

OAKS32R

ソフトウェアマニュアル

安全設計に関するお願い

- ・弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- ・本資料は、お客様が用途に応じた適切な製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてオクス電子および情報を提供いただいた各社が所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- ・本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、オクス電子は責任を負いません。
- ・本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、オクス電子は特性改良などにより予告なしに変更することがあります。
- ・本資料に記載の図、表に示す技術的な内容、及びプログラム、アルゴリズムを流用する場合、お客様の責任において実施してください。また、組み込んだプログラム、アルゴリズム単体で評価するだけでなく、システム全体で十分に評価してください。オクス電子は、一切責任を負いません。
- ・本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、オクス電子へご照会ください。
- ・本資料の転載、複製については、文書によるオクス電子の事前の承諾が必要です。
- ・本資料に関し詳細についてのお問い合わせ、その他お気づきの点がございましたらオクス電子までご照会ください。

1. 概要.....	3
2. 動作環境.....	3
2.1. プログラム作成環境.....	3
2.2. プログラム動作環境.....	3
3. ユーザプログラム作成手順.....	4
3.1. コンフィグレーション設定.....	4
3.2. プログラム配置.....	5
3.2.1. メモリ配置.....	5
3.2.2. セクション配置.....	5
3.3. プログラムの起動手順.....	7
3.4. プログラム作成手順例.....	8
3.5. プログラム作成時の注意事項.....	10
3.5.1. ポートの設定について.....	10
3.5.2. タスクの優先度.....	10
3.5.3. EIT.....	10
4. 関数一覧.....	11
4.1. リアルタイム OS(M3T-MR32R).....	11
4.2. TCP/IP プロトコルスタック(M3S-TCP32R).....	14
4.3. I/O ポート操作.....	15
4.4. リアルタイム OS 拡張機能.....	15
5. ミドルウェア関数詳細.....	17
5.1. リアルタイム OS(M3T-MR32R).....	17
5.2. TCP/IP プロトコルスタック(M3S-TCP32R).....	17
5.3. I/O ポート操作.....	19
5.4. リアルタイム OS 拡張機能.....	24

本書では、OAKS32R に ROM 化して添付しているソフトウェアの使用方法を説明します。

1. 概要

ボードのフラッシュ ROM には下記のソフトウェアが入っています。

- ・リアルタイム OS M3T-MR32R
- ・TCP/IP プロトコルスタック M3S-TCP32R
- ・スタートアップ

このため、上記のミドルウェアや OS を使用した応用製品を開発する場合、ソフトウェアの動作確認や性能検証が簡単にできます。実際の応用製品にソフトウェアを搭載する場合には、上記ミドルウェアや OS は別途購入いただく必要があります。

2. 動作環境

2.1. プログラム作成環境

ROM 上の OS やミドルウェアを使用したプログラムは、添付 CD-ROM に入っているコンパイラ(M3T-TW32R 機能限定版)を使用し、PC 上で作成します。プログラム作成方法の詳細は第 3 章を参照ください。

2.2. プログラム動作環境

電源を投入すると、まず最初にハードウェアの初期化プログラムが実行され、以下の初期化が実行されます。

- ・メモリマップ
- ・CPU の動作周波数 (66.6MHz)
- ・プログラムの転送 (フラッシュ ROM から SDRAM へ)
- ・データの初期化
- ・OS の初期化
- ・初期設定用タスクの起動
 - ユーザーのコンフィグレーション関数呼び出し
 - LAN コントローラの初期化

3. ユーザプログラム作成手順

3.1. コンフィグレーション設定

タスクやイベントフラグなどは configuration という関数で、cre_tsk 等のシステムコールを使用して生成します。この configuration 関数は、システム起動時に自動的に起動されるタスク（優先度 1）からサブルーチンコールされます。

オブジェクトを生成する場合、以下の点に注意ください。

1. 以下のオブジェクト ID 番号は使用できません。

[タスク] 1 : TCP/IP 用タスク

2 : 初期設定用タスク

[セマフォ] 1、2 : TCP/IP 用

[周期起動ハンドラ] 1, 2, 3, 4, 5 : TCP/IP 用

[割り込みハンドラ] 1 : TCP/IP 用

16 : OS のシステムタイマ (MFT00)

2. 各オブジェクトの最大値は以下のとおりです。

タスク数 : 30 (上記 2 つを含む)

各タスクのスタックサイズ : 最大 32K バイト

: 合計 512K バイト

イベントフラグ : 15

セマフォ : 15 (上記 2 つを含む)

メールボックス : 15

メッセージ数 : 最大 256

: 合計 15KB

メッセージバッファ : 10

メッセージバッファサイズ : 最大 1K バイト

: 合計 10KB

ランデブポート : 5

優先度付きメールボックス : 5

固定長メモリーブール : 15

メモリーブロック数 × ブロックサイズ : 最大 4K バイト

: 合計 60KB

可変長メモリーブール : 5

メモリーブールサイズ : 最大 1024K バイト

: 合計 2048K バイト

周期起動ハンドラ : 20 (上記 5 つを含む)

割り込みハンドラ : 64

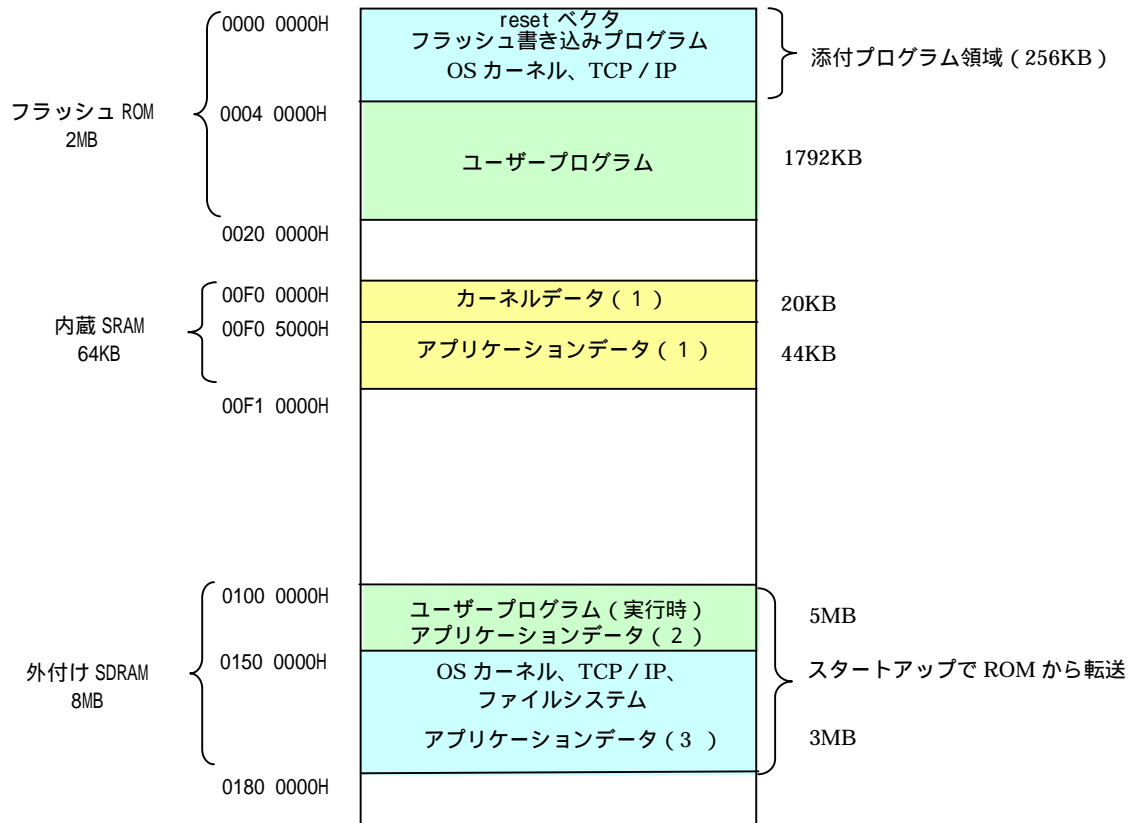
(実際に使用できる割り込みはハードウェアの仕様に依存します)

3.2. プログラム配置

プログラムは SDRAM (01000000H ~ 017FFFFFFH 番地) 上にコピーされ、SDRAM 上で実行されます。

3.2.1. メモリ配置

メモリ配置は以下のとおりです。



アプリケーションデータ (3) は可変長メモリプールの機能を使用することで獲得できます。

3.2.2. セクション配置

標準では以下のセクションを配置します。

- .text セクション

C 言語のプログラムが配置されるセクションです。

フラッシュ ROM 上に配置され、初期化時に SDRAM にコピーされ、実行されます。

- .rodata セクション

C 言語の定数データが配置されるセクションです。

フラッシュ ROM 上に配置され、初期化時に SDRAM にコピーされます。

- .data, .sdata セクション

C 言語の初期値有りデータが配置されるセクションです。

フラッシュ ROM 上に配置され、初期化時に SDRAM にコピーされます。

3 ユーザプログラム作成手順

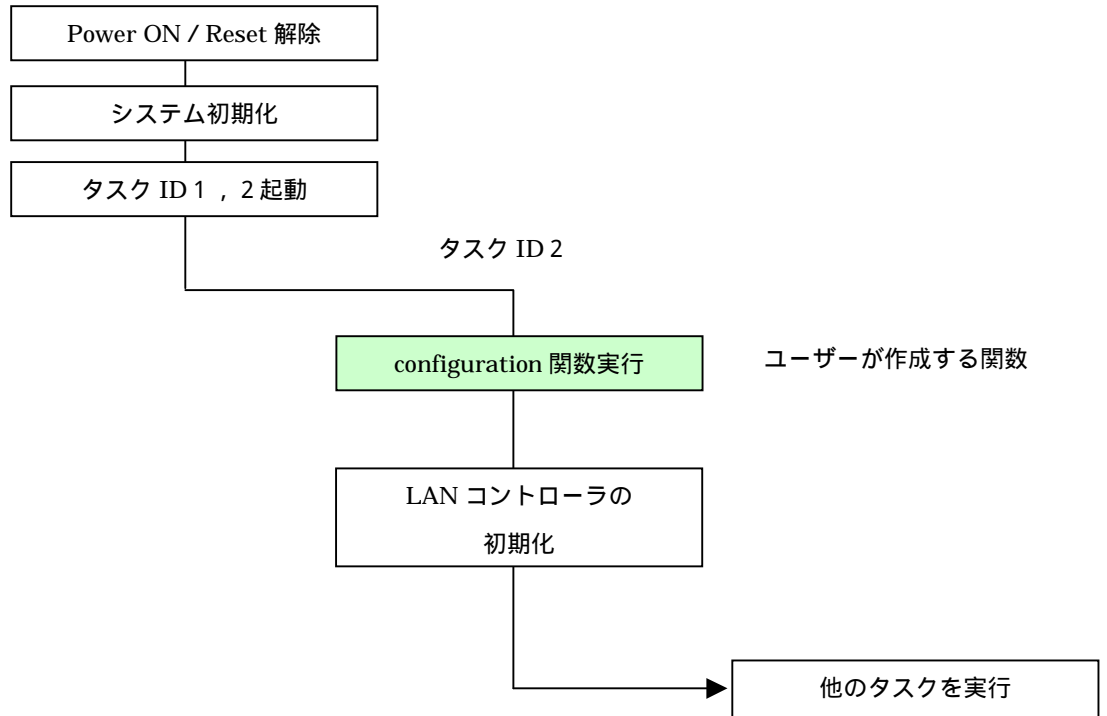
- **.bss,.sbss セクション**
C 言語の初期値無しデータが配置されるセクションです。
SDRAM 上に配置され、初期化時に 0 クリアされます。
- **.CONFIG セクション**
タスクなどのオブジェクトの生成を行う関数を呼び出すルーチンが入っています。
このセクションは 40000H 番地に配置するようにしなければいけません。
- **.OS_START セクション**
フラッシュ ROM 上のプログラムやデータを SDRAM にコピーするプログラムが置かれています。このセクションは 40100H 番地に配置するようにしなければいけません。
- **.mwif セクション**
ミドルウェアのインタフェースルーチンが配置されるセクションです。
フラッシュ ROM 上に配置され、初期化時に SDRAM にコピーされ、実行されます。
- **.MR_KERNEL セクション**
M3TR-MR32R システムコールのインタフェースルーチンが配置されるセクションです。
フラッシュ ROM 上に配置され、初期化時に SDRAM にコピーされ、実行されます。

【セクション配置に関する注意事項】

- **.text,.rodata,.data,.sdata,.bss,.sbss** セクションはコンパイラの出力する標準セクション名です。これらのセクションはシステムが自動的にコピー、初期化します。
- **.OS_START, .CONFIG, .meif, .MR_KERNEL** セクションはシステムが使用するセクション名です。
- セクションを追加することは可能です。セクションを追加した場合に ROM 領域から RAM 領域に転送して使用する必要があるものについては、別途転送プログラムを作成し、リンクする必要があります。

3.3. プログラムの起動手順

プログラムは以下のように起動されます。



3.4. プログラム作成手順例

プログラムのコーディングから実行までを、例を用いて説明します。

1. コーディング

プログラムとして task1,task2 の 2 つのタスクでメッセージをやり取りするプログラムを想定します。

使用するオブジェクトはタスク 2 つ、メールボックス 1 つですので、ソースファイルにはそれを生成する関数 (関数名 : configuration) と 2 つのタスク関数を記述します。

smp.c というファイル名で記述した例を示します。

smp.c の例

```
#include <mr32r.h>

#define ID_task1  3
#define ID_task2  4
#define ID_mbx1  1

void configuration(void) /* 必ず configuration という関数名にする */
{
    T_CTSK  ctask;
    T_CMBX  cmbx;

    /* task1 の生成と起動 */
    ctask.tskatr = __MR_EXT;
    ctask.task = task1;
    ctask.itskpri = 7;
    ctask.stksz = 1024;
    cre_tsk(ID_task1, &ctask);
    sta_tsk( ID_task1, 0);

    /* task2 の生成と起動 */
    ctask.tskatr = __MR_EXT;
    ctask.task = task2;
    ctask.itskpri = 8;
    ctask.stksz = 1024;
    cre_tsk(ID_task2, &ctask);
    sta_tsk( ID_task2, 0);

    /* メールボックスの生成 */
    cmbx.mbxatr = __MR_EXT;
    cmbx.bufcnt = 1;
    cre_mbx( ID_mbx1, &cmbx );
}

void task1(void) /* メッセージ受信タスク */
{
    long  msg;

    while(1){
        rcv_msg( &msg, ID_mbx1 );
    }
}

void task2(void) /* メッセージ送信タスク */
{
```

3 ユーザプログラム作成手順

```
long msg;

msg = 1;
while(1){
    snd_msg( ID_mbx1, msg );
    msg++;
}
}
```

2 . オブジェクトの作成

上記で作成したファイルをコンパイル、リンクします。

- (1) コンパイル時には、「-D TRON -DINET -D KERNEL -D NOT_FASTMEMCPY」をコンパイルオプションとして指定します。また、コンパイル時に TCP/IP 関連のヘッダファイルでワーニングが発生することがありますが特に問題となることはありません。
- (2) リンク時には、“mwif.o”および”libc32rnr.a”をリンクするようにします。
- (3) 「3.2.2 セクション配置」に示すように各セクションを配置します。

3 . ダウンロード

作成したオブジェクトファイルをダウンロードして、実行します。

ダウンロードするには以下の2つの方法があります。

- (1) デバッガでダウンロード
- (2) シリアルからダウンロード

以下にシリアルからダウンロードする方法を説明します。

【シリアルからダウンロード】

- (a) オブジェクトファイルをモトローラ S フォーマットに変換
- (b) ボード上のジャンパピン(JP1)をオープンにして電源を投入
- (c) シリアルダウンロードモードになると LED0 が点滅
- (d) PC からシリアルにモトローラ S フォーマットのファイルを転送

シリアルは以下のように設定します。シリアルを受信すると、LED0 と LED1 が交互に点滅します。

```
115200bps
データビット : 8 ビット
ストップビット : 1 ビット
パリティビット : なし
フロー制御 : なし
```

- (e) フラッシュ ROM 書き込み中は LED0 が点滅、LED1 は消灯します
- (f) ダウンロードが完了すると LED0,LED1 が短い間隔で点滅
- (g) 電源を切り、ジャンパピン(JP1)をクローズし、再度電源を入れます。

3.5. プログラム作成時の注意事項

3.5.1. ポートの設定について

ポートの設定は4.3節に示すポート操作関数を使用しなければなりません。

3.5.2. タスクの優先度

タスクの優先度は 31 までが設定可能です。

初期設定用タスクが優先度 10 で初期設定終了後は起動されることはありません。

TCP/IP 用タスクが優先度 5 です。したがって、優先度 1～4 のタスクが実行されたままになると TCP/IP の通信処理ができなくなります。

3.5.3. EIT

- ・ 外部割り込みは、割り込みハンドラを定義する必要があります。
- ・ TRAP やその他例外を定義することはできません。

4 関数一覧

4. 関数一覧

ユーザーが利用する関数は、インターフェースルーチンをライブラリで提供します。そのインターフェースルーチンはROM内の関数にテーブルジャンプを行います。

以下に関数の一覧を示します。

4.1. リアルタイム OS(M3T-MR32R)

システムコール名	機能
cre_tsk	タスクを生成します
del_tsk	タスクを削除します
sta_tsk	タスクを起動します
ista_tsk	タスクを起動します(ハンドラ専用)
ext_tsk	自タスクを正常終了します
exd_tsk	自タスクを終了後削除します
ter_tsk	他タスクを強制的に異常終了します
chg_pri	タスクの優先度を変更します
ichg_pri	タスクの優先度を変更します(ハンドラ専用)
dis_dsp	タスクのディスパッチを禁止します
ena_dsp	タスクのディスパッチを許可します
rot_rdq	レディキューを回転します
irotd_rdq	レディキューを回転します(ハンドラ専用)
rel_wai	待ち状態を強制解除します
irel_wai	待ち状態を強制解除します(ハンドラ専用)
get_tid	自タスクの ID を得ます
ref_tsk	タスクの状態を参照します
sus_tsk	タスクを強制待ち状態に移行します
isus_tsk	タスクを強制待ち状態に移行します(ハンドラ専用)
rsm_tsk	強制待ち状態のタスクを再開する
irms_tsk	強制待ち状態のタスクを再開する(ハンドラ専用)
slp_tsk	タスクを待ち状態へ移行する
tslp_tsk	タスクを一定時間待ち状態へ移行する
wup_tsk	待ち状態のタスクを起床します
iwup_tsk	待ち状態のタスクを起床します(ハンドラ専用)
can_wup	タスクの起床要求を無効にします
cre_flg	イベントフラグを生成します
del_flg	イベントフラグを削除します
set_flg	イベントフラグをセットします
iset_flg	イベントフラグをセットします (ハンドラ専用)
clr_flg	イベントフラグをクリアします
wai_flg	イベントフラグを待ちます
twai_flg	イベントフラグを待ちます(タイムアウトあり)
pol_flg	イベントフラグを得ます(待ちなし)
ref_flg	イベントフラグの状態を参照します
cre_sem	セマフォを生成します
del_sem	セマフォを削除します
sig_sem	セマフォ資源を返却します
isig_sem	セマフォ資源を返却します(ハンドラ専用)
wai_sem	セマフォ資源を獲得します
twai_sem	セマフォ資源を獲得します(タイムアウトあり)
preq_sem	セマフォ資源を獲得します(ポーリング)
ref_sem	セマフォの状態を参照します
cre_mbx	メールボックスを生成します

4 関数一覧

システムコール名	機能
del_mbx	メールボックスを削除します
snd_msg	メッセージを送信します
isnd_msg	メッセージを送信します(ハンドラ専用)
rcv_msg	メッセージを受信します
trcv_msg	メッセージを受信します(タイムアウトあり)
prcv_msg	メッセージを受信します(待ちなし)
ref_mbx	メールボックスの状態を参照します
cre_mbf	メッセージバッファを生成します
del_mbf	メッセージバッファを削除します
snd_mbf	メッセージバッファへ送信します
tsnd_mbf	メッセージバッファへ送信します(タイムアウトあり)
psnd_mbf	メッセージバッファへ送信します(待ちなし)
rcv_mbf	メッセージバッファから受信します
trcv_mbf	メッセージバッファから受信します(タイムアウトあり)
prcv_mbf	メッセージバッファから受信します(待ちなし)
ref_mbf	メッセージバッファの状態を参照します
cre_por	ランデブ用ポートを生成します
del_por	ランデブ用ポートを削除します
cal_por	ポートに対するランデブ呼出を行います
tcac_por	ポートに対するランデブ呼出を行います(タイムアウトあり)
pcac_por	ポートに対するランデブ呼出を行います(待ちなし)
acp_por	ポートに対するランデブ受付を行います
tacp_por	ポートに対するランデブ受付を行います(タイムアウトあり)
pacp_por	ポートに対するランデブ受付を行います(待ちなし)
fwd_por	ランデブを別のポートに回送します
rpl_rdv	ランデブ返答を行います
ref_por	ランデブ用ポートを参照します
def_int	割り込みハンドラを定義します
ret_int	割り込みハンドラから復帰します
loc_cpu	割り込みとディスパッチを禁止します
unl_cpu	割り込みとディスパッチを許可します
cre_mpf	固定長メモリプールを生成します
del_mpf	固定長メモリプールを削除します
get_blf	メモリブロックを獲得します
tget_blf	メモリブロックを獲得します(タイムアウトあり)
pget_blf	メモリブロックを獲得します(待ちなし)
rel_blf	メモリブロックを解放します
irel_blf	メモリブロックを解放します
ref_mpf	固定長メモリプールの状態を参照します
cre_mpl	可変長メモリプールを生成します
del_mpl	可変長メモリプールを削除します
get_blk	メモリブロックを獲得します
tget_blk	メモリブロックを獲得します(タイムアウトあり)
pget_blk	メモリブロックを獲得します(待ちなし)
rel_blk	メモリブロックを解放します
ref_mpl	可変長メモリプールの状態を参照します
set_tim	システムクロックを設定します
get_tim	システムクロックを参照します
dly_tsk	タスクの実行を一定時間遅延します

4 関数一覧

システムコール名	機能
act_cyc	周期起動ハンドラの活性制御を行います
ref_cyc	周期起動ハンドラの状態を参照します
ref_alm	アラームハンドラの状態を参照します
get_ver	OSのバージョンを参照します
ref_sys	システムの状態を参照します
def_exc	例外ハンドラの定義を行います
vclr_ems	例外マスクをクリアします
vset_ems	例外マスクをセットします
vras_fex	強制例外ハンドラを起動します
vrst_msg	メールボックスをクリアします
vrst_mbf	メッセージバッファをクリアします
vrst_blf	固定長メモリプールを解放します
vrst_blk	可変長メモリプールを解放します
vcre_mbx	優先度付きメールボックスを生成します
vdel_mbx	優先度付きメールボックスを削除します
vsnd_mbx	優先度付きメッセージを送信します
visnd_mbx	優先度付きメッセージを送信します(ハンドラ専用)
vrcv_mbx	優先度付きメッセージを受信します(タイムアウトなし)
vtrcv_mbx	優先度付きメッセージを受信します(タイムアウトあり)
vprcv_mbx	優先度付きメッセージを受信します(まちなし)
vrst_mbx	優先度付きメールボックスをクリアします
vrref_mbx	優先度付きメールボックスの状態を参照します

4.2. TCP/IP プロトコルスタック(M3S-TCP32R)

関数名	機能
unix_socket	ソケットの生成
unix_bind	ソケットへの名前のバインド
unix_listen	ソケット上での接続の受け入れ準備
unix_accept	ソケットへの接続の受け付け
unix_connect	ソケットへの接続の開始
unix_send	ソケットからのメッセージ送信
unix_sendto	ソケットからのメッセージ送信
unix_recv	ソケットからのメッセージ受信
unix_recvfrom	ソケットからのメッセージ受信
unix_read	入力の読み取り (ソケットからのメッセージ受信)
unix_write	出力の書き込み (ソケットからのメッセージ送信)
unix_shutdown	全 2 重接続の部分シャットダウン
unix_close	ソケットのクローズ
unix_ioctl	デバイスの制御 (ソケットの制御)
unix_getsockopt	ソケットオプションの取得
unix_setsockopt	ソケットオプションの設定
unix_getsockname	ソケット名の取得
unix_getpeername	接続している相手側の名前の取得
inet_addr	IP アドレスをネットワークバイトオーダーのバイナリ値へ変換
inet_ntoa	ネットワークバイトオーダーの IP アドレスをドット付き ASCII 文字列へ変換
ifconfig	IP アドレスの初期化
getErrno	errno に設定された値を取得する

4 関数一覧

4.3. I/O ポート操作

関数名	機能
sfrReadPxMOD	ポート動作モードレジスタ読み出し
sfrWritePxMOD	ポート動作モードレジスタへの書き込み
sfrReadPxDIR	ポート方向レジスタの読み出し
sfrWritePxDIR	ポート方向レジスタへの書き込み
sfrReadPxDATA	ポートデータレジスタへの読み出し
sfrWritePxDATA	ポートデータレジスタへの書き込み
sfrReadPxODCR	ポートオーブンドレインレジスタの読み出し
sfrWritePxODCR	ポートオーブンドレインレジスタへの書き込み
sfrWritePxIEN	ポート入力制御
sfrUpDatePxDIR	ポート方向レジスタの更新

4.4. リアルタイム OS 拡張機能

関数名	機能
def_cyc	周期起動ハンドラの定義

5. ミドルウェア関数詳細

本章では、各ミドルウェア関数の詳細を示します。各関数詳細の読み方は以下のとおりです。

関数名		分類
機能概要		
書式	関数の呼び出し形式を示します。#include “ヘッダファイル”で示すヘッダファイルは、この関数の実行に必要な標準ヘッダファイルです。必ずインクルードしてください。 I,O は、引数がそれぞれ入力データ、出力データであることを意味します。	
戻り値	関数の戻り値を示します。戻り値の後に「:」を付けて記載されているコメントは、その戻り値についての説明(リターン条件等)です。	
解説	関数の仕様について説明します。	
注意	注意事項があればここに示します。	
使用例	関数の使用例をここで示します。	
作成例	関数の作成例をここで示します。	

図 5.1 ライブラリ関数詳細の見方

5.1. リアルタイム OS(M3T-MR32R)

M3T-MR32R のマニュアルを参照ください。

5.2. TCP/IP プロトコルスタック(M3S-TCP32R)

以下に示す ifconfig 関数以外の関数に関しては、M3S-TCP32R のマニュアルを参照ください。

ifconfig

IP アドレス設定関数

IP アドレス設定

書式

```
#include <netcfg.h>
int ifconfig ( NETCFG *netcfg )
    NETCFG *netcfg          |   IP アドレス情報
```

netcfg の内容

unsigned long	myaddr	自 IP アドレス
unsigned long	maskaddr	マスクアドレス
unsigned long	broadaddr	ブロードキャストアドレス
unsigned long	gwaddr	ゲートウェイアドレス

戻り値 0 : 正常終了
 -1 : 異常終了

解説 自 IP アドレスを設定し、ネットワークを有効化します。
 ゲートウェイアドレスが"0"の場合は、ゲートウェイの設定を行いません。

getErrno

エラー番号取得関数

エラー番号取得

書式

```
#include <netcfg.h>
#include <errno.h>
int getErrno( void )
```

戻り値 エラー番号を返します

解説 errno に設定された値を取得します。

sfrReadPxMOD

ポート操作関数

ポート動作モードレジスタ読み出し

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrReadPxMOD(int id);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
```

戻り値 ポート動作モードレジスタ値

解説 id で指定したポートの動作モードレジスタの内容を読み出します。
読み出しデータは内部動作モードレジスタ値保存バッファにコピーしてあるレジスタ値です。

sfrWritePxMOD

ポート操作関数

ポート動作モードレジスタへの書き込み

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrWritePxMOD(int id,unsigned short data);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
    unsigned short data   |   動作モードレジスタへ書き込む値
```

戻り値 書き込んだデータ

解説 id で指定したポートの動作モードレジスタへ data で指定した値を書き込みます。
書き込みデータは内部動作モードレジスタ保存バッファにも書き込まれます。

sfrReadPxDIR

ポート操作関数

ポート方向レジスタの読み出し

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned char sfrReadPxDIR(int id);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
```

戻り値 ポート方向レジスタ値

解説 id で指定したポートの方向レジスタの内容を読み出します。
読み出しデータは内部方向レジスタ値保存バッファにコピーしてあるレジスタ値です。

sfrWritePxDIR

ポート操作関数

ポート方向レジスタへの書き込み

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned char sfrWritePxDIR(int id,unsigned char data);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
    unsigned char data    |   ポート方向レジスタへ書き込む値
```

戻り値 書き込んだデータ

解説 id で指定したポートの方向レジスタへ data で指定した値を書き込みます。
書き込みデータは内部方向レジスタ保存バッファにも書き込まれます。

sfrReadPxDATA

ポート操作関数

ポートデータレジスタの読み出し

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrReadPxDATA (int id);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
```

戻り値 ポートデータレジスタ値

解説 id で指定したポートのデータレジスタの内容を読み出します。

sfrWritePxDATA

ポート操作関数

ポートデータレジスタへの書き込み

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrWritePxDATA (int id, unsigned short data);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
    unsigned short data   |   出力制御レジスタへ書き込む値
```

戻り値 書き込んだデータ

解説 id で指定したポートのデータレジスタへ data で指定した値を書き込みます。
書き込みデータは PiDATA レジスタのみに書き込み、保存バッファにリード参照用保存データを作成しません。

sfrReadPxODCR

ポート操作関数

オープンドレインレジスタの読み出し

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrReadPxODCR (int id);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
```

戻り値 オープンドレインレジスタ値

解説 id で指定したポートのオープンドレインレジスタの内容を読み出します。

sfrWritePxODCR

ポート操作関数

オープンドレインレジスタへの書き込み

書式

```
#include <m32102.h>
#include <sfr_func.h>
unsigned short sfrWritePxODCR (int id,unsigned short data);
    int id                |   ポート ID ( ID_P0 ~ ID_P7 )
    unsigned short data   |   オープンドレインレジスタへ書き込む値
```

戻り値 書き込んだデータ

解説 id で指定したポートのオープンドレインレジスタへ data で指定した値を書き込みます。
書き込みデータはオープンドレインレジスタ保存バッファにも書き込まれます。

sfrSetPortInput

ポート操作関数

ポート入力制御

書式

```
#include <m32102.h>
#include <sfr_func.h>
void sfrSetPortInput (unsigned char mode);
    unsigned char mode      |      0 : ポート入力禁止 (貫通電流防止)
                           |      1 : ポート入力許可
```

戻り値 なし

解説

ポート入力許可レジスタに mode で指定した値を書き込み、ポートの入力許可/禁止を制御します。

ポート入力許可レジスタの初期値は'0'で、入力禁止状態です。

ポートを入力ポートとして使用する場合は、本関数にてポート入力を許可にする必要があります。

sfrUpdatePxDIR

ポート操作関数

ポート方向レジスタの更新

書式

```
#include <m32102.h>
#include <sfr_func.h>
void sfrUpdatePxDIR(void);
```

戻り値 なし

解説

全ポート方向レジスタを更新します。

ノイズ等により方向レジスタの内容変化に対する処理例です。

def_cyc

周期起動ハンドラ定義

書式

```
#include <mr32r.h>
#include <os_ext.h>
int def_cyc(HNO cycno, T_DCYC *pk_dcyc);
    HNO cycno          |   周期起動ハンドラ番号
    T_DCYC *pk_dcyc   |   周期起動ハンドラ定義情報
```

pk_dcyc の内容

VP	exinf	拡張情報
ATR	cycatr	周期起動ハンドラ属性
FP	cychdr	周期起動ハンドラ開始アドレス
UINT	cycact	周期起動ハンドラ活性状態
CYCTIME	cyctim	周期起動時間間隔

戻り値

E_OK 正常終了

解説

cycno で指定された番号に対応する周期起動ハンドラを定義します。
exinf・cycatr は、本システムでは使用しません。cychdr は、周期起動ハンドラの開始アドレス、cyctim は、周期起動ハンドラの起動間隔、cycact は周期起動ハンドラの活性状態を指定します。def_cyc で定義された周期起動ハンドラは、act_cyc によって TCY_OFF にするか周期起動ハンドラを再定義するまで周期起動が繰り返されます。
本システムコールによって周期起動ハンドラの定義を解除することはできません。本システムコールは、タスクからのみ発行可能です。

OAKS32R ソフトウェアマニュアル Rev 0.80
2003年2月発行

編集 オークス電子株式会社
発行 オークス電子株式会社
禁無断転載

本説明書の一部又は全部を、当社に断りなく、いかなる形でも転載又は複製することを堅くお断りします。