TINET-1.3 におけるイーサネットの実装

1　はじめに

TINET リリース 1.3 におけるイーサネットの実装を解説し、TINET 用のイーサネット・ディバイスドライバの実装例を示します。TOPPERS/JSP のリリースは 1.4.2 で、例とするシステムは、品川通信計装サービス製 NKEV-010H8 ボードです。本ボードに搭載されている NE2000 互換 NIC（Realtek RTL8019AS）用イーサネット・ディバイスドライバを実装例として解説します。

なお、TINET リリース 1.3 のソースを元に解説していますが、割り込み関係以外は、ほぼ、TINET リリース 1.2 でも変わりはありません。

2　TINET の概要

2.1　TINET のディレクトリとファイルの構成

リスト 1 に、TINET のディレクトリ構成を示します。主要なディレクトリと含まれているファイルの概要を解説しますが、以後、$() で囲われたディレクトリは総称名です。意味と品川通信計装サービス製 NKEV-010H8 ボードの固有名を以下に示します。

- $(CPU) はプロセッサの総称名、h8
- $(SYS) はシステムの総称名、nkev_101h8
- $(NIC) はイーサネット・ディバイスドライバの総称名、if_ed

他に$(UNAME) はアプリケーションプログラム名を意味しています。以下に、イーサネットとイーサネット・ディバイスドライバに関係するファイルの概要を示します。

(1) 汎用ネットワーク制御に関するファイル

ディレクトリ tinet/net には、ネットワークシステムに依存しない汎用ネットワーク制御に関するファイルが入っています。

(2) TCP/IP プロトコルスタックのファイル

リスト 1　TINET のソースのディレクトリ構成

```
config          # JSP ターゲット依存部
  /h8           # H8 プロセッサ依存部
    /nkev_010H8 # NKEV-010H8 システム依存部
tinet           # TINET のルートディレクトリ
  /net          # 汎用ネットワークのソース
  /netinet      # TCP/IP 本体のソース
  /netinet6     # IPv6 のソース
  /netdev       # ディバイスドライバのルートディレクトリ
    /if_ed      # NE2000 互換 NIC ドライバのソース
  /netapp       # サンプルアプリケーションプログラムのソース
  /cfg          # TINET コンフィギュレータ
  /doc          # ドキュメント類
echos           # IPv4 用 TCP ECHO サーバのサンプル
nserv           # クライアントサーバ・プログラムのサンプル
sample1         # JSP のサンプルプログラム
sample1n        # JSP のサンプルプログラム sample1 のネットワーク対応
```

表 1　TINET 内部パラメータ調整用ファイル

ファイル名	ディレクトリ	定義内容
tinet_config.h	tinet	以下のファイルをインクルードします。
tinet_app_config.h	$(UNAME)	アプリケーションプログラムの依存定義
tinet_cpu_config.h	config/$(CPU)	プロセッサ依存定義
tinet_sys_config.h	config/(CPU)(SYS)	システム依存定義
tinet_nic_config.h	tinet/netdev/$(NIC)	イーサネットディバイスドライバ依存定義

表 2　TINET に対するハードウェア依存性定義ファイル

ファイル名	ディレクトリ	定義内容
tinet_defs.h	tinet	以下のファイルをインクルードします。
tinet_cpu_defs.h	config/$(CPU)	プロセッサ依存定義
tinet_nic_defs.h	tinet/netdev/$(NIC)	イーサネット・ディバイスドライバ依存定義

ディレクトリ tinet/netinet には、TCP、UDP と IPv4 プロトコルスタックのファイルが入っています。また、ディレクトリ tinet/netinet6 には、IPv6 プロトコルスタックのファイルが入っています。

(3) イーサネット・ディバイスドライバに関するファイル

イーサネット・ディバイスドライバに関するファイルの中で、JSP ターゲットに依存しないファイルは、ディレクトリ tinet/netdev/$(NIC) に、JSP ターゲットに依存するファイルは、ディレクトリ config/$(CPU) と config/$(CPU)$(SYS) に入っています。

(4) TINET 内部パラメータ調整用ファイル

JSP では、ターゲット依存部で提供すべきカーネル用のデータ型、関数及び定数は、cpu_config.h と sys_config.h により指定します。TINET にも、これらと同様のファイルがあります。ただし、JSP とは少し性格が異なり、TINET 内部パラメータ調整用ファイルで、ディレクトリ tinet にある tinet_config.h で一括してインクルードしています。表 1 に、ファイル名、ディレクトリ、及び定義内容を示します。詳しい内容は、ディレクトリ tinet/doc にある tinet_config.txt を参照してください。

(5) TINET に対するハードウェア依存性定義ファイル

JSP と同様、TINET に対するハードウェア依存性を定義するファイルがあり、ディレクトリ tinet にある tinet_defs.h で一括してインクルードしています。表 2 に、ファイル名、ディレクトリ、及び定義内容を示します。詳しい内容は、ディレクトリ tinet/doc にある tinet_defs.txt を参照してください。

(6) アプリケーションプログラムに関係するファイル

アプリケーションプログラムの Makefile は、ディレクトリ sample にある Makefile.tinet を元に作成しなければなりません。詳細については TINET ユーザスマニュアル（tinet.txt、tinet.pdf）を参照してください。作成されたアプリケーションプログラムの Makefile には、リスト 2 に示す内容が含まれています。ネットワークインタフェースがイーサネットの場合は NET_IF = ether を選択します。これにより、コンパイルオプション SUPPORT_ETHER が有効になります。また、NET_DEV には、イーサネット・ディバイスドライバを指定します。

リスト 2　アプリケーションプログラムの Makefile

```
#   ネットワークインタフェースの選択、
#   何れか一つ選択する。
#NET_IF = loop
#NET_IF = ppp
NET_IF = ether

#   イーサネット・ディバイスドライバの選択
NET_DEV = if_ed

#   トランスポート層の選択
SUPPORT_TCP = true
SUPPORT_UDP = true

...

#   ミドルウェアの Makefile のインクルード
include $(SRCDIR)/tinet/Makefile.tinet
```

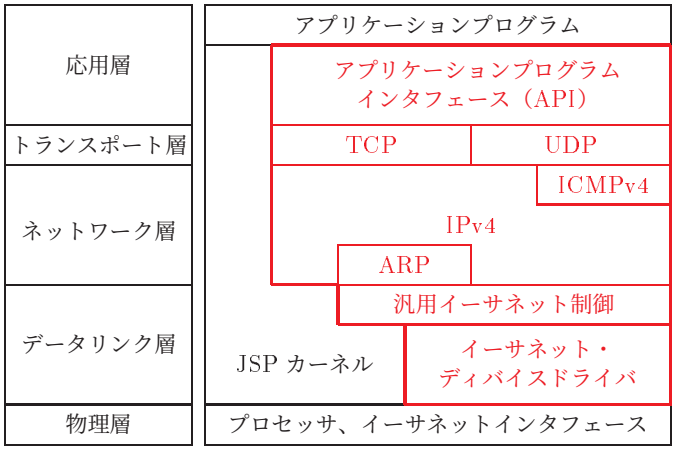

図 1　IPv4 のネットワーク階層構成

2.2　TINET のネットワーク階層構成

ネットワーク層として IPv4、ネットワークインタフェースとしてイーサネットを組み込んだ場合の TINET のネットワーク階層構成を図 1 に示します。また、ネットワーク層として IPv6 を組み込んだ場合の TINET のネットワーク階層構成を図 2 に示します。　　で囲った部分が、TINET で実装した階層です。TINET は、プロセッサやイーサネットインタフェースへの依存性をできるだけ排除し、多様な RTOS に対応するため、RTOS カーネルとアプリケーションプログラムの中間のミドルウェアとして実装しており、TINET 内でも、μITRON 4.0 仕様のスタンダードプロファイルに準拠した RTOS カーネルが提供する各種サービスを利用しています。

2.3　ネットワーク用メモリブロックの実装

TINET では、BSD UNIX の TCP/IP プロトコルスタックの mbuf に相当するメモリブロックとして、固定長のネットワークバッファを使用しています。TCP の接続確立時に、最大セグメントサイズ（MSS）オプションが付加される以外は、IP と TCP のオプションは付加されません。従って、メモリブロックを割り当てるとき、あらかじめ各階層のヘッダとペイロード長が明らかであるため、固定長のメモリブロックであるネットワークバッファを割当てます。また、TCP/IP プロトコルスタック内での処理中に動的なメモリ割当ても行

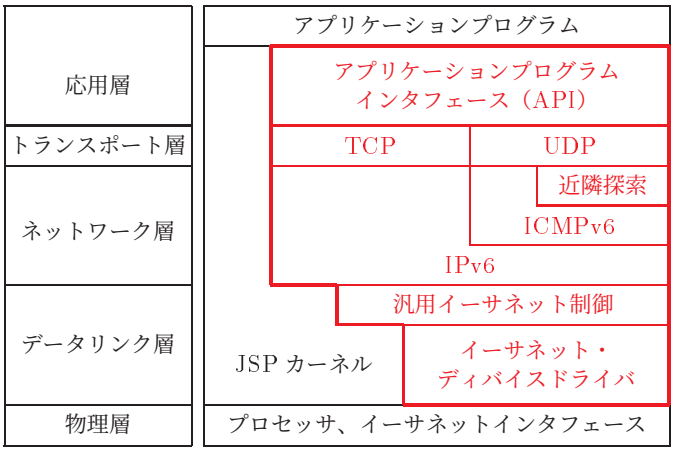

図 2　IPv6 のネットワーク階層構成

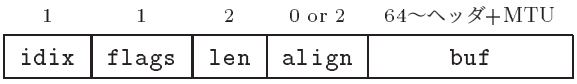

図 3　TINET のネットワークバッファの構造

いません。ネットワークバッファの構造を図 3 に示します。

- `idix` は、固定長メモリプールの ID 番号等の情報を保持する配列のインデックスです。固定長メモリプールの ID 番号以外にも、TINET で管理すべき情報があるため、間接的に固定長メモリプールを参照するようにしています。また、固定長メモリプールの ID 番号には 4 バイト必要になるため、配列のインデックスとした方がサイズの節約にもなります。
- `flags` は、ネットワークバッファの制御用で、TINET-1.3 からは、ネットワークインタフェース出力後にネットワークバッファを開放しないフラグを追加しました。
- `len` は、データ長で、種類は 64、128、256、512、1,024 と、ネットワークインタフェースの MTU にヘッダを加えた長さです。種類と個数は、μITRON のリアルタイム OS のシステムコンフィグレーションファイルで、静的 API「固定長メモリプールの生成」により指定します。
- `align` は、IP ヘッダ以降を 4 バイト境界に整列するためのダミーです。TCP/IP の各ヘッダは、32 ビット単位で処理すると効率が良いため、BSD UNIX の TCP/IP プロトコルスタックでは、4 バイト境界にあることを前提にしています。また、32 ビット境界に整合する必要のあるプロセッサも存在します。
- `buf` は、ヘッダとペイロードが入ります。受信時には、ネットワークインタフェースの MTU のペイロードとヘッダを保持できる大きさの `buf` から構成されるネットワークバッファを割当てます。例えば、イーサネットで、`buf` の大きさは、標準的な MTU の 1,500 オクテットにイーサネットヘッダの 14 オクテットを加えた 1,514 オクテットです。

表 3　TINET の汎用ネットワークインタフェース制御マクロ

マクロ名	設定内容
MAX_IF_MADDR_CNT	インタフェースのマルチキャストアドレス配列の最大サイズ
IF_MTU	インタフェースの MTU
IF_HDR_ALIGN	ヘッダのアライン単位
IF_IPV4_ADDR_LOCAL	インタフェースの IPv4 アドレス
IF_IPV4_ADDR_LOCAL_MASK	インタフェースの IPv4 サブネットマスク
IF_IPV4_ADDR_LOCAL_BC	インタフェースの IPv4 ブロードキャストアドレス
IF_PROTO_IP	インタフェースの IPv4 プロトコルの値
IF_PROTO_ARP	インタフェースの ARP プロトコルの値
IF_PROTO_IPV6	インタフェースの IPv6 プロトコルの値
IF_MADDR_INIT	インタフェースのマルチキャストアドレス配列の初期化
T_IF_HDR	インタフェースのヘッダ構造体
T_IF_ADDR	インタフェースのアドレス
IF_SOFTC_TO_IFADDR(s)	ソフトウェア情報から MAC アドレスを取り出す関数
IF_OUTPUT(o,d,g,t)	インタフェースの出力関数（アドレス解決あり）
IF_RAW_OUTPUT(o,t)	インタフェースの出力関数（アドレス解決無し）
IF_SET_PROTO(b,p)	インタフェースの上位プロトコル設定関数
IF_GET_IFNET()	ネットワークインタフェース構造体を取り出す関数
IF_ADDMULTI(s)	マルチキャストアドレスの登録
IF_IN6_NEED_CACHE(i)	近隣探索キャッシュの使用の有無
IF_IN6_IFID(i,a)	インタフェース識別子の設定
IF_IN6_RESOLVE_MULTICAST(i,m)	インタフェースのマルチキャストアドレスへの変換

3　イーサネットインタフェースの実装

3.1　汎用ネットワークインタフェースの実装

TINET では、ネットワーク末端の終端ノードで、単一ネットワークインタフェースのみ想定しているため、IP 層から汎用ネットワークインタフェースの各関数を直接呼出すことが可能であり、関数呼出しのオーバヘッドを削減することができます。ただし、ネットワークインタフェースを抽象化し、コンパイル時にネットワークインタフェースを選択可能なように、表 3 に示す汎用ネットワークインタフェース制御マクロを定義しています。IP 層では、これらのマクロを使用し、特定のネットワークインタフェースに依存しないように記述しています。また、ネットワーク層として IPv6 を実装するためには、データリンク層でマルチキャストアドレスへの対応が不可欠です。このため、マルチキャストアドレスへの対応した汎用ネットワークインタフェース制御マクロもあります。

リスト 3 に示す IP 層の出力関数 ip_output を例に説明します（[] 内はファイル名。以降同じ）。この関数では、ネットワークバッファ output へのイーサネットヘッダの設定と、ネットワークインタフェース出力関数を呼び出す部分にマクロが使われています。イーサネットの場合は以下のように展開されます。

- IF_SET_PROTO は、イーサネットフレームのヘッダに上位層のプロトコルを設定するマクロで、以下のように展開されます。

```
(GET_ETHER_HDR(output)->type = htons(IF_PROTOIP))
```

リスト 3　IP 出力関数（ip_output［tinet/netinet/ip_output.c］）

```
ER ip_output (T_NET_BUF *output, TMO tmout)
{
    /* 途中省略 */
    gw = rtalloc(ntohs(ip4h->dst));
    IF_SET_PROTO(output, IF_PROTO_IP);
    if ((error = IF_OUTPUT(output, (VP)&gw, NULL, tmout)) != E_OK)
        /* 送信エラーを記録する */;
    return error;
    }
```

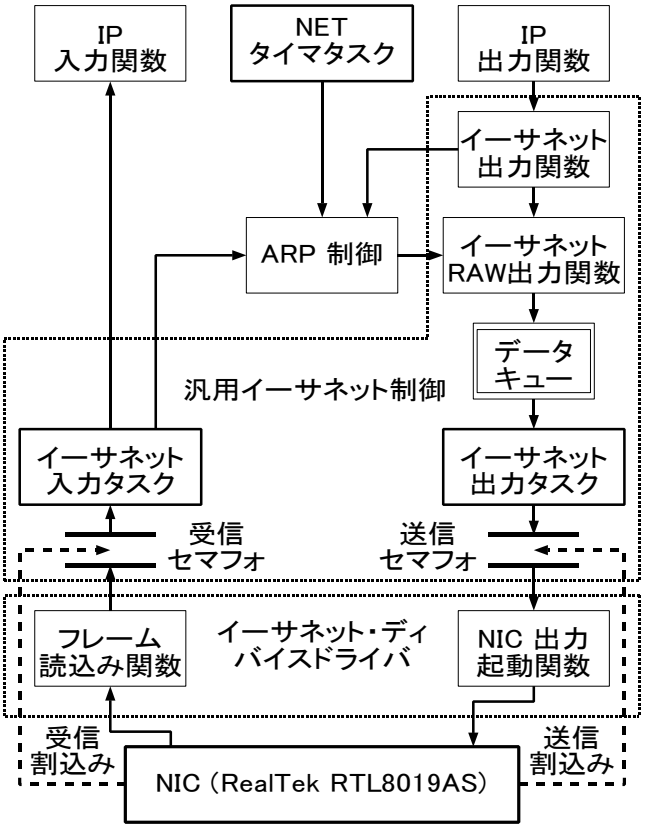

図 4　イーサネットの制御とデータの流れ

- IF_PROTO_IP は、IP を表すネットワークインタフェース固有の上位プロトコル番号に展開されるマクロで、ETHER_TYPE_IP に展開され、さらに ETHER_TYPE_IP は、0x0800 に展開されます。
- IF_OUTPUT は、ネットワークインタフェース出力関数のマクロで、以下のように展開されます。

```
ether_output(output, (VP)&gw, NULL, tmout)
```

3.2　汎用イーサネット制御の実装

図 4 にイーサネットの制御とデータの流れを示します。イーサネットインタフェースは、NIC に依存しない汎用イーサネット制御と、NIC に依存するイーサネット・ディバイスドライバから構成されています。

(1) イーサネット出力関数（ether_output）

リスト 4 に示すイーサネット出力関数 ether_output は、ネットワークバッファ output のイーサネットヘッダに、送信元の MAC アドレスを設定します。次に、イーサネットヘッダに設定されている上

リスト 4　イーサネット出力関数（ether_output［tinet/net/ethernet.c］）

```
ER ether_output (T_NET_BUF *output, VP dst, T_IF_ADDR *gw, TMO tmout)
{
    /* 送信元 MAC アドレスを設定する */
    ic = IF_ETHER_NIC_GET_SOFTC();
    memcpy(GET_ETHER_HDR(output)->shost, ic->ifaddr.lladdr, ETHER_ADDR_LEN);
    switch(ntohs(GET_ETHER_HDR(output)->type)) {
    case ETHER_TYPE_IP:        /* IPv4 */
        if (arp_resolve(&ic->ifaddr, output, *(UW*)dst))
            /* TRUE ならアドレス解決済 */
            error = ether_raw_output(output, tmout);
        break;
    case ETHER_TYPE_IPV6: /* IPv6 */
        error = nd6_storelladdr((T_ETHER_ADDR*)GET_ETHER_HDR(output)->dhost,
                                (T_IN6_ADDR*)dst, gw);
        if (error == E_OK)
            /* E_OK ならアドレス解決済 */
            error = ether_raw_output(output, tmout);
        break;
        }
    return error;
    }
```

リスト 5　イーサネット RAW 出力関数（ether_raw_output［tinet/net/ethernet.c］）

```
ER ether_raw_output (T_NET_BUF *output, TMO tmout)
{
    /* Ethernet 出力キューに投入する。*/
    if ((error = tsnd_dtq(DTQ_ETHER_OUTPUT, (VP_INT)output, tmout)) != E_OK) {
        /* E_OK でなければ NIC をリセットする。*/
        IF_ETHER_NIC_RESET(IF_ETHER_NIC_GET_SOFTC());
        }
    return error;
    }
```

位プロトコルにより分岐します。上位プロトコルが IPv4 の時は、ARP により送信先の MAC アドレスを解決し、上位プロトコルが IPv6 の時は、近隣探索により送信先の MAC アドレスを解決します。解決後、に示すイーサネット RAW 出力関数 ether_raw_output を呼出します。

(2) イーサネット RAW 出力関数（ether_raw_output）

リスト 5 に示すイーサネット RAW 出力関数 ether_raw_output は、ネットワークバッファをデータキュー DTQ_ETHER_OUTPUT に投入します。

(3) イーサネット出力タスク（ether_output_task）

リスト 6 にイーサネット出力タスクを示します。イーサネット出力タスク ether_output_task は、データキュー DTQ_ETHER_OUTPUT からネットワークバッファ output を取り出し，送信セマフォ ic->semid_txb_ready で，NIC 割込みハンドラと if_ed_handler 同期した後，NIC 出力起動関数 IF_ETHER_NIC_START（マクロ展開後は、リスト 20 に示す NIC 出力起動関数 ed_start ）を呼出します。

リスト 6　イーサネット出力タスク（ether_output_task［tinet/net/ethernet.c］）

```
void ether_output_task(VP_INT exinf)
{
    ic = IF_ETHER_NIC_GET_SOFTC();
    while (TRUE) {
        while (rcv_dtq(DTQ_ETHER_OUTPUT, (VP_INT*)&output) == E_OK) {
            syscall(wai_sem(ic->semid_txb_ready));
            IF_ETHER_NIC_START(ic, output);
            syscall(rel_net_buf(output));
            }
        }
    }
```

表 4　TINET-1.3 の汎用イーサネット・ディバイスドライバマクロ

マクロ名	展開値	設定内容
IF_ETHER_NIC_START(i,o)	ed_start(i,o)	NIC 出力起動関数
IF_ETHER_NIC_GET_SOFTC()	ed_get_softc()	NIC ディバイスドライバ共通構造体取得
IF_ETHER_NIC_PROBE(i)	ed_probe(i)	NIC 検出関数
IF_ETHER_NIC_INIT(i)	ed_init(i)	NIC 初期化関数
IF_ETHER_NIC_READ(i)	ed_read(i)	NIC イーサネットフレーム読込み関数
IF_ETHER_NIC_RESET(i)	ed_reset(i)	NIC リセット関数
IF_ETHER_NIC_WATCHDOG(i)	ed_watchdog(i)	NIC ワッチドック関数
IF_ETHER_NIC_IN6_IFID(i,a)	get_mac6_ifid(i,a)	NIC インタフェース識別子の設定
IF_ETHER_NIC_ADDMULTI(s)	ed_addmulti(s)	NIC マルチキャストアドレス追加関数
T_IF_ETHER_NIC_SOFTC	struct t_ed_softc	NIC 固有構造体

(4) イーサネット入力タスク（ether_input_task）

リアルタイム OS のミドルウェアとしての実装では、割込みコンテキスでの実行時間をできるだけ短縮することが重要であるため、リスト 19 に示す NIC 割込みハンドラ if_ed_handler は、入力は一切行わず、受信セマフォ ic->semid_rxb_ready により、リスト 7 に示すイーサネット入力タスク ether_input_task と同期を取ります。

イーサネット入力タスク ether_input_task は受信セマフォ ic->semid_rxb_ready で、NIC 割込みハンドラ if_ed_handler との同期を行った後、イーサネットフレーム読込み関数 IF_ETHER_NIC_READ（マクロ展開後は ed_read ）を呼出し、受信バッファからイーサネットフレームを入力します。次に、イーサネットヘッダに設定されている上位プロトコルにより switch 文で分岐します。

(5) 汎用イーサネット・ディバイスドライバマクロ

汎用ネットワークインタフェース制御同様に、汎用イーサネット制御からディバイスドライバの各関数を直接呼出すことは可能ですが、NIC への依存を避けるため、各イーサネット・ディバイスドライバは、表 4 に示す汎用イーサネット・ディバイスドライバマクロを定義しています。表 4 の展開値は、NE2000 互換 NIC の場合の例です。

(6) 多様な NIC への対応

NIC のハードウェア上の制約や、同じ NIC であっても、ディバイスドライバの内部構成により、TCP/IP プロトコルスタック内の構造体と制御アルゴリズムを調整する必要があります。

NIC のハードウェア上の制約は、表 2 に示した TINET に対するハードウェア依存性定義ファイルで調整を行うことが可能です。NIC に関しては、tinet/netdev/$(NIC) にある tinet_nic_defs.h で調整します。tinet_nic_defs.h には、以下に示すパラメータが定義されています。

リスト 7　イーサネット入力タスク（ether_input_task［tinet/net/ethernet.c］）

```
void
ether_input_task(VP_INT exinf)
{
    ifinit();                   /* ネットワークインタフェース管理を初期化する */

    ic = IF_ETHER_NIC_GET_SOFTC();  /* NIC を初期化する */
    IF_ETHER_NIC_PROBE(ic);
    IF_ETHER_NIC_INIT(ic);

    syscall(act_tsk(ETHER_OUTPUT_TASK));    /* Ethernet 出力タスクを起動する */
    syscall(act_tsk(NET_TIMER_TASK));       /* ネットワークタイマタスクを起動する */

    arp_init();                 /* ARP を初期化する。*/

    ether_ifnet.ic = ic;
    while (TRUE) {
        syscall(wai_sem(ic->semid_rxb_ready));
        if ((input = IF_ETHER_NIC_READ(ic)) != NULL) {
            eth = GET_ETHER_HDR(input);
            proto = ntohs(eth->type);
            switch(ntohs(proto) {
            case ETHER_TYPE_IP:    /* IP */
                ip_input(input);
                break;
            case ETHER_TYPE_ARP:   /* ARP */
                arp_input(&ic->ifaddr, input);
                break;
            case ETHER_TYPE_IPV6:  /* IPv6    */
                ip6_input(input);
                break;
            default:
                /* 入力エラーを記録する */;
                syscall(rel_net_buf(input));
                break;
                }
            }
        }
    }
```

- IF_ETHER_NIC_HDR_ALIGN
  ネットワークバッファにおけるイーサネットフレームのアラインを調整します。図 3「ネットワークバッファ構造体」に示した、align フィールドにより IP ヘッダ以降を 4 バイト境界に整列します。

ディバイスドライバの内部構成は、表 1 に示した TINET 内部パラメータ調整用ファイルで調整を行うことが可能です。NIC に関しては、以下に示すパラメータが定義されています。

- ETHER_NIC_CFG_RELEASE_NET_BUF
  イーサネット出力タスクにおけるネットワークバッファの開放を制御します。

表 5　RTL8019AS NE2000 互換レジスタ

ページ/No.	略称	R/W	範囲	機能
0/00	CR	R/W		コマンドレジスタ
0/03	BNRY	R/W		受信限界ページレジスタ
0/04	TSR	R		送信状態レジスタ
0/05	NCR	R		コリジョンカウンタ
0/07	ISR	R/W		割込み状態レジスタ
0/0D	CNTR	R	0-2	エラーカウンタ
0/01	PSTART	W		受信バッファ開始ページレジスタ
0/02	PSTOP	W		受信バッファ終了ページレジスタ
0/04	TPSR	W		送信開始ページレジスタ
0/05	TBCR	W	0-1	送信オクテット数レジスタ
0/08	RSAR	W	0-1	データ R/W 開始アドレスレジスタ
0/0A	RBCR	W	0-1	データ R/W バイト数レジスタ
0/0C	RCR	W		受信構成レジスタ
0/0D	TCR	W		送信構成レジスタ
0/0E	DCR	W		データ構成レジスタ
0/0F	IMR	W		割込みマスクレジスタ
1/01	PAR	R/W	0-5	MAC アドレスレジスタ
1/07	CURR	R/W		受信開始ページレジスタ
1/08	MAR	R/W	0-7	マルチキャストアドレスレジスタ

4　NE2000 互換 NIC REALTEK 製 RTL8019AS

イーサネット・ディバイスドライバを解説する前に、今回の実装ターゲットとした REALTEK 製 RTL8019AS を解説します。TINET では、表 5 に示す NE2000 互換レジスタのみ使用しています。多数レジスタがあるため、各レジスタは 4 ページのレジスタページに分かれており、レジスタ CR のビット PS0 と PS1 でレジスタページを選択します。表 5 のページ/No. のページはレジスタページを、No. はレジスタアドレス（16 進数）を表しています。読込みと書込みで異なるレジスタをアクセスすることもあります。例えば、0/04 は、読込み時にはレジスタ TSR、書込み時にはレジスタ TPSR になります。また、範囲はレジスタ内インデックスです。例えば、レジスタ TBCR は、0/05 が TBCR0、0/06 が TBCR1 と、複数バイトで構成されています。

MAC アドレスが書き込まれているシリアル EEPROM と、送受信バッファとして使用される NIC 内蔵 SRAM は、レジスタページに関係なく、レジスタアドレス 0x10 でアクセスします。レジスタ RSAR0 と RSAR1 に開始アドレス、レジスタ RBCR0 と RBCR1 に転送バイト数、レジスタ CR に読み書き方向を設定し、レジスタアドレス 0x10 を使って読み書きします。シリアル EEPROM のアドレスは `0x0000` から `0x000b` です。

RTL8019AS には 8 ビットモードと 16 ビットモードがありますが、TINET の NE2000 互換 NIC イーサネット・ディバイスドライバでは、8 ビットモードのみ実装しており、NIC 内蔵 SRAM のアドレスは `0x4000` から `0x5fff` です。なお、ディバイスドライバからはバイト単位にアクセスしますが、NIC 内のイーサネットフレームの送受信は 256 バイトを 1 ページとするページ単位で行われています。

品川通信計装サービス製 NKEV-010H8 ボードでは、RTL8019AS のレジスタは `0x200000` からマップされています。また、割込みに関しては、RTL8019AS の INT4 が H8/3069F の IRQ5 に接続されています。

表 6　JSP ターゲットに依存するファイル

ファイル名	内容
tinet_sys_config.h	JSP ターゲット依存の TINET 内部パラメータ調整用ファイル。
tinet_sys_config.c	JSP ターゲット依存の TINET 関数定義用ファイル。
Makefile.tinet	JSP ターゲット依存の Makefile。

表 7　ディバイスドライバ共通ファイル

ファイル名	内容
Makefile.tinet	コンパイルオプション、依存性探索パス及びオブジェクトの指定
nic.cfg	カーネルオブジェクトの指定。NE2000 互換 NIC では if_ed.cfg をインクルード
nic_rename.h	ディバイスドライバ全域名とアプリケーションプログラム全域名との衝突回避
tinet_nic_config.h	TINET の内部のパラメータ調整用のファイル。 nic_rename.h と if_ed.h をインクルード
tinet_nic_defs.h	TINET に対する NIC のハードウェア依存性定義ファイル。 イーサネットヘッダのアライン調整量の指定

表 8　NIC 固有ファイル

ファイル名	内容
if_ed.c	NE2000 互換 NIC の本体プログラム
if_ed.cfg	割込みハンドラと汎用イーサネット制御との同期用送受信セマフォの定義
if_ed.h	汎用イーサネット・ディバイスドライバマクロの定義と関数のプロトタイプ宣言
if_edreg.h	NE2000 互換 NIC のレジスタ等の定義

5　イーサネット・ディバイスドライバのファイル構成

TINET のイーサネット・ディバイスドライバは、JSP ターゲットに依存するファイルと、依存しないファイルに分かれます。さらに、JSP ターゲットに依存しないファイルは、NIC に依存しないのディバイスドライバ共通ファイルと、NIC に依存する NIC 固有ファイルから構成されています。

(1) JSP ターゲットに依存するディレクトリとファイル

NIC によって、JSP のターゲットに依存するファイルは異なると思いますが、プロセッサから見た NIC のレジスタやメモリのアドレス、割込み等を JSP ターゲット依存部のファイルで定義することになります。JSP ターゲット依存部で関係するファイルは、ディレクトリ config/$(CPU)/$(SYS) にあります。NE2000 互換 NIC の場合、JSP ターゲットに依存するファイルと内容を表 6 に示します。

(2) JSP ターゲットに依存しないディレクトリとファイル

JSP ターゲットに依存しないファイルは、ディレクトリ tinet/netdev/if_ed にあり、NIC に依存しないのディバイスドライバ共通ファイルと、NIC に依存する NIC 固有ファイルから構成されています。

表 7 に、ディバイスドライバ共通ファイルと内容を、表 8 に、NE2000 互換 NIC 固有ファイルと内容を示します。NE2000 互換 NIC の本体プログラムファイル if_ed.c には、これから解説する全ての変数、全域関数及び局所関数が定義されています。

リスト 8　TINET 内部パラメータ調整用ファイル（[config/h8/nkev_010h8/tinet_sys_config.h]）

```
/* NIC (NE2000 互換) に関する定義 */
#define NUM_IF_ED_TXBUF         1               /* 送信バッファ数 */
#define TMO_IF_ED_GET_NET_BUF   1               /* [ms]、受信用 net_buf 獲得タイムアウト */
#define TMO_IF_ED_XMIT          (2*IF_TIMER_HZ) /* [s]、 送信タイムアウト */
/*#define IF_ED_CFG_ACCEPT_ALL  マルチキャスト、エラーフレームも受信するときはコメントを外す。*/

/* NIC (RTL8019AS) に関する定義 */
#define ED_BASE_ADDRESS         0x00200000      /* NIC のレジスタベースアドレス */
#define INHNO_IF_ED             IRQ_EXT5        /* IRQ5 */
#define ED_IER                  H8IER
#define ED_IER_IP_BIT           H8IER_IRQ5E_BIT
#define ED_IPR                  H8IPRA
#define ED_IPR_IP_BIT           H8IPR_IRQ5_BIT

/* 割込みの制御に関する定義 */
#define ED_IPM                  IPM_LEVEL1      /* 割込みプライオリティ・レベル */
#define if_ed_handler_intmask   IPM_LEVEL2      /* 割込みハンドラ実行中の割込みマスクの値 */

#define ed_ena_inter(ipm) chg_ipm(ipm)

extern IPM ed_dis_inter (void);
extern void ed_bus_init (void);
extern void ed_inter_init (void);
```

6　JSP ターゲットに依存するディレクトリとファイル

6.1　TINET 内部パラメータ調整用ファイル（tinet_sys_config.h）

TINET 内部パラメータ調整用ファイル（tinet_sys_config.h）をリスト 8 に示します。このファイルに定義されている項目を以下に示します。

(1) NIC (NE2000 互換) に関する定義
NE2000 互換 NIC 共通の定義で、送信バッファ数、タイムアウト等を定義します。

(2) NIC (RTL8019AS) に関する定義
ターゲットシステムに依存するパラメータを定義します。品川通信計装サービス製 NKEV-010H8 ボードでは、RTL8019AS のレジスタのベースアドレスと割り込み入力番号、プロセッサの割り込み制御レジスタを定義しています。リスト 8 に示すように、RTL8019AS のレジスタは 0x200000 からマップされ、RTL8019AS の割込みラインが H8/3069F の IRQ5 に接続されていることが定義されています。

(3) 割込みの制御に関する定義
JSP リリース 1.4.2 から、H8 依存部の割り込み制御方式が変更されています。割込みの許可・禁止の制御等が共通化されたことにより、依存部における定義が簡略化され、割り込みベクタも自動的に生成されるようになっています。このため、リスト 8 に示すように、割込みプライオリティ・レベル、割込みハンドラ実行中の割込みマスクの値と、割込みの許可・禁止関数等の定義のみが必要となっています。

リスト 9　割込みプライオリティレベル設定変数定義（ed_irc［config/h8/nkev_010h8/tinet_sys_config.c］）

```
static IRC ed_irc = {
    (UB*)ED_IPR,    /* IPR レジスタの番地 */
    ED_IPR_IP_BIT,  /* IPR レジスタの該当するビット番号 */
    ED_IPM,         /* 割込みレベル */
    };
```

リスト 10　割込み禁止関数（ed_dis_inter［config/h8/nkev_010h8/tinet_sys_config.c］）

```
IPM ed_dis_inter(void)
{
    IPM ipm;
    syscall(get_ipm(&ipm));
    syscall(chg_ipm(if_ed_handler_intmask));
    return ipm;
    }
```

6.2　TINET システム依存関数定義ファイル（tinet_sys_config.c）

TINET システム依存関数定義ファイル（tinet_sys_config.c）では、リスト 8 に定義されている割込みの許可・禁止関数等の関数を実装しています。以下に、各関数の概要を示します。

6.2.1　割込みプライオリティレベル設定変数（ed_irc）

リスト 9 に、割込みプライオリティレベル設定用のデータの定義を示します。割込みプライオリティレベル設定用データ構造体 IRC は、config/h8/cpu_defs.h に宣言されています。ed_irc には、コメントに示すデータを定義しています。

6.2.2　割込み禁止関数（ed_dis_inter）

リスト 10 に、割込み禁止関数を示します。リスト 19 に示す NIC 割込みハンドラ if_ed_handler 入口と出口で、割込みの禁止と許可が暗黙的に行われていますが、割込み禁止関数 ed_dis_inter は、これ以外に、割込みの禁止を行いたいときに呼ばれる関数で、イーサネットディバイスドライバ内の各関数で呼出されています。

動作は、現在の割込みレベルを記憶し、割込みレベルを、割込みハンドラ実行中の割込みマスクの値に設定した後、記憶した割込みレベルを関数の戻り値として返します。

なお、割込み許可関数はリスト 8 に示した、TINET 内部パラメータ調整用ファイル tinet_sys_config.h でマクロ定義されています。動作は、記憶してあった、元の割込みレベルに戻します。

6.2.3　ターゲット依存のバス初期化関数（ed_bus_init）

11 にターゲット依存のバス初期化関数を示します。この関数は、リスト 22 に示す NIC 検出関数 ed_probe から呼出され、ターゲットシステムのプロセッサのバス設定を行います。

リスト 11　ターゲット依存のバス初期化関数（ed_bus_init［config/h8/nkev_010h8/tinet_sys_config.c］）

```
void ed_bus_init (void)
{
#ifdef INMEM_ONLY
    /* 外部バスを有効にする。*/
    sil_orb_ddr(IO_PORT1,    0x1f);         /* Enable A0 - A4  */
    sil_orb_ddr(IO_PORT3,    0xff);         /* Enable D8 - D15 */
    sil_orb_ddr(IO_PORT8,    H8P8DDR_CS1);  /* Enable CS1      */
    sil_orb_ddr(IO_PORTB,    H8PBDDR_UCAS); /* Enable UCAS     */
#endif    /* of #ifdef INMEM_ONLY */
}
```

リスト 12　ターゲット依存の割込み初期化関数（ed_inter_init［config/h8/nkev_010h8/tinet_sys_config.c］）

```
void ed_inter_init (void)
{
    /* 割込み要求時用プライオリティ・レベルを設定する。*/
    define_int_plevel(&(ed_irc));

    /* NIC の割り込みを許可する。*/
    bitset((UB*)ED_IER, ED_IER_IP_BIT);
    }
```

リスト 13　TINET システム依存 Makefile（[config/h8/nkev_010h8/Makefile.tinet]）

```
TINET_COBJS := $(TINET_COBJS) tinet_sys_config.o
```

6.2.4　ターゲット依存の割込み初期化関数（ed_inter_init）

12 にターゲット依存の割込み初期化関数を示します。この関数は、リスト 39 に示す NIC 初期化本体関数（ed_init_sub から呼出され、ターゲットシステムのプロセッサの割込みの初期化を行います。

6.3　TINET システム依存 Makefile（Makefile.tinet）

TINET システム依存 Makefile（Makefile.tinet）をリスト 13 に示します。リストに示すように、TINET システム依存関数定義ファイル（tinet_sys_config.c）を組込むことを指定しています。

リスト 14　ディバイスドライバ共通構造体宣言（T_IF_SOFTC［tinet/net/ethernet.h］）

```
struct t_if_softc {
    T_IF_ADDR               ifaddr;             /* MAC アドレス */
    UH                      timer;              /* 送信タイマ */
    T_IF_ETHER_NIC_SOFTC    *sc;                /* NIC 固有構造体へのポインタ */
    ID                      semid_txb_ready;    /* 送信セマフォ */
    ID                      semid_rxb_ready;    /* 受信セマフォ */
#ifdef SUPPORT_INET6
    T_IF_ADDR     maddrs[MAX_IF_MADDR_CNT];    /* マルチキャストアドレスリスト */
#endif  /* of #ifdef SUPPORT_INET6 */
    } T_IF_SOFTC;
```

リスト 15　ディバイスドライバ共通構造体変数定義（if_softc［tinet/netdev/if_ed/if_ed.c］）

```
T_IF_SOFTC if_softc = {
    {}                      /* MAC アドレス */
    0,                      /* 送信タイマ */
    &ed_softc,              /* NIC 固有構造体へのポインタ */
    SEM_IF_ED_SBUF_READY,   /* 送信セマフォ */
    SEM_IF_ED_RBUF_READY,   /* 受信セマフォ */
#ifdef SUPPORT_INET6
    IF_MADDR_INIT,          /* マルチキャストアドレスリスト */
#endif  /* of #ifdef SUPPORT_INET6 */
    };
```

リスト 16　NE2000 互換 NIC 固有構造体宣言（T_ED_SOFTC［tinet/netdev/if_ed/if_ed.c］）

```
typedef struct t_ed_softc {
    UW  nic_addr;                   /* NIC のベースアドレス */
    UW  asic_addr;                  /* ASIC のベースアドレス */
    UH  txb_len[NUM_IF_ED_TXBUF];   /* 送信バッファのオクテット数 */
    UB  txb_inuse;                  /* 使用中の送信バッファ数 */
    UB  txb_insend;                 /* 送信中の送信バッファ数 */
    UB  txb_write;                  /* 書き込む送信バッファ */
    UB  txb_send;                   /* 送信する送信バッファ */
    UB  rxb_read;                   /* 読込む受信ページ */
    } T_ED_SOFTC;
```

7　イーサネット・ディバイスドライバ制御構造体と変数

イーサネット・ディバイスドライバを制御するために必要となる変数は、イーサネット・ディバイスドライバ制御構造体で定義しています。この構造体は、NIC に依存しないディバイスドライバ共通構造体と、NIC に依存する NIC 固有構造体へのポインタから構成されています。

7.1　ディバイスドライバ共通構造体（T_IF_SOFTC）と変数（if_softc）

リスト 14 に、ディレクトリ tinet/net にある ethernet.h で宣言されてるディバイスドライバ共通構造体 T_IF_SOFTC を示します。また、リスト 15 に、この構造体の実体となる変数 if_softc を示します。各イーサネット・ディバイスドライバは、T_IF_SOFTC のメンバ sc の T_IF_ETHER_NIC_SOFTC を定義する必要があります。NE2000 互換 NIC の場合は、表 4 に示す汎用イーサネット・ディバイスドライバマクロにより、struct t_ed_softc に展開されますが、依存性を避けるため、実体は if_ed.c に宣言されています。また、メンバ maddrs は、ネットワーク層として IPv6 を選択したときのみ有効のマルチキャストアドレスリストです。

リスト 17　NE2000 互換 NIC 固有構造体変数定義（ed_softc［tinet/netdev/if_ed/if_ed.c］）

```
static T_ED_SOFTC ed_softc = {
    ED_BASE_ADDRESS + ED_NIC_OFFSET,  /* NIC のベースアドレス  */
    ED_BASE_ADDRESS + ED_ASIC_OFFSET, /* ASIC のベースアドレス */
    };
```

リスト 18　NE2000 互換 NIC レジスタ定義ファイル（［tinet/netdev/if_ed/if_edreg.h］）

```
#define ED_INT_RAM_SIZE     0x2000
#define ED_INT_RAM_BASE     0x4000

#define ED_PAGE_SIZE        256
#define NUM_IF_ED_TXBUF_PAGE     ((ETHER_MAX_LEN + ED_PAGE_SIZE - 1) / ED_PAGE_SIZE)
#define IF_ED_TXBUF_PAGE_SIZE    (NUM_IF_ED_TXBUF_PAGE * NUM_IF_ED_TXBUF)
#define IF_ED_RXBUF_PAGE_SIZE    (ED_INT_RAM_SIZE / ED_PAGE_SIZE - IF_ED_TXBUF_PAGE_SIZE)

#define ED_NIC_OFFSET       0x00
#define ED_ASIC_OFFSET      0x10
#define ED_DATA_OFFSET      0x00
```

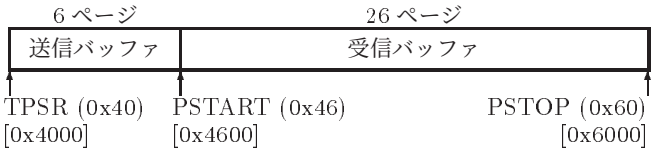

図 5　送受信リングバッファ

7.2　NE2000 互換 NIC 固有構造体（T_ED_SOFTC）と変数（ed_softc）

リスト 16 に、if_ed.c で宣言されている NE2000 互換 NIC 固有構造体 T_ED_SOFTC を示します（表 4 では struct t_ed_softc ）。また、リスト 17 に、この構造体の実体となる変数の定義を示します。例えば、メンバ nic_addr の設定に使われている ED_BASE_ADDRESS は、リスト 8 に示した TINET 内部パラメータ調整用ファイル tinet_sys_config.h に定義されており、値は 0x00200000 です。ED_NIC_OFFSET はリスト 18 に示す NE2000 互換 NIC レジスタ定義ファイル if_edreg.h （送受信バッファとレジスタアドレスの定義部分のみ）に定義されており、値は 0x00 です。従って、NE2000 互換 NIC 固有構造体 t_ed_softc のメンバ nic_addr の値は 0x00200000 となり、表 5 に示したレジスタのベースアドレスとなります。同様に、asic_addr の値は 0x00200010 となり、この値が NIC の内蔵 SRAM をアクセスするためのアドレスとなります。

図 5 に送受信リングバッファを示します。送受信リングバッファの設定レジスタには、256 バイトを 1 ページとするページ単位の値を設定します。送受信とも同じ NIC 内蔵 SRAM を使用しますので、送信バッファと受信バッファが重ならないようにします。受信バッファの開始ページをレジスタ PSTART に、終了ページ（正確には、終了ページの次のページ）をレジスタ PSTOP に設定します。受信フレームは、受信バッファに順次格納され、終了ページに達すると、開始ページに折り返されるリングバッファ構造になっています。送信バッファの開始ページはレジスタ TPSR で指定します。図 5 で（）内は、ページアドレスで、レジスタ PSTART、PSTOP 及び TPSR に設定する値です。［］内は、ディバイスドライバから見たバイト単位のアドレスです。

一般に、送信バッファ 1 個のページ数は、最大長のイーサネットフレーム（1,518 オクテット）が入る 6 ページ分（1,536 バイト）を確保します。送信バッファ数 NUM_IF_ED_TXBUF は、リスト 8 に示した TINET 内部パラメータ調整用ファイル tinet_sys_config.h で 1 に定義されており、図 5 に示す送信バッファ全体

表 9　イーサネット・ディバイスドライバ全域関数

関数名	機能
if_ed_handler	NIC 割込みハンドラ
ed_start	NIC 出力起動関数
ed_get_softc	NIC ディバイスドライバ共通構造体取得関数
ed_probe	NIC 検出関数
ed_init	NIC 初期化関数
ed_read	NIC イーサネットフレーム読込み関数
ed_reset	NIC リセット関数
ed_watchdog	NIC ワッチドッグ関数
ed_addmulti	NIC マルチキャストアドレス追加関数

のページ数 IF_ED_TXBUF_PAGE_SIZE は 6 ページとなります。また、図 5 に示す受信バッファ全体のページ数 IF_ED_RXBUF_PAGE_SIZE は、NIC 内蔵 SRAM の残りのページ数で、26 ページとなります。送信バッファ数を増加することも可能ですが、受信バッファ数が減少することに注意してください。

8　汎用イーサネット・ディバイスドライバマクロと NE2000 互換 NIC 全域関数

汎用ネットワークインタフェース制御同様に、汎用イーサネット制御からディバイスドライバの各関数を直接呼出すことは可能ですが、NIC への依存を避けるため、各イーサネット・ディバイスドライバは、表 4 に示す汎用イーサネット・ディバイスドライバマクロを定義する必要があります。NE2000 互換 NIC の場合は、if_ed.h に定義されています。

表 9 に、NE2000 互換 NIC のイーサネット・ディバイスドライバ全域関数を示します。この全域関数は、表 4 の汎用イーサネット・ディバイスドライバマクロに含まれており、ディバイスドライバに必須の関数です。

以下の事項については、説明が煩雑になることを避けるため省略しています。

- 関数によっては、入り口で割り込みを禁止し、出口で割り込みを許可する。
- レジスタを変更する前にレジスタページを選択して NIC を停止し、変更後に NIC を起動する必要がある。
- レジスタページの切り替え。

8.1　NIC 割込みハンドラ（if_ed_handler）

NIC からの割込みは、送受信ともリスト 19 に示す JSP カーネルに登録した NIC 割込みハンドラ if_ed_handler で処理します。以下は動作概要です。

(1) 送信完了と送信エラーにより送信割込みが発生します。NIC 固有構造体のメンバ sc->txb_insend と sc->txb_inuse は、リスト 16 に示すように、送信中バッファ数と使用中バッファ数を表しており、送信が完了したので、それぞれデクリメントします。次に、sc->txb_inuse が 0 以上なら、未送信フレームが送信バッファに残っているため、関数 ed_xmit で、NIC に送信を指示します。最後に、送信セマフォ ic->semid_txb_ready によりイーサネット出力タスクと同期を取ります。

(2) 受信完了と受信エラーにより受信割込みが発生します。受信完了は、受信セマフォ ic->semid_rxb_ready によりイーサネット入力タスクと同期を取るだけで終了します。

リスト 19　NIC 割込みハンドラ（if_ed_handler［tinet/netdev/if_ed/if_ed.c］）

```
void if_ed_handler (void)
{
    ic = &if_softc;
    sc = ic->sc;

    /* 割込み状態レジスタの内容を読み出す。*/
    isr = sil_reb_mem((VP)(sc->nic_addr + ED_P0_ISR));

    /* 全ての割り込みフラグをリセットする。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_ISR), isr);

    /* 送信割込み処理 */
    if (isr & (ED_ISR_PTX | ED_ISR_TXE)) {
        if (isr & ED_ISR_TXE)
            /* 送信エラーを記録する */

        if (sc->txb_insend)
            sc->txb_insend --;
        if (sc->txb_inuse)
            sc->txb_inuse  --;

        ic->timer = 0;          /* 送信タイムアウトをリセットする。*/

        if (sc->txb_inuse)      /* まだ送信バッファに残っていれば送信する */
            ed_xmit(ic);
        if (isig_sem(ic->semid_txb_ready) != E_OK)
            /* セマフォエラーを記録する */
        }

    /* 受信割込み処理 */
    if (isr & (ED_ISR_PRX | ED_ISR_RXE | ED_ISR_OVW)) {
        if (isr & ED_ISR_OVW)
            /* 上書きエラーとリセットを記録する */
        else {
            if (isr & ED_ISR_RXE)
                /* 受信エラーを記録する */
            if (isig_sem(ic->semid_rxb_ready) != E_OK)
                /* セマフォエラーを記録する */
            }
        }
    }
```

8.2　NIC 出力起動関数（ed_start）

リスト 20 に、NIC 出力起動関数 ed_start を示します。ed_start は、リスト 30 に示す NIC 内蔵 SRAM 書込み関数 ed_pio_writemem によりネットワークバッファ output のイーサネットフレームを送信バッファに書込みます。sc->txb_write は、送信フレームを書き込む送信バッファのインデックスです。

次に、書き込んだオクテット数を、sc->txb_write をインデックスとして、配列 sc->txb_len に設定します。ここでは、スロット時間[*1]を考慮して、イーサネットフレームが、最短イーサネットフレーム長である 64 オクテットから CRC の 4 オクテットを除いた 60 オクテットより短い時は、60 オクテットを設定します。

送信バッファを切り替えた後、使用中の送信バッファ数 sc->txb_inuse をインクリメントします。最後に、現在送信中でなければ、関数 ed_xmit を呼出して、イーサネットフレームの送信を開始して終了します。

[*1] イーサネットの制御方式である CSMA/CD における衝突検出のため送信しつづけなければならない最短時間で、64 オクテットを送信する時間

リスト 20　NIC 出力起動関数（ed_start［tinet/netdev/if_ed/if_ed.c］）

```
void ed_start (T_IF_SOFTC *ic, T_NET_BUF *output)
{
    T_ED_SOFTC *sc = ic->sc;

    ipm = ed_dis_inter();       /* NIC からの割り込みを禁止する。*/

    /* 送信バッファに書込む。*/
    ed_pio_writemem(sc, output->buf,
                    ED_INT_RAM_BASE + sc->txb_write * NUM_IF_ED_TXBUF_PAGE * ED_PAGE_SIZE,
                    output->len);

    /* 送信バッファに書込んだオクテット数を記録する。*/
    if (output->len > ETHER_MIN_LEN-ETHER_CRC_LEN)
        sc->txb_len[sc->txb_write] = output->len;
    else
        sc->txb_len[sc->txb_write] = ETHER_MIN_LEN - ETHER_CRC_LEN;

    sc->txb_write ++;           /* 送信バッファを切り替える。*/
    if (sc->txb_write == NUM_IF_ED_TXBUF)
        sc->txb_write = 0;

    sc->txb_inuse ++;

    if (sc->txb_insend == 0)    /* もし送信中でなければ、送信を開始する。*/
        ed_xmit(ic);

    ed_ena_inter(ipm);          /* NIC からの割り込みを許可する。*/
    }
```

リスト 21　ディバイスドライバ共通構造体取得関数（ed_get_softc［tinet/netdev/if_ed/if_ed.c］）

```
T_IF_SOFTC *ed_get_softc (void)
{
    return &if_softc;
    }
```

8.3　ディバイスドライバ共通構造体取得関数（ed_get_softc）

リスト 21 にディバイスドライバ共通構造体取得関数を示します。この関数はリスト 15 に示した局所変数のディバイスドライバ共通構造体の実体である変数 if_softc を返すだけの関数です。

8.4　NIC 検出関数（ed_probe）

リスト 22 に、NIC 検出関数を示します。NIC 検出関数の本来の機能は、NE2000 互換 NIC がインストールされていることを確かめ、互換 NIC でもベンダ毎に異なるメモリやレジスタの違いを記録することです。しかし、TINET では、コンパイル時に NIC が確定しているので、この NIC 検出関数では、以下に示す操作のみ実行します。

(1) ターゲット依存部のバスを初期化する。
(2) NIC をリセットする。
(3) データ構成レジスタ DCR を設定する。
(4) シリアル EEPROM から MAC アドレスを読み出す。
(5) すべての割込みフラグをリセットする。

リスト 22　NIC 検出関数（ed_probe［tinet/netdev/if_ed/if_ed.c］）

```
void ed_probe (T_IF_SOFTC *ic)
{
    ed_bus_init();          /* ターゲット依存部のバスの初期化 */

    /* リセットする */
    tmp = sil_reb_mem((VP)(sc->asic_addr + ED_RESET_OFFSET));
    sil_wrb_mem((VP)(sc->asic_addr + ED_RESET_OFFSET), tmp);
    dly_tsk(5);

    /* 途中省略 */

    /* DCR レジスタを設定する */
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_DCR), ED_DCR_FT1 | ED_DCR_LS);

    /* MAC アドレスを読込む */
    ed_pio_readmem(sc, 0, romdata, ETHER_ADDR_LEN * 2);
    for (ix = 0; ix < ETHER_ADDR_LEN; ix ++)
        ic->ifaddr.lladdr[ix]
            = romdata[ix * 2 + 1];

    /* 全ての割込みフラグをリセットする */
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_ISR), 0xff);
    }
```

リスト 23　NIC 初期化関数（ed_init［tinet/netdev/if_ed/if_ed.c］）

```
void ed_init (T_IF_SOFTC *ic)
{
    IPM    ipm;

    ipm = ed_dis_inter();       /* NIC からの割り込みを禁止する。*/
    ed_init_sub(ic);            /* ed_init 本体を呼び出す。*/
    ed_ena_inter(ipm);           /* NIC からの割り込みを許可する。*/
    }
```

8.5　NIC 初期化関数（ed_init）

リスト 23 に、NIC 初期化関数を示します。この関数は、NIC からの割込みを禁止して、リスト 39 に示す NIC 初期化本体関数 ed_init_sub を呼出します。

8.6　イーサネットフレーム読込み関数（ed_read）

図 6 に受信バッファを示します。受信バッファの開始ページをレジスタ PSTART に、終了ページをレジスタ PSTOP に設定します。受信フレームは、レジスタ CURR に設定されているページに格納されて行きます。レジスタ CURR は自動的に次のページに進み、終了ページに達すると開始ページに戻ります。レジスタ BNRY は、レジスタ CURR が進むことができる限界ページです。sc->rxb_read は、次にディバイスドライバが読込む受信フレームのページを指しています。

リスト 24 にイーサネットフレーム読込み関数を示します。以下は動作概要です。

(1) sc->rxb_read とレジスタ CURR の値が異なっていれば、受信フレームの読込みを開始します。
(2) 受信フレームの先頭ページアドレスを計算します。
(3) 受信フレームの情報は、リスト 25 に示す受信フレームヘッダ構造体 frame_hdr に格納されて、受信フレームの直前に置かれています。この frame_hdr を読込み、プロセッサのバイトオーダがビックエン

リスト 24　イーサネットフレーム読込み関数（ed_read［tinet/netdev/if_ed/if_ed.c］）

```
T_NET_BUF *ed_read (T_IF_SOFTC *ic)
{
    /* 未読の受信フレームがあるかチェックする。*/
    curr = sil_reb_mem((VP)(sc->nic_addr + ED_P1_CURR));
    if (sc->rxb_read != curr) {
        /* 受信フレームの先頭ページを計算する。*/
        frame_ptr = sc->rxb_read * ED_PAGE_SIZE;

        /* 受信フレームヘッダ構造体を取り出す。*/
        ed_pio_readmem(sc, frame_ptr, (char *)&frame_hdr, sizeof(frame_hdr));

#if SIL_ENDIAN == SIL_ENDIAN_BIG
        frame_hdr.count
          = (frame_hdr.count << 8)
          | (frame_hdr.count >> 8);
#endif

        len = frame_hdr.count;
        if (len> sizeof(T_ED_FRAME_HDR) &&
            len<=IF_MTU+sizeof(T_ETHER_HDR) + sizeof(T_ED_FRAME_HDR) &&
            frame_hdr.next>=ED_INT_RXBUF_START &&
            frame_hdr.next< ED_INT_RXBUF_STOP)
            input =
              ed_get_frame(sc, frame_ptr+sizeof(T_ED_FRAME_HDR),
                               len-sizeof(T_ED_FRAME_HDR));
        else
            /* NIC を停止し、リセット後、割込みを許可する。*/
            return NULL;

        /* 次に読込む受信フレームを更新する。*/
        sc->rxb_read = frame_hdr.next;

        /* 受信フレームの境界ページを更新する。*/
        boundry = sc->rxb_read - 1;
        if (boundry < ED_INT_RXBUF_START)
            boundry = ED_INT_RXBUF_STOP - 1;
        sil_wrb_mem((VP)(sc->nic_addr + ED_P0_BNRY), boundry);
        }

    /* 受信リングバッファにデータが残っていれば、
       受信処理を継続する。*/
    curr = sil_reb_mem((VP)(sc->nic_addr + ED_P1_CURR));
    if (sc->rxb_read != curr)
        sig_sem(ic->semid_rxb_ready);
    return input;
    }
```

ディアンの場合は、受信フレーム長を表す frame_hdr.count のバイトを交換します[*2]。

(4) frame_hdr.count と、次の受信フレームのページを表す frame_hdr.next が正しいかチェックし、正しければ受信フレームを読込みます。誤っていれば、NIC を停止し、リセットして関数を終了します。

(5) sc->rxb_read を frame_hdr.next で更新します。

(6) 受信フレームの境界ページを表すレジスタ BNRY に frame_hdr.next の前のページを設定します。

(7) もう一度、sc->rxb_read とレジスタ CURR を比較して、値が異なっていれば、まだ未読のイーサネットフレームがあることを意味していますので、受信セマフォ ic->semid_rxb_ready で、イーサネット入力タスクと同期を取ります。

[*2] RTL8019AS のデータシートに記載されてますが、レジスタ DCR のビット BOS（Byte Order Select）は未実装です

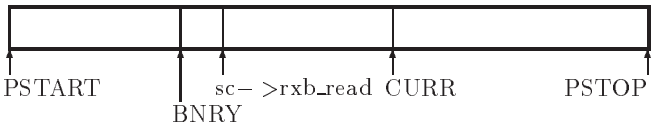

図 6　受信バッファ

リスト 25　受信リングバッファヘッダ構造体宣言（T_ED_RING［tinet/netdev/if_ed/if_edreg.h］）

```
typedef struct t_ed_frame_hdr {
    UB  rsr;    /* 受信ステータス */
    UB  next;   /* 次の受信フレームのページ */
    UH  count;  /* 受信フレーム長（長さ+4）*/
    } T_ED_FRAME_HDR;
```

リスト 26　NIC リセット関数（ed_reset［tinet/netdev/if_ed/if_ed.c］）

```
void ed_reset (T_IF_SOFTC *ic)
{
    IPM ipm;

    /* NIC からの割り込みを禁止する。*/
    ipm = ed_dis_inter();

    ed_stop(ic->sc);
    ed_init_sub(ic);

    /* NIC からの割り込みを許可する。*/
    ed_ena_inter(ipm);
    }
```

8.7　NIC リセット関数（ed_reset）

リスト 26 に、NIC リセット関数を示します。この関数は、NIC からの割込みを禁止して、リスト 33 に示す NIC 停止関数 ed_stop を呼出します。NIC を停止した後、リスト 39 に示す NIC 初期化本体関数 ed_init_sub を呼出します。

8.8　NIC ワッチドッグ関数（ed_watchdog）

T_IF_SOFTC にある送信タイマ timer には、イーサネットフレーム送信関数 ed_xmit で、以下のようにタイムアウトが設定されます。

```
ic->timer = TMO_IF_ED_XMIT;
```

このタイマは、ディレクトリ tinet/net の if.c にある関数 if_slowtimo で監視されており、タイムアウトすると、この関数が呼出されます。リスト 27 に、NIC ワッチドック関数を示します。この関数は、NIC ワッチドッグ関数は、リスト 26 に示す NIC リセット関数 ed_reset を呼出すだけで終了します。

8.9　NIC マルチキャストアドレス追加関数（ed_addmulti）

リスト 28 に、NIC マルチキャストアドレス追加関数を示します。この関数は、リスト 35 に示す IPv6 用 NIC 受信構成レジスタ設定関数 ed_setrcr を呼出すだけで終了します。

リスト 27　NIC ワッチドック関数（ed_watchdog［tinet/netdev/if_ed/if_ed.c］）

```
void ed_watchdog (T_IF_SOFTC *ic)
{
    ed_reset(ic);
    }
```

リスト 28　NIC マルチキャストアドレス追加関数（ed_addmulti［tinet/netdev/if_ed/if_ed.c］）

```
ER ed_addmulti (T_IF_SOFTC *ic)
{
    ed_setrcr(ic);
    return E_OK;
    }
```

表 10　NE2000 互換 NIC イーサネット・ディバイスドライバ局所関数

関数名	機能
ed_pio_readmem	NIC 内蔵 SRAM 読込み関数
ed_pio_writemem	NIC 内蔵 SRAM 書込み関数
ed_get_frame	NIC からのイーサネットフレーム読込み関数
ed_xmit	NIC へのイーサネットフレーム送信関数
ed_stop	NIC 停止関数
ed_init_sub	NIC 初期化本体関数
ed_setrcr	NIC 受信構成レジスタ設定関数
ds_crc	イーサネットアドレス CRC 計算関数（IPv6 のみ）
ds_getmcaf	マルチキャストアドレスフィルタ計算関数（IPv6 のみ）

9　NE2000 互換 NIC 局所関数

表 10 に、イーサネット・ディバイスドライバ内でのみ使用される NE2000 互換 NIC のイーサネット・ディバイスドライバ局所関数を示します。

9.1　NIC 内蔵 SRAM 読込み関数（ed_pio_readmem）

リスト 29 に、NIC 内蔵 SRAM 読込み関数を示します。まず、レジスタ RBCR0 と RBCR1 に転送数を設定し、レジスタ RSAR0 と RSAR1 に、NIC 内蔵 SRAM の転送元アドレスを設定します。次に、レジスタ CR により NIC 内蔵 SRAM の読込みを選択し、NIC 内蔵 SRAM をアクセスするレジスタアドレス

```
sc->asic_addr + ED_DATA_OFFSET
```

から、転送数分の読み出しを行います。

9.2　NIC 内蔵 SRAM 書込み関数（ed_pio_writemem）

リスト 30 に、NIC 内蔵 SRAM 書込み関数を示します。NIC 内蔵 SRAM 書込み関数は、読込みが書込みに変わるだけで、リスト 29 に示す NIC 内蔵 SRAM 読込み関数 ed_pio_readmem とほぼ同じです。

リスト 29　NIC 内蔵 SRAM 読込み関数（ed_pio_readmem［tinet/netdev/if_ed/if_ed.c］）

```
static void
ed_pio_readmem (T_ED_SOFTC *sc, UW src,
                    UB *dst, UH amount)
{
    /* 転送数を設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR0), amount);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR1), amount >> 8);

    /* NIC 内蔵 SRAM の転送元アドレスを設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RSAR0), src);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RSAR1), src >> 8);

    /* NIC 内蔵 SRAM から読込む */
    while (amount -- > 0)
        *dst ++ = sil_reb_mem((VP)(sc->asic_addr + ED_DATA_OFFSET));
    }
```

リスト 30　NIC 内蔵 SRAM 書込み関数（ed_pio_writemem［tinet/netdev/if_ed/if_ed.c］）

```
static void
ed_pio_writemem (T_ED_SOFTC *sc, UB *src,
                        UW dst, UH amount)
{
    /* 転送数を設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR0), amount);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR1), amount >> 8);

    /* NIC 内蔵 SRAM の転送元アドレスを設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RSAR0), dst);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RSAR1), dst >> 8);

    /* NIC 内蔵 SRA に書き込む */
    while (amount -- > 0)
        outb(sc->asic_addr + ED_DATA_OFFSET, *src ++);

    /* 完了するまで約 240us 待つ。*/
    }
```

9.3　NIC からのイーサネットフレーム読込み関数（ed_get_frame）

イーサネットフレームを読込む時に、図 7 に示すように、IP ヘッダの先頭を 4 バイト境界に調整する必要があります。このために、ネットワークバッファには、図 3 と表 3 に示したように、調整用のメンバ align が宣言されており、表 3 の IF_HDR_ALIGN の定義に従って宣言されます。イーサネットでは、ヘッダ長が 14 オクテットであるため、IF_HDR_ALIGN は 2 と定義されています。また、TCP と UDP の入力では、それぞれの擬似ヘッダと SDU [*3]でチェックサムを計算しますが、図 7 の SDU 長 n が 4 オクテット境界でない時は、SDU の後の 4 オクテット境界まで 0 のデータを書き込むので、この分を考慮してネットワークバッファを確保します。

リスト 31 に、NIC からのイーサネットフレーム読込み関数を示します。変数 align は、SDU の後の 4 オクテット境界までのバッファ長です。図 6 に示したように、受信バッファはリングバッファのため、受信フレームが終了ページを超える時は、開始ページに折り返されていますので、引数 ring が指すページからレジスタ PSTOP が指すページまでと、レジスタ PSTART が指すページから受信フレームの終わりのページまで

[*3] SDU (Service Data Unit) は OSI の用語で、プロトコルが運ぶデータ（ペイロード）

リスト 31　NIC からのイーサネットフレーム読込み関数（ed_get_frame［tinet/netdev/if_ed/if_ed.c］）

```
static T_NET_BUF *
ed_get_frame (T_ED_SOFTC *sc, UW ring, UH len)
{
    align = ((((len - sizeof(T_IF_HDR)) + 3) >> 2) << 2) + sizeof(T_IF_HDR);

    error = tget_net_buf(&input, align, TMO_IF_ED_GET_NET_BUF);
    if (error == E_OK && input != NULL) {
        dst = input->buf;
        if (ring + len > ED_INT_RAM_BASE + ED_INT_RAM_SIZE) {
            sublen = (ED_INT_RAM_BASE + ED_INT_RAM_SIZE) - ring;
            ed_pio_readmem(sc, ring, dst, sublen);
            len -= sublen;
            dst += sublen;
            ring = ED_INT_RXBUF_START * ED_PAGE_SIZE;
            }
        ed_pio_readmem(sc, ring, dst, len);
        }
    else
        /* エラーを記録する */
    return input;
    }
```

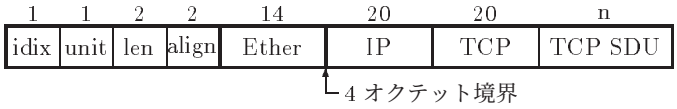

図 7　アライン調整

に分けて、受信バッファから受信フレームを読込みます。

9.4　NIC へのイーサネットフレーム送信関数（ed_xmit）

リスト 32 に、NIC へのイーサネットフレーム送信関数を示します。ed_xmit を呼出す前に、NIC 出力起動関数 ed_start で、送信フレームを送信バッファに書込み込んでいるため、ed_xmit では、NIC に送信を指示するだけです。以下に動作概要を示します。

(1) 送信する送信バッファのページを、レジスタ TPSR に設定します。sc->txb_send は、送信する送信バッファのインデックスです。
(2) ed_start で設定された配列 sc->txb_len から送信するオクテット数を読み、レジスタ TBCR0 と TBCR1 に設定します。
(3) レジスタ CR のビット TXP をセットして送信を開始します。
(4) 送信バッファを切り替え、送信中のバッファ数 sc->txb_insend をインクリメントします。
(5) 送信タイムアウトを設定します。

9.5　NIC 停止関数（ed_stop）

リスト 33 に、NIC 停止関数を示します。この関数は、レジスタ CR により NIC を停止し、5 [ms] を限度に、NIC がリセット状態から復帰するまで待ちます。

リスト 32　NIC へのイーサネットフレーム送信関数（ed_xmit［tinet/netdev/if_ed/if_ed.c］）

```
static void ed_xmit (T_IF_SOFTC *ic)
{
    /* 送信するページを設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TPSR),
         ED_INT_TXBUF_START + sc->txb_send * NUM_IF_ED_TXBUF_PAGE);

    /* 送信するオクテット数を設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TBCR0), sc->txb_len[sc->txb_send]);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TBCR1), sc->txb_len[sc->txb_send] >> 8);

    /* 送信する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_CR),
                ED_CR_RD2 | ED_CR_PAGE0 | ED_CR_TXP | ED_CR_STA);

    /* 送信バッファを切り替える。*/
    sc->txb_send ++;
    if (sc->txb_send == NUM_IF_ED_TXBUF)
        sc->txb_send = 0;

    sc->txb_insend ++;

    /* 送信タイムアウトを設定する。*/
    ic->timer = TMO_IF_ED_XMIT;
    }
```

リスト 33　NIC 停止関数（ed_stop［tinet/netdev/if_ed/if_ed.c］）

```
static void ed_stop (T_ED_SOFTC *sc)
{
    int wait;

    /* DMA を停止する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_CR),
                ED_CR_RD2 | ED_CR_PAGE0 | ED_CR_STP);

    /* NIC が停止状態になるまで待つ。ただし、5[ms] を上限にしている。*/
    for (wait = 5; ((sil_reb_mem((VP)(sc->nic_addr + ED_P0_ISR)) &
                   ED_ISR_RST) == 0) && wait -- > 0; )
        syscall(dly_tsk(1));
}
```

9.6　受信構成レジスタ設定関数（ed_setrcr）

TINET に組込まれるネットワーク層により、設定内容が異なっているため、IPv4 用と IPv6 用に分けて解説します。

9.6.1　IPv4 用受信構成レジスタ設定関数（ed_setrcr）

リスト 34 に、IPv4 用受信構成レジスタ設定関数を示します。内部パラメータ調整用ファイルで指定されている IF_ED_CFG_ACCEPT_ALL により、設定が異なります。

(1) 定義されている場合は、マルチキャスト設定レジスタ MAR0 から MAR7 の全ビットを 1 に設定し、全てのマルチキャストフレームを受信します。また、表 11 に示すアドレスのイーサネットフレームを受信するようにレジスタ RCR に設定します。

リスト 34　IPv4 用受信構成レジスタ設定関数（ed_setrcr［tinet/netdev/if_ed/if_ed.c］）

```
static void ed_setrcr (T_IF_SOFTC *ic)
{
#ifdef IF_ED_CFG_ACCEPT_ALL

    /* マルチキャストの受信設定 */
    for (ix = 0; ix < 8; ix ++)
        /* マルチキャストを全て受信する。*/
        outb(sc->nic_addr + ED_P1_MAR(ix), 0xff);

    /* マルチキャストとエラーフレームも受信するように設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RCR),
                ED_RCR_PRO| ED_RCR_AM | ED_RCR_AB |ED_RCR_SEP);

#else

    /* マルチキャストの受信設定 */
    for (ix = 0; ix < 8; ix ++)
        /* マルチキャストを全て受信しない。*/
        outb(sc->nic_addr + ED_P1_MAR(ix), 0x00);

    /* 自分とブロードキャストのみ受信するように設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RCR), ED_RCR_AB);

#endif
    }
```

表 11　受信構成レジスタ RCR の設定

ビット	受信するフレーム
PRO	全てのアドレスのフレーム
AM	マルチキャストフレーム
AB	ブロードキャストフレーム
SEP	エラーフレーム

(2) 定義されていない場合は、MAR0 から MAR7 の全ビットを 0 に設定し、全てのマルチキャストフレームを受信しないようにします。また、レジスタ RCR はブロードキャストフレームのみ受信するように設定します。

9.6.2　IPv6 用受信構成レジスタ設定関数（ed_setrcr）

リスト 35 に、IPv6 用受信構成レジスタ設定関数を示します。以下は動作概要です。

(1) リスト 36 に示すマルチキャストフィルタ計算関数を呼出し、マルチキャストフィルタを計算します。
(2) マルチキャストアドレスレジスタ MAR に、マルチキャストフィルタを設定します。
(3) マルチキャストとユニキャストアドレスのみ受信するように、RCR を設定します。

9.6.3　マルチキャストフィルタ計算関数（ds_getmcaf）

リスト 36 に、マルチキャストフィルタ計算関数を示します。マルチキャストアドレス毎に、リスト 37 に示すイーサネットアドレス CRC 計算関数を呼出し、CRC を計算し、マルチキャストフィルタ mcaf に設定します。

リスト 35　IPv6 用受信構成レジスタ設定関数（ed_setrcr［tinet/netdev/if_ed/if_ed.c］）

```
static void ed_setrcr (T_IF_SOFTC *ic)
{
    /* マルチキャストフィルタを計算する。*/
    ds_getmcaf(ic, mcaf);

    /* マルチキャストの受信設定 */
    for (ix = 0; ix < 8; ix ++)
        sil_wrb_mem((VP)(sc->nic_addr + ED_P1_MAR(ix)), ((UB *)mcaf)[ix]);

    /* マルチキャストとユニキャストアドレスのみ受信するように設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RCR), ED_RCR_AM);
    }
```

リスト 36　マルチキャストフィルタ計算関数（ds_getmcaf［tinet/netdev/if_ed/if_ed.c］）

```
static void ds_getmcaf (T_IF_SOFTC *ic, UW *mcaf)
{
    UB *af = (UB*)mcaf;

    mcaf[0] = mcaf[1] = 0;

    for (count = MAX_IF_MADDR_CNT; count -- > 0; ) {
        index = ds_crc(ic->maddrs[count].lladdr) >> 26;
        af[index >> 3] |= 1 << (index & 7);
        }
    }
```

リスト 37　イーサネットアドレス CRC 計算関数（ds_crc［tinet/netdev/if_ed/if_ed.c］）

```
static UW ds_crc (UB *addr)
{
    UW crc = 0xffffffff;

    for (len = ETHER_ADDR_LEN; len -- > 0; ) {
        byte = *addr ++;
        for (bit = 8; bit -- > 0; ) {
            carry   = ((crc & 0x80000000) ? 1 : 0) ^ (byte & 0x01);
            crc   <<= 1;
            byte   >>= 1;
            if (carry)
                crc = (crc ^ POLYNOMIAL) | carry;
            }
        }
    return crc;
    }
```

9.6.4　イーサネットアドレス CRC 計算関数（ds_crc）

リスト 37 に、イーサネットアドレス CRC 計算関数を示します。計算対象はマルチキャストのイーサネットアドレスです。

リスト 38　送受信リングバッファ割当の定義（[tinet/netdev/if_ed/if_ed.c]）

```
#define ED_INT_TXBUF_START (ED_INT_RAM_BASE / ED_PAGE_SIZE)
#define ED_INT_TXBUF_STOP  (ED_INT_RAM_BASE / ED_PAGE_SIZE + IF_ED_TXBUF_PAGE_SIZE)
#define ED_INT_RXBUF_START ED_INT_TXBUF_STOP
#define ED_INT_RXBUF_STOP  ((ED_INT_RAM_BASE + ED_INT_RAM_SIZE) / ED_PAGE_SIZE)
```

表 12　割込みマスクレジスタ IMR の設定

ビット	割込み要因
PRX	受信（エラーなし）
PTX	送信（エラーなし）
RXE	受信エラー
TXE	コリジョン超過による送信エラー
OVW	受信バッファ超過

9.7　NIC 初期化本体関数（ed_init_sub）

リスト 39 に、NIC 初期化本体関数を示します。以下は動作概要です。

(1) レジスタ BNRY との関係で、次に読む受信フレームのページを指す sc->rxb_read は受信バッファの先頭アドレスの次のページに設定します。
(2) 送信バッファ制御に関係するメンバをゼロクリアします。
(3) 送信タイマをリセットします。
(4) 各種レジスタの設定中に動作が不安定にならないように、以下に示す操作により NIC を停止します。
　[1] レジスタ CR により送受信を停止する。
　[2] NIC 内蔵 SRAM へのアクセスも発生しないように、転送数を表すレジスタ RBCR をクリアする。
　[3] レジスタ RCR によりモニタモードにする。
　[4] レジスタ TCR によりループバックモードにする。
(5) リスト 38 に示す送受信リングバッファ割当に従って、送受信リングバッファの開始・終了ページアドレスを設定します。
(6) 全ての割込みフラグをリセットします。
(7) 割込みマスクレジスタ IMR に、表 12 に示す割込み許可を設定します。
(8) レジスタ PAR0 から PAR5 に MAC アドレスを設定します。
(9) レジスタ CURR に受信フレームを書込むページを設定します。
(10) レジスタ設定関数 ed_setrcr により受信構成レジスタ RCR を設定します。
(11) ループバックモードを終了し、送受信を有効にします。
(12) 送信可能セマフォを初期化します。

リスト 39　NIC 初期化本体関数（ed_init_sub［tinet/netdev/if_ed/if_ed.c］）

```
static void ed_init_sub (T_IF_SOFTC *ic)
{
    /* 次に読む受信フレームのページを設定する。 */
    sc->rxb_read = ED_INT_RXBUF_START + 1;

    /* 送信バッファ制御メンバを設定する。 */
    sc->txb_inuse = sc->txb_insend = 0;
    sc->txb_write = sc->txb_send   = 0;

    /* 送信タイマをリセットする。*/
    ic->timer = 0;

    /* NIC の動作が不安定にならないように送受信と 内蔵 SRAM へのアクセスを停止する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_CR),  ED_CR_PAGE0 | ED_CR_STP);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_DCR), ED_DCR_FT1 | ED_DCR_LS);

    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR0), 0);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RBCR1), 0);

    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_RCR), ED_RCR_MON);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TCR), ED_TCR_LB0);

    /* 送受信リングバッファを設定する。 */
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TPSR),   ED_INT_TXBUF_START);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_PSTART), ED_INT_RXBUF_START);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_PSTOP),  ED_INT_RXBUF_STOP);
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_BNRY),   ED_INT_RXBUF_START);

    /* 全ての割込みフラグをリセットする。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_ISR), 0xff);

    /* 割込みを許可する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_IMR),
         ED_IMR_PRX | ED_IMR_PTX | ED_IMR_RXE | ED_IMR_TXE | ED_IMR_OVW);

    /* MAC アドレスを設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_CR), ED_CR_RD2 | ED_CR_PAGE1 | ED_CR_STP);
    for (ix = 0; ix < ETHER_ADDR_LEN; ix ++)
        outb(sc->nic_addr + ED_P1_PAR(ix), ic->ifaddr.lladdr[ix]);

    /* 受信フレームを書き込むページを設定する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P1_CURR), sc->rxb_read);

     /* 受信構成レジスタ (RCR) を設定する。*/
     ed_setrcr(ic);

/* ループバックモードを終了する。*/
    sil_wrb_mem((VP)(sc->nic_addr + ED_P0_TCR), 0);

    /* 送信可能セマフォを初期化する。*/
    for (ix = NUM_IF_ED_TXBUF; ix --; )
        sig_sem(ic->semid_txb_ready);

    /* ターゲット依存部の割込み初期化 */
    ed_inter_init();
    }
```