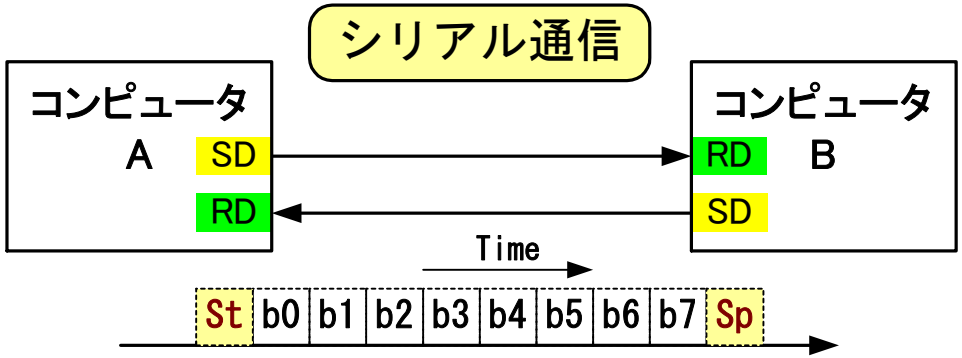
シリアル通信、調歩同期とは

コンピュータ世界の通信とは、コンピュータ同士、あるいは、コンピュータと いろんな周辺機器を接続してデータ通信を行わせる事です。

接続の形態で、シリアル通信とパラレル通信があります。

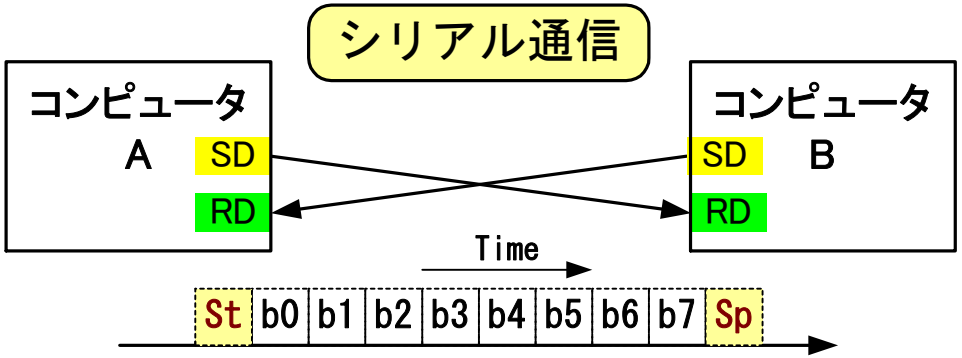
シリアル通信とは、ビットシリアルといって 1つの伝送路に、データを 1ビットずつ載せて順次一定速度で 伝送するやりかたです。 通常、送信線と、受信線の 2つの伝送路を組みにして扱う場合が多いです。 電線が少なくて済むため遠距離に伝送する場合にも 使われます。

パラレル通信は、8ビット分のデータ線、8本を持っており、一度に並列8ビット(1バイト)単位でデータを 転送出来ます。 比較的近距離で、データを高速に転送する用途で使用されていました。 パラレル通信は まだありますが、最近 あまり見なくなってきました。



シリアル通信とは、1本の信号線に、時分割で1bit単位のデータを載せて転送するやり方です。

上の例では、調歩同期のビット並び例を示しています、調歩同期では、送り側と 受け側で 同じ転送速度で、データを 送受信します。 調歩同期では、1byteのデータの 始まりが分かるように先頭に St スタートビットが、付きます。 同様に1byteのデータの終りに Sp ストップビットが付きます。 あいだの 8bitデータは、b0から順次送信されます。 この方式のメリットは、I2Cや SPIのような シリアルクロック線が、必要ない事です。



上記の図で、2台のコンピュータAと Bの 間の2本の矢印の付いた線ですが、矢印の向きはデータ転送される方向を示します。 線の両端には、SD、RDの名前を付けてますが、 センドデータ 送信線、レシーブデータ 受信線の意味です。

この際、注意する必要があるのは、自分のSD信号は、相手の RD信号になるという事です。相手の SD信号は、自分の RD信号になります。
この関係を明確化するため、この図では、矢印の信号線を、クロスした形で描きました。

シリアル通信で、調歩同期の場合、伝送速度ボーレイトの設定が、決められています。
110bps、150bps、300bps、600bps、1200bps、2400bps、4800bps、9600bps、14400bps、19200bps、38400bps、57600bps、115200bps、230400bps、460800bps、921600bps これらはテラタームの設定値を参照しました。

RX220では、ボーレイトの分周値の問題で安定して使えるのは 57600bpsまでと思います。HOCO 32MHzの高速クロックの場合 115200bpsが 何とか使えました。クロック分周比の誤差が -3.99%に なっているので 1%しか余裕が無い上に、HOCOクロックの 誤差が 1%以内のようで際どい誤差で動いている事が考えられます。
使用する環境によっては、115200bpsは、動作が不安定になる恐れがあります。
設定するボーレイトと、誤差を計算した Excelの表を、次のページに示します。

外部水晶 20MHzメインクロック 使用の ボーレイト換算表

RX220の SCI1の ボーレイトに関わる設定
SCI1.BRR = BRR値(8 ～ 255)
SCI1.SNR.BIT.CKS = 0 ～ 2 (これは 0 = 1/1、1 = 1/4、2 = 1/16)

BPS が ボーレイトです。
PCLKが、SCI1に入力されるクロック周波数です。

目的のボーレイトを作り出すための分周値 (PCLKを 1/1で使用)

No.	PCLK[MHz]	BPS	STATUS	BRR理想値	BRR値	誤差[%]	分周値	オシロ確認用 9bit データ長 [us]
1	20	300	OK	129.22	129	0.17	1/16	30,000.0
2	20	600	OK	64.11	64	0.17	1/16	15,000.0
3	20	1200	OK	129.22	129	0.17	1/4	7,500.0
4	20	2400	OK	64.10	64	0.16	1/4	3,750.0
5	20	4800	OK	129.21	129	0.16	1/1	1,875.0
6	20	9600	OK	64.10	64	0.16	1/1	937.5
7	20	19200	OK	31.55	32	-1.40	1/1	468.8
8	20	31250	OK	19.00	19	0.00	1/1	288.0
9	20	38400	OK	15.28	15	1.84	1/1	234.4
10	20	57600	OK	9.85	10	-1.49	1/1	156.3
11	20	115200	NG	4.43	4	10.63	1/1	78.1

内部HOCOクロック使用の ボーレイト換算表

オシロ観測用 9bit Data長 [us]は、設定したボーレイトが正しいか確認するため、0x00をデータとして出力すると、スタートビットから、b7までの 9bitが、全て Lowレベルになるので、それをオシロで、パルス幅を 計って調べるものです。

No.	PCLK[MHz]	BPS	STATUS	BRR理想値	BRR値	誤差[%]	分周値	オシロ観測用 9bit Data長
1	32	300	OK	207.33	207	0.16	1/16	30,000.0
2	32	600	OK	103.17	103	0.16	1/16	15,000.0
3	32	1200	OK	207.33	207	0.16	1/4	7,500.0
4	32	2400	OK	103.17	103	0.16	1/4	3,750.0
5	32	4800	OK	207.33	207	0.16	1/1	1,875.0
6	32	9600	OK	103.17	103	0.16	1/1	937.5
7	32	19200	OK	51.08	51	0.16	1/1	468.8
8	32	31250	OK	31.00	31	0.00	1/1	288.0
9	32	38400	OK	25.04	25	0.17	1/1	234.4
10	32	57600	OK	16.36	16	2.26	1/1	156.3
11	32	115200	OK	7.68	8	-3.99	1/1	78.1

誤差は、±5% 未満である事

今回作成した シリアル通信 プログラム／ヘッダファイル

シリアル通信に関わる、関数の一覧です。

```
//  Source File : RX22_iocs_sci1.c  プロトタイプ宣言
// ------------------------------------------------------------------
_UBYTE setup_sci1( _UBYTE bsel, _UBYTE pb );     // SCI1 オープン処理
void send_sci1( _UBYTE c );                       //  １文字 送信
shortget_sci1_recv_len( void );                  // SCI1 受信バッファ内の 格納文字列長 取得
shortrecv_sci1( void );                           //  １文字 受信
void prin_sci1( char *txt );                     // Null終端の文字列を SCI1から送信
void print_sci1( char *txt );                    // 文字列の送信 Cr, Lf 付き
void send_block_sci1( Uchar buf[], short len );  // バイナリ固定長データの送信
shortrecv_text_sci1( char txt[], short maxlen ); // 文字列を受信、Crコード検出で 打ち切る
shortrecv_block_sci1( Uchar buf[], short len );  // 受信した固定長バイナリデータの取り出し
```

シリアル通信処理
初期化(SCI1)

初期化の設定例を
いくつか示します。

① ボーレイト: 9600bps
パリティ: 無し
```
setup_sci1( 5, 0 );
```

② ボーレイト: 38400bps
パリティ: 偶数
```
setup_sci1( 8, 1 );
```

③ ボーレイト: 57600bps
パリティ: 奇数
```
setup_sci1( 9, 2 );
```

```
//********************************************************************
//**  シリアル通信処理 初期化処理                                     **
//** ---------------------------------------------------------------- **
//**  bsel : ボーレイト                                               **
//**   0 =    300 b/s                                                 **
//**   1 =    600 b/s                                                 **
//**   2 =   1200 b/s                                                 **
//**   3 =   2400 b/s                                                 **
//**   4 =   4800 b/s                                                 **
//**   5 =   9600 b/s                                                 **
//**   6 =  19200 b/s                                                 **
//**   7 =  31250 b/s                                                 **
//**   8 =  38400 b/s                                                 **
//**   9 =  57600 b/s                                                 **
//**  10 = 115200 b/s ( 32MHzの時のみ何とか使用可 )                   **
//**                  ( 20MHzでは 115200 b/sは使えない )               **
//** ---------------------------------------------------------------- **
//**  pb : パリティビット 0 = 無し 、1 = 偶数 、2 = 奇数              **
//** ---------------------------------------------------------------- **
//**  データ長: 8 bit 、ストップbit長: 1 bit   固定                   **
//** ---------------------------------------------------------------- **
//**  関数値: = 0 :  使用不能                                         **
//**    <> 0 :  使用可能 ( CPUクロック値を返す )                      **
//********************************************************************
_UBYTE  setup_sci1( _UBYTE bsel, _UBYTE pb )
```

1文字受信、1文字送信処理

シリアル通信処理で、初期化以外で、一番基本的な、1文字送信関数、1文字受信関数（ 正確には、受信リングバッファから1文字取り出し ）、それと受信リングバッファに、何バイトデータが、格納されているかの、受信バイト数 取り出し関数が、基本となる関数です。

ちなみに 1文字送信関数 send_sci1() は 直接 周辺回路SCI1をアクセスして送信しています。

それに対して、受信処理はいつ相手からデータが、送られてくるか分からないので SCI1の受信処理は、SCI1の割り込み機能を利用して256バイトの リングバッファに書き込んでいます。 それとは非同期に recv_sci1() にて 受信データを取り出す事が出来ます。

```
//**************************************
//**   1文字送信                        **
//** ---------------------------------- **
//**   c :  送信データ 0 ～ 255         **
//**************************************
void  send_sci1( _UBYTE c )

//**************************************
//**   1文字 受信                       **
//** ---------------------------------- **
//**   関数値 :  0 ～ 255 = 正常データ  **
//**             -1 = 受信データ無し    **
//**************************************
short recv_sci1( void )

//**************************************************
//**   SCI1 受信バッファ内の 格納文字列長 取得    **
//** ------------------------------------------- **
//**   関数値 :  0 ～ 255   受信バイト数          **
//**************************************************
short get_sci1_recv_len( void )
```

シンプルなデータ折り返し処理

今回作成したプログラム内のデータ折り返し関数から、LEDのタイマーによる点滅を外した物です。 受信データが -1は 受信データが無い事を意味します。

```
//**********************************
//**  ループテスト２              **
//**********************************
void  test_loop_2( void )
{
      short dt;

      while( 1 )
      {
            dt = recv_sci1();     // １文字受信
            if( dt == -1 )    continue;

            send_sci1( dt );      // １文字送信
      }
}
```

リングバッファとは

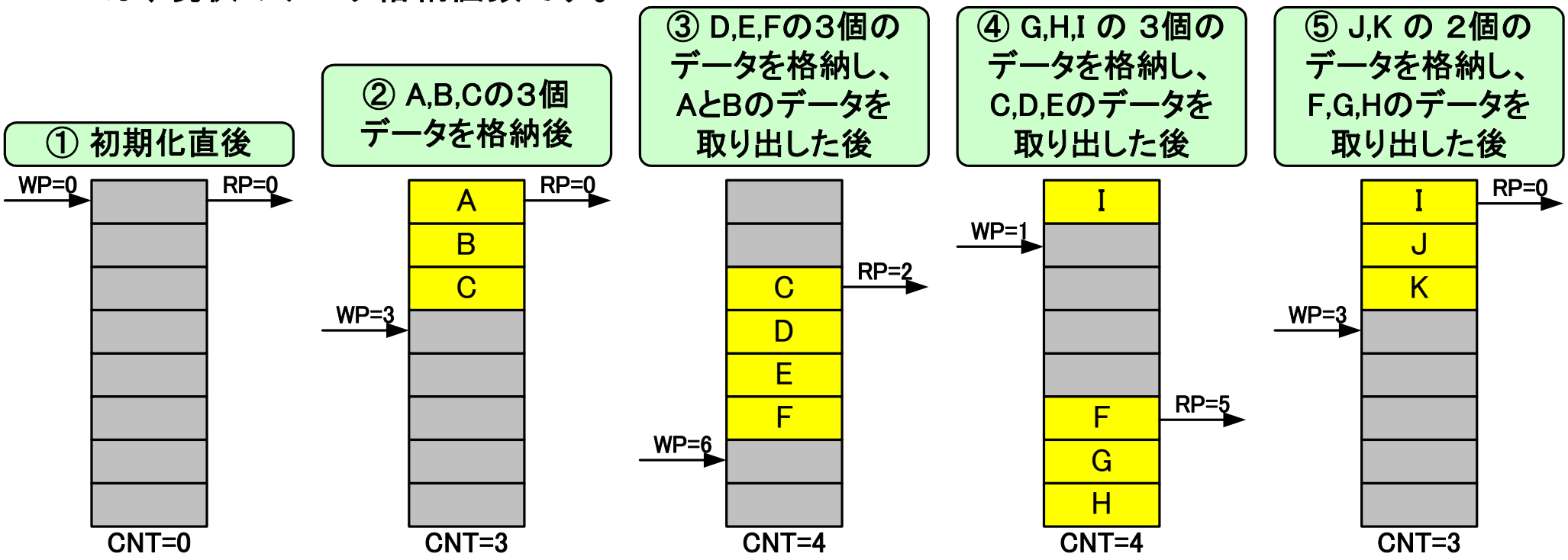
リングバッファは、FIFOバッファです。
一番最初に書き込んだデータが、一番最初に取り出せます。

実際にメモリが リング状になってる訳ではないですが、一番最後のアドレスまでデータを格納したら、また先頭に戻ってデータを書き続けるバッファです。 そんな事したら先頭のデータが壊れると思う方もいるかもしれませんが、書き込みポインタを、追いかけるように読み出しポインタもデータを読み出すので、データを読み出してしまったら、そのデータの格納されていた場所は、空きエリアとみなされるのです。
ちょっと、文字の説明だけでは難しいですね。
次のページに図で示します。

リングバッファの動作

格納データ数8個の 小さなリングバッファで説明します。 WPは 次の書き込み位置ポインタで、 RPは 次の読み出し位置ポインタで、CNTは、現状のデータ格納個数です。

リングバッファのイメージが、多少なりと理解出来ましたでしょうか。？



リングバッファ アクセス時の注意

今回のリングバッファ アクセスの関数は、表に出してませんが、書き込み wr_rbuf関数と 読み出し rd_rbuf関数の2つが、あります。 wr_rbuf関数は、SCI1割り込み処理内にて呼び出されます。 rd_rbuf関数は、アイドルループ内で呼び出されます。
rd_rbuf関数内で disable_irq関数 割り込み禁止と、enable_irq関数 割り込み許可を 呼び出しています。

052の動画で、チラッと話をしましたが、disable_irq関数と enable_irq関数に挟まれている区間が、クリティカルパスになります。 どういうことかというと、アイドルループ内で recv_sci1関数を呼び出すと、その中で rd_rbuf関数を呼び出します。rd_rbuf関数実行中で Rbufに関わる読み出し 書き込みをしている最中に wr_buf関数が、割り込んで来ると、Rbufのパラメータが 壊される恐れがあるのです。 割り込み処理が割り込んで来ないように、一時的にクリティカルセクションの間だけ、割り込み禁止にしているという事です。

```
void   wr_rbuf( Uchar c )
{
    if( Rbuf.ct == 255 )  return;   // 満杯であれば格納しない

    Rbuf.rng[Rbuf.wp].dt = c;  // 1byte 受信データ格納
    Rbuf.wp++;                  // 書き込みポインタ更新
    Rbuf.ct++;                  // 格納個数 更新
}

short  rd_rbuf( void )
{
    short   dt;

    if( Rbuf.ct == 0 )
       return  -1;  // データが無ければ -1を返す

    disable_irq();   // 割り込みを 一時的に禁止する
    dt = Rbuf.rng[Rbuf.rp].dt;  // 1byte 受信データ読み出し
    Rbuf.rp++;        // 書き込みポインタ更新
    Rbuf.ct--;        // 読み出し個数 更新
    enable_irq();    //  割り込みを許可する

       return dt;
}
```

ややこしいので、エラーフラグ処理を外してます。