

# ルネサスマイクロコンピュータ

R20UW0031JJ0106

Rev.1.06

## 組み込み用 TCP/IP M3S-T4-Tiny: ユーザーズマニュアル

---

2014.04.01

### 要旨

本ユーザーズマニュアルは、M3S-T4-Tiny ソフトウェアライブラリの使用方法を説明します。

### 動作確認デバイス

ルネサスマイクロコンピュータ

### 目次

1. はじめに.....	5
1.1 概要 .....	5
1.2 TCP/IP の基礎知識.....	7
1.2.1 コネクションについて .....	7
1.2.2 クライアント/サーバモデルについて .....	7
1.2.3 IP アドレスとポート番号 .....	7
1.2.4 ソケットについて .....	7
1.3 TCP/IP の基礎知識.....	8
1.3.1 コネクションの扱い.....	8
1.3.2 クライアント/サーバモデルの扱い.....	8
1.3.3 受付口および通信端点について.....	8
1.4 用語の定義 .....	9
1.4.1 受付口 .....	9
1.4.2 通信端点 .....	9
1.4.3 タイムアウト.....	9
1.4.4 ノンブロッキングコール、コールバック .....	9
1.4.5 ポーリング .....	9
1.4.6 Pending .....	9
1.4.7 MSS (Maximum Segment Size).....	10
1.4.8 IP フラグメント .....	10
1.4.9 受信(送信)ウィンドウ .....	10
1.4.10 PPP (Point to Point Protocol).....	10
1.4.11 PAP (Password Authentication Protocol) .....	10
1.4.12 ICMP (Internet Control Message Protocol).....	10
1.4.13 ARP (Address Resolution Protocol).....	10
1.5 プログラム開発手順.....	11
2. T4 の概要 .....	12
2.1 製品構成 .....	12
2.1.1 ドキュメント.....	12
2.1.2 T4 ライブラリ .....	12
2.1.3 サンプルプログラム.....	12
2.2 ライブラリの概略仕様 .....	13
3. T4 Library マクロ定義.....	14
4. T4 ライブラリ構造体.....	15
4.1 TCP オブジェクト構造体.....	15
4.2 UDP オブジェクト構造体 .....	15
5. T4 ライブラリ定数 .....	16
5.1 TCP 機能の戻り値.....	16
5.2 UDP 機能の戻り値 .....	16
5.3 API で使用されているエラーコード .....	16
5.4 タイムアウト指定 .....	16
5.5 特殊な IP アドレスとポート番号 .....	16

6. T4 コンフィグレーションファイル.....	17
6.1 LAN ポート/シリアルポート数の定義.....	18
6.2 TCP 受付口の定義.....	19
6.3 TCP 通信端点の定義.....	20
6.4 UDP 通信端点の定義.....	21
6.5 IP ヘッダ関連の定義.....	22
6.6 TCP オプションの定義.....	23
6.6.1 TCP の MSS.....	23
6.6.2 シーケンス番号の初期値.....	23
6.6.3 2-MSL 待ち時間.....	24
6.6.4 再転送タイムアウトの最大値.....	24
6.6.5 TCP 多重送信の有無.....	24
6.7 UDP オプションの定義.....	25
6.7.1 UDP ゼロチェックサムの動作定義.....	25
6.8 自局の設定 (Ethernet).....	26
6.9 自局の設定(PPP).....	28
6.9.1 自局の設定(PPP クライアント).....	29
6.9.2 自局の設定(PPP サーバ).....	29
6.9.3 モデムコマンドの設定(PPP サーバ/PPP クライアント共通).....	29
7. T4 コンフィグレーションファイル(複数 LAN ポート).....	30
7.1 LAN ポート数の定義.....	31
7.2 TCP 受付口の定義.....	32
7.3 TCP 通信端点の定義.....	33
7.4 UDP 通信端点の定義.....	34
7.5 IP ヘッダ関連の定義.....	35
7.6 TCP オプションの定義.....	36
7.7 UDP オプションの定義.....	37
7.7.1 UDP ゼロチェックサムの動作定義.....	37
7.8 自局の設定 (Ethernet).....	38
8. T4 ライブラリの API.....	40
8.1 tcp_acp_cep.....	41
8.2 tcp_con_cep.....	42
8.3 tcp_sht_cep.....	44
8.4 tcp_cls_cep.....	45
8.5 tcp_snd_dat.....	46
8.6 tcp_rcv_dat.....	47
8.7 tcp_can_cep.....	48
8.8 udp_snd_dat.....	49
8.9 udp_rcv_dat.....	51
8.10 udp_can_cep.....	53
8.11 ppp_open.....	54
8.12 ppp_close.....	55
8.13 ppp_status.....	56
8.14 ppp_api_req.....	57
8.15 callback.....	59

8.16	tcpudp_get_ramsize .....	60
8.17	tcpudp_open.....	61
8.18	_process_tcpip .....	62
8.19	tcpudp_close .....	63
9.	Ethernet/PPP ドライバ関連の API 仕様 .....	64
9.1	lan_open.....	66
9.2	lan_close .....	67
9.3	sio_open.....	68
9.4	sio_close .....	69
9.5	modem_active_open.....	70
9.6	modem_passive_open .....	71
9.7	modem_close.....	72
9.8	get_random_number.....	73
10.	サンプルプログラム .....	74
10.1	main 関数のフロー .....	74
10.2	TCP エコーバックサーバ関数(ブロッキングコールの場合)のフロー .....	74
10.2.1	エコーバックサーバ関数 .....	74
10.3	TCP エコーバックサーバ関数(ノンブロッキングコールの場合)のフロー .....	74
10.3.1	エコーバックサーバ関数 .....	74
10.3.2	コールバック関数のフロー .....	74
10.4	UDP エコーバックサーバ関数(ブロッキングコールの場合)のフロー .....	74
10.4.1	エコーバックサーバ関数 .....	74
10.5	UDP エコーバックサーバ関数(ノンブロッキングコールの場合)のフロー .....	74
10.5.1	エコーバックサーバ関数 .....	74
10.5.2	コールバック関数のフロー .....	74
10.6	実行環境 .....	81
10.7	実行方法 .....	82
10.8	実行環境 (複数 LAN ポート).....	83
10.9	実行方法 (複数 LAN ポート).....	84
11.	T4 制限・注意事項 .....	85

## 1. はじめに

### 1.1 概要

Tiny な TCP/IP ライブラリ M3S-T4-Tiny(以下、T4 と呼びます)は、ルネサスエレクトロニクス製マイクロコンピュータに対応したネットワークソフトウェアライブラリです。本マニュアルでは T4 を使用してアプリケーションプログラムを作成するための共通情報を提供します。機種に依存する情報(開発環境など)は、各導入ガイドで提供します。

※対応 CPU や Ethernet、PPP 対応は順次拡大する予定です

(詳細情報はルネサスエレクトロニクスの web サイトをご参照ください)

T4 の位置付けを図 1 に示します。

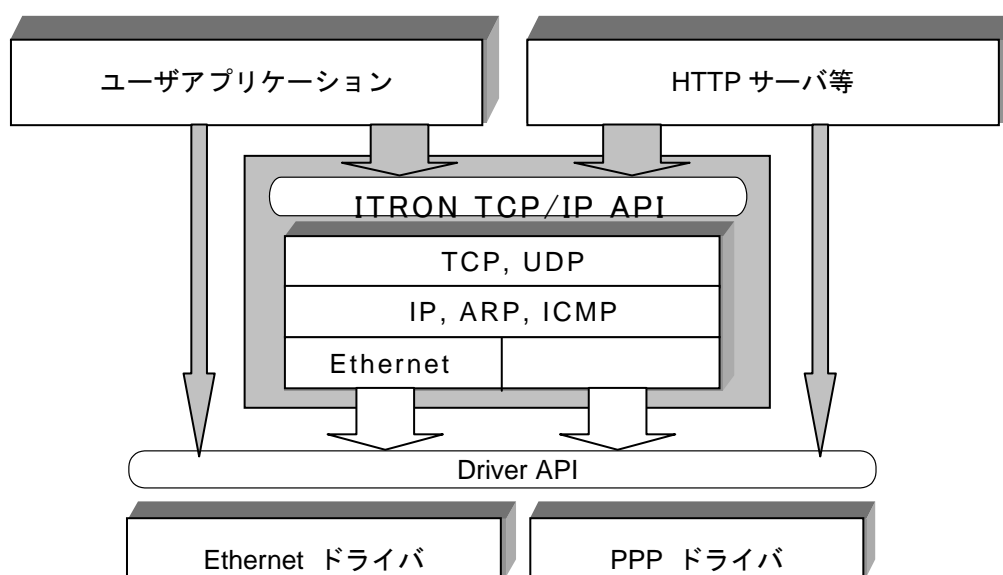


図 1 T4 の位置付け

図 1 では、ネットワークミドルウェアに係わるソフトウェアの全体像を示し、その中の T4 の位置付けを表しています。中央のグレーの背景で示した部分が T4 です。矢印は、関数呼び出しを表しています。

T4 は、Ethernet ドライバ、または PPP ドライバとアプリケーションプログラムの中でプロトコル処理を行い、ネットワークに対してデータの送受信を行います。

T4 には、プロトコル処理として TCP、UDP、IP、ICMP、ARP、PPP が含まれています。ユーザアプリケーションや上位プロトコルのプログラムから TCP や UDP の通信を行う場合、T4 で規定する API (ITRON TCP/IP API に準拠) を利用してライブラリ内の関数を呼び出します。IP、ICMP、ARP、PPP については、TCP と UDP の下に位置するプロトコルであり、これらのプロトコル処理関数をユーザプログラムから直接、呼び出すことはありません。

T4 では、データリンク層/物理層として Ethernet と PPP をサポートします。これらの内、Ethernet コントローラやシリアル I/O 等、ご使用になるハードウェアに依存する部分については、ドライバとして T4 のライブラリから切り離しています。

ドライバ API の仕様を規定していますので、お使いになるハードウェアに合わせてドライバを作成してください。ドライバ API の仕様は、ドライバのユーザーズマニュアルに記載しています。

※ドライバのユーザーズマニュアルは、本マニュアルとは別に、本製品パッケージに付属しています。

なお T4 を使用してプログラムを作成する場合、ドライバ内の初期設定関数および終了関数を直接、呼び出していただく必要があります(図 1 ではグレーの矢印で示しました)。これらの API 関数仕様については、ドライバのユーザーズマニュアルだけでなく、本 T4 のマニュアル中でも説明しています。

複数 LAN ポートをサポートしています。ドライバの設定で自己 IP アドレスを LAN ポート数の分だけ設定することが出来ます。

本製品パッケージには、ドライバのサンプルプログラムが付属しています。お客様独自のドライバを作成する際にご参照ください。

T4 のライブラリでは、リアルタイム OS の機能を使用しておりません。

## 1.2 TCP/IP の基礎知識

本節では、T4 をご使用いただく上で必要となる TCP/IP の基礎知識について説明します。T4 では、組み込みシステムに適した API を採用しており、BSD や Linux のような一般の TCP/IP とは異なる部分があります。これらについては後節で説明しますが、その際、本節で説明する一般の TCP/IP と対比できるようにしました。

なお本節で説明する内容は、本マニュアルをお読みいただく上で必要となる最小限の内容に限定しています。TCP/IP についてより詳しい説明が必要な場合は、市販の書籍をご活用いただきますようお願いいたします。

### 1.2.1 コネクションについて

TCP では、仮想的な通信路を作って通信します。この仮想的な通信路のことを「コネクション」と呼びます。そして通信相手とコネクションを持つことを「コネクションを確立する」と言います。1 つのプログラム（通信するコンピュータ上で稼動するプログラムのこと）は、複数のコネクションを持つことができ、これにより複数のコンピュータプログラムと通信できます。ただし 1 つのコネクションで通信できる相手は 1 つだけです。

1 つのコネクションを確立するには、通信する相手との間（すなわち 2 者間）で交渉する必要があります。そして 1 つのコネクションが確立できると、双方向で通信できるようになります（送信用、受信用に異なる 2 つのコネクションを用意する必要はありません）。このコネクション確立のための手続きを「コネクションを要求する」と表現します。

これに対して、UDP では通信の際にコネクションを確立しません。相手の通信状態を確認せずに、一方的に通信を試みるため、コネクションの確立や、正常に通信できたか確認する際の交渉は行いません。

### 1.2.2 クライアント/サーバモデルについて

TCP では、前記のコネクションを確立する際の交渉において、クライアント/サーバモデルを採用しています。したがって通信する両者の内、どちらか一方が「クライアント」、もう片方が「サーバ」の役目を担います。コネクションを確立する際、サーバ側は、通信相手を特定せずに、誰かが自分に対してコネクションを要求してくるのを待ちます。これに対して、クライアント側は、通信相手となるサーバのアドレスを指定し、通信相手を特定した上でコネクションを要求します。

この両者の手続きによってコネクションを確立した後は、クライアント/サーバに係わらず、自由に送受信できるようになります。

ただし一般的には、TCP/IP の通信プログラム自体をクライアント/サーバモデルで実装することが多く、その場合、サーバ側はクライアント側からの要求を待って、クライアント側に情報を返信するように実装することになります。

### 1.2.3 IP アドレスとポート番号

IP アドレスとポート番号は、TCP/IP および UDP/IP の通信において、通信相手や自分を指し示すためのアドレス情報です。

- IP アドレスは、通信対象の通信ポートを示すユニークなアドレスです。
- ポート番号は、通信対象のアプリケーションプログラムを示す番号です。

IP アドレスは、インターネットに接続しないローカルなネットワークの場合を除き、全世界でユニークな番号である必要があります（IP アドレスを割り当てる専門機関があります）。

ポート番号は、同じコンピュータ上で稼動する全ての通信プログラムにおいて、ユニークに設定する必要があります。

### 1.2.4 ソケットについて

一般の TCP/IP では、通信においてソケットと呼ばれる API を利用します。TCP/IP および UDP/IP では、パケット通信において複雑な処理が行われていますが、アプリケーションプログラムを作成する際は、初期設定、コネクション要求、送信、受信、終了処理等の API が用意されており、複雑なパケット処理を意識することなく、C 言語のファイル入出力に似た感覚で通信処理をプログラミングできます。API では、socket と呼ばれる抽象的なデータ構造（構造体）を用います。

## 1.3 ITRON TCP/IP API 仕様の基礎知識

本節では、T4 において採用した、ITRON TCP/IP API 仕様（以下、ITRON 仕様と呼びます）の基礎知識について説明します。ただし本節で説明する内容は、本書をお読みいただく上で必要になる最小限の内容に限定しています。ITRON TCP/IP API 仕様についてより詳しい説明が必要な場合は、下記のホームページに掲載されている(社)トロン協会 ITRON 専門委員会 Embedded TCP/IP 技術委員会発行の仕様書をご活用いただきますようお願いいたします。

URL: <http://www.ertl.jp/ITRON/SPEC/tcpip-j.html>

### 1.3.1 コネクションの扱い

ITRON TCP では、前節のコネクションのことを日本語で「接続」と表現します。そしてコネクションを要求する手続きを「接続要求」「接続要求待ち」と言います。この両者の違いについては、後で説明します。

### 1.3.2 クライアント/サーバモデルの扱い

ITRON TCP でも、一般の TCP/IP と同様に、接続の手続きにおいてクライアント/サーバモデルを採用しています。しかし ITRON TCP の仕様書中では、クライアント/サーバ という用語は用いません。その代わりに、接続に関する API として、サーバ用に「接続要求待ち（受動オープン）」、クライアント用に「接続要求（能動オープン）」の 2 つを用意して区別しています。すなわち「接続を要求するのがクライアント」で、「その接続要求を待つのがサーバ」です。

### 1.3.3 受付口および通信端点について

ITRON TCP では、通信において socket のデータ構造は用いず、その代わりに「受付口」「通信端点」と呼ばれる構造体を用います。これは、組み込みシステムにおいて求められる以下の性質を考慮した結果です。

- (1)バッファのためのメモリ領域や、データのコピーの回数が最小限であること。
- (2)できる限りプロトコルスタック内部で動的なメモリ管理を使う必要がないこと。
- (3)非同期インターフェイスないしはノンブロッキングコールをサポートすること。
- (4)API ごとのエラーの詳細がわかることが望ましい。

((社)トロン協会 ITRON 専門委員会 Embedded TCP/IP 技術委員会発行の仕様書より抜粋)



## 1.4 用語の定義

### 1.4.1 受付口

受付口は、TCP の通信において API で使用する構造体です。具体的には（サーバ側が）通信相手（クライアント）からの接続要求を待つための API（関数 `tcp_acp_cep`）で使われます。受付口は、プログラム全体で複数持つことができ、プログラム全体で 1 つの構造体配列として定義します。

接続要求待ちの API では、使用する受付口をパラメータで指定しますが、その際、構造体のポインタを指定するのではなく、1 から始まるユニークな ID 番号で指定します。この ID 番号はプログラマ自身で定義するのではなく、受付口の構造体配列を定義する際に T4 が先頭から順に 1、2... の ID 番号を割り当てます。したがってプログラマは、構造体配列の先頭で定義した受付口を 1 番、2 番目の受付口を 2 番として、それぞれの ID 番号が何番であるか把握できます。

トロン協会が規定する仕様では、受付口はメンバとして属性、自局の IP アドレス、自局のポート番号を持っています。

### 1.4.2 通信端点

通信端点は、TCP と UDP の通信において各種 API で使用する構造体です。通信端点は（受付口と同じように）構造体配列として定義し、先頭の構造体を 1 番とする ID 番号で表します。この通信端点 ID を使用して、通信相手との接続の確立、データの送受信、接続の切断を行います。

トロン協会が規定する仕様では、TCP の通信端点はメンバとして属性、送信ウィンドウの先頭アドレス、送信ウィンドウのサイズ、受信ウィンドウの先頭アドレス、受信ウィンドウのサイズ、コールバックルーチンのアドレスを持っています。

TCP では、接続が完全に切断されている状態の通信端点を未使用状態と呼びます。

トロン協会が規定する仕様では、UDP の通信端点はメンバとして属性、自局の IP アドレス、自局のポート番号、コールバックルーチンのアドレスを持っています。

### 1.4.3 タイムアウト

API をコールすると処理に時間がかかり、関数のなかで待ち状態になることがあります。この時、指定した時間（タイムアウト時間）を過ぎても処理が終了しない場合、処理をキャンセルして API からリターンすることができます。この現象をタイムアウトと呼びます。

### 1.4.4 ノンブロッキングコール、コールバック

API のなかで待ち状態となった場合、処理を継続したまま API からリターンすることをノンブロッキングコールと呼びます。そして API からリターンした後、継続された処理の終了をアプリケーションに通知する機能をコールバックと呼びます。.

### 1.4.5 ポーリング

タイムアウト時間を 0 に設定したタイムアウト処理と同じ動作をします。

### 1.4.6 Pending

API のなかで待ち状態になっていることをペンディングしていると呼びます。

### 1.4.7 MSS (Maximum Segment Size)

MSS は、TCP のプロトコル処理において、一回に送信できるセグメントの最大サイズのことです。システムで一意に決定します（※T4 でも MSS を定義する必要があります）。アプリケーションプログラムから API で渡された送信データが MSS より大きなサイズの場合、TCP のプロトコル処理部は MSS のサイズで分割して複数のセグメントで送信します。分割の対象となるのは、TCP ヘッダを除いた純粋なデータ部分です。

TCP では、通信の初期段階で通信相手と自局の MSS 情報を交換し、両者の内、小さい MSS で送信するようにします。このときの MSS は、TCP のヘッダのオプション部に置かれます。したがってこの機能を「ヘッダオプションの MSS」と表現します。

※T4 はこの機能をサポートしています。

### 1.4.8 IP フラグメント

IP プロトコル処理のオプション機能の 1 つで、ルータが MTU（ネットワーク最大転送単位）の小さな通信路を使用する場合、MTU よりサイズが大きなデータを送信するために、データを MTU 以下のサイズに分割して送信し、最終的に受信した側で再構築する仕組みのことです。例えば Ethernet では、MTU が 1500bytes のため、これ以上のデータを送信する場合、分割が必要となります。

※T4 では、この機能をサポートしていません。従って、MTU の小さな通信路を使って通信する可能性のある場合は、MSS を MTU より小さなサイズに設定する必要があります。

### 1.4.9 受信(送信)ウィンドウ

受信したデータを溜めておくためのバッファ領域のことです。TCP では、通常、セグメントを送信する度に相手からの確認応答を待ちますが、毎回、確認応答を待っていたのではオーバーヘッドが生じます。したがって通信の際に TCP のヘッダを使って受信側から受信ウィンドウの残りサイズを送信側に通知し、送信側は受信ウィンドウが一杯になるまでは、確認応答を待たずにセグメントを連続して送信するようにします。

通信中に受信ウィンドウが一杯になると、その通知を受けた送信側は送信を止め、受信側から受信ウィンドウが空いたことを通知してくるのを待ちます。

T4 では、2 回送信する毎に受信側の確認応答を待ってから次の送信を再開します。

### 1.4.10 PPP (Point to Point Protocol)

シリアル回線上に IP プロトコルを載せるためのプロトコルです。付随したプロトコルとして回線接続後のユーザ認証手順があります。

### 1.4.11 PAP (Password Authentication Protocol)

PPP 接続時の認証プロトコルのひとつです。発信側が送信したパスワードの正誤を認証します。パスワードは、サーバ側では暗号化されていますが、通信回線上では暗号化されません。同様のプロトコルに CHAP がありますが、こちらはパスワードを暗号化するため、PAP に比べ信頼性が高いです。

### 1.4.12 ICMP (Internet Control Message Protocol)

IP パケットの配送に伴うエラーを通知したり、各ホストの IP の状態を調べたり通知したりするためのプロトコルで、IP と常に一緒に使われます。

### 1.4.13 ARP (Address Resolution Protocol)

IP アドレスから MAC アドレス（例えば、Ethernet アドレス）に変換するためのプロトコルです。

## 1.5 プログラム開発手順

T4 を使用したアプリケーションプログラムの開発フローを  
図 2 に示します。

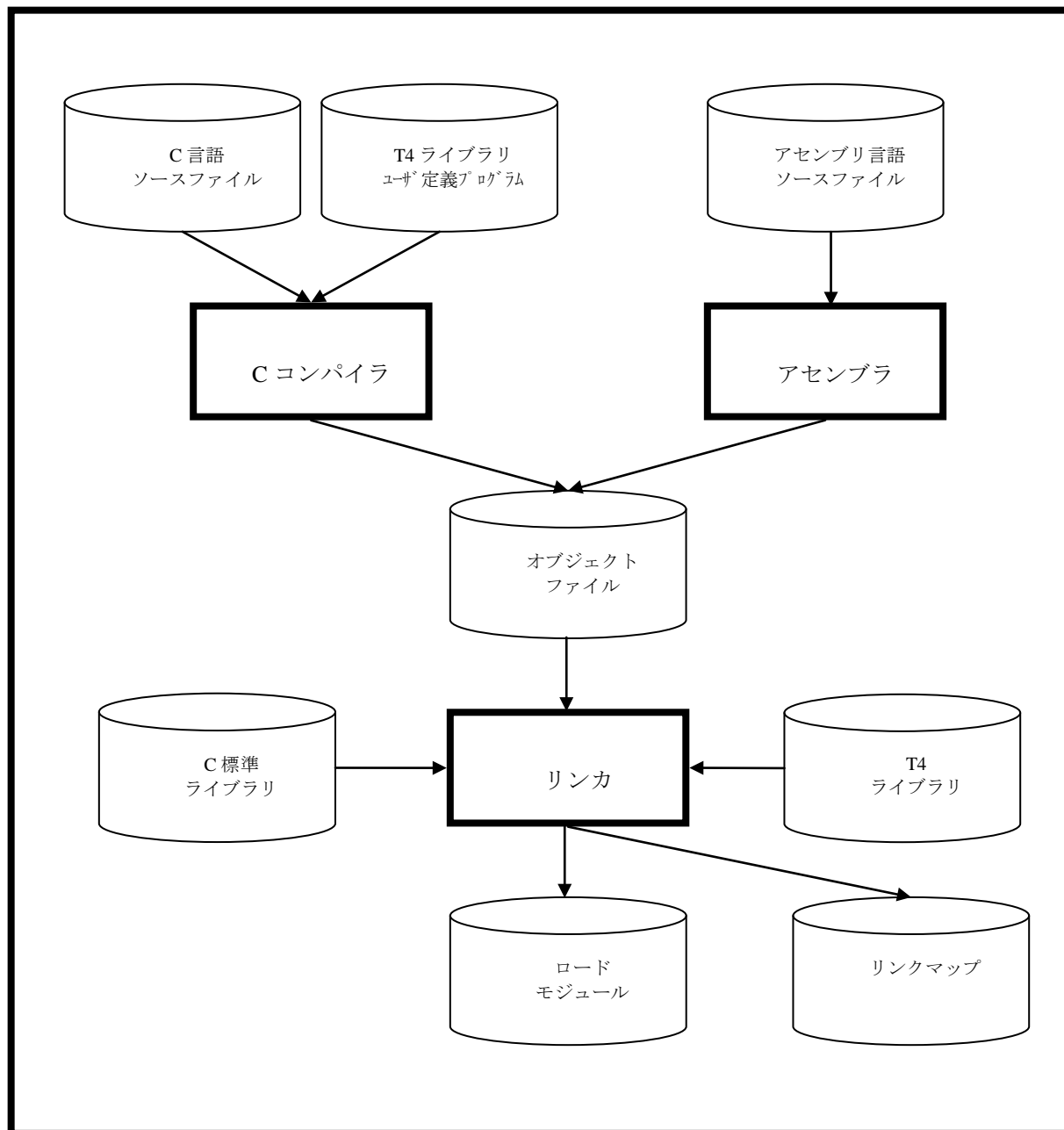


図 2 アプリケーションプログラムの開発フロー

## 2. T4 の概要

### 2.1 製品構成

本製品には、表 1 のものが含まれています。.

表 1. T4 製品構成

	内容
ドキュメント (doc)	
rzzz1_t4tiny.pdf	ユーザーズマニュアル ( 本書 )
rzzz2_t4tiny.pdf	Ethernet ドライバ・インタフェース仕様書
rzzz3_t4tiny.pdf	PPP ドライバ・インタフェース仕様書
rzzz4_xxx_t4.pdf	導入ガイド
T4 Library (lib)	
T4_Library_xxx_ether_yyy.lib	ライブラリファイル (for Ethernet)
T4_Library_xxx_ppp_yyy.lib	ライブラリファイル (for PPP)
r_t4_itcpip.h	ヘッダファイル
ライブラリ生成環境 (make_lib)	
make_lib.zip	ライブラリ生成環境(ソースコード入り)

xxx : CPU name (ex. rx600, h8s2600, sh2a) yyy : CPU settings (ex. big, little, advanced)

zzz1-zzz4 : document name

#### 2.1.1 ドキュメント

T4 を使用するために必要な情報を記載した資料一式です。導入ガイドにはマイコンに依存している情報やバージョン情報、更新履歴などを記載します。

#### 2.1.2 T4 ライブラリ

- T4 ライブラリファイル (バイナリファイル)

API 関数や各種プロトコル処理のプログラムを含みます。アプリケーションプログラムと共にリンクして使用します。

Ethernet をお使いになる場合は T4\_Library\_xxx\_ether\_yyy.lib を、PPP の場合は T4\_Library\_xxx\_ppp\_yyy.lib をお使いください。

※ xxx には対応する CPU の種類、yyy には CPU 設定が入ります。

xxx ( rx600, h8s2600, sh2a 等 ) : yyy (big, little, advanced 等)

※ CPU 毎に Ethernet、PPP の対応状況は異なります

- T4 ヘッダファイル (r\_t4\_itcpip.h)

T4 の API のプロトタイプ宣言やマクロ定義等が記述されています。T4 の API を呼び出すアプリケーションプログラムでは、このヘッダファイルをインクルードしてください。

#### 2.1.3 サンプルプログラム

アプリケーションプログラム (Telnet 使用) のサンプルソースを用意しています。アプリケーションプログラムの動作方法は、9章を参照ください。

このサンプルプログラムは、物理層のサンプルドライバのソースコード(Ethernet またはシリアル)を含みます。ユーザシステムに合わせて適宜変更して下さい。

## 2.2 ライブラリの概略仕様

T4 の概略仕様を表 2 に示します。本ライブラリは TCP、UDP、IP、ICMP、ARP、PPP のプロトコルをサポートしています。API は、ITRON TCP/IP API 仕様に準拠しています。

表 2. プロトコルごとの概略仕様

Protocol	Item	Specification
TCP	API	ITRON TCP/IP API 仕様に準拠
	ノンブロッキングコール	サポート
	コールバック機能	サポート
	キューイング	未サポート
	最大サイズ※	データセグメント長(データ長):1480(1460)Byte
	TCP のヘッダオプション	MSS のみサポート 受信: MSS 以外のヘッダオプションは無視 送信: MSS 以外のヘッダオプションは不可
	受付口数	最大 30
	通信端点数	最大 30
UDP	API	ITRON TCP/IP API 仕様に準拠
	ノンブロッキングコール	サポート
	コールバック機能	サポート
	キューイング機能	未サポート
	最大サイズ※	UDP データグラム長(データ長): 1480(1472)Byte
	通信端点数	最大 30
	マルチキャスト	受信のみサポート (224.0.0.0~239.255.255.255) ローカルエリアへの送信のみサポート (IGMP 未対応)
IP	バージョン	IPv4 (version 4) のみ
	フラグメント	未サポート 受信: フラグメント化されたデータグラムは破棄 送信: データグラムのフラグメント化は不可
	ヘッダオプション	未サポート 受信: ヘッダオプションが含まれるデータグラムは破棄 送信: ヘッダオプションは不可
	最大サイズ※	IP データグラム長(データ長): 1,500 (1,480) bytes
ICMP	メッセージタイプ	エコー応答のみサポート 受信: エコー要求のみ、他のメッセージは破棄 送信: エコー応答のみ
ARP	キャッシュエントリー数	ユーザ定義による
	キャッシュ保持期間	約 10 分
PPP	サーバ/クライアント	クライアント/サーバ機能サポート
	最大サイズ※	PPP フレーム ( データ長 ): 1,504 (1,500) bytes
	認証方式	PAP CHAP with MD5
	圧縮オプション	圧縮関連オプション・未サポート プロトコルフィールド圧縮、 アドレスと制御フィールド圧縮、 TCP/IP ヘッダ圧縮の設定要求を拒否

※TCP と UDP と IP と PPP の最大サイズはドライバの送受信バッファサイズに依存します。

※Ethet は複数チャネル対応、PPP は単一チャネルのみ対応

### 3. T4 Library マクロ定義

このセクションは、ライブラリで使われる定義について、詳細を伝えます。

Datatype	Typedef
signed char	int8_t
unsigned char	uint8_t
signed short	int16_t
unsigned short	uint16_t
signed long	int32_t
unsigned long	uint32_t
signed char	B
signed short	H
signed long	W
unsigned char	UB
unsigned short	UH
unsigned long	UW
signed char	VB
signed short	VH
signed long	VW
void far	*VP
void	(*FP)();
signed long	INT
unsigned long	UINT
signed short	ID
signed short	PRI
signed long	TMO
signed short	HNO
signed long	ER
unsigned short	ATR
signed long	FN

## 4. T4 ライブラリ構造体

このセクションは、ライブラリで使用される構造体の詳細を伝えます。

ユーザはグレイアウトしているメンバは使用する必要はありません。

### 4.1 TCP オブジェクト構造体

T4 ライブラリ用環境変数: TCPUDP\_ENV

Datatype	構造体要素	説明
UB	ipaddr[4]	自局 IP アドレス
UB	maskaddr[4]	サブネットマスク
UB	gwaddr[4]	ゲートウェイアドレス

IP アドレスとポート番号: T\_IPV4EP

Datatype	構造体要素	説明
UW	ipaddr	IP アドレス
UH	portno	ポート番号

TCP 受付口: T\_TCP\_CREP

Datatype	構造体要素	説明
ATR	repatr	TCP 受付口の属性
T_IPV4EP	myaddr	自局の IP アドレスとポート番号

TCP 通信端点: T\_TCP\_CCEP

Datatype	構造体要素	説明
ATR	cepatr	TCP 通信端点属性 (LAN ポートの番号: 0 ~ LAN ポート数-1 LAN ポート数については 6.1 章参照)
VP	sbuf	送信用ウィンドウバッファの先頭アドレス
INT	sbufsz	送信用ウィンドウバッファのサイズ
VP	rbuf	受信用ウィンドウバッファの先頭アドレス
INT	rbufsz	受信用ウィンドウバッファのサイズ
ER	(*callback)(ID cepid, FN fncd, VP p_parblk)	コールバックルーチン

### 4.2 UDP オブジェクト構造体

UDP 通信端点: T\_UDP\_CCEP

Datatype	構造体要素	説明
ATR	cepatr	UDP 通信端点属性 (LAN ポートの番号: 0 ~ LAN ポート数-1 LAN ポート数については 6.1 章参照)
T_IPV4EP	myaddr	自局の IP アドレスとポート番号
ER	(*callback)(ID cepid, FN fncd, VP p_parblk)	コールバックルーチン

## 5. T4 ライブラリ定数

このセクションは、ライブラリで使用する定数の詳細を伝えます。

### 5.1 TCP 機能の戻り値

名称	値	意味
TFN_TCP_ACP_CEP	-0x205	TCP 接続要求待ち
TFN_TCP_CON_CEP	-0x206	TCP 接続要求(能動要求)
TFN_TCP_SHT_CEP	-0x207	TCP データ送信の終了
TFN_TCP_CLS_CEP	-0x208	TCP 通信端点のクローズ
TFN_TCP_SND_DAT	-0x209	TCP データの送信
TFN_TCP_RCV_DAT	-0x20A	TCP データの受信
TFN_TCP_ALL	0	TCP 関連の全 API 指定

### 5.2 UDP 機能の戻り値

名称	値	意味
TFN_UDP_SND_DAT	-0x223	UDP データの送信
TFN_UDP_RCV_DAT	-0x224	UDP データの受信 (udp_rcv_dat() コール後)
TEV_UDP_RCV_DAT	0x221	UDP データの受信 (udp_rcv_dat() コール前)
TFN_UDP_ALL	0	UDP 関連の全 API 指定

### 5.3 API で使用されているエラーコード

名称	Value	意味
E_OK	0	正常
E_NOSPT	-9	指定された機能は未サポート
E_PAR	-17	パラメータエラー
E_OBJ	-41	オブジェクト状態エラー
E_QOVR	-43	キューイングオーバーフロー
E_WBLK	-57	ノンブロッキングコール受付
E_TMOUT	-50	タイムアウト
E_RLWAI	-49	API キャンセル処理完了
E_CLS	-52	接続失敗
E_BOVR	-58	バッファオーバーフロー

### 5.4 タイムアウト指定

名称	Value	意味
TMO_POL	0	ポーリング
TMO_FEVR	-1	永久待ち
TMO_NBLK	-2	ノンブロッキングコール

### 5.5 特殊な IP アドレスとポート番号

名称	Value	意味
TCP_PORTANY	0	TCP ポート番号の指定省略
NADR	0	無効アドレス



## 6. T4 コンフィグレーションファイル

使用する LAN ポート数、TCP 受付け、TCP 通信端点、UDP 通信端点、IP ヘッダ関連、TCP 関連オプション、自局の設定 (Ethernet、PPP) を定義します。これらの定義については、T4 の初期化よりも前に設定するのであれば、アプリケーションプログラム中でも可能です。ただし、使用状況にかかわらず、全ての項目を設定する必要があります。

本製品のコンフィグレーションファイル `config_tcpudp.c` を用いて、  
次の各定義について説明します。

## 6.1 LAN ポート/シリアルポート数の定義

```
/* **** */
/* **** General definition **** */
/* **** */
/* Number of LAN port, Number of Serial port */
const UB _t4_channel_num = 1; /* If user uses PPP, this value can be set "1" only.
*/
```

図 3. LAN ポート/シリアルポート数の定義(config\_tcpudp.c より抜粋)

変数 `_t4_channel_num` には、LAN ポート数の値(=1)を設定してください。

変数 `_t4_channel_num` には、1 以外を設定しないでください。

## 6.2 TCP 受付口の定義

```

/*****
/***** TCP-related definition *****/
/*****

/** Definition of TCP reception point (only port number needs to be set) */
T_TCP_CREP tcp_crep[] =
{
/* { 受付口属性, {自局の IP アドレス, 自局のポート番号}} */
  { 0, { 0, 1024 }}, /*TCP 受付口 ID:1、ポート番号:1024 */
  { 0, { 0, 1025 }}, /*TCP 受付口 ID:2、ポート番号:1025 */
};

/* Total number of TCP reception points */
const H __tcpcrepn = sizeof(tcp_crep)/sizeof(T_TCP_CREP); /* TCP 受付口の総数 */

```

図 4. TCP 受付口の定義(config\_tcpudp.c より抜粋)

TCP 受付口は、T\_TCP\_CREP構造体（第4章参照）を用いて静的に生成します。図 4に、TCP 受付口の定義の記述例(受付口が 2 の場合)を示します。

現在、これら 3 つのフィールドの内、自局のポート番号のみ設定が必要です。

属性は、未サポートのため設定値は無効となります。

自局の IP アドレスについては、T4 の初期化において、変数 `tcpudp_env` に設定された自局の IP アドレスが設定されるため、TCP 受付口の定義で設定された自局の IP アドレスは無効となります。

TCP 受付口は複数設定可能です。複数設定した場合、必要 RAM サイズが増加します。

また、異なる通信端点で同じ受付口(受付ポート番号) を使用可能です。

変数 `__tcpcrepn` には、TCP 受付口の総数が自動的に設定されるため、変更の必要はありません。

本定義により、指定したポート番号での TCP を用いた接続要求待ちからの接続が可能となります。また、TCP を使用しない場合も定義してください。

## 6.3 TCP 通信端点の定義

```

/*****
/***** TCP-related definition *****/
/*****
:
:
/** Definition of TCP communication end point
(only receive window size needs to be set) */
T_TCP_CCEP tcp_ccep[] =
{
/* { attribute of TCP communication end point,
top address of transmit window buffer, size of transmit window buffer,
top address of receive window buffer, size of receive window buffer,
address of callback routine }

*/
{ 0, 0, 0, 0, 64, callback } /* TCP communication end point ID: 1 */
{ 0, 0, 0, 0, 64, callback } /* TCP communication end point ID: 2 */

};
/* Total number of TCP communication end points */
const H __tcpcepn = sizeof(tcp_ccep)/sizeof(T_TCP_CCEP);

```

図 5. TCP 通信端点の定義 (config\_tcpudp.c より抜粋)

TCP 通信端点は、T\_TCP\_CCEP構造体（第4章参照）を用いて静的に生成します。図 5に、TCP 通信端点の定義の記述例(端点数が2の場合)を示します。

1つのTCP通信端点の定義は、以下で示すように6つのフィールドで構成されています。

{TCP 通信端点属性(LAN ポート番号), 送信ウィンドウの先頭アドレス, 送信ウィンドウのサイズ,  
受信ウィンドウの先頭アドレス, 受信ウィンドウのサイズ,  
コールバックルーチンアドレス(関数名) }

現在、これら6つのフィールドの内、受信ウィンドウのサイズ、コールバックルーチンアドレスの設定が必要です。TCP 通信端点属性(LAN ポート番号)は'0'を設定して下さい。

送信ウィンドウおよびそのサイズは未サポートのため、設定値は無効となります。

受信ウィンドウの先頭アドレスのフィールドについては、T4の初期化で自動的に割り当てられるため、コンフィグレーションファイルでの設定値は無効です。

コールバック機能を使用する場合には、コールバックルーチンアドレスにコールバックルーチンへのアドレスを設定してください。コールバック機能を使用しない場合には、0を設定してください。

TCP 通信端点は複数設定可能です。複数設定した場合、必要 RAM サイズが増加します。

変数\_\_tcpcepn には、TCP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、TCP を用いた接続の確立、データの送受信、接続の切断が可能となります。また、TCP を使用しない場合も定義してください。

## 6.4 UDP 通信端点の定義

```

/*****
/*****  UDP-related definition  *****/
/*****
/** Definition of UDP communication end point */
T_UDP_CCEP udp_ccep[] =
{
    /* Only setting port number */
    { 0, { 0, 1365 }, callback }, /* ID: 1, port number: 1365 */
    { 0, { 0, 1366 }, callback }, /* ID: 2, port number: 1366 */
};
/* Total number of UDP communication end points */
const H __udpcepn = (sizeof(udp_ccep)/sizeof(T_UDP_CCEP));

```

図 6. UDP 関連の設定 (config\_tcpudp.c より抜粋)

UDP 通信端点は、T\_UDP\_CCEP構造体（第4章参照）を用いて静的に生成します。図 6に、UDP 関連の定義の記述例を示します。

1 つの UDP 通信端点の定義は、以下で示すように 4 つのフィールドで構成されています。

{ UDP 通信端点属性(LAN ポート番号), { 自局の IP アドレス , 自局のポート番号 },  
コールバックルーチンアドレス(関数名) }

現在、これら 4 つのフィールドの内、自局のポート番号、コールバックルーチンアドレスの設定が必要です。UDP 通信端点属性(LAN ポート番号)は'0'を設定してください。

自局の IP アドレスは、T4 の初期化において、変数 tcpudp\_env に設定された自局の IP アドレスが設定されるため、コンフィグレーションファイルでの設定値は無効となります。

コールバック機能を使用する場合には、コールバックルーチンアドレスにコールバックルーチンへのアドレスを設定してください。コールバック機能を使用しない場合には、0 を設定してください。

UDP 通信端点は複数設定可能です。複数設定した場合、必要 RAM サイズが増加します。

変数\_\_udpcepn には、UDP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、指定したポート番号における UDP を用いたデータの送受信が可能となります。また、UDP を使用しない場合も定義してください。

## 6.5 IP ヘッダ関連の定義

```
/** TTL for multicast transmission */  
const UB __multi_TTL = 1;
```

図 7. IP ヘッダ関連の定義 (config\_tcpudp.c より抜粋)

変数\_\_multi\_TTLには、マルチキャスト宛に送信する IP データグラムの TTL を設定します。TTL は、IP ヘッダの要素であり、IP データグラムが通過できるルータの数をあらわします。

マルチキャスト宛以外の IP データグラムを送信する場合、IP ヘッダの TTL にはデフォルトの 80 が設定されます。

通常、マルチキャスト宛 IP データグラムのプロトコルは UDP のため、マルチキャスト宛の UDP データの送信時にこの設定値を使用します。ただし、T4 では基本的に API で指定されたパラメータの妥当性をチェックしません。そのため、TCP の接続要求 API で宛先 IP アドレスをマルチキャストアドレスに設定した場合、マルチキャスト宛の不正な TCP パケットが送信されます。TCP では宛先 IP アドレスをマルチキャストアドレスに設定しないでください。

また、マルチキャストは UDP のみで使用してください。TCP では使用出来ません。

## 6.6 TCP オプションの定義

```

/*****
/***** TCP-related definition *****/
/*****
:
:
/** TCP MSS */
const UH _tcp_mss = 64;          /* Maximum: 1,460 bytes */

/** Initial value of sequence number (Set any value other than 0) */
UW _tcp_initial_seqno = 1;

/** 2MSL wait time (unit: 10 ms) */
const UH _tcp_2msl = (1*60*1000/10); /* 1 minute */

/** Maximum value of retransmission timeout period (unit: 10 ms) */
const UH _tcp_rt_tmo_rst = (10*60*1000/10); /* 10 minute */

/** Transmit for delay ack (ON=1/OFF=0) */
UB _tcp_dack = 1;

```

図 8. TCP のオプション定義 (config\_tcpudp.c より抜粋)

図 8 に TCP 関連のオプション設定の記述例を示します。

TCP 関連のオプション設定では、MSS(Maximum Segment Size)のデフォルト値、シーケンス番号の初期値、2MSL 待ち時間、再転送タイムアウトの最大値の設定が必要です。以下に、各々のオプションについて説明します。TCP オプションは全ての端点において共通の設定です。

### 6.6.1 TCP の MSS

変数\_tcp\_mss には、TCP による通信において、ひとつの TCP セグメントで送ることのできる最大データ数 (バイト) を指定します。ヘッダのサイズは含みません。1~1460 までの値を設定することが可能です。それ以外を設定した場合は、RFC によるデフォルトの 536 に設定されます。

この値は TCP 接続時に、TCP ヘッダオプションとして通信相手に送信されます。相手もこのオプションを送信してきた場合、両者のうち小さい方の値を以後この接続における MSS として用います。相手がこのオプションを送信してこなかった場合は、相手がデフォルトの 536 を送信してきたものとして処理します。

ただし、使用する通信端点の受信ウィンドウサイズが\_tcp\_mss よりも小さい場合、受信ウィンドウサイズの値を MSS として相手に送信します。

また、T4 では、IP フラグメントをサポートしていないため、MTU の小さな通信路を使って通信する可能性のある場合は、\_tcp\_mss を MTU より小さなサイズに設定する必要があります。

### 6.6.2 シーケンス番号の初期値

送信する TCP セグメントのシーケンス番号の初期値を設定します。変数 \_tcp\_initial\_seqno に、正の値 (0 は含まない) を設定してください。

また、変数 \_tcp\_initial\_seqno は、必ず RAM 領域に配置してください。

### 6.6.3 2-MSL 待ち時間

TCP の状態遷移において、TIME-WAIT 状態に留まる時間を設定してください。

時間の単位は 10ms です。上記の設定では、

$$(1 \times 60 \times 1000 \div 10) \times 10 \text{ ms} = 60,000 \text{ ms} = 60 \text{ 秒} = 1 \text{ 分}$$

になります。

### 6.6.4 再転送タイムアウトの最大値

TCP の再転送を行う時間を設定します。再転送を繰り返し、最初にセグメントを送信してからの経過時間がこの設定値に達すると、通信相手にリセットセグメントを送信してコネクションを破棄します。時間の単位は 10ms です。上記の設定では、

$$(10 \times 60 \times 1000 \div 10) \times 10 \text{ ms} = 600,000 \text{ ms} = 600 \text{ 秒} = 10 \text{ 分}$$

となります。

再転送までの時間は、指数バックオフのアルゴリズムにより、指数的に長くなります。この再送までの時間の最大値は 60 秒です。再転送までの時間が 60 秒に達すると、それ以降は 60 秒間隔で再転送が繰り返されます。

### 6.6.5 TCP 多重送信の有無

変数\_tcp\_dack には、TCP による送信において、遅延 ACK 受信に対応する TCP 多重送信をするかしないかを指定します。TCP 多重送信をする場合、変数\_tcp\_dack に"1"を指定してください。またこの機能が無効にする場合は変数\_tcp\_dack に"0"を指定してください。

TCP 多重送信をする場合、遅延 ACK が有効になっている通信相手と通信する際に無駄な待ち時間が発生しなくなります。RFC では受信側で ACK を遅らせて送信側にデータをまとめさせる(Nagle アルゴリズム)方法が規程されており、仮に送信側が 1 個だけ TCP パケットを送信する場合に受信側が ACK を遅らせる時間は最大 0.5 秒となります。(Windows の場合 0.2 秒)

T4 では、本機能を有効にした場合、送信データ長と MSS により以下のようにデータを分割した後に ACK を待ちます。送信データ長は送信 API で指定したデータ長です。送信データ長は受信側からの ACK が返る毎に減っていき、0 になると送信 API が完了します。また、受信側のウィンドウサイズを超えてデータを送信する場合も同様にデータ分割されます。

MSS バイトを 2 回送信して ACK 待ち :

(送信データ長  $\geq$  MSS $\times$ 2)

MSS バイトを 1 回、(送信データ長-MSS)バイトを 1 回送信して ACK 待ち :

(MSS $\times$ 2 > 送信データ長 > MSS)

(送信データ長-1)バイトを 1 回、1 バイトを 1 回送信して ACK 待ち :

MSS  $\geq$  送信データ長 > 1

1 バイトを 1 回送信して ACK 待ち :

送信データ長 = 1.

※送信 API で指定するデータ長は MSS の 2 倍を指定すると効率よくデータ送信ができます。

※\_tcp\_dack 値を"0"にすると、T4 はデータ送信毎に ACK を待つようになります。

※\_tcp\_dack 値は TCP 接続が確立している状態で変更しないでください。



## 6.7 UDP オプションの定義

```
/** Behavior of UDP zero checksum */  
const UB _udp_enable_zerochecksum = 0; /* 0 = disable, other = enable  
*/
```

図 9. UDP オプションの定義 (config\_tcpudp.c より抜粋)

### 6.7.1 UDP ゼロチェックサムの動作定義

UDP パケットを受信した際に、チェックサム値がゼロだった場合、エラーパケットとして扱い、破棄するかどうかを決める変数です。

UDP の仕様として、受信側は UDP チェックサム値にゼロが格納されている場合、そのチェックサム値をチェックしません。また、送信側が送信データから演算したチェックサム値がゼロになった場合はチェックサム値に(0xffff)を格納します。このとき、受信側はそのチェックサム値を 0 として扱いチェックサム値をチェックします。

この変数をゼロにした場合(デフォルト)、受信した UDP チェックサム値がゼロのときチェックサム演算を実行せず、正常パケットとして扱います。

この変数をゼロ以外にした場合、受信した UDP チェックサム値がゼロのときチェックサム演算を実行し、エラーパケットとして扱い、破棄します。

## 6.8 自局の設定 (Ethernet)

```

/*****
/***** IP-related definition *****/
/*****
const UH _ip_tblcnt = 3;
#define MY_IP_ADDR 192,168,0,3 /* Local IP address */
#define GATEWAY_ADDR 0,0,0,0 /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK 255,255,255,0 /* Subnet mask */
:
:
TCPUDP_ENV tcpudp_env = {
    {MY_IP_ADDR}, /* IP address */
    {SUBNET_MASK}, /* Subnet mask */
    {GATEWAY_ADDR} /* Gateway address */
};
/*****
/***** Driver-related definition *****/
/*****
/*-----*/
/* Set of Ethernet-related */
/*-----*/
/* Local MAC address (Set all 0s when unspecified) */
#define MY_MAC_ADDR 0x20,0x00,0x00,0x00,0x00,0x00

UB _myethaddr[6]={MY_MAC_ADDR}; /* MAC address */

```

図 10. Ethernet 使用時の自局の設定 (config\_tcpudp.c より抜粋)

Ethernet を使用する場合の自局に関する情報として、IP アドレス、サブネットマスク、ゲートウェイアドレス、イーサネットアドレス (MAC アドレス) を定義します。図 10 に記述例を示します。

自局の設定は、以下で示す 3 つのフィールドを含む TCPUDP\_ENV 構造体 (第 4 章参照) とイーサネットアドレス (MAC アドレス) で構成されています。

```

tcpudp_env = { IP アドレス , サブネットマスク , ゲートウェイアドレス }
_myethaddr = { MAC アドレス }

```

各情報は、図 10 に示すように define 定義を用いることもできます。各情報の値は、コンマで区切って指定してください。

IP アドレス、サブネットマスクの設定は必須です。

ゲートウェイアドレスは、使用する場合にはそのアドレスを、そうでない場合には {0, 0, 0, 0} と、全て 0 を設定してください。ゲートウェイアドレスの設定は 1 つのみ可能です。

MAC アドレスには、プログラムにより Ethernet コントローラに対して MAC アドレスを指定する場合にはそのアドレスを、EEPROM 等により Ethernet コントローラの MAC アドレスを自動設定する場合には {0, 0, 0, 0, 0, 0} と、全て 0 を設定してください。

また、変数 tcpudp\_env、変数\_myethaddr は、必ず RAM 領域に配置してください。

変数\_ip\_tblcnt には、T4 で使用する ARP のキャッシュエントリ数を設定します。T4 では通信するホスト 1 台につき、1 つのキャッシュエントリを使用します。

「同時に通信する可能性のあるホスト数+1」以上の値を推奨します。「同時に通信する可能性のあるホスト数」よりも小さな値を設定した場合、動作は不定です。例えば、TCP、UDP を同時に使用し、それぞれ別ホストと通信する場合、3 以上の値を設定するようにしてください。

また、キャッシュエントリの保持期間は 10 分のため、その間に、頻繁に多数のホストと通信する場合、ホスト数以上の値を設定すると通信効率が上がります。例えば、4 つのホストと順に UDP で通信する場合、4 以下のエントリ数を設定すると毎回 ARP の解決が必要となります。

## 6.9 自局の設定(PPP)

```

/*****
/*****      PPP definition      *****/
/*****
UB ppp_mode = PPP_MODE_CLIENT;
//UB ppp_mode = PPP_MODE_SERVER;

const UH ppp_auth = AUTH_PAP;
/* PAP = AUTH_PAP, CHAP with MD5 = AUTH_CHAP, NONE = AUTH_NON */

/*****
/***** Driver-related definition ( for PPP server) *****/
/*****
UB user1_name[] = "abcde";
UB user1_passwd[] = "abc00";
UB user2_name[] = "xxx";
UB user2_passwd[] = "yyy";
UB *user_table[] = {          /* user authentication data table */
    user1_name, user1_passwd,
    user2_name, user2_passwd,
};
H __usern = sizeof(user_table)/sizeof(UB*)/2;    /* Number of user */
UB client_ip_address[] = {192,168,0,100}; /* IP address that PPP server allocates
*/
UB serverName[] = "T4 PPP SERVER";
UB serverName_len = sizeof(serverName)-1;

/*****
/***** Driver-related definition ( for PPP client) *****/
/*****
UB user_name[6] = "abcde";      /* user name */
UB user_passwd[6] = "abc00";    /* password */

/* Dial up-related setting */
const UB peer_dial[] = "0,123"; /* Destination telephone number */
const UB at_commands[] = "ATW2S0=2&C0&D0&S0M0X3"; /* Modem initialization command
*/

```

図 11. PPP 使用時の自局の設定 (config\_tcpudp.c より抜粋)

### 6.9.1 自局の設定(PPP クライアント)

PPP を使用する場合の自局に関する情報として、IP アドレス、PAP 関連、ダイヤルアップ関連を定義します。図 11 に記述例を示します。

IP アドレスの設定には、Ethernet の場合と同様に、以下に示す TCPUDP\_ENV 構造体（第4章参照）を用います。サブネットマスクおよびゲートウェイアドレスには全て 0 を設定してください。

```
tcpudp_env = { IP アドレス, サブネットマスク, ゲートウェイアドレス }
```

図 11 に示すように define 定義を用いることもできます。値は、コンマで区切って指定してください。

また、変数 tcpudp\_env は、必ず RAM 領域に配置してください。

T4 を PPP クライアントとして使用し、IP アドレスを PPP サーバで割り当てるように要求する場合、図 11 のように自局の IP アドレスに {0, 0, 0, 0} と、全て 0 を設定します。ppp\_mode 変数には PPP\_MODE\_CLIENT をセットします。また、特定の IP アドレスを PPP サーバに対して要求する場合には、要求する IP アドレスを設定します。ただし、PPP サーバが PPP クライアントの要求した IP アドレスを許可しなかった場合、IP アドレスは PPP サーバが割り当てた IP アドレスとなります。

PPP サーバから IP アドレスが割り当てられる場合、変数 tcpudp\_env の IP アドレスは PPP サーバが割り当てた IP アドレスに更新されます。

サブネットマスクは、本来 PPP において意味を持ちませんが、T4 ではサブネットマスクを用いて、自局と通信相手が同一ネットワークかどうかをチェックしています。そのため、サブネットマスクが全て 0 の場合は正常に通信できますが、それ以外の場合、API にエラーが返り、通信できないことがあります。

認証方式として PAP 及び CHAP を使用する場合、認証方式、ユーザ名とパスワードの設定が必要です。変数 ppp\_auth に AUTH\_PAP または AUTH\_CHAP\_MD5、変数 user\_name にユーザ名、変数 user\_passwd にパスワードを設定してください。認証を用いない場合には、変数 ppp\_auth に AUTH\_NON を設定してください。この場合、変数 user\_name, user\_passwd への値の設定は不要です(設定値は無視されます)。

### 6.9.2 自局の設定(PPP サーバ)

ここでは PPP クライアントとの違いのみ説明します。

T4 を PPP サーバとして用いる場合、ppp\_mode 変数には PPP\_MODE\_SERVER をセットします。T4 は client\_ip\_address 変数に設定された IP アドレスを PPP クライアントに割り当てます。(同時に接続可能な PPP クライアントは 1 個です)

認証方式として PAP 及び CHAP を使用する場合、認証方式、ユーザ名とパスワードの設定が必要です。変数 ppp\_auth に AUTH\_PAP または AUTH\_CHAP、変数 user\_table 変数のユーザ名(user\_name)とパスワード(passwd)を設定してください。認証を用いない場合には、変数 ppp\_auth に AUTH\_NON を設定してください。この場合、変数 user\_table への値の設定は不要です(設定値は無視されます)。

### 6.9.3 モデムコマンドの設定(PPP サーバ/PPP クライアント共通)

ダイヤルアップ関連の定義として、宛先の電話番号と、モデムに対する初期化コマンドの設定が必要です。宛先の電話番号は変数 peer\_dial に、モデムの初期化コマンドは変数 at\_commands に設定してください。また、モデムの初期化コマンドについては、モデムのマニュアルを参照ください。

## 7. T4 コンフィグレーションファイル(複数 LAN ポート)

使用する LAN ポート数、TCP 受付け、TCP 通信端点、UDP 通信端点、IP ヘッダ関連、TCP 関連オプション、自局の設定 (Ethernet、PPP) を定義します。これらの定義については、T4 の初期化よりも前に設定するのであれば、アプリケーションプログラム中でも可能です。ただし、使用状況にかかわらず、全ての項目を設定する必要があります。

本製品のコンフィグレーションファイル `config_tcpudp.c` を用いて、  
次の各定義について説明します。

PPP を使用する場合は 6 章を参照してください。

## 7.1 LAN ポート数の定義

```
/* **** */
/* **** General definition **** */
/* **** */
/* Number of LAN port, Number of Serial port */
const UB _t4_channel_num = 2; /* If user uses PPP, this value can be set "1" only.
*/
```

図 12. LAN ポート数の定義(config\_tcpudp.c より抜粋)

変数 `t4_channel_num` には、使用する LAN ポート数の値を設定してください。

変数 `t4_channel_num` には、0 を設定しないでください。

## 7.2 TCP 受付口の定義

```

/*****
/*****          TCP 関連の定義          *****/
/*****

/**** TCP 受付口の定義 (ポート番号のみ設定要) ****/
T_TCP_CREP tcp_crep[] = {
/* { 受付口属性, { 自局の IP アドレス, 自局のポート番号} } */
  { 0, { 0, 1024 } }, /*TCP 受付口 ID:1、ポート番号:1024 */
  { 0, { 0, 1025 } }, /*TCP 受付口 ID:2、ポート番号:1025 */
  { 0, { 0, 1026 } }, /*TCP 受付口 ID:3、ポート番号:1026 */
  { 0, { 0, 1027 } }, /*TCP 受付口 ID:4、ポート番号:1027 */
};

/* Total number of TCP reception points */
const H __tcpcrepn = sizeof(tcp_crep)/sizeof(T_TCP_CREP); /* TCP 受付口の総数 */

```

図 13. TCP 受付口の定義(config\_tcpudp.c より抜粋)

TCP 受付口は、T\_TCP\_CREP構造体（第4章参照）を用いて静的に生成します。図 13 に、TCP 受付口の定義の記述例(受付口が 4 の場合)を示します。

オプションの各変数の設定値の詳細については 6.2 章を参照してください。



## 7.3 TCP 通信端点の定義

```

/*****
/***** TCP-related definition *****/
/*****
:
:
/** Definition of TCP communication end point
(only receive window size needs to be set) */
T_TCP_CCEP tcp_ccep[] =
{
/* { attribute of TCP communication end point,
top address of transmit window buffer, size of transmit window buffer,
top address of receive window buffer, size of receive window buffer,
address of callback routine }
*/
{ 0, 0, 0, 0, 64, callback } /* TCP communication end point ID: 1 */
{ 1, 0, 0, 0, 64, callback } /* TCP communication end point ID: 2 */
{ 0, 0, 0, 0, 64, callback } /* TCP communication end point ID: 3 */
{ 1, 0, 0, 0, 64, callback } /* TCP communication end point ID: 4 */
};
/* Total number of TCP communication end points */
const H __tcpcepn = sizeof(tcp_ccep)/sizeof(T_TCP_CCEP);

```

図 14. TCP 通信端点の定義 (config\_tcpudp.c より抜粋)

TCP 通信端点は、T\_TCP\_CCEP構造体（第4章参照）を用いて静的に生成します。図 14 に、TCP 通信端点の定義の記述例(端点数が 4 の場合)を示します。

1 つの TCP 通信端点の定義は、以下で示すように 6 つのフィールドで構成されています。

{TCP 通信端点属性(LAN ポート番号), 送信ウィンドウの先頭アドレス, 送信ウィンドウのサイズ,  
受信ウィンドウの先頭アドレス, 受信ウィンドウのサイズ,  
コールバックルーチンアドレス(関数名) }

現在、これら 6 つのフィールドの内、TCP 通信端点属性(LAN ポート番号)、受信ウィンドウのサイズ、コールバックルーチンアドレスの設定が必要です。

送信ウィンドウおよびそのサイズは未サポートのため、設定値は無効となります。

TCP 通信端点属性(LAN ポート番号)のフィールドに、使用する LAN ポートの番号を設定してください。この値は Ethernet ドライバ・インタフェースの各関数の引数に渡されます。Ethernet ドライバではこの値を確認し、適切な LAN ポートを使用してください。

この値の上限値は、「LAN ポート数-1」です。

受信ウィンドウの先頭アドレスのフィールドについては、T4 の初期化で自動的に割り当てられるため、コンフィグレーションファイルでの設定値は無効です。

コールバック機能を使用する場合には、コールバックルーチンアドレスにコールバックルーチンへのアドレスを設定してください。コールバック機能を使用しない場合には、0 を設定してください。

TCP 通信端点は複数設定可能です。複数設定した場合、必要 RAM サイズが増加します。

変数\_\_tcpcepn には、TCP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、TCP を用いた接続の確立、データの送受信、接続の切断が可能となります。また、TCP を使用しない場合も定義してください。

## 7.4 UDP 通信端点の定義

```

/*****
/*****  UDP-related definition  *****/
/*****
/**** Definition of UDP communication end point ****/
T_UDP_CCEP udp_ccep[] =
{
    /* Only setting port number */
    { 0, { 0, 1365 }, callback }, /* ID: 1, port number: 1365 */
    { 1, { 0, 1366 }, callback }, /* ID: 2, port number: 1366 */
};
/* Total number of UDP communication end points */
const H __udpcepn = (sizeof(udp_ccep)/sizeof(T_UDP_CCEP));

```

図 15. UDP 関連の設定 (config\_tcpudp.c より抜粋)

UDP 通信端点は、T\_UDP\_CCEP構造体（第4章参照）を用いて静的に生成します。図 15 に、UDP 関連の定義の記述例を示します。

1 つの UDP 通信端点の定義は、以下で示すように 4 つのフィールドで構成されています。

{UDP 通信端点属性(LAN ポート番号), { 自局の IP アドレス , 自局のポート番号 } ,  
コールバックルーチンアドレス(関数名) }

現在、これら 4 つのフィールドの内、UDP 通信端点属性(LAN ポート番号)、自局のポート番号、コールバックルーチンアドレスの設定が必要です。

UDP 通信端点属性(LAN ポート番号)のフィールドに設定できる値は、使用する LAN ポートの番号を設定してください。この値は Ethernet ドライバ・インタフェースの各関数の引数に渡されます。Ethernet ドライバではこの値を確認し、適切な LAN ポートを使用してください。

この値の上限値は、「LAN ポート数-1」です。

自局の IP アドレスは、T4 の初期化において、変数 tcpudp\_env に設定された自局の IP アドレスが設定されるため、コンフィグレーションファイルでの設定値は無効となります。

コールバック機能を使用する場合には、コールバックルーチンアドレスにコールバックルーチンへのアドレスを設定してください。コールバック機能を使用しない場合には、0 を設定してください。

UDP 通信端点は複数設定可能です。複数設定した場合、必要 RAM サイズが増加します。

変数\_\_udpcepn には、UDP 通信端点の総数が自動的に設定されるため、変更の必要はありません。

本定義により、指定したポート番号における UDP を用いたデータの送受信が可能となります。また、UDP を使用しない場合も定義してください。

## 7.5 IP ヘッダ関連の定義

```
/** TTL for multicast transmission */  
const UB __multi_TTL[] = { 1, 1};
```

図 16. IP ヘッダ関連の定義 (config\_tcpudp.c より抜粋)

IP ヘッダ定義の設定値の詳細については 6.5 章を参照してください。

## 7.6 TCP オプションの定義

```

/*****
/***** TCP-related definition *****/
/*****
:
:
/** TCP MSS */
const UH _tcp_mss[] = {64, 64};          /* Maximum: 1,460 bytes */

/** Initial value of sequence number (Set any value other than 0) */
UW _tcp_initial_segno[] = { 1, 1};

/** 2MSL wait time (unit: 10 ms) */
const UH _tcp_2msl[] = {(1*60*1000/10), (1*60*1000/10)};          /* 1 minute */

/** Maximum value of retransmission timeout period (unit: 10 ms) */
const UH _tcp_rt_tmo_rst[] = {(10*60*1000/10), (10*60*1000/10)}; /* 10 minute */

/** Transmit for delay ack (ON=1/OFF=0) */
UB _tcp_dack[] = {1, 1};

```

図 17. TCP のオプション定義 (config\_tcpudp.c より抜粋)

図 17 に TCP 関連のオプション設定の記述例を示します。

オプションの各変数の設定値の詳細については 6.6 章を参照してください。

## 7.7 UDP オプションの定義

```
/** Behavior of UDP zero checksum */  
const UB _udp_enable_zerochecksum[] = {0, 0}; /* 0 = disable, other = enable */
```

図 18. UDP オプションの定義 (config\_tcpudp.c より抜粋)

### 7.7.1 UDP ゼロチェックサムの動作定義

変数の設定値の詳細については 6.7.1 章を参照してください。

## 7.8 自局の設定 (Ethernet)

```

/*****
/***** IP-related definition *****/
/*****
const UH _ip_tblcnt[] = {3,3};
#define MY_IP_ADDR0 192,168,0,3 /* Local IP address */
#define GATEWAY_ADDR0 0,0,0,0 /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK0 255,255,255,0 /* Subnet mask */

#define MY_IP_ADDR1 192,168,0,10/* Local IP address */
#define GATEWAY_ADDR1 0,0,0,0 /* Gateway address (invalid if all 0s)*/
#define SUBNET_MASK1 255,255,255,0 /* Subnet mask */
:
:
TCPUDP_ENV tcpudp_env[] =
{
    {{MY_IP_ADDR0},{SUBNET_MASK0},{GATEWAY_ADDR0}},
    {{MY_IP_ADDR1},{SUBNET_MASK1},{GATEWAY_ADDR1}}
};
/*****
/***** Driver-related definition *****/
/*****
/*-----
/* Set of Ethernet-related */
/*-----*/
/* Local MAC address (Set all 0s when unspecified) */
#define MY_MAC_ADDR0 0x20,0x00,0x00,0x00,0x00,0x00
#define MY_MAC_ADDR1 0x40,0x00,0x00,0x00,0x00,0x00

UB _myethaddr[][6]={ {MY_MAC_ADDR0}, {MY_MAC_ADDR1}}; /* MAC address */

```

図 19. Ethernet 使用時の自局の設定 (config\_tcpudp.c より抜粋)

Ethernet を使用する場合の自局に関する情報として、IP アドレス、サブネットマスク、ゲートウェイアドレス、イーサネットアドレス (MAC アドレス) を定義します。図 19 に記述例を示します。

自局の設定は、以下で示す 3 つのフィールドを含む TCPUDP\_ENV 構造体 (第 4 章参照) とイーサネットアドレス (MAC アドレス) で構成されています。

tcpudp\_env = { IP アドレス , サブネットマスク , ゲートウェイアドレス }

\_myethaddr = { MAC アドレス }

各情報は、図 19 に示すように define 定義を用いることもできます。各情報の値は、コンマで区切って指定してください。

IP アドレス、サブネットマスクの設定は必須です。

ゲートウェイアドレスは、使用する場合にはそのアドレスを、そうでない場合には {0,0,0,0} と、全て 0 を設定してください。ゲートウェイアドレスの設定は 1 つのみ可能です。

MAC アドレスには、プログラムにより Ethernet コントローラに対して MAC アドレスを指定する場合にはそのアドレスを、EEPROM 等により Ethernet コントローラの MAC アドレスを自動設定する場合には {0,0,0,0,0,0} と、全て 0 を設定してください。

また、変数 tcpudp\_env、変数\_myethaddr は、必ず RAM 領域に配置してください。

変数\_ip\_tblcnt には、T4 で使用する ARP のキャッシュエントリ数を設定します。T4 では通信するホスト 1 台につき、1 つのキャッシュエントリを使用します。

「同時に通信する可能性のあるホスト数+1」以上の値を推奨します。「同時に通信する可能性のあるホスト数」よりも小さな値を設定した場合、動作は不定です。例えば、TCP、UDP を同時に使用し、それぞれ別ホストと通信する場合、3 以上の値を設定するようにしてください。

また、キャッシュエントリの保持期間は 10 分のため、その間に、頻繁に多数のホストと通信する場合、ホスト数以上の値を設定すると通信効率が上がります。例えば、4 つのホストと順に UDP で通信する場合、4 以下のエントリ数を設定すると毎回 ARP の解決が必要となります。

## 8. T4 ライブラリの API

T4 でサポートする API の一覧を表 3 に示します。TCP、UDP 関連の各 API は、ITRON TCP/IP API 仕様に準拠しています。

同時に実行可能な API の数は、TCP と UDP の端点毎に各ひとつです。(キャンセル API を除く)

表 3. T4 の API 一覧

T4 API 一覧		C 言語 API
TCP	TCP 接続要求待ち(受動オープン)	tcp_acp_cep()
	TCP 接続要求 (能動オープン)	tcp_con_cep()
	TCP データ送信の終了	tcp_sht_cep()
	TCP 通信端点のクローズ	tcp_cls_cep()
	TCP データ送信	tcp_snd_dat()
	TCP データ受信	tcp_rcv_dat()
	TCP API キャンセル	tcp_can_cep()
UDP	UDP データ送信	udp_snd_dat()
	UDP データ受信	udp_rcv_dat()
	UDP API キャンセル	udp_can_cep()
TCP/UDP 共通	コールバック関数(ユーザ定義関数)	callback()
PPP	PPP 開始処理	ppp_open()
	PPP 終了処理	ppp_close()
	PPP 状態参照	ppp_status()
	PPP 処理関数への要求発行	ppp_api_req()
初期化	ワークメモリ使用サイズの取得	tcpudp_get_ramsize()
	T4 ライブラリ・オープン	tcpudp_open ()
周期処理	TCP/IP プロトコル処理	_process_tcpip()
終了処理	T4 ライブラリ・クローズ	tcpudp_close()

※上記 API のスタックサイズは、各導入ガイドを参照ください。

以降では、API で使用するデータ構造とマクロ定義、各 API の詳細について説明します。

各 API の詳細は、以下のフォーマットに沿っています。

### < API 書式 >

API の書式を示します。

### < 機能 >

各 API の機能・動作、使用上の注意点を示します。

### < 引数 >

API に与える引数の意味、値の制限を示します。

### < 戻り値・エラーコード >

API からの戻り値やエラーコードの種類、発生する条件を示します。



## 8.1 tcp\_acp\_cep

### < API 書式 >

```
ER tcp_acp_cep( ID cepid, ID repid, T_IPV4EP *p_dstaddr, TMO tmout )
```

### < 機能 >

TCP 受付口 repid に対する接続要求を待ちます。接続要求があった場合には、指定した TCP 通信端点 cepid を用いて接続を確立し、接続を要求してきた相手の IP アドレスとポート番号を引数 p\_dstaddr が指す領域に格納して返します。

タイムアウト指定 tmout に TMO\_FEVR を指定した場合は、接続が確立するまでは待ち状態となりますが、この待ち時間に対してタイムアウト指定することができます。指定時間内に接続が確立されない場合、エラーコード E\_TMOUT を返します。

接続が確立された場合、戻り値 E\_OK が返されます。確立されない場合は、各原因により上記 E\_OK 以外のエラーコードを返します。エラーコードに対応した原因を ( ) 内に示します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合は、ノンブロッキングコールとなり、本 API では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 E\_WBLK が返ります。この場合、接続が確立された時点でコールバック関数が呼び出されます。コールバック関数には、引数として TCP 通信端点 ID、機能コード TFN\_TCP\_ACP\_CEP、エラーコードへのポインタが渡されます。

コールバック関数内から本 API を使用する場合、タイムアウト指定に TMO\_NBLK 以外を指定すると、パラメータエラー (E\_PAR) となります。

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1~30 までの任意設定値)
repid	ID	TCP 受付口 ID 指定(1~30 までの任意設定値)
p_dstaddr	T_IPV4EP	相手の IP アドレスとポート番号取得 接続を要求してきた相手の IP アドレスとポート番号を取得
tmout	TMO	タイムアウト指定 正の値： 接続が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR： 接続が完了するまで待ち状態(永久待ち) TMO_NBLK： ノンブロッキング呼び出し

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了 (接続が確立)
E_PAR	パラメータエラー (cepid、tmout が不正な値)
E_QOVR	キューイングオーバーフロー (同一端点に対し複数の API が発行された)
E_OBJ	オブジェクト状態エラー (使用中の通信端点 ID を指定)
E_TMOUT	タイムアウト (tmout に設定した時間を経過)
E_WBLK	ノンブロッキングコール受付

## 8.2 tcp\_con\_cep

### < API 書式 >

```
ER tcp_con_cep( ID cepid, T_IPV4EP *p_myaddr, T_IPV4EP *p_dstaddr, TMO tmout )
```

### < 機能 >

TCP 通信端点 cepid を用いて、接続したい相手の IP アドレス/ポート番号に対して接続を要求し、タイムアウト指定 tmout に TMO\_FEVR を指定した場合は、接続が完了するまで待ち状態となります。

接続が確立するまでは待ち状態となりますが、この待ち時間に対してタイムアウト指定することができません。指定時間内に接続が確立されない場合、E\_TMOUT を返します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合は、ノンブロッキングコールとなり、本 API では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 E\_WBLK が返ります。この場合、接続が確立された時点でコールバック関数が呼び出されます。コールバック関数には、引数として TCP 通信端点 ID、機能コード TFN\_TCP\_CON\_CEP、エラーコードへのポインタが渡されます。

タイムアウトにより接続の要求がキャンセルされた場合、および相手がサポートしていないポート番号を指定するなど接続が拒否された場合には、通信端点 cepid は未使用状態に戻ります。

この API がコールされるとまず通信端点が使用中でないかをチェックし、以下の設定を行います。

自局の IP アドレスには、変数 tcpudp\_env (4章参照)に設定された自局の IP アドレスが設定されます。

自局のポート番号に 0 以外の値を指定した場合、その値が設定されます。また、自局のポート番号に TCP\_PORTANY を指定した場合、T4 で 1024～5000 番の範囲の中から割り当てます。

自局の IP アドレス/ポート番号 p\_myaddr に NADR を指定した場合には、自局の IP アドレスには変数 tcpudp\_env に設定された IP アドレスを、自分側のポート番号には T4 で 1024～5000 番の範囲の中からポート番号を割り当てます。

ブロッキングコールで接続が確立された場合、戻り値 E\_OK が返されます。確立されない場合は、各原因により上記 E\_OK 以外のエラーコードを返します。エラーコードに対応した原因を ( ) 内に示します。

コールバック関数内から本 API を使用する場合、タイムアウト指定に TMO\_NBLK 以外を指定すると、パラメータエラー (E\_PAR) となります。

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1～30 までの任意設定値)
p_myaddr	T_IPV4EP	自局の IP アドレスとポート番号指定
p_dstaddr	T_IPV4EP	接続したい相手側の IP アドレスとポート番号指定
tmout	TMO	タイムアウト指定 正の値： 接続が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR： 接続が完了するまで待ち状態(永久待ち) TMO_NBLK： ノンブロッキング呼び出し

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了 (接続が確立)
E_PAR	パラメータエラー (cepid、tmout が不正な値)
E_QOVR	キューイングオーバーフロー (同一端点に対し複数の API が発行された)
E_OBJ	オブジェクト状態エラー (使用中の通信端点 ID、または自局ポート番号を指定)
E_TMOUT	タイムアウト (tmout に設定した時間を経過)
E_CLS	接続の失敗 (接続が拒否された)
E_WBLK	ノンブロッキングコール受付

ヒント:

IP アドレスとポート番号を T\_IP\_V4EP 構造体に格納して本関数に指定してください。

例: IP アドレス 192.168.123.250、ポート 80 番に接続する場合.

```
T_IPV4EP dst, src;
dst.ipaddr = 0xc0a87bfa; // 192 = 0xc0, 168 = 0xa8, 123= 0x7b, 250
= 0xfa
dst.portno = 80;
tcp_con_cep(1, &src, &dst, TMO_NBLK);
```

### 8.3 tcp\_sht\_cep

#### < API 書式 >

```
ER tcp_sht_cep( ID cepid )
```

#### < 機能 >

TCP 通信端点 cepid に対する接続の切断処理の準備として、データ送信の終了手続きを行います。具体的には、送信したデータに対する ACK を受信した時点で、FIN を送信します。本 API は切断処理の手配を行うだけのため、待ち状態にはなりません。

本 API 発行後、指定した TCP 通信端点 cepid に対してデータを送信することはできず、送信しようとした場合、エラーコード E\_OBJ を返します。データの受信は可能です。

正常にデータ転送が行われた場合は、E\_OK を返します。データ転送が失敗した場合は各原因により上記 E\_OK 以外のエラーコードを返します。エラーコードに対応した原因を ( ) 内に示します。

コールバックルーチンを持たない通信端点(cepid)に対して本 API を発行した場合、本 API はブロッキングコール指定で呼び出されます。

コールバックルーチンを持つ通信端点(cepid)に対して本 API を発行した場合、本 API はノンブロッキングコール指定で呼び出されます。

本 API はシャットダウンパケットを送信すると完了します。

コールバック関数内から本 API を使用すると、サポート外エラー (E\_NOSPT) となります。

#### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1~30 までの任意設定値)

#### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了 (データ送信が終了)
E_PAR	パラメータエラー (cepid が不正な値)
E_OBJ	オブジェクト状態エラー (指定した通信端点が未接続)
E_NOSPT	コールバックルーチン内からコールされた時 (サポート外エラー)
E_WBLK	ノンブロッキングコール受付

## 8.4 tcp\_cls\_cep

### < API 書式 >

```
ER tcp_cls_cep( ID cepid, TMO tmout )
```

### < 機能 >

TCP 通信端点 cepid の接続を切断します。本 API を発行後は、相手から送信されたデータを破棄します。

タイムアウト指定 tmout に TMO\_FEVR を指定した場合の、タイムアウトにより接続の切断処理がキャンセルされた場合、本 API で指定した TCP 通信端点から RST を送信し、強制的に接続を切断します。この場合、正常切断ではないため、エラーコード E\_TMOUT を返します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合は、ノンブロッキングコールとなり、本 API では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 E\_WBLK が返ります。この場合、接続が切断された時点でコールバック関数が呼び出されます。コールバック関数には、引数として TCP 通信端点 ID、機能コードTFN\_TCP\_CLS\_CEP、エラーコードへのポインタが渡されます。

本 API は、タイムアウト指定 tmout に TMO\_FEVR を指定した場合、正常切断、強制切断のどちらの場合も通信端点が未使用状態になるのを待って API からリターンするため、本 API からリターンした後は TCP 通信端点 cepid をすぐに利用することが可能です。TCP 通信端点は、接続が完全に終了するまで未使用状態となりません。TCP/IP の規格に従うと接続が完全に終了するまでに TIME\_WAIT 状態に留まる場合があります。TIME\_WAIT 状態の時間 2MSL は、T4 のコンフィグレーションファイルで設定できます（6章参照）。

ブロッキングコールで接続が正常に切断された場合、戻り値 E\_OK が返されます。タイムアウトにより強制的に切断された場合は、エラーコード E\_TMOUT を返します。これらのコードが返された場合、接続は完全に終了しているため、指定した TCP 通信端点は未使用状態となります。また通信端点が未接続の場合には本 API は、E\_OBJ を返します。

コールバック関数内から本 API を使用する場合、タイムアウト指定に TMO\_NBLK 以外を指定すると、パラメータエラー (E\_PAR) となります。

tcp\_sht\_cep()発行後、ケーブルが切断された等で通信不可の状態になったときにtcp\_cls\_cep()を TMO\_FEVR 指定で発行すると、待ち状態が継続される場合があります。tcp\_cls\_cep()はタイムアウト指定、またはノンブロッキングコール後アプリケーションでタイムアウト処理とキャンセル API を発行してください。

### 【補足事項】

ITRON 仕様では、データの送信が終了していない場合は、送信が完了するのを待ってから FIN を送信し、接続の切断を行います。T4 では、同時に複数の API を実行できない為、本 API を発行するときには、すでにデータの送信は完了しているため、FIN の送信が待たされることはありません。

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1~30 までの任意設定値)
tmout	TMO	タイムアウト指定 正の値 : 接続がクローズするのを待つ時間。時間の単位は 10msec TMO_FEVR : 接続がクローズするまで待ち状態(永久待ち) TMO_NBLK : ノンブロッキング呼び出し

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了（接続が正常切断）
E_PAR	パラメータエラー（cepid、tmout が不正な値）
E_QOVR	キューイングオーバーフロー（同一端点に対し複数の API が発行された）
E_OBJ	オブジェクト状態エラー（指定した通信端点が未接続）
E_TMOUT	タイムアウト（tmout に設定した時間を経過。接続を強制切断）
E_WBLK	ノンブロッキングコール受付

## 8.5 tcp\_snd\_dat

### < API 書式 >

```
ER tcp_snd_dat( ID cepid, VP data, INT len, TMO tmout )
```

### < 機能 >

TCP 通信端点 cepid からデータを送信します。正常に送信した場合、API は送信したデータサイズを返します。

T4 では、RAM 使用の効率化、送信速度の向上のため、ITRON TCP の仕様と異なる点があります。本ライブラリでは、送信データが格納されている領域（以下、ユーザ送信バッファと呼びます）が、ITRON TCP で定義されている送信ウィンドウを兼ねています。同様に、送信ウィンドウサイズは、送信データの長さとなり、本 API の使用状況によりサイズが異なります。

ITRON 仕様では、ユーザ送信バッファのデータを送信ウィンドウにコピーした時点で、API からリターンしますが、本 API ではタイムアウト指定 tmout に TMO\_FEVR を指定した場合、データを送信し、送信データに対する ACK を受信した後、API からリターンします。具体的には、MSS や相手の受信ウィンドウサイズにより TCP レベルの分割が発生した場合、1 つ目の分割データの送信に対する ACK ではなく、全てのデータの送信に対する ACK を受信した時点で API からリターンします。

コールバック関数には、引数として TCP 通信端点 ID、機能コード TFN\_TCP\_SND\_DAT、エラーコードへのポインタが渡されます。正常に送信完了した場合、エラーコードに送信したデータサイズがセットされます。

コールバック関数内から本 API を使用する場合、タイムアウト指定に TMO\_NBLK 以外を指定すると、パラメータエラー (E\_PAR) となります。

### 【補足事項】

ITRON 仕様では、送信ウィンドウの空き領域のサイズにより、送信データのサイズが決まります。このため、API の正常終了時の戻り値は必ず第 3 引数 len と一致するわけではありません。T4 以外の ITRON TCP/IP API 仕様準拠のプロトコルスタック上に移植される場合は、ご注意ください。

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1~30 までの任意設定値)
data	VP	送信データの先頭アドレス指定 ユーザが送信するデータの先頭アドレス
Len	INT	送信データの長さ指定 正の値
tmout	TMO	タイムアウト指定 正の値：送信が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR：送信が完了するまで待ち状態(永久待ち) TMO_NBLK：ノンブロッキング呼び出し

### < 戻り値・エラーコード >

戻り値	意味
正の値	正常終了（送信したデータのサイズ=第 3 引数 len の値。単位：バイト）
E_PAR	パラメータエラー（cepid、tmout が不正な値）
E_QOVR	キューイングオーバーフロー（同一端点に対し複数の API が発行された）
E_OBJ	オブジェクト状態エラー（未接続、送信終了）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_CLS	相手から接続を切断された（RST の送受信により異常切断した場合）
E_WBLK	ノンブロッキングコール受付



## 8.6 tcp\_rcv\_dat

### < API 書式 >

```
ER tcp_rcv_dat( ID cepid, VP data, INT len, TMO tmout )
```

### < 機能 >

TCP 通信端点 cepid からデータを受信します。

TMO\_FEVR を指定した場合は、本 API は、受信ウィンドウから引数 data が指すユーザ領域にデータをコピー（以下、データの取り出しと呼びます）し、リターンします。受信ウィンドウが空の場合は、データを受信するまで本 API は待ち状態となります。

受信ウィンドウに入っているデータのサイズが、受信しようとしたデータのサイズ len よりも短い場合、受信ウィンドウが空になるまでデータを取り出し、取り出したデータのサイズを戻り値として返します。

相手から接続が正常に切断され、受信ウィンドウのデータを全て取り出し、データがなくなると、API から 0 が返ります。

データを受信した場合、受信したデータのサイズが返されます。受信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を（）内に示します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合は、ノンブロッキングコールとなり、本 API では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 E\_WBLK が返ります。この場合、エラーが発生した時点、または全受信データをユーザの受信バッファに格納した時点でコールバック関数が呼び出されます。コールバック関数には、引数として TCP 通信端点 ID、機能コード TFN\_TCP\_RCV\_DAT、エラーコードへのポインタが渡されます。受信したデータサイズはエラーコードに示され、受信したデータはノンブロッキングコールを行った API で指定したユーザ領域 data に格納されています。

コールバック関数内から本 API を使用する場合、タイムアウト指定に TMO\_NBLK、TMO\_POL 以外を指定すると、パラメータエラー（E\_PAR）となります。

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定（1～30 までの任意設定値）
data	VP	受信データを格納する領域の先頭アドレス指定 ユーザが確保した受信データを格納するバッファの先頭アドレス
len	INT	受信データを格納する最大サイズ 正の値
tmout	TMO	タイムアウト指定 正の値： 受信が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR： 受信が完了するまで待ち状態(永久待ち) TMO_NBLK： ノンブロッキング呼び出し TMO_POL： ポーリング

### < 戻り値・エラーコード >

戻り値	意味
正の値	正常終了（受信したデータのサイズ。単位：バイト）
0	データ終了（接続が正常切断。切断までのデータは全て受信）
E_PAR	パラメータエラー（cepid、tmout が不正な値）
E_QOVR	キューイングオーバーフロー（同一端点に対し複数の API が発行された）
E_OBJ	オブジェクト状態エラー（未接続）
E_TMOUT	タイムアウト（tmout に設定した時間を経過）
E_CLS	接続を切断され、受信ウィンドウが空（RST の受信により異常切断した場合）
E_WBLK	ノンブロッキングコール受付

## 8.7 tcp\_can\_cep

### < API 書式 >

```
ER tcp_can_cep( ID cepid, FN fncd )
```

### < 機能 >

ノンブロッキングコールで呼び出した他の TCP API をキャンセルし、また別の API を呼び出したい場合に使用します。本 API が正常終了(キャンセル処理受付)し、キャンセル処理が終了するとコールバック関数が呼び出されます。その際の事象の種類は「キャンセルした API の機能コード」、事象に固有なパラメータブロックのアドレスには、エラーコード「E\_RLWAI」が格納されている領域のアドレスが格納されます。本 API を呼び出した後、別の API を呼び出すことが可能になるタイミングは以下の通りです。

#### (1)E\_OK が戻り値の場合

コールバック関数で E\_RLWAI が通知された時

#### (2)E\_OBJ が戻り値の場合

キャンセル API が終了した時

### < 引数 >

名称	型	機能
cepid	ID	TCP 通信端点 ID 指定(1~30 までの任意設定値)
fncd	FN	API の種類(以下対応機能コード) TFN_TCP_ACP_CEP, TFN_TCP_CON_CEP, TFN_TCP_CLS_CEP, TFN_TCP_SND_DAT, TFN_TCP_RCV_DAT, TFN_TCP_ALL

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了 (キャンセル処理受付完了した場合)
E_OBJ	オブジェクト状態エラー(fncd で指定の API がペンディングしていない場合)
E_PAR	パラメータエラー(cepid、fncd が不正な値)
E_NOSPT	未サポート(コールバック関数内で本 API をコールした場合)



## 8.8 udp\_snd\_dat

### < API 書式 >

```
ER udp_snd_dat( ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout )
```

### < 機能 >

UDP 通信端点 cepid から、相手側の IP アドレスとポート番号を指定して、UDP データグラムを送信します。UDP データグラムを送信バッファに入れた時点で本 API からリターンします。ここで指す送信バッファとは、T4 内部で確保したものではなく、Ethernet コントローラ内部の送信バッファ (Ethernet の場合)、シリアルドライバで確保した送信バッファ (PPP の場合) のことです。データが送信バッファに格納された場合、送信したデータのサイズ (第 4 引数 len の値) が返されます。送信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を () 内に示します。

本 API では、宛先アドレスにユニキャストアドレス、または、マルチキャストアドレス、ブロードキャストアドレス、ディレクテッドブロードキャストアドレスを指定してデータを送信することができます。

T4 では、1 度に送信できるデータサイズの最大値は、送信バッファのサイズに依存します。ドライバ・インタフェース関数 lan\_write, ppp\_write により、最大サイズが M (バイト) の Ethernet フレーム、PPP フレームを送信 (送信バッファに格納) できる場合、1 度に送信可能なデータの最大サイズ N (バイト) は、

$$N = M - \text{フレーム・ヘッダサイズ}(F) - \text{IP ヘッダの最小サイズ}(I) - \text{UDP ヘッダサイズ}(U)$$

となります。ここで、M, F, I, U の各値は以下の通りです。

M ≤ 1514 (Ethernet の場合)、M ≤ 1504 (PPP の場合)

F = 14 (Ethernet の場合)、F = 4 (PPP の場合)

I = 20

U = 8

T4 では、IP フラグメントに対応していないため、これより大きなサイズのデータを送信する場合には、データを分割し、N バイト以下にする必要があります。N バイト以上のデータサイズを指定した場合の動作、戻り値は不定です。

タイムアウト指定 tmout に TMO\_FEVR を指定した場合は、送信バッファにデータが入るまで待ち状態になります。タイムアウト指定 tmout に正の値を指定した場合は、指定時間まで待ち状態になります。指定時間までに送信バッファにデータが入らなかった場合には、エラーコード E\_TMOUT を返します。データが送信バッファに格納された場合、格納されたデータのサイズを返します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合、ノンブロッキングコールとなり、本 API では待ち状態になりません。送信要求が受け付けられるとノンブロッキングコール受付 E\_WBLK を返します。この場合、送信バッファにデータが格納された時点でコールバック関数が呼び出されます。コールバック関数には、引数として UDP 通信端点 ID、機能コード TFN\_UDP\_SND\_DAT、エラーコードへのポインタが渡されます。エラーコードには、格納されたデータサイズが示されます。

本 API をブロッキングコールしてから API を抜けるまでの間、ノンブロッキングコールしてからコールバック関数 (機能コード TFN\_UDP\_SND\_DAT) が呼び出されるまでの間は、UDP 送信処理中とみなされます。そのため、処理中は送信データを書き換えしないでください。

T4 では、同時に複数の UDP 関数 (udp\_rcv\_dat(), および、udp\_snd\_dat()) を発行できません (※)。複数の UDP 関数を発行しなければ、コールバック中でも UDP 関数は呼び出し可能です。UDP 関数の処理中に本 API を発行した場合、エラーコード E\_QOVR を返します。

※事象コード TEV\_UDP\_RCV\_DAT により呼び出されたコールバック関数の中でポーリング指定 (TMO\_POL)により udp\_rcv\_dat()を呼び出す場合を除く。

## < 引数 >

名称	型	機能
cepid	ID	UDP 通信端点 ID 指定 (1~30 までの任意設定値)
*p_dstaddr	T_IPV4EP	送信したい相手側の IP アドレスとポート番号
data	VP	送信データの先頭アドレス指定 ユーザが送信するデータの先頭アドレス
len	INT	送信データの長さ指定 0 以上 1472 以下
tmout	TMO	タイムアウト指定 正の値 : 送信が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR : 送信が完了するまで待ち状態(永久待ち) TMO_NBLK : ノンブロッキング呼び出し

## < 戻り値・エラーコード >

戻り値	意味
0, 正の値	正常終了 (送信したデータのサイズ=第 4 引数 len の値。単位 : バイト)
E_PAR	パラメータエラー (cepid、tmout が不正な値)
E_QOVR	キューイングオーバーフロー (同一端点に対し複数の API が発行された)
E_TMOUT	タイムアウト (tmout に設定した時間を経過)
E_WBLK	ノンブロッキングコール受付
E_CLS	接続の失敗(ARP 交換ができなかった)

## 8.9 udp\_rcv\_dat

### < API 書式 >

```
ER udp_rcv_dat( ID cepid, T_IPV4EP *p_dstaddr, VP data, INT len, TMO tmout )
```

### < 機能 >

UDP 通信端点 cepid から UDP データグラムを受信し、相手側の IP アドレスとポート番号を取得します。データを受信した場合、受信したデータのサイズが返されます。受信に失敗した場合は、各原因により上記エラーコードを返します。エラーコードに対応した原因を ( ) 内に示します。

T4 では、1 度に受信できるデータサイズの最大値は、ドライバの受信バッファのサイズに依存します。ドライバ・インタフェース関数 lan\_read, ppp\_read により、最大サイズが M (バイト) の Ethernet フレーム、PPP フレームを受信(受信バッファに格納)できる場合、1 度に受信可能なデータの最大サイズ N (バイト) は、

$$N = M - \text{フレーム・ヘッダサイズ(F)} - \text{IP ヘッダの最小サイズ(I)} - \text{UDP ヘッダサイズ(U)}$$

となります。ここで、M,F,I,U の各値は以下の通りです。

M ≤ 1514 (Ethernet の場合)、M ≤ 1504 (PPP の場合)

F = 14 (Ethernet の場合)、F = 4 (PPP の場合)

I = 20

U = 8

T4 では、IP フラグメントに対応していないため、これより大きなサイズのデータを受信した場合には破棄されます。

タイムアウト指定 tmout に TMO\_FEVR を指定した場合は、UDP データグラムを受信するまで待ち状態になります。タイムアウト指定 tmout に正の値を指定した場合は、指定時間まで待ち状態になります。指定時間までに UDP データグラムを受信できなかった場合、エラーコード E\_TMOUT を返します。

タイムアウト指定 tmout に TMO\_NBLK を指定した場合は、ノンブロッキングコールとなり、本 API では待ち状態になりません。要求が受け付けられるとノンブロッキングコール受付 E\_WBLK が返ります。この場合、受信データをユーザの受信バッファに格納した時点でコールバック関数が呼び出されます。コールバック関数には、引数として UDP 通信端点 ID、機能コード TFN\_UDP\_RCV\_DAT、エラーコードへのポインタが渡されます。受信したデータサイズはエラーコードに示され、受信したデータはノンブロッキングコールを行った API で指定したユーザ領域 data に格納されています。

本 API をブロッキングコールしてから API を抜けるまでの間、ノンブロッキングコールしてからコールバック関数(機能コード TFN\_UDP\_RCV\_DAT)が呼び出されるまでの間は、UDP データ受信待ち状態であり、UDP 受信処理中とみなされます。

受信 API が発行された状態で UDP データを受信すると、受信した UDP データは API で指定したデータ領域にコピーされます。

受信したデータのサイズが API で指定したサイズと等しい、または指定したサイズより小さい場合、受信したデータの全てはコピーされます。受信したデータのサイズが指定したサイズより大きい場合、受信したデータは指定されたデータサイズと等しいバイト数コピーされ、残りは破棄されます。

前者の場合、戻り値は受信データ長です。

後者の場合、戻り値は E\_BOVR です。

受信 API が発行されていない状態で UDP データを受信すると、イベントコード「TEV\_UDP\_RCV\_DAT」を伴ってコールバック関数が呼ばれます。

コールバック関数は、引数として、端点 ID、イベントコード、エラーコードへのポインタを指定します。

受信したデータのサイズはエラーコードで示されます。このコールバック中に、UDP 受信 API の `tmout` 引数に `TMO_POL` を指定して呼び出すことで UDP 受信データを読み出すことが出来ます。

T4 では、同時に複数の UDP 関数 (`udp_rcv_dat()`、および、`udp_snd_dat()`) を発行できません(※)。複数の UDP 関数を発行しなければ、コールバック中でも UDP 関数は呼び出し可能です。UDP 関数の処理中に本 API を発行した場合、エラーコード `E_QOVR` を返します。

※事象コード `TEV_UDP_RCV_DAT` により呼び出されたコールバック関数の中でポーリング指定 (`TMO_POL`)により `udp_rcv_dat()` を呼び出す場合を除く。

本 API では、宛先 IP アドレスがユニキャストアドレス、または、マルチキャストアドレス (224.0.0.0～239.255.255.255)、または、ブロードキャストアドレス(255.255.255.255)の UDP データグラムを受信することができます。また、ディレクテッドブロードキャストアドレス(例: 192.168.0.0/24 の場合、192.168.0.255)も受信可能です。ただし、T4 では IGMP をサポートしていませんので、ルータに対してマルチキャストへの参加を通知することはできません。

## 【補足事項】

ノンブロッキングコールを使用する場合、コールのタイミングによっては、事象コード `TEV_UDP_RCV_DAT` のコールバック関数が先に呼ばれることがあります。その場合、次の UDP データの受信により機能コード `TFN_UDP_RCV_DAT` のコールバック関数が呼ばれます。

## < 引数 >

名称	型	機能
<code>cepid</code>	ID	UDP 通信端点 ID 指定 (1～30 までの任意設定値)
<code>*p_dstaddr</code>	T_IPV4EP	相手の IP アドレスとポート番号取得 データを送信してきた相手の IP アドレスとポート番号を取得
<code>data</code>	VP	受信データを格納する領域の先頭アドレス指定 ユーザが確保した受信データを格納するバッファの先頭アドレス
<code>len</code>	INT	受信データを格納する最大サイズ 0 以上 1472 以下
<code>tmout</code>	TMO	タイムアウト指定 正の値: 受信が完了するのを待つ時間。時間の単位は 10msec TMO_FEVR: 受信が完了するまで待ち状態(永久待ち) TMO_NBLK: ノンブロッキング呼び出し TMO_POL: ポーリング 事象コード <code>TEV_UDP_RCV_DAT</code> による コールバック関数の中でのみ指定可能。

## < 戻り値・エラーコード >

戻り値	意味
0, 正の値	正常終了 (受信したデータのサイズ。単位: バイト)
<code>E_PAR</code>	パラメータエラー ( <code>cepid</code> 、 <code>tmout</code> が不正な値)
<code>E_QOVR</code>	キューイングオーバーフロー (同一端点に対し複数の API が発行された)
<code>E_TMOUT</code>	タイムアウト ( <code>tmout</code> に設定した時間を経過)
<code>E_WBLK</code>	ノンブロッキングコール受付
<code>E_BOVR</code>	バッファオーバーフロー (受信データを格納する領域以上のデータが届いた)

## 8.10 udp\_can\_cep

### < API 書式 >

```
ER udp_can_cep( ID cepid, FN fncd )
```

### < 機能 >

ノンブロッキングコールで呼び出した他の UDP API をキャンセルし、また別の API を呼び出したい場合に使用します。本 API が正常終了(キャンセル処理受付)し、キャンセル処理が終了するとコールバック関数が呼び出されます。その際の事象の種類は「キャンセルした API の機能コード」、事象に固有なパラメータブロックのアドレスには、エラーコード「E\_RLWAI」が格納されている領域のアドレスが格納されます。

本 API を呼び出した後、別の API を呼び出すことが可能になるタイミングは以下の通りです。

#### (1)E\_OK が戻り値の場合

コールバック関数で E\_RLWAI が通知された時

#### (2)E\_OBJ が戻り値の場合

キャンセル API が終了した時

### < 引数 >

名称	型	機能
cepid	ID	UDP 通信端点 ID 指定(1~30 までの任意設定値)
fncd	FN	API の種類(以下対応 API コード) TFN_UDP_SND_DAT, TFN_UDP_RCV_DAT, TFN_UDP_ALL

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了 (キャンセル処理受付完了した場合)
E_OBJ	オブジェクト状態エラー(fncd で指定の API がペンディングしていない場合)
E_PAR	パラメータエラー(cepid、fncd が不正な値)
E_NOSPT	未サポート(コールバック関数内で本 API をコールした場合)

## 8.11 ppp\_open

### < API 書式 >

```
ER ppp_open( void )
```

### < 機能 >

PPP サーバとのリンクを確立し、PAP による認証を行い、IPCP によりネットワークをオープン状態にします。PAP では、T4 のコンフィグレーションファイルで設定するグローバル変数 UB user\_name[], UB user\_passwd[] に格納されたユーザ名とパスワードを用いて認証を行います。IPCP では、T4 のコンフィグレーションファイルで設定する変数 tcpudp\_env の IP アドレスが 0 でない場合、PPP サーバに特定の IP アドレスの割り当てを、また、変数 tcpudp\_env の IP アドレスが 0 の場合には、PPP サーバによる IP アドレスの自動割当を要求します。特定の IP アドレスを要求した場合でも、PPP サーバがその IP アドレスを許可するとは限りません。最終的に PPP サーバが許可した IP アドレスは、変数 tcpudp\_env の IP アドレスに格納されます。

本 API は、T4 ライブラリの初期化関数 tcpudp\_open() の後で呼び出してください。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	初期化失敗

## 8.12 ppp\_close

### < API 書式 >

```
ER ppp_close( void )
```

### < 機能 >

PPP サーバとの接続の切断処理を行います。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	終了処理失敗

## 8.13 ppp\_status

### < API 書式 >

```
UH ppp_status( void )
```

### < 機能 >

PPP の接続状態を返します。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

PPP の接続状態に応じて、以下の値を返します。

戻り値	意味
0x0001(PS_DEAD)	リンク切断状態
0x0002(PS_ESTABLISH)	LCP フェーズ
0x0004(PS_AUTHENTICATE)	認証フェーズ (PAP)
0x0008(PS_NETWORK)	NCP フェーズ (IPCP)
0x0010(PS_NETOPEN)	ネットワーク確立
0x0020(PS_TERMINATE)	リンク切断中
Negative value	初期化失敗



## 8.14 ppp\_api\_req

### < API 書式 >

```
ER ppp_api_req( UH type, void *parblk, H tmout)
```

### < 機能 >

T4 ライブラリの PPP 処理に対して、AT コマンド送信、応答コードの受信、所定の時間待ちの要求を発行します。各要求は、要求番号 `type` で指定します。また、各要求に応じて必要となるパラメータは、ポインタ `parblk` で指定します。発行した要求が、所定の時間までに完了しない場合には、タイムアウト `E_TMOUT(-85)` が返ります。

本関数で要求を発行した場合、PPP 処理関数では以下のドライバ関数を実行します。

- (1) `type` が `PPP_SNDCOMMAND` の場合

`modem_write()`

- (2) `type` が `PPP_RCVRZLT` の場合

`modem_read()`

- (3) `type` が `PPP_WAIT` の場合

なし

本関数の戻り値は、以下の通りです。

- (1) `type` が `PPP_SNDCOMMAND` の場合

`modem_write()`の戻り値

- (2) `type` が `PPP_RCVRZLT` の場合

`modem_read()`の戻り値

- (3) `type` が `PPP_WAIT` の場合

`E_TMOUT(-85)`

### < 引数 >

名称	型	機能
<code>type</code>	UH	PPP 処理関数への要求番号 <code>type</code> には以下を設定して下さい。他の値は予約されていますので、使用しないで下さい。 3 : (PPP_API_SNDCOMMAND)AT コマンド送信要求 4 : (PPP_API_RCVRZLT) 応答コード受信要求 5 : (PPP_API_WAIT) 所定の時間待ち要求
<code>parblk</code>	void *	要求に対応したパラメータへのポインタ。 <code>type</code> が <code>PPP_API_WAIT</code> の場合は、無効です (無視されます)。 (ポインタ <code>parblk</code> が指すデータの形式は、カスタマイズ可能です)
<code>tmout</code>	H	タイムアウト指定。 正の値 : タイムアウトまでの時間 (10msec 単位) -1 : 無限待ち 上記以外 : 予約 (設定不可)

### < 戻り値・エラーコード >

PPP の接続状態に応じて、以下の値を返します。

戻り値	意味
0 以上の値	正常終了
-85 (E_TMOUT)	タイムアウトにより終了
負の値	その他のエラーにより終了。(ドライバの実装に依存)

### < 注意事項 >

本関数は、T4 ライブラリに含まれる関数です。ユーザ側で作成する必要はありません。ドライバ API 関数を作成する場合に使用して下さい。

## 8.15 callback

### < API 書式 >

```
ER callback( ID cepid, FN fncd, VP p_parblk )
```

### < 機能 >

TCP・UDP コールバック関数は、TCP・UDP 関数をノンブロッキングコールした場合の完了通知を行います。

TCP・UDP コールバック関数の中身は各通知における処理であり、ユーザが作成します。引数 fncd が、通知の種類をあらわします。事象の種類とコールバック関数が呼ばれるタイミングを付録A3に示します。

p\_parblk が指し示す領域には完了した API の戻り値(ER 型)が格納されます。たとえば、tcp\_snd\_dat()を TMO\_NBLK(ノンブロッキングコール)指定で呼び出し、送信が完了した後、callback()関数が p\_parblk に 送信データ長 を格納した状態で呼び出されます。

コールバック関数は config\_tcpudp.c で通信端点毎に関数名を設定可能です。設定方法は、6.3 TCP 通信端点の定義、または 6.4 UDP 通信端点の定義を参照してください。

### < 引数 >

名称	型	機能
cepid	ID	完了した API で使用されていた通信端点 ID を格納しています
fncd	FN	完了した API を示す機能コードが格納されています。
p_parblk	VP	完了した API の戻り値を格納した領域のポインタが格納されています。

### < 戻り値・エラーコード >

戻り値	意味
0	常に 0 を戻り値に指定してください。

## 8.16 tcpudp\_get\_ramsize

## &lt; API 書式 &gt;

```
W tcpudp_get_ramsize( void )
```

## &lt; 機能 &gt;

T4 が使用するワーク領域のサイズ(RAM サイズ)をリターン値として返します。ワーク領域は、TCP の受信ウィンドウなどに使用するメモリ領域のことで、プログラム中で確保し、API の初期化関数tcpudp\_open()の引数で初期化する必要があります。ワーク領域の先頭アドレスは、4 バイト境界に配置してください。

例： tcpudp\_get\_ramsize()の戻り値が 100 の場合、T4 が使用するワーク領域 work は以下のように定義することができます。

```
UW      work[100/4];
```

本 API は、以下の目的で使用できます。

## 1) 静的な配列でワーク領域を確保する場合

本 API では、T4 のコンフィグレーションファイルで設定した内容により、T4 で使用するワーク領域のサイズを算出しますが、予めユーザ自身でこのサイズを算出するのは困難です。従って T4 のコンフィグレーションファイルの内容が決定した時点で、プログラムをビルドしてデバッグで本 API を実行して戻り値を調べます。そして戻り値が示すサイズ分、ワーク領域のメモリ配列を確保するようにします。

なお T4 のコンフィグレーションファイルを変更した場合には、必要なワーク領域のサイズが変わりますので、本 API を用いてワーク領域のサイズを再計算してください。またエラー判定処理として、初期設定の処理の中で必ず本 API を呼び出して、戻り値と（ユーザが最終的に決めた）ワーク領域のサイズを比較し、異なっていた場合はエラー処理に分岐させて、デバッグやテストの段階で間違いが発見できるようにしておいてください。

## 2) 動的メモリからワーク領域を確保する場合

アプリケーションプログラムの初期設定で本 API を呼び出してワーク領域のサイズを算出します。算出したサイズのワーク領域を動的メモリから確保し、初期化関数tcpudp\_open()に渡して初期化してください。

## &lt; 引数 &gt;

名称	型	機能
なし	なし	-

## &lt; 戻り値・エラーコード &gt;

戻り値	意味
正の値	T4 が使用するワーク領域のサイズ（単位：バイト）

## 8.17 tcpudp\_open

### < API 書式 >

```
ER tcpudp_open( UW *work )
```

### < 機能 >

T4 ライブラリの初期化を行います。ライブラリの初期化処理では、内部管理領域のメモリ割り当てとその初期化およびライブラリが使用する TCP/IP 周期処理関数の起動を行います。

T4 が使用するワーク領域の先頭アドレスを、work に指定します。必要なワーク領域のサイズは、tcpudp\_get\_ramsize() のリターン値で得られます。

### < 引数 >

名称	型	機能
*work	UW	T4 が使用するワーク領域のアドレス

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	初期化失敗

## 8.18 \_process\_tcpip

## &lt; API 書式 &gt;

```
void _process_tcpip( void )
```

## &lt; 機能 &gt;

T4 ライブラリの TCP/IP 部の処理を行います。

本 API からドライバ層へのリードライト関数など、種々ドライバ関数が呼び出されます。

本 API の処理時間は通信状態やドライバ層の実装方法によって変化します。

本 API のスタックはドライバ層の実装方法によって変化します。

本 API は任意の間隔(10msec 推奨)で定期的に起動してください。(タイマ割り込みなどを使用)

また、本 API は送受信割り込み検知でも起動してください。

T4 は時間管理を `tcpudp_get_time()` を使用して行います。`tcpudp_get_time()` の戻り値は 10msec でインクリメントされるべきで、インクリメントの間隔が 10msec になっていない場合、

- ・ API で指定したタイムアウトが所定の時間に発生しない。
- ・ 再転送の間隔が所定の間隔にならない。
- ・ ゼロウィンドウブロープの間隔が所定の間隔にならない。
- ・ コネクション切断後の 2MSL 待ち時間が所定の時間にならない。

等の問題が発生します。

データ送受信割り込みから起動が可能な場合、通信相手からの ACK 受信に呼応して次のデータを送信したり、通信相手からのデータ受信に呼応して ACK を送信したり出来るため、受信割り込みを使用しない場合に比べ通信速度が向上します。

## &lt; 引数 &gt;

名称	型	機能
なし	なし	-

## &lt; 戻り値・エラーコード &gt;

戻り値	意味
なし	-

## 8.19 tcpudp\_close

### < API 書式 >

```
ER tcpudp_close( void )
```

### < 機能 >

T4 ライブラリの終了処理を行います。ライブラリの終了処理では、ライブラリが使用する TCP/IP 周期処理関数を停止します。

本 API を呼び出す前に、TCP 通信端点を切断・未使用状態にしてください。

本 API の実行後、ライブラリ・オープン関数 `tcpudp_open()` で指定したワーク領域は開放されますので、アプリケーションプログラムでワーク領域が使用可能となります。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	終了処理失敗

## 9. Ethernet/PPP ドライバ関連の API 仕様

T4 ドライバ関連の API の一覧を表 4、表 5 および 表 6 に示します。これらの API は、ITRON TCP/IP API 仕様とは無関係の弊社オリジナルの API です。

表 4. Ethernet ドライバの API 一覧

Ethernet ドライバの API	C 言語 API	備考
Ethernet ドライバ・オープン	ER lan_open(void)	ユーザアプリケーションからコールされる API
Ethernet ドライバ・クローズ	ER lan_close(void)	
データ受信	H lan_read(UB, B **)	ライブラリ内部からコールされる API
データ送信	H lan_write(UB, B *, H, B *, H)	
Ethernet コントローラのリセット	void lan_reset(UB)	

※ 表 4 の API のプロトタイプ宣言は、r\_t4\_itcpip.h ファイルに含まれています。

表 5. PPP ドライバの API 一覧

PPP ドライバの API	C 言語 API	備考
serial I/O 開始	void sio_open(UB)	ユーザアプリケーションからコールされる API
serial I/O 終了	void sio_close(void)	
モデムの接続 (T4 が PPP クライアントの場合)	ER modem_active_open(void)	
モデムの接続 (T4 が PPP サーバの場合)	ER modem_passive_open(void)	
モデムの切断	ER modem_close(void)	
PPP 状態参照	UH ppp_status(void)	
PPP フレームの受信	H ppp_read(UB **)	ライブラリ内部からコールする API
PPP フレームの送信	H ppp_write(B *, H, B **, H *, H)	
モデムの応答コードの受信	H modem_read(UB **)	
モデムのコマンドの送信	H modem_write(void far *)	
PPP ドライバのステータス取得	UH ppp_drv_status(void)	
PPP 関連 API 完了待ち (完了チェック待ち)	void ppp_api_slp(void)	
PPP 関連 API 完了待ち解除	void ppp_api_wup(void)	
CHAP 用乱数生成	void get_random_number(UB *, UW)	

※ 表 5 の API のプロトタイプ宣言は、r\_t4\_itcpip.h ファイルに含まれています。



表 6. Ethernet/PPP ドライバ共通の API 一覧

Ethernet/PPP ドライバの API	C 言語 API	備考
受信バッファ開放	H rcv_buff_release(UB)	ライブラリ内部から コールする API
TCP 関連 API 完了待ち (完了チェック待ち)	void tcp_api_slp(ID)	
TCP 関連 API 完了待ち解除	void tcp_api_wup(ID)	
UDP 関連 API 完了待ち (完了チェック待ち)	void udp_api_slp(ID)	
UDP 関連 API 完了待ち解除	void udp_api_wup(ID)	
周期起動制御	void tcpudp_act_cyc (UB)	
時刻の取得	UH tcpudp_get_time(void)	
割り込み再開処理	void ena_int(void)	
割り込み一時停止処理	void dis_int(void)	
エラー通知	void report_error(UB, H, UB*)	

※ 表 6 の API のプロトタイプ宣言は、r\_t4\_itcpip.h ファイルに含まれています。

表 4、表 5 および表 6 に示した API は、下記の 3 つに分類できます。

1. アプリケーションプログラムの中から呼び出す必要のある API
2. T4 ライブラリ内部から呼び出される API で、アプリケーションからは呼び出す必要のない API（表中ではグレーの背景で記載）
3. ドライバ API の作成を補助する API で、アプリケーションからは呼び出す必要のない API。

本マニュアルでは、アプリケーションプログラムから呼び出す必要のある上記 1 に分類される API について、使用方法を説明します。

各 API の仕様については、別紙の「Ethernet ドライバインタフェース仕様書」「PPP ドライバインタフェース仕様書」を参照いただき、仕様に従ってドライバ関数を作成してください。

#### 【補足事項 1】

表 5 に示した API の内、関数 ppp\_open、ppp\_close、ppp\_api\_req は PPP ドライバを制御する関数ですが、実は T4 ライブラリに組み込まれています（スタックサイズは各導入ガイドを参照ください）。

これに対して、その他の API はすべて Ethernet ドライバまたは PPP ドライバとして T4 ライブラリから分離されており、各 API のサンプルソースはドライバのサンプルプログラムとして提供されます。

## 9.1 lan\_open

### < API 書式 >

```
ER lan_open( void )
```

### < 機能 >

Ethernet コントローラ、Ethernet ドライバの初期化を行います。T4 のコンフィグレーションファイルで設定するグローバル変数\_myethaddrが 0 の場合には、ROM に格納されている Ethernet アドレスを Ethernet コントローラに設定し、かつ、\_myethaddr にコピーします。\_myethaddrが 0 でない場合には、その値を Ethernet コントローラに設定します。グローバル変数\_myethaddrの設定方法は、6章を参照ください。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	初期化失敗

## 9.2 lan\_close

< API 書式 >

```
ER lan_close( void )
```

< 機能 >

Ethernet コントローラの停止、Ethernet ドライバの終了処理を行います。

< 引数 >

名称	型	機能
なし	なし	-

< 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了
負の値	終了処理失敗

### 9.3 sio\_open

#### < API 書式 >

```
void sio_open( UB rate )
```

#### < 機能 >

使用するシリアル I/O の初期設定を行い、送受信できる状態にします。ボーレートは、パラメータ `rate` に各ボーレートを示す値 (0~5) で渡されます。これらの値は、ヘッダファイル `r_t4_itcpip.h` 内で BR96~BR1152 に定義されています。

#### < 引数 >

名称	型	機能
Rate	UB	<ul style="list-style-type: none"> <li>I/O のボーレート設定 <ul style="list-style-type: none"> <li>0: (BR96) 9600bps</li> <li>1: (BR192) 19200bps</li> <li>2: (BR288) 28800bps</li> <li>3: (BR384) 38400bps</li> <li>4: (BR576) 57600bps</li> <li>5: (BR1152) 115200bps</li> </ul> </li> </ul>

#### < 戻り値・エラーコード >

戻り値	意味
なし	-

## 9.4 sio\_close

### < API 書式 >

```
void sio_close( void )
```

### < 機能 >

使用するシリアル I/O の送受信割り込みを禁止し、送受信できない状態にします。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
なし	-

## 9.5 modem\_active\_open

### < API 書式 >

```
ER modem_active_open( void )
```

### < 機能 >

モデムの初期化と、電話回線の接続を行い、モデム接続を確立します。モデムの初期化では、ユーザ指定の初期化 AT コマンド `at_commands[]` を用いてモデムの動作設定を行います。また、電話回線の接続では、モデムを用いて宛先電話番号 `peer_dial[]` に電話を掛けます。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了(モデム接続成功)
-1	異常終了(モデム接続失敗)

## 9.6 modem\_passive\_open

### < API 書式 >

```
ER modem_passive_open( void )
```

### < 機能 >

モデムの初期化と、電話回線の接続を行い、モデム接続を確立します。モデムの初期化では、ユーザ指定の初期化 AT コマンド `at_commands[]` を用いてモデムの動作設定を行います。また、電話回線の接続では、モデムを用いて着信を待ち受けます。本関数は着信があるまで待ち続けます。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了(モデム接続成功)
-1	異常終了(モデム接続失敗)

## 9.7 modem\_close

### < API 書式 >

```
ER modem_close( void )
```

### < 機能 >

モデムをデータ通信モードからコマンドモードに遷移させて、電話回線の切断を行います。

### < 引数 >

名称	型	機能
なし	なし	-

### < 戻り値・エラーコード >

戻り値	意味
E_OK	正常終了(モデム切断成功)
-1	異常終了(モデム切断失敗)



## 9.8 get\_random\_number

### < API 書式 >

```
void get_random_number(UB *data, UW len)
```

### < 機能 >

指定されたポインタに指定されたデータ長の乱数を設定します。

### < 引数 >

名称	型	機能
data	UB *	乱数を書き込むデータポインタ
len	UW	必要な乱数のバイト長

### < 戻り値・エラーコード >

戻り値	意味
なし	-

## 10. サンプルプログラム

以下の 4 パターンのエコーバックサーバのソースコードを含むプロジェクトを用意しています。

- TCP ブロッキングコール(echo\_srv\_tcp\_blocking ディレクトリ)
- TCP ノンブロッキングコール(echo\_srv\_tcp\_nonblocking ディレクトリ)
- UDP ブロッキングコール(echo\_srv\_udp\_blocking ディレクトリ)
- UDP ノンブロッキングコール(echo\_srv\_udp\_nonblocking ディレクトリ)

サンプルプログラムは共通の main 関数を持ちます。main 関数は echo\_srv()関数を呼び出します。上記 4 パターンはそれぞれエコーバックサーバの実装例です。いずれか 1 パターンのプロジェクトを選択してください。

### 10.1 main 関数のフロー

図 20. main 関数の処理フロー参照

### 10.2 TCP エコーバックサーバ関数(ブロッキングコールの場合)のフロー

#### 10.2.1 エコーバックサーバ関数

図 21. TCP のエコーバックサーバ (ブロッキングコール) の処理フロー参照

### 10.3 TCP エコーバックサーバ関数(ノンブロッキングコールの場合)のフロー

#### 10.3.1 エコーバックサーバ関数

図 22. TCP のエコーバックサーバ (ノンブロッキングコール) の処理フロー参照

#### 10.3.2 コールバック関数のフロー

図 23. TCP のコールバック (ノンブロッキングコール) の処理フロー参照

### 10.4 UDP エコーバックサーバ関数(ブロッキングコールの場合)のフロー

#### 10.4.1 エコーバックサーバ関数

図 24. UDP のエコーバックサーバ (ブロッキングコール) の処理フロー参照

### 10.5 UDP エコーバックサーバ関数(ノンブロッキングコールの場合)のフロー

#### 10.5.1 エコーバックサーバ関数

図 25. UDP のエコーバックサーバ (ノンブロッキングコール) の処理フロー参照

#### 10.5.2 コールバック関数のフロー

図 26. UDP のコールバック (ノンブロッキングコール) の処理フロー参照

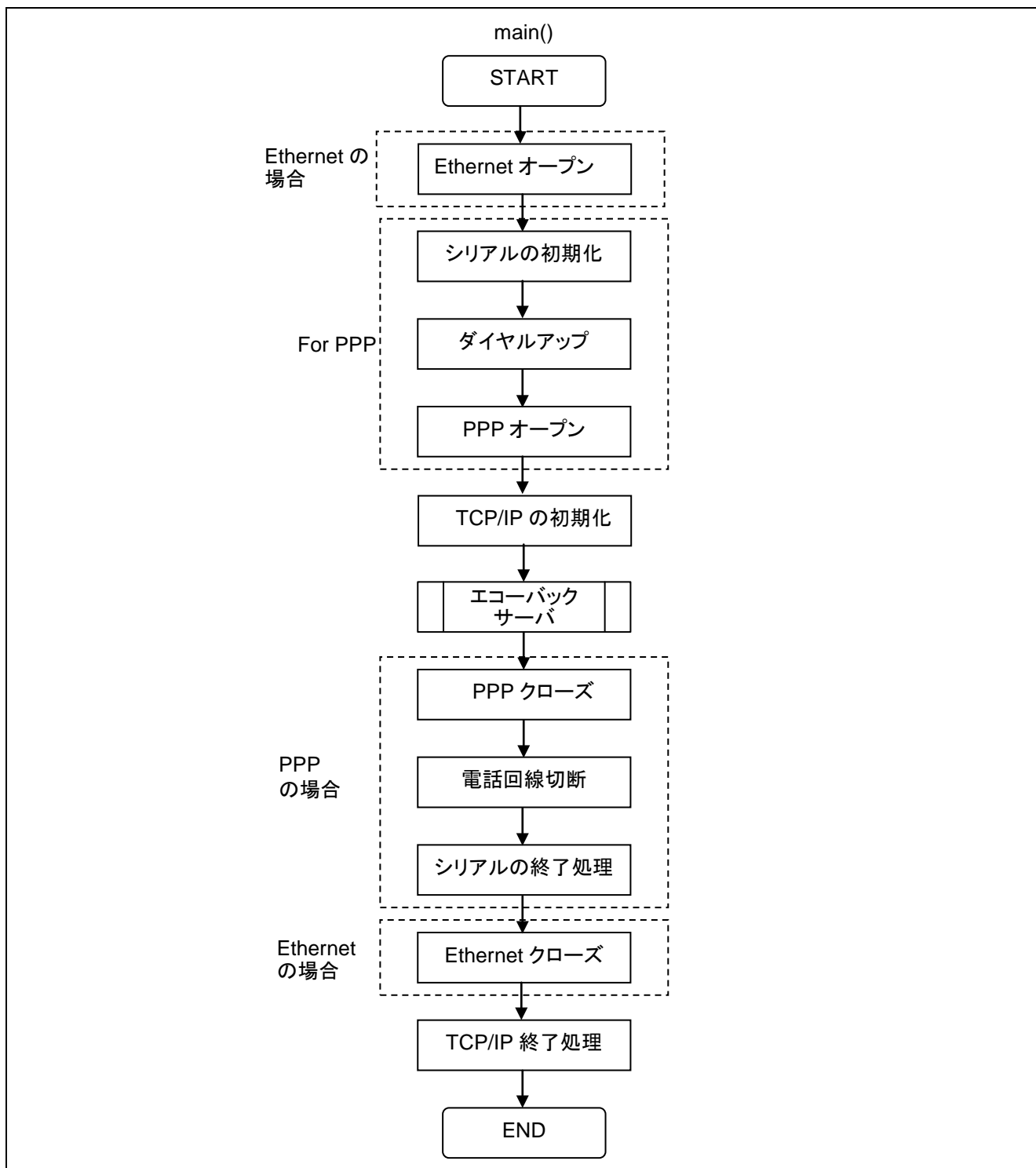
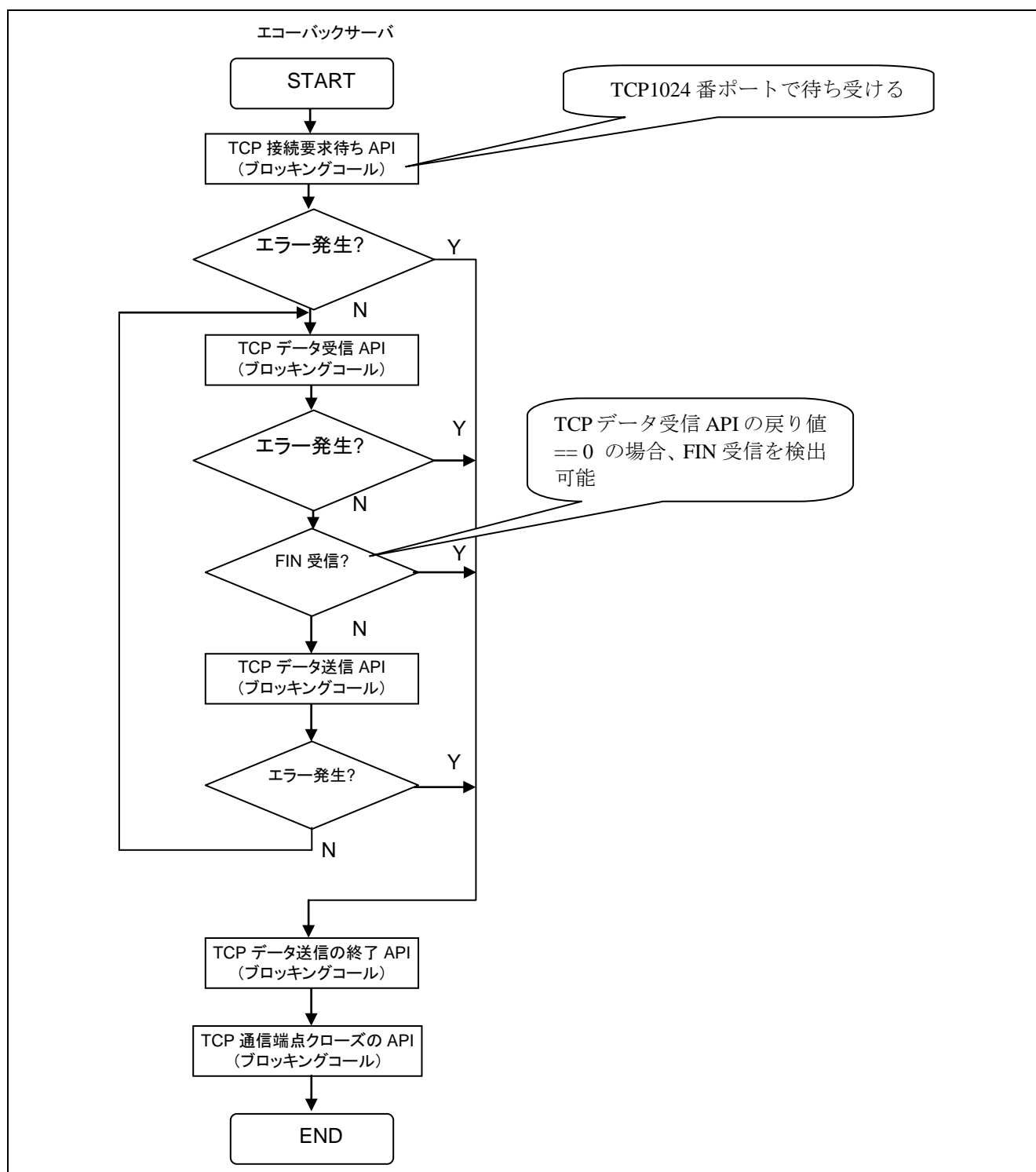


図 20. main 関数の処理フロー



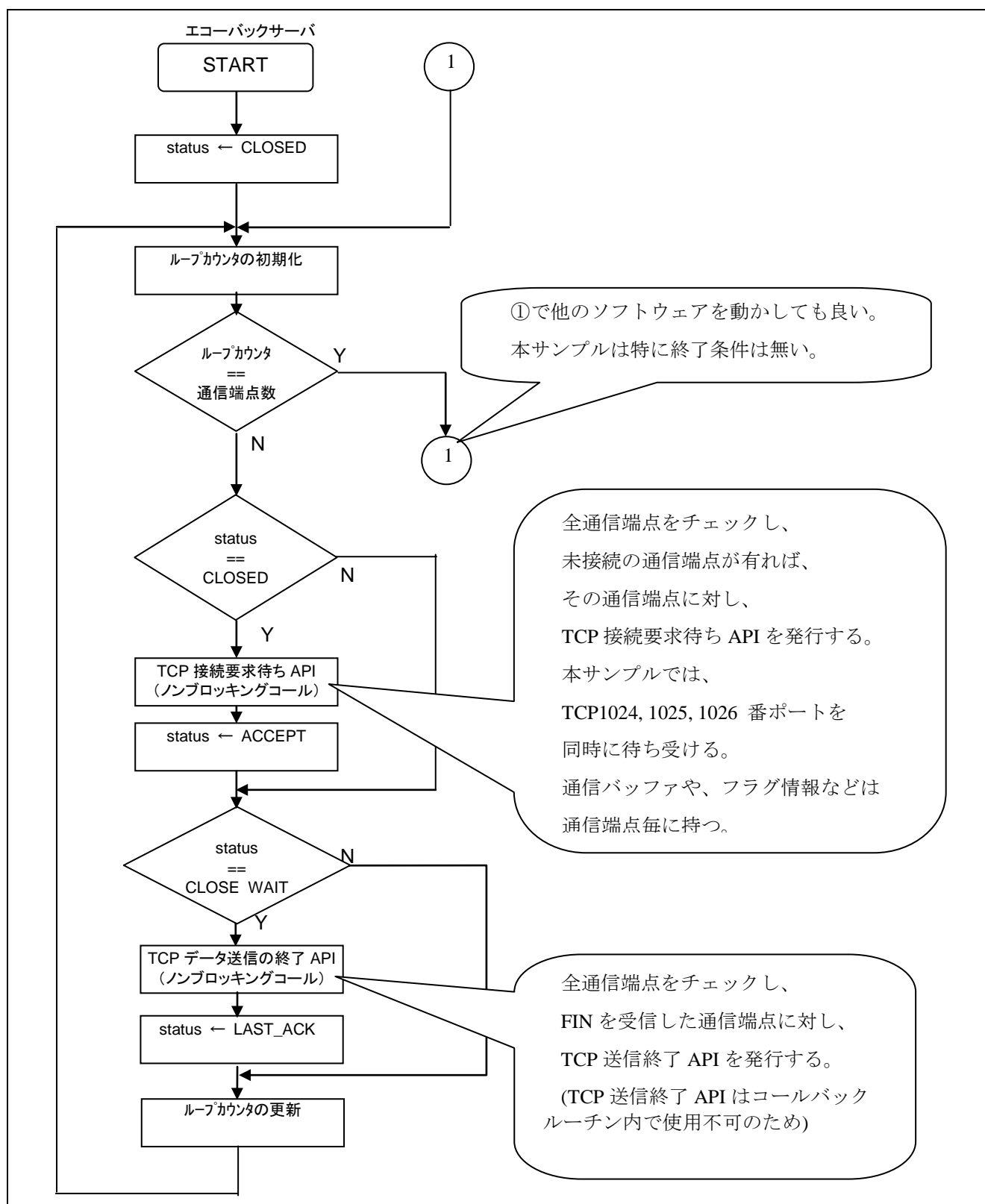


図 22. TCP のエコーバックサーバ（ノンブロッキングコール）の処理フロー

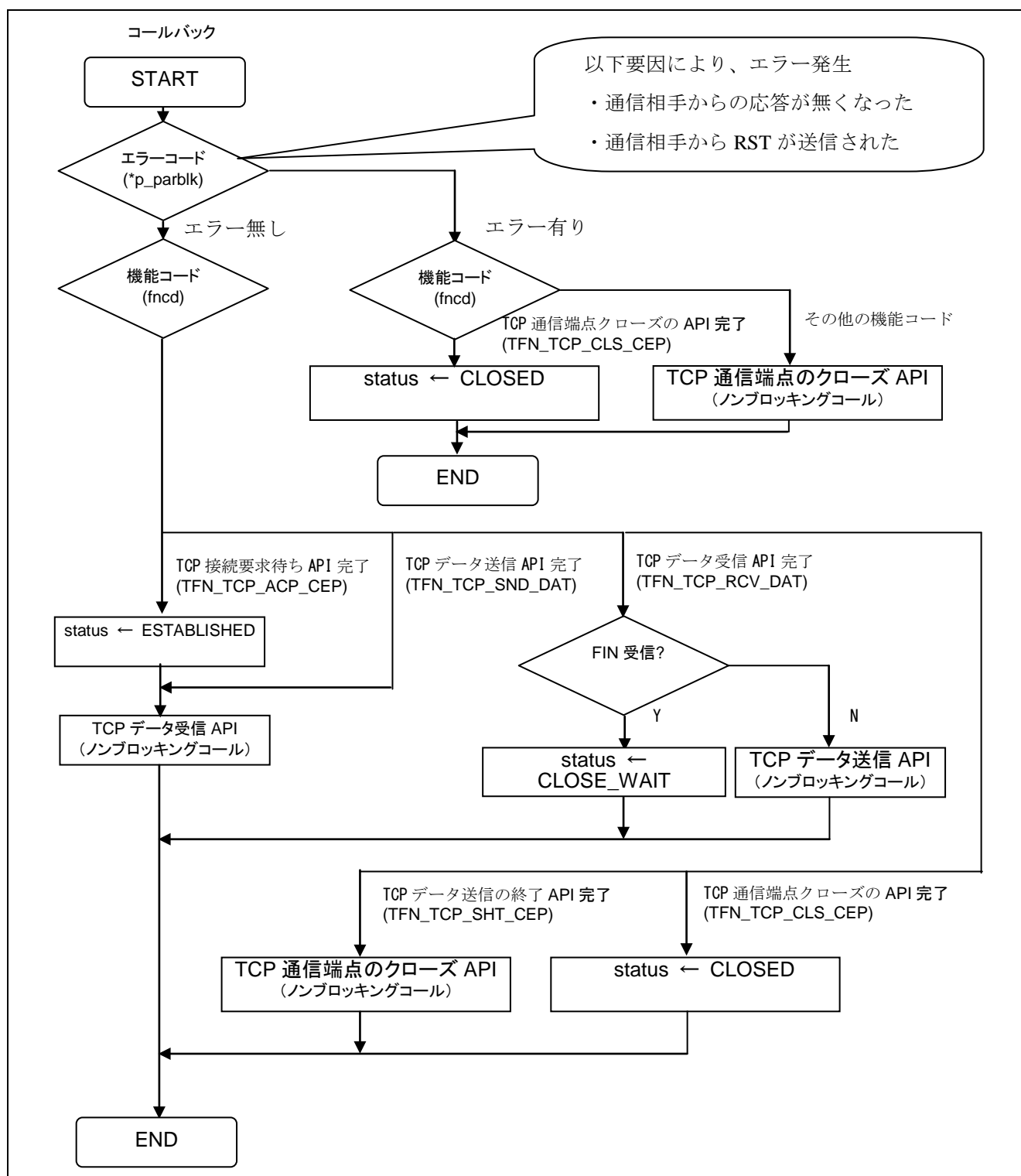


図 23. TCP のコールバック（ノンブロッキングコール）の処理フロー

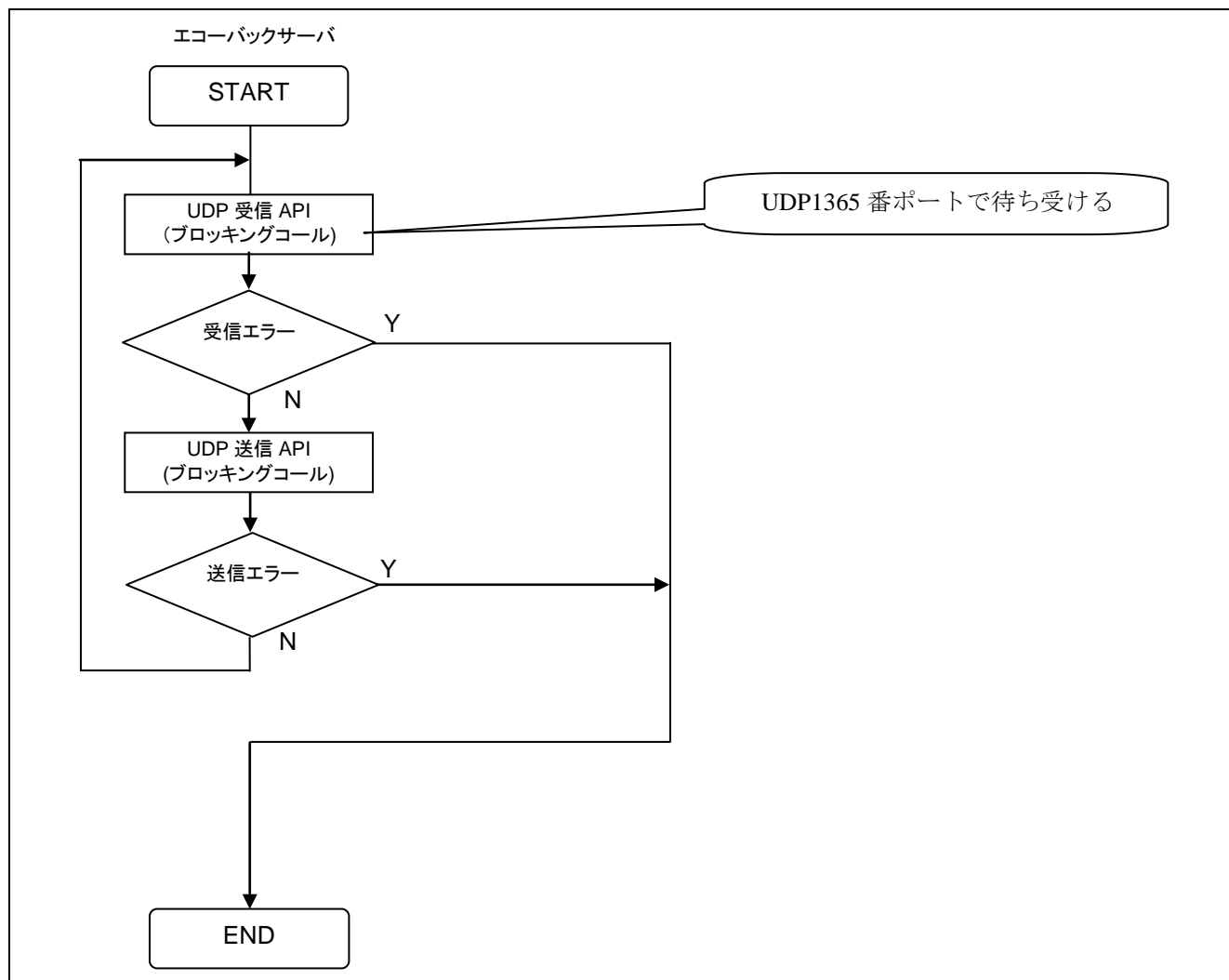


図 24. UDP のエコーバックサーバ（ブロッキングコール）の処理フロー

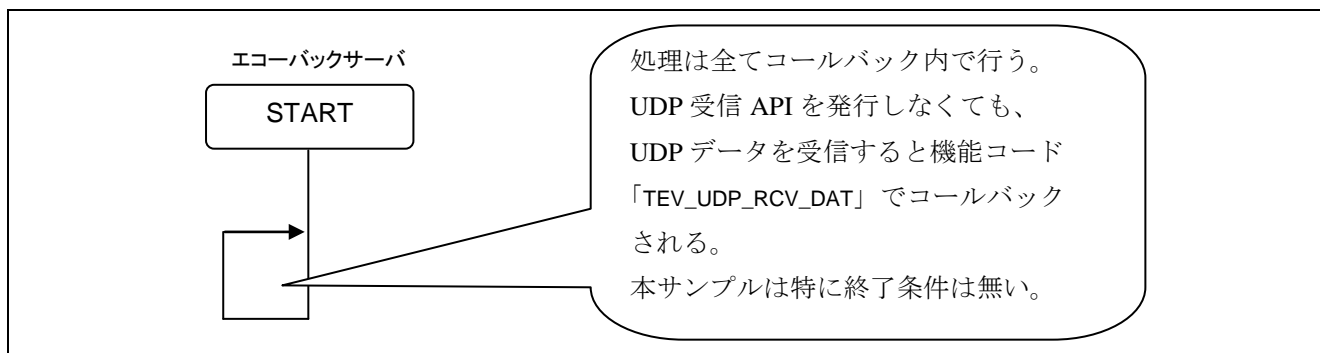


図 25. UDP のエコーバックサーバ（ノンブロッキングコール）の処理フロー

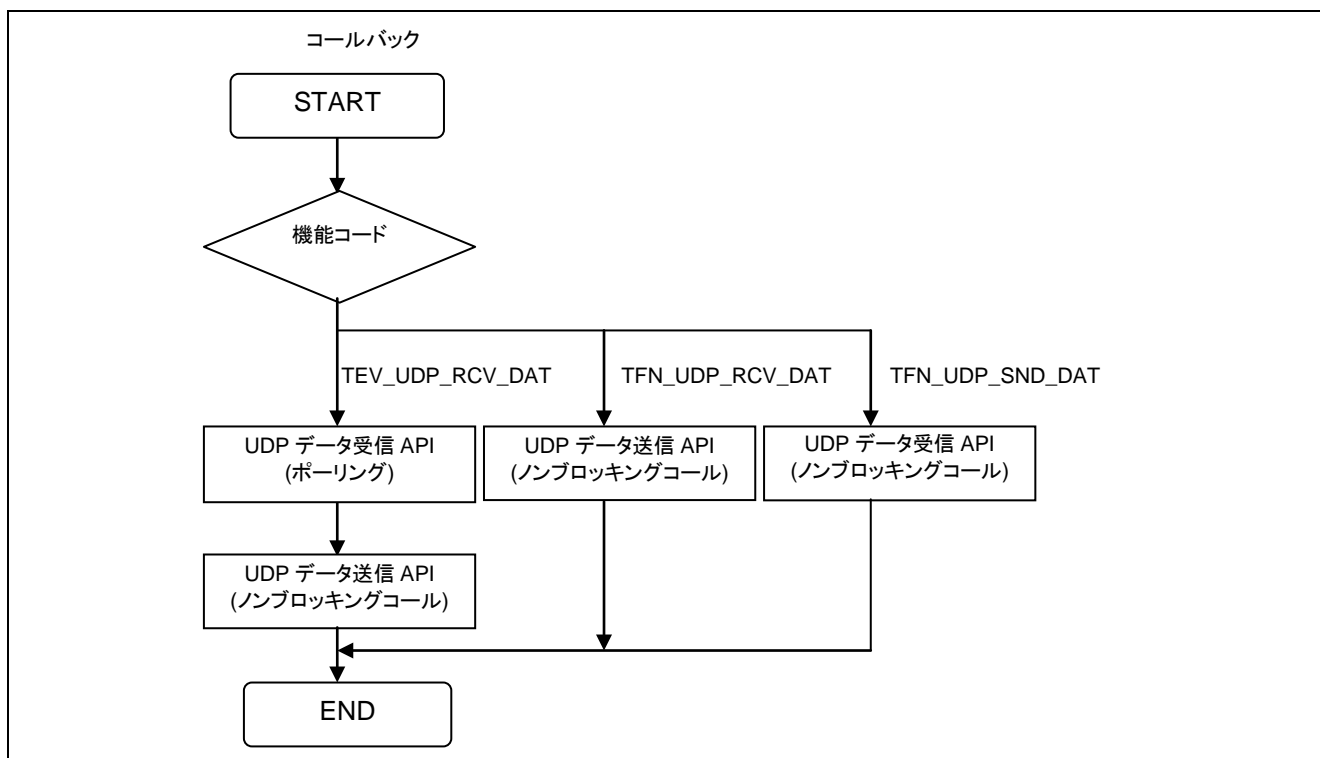


図 26. UDP のコールバック（ノンブロッキングコール）の処理フロー



## 10.6 実行環境

本サンプルプログラムを実行する場合のハードウェアの接続方法を、Ethernet については図 27 に、PPP の場合については 図 28 に示します。

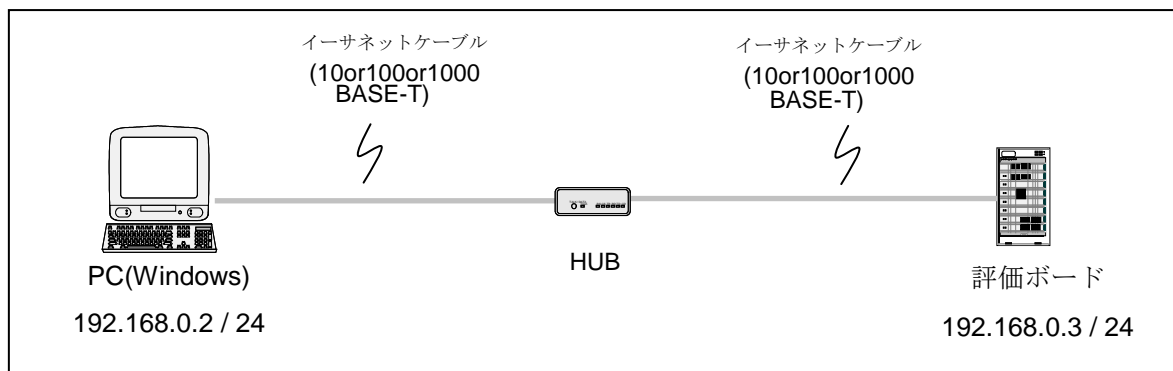


図 27. サンプルプログラムの実行環境 (Ethernet)

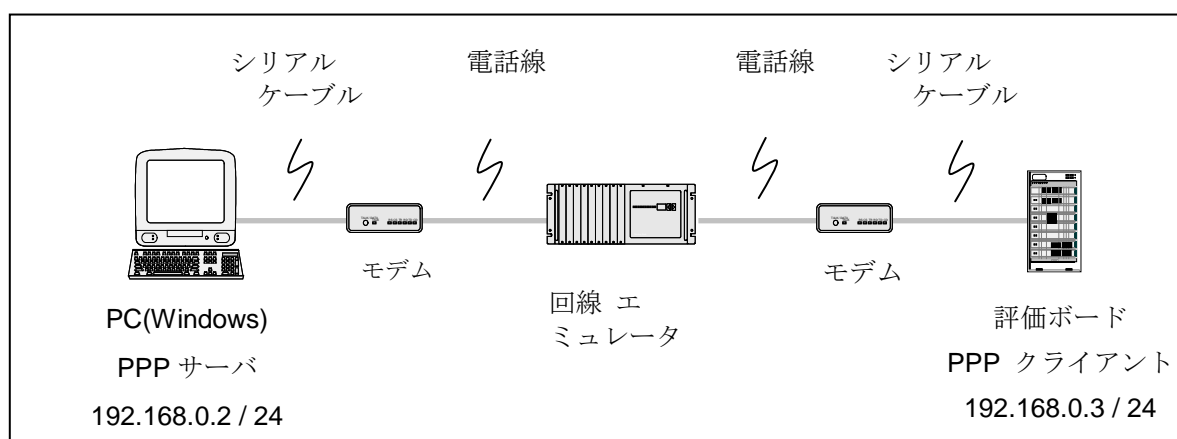


図 28. サンプルプログラムの実行環境 (PPP)

## 10.7 実行方法

以下の (1) ～ (7) に、サンプルプログラムの実行方法を示します。Ethernet の場合には (2) と (4)～(6)、PPP の場合には (1) ～ (6) に従って実行してください。

- (1) PC 上のダイヤルアップサーバの起動  
 予めダイヤルアップサーバの設定で、ユーザ名: "abcde", パスワード: "abc00" を登録しておきます。  
 また、ダイヤルアップサーバの TCP/IP の設定で、クライアントに割り当てる IP アドレスが 192.168.0.3 になるように設定しておきます。
- (2) サンプルプログラムのダウンロード  
 評価ボードにサンプルプログラムをダウンロードし、実行します。
- (3) PPP サーバへの接続処理  
 評価ボードからダイヤルアップにより PC 上の PPP サーバへ接続し、PPP のネゴシエーションを実行します。
- (4) コネクションの設定  
 【TCP ブロックリングコールの場合】  
 サンプルプログラムを実行する環境に合わせて、以下のいずれかを動作させてください。  
 PC 上の MS-DOS プロンプトで下記コマンドを実行し、コネクションを確立します。

```
telnet 192.168.0.3 1024
```

【TCP ノンブロックリングコールの場合(複数同時接続可能)】

```
telnet 192.168.0.3 1024
```

```
telnet 192.168.0.3 1025
```

```
telnet 192.168.0.3 1026
```

Windows7 をご使用の場合:

1. コントロールパネル→プログラム→プログラムと機能  
 →Windows の機能の有効化または無効化 をクリック
2. Telnet クライアント をチェックし OK をクリック

【UDP ブロックリングコールの場合】

PC 上から UDP 送受信フリーソフトを使用します。設定は以下の通り。

相手先 IP アドレス 192.168.0.3 、使用ポート番号 1365

【UDP ノンブロックリングコールの場合(複数同時通信可能)】

相手先 IP アドレス 192.168.0.3 、使用ポート番号 1365

相手先 IP アドレス 192.168.0.3 、使用ポート番号 1366

相手先 IP アドレス 192.168.0.3 、使用ポート番号 1367

- (5) 文字の送信  
 TELNET の画面上及び、UDP 送受信フリーソフト上でキーボードから文字を入力すると、その文字が評価ボードに送信されます。評価ボードでは受信した文字を返信するため、TELNET の画面上及び、UDP 送受信フリーソフト上に返信された文字が表示されます。  
 UDP データが送受信可能なフリーソフト:  
 Socket Debugger Free <http://sdg.ex-group.jp/distinction.html>
- 
- (6) コネクションの切断と再接続 (TCP のみ)  
 TELNET 画面で CTRL+] を押下して "quit" と打ち込むと切断できます。  
 再接続には (4) から繰り返してください。

## 10.8 実行環境 (複数 LAN ポート)

本サンプルプログラムを実行する場合のハードウェアの接続方法を、Ethernet については図 29 に示します。

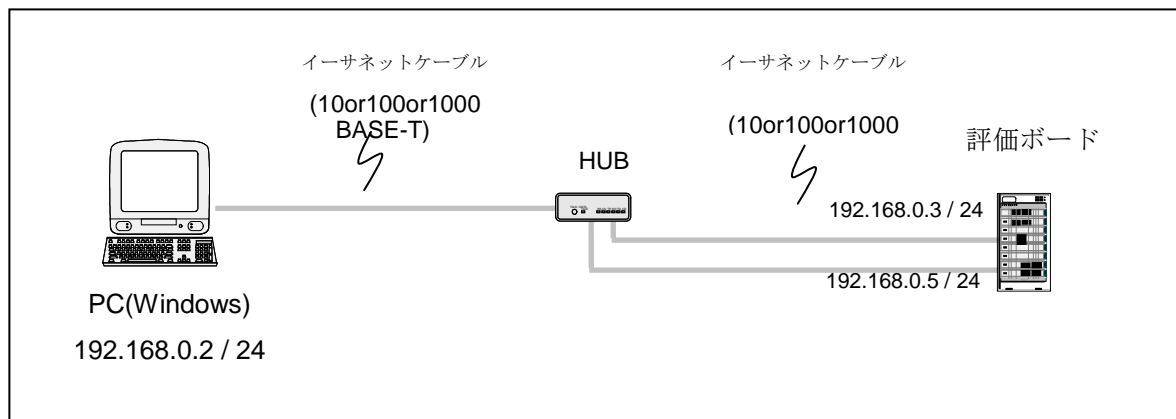


図 29. サンプルプログラムの実行環境 (複数 LAN ポート)

## 10.9 実行方法 (複数 LAN ポート)

- (1) サンプルプログラムのダウンロード  
評価ボードにサンプルプログラムをダウンロードし、実行します。
- (2) コネクションの設定

### 【TCP ブロックリングコールの場合】

サンプルプログラムを実行する環境に合わせて、以下のいずれかを動作させてください。

PC 上の MS-DOS プロンプトで下記コマンドを実行し、コネクションを確立します。

```
telnet 192.168.0.3 1024
telnet 192.168.0.10 1024
```

### 【TCP ノンブロックリングコールの場合】

```
telnet 192.168.0.3 1024
telnet 192.168.0.3 1025
telnet 192.168.0.3 1026
```

```
telnet 192.168.0.10 1024
telnet 192.168.0.10 1025
telnet 192.168.0.10 1026
```

Windows7をご使用の場合:

1. コントロールパネル→プログラム→プログラムと機能  
→Windowsの機能の有効化または無効化 をクリック
2. Telnet クライアント をチェックし OK をクリック

### 【UDP ブロックリングコールの場合】

PC 上から UDP 送受信フリーソフトを使用します。設定は以下の通り。

```
相手先 IP アドレス 192.168.0.3 、使用ポート番号 1365
相手先 IP アドレス 192.168.0.10 、使用ポート番号 1365
```

### 【UDP ノンブロックリングコールの場合(複数同時通信可能)】

```
相手先 IP アドレス 192.168.0.3 、使用ポート番号 1365
相手先 IP アドレス 192.168.0.3 、使用ポート番号 1366
相手先 IP アドレス 192.168.0.3 、使用ポート番号 1367
```

```
相手先 IP アドレス 192.168.0.10 、使用ポート番号 1365
相手先 IP アドレス 192.168.0.10 、使用ポート番号 1366
相手先 IP アドレス 192.168.0.10 、使用ポート番号 1367
```

- (3) 文字の送信

TELNET の画面上及び、UDP 送受信フリーソフト上でキーボードから文字を入力すると、その文字が評価ボードに送信されます。評価ボードでは受信した文字を返信するため、TELNET の画面上及び、UDP 送受信フリーソフト上に返信された文字が表示されます。

UDP データが送受信可能なフリーソフト:

Socket Debugger Free <http://sdg.ex-group.jp/distinction.html>

- 
- (4) コネクションの切断と再接続 (TCP のみ)  
TELNET 画面で CTRL+] を押下して” quit” と打ち込むと切断できます。  
再接続には(4)から繰り返してください。

## 11. T4 制限・注意事項

T4 の制限・注意事項を以下に示します。

- (1) 関数 `tcp_get_buf()`, `tcp_snd_buf()`, `tcp_rcv_buf()`, `tcp_rel_buf()` はサポートしていません。
- (2) 緊急データの送受信関数 `tcp_snd_oob()`, `tcp_rcv_oob()` はサポートしていません。
- (3) オプションの設定/取得の関数 `tcp_set_opt()`, `tcp_get_opt()`, `udp_set_opt()`, `udp_get_opt()` はサポートしていません。
- (4) TCP の API では、宛先 IP アドレスにマルチキャストアドレス、ブロードキャストアドレス、ループバックアドレス、自局の IP アドレスを設定できません。
- (5) UDP の API では、宛先 IP アドレスにループバックアドレス、自局の IP アドレスを設定できません。
- (6) UDP 受信関数 `udp_rcv_dat()` のポーリング(TMO\_POL)指定は、事象コード `TEV_UDP_RCV_DAT` により呼び出されたコールバック関数の中でのみ可能です。それ以外でポーリング(TMO\_POL)指定した場合には、パラメータエラー(E\_PAR)が返ります。
- (7) IGMP をサポートしていませんので、ルータに対して宛先 IP アドレスがマルチキャストアドレスの IP データグラムの転送を要求することはできません。
- (8) T4 で同時に実行可能な API の数は、TCP と UDP で各々、端点毎に 1 つです。同時に複数の API を実行した場合、戻り値に `E_QOVR` が返ります。
- (9) TCP では、ヘッダオプションは MSS のみサポートしています。TCP セグメントを受信した場合、MSS 以外のオプションについては無視します。
- (10) IP では、IP オプション、フラグメントはサポートしていません。受信した IP データグラムに IP オプションが含まれている、または、フラグメント化されている場合には、そのデータグラムを破棄します。
- (11) ICMP では、エコー要求の受信とそれに対するエコー応答の送信のみサポートしています。その他の ICMP メッセージを受信した場合、そのパケットを破棄します。
- (12) PPP では、圧縮関連のオプション(プロトコルフィールド圧縮、アドレスと制御フィールド圧縮、TCP/IP ヘッダ圧縮) はサポートしていません。圧縮関連のオプションの設定要求を受信した場合には、設定拒否の応答を送信します。
- (13) 関数 `tcpudp_close()` をコールする場合、コール前に通信端点を切断・未使用の状態にしてください。通信状態で `tcpudp_close()` をコールした場合、動作は未定義です。
- (14) API の引数に指定する `T_IPV4EP` 構造体は、グローバル変数に配置してください。

## 付録 A.TCP 用 API の戻り値

### A1. 意図せず回線が切断された場合の API の動作

ケーブルが外れる等して回線が切断された場合、API に通知は行いません。API は、通信相手がハングアップ/リブート等して正常に通信できなくなった場合と同様の動作（戻り値）となります。引数でタイムアウトを設定する API については、タイムアウトにより異常を検知できます。回線が切断された場合に API へ戻り値が返るのは、基本的には、

(time1) APIの引数で指定したタイムアウトに達した時点

または、

(time2) TCPセグメントを送信中に再転送タイムアウトの最大値に達した時点

です。

APIへ戻り値が返るタイミングとしては、APIでタイムアウトにTMO\_FEVRを指定した場合は(time2)に該当します。それ以外の場合は、(time1)または(time2)の時間が短い方に該当します。このため、(time2)の時間の方が短い場合、(time2)で指定した時間に達した時点でタイムアウトし、戻り値は(time2)のものとなります。

## A2. 各 TCP 用 API の戻り値と通信端点の状態

上記 A1. に示す(time1)の場合、(time2)の場合、相手から RST を受信した場合(R)の各 API の戻り値と通信端点の状態について説明します。

tcp\_acp\_cep()

【戻り値】

time1: E\_TMOUT  
time2: 戻り値は返さない  
R: 戻り値は返さない

【通信端点の状態】

未使用状態  
引き続き待ち状態(LISTEN)になる  
引き続き待ち状態(LISTEN)になる

tcp\_con\_cep()

【戻り値】

time1: E\_TMOUT  
time2: E\_CLS  
R: E\_CLS

【通信端点の状態】

未使用状態  
未使用状態  
未使用状態

tcp\_cls\_cep()

【戻り値】

time1: E\_TMOUT  
time2: E\_OK  
R: E\_OK

【通信端点の状態】

未使用状態  
未使用状態  
未使用状態

tcp\_snd\_dat() ※1

【戻り値】

time1: E\_TMOUT  
time2: E\_CLS  
R: E\_CLS

【通信端点の状態】

接続状態  
切断状態  
切断状態

tcp\_rcv\_dat()

【戻り値】

time1: E\_TMOUT  
time2: データ等を送信しないので発生しない  
R: E\_CLS

【通信端点の状態】

接続状態  
切断状態※2

[( ※1 )] tcp\_snd\_dat()の戻り値

通信相手から FIN を受信しても、自局から FIN を送信するまではデータを送信できます。このため、相手から FIN を受信しても、tcp\_snd\_dat()は戻り値 E\_CLS を返しません。  
相手から RST を受信した場合は、戻り値 E\_CLS を返します。

[( ※2 )] tcp\_rcv\_dat()の RST 受信時の処理

通信相手から RST を受信した場合、受信ウィンドウのデータをすべて読み出してから戻り値 E\_CLS を返します。受信ウィンドウからデータを読み出している間は、戻り値は読み出したデータサイズとなります。

【補足 1】 tcp\_sht\_cep()の戻り値

ペンディング状態にならないため、回線が切断されていても通常と同じ戻り値が返ります。

### A3. 各状態における TCP 用 API の戻り値と動作

表. CLOSED 状態における各 API の戻り値と動作

CLOSED	呼出条件	戻り値	動作
tcp_acp_cep()	1	E_OK	接続確立後、API から戻る
	2	E_WBLK	接続確立後、E_OK でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	接続確立後、E_OK でコールバックルーチンが呼ばれる
tcp_con_cep()	1	E_OK	接続確立後、API から戻る
	2	E_WBLK	接続確立後、E_OK でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	接続確立後、E_OK でコールバックルーチンが呼ばれる
tcp_snd_dat()	1	E_OBJ	API コール後最初の周期処理後、API から戻る
	2	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
tcp_rcv_dat()	1	E_OBJ	API コール後最初の周期処理後、API から戻る
	2	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
tcp_sht_cep()	5	E_OBJ	API コール後最初の周期処理後、API から戻る
	6	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	7	E_NOSPT	API コール後、すぐに戻る
	8	E_NOSPT	API コール後、すぐに戻る
tcp_cls_cep()	1	E_OBJ	API コール後最初の周期処理後、API から戻る
	2	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	3	E_OBJ	API コール後最初の周期処理後、API から戻る
	4	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
tcp_can_cep()	5	-	API コール後、すぐに戻る
	6	E_OK	キャンセル完了後、E_RLWAI でキャンセル指定した端点のコールバックルーチンが呼ばれる
	7	E_NOSPT	API コール後、すぐに戻る
	8	E_NOSPT	API コール後、すぐに戻る

呼出条件 : 1. メイン内で TMO\_FEVR  
2. メイン内で TMO\_NBLK  
3. コールバック内で TMO\_FEVR  
4. コールバック内で TMO\_NBLK  
5. メイン内でブロッキングコール(コールバック登録無し)  
6. メイン内でノンブロッキングコール(コールバック登録有り)  
7. コールバック内でブロッキングコール(コールバック登録無し)  
8. コールバック内でノンブロッキングコール(コールバック登録有り)

※ 1～4 はタイムアウト引数有りの API の条件、5～8 はタイムアウト引数無しの API の条件



表. ESTABLISHED 状態における各 API の戻り値と動作

ESTABLISHED	—	戻り値	動作
tcp_acp_cep()	1	E_OBJ	API コール後最初の周期処理後、API から戻る
	2	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
tcp_con_cep()	1	E_OBJ	API コール後最初の周期処理後、API から戻る
	2	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	API コール後最初の周期処理後、E_OBJ でコールバックルーチンが呼ばれる
tcp_snd_dat()	1	Positive value	正常送信完了後、API から戻る
	2	E_WBLK	正常送信完了後、正数でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	正常送信完了後、正数でコールバックルーチンが呼ばれる
tcp_rcv_dat()	1	Positive value	正常受信完了後、API から戻る
	2	E_WBLK	正常受信完了後、正数でコールバックルーチンが呼ばれる
	3	E_PAR	API コール後、すぐに戻る
	4	E_WBLK	正常受信完了後、正数でコールバックルーチンが呼ばれる
tcp_sht_cep()	5	E_OK	API コール後最初の周期処理後、API から戻る
	6	E_WBLK	API コール後最初の周期処理後、E_OK でコールバックルーチンが呼ばれる
	7	E_NOSPT	API コール後、すぐに戻る
	8	E_NOSPT	API コール後、すぐに戻る
tcp_cls_cep()	1	E_OK	正常切断後、API から戻る
	2	E_WBLK	正常切断後、E_OK でコールバックルーチンが呼ばれる
	3	E_OK	正常切断後、API から戻る
	4	E_WBLK	正常切断後、E_OK でコールバックルーチンが呼ばれる
tcp_can_cep()	5	-	API コール後、すぐに戻る
	6	E_OK	キャンセル完了後、E_RLWAI でキャンセル指定した端点のコールバックルーチンが呼ばれる
	7	E_NOSPT	API コール後、すぐに戻る
	8	E_NOSPT	API コール後、すぐに戻る

呼出条件 : 1. メイン内で TMO\_FEVR

2. メイン内で TMO\_NBLK

3. コールバック内で TMO\_FEVR

4. コールバック内で TMO\_NBLK

5. メイン内でブロッキングコール(コールバック登録無し)

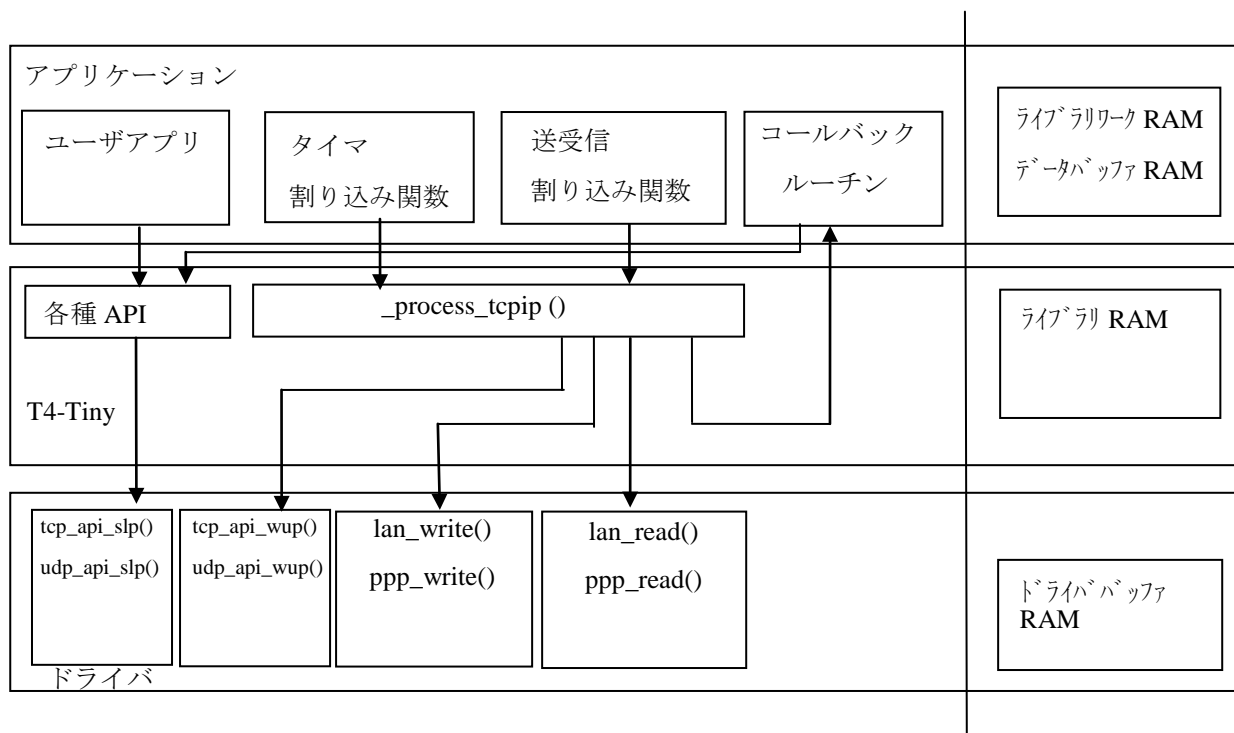
6. メイン内でノンブロッキングコール(コールバック登録有り)

7. コールバック内でブロッキングコール(コールバック登録無し)

8. コールバック内でノンブロッキングコール(コールバック登録有り)

※ 1~4 はタイムアウト引数有りの API の条件、5~8 はタイムアウト引数無しの API の条件

付録 B. T4 ソフトウェア構造



## ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

<http://japan.renesas.com/>

お問合せ先

<http://japan.renesas.com/contact/>

すべての商標および登録商標は、それぞれの所有者に帰属します。

## 改訂記録

T4 ver.	Rev.	発行日	改訂内容	
			ページ	ポイント
2.00	1.06	2014.04.01	-	複数 LAN ポートに対応しました。
1.06	1.05	2013.06.21	10	ライブラリの型定義表を修正しました。
			36	udp_snd_dat()の仕様を変更しました。
			40	udp_rcv_dat()の仕様を変更しました。
			44	コールバックルーチンの情報を追記しました。
			49	T4 制限・注意事項を変更しました。
1.05	1.04	2012.04.01	-	api_wup()、api_slp()の仕様を変更しました。 modem_open()の仕様を変更しました。 _process_tcpip()の仕様を変更しました。 tcp_rcv_dat()でメイン内 TMO_POL 指定を可能にしました。 PPP サーバ、PPP クライアントの切り替え機能を実装しました。 PPP の認証に CHAP(MD5)を追加しました。
1.04	1.03	2011.08.23	20	UDP ゼロチェックサムの動作定義を追加しました。
			44	ユーザ定義関数の report_error()を追加しました。
			54	フロー処理を修正しました。
-	1.02	2011.01.24	10	ライブラリの型定義表を修正しました。
1.01	1.01	2011.01.06	—	api_wup()の仕様を変更しました。
1.00	1.00	2010.10.07	—	初版発行

## 製品ご使用上の注意事項

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

### 1. 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（予約領域）のアクセス禁止

【注意】リザーブアドレス（予約領域）のアクセスを禁止します。

アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

### 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。

### 5. 製品間の相違について

【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。

同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

## ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、  
家電、工作機械、パーソナル機器、産業用ロボット等

高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、  
防災・防犯装置、各種安全装置等

当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図し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損害が生じて、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。

6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は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、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。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。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

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と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。



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 〒100-0004 千代田区大手町2-6-2（日本ビル）

■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：<http://japan.renesas.com/contact/>