User's Manual

R-IN32M3 シリーズ

ユーザーズ・マニュアル　Modbus スタック編

・R-IN32M3-EC
・R-IN32M3-CL

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

資料番号：R18UZ0028JJ0104
発行年月：2019.04.19

ルネサス エレクトロニクス
www.renesas.com

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）
   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等） を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等

8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。お客様が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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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その総株主の議決権の過半数を直接または間接に保有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。

製品ご使用上の注意事項

ここでは、CMOS デバイスの一般的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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレスのアクセス禁止
   【注意】リザーブアドレスのアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。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。
4. クロックについて
   【注意】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。

このマニュアルの使い方

1. 目的と対象者

このマニュアルはイーサネット通信 LSI「R-IN32M3 シリーズ」の機能を理解し、それを用いた応用設計をするユーザを対象とします。このマニュアルを使用するには、電気回路、論理回路マイクロコンピュータに関する基本的な知識が必要です。

本製品は、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています。

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。
本文中の★印は、本版で改訂された主な箇所を示しています。この"★"を PDF 上でコピーして「検索する文字列」に指定することによって、改版箇所を容易に検索できます

関連資料　関連資料は暫定版の場合がありますが、この資料では「暫定」の表示をしておりません。あらかじめご了承ください。また各コアの開発・企画段階で資料を作成しているため、関連資料は個別のお客様向け資料の場合があります。下記資料番号の末尾****部分は版数です。当社ホームページより最新版をダウンロードして参照ください。

R-IN32M3に関する資料

資料名	資料番号
R-IN32M3 シリーズ　データシート	R18DS0007JJ****
R-IN32M3 シリーズ　ユーザーズ・マニュアル R-IN32M3-EC	R18UZ0002JJ****
R-IN32M3 シリーズ　ユーザーズ・マニュアル R-IN32M3-CL	R18UZ0004JJ****
R-IN32M3 シリーズ　ユーザーズ・マニュアル 周辺機能編	R18UZ0006JJ****
R-IN32M3 シリーズ　プログラミング・マニュアル（ドライバ編）	R18UZ0008JJ****
R-IN32M3 シリーズ　プログラミング・マニュアル（OS 編）	R18UZ0010JJ****
R-IN32M3 シリーズ　ユーザーズ・マニュアル TCP/IP スタック編	R18UZ0018JJ****
R-IN32M3 シリーズ　ユーザーズ・マニュアル Modbus スタック編	このマニュアル

2. 数や記号の表記

データ表記の重み　　：左が上位桁、右が下位桁
アクティブ・ローの表記：
　　xxxZ（端子、信号名称のあとにZ ）
　　またはxxx_N （端子、信号名称のあとに_N ）
　　またはxxnx （端子、信号名称にnを含む）
注：
　　本文中につけた注の説明
注意：
　　気をつけて読んでいただきたい内容
備考：
　　本文の補足説明
数の表記：
　　2 進数 … xxxx , xxxxBまたはn’bxxxx（nビット）
　　10 進数 … xxxx
　　16 進数 … xxxxHまたはn’hxxxx（nビット）

2のべき数を示す接頭語（アドレス空間、メモリ容量）：
　　K （キロ）… $2^{10} = 1024$
　　M （メガ）… $2^{20} = 1024^2$
　　G （ギガ）… $2^{30} = 1024^3$
データ・タイプ　：
　　ワード … 32 ビット
　　ハーフワード … 16 ビット
　　バイト … 8 ビット

目次

1. 概説

本書は、R-IN32M3 シリーズ用 Modbus プロトコルスタックに関する資料です。本パッケージでは、イーサネット・ベースの Modbus TCP と、RS-485、RS-232C および RS-422 といったシリアル通信ベースの Modbus RTU、Modbus ASCII プロトコルに対応しています。

本書は R-IN32M3 Modbus プロトコルスタックを使ったアプリケーションを開発、実装する際に参照頂くことを意図したものであり、機能の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されています。

1.1 特徴

R-IN32M3 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。

- Modbus RTU スレーブ
- Modbus ASCII スレーブ
- Modbus RTU マスター
- Modbus ASCII マスター
- Modbus TCP サーバ
- Modbus TCP ゲートウェイ

サポートクラスおよびファンクションコードは以下のようです。

- Modbus コンフォーマンス・クラス 0、1、および 2 の一部
- サポートされるファンクションコード:
  - Read Coils (FC 1)
  - Read Discrete Inputs (FC 2)
  - Read Holding Registers (FC 3)
  - Read Input Registers (FC 4)
  - Write Single Coil (FC 5)
  - Write Single Register (FC 6)
  - Write Multiple Coils (FC 15)
  - Write Multiple Registers (FC 16)
  - Read/Write Multiple Registers (FC 23)

1.2　開発環境

本章では、Modbus プロトコルスタックの開発環境について記載します。

1.2.1　開発ツール

サンプルソフトは下記のツールチェーンで動作を確認しています。
また、サンプルソフトでは Arm® Cortex®マイクロコントローラ・ソフトウェア・インタフェース規格（CMSIS）の V2.10 を採用しています。詳細は CMSIS のドキュメントを参照してください。

表 1.1　ソフトウェア開発ツール一覧（ツールチェーン）

ツールチェイン	IDE	コンパイラ	デバッガ	ICE
IAR	Embedded Workbench for ARM (最新版をお使いください) (IAR Systems)			i-Jet JTAGjet-Trace-CM (IAR Systems)

1.2.2　評価ボード

Modbus スタックのサンプルアプリケーションは以下の R-IN32M3 用評価ボード上で動作します。各評価ボードの詳細については、弊社および IAR またはテセラ・テクノロジー株式会社の WEB サイトをご覧ください。

[サポートする評価ボード]

- Modbus TCP プロトコル
  - ・ TS-R-IN32M3-EC　：テセラ・テクノロジー株式会社
  - ・ TS-R-IN32M3-CL　：テセラ・テクノロジー株式会社
  - ・ TS-R-IN32M3-CEC　：テセラ・テクノロジー株式会社
  - ・ KSK-RIN32M3EC-LT-IL　：IAR 製キックスタートキット同梱品

- Modbus RTU/ASCII プロトコル
  - ・ TS-R-IN32M3-EC　：テセラ・テクノロジー株式会社
  - ・ TS-R-IN32M3-CL　：テセラ・テクノロジー株式会社
  - ・ TS-R-IN32M3-CEC　：テセラ・テクノロジー株式会社

注意　Modbus RTU/ASCII プロトコルを RS-485 通信で動作させる場合、RS485 トランシーバ IC またはモジュールを準備し、以下の関連信号と接続する必要があります。
P20 (RXD0) → TX,　P21 (TXD0) → RX,　RP17(GPIO) → DE(/RE)

Modbus RTU/ASCII 通信におけるハードウェア接続例を 7 章にて記載していますので、そちらもご参照ください。

1.3　リソース

- スタックはハードウェア RTOS の機能を使用しますので、スタックが利用可能である必要があります。
- スタックは、Modbus シリアル通信実行時にパケットタイミング計測用にタイマを 1 チャネル分使用します。
- スタックは、Modbus シリアル通信で UART を 1 チャネル分使用します。
- RS485 トランシーバ制御用に GPIO ピンがアサインされ、スタックが利用可能である必要があります。

1.4　ネットワーク

- Modbus シリアルスタックは、標準的な RS485 ネットワークを介して通信が可能です。
- Modbus TCP スタックは、標準的なイーサネットネットワークを介して通信が可能です。

1.5　スタック実行時の制約および注意点

- スタックは、任意の UART チャネル、タイマーチャネルおよび GPIO ピンをシリアルモード実行中に使用します。スタック実行中は、他のプログラムでこれらのリソースを使用することはできません。
- スタックは、ハードウェア RTOS のいくつかの機能を使用します。
- スタックがスレーブモードで動作している場合、ユーザにより適切な Modbus アプリケーションオブジェクトとのハンドシェークおよびデータ更新が行われる必要があります。
  - スレーブモードでは、ユーザは Modbus オブジェクトにアクセスするための関数を記述し、API 関数 Modbus_slave_map_init を用いて Modbus ファンクションコードにマップする必要があります。
  - 実装された関数において、ユーザは 2 つ以上のタスクから一度にメモリアクセスしないことを保証する必要があります。

2. R-IN32M3 用 Modbus スタックの基本概念

2.1 サポートプロトコル

本スタックは、シリアル通信上で動作する Modbus RTU および Modbus ASCII と、イーサネットネットワーク上で動作する Modbus TCP の各プロトコルをサポートします。ただし、ModbusTCP クライアントとしては動作しません。

これらの機能に対応して、本スタックは以下の 6 つのモードで動作します。

- Modbus RTU マススタック
- Modbus RTU スレーブスタック
- Modbus ASCII マススタック
- Modbus ASCII スレーブスタック
- Modbus TCP サーバスタック
- Modbus TCP サーバゲートウェイスタック

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。この初期化でサポートされる Modbus ファンクションコードも指定されます。本スタックでは、以下の 9 つのファンクションコードが実装可能です。

- 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

2.2　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、下図に示すような構造になっています。
2. 本スタックはハードウェアRTOSの機能を使用してタスクを作成します。スタックはRTOSを使用してマルチスレッドで動作します。
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。

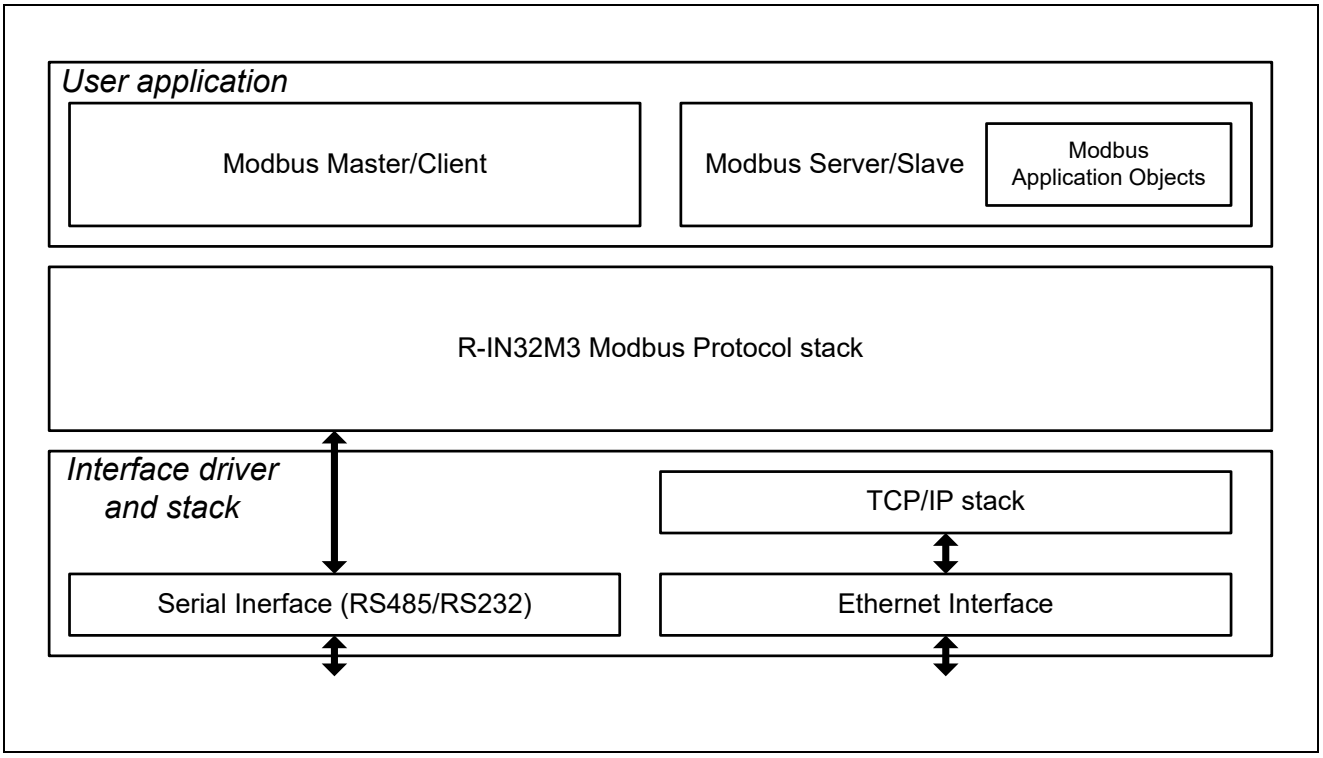


図 2.1　R-IN32M3 用 Modbus スタック概要 ★

3. システム構成 - Modbus シリアルプロトコルスタック

図 3.1 に Modbus プロトコルスタックの全体的な構成を示します。図が示すように、スタックは 4 つの機能層に分割されています。

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、マスターまたはサーバ/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。これらのモード選択などは初期設定 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。

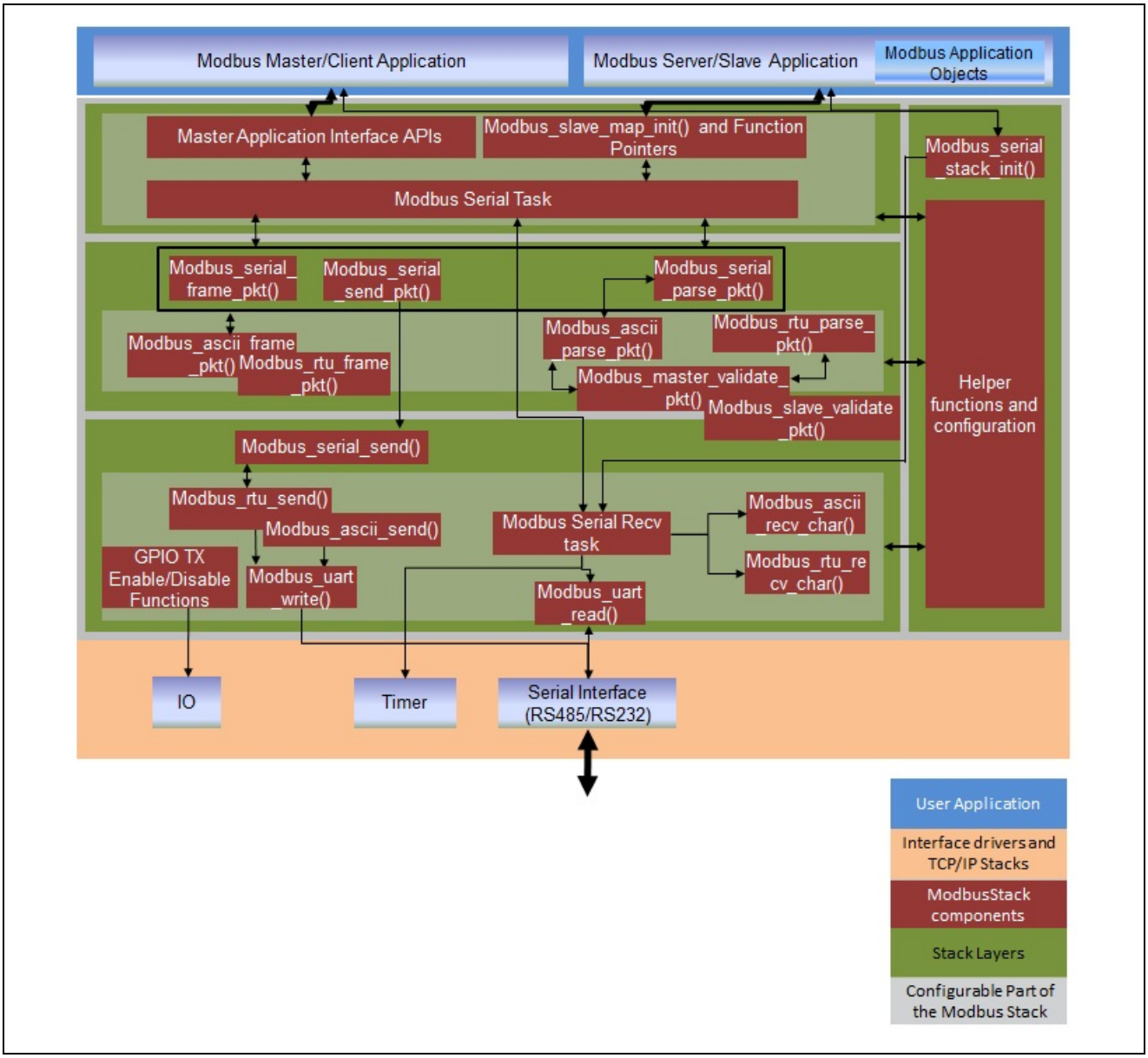


図 3.1 Modbus スタックソフトウェア構成 ★

以降のセクションで、各機能層について説明します。

3.1　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ーフェース層は、マスター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。

以上の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。

3.1.1　アプリケーションインターフェース層

アプリケーション･インターフェース層は、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。構成されたマスターまたはスレーブスタックモードに基づいて、スレッドはそのモードで動作可能な機能をユーザに提供します。この層は、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ーフェース層の主な構成要素は、シリアルタスクと Modbus_serial_slave_map_init()になります。Modbus_serial_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で行われます。シリアルタスクは、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ー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、その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 マスターモードにおけるアプリケーションインターフェース層の主な構成要素は、シリアルタスクとユーザアプリケーションとのインター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。シリアルタスクはその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。

3.1.1.1 Modbus シリアルタスク

シリアルタスクは、モード（RTU/ASCII）にかかわらず、Modbus マスター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 マスターとして定義されると、シリアルタスクは Modbus マスターのタスクとして機能します。マスター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 3.2 は、シリアルタスクがスレーブモード時の状態遷移図になります。Modbus_serial_stack_init() 関数がシリアルタスクを起動し、メールボックスにメッセージが送信されるのを待ちます。メッセージボックスで受信したメッセージの種類に応じて、マスターまたはスレーブとして機能します。タスクがスレーブとして機能するとき、Modbus マスターからの Modbus 要求を受信するまで、タスクの状態を保持します。

クライアントから要求を受信すると、タスク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、マスターデバイスへ送信します。

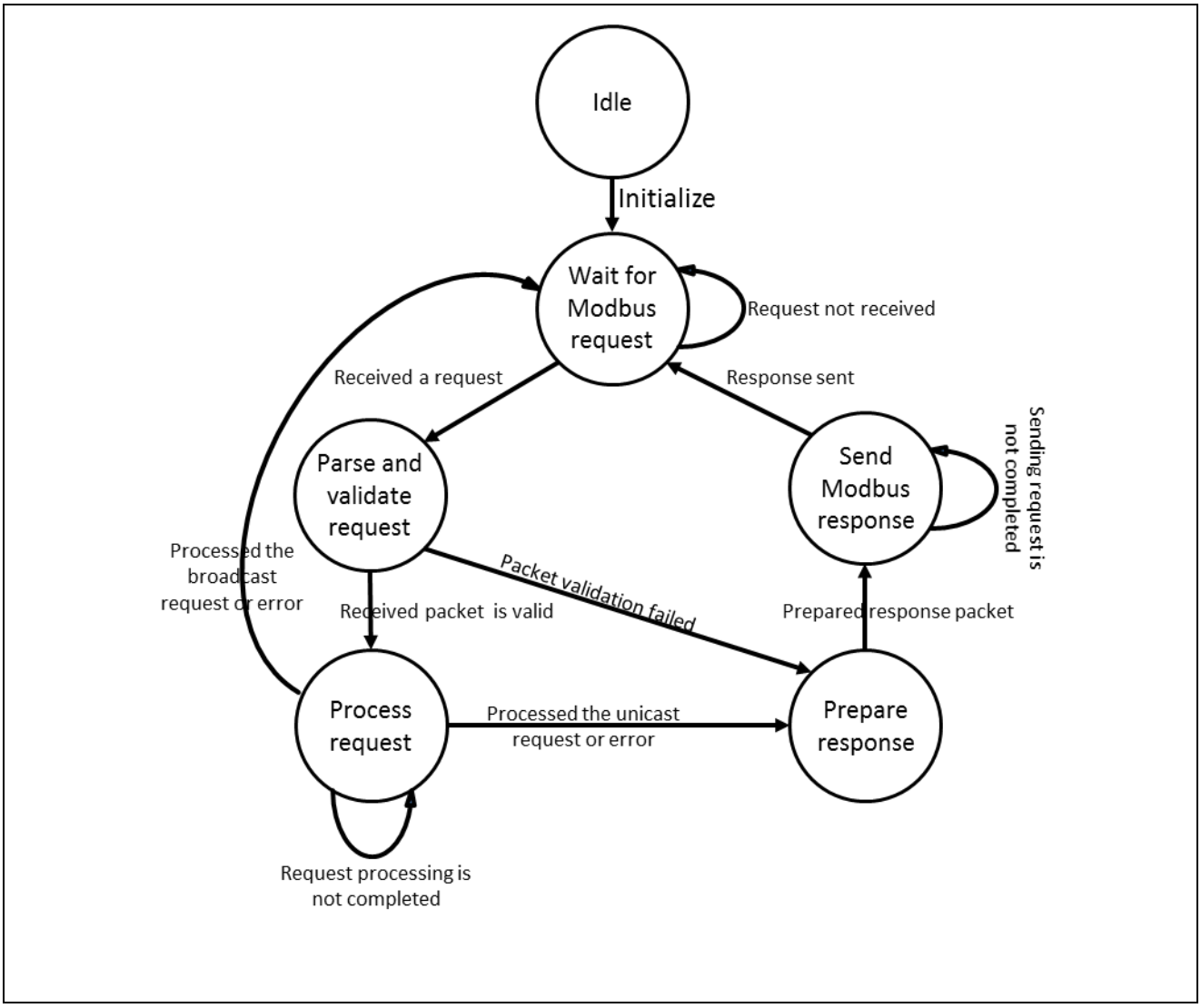


図 3.2 スレーブモード時の Modbus シリアルタスクの状態遷移図★

図 3.3 は、シリアルタスクがマスターモードの時の状態遷移図になります。

マスター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た際にシリアルタスクが起動され、メッセージ待ち状態になります。ユーザアプリケーションが、API を呼び出して、シリアルタスクにスレーブデバイスとの Modbus 通信を要求します。 タスクが要求を受信したら、以下を実施します。

- Modbus 要求パケットを準備し、それを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、タスクは「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、タスクが「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。
- タスクは、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。
- タスクが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、タスクは最大リトライ回数まで同じ要求を送信します。
- タスクは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。この場合、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、他タスク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。

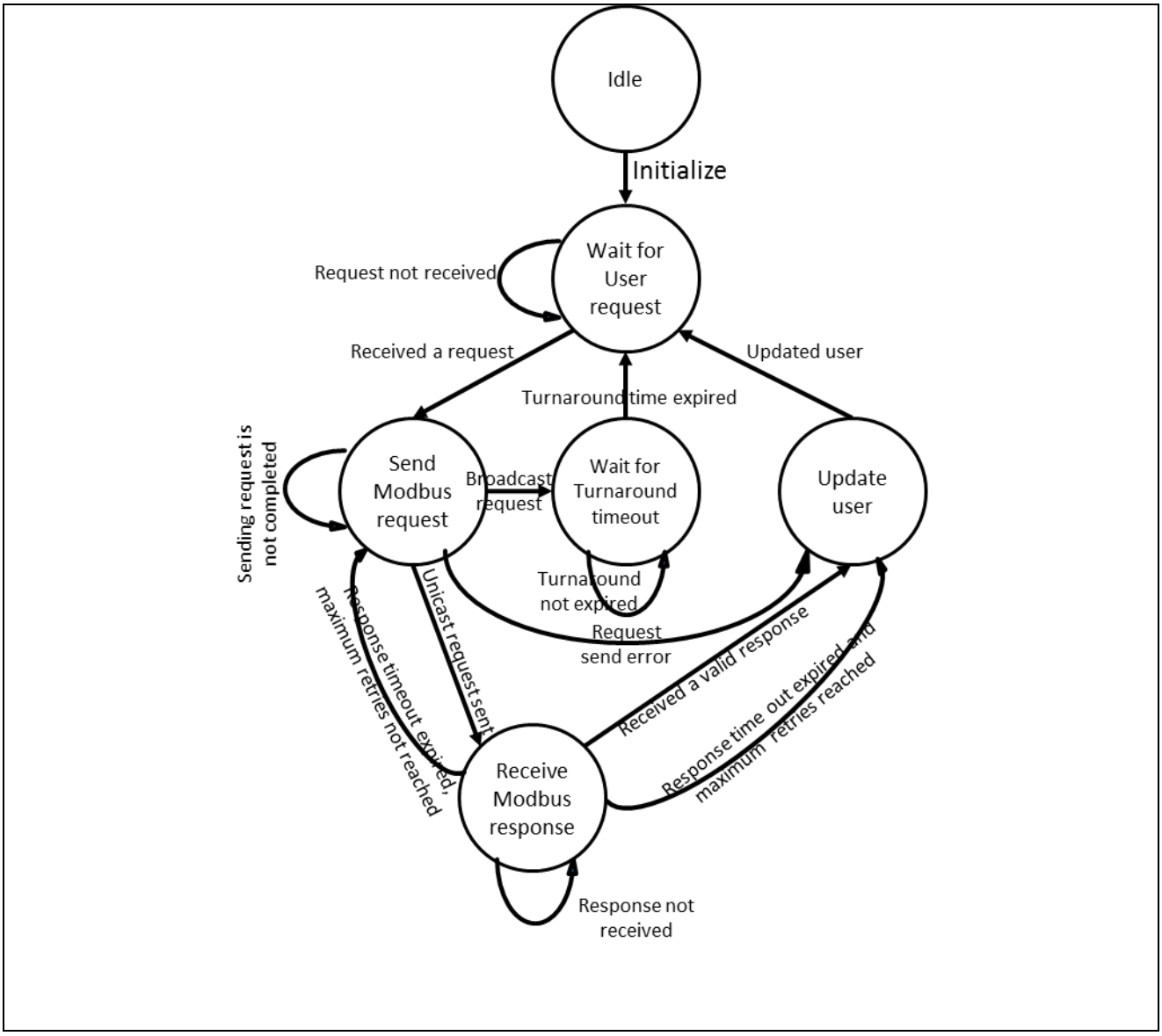


図 3.3 マスターモード時の Modbus シリアルタスクの状態遷移図 ★

3.1.1.2　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、シリアルタスクは例外応答を構築し、それをマスター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、指定されたパラメタの基本的なチェックを行い、その状態を返します。
- 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 マスター

- API 関数は、パラメタの基本的なチェックを行います。
- タスクは、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。
- パケット構築用にメモリを動的に確保します。メモリが確保できない場合はエラーを報告します。

3.1.2　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。

(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、それをマスター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ー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。

(2)　送信パケットの構築

マスター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。

3.1.2.1　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。

3.1.3　通信接続管理およびフレーム送受信層

このレイヤは、通信インター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、このレイヤで行われます。

3.1.3.1　シリアル受信タスク

シリアル受信タスクは、Modbus シリアルスタックが初期化されたときに起動されます。UART およびタイマ割り込みがハードウェア ISR でイベントとして登録されます。シリアル受信タスクは、ハードウェア ISR で定義されたパターンのイベントフラグがセットされるのを待ちます。

UART 割り込みイベントが発生すると、ドライバ関数 uart_read()で各バイトを読み込みます。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。

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

3.1.3.2　Modbus シリアルインターフェースコンフィグレーション

- このモードでは、スタックの初期化中に設けられた構成パラメタに従って、UART 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、「ユーザーズ・マニュアル 周辺機能編」を参照ください。
- 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。

3.1.3.3　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。

3.1.4　スタック設定および管理モジュール

本レイヤは、前記 3 レイヤに跨って使用されるソフトウェアモジュールになります。これには、各種マクロ定義、グローバル変数、データ構造体およびコンフィグレーション API などが含まれます。
詳細は以下の機能に分けられます。

3.1.4.1　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 3.1　エラーコードの説明

ERR_OK	正常終了
ERR_ILLEGAL_FUNCTION	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。このエラーは Exception code ‘1’に該当しますので、値を変更することはできません。
ERR_ILLEGAL_DATA_ADDRESS	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。このエラーは Exception code ‘2’に該当しますので、値を変更することはできません。
ERR_ILLEGAL_DATA_VALUE	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。このエラーは Exception code ‘3’に該当しますので、値を変更することはできません。
ERR_SLAVE_DEVICE_FAILURE	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。このエラーは Exception code ‘4’に該当しますので、値を変更することはできません。
ERR_STACK_INIT	スタックの初期化に失敗
ERR_ILLEGAL_SERV_BSY	デバイスが実行中でリクエストを受け付けられない。このエラーは Exception code ‘6’に該当しますので、値を変更することはできません。
ERR_CRC_CHECK	CRC チェックに失敗
ERR_LRC_CHECK	LRC チェックに失敗
ERR_INVALID_SLAVE_ID	スレーブ ID が異常
ERR_TCP_SND_MBX_FULL	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	スタック終了処理で異常
ERR_TIME_OUT	タイムアウト発生
ERR_MEM_ALLOC	メモリ確保に失敗
ERR_SYSTEM_INTERNAL	メールボックスの送受信に失敗
ERR_ILLEGAL_NUM_OF_COILS	number of coils に仕様範囲外の値が指定された
ERR_ILLEGAL_NUM_OF_INPUTS	number of inputs に仕様範囲外の値が指定された
ERR_ILLEGAL_NUM_OF_REG	number of registers に仕様範囲外の値が指定された
ERR_ILLEGAL_OUTPUT_VALUE	レジスタ値の指定が異常
ERR_ILLEGAL_NUM_OF_OUTPUTS	number of outputs の指定が異常
ERR_INVALID_STACK_INIT_PARAMS	スタックの初期化で指定されたパラメタに異常
ERR_INVALID_STACK_MODE	指定されたスタックモードが異常
ERR_FUN_CODE_MISMATCH	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	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	正常終了（ブロードキャスト要求）

3.1.4.2　スタックモード選択

用意されている 6 つのスタックモードから 1 つを選択して開発を行います。TCP サーバゲートウェイの場合、TCP ゲートウェイデバイスと接続するシリアルデバイスのスタックモードを MODBUS_RTU_MASTER_MODE または MODBUS_ASCII_MASTER_MODE で選択します。

3.1.4.3　ファンクションコード選択

Modbus スタックは、クライアント側からリクエストを受信したとき、ユーザが登録した関数を呼び出します。

その際、ファンクションコードに対応する関数が登録されていない（実際には登録テーブルの関数ポインタに NULL が設定されている）場合、そのファンクションコードはそのアプリケーション/デバイスでサポートされていないことを意味します。

3.1.4.4　エラー識別および報告

コンフィグレーションパラメタの検証は参照される API 関数内で行われます。指定にエラーがあった場合、当該 API よりエラーが報告されます。

4.　システム構成 – Modbus TCP プロトコルスタック

この章では、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。図 4.1 にスタックの概要を示します。図が示すように、TCP サーバスタックおよびゲートウェイ機能付き TCP サーバスタックとして使用することができます。またゲートウェイスタックのみとしても使用することもできます。

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の Modbus RTU/ ASCII マススタックを使用することになります。Modbus RTU/ASCII マス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。

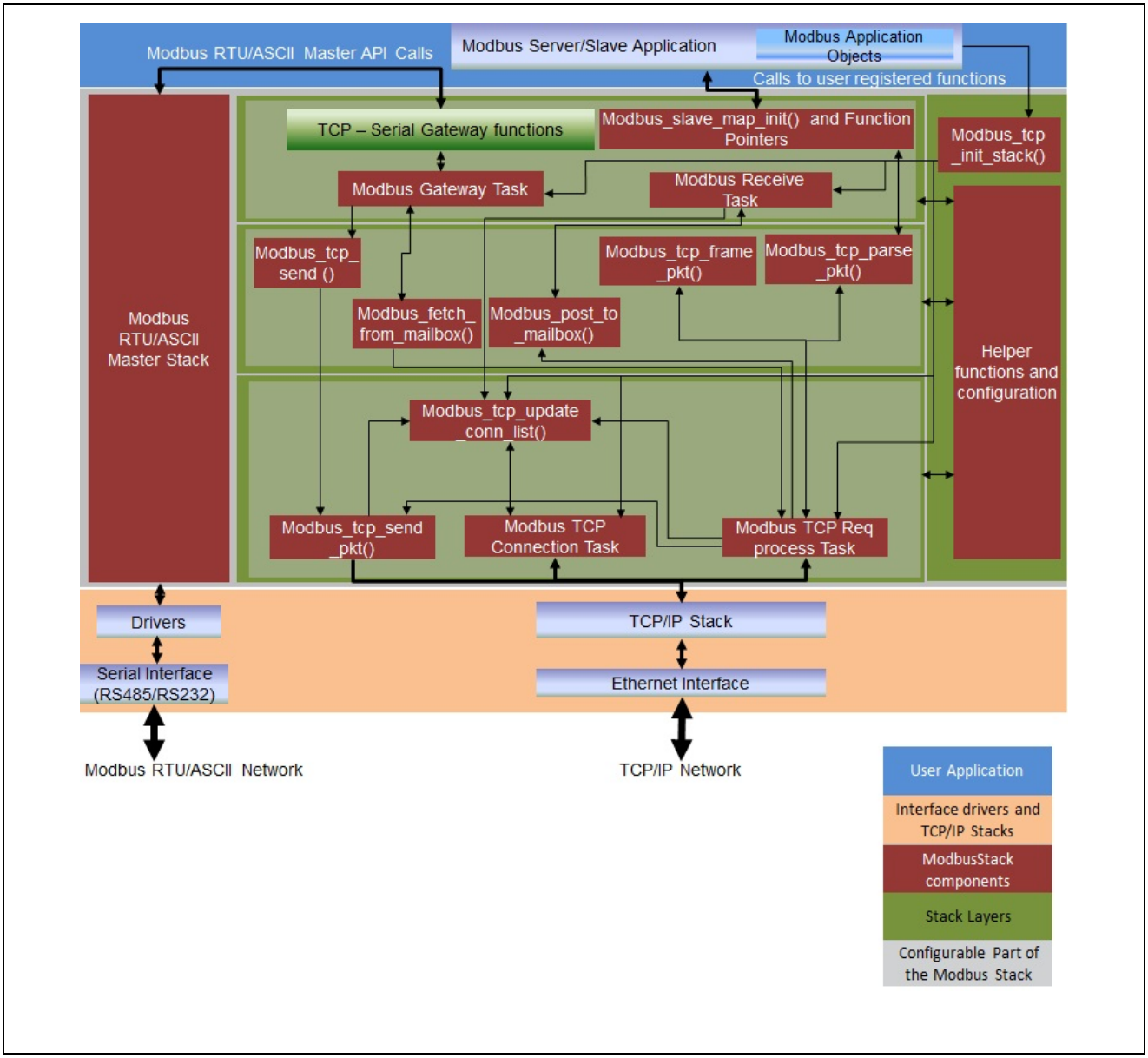


図 4.1 Modbus TCP およびゲートウェイソフトウェア構成 ★

図 4.1 のように、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。

最上位層のアプリケーションインターフェース層は、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フレーム送受信層は、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いて、この層のすべての関数は、この層自体のタスクで実行されます。

コンフィギュレーション層は上記３層に跨って呼び出されるレイヤであり、構成用 API と共に必要な機能が含まれています。

4.1　モジュール構成

4.1.1　アプリケーションインターフェース層

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。

4.1.1.1　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、このタスクはそれをコピーし処理します。ゲートウェイ機能の有無で少し動作が異なります。

ゲートウェイ機能なしの場合、このタスクはスレーブ 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にその要求パケットを転送します。

図 4.2 と図 4.3 に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このタスクの状態遷移図をそれぞれ示します。

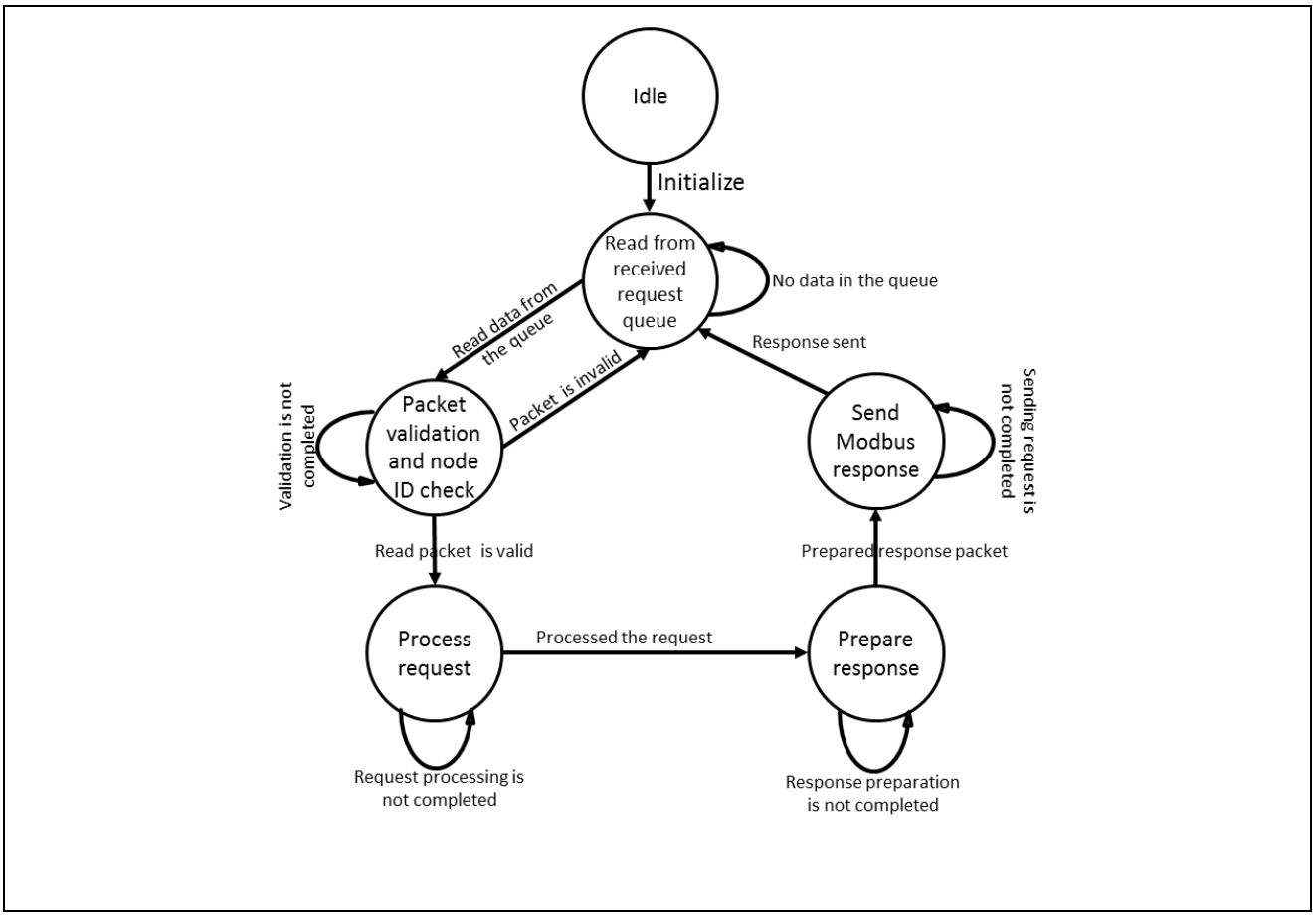


図 4.2 Modbus TCP サーバタスク （ゲートウェイ機能なし） ★

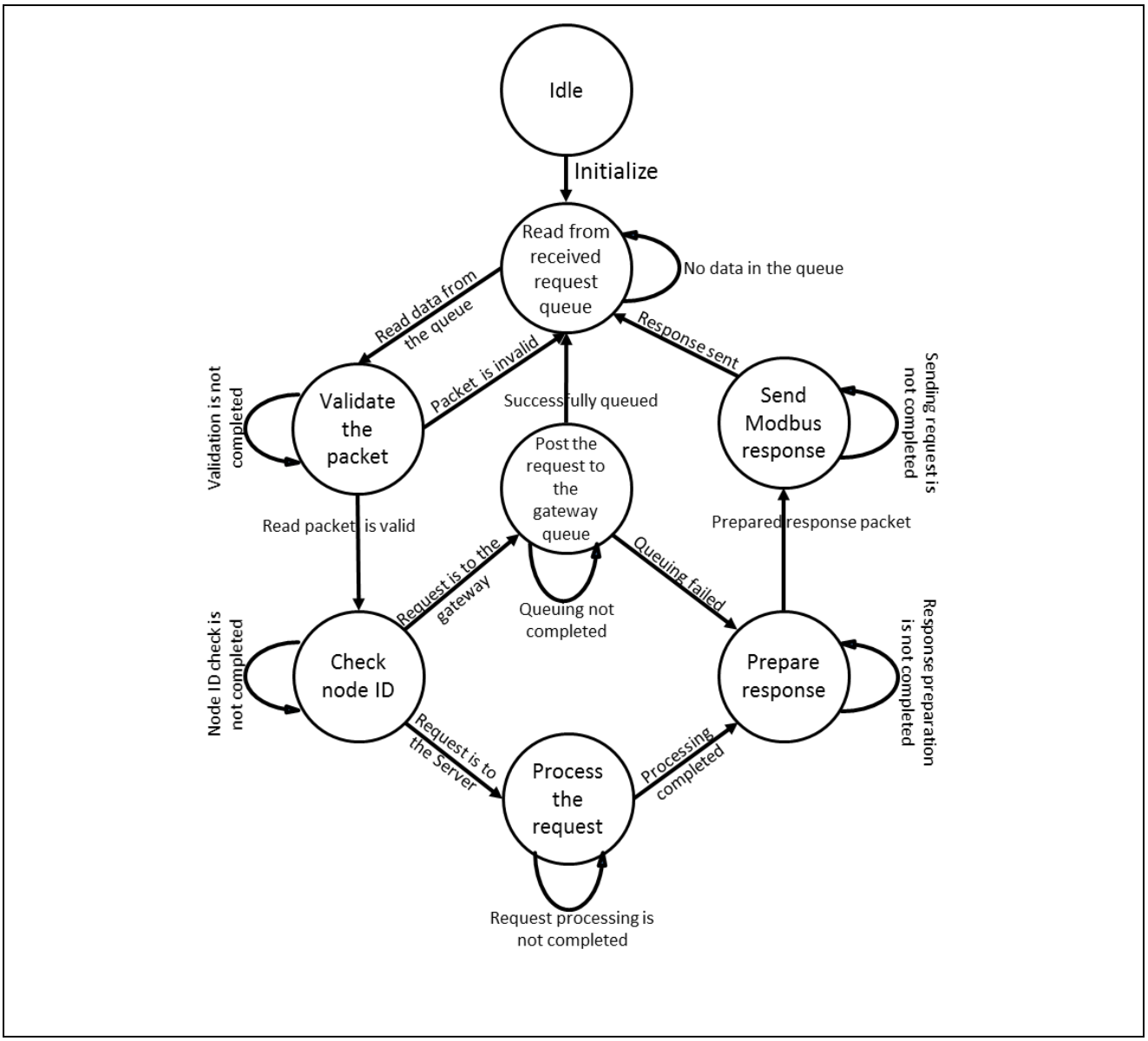


図 4.3　　Modbus TCP サーバタスク　（ゲートウェイ機能あり）　★

4.1.1.2　Modbus TCP – シリアルゲートウェイタスク

このタスクは Modbus RTU/ASCII マスター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。図 4.4 にこのタスクの主な状態を示します。図に示すように、このタスクは、クライアントからのスレーブ ID が 0xFF 以外の要求を受信したとき、その要求が TCP サーバタスクからメールボックスに転送されるのを待ちます。メールボックスに要求を受信すると、それを検証し、Modbus ゲートウェイ関数を呼び出して、RTU/ASCII マスタースタックにそれを送信します。このタスクは、受信したパケットのファンクションコードに基づいて、ゲートウェイ関数を呼び出し、マスタータスクから応答が返ってきたとき、TCP 接続に対して応答を送り返します。一方、RTU/ASCII マスタータスクは、シリアルネットワーク上のスレーブデバイスと通信し、応答を取得します。

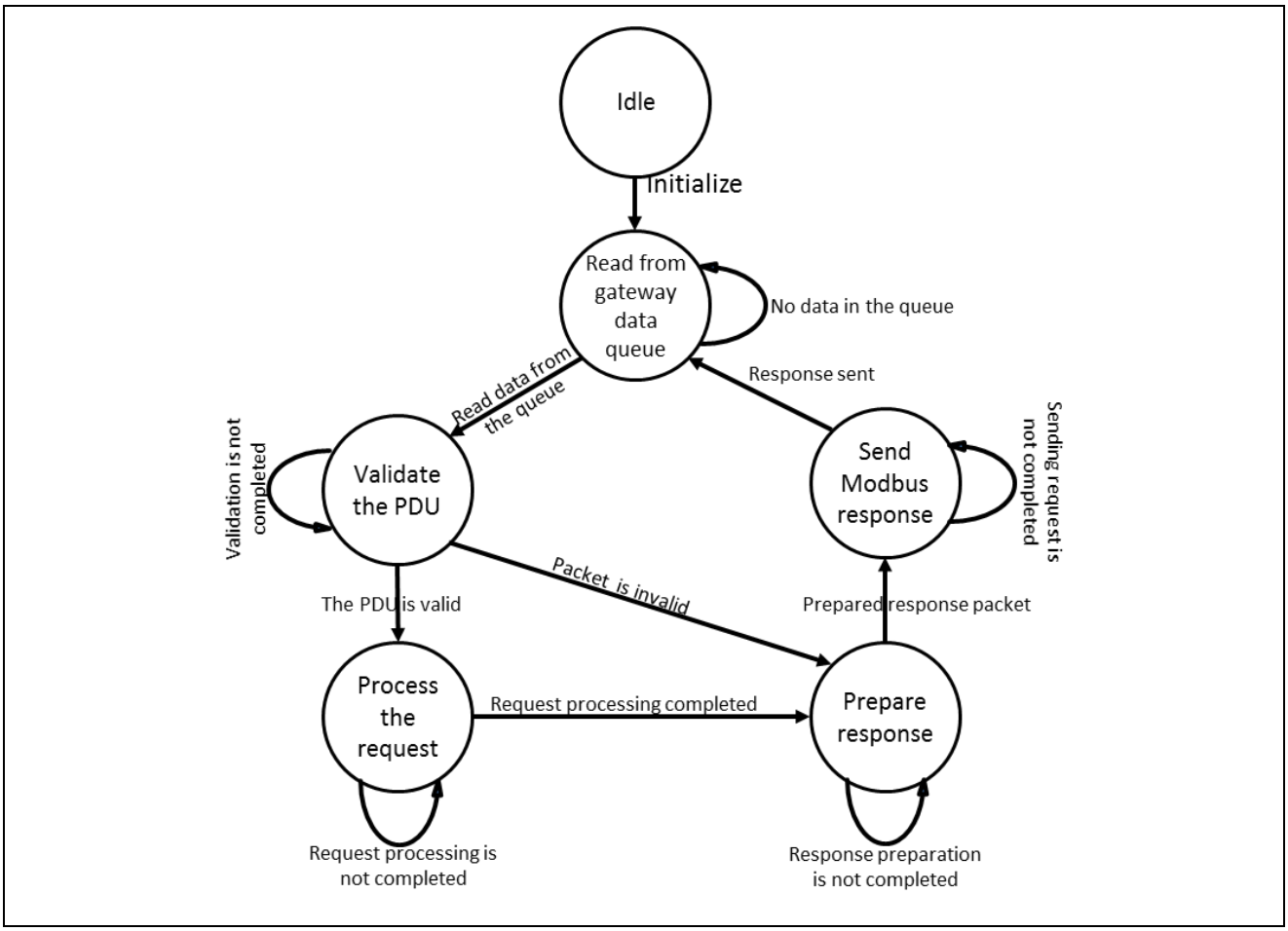


図 4.4　MODBUS TCP-シリアルゲートウェイタスク ★

4.1.1.3　エラー判定および報告

- 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 サーバタスクは要求パケットに対して Exception code 6(Server Busy)を応答パケットとして返信します。

4.1.2　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。

(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。

(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。

4.1.2.2　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値がプロトコルに適合しているかを検証します。
- 受信パケット解析では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。

4.1.3　通信接続管理およびパケット送受信層

この層は、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。

4.1.3.1　Modbus TCP 接続受け入れタスク

この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。図 4.5 にこのタスクの状態遷移図を示します。接続の最大数は、MAXIMUM_NUMBER_OF_CLIENTS に制限されます。

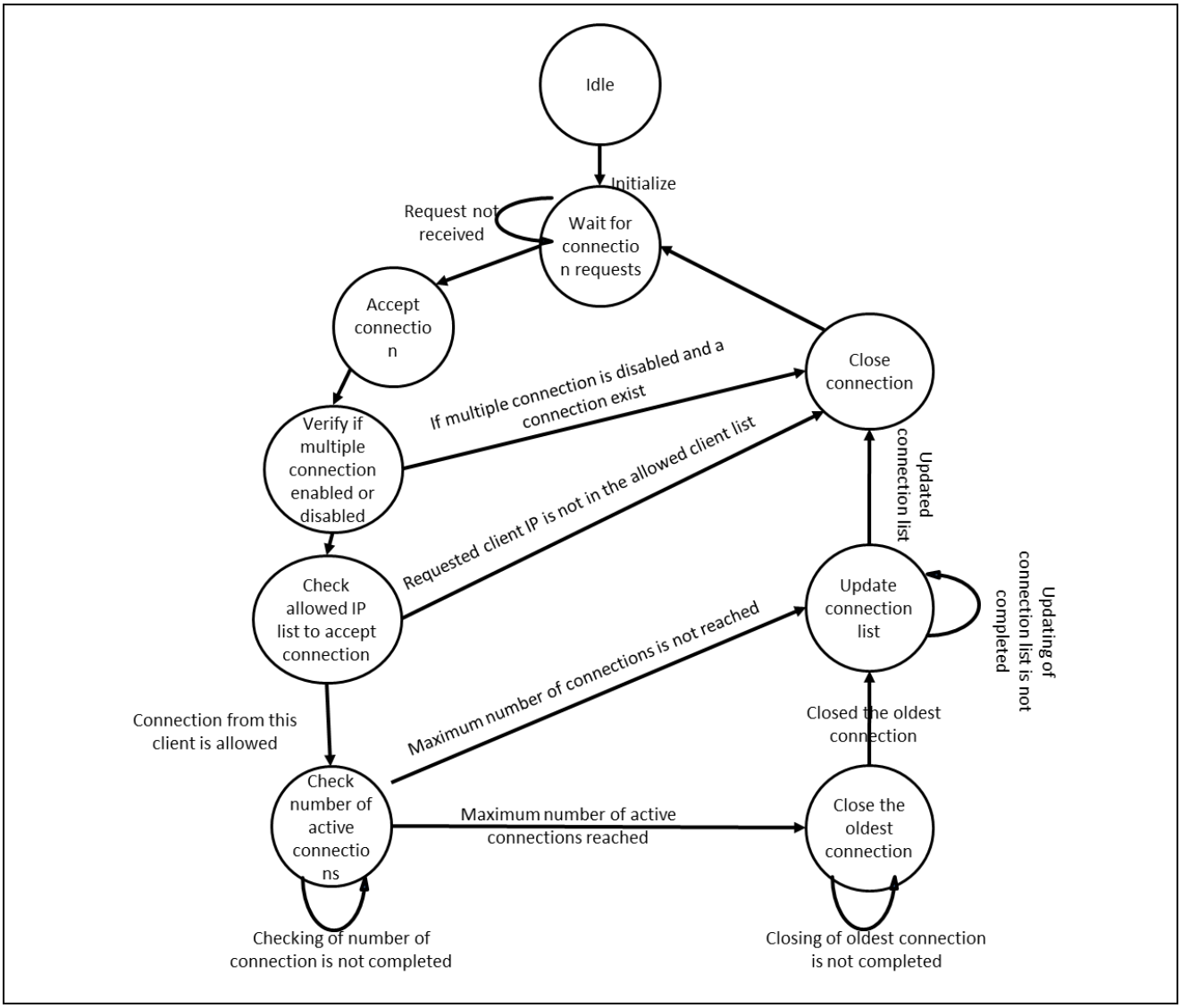


図 4.5 Modbus TCP 接続受け入れタスク ★

4.1.3.2　Modbus TCP 受信データタスク

この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。図 4.6 にこのタスクの状態遷移図を示します。

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。この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。

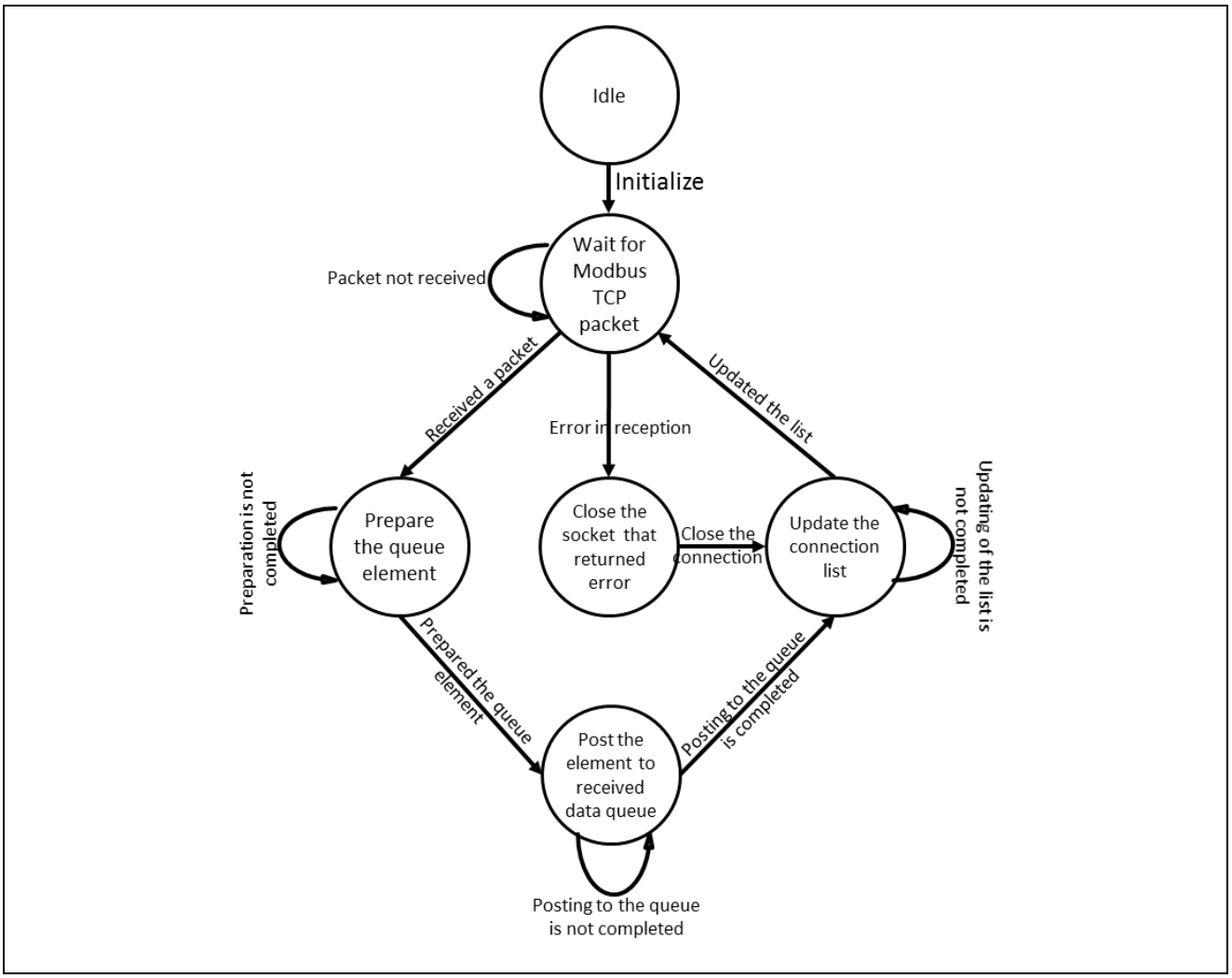


図 4.6 Modbus TCP 受信データタスク ★

4.1.3.3　エラー判定および報告

- 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 Exception code 6(Server Busy)を応答パケットとして返信します。

5.　アプリケーションプログラミングインターフェースの説明

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。

5.1　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。

5.1.1　Modbus TCP/IP

5.1.1.1　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タで呼び出します。

本 API では、以下のパラメタを指定します。

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 マスターモードを選択
MODBUS_RTU_SLAVE_MODE	Modbus Stack RTU スレーブモードを選択
MODBUS_ASCII_MASTER_MODE	Modbus Stack ASCII マスターモードを選択
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ーチャネル  */
    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タとして使用されます。

ボーレート指定	意味
UART_BAUD_9600	9600bps
UART_BAUD_19200	19200bps
UART_BAUD_31250	31250bps
UART_BAUD_38400	38400bps
UART_BAUD_76800	76800bps
UART_BAUD_115200	115200bps
UART_BAUD_153600	153600bps

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

UART チャネル指定	意味
UART_CHANNEL_0	チャネル０を使用
UART_CHANNEL_1	チャネル１を使用

タイマーチャネル指定	意味
TIMER_CHANNEL_0	チャネル０を使用
TIMER_CHANNEL_1	チャネル１を使用

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 p_serial_slave_map_init_t);

【引数】

p_slave_map_init_t	p_serial_slave_map_init_t	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、5.1.2.2 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 descrete 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 descrete 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 descrete 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

5.1.1.2　IP アドレス管理

IP アドレス管理に使用される 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
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。

5.1.1.3 タスク

以下の関数は、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_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_terminate_gateway_stack()を呼び出します。

5.1.2　Modbus シリアル

5.1.2.1　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 マスターモードを選択
MODBUS_RTU_SLAVE_MODE	Modbus Stack RTU スレーブモードを選択
MODBUS_ASCII_MASTER_MODE	Modbus Stack ASCII マスターモードを選択
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。

パラメタの詳細は、5.1.1.1 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 p_tcp_slave_map_init_t);

【引数】

p_slave_map_init_t	p_tcp_slave_map_init_t	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、5.1.1.1 章を参照してください。

5.1.2.2　マスターモード API

以下の API はマスターモードで使用されます。

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 descrete 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_regs;        /* 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_regs;        /* 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ー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。

5.1.2.3　タスク

以下の関数は、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ー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ーモードの場合、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ます。送信後、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ー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ー側に送信します。

Modbus_serial_recv_task	Modbus シリアルデータ受信タスク

【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、UART からのデータ受信に使用されます。ハードウェア ISR で登録した各割り込みイベントを監視し、発生したイベントに応じて処理を呼び出します。

UART 受信割り込みを検出すると、UART 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。

UART 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、「ユーザーズ・マニュアル 周辺機能編」を参照ください。

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。

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。

5.2　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。

5.2.1　パケット構築および解析 API

5.2.1.1　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、マスターモードの場合は 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、UART 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）

【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）

【解説】

本 API は、UART 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、マスターモードの場合は 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）

【解説】

本 API は、UART 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、UART 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、マスターモードで UART 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 UART 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　　マスター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、スタックがマスター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_uart_write	Modbus UART 書き込み

【書式】

```
void_t Modbus_uart_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 UART に書き込みます。ドライバ関数 uart_write()を使用し、MB_UART_CHANNEL 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_uart_read	Modbus UART 読み込み

【書式】

```
uint32_t Modbus_uart_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、UART チャネルから 1 バイトデータを読み込みます。ドライバ関数 uart_read()を使用し、MB_UART_CHANNEL 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。

5.2.1.2　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_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_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。

5.2.2　スタックコンフィグレーションおよび管理 API

5.2.2.1　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。

5.2.2.2　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

5.2.2.3　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。

5.2.2.4　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_create_mailbox	Modbus メールボックス生成

【書式】

```
uint32_t Modbus_create_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	メールボックス生成に失敗

【解説】

本 API は、タスク間のメッセージのやり取りに使用するメールボックスを生成します。

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。

5.2.3　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ー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ー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ーデバイスからファンクションコード Read descrete 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ー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ー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ー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ー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ー 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ー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ー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】

[in]	puint8_t	pu8_recvd_pkt	受信パケットを格納した領域へのポインタ
[in]	uint32_t	u32_recv_pkt_len	受信パケットの長さ
[out]	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ー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ー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ー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ー 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ー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ー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ー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ー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。

6. 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。

6.1 Modbus TCP

本章では、Modbus TCP スタックの実装について説明します。Modbus TCP スタックを実装する場合、TCP/IP プロトコルスタックも実装する必要があります。

TCP/IP プロトコルスタックの実装方法については、「プログラミングマニュアル TCP/IP 編」を参照してください。

6.1.1 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	機能
Modbus_tcp_soc_wait_task	TCP 接続待ちタスク
Modbus_tcp_recv_data_task	TCP 受信データタスク
Modbus_tcp_req_process_task	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) タスク生成

Modbus スタックで使用するタスクを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const TSK_TBL static_task_table[] = {
// CRE_TSK( tskid,          {tskatr,     exinf,  task,                          itskpri,  stksz,   stk});
    …
      {ID_CONN_TASK,   {TA_HLNG,   0,   (FP)Modbus_tcp_soc_wait_task,   8,   0x400,   NULL}},
      {ID_RECV_SOC,    {TA_HLNG,   0,   (FP)Modbus_tcp_recv_data_task,  8,   0x400,   NULL}},
      {ID_SERV_TSK,    {TA_HLNG,   0,   (FP)Modbus_tcp_req_process_task, 8,  0x400,   NULL}},
    …
};
```

(4)　メールボックス生成

Modbus スタックで使用するメールボックスを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const MBX_TBL static_mailbox_table[] = {
// CRE_MBX(   mbxid,                    {mbxatr,        maxmpri,   mprihd});
    …
               {ID_MB_TCP_RECV_MBX,      {TA_TFIFO,      0,         NULL}},
    …
};
```

(5)　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。

- IP アドレス登録
- 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 の記述方法については、5.1.1.1 章および 5.1.1.2 章を参照してください。以下に例を示します。

```
/* Enable IP table */
Modbus_tcp_init_ip_table(ENABLE, REJECT);

/* register IP address */
ercd = Modbus_tcp_add_ip_addr("192.168.1.100");
if (ercd != ERR_OK){
        return ercd;
}

/* 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);

/* Initialize MODBUS stack by TCP server mode */
ercd = Modbus_tcp_init_stack(MODBUS_TCP_SERVER_MODE,
                        MODBUS_TCP_GW_SLAVE_ENABLE,
                        ENABLE_MULTIPLE_CLIENT_CONNECTION,
                        0,
                        NULL,
                        NULL);
if (ercd != ERR_OK){
        return ERR_OK;
}
```

(6)　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ーフェース仕様については、5.1.1.1 章の Modbus_slave_map_init の項を参照してください。

6.1.2　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	機能
Modbus_gateway_task	TCP サーバ⇔シリアル通信間ゲートウェイタスク
Modbus_tcp_soc_wait_task	TCP 接続待ちタスク
Modbus_tcp_recv_data_task	TCP 受信データタスク
Modbus_tcp_req_process_task	TCP 要求処理タスク
Modbus_serial_recv_task	シリアル受信データタスク
Modbus_serial_task	シリアル要求処理タスク

(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	意味
ID_FLG_SERIAL	タイマおよび UART 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)　タスク生成

Modbus スタックで使用するタスクを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const TSK_TBL static_task_table[] = {
// CRE_TSK( tskid,                          {tskatr,      exinf,   task,                                itskpri,    stksz,      stk});
        …
            {ID_CONN_TASK,                  {TA_HLNG,   0,   (FP)Modbus_tcp_soc_wait_task,        8,      0x400,   NULL}},
            {ID_RECV_SOC,                   {TA_HLNG,   0,   (FP)Modbus_tcp_recv_data_task,       8,      0x400,   NULL}},
            {ID_SERV_TSK,                   {TA_HLNG,   0,   (FP)Modbus_tcp_req_process_task,     8,      0x400,   NULL}},
            {ID_GATEWAY_TSK,                {TA_HLNG,   0,   (FP)Modbus_gateway_task,             8,      0x400,   NULL}},
            {ID_MB_SERIAL_RECV_TSK,         {TA_HLNG,   0,   (FP)Modbus_serial_recv_task,         8,      0x400,   NULL}},
            {ID_MB_SERIAL_TSK,              {TA_HLNG,   0,   (FP)Modbus_serial_task,              8,      0x400,   NULL}},
        …
};
```

(5)　イベントフラグ生成

Modbus スタックで使用するイベントフラグを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const FLG_TBL static_eventflag_table[] = {
// CRE_FLG(     flgid,                  {flgatr,            iflgptn});
    …
        {ID_FLG_SERIAL,             {TA_TFIFO,      0}},
        {ID_FLG_RESP_RDY,           {TA_TFIFO,      0}},
        {ID_SERIAL_RESP,            {TA_TFIFO,      0}},
    …
};
```

(6)　メールボックス生成

Modbus スタックで使用するメールボックスを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const MBX_TBL static_mailbox_table[] = {
// CRE_MBX(   mbxid,                   {mbxatr,            maxmpri,    mprihd});
    …
        {ID_MB_SERIAL_MBX,      {TA_TFIFO,          0,          NULL}},
        {ID_MB_GATEWAY_MBX,     {TA_TFIFO,          0,          NULL}},
        {ID_MB_TCP_RECV_MBX,    {TA_TFIFO,          0,          NULL}},
    …
};
```

(7)　ハードウェア ISR 登録

Modbus スタックで使用するハードウェア ISR の登録を行います。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const HWISR_TBL static_hwisr_table[] = {
//    inhno,              hwisr_syscall,      id,                 setptn
        …
      {UAJ0TIR_IRQn,      HWISR_SET_FLG,    ID_FLG_SERIAL,      RECV_FLG},
      {TAUJ2I1_IRQn,      HWISR_SET_FLG,    ID_FLG_SERIAL,      TIMER_FLG},
      {UAJ0TIS_IRQn,      HWISR_SET_FLG,    ID_FLG_SERIAL,      UART_STS_FLG},
        …
};
```

上記の例は、UART チャネル 0 とタイマーチャネル 1 を設定した場合の例になります。他のチャネルを使用する場合は、適宜変更してください。

(8)　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 の記述方法の詳細については、5.1.1.1 章 および 5.1.1.2 章を参照してください。以下に例を示します。

```
/* Enable IP table */
Modbus_tcp_init_ip_table(ENABLE, REJECT);

/* register IP address */
ercd = Modbus_tcp_add_ip_addr("192.168.1.100");
if (ercd != ERR_OK){
        return ercd;
}

/* serial connection setting */
st_init_info.u32_baud_rate                  = BAUD_38400;
st_init_info.u8_parity                      = PARITY_NONE;
st_init_info.u8_stop_bit                    = STOP_BIT_ONE;
st_init_info.u8_uart_channel                = UART_CHANNEL_ZERO;
st_init_info.u8_timer_channel               = TIMER_CHANNEL_ONE;
st_init_info.u32_response_timeout_ms    = 1000;
st_init_info.u32_turnaround_delay_ms    = 200;
st_init_info.u32_interframe_timeout_us = 1750;
st_init_info.u32_interchar_timeout_us   = 750;
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 gateway 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);
if (ercd != ERR_OK){
        return ERR_OK;
}
```

6.2 Modbus RTU/ASCII

本章は、Modbus RTU/ASCII スタックについて説明します。

6.2.1 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	機能
Modbus_serial_recv_task	シリアル受信データタスク
Modbus_serial_task	シリアル要求処理タスク

(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	意味
ID_FLG_SERIAL	タイマおよび UART 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	受信要求イベント

(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	シリアル通信イベントメールボックス

(4) タスク生成

Modbus スタックで使用するタスクを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const TSK_TBL static_task_table[] = {
// CRE_TSK( tskid,                  {tskatr,     exinf,     task,                          itskpri,    stksz,   stk});
        …
    {ID_MB_SERIAL_RECV_TSK,   {TA_HLNG,   0,     (FP)Modbus_serial_recv_task, 8,      0x400, NULL}},
    {ID_MB_SERIAL_TSK,        {TA_HLNG,   0,     (FP)Modbus_serial_task,      8,      0x400, NULL}},
        …
};
```

(5)　イベントフラグ生成

Modbus スタックで使用するイベントフラグを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const FLG_TBL static_eventflag_table[] = {
// CRE_FLG(     flgid,                  {flgatr,          iflgptn});
    …
        {ID_FLG_SERIAL,          {TA_TFIFO,     0}},
        {ID_FLG_RESP_RDY,        {TA_TFIFO,     0}},
        {ID_SERIAL_RESP,         {TA_TFIFO,     0}},
    …
};
```

(6)　メールボックス生成

Modbus スタックで使用するメールボックスを生成します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const MBX_TBL static_mailbox_table[] = {
// CRE_MBX(    mbxid,                  {mbxatr,          maxmpri,   mprihd});
    …
        {ID_MB_SERIAL_MBX,       {TA_TFIFO,     0,         NULL}},
    …
};
```

(7)　ハードウェア ISR 登録

Modbus スタックで使用するハードウェア ISR を登録します。記述方法の詳細については、「プログラミングマニュアル OS 編」を参照してください。以下に例を示します。

```
const HWISR_TBL static_hwisr_table[] = {
//    inhno,              hwisr_syscall,          id,                 setptn
        …
    {UAJ0TIR_IRQn,      HWISR_SET_FLG,      ID_FLG_SERIAL,      RECV_FLG},
    {TAUJ2I1_IRQn,      HWISR_SET_FLG,      ID_FLG_SERIAL,      TIMER_FLG},
    {UAJ0TIS_IRQn,      HWISR_SET_FLG,      ID_FLG_SERIAL,      UART_STS_FLG},
        …
};
```

上記の例は、UART チャネル 0 とタイマーチャネル 1 を設定した場合の例になります。他のチャネルを使用する場合は、適宜変更してください。

(8)　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、それらはマスター側の設定と一致させてください。

各 API の記述方法については、 5.1.2.1 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                = BAUD_38400;
st_init_info.u8_parity                    = PARITY_NONE;
st_init_info.u8_stop_bit                  = STOP_BIT_ONE;
st_init_info.u8_uart_channel              = UART_CHANNEL_ZERO;
st_init_info.u8_timer_channel             = TIMER_CHANNEL_ONE;
st_init_info.u32_response_timeout_ms      = 1000;
st_init_info.u32_turnaround_delay_ms      = 200;
st_init_info.u32_interframe_timeout_us = 1750;
st_init_info.u32_interchar_timeout_us   = 750;
st_init_info.u8_retry_count               = 3;

/* register function 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 RTU slave mode   */
ercd = Modbus_serial_stack_init(&st_init_info,
                        &st_gpio_cfg,
                        MODBUS_RTU_SLAVE_MODE,
                        1);                     /* Slave ID */
```

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。

(9)　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ーフェース仕様については、5.1.1.1 章の Modbus_slave_map_init の項を参照してください。

6.2.2　　マスターモード

マスターモードでは、スレーブモードと同様の OS リソースを使用しますので、前章の(1)～(7)を参照してください。以下、マスター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

マスターモードでの初期化は、Modbus_serial_stack_init のみで行います。API の記述方法については、5.1.2.1 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                  = BAUD_38400;
st_init_info.u8_parity                      = PARITY_NONE;
st_init_info.u8_stop_bit                    = STOP_BIT_ONE;
st_init_info.u8_uart_channel                = UART_CHANNEL_ZERO;
st_init_info.u8_timer_channel               = TIMER_CHANNEL_ONE;
st_init_info.u32_response_timeout_ms    = 1000;
st_init_info.u32_turnaround_delay_ms    = 200;
st_init_info.u32_interframe_timeout_us = 1750;
st_init_info.u32_interchar_timeout_us   = 750;
st_init_info.u8_retry_count                 = 3;

/* register function 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;

/* Modbus stack be initilized in RTU master mode   */
ercd = Modbus_serial_stack_init(&st_init_info,
                                &st_gpio_cfg,
                                MODBUS_RTU_MASTER_MODE,
                                0);
```

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。

7.　サンプルアプリケーションを使ったチュートリアル

本章では、Modbus スタックサンプルに同梱しているサンプルアプリケーションを使用して、プログラムの動作確認を行います。

7.1　Modbus TCP サーバ通信

7.1.1　サンプルプロジェクト概要

ここでは、PC と Modbus TCP サーバを接続して通信を行う設定手順について記載します。サンプルでは、Windows PC 上で動作するアプリケーションから、Read 系および Write 系コマンドを発行してボード上の LED 点滅パターンを変更させます。

7.1.2　ハードウェア接続

サンプルソフト用評価ボードとして、テセラ・テクノロジー社製 EC/CL/CEC ボードまたは、IAR 社製 IAR キックスタートキットが使用できます。PC または PLC との接続には RJ45 ポートを使用します。

下図は、CEC ボードで Modbus TCP 接続をした際のイメージになります。

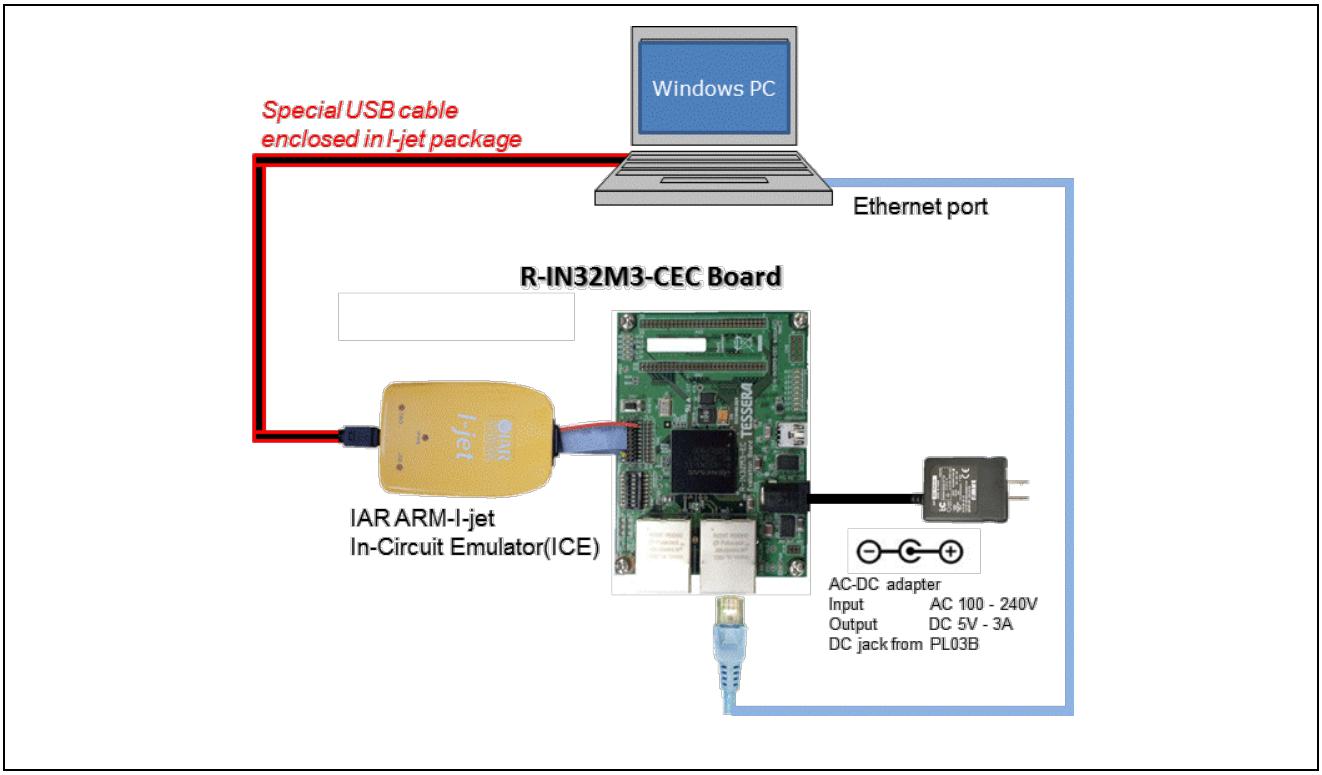


図 7.1　CEC ボードによる Modbus TCP 開発環境接続例

7.1.3　ボード IP アドレス設定

以下の手順で IP アドレスを設定してください。

(1) net_cfg.c 内のサーバーネットワークアドレスを設定します。図 7.2 に例を示します。

```
/*****************************************
     Define Local IP Address
*****************************************/
T_NET_ADR gNET_ADR[] = {
    {
        0x0,                /* Reserved */
        0x0,                /* Reserved */
        0xC0A80164,         /* IP address (192.168.   1.100) */
        0xC0A80101,         /* Gateway        (192.168.   1.   1) */
        0xFFFFFF00,         /* Subnet mask (255.255.255.   0) */
    }
};
```

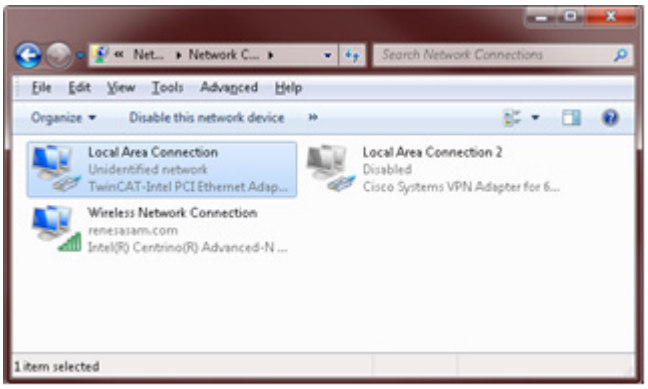
図 7.2 IP アドレスの設定例 (IP アドレスが 192.168.1.100 の場合)

(2) 使用する PC の IP アドレスは R-IN32M3 ボードと同じドメインである必要があります。（設定方法の詳細については次頁も参照してください。）
設定例では以下のアドレスを使用しています。
サブネットマスク: 255.255.255.0
PC IP アドレス　: 192.168.1.101
これらはサーバとクライアントのドメインを同じにするためです。

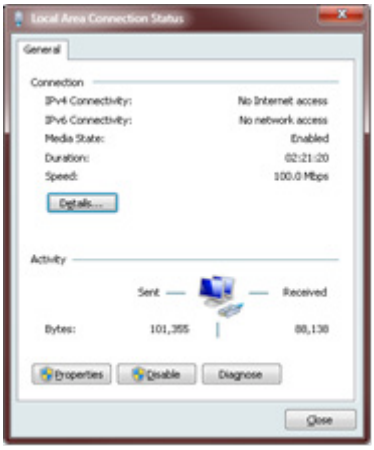
参考 : IP アドレスの設定方法

・ ネットワーク接続を開きます。

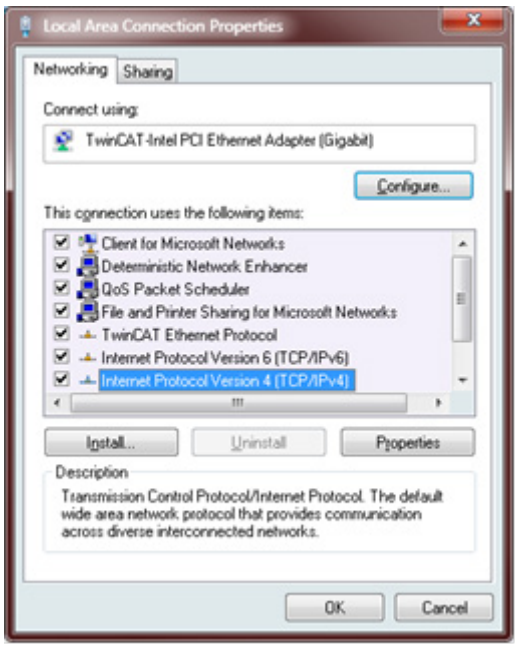
Windows7 の場合: コントロールパネル->ネットワークと共有センター->アダプターの設定の変更



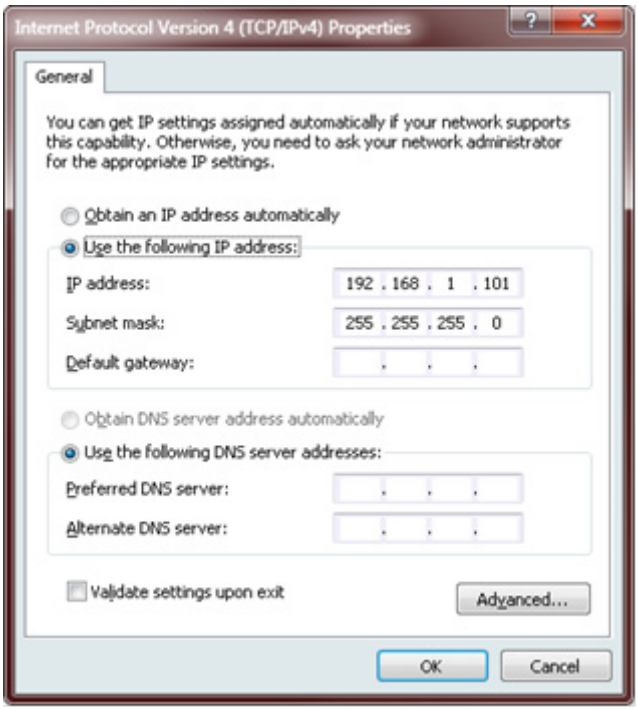
・ ローカルエリア接続をダブルクリックまたは右クリックをして”プロパティ”を選択します。



・ TCP/IPv4 を選択し、プロパティボタンを押します。



・ IP アドレスに 192.168.1.101 を、サブネットマスクに 255.255.255.0 を設定します。



以上

7.1.4　動作確認

本スタックを含んだサンプルプロジェクトを用いて、簡単な動作確認をすることができます。

7.1.4.1　動作仕様

Modbus TCP プロトコルを介して PC と通信することにより、LED 点滅パターンを動的に制御します。この制御には Read Descrete Input および Write Multiple Coil コマンドが使用されています。

詳細は、以下の手順で実行されます。

(1)　PC アプリは、Read Descrete Input コマンドを用いて、入力スイッチ注の状態をチェックします。

(2)　入力スイッチの設定により、LED の更新間隔を設定します。

更新間隔　=([SW 設定値] + 1) x 10 [msec]　: SW 設定値が 0x7F 未満の場合

10msec 固定　: SW 設定値が 0x7F 以上の場合

注　- TS-R-IN32M3-“CEC” ボードの場合

入力スイッチが実装されていません。そのため、SW 設定値は 0xFF 固定値が読み込まれる仕様としています。（更新間隔が 10msec 固定となります）

- TS-R-IN32M3-“EC” ボードの場合

“SW6”が入力スイッチとなり、接続された８ビット分のデータが入力値となります。

- TS-R-IN32M3-EC “Lite”ボード (IAR キックスタートキット)の場合

“SW3”が入力スイッチとなり、接続された８ビット分のデータが入力値となります。

- TS-R-IN32M3-“CL” ボードの場合

“SW19”が入力スイッチとなり、接続された４ビット分のデータが入力値となります。

7.1.4.2　　設定方法

以下のパスにあるワークベンチファイルをご使用ください。

[IAR project file]
¥r-in32m3_samplesoft¥Device¥Renesas¥RIN32M3¥Source¥Project¥modbus_TCP¥IAR¥main.eww

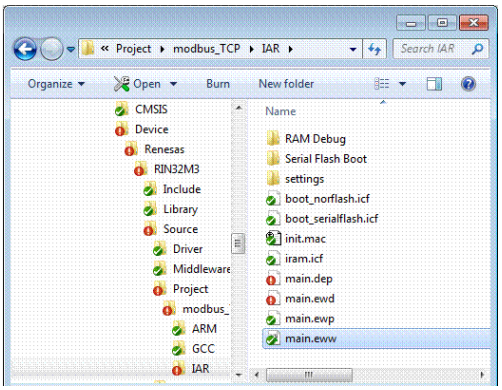


図 7.3　動作確認用ワークベンチ

次に、左上側のプルダウンメニューから RAM Debug,Serial Flash Boot,NOR Boot のいずれかをアクティブプロジェクトに選択し、評価ボード上のディップスイッチも同様の設定とします。

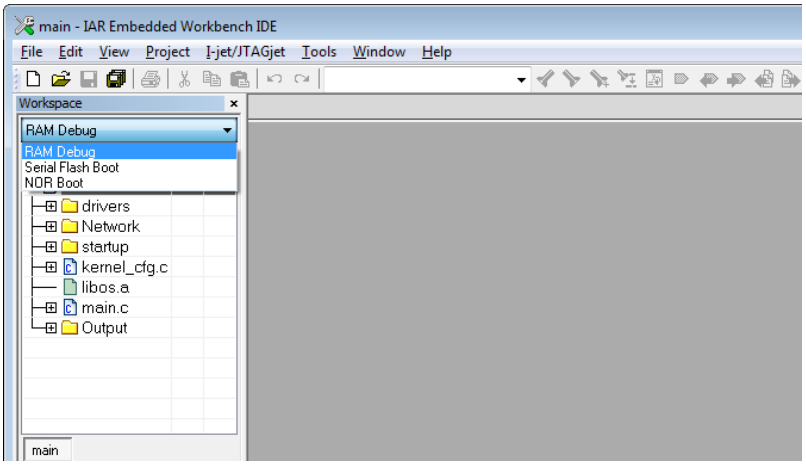


図 7.4　プロジェクトの選択

以下の手順で設定してください。

(1) コンパイルおよびダウンロードを行い、アプリケーションを実行します。

備考　TS-R-IN32M3-CL ボードをご使用の場合、“RIN32M3_CL”を C/C++コンパイラオプションのプリプロセッサタブ上のシンボル定義に追加してください。

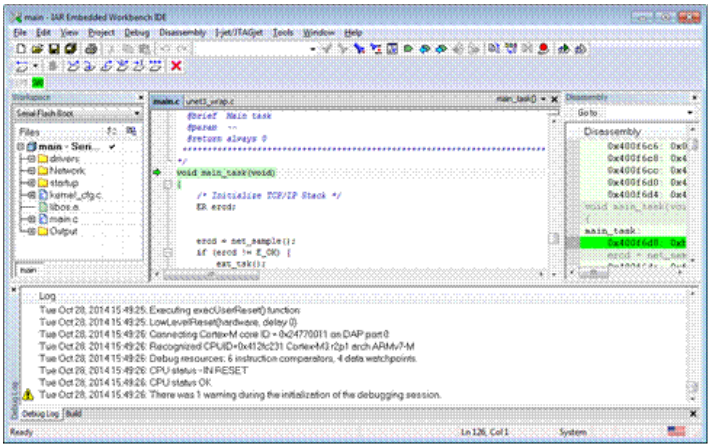


図 7.5　シリアルブートモードでの接続例

(2) Modbus サンプルプログラムに含まれる“ModbusDemoApplication.**exe**” を起動します。

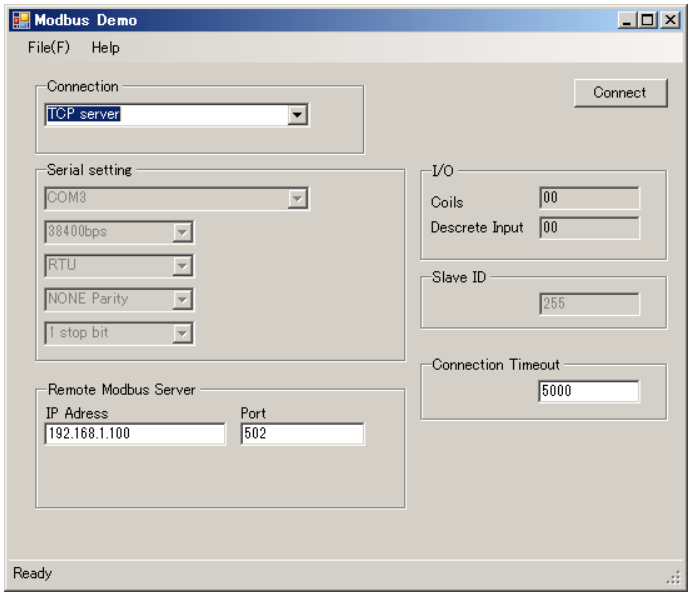


図 7.6　サンプルアプリケーション起動後

(3) “Connection” で TCP server を選択し、サーバ側の IP アドレス（例：”192.168.1.100”）およびポート番号（例：”502”）を設定します。

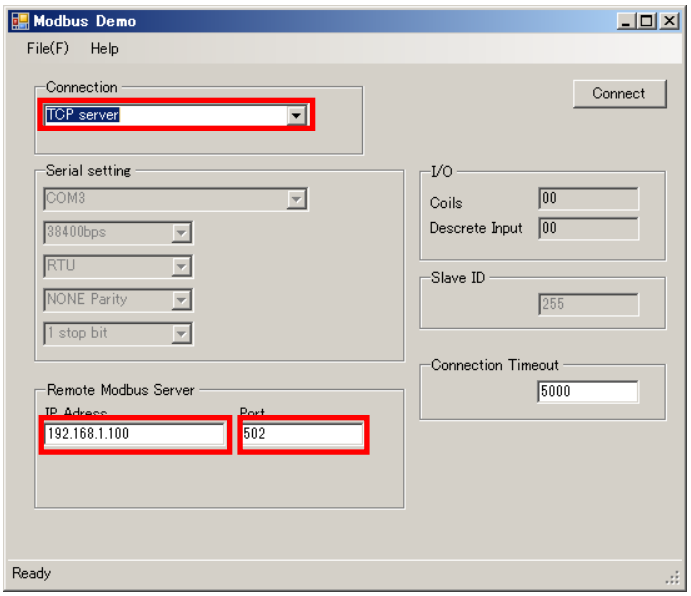


図 7.7　IP アドレスおよびポート番号の設定

(4) “Connect”を押すと、Modbus 通信が開始され LED が点滅します。

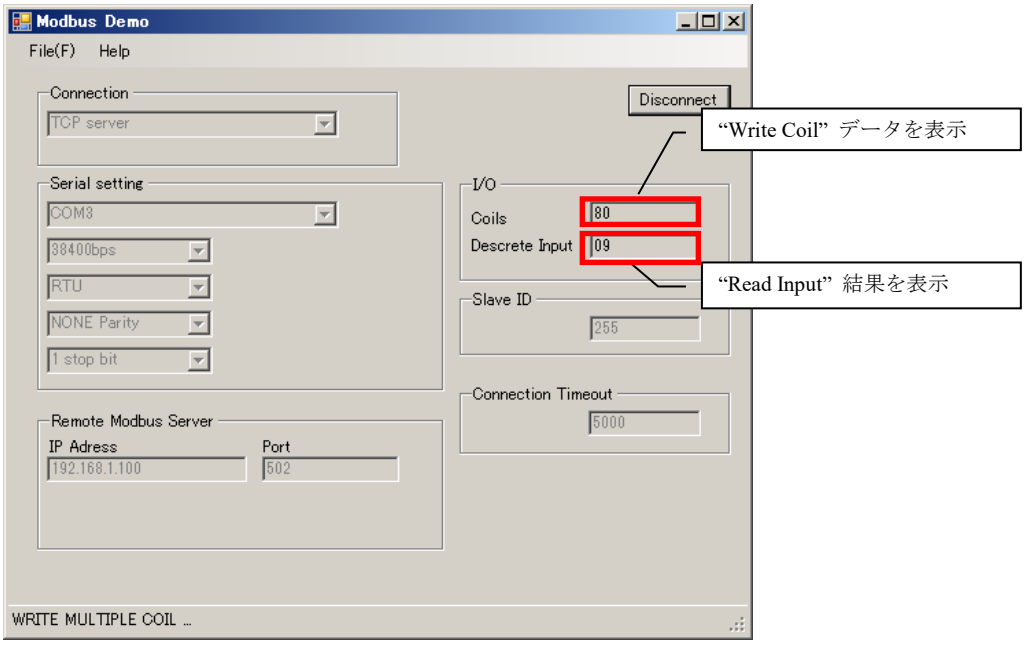


図 7.8　接続開始後

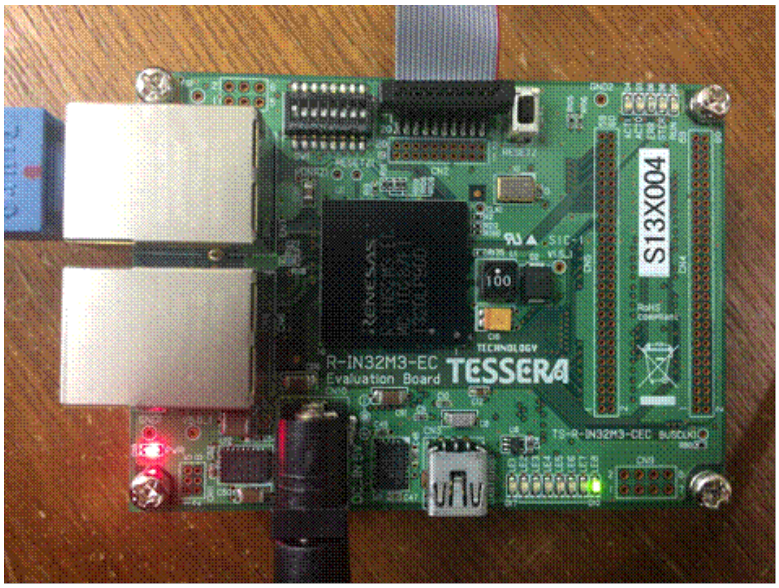


図 7.9　LED 点滅イメージ

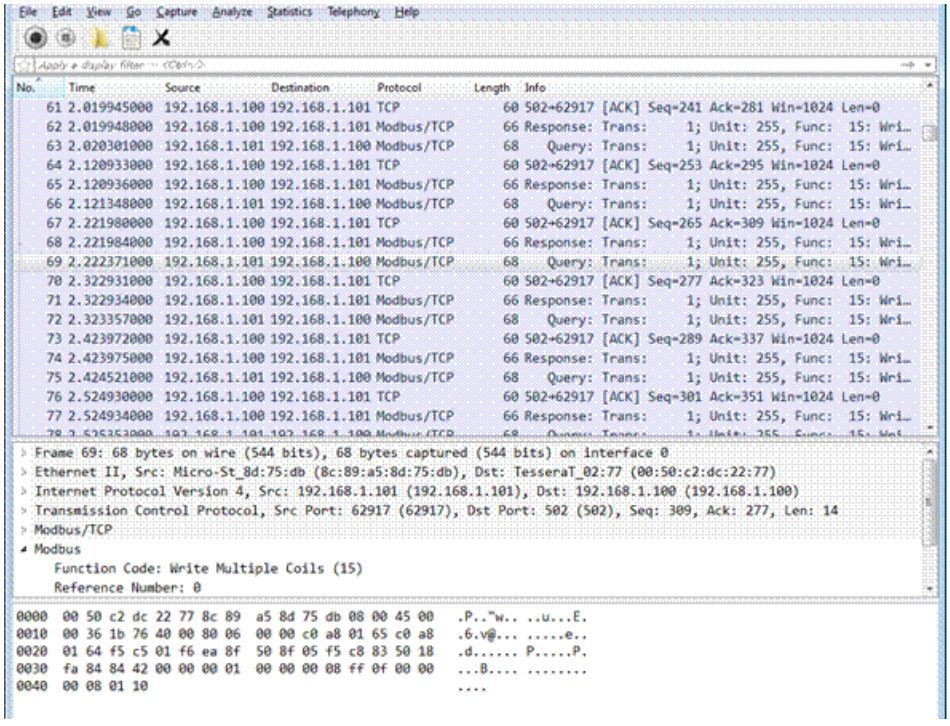


図 7.10　Wireshark で取得した通信ログ例

7.2　Modbus RTU/ASCII スレーブ接続

7.2.1　サンプルプロジェクト概要

ここでは、PC との Modbus RTU/ASCII スレーブ通信の設定手順について記載します。Windows PC 上で動作するアプリケーションと、Modbus コマンドを介して評価ボード上の LED 点滅パターンを変更させるデモを行います。

7.2.2　ハードウェア接続

サンプルソフト用評価ボードとして、テセラ・テクノロジー社製 EC/CL/CEC ボードが使用できます。PC または PLC との接続には RS-485 通信を使用します。

備考　IAR キックスタートキットはサポートしておりません。（ボード仕様が Ethernet 用のため）

すべてのボードは、RS-485 通信用に RS485 トランシーバモジュールが必要となります。表 7.1 には RS-485 トランシーバとの接続が想定されるピンを示しています。

図 7.11 は Modbus RTU/ASCII 通信を CEC ボードでセットアップした際の接続例になります。また、図 7.12 は RS-485 インターフェースの接続ピンの詳細になります。

表 7.1　Modbus RTU/ASCII 用 RS-485 I/F 用接続ピン

RS-485 トランシーバ側の接続ピン	R-IN32M3 側ポート	R-IN32M3-EC 評価ボード (TS-R-IN32M3-EC)	R-IN32M3-CL 評価ボード (TS-R-IN32M3-CL)	R-IN32M3-EC 評価ボード (TS-R-IN32M3-CEC)	R-IN32M3-EC キックスタートキット (KSK-RIN32M3EC-LT-IL)
TX	P20 (RXD0)	R125 を外して、反対側のピンを LED に接続	J22 : 1 ピン (ジャンパを外す)	CN5 : 44 ピン	RTU/ASCII 非サポート
RX	P21 (TXD0)	R126 を外して、反対側のピンを LED に接続	J27 : 3 ピン (ジャンパを外す)	CN5 : 46 ピン	
DE(/RE)	P27	CN14 : 13 ピン	CN4 : 1 ピン	CN5 : 52 ピン	
VCC	3.3V	+3.3V	+3.3V	V3.3_1	
GND	GND	GND	GND	GND2	

注　弊社では RS485 通信を以下のモジュールで確認しております：

[UART-RS485 変換]

- Sparkfun 社製“Transceiver Breakout – RS485”

https://www.sparkfun.com/products/10124

[RS485-USB 変換]

- Sparkfun 社製”USB to RS-485 Converter”

https://www.sparkfun.com/products/9822

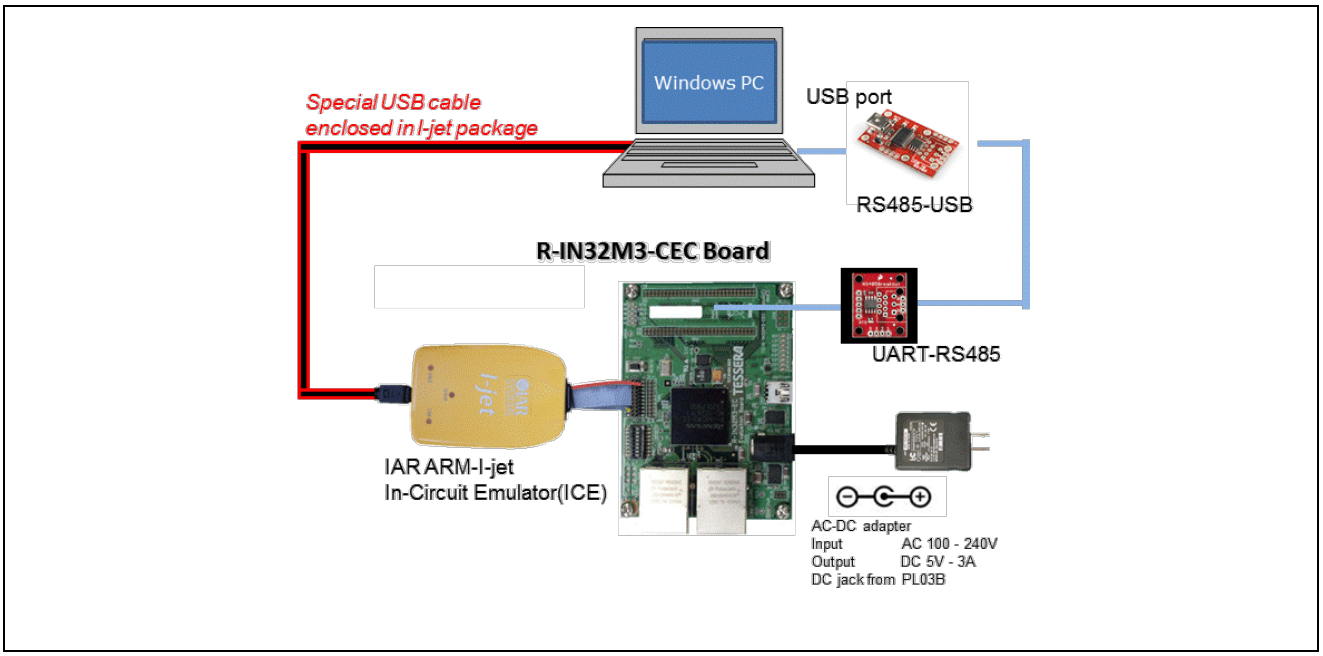


図 7.11 CEC ボードによる Modbus RTU/ASCII 開発環境接続例

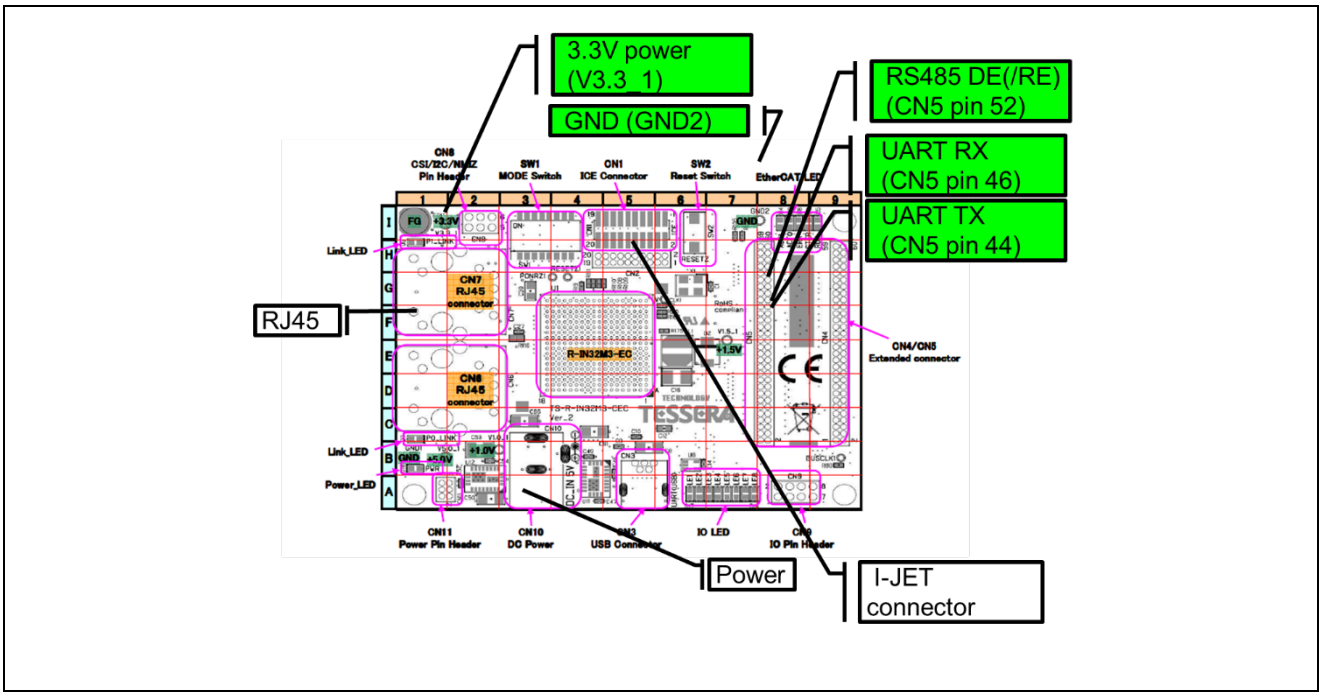


図 7.12　RS-485 接続ピン詳細

7.2.3　動作確認

本スタックを含んだサンプルプロジェクトを用いて、簡単な動作確認をすることができます。

7.2.3.1　動作仕様

Modbus RTU/ASCII プロトコルを介して PC と通信することにより、LED 点滅パターンを動的に制御します。この制御には Read Descrete Input および Write Multiple Coil コマンドが使用されています。

詳細は、以下の手順で実行されます。

(1)　PC アプリは Read Descrete Input コマンドを用いて、入力スイッチ[注]の状態をチェックします。

(2)　入力スイッチの設定により、LED の更新間隔を決定します。

更新間隔　=([SW 設定値] + 1) x 10 +200[msec]　: SW 設定値が 0x7F 未満の場合

200msec 固定　: SW 設定値が 0x7F 以上の場合

注　- TS-R-IN32M3-“CEC” ボードの場合

入力スイッチが実装されていません。そのため、SW 設定値は 0xFF 固定値が読み込まれる仕様としています。（更新間隔が 200msec 固定となります）

- TS-R-IN32M3-“EC” ボードの場合

“SW6”が入力スイッチとなり、接続された８ビット分のデータが入力値となります。

- TS-R-IN32M3-EC “Lite”ボード (IAR キックスタートキット)の場合

“SW3”が入力スイッチとなり、接続された８ビット分のデータが入力値となります。

- TS-R-IN32M3-“CL” ボードの場合

“SW19”が入力スイッチとなり、接続された４ビット分のデータが入力値となります。

(3)　PC アプリは決定した更新間隔毎に Write Multiple Coil コマンドを用いて、評価ボード側に LED の表示データを送付します。

7.2.3.2　　設定方法

以下のパスにあるワークベンチファイルをご使用ください。

[IAR project file]
¥r-in32m3_samplesoft¥Device¥Renesas¥RIN32M3¥Source¥Project¥modbus_Serial¥IAR¥main.eww

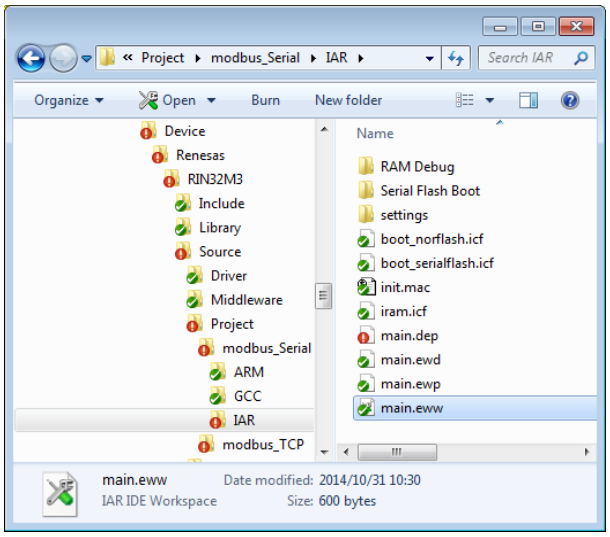


図 7.13　動作確認用ワークベンチ

次に、左上側のプルダウンメニューから RAM Debug,Serial Flash Boot,NOR Boot のいずれかをアクティブプロジェクトに選択し、評価ボード上のディップスイッチも同様の設定とします。

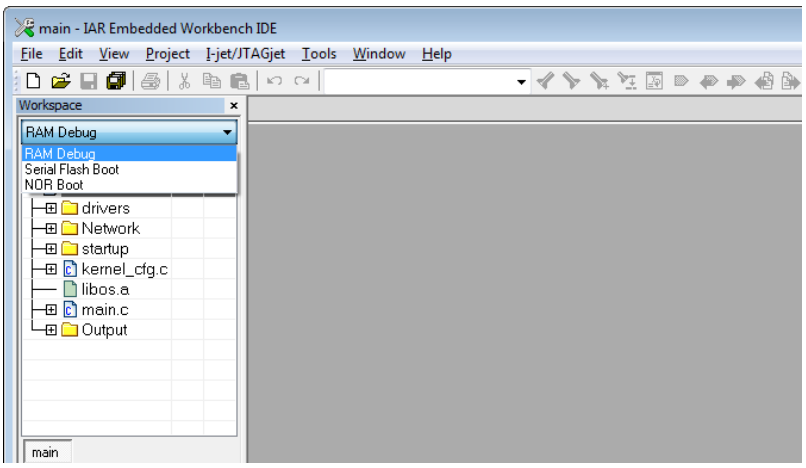


図 7.14　プロジェクトの選択

その後、ご使用になる 4 種類の Modbus シリアルモード（RTU マスター、RTU スレーブ、ASCII マスター、ASCII スレーブ）に応じて、コンパイルオプションを設定します。

IAR ワークベンチをご利用の場合、アクティブプロジェクトのオプションから設定を変更します。

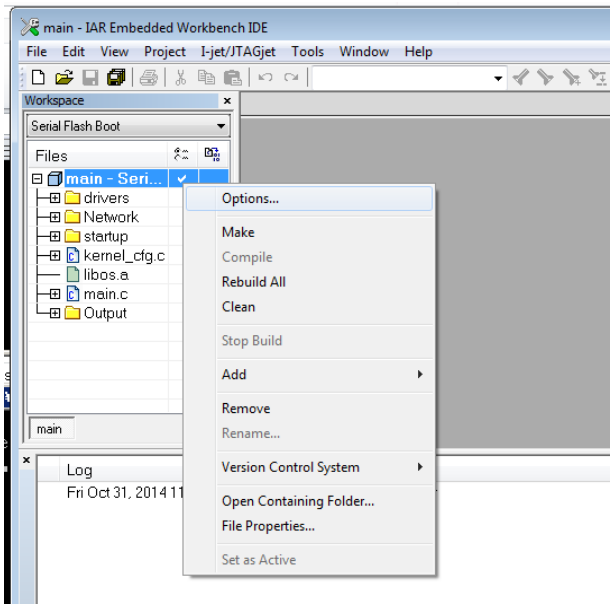


図 7.15　IAR EWARM オプション設定

“C/C++コンパイラ” カテゴリの”プリプロセッサ”タブを選択します。

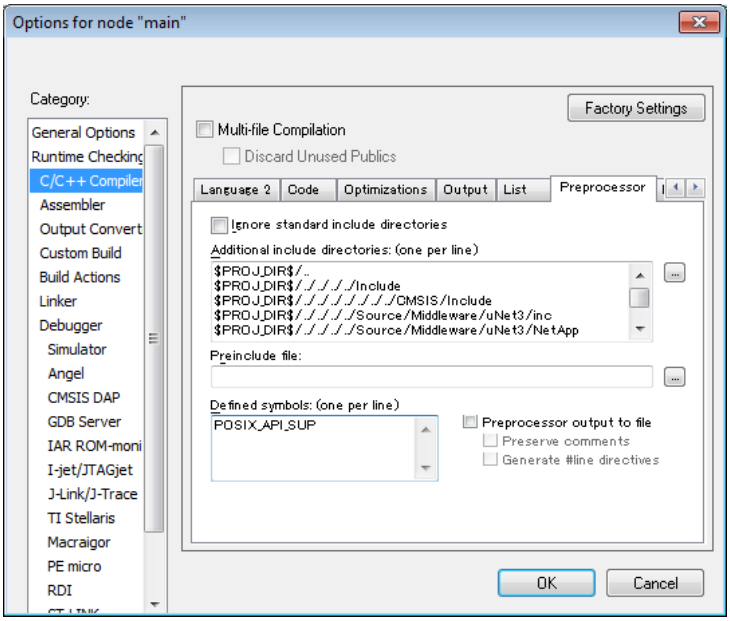


図 7.16　プリプロセッサ設定

シンボル定義に以下の設定を追加してください。

[コンパイルオプション設定]

- Modbus RTU スレーブの場合　: (追加は必要ありません)
- Modbus RTU マスターの場合　: “MODBUS_MASTER”を追加
- Modbus ASCII スレーブの場合: “MODBUS_ASCII”を追加
- Modbus ASCII マスターの場合: “MODBUS_MASTER”および“MODBUS_ASCII”を追加

次に以下の手順で設定してください。

(1) サンプルプロジェクトでコンパイルおよびダウンロードを行い、実行します。

備考　TS-R-IN32M3-CL ボードをご使用の場合、“RIN32M3_CL”を C/C++コンパイラオプションのプリプロセッサタブ上のシンボル定義に追加してください。

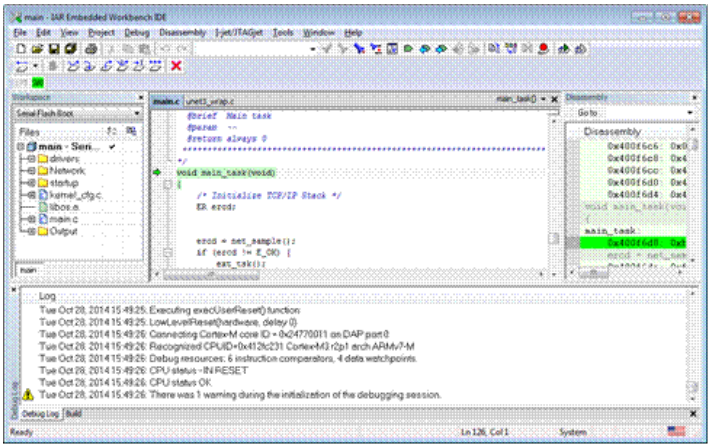


図 7.17　シリアルフラッシュへのダウンロード後

(2) 接続する Windows PC のデバイスマネージャ内にある COM 設定（ボーレート、データビット、パリティ、ストップビット、フロー制御）とサンプルプログラム内の関数 modbus_int()のスタック初期化 API の設定を合わせます。

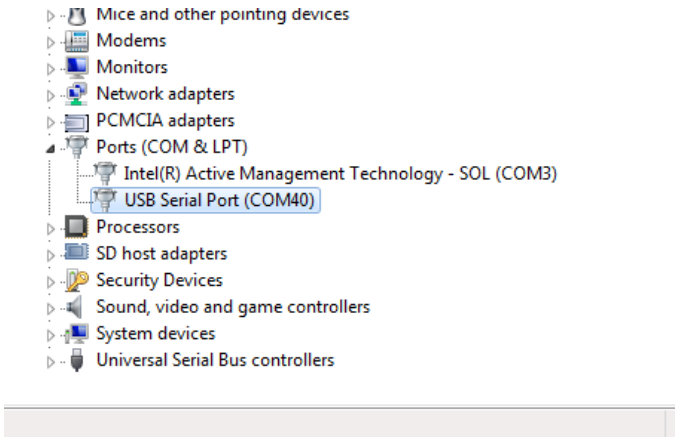
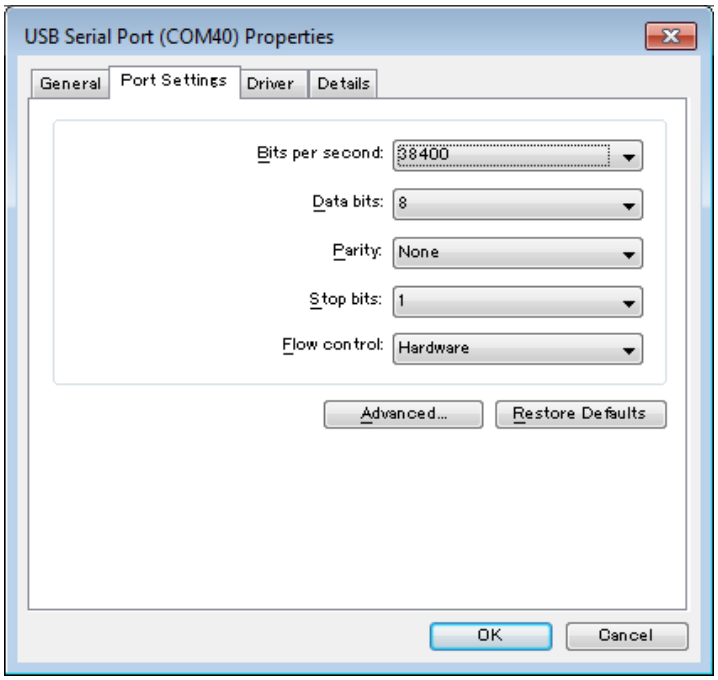


図 7.18 Windows PC のデバイスマネージャ



ポートの設定と関数 modbus_init()の設定内容を合わせる

```
    modbus_slave_map_init(&st_slave_map);
#endif

    /* serial connection setting */
    st_init_info.u32_baud_rate              = BAUD_38400;       /* Baud rate for ser
    st_init_info.u8_parity                  = PARITY_NONE;      /* Parity for serial
    st_init_info.u8_stop_bit                = STOP_BIT_ONE;     /* Stop bit for seri
    st_init_info.u8_uart_channel            = UART_CHANNEL_ZERO;/* The hardware UART
    st_init_info.u8_timer_channel           = TIMER_CHANNEL_ONE;/* The hardware time
    st_init_info.u32_response_timeout_ms    = 1000;             /* Response shall be
    st_init_info.u32_turnaround_delay_ms    = 200;              /* Delay in between
    st_init_info.u32_interframe_timeout_us = 1750;                 /* Inter frame d
    st_init_info.u32_interchar_timeout_us   = 750;          /* Inter char delay for
    st_init_info.u8_retry_count             = 3;                /* Number of retries

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

図 7.19 Windows PC のデバイスマネージャ上のシリアルポート設定

(3) Modbus サンプルプログラムに含まれる“ModbusDemoApplication**.exe**” を起動します。

(4) “Connection”で Serial Slave を選択し、COM ポートおよびシリアル通信パラメタを選択します。

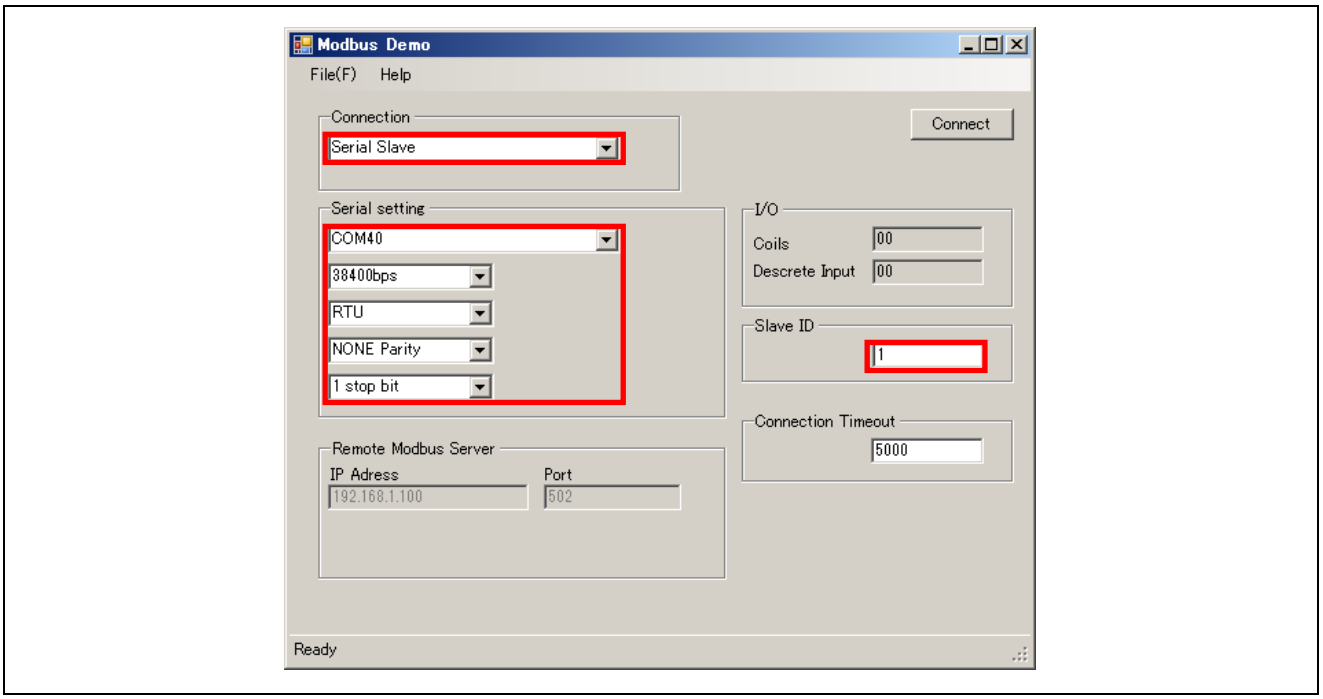


図 7.20　COM ポートおよびシリアル設定

(5) “Connet”ボタンを押下すると、Modbus 通信が開始されます。

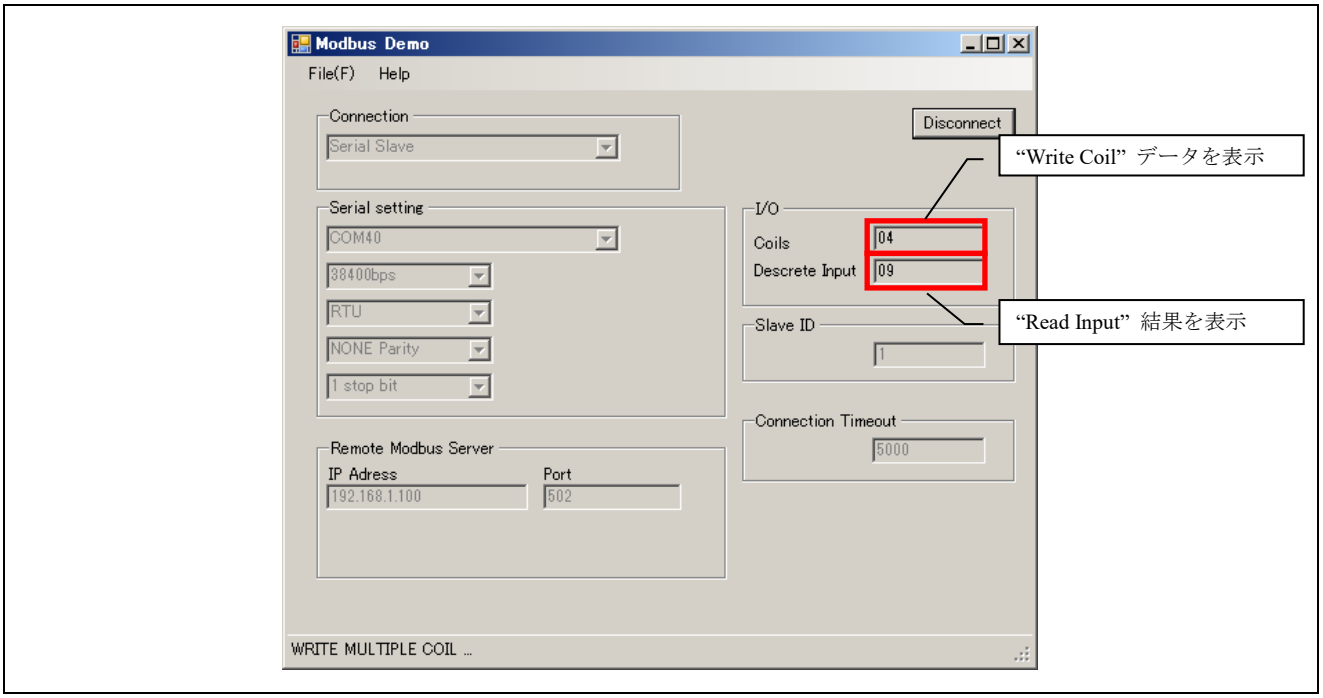


図 7.21　接続開始後

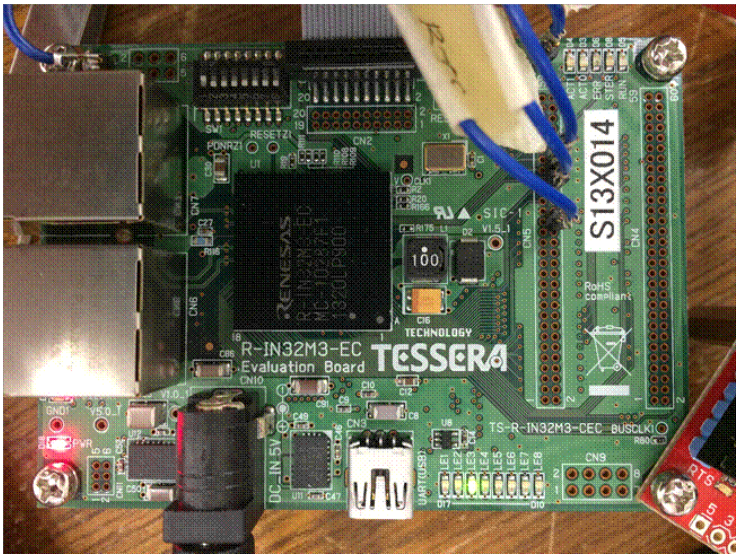


図 7.22　デモによる LED 点灯イメージ

7.3　Modbus RTU/ASCII マスター接続

7.3.1　サンプルプロジェクト概要

ここでは、PC との Modbus RTU/ASCII マスター通信の設定手順について記載します。Windows PC 上で動作するアプリケーションと、Read Coil コマンドを介して評価ボード上の LED 表示を変更させるデモを行います。

7.3.2　ハードウェア接続

Modbus RTU/ASCII スレーブと同様の接続方法になります。 7.2.2 ハードウェア接続
を参照してください。

7.3.3　動作確認

7.3.3.1　動作仕様

Modbus RTU/ASCII プロトコルを介して PC と通信することにより、LED 表示を動的に制御します。この制御には Read Coil コマンドが使用されています。
詳細は、以下の手順で実行されます。

(1) サンプルプログラムは、Read Coil コマンドを 1 秒間隔でスレーブ ID=1 に送信します。
(2) ＰＣアプリは、Read Coil コマンドを受信すると、”Coil”テキストボックスに設定された値を返却データとして返します。
(3) サンプルプログラムは、Read Coil コマンドの返却データを LED に出力します。

7.3.3.2　　設定方法

Modbus RTU/ASCII スレーブとほぼ同じになります。詳細は、7.3.3.2 設定方法 を参照してください。スレーブ設定とは以下の点が異なります。

- コンパイルオプションのシンボル定義を追加します。

[コンパイルオプション設定]

- Modbus RTU マスターの場合 : “MODBUS_MASTER”を追加
- Modbus ASCII マスターの場合: “MODBUS_MASTER”および“MODBUS_ASCII”を追加

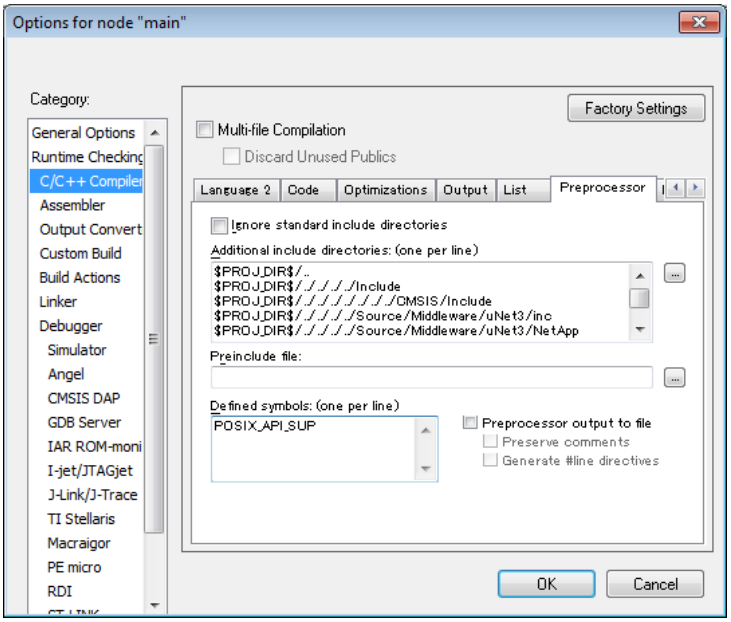


図 7.23　プリプロセッサ設定

(1) コンパイルおよびダウンロード後、サンプルソフトを実行します。

(2) サンプルソフトパッケージに含まれている“ModbusDemoApplication**.exe**”を起動します。

(3) “Connection”で”Serial Master”を、”Serial setting”で COM ポートおよびシリアル接続パラメタを指定します。

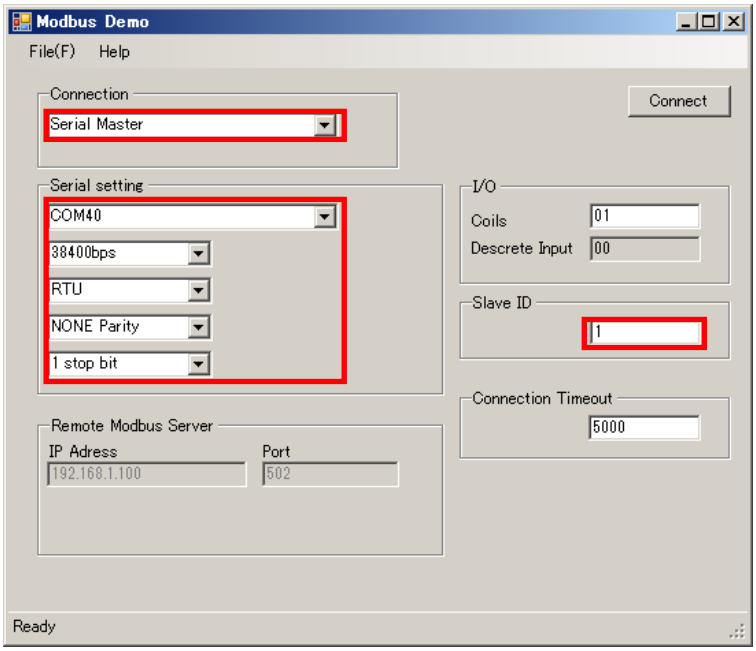


図 7.24　COM ポートおよびシリアル設定

(4) “Connect”ボタンを押すと、Modbus 通信可能な状態になります。

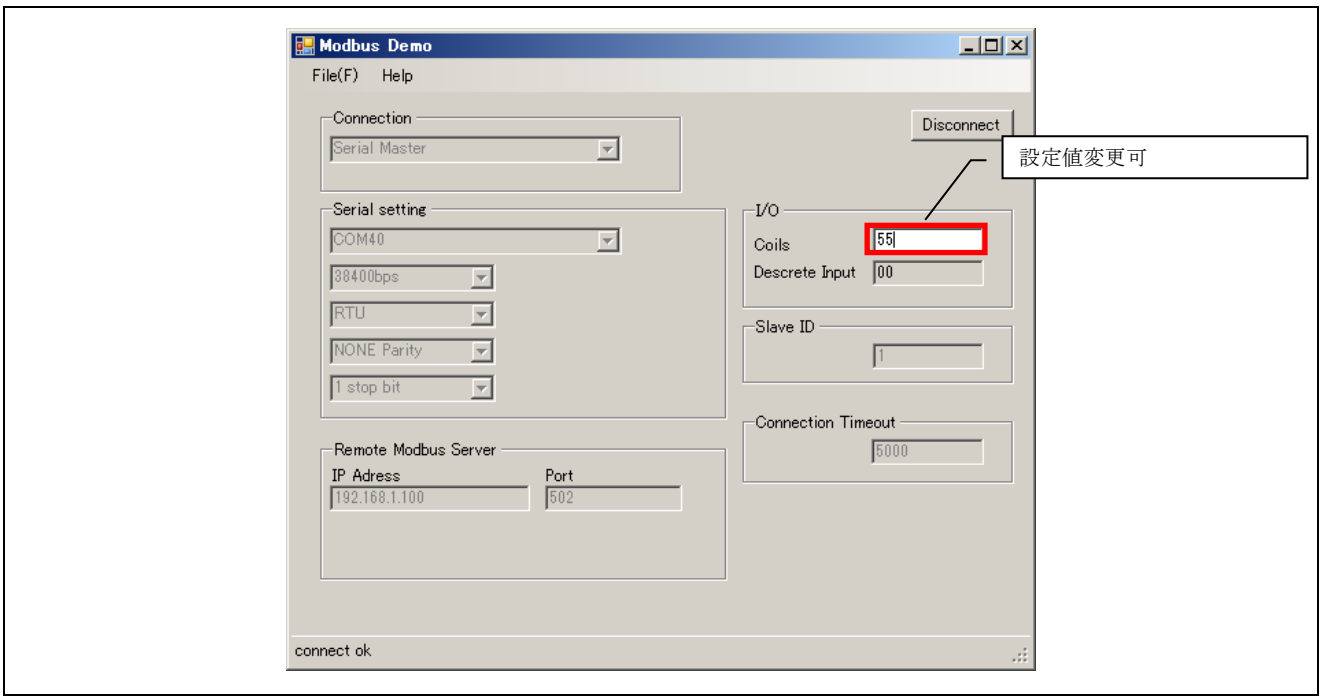


図 7.25　通信開始後

(5) もう一度評価ボードをリセットすると、LED 点滅が開始されます。 ★

注意　サンプルアプリケーションは、スレーブデバイスの動作開始後に起動してください。

7.4　Modbus TCP サーバ – RTU/ASCII マスター間ゲートウェイ通信

7.4.1　サンプルプロジェクト概要

ここでは、PC との Modbus TCP サーバ—Modbus RTU/ASCII マスター間ゲートウェイ通信の設定手順について記載します。Windows PC 上で動作するアプリケーションと、Modbus コマンドを介してゲートウェイ先の Modbus RTU/ASCII スレーブデバイス上の LED 表示を変更させるデモを行います。

7.4.2　ハードウェア接続

動作確認には２つの評価ボードが必要となります。一つはゲートウェイモジュール用、もう一つは RTU/ASCII スレーブデバイスになります。

下図は、Modbus ゲートウェイ通信用に CEC ボードでセットアップした際の接続例になります。

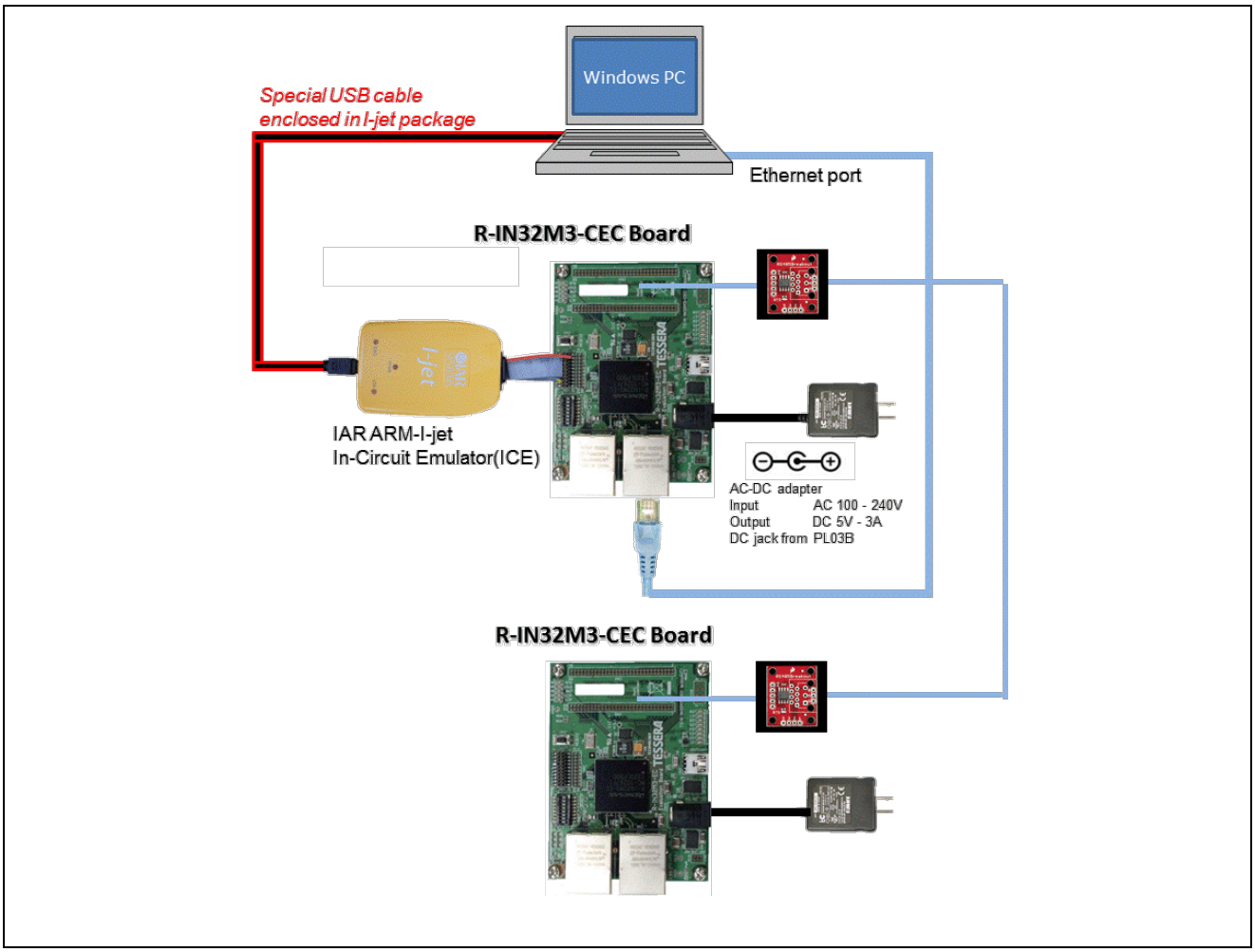


図 7.26　CEC ボードによる Modbus ゲートウェイ開発環境接続例

7.4.3　動作確認

本スタックを含んだサンプルプロジェクトを用いて、簡単な動作確認をすることができます。

7.4.3.1　動作仕様

7.1.4 章の Modbus TCP と同じ仕様になります。Modbus TCP プロトコルを介して PC と接続し、LED 点滅パターンを動的に制御します。

7.4.3.2　設定方法

(1) Modbus RTU/ASCII スレーブモジュール用
Modbus RTU/ASCII スレーブと同じになります。7.2.3.2 設定方法を参照してください。.

(2) Modbus ゲートウェイモジュール用
Modbus TCP サーバとほぼ同じになります。7.2.3.2 設定方法を参照してください。

- コンパイルの際、“Defined symbols”に定義を追加します。

[コンパイルオプション設定]
- Modbus ゲートウェイの場合　: “MODBUS_GATEWAY”を追加

(3) Modbus サンプルソフトパッケージに含まれる、 “ModbusDemoApplication**.exe**”を起動します。

(4) “Connection”で”TCP gateway”を選択し、IP アドレス(例：“**192.168.1.100**”)、ポート番号(例：“**502**”)およびスレーブ ID(例：“**1**”)を設定します。

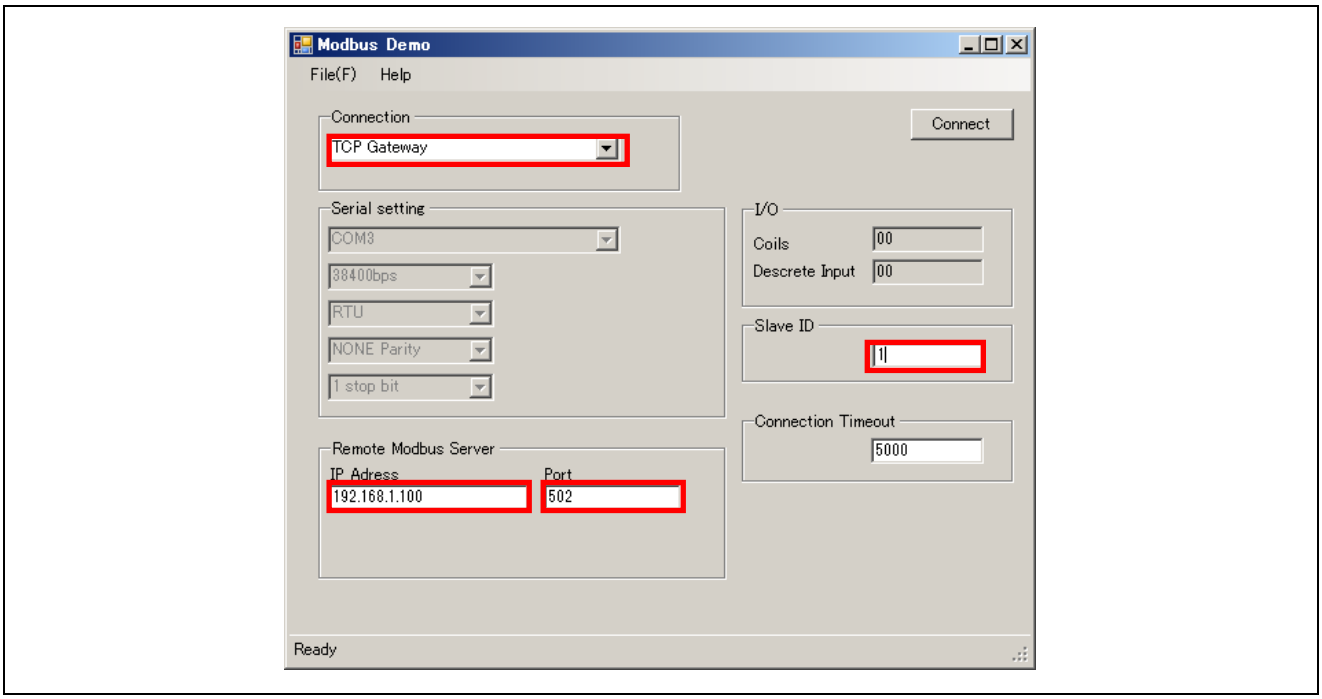


図 7.27　接続パラメタ設定

(5) “Connect”を押下すると、Modbus 通信により LED 点灯が開始されます。

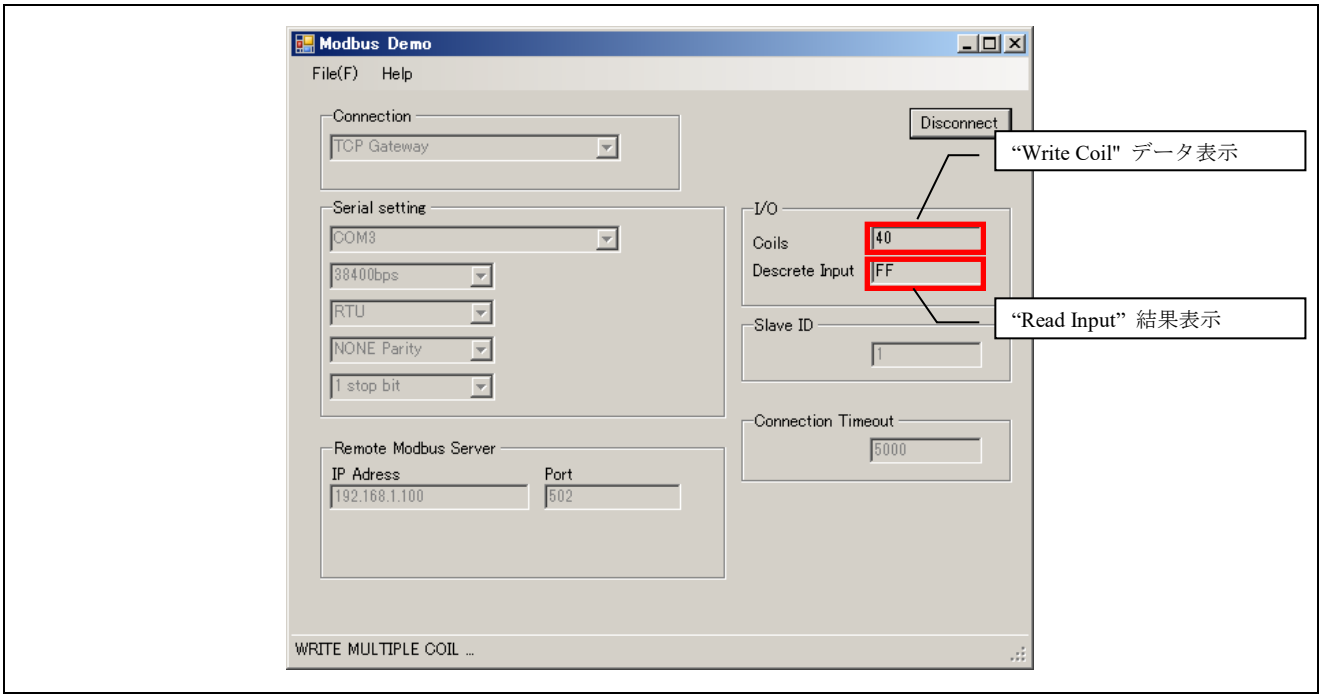


図 7.28　実行開始後

8. 制限事項

- ゲートウェイ機能は Modbus シリアルマスターモードの機能を使用しています。したがって、ゲートウェイモードでは、シリアルマスターモードでサポートされている機能コードでのみ Modbus 通信が可能となります。
- TCP 接続に優先順位はありません。新しい TCP 接続を受信すると、古い接続からクローズしていきます。

改訂記録	R-IN32M3 シリーズ　ユーザーズ・マニュアル　Modbus スタック編

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2015.04.08	-	日本語版 初版発行
1.01	2015.08.31	29	マクロ名の誤記を修正
		116 117	DHCP 機能は使用できないため説明を削除
1.02	2015.12.25	3	開発ツールの対応バージョンを更新
		119 126	各評価ボードにおける入力スイッチの記述を修正
		126	PC アプリケーションによる更新時間を修正
1.03	2017.02.28	1	正式版に統一の為、暫定版を削除
		2	開発ツール対応表を更新
1.04	2019.04.19	2	1.2.1　開発ツール 表記変更

[メ　モ]

R-IN32M3 シリーズ　ユーザーズ・マニュアル
Modbusスタック編

発行年月日　2015年04月08日　Rev.1.00
2019年04月19日　Rev.1.04

発行　ルネサス エレクトロニクス株式会社
〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

R-IN32M3 シリーズ　ユーザーズ・マニュアル

Modbus スタック編

ルネサス エレクトロニクス株式会社

R18UZ0028JJ0104

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社 〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

Colophon 3.1