カーネル勉強会資料

金光　恭治

2001 年 9 月 17 日

1　タイマ

1.1　タイマ関連モジュール構成

タイマに関するモジュールの構成は図 1 に示すように，ハードウェアタイマ，CPU 依存タイマモジュール (hw_timer.h)，システムクロックドライバ (timer.c,timer.h) があり，その上に時間管理に関するシステムコールがある．

ハードウェアタイマは SH3 がハードウェアとして持つタイマユニット (TMU) であり，CPU 依存タイマモジュールはそれとのインターフェースである．システムクロックドライバは汎用的な時間管理機能を提供する．

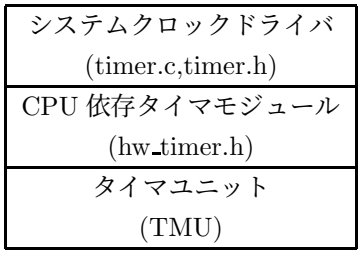


図 1　タイマ関連モジュール構成

1.2　SH3 のタイマユニット (TMU)

TMU は 3 つのチャネル（チャネル 0〜2）の 32 ビットタイマにより構成される．

1.2.1　TMU のレジスタ構成

各チャネルには，32 ビットのタイマカウンタ (TCNT) と 32 ビットのタイマコンスタントレジスタ (TCOR)，およびタイマコントロールレジスタ (TCR) がある．TCNT カウンタは，ダウンカウント動作を行う．また，全チャネル共通のレジスタとして，タイマスタートレジスタ (TSTR) がある．表 1 に TMU のレジスタ構成，表 2 に TMU のレジスタ概要を示す．

表 1　TMU レジスタ構成

チャネル	レジスタ	略称	R/W	アドレス	サイズ
共通	タイマアウトプットコントロールレジスタ	TOCR	R/W	0xfffffe90	8
	タイマスタートレジスタ	TSTR	R/W	0xfffffe92	8
0	タイマコンスタントレジスタ 0	TCOR0	R/W	0xfffffe94	32
	タイマカウンタ 0	TCNT0	R/W	0xfffffe98	32
	タイマコントロールレジスタ 0	TCR0	R/W	0xfffffe9C	16
1	タイマコンスタントレジスタ 1	TCOR1	R/W	0xfffffeA0	32
	タイマカウンタ 1	TCNT1	R/W	0xfffffeA4	32
	タイマコントロールレジスタ 1	TCR1	R/W	0xfffffeA8	16
2	タイマコンスタントレジスタ 2	TCOR2	R/W	0xfffffeAC	32
	タイマカウンタ 2	TCNT2	R/W	0xfffffeB0	32
	タイマコントロールレジスタ 2	TCR2	R/W	0xfffffeB4	16
	インプットキャプチャレジスタ 2	TCPR2	R	0xfffffeB8	32

表 2　TMU レジスタ概要

レジスタ	チャネル	概要
タイマアウトプット コントロールレジスタ (TOCR)	共通	外部端子の TCLK を外部クロックもしくはインプットキャプチャ制御用の入力端子とするか，内臓 RTC（リアルタイムクロック）の出力クロック用の出力端子とするかを選択
タイマスタートレジスタ (TSTR)	共通	各チャネルのタイマカウンタを動作させるか， 停止させるか選択
タイマコンスタント レジスタ (TCOR)	0～2	タイマカウンタのアンダーフローが発生したとき， タイマカウンタにセットする値を指定
タイマカウンタ (TCNT)	0～2	入力したカウントクロックおきにカウントダウン動作を行う
タイマコントロールレジスタ (TCR)	0～2	タイマカウンタの制御および割り込みの制御を行う
インプットキャプチャレジスタ (TCPR2)	2	インプットキャプチャ用

1.2.2　動作説明

TSTR の STR0～STR2 ビットを 1 にすると対応するチャネルの TCNT はカウント動作を開始する．TCNT カウンタがアンダーフローすると対応する TCR の UNF フラグがセットされる．このとき，TCR レジスタの UNIE ビットが 1 ならば，CPU に割り込み要求を出す．また，このとき TCNT カウンタには TCOR レジスタから値がコピーされ，ダウンカウント動作を再実行する．なお，カウントクロックは TCR の TPSC2～TPSC0 の値で決定される．図 2 に TMU の動作を示す．

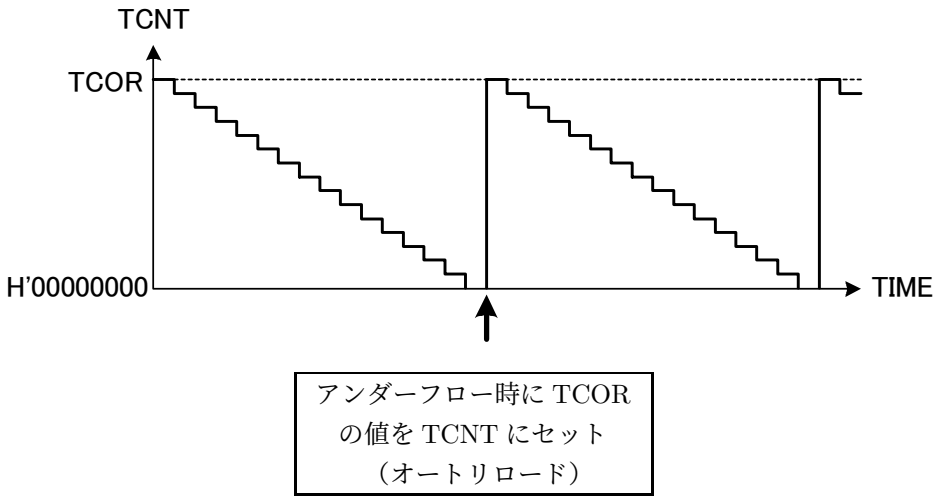


図 2　TMU の動作

1.2.3 TIMER_CLOCK の設定値

システムクロックドライバが isig_tim を呼び出すタイムティックの周期は，sys_defs.h 中の TIC_NUME（周期の分子）と TIC_DENO（周期の分母）で定義されている（初期値は共に 1）．この値を使って，hw_timer.h 内のマクロ関数 TO_CLOCK で TIMER_CLOCK から TCOR への設定値を計算している．TIMER_CLOCK はシステムの周辺モジュール用クロック $P\phi$（CPU クロックと同じ，もしくは 1/2，1/4）の 1/4，1/16，1/64，1/256 のいずれのカウンタクロックにおいても，タイムティック供給間隔が同じになるように，それぞれの値に対応する値を設定する必要がある．JSP カーネルでは，タイムティック供給間隔が $1msec(= 1000\mu sec)$ になるように次の式で TIMER_CLOCK 値を設定している．

1 カウントにかかる時間 $[\mu sec]$ = 1（周期） / カウントクロック [MHz]
TIMER_CLOCK = $1000[\mu sec]$ / 1 カウントにかかる時間 $[\mu sec]$

例：DVE-SH7700 用の設定値
$P\phi$ = 30MHz において

$P\phi/4$ = 7.5MHz のとき	1 / 7.5 = $0.133[\mu sec]$	TIMER_CLOCK = 1000 / 0.133 = 7500
$P\phi/16$ = 1.875MHz のとき	1 / 1.875 = $0.533[\mu sec]$	TIMER_CLOCK = 1000 / 0.533 = 1875
$P\phi/64$ = 0.469MHz のとき	1 / 0.469 = $2.133[\mu sec]$	TIMER_CLOCK = 1000 / 2.133 = 469
$P\phi/256$ = 0.117MHz のとき	1 / 0.117 = $8.545[\mu sec]$	TIMER_CLOCK = 1000 / 8.545 = 117

以上の例のようにそれぞれのカウントクロックに対応した TIMER_CLOCK が得られる．ただし，32 ビットカウンタの最高動作周波数は 2MHz なので，それ以下のカウンタクロックを設定し，それに対応した TIMER_CLOCK を設定しておく．つまり，例の $P\phi/4$ としてのカウントクロックは設定することができない．また，TIMER_CLOCK の値が大きい方が細かいカウントが行えるため，出来るだけ大きい TIMER_CLOCK 値を利用する．これらの条件より，JSP カーネルでは DVE-SH7700 用の $P\phi$ = 30MHz 時の TIMER_CLOCK に，$P\phi/16$ = 1.875MHz のときの値 1875 を sys_config.h 上で設定している．

1.3 CPU 依存タイマモジュール (hw_timer.h)

CPU 依存タイマモジュールである hw_timer.h は，ハードウェアタイマの制御，参照を直接行う低レベルのインターフェースである．JSP カーネルでは，SH3 の TMU が持つ 3 つのチャネルのうち，チャネル 0 を使用する．

1.3.1 hw_timer.h におけるレジスタへの設定値

I. タイマスタートレジスタ

7	6	5	4	3	2	1	0
—	—	—	—	—	STR2	STR1	STR0

マクロ	マクロ値	対象ビット	機能
TMU_STR0	0x01	STR0	1:TCNT0 はカウント動作，0:カウント動作停止

II. タイマコントロールレジスタ 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	—	—	—	—	—-	—	UNF	—-	—	UNIE	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0

マクロ	マクロ値	対象ビット	機能
TCR0_TPSC	0x0001 or 0x0000	TPSC2〜1	0x0001:P φ/16，0x0000:P φ/4 でカウント
—	0x020	UNIE	1:UNF による割込みを許可，0:割込みを許可しない
TCR_UNF	0x0100	UNF	1:TCNT がアンダーフローした，0:アンダーフローしていないことを示す

III. タイマコンスタントレジスタ 0，タイマカウンタ 0

タイマコンスタントレジスタ 0 およびタイマカウンタ 0 は 32 ビットレジスタなので，0xffffffff までの値を設定できる．JSP カーネルではタイマを動作させる前の初期化段階で，MAX_CLOCK(=0xffffffff) 以下の値を設定しておく．

1.3.2 TMU のデータ構造

JSP カーネルでは，TMU のデータ構造を./config/sh3/sh3.h において，次の構造体により定義している．

```
/*
 * タイマーレジスタ
 */
typedef struct{
    IOREG  TOCR;
    IOREG  dummy1;
    IOREG  TSTR;
    IOREG  dummy2;
    LIOREG TCOR0;
    LIOREG TCNT0;
    HIOREG TCR0;
    LIOREG TCOR1;
    LIOREG TCNT1;
    HIOREG TCR1;
    LIOREG TCOR2;
    LIOREG TCNT2;
    HIOREG TCR2;
} tmu;

#define TMU  (*(volatile tmu *)0xfffffe90)
```

なお，メンバ dummy はアドレスをアラインするために用意した構造体のメンバである．例えば，表 1 よりタイマアウトプットコントロールレジスタ (TOCR) のレジスタサイズが 8 ビットであるため，メンバ TOCR を 0xfffffe90 から IOREG(=unsigned char) の 8 ビットで宣言している．よって，メンバ dummy1 なしでメンバ TSTR を宣言すると 0xfffffe91 に用意されることになる．しかし，次のタイマスタートレジスタ (TSTR) は，0xfffffe92 からはじまらなければならない．そのため，0xfffffe91 からはじまる IOREG 型のメンバ dummy1 を用意することにより，構造体による TMU のデータ構造を実現している．

1.3.3 hw_timer.h の関数

・void hw_timer_initialize()：タイマの起動処理
　タイマを初期化し，周期的なタイマ割込み要求を発生させる．

・void hw_timer_int_clear()：タイマ割込み要求のクリア
　タイマ割込み要求を知らせるチャネル 0 のタイマコントロールレジスタ (TCR0) のアンダーフローフラグ (UNF) をクリアする．タイマ割込みハンドラから呼ばれる．

・void hw_timer_terminate()：タイマの停止処理
　TSTR のカウンタスタート 0(STR0) を 0 にし，カウント動作を停止させる（＝タイマ停止）．
　さらに，TCR0 に 0 を書き込み割込み要求をクリアする．

・CLOCK hw_timer_get_current(void)：タイマの現在値の読みだし
　現在のカウント値を返す．vxget_tim()(./kernel/time_manage.c) で使用．

・BOOL hw_timer_fetch_interrupt(void)：タイマの割込み要求の確認
　タイマ割込み要求が発生したかどうかを返す．vxget_tim()(./kernel/time_manage.c) で使用．

1.4 システムクロックドライバ (timer.c,timer.h)

システムクロックドライバは，ハードウェアタイマを用いて周期的に割込みを発生させ，isig_tim を呼び出してカーネルにタイムティックを供給する．システムクロックドライバは，タイマの起動処理，タイマ割込みハンドラ，タイマの停止処理の三つの関数で構成される．

このドライバはタイムイベントハンドラ（周期ハンドラ，アラームハンドラ，オーバランハンドラ）およびシステム時刻管理を必要とするカーネルが使用する．

・void timer_initialize()：タイマの起動処理
　hw_timer_initialize() を呼び出し，タイマの初期化およびタイマ割込み要求を発生させる．
　システムクロックドライバのコンフィギュレーションファイル (timer.cfg) の ATT_INI
　(初期化ルーチンの追加を行う静的 API) から呼ばれる．

・void timer_handler()：タイマ割込みハンドラ
　hw_timer_int_clear() による割込み要求のクリアを行い，isig_tim を呼び出してタイムティックを供給する．
　システムクロックドライバのコンフィギュレーションファイル (timer.cfg) の DEF_INH
　(割込みハンドラの定義を行う静的 API) から呼ばれる．

・void timer_terminate()：タイマの停止処理
　hw_timer_terminate() を呼び出し，タイマの停止を行う．
　タイマの停止処理は，JSP カーネルの現バージョンではうまく組み込む方法がない．

2　シリアルドライバ

2.1　シリアルドライバ関連モジュール構成

タイマに関するモジュールの構成は図 3 に示すように，ハードウェアシリアル，システム依存シリアル I/O モジュール (hw_serial.h)，シリアルインタフェースドライバ (serial.c,serial.h) がある．

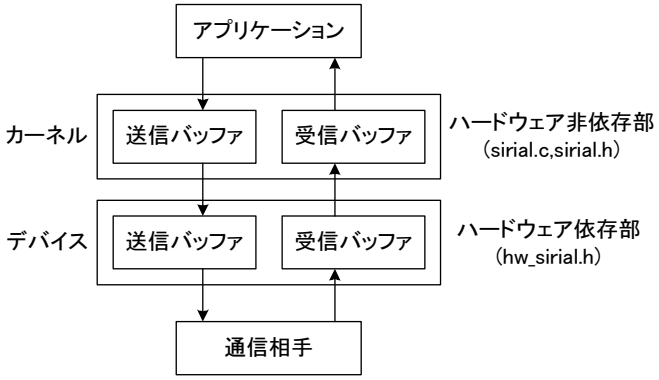


図 3　シリアルドライバの構成

2.2　SH3 のシリアルコミュニケーションインターフェース

SH3 は標準でシリアルコミュニケーションインターフェース (SCI) を持つ．また，システムによっては独自で持つシリアルコントローラがある場合がある．JSP カーネルは，SH3 を搭載したシステムとして，SH-CARD CARD-E09A，MS7709ASE01，MU-200-RSH3，DVE-SH7700 をサポートしているが，そのうち SH-CARD CARD-E09A，MS7709ASE01 はシステム独自が持つコントローラを JSP カーネルは利用する．一方，MU-200-RSH3，DVE-SH7700 は SH3 が持つ SCI をカーネルは利用している．そのため，今回説明する hw_serial.h には，DVE-SH7700 用を用いる．これにより，独自コントローラよりも資料が入手しやすい SH3 で説明がおこなえるため，明確に構造も理解していただけると思われる．

SCI には，調歩同期式通信とクロック同期式通信の 2 方式でシリアル通信をすることが可能である．

2.2.1　SCI のレジスタ構成

SCI には，次に示す内部レジスタがある．これらのレジスタにより通信方式の指定，データフォーマットの指定，ビットレートの指定，送信部／受信部の制御を行う．表 3 にレジスタ構成，表 4 にレジスタ概要を示す．

表 3　SCI レジスタ構成

レジスタ	略称	R/W	アドレス	サイズ
シリアルモードレジスタ	SCSMR	R/W	0xffffe80	8
ビットレートレジスタ	SCBRR	R/W	0xffffe82	8
シリアルコントロールレジスタ	SCSCR	R/W	0xffffe84	8
トランスミットデータレジスタ	SCTDR	R/W	0xffffe86	8
シリアルステータスレジスタ	SCSSR	R/(W)[1]	0xffffe88	8
レシーブデータレジスタ	SCRDR	R/W	0xffffe8A	8
ポート SC データレジスタ	SCPDR	R/W	0x04000136	8
ポート SC コントロールレジスタ	SCPCR	R/W	0x04000116	16
トランスミットシフトレジスタ	SCTSR	CPU からのアクセス不可		8
レシーブシフトレジスタ	SCRSR	CPU からのアクセス不可		8

1：フラグをクリアするために 0 のみ書き込むことが出来る．

表 4　SCI レジスタ概要

レジスタ	概要
シリアルモードレジスタ	SCI のシリアル通信フォーマットの設定と，ボーレートジェネレータのクロックソースを選択する
ビットレートレジスタ	SCSMR の CKS1,CKS0 ビットで選択されるボーレートジェネレータの動作クロックとあわせて，シリアル送受信のビットレートを設定する
シリアルコントロールレジスタ	SCI の送受信動作，調歩同期式モードでのシリアルクロック出力，割込み要求の許可／禁止，および送受信クロックソースの選択を行う
トランスミットデータレジスタ	シリアル送信するデータを格納する
シリアルステータスレジスタ	SCI の動作状態を示すステータスフラグと，マルチプロセッサビットを内蔵
レシーブデータレジスタ	受信したシリアルデータを格納する
ポート SC データレジスタ	SCI 端子と兼用されているポートの入出力方向とデータを制御する
ポート SC コントロールレジスタ	
トランスミットシフトレジスタ	シリアルデータを送信する
レシーブシフトレジスタ	シリアルデータを受信する

2.2.2　動作説明

Ⅰ. 動作モード

・調歩同期式モード
スタート／ストップビットによりキャラクタ単位で同期をとる方式

・クロック同期式モード
クロックに同期してシリアルデータ通信を行う方式

JSP カーネルでは，調歩同期式モードを利用しているので，調歩同期の動作を説明する．

◎送信時

(1)SCI は，シリアルステータスレジスタ (SCSSR) の TDRE フラグを監視し，0 であるとトランスミットデータレジスタ (SCTDR) にデータが書き込まれたと認識し，SCTDR からトランスミットシフトレジスタ (SCTSR) にデータを転送する．

(2)SCTDR から SCTSR へデータを転送した後に TDRE フラグを 1 にセットし，送信を開始する．
このとき，シリアルコントロールレジスタ (SCSCR) の TIE ビットが 1 にセットされていると送信データエンプティ割り込み (TXI) 要求を発生する．

シリアル送信データは，以下の順に TxD 端子から送り出される．

(a) スタートビット：1 ビットの 0 が出力される．
(b) 送信データ：8 ビット，または 7 ビットのデータが LSB から順に出力される．
(c) パリティビットまたはマルチプロセッサビット：1 ビットのパリティビット (偶数パリティ，または奇数パリティ)，または 1 ビットのマルチプロセッサビットが出力される．
なお，パリティビット，またはマルチプロセッサビットを出力しないフォーマットも選択できる．
(d) ストップビット：1 ビットまたは 2 ビットの 1(ストップビット) が出力される．
(e) マーク状態：次の送信を開始するスタートビットを送り出すまで 1 を出力し続ける．

(3)SCI は，ストップビットを送出するタイミングで TDRE フラグをチェックする．
TDRE フラグが 0 であると SCTDR から SCTSR にデータを転送し，ストップビットを送り出した後，次フレームのシリアル送信を開始する．
TDRE フラグが 1 であるとシリアルステータスレジスタ (SCSSR) の TEND フラグに 1 をセットし，ストップビットを送り出した後，1 を出力するマーク状態になる．
このとき SCSCR の TEIE ビットが 1 にセットされていると TEI 要求を発生する．

◎受信時

(1)SCI は通信回線を監視し，スタートビットの 0 を検出すると内部を同期化し，受信を開始する．

(2) 受信したデータを SCRSR の LSB から MSB の順に格納する．

(3) パリティビット，およびストップビットを受信する．

受信後，SCI は以下のチェックを行う．

(a) パリティチェック:受信データの 1 の数をチェックし，これがシリアルモードレジスタ (SCSMR) の O/E ビットで設定した偶数／奇数パリティになっているかをチェックする．
(b) ストップビットチェック：ストップビットが 1 であるかをチェックする．
ただし，2 ストップビットの場合，1 ビット目のストップビットのみをチェックする．
(c) ステータスチェック：RDRF フラグが 0 であり，受信データをレシーブシフトレジスタ (SCRSR) から SCRDR に転送できる状態であるかをチェックする．

以上のチェックがすべてパスしたとき，RDRF フラグが 1 にセットされ，SCRDR に受信データが格納される．
エラーチェックで受信エラーを発生すると表 5 のように動作する．
【注】受信エラーが発生した状態では，以後の受信動作ができない．
また，受信時に RDRF フラグが 1 にセットされないので，必ずエラーフラグを 0 にクリアする．

(4)RDRF フラグが 1 になったとき，SCSCR の RIE ビットが 1 にセットされていると受信データフル割り込み (RXI) 要求を発生する．
また，ORER，PER，FER フラグのどれかが 1 になったとき，SCSCR の RIE ビットが 1 にセットされていると受信エラー割り込み (ERI) 要求を発生する．

表 5　受信エラーと発生条件

受信エラー名	略称	発生条件	データ転送
オーバランエラー	ORER	SCSSR の RDRF フラグが 1 にセットされたまま次のデータ受信を完了したとき	SCSSR から SCRDR に受信データは転送されない
フレーミングエラー	FER	ストップビットが 0 のとき	SCRSR から SCRDR に受信データが転送される
パリティエラー	PER	SCSMR で設定した偶数／奇数パリティの設定と受信したデータが異なるとき	SCRSR から SCRDR に受信データが転送される

II. フロー制御 (XON/XOFF)

(1) 概要

フロー制御とは，あまりに速いバイトの流れにより，システムがオーバーランするのを防ぐことである．オーバーランとは，システム等の受信バッファがあふれ，十分な余裕を持って受信処理をできない時に，データの損失やその他のエラーが生じることである．フロー制御により，通信相手がデータ受信の準備ができるまでバイトの流れを止めておく．フロー制御はバイトの流れを止めたいシステムの相手側のシステムに対してフロー停止信号を送る．そのため，端末とコンピュータの両方で設定されねばならない．

(2) フロー制御の必要性

オーバーランしないためには，受信バッファがあふれることのないような送信速度で送ればよい．しかし，送信速度を下げると受信バッファに余裕があるときでもその速度で通信しなくてはならない．また，シリアル通信の場合，常にユーザーの望む通信速度で通信できるわけではない．9600bps，14400bps，19200bps のように決められた通信速度を利用しなければならないため，望む速度以上で使う必要が出てくる．そのため，受信バッファがあふれることになりかねない．そのような状況のためにフロー制御は必要となる．

(3) バッファについて

バッファは，短い間だけなら最悪の状況を処理する助けとなる．最悪の状況とは，受信ならば通信が途絶える場合および一度に処理しきれないほど速く大量にデータが流れてきた場合が考えられる．送信ならば送信量が極端に少なくなってしまった場合やもしくは通信速度以上の速さでデータを送り出そうとする場合が考えられる．バッファがあれば，データをバッファに格納し，途絶えた場合でもバッファ分だけは処理が継続できる．また，大量にデータが流れてきた場合には，後で処理するために溜めておくことができる．

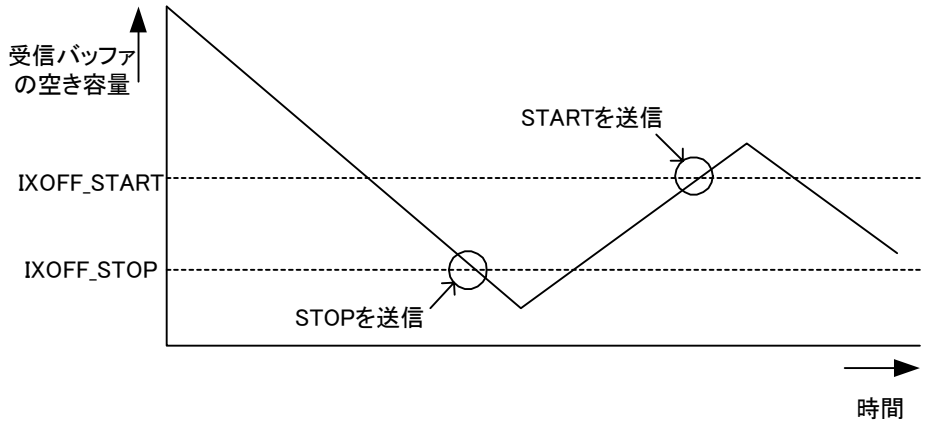


図 4　フロー制御 (XON/XOFF)

2.3 ターゲット依存シリアル I/O モジュール (hw_serial.h)

ターゲット依存シリアル I/O モジュールである hw_sirial.h は，ハードウェアシリアルの制御，参照を直接行う低レベルのインターフェースである．DVE-SH7700 用の hw_serial.h では，SH3 の SCI0 を使用する．

2.3.1 hw_serial.h におけるレジスタへの設定値

I. シリアルコントロールレジスタ

7	6	5	4	3	2	1	0
TIE	RIE	TE	RE	MPIE	TEIE	CKE1	CKE0

マクロ	マクロ値	ビット	機能
—	0x70	RIE	1:受信データフル割込み (RXI) 要求と受信エラー割込み (ERI) 要求を許可，0:RXI と ERI を禁止
		TE	1:送信動作を許可，0:送信動作を禁止
		RE	1:受信動作を許可，0:受信動作を禁止
SCSCR_RIE	0x0040	RIE	1:受信データフル割込み (RXI) 要求と受信エラー割込み (ERI) 要求を許可，0:RXI と ERI を禁止
SCSCR_TIE	0x0080	TIE	1:送信データエンプティ割込み (TXI) 要求を許可，0:TXI とを禁止

II. シリアルステータスレジスタ

7	6	5	4	3	2	1	0
TDRE	RDRF	ORER	FER	PER	TEND	MPB	MPBT

マクロ	マクロ値	ビット	機能
—	0x0060	RDRF	1:SCRDR に有効な受信データが格納されていることを表示，0:受信されていないことを表示
		ORER	1:受信時にオーバランエラーが発生したことを表示，0:受信中，または正常に受信したことを表示
SCSSR_RDRF	0x40	RDRF	1:SCRDR に有効な受信データが格納されていることを表示，0:受信されていないことを表示
SCSSR_TDRE	0x80	TDRE	1:SCTDR に有効な送信データがないことを表示，0:書き込まれていることを表示
SCSSR_TEND	0x04	TEND	1:送信を終了したことを表示，0:送信中であることを表示

III. シリアルモードレジスタ

表 6 に SCSMR の設定値とシリアル通信フォーマットの関係を示す．

7	6	5	4	3	2	1	0
C/$\bar{A}$	CHR	PE	O/$\bar{E}$	STOP	MP	CKS1	CKS0

表 6　SCSMR の設定値とシリアル送信／受信フォーマット

SCSMR の設定値					モード	SCI の送信／受信フォーマット			
ビット 7	ビット 6	ビット 2	ビット 5	ビット 3		データ長	マルチプロセッサビット	パリティビット	ストップビット長
C/A	CHR	MP	PE	STOP					
0	0	0	0	0	調歩同期式	8 ビットデータ	なし	なし	1 ビット
				1					2 ビット
			1	0				あり	1 ビット
				1					2 ビット
	1		0	0		7 ビットデータ		なし	1 ビット
				1					2 ビット
			1	0				あり	1 ビット
				1					2 ビット
	0	1	—	0	調歩同期式 (マルチプロセッサフォーマット)	8 ビットデータ	あり	なし	1 ビット
			—	1					2 ビット
	1		—	0		7 ビットデータ			1 ビット
			—	1					2 ビット
1	—	—	—	—	クロック同期式	8 ビット	なし		なし

DVE-SH7700 用の hw_serial.h では，シリアル I/O ポートの初期化の段階で 0 を設定しているので，表 6 の一番上のフォーマットで送受信が行われる．

IV．ビットレートレジスタ

マクロ	マクロ値	ビット	機能
SCI_BPS_VALUE	3 or 7	all	ビットレートを設定

SCBRR への設定値は以下の計算式で求められる．

$$N = \frac{P\phi}{64 \times 2^{2n-1} \times B} \times 10^6 - 1$$

B ：ビットレート (bit/s)
N ：ボーレートジェネレータの SCBRR の設定値 ($0 \leqq N \leqq 255$)
$P\phi$：周辺モジュール用動作周波数 (MHz)
n ：ボーレートジェネレータ入力クロック (n=0,1,2,3)
(n とクロックの関係は，表 7 参照)

表 7　n と CKS1，CKS0 の対応表

n	クロック	SCSMR の設定値	
		CKS1	CKS0
0	$P\phi$	0	0
1	$P\phi/4$	0	1
2	$P\phi/16$	1	0
3	$P\phi/64$	1	1

DVE-SH7700 の SH3 プロセッサ SH7708 は周辺モジュール用動作周波数が 16MHz, もしくは 30MHz で設定可能である．また，SCSMR の CKS1 および CKS0 ビットは共に 0 なので，表 7 に基づき $n = 0$．ビットレートは 115200bps で利用できるように設定したいので，上記の式により計算すると，16MHz のとき N=3, 30MHz のとき N=7 となる．

2.3.2　SCI のデータ構造

JSP カーネルでは，SCI のデータ構造を./config/sh3/sh3.h において，次の構造体により定義している．

```
/*
 *  シリアルコミュニケーションインターフェース (SCI)
 */
typedef struct{
    IOREG  SCSMR;
    IOREG  dummy1;
    IOREG  SCBRR;
    IOREG  dummy2;
    IOREG  SCSCR;
    IOREG  dummy3;
    IOREG  SCTDR;
    IOREG  dummy4;
    IOREG  SCSSR;
    IOREG  dummy5;
    IOREG  SCRDR;
} sci;

#define SCI (*(volatile sci *)0xfffffe80)
```

2.3.3 hw_serial.h の関数

・BOOL hw_port_initialize()：シリアル I/O ポートの初期化
　　通信方式およびビットレート設定，割込み関連の設定
・void hw_port_terminate()：シリアル I/O ポートの終了
　　シリアルコントローラの停止および割込みマスクのクリア.
・void hw_serial_handler_in()：シリアル I/O ポートの割込みハンドラ
　　受信に関する割込みハンドラのエントリをおこなう.
・void hw_serial_handler_out()：シリアル I/O ポートの割込みハンドラ
　　送信に関する割込みハンドラのエントリをおこなう.
・BOOL hw_port_getready()：受信チェック
　　文字を受信したかをチェック.
・BOOL hw_port_putready()：送信チェック
　　文字を送信できるかをチェック.
・byte hw_port_getchar()：受信文字の取り出し
　　受信した文字 1 バイトを取り出す.
・byte hw_port_putchar()：送信文字の書き込み
　　送信する文字 1 バイトを書き込む.
・void hw_port_sendstart()：送信制御関数
　　送信割り込みを許可して，送信を開始する.
・void hw_port_sendstop()：送信制御関数
　　送信割り込みの停止して，送信を停止する.

2.4 シリアルインタフェースドライバ (sirial.c,sirial.h)

システムインタフェースドライバはカーネルがシリアル I/O ポートを扱うためのドライバである.

2.4.1 シリアルポート管理ブロック構造体

```
#define SERIAL_BUFSZ 256 /* シリアルインタフェース用バッファのサイズ */

#define inc(x) (((x)+1 < SERIAL_BUFSZ) ? (x)+1 : 0)
#define INC(x) ((x) = inc(x))

typedef struct serial_port_control_block {
    BOOL     init_flag;                 /* 初期化済か？ */
    HWPORT   hwport;                    /* ハードウェア依存情報 */
    ID       in_semid;                  /* 受信バッファ管理用セマフォの ID */
    ID       out_semid;                 /* 送信バッファ管理用セマフォの ID */

    int      in_read_ptr;               /* 受信バッファ読み出しポインタ */
    int      in_write_ptr;              /* 受信バッファ書き込みポインタ */
    int      out_read_ptr;              /* 送信バッファ読み出しポインタ */
    int      out_write_ptr;             /* 送信バッファ書き込みポインタ */
    UINT     ioctl;                     /* ioctl による設定内容 */
    BOOL     send_enabled;              /* 送信をイネーブルしてあるか？ */
    BOOL     ixon_stopped;              /* STOP を受け取った状態か？ */
    BOOL     ixoff_stopped;             /* 相手に STOP を送った状態か？ */
    char     ixoff_send;                /* 相手に START/STOP を送るか？ */

    char     in_buffer[SERIAL_BUFSZ];   /* 受信バッファエリア */
    char     out_buffer[SERIAL_BUFSZ];  /* 送信バッファエリア */
} SPCB;
```

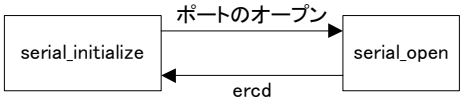
2.4.2　バッファ管理用セマフォ

バッファは通信の際に最悪の状況が発生したときの影響を軽減する．バッファには，受信したデータが受信した順番に，もしくは送信するデータが送信する順番に格納されなければならない．もし，その順番が変わると，送受信エラーや受信側の誤動作の原因を引き起こすことになりかねない．そのため，通信する一文字を取り扱うごとにセマフォを確保する．それにより，その間にその次以降に扱う文字を通信しようとしてもセマフォが確保できないため，タスクの待ち行列に入るため，通信する文字の順番が守られる．

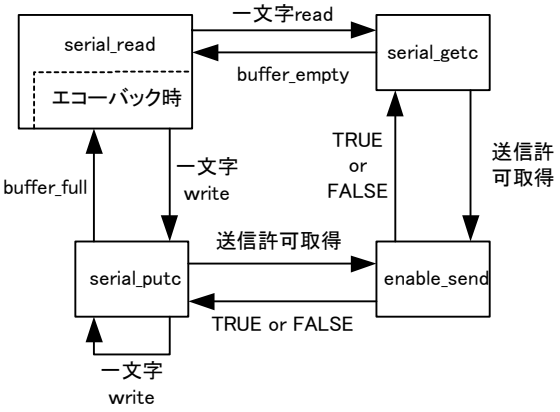
2.4.3　serial.c の関数

Ⅰ. 各関数の関係

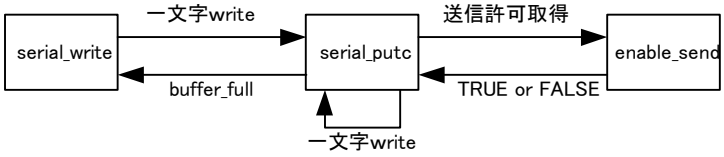
(1) ポートのオープン（初期化）



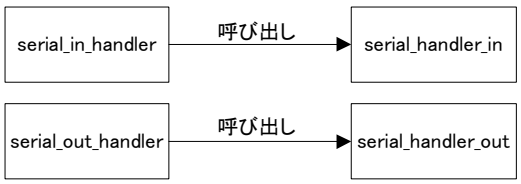
(2) 受信



(3) 送信



(4) 割込み



II. 各関数概要

・void serial_initialize(VP_INT portid)：シリアルインターフェースドライバの起動
　シリアルインターフェースドライバの起動．シリアルポートのオープンを呼び出す．

・ER serial_open(ID portid)：ポートのオープン
　portid で示されるシリアルポートを初期化し，オープンする．

・ER serial_close(ID portid, BOOL flush)：ポートのクローズ
　portid で示されるシリアルポートのクローズ処理を行う．flush が 0 でない場合，送信バッファが空になるまでループする．

・BOOL enable_send(SPCB *spcb, char c)：ユーティリティルーチン
　送信（割込み）を許可する．デバイスが送信可能であれば，1 バイト送信する．

・BOOL serial_getc(SPCB *spcb, char c)：シリアルポートからの受信
　1 バイト受信する．

・ER_UINT serial_read(ID portid, char *buf, UINT len)：シリアルポートからの受信
　portid で示されるシリアルポートから，len バイトの文字列を読み込み，buf からの領域に入れる．戻り値には読み込んだ文字数を返す．

・BOOL serial_putc(SPCB *spcb, char c)：シリアルポートへの送信
　1 バイト送信する．

・ER_UINT serial_write(ID portid, char *buf, UINT len)：シリアルポートへの送信
　portid で示されるシリアルポートに，buf からの len バイトの文字列を書き出す．戻り値には書き出した文字数を返す．

・ER serial_ioctl(ID portid, UINT ioctl)：シリアルポートの制御
　portid で示されるシリアルポートの制御情報を設定／解除する．設定できる値は serial.h にマクロで定義されており，ビット毎に論理和をとって利用する．マクロを表 8 に示す．

表 8　ioctl 設定マクロ

マクロ	値	概要
IOCTL_NULL	0	指定なし
IOCTL_ECHO	0x0001	入力した文字をエコーバック
IOCTL_CRLF	0x0010	LF(line feed) を出力する際に CR(carriage return) を付加
IOCTL_RAW	0x0100	1 文字単位で入力
IOCTL_CANONICAL	0x0200	LF が来るまで入力
IOCTL_IXON	0x1000	出力に対して XON/XOFF 制御
IOCTL_IXANY	0x2000	どのような文字でも出力開始
IOCTL_IXOFF	0x4000	入力に対して XON/XOFF 制御

◎使用例

```
serial_ioctl(0, (IOCTL_CRLF |IOCTL_RAW |IOCTL_IXON |IOCTL_IXOFF));
```

・void serial_handler_in(ID port)：シリアルポート割込みサービスルーチン
　受信用割込みサービスルーチン．

・void serial_handler_out(ID port)：シリアルポート割込みサービスルーチン
　送信用割込みサービスルーチン．

・void serial_in_handler()：割込みハンドラ
　受信用割込みハンドラ．
　シリアルインタフェースドライバのコンフィギュレーションファイル (serial.cfg) の DEF_INH (割込みハンドラの定義を行う静的 API) から呼ばれる．

・void serial_out_handler()：割込みハンドラ
　送信用割込みハンドラ．
　シリアルインタフェースドライバのコンフィギュレーションファイル (serial.cfg) の DEF_INH (割込みハンドラの定義を行う静的 API) から呼ばれる．

・void serial_handler()：割込みハンドラ
　割込みハンドラ．他のターゲット用．