RX72M Group

CPU カード Modbus スタートアップマニュアル

R01AN6757JJ0100
Rev.1.00
2023/1/31

要旨

本書は、産業ネットワーク評価用 RX72M CPU カードで Modbus 通信を行うためのクイックスタートガイドです。

本スタックは、イー・フォース社のリアルタイム OS「μC3（マイクロ・シー・キューブ）」と、同じく TCP/IP プロトコルスタック「μNet3（マイクロ・ネット・キューブ）」上で動作します。

動作確認デバイス

RX72M

目次

1. 概要

本書は、RX72M で動作する Modbus プロトコルスタックの資料であり、プロトコルスタックを使ったアプリケーションを開発、実装する際の機能概要やアプリケーション・プログラミング・インタフェース（API）、アプリケーションサンプルについて記載しています。

RX72M 用 Modbus プロトコルスタックは、イーサネット・ベースの Modbus TCP と、RS-485 シリアル通信ベースの Modbus RTU、Modbus ASCII の各プロトコルに対応しています。

本パッケージでは、RX72M CPU カードでの動作としてイーサネット・ベースの Modbus TCP server に対応しています。

1.1 特長

Modbus プロトコルは、Modicon Inc.（Schneider Electric SA.）がプログラマブルロジックコントローラ（PLC）向けに開発した通信プロトコルであり、仕様は公開されています。

プロトコル仕様書（PI-MBUS-300 Rev.J）を参照ください。

RX72M 用 Modbus プロトコルスタックは、次のアプリケーションの迅速かつ容易な開発を可能とします。スタックモードは、アプリケーション実行時に初期化 API によって指定されます。

- Modbus TCP サーバ （ビルド構成対応）
- Modbus TCP ゲートウェイ
- Modbus RTU マスタスタック
- Modbus RTU スレーブスタック
- Modbus ASCII マスタスタック
- Modbus ASCII スレーブスタック

RX72M 用 Modbus プロトコルスタックでは、以下の 9 つのファンクションコードをサポートします。

- 1(0x01) – Read coils
- 2(0x02) – Read discrete input
- 3(0x03) – Read holding registers
- 4(0x04) – Read input registers
- 5(0x05) – Write single coil
- 6(0x06) – Write single register
- 15(0x0F) – Write multiple coils
- 16(0x10) – Write multiple registers
- 23(0x17) – Read/Write multiple registers

Modbus に関する詳細は、以下のサイトを参照して下さい。

http://www.modbus.org

「Modicon Modbus Protocol Reference Guide Rev.J」 (PI_MBUS_300.pdf)

「Modbus Application Protocol Specification V1.1b3」(Modbus_Application_Protocol_V1_1b3.pdf)

※更新によりバージョン番号は異なる場合がございます。最新のマニュアルを参照ください。

1.2 動作環境

本マニュアルのサンプルプログラムは表 1.1 の環境を想定しています。

表 1.1 動作環境

項目	内容
使用ボード	ルネサスエレクトロニクス製 RX72M CPU Card 型名：RTK0EMXDE0C00000BJ
CPU	RX CPU (RXv3)
動作周波数	CPU クロック (CPUCLK)：240MHz
動作電圧	3.3V
動作モード	シングルチップモード ブートモード(SCI インタフェース) ブートモード(USB インタフェース) ブートモード(FINE インタフェース)
使用デバイス	R5F572MNDDBD ・コードフラッシュメモリ 容量：4Mバイト ROM キャッシュ：8K バイト ・データフラッシュメモリ 容量：32K バイト ・RAM / 拡張 RAM 512k バイト/ 512k バイト
通信プロトコル	Modbus
統合開発環境	e2Studio V7.5.0 以降
ツールチェーン	RX ファミリ用 C/C++コンパイラパッケージ V3.01.00 以降
エミュレータ	ルネサスエレクトロニクス製 オンボードエミュレータ E2 On-Board (以下、E2OB)
評価用ツール	ModbusDemoApplication.exe：Modbus 評価用テストプログラム

1.3 参考ドキュメント

Modbus に関する技術情報は Modbus Organization のサイトから、RX72M CPU カードに関する情報はルネサス エレクトロニクスのサイトから入手できます。

・Modbus Organization のサイト : http://www.modbus.org

・ルネサス エレクトロニクスのサイト : http://www.renesas.com

表 1.2 Modbus 関連ドキュメント

項番	ドキュメント
1	Modbus_Application_Protocol_V1_1b3.pdf
2	PI_MBUS_300.pdf
3	Modbus_over_serial_line_V1_02.pdf
4	Modbus_Messaging_Implementation_Guide_V1_0b.pdf

表 1.3 μC3/μNet3 関連ドキュメント

項番	ドキュメント
1	ProcessorDependentManual_RXv3.pdf : μC3/Standard ユーザーズガイドプロセッサ依存部 RXv3 編
2	TutorialGuide_RXv3_RX72M.pdf : μC3/Standard チュートリアルガイド RXv3 - RX72M 編
3	uC3Std_UsersGuide.pdf : μC3/Standard ユーザーズガイド
4	uNet3_BSD_UsersGuide.pdf : μNet3/BSD ユーザーズガイド
5	uNet3_DriverGuide.pdf : μNet3Ethernet ドライバインタフェース
6	uNet3_UsersGuide.pdf : μNet3 ユーザーズガイド

表 1.4 RX72M CPU カード関連ドキュメント

項番	ドキュメント
1	RX72M グループユーザーズマニュアル ハードウェア編（R01UH0804JJ）
2	RX72M CPU Card with RDC-IC ユーザーズマニュアル（R12UZ0098JJ0100）
3	RX72M CPU Card with RDC-IC 回路図（R12TU0146EJ0100）

表 1.5 エミュレータ関連ドキュメント

項番	ドキュメント
1	E1/E20 エミュレータ, E2 エミュレータ Lite ユーザーズマニュアル別冊（RX ユーザシステム設計編）（R20UT0399JJ）
2	RX ファミリ用 E1/E20 エミュレータユーザーズマニュアル（R20UT0398JJ）

2. ハードウェア

RX72M CPU カードの詳細情報に関しては、「RX72M CPU Card with RDC-IC ユーザーズマニュアル」（R12UZ0098JJ0100）をご参照ください。

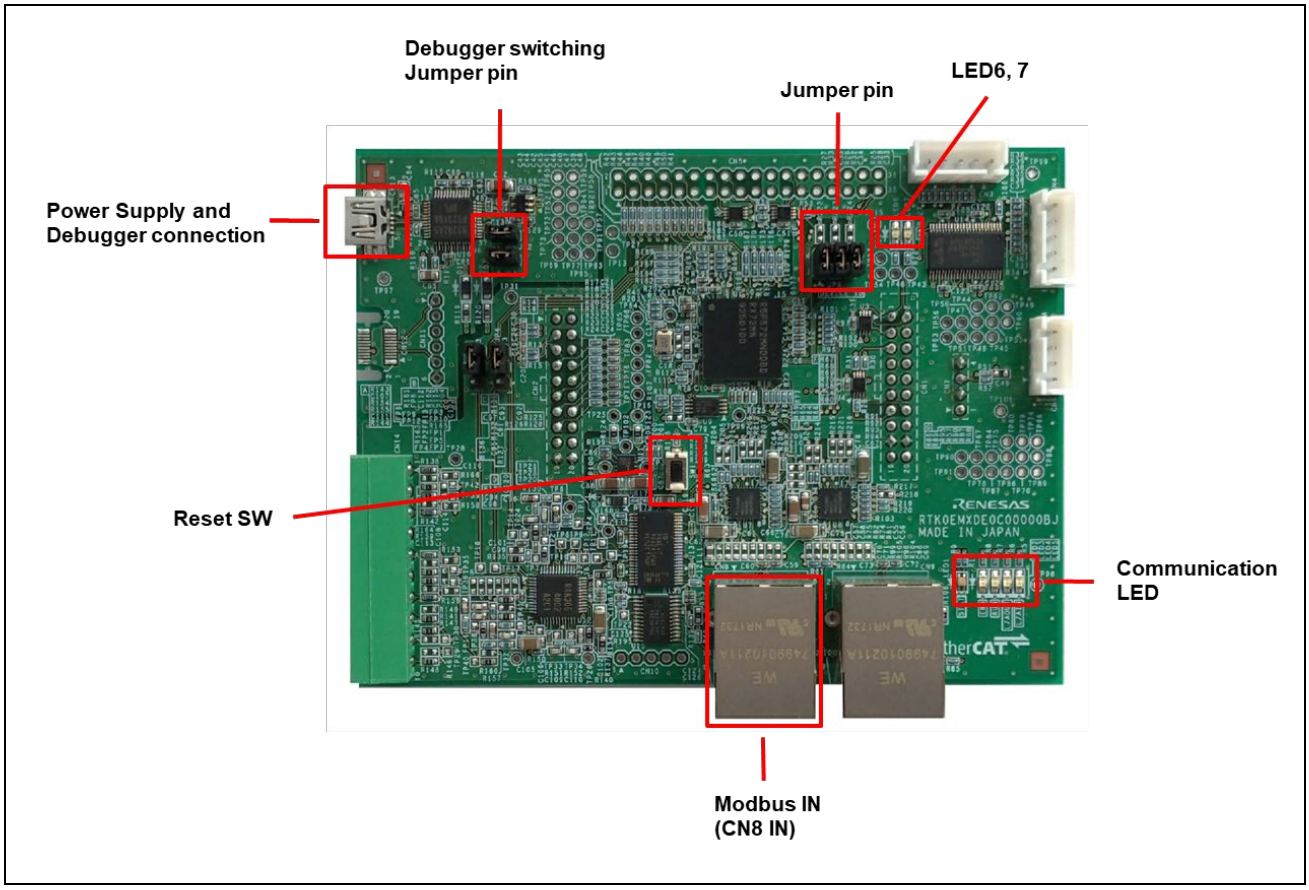

図 2.1　　RX72M CPU カード構成

2.1　　CPU カードの設定

CPU カードの電源投入前に、ジャンパーピンの設定を行います。

E2OB を使用する場合　：Debugger switching Jumper pin をショートします。

2.2　　電源の接続

CPU カードは DC Jack がありません、USB コネクタより、DC5V を入力してください。

3. e2studio のインストール

以下の web サイトから、RX72M 対応の e2studio(V7.5.0 以降)をダウンロードしてください。

https://www.renesas.com/e2studio_download

3.1 CC-RX コンパイラ V3.01.00 インストール

e2studio のインストール中にコンパイラ選択画面が現れます。[Renesas CCRX v3.01.00]を選択して[次へ]を選択することで、RX72M 対応の CC-RX V3.01.00 コンパイラが合わせてインストールされます。

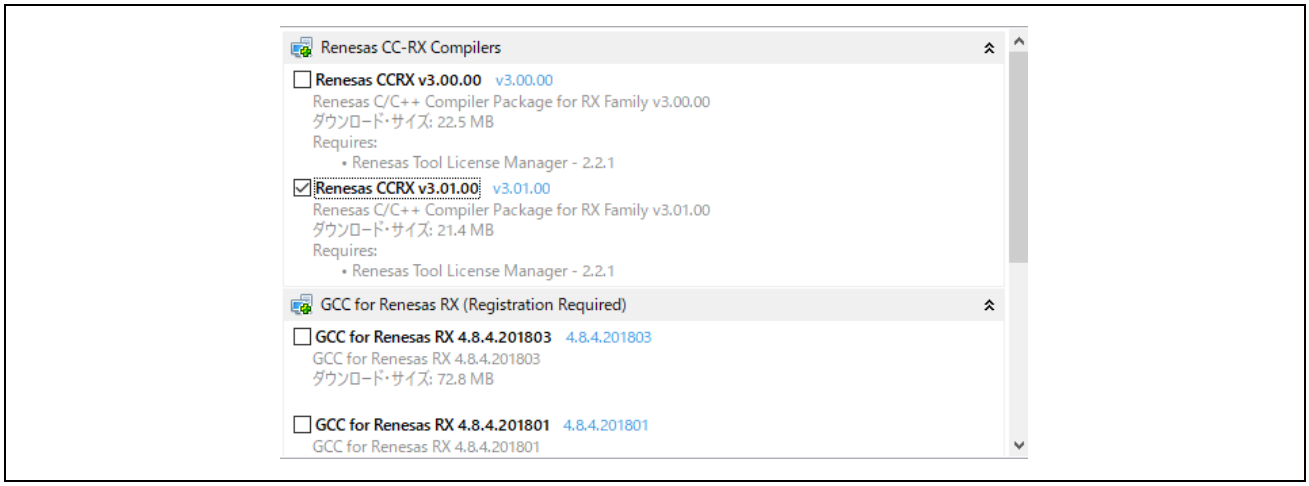

図 3.1 e2studio – コンパイラ選択画面

e2studio を起動するには、インストールされた下記フォルダにある”e2studio.exe”を実行してください。

e2_studio_rx72m¥eclipse

3.2 インストールされたコンパイラの確認方法

RX72M 対応 e2studio で CC-RX コンパイラ V3.01.00 版が利用できるようにします。

(1) e2studio を起動します。

(2) [ファイル]→[新規]→[C/C++Project]を選択します。[次へ]

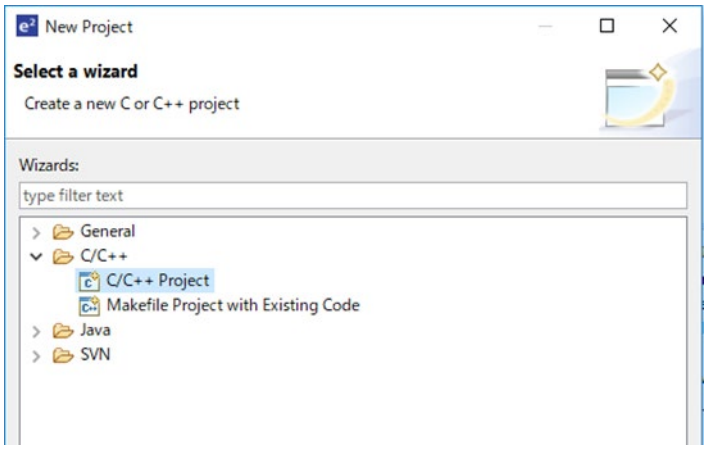

図 3.2 e2studio - プロジェクト選択画面

(3) [Templates for New C/C++ Project]ダイアログで[Renesas RX]→[Renesas CC-RX C/C++ Executable Project]→[次へ]を選択します。

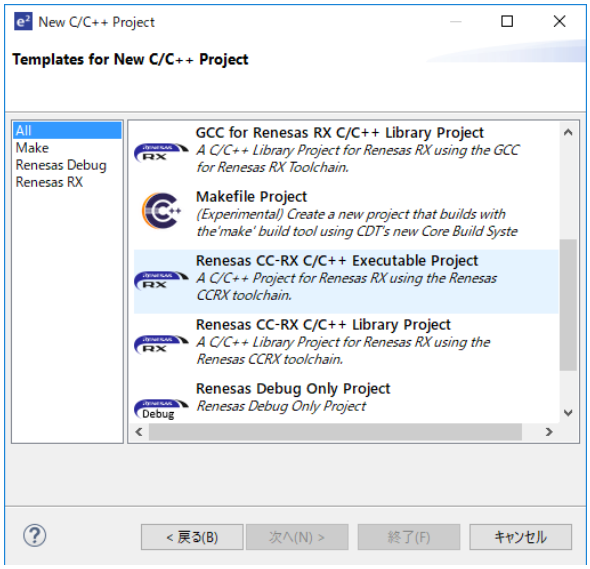

図 3.3 e2studio - プロジェクト選択画面

(4) [New Renesas CC-RX C/C++ Executable Project]ダイアログで任意のプロジェクト名を入力し[次へ]を選択します。

(5) [Select toolchain, device & debug settings]ダイアログで[Toolchain Settings]の[ツールチェーンの管理]をクリックします。

(6) [Renesas ツールチェーン管理]ダイアログで[追加]→[参照]をクリックし、インストールフォルダ“C:\Renesas\RX\3_0_1”を参照してください。

”Renesas CCRX”に”v3.01.00”が追加されていれば OK です。

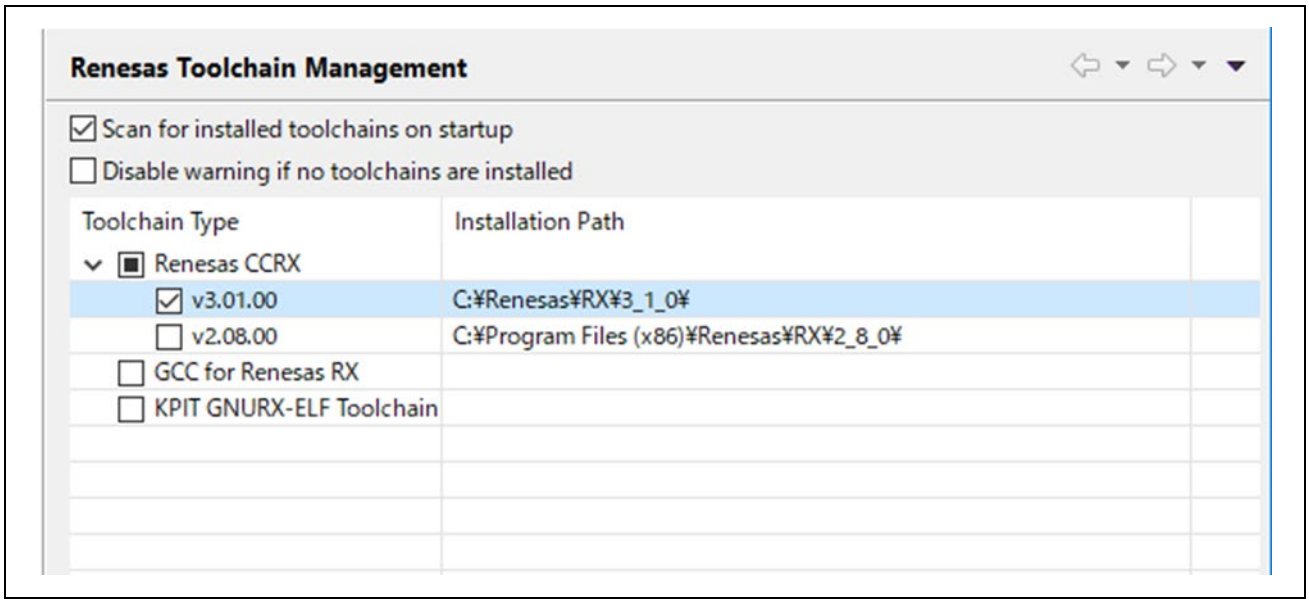

図 3.4 e2studio – Renesas ツールチェーン管理画面

4. サンプルアプリケーション

4.1 概要

本サンプルアプリケーションは、大きく分けて 3 つのブロックに分かれます。

1. イー・フォース社のリアルタイム OS「μC3（マイクロ・シー・キューブ）」と、同社製 TCP/IP プロトコルスタック「μNet3（マイクロ・ネット・キューブ）」
2. 上記 RTOS および TCP/IP スタックを使用する Modbus プロトコルスタックサンプルプログラム
3. 上記 Modbus プロトコルスタックを使用するアプリケーションサンプルプログラム

4.1.1 μC3/μNet3 (評価版)

本サンプルアプリケーションは、イー・フォース社のリアルタイム OS であるμC3（マイクロ・シー・キューブ）の評価版、および同社製 TCP/IP プロトコルスタックであるμNet3（マイクロ・ネットキューブ）の評価版を含みます。

詳細は、6 章「RX72M 用 TCP/IP スタックの基本概念」を参照ください。

4.1.2 Modbus プロトコルスタック（サンプルプログラム）

本サンプルアプリケーションは、Modbus プロトコルに基づく通信機能を提供するプロトコルスタックのサプルプログラムを含みます。本サンプルプログラムは RTOS としてμC3 を、また TCP/IP スタックとしてμNet3（Modbus TCP のみ）を使用します。

詳細は、7 章「RX72M 用 Modbus スタックの基本概念」~ 11 章「実装方法」を参照ください。

4.1.3 Modbus アプリケーション（サンプルプログラム）

本サンプルアプリケーションは、μC3、μNet3、および Modbus プロトコルスタックのサンプルプログラムを用いて、Modbus プロトコル通信をデモンストレーションするサンプルプログラムです。

詳細は、本章、および 5 章「サンプルアプリケーションによる通信のテスト」を参照ください。

4.2 ブロック図

サンプルアプリケーションのブロック図を、図 4.1 に示します。

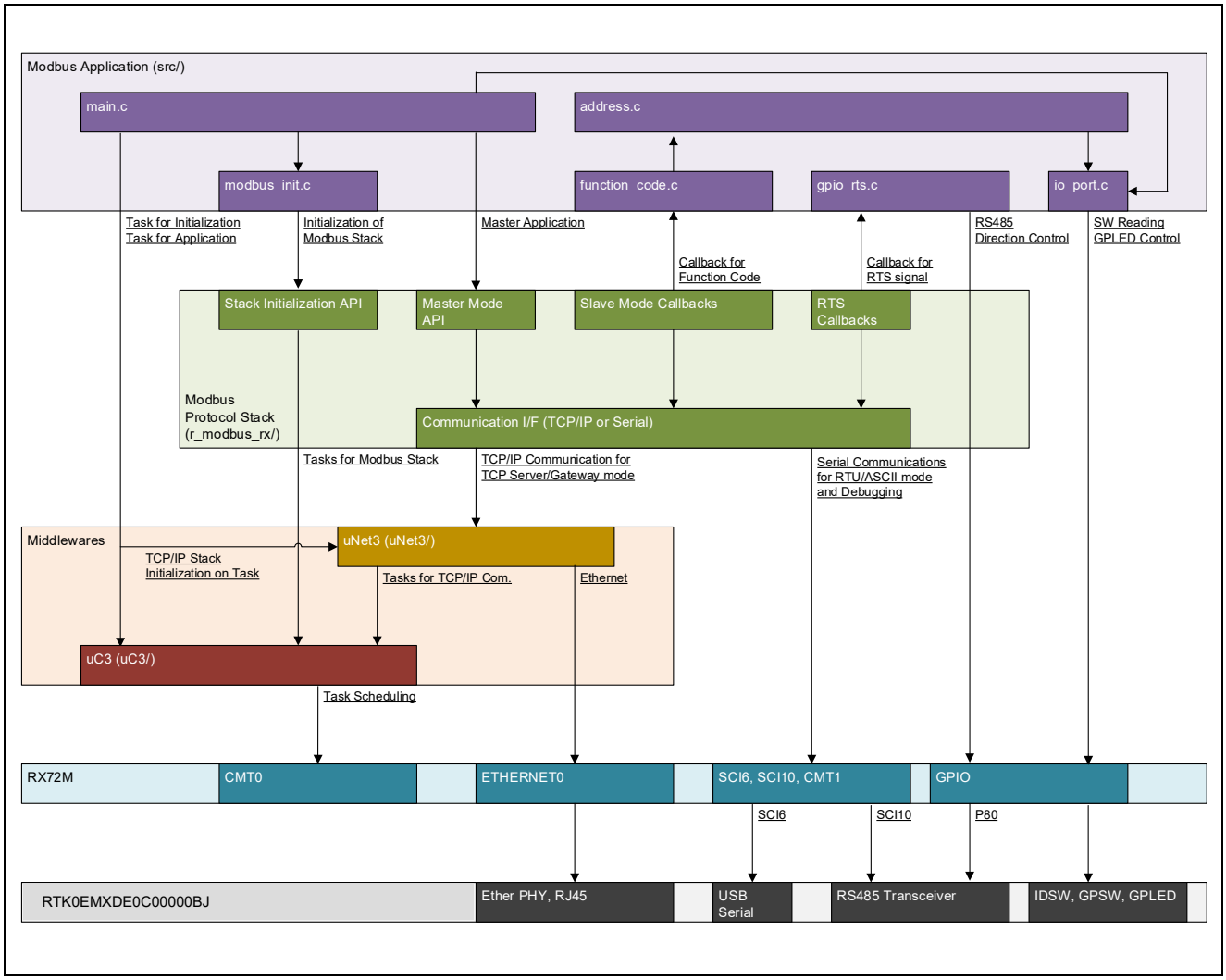

図 4.1　サンプルアプリケーション機能ブロック図

4.3 ファイル構成

本サンプルアプリケーションのファイル構成（一部省略）を表 4.1 に示します。

表 4.1 ファイル構成

フォルダ/ファイル名	説明
src/	Modbus アプリケーション サンプルプログラム
main.c	Modbus アプリケーション初期化処理、およびアプリケーションタスク
io_port.c (.h)	Modbus アプリケーション用 GPIO ポートアクセス実装
modbus_init.c (.h)	Modbus プロトコルスタック 初期化実装
mbapp_serial/	Modbus RTU/ASCII シリアル通信用実装
gpio_rts.c (.h)	Modbus RTU/ASCII シリアル通信用 RS485 RTS 制御実装
mbapp_slave/	Modbus スレーブモード用実装
function_code.c (.h)	Modbus スレーブモード用ファンクションコードコールバック実装
address.c (.h)	Modbus スレーブモード用データアドレス実装
r_modbus_rx/	Modbus プロトコルスタック サンプルプログラム
inc/ (src/)	Modbus プロトコルスタック サンプルプログラム ヘッダ＆ソースファイル
master/	Modbus プロトコルスタック マスタモード実装
port/	Modbus プロトコルスタック 通信ポート実装
serial/	Modbus プロトコルスタック Modbus RTU/ASCII プロトコル実装
slave/	Modbus プロトコルスタック スレーブモード実装
tcp/	Modbus プロトコルスタック Modbus TCP プロトコル実装
modbusCommon.h	Modbus プロトコル一般定義ヘッダファイル
modbusTyperdef.h	Modbus プロトコルスタック typedef 型定義ヘッダファイル
modbus.h	Modbus プロトコルスタック API インクルードファイル
uNet3/	μNet3 TCP/IP スタック実装
uC3/	μC3 RTOS 実装
generate/	e2studio 生成ファイル（ヒープコンフィグレーション）

4.4 ビルド構成

本サンプルアプリケーションの e2studio プロジェクトにおけるビルド構成を表 4.2 に示します。本サンプルアプリケーションは、Modbus プロトコルの各動作モードに対応したビルド構成を準備しています。本サンプルアプリケーションでは RX72M CPU カードでの動作として Modbus TCP サーバスタックのみ対応しています。

表 4.2 ビルド構成

スタックモード	ビルド構成名
Modbus TCP サーバスタック	TCP_SERVER_UC3
Modbus TCP ゲートウェイ機能付きサーバスタック	非対応
Modbus RTU マスタスタック	非対応
Modbus RTU スレーブスタック	非対応
Modbus ASCII マスタスタック	非対応
Modbus ASCII スレーブスタック	非対応

4.5 リソース構成

4.5.1 使用する HW モジュール

本サンプルプログラムの各 SW ブロックが使用する RX72M 搭載 HW モジュールを表 4.3 に示します。本サンプルプログラムに機能を追加する場合は、リソースの競合にご注意ください。

表 4.3 使用 HW モジュール

SW ブロック	HW モジュール	入出力端子	用途
μC3 （リアルタイム OS）	CMT0	-	RTOS タスクスケジューリング
μNet3 （TCP/IP スタック）	ETHERNET0	P74~5 PC2 PK0~1, PK4~5 PL2~7 PM4~7	Modbus TCP Server / TCP Gateway モード通信
Modbus アプリケーション	汎用 I/O ポート	P15 PH3 PK6 PK7	LED 制御 （CPU カード LED2～5）
		PB4 PB6 PB7	ジャンパスイッチ入力 （CPU カード JP5）

* CMT : Compare match timer

4.5.2 ドライバ SW モジュール

各 HW モジュールのデバイスドライバは、RX72M 向けμC3/Configurator によって生成されたドライバ SW モジュールを、サンプルアプリケーション向けに編集して使用しています。

- **DDR_COM.c**：Micro C Cube Standard, DEVICE DRIVER Standard Communication Interface
  - シリアルコミュニケーションインタフェース共通モジュール。

- **DDR_RX_CMT0.c**：Micro C Cube Compact, DEVICE DRIVER Interval Timer code for RX CMT
  - CMT0 用ドライバであり、RTOS μC3 が使用する。

- **DDR_RX_CMT1.c**：Micro C Cube Compact, DEVICE DRIVER Interval Timer code for RX CMT
  - CMT1 用ドライバであり、Modbus プロトコルスタックが使用する。

- **DDR_RX_SCI3.c**：Micro C Cube Compact, DEVICE DRIVER Serial Interface for RX
  - SCI6 用ドライバであり、Modbus プロトコルスタックのデバッグコンソール機能が使用する。
  - SCI10 用ドライバであり、Modbus RTU/ASCII 通信に使用する。

- **DDR_RX_ETH0.c**：Micro C Cube Compact, DEVICE DRIVER Ethernet driver for RX
  - ETHERNET0 用ドライバであり、Modbus TCP 通信に使用する。

4.5.3 リアルタイム OS 構成の設定

RX72M 向けμC3/Configurator によって生成された各種設定ファイルを、下記のように編集しています。

- **” uC3/ kernel_cfg.h”**
  - リアルタイム OS の各種パラメータを設定しています。

- **” uC3/ hw_init.c”**
  - init_peripheral ()
    - ETHERNET モジュールの端子設定、および Ether PHY クロック設定を記述しています。
  - _ddr_init ()
    - CMT ドライバの初期化関数をコールします。

- **”uC3/kernel_cfg.c”**
  - リアルタイム OS のインスタンスを生成し、各ビルド構成に合わせてタスクを生成します。
    - タスク生成前に_ddr_init ()をコールし、CMT を起動します。
  - スタックサイズを次のように変更しています。
    - `#pragma stacksize su = 0x400 /* Time Event Handler */`
    - `#pragma stacksize si = 0x400 /* Isr Service Routine */`

4.5.4 ヒープサイズの設定

下記ファイルを編集し、ヒープサイズを変更しています。

- **” generate/sbrk.h”**
    - `#define HEAPSIZE 0x10000`

4.6 Modbus アプリケーション概要

本節では、Modbus プロトコルスタックサンプルプログラムを用いた Modbus アプリケーションサンプルプログラム（src/ ディレクトリ下）の動作概要を解説

4.6.1 初期化処理、およびアプリケーション実装

4.6.1.1 main.c

本ファイルは、次の手順でドライバ、RTOS、TCP/IP スタックを初期化し、Modbus アプリケーション起動します。

1. 周辺 HW モジュール、およびドライバ SW を初期化します。

2. μC3 RTOS を起動し、各種タスクを生成します。
   - μC3 起動後、自動的に初期化タスクが起動します。
   - 以降の初期化処理は、初期化タスク上で実行します。

3. デバッグコンソール用シリアルポートをオープンします。
   - 以降、debug_printf() 関数によるシリアルコンソール出力が使用できます。
   - この機能は Modbus プロトコルスタックサンプルプログラムに実装されています。

4. μNet3 TCP/IP スタックを起動し、TCP/IP 通信用タスクを起動します。
   - 正常に起動した場合、シリアルコンソール出力経由で、デバイスに設定された IP を表示します。
   - この初期化は、Modbus TCP 通信を使用する際にのみ必要となります。

5. Modbus プロトコルスタックサンプルプログラムを起動します。
   - 詳細は modbus_init.c (.h) の解説を参照ください。
   - Modbus プロトコルスタックが RTU/ASCII 通信を使用する場合、シリアルコンソール出力経由で、RTU/ASCII 通信に使用するシリアル通信構成を表示します。

6. メインタスクを起動し、各 Modbus スタックモードのアプリケーションへ移行します。
   - RTU/ASCII マスタモード用アプリケーションのみ実装しています。動作は 5 章で解説します
   - その他のモードは、各種タスク上で、対向マスタデバイスのリクエストを受信したときに、パッシブに動作するため、アプリケーションは実装していません。

4.6.2 Modbus プロトコルスタック初期化処理

4.6.2.1 modbus_init.c (.h)

Modbus プロトコルスタックサンプルプログラムの初期化、起動処理の実装例を記述しています。使用する API の詳細は、10 章を参照ください。

本ファイルは、次の手順で Modbus プロトコルスタックを起動します。

- Modbus ファンクションコードのコールバック関数をマッピングします。
  - 使用 API：Modbus_slave_map_init
  - Modbus マスタ側から要求される Modbus ファンクションコードに対する処理をコールバック関数として実装し、実装した関数を専用の関数ポインタにマッピングします。
    - 実装例は、mbapp_slave/function_code.c (.h) の節にて説明します。
  - この初期化処理は、Modbus RTU/ASCII スレーブモード、および Modbus TCP サーバモードのみ必要となります。
    - 例外として、Modbus TCP ゲートウェイモードでも、初期化処理が必要になりますが、コールバック関数を専用の関数ポインタにセットする必要はありません。

- ホスト IP リストを有効にし、アクセス許可を与える IP を登録します。
  - 使用 API：Modbus_tcp_init_ip_table、Modbus_tcp_add_ip_addr
  - ホスト IP リストを有効にし、アクセス可能なマスタデバイスの IP を設定します。
  - この機能は Modbus TCP サーバ、Modbus TCP ゲートウェイモードのみ使用できます。
  - サンプルコードでは、チュートリアルの簡素化のために、この処理を除外とすることで、全ての IP のアクセスを許可しています。

- シリアル通信構成を設定し、各モードで Modbus プロトコルスタックを起動します。
  - 使用 API：Modbus_serial_stack_init、Modbus_tcp_init_stack
  - 専用の構造体変数に対して、Modbus RTU/ASCII 通信に使用するシリアル通信構成（ボーレートなど）をセットします。
  - また、別の専用の構造体に対して、RS485 トランシーバ制御用の RTS 信号を制御するコールバック関数をセットします。
    - RTS 信号制御用のコールバック関数の実装例は、mbapp_serial/gpio_rts.c (.h) の節にて説明します。
  - 各初期化 API に対してこれらの構造体を引数として入力し、各種スタックモードにて Modbus プロトコルスタックを起動します。
  - Modbus TCP サーバモードのみ、シリアル通信を使用しないため、構造体引数に対して NULL を渡します。

- Modbus ファンクションコードのコールバック関数実装サンプルの初期化
  - Modbus ファンクションコードのコールバック関数サンプルがアクセスする Modbus データモデルを初期化します。Modbus データモデルの実装例は、mbapp_slave/address.c (.h) にて示します。

4.6.3 スレーブ（サーバ）モード用実装

以下のファイルは、Modbus プロトコルスタックがスレーブ（サーバ）モードとして動作するビルド構成（RTU_SLAVE_UC3、ASCII_SLAVE_UC3、TCP_SERVER_UC3）を指定した場合に、ビルド対象となります。

4.6.3.1 mbapp_slave/function_code.c (.h)

マスタから受信したリクエスト上の Modbus ファンクションコードに基づき各種動作を処理するコールバック関数の実装例を記述しています。

- Modbus プロトコルスタックは、表 4.4 のファンクションコードに対応するコールバック関数を登録できます。本ソースファイルは、全てのコールバック関数の実装例を示しています。
- Modbus プロトコルは固有のデータモデルを持ち、データモデルは 4 つのデータ型と、各データ型に対応するアドレス空間で構成される。本プロトコルスタックが対応する Modbus データ型を表 4.5 に示します。
- ファンクションコードを処理するコールバック関数は、各々対応するデータ型にアクセスする処理が要求されます。
- Modbus プロトコルにおいて、スレーブは各データ型に対して最大 65536（0x10000）個のデータを持つことができ、それぞれ 1~65536(0x00001~0x10000)の範囲で参照アドレスを割り振る事が出来ます。
- また、参照アドレスは任意の物理アドレスを参照することが出来ます。
- 本 Modbus プロトコルスタックは、ユーザによる任意の Modbus データモデルの設計をサポートするために、それらの Modbus データモデルにアクセスするファンクションコード処理を、コールバック関数として実装します。

表 4.4 Modbus プロトコルスタックサンプルプログラム対応 Modbus ファンクションコード

Code	Code (Hex)	Function Name	Description
1	1h	Read Coils	指定した複数の Coil アドレスのデータを読み出す
2	2h	Read Discrete Inputs	指定した複数の Discrete Inputs アドレスのデータを読み出す
3	3h	Read Holding Registers	指定した複数の Holding Registers アドレスのデータを読み出す
4	4h	Read Input Registers	指定した複数の Input Registers アドレスのデータを読み出す
5	5h	Write Single Coil	指定した単一の Coil アドレスにデータを書き込む
6	6h	Write Single Register	指定した単一の Holding Register アドレスにデータを書き込む
15	Fh	Write Multiple Coils	指定した複数の Coil アドレスにデータを書き込む
16	10h	Write Multiple Registers	指定した複数の Holding Register アドレスにデータを書き込む
23	17h	Read/Write Multiple Registers	指定した複数の Holding Register アドレスにデータを書き込んだ後、別に指定した複数の Holding Register アドレスのデータを読み出す

表 4.5 Modbus プロトコルスタックサンプルプログラム対応 Modbus データ型

Name	Bits	Type of access	Description
Discrete Inputs	1	Read	I/O システムによって提供できるデータの型
Coils	1	Read/Write	アプリケーションプログラムによって変更できるデータ型
Input Registers	16	Read	I/O システムによって提供できるデータの型
Holding Registers	16	Read/Write	アプリケーションプログラムによって変更できるデータ型

(Continued on next page)

Modbus ファンクションコードのコールバック関数すべき処理は以下の 2 つとなります。

1. ポインタ引数のリクエスト構造体を参照して、ポインタ引数のレスポンス構造体にファンクションコードの処理結果をセットする。
2. 設計された Modbus データモデルが持たない参照アドレスが参照された場合、例外コード 0x02 をレスポンス構造体にセットする。

また、ファンクションコード処理中に発生した回復不能なエラーに対して、例外コード 0x04 をセットすることができます。

表 4.6　ファンクションコードコールバック関数で実装する例外コード

Code	Code (Hex)	Function Name	Description
2	2h	Illegal Data Address	指定アドレス値、もしくは指定アドレス範囲に、設計された Modbus データモデルにとって不正なアドレスが含まれる場合に返す。
4	4h	Slave Device Failure	ファンクションコードの処理中に回復不能なエラーが発生した場合に返す。

4.6.3.2 mbapp_slave/address.c (.h)

本ソースファイルは、Modbus データモデルの実装例を記述しています。各々の Modbus データ型の参照アドレス設計、および物理メモリマッピングを図 4.2 に示します

- 各 Modbus データ型は、32768 個のデータを持ち、参照アドレス 0x0001~0x8000 を使用します（※1）。
- 参照アドレスは、静的グローバル変数の配列バッファとして RAM メモリに割り当てます。
- Modbus ファンクションコードのコールバック関数で使用される、各 Modbus データ型の各配列バッファにアクセス（Read/Write）する関数を実装しています。
- Read/Write アクセス関数は、特定の参照アドレスにアクセスした場合、バッファ配列と RX72M の周辺機能レジスタへリンクします（※2）。
  - Coils アドレス 0x0001~0x0004 番地 ： LED に対応する GPIO ポート
  - Discrete Input アドレス 0x0001~0x0008 番地：SW5 に対応する GPIO ポート
- 各 Modbus データアドレスの 0x5002 番地にアクセスした場合に、回復不能なエラーが発生したとして、例外コード 0x04：SLAVE DEVICE FAILURE を返します（※3）。
- 各 Modbus データ型で指定したアドレス範囲に、不正なアドレスが含まれているかをチェックする関数を実装しています。
- 各 Modbus ファンクションコードに対応するコールバック関数は、これらのチェック関数を使用することで、バッファ配列にアクセスする前に、不正なアドレスを検知して例外コード 0x02：ILLEGAL DATA ADDRESS を返すことができます（※4）。

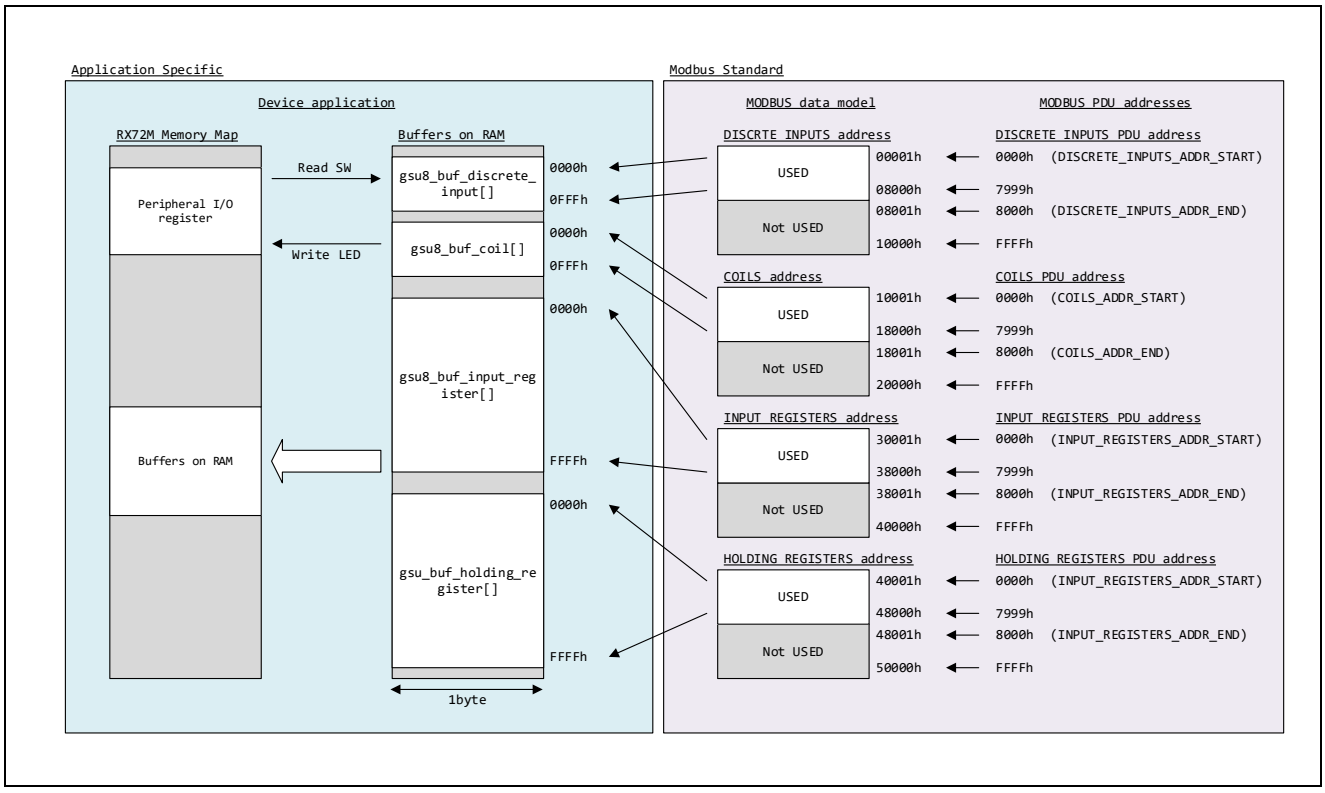

図 4.2　Modbus data model とメモリマッピング

（次ページへ続く）

※1

使用する参照アドレスの範囲は、各 Modbus データ型に対応する定数マクロで指定できます。

- 例えば、COILS_ADDR_START、COILS_ADDR_END 定数マクロは、データ型 Coils の参照アドレスの開始アドレスと終了アドレスを指定できます。
- デフォルトでは、各 Modbus データアドレスに開始アドレスとして 0x0000、終了アドレスとして 0x8000 を指定しています。
- ここで指定されるアドレス値は、0x0001 開始ではなく、0x0000 開始であることに注意します。
- これは、Modbus リクエストを構成するプロトコルデータユニット（PDU）に記載されているアドレス値表現に準拠するためです。以下、これアドレス表現を PDU アドレスと呼びます。
- このサンプルプログラム上では、参照アドレスを全て PDU アドレスで処理をしています。
- また、COILS_ADDR_END などの有効アドレス終了指定値は排他としており、指定した値のアドレスは無効なアドレスとなることに注意します。

※2

本サンプルプログラムでは、RX72M 周辺機能レジスタへ割り当てられる参照アドレスは、それぞれ COILS_ADDR_START、および DISCRETE_INPUT_ADDR_START との相対値で決定しています。

- 例えば、COILS_ADDR_START を 0x1000 にした場合、Coil アドレス 0x1001~0x1004 番地が、CPU カード上の LED に対応する GPIO ポートとリンクします。

※3

例外コード 0x04 を返すアドレス値は、定数マクロ PUSEDO_SLAVE_FAILURE_ADDR で指定できます。指定値は PDU アドレスであることに注意します。

※4

不正アドレスのチェック関数を変更することで、任意のアドレスを不正なアドレスとして設計することができます。

4.6.4 アプリケーション用 IO ポート実装

4.6.4.1 ioport.c/.h

CPU カードの LED、ジャンパスイッチ（JP5）に対応する GPIO ポートの処理を記述しています。

Modbus プロトコルスタックがスレーブモードで動作する場合は、address.c に実装されている Modbus データアドレスアクセス関数から操作されます。

Modbus プロトコルスタックがマスタモードで動作する場合は、main.c に実装されているマスタプログラム関数から操作されます。

4.7 その他

4.7.1 μC3/μNet3 製品版の導入について

μC3/μNet3 評価版（無償版）は一部動作に制限を含みます。製品版（有償版）の入手は、下記 Web サイトよりイー・フォース社へお問い合わせください。

- https://www.eforce.co.jp/

本サンプルアプリケーションでは、表 4.7 左列に記載するμC3/μNet3 の評価版ライブラリファイルを、表 4.7 右列にμC3/μNet3 製品版のライブラリファイルに差し替えることで、μC3/μNet3 製品版を適用する事ができます。

表 4.7 μC3/μNet3 ライブラリ

	差し替え前のファイル（評価版）	差し替え後のファイル（製品版）とその格納場所
uC3	{project_root}\uC3\ uC3RXv3b.lib uC3Rxv3l.lib uC3RXv3rbb.lib uC3Rxc3rbl.lib	{install_root}\Kernel\Standard\lib\RXv3\e2studio uC3RXv3b.lib uC3Rxv3l.lib uC3RXv3rbb.lib uC3Rxc3rbl.lib
uNet3	{project_root}\uNet3\ uNet3BSDRXv3b.lib uNet3BSDRXv3l.lib uNet3RXv3b.lib uNet3RXv3l.lib	{install_root}\Network\TCPIP\lib\RXv3\e2studio uNet3BSDRXv3b_Std.lib uNet3BSDRXv3l_Std.lib uNet3RXv3b_Std.lib uNet3RXv3l_Std.lib

{project_root}：本サンプルアプリケーションのルートフォルダです。

{install_root}：RX72M 向けμC3/μNet3 製品版のインストールフォルダです。

5. サンプルアプリケーションによる通信のテスト

5.1 ハードウェア接続

Modbus プロトコルスタックはスタックモードにより、ハードウェア接続の仕方が異なります。

5.1.1 Modbus TCP サーバスタックモード

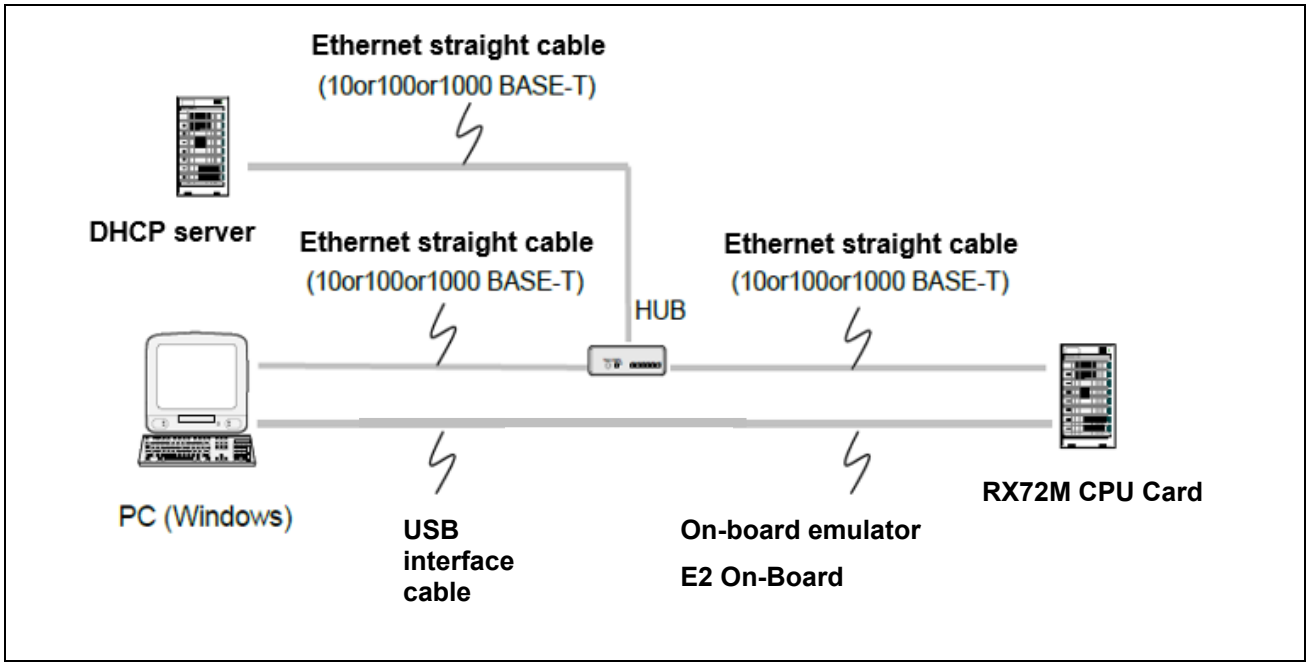

図 5.1Modbus TCP サーバスタックモード時のハードウェア接続例

本サンプルプログラムでは、IP アドレスは DHCP サーバから自動で取得します。

一定時間経過後に、固定 IP（192.168.1.103）に切り替わります。

5.2 サンプルアプリケーションの起動方法

サンプルアプリケーションによる通信テストを実行する際のアプリケーション操作方法を説明します。

事前に 5.1 ハードウェア接続を参考にサンプルアプリケーションにて動作させるプロトコルスタックモードに合わせてハードウェア接続を完了させてください。

1) e2studio を起動後、「ファイル」→「インポート」をクリックします。
2) 「選択」ダイアログで「一般」→「既存プロジェクトをワークスペースへ」を選択し「次へ」をクリックします。

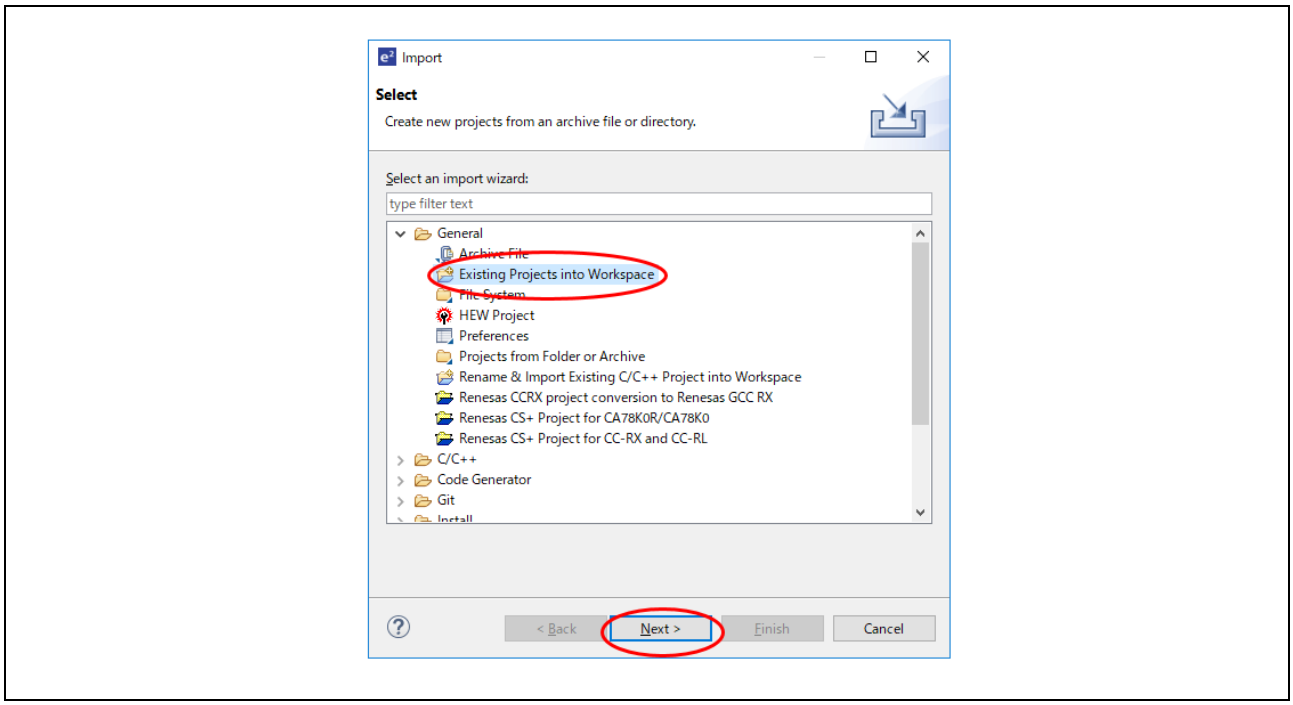

図 5.2プロジェクトのインポート

3) 「プロジェクトのインポート」ダイアログの「アーカイブファイルの選択」チェックボックスを選択し、「参照」をクリックします。” rx72m_cpucard_modbus_eva.zip”を選択し「開く」をクリック。「終了」をクリックしプロジェクトのインポートを完了します。

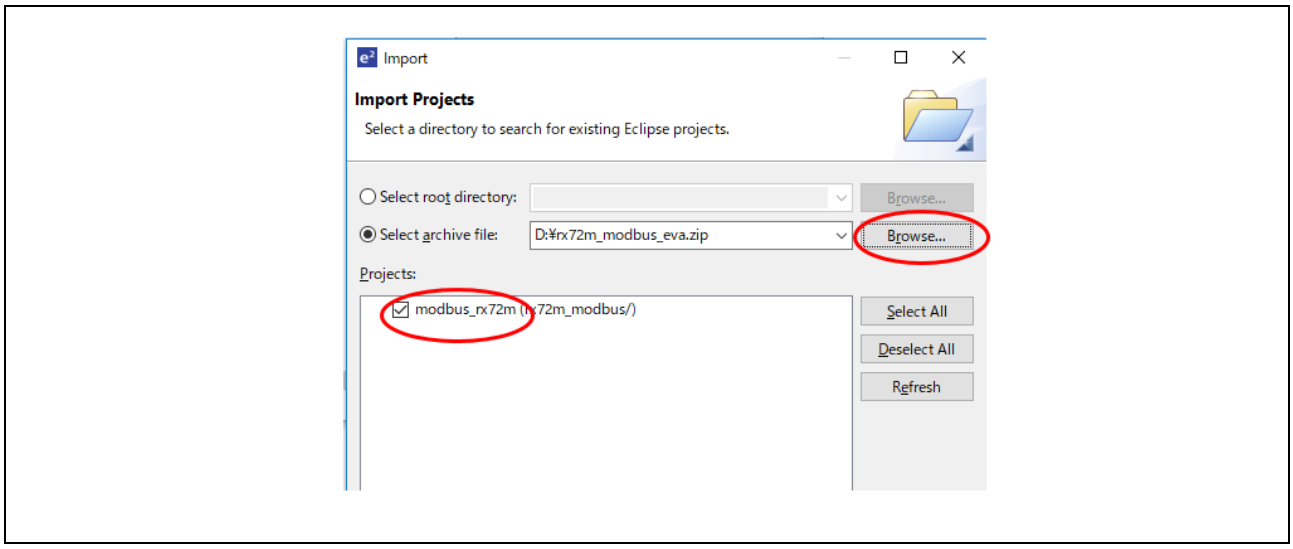

図 5.3アーカイブファイルの選択

4) サンプルプロジェクトを右クリックし、「ビルド構成」の「アクティブにする」から
“TCP_SERVER_UC3”がアクティブとなっていることを確認し、ビルドを実行します。

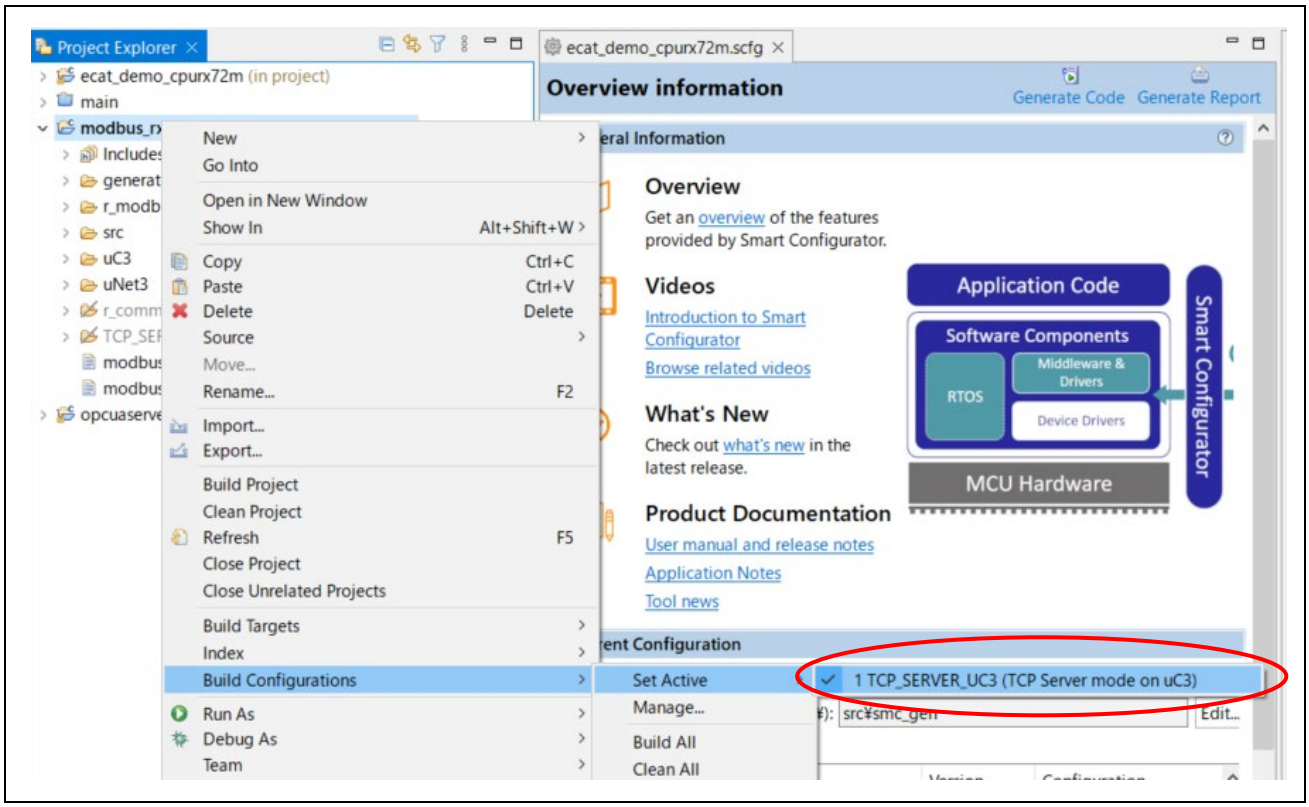

図 5.4ビルド構成設定

5) デバッグは「実行」の「デバッグの構成」から起動します。

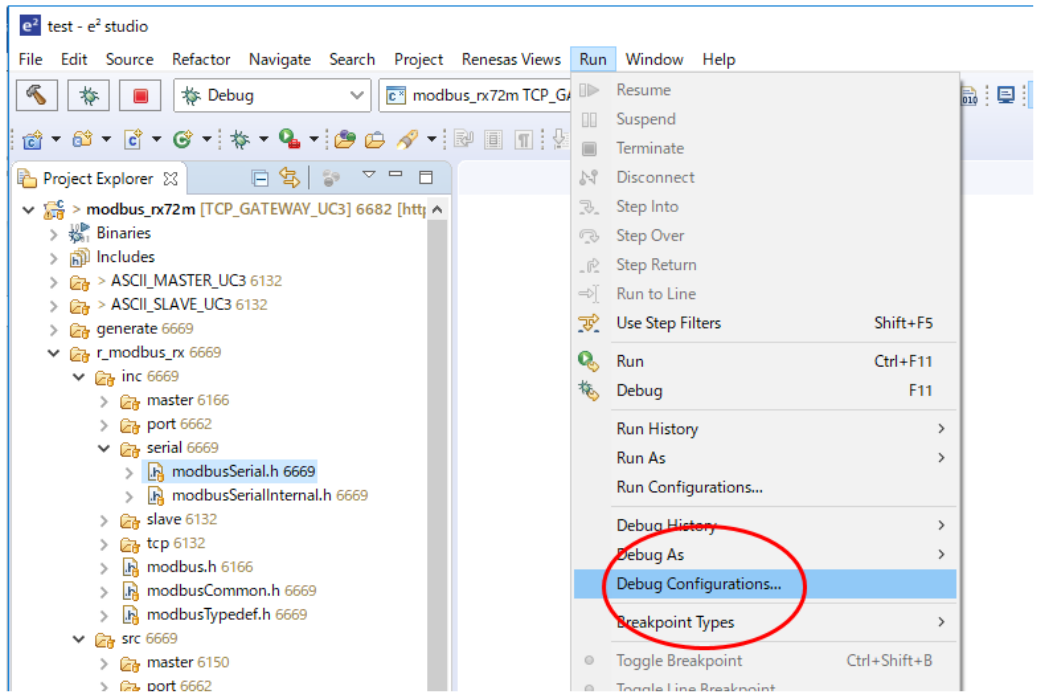

図 5.5デバッグの構成の選択

6) ビルド構成に合わせた「デバッグ構成」を Renesas GDB Hardware Debugging 以下から選択、「デバッグ」ボタンを押下しデバッガを起動します。

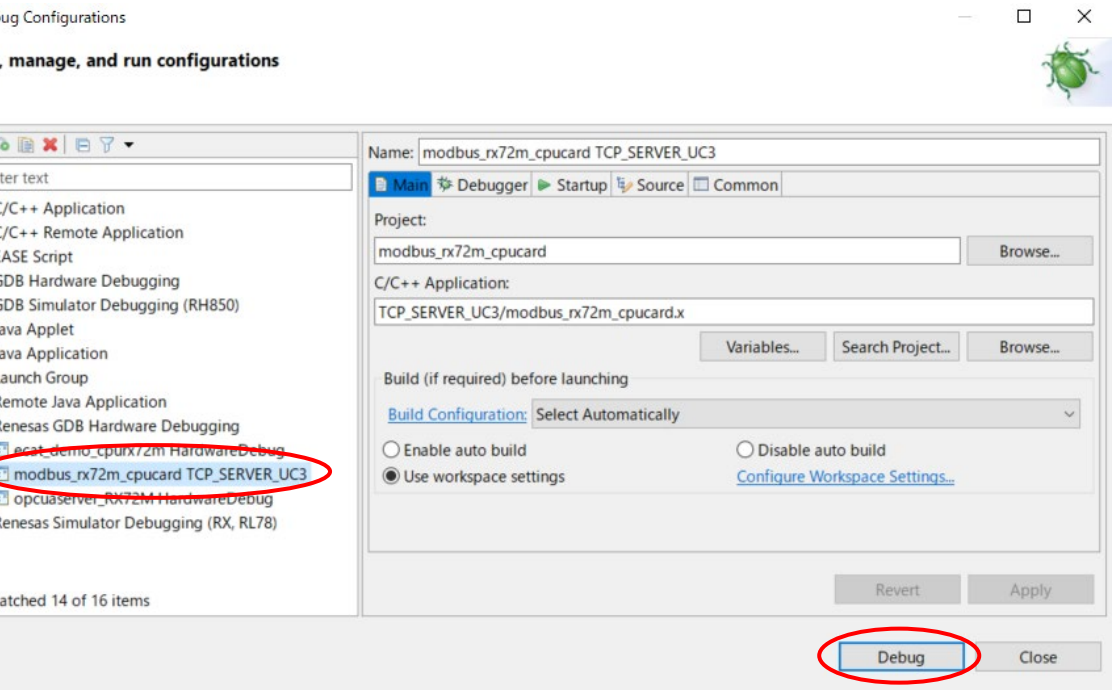

図 5.6デバッガ起動

7) 「リスタート」ボタンを押下後、「再開」ボタンを押下しサンプルアプリケーションが動作します。

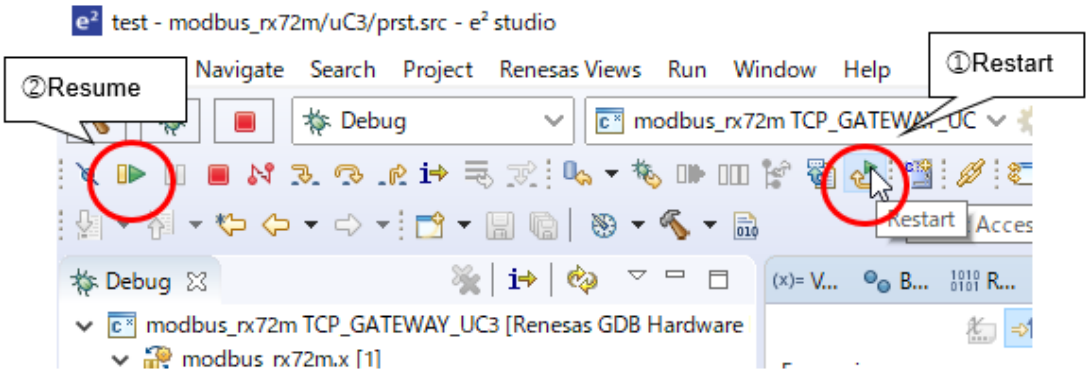

図 5.7サンプルアプリケーション動作開始

5.3 評価用ツールを用いた通信テスト

評価用ツール（ModbusDemoApplication.exe）を使用して、Modbus プロトコルスタックサンプルプログラム、および Modbus アプリケーションサンプルプログラムの動作を確認します。

5.3.1 評価用ツールの動作概要

評価用ツールは PC 上で動作し、RX72M CPU カード上で動作する Modbus アプリケーションサンプルプログラムの対向ソフトウェアとして動作します。

RX72M CPU カードは、評価用ツールとの Modbus 通信に応じて JP5 の状態の送信、もしくは LED の点灯の制御を行います。LED に対応する Coil アドレスを表 5.3 に、SW に対応する Discrete Input アドレスを表 5.4 に示します。

表 5.1 Coil アドレスと対応する LED

Coil アドレス	対応する LED
0001h	LED5
0002h	LED4
0003h	LED3
0004h	LED2

表 5.2 Discrete Input アドレスと対応する SW

Discrete Input アドレス	対応する SW
0001h	JP5 1-4
0002h	JP5 2-5
0003h	JP5 3-6

（次ページへ続く）

サンプルアプリケーションが Modbus スレーブの場合、評価用ツールは Modbus マスタとして動作し、以下の手順で動作をできます。

1. 評価用ツールは、Read Discrete Inputs リクエストを送信します。
2. サンプルアプリケーションは、Read Discrete Inputs リクエストを受信し、RX72M CPU カード JP5 の状態をレスポンスとして送信します。
3. 評価用ツールは、Read Discrete Inputs リクエストのレスポンスとして受信した JP5 の状態に応じて、RX72M CPU カードの LED の更新方法を決定し、Write Multiple Coils リクエストを送信します。

   JP5 1-4 オープン ：一定間隔で点灯する LED を変更する。

   JP5 1-4 ショート ：評価用ツールの Coils テキストボックスで指定した値を LED に反映する。
4. サンプルアプリケーションは Write Multiple Coils リクエストを受信し、指定された RX72M CPU カード LED の点灯状態を更新します。

サンプルアプリケーションが Modbus マスタの場合、評価用ツールは Modbus スレーブとして動作し、以下の手順で動作を確認できます。

1. サンプルアプリケーションは、Read Coils リクエストを 1 秒間隔で送信します。
2. 評価用ツールは、Read Coil リクエストを受信し、Coils テキストボックスに設定した値を、レスポンスとして送信します。
3. サンプルアプリケーションは、Read Coils リクエストのレスポンスとして受信した値を、LED に反映します。

5.3.2 評価用ツールの使用方法

“Connection”で動作モードを選択し、各種パラメータを設定してください。

- Connection
  - TCP server を選択します。

- Serial setting
  - 設定は不要です。

- Remote Modbus Server
  - デバイスに合わせて IP アドレス、ポート番号を設定します。
  - 固定 IP アドレスで接続する場合、下記に設定してください。
    IP Address: 192.168.1.103　Port: 502

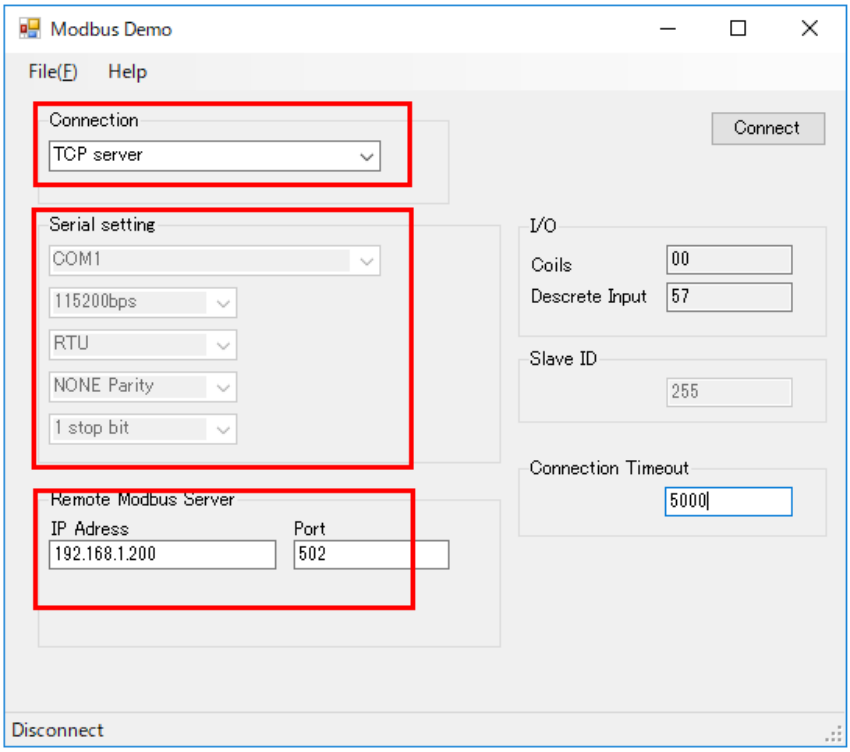

図 5.8評価用ツール画面

（次ページへ続く）

評価ツールが、Modbus マスタとして動作する場合、

- RX72M CPU カードの JP5 1-4 がオープンの場合、Coils テキストボックスが自動的に変更されます。
- RX72M CPU カードの JP5 1-4 がショートの場合、Coils テキストボックスが手動で変更できます。
- Coils テキストボックスの値が、一定間隔で RX72M CPU カード上の LED に反映されます。

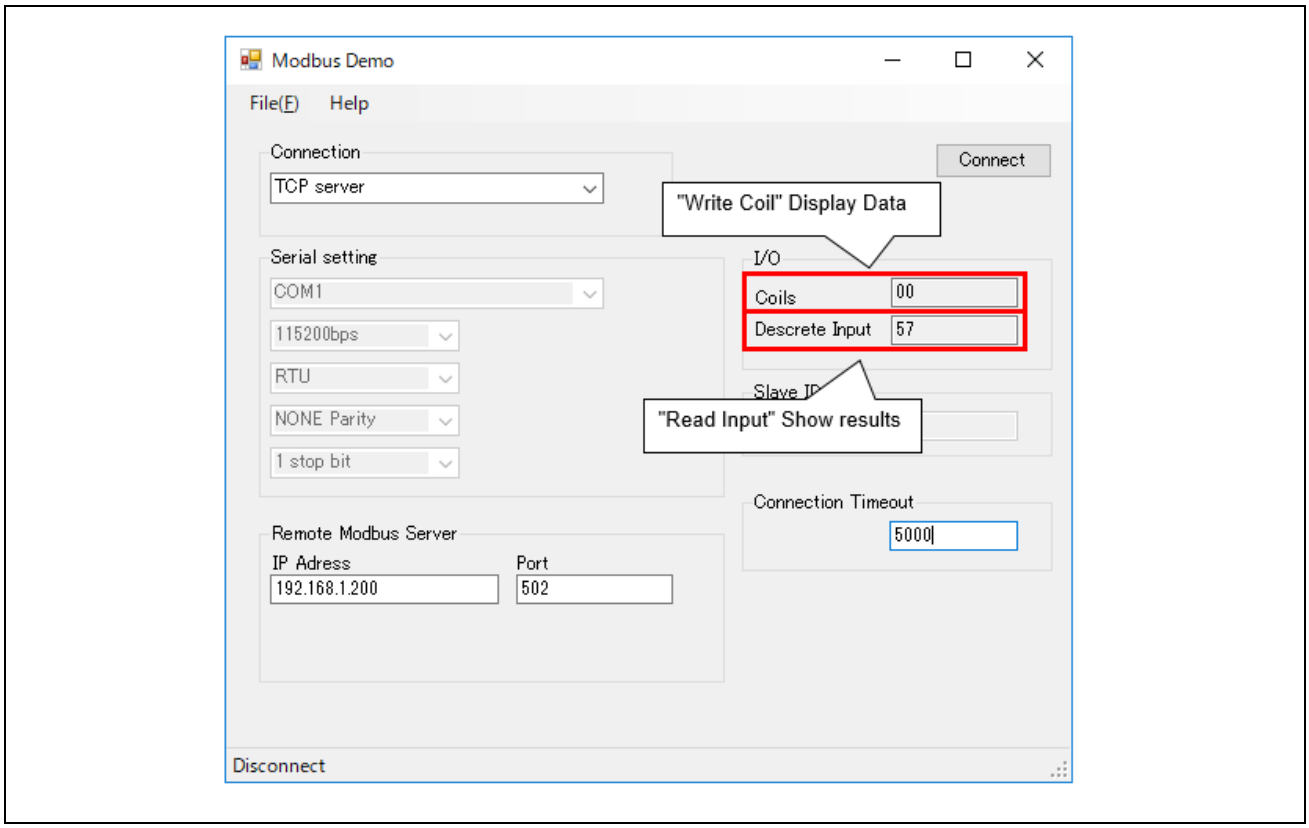

図 5.9Modbus マスタとして動作する評価ツール

（次ページへ続く）

評価ツールが Modbus スレーブとして動作する場合、

- Coils テキストボックスの値を、手動で変更できます。
- Coils テキストボックスの値が、一定間隔で RX72M CPU カード上の LED に反映されます。

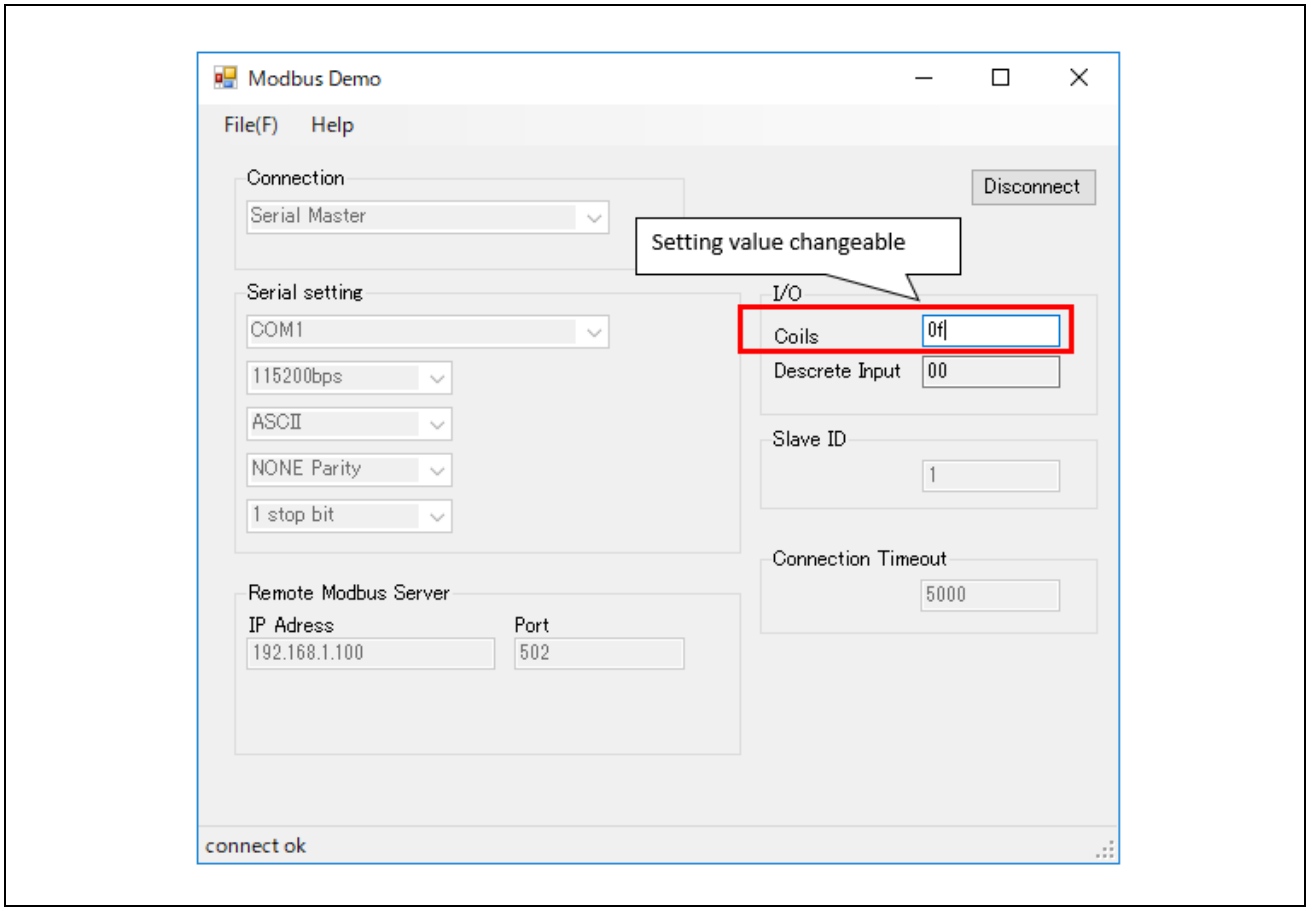

図 5.10 Modbus スレーブとして動作する評価ツール

注意 サンプルアプリケーション側がサーバとなりますので、スレーブデバイスの動作開始前に通信可能状態にしておいてください。

6. RX72M 用 TCP/IP スタックの基本概念

RX72M 用 Modbus TCP スタックは、イーサネットネットワークインタフェースに、イー・フォース社製の TCP/IP プロトコルスタック「μNet3」を使用しています。これは、同じくイー・フォース社製の RTOS「μC3」向けに実装される TCP/IP プロトコルスタックです。

6.1 μNet3 モジュール構成

μNet3 は「アプリケーションインタフェース」「TCP/IP プロトコルスタック」「ネットワークデバイス制御 API」の 3 つのモジュールから構成されています。

- アプリケーションインタフェース
  リモートホストとの接続の確立、データの送信・受信等といった様々なネットワークサービスを利用するためのインタフェース（API）を提供します。
- TCP/IP プロトコルスタック
  TCP、UDP、ICMP、IGMP、IP、ARP といったネットワークプロトコルを処理します。
- ネットワークデバイス制御 API
  ネットワークシステムには様々なネットワークデバイスが存在している可能性があり、デバイス毎にデバイスドライバが必要になります。ネットワークデバイス制御 API は、これらのデバイスの違いを吸収し、統一的にアクセスするためのインタフェースを提供します。アプリケーションプログラムからデバイス番号を使用して各デバイスにアクセスします。

詳細は、次のドキュメントを参考にして下さい。

- uNet3_UsersGuide.pdf：μNet3 ユーザーズガイド
- uNet3_DriverGuide.pdf：μNet3Ethernet ドライバインタフェース
- uNet3_BSD_UsersGuide.pdf：μNet3/BSD ユーザーズガイド

6.2 開発手順

ネットワークのコンフィグレーション（IP アドレス、ソケット定義）、デバイスドライバのコンフィグレーション（MAC アドレス、デバイス I/F 定義、ドライバ固有のフィーチャ）、μNet3 初期化ルーチン呼び出しなどのコンフィグレーションを net_cfg.c ソースファイルに、コンフィグレーション定義を net_cfg.h ヘッダファイルにそれぞれ記述します。

6.2.1 コンフィグレーション定義一覧

表 6.1 にコンフィグレーション可能なパラメータと Modbus プロトコルスタックにおける設定値の一覧を示します。

表 6.1　コンフィグレーション定義一覧

コンフィグレーション定義	値	内容
CFG_NET_DEV_MAX	1	データリンクデバイス数
CFG_NET_SOC_MAX	10	使用ソケットの上限数
CFG_NET_TCP_MAX	8	使用 TCP ソケットの上限数
CFG_NET_ARP_MAX	8	ARP エントリ数
CFG_NET_MGR_MAX	8	マルチキャストエントリ数
CFG_NET_IPR_MAX	2	IP 再構築キュー数

CFG_NET_BUF_SZ	1576	ネットワークバッファサイズ
CFG_NET_BUF_CNT	8	ネットワークバッファ数
CFG_NET_BUF_OFFSET	2	ネットワークバッファデータ書き込み位置
CFG_PATH_MTU	1560	MTU サイズ
CFG_ARP_RET_CNT	3	ARP リトライ回数
CFG_ARP_RET_TMO	1*1000	ARP リトライタイムアウト
CFG_ARP_CLR_TMO	20*60*1000	ARP キャッシュクリアタイムアウト
CFG_IP4_TTL	64	IP ヘッダ TTL 値
CFG_IP4_TOS	0	IP ヘッダ TOS 値
CFG_IP4_IPR_TMO	10*1000	IP リアセンブルパケット待ち時間
CFG_IP4_MCAST_TTL	1	IP ヘッダ TTL（マルチキャストパケット）
CFG_IGMP_V1_TMO	400*1000	IGMPV1 タイムアウト
CFG_IGMP_REP_TMO	10*1000	IGMP レポートタイムアウト
CFG_TCP_MSS	1460	MSS
CFG_TCP_RTO_INI	3	TCP リトライタイムアウト初期値
CFG_TCP_RTO_MIN	500	TCP リトライタイムアウト最小値
CFG_TCP_RTO_MAX	60	TCP リトライタイムアウト最大値
CFG_TCP_SND_WND	1024	送信バッファサイズ
CFG_TCP_RCV_WND	1024	受信バッファサイズ（Window サイズ）
CFG_TCP_DUP_CNT	4	リトライ開始重複 ACK 数
CFG_TCP_CON_TMO	75*1000	SYN タイムアウト
CFG_TCP_SND_TMO	64*1000	送信タイムアウト
CFG_TCP_CLS_TMO	75*1000	FIN タイムアウト
CFG_TCP_CLW_TMO	20*1000	Close Wait タイムアウト
CFG_TCP_ACK_TMO	200	ACK タイムアウト
CFG_TCP_KPA_CNT	0	切断するまでの KeepAlive 通知回数
CFG_TCP_KPA_INT	1*1000	KeepAlive を開始後の通知間隔
CFG_TCP_KPA_TMO	7200*1000	KEepAlive を開始するまでの無通信時間
CFG_PKT_RCV_QUE	1	受信パケットキューイング数
CFG_TCP_RCV_OSQ_MAX	6	受信シーケンス保障キューイング数
CFG_PKT_CTL_FLG	0x0000	受信パケットチェックサム検証無効フラグ
CFG_ARP_PRB_WAI	1*1000	net_acd()実行時の ARP Probe 送信待ち時間
CFG_ARP_PRB_NUM	3	net_acd()実行時の ARP Probe 送信回数
CFG_ARP_PRB_MIN	1*1000	次の ARP Probe 送信までの最小待ち時間
CFG_ARP_PRB_MAX	2*1000	次の ARP Probe 送信までの最大待ち時間
CFG_ARP_ANC_WAI	2*1000	ARP Probe を送信してからの検出待ち時間
CFG_ARP_ANC_NUM	2	ARP Announce の送信回数
CFG_ARP_ANC_INT	2*1000	次の ARP Announce の送信までの待ち時間

6.2.2 プロトコルスタックの初期化

イー・フォース社製の TCP/IP プロトコルスタック「μNet3」を使用するには、プロトコルスタックの初期化とネットワークデバイスの初期化が必要になります。

以下に初期化のサンプルを示します。

- プロトコルスタックの初期化

```
ercd = net_ini();
if (ercd != E_OK) {
  return ercd;
}
```

- ネットワークデバイス（デバイス番号 N）の初期化

```
ercd = net_dev_ini(N);
if (ercd != E_OK) {
  return ercd;
}
```

初期化コードは、net_cfg.c の net_setup 関数に実装しています。

6.3 Ethernet ドライバインタフェース

ネットワークデバイス上にμNet3 プロトコルスタックを実装するためのポーティング方法とネットワークデバイスドライバの開発に必要なインタフェースを説明します。

6.3.1 ファイル構成

ネットワークデバイスドライバを実装するにあたり、以下のファイルにソースコードを記述します。

- net_cfg.c
  μNet3 のコンフィグレーションやネットワークデバイスドライバとの I/F を記述します。
- DDR_RX_ETH0.c / DDR_RX_ETH0.h
  ネットワークデバイスドライバ本体です。デバイスの名前等を記述します。本ファイルは新しくネットワークデバイスドライバを開発する場合は新たに準備する必要があります。
- DDR_RX_ETH0_cfg.h
  PHY ID や MII/RMII モードなどネットワークデバイスドライバのコンフィグレーションマクロを記述します。本ファイルは新しくネットワークデバイスドライバを開発する場合は新たに準備する必要があります。

6.3.2 インタフェース

μNet3 はデバイス番号を使ってネットワークデバイスにアクセスします。ネットワークデバイスドライバは、デバイスの初期化関数、フレームの送信関数、デバイスの解放関数、デバイスの制御関数、デバイスの状態取得関数を自身のデバイス番号が指す gNET_DEV[ ]（*1）に設定して実装します。

また、ネットワークデバイスから受信したフレームをμNet3 で処理するためには、ネットワークデバイスドライバで適切な受信処理関数を実装する必要があります。

(*1) μNet3 はすべてのネットワークデバイスドライバの制御を gNET_DEV[ ]（デバイスオブジェクト）によって管理します。gNET_DEV[ ]はデバイス番号順の T_NET_DEV 型の配列で、net_cfg.c ファイルに定義します。

- デバイスの初期化関数：ER eth_ini(UH dev_num)
  アプリケーションがデバイスの初期化関数 net_devini(デバイス番号)API を呼び出すと、μNet3 はデバイス番号に該当するネットワークデバイスドライバの初期化関数を呼び出します。

  デバイス番号に該当する gNET_DEV[ ]のメンバ「ini」に設定して下さい。

- フレーム送信関数：ER eth_snd(UH dev_num, T_NET_BUF *pkt)
  アプリケーションが送信ソケット snd_soc()API を呼んだ場合や、μNet3 がパケットを送信する場合、μNet3 はデバイス番号に該当するネットワークデバイスドライバの送信関数を呼び出します。

  デバイス番号に該当する gNET_DEV[ ]のメンバ「out」に設定して下さい。

- デバイスの解放関数：ER eth_cls(UH dev_num)
  アプリケーションがデバイスの解放関数 net_dev_cls(デバイス番号)API を呼び出すと、μNet3 はデバイス番号に該当するネットワークでナイスドライバの解放関数を呼び出します。

  デバイス番号に該当する gNET_DEV[ ]のメンバ「cls」に設定して下さい。

- デバイスの制御関数：ER eth_ctl(UH dev_num, UH opt, VP val)
  アプリケーションがデバイスの制御関数 net_dev_ctl(デバイス番号、制御コード、設定値)API を呼び出すと、μNet3 はデバイス番号に該当するネットワークデバイスドライバの制御関数を呼び出します。

  デバイス番号に該当する gNET_DEV[ ]のメンバ「ctl」に設定して下さい。

- デバイスの状態取得関数：ER eth_ref(UH dev_num, UH opt, VP val)
  アプリケーションがデバイスの状態取得関数 net_dev_sts(デバイス番号、制御コード、取得値)API を呼び出すと、μNet3 はデバイス番号に該当するネットワークデバイスドライバの状態取得関数を呼び出します。

  デバイス番号に該当する gNET_DEV[ ]のメンバ「ref」に設定して下さい。

- フレームの受信処理：net_pkt_rcv()
  ネットワークデバイスから受信したフレームをμNet3 に転送するためには、以下の手順で net_pkt_rcv()API を呼び出します。使用するネットワークバッファは、μNet3 側で解放されます。

1) net_buf_get()API を使用して、ネットワークバッファ（T_NET_BUF *）を取得します。
2) 自身の gNET_DEV[ ]をネットワークバッファに設定します。
3) 受信したフレームをネットワークバッファに書き込みます。
4) 書き込んだフレームの先頭アドレス（ヘッダアドレス）をネットワークバッファに設定します。
5) 書き込んだフレームのデータアドレスをネットワークバッファに設定します。
6) 書き込んだフレームのヘッダサイズ（Ether ヘッダサイズ）をネットワークバッファに設定します。
7) 書き込んだフレームのデータサイズ（フレームサイズ - ヘッダサイズ）をネットワークバッファに設定します。
8) ネットワークバッファを引数に net_pkt_rcv()を呼び出します。

6.4 μNet3/BSD

μNet3/BSD は、μNet3 上で BSD アプリを動作させるために BSD インタフェースを提供するものです。μNet3/BSD を使用することにより、Linux や BSD で動作するソケットアプリは、μNet3 上でシームレスに動作可能となります。

μNet3/BSD は、4.4BSD-Lite 相当のソケット API を提供します。

μNet3/BSD を使用することでアプリケーションは、BSD ソケット API と、μNet3 独自 API を使用することができます。

- ソケット API の多重呼び出し
- select()関数
- ループバックアドレス
- ソケット単位のマルチキャストグループ
- TCP ソケットの Listen キュー
- ソケットエラー

6.4.1 シンボル名の互換性

コンパイラ環境によるシンボル衝突を避けるためμNet3/BSD が提供する API、構造体、マクロには独自接頭辞“unet3_”が冠してあります。

アプリケーションは/bsd/unet3_posix/unet3_socket.h をインクルードすることにより、アプリケーション内で使用している POSIX 標準のシンボル名がこれら独自接頭辞付きのシンボルに置き換わります。そのため BSD ソケットを使用するアプリケーションはそのままのソースファイルでμNet3/BSD で動作します。

6.4.2 ソケット API

イー・フォース社製の TCP/IP プロトコルスタック「μNet3/BSD」が提供する API 一覧を表 6.2 に示します。

表 6.2 ソケット API 一覧

μNet3 API	μNet3/BSD API	機能
unet3_bsd_init		μNet3/BSD を初期化する
unet3_socket	socket	新しいソケットの作成
unet3_bind	bind	ソケットに名前を付ける
unet3_listen	listen	ソケット上の接続を待つ
unet3_accept	accept	ソケットへの接続を受ける
unet3_connect	connect	ソケットの接続を行う
unet3_send	send	ソケットへメッセージを送る
unet3_sendto	sendto	ソケットへメッセージを送る
unet3_recv	recv	ソケットからメッセージを受け取る
unet3_recvfrom	recvfrom	ソケットからメッセージを受け取る
unet3_shutdown	shutdown	全二重接続の一部を閉じる
unet3_close	close	ソケットを閉じる
unet3_getsockname	getsockname	ソケットの名前を取得する
unet3_getpeername	getpeername	接続している相手ソケットの名前を取得する
unet3_getsockopt	getsockopt	ソケットのオプションの取得を行う
unet3_setsockopt	setsockopt	ソケットのオプションの設定を行う
unet3_inet_aton	inet_aton	インタネットアドレス操作ルーチン
unet3_inet_addr	inet_addr	インタネットアドレス操作ルーチン
unet3_inet_ntoa	inet_ntoa	インタネットアドレス操作ルーチン
unet3_rresvport	rresvport	ポートにバインドされたソケットを取得する
unet3_getifaddrs	getifaddrs	インタフェースのアドレス取得
unet3_freeifaddrs	freeifaddrs	インタフェース情報の開放

6.4.3 BSD アプリケーションの設定

1) ソースコード

uNet3/直下の 4 つのソースコードをプロジェクトに取り込みます。

- unet3_lodev.c 仮想ループバックデバイス
- unet3_option.c ソケットオプション関数群
- unet3_socket.c BSD ソケット API
- unet3_wrap.c μNet3/Wrapper タスク

また、uNet3 本体となるライブラリも BSD 向けのライブラリをリンクします。

- uNet3BSDRXv3l.lib BSD 用のμNet3 ライブラリ

2) インクルードパス

インクルードパスの設定を追加します。

ヘッダファイルは POSIX 準拠のファイルも含めて uNet3/bsd/unet3_posix/フォルダ配下にあります。

3) コンフィグレーション

アプリケーションで使用する最大ソケット数とアプリケーションのタスク数を unet3_cfg.h にマクロ定義します。

- 最大ソケット数

```
#define   BSD_SOCKET_MAX          CFG_NET_SOC_MAX
```

- アプリケーションのタスク数

```
#define   NUM_OF_TASK_ERRNO       (CFG_TASK_MAX+1)
```

4) リソース定義

μNet3/BSD を動作させるために必要なリソースを用意します。リソースはμNet3/BSD が情報を管理するためのテーブルです。

- BSD ソケット管理テーブル

```
T_UNET3_BSD_SOC  gNET_BSD_SOC[ BSD_SOCKET_MAX ];
```

- エラー番号管理テーブル

```
UW  tsk_errno[ NUM_OF_TASK_ERRNO ];
```

5) カーネルオブジェクト
μNet3/BSD が使用するカーネルオブジェクトは表 6.3 のとおりです。

表 6.3 カーネルオブジェクト一覧

リソース名	用途	ID
タスク	BSD Wrapper タスク	ID TSK_BSD_API
	ループバックデバイスタスク	ID_LO_IF_TSK
メールボックス	BSD Wrapper タスク間通信	ID MBX_BSD_REQ
	ループバックデバイスタスク間通信	ID_LO_IF_MBX
メモリプール	メッセージバッファ	ID MPF_BSD_MSG

6) 初期化
アプリケーションはソケット API を使用する前に unet3_bsd_init()関数を呼び出してμNet3/BSD モジュールを初期化する必要があります。尚、μNet3/BSD を初期化するにはμNet3 の初期化とデバイスドライバの初期化が正常に終了している必要があります。

7. RX72M 用 Modbus スタックの基本概念

7.1 設計手法

1. ネットワーク上である機能を実現するために必要な機能を選択し、階層状に積み上げたソフトウェア群をプロトコルスタックと呼びます。本スタックは、図 7.1 に示すような構造になっています。
2. 本スタックは、イー・フォース社製の RTOS「μC3」を使用してタスクを作成します。スタックは RTOS を使用してマルチタスクで動作します。
3. 本スタックは、Modbus のフレームタイミングに複数のタイマ· チャネルを使用することはできません。

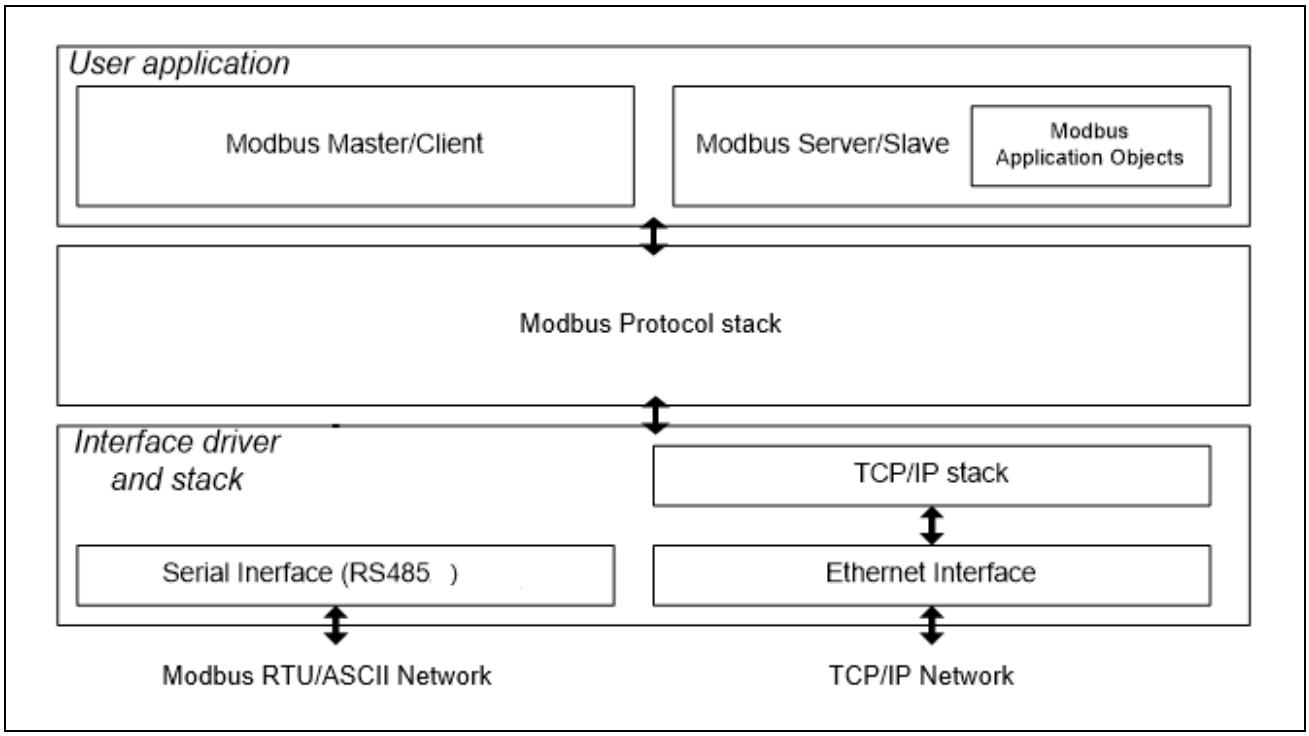

図 7.1RX72M 用 Modbus スタック概要

7.2 通信フォーマット

Modbus TCP の通信フォーマットは、Modbus RTU の CRC を除く部分を含んだ形となっています。Modbus RTU では誤りチェックのため、CRC チェックコードを末尾に付加する必要がありましたが、Modbus TCP では TCP/IP プロトコルが持つチェック機構を利用するため不要となります。

Modbus TCP の通信フォーマットでは、Modbus RTU には存在しなかった、トランザクション識別子、プロトコル識別子、メッセージ長、ユニット識別子を付加する必要があります。

Modbus RTU フォーマットと Modbus TCP/IP フォーマットの違いを図 7.2 に示します。

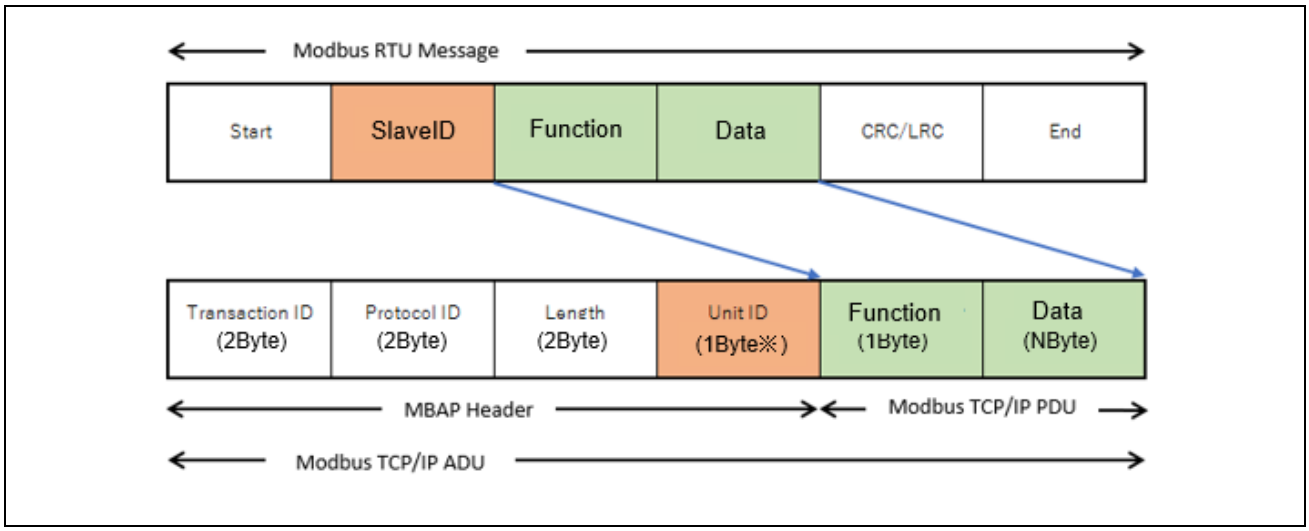

図 7.2Modbus RTU フォーマットと Modbus TCP/IP フォーマットの違い

ADU：Application Data Unit

MBAP Header：Modbus Application Header

PDU：Protocol Data Unit

8. システム構成 - Modbus TCP プロトコルスタック

Modbus TCP および Modbus TCP シリアルゲートウェイスタックの詳細について説明します。

Modbus TCP シリアルゲートウェイモードでは、シリアルネットワークへのゲートウェイとして Modbus RTU/ASCII マスタスタックを使用することになります。Modbus RTU/ASCII マスタスタックの初期化は、ゲートウェイスタックの初期化内で行われます。ユーザは、Modbus RTU または Modbus ASCII ゲートウェイスタックのいずれかを選択することができます。

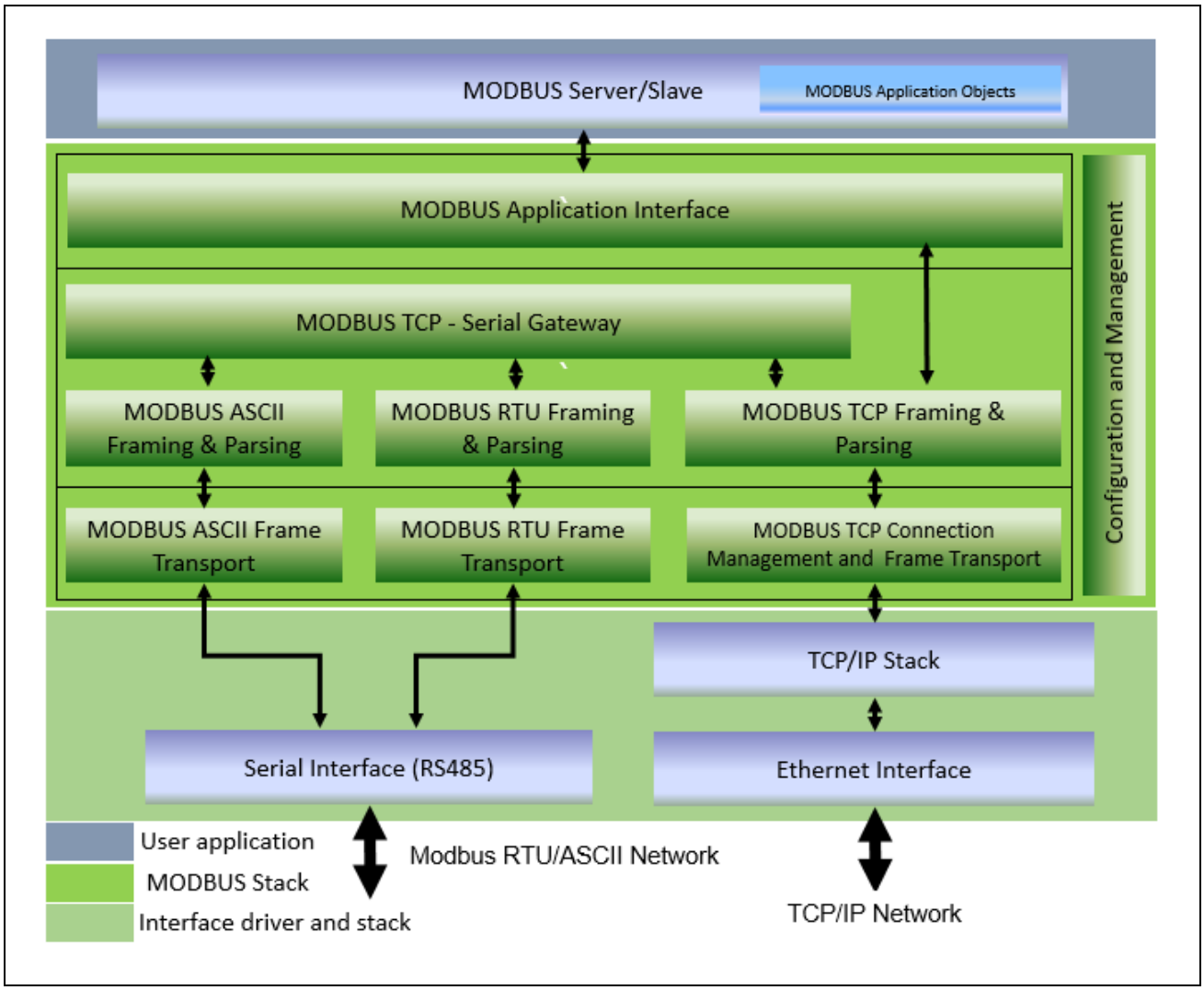

図 8.1Modbus TCP スタックソフトウェア構成

8.1 モジュール構成

TCP サーバおよびゲートウェイスタックは機能に基づいて、いくつかのレイヤに分割されます。

最上位層にアプリケーションインタフェース層は、2 つのタスクとコールバック関数を各ファンクションコードにマッピングする API 関数で構成されます。

中間層のパケット構築および解析層は、パケットの構築・解析およびメールボックスを操作するための関数およびキューで構成されています。これらの関数はすべて、上位層のタスクで実行されます。

最下層の接続管理およびパケット送受信層は、TCP 通信の接続および送受信を行う関数およびタスクで構成されます。応答 TCP メッセージを送信するための機能を除く全ての関数は、タスクで実行されます。

コンフィグレーション層は最上位層から最下層の 3 層に跨って呼び出されるレイヤで、構成用 API と共に必要な機能が含まれています。

8.1.1 アプリケーションインタフェース層

本レイヤは２つのタスクと複数の関数で構成され、指定されたモードを基にタスクと関数が動作します。スタックが TCP サーバモードの場合、ゲートウェイタスクは動作しません。ゲートウェイ機能が動作する場合のみ起動されます。しかし、ゲートウェイ機能のみの場合でも、TCP サーバタスクは動作します。

TCP サーバモード時の処理

1) サンプルプログラムのメインタスク（main_task）より、Modbus 初期化関数（modbus_init()）を呼び出します。
2) Modbus 初期化関数は、ModbusTCP 通信で必要な以下のタスクを起動します。
   - 接続待ちタスク（Modbus_tcp_soc_wait_task）
   - パケット受信タスク（Modbus_tcp_recv_data_task）
   - Modbus ファンクション処理タスク（Modbus_tcp_req_process_task）

TCP シリアルゲートウェイモード時の処理

1) サンプルプログラムのメインタスク（main_task）より、Modbus 初期化関数（modbus_init()）を呼び出します。
2) Modbus 初期化関数は、ModbusTCP 通信で必要な以下のタスクを起動します。
   - 接続待ちタスク（Modbus_tcp_soc_wait_task）
   - パケット受信タスク（Modbus_tcp_recv_data_task）
   - Modbus ファンクション処理タスク（Modbus_tcp_req_process_task）
   - シリアルゲートウェイタスク（Modbus_gateway_task）
   - シリアルタスク（Modbus_serial_task）
   - シリアルデータ受信タスク（Serial_recv_task）
   - シリアルパケット受信タスク（Modbus_serial_recv_task）

8.1.1.1 Modbus TCP サーバタスク

本タスクは、スタックが Modbus TCP サーバとして設定された場合に動作します。受信した Modbus 要求が TCP 受信データタスクに送られたとき、メールボックスからデータを取得するために待ちます。メールボックスにパケットが到着したとき、パケットをコピーし処理します。

ゲートウェイ機能の有無で動作が異なります。

ゲートウェイ機能なしの場合、スレーブ ID が 0xFF 以外のパケットを破棄し、スレーブ ID が 0xFF のパケットのみ処理します。一方、ゲートウェイ機能ありの場合、スレーブ ID が 0xFF 以外の要求を受けると、TCP - シリアルゲートウェイタスクへ要求パケットを転送します。

ゲートウェイ機能ありとゲートウェイ機能なしの場合のタスク状態遷移図をそれぞれ示します。

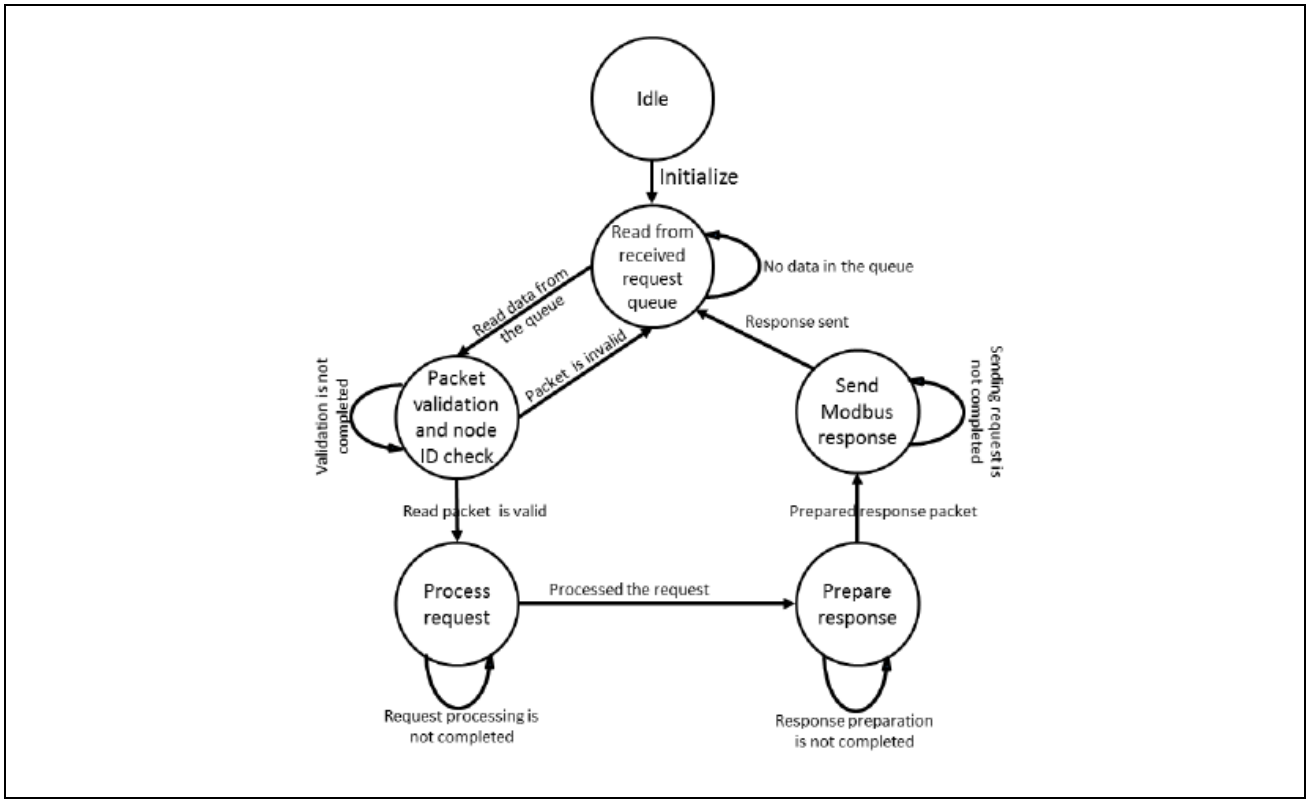

図 8.2Modbus TCP サーバタスク（ゲートウェイ機能なし）

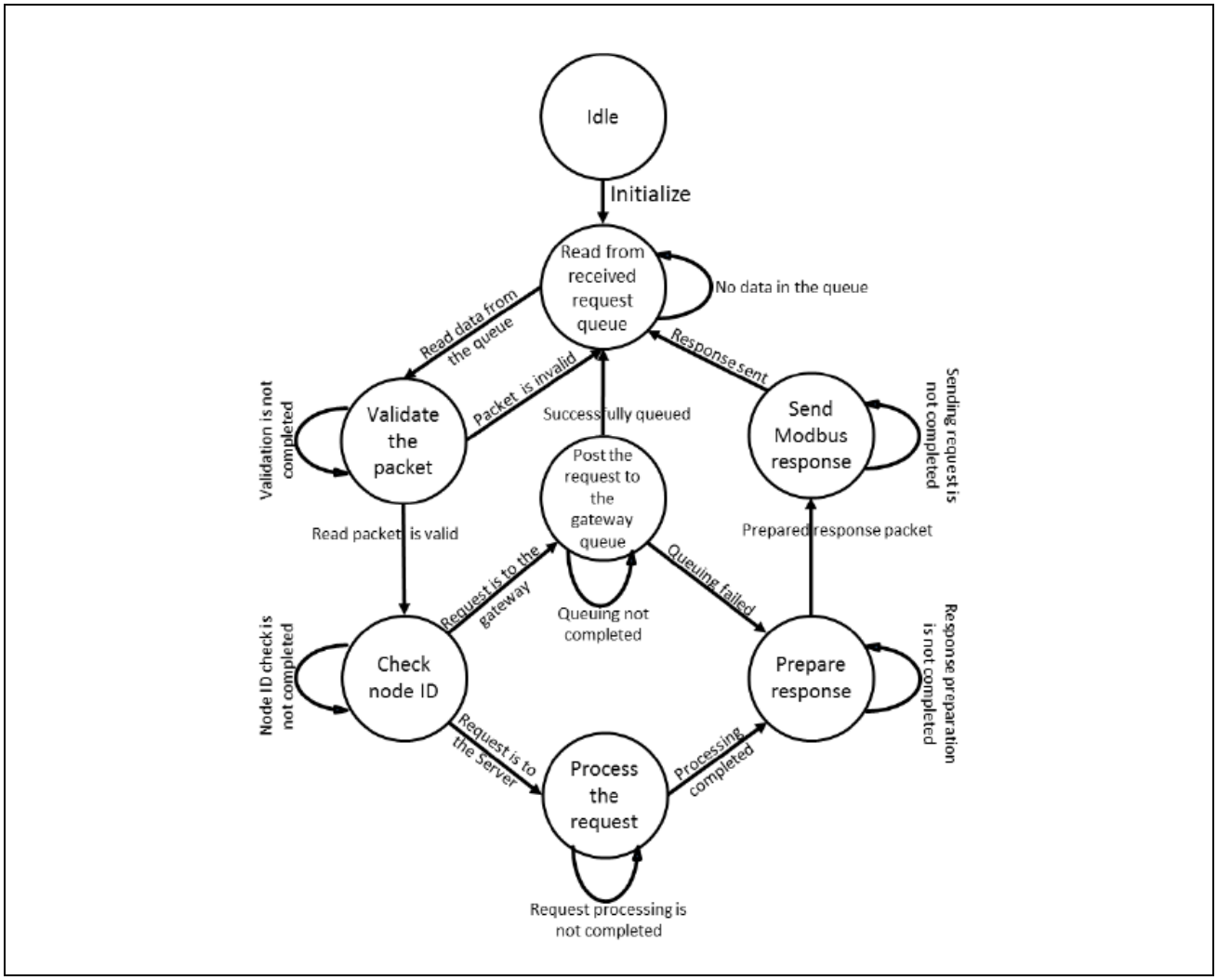

図 8.3Modbus TCP サーバタスク（ゲートウェイ機能あり）

8.1.1.2 Modbus TCP – シリアルゲートウェイタスク

本タスクは Modbus RTU/ASCII マスタスタックの機能を使用し、シリアルインターフェースによる通信に応答します。

- Modbus TCP クライアントからの要求を受ける。
- スレーブ ID が 0xFF 以外の要求を受信したとき、RTU/ASCII マスタスタックに要求を送信します。
- RTU/ASCII マスタスタックから応答を受け取り、Modbus TCP クライアントへ応答を返します。

図 8.4 にタスク状態遷移図を示します。

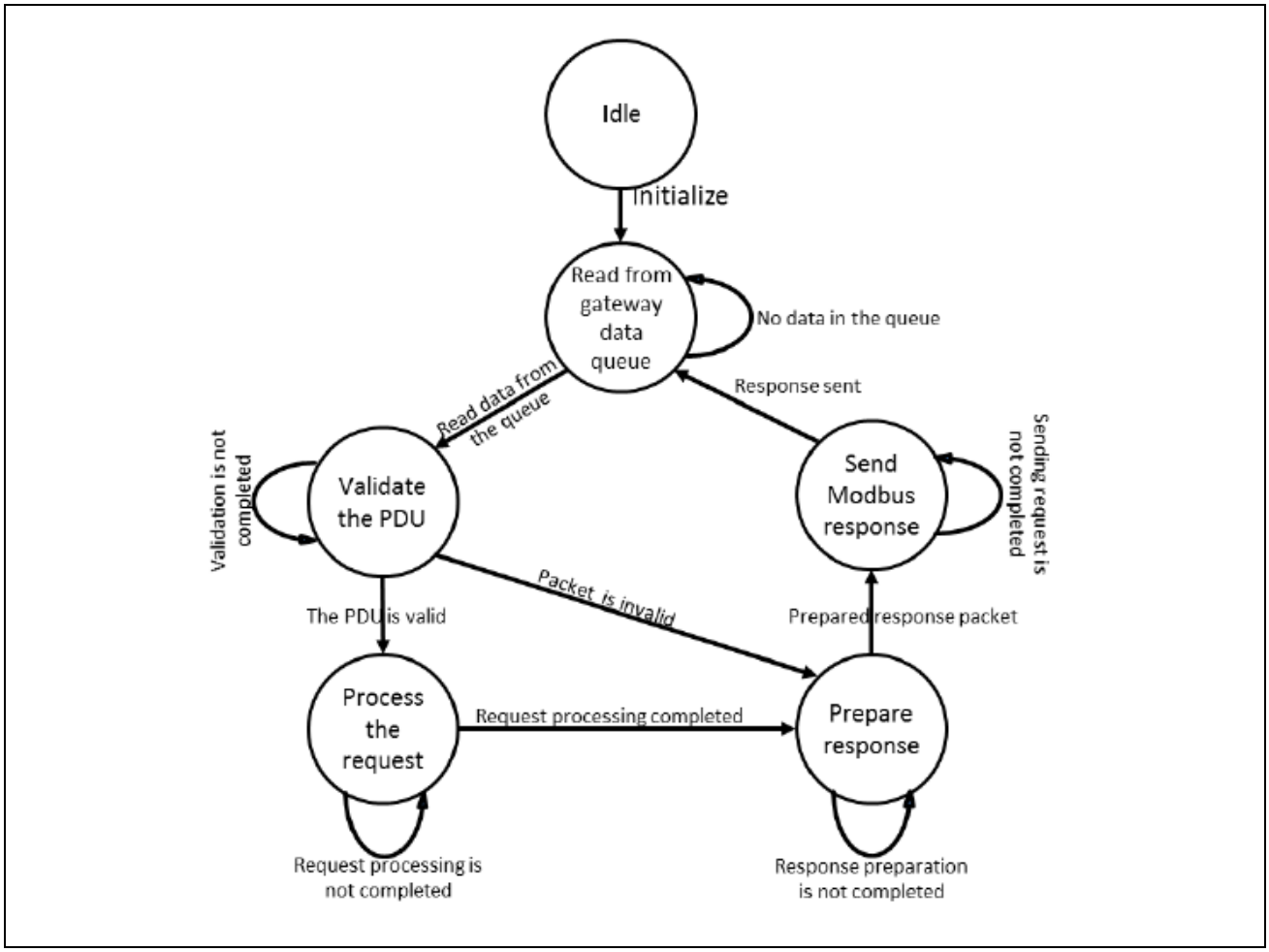

図 8.4Modbus TCP - シリアルゲートウェイタスク（Modbus_gateway_task）

8.1.1.3 エラー判定及び報告

- パケット構築用に動的メモリを確保します。メモリが確保できない場合はエラーを報告します。
- ゲートウェイタスクは、最大 MAX_GW_MBX_SIZE までメッセージをキューイングします。ゲートウェイタスクがキューイングできない場合、TCP サーバは要求パケットに対して例外コード（06）を応答パケットとして返信します。

8.1.2 パケット構築および解析層

本レイヤは、Modbus パケットの構築および解析を実施するレイヤです。Modbus パケットの解析および構築をする関数ならびにデータ構造で構成されます。

- 受信パケット解析（Modbus_tcp_parse_pkt）
- 受信パケット検証（Modbus_tcp_validate_pkt）
- 送信パケット構築（Modbus_tcp_frame_pkt）

8.1.2.1 受信パケットの解析

パケット長および指定値の整合性などを検証し異常があった場合、受信パケットは破棄されエラーを報告します。受信パケットが正常ならば、リクエストを処理するためにユーザが登録したコールバック関数を呼び出します。

8.1.2.2 送信パケットの構築

各コールバック関数での実行結果を基に、応答パケットを構築して送り返します。また、サポートされていないファンクションコードが指定されたなど Exception code を返却する必要がある場合も応答パケットを構築し送信します。

8.1.2.3 エラー判定および報告

- ファンクションコードに基づいて、受信パケットにおけるパケット長および指定されたデータ長がプロトコルに適合しているかを検証し、異常があった場合はエラーを報告します。
- 受信パケット解析では動的にメモリを確保します。メモリが確保できない場合はエラーを報告します。

8.1.3 通信接続管理およびパケット送受信層

本レイヤはクライアントからの接続受け付けやパケットの送受信を行うタスクおよび機能で構成されます。

8.1.3.1 Modbus TCP 接続受け入れタスク

本タスクはユーザによりスタックが初期化されたとき、初期化され、ポート 502 またはユーザが設定したポート（スタック初期化時にユーザにより指定された場合のみ）に対するクライアントからの接続要求待ちを開始します。

タスクが接続要求を受信すると、IP リストと照らし合わせて、接続済みか受け入れ可能かをチェックします。接続受け入れ後、接続済みリストに IP を登録します。

接続の最大数は、r_modbus_rx フォルダ内の”modbusTcpConfig.h”内にて定義されている「MAXIMUM_NUMBER_OF_CLIENTS」マクロ の設定値により制限されます。

図 8.5 に本タスク状態遷移図を示します。

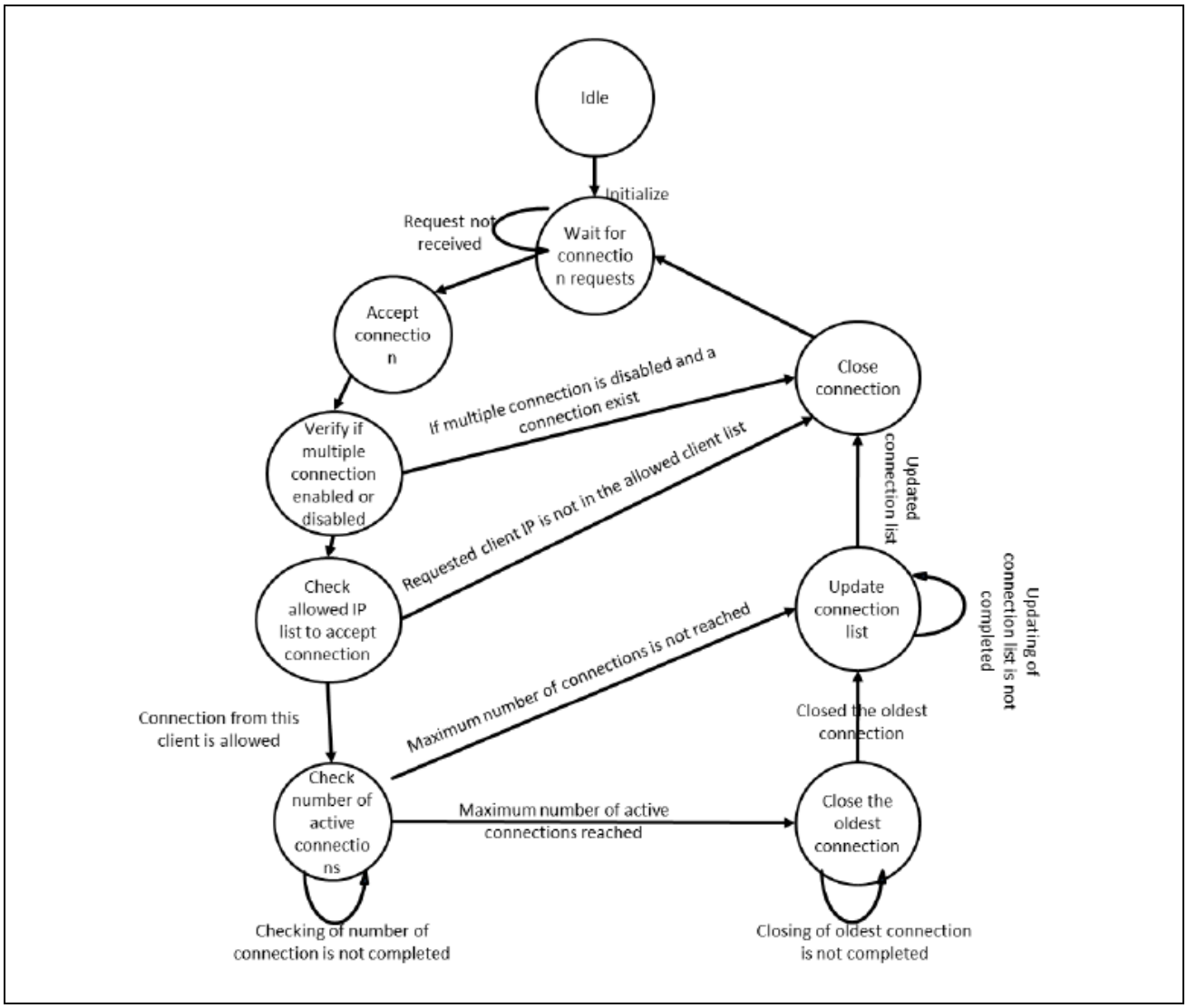

図 8.5Modbus TCP 接続受け入れタスク（Modbus_tcp_soc_wait_task）

8.1.3.2 Modbus TCP 受信データタスク

本タスクはスタックが初期化されたときに初期化が行われます。タスクは接続したクライアントからのデータを待ち、有効なパケットを受信すると、メールボックスに転送します。

クライアントから要求を受信すると、Modbus_post_to_mailbox()を呼び出して要求をメールボックスに送ります。メールメッセージは、Modbus_fetch_from_mailbox()を使って、TCP サーバタスクにより読み取られます。

図 8.6 に本タスク状態遷移図を示します。

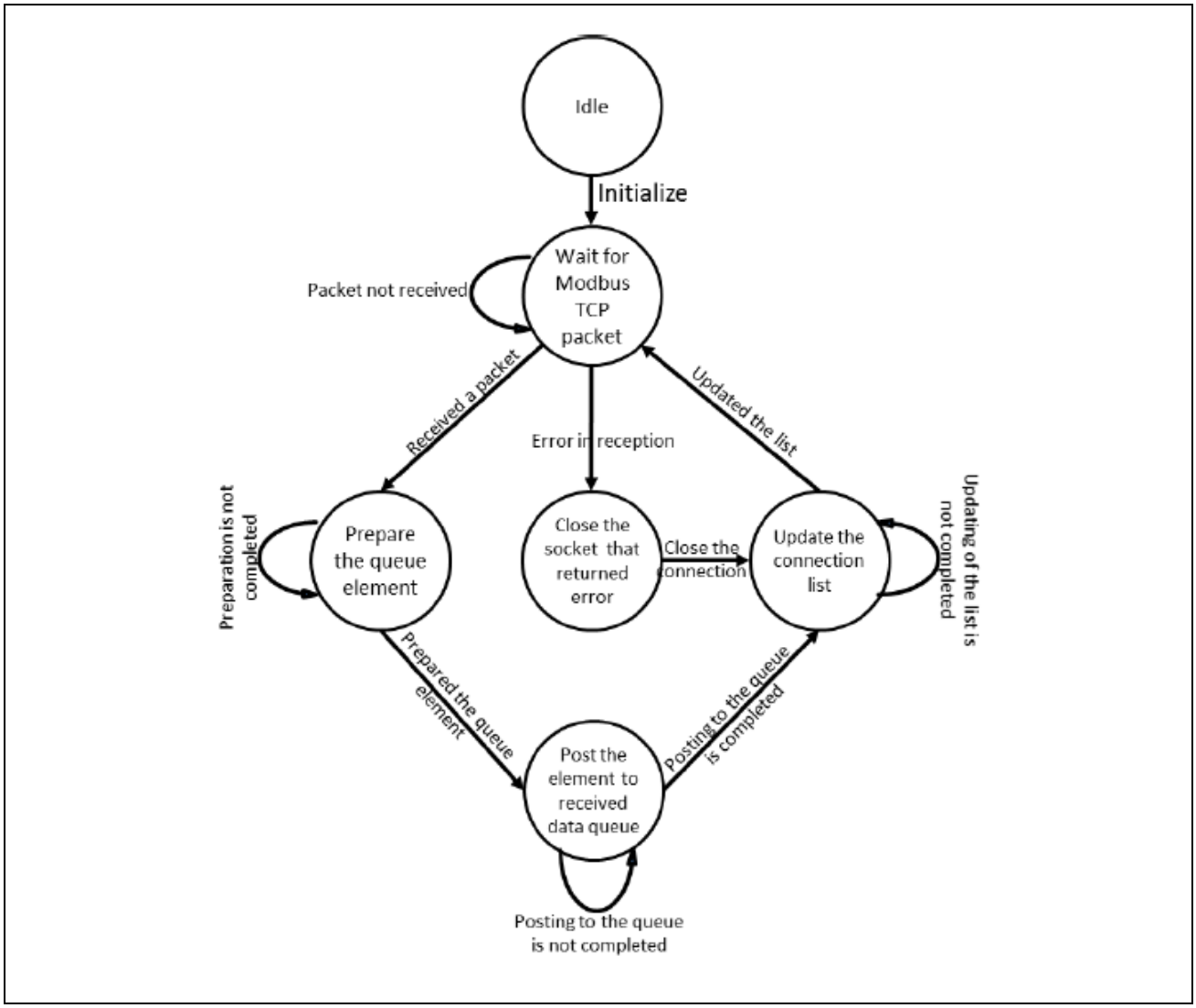

図 8.6Modbus TCP 受信データタスク（Modbus_tcp_recv_data_task）

8.1.3.3 エラー判定及び報告

- パケット解析用に動的メモリを確保します。メモリが確保できない場合はエラーを報告します。
- TCP サーバタスクは最大 MAX_RCV_MBX_SIZE までメッセージをキューイングします。TCP サーバタスクがキューイングできない場合、TCP 受信データタスクは要求パケットに対して例外コード（06）を応答パケットとして返信します。

9. システム構成 - Modbus RTU/ASCII プロトコルスタック

Modbus RTU/ASCII スタックの詳細について説明します。

本スタックは必要なコンフィグレーションを設定することにより、マスタまたはサーバ/スレーブアプリケーションとして使用できます。スタックは、Modbus RTU または Modbus ASCII のいずれか一つをサポートするよう構成できます。モード選択は初期設定 API をユーザアプリケーション上で呼び出す際に行います。

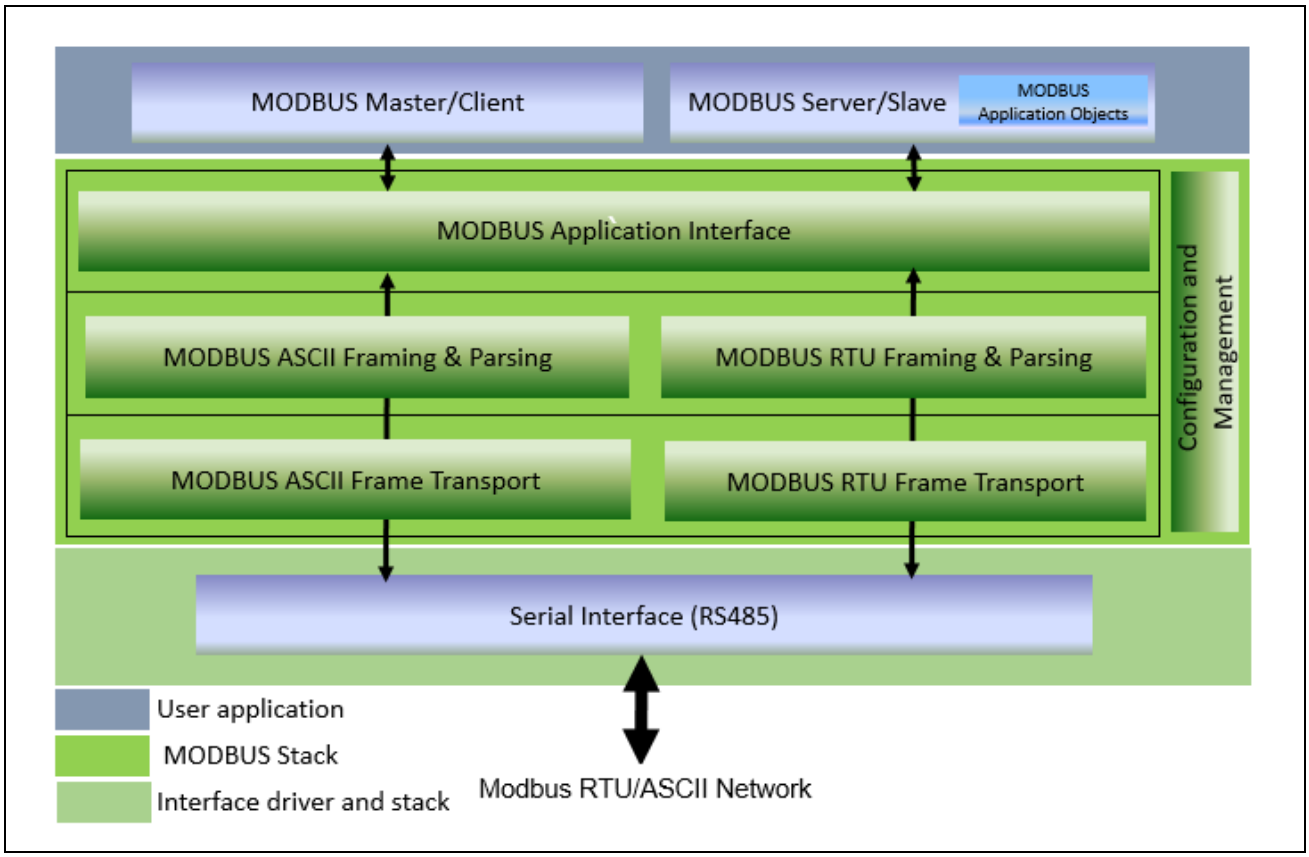

図 9.1Modbus RTU/ASCII スタックソフトウェア構成

以降のセクションで、各機能層について説明します。

9.1 モジュール構成

スタックは機能に基づいて、以下のレイヤに分割されます。
最上位層にあたるアプリケーションインタフェース層は、マスタおよびスレーブの両モードで、ユーザアプリケーションと直接やりとりをします。
中間層であるパケット構築および解析層は、Modbus フレームの構築、解析、検証を担当しています。
最下層の通信接続管理およびフレーム送受信層は、通信の論理接続との Modbus フレームの送受信を管理します。
最上位層から最下層の 3 つの層に跨る形で、構成用 API などを含むコンフィグレーション層があります。以下に各レイヤの詳細を示します。

9.1.1 アプリケーションインタフェース層

アプリケーション· インタフェース層は、ユーザアプリケーションとのやり取りに必要な機能を含んでいます。また、スタックの状態を維持するスレッドが含まれています。構成されたマスタまたはスレーブスタックモードに基づいて、スレッドはスタックモードで動作可能な機能をユーザに提供します。本レイヤは、通信モードが RTU または ASCII でも機能は変わりません。
Modbus サーバ/スレーブモードにおけるアプリケーションインタフェース層の主な構成要素は、シリアルタスク関数と Modbus_slave_map_init()になります。Modbus_slave_map_init()を使用して、ユーザアプリケーションは、有効な Modbus 要求を受信したときに特定のファンクションコードに対応したコールバック関数が呼び出されるよう登録します。
要求メッセージの解析と応答メッセージの構築はシリアルタスク関数で行われます。シリアルタスク関数は、有効な Modbus 要求を受信したとき、適切なコールバックハンドラを呼び出します。また、コールバックハンドラからの応答メッセージを要求元のマスタに対して送り返します。

> **備考　ユーザ指定のコールバックハンドラを使用する場合、実行時間に注意してください。ハンドラ内でエラー等が発生して処理が遅延すると、その間スタックは次のコマンドを処理することができません。**

Modbus マスタモードにおけるアプリケーションインタフェース層の主な構成要素は、シリアルタスク関数とユーザアプリケーションとのインタフェースとなる各種 API になります。スタックがユーザにより初期化されるとシリアルタスク関数が起動され、ユーザアプリケーションから各種 API が呼び出されることで Modbus 通信が行われます。
ユーザアプリケーションは、各ファンクションコードに対応したユーザアプリケーションインターフェース API を呼び出して、スタックから Modbus リクエストを Modbus スレーブデバイスに送ります。シリアルタスク関数はリクエストの送受信を行います。
ユーザアプリケーションインターフェース API は、ブロッキングモードまたはノンブロッキングモードで呼び出しができます。ユーザによりコールバック関数が API に引数として提供されていれば、ノンブロッキングモードで呼び出され、シリアルタスク関数は応答受信時に提供されたコールバック関数を呼び出すかタイムアウトを発生させます。コールバック関数が提供されない場合、これらの API はスレーブからの応答を受信するかタイムアウト発生までブロックされます。

9.1.1.1 Modbus シリアルタスク関数

Modbus シリアルタスク関数は、モード（ RTU/ASCII）にかかわらず、Modbus マスタおよびスレーブスタックの両方で使用されます。

例えば、スタックモードが Modbus RTU マスタとして定義されると、Modbus シリアルタスク関数は Modbus マスタの関数として機能します。マスタおよびスレーブの切り替えは初期化 API でモードが指定された際に決定されます。

図 9.2 は Modbus シリアルタスク関数がスレーブモード時の状態遷移図になります。**main_task**(**void**)関数が Modbus シリアルタスク関数を呼び出し、メッセージが送信されるのをチェックします。受信したメッセージの種類に応じて、マスタまたはスレーブとして機能します。スレーブとして機能するとき、Modbus マスタからの Modbus 要求を受信するまで、状態を保持します。

クライアントから要求を受信すると、関数は以下の動作をします。

- 受信した要求パケットの解析および検証を行います。
- 受信パケットが正常ならば、応答パケットを構築し、マスタデバイスへ送信します。

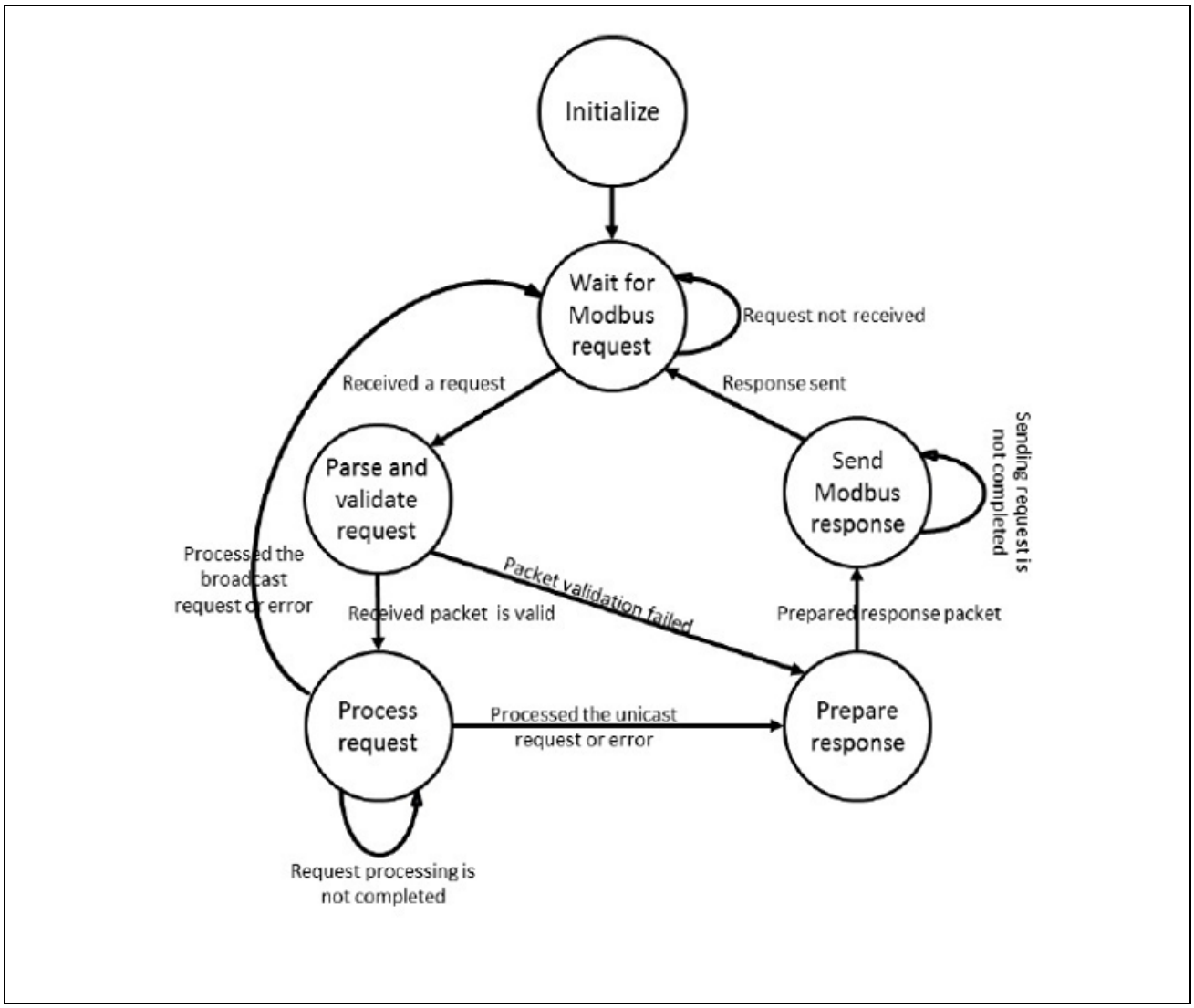

図 9.2スレーブモード時の Modbus シリアルタスク関数の状態遷移図

図 9.3は、シリアルタスク関数がマスタモードの時の状態遷移図になります。

マスタモードの場合もスレーブモードと同様、Modbus_serial_stack_init ()でスタックが初期化されます。ユーザアプリケーションが、API を呼び出して、シリアルタスク関数にスレーブデバイスとの Modbus 通信を要求します。要求を受信したら、以下を実施します。

- Modbus 要求パケットを準備しスレーブデバイスへ送信します。
- 送信リクエストがブロードキャスト要求の場合、「ターンアラウンド遅延時間」までスレーブデバイスからの応答が返るのを待ちます。
- 送信リクエストがユニキャスト要求の場合、「応答タイムアウト時間」まで、スレーブデバイスからの応答を待ちます。
- 応答タイムアウトまでにスレーブデバイスから有効な応答を受信した場合、受信されたデータから応答テーブルを更新しユーザアプリケーションに引き渡します。
- 応答タイムアウト時間内に応答を受信できなかった場合、最大リトライ回数まで同じ要求を送信します。
- リトライに対する応答を受信できなかった場合、コールバックハンドラに対してタイムアウトエラーを引き渡します。

コマンド要求の処理が完了したときに通知を必要とする場合、ユーザアプリケーションは、API 関数呼び出しと一緒にコールバックハンドラを提供することができます。また、API 関数コールはノンブロッキングモードで動作し、リクエストが完了するまでの間、他関数を動作させることができます。コールバック関数が提供されない場合、API 関数コールはブロッキングモードで動作します。

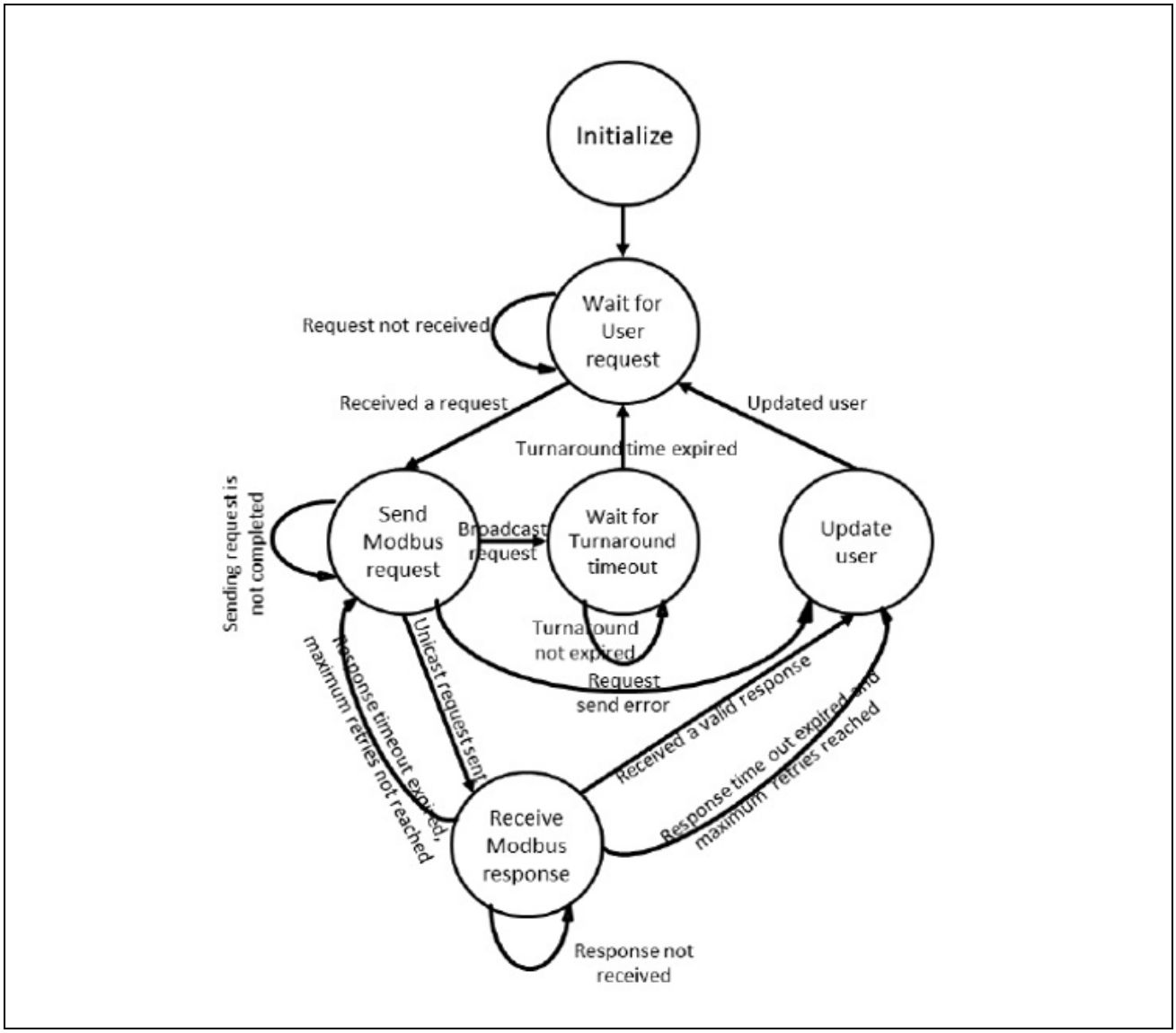

図 9.3マスタモード時の Modbus シリアルタスク関数の状態遷移図

9.1.1.2 エラー判定および報告

(1) Modbus RTU/ASCII スレーブ

- ユニキャスト要求の場合、ユーザによりコールバック関数が登録されていないファンクションコードに対するリクエストを受信したとき、シリアルタスク関数は例外応答を構築しマスタデバイスに返します。
- ユーザによって記述されたコールバック関数は、実装されていないレジスタまたはコイルへの要求に対して、例外応答を生成する必要があります。
- 初期化 API では、指定されたパラメータの基本的なチェックを行い、状態を返します。
- ブロードキャスト要求の場合、クライアントに応答を返しません。

(2) Modbus RTU/ASCII マスタ

- 初期化 API では、指定されたパラメータの基本的なチェックを行います。
- 規定回数分のリトライを実施しても、リクエストに対する応答を受信できない場合、タイムアウトエラーを返します。

9.1.2 パケット構築および解析層

本レイヤは、Modbus パケットの構築および解析を実施するレイヤです。Modbus パケットの解析および構築をする関数ならびにデータ構造で構成されます。これらの機能は、設定されたスタックモードに従い処理を行いますが、Modbus パケットは内部的に RTU 形式で処理されるので、ASCII モードの場合も一度 RTU 形式に変換して処理されます。

- 受信パケット解析（Modbus_master_parse_pkt , Modbus_slave_parse_pkt）
- 受信パケット検証（Modbus_master_validate_pkt , Modbus_slave_validate_pkt）
- 送信パケット構築（Modbus_master_frame_request , Modbus_slave_frame_response）

9.1.2.1 受信パケットの解析

- ASCII モードの場合、ASCII→RTU 形式へのパケット変換を行います。
- 受信パケットの検証で、長さチェック、パケットの整合性およびスレーブ ID の不一致などが発生した場合、受信パケットを破棄します。
- 受信パケットが正常ならば、リクエストを処理するために、ユーザが登録したコールバック関数を呼び出します。
- ユニキャスト要求を受信した場合、ファンクションコードの処理に異常があれば、例外応答メッセージを作成しマスタデバイスに送ります。
- ブロードキャスト要求を受信した場合、受信したパケットがライト系ファンクションコードならば正常パケットとして受け付けますが応答メッセージをマスタデバイスに送信しません。また、受信したパケットがリード系ファンクションコードならば、スレーブ ID 異常として要求パケットを破棄します。

9.1.2.2 送信パケットの構築

マスタモードの場合は要求パケットが、スレーブモードの場合は応答パケットが、API で生成された内容に基づいて構築されます。その後、構築したパケットに CRC/LRC を付加します。ASCII モードの場合、内部的に RTU 形式で処理されていますので、RTU→ASCII 変換が行われます。

9.1.2.3 エラー判定および報告

- ファンクションコードに基づいて、受信および送信パケットにおけるパケットの長さ、指定されたデータおよびスレーブ ID について、プロトコルに適合しているかを検証します。
- 受信したパケットに埋め込まれた CRC/ LRC を使用して、パケットの整合性を検証します。
- パケット解析用に動的メモリを確保します。メモリが確保できない場合はエラーを報告します。

9.1.3 通信接続管理およびフレーム送受信層

本レイヤは、通信インタフェースを介してデータを送受信するための機能およびデータ構造を含みます。シリアル通信のフレームタイミングの管理は、本レイヤで行われます。

9.1.3.1 シリアル受信タスク関数

Modbus RTU/ASCII スタックの初期化処理でシリアル I/F およびタイマ割り込みが登録されます。シリアル受信タスク関数は、シリアル I/F およびタイマ割り込みが発生すると呼び出されます。

シリアル受信タスク関数は、シリアル I/F 割り込みが発生すると、シリアル I/F ドライバ関数にて受信データを読み込みます。データの受信に成功したのち、スタックモードに応じて、Modbus_ascii_recv_char() または Modbus_rtu_recv_char() が呼び出されます。それらの関数で、データがバッファに格納されます。

タイマ割り込みが発生すると、Modbus_timer_handler()が呼び出されます。本関数でフレームタイミングの判定が行われます。

9.1.3.2 Modbus シリアルインターフェースコンフィグレーション

- スタックの初期化中に設けられた構成パラメータに従って、シリアルインターフェースを使用しパケットを送受信します。
- 受信動作中にエラーが発生した場合、ステータス割り込みイベントを発生させています。受信エラーの詳細については、「RX72M CPU カード ハードウェアマニュアル」を参照してください。
- タイマは無通信時間の測定に利用されます。
- RS485 のモード切り替えは、GPIO ピンを使って行われます。

9.1.3.3 エラー識別および報告

受信中にエラーが発生した場合、受信したパケットを破棄し、処理を続行します。

10. アプリケーションプログラミングインタフェースの説明

本章では、アプリケーションプログラミングインターフェース(API)の仕様の詳細について説明します。

10.1 ユーザインターフェース API

本章では、ユーザアプリケーションで使用する API について説明します。

10.1.1 Modbus TCP/IP

10.1.1.1 プロトコルスタックの初期化

プロトコルスタックの初期化で使用する API は以下になります。

Modbus_tcp_init_stack	Modbus TCP スタック初期化 API

【書式】

```
uint32_t Modbus_tcp_init_stack(uint8_t u8_stack_mode,
                               uint8_t u8_tcp_gw_slave,
                               uint8_t u8_tcp_multiple_client,
                               uint32_t u32_additional_port,
                               p_serial_stack_init_info_t pt_serial_stack_init_info,
                               p_serial_gpio_cfg_t pt_serial_gpio_cfg_t);
```

【引数】

uint8_t	u8_stack_mode	スタックモード指定
uint8_t	u8_tcp_gw_slave	ゲートウェイモード指定
uint8_t	u8_tcp_multiple_client	マルチクライアント対応指定
uint32_t	u32_additional_port	追加ポート番号
p_serial_stack_init_info_t	pt_serial_stack_init_info	シリアル通信パラメータテーブルへのポインタ
p_serial_gpio_cfg_t	pt_serial_gpio_cfg_t	I/O ポートコンフィグレーションテーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_INIT	初期化失敗

【解説】

本 API はユーザが指定した情報に従って、Modbus スタックを初期化します。引数 p_serial_stack_init_info_t の指定が NULL の場合、Modbus_tcp_server_init_stack()を呼び出します。NULL 以外が指定された場合、Modbus_tcp_init_gateway_stack()を指定されたシリアル通信パラメータで呼び出します。

本 API では、以下のパラメータを指定します。

a. u8_stack_mode はスタックの種別を指定します。指定された値により、スタックは以下のいずれかのモードで動作します。ゲートウェイモードで使用する場合、シリアルデバイスとの通信で使用するモードを指定してください。

スタックモード指定	意味
MODBUS_RTU_MASTER_MODE	Modbus Stack RTU マスタモードを選択
MODBUS_RTU_SLAVE_MODE	Modbus Stack RTU スレーブモードを選択
MODBUS_ASCII_MASTER_MODE	Modbus Stack ASCII マスタモードを選択
MODBUS_ASCII_SLAVE_MODE	Modbus Stack ASCII スレーブモードを選択
MODBUS_TCP_SERVER_MODE	Modbus Stack TCP サーバモードを選択

b. u8_tcp_gw_slave は、ゲートウェイモードでの動作を指定します。指定には以下のマクロを使用します。

ゲートウェイモード指定	意味
MODBUS_TCP_GW_SLAVE_DISABLE	Modbus スタックゲートウェイスレーブ無効
MODBUS_TCP_GW_SLAVE_ENABLE	Modbus スタックゲートウェイスレーブ有効

c. u8_tcp_multiple_client は、複数クライアントからの通信を受け付けるかどうかを指定します。指定には以下のマクロを使用します。

マルチクライアント対応指定	意味
DISABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続無効
ENABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続有効

d. u32_additional_port は、通信ポートにデフォルト（502）以外を使用する場合に指定します。追加しない場合は 0 を指定してください。

e. p_serial_stack_init_info_t 構造体はシリアル通信に関連する設定を指定します。TCP サーバモードで使用する場合は NULL を指定してください。

・シリアル通信パラメータテーブル(serial_stack_init_info_t)

```
typedef struct _stack_init_info{
    uint32_t    u32_baud_rate;              /* シリアルポートのボーレート指定 */
    uint8_t     u8_parity;                  /* シリアルポートのパリティビット指定 */
    uint8_t     u8_stop_bit;                /* シリアルポートのストップビット指定 */
    uint8_t     u8_uart_channel;            /* Modbus スタックで使用する UART チャネル */
    uint8_t     u8_timer_channel;           /* Modbus スタックで使用するタイマチャネル */
    uint32_t    u32_response_timeout_ms;    /* 応答タイムアウト[msec] */
    uint32_t    u32_turnaround_delay_ms;    /* ブロードキャストでリクエストを送信する場合の遅延時間
                                               [msec] */
    uint32_t    u32_interframe_timeout_us;  /* RTU パケットのインターバル時間[usec] */
    uint32_t    u32_interchar_timeout_us;   /* ASCII パケットのインターバル時間[usec] */
    uint8_t     u8_retry_count;             /* リトライ回数 */
}serial_stack_init_info_t, *p_serial_stack_init_info_t;
```

この構造体には以下のマクロがパラメータとして使用されます。

ボーレート指定	意味
MODBUS_BAUDRATE	9600bps , 19200bps , 115200bps より任意のボーレートを指定

パリティ指定	意味
UART_PARITY_NONE	パリティなし
UART_PARITY_ODD	奇数パリティ
UART_PARITY_EVEN	偶数パリティ

ストップビット指定	意味
UART_STOPBIT_1	ストップビット１ビット
UART_STOPBIT_2	ストップビット２ビット

f. p_serial_gpio_cfg_t 構造体は RS485 通信に使用する GPIO ポート制御用関数へのポインタを指定します。TCP サーバモードで使用する場合は、NULL を指定してください。

・I/O ポートコンフィグレーションテーブル(serial_gpio_cfg_t)

```
typedef struct _serial_gpio_cfg_t{
    fp_gpio_callback_t    fp_gpio_init_ptr;    /* RS485 送受信方向制御用初期化関数へのポインタ */
    fp_gpio_callback_t    fp_gpio_set_ptr;     /* RS485 送受信方向制御用送信設定関数へのポインタ */
    fp_gpio_callback_t    fp_gpio_reset_ptr;   /* RS485 送受信方向制御用受信設定関数へのポインタ */
}serial_gpio_cfg_t, *p_serial_gpio_cfg_t;
```

Modbus_slave_map_init	Modbus ファンクションコードマッピング API

【書式】

uint32_t Modbus_slave_map_init(p_slave_map_init_t pt_slave_func_tbl);

【引数】

p_slave_map_init_t	pt_slave_func_tbl	ファンクションコードマッピングテーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_INVALID_STACK_INIT_PARAMS	引数が NULL
ERR_MEM_ALLOC	メモリ確保に失敗

【解説】

本 API は、クライアントからの各ファンクションコードに対する処理要求をユーザ定義の関数に関連付けます。Modbus スレーブスタックが要求を受信した際、登録された関数を呼び出します。

本 API はスレーブモードの場合のみ有効となります。

・ファンクションコードマッピングテーブル (slave_map_init_t)

```
typedef struct _slave_map_init{
    fp_function_code1_t     fp_function_code1;     /* ファンクションコード 1(Read coils)用コールバック関数へのポインタ */
    fp_function_code2_t     fp_function_code2;     /* ファンクションコード 2(Read discrete inputs)用コールバック関数へのポインタ */
    fp_function_code3_t     fp_function_code3;     /* ファンクションコード 3(Read holding registers)用コールバック関数へのポインタ */
    fp_function_code4_t     fp_function_code4;     /* ファンクションコード 4(Read input registers)用コールバック関数へのポインタ */
    fp_function_code5_t     fp_function_code5;     /* ファンクションコード 5(Write single coil)用コールバック関数へのポインタ */
    fp_function_code6_t     fp_function_code6;     /* ファンクションコード 6(Write single register)用コールバック関数へのポインタ */
    fp_function_code15_t    fp_function_code15;    /* ファンクションコード 15(Write multiple coils)用コールバック関数へのポインタ */
    fp_function_code16_t    fp_function_code16;    /* ファンクションコード 16(Write multiple registers)用コールバック関数へのポインタ */
    fp_function_code23_t    fp_function_code23;    /* ファンクションコード 23(Read/Write multiple registers)用コールバック関数へのポインタ */
}slave_map_init_t, *p_slave_map_init_t;
```

各ファンクションコードに対応したコールバック関数は、以下の書式で定義されます。各コールバック関数の引数に使用されている構造体の詳細については 10.1.2.2 章の各 API を参照してください。

fp_function_code1_t	Modbus ファンクションコード 1(Read coils)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code1_t)(p_req_read_coils_t pt_req_read_coils,
                                p_resp_read_coils_t pt_resp_read_coils );
```

【引数】

p_req_read_coils_t	pt_req_read_coils	Read coil 要求情報が格納された構造体へのポインタ
p_resp_read_coils_t	pt_resp_read_coils	Read coil 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常,1 : 異常

fp_function_code2_t	Modbus ファンクションコード 2(Read discrete inputs)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code2_t)(p_req_read_inputs_t pt_req_read_inputs,
                                p_resp_read_inputs_t pt_resp_read_inputs );
```

【引数】

p_req_read_inputs_t	pt_req_read_inputs	Read discrete inputs 要求情報が格納された構造体へのポインタ
p_resp_read_inputs_t	pt_resp_read_inputs	Read discrete inputs 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code3_t	Modbus ファンクションコード 3(Read holding register)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code3_t)(p_req_read_holding_reg_t pt_req_read_holding_reg,
                                p_resp_read_holding_reg_t pt_resp_read_holding_reg);
```

【引数】

p_req_read_holding_reg_t	pt_req_read_holding_reg	Read holding register 要求情報が格納された構造体へのポインタ
p_resp_read_holding_reg_t	pt_resp_read_holding_reg	Read holding register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code4_t	Modbus ファンクションコード 4(Read input register)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code4_t)(p_req_read_input_reg_t pt_req_read_input_reg,
                                p_resp_read_input_reg_t pt_resp_read_input_reg);
```

【引数】

p_req_read_input_reg_t	pt_req_read_input_reg	Read Input register 要求情報が格納された構造体へのポインタ
p_resp_read_input_reg_t	pt_resp_read_input_reg	Read holding register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code5_t	Modbus ファンクションコード 5(Write single coil)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code5_t)(p_req_write_single_coil_t pt_req_write_single_coil,
                                p_resp_write_single_coil_t pt_resp_write_single_coil );
```

【引数】

p_req_write_single_coil_t	pt_req_write_single_coil	Write single coil 要求情報が格納された構造体へのポインタ
p_resp_write_single_coil_t	pt_resp_write_single_coil	Write single coil 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code6_t	Modbus ファンクションコード 6(Write single register)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code6_t)(p_req_write_single_reg_t pt_req_write_single_reg,
                                p_resp_write_single_reg_t pt_resp_write_single_reg);
```

【引数】

p_req_write_single_reg_t	pt_req_write_single_reg	Write single register 要求情報が格納された構造体へのポインタ
p_resp_write_single_reg_t	pt_resp_write_single_reg	Write single register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code15_t	Modbus ファンクションコード 15(Write multiple coils)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code15_t) (p_req_write_multiple_coils_t pt_req_write_multiple_coils,
                                  p_resp_write_multiple_coils_t pt_resp_write_multiple_coils);
```

【引数】

p_req_write_multiple_coils_t	pt_req_write_multiple_coils	Write multiple coils 要求情報が格納された構造体へのポインタ
p_resp_write_multiple_coils_t	pt_resp_write_multiple_coils	Write multiple coils 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code16_t	Modbus ファンクションコード 16(Write multiple registers)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code16_t) (p_req_write_multiple_reg_t pt_req_write_multiple_reg,
                                  p_resp_write_multiple_reg_t pt_resp_write_multiple_reg);
```

【引数】

p_req_write_multiple_reg_t	pt_req_write_multiple_reg	Write multiple registers 要求情報が格納された構造体へのポインタ
p_resp_write_multiple_reg_t	pt_resp_write_multiple_reg	Write multiple registers 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

fp_function_code23_t	Modbus ファンクションコード 23(Read/Write multiple registers)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code23_t) (p_req_read_write_multiple_reg_t pt_req_read_write_multiple_reg,
                                  p_resp_read_write_multiple_reg_t pt_resp_read_write_multiple_reg);
```

【引数】

p_req_read_write_multiple_reg_t	pt_req_read_write_multiple_reg	Read/Write multiple registers 要求情報が格納された構造体へのポインタ
p_resp_read_write_multiple_reg_t	pt_resp_read_write_multiple_reg	Read/Write multiple registers 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

10.1.1.2 IP アドレス管理

プロトコルスタックの初期化で使用する API は以下になります。

Modbus_tcp_init_ip_table	ホスト IP リストの状態設定

【書式】

```
void_t Modbus_tcp_init_ip_table(ENABLE_FLAG e_flag,
                                TABLE_MODE e_mode);
```

【引数】

ENABLE_FLAG	e_flag	ホスト IP リストの有効/無効指定 有効: ENABLE, 無効: DISABLE
TABLE_MODE	e_mode	リストに含まれている IP アドレスのアクセス権指定 アクセス許可: ACCEPT, アクセス拒否: REJECT

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、ホスト IP リストの有効/無効およびホスト IP リストに登録された IP アドレスでのアクセス権を指定します。

ホスト IP リスト無効がデフォルト動作となります。

Modbus_tcp_add_ip_addr	ホスト IP リストへの追加

【書式】

```
uint32_t Modbus_tcp_add_ip_addr(pchar_t pu8_add_ip);
```

【引数】

pchar_t	pu8_add_ip	ホスト IP アドレス(IPv4 表記) 例. 192.168.1.100

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_IP_ALREADY_PRESENT	指定アドレスが登録済み
ERR_MAX_CLIENT	登録数が最大値に達している
ERR_TABLE_DISABLED	ホスト IP リストが無効状態

【解説】

本関数は、ホスト IP リストに指定した IP アドレスを追加します。

Modbus_tcp_delete_ip_addr	ホスト IP リストからの削除

【書式】

uint32_t Modbus_tcp_delete_ip_addr(pchar_t pu8_del_ip);

【引数】

pchar_t	pu8_del_ip	ホスト IP アドレス(IPv4 表記)

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_IP_NOT_FOUND	指定した IP アドレスが見つからない
ERR_TABLE_EMPTY	リストが空
ERR_TABLE_DISABLED	リストが無効状態。すなわち、サーバはいずれのホストからの要求も受け付ける。

【解説】

本関数は、指定した IP アドレスをホスト IP リストから削除します。

10.1.1.3 タスク

プロトコルスタックで動作するタスクのメイン処理になります。

Modbus_tcp_recv_data_task	TCP 受信データタスク

【書式】

void_t Modbus_tcp_recv_data_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

本タスクは、接続済みのソケット ID に対する要求を受信するのを待ちます。受信したパケットが Modbus プロトコルのパケットかどうかを検証し、そうならば、受信処理用キュー(メールボックス)に要求があったことを書き込みます。キューがフル状態ならば、クライアント側にサーバビジー状態を送信します。

Modbus_tcp_req_process_task	TCP サーバタスク

【書式】

void_t Modbus_tcp_req_process_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

このタスクは受信処理用キュー(メールボックス)に要求が入るのを待ちます。TCP サーバまたはシリアルで接続されたデバイスに対するパケットであることを判断するために、要求パケットのスレーブ ID を検証します。要求パケットが TCP サーバに対する処理ならば、応答パケットを作成し、TCP クライアントに送信します。要求パケットがシリアルデバイスに対するものならば、ゲートウェイ処理用キューに要求を転送します。

Modbus_gateway_task	TCP-シリアルゲートウェイタスク

【書式】

void_t Modbus_gateway_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

このタスクは、TCP サーバに接続されたシリアルデバイスへのデータを処理するためにゲートウェイ処理用キューに対する要求を待ちます。

キューから要求を読み出して処理したのち、応答パケットを作成し、TCP クライアントに送信します。

Modbus_tcp_soc_wait_task	TCP 接続受け入れタスク

【書式】

void_t Modbus_tcp_soc_wait_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

このタスクは、Modbus デフォルトのポート 502 とユーザに追加されたポート番号に対するクライアントからの接続を待ちます。ホスト IP リストが有効な場合、指定されているアクセス権に応じて指定された IP アドレスに対する処理を行います。接続が有効な場合、取得したソケット ID を接続リストへ登録します。

Modbus_tcp_terminate_stack	TCP スタック終了処理

【書式】

uint32_t Modbus_tcp_terminate_stack(void_t);

【引数】

void_t	

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_TERM	スタックの終了に失敗

【解説】

本 API は Modbus スタックを終了させます。スタックモードに応じて、各内部 API が呼び出されます。スタックモードが MODBUS_TCP_SERVER_MODE の場合、Modbus_tcp_server_terminate_stack ()を呼び出します。スタックモードがゲートウェイモードの場合、Modbus_tcp_server_terminate_stack()に加えて、Modbus_tcp_terminate_gateway_stack()を呼び出します。

10.1.2 Modbus シリアル

10.1.2.1 プロトコルスタックの初期化

プロトコルスタックの初期化で使用する API は以下になります。

Modbus_serial_stack_init	Modbus シリアルスタックの初期化

【書式】

```
uint32_t Modbus_serial_stack_init(p_serial_stack_init_info_t pt_serial_stack_init_info,
                                  p_serial_gpio_cfg_t pt_serial_gpio_cfg_t,
                                  uint8_t u8_stack_mode,
                                  uint8_t u8_slave_id);
```

【引数】

p_serial_stack_init_info_t	pt_serial_stack_init_info	シリアル通信パラメータテーブルへのポインタ
p_serial_gpio_cfg_t	pt_serial_gpio_cfg_t	I/O ポートコンフィグレーションテーブルへのポインタ
uint8_t	u8_stack_mode	スタックの動作モードを指定
uint8_t	u8_slave_id	デバイスのスレーブ ID　スレーブモードの場合のみ有効

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_INVALID_STACK_MODE	スタックモードの指定が無効
ERR_INVALID_SLAVE_ID	スレーブ ID が無効
ERR_INVALID_STACK_INIT_PARAMS	ユーザ指定の情報に無効なパラメータがある
ERR_STACK_INIT	スタックの起動に失敗

【解説】

本 API はユーザに指定されたパラメータでシリアルスタックの初期化を行います。

本 API では、以下のパラメータを指定します。

a. p_serial_stack_init_info_t 構造体はシリアル通信に関連する設定を指定します。
b. p_serial_gpio_cfg_t 構造体は RS485 通信に使用する GPIO ポート制御用関数へのポインタを指定します。
c. u8_stack_mode はスタックの種別を指定します。指定された値により、スタックは以下のいずれかのモードで動作します。

スタックモード指定	意味
MODBUS_RTU_MASTER_MODE	Modbus Stack RTU マスタモードを選択
MODBUS_RTU_SLAVE_MODE	Modbus Stack RTU スレーブモードを選択
MODBUS_ASCII_MASTER_MODE	Modbus Stack ASCII マスタモードを選択
MODBUS_ASCII_SLAVE_MODE	Modbus Stack ASCII スレーブモードを選択

d. u8_slave_id には、スレーブモードにおけるデバイス ID を指定します。引数で指定できる範囲は、1～247 になります。

p_serial_stack_init_info_t 構造体及び p_serial_gpio_cfg_t 構造体のパラメータ詳細については、10.1.1.1 章を参照してください。

Modbus_slave_map_init	Modbus ファンクションコードマッピング API

【書式】

uint32_t Modbus_slave_map_init (p_slave_map_init_t pt_slave_func_tbl);

【引数】

p_slave_map_init_t	pt_slave_func_tbl	ファンクションコードマッピングテーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_INVALID_STACK_INIT_PARAMS	引数が NULL
ERR_MEM_ALLOC	メモリ確保に失敗

【解説】

この API は、Modbus TCP 時と同じ関数になります。詳細は、10.1.1.1 章を参照してください。

10.1.2.2 マスタモード API

マスタモードで使用する API は以下になります。

Modbus_read_coils	Read coils 実行

【書式】

```
uint32_t Modbus_read_coils(p_req_read_coils_t pt_req_read_coils,
                           p_resp_read_coils_t pt_resp_read_coils,
                           fp_callback_notify_t fp_callback_notify);
```

【引数】

p_req_read_coils_t	pt_req_read_coils	read coil 要求データ構造体へのポインタ
p_resp_read_coils_t	pt_resp_read_coils	read coil 応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスへの送受信に失敗
ERR_ILLEGAL_NUM_OF_COILS	読み出し数（number of coils）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Read coils をスレーブデバイスに対して要求する場合に使用します。API がエラーを返した場合、応答データは無効となります。

・Read coils 要求テーブル (req_read_coils_t)

```
typedef struct _req_read_coils{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする coil の開始アドレス */
    uint16_t    u16_num_of_coils;       /* リードする coil の個数 */
}req_read_coils_t, *p_req_read_coils_t;
```

・Read coils 応答テーブル (resp_read_coils_t)

```
struct _resp_read_coils{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID（自局 ID） */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                           正常終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru8_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA]; /* リードデータ */
}resp_read_coils_t, *p_resp_read_coils_t;
```

Modbus_read_discrete_inputs	Read discrete inputs 実行

【書式】

uint32_t Modbus_read_discrete_inputs(p_req_read_inputs_t pt_req_read_inputs,
p_resp_read_inputs_t pt_resp_read_inputs,
fp_callback_notify_t fp_callback_notify);

【引数】

p_req_read_inputs_t	pt_req_read_inputs	Read input 要求データ構造体へのポインタ
p_resp_read_inputs_t	pt_resp_read_inputs	Read input 応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_INPUTS	読み出し数（number of inputs）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Read discrete inputs をスレーブデバイスに対して要求する場合に使用します。API がエラーを返した場合、応答データは無効となります。

・Read inputs 要求テーブル(req_read_inputs_t)

```
typedef struct _req_read_inputs{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする discrete inputs の開始アドレス */
    uint16_t    u16_num_of_inputs;      /* リードするレジスタの個数 */
}req_read_inputs_t, *p_req_read_inputs_t;
```

・Read inputs 応答テーブル (resp_read_inputs_t)

```
typedef struct _resp_read_inputs{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                           正常終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru8_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA]; /* リードデータ */
}resp_read_inputs_t, *p_resp_read_inputs_t;
```

Modbus_read_holding_registers	Read holding registers 実行

【書式】

```
uint32_t Modbus_read_holding_registers(p_req_read_holding_reg_t pt_req_read_holding_reg,
                                       p_resp_read_holding_reg_t pt_resp_read_holding_reg,
                                       fp_callback_notify_t fp_callback_notify);
```

【引数】

p_req_read_holding_reg_t	pt_req_read_holding_reg	Read holding reg.要求データ構造体へのポインタ
p_resp_read_holding_reg_t	pt_resp_read_holding_reg	Read holding reg.応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	読み出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Read holding registers をスレーブデバイスに対して要求する場合に使用します。API がエラーを返した場合、応答データは無効となります。

・Read holding registers 要求テーブル(req_read_holding_reg_t)

```
typedef struct _req_read_holding_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする holding register の開始アドレス */
    uint16_t    u16_num_of_reg;         /* リードするレジスタの個数 */
}req_read_holding_reg_t, *p_req_read_holding_reg_t;
```

・Read holding registers 応答テーブル(resp_read_holding_reg_t)

```
typedef struct _resp_read_holding_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常
                                           終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru16_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];   /* リードデータ */
}resp_read_holding_reg_t, p_resp_read_holding_reg_t;
```

Modbus_read_input_registers	Read input registers 実行

【書式】

uint32_t Modbus_read_input_registers(p_req_read_input_reg_t pt_req_read_input_reg,
p_resp_read_input_reg_t pt_resp_read_input_reg,
fp_callback_notify_t fp_callback_notify);

【引数】

p_req_read_input_reg_t	pt_req_read_input_reg	Read input reg.要求データ構造体へのポインタ
p_resp_read_input_reg_t	pt_resp_read_input_reg	Read input reg.応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	読み出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Read input registers をスレーブデバイスに対して要求する場合に使用します。API がエラーを返した場合、応答データは無効となります。

・Read input registers 要求テーブル (req_read_input_reg_t)

```
typedef struct _req_read_input_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする input register の開始アドレス */
    uint16_t    u16_num_of_reg;         /* リードするレジスタの個数 */
}req_read_input_reg_t, *p_req_read_input_reg_t;
```

・Read input registers 応答テーブル(resp_read_input_reg_t)

```
typedef struct _resp_read_input_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常
                                           終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru16_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA]; /* リードデータ */
}resp_read_input_reg_t, p_resp_read_input_reg_t;
```

Modbus_write_single_coil	Write single coil 実行

【書式】

```
uint32_t Modbus_write_single_coil(p_req_write_single_coil_t pt_req_write_single_coil,
                                  p_resp_write_single_coil_t pt_resp_write_single_coil,
                                  fp_callback_notify_t fp_callback_notify);
```

【引数】

p_req_write_single_coil_t	pt_req_write_single_coil	Write single coil 要求データ構造体へのポインタ
p_resp_write_single_coil_t	pt_resp_write_single_coil	Write single coil 応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_OUTPUT_VALUE	出力指定値が不正
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Write single coil をスレーブデバイスに対して要求する場合に使用します。

・Write single coil 要求テーブル(req_write_single_coil_t)

```
typedef struct _req_write_single_coil
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_output_addr;        /* ライトする coil のアドレス */
    uint16_t    u16_output_value;       /* ライトする値 */
}req_write_single_coil_t, *p_req_write_single_coil_t;
```

・Write single coil 応答テーブル(resp_write_single_coil_t)

```
typedef struct _resp_write_single_coil{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終
                                           了の場合は 0 を設定。 */
    uint16_t    u16_output_addr;        /* ライトしたアドレス */
    uint16_t    u16_output_value;       /* ライトしたデータ */
}resp_write_single_coil_t, *p_resp_write_single_coil_t;
```

Modbus_write_single_reg	Write single register 実行

【書式】

uint32_t Modbus_write_single_reg(p_req_write_single_reg_t pt_req_write_single_reg,
p_resp_write_single_reg_t pt_resp_write_single_reg,
fp_callback_notify_t fp_callback_notify);

【引数】

p_req_write_single_reg_t	pt_req_write_single_reg	Write single reg.要求データ構造体へのポインタ
p_resp_write_single_reg_t	pt_resp_write_single_reg	Write single reg.応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Write single register をスレーブデバイスに対して要求する場合に使用されます。

・Write single register 要求テーブル(req_write_single_reg_t)

```
typedef struct _req_write_single_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_register_addr;      /* ライトするレジスタのアドレス */
    uint16_t    u16_register_value;     /* ライトするデータ */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write single register 応答テーブル(resp_write_single_reg_t)

```
typedef struct _resp_write_single_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終了の
                                           場合は 0 を設定。 */
    uint16_t    u16_register_addr;      /* ライトしたレジスタのアドレス */
    uint16_t    u16_register_value;     /* ライトしたデータ */
}resp_write_single_reg_t, *p_resp_write_single_reg_t;
```

Modbus_write_multiple_coils	Write multiple coils 実行

【書式】

```
uint32_t Modbus_write_multiple_coils(p_req_write_multiple_coils_t pt_req_write_multiple_coils,
                                     p_resp_write_multiple_coils_t pt_resp_write_multiple_coils,
                                     fp_callback_notify_t fp_callback_notify);
```

【引数】

p_req_write_multiple_coils_t	pt_req_write_multiple_coils	Write multiple coils 要求データ構造体へのポインタ
p_resp_write_multiple_coils_t	pt_resp_write_multiple_coils	Write multiple coils 応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_OUTPUTS	書き出し数（number of outputs）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Write multiple coils をスレーブデバイスに対して要求する場合に使用します。

・Write multiple coils 要求テーブル(req_write_multiple_coils_t)

```
typedef struct _req_write_single_reg{
    uint16_t    u16_transaction_id;             /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;                /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                    /* スレーブ ID */
    uint16_t    u16_start_addr;                 /* ライトする coil の開始アドレス */
    uint16_t    u16_num_of_outputs;             /* ライトする coil の個数 */
    uint8_t     u8_num_of_bytes;                /* ライトするデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA];   /* ライトデータ */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write multiple coils 応答テーブル(resp_write_multiple_coils_t)

```
typedef struct _resp_write_multiple_coils{
    uint16_t    u16_transaction_id;             /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;                /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                    /* スレーブ ID */
    uint8_t     u8_exception_code;              /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                                   正常終了の場合は 0 を設定。 */
    uint16_t    u16_start_addr;                 /* ライトした開始アドレス */
    uint16_t    u16_num_of_outputs;             /* ライトしたデータ */
}resp_write_multiple_coils_t, *p_resp_write_multiple_coils_t;
```

Modbus_write_multiple_reg	Write multiple registers 実行

【書式】

```
uint32_t Modbus_write_multiple_reg(p_req_write_multiple_reg_t pt_req_write_multiple_reg,
                                   p_resp_write_multiple_reg_t pt_resp_write_multiple_reg,
                                   fp_callback_notify_t fp_callback_notify);
```

【引数】

p_req_write_multiple_reg_t	pt_req_write_multiple_reg	Write multiple reg.要求データ構造体へのポインタ
p_resp_write_multiple_reg_t	pt_resp_write_multiple_reg	Write multiple reg.応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	書き出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Write multiple registers をスレーブデバイスに対して要求する場合に使用されます。

・Write multiple registers 要求テーブル(req_write_multiple_reg_t)

```
typedef struct _req_write_multiple_reg{
    uint16_t    u16_transaction_id;         /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;            /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                /* スレーブ ID */
    uint16_t    u16_start_addr;             /* ライトするレジスタの開始アドレス */
    uint16_t    u16_num_of_reg;             /* ライトするレジスタの個数 */
    uint8_t     u8_num_of_bytes;            /* ライトするデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];   /* ライトデータ */
}req_write_multiple_reg_t, *p_req_write_multiple_reg_t;
```

・Write multiple registers 応答テーブル(resp_write_multiple_reg_t)

```
typedef struct _resp_write_multiple_reg{
    uint16_t    u16_transaction_id;         /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;            /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                /* スレーブ ID */
    uint8_t     u8_exception_code;          /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終
                                               了の場合は 0 を設定。 */
    uint16_t    u16_start_addr;             /* ライトした開始アドレス */
    uint16_t    u16_num_of_reg;             /* ライトしたレジスタの個数 */
}resp_write_multiple_reg_t, *p_resp_write_multiple_reg_t;
```

Modbus_read_write_multiple_reg	Read/Write multiple registers 実行

【書式】

uint32_t Modbus_read_write_multiple_reg(p_req_read_write_multiple_reg_t pt_req_read_write_multiple_reg,
p_resp_read_write_multiple_reg_t pt_resp_read_write_multiple_reg,
fp_callback_notify_t fp_callback_notify);

【引数】

p_req_read_write_multiple_reg_t	pt_req_read_write_multiple_reg	Read/Write multiple reg.要求データ構造体へのポインタ
p_resp_read_write_multiple_reg_t	pt_resp_read_write_multiple_reg	Read/Write multiple reg.応答データ構造体へのポインタ
fp_callback_notify_t	fp_callback_notify	ノンブロッキングモード時、通知用コールバック関数へのポインタを指定します。NULL が指定された場合、API はブロッキングモードで動作します。

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_OUTPUT_VALUE	読み出し/書き込み数に異常がある
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、ファンクションコード Read/Write multiple registers をスレーブデバイスに対して要求する場合に使用されます。API がエラーを返した場合、応答データは無効となります。

・Read/Write multiple registers 要求テーブル(req_read_write_multiple_reg_t)

```
typedef struct _req_read_write_multiple_reg{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint16_t    u16_read_start_addr;         /* リードするレジスタの開始アドレス */
    uint16_t    u16_num_to_read;             /* リードするレジスタの個数 */
    uint16_t    u16_write_start_addr;        /* ライトするレジスタの開始アドレス */
    uint16_t    u16_num_to_write;            /* ライトするレジスタの個数 */
    uint8_t     u8_write_num_of_bytes;       /* ライトするデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];    /* ライトするデータ */
}req_read_write_multiple_reg_t, *p_req_read_write_multiple_reg_t;
```

・Read/Write multiple registers 応答テーブル(resp_read_write_multiple_reg_t)

```
typedef struct _resp_read_write_multiple_reg{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint8_t     u8_exception_code;           /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                                正常終了の場合は 0 を設定。0 以外が設定された場合、
                                                aru16_read_data は無効となります。 */
    uint16_t    u8_num_of_bytes;             /* 更新したデータのバイト数 */
    uint16_t    aru16_read_data[MAX_REG_DATA]; /* リードデータ */
}resp_read_write_multiple_reg_t, *p_resp_read_write_multiple_reg_t;
```

Modbus_callback_notify	通知用コールバック関数

【書式】

void Modbus_callback_notify(uint32_t u32_resp_code);

【引数】

uint32_t	u32_resp_code	応答エラーコード

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、呼び出し元でコールバック関数を登録していない場合に、スタックによって呼び出されるデフォルトのコールバック関数になります。これは、本スタックがマスタモードの場合のみ適用されます。

Read/Write 要求がスレーブから応答データを取得するとき、スタックは登録されたコールバック関数を呼び出します。

10.1.2.3 タスク

以下の API はタスクのメイン処理になります。

Modbus_serial_task	Modbus シリアル処理タスク

【書式】

void_t Modbus_serial_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

このタスクは、スタックモードに応じて、スレーブまたはマスタモードのタスクとして実行されます。

スタックモードがマスタモードの場合、タスクはユーザからの要求を待ちます。ユーザにより各ファンクションコード API から要求が出されたら、要求の内容を検証します。問題がなければ、パケットを構築し、スレーブデバイスにそのパケットを送信し、スレーブからの応答を待ちます。ユーザによってコールバック関数が提供されていれば、タスクは応答データ受信時にそのコールバック関数を呼び出します。

スタックモードがスレーブモードの場合、このタスクはマスタデバイスからのデータ受信を待ちます。データを受信すると、受信パケットを解析し Modbus パケットならば、各ファンクションコードに対応した処理を実行したのち、応答パケットを構築しマスタ側に送信します。

Modbus_serial_recv_task	Modbus シリアルパケット受信タスク

【書式】

void_t Modbus_serial_recv_task(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

このタスクは、シリアル I/F からのデータ受信に使用されます。

シリアル I/F 受信割り込みイベントが発生すると、シリアル I/F から受信データの読み出しを行い、RTU/ASCII 各モードに対応したバッファリング処理を呼び出します。

シリアル I/F ステータス割り込みを検出すると、ステータス割り込み用ドライバ関数を呼び出します。ステータス割り込み発生時の詳細については、「RX72M CPU カード ハードウェアマニュアル」を参照してください。

タイマ割り込みを検出すると、バッファリング停止処理を呼び出します。

Serial_recv_task	Modbus シリアルデータ受信タスク

【書式】

void Serial_recv_task(void);

【引数】

void

【戻り値】

void

【エラーコード】

―

【解説】

このタスクは、シリアル I/F からのデータ受信割り込みを検出し、シリアル I/F 受信割り込みイベントを発生させます。

Modbus_serial_stack_terminate	Modbus シリアルタスク終了処理

【書式】

uint32_t Modbus_serial_stack_terminate(void_t);

【引数】

void_t

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_TERM	スタック終了に失敗

【解説】

本 API は、動作中のシリアルスタックを終了させます。

10.1.3 ユーザ定義関数

ユーザ定義関数を以下に示すファイルで定義しています。
/ r_modbus_rx/src/src/modbus_user.c

スレーブモード時に、各ファンクションを処理するために、ユーザ定義の Read/Write 関数を使用します。
Coil/Discrete Input/holding resister/input register の各アドレスの対応する、Read/write 関数とそのテーブルを用意しています。

以下に示すようにユーザ定義関数を設定すると、Coil アドレス 00001(オフセット 0)の読み込み要求が来た場合、ファンクションコード 1 の処理関数 cd_fun_code01()は、Coil の Read 処理関数テーブル MB_Coils_Read[]から、Coil アドレス 00001 の Coil Read 処理関数 MB_Coil_Read_00001 を呼び出します。

```
/*Read processing function of Coil address 00001 (offset 0) */
uint8_t MB_Coil_Read_00001(uint8_t *coil)
{
   *coil = 0;
   if (LED0 == 0)
   {
      *coil |= 1;
   }
   g_Coils_Area[0]=*coil;
   return ERR_OK;
}

/* Coil Read Processing Function Table */
uint8_t (*MB_Coils_Read[])(uint8_t *data)={
   MB_Coil_Read_00001,    /* 00001 */
   MB_Coil_Read_00002,    /* 00002 */
...
```

・Coil 用関数(Read) 関数テーブル (MB_Coils_Read)

```
uint8_t (*MB_Coils_Read[])(uint8_t *data)={
        MB_Coil_Read_00001,      /* 00001 */
        MB_Coil_Read_00002,      /* 00002 */
        MB_Coil_Read_00003,      /* 00003 */
        MB_Coil_Read_00004,      /* 00004 */
        MB_Reg_Exp_Addr_p8       /* 00005 */
        MB_Reg_Exp_Addr_p8       /* 00006 */
        MB_Reg_Exp_Addr_p8       /* 00007 */
        MB_Reg_Exp_Addr_p8       /* 00008 */
};
```

・Coil 用関数(Write) 関数テーブル (MB_Coils_Write)

```
uint8_t (*MB_Coils_Write[])(uint8_t data)={
        MB_Coil_Write_00001,        /* 00001 */
        MB_Coil_Write_00002,        /* 00002 */
        MB_Coil_Write_00003,        /* 00003 */
        MB_Coil_Write_00004,        /* 00004 */
        MB_Reg_Exp_Addr_p8          /* 00005 */
        MB_Reg_Exp_Addr_p8          /* 00006 */
        MB_Reg_Exp_Addr_p8          /* 00007 */
        MB_Reg_Exp_Addr_p8          /* 00008 */
};
```

・Discrete Input 用関数テーブル (MB_Discretes_Input)

```
uint8_t (*MB_Discretes_Input[])(uint8_t *data)={
        MB_D_Read_10001,            /* 10001 */
        MB_D_Read_10002,            /* 10002 */
        MB_D_Read_10003,            /* 10003 */
        MB_D_Read_10004,            /* 10004 */
        MB_D_Read_10005,            /* 10005 */
        MB_D_Read_10006,            /* 10006 */
        MB_D_Read_10007,            /* 10007 */
        MB_D_Read_10008,            /* 10008 */
        MB_D_Read_10009,            /* 10009 */
        MB_Reg_Exp_Addr_p8,         /* 10010 */
        MB_D_Read_10011,            /* 10011 */
        MB_D_Read_10012,            /* 10012 */
};
```

・Holding resister 用関数(Read) テーブル (MB_HoldingRegs_Read)

```
uint8_t (*MB_HoldingRegs_Read[])(uint16_t *data)={
        MB_Reg_Read_40001,          /* 40001 */
        MB_Reg_Read_40002,          /* 40002 */
        MB_Reg_Read_40003,          /* 40003 */
        MB_Reg_Exp_Addr_p16,        /* 40004 */
        MB_Reg_Exp_Addr_p16,        /* 40005 */
        MB_Reg_Exp_Addr_p16,        /* 40006 */
        MB_Reg_Read_40007,          /* 40007 */
};
```

・Input register 用関数テーブル (MB_Input_Regs)

```
uint8_t (*MB_HoldingRegs_Read[])(uint16_t *data)={
        MB_IReg_Read_30001,         /* 30001 */
        MB_IReg_Read_30002,         /* 30002 */
        MB_IReg_Read_30003,         /* 30003 */
        MB_Reg_Exp_Addr_p16,        /* 30004 */
        MB_Reg_Exp_Addr_p16,        /* 30005 */
        MB_Reg_Exp_Addr_p16,        /* 30006 */
        MB_IReg_Read_30008,         /* 30007 */
};
```

・Holding resister 用関数(Write) テーブル (MB_HoldingRegs_Write)

```
uint8_t (*MB_HoldingRegs_Write[])(uint16_t data)={
        MB_Reg_Write_40001,         /* 40001 */
        MB_Reg_Write_40002,         /* 40002 */
        MB_Reg_Write_40003,         /* 40003 */
        MB_Reg_Exp_Addr_p16,        /* 40004 */
        MB_Reg_Exp_Addr_p16,        /* 40005 */
        MB_Reg_Exp_Addr_p16,        /* 40006 */
        MB_Reg_Write_40007,         /* 40007 */
};
```

10.2 内部 API

本章では、スタック内部で使用される API について説明します。

10.2.1 パケット構築および解析 API

10.2.1.1 シリアル接続管理

以下の API は、シリアル通信のパケット処理に使用されます

Modbus_serial_frame_pkt	Modbus シリアルパケット構築	

【書式】

```
void_t Modbus_serial_frame_pkt(puint8_t pu8_mb_snd_pkt,
                               puint32_t pu32_snd_pkt_len,
                               p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_snd_pkt	送信するパケットを格納する配列へのポインタ
puint32_t	pu32_snd_pkt_len	構築したパケットの長さ
pt_mbserial_queue_elmnt	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、ユーザアプリケーションによって与えられた情報を元に送信するパケットを構築します。スタックモードに応じて、対応する関数へと情報を引き渡します。

まず、マスタモードの場合は Modbus_master_frame_request()を、スレーブモードの場合は Modbus_slave_frame_response()をそれぞれ呼び出して、必要な情報を収集します。

次に、収集した情報を元に、RTU モードの場合は Modbus_rtu_frame_pkt()を、ASCII モードの場合は Modbus_ascii_frame_pkt()をそれぞれ呼び出し、パケットを構築します。

・シリアルパケットキューテーブル (mbserial_queue_elmnt_t)

```
typedef struct _mbserial_queue_elmnt{
    fp_callback_notify_t    fpt_callback_notify;        /* ノンブロッキングモード時に呼び出される
                                                           通知用コールバック関数へのポインタ
                                                           このメンバが NULL の場合、ブロッキングモ
                                                           ードとなります */
    void*                   pu8_output_response;        /* 各ファンクションコード用応答テーブルへ
                                                           のポインタ */
    void*                   pu8_input_request;          /* 各ファンクションコード用要求テーブルへ
                                                           のポインタ */
    uint32_t                u32_num_of_bytes;           /* データパケットの長さ */
    uint8_t                 aru8_data_packet[MAX_DATA_SIZE]; /* パケットデータ */
    uint8_t                 u8_cmd_mode;                /* 当該パケットの処理モード */
    uint8_t                 u8_slave_id;                /* スレーブ ID */
    uint8_t                 u8_func_code;               /* ファンクションコード */
}mbserial_queue_elmnt_t, *p_mbserial_queue_elmnt_t;
```

この構造体には以下のマクロが引数として使用されます。

パケット処理モード指定	意味
UNICAST_MODE	当該パケットをユニキャストとして処理
BROADCAST_MODE	当該パケットをブロードキャストとして処理

Modbus_rtu_frame_pkt	Modbus RTU パケット構築

【書式】

```
void_t Modbus_rtu_frame_pkt(puint8_t pu8_mb_snd_pkt,
                            puint32_t pu32_snd_pkt_len,
                            p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【Paramter】

puint8_t	pu8_mb_snd_pkt	送信するパケットを格納する配列へのポインタ
puint32_t	pu32_snd_pkt_len	構築したパケットの長さ
pt_mbserial_queue_elmnt	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、ユーザアプリケーションによって提供された情報を元に、RTU デバイスに送信するパケットを構築します。CRC の計算には calculate_crc()を使用しています。

Modbus_ascii_frame_pkt	Modbus ASCII パケット構築

【書式】

```
void_t Modbus_ascii_frame_pkt(puint8_t pu8_mb_snd_pkt,
                              puint32_t pu32_snd_pkt_len,
                              p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_snd_pkt	送信するパケットを格納する配列へのポインタ
puint32_t	pu32_snd_pkt_len	構築したパケットの長さ
pt_mbserial_queue_elmnt	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、ユーザアプリケーションによって提供された情報を元に、ASCII デバイスに送信するパケットを構築します。LRC の計算には calculate_lrc()を使用しています。

Modbus_serial_send_pkt	Modbus シリアルパケット送信

【書式】

```
void_t Modbus_serial_send_pkt(puint8_t pu8_mb_snd_pkt,
                              uint32_t u32_snd_pkt_len);
```

【引数】

puint8_t	pu8_mb_snd_pkt	送信パケットへのポインタ
uint32_t	u32_snd_pkt_len	送信パケットの長さ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、Modbus_serial_send()のラッパ関数になります。

Modbus_serial_send	Modbus シリアルパケット送信

【書式】

```
void_t Modbus_serial_send(puint8_t u8_mb_snd_pkt,
                          uint32_t u32_snd_pkt_len);
```

【引数】

puint8_t	pu8_mb_snd_pkt	送信パケットへのポインタ
uint32_t	u32_snd_pkt_len	送信パケットの長さ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、シリアル I/F を介してパケットを送信します。送信時、スタック初期化関数で登録された RS485 送受信方向制御関数によって、通信方向を送信側に切り替えます。

Modbus_serial_parse_pkt	Modbus シリアルパケット解析

【書式】

```
uint32_t Modbus_serial_parse_pkt(puint8_t pu8_mb_recv_pkt,
                                 puint32_t pu32_recv_pkt_len,
                                 p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信パケットを格納した配列へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求したファンクションコード以外の応答が返ってきた）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_OK_WITH_NO_RESPONSE	正常終了（ブロードキャスト指定により応答データなし）
ERR_MSG_SIZE_OVER	受信した要求または応答データサイズが最大長を超えている

【解説】

本関数は、ユーザアプリケーションによって与えられた情報を元に受信したパケットの解析を行います。スタックのモードに応じて、RTU モードの場合は Modbus_rtu_parse_pkt()を、ASCII モードの場合は Modbus_ascii_parse_pkt()をそれぞれ呼び出します。

Modbus_rtu_parse_pkt	Modbus RTU パケット解析

【書式】

```
uint32_t Modbus_rtu_parse_pkt(puint8_t pu8_mb_recv_pkt,
                              puint32_t pu32_recv_pkt_len,
                              p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信パケットを格納した配列へのポインタ
uint32_t	pu32_recv_pkt_len	受信パケットの長さ
p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブ ID が不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求を出したファンクションコードとコードで応答が返ってきた）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_OK_WITH_NO_RESPONSE	正常終了（ブロードキャスト指定により応答データなし）
ERR_MSG_SIZE_OVER	受信した要求または応答データサイズが最大長を超えている

【解説】

本 API は、シリアル I/F から受信したパケットを解析します。スタックのモードに応じて、マスタモードの場合は Modbus_master_parse_pkt()を、スレーブモードの場合は Modbus_slave_parse_pkt()をそれぞれ呼び出します。

Modbus_ascii_parse_pkt	Modbus ASCII パケット解析

【書式】

```
uint32_t Modbus_ascii_parse_pkt (puint8_t pu8_mb_recv_pkt,
                                 puint32_t pu32_recv_pkt_len,
                                 p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信パケットを格納した配列へのポインタ
uint32_t	pu32_recv_pkt_len	受信パケットの長さ
p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブ ID が不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求を出したファンクションコードとコードで応答が返ってきた）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_OK_WITH_NO_RESPONSE	正常終了（ブロードキャスト指定により応答データなし）
ERR_MSG_SIZE_OVER	受信した要求または応答データサイズが最大長を超えている

【解説】

本 API は、シリアル I/F から受信したパケットを解析します。本関数では、指定された ASCII パケットを RTU パケットに変換した後、スタックモードに対応した各パケット解析 API を呼び出します。

Modbus_master_parse_pkt	Modbus マスターパケット解析

【書式】

```
uint32_t Modbus_master_parse_pkt(puint8_t pu8_mb_recv_pkt,
                                 puint32_t pu32_recv_pkt_len,
                                 p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信パケットを格納した配列へのポインタ
uint32_t	pu32_recv_pkt_len	受信パケットの長さ
p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブ ID が不一致（要求を出したスレーブ以外から応答が返ってきた）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致（要求を出したファンクションコードとコードで応答が返ってきた）
ERR_LRC_CHECK	LRC チェックで異常（ASCII モード）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、シリアル I/F から受信したパケットを解析します。解析結果をもとに、ユーザ情報構造体（pt_mbserial_queue_elmnt）を更新します。

Modbus_slave_parse_pkt	Modbus スレーブパケット解析

【書式】

```
uint32_t Modbus_slave_parse_pkt(puint8_t pu8_mb_recv_pkt,
                                puint32_t pu32_recv_pkt_len,
                                p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信パケットを格納した配列へのポインタ
uint32_t	pu32_recv_pkt_len	受信パケットの長さ
p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_LRC_CHECK	LRC チェックで異常（ASCII モード）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_OK_WITH_NO_RESPONSE	正常終了（ブロードキャスト指定により応答データなし）

【解説】

本 API は、指定されたパケットを解析し、ユーザが登録した各ファンクションコードに対応したコールバック関数を実行します。コールバック実行後、その実行結果を元にユーザ情報構造体（pt_mbserial_queue_elmnt）を更新します。本関数では、各コールバック実行用に要求および応答テーブルのメモリを動的に確保します。要求テーブルは本関数内で解放されますが、応答テーブルは応答パケットを構築する段階で解放されます。

Modbus_master_validate_pkt	マスターモードパケット検証

【書式】

uint32_t Modbus_master_validate_pkt(p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);

【Paramter】

p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブ ID が不一致
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_LRC_CHECK	LRC チェックで異常（ASCII モード）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある

【解説】

本 API は、マスタモードでシリアル I/F から受信したパケットに異常がないかを検証します。本関数では、パケット中のスレーブ ID およびファンクションコードがチェックされます。

Modbus_slave_validate_pkt	スレーブモードパケット検証

【書式】

uint32_t Modbus_slave_validate_pkt(p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);

【引数】

p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_LRC_CHECK	LRC チェックで異常（ASCII モード）
ERR_CRC_CHECK	CRC チェックで異常（RTU モード）
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_INVALID_SLAVE_ID	スレーブIDが異常
ERR_SLAVE_ID_MISMATCH	要求パケット内のスレーブ ID が自局またはブロードキャスト ID でない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_OK_WITH_NO_RESPONSE	正常終了（ブロードキャスト指定により応答データなし）

【解説】

本 API は、スレーブモードでシリアル I/F から受信したパケットに異常がないかを検証します。本関数では、パケット中のスレーブ ID がチェックされます。

Modbus_master_frame_request	マスタモード要求パケット構築

【書式】

void_t Modbus_master_frame_request(p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);

【引数】

p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、スタックがマスタモードの場合に呼び出されます。ユーザアプリケーションによって提供されたリクエスト情報を元に、引数で指定される p_mbserial_queue_elmnt_t 構造体にリクエストパケットを構築します。

Modbus_slave_frame_response	スレーブモード応答パケット構築

【書式】

void_t Modbus_slave_frame_response(p_mbserial_queue_elmnt_t pt_mbserial_queue_elmnt);

【引数】

p_mbserial_queue_elmnt_t	pt_mbserial_queue_elmnt	ユーザ情報を含んだ構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、スタックがスレーブモードの場合に呼び出されます。パケット解析 API で作成される応答データ構造体の情報を元に、引数で指定される p_mbserial_queue_elmnt_t 構造体に応答パケットを構築します。

Modbus_serial_write	Modbus シリアル I/F 書き込み

【書式】

```
void_t Modbus_serial_write(puint8_t pu8_mb_snd_data,
                           uint32_t u32_data_size);
```

【引数】

puint8_t	pu8_mb_snd_data	送信データを格納した領域へのポインタ
uint32_t	u32_data_size	送信データのサイズ

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、指定されたデータ数をシリアル I/F に書き込みます。シリアル I/F ドライバ関数を使用し、g_modbus_sci_handle で定義されたチャネル番号に対して書き込みを行います。

Modbus_serial_read	Modbus シリアル I/F 読み込み

【書式】

```
uint32_t Modbus_serial_read(puint8_t pu8_mb_read_char);
```

【引数】

puint8_t	pu8_mb_read_char	読み込んだ文字を格納した領域へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_UART_RECV_OPERATION	リード失敗

【解説】

本 API は、シリアル I/F チャネルから 1 バイトデータを読み込みます。シリアル I/F ドライバ関数を使用し、g_modbus_sci_handle で定義されたチャネル番号から読み込みを行います。

Modbus_rtu_crc_calculate	CRC（Cyclic Redundancy Check）計算

【書式】

```
uint32_t Modbus_rtu_crc_calculate(puint8_t pu8_mb_pkt,
                                  uint32_t u32_pkt_len);
```

【引数】

puint8_t	pu8_mb_pkt	パケットを格納した配列へのポインタ
uint32_t	u32_pkt_len	パケットの長さ

【戻り値】

uint32_t	CRC 計算結果

【エラーコード】

—

【解説】

本関数は、指定された配列の CRC を計算します。

Modbus_rtu_crc_validate	CRC（Cyclic Redundancy Check）検証

【書式】

```
uint32_t Modbus_rtu_crc_validate(puint8_t pu8_mb_pkt,
                                 uint32_t u32_pkt_len);
```

【引数】

puint8_t	pu8_mb_pkt	パケットが格納された配列へのポインタ
uint32_t	u32_pkt_len	パケットの長さ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_CRC_CHECK	CRC 異常

【解説】

本関数は、Modbus パケットの CRC に異常がないか検証します。指定された配列の末尾２バイトを CRC とし、それ以前のデータで CRC を計算して比較を行います。

Modbus_ascii_lrc_calculate	LRC（Longitudinal Redundancy Check）計算

【書式】

```
uint8_t Modbus_ascii_lrc_calculate(puint8_t pu8_mb_pkt,
                                   uint32_t u32_pkt_len);
```

【引数】

puint8_t	pu8_mb_pkt	パケットが格納された配列へのポインタ
uint32_t	u32_pkt_len	パケットの長さ

【戻り値】

uint32_t	LRC 計算結果

【エラーコード】

—

【解説】

本関数は、指定された配列の LRC を計算します。

Modbus_ascii_lrc_validate	LRC（Longitudinal Redundancy Check）検証

【書式】

```
uint32_t Modbus_ascii_lrc_validate(puint8_t pu8_mb_pkt,
                                   uint32_t u32_pkt_len);
```

【引数】

puint8_t	pu8_mb_pkt	パケットが格納された配列へのポインタ
uint32_t	u32_pkt_len	パケットの長さ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_LRC_CHECK	LRC 異常

【解説】

本関数は、Modbus パケットの LRC に異常がないか検証します。指定された配列の末尾２バイトを LRC とし、それ以前のデータで LRC を計算して比較を行います。

Modbus_rtu_to_ascii RTU→ASCII 変換

【書式】

```
void_t Modbus_rtu_to_ascii(puint8_t pu8_rtu_pkt,
                           uint32_t u32_rtu_pkt_size,
                           puint8_t pu8_ascii_pkt,
                           puint32_t pu32_ascii_pkt_size);
```

【引数】

puint8_t	pu8_rtu_pkt	RTU パケットを格納した領域へのポインタ
uint32_t	u32_rtu_pkt_size	RTU パケットのサイズ
puint8_t	pu8_ascii_pkt	ASCII パケットを格納する領域へのポインタ
puint32_t	pu32_ascii_pkt_size	ASCII パケットの長さ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、指定された RTU パケットを ASCII パケットに変換します。

Modbus_ascii_to_rtu ASCII→RTU 変換

【書式】

```
void_t Modbus_ascii_to_rtu(puint8_t pu8_ascii_pkt,
                           uint32_t u32_ascii_pkt_size,
                           puint8_t pu8_rtu_pkt,
                           puint32_t pu32_rtu_pkt_size);
```

【引数】

puint8_t	pu8_ascii_pkt	ASCII パケットを格納した領域へのポインタ
uint32_t	u32_ascii_pkt_size	ASCII パケットのサイズ
puint8_t	pu8_rtu_pkt	RTU パケットを格納する領域へのポインタ
puint32_t	pu32_rtu_pkt_size	RTU パケットの長さ

【戻り値】

void_t

【エラーコード】

—

【解説】

本関数は、指定された ASCII パケットを RTU パケットに変換します。

Modbus_RS485_TX_enable　RS485 送信有効

【書式】

void_t Modbus_RS485_TX_enable(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、RS485 を送信モードに切り替えます。

Modbus_RS485_TX_disable　RS485 送信無効

【書式】

void_t Modbus_RS485_TX_disable(void_t);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、RS485 を受信モードに切り替えます。

Modbus_ascii_recv_char	Modbus ASCII 用受信データバッファリング

【書式】

void_t Modbus_ascii_recv_char(uint8_t u8_read_char);

【引数】

uint8_t	u8_read_char	受信データ

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、Modbus ASCII モード時に受信データのバッファリングを行います。バッファリングは終端文字を検出するか、最大文字数（MAX_ASCII_PACKET_LEN）まで行われます。終端文字を検出すると、スタックのモードに応じて各タスクにパケットが受信できた旨を報告します。

また、本関数は呼び出されると、無通信時間を測定するためにスタック初期化時に指定されたインターバル時間でタイマを起動します。

Modbus_rtu_recv_char	Modbus RTU 用受信データバッファリング

【書式】

void_t Modbus_rtu_recv_char(uint8_t u8_recv_char)

【引数】

uint8_t	u8_recv_char	受信データ

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数は、Modbus RTU モード時に受信データのバッファリングを行います。バッファリングは最大文字数（MAX_RTU_PACKET_LEN）まで行われます。パケットの終端判定は無通信時間を検出するタイマハンドラで行われます。

また、本関数は呼び出されると、無通信時間を測定するためにスタック初期化時に指定されたインターバル時間でタイマを起動します。

Modbus_timer_handler	バッファリング停止処理

【書式】

void Modbus_timer_handler(void);

【引数】

void_t

【戻り値】

void_t

【エラーコード】

―

【解説】

本関数はシリアルデータ受信タスクでタイマ割り込みイベントが発生した際に呼び出されます。

ASCII モードの場合、受信データのバッファリング動作をリセットします。終端文字検出前に発生すると受信中のパケットは破棄されます。

RTU モードの場合、受信データのバッファリングを停止し、スタックのモードに応じて、各タスクへパケットの受信が完了したことを報告します。

10.2.1.2 TCP/IP 接続管理

以下の API は、TCP/IP 処理に使用されます

Modbus_tcp_send	Modbus TCP パケット送信

【書式】

```
uint32_t Modbus_tcp_send(puint8_t pu8_mb_snd_pkt,
                         uint32_t u32_snd_pkt_len,
                         uint8_t u8_soc_id);
```

【引数】		
puint8_t	pu8_mb_snd_pkt	送信パケットを格納した配列へのポインタ
uint32_t	u32_snd_pkt_len	送信パケットの長さ
uint8_t	u8_soc_id	ソケット ID
【戻り値】		
uint32_t	エラーコード	
【エラーコード】		
ERR_OK	正常終了	
ERR_SEND_FAIL	送信失敗	

【解説】

本 API を使用して Modbus ファンクションパケットの送信を行います。実際の送信処理は、Modbus_tcp_send_pkt 関数で行っています。

Modbus_tcp_send_pkt	Modbus TCP パケット送信

【書式】

```
uint32_t Modbus_tcp_send_pkt(puint8_t pu8_mb_snd_pkt,
                             uint32_t u32_snd_pkt_len,
                             uint8_t u8_soc_id);
```

【引数】		
puint8_t	pu8_mb_snd_pkt	送信パケットを格納した配列へのポインタ
uint32_t	u32_snd_pkt_len	送信パケットの長さ
uint8_t	u8_soc_id	ソケット ID
【戻り値】		
uint32_t	エラーコード	
【エラーコード】		
ERR_OK	正常終了	
ERR_SEND_FAIL	送信失敗	

【解説】

本 API は、指定されたパケットを接続済みのソケットに対して書き出します。書き出しには TCP/IP スタック API を使用します。

Modbus_tcp_recv	Modbus TCP パケット受信

【書式】

```
uint32_t Modbus_tcp_recv(puint8_t pu8_mb_recv_pkt,
                         uint32_t u32_recv_pkt_len,
                         uint8_t u8_soc_id);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信バッファのポインタ
uint32_t	u32_recv_pkt_len	受信バッファサイズ
uint8_t	u8_soc_id	ソケット ID

【戻り値】

uint32_t	受信パケットサイズ

【エラーコード】

—

【解説】

本 API を使用して Modbus ファンクションパケットの受信を行います。実際の受信処理は、Modbus_tcp_recv_pkt 関数で行っています。

Modbus_tcp_recv_pkt	Modbus TCP パケット受信

【書式】

```
uint32_t Modbus_tcp_recv_pkt(puint8_t pu8_mb_recv_pkt,
                             uint32_t u32_recv_pkt_len,
                             uint8_t u8_soc_id);
```

【引数】

puint8_t	pu8_mb_recv_pkt	受信バッファのポインタ
uint32_t	u32_recv_pkt_len	受信バッファサイズ
uint8_t	u8_soc_id	ソケット ID

【戻り値】

uint32_t	受信パケットサイズ

【エラーコード】

—

【解説】

本 API は unet3_recv()関数を使用して通信相手からの Modbus ファンクションパケットを受信します。

Modbus_tcp_frame_pkt	Modbus TCP パケット構築

【書式】

```
void_t Modbus_tcp_frame_pkt(puint8_t pu8_mb_snd_pkt,
                            puint32_t pu32_snd_pkt_len,
                            p_mb_tcp_pkt_info_t pt_mb_tcp_pkt_info);
```

【引数】

puint8_t	pu8_mb_snd_pkt	送信パケットを格納する配列へのポインタ
puint32_t	pu32_snd_pkt_len	構築パケットの長さを格納する領域へのポインタ
p_mb_tcp_pkt_info_t	pt_mb_tcp_pkt_info	応答情報を含む構造体へのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、指定された応答情報を元に TCP パケットを構築します。

Modbus_tcp_parse_pkt	Modbus TCP パケット解析

【書式】

```
uint32_t Modbus_tcp_parse_pkt(puint8_t pu8_mb_rcv_pkt,
                              uint32_t u32_recv_pkt_len,
                              p_mb_tcp_pkt_info_t pt_mb_tcp_pkt_info);
```

【引数】

puint8_t	pu8_mb_rcv_pkt	受信パケットが格納されている領域へのポインタ
puint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_mb_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
EXP_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_MEM_ALLOC	メモリ確保に失敗

【解説】

本 API は、指定された TCP パケットを解析し、ユーザが登録した各ファンクションコードに対応したコールバック関数を実行します。コールバック実行後、その実行結果を元に TCP パケット情報テーブル（pt_mb_tcp_pkt_info）を更新します。本関数では、各コールバック実行用に要求および応答テーブルのメモリを動的に確保します。要求テーブルは本関数内で解放されますが、応答テーブルは応答パケットを構築する段階で解放されます。

Modbus_tcp_validate_pkt	Modbus TCP パケット検証

【書式】

```
uint32_t Modbus_tcp_validate_pkt(p_mb_tcp_pkt_info_t pt_mb_tcp_pkt_info,
                                 uint32_t u32_pdu_len);
```

【引数】

p_mb_tcp_pkt_info_t	pt_mb_tcp_pkt_info	TCP パケット情報テーブルへのポインタ
uint32_t	u32_pdu_len	受信した PDU（Protocol Data Unit）の長さ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
EXP_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない

【解説】

本 API は、指定された TCP パケット情報テーブルに格納されているパケットデータに異常がないかを検証します。

Modbus_tcp_init_socket	function for creating server socket

【書式】

```
int8_t Modbus_tcp_init_socket(uint16_t u16_port,
                              pint32_t ps32_listen_fd);
```

【引数】

uint16_t	u16_port	ポート番号
pint32_t	ps32_listen_fd	ソケット識別子を格納する領域へのポインタ

【戻り値】

int8_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SOCK_ERROR	ソケット生成（socket）に失敗
ERR_BIND_ERROR	ソケット登録（bind）に失敗
ERR_LISTEN_ERROR	ソケット接続準備（listen）に失敗

【解説】

本関数は、ソケットの生成をし、クライアントからの接続準備までを行います。

Modbus_tcp_frame_response	Modbus TCP 応答パケット構築

【書式】

```
uint32_t Modbus_tcp_frame_response(uint8_t u8_fn_code,
                                   p_mb_tcp_pkt_info_t pt_mb_tcp_pkt_info);
```

【引数】

uint8_t	u8_fn_code	ファンクションコード
p_mb_tcp_pkt_info_t	pt_mb_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了

【解説】

本 API は、指定された TCP パケット情報テーブルの情報を元に応答用の TCP パケットを構築します。構築したパケットは、TCP パケット情報テーブルに格納されます。

10.2.2 スタックコンフィグレーションおよび管理 API

10.2.2.1 プロトコルスタックの初期化

以下の API は、スタックの初期化処理に使用されます

Modbus_tcp_server_init_stack	Modbus TCP サーバスタック（ゲートウェイ以外）初期化

【書式】

```
uint32_t Modbus_tcp_server_init_stack(uint32_t u32_additional_port,
                                      uint8_t u8_tcp_multiple_client);
```

【引数】		
uint32_t	u32_additional_port	ユーザ指定の追加ポート番号
uint8_t	u8_tcp_multiple_client	マルチクライアント指定
【戻り値】		
uint32_t	エラーコード	
【エラーコード】		
ERR_OK	正常終了	
ERR_STACK_INIT	初期化に失敗	

【解説】

本 API は、TCP スタックの初期化に使用します。具体的には、スタックの動作に必要な以下の３つのタスクを起動します。

・ユーザに指定されたポート番号（デフォルトでは 502）で、クライアントからの接続を監視するタスク。
・クライアント側から送られたデータを受信するタスク。
・受信データを解析し、ユーザによって提供された各ファンクションコードに対応した動作を実行するタスク。

Modbus_tcp_init_gateway_stack	Modbus TCP ゲートウェイ初期化

【書式】

```
uint32_t Modbus_tcp_init_gateway_stack(uint8_t u8_stack_mode,
                                       uint8_t u8_tcp_gw_slave,
                                       p_serial_stack_init_info_t pt_serial_stack_init_info,
                                       p_serial_gpio_cfg_t pt_serial_gpio_cfg_t);
```

【引数】

uint8_t	u8_stack_mode	スタックモード指定
uint8_t	u8_tcp_gw_slave	TCP スタックのゲートウェイモード指定
p_serial_stack_init_info_t	pt_serial_stack_init_info	シリアル通信パラメータテーブルへのポインタ
p_serial_gpio_cfg_t	pt_serial_gpio_cfg_t	I/O ポートコンフィグレーションテーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_INIT	初期化失敗

【解説】

本 API は、TCP スタックをゲートウェイ機能有りで初期化します。初期化する際、シリアルスタックも初期化されます。TCP デバイスに接続されたシリアルデバイスの要求を処理するためにゲートウェイタスクを起動します。

10.2.2.2 IP アドレス管理

以下の API は、IP アドレス管理に使用されます

Modbus_tcp_search_ip_addr	IP アドレス検索

【書式】

```
uint32_t Modbus_tcp_search_ip_addr(pchar_t pu8_search_IP,
                                   puint8_t pu8_ip_idx);
```

【引数】

pchar_t	pu8_search_IP	検索する IP アドレス
puint8_t	pu8_ip_idx	ホスト IP リスト上のインデックス値を格納する領域へのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_IP_NOT_FOUND	指定された IP アドレスが見つからない
ERR_TABLE_EMPTY	ホスト IP リストが空
ERR_TABLE_DISABLED	ホスト IP リストが無効状態

【解説】

本 API は、指定された IP アドレスをホスト IP リストから検索します。

Modbus_tcp_shift_conn_list	Modbus TCP 接続リストシフト

【書式】

```
void_t Modbus_tcp_shift_conn_list(puint8_t pu8_conn_list,
                                  puint8_t pu8_conn_idx);
```

【引数】

puint8_t	pu8_conn_list	接続リストへのポインタ
puint8_t	pu8_conn_idx	シフト開始位置

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、指定された位置から接続リストの内容をインデックス値が小さいほうへシフトさせます。

Modbus_tcp_remove_from_conn_list	Modbus TCP 接続リストからの除外

【書式】

```
void_t Modbus_tcp_remove_from_conn_list(puint8_t pu8_soc_id,
                                        puint8_t pu8_conn_list);
```

【引数】

puint8_t	pu8_soc_id	ソケット ID
puint8_t	pu8_conn_list	接続リストへのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、スタックに保持された接続リストから、指定のソケット ID を除外します。

Modbus_tcp_add_to_conn_list	Modbus TCP 接続リストへの追加

【書式】

```
void_t Modbus_tcp_add_to_conn_list(puint8_t pu8_soc_id,
                                   puint8_t pu8_conn_list);
```

【引数】

puint8_t	pu8_soc_id	ソケット ID
puint8_t	pu8_conn_list	接続リストへのポインタ

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、スタックに保持された接続リストに、指定のソケット ID を追加します。

Modbus_tcp_get_loc_from_list TCP 接続リスト内位置取得

【書式】

```
uint32_t Modbus_tcp_get_loc_from_list(puint8_t pu8_soc_id,
                                      puint8_t pu8_conn_list,
                                      puint8_t pu8_soc_loc);
```

【引数】

puint8_t	pu8_soc_id	ソケット ID
puint8_t	pu8_conn_list	接続リストへのポインタ
puint8_t	pu8_soc_loc	指定されたソケット ID が入っている接続リスト上のインデックス値

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SOC_NOT_FOUND	指定されたソケット ID がみつからない

【解説】

本 API は、接続リストから指定されたソケット ID が入っている位置を取得します。.

Modbus_tcp_update_conn_list TCP 接続リスト更新

【書式】

```
void_t Modbus_tcp_update_conn_list(uint8_t u8_soc_id,
                                   puint8_t pu8_conn_list,
                                   uint8_t u8_add_remove);
```

【引数】

uint8_t	u8_soc_id	ソケット ID
puint8_t	pu8_conn_list	接続リストへのポインタ
uint8_t	u8_add_remove	更新種別指定

【戻り値】

void_t

【エラーコード】

—

【解説】

本 API は、接続リストの更新を行います。接続リストは、最新の接続が配列の最後に、最も古い接続が配列の最初に配置されます。本関数では、以下のマクロが引数として使用されます。

更新種別指定	意味
ADD_TO_CONN_LIST	ソケット ID 追加
REMOVE_FROM_CONN_LIST	ソケット ID 除外

10.2.2.3 タスク終了処理

以下の API は、タスク終了処理に使用されます

Modbus_tcp_server_terminate_stack	Modbus TCP サーバスタック終了

【書式】

uint32_t Modbus_tcp_server_terminate_stack(void_t);

【引数】

void_t	

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_TERM	終了失敗

【解説】

本 API は、Modbus TCP スタック関連のタスクおよびメールボックスを終了させます。

Modbus_tcp_gateway_terminate_stack	Modbus TCP ゲートウェイスタック終了

【書式】

uint32_t Modbus_tcp_gateway_terminate_stack(void_t);

【引数】

void_t	

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_TERM	終了失敗

【解説】

本 API は、Modbus TCP ゲートウェイスタック関連のタスクおよびメールボックスを終了させます。

10.2.2.4 メールボックス

以下の API は、メールボックス管理に使用されます

Modbus_post_to_mailbox	メールボックスへのリクエスト送信

【書式】

```
uint32_t Modbus_post_to_mailbox(uint16_t u16_mbx_id,
                                p_mb_req_mbx_t pt_req_recvd);
```

【引数】		
uint16_t	u16_mbx_id	宛先メールボックス ID
p_mb_req_mbx_t	pt_req_recvd	メールボックスキューテーブルへのポインタ

【戻り値】	
uint32_t	エラーコード

【エラーコード】	
ERR_OK	正常終了
ERR_MAILBOX	メールボックス書き込み失敗
ERR_TCP_SND_MBX_FULL	処理中のメッセージ数が最大に達している

【解説】

本 API は、クライアントから受信したリクエストを受信メールボックスまたはゲートウェイメールボックスに送る際に使用します。リクエストの送信が成功すると、処理中のメッセージカウンタを更新します。

・メールボックスキューテーブル (mb_req_mbx_t)

```
typedef struct _req_mbx{
    uint32_t    u32_soc_id;     /* ソケット ID */
    puint8_t    pu8_req_pkt;    /* パケットの格納先へのポインタ */
    uint32_t    u32_pkt_len;    /* パケットの長さ */
}mb_req_mbx_t, *p_mb_req_mbx_t;
```

Modbus_fetch_from_mailbox	メールボックスからのリクエスト取り出し

【書式】

```
uint32_t Modbus_fetch_from_mailbox(uint16_t u16_mbx_id,
                                   p_mb_req_mbx_t* pt_req_recvd);
```

【引数】		
uint16_t	u16_mbx_id	メールボックス ID
p_mb_req_mbx_t	pt_req_recvd	メールボックスキューテーブルへのポインタ
【戻り値】		
uint32_t	エラーコード	
【エラーコード】		
ERR_OK	正常終了	
ERR_MAILBOX	メールボックス読み取り失敗	

【解説】

本 API は、受信メールボックスまたはゲートウェイメールボックスに送られたメッセージを読み取る際に使用します。リクエストの送信が成功すると、処理中のメッセージカウンタを更新します。

Modbus_check_mailbox	メッセージボックス処理数チェック

【書式】

```
uint32_t Modbus_check_mailbox(uint16_t u16_mbx_id);
```

【引数】		
uint16_t	u16_mbx_id	メールボックス ID
【戻り値】		
uint32_t	処理中のメッセージ数またはエラーコード	
【エラーコード】		
ERR_TCP_SND_MBX_FULL	メッセージフル状態	

【解説】

本 API は、メールボックスで処理中のメッセージ数をチェックします。各メールボックスで処理できる最大数は以下のマクロで定義されており、処理中のメッセージ数が最大値に達している場合は、メッセージフル状態と判定します。

マクロ名	意味
MAX_RCV_MBX_SIZE	受信メールボックス最大値
MAX_GW_MBX_SIZE	ゲートウェイメールボックス最大値

Modbus_delete_mailbox	Modbus メールボックス削除

【書式】

uint32_t Modbus_delete_mailbox(uint16_t u16_mbx_id);

【引数】

uint16_t	u16_mbx_id	メールボックス ID

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_MAILBOX	メールボックスの削除に失敗

【解説】

本 API は、指定されたメールボックス ID のメールボックスを削除します。

10.2.3 ゲートウェイモード用 API

本章では、ゲートウェイタスクから呼び出される関数を記述します。

Modbus_gw_read_coils	Read coils ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_read_coils(puint8_t pu8_recvd_pkt,
                              uint32_t u32_recv_pkt_len,
                              p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスへの送受信に失敗
ERR_ILLEGAL_NUM_OF_COILS	読み出し数（number of coils）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Read coils を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用したテーブルのメモリを解放します。

・TCP パケット情報テーブル (mb_tcp_pkt_info_t)

```
typedef struct _mb_tcp_pkt_info{
    puint8_t    pu8_output_response;          /* 応答情報テーブルへのポインタ */
    uint16_t    u16_transaction_id;           /* トランザクション ID */
    uint16_t    u16_protocol_id;              /* プロトコル ID */
    uint8_t     u8_slave_id;                  /* スレーブ ID */
    uint16_t    u16_num_of_bytes;             /* PDU のバイト数 */
    uint8_t     aru8_data_packet[MAX_DATA_SIZE];  /* パケットデータ */
}mb_tcp_pkt_info_t, *p_mb_tcp_pkt_info_t;
```

Modbus_gw_read_discrete_inputs	Read discrete inputs ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_read_discrete_inputs(puint8_t pu8_recvd_pkt,
                                        uint32_t u32_recv_pkt_len,
                                        p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_INPUTS	読み出し数（number of inputs）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Read discrete inputs を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_read_holding_regs	Read holding registers ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_read_holding_regs(puint8_t pu8_recvd_pkt,
                                     uint32_t u32_recv_pkt_len,
                                     p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	読み出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Read holding registers を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_read_input_regs	Read input register ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_read_input_regs(puint8_t pu8_recvd_pkt,
                                   uint32_t u32_recv_pkt_len,
                                   p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	読み出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Read input registers を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_write_single_coil	Write single coil ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_write_single_coil(puint8_t pu8_recvd_pkt,
                                     uint32_t u32_recv_pkt_len,
                                     p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_OUTPUT_VALUE	出力指定値が不正
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Write single coil を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_write_single_reg	Write single register ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_write_single_reg(puint8_t pu8_recvd_pkt,
                                    uint32_t u32_recv_pkt_len,
                                    p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Write single register を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_write_multiple_coils	Write multiple coils ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_write_multiple_coils(puint8_t pu8_recvd_pkt,
                                        uint32_t u32_recv_pkt_len,
                                        p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_OUTPUTS	書き出し数（number of outputs）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Write multiple coils を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_write_multiple_reg	Write multiple registers ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_write_multiple_reg(puint8_t pu8_recvd_pkt,
                                      uint32_t u32_recv_pkt_len,
                                      p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_REG	書き出し数（number of registers）が仕様値に収まっていない
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Write multiple registers を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

Modbus_gw_read_write_multiple_reg	Read/Write multiple registers ゲートウェイ関数

【書式】

```
uint32_t Modbus_gw_read_write_multiple_reg(puint8_t pu8_recvd_pkt,
                                           uint32_t u32_recv_pkt_len,
                                           p_mb_tcp_pkt_info_t pt_gw_tcp_pkt_info);
```

【引数】

puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
uint32_t	u32_recv_pkt_len	受信パケットの長さ
p_mb_tcp_pkt_info_t	pt_gw_tcp_pkt_info	TCP パケット情報テーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_OUTPUT_VALUE	読み出し/書き込み数に異常がある
ERR_INVALID_SLAVE_ID	スレーブIDが有効でない
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SLAVE_ID_MISMATCH	受信した応答データのスレーブIDが不一致
ERR_CRC_CHECK	CRCチェックで異常（RTUモード）
ERR_LRC_CHECK	LRCチェックで異常（ASCIIモード）
ERR_FUN_CODE_MISMATCH	受信した応答データのファンクションコードが不一致
ERR_ILLEGAL_FUNCTION	当該ファンクションコードが有効になっていない
ERR_ILLEGAL_DATA_VALUE	指定データに異常がある
ERR_INSUFFICIENT_DATA	受信パケットの長さが異常

【解説】

本 API は、ゲートウェイモードでマスタデバイスからファンクションコード Read/Write multiple registers を受信した際にゲートウェイタスクから呼び出されます。関数内では、リクエストおよび応答に使用する構造体テーブルを動的に確保し、受信したパケットの情報で初期化します。次に各動作に対応するシリアル用マスタ API を呼び出し、その実行結果を TCP パケット情報テーブルに反映します。その後、リクエストおよび応答に使用した構造体のメモリを解放します。

10.3 エラーコード

エラーコードは、コマンド処理状況をユーザアプリケーションに知らせるために報告されます。要求/応答を処理中に発生したエラー別にコードが生成されます。以下に使用されるエラーコードを示します。

表 10.1　エラーコード一覧

エラーコード	値	エラー内容
ERR_OK	0x00	正常終了
ERR_ILLEGAL_FUNCTION	0x01	リクエストで受信したファンクションコードに、サーバ（またはスレーブ）で許可されていない、または、実装されていないファンクションコードが指定されている。例外コード(01)に該当しますので、値の変更をすることはできません。
ERR_ILLEGAL_DATA_ADDRESS	0x02	リクエストで受信したデータアドレスに、サーバ（またはスレーブ）で許可されていない、または、実装されていないアドレスが指定されている。例外コード(02)に該当しますので、値の変更をすることはできません。
ERR_ILLEGAL_DATA_VALUE	0x03	リクエストのデータフィールドで指定された値に、サーバ（またはスレーブ）で許可されない値が指定されている。例外コード(03)に該当しますので、値の変更をすることはできません。
ERR_SLAVE_DEVICE_FAILURE	0x04	サーバ（またはスレーブ）でリクエストを実行しようとして回復不能なエラーが発生した。例外コード(04)に該当しますので、値の変更をすることはできません。
ERR_STACK_INIT	0x05	スタックの初期化に失敗
ERR_ILLEGAL_SERV_BSY	0x06	デバイスが実行中でリクエストを受け付けられない。例外コード(06)に該当しますので、値の変更をすることはできません。
ERR_CRC_CHECK	0x07	CRC チェックに失敗
ERR_LRC_CHECK	0x08	LRC チェックに失敗
ERR_INVALID_SLAVE_ID	0x09	スレーブ ID が異常
ERR_TCP_SND_MBX_FULL	0x0A	メールボックスが満杯でメッセージが受け取れない
ERR_STACK_TERM	0x0B	スタック終了処理で異常
ERR_TIME_OUT	0x0C	タイムアウト発生
ERR_MEM_ALLOC	0x0D	メモリ確保に失敗
ERR_SYSTEM_INTERNAL	0x0E	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_COILS	0x0F	number of coils に仕様範囲外の値が指定された
ERR_ILLEGAL_NUM_OF_INPUTS	0x10	number of inputs に仕様範囲外の値が指定された
ERR_ILLEGAL_NUM_OF_REG	0x11	number of registers に仕様範囲外の値が指定された
ERR_ILLEGAL_OUTPUT_VALUE	0x12	レジスタ値の指定が異常
ERR_ILLEGAL_NUM_OF_OUTPUTS	0x13	number of outputs の指定が異常
ERR_INVALID_STACK_INIT_PARAMS	0x14	スタックの初期化で指定されたパラメータに異常
ERR_INVALID_STACK_MODE	0x15	指定されたスタックモードが異常
ERR_FUN_CODE_MISMATCH	0x16	要求を出したファンクションコード以外で応答パケットを受信した
ERR_SLAVE_ID_MISMATCH	0x17	要求を出したスレーブ以外から応答パケットを受信した
ERR_OK_WITH_NO_RESPONSE	0x18	正常終了（ブロードキャスト要求）
ERR_MSG_SIZE_OVER	0x19	受信した要求または応答データサイズが最大長を超えている

11. 実装方法

本章では、ソフトウェアの実装方法について説明します。

11.1 Modbus TCP

本章では、Modbus TCP スタックの実装について説明します。

11.1.1 サーバモード

サーバモードで使用する場合、以下の項目の設定を行います。

(1) タスク ID 定義

Modbus スタックがタスクとして使用する以下の API について、タスク ID を任意の値で割り当てます。

タスク API	タスク ID	機能
Modbus_tcp_soc_wait_task	ID_CONN_TASK	TCP 接続待ちタスク
Modbus_tcp_recv_data_task	ID_RECV_SOC	TCP 受信データタスク
Modbus_tcp_req_process_task	ID_SERV_TSK	TCP 要求処理タスク

(2) メールボックス ID 定義

Modbus スタックが使用する以下のメールボックス ID を設定します。

メールボックス ID	意味
ID_MB_TCP_RECV_MBX	TCP 受信メールボックス

(3) Modbus スタックの初期化

各種初期化を実行し、Modbus スタックを開始します。これらの初期化は TCP/IP スタックの初期化の後に行われる必要があります。

TCP サーバモードでは、各 API によって以下の処理を行う必要があります。

- 各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
- Modbus スタックの初期化および関連タスクの起動

各 API 仕様については、10.1.1.1 章を参照してください。以下に例を示します。

```
/* register callback functions */
st_slave_map.fp_function_code1  = cb_func_code01;  /* Read Coils operation */
st_slave_map.fp_function_code2  = cb_func_code02;  /* Read Discrete Inputs operation */
st_slave_map.fp_function_code3  = cb_func_code03;  /* Read Holding Registers operation */
st_slave_map.fp_function_code4  = cb_func_code04;  /* Read Input Registers operation */
st_slave_map.fp_function_code5  = cb_func_code05;  /* Write Single Coil operation */
st_slave_map.fp_function_code6  = cb_func_code06;  /* Write Single Register operation */
st_slave_map.fp_function_code15 = cb_func_code15;  /* Write Multiple Coils operation */
st_slave_map.fp_function_code16 = cb_func_code16;  /* Write Multiple Registers operation */
st_slave_map.fp_function_code23 = cb_func_code23;  /* Read/Write Multiple Registers operation */
Modbus_slave_map_init(&st_slave_map);

/* Initialize MODBUS stack by TCP server mode */
ercd = Modbus_tcp_init_stack(MODBUS_TCP_SERVER_MODE,
MODBUS_TCP_GW_SLAVE_DISABLE,
ENABLE_MULTIPLE_CLIENT_CONNECTION,
0,
NULL,
NULL);
```

(4) コールバック関数の実装

ファンクションコードに対応する機能を実装する場合、それらに対応するコールバック関数を実装します。

コールバック関数のインタフェース仕様については、10.1.1.1 章の Modbus_slave_map_init の項を参照してください。

11.1.2 ゲートウェイモード

ゲートウェイモードは、Modbus シリアルと Modbus TCP とそれぞれ通信を行う機能で構成されます。ゲートウェイモードで使用する場合、以下の項目を設定する必要があります。

(1) タスク ID 定義

Modbus スタックがタスクとして使用する以下の API について、タスク ID を任意の値で割り当てます。

タスク API	タスク ID	機能
Modbus_tcp_soc_wait_task	ID_CONN_TASK	TCP 接続待ちタスク
Modbus_tcp_recv_data_task	ID_RECV_SOC	TCP 受信データタスク
Modbus_tcp_req_process_task	ID_SERV_TSK	TCP 要求処理タスク
Modbus_gateway_task	ID_GATEWAY_TSK	TCP サーバ⇔シリアル通信間ゲートウェイタスク
Modbus_serial_recv_task	ID_MB_SERIAL_TSK	シリアルパケット受信タスク
Modbus_serial_task	ID_MB_SERIAL_RECV_TSK	シリアルタスク
Serial_recv_task	ID_SERIAL_TSK	シリアルデータ受信タスク

(2) イベントフラグ ID 定義

Modbus スタックが使用する以下のイベントフラグ ID を設定します。

イベントフラグ ID	意味
ID_FLG_SERIAL	タイマおよびシリアル I/F 割り込みイベント
ID_FLG_RESP_RDY	ブロッキングモードにおける応答イベント
ID_SERIAL_RESP	受信応答イベント

(3) メールボックス ID 定義

Modbus スタックが使用する以下のメールボックス ID を設定します。

メールボックス ID	意味
ID_MB_GATEWAY_MBX	ゲートウェイ処理受信メールボックス
ID_MB_TCP_RECV_MBX	TCP 受信メールボックス
ID_MB_SERIAL_MBX	シリアルイベントメールボックス

(4) Modbus スタックの初期化

TCP サーバモードと同様の初期化を行いますが、以下の項目が異なります。

- Modbus_slave_map_init()でコールバック関数の登録は行いません。(ただし、内部メモリ確保のため、API の呼び出しは行います。)
- Modbus_tcp_init_stack はゲートウェイモードで初期化し、シリアル通信用の設定を追加します。

各 API 仕様については、10.1.1.1 章を参照してください。以下に例を示します。

```
/* serial connection setting */
st_init_info.u32_baud_rate           = MODBUS_BAUDRATE;
st_init_info.u8_parity               = MODBUS_PARITY;
st_init_info.u8_stop_bit             = MODBUS_STOPBITS;
st_init_info.u8_uart_channel         = SCI_CH;
st_init_info.u32_response_timeout_ms   = 2000;
st_init_info.u32_turnaround_delay_ms   = 200;
st_init_info.u32_interframe_timeout_us = INTER_FRAME_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u32_interchar_timeout_us  = INTER_CHAR_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u8_retry_count          = 3;

/* register functions that performs RS485 direction control */
st_gpio_cfg.fp_gpio_init_ptr  = gpio_init;
st_gpio_cfg.fp_gpio_set_ptr   = gpio_set;
st_gpio_cfg.fp_gpio_reset_ptr = gpio_reset;

/* register callback functions(only memory allocation) */
Modbus_slave_map_init(&st_slave_map);

/* Initialize MODBUS stack by TCP server mode */
ercd = Modbus_tcp_init_stack(MODBUS_RTU_MASTER_MODE,
                             MODBUS_TCP_GW_SLAVE_ENABLE,
                             ENABLE_MULTIPLE_CLIENT_CONNECTION,
                             0,
                             &st_init_info,
                             &st_gpio_cfg);
```

11.2 Modbus RTU/ASCII

本章では、Modbus RTU/ASCII スタックの実装について説明します。

11.2.1 スレーブモード

スレーブモードで使用する場合、以下の設定が必要になります。

(1) タスク ID 定義

Modbus スタックがタスクとして使用する以下の API について、タスク ID を任意の値で割り当てます。

タスク API	タスク ID	機能
Modbus_serial_recv_task	ID_MB_SERIAL_TSK	シリアルパケット受信タスク
Modbus_serial_task	ID_MB_SERIAL_RECV_TSK	シリアルタスク
Serial_recv_task	ID_SERIAL_TSK	シリアルデータ受信タスク

(2) イベントフラグ ID 定義

Modbus スタックが使用する以下のイベントフラグ ID を設定します。

イベントフラグ ID	意味
ID_FLG_SERIAL	タイマおよびシリアル I/F 割り込みイベント
ID_FLG_RESP_RDY	ブロッキングモードにおける応答イベント
ID_SERIAL_RESP	受信応答イベント

(3) メールボックス ID 定義

Modbus スタックが使用する以下のメールボックス ID を設定します。

メールボックス ID	意味
ID_MB_SERIAL_MBX	シリアルイベントメールボックス

(4) Modbus スタックの初期化

各種初期化を実行し、Modbus スタックを開始します。シリアルスレーブモードでは、各 API によって以下の処理を行う必要があります。

- 各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。
- Modbus スタックの初期化および関連タスクの起動。この初期化にて、シリアル通信関連の設定を行いますが、それらはマスタ側の設定と一致させてください。

各 API 仕様については、10.1.2.1 章を参照してください。以下に RTU モードでの例を示します。

```
/* register callback functions */
st_slave_map.fp_function_code1  = cb_func_code01;
st_slave_map.fp_function_code2  = cb_func_code02;
st_slave_map.fp_function_code3  = cb_func_code03;
st_slave_map.fp_function_code4  = cb_func_code04;
st_slave_map.fp_function_code5  = cb_func_code05;
st_slave_map.fp_function_code6  = cb_func_code06;
st_slave_map.fp_function_code15 = cb_func_code15;
st_slave_map.fp_function_code16 = cb_func_code16;
st_slave_map.fp_function_code23 = cb_func_code23;
Modbus_slave_map_init(&st_slave_map);

/* serial connection setting */
st_init_info.u32_baud_rate           = MODBUS_BAUDRATE;
st_init_info.u8_parity               = MODBUS_PARITY;
st_init_info.u8_stop_bit             = MODBUS_STOPBITS;
st_init_info.u8_uart_channel         = SCI_CH;
st_init_info.u32_response_timeout_ms   = 2000;
st_init_info.u32_turnaround_delay_ms   = 200;
st_init_info.u32_interframe_timeout_us = INTER_FRAME_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u32_interchar_timeout_us  = INTER_CHAR_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u8_retry_count          = 3;

/* register functions that performs RS485 direction control */
st_gpio_cfg.fp_gpio_init_ptr  = gpio_init;
st_gpio_cfg.fp_gpio_set_ptr   = gpio_set;
st_gpio_cfg.fp_gpio_reset_ptr = gpio_reset;

/* Initialize MODBUS stack by Serial mode */
ercd = Modbus_serial_stack_init(&st_init_info,
                        &st_gpio_cfg,
                        MODBUS_RTU_SLAVE_MODE,
                        1);
```

ASCII モードで使用する場合、Modbus_serial_stack_init の引数を MODBUS_RTU_SLAVE_MODE から、MODBUS_ASCII_SLAVE_MODE に変更することで対応できます。

(5) コールバック関数の実装

ファンクションコードに対応する機能を実装する場合、それらに対応するコールバック関数を実装します。

コールバック関数のインタフェース仕様については、10.1.1.1 章の Modbus_slave_map_init の項を参照してください。

11.2.2 マスタモード

マスタモードでは、スレーブモードと同様の OS リソースを使用しますので 11.2.1 章を参照してください。以下、マスタモードで使用する際に必要な項目を示します。

(1) Modbus スタックの初期化

マスタモードでの初期化は、Modbus_serial_stack_init のみで行います。

各 API 仕様については、10.1.2.1 章を参照してください。以下に RTU モードでの例を示します。

```
/* serial connection setting */
st_init_info.u32_baud_rate            = MODBUS_BAUDRATE;
st_init_info.u8_parity                = MODBUS_PARITY;
st_init_info.u8_stop_bit              = MODBUS_STOPBITS;
st_init_info.u8_uart_channel          = SCI_CH;
st_init_info.u32_response_timeout_ms  = 2000;
st_init_info.u32_turnaround_delay_ms  = 200;
st_init_info.u32_interframe_timeout_us = INTER_FRAME_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u32_interchar_timeout_us  = INTER_CHAR_DELAY_TIMEOUT_US(MODBUS_BAUDRATE);
st_init_info.u8_retry_count           = 3;

/* register functions that performs RS485 direction control */
st_gpio_cfg.fp_gpio_init_ptr  = gpio_init;
st_gpio_cfg.fp_gpio_set_ptr   = gpio_set;
st_gpio_cfg.fp_gpio_reset_ptr = gpio_reset;

/* Initialize MODBUS stack by Serial mode */
ercd = Modbus_serial_stack_init(&st_init_info,
                        &st_gpio_cfg,
                        MODBUS_RTU_SLAVE_MODE,
                        1);
```

ASCII モードで使用する場合、Modbus_serial_stack_init の引数を MODBUS_RTU_MASTER_MODE から、MODBUS_ASCII_MASTER_MODE に変更することで対応できます。

12. 制限事項

μC3/Standard+μNet3-Professional RX700(RX72M)シリーズ e2studio 版 Release 2.0.0 に関する制限事項

- RX72M CPU カードの Ethernet 通信
  本サンプルは、Ethernet1 のドライバが未対応となります。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2023.01.31	-	初版発行

製品ご使用上の注意事項

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

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

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

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

www.renesas.com/contact/

商標について

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