reg_map(MBMAP_INPUT_DISCRETE,       0x0080, CNVL_BT(bylen), gMBUF_di, 0);
bylen = sizeof(gMBUF_co);                    /* co: 0x0090 - 0x00CF (no use 0x0080-0x008F) */
ercd = mbs_reg_map(MBMAP_COILS,                      0x0090, CNVL_BT(bylen), gMBUF_co, 0);
bylen = sizeof(gMBUF_ir);                   /* ir: 0x0008 - 0x000B */
ercd = mbs_reg_map(MBMAP_INPUT_REGISTERS,     0x0008, CNVL_WD(bylen), gMBUF_ir, 0);

/* HoldingRegister用バッファをエンディアン設定別にアクセスするアドレスの割り当て   */
bylen = (UH)sizeof(gMBUF_hr);
mbmap = (UB)MBMAP_HOLDING_REGISTERS;
/*  通常アクセス用領域  */
ercd = mbs_reg_map(mbmap, 0x0000U, CNVL_WD(bylen), gMBUF_hr, 0);
/* 32bitアクセス) hr: 0x00X0 - 0x0XF --> X=1:Big, X=2:Big&Swap, X=3:Little, X=4:Little&Swap   */
ercd = mbs_reg_map(mbmap, 0x0010U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0020U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0030U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0040U, CNVL_WD(bylen), gMBUF_hr, 0);
/* 64bitアクセス) hr: 0x01X0 - 0x1XF --> X=1:Big, X=2:Big&Swap, X=3:Little, X=4:Little&Swap   */
ercd = mbs_reg_map(mbmap, 0x0110U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0120U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0130U, CNVL_WD(bylen), gMBUF_hr, 0);
ercd = mbs_reg_map(mbmap, 0x0140U, CNVL_WD(bylen), gMBUF_hr, 0);
```

本サンプルでは ModBus 各テーブルのアドレスに対して、グローバル変数のバッファ(gMBUF_xxx)を割り当てています。上記コードでは以下の割当になります。

テーブル名	アドレス	割当バッファ／備考
Discretes Input	0x0000 – 0x007F	gMBUF_common
Discretes Input	0x0080 – 0x009F	gMBUF_di
Coils	0x0000 – 0x007F	gMBUF_common
Coils	0x0090 – 0x00CF	gMBUF_co
Input Registers	0x0000 – 0x0007	gMBUF_common
Input Registers	0x0008 – 0x000B	gMBUF_ir
Holding Registers	0x0000 – 0x0000F	gMBUF_hr
Holding Registers	0x0010 – 0x0001F	gMBUF_hr 32bit アクセス用 BigEndian
Holding Registers	0x0020 – 0x0002F	〃 用 BigEndian & Swap
Holding Registers	0x0030 – 0x0003F	〃 用 LittleEndian
Holding Registers	0x0040 – 0x0004F	〃 用 LittleEndian & Swap
Holding Registers	0x0110 – 0x0011F	gMBUF_hr 64bit アクセス用 BigEndian
Holding Registers	0x0120 – 0x0012F	〃 用 BigEndian & Swap
Holding Registers	0x0130 – 0x0013F	〃 用 LittleEndian
Holding Registers	0x0140 – 0x0014F	〃 用 LittleEndian & Swap

Copyright © eForce Co.,Ltd. All Rights Reserved

ソース解説: modbus スレーブタスク(TCP 動作)

ModBus/TCP のスレーブ動作を定義しています。

modbus_slave_user.c – 抜粋

```
#if USE_MODBUS_TCP          /* modbus/TCP */
void modbus_slave_tcp_task(VP_INT exinf)
{
      T_MODBUS_SLAVE *ms;
      ER ercd;
#ifndef NET_C_OS
      T_NODE nod = {0U};
#endif

      ms = &g_ms[G_MS_IDX_TCP];
/*  ★1 ModBusスレーブ TCP設定  */
#ifdef NET_C_OS
      ms->type = MODBUS_TYPE_TCP;
      ms->id = (UH)ID_MODBUS_SOC;    /* socket id */
      ercd = cfg_soc(ms->id, SOC_PRT_LOCAL, (VP)MODBUS_TCP_PORT);
      dbg_assert((0 == ercd));
#else
      /* create TCP socket */
      nod.ver = IP_VER4;
      nod.port = MODBUS_TCP_PORT;
      ercd = cre_soc(IP_PROTO_TCP, &nod);
      dbg_assert((0 < ercd));

      /* socket Timeout settings */
      (void)cfg_soc(ercd, SOC_TMO_CON, (VP)5000);
      (void)cfg_soc(ercd, SOC_TMO_SND, (VP)5000);
      (void)cfg_soc(ercd, SOC_TMO_RCV, (VP)-1);
      (void)cfg_soc(ercd, SOC_TMO_CLS, (VP)5000);
      ms->type = MODBUS_TYPE_TCP;
      ms->id = ercd;            /* socket id */
#endif
      /*  ★2 各テーブルのバッファ割当、コールバック関数登録  */
      user_ini_modbus_slave();

      ercd = modbus_slave(&ms);    /*  ★3 ModBusスレーブ処理開始  */}
#endif
```

★1) ModBus スレーブ TCP 制御情報の値設定

TCP ソケットの設定／生成、ms.type に値 MODBUS_TYPE_TCP を入れます。

★2) 各テーブルのバッファ割当、コールバック関数登録

前述した各テーブルのバッファ割当、コールバック関数登録を行います。

★3) スレーブ処理開始

設定が完了したので ModBus スレーブの処理を開始します。

Copyright © eForce Co., Ltd. All Rights Reserved

ソース解説: modbus スレーブタスク(RTU 動作)

ModBus/RTU のスレーブ動作を定義しています。

modbus_slave_user.c – 抜粋

```
#if USE_MODBUS_RTU          /* modbus/RTU */
void modbus_slave_com_task(VP_INT exinf)
{
      T_MODBUS_SLAVE *ms;
      ER ercd;

      ms = &g_ms[G_MS_IDX_RTU];

      /*  ★4 ModBusスレーブ RTU設定  */
      /* init COM driver */
      ercd = ini_com(SPCFG_DID_COM, &rtu_com_ini);
      dbg_assert(E_OK == ercd);
      ercd = ctr_com(SPCFG_DID_COM, STA_COM, 0);
      dbg_assert(E_OK == ercd);

      /* modbus/RTU settings */
      user_board_setup();
      g_mc.slave_id = SPCFG_SLAVE_ID;
      g_mc.tmr_ch = SPCFG_TIMER_CH;
      g_mc.baud_rate = SPCFG_COM_BAUDRATE;
      g_mc.cbk_sw_tr = cbk_rs485_tx_enable;
      ms->type = MODBUS_TYPE_RTU;
      ms->id = (UH)SPCFG_DID_COM;
      ms->com = &g_mc;

      /*  ★2 各テーブルのバッファ割当、コールバック関数登録  */
      user_ini_modbus_slave();;

      ercd = modbus_slave(&ms);    /*  ★3 ModBusスレーブ処理開始  */}
}
#endif
```

★4) ModBus スレーブ RTU 制御情報の値設定

COM ドライバの初期化(ini_com)、デバイス ID の設定、ms.type に値 MODBUS_TYPE_RTU を入れます。また、RTU 動作時は T_MODBUS_SLAVE_COM 構造体変数に対して、スレーブ ID、COM ドライバのデバイス ID、動作ボーレート、使用タイマドライバのチャンネルの登録が必須となり、必要に応じて RS485 用にコールバック関数の登録も行います。

Copyright © eForce Co.,Ltd. All Rights Reserved

ソース解説: 共通チェック用コールバック関数

要求メッセージのチェック用コールバック関数です。サンプルでは共通のコールバック関数で各ファンクションコードのチェック処理を行っています。

modbus_slave_user.c – 抜粋

```
ER cbk_chk_req(T_MODBUS_SLAVE *ms, UB *buf)
{
	T_MODBUS_DATA_REQ1 *req = (VP)buf;   /* ★5 */
	INT ret;
	UB ec = 0;
	/* ENTRY */

	/* MB Server receives mb_req_pdu */
	do {
		/* check - Function code supported */
		/* このコールバック処理に入る前にチェックする */

		/* ★6 check Quantity of XXXs */
		if (E_OK != mbs_is_valid_quant(req->func_code, req->quant)) {
			ec = EC03_ILLEGAL_DATA_VALUE;
			break;
		}

		/* ★7 Starting Address == OK   AND   Starting Address + Quantity of Outputs == OK */
		ret = mbs_is_valid_addr(req->func_code, req->addr, req->quant, 0);
		if (0 != ret) {
			if (E_OBJ == ret) {
				;   /* テーブル バッファ未登録 */
			}
			ec = EC02_ILLEGAL_DATA_ADDRESS;
			break;
		}

		/* ★8 Check completed before Request processing */
		ec = 0;
	} while(0);

	return (ER)(ec);
}
```

★5) modbus メッセージアクセス用構造体へのキャスト

コールバック関数の第 2 引数 buf には PDU 先頭アドレスが入ります。アクセス簡便化のため、アクセス用構造体ポインタにキャストしています。(バイト単位アクセスも可能です)

★6) 数量要素のチェック

mbs_is_valid_quant()を利用して、数量要素が範囲内にあるかをチェックします。チェック異常の場合、後続処理を中断し EC03_ILLEGAL_DATA_VALUE を関数戻り値に設定します。

★7) アクセス可能アドレスのチェック

mbs_is_valid_addr()を利用して、有効アドレスへのアクセスかをチェックします。チェック異

Copyright © eForce Co., Ltd. All Rights Reserved

常の場合、後続処理を中断し EC02_ILLEGAL_DATA_ADDRESS を関数戻り値に設定します。ここで mbs_is_valid_addr()の引数 chk_next=1 を指定しており、指定アドレス・数量が登録バッファを跨る場合も対応可能にしています。

★8) チェック処理が正常
チェック処理が正常な場合、関数戻り値に 0 を設定します。コールバック呼び元では、戻り値 0 の判定を受けて、ファンクション用コールバック関数を呼びます。

ソース解説: 共通ファンクションコード用コールバック関数

要求メッセージのファンクション用コールバック関数です。サンプルでは共通のコールバック関数でファンクションコードを判定後、各ファンクションコード用処理(f)を呼んでいます。

※ 本関数は長めのため、処理部を分けて解説いたします。

modbus_slave_user.c – 抜粋 (1/3)

```
ER cbk_func_req(T_MODBUS_SLAVE *ms, UB *buf)
{
	T_MODBUS_DATA *md;
	T_MODBUS_SLAVE_MAP *sm;
	T_MODBUS_DATA_REQ1 abs, rel;
	ER ercd;
	UB *tmp;

	/* ★9 */
	md = (VP)buf;
	abs = md->req1;

	/* ★10 テーブル登録バッファの取得 */
	sm = mbs_get_map(abs.func_code, abs.addr);
	if (!sm) {
		return EC04_SERVER_DEVICE_FAILURE;
	}

	/* ★11 作業用ポインタの設定(out:応答データに出力、in:要求データから入力) */
	switch ((INT)abs.func_code) {
	case F01_READ_COILS:
	case F02_READ_DISCRETE_INPUTS:
		tmp = md->rsp1.stats;				/* out: status */
		break;

	case F03_READ_HOLDING_REGISTERS:
	case F04_READ_INPUT_REGISTERS:
		tmp = (UB*)md->rsp2.regs;		/* out: registers */
		break;

	case F05_WRITE_SINGLE_COIL:
	case F06_WRITE_SINGLE_REGISTER:
		tmp = NULL;
		break;
```


Copyright © eForce Co.,Ltd. All Rights Reserved

```
    case F15_WRITE_MULTIPLE_COILS:
          tmp = md->req2.outs;                              /* in: outputs */
        break;

    case F16_WRITE_MULTIPLE_REGISTERS:
          tmp = (UB*)md->req3.regs;                 /* in: registers */
        break;

    default:
        ercd = (ER)EC01_ILLEGAL_FUNCTION;
        return ercd;
    }
…(2/3)に説明を移る…
```

★9) modbus メッセージアクセス用構造体へのキャスト、要求メッセージのヘッダ保持
コールバック関数の第 2 引数 buf には PDU 先頭アドレスが入ります。アクセス簡便化のため、アクセス用構造体ポインタにキャストしています。サンプルでは md->req{n} が要求メッセージへのアクセス、md->rsp{n} が応答メッセージへのアクセスを示します。
(コールバック関数では要求メッセージと応答メッセージのバッファは共通です。そのため、要求メッセージの内容取得後に応答メッセージを作成するようにして下さい。)
サンプルコードでは、要求メッセージのヘッダ情報(ファンクションコード・アドレス・数量)を頻繁に参照するため、変数 abs に保持しています。

★10) ファンクションコードに対応したテーブルバッファの取得
mbs_get_map()を利用して、登録テーブルバッファを取得します。サンプルでは各ファンクションコードの入出力先がこのバッファとなります。テーブル未登録の場合、サンプルではエラー判定とし EC04_SERVER_DEVICE_FAILURE を返します。

★11) 作業用ポインタの設定
各ファンクションコードの要求メッセージ or 応答メッセージには可変フィールドが存在する場合があります。それを順次読込・書込する必要があります。そのため、該当フィールドを作業用ポインタに入れアクセスをしやすくしています。

Copyright © eForce Co., Ltd. All Rights Reserved

modbus_slave_user.c – 抜粋 (2/3)

```
/* (1/3)からの処理続き */
    /* ★12 要求メッセージのアドレス・数量に複数バッファが該当する場合があるのでループ処理 */
    do {
        /* ★13 要求メッセージのアドレス・数量を対象バッファに合わせて相対値を求める */
        ercd = mbs_get_map_req(&rel, &abs, sm);
        if (E_OK != ercd) {
                /* 後続バッファが要求メッセージのアドレス対象外の場合は処理を抜ける */
                break;
        }
        /* ★14 ファンクションコードに応じた処理を実施 */
        ercd = E_OK;
        switch ((INT)rel.func_code) {
        case F01_READ_COILS:
            ercd = F01_read_coils(tmp, sm->dat, rel.addr, rel.quant);
            tmp += ercd;
            break;

        case F02_READ_DISCRETE_INPUTS:
            ercd = F02_read_discrete_inputs(tmp, sm->dat, rel.addr, rel.quant);
            tmp += ercd;
            break;

        case F03_READ_HOLDING_REGISTERS:
            ercd  = F03_read_holding_registers(tmp, sm->dat, rel.addr, rel.quant);
            tmp += ercd;
            break;

        case F04_READ_INPUT_REGISTERS:
          ercd  = F04_read_inputs_registers(tmp, sm->dat, rel.addr, rel.quant);
          tmp += ercd;
            break;

        case F05_WRITE_SINGLE_COIL:
          /* rel.quantにはコイルの設定値が入る */
            F05_write_single_coil(sm->dat, rel.addr, rel.quant);
            break;

        case F06_WRITE_SINGLE_REGISTER:
          /* rel.quantにはレジスタの設定値が入る */
            F06_write_single_register(sm->dat, rel.addr, rel.quant);
            break;

        case F15_WRITE_MULTIPLE_COILS:
            F15_write_multiple_coils(sm->dat, rel.addr, rel.quant, 0, tmp);
            tmp += rel.quant;
            break;

        case F16_WRITE_MULTIPLE_REGISTERS:
          F16_write_multiple_registers(sm->dat, rel.addr, rel.quant, 0, tmp);
            tmp += rel.quant;
            break;

        default:
            /* abnormal branch */
            break;
        }
          sm = mbs_get_map_next(sm);   /* ★15 */
    } while(sm);
```


Copyright © eForce Co.,Ltd. All Rights Reserved

…中略…

★12) 複数バッファ対応のループ処理
要求メッセージの指定アドレス・数量によっては、登録バッファの複数が該当する場合があります。該当バッファ数分のループ処理を行います。

★13) 要求メッセージのアドレス・数量を対象バッファに合わせた相対値の取得
mbs_get_map_req()を使い、指定バッファ sm のアクセス用に要求メッセージ情報 abs から相対値 rel を取得します。API エラーの場合、指定バッファは対象外なのでループから抜けます。

★14) ファンクションコードに応じた処理の実施
ファンクションコードを判定して、処理を分岐します。サンプルではファンクションコード 1,2,3,4,5,6,15,16 に対応した処理を実施します。(各ファンクション処理関数内では mbu_copy_bits(), mbu_copy_words() の処理を使い、実装しています。)
★11 で設定した作業用ポインタ入出力の対象に利用します。

★15) 登録バッファの隣接後続バッファの取得
mbs_get_map_next()を使い、指定バッファ sm の隣接後続バッファを取得して sm に設定します。隣接後続バッファがない場合、sm は NULL となりループ処理を終了します。

Copyright © eForce Co., Ltd. All Rights Reserved

modbus_slave_user.c – 抜粋 (3/3)

```
/* (2/3)からの処理続き */
    /* ★16 応答メッセージのサイズ確定 */
    switch ((INT)abs.func_code) {
    case F01_READ_COILS:
          md->rsp1.count = tmp - md->rsp1.stats;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F01 + md->rsp1.count));
        break;

    case F02_READ_DISCRETE_INPUTS:
        md->rsp1.count = tmp - md->rsp1.stats;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F02 + md->rsp1.count));
        break;

    case F03_READ_HOLDING_REGISTERS:
        md->rsp2.count = tmp - (UB*)md->rsp2.regs;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F03 + md->rsp2.count));
        break;

    case F04_READ_INPUT_REGISTERS:
        md->rsp2.count = tmp - (UB*)md->rsp2.regs;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F04 + md->rsp2.count));
        break;

    case F05_WRITE_SINGLE_COIL:
          md->rsp3.val = abs.quant;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F05));
        break;

    case F06_WRITE_SINGLE_REGISTER:
          md->rsp3.val = abs.quant;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F06));
        break;

    case F15_WRITE_MULTIPLE_COILS:
          md->rsp3.val = abs.quant;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F15));
        break;

    case F16_WRITE_MULTIPLE_REGISTERS:
          md->rsp3.val = abs.quant;
        ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F16));
        break;

    default:
        ercd = (ER)EC01_ILLEGAL_FUNCTION;
        break;
    }
    return ercd;
}
```

★16) 応答メッセージサイズの更新

応答メッセージのサイズに関係するフィールドを更新後、mbs_upd_buflen()を利用して応答メッセージの PDU サイズを指定します。応答メッセージの送信サイズが決定します。

Copyright © eForce Co.,Ltd. All Rights Reserved

HoldingRegisters 用ファンクションコード**(3,6,16)**は **32/64** ビットアクセスするサンプル処理を入れています。そのため、共通コールバック関数とは別にしています。

ソース解説: HoldingRegisters チェック用コールバック関数

modbus_slave_user.c – 抜粋

```
ER cbk_chk_req_hr(T_MODBUS_SLAVE *ms, UB *buf)
{
	T_MODBUS_DATA_REQ1 *req = (VP)buf; /*  ★17 */
	INT ret;
	ER ec = 0;
	UH num;
	/* ENTRY */

	/* MB Server receives mb_req_pdu */
	do {
		/*  ★6 check Quantity of XXXs */
		if (E_OK != mbs_is_valid_quant(req->func_code, req->quant)) {
			ec = EC03_ILLEGAL_DATA_VALUE;
			break;
		}

		/*  ★18 Starting Address == OK    AND    Starting Address + Quantity of Outputs == OK */
		ret = mbs_is_valid_addr(req->func_code, req->addr, req->quant, 0);
		if (0 != ret) {
			if (E_OBJ == ret) {
				;	/*  テーブル  バッファ未登録  */
			}
			ec = EC02_ILLEGAL_DATA_ADDRESS;
			break;
		}

		/*  ★19 0x0010番地以降を32/64bitアクセスのみ許可とする  */
		if (0x0010U <= req->addr) {
		  /*  対象番地に  WriteSingleRegister(6)  が来た場合はエラーとする  */
		  if (F06_WRITE_SINGLE_REGISTER == req->func_code) {
				ec = EC02_ILLEGAL_DATA_ADDRESS;
				break;
		  }

		  /* 0x0100ビットが立つ番地は64bitアクセスとする  */
		  num = (0x0100 & req->addr) ? 4U : 2U ;   /*  使用ワード数  */
			if ((0U != (req->addr % num)) || (0U != (req->quant % num))) {
                                        ec = EC02_ILLEGAL_DATA_ADDRESS;
                                        break;
			}
		}

		/* Check completed before Request processing */
		ec = 0;
	} while(0);

	return ec;
}
```


Copyright © eForce Co., Ltd. All Rights Reserved

★17) modbus メッセージアクセス用構造体へのキャスト
コールバック関数の第 2 引数 buf には PDU 先頭アドレスが入ります。アクセス簡便化のため、アクセス用構造体ポインタにキャストしています。

★18) アクセス可能アドレスのチェック
mbs_is_valid_addr()を利用して、有効アドレスへのアクセスかをチェックします。チェック異常の場合、後続処理を中断し EC02_ILLEGAL_DATA_ADDRESS を関数戻り値に設定します。ここで mbs_is_valid_addr()の引数 chk_next=0 を指定しており、指定アドレス・数量が登録バッファを跨る場合にはエラーとしています。

★19) 32/64bit アクセスのための妥当性チェック
アドレスが 0x0010 番地以降は 32bit アクセス、0x0100 番地以降は 64bit アクセスにしているため、要求アドレスの番地、数量が 2or4 ワード単位であることをチェックしています。条件に合わない場合、EC02_ILLEGAL_DATA_ADDRESS を関数戻り値に設定します。

Copyright © eForce Co.,Ltd. All Rights Reserved

ソース解説: HoldingRegisters ファンクション用コールバック関数

modbus_slave_user.c – 抜粋

```
ER cbk_func_req_hr(T_MODBUS_SLAVE *ms, UB *buf)
{
	T_MODBUS_DATA *md;
	T_MODBUS_SLAVE_MAP *sm;
	T_MODBUS_DATA_REQ1 abs;
	T_MODBUS_DATA_REQ1 rel;
	ER ercd;
	UD *ud;
	UW *uw;
	UH quant;
	UB end_swp;
	UB n;

	/*  ★9 */
	md = (VP)buf;
	abs = md->req1;

	/*  ★10  テーブル登録バッファの取得  */
	sm = mbs_get_map(abs.func_code, abs.addr);
	if (!sm) {
		return EC04_SERVER_DEVICE_FAILURE;
	}

	/*  ★13  バッファの開始アドレスに合わせた要求アドレスに変更する  */
	ercd = mbs_get_map_req(&rel, &abs, sm);
	if (E_OK != ercd) {
		goto exit_cbk_func_req_hr;
	}

	/*  ★20  ファンクションコードに応じた処理を実施  */
	switch ((INT)rel.func_code) {
	case F03_READ_HOLDING_REGISTERS:	/*  ★21 */
		md->rsp2.count = F03_read_holding_registers(
				(UB*)md->rsp2.regs, sm->dat, rel.addr, rel.quant);
		ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F04 + md->rsp2.count));
		if (E_OK == ercd) {
			/* 0x00[1234]0番地は32/64bitアクセス  */
			switch (abs.addr & 0x00F0U) {
			case 0x0010U:	end_swp = ES_BIGE_NOSWP;	break;
			case 0x0020U:	end_swp = ES_BIGE_SWP;		break;
			case 0x0030U:	end_swp = ES_LITE_NOSWP;	break;
			case 0x0040U:	end_swp = ES_LITE_SWP;		break;
			default:		end_swp = 0U;			break;
			}
			if (0U != end_swp) {
				/*  ワード値は既にバイトオーダー変換されているため調整フラグを設定  */
				end_swp |= ES_DONE_BO;
				if (abs.addr & 0x0100) {	/* 64bit */
					for (n = 0U; n < (UB)quant; n+=4U) {	/* 4word */
						ud = (VP)&md->rsp2.regs[n];
						*ud = mbu_swap_4xword(*ud, end_swp);
					}
				}
				else {				/* 32bit */
					for (n = 0U; n < (UB)quant; n+=2U) {	/* 2word */
```


Copyright © eForce Co., Ltd. All Rights Reserved

```
                            uw = (VP)&md->rsp2.regs[n];
                            *uw = mbu_swap_2xword(*uw, end_swp);
                        }
                    }
                }
            }
            break;

        case F16_WRITE_MULTIPLE_REGISTERS:    /* ★22*/
            /* 0x00[1234]0番地は32/64bitアクセス */
            switch (abs.addr & 0x00F0U) {
            case 0x0010U:     end_swp = ES_BIGE_NOSWP;       break;
            case 0x0020U:     end_swp = ES_BIGE_SWP;         break;
            case 0x0030U:     end_swp = ES_LITE_NOSWP;       break;
            case 0x0040U:     end_swp = ES_LITE_SWP;         break;
            default:          end_swp = 0U;                  break;
            }
            if (0U != end_swp) {
                /* ワード値は既にバイトオーダー変換されているため調整フラグを設定 */
                end_swp |= ES_DONE_BO;
                if (abs.addr & 0x0100) {      /* 64bit */
                    for (n = 0U; n < (UB)md->req3.quant; n+=4U) {    /* 4word */
                        ud = (VP)&md->req3.regs[n];
                        *ud = mbu_swap_4xword(*ud, end_swp);
                    }
                }
                else {                        /* 32bit */
                    for (n = 0U; n < (UB)md->req3.quant; n+=2U) {    /* 2word */
                        uw = (VP)&md->req3.regs[n];
                        *uw = mbu_swap_2xword(*uw, end_swp);
                    }
                }
            }
            F16_write_multiple_registers(sm->dat, rel.addr, rel.quant, 0, (UB*)md->req3.regs);
            ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F16));
            break;

        case F06_WRITE_SINGLE_REGISTER:    /* ★23*/
            F06_write_single_register(sm->dat, rel.addr, rel.quant);
            md->rsp3.val = abs.quant;
            ercd = mbs_upd_buflen(ms, ((UB)MBL_PDULEN_RSP_F06));
            break;

        default:      /* ★24*/
            ercd = E_SYS;     /* abnormal branch */
            goto exit_cbk_func_req_hr;
            break;
        }

        if (0) {
exit_cbk_func_req_hr:
            if (E_OK != ercd) {
                ercd = EC04_SERVER_DEVICE_FAILURE;
            }
        }

        return ercd;
}
```


Copyright © eForce Co.,Ltd. All Rights Reserved

★20) ファンクションコードに応じた処理の実施
HoldingRegisters テーブルが対象のファンクションコード 3,6,16 に応じた処理を実施します。

★21) F03_READ_HOLDING_REGISTERS の処理
F03_read_holding_registers()にて対象バッファからデータを取得後、取得したデータのアドレスに応じて 32/64bit のエンディアン・スワップ変換処理を行います。
変換処理後、応答メッセージのレジスタデータとして送信されます。

★22) F16_WRITE_MULTIPLE_REGISTERS 処理
要求メッセージのレジスタデータをアドレスに応じて 32/64bit のエンディアン・スワップ変換処理を行います。変換したレジスタデータを F16_write_multiple_registers()を使い、対象バッファの該当アドレスに適用します。

★23) F06_WRITE_SINGLE_REGISTER 処理
F06_write_single_register()にて、要求メッセージのアドレス・データを該当バッファのアドレスに書き込みます。

★24) ファンクションコードが上記以外の場合
ファンクションコードが想定外の場合、コールバック関数をエラーで終了します。
(通常この分岐に飛ぶことはありません。例外処理となります。)

Copyright © eForce Co., Ltd. All Rights Reserved

第5章　付録

5. 1 パケット形式

5. 1. 1　modbus メッセージアクセス用構造体

modbus メッセージの各要素にアクセスするために用意された構造体です。コールバック関数のバッファ req, buf をこれらの構造体ポインタにキャストして使用できます。(これらの構造体はメンバ間パディングが入らないよう pack オプションが有効で定義されており、他の構造体にコピーする際などは注意して下さい。)

T_MODBUS_DATA

メッセージアクセス用構造体のいずれにも変換可能な共用体です。

```
typedef union t_modbus_data {
    T_MODBUS_PDU        pdu;        PDU 形式
    T_MODBUS_DATA_REQ1  req1;       要求メッセージ 1
    T_MODBUS_DATA_REQ2  req2;       要求メッセージ 2
    T_MODBUS_DATA_REQ3  req3;       要求メッセージ 3
    T_MODBUS_DATA_RSP1  rsp1;       応答メッセージ 1
    T_MODBUS_DATA_RSP2  rsp2;       応答メッセージ 2
    T_MODBUS_DATA_RSP3  rsp3        応答メッセージ 3
    T_MODBUS_DATA_ERR   err;        例外応答メッセージ
} T_MODBUS_DATA;
```

T_MODBUS_PDU

{func_code,data}のシンプルな PDU フォーマットの構造体です。

```
typedef struct t_modbus_pdu {
    UB func_code;                   ファンクションコード
    UB data[1];                     データフィールド
} T_MODBUS_PDU;
```

T_MODBUS_DATA_REQ1

ファンクションコード 1,2,3,4,5,6 で利用可能な要求メッセージの構造体です。

```
typedef struct t_modbus_data_req1 {
    UB func_code;                   ファンクションコード
    UH addr;                        アドレス
    UH quant;                       数量
} T_MODBUS_DATA_REQ1 ;
```


Copyright © eForce Co.,Ltd. All Rights Reserved

T_MODBUS_DATA_REQ2

ファンクションコード 15 で利用可能な要求メッセージの構造体です。

```
typedef struct t_modbus_data_req2 {
    UB func_code;                   ファンクションコード
    UH addr;                        アドレス
    UH quant;                       数量
    UB count;                       Byte count
    UB outs[1]                      Outputs Value
} T_MODBUS_DATA_REQ2 ;
```

T_MODBUS_DATA_REQ3

ファンクションコード 16 で利用可能な要求メッセージの構造体です。

```
typedef struct t_modbus_data_req3 {
    UB func_code;                   ファンクションコード
    UH addr;                        アドレス
    UH quant;                       数量
    UB count;                       Byte count
    UH regs[1]                      Registers Value
} T_MODBUS_DATA_REQ3 ;
```

T_MODBUS_DATA_RSP1

ファンクションコード 1,2 で利用可能な応答メッセージの構造体です。

```
typedef struct t_modbus_data_rsp1 {
    UB func_code;                   ファンクションコード
    UB count;                       Byte count
    UB stats[1];                    ステータス
} T_MODBUS_DATA_RSP1 ;
```

T_MODBUS_DATA_RSP2

ファンクションコード 3,4 で利用可能な応答メッセージの構造体です。

```
typedef struct t_modbus_data_rsp2 {
    UB func_code;                   ファンクションコード
    UB count;                       Byte count
    UH regs[1];                     レジスタ値
} T_MODBUS_DATA_RSP2 ;
```


Copyright © eForce Co., Ltd. All Rights Reserved

T_MODBUS_DATA_RSP3

ファンクションコード 5,6,15,16 で利用可能な応答メッセージの構造体です。

```
typedef struct t_modbus_data_rsp3 {
    UB func_code;                   ファンクションコード
    UH addr;                        Byte count
    UH val;                         レジスタ値
} T_MODBUS_DATA_RSP3 ;
```

T_MODBUS_DATA_F01、T_MODBUS_DATA_F02

ファンクションコード 1,2 用の要求・応答・例外応答メッセージの共用体です。

```
typedef union t_modbus_data_f01 {
    T_MODBUS_DATA_REQ1   req;       要求メッセージ
    T_MODBUS_DATA_RSP1   rsp;       応答メッセージ
    T_MODBUS_DATA_ERR    err;       例外応答メッセージ
} T_MODBUS_DATA_F01;
typedef T_MODBUS_DATA_F01    T_MODBUS_DATA_F02;
```

T_MODBUS_DATA_F03、T_MODBUS_DATA_F04

ファンクションコード 3,4 用の要求・応答・例外応答メッセージの共用体です。

```
typedef union t_modbus_data_f03 {
    T_MODBUS_DATA_REQ1   req;       要求メッセージ
    T_MODBUS_DATA_RSP2   rsp;       応答メッセージ
    T_MODBUS_DATA_ERR    err;       例外応答メッセージ
} T_MODBUS_DATA_F03;
typedef T_MODBUS_DATA_F03    T_MODBUS_DATA_F04;
```

T_MODBUS_DATA_F05、T_MODBUS_DATA_F06

ファンクションコード 5,6 用の要求・応答・例外応答メッセージの共用体です。

```
typedef union t_modbus_data_f05 {
    T_MODBUS_DATA_REQ1   req;       要求メッセージ
    T_MODBUS_DATA_RSP3   rsp;       応答メッセージ
    T_MODBUS_DATA_ERR    err;       例外応答メッセージ
} T_MODBUS_DATA_F05;
typedef T_MODBUS_DATA_F05    T_MODBUS_DATA_F06;
```


Copyright © eForce Co.,Ltd. All Rights Reserved

5.2 定数とマクロ

5.2.1 ModBus ファンクションコード

uNet3-ModBus で解釈可能なファンクションコードを次の名称で定義しています。

```
enum modbus_func_codes {
        F01_READ_COILS                  = 0x01,
        F02_READ_DISCRETE_INPUTS        = 0x02,
        F03_READ_HOLDING_REGISTERS      = 0x03,
        F04_READ_INPUT_REGISTERS        = 0x04,
        F05_WRITE_SINGLE_COIL           = 0x05,
        F06_WRITE_SINGLE_REGISTER       = 0x06,
        F15_WRITE_MULTIPLE_COILS        = 0x0F,
        F16_WRITE_MULTIPLE_REGISTERS    = 0x10,
};
```

各構造体メンバの func_code に入る値です。(例: T_MODBUS_PDU::func_code)

5.2.2 ModBus 例外コード

ModBus の例外コードを次の名称で定義しています。

```
enum modbus_exception_codes {
        EC01_ILLEGAL_FUNCTION           = 0x01,
        EC02_ILLEGAL_DATA_ADDRESS       = 0x02,
        EC03_ILLEGAL_DATA_VALUE         = 0x03,
        EC04_SERVER_DEVICE_FAILURE      = 0x04,
        EC05_ACKNOWLEDGE                = 0x05,
        EC06_SERVER_DEVICE_BUSY         = 0x06,
        EC08_MEMORY_PARITY_ERROR        = 0x08
};
```

コールバック関数の戻り値とすることで、例外応答メッセージを返せます。

Copyright © eForce Co., Ltd. All Rights Reserved

5.2.3 PDU データサイズ

ModBus の PDU データサイズの固定値部分を次の名称で定義しています。

```
enum modbus_len {
…
        MBL_PDULEN_RSP1 = 2,      /* + count */
        MBL_PDULEN_RSP2 = 2,      /* + count */
        MBL_PDULEN_RSP3 = 5,
        MBL_PDULEN_RSP_F01 = MBL_PDULEN_RSP1,
        MBL_PDULEN_RSP_F02 = MBL_PDULEN_RSP1,
        MBL_PDULEN_RSP_F03 = MBL_PDULEN_RSP2,
        MBL_PDULEN_RSP_F04 = MBL_PDULEN_RSP2,
        MBL_PDULEN_RSP_F05 = MBL_PDULEN_RSP3,
        MBL_PDULEN_RSP_F06 = MBL_PDULEN_RSP3,
        MBL_PDULEN_RSP_F15 = MBL_PDULEN_RSP3,
        MBL_PDULEN_RSP_F16 = MBL_PDULEN_RSP3,
…
};
```

コールバック関数の中でバッファデータを更新した後、mbs_upd_buflen()の引数として利用可能です。

Copyright © eForce Co.,Ltd. All Rights Reserved

5. 3 ModBus 用語

マスタ・スレーブ方式

ModBus プロトコルでは接続機器はマスタとスレーブに分かれます。マスタ：スレーブの接続台数は、シリアル接続時は１：多、TCP 接続時は１：１となります。TCP 接続では ModBus スレーブ側が TCP サーバ、ModBus マスタ側が TCP クライアントになります。

マスタがスレーブに対して要求メッセージを送信し、スレーブは要求に対する動作を行った後に応答メッセージをマスタに返します。

PDU、ADU

ModBus プロトコルでは基本的なデータ単位を PDU(Protocol Data Unit)とよび、ファンクションコードとデータフィールドで構成されます。ADU(Application Data Unit)は実際に通信されるデータ単位となり、PDU に加えてアドレスや CRC など通信レイヤで必要なフィールドを含めます。PDU は通信方式に関わらず共通、ADU は通信方式で異なります。

以下 ModBus ドキュメントより抜粋。

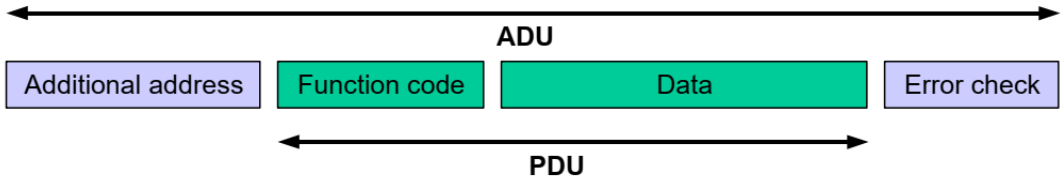


Figure 3: General MODBUS frame

ファンクションコード

Modbus メッセージ内の操作の種類を指定する 1 バイトのコード。例えば、読み取りや書き込み操作を指示します。

データフィールド

ファンクションコードに対応するデータを含むフィールド。例えば、読み取るレジスタのアドレスや書き込む値が含まれます。

アドレスフィールド

スレーブデバイスのアドレスを指定するフィールド。マスタがどのスレーブにメッセージを送信するかを決定します。スレーブ側は宛先が自分でない場合、メッセージを破棄します。

Copyright © eForce Co., Ltd. All Rights Reserved

要求メッセージ (Request PDU, mb_req_pdu)

マスタからスレーブに送信されるメッセージです。スレーブ側は本メッセージ受信後に自アドレス当ての場合にのみ応答します。(TCP 接続時は必ず応答します)

応答メッセージ (Response PDU, mb_rsp_pdu)

要求メッセージに対してスレーブから応答されるメッセージです。要求メッセージに対して、正常なファンクション処理実施後に返されます。

例外応答メッセージ (Exception Response PDU, mb_excep_rsp_pdu)

要求メッセージに対してスレーブから応答されるメッセージです。要求メッセージに対して、正常な応答が出来ない場合(未サポートのファンクション、データ範囲不正、アドレス不正など)に返されます。

Copyright © eForce Co.,Ltd. All Rights Reserved

μNet3/ModBus ユーザーズガイド

2024 年 9 月　　第 1 版
2024 年 9 月 30 日　第 2 版
2025 年 3 月　　第 3 版

イー・フォース株式会社　https://www.eforce.co.jp/
Copyright (C) 2024-2025 eForce Co.,Ltd. All Rights Reserved.

Copyright © eForce Co., Ltd. All Rights Reserved