11 シリアル通信の利用

11.1 シリアル通信 (RS-232C)

パソコン上で動かすプログラムでは、演算結果をディスプレイに表示することができます。C 言語の標準関数である printf などを用いると、出力はかなりの自由度で制御できるのです。

しかし、マイコンの場合にはこのような表示器が備わっていない場合が少なくありません。LED やブザーだけでは演算の結果を充分に表示することができない場合も多いでしょう。最終的なシステムにおいては必要でなくても、開発途中にデバッグを行う際には、演算結果を表示できると非常に便利です。

マイコンシステムにおいては、プログラムの開発の際にパソコンを用います。書き込みにおいてはRS232Cなどの通信を用いて書き込みを行います。そこで、この通信を用いて演算の結果をパソコン (ディスプレイ) に表示してみようというのがここでの目標です。

RS-232C は多くのマイコンで周辺機能として組み込まれていますので、これを利用することができればとても便利なわけです。機能が豊富なのでレジスタの設定は大変ですが、充分に理解して使いこなせるようになってください。

11.1.1 シリアル通信とは

1 本の信号線を用いて 1 ビットずつ転送するのをシリアル通信といいます。複数の信号線を用いて一度に複数のビットデータを転送するものをパラレル通信と言います。シリアル通信には、

- RS-232
- RS-422
- RS-485
- USB(Universal Serial Bus)
- イーサネット

などがあります。

RS-232(Recommended Standard 232) は、パソコンとモデムなどを接続するシリアル通信方式のインターフェース（界面）規格案の一つです。RS-232C という名前が一般的に使われます。データ出入り口はポートとも呼ばれるため、シリアルポートと呼ばれることもあります。正式な企画は EIA-232-D/E として発行されています。

TIA(Telecommunications Industry Association) が受け継いで「ANSI/TIA/EIA-232-F-1997」が現在の正式規格名となっていますが、この新しい規格に準拠しているものは多くないようです。

最近ではシリアルポートを持たないパソコンのほうが多く、変換器を用いて USB ポートにつなぐのが一般的になってきました。

11.1.2 端子

RS-232C の端子は、D サブ 9 ピンもしくは D サブ 25 ピンです。最近では D サブ 9 ピンが多く使われています。

D サブ (D-sub) は、D-subminiature（ディー・サブミニアチュア) の略です。2 もしくは 3 列に並んだピン (ピンコンタクト) またはソケット (ソケットコンタクト) がアルファベットの D 形の金属シールドに囲まれているためにこの名前がついているようです。ピンをオス、ソケットをメスと言います。

現在最も一般的に利用されている 9 ピン DSUB コネクタのピン配置は以下のようになります。

ピン番号	記号	入出力方向	意味
1	DCD	IN	キャリア検出
2	RxD	IN	受信データ
3	TxD	OUT	送信データ
4	DTR	OUT	データ端末レディー
5	GND		グランド
6	DSR	IN	データ・セット・レディ
7	RTS	OUT	送信要求
8	CTS	IN	受信可能
9	RI	IN	被呼表示

RxD はデータの受信、TxD はデータの送信、GND はグランドです。その他は、通信の確認などに使われますが、マイコンの通信では、RxD、TxD、GND の 3 本だけで行うことも多いようです。

以下では、この 3 線による通信に限定して考えてみましょう。

11.1.3 RS232C(9 ピン DSUB コネクタ) メス

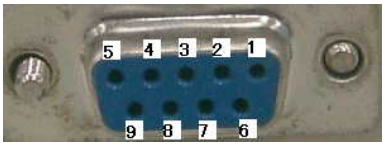


図 11.1　RS232C(9 ピン DSUB コネクタ) メス

RS232C(9 ピン DSUB コネクタ) メスのピン配置は、図 11.1 のようになっています。

11.1.4 RS232C(9 ピン DSUB コネクタ) オス

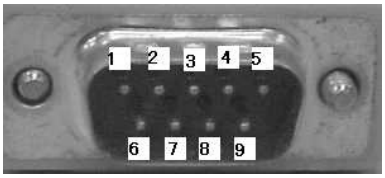


図 11.2 RS232C(9 ピン DSUB コネクタ) オス

RS232C(9 ピン DSUB コネクタ) オスのピン配置は、図 11.2 のようになっています。

RS232C(9 ピン DSUB コネクタ) メスのピン番号 (図 11.1) と割り振りが異なりますが、同じ番号同士がつながるようになっているわけです。

11.1.5 ストレートケーブル (パソコン-周辺機器)

パソコンと周辺機器をつなぐのには、通常ストレートケーブルを用います。周辺機器の側がオス、パソコンの側がメスになっているものが多いかと思います。このケーブルは 2 ピン同士、3 ピン同士をつなぎます。送信、受信同士をつなぐのは奇妙な感じですが、周辺機器の中でクロスさせているのでこのようなつなぎ方になります。オス-メスのストレートケーブルは比較的入手しやすいかと思います。

また、3 線で通信を行う場合は、周辺機器の側で、 RTS と CTS などは直接結線することになります。

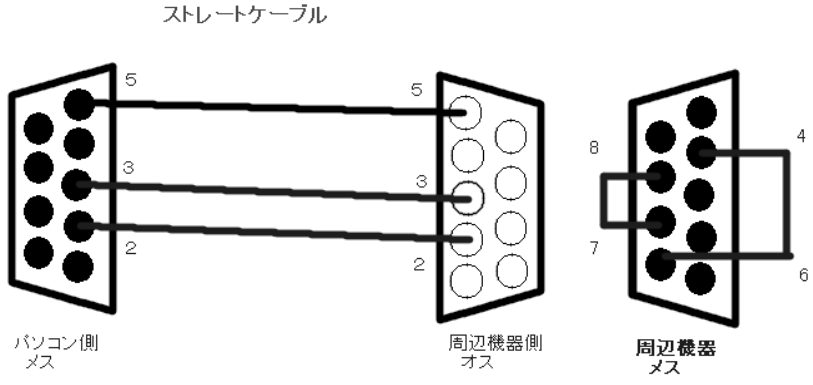


図 11.3 ストレートケーブルの接続例

11.1.6 クロスケーブル (パソコン-パソコン)

パソコン同士をつなぐ場合には、送信データのピンを受信データのピンにつなぎます。ストレートケーブルと言います。

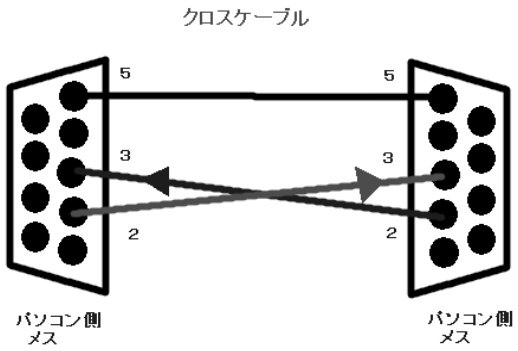


図 11.4　クロスケーブルの接続例

11.1.7 半二重通信と全二重通信

全二重通信は、送信と受信それぞれ伝送路が別にあり、送受信を同時に行うことができます。半二重通信は、1 つの伝送路を用いて送信と受信を切り替えながら通信します。そのため、送信と受信を同時に行うことはできません。

H8/3694F では全二重通信が可能です。

また、送信レジスタ、受信レジスタともに二つあり (ダブルバッファ)、連続送信動作、連続受信動作が可能です。

11.1.8 調歩同期式とクロック同期式

RS232C 通信では、受信側で正確にデータを受信するために、送信側がどのような速度でデータを送信しているか把握しておかなければなりません。

H8/3694F ではでは、通信方式として「調歩同期式」と「クロック同期式」を利用することができます。全二重調歩同期式通信が一般的に利用されているようです。

調歩同期式通信は、非同期の通信で、送信側受信側の間でクロックのやり取りは行いません。互いに自分のクロックを用いてデータの送受信を行います。このため伝送速度が一致していないと、正常な通信ができません。最初にお互いの通信条件設定を合わせる必要があるわけです。

クロック同期式では、どちらかの発生するクロックに同期してデータの送受信を行う方式です。送信側から 1 ビット毎に付加された同期信号をもとに通信を行います。データの転送効率は良いですが、通信手順が複雑になるデメリットがあります。

前述のように、今回は一般的に利用されている、全二重調歩同期式通信を用いることにします。

11.1.9 通信速度

通信速度は bps(Bits Per Second) で表します。1 秒間に 1 ビットのデータを転送できる速度が、1bps です。19200bps、38400bps、57600bps など標準的な通信速度のなかから選択することが多いようです。

11.1.10 ストップビット長・スタートビット長

ストップビット長とは、データの終端を表すビットの長さです。通常は、1 ビット、2 ビットから選択します。

データの始まりを表すスタートビット長は、1 ビットに固定されています (通常は設定する必要がありません)。

11.1.11 データビット長

データビット長では、ひとつのデータが何ビットで構成されているかを指定します。7 ビットもしくは 8 ビットから選択します。

11.1.12 パリティチェック

パリティチェックは、データの誤り検出を行う機能です。偶数パリティ、奇数パリティ、パリティチェック無しから選択します。

パリティチェックは送信側でデータに 1 または 0 のパリティビットを付加します。偶数パリティならば 1 が偶数になるように、奇数パリティならば 1 が奇数になるように付加するわけです。

受信側で 1 の数を数え、データが正常かどうかの判断を行います。

11.1.13 データの形式

データは 1 バイトごとに以下のように送信されます (通常使われるデータ 8 ビット、パリティーなし、ストップビット 1 の場合)。

- スタートビット (1 ビット)
- データ (8 ビット)
- ストップビット (1 ビット)

11.1.14 信号の電圧レベル

RS232C 通信の信号の電圧レベルは以下のように規格されています。

表 11.1　RS232C 通信の信号の電圧レベル

項目	スペース	マーク
出力信号	0(On)	1(Off)
出力電圧	+5V〜+15V	-5〜-15V
入力条件	+3V 以上	-3V 以下

この規格から分かるように、RS232C 通信では出力信号 0 は+3V 以上です。一方 H8/3694F の出力信号 0 は 0V です。同様に RS232C 通信では出力信号 1 は-3V 以下ですが、H8/3694F の出力信号 1 は 5V です。

このように電圧が異なるために、電圧のレベル変換を行う必要があります。通常この目的のために、専用の IC が利用されます。今回利用する書込み・拡張 I/O ボード MB-RS10 では、この変換のために ADM232AAN という IC が利用されています (図 11.5)。

11.1.15 回路図

以下にレベル変換 IC ADM232AAN の回路を示しておきます。この回路は、書込み・拡張 I/O ボード MB-RS10 に実装されているものですので、作る必要はありません。

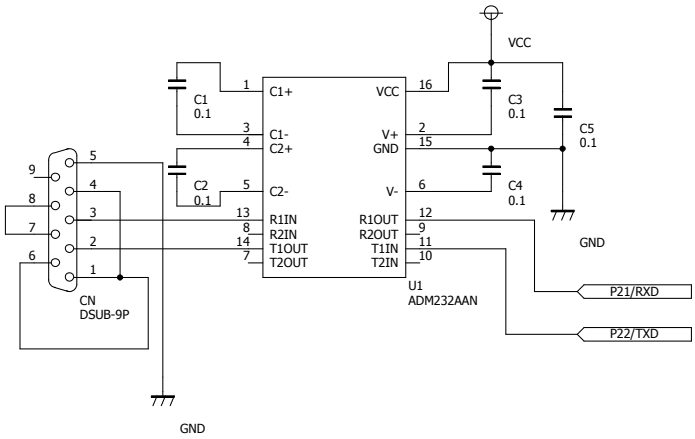


図 11.5　シリアル通信の回路

プログラムする際には、送信が P22/TXD、受信が P21/RXD であることだけ理解していれば充分です。

11.1.16 関連レジスタ

H8/3694F にはシリアルコミュニケーションインタフェース 3(SCI3) が一つあります。

プログラムに必要な、SCI3 関連レジスタは以下のとおりです。

- レシーブデータレジスタ (RDR)

- トランスミットデータレジスタ (TDR)
- シリアルモードレジスタ (SMR)
- シリアルコントロールレジスタ 3(SCR3)
- シリアルステータスレジスタ (SSR)
- ビットレートレジスタ (BRR)

上記のレジスタは次のアドレスに割り振られています。

表 11.2 SCI3 レジスタのアドレス

レジスタ名	アドレス
シリアルモードレジスタ (SMR)	H'FFA8
ビットレートレジスタ (BRR)	H'FFA9
シリアルコントロールレジスタ 3(SCR3)	H'FFAA
トランスミットデータレジスタ (TDR)	H'FFAB
シリアルステータスレジスタ (SSR)	H'FFAC
レシーブデータレジスタ (RDR)	H'FFAD

なお、これらのレジスタのほかに、トランスミットシフトレジスタ (TSR) とレシーブシフトレジスタ (SRS) が存在します。これらは上記のダブルバッファの機能を実現しているレジスタです。プログラムからは操作することができません。

図 11.6 のように、送信データはトランスミットデータレジスタ (TDR) に書き込みますが、その値は一旦トランスミットシフトレジスタ (TSR) に転送され、そののちに TXD 端子へと送りだされます (図 11.6 参照)。

同様に RXD 端子から受け取ったデータは、一旦レシーブシフトレジスタ (SRS) に保存され、転送が可能になれば、レシーブデータレジスタ (RDR) にデータが移されます (図 11.6 参照)。

このようにして、2 段階で送受信を行うことで、スムーズな動作を実現しているわけです。

なお、受信バッファはこれだけでは不十分であるため、後ほど自作することになります (P.273)。

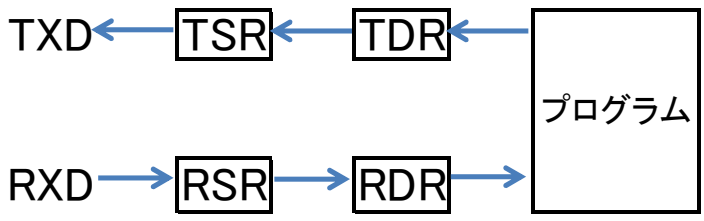


図 11.6 ダブルバッファ

レシーブデータレジスタ (RDR)

RDR は受信データを格納するための 8 ビットのレジスタです。RDR のリードは SSR の RDRF が 1 にセットされていることを確認して 1 回だけ行います。RDR は CPU からライトできません。RDR の初期値は H'00 です。

トランスミットデータレジスタ (TDR)

TDR は送信データを格納するための 8 ビットのレジスタです。シリアル送信を確実に行うため、TDR への送信データのライトは必ず SSR の TDRE が 1 にセットされていることを確認して 1 回だけ行います。TDR の初期値は H'FF です。

シリアルモードレジスタ (SMR)

SMR はシリアルデータ通信フォーマットと内蔵ボーレートジェネレータのクロックソースを選択するためのレジスタです。

表 11.3 シリアルモードレジスタ (SMR)

ビット	ビット名	初期値	R/W	説 明
7	COM	0	R/W	コミュニケーションモード 0：調歩同期式モードで動作します。 1：クロック同期式モードで動作します。
6	CHR	0	R/W	キャラクタレングス（調歩同期式モードのみ有効） 0：データ長 8 ビットのフォーマットで送受信します。 1：データ長 7 ビットのフォーマットで送受信します。
5	PE	0	R/W	パリティイネーブル（調歩同期式モードのみ有効） このビットが 1 のとき、送信時はパリティビットを付加し、 受信時はパリティチェックを行います。
4	PM	0	R/W	パリティモード（調歩同期式モードで PE ＝ 1 のときのみ有効） 0：偶数パリティで送受信します。 1：奇数パリティで送受信します。
3	STOP	0	R/W	ストップビットレングス（調歩同期式モードのみ有効） 送信時のストップビットの長さを選択します。 0 : 1 ストップビット 1 : 2 ストップビット 受信時はこのビットの設定値にかかわらず ストップビットの 1 ビット目のみチェックし、 2 ビット目が 0 の場合は 次の送信キャラクタのスタートビットとみなします。
2	MP	0	R/W	マルチプロセッサモード このビットが 1 のときマルチプロセッサ通信機能が イネーブル (有効) になります。 PE、PM ビットの設定値は無効になります。 クロック同期式モードではこのビットは 0 に設定ます。
1 0	CKS1 CKS0	0 0	R/W R/W	クロックセレクト 1〜0 内蔵ボーレートジェネレータのクロックソースを選択します。 00：φクロック（n=0） 01：φ/4 クロック（n=1） 10：φ/16 クロック（n=2） 11：φ/64 クロック（n=3） このビットの設定値とボーレートの関係については P.266「ビットレートレジスタ（BRR）」を参照してください n は設定値の 10 進表示で P.266「ビットレートレジスタ（BRR）」中の n の値を表します。

シリアルコントロールレジスタ 3(SCR3)

SCR3 は送受信動作と割り込み制御、送受信クロックソースの選択を行うためのレジスタです。

表 11.4　シリアルコントロールレジスタ 3(SCR3)

ビット	ビット名	初期値	R/W	説 明
7	TIE	0	R/W	トランスミットインタラプトイネーブル このビットを 1 にセットすると、TXI 割り込み要求がイネーブルになります。
6	RIE	0	R/W	レシーブインタラプトイネーブル このビットを 1 にセットすると、 RXI および ERI 割り込み要求がイネーブルになります。
5	TE	0	R/W	トランスミットイネーブル このビットが 1 のとき送信動作が可能になります。
4	RE	0	R/W	レシーブイネーブル このビットが 1 のとき受信動作が可能になります。
3	MPIE	0	R/W	マルチプロセッサインタラプトイネーブル （調歩同期式モードで SMR の MP ＝ 1 のとき有効）
2	TEIE	0	R/W	トランスミットエンドインタラプトイネーブル このビットを 1 にセットすると TEI 割り込み要求がイネーブルになります。
1 0	CKE1 CKE0	0 0	R/W R/W	クロックイネーブル 1〜0 クロックソースを選択します。 調歩同期式の場合 00：内部ボーレートジェネレータ 01：内部ボーレートジェネレータ （SCK3 端子からビットレートと同じ周波数のクロックを出力します） 10：外部クロック （SCK3 端子からビットレートの 16 倍の周波数のクロックを入力してください。） 11：リザーブ

シリアルステータスレジスタ (SSR)

SSR は SCI3 のステータスフラグなどで構成されます。TDRE、RDRF、OER、PER、FER はクリアのみ可能です。

表 11.5 シリアルステータスレジスタ (SSR)

ビット	ビット名	初期値	R/W	説 明
7	TDRE	1	R/W	トランスミットデータレジスタエンプティ TDR 内の送信データの有無を表示します。 ［セット条件］ ・SCR3 の TE が 0 のとき ・TDR から TSR にデータが転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき ・TDR へ送信データをライトしたとき
6	RDRF	0	R/W	レシーブデータレジスタフル RDR 内の受信データの有無を表示します。 ［セット条件］ ・受信が正常終了し、RSR から RDR へ受信データが転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき ・RDR のデータをリードしたとき
5	OER	0	R/W	オーバランエラー ［セット条件］ ・受信中にオーバランエラーが発生したとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
4	FER	0	R/W	フレーミングエラー ［セット条件］ ・受信中にフレーミングエラーが発生したとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
3	PER	0	R/W	パリティエラー ［セット条件］ ・受信中にパリティエラーが発生したとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
2	TEND	1	R	トランスミットエンド ［セット条件］ ・SCR3 の TE が 0 のとき ・送信キャラクタの最後尾ビットの送信時、TDRE が 1 のとき ［クリア条件］ ・TDRE = 1 の状態をリードした後、TDRE フラグに 0 をライトしたとき ・TDR へ送信データをライトしたとき
1	MPBR	0	R	マルチプロセッサビットレシーブ 受信キャラクタ中のマルチプロセッサビットを格納します。 SCR3 の RE = 0 のときは変化しません
0	MPBT	0	R/W	マルチプロセッサビットトランスファ 送信キャラクタに付加するマルチプロセッサビットの値を指定します。

ビットレートレジスタ (BRR)

BRR はビットレート (通信速度) を設定する 8 ビットのレジスタです。BRR の初期値は H'FF です。動作周波数とビットレートの組み合わせに対する BRR の設定値Nは以下の計算式で求まります。

〔調歩同期式モード〕

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

B：ビットレート（bit/s）

N：ボーレートジェネレータの BRR の設定値（0 ≦ N ≦ 255）

Φ：動作周波数（MHz）

n：SMR の CKS1、CKS0 の設定値（0 ≦ n ≦ 3）(表 11.3 参照)

ポートモードレジスタ 1（PMR1）

PMR1 はポート 1 とポート 2 の端子の機能を切り替えるために使います。シリアル通信に関連するのは TXD ビットで、P22/TXD 端子の機能を選択します。

すでに P.170 の表 7.3 に掲載しましたが、再度載せておきます。

表 11.6　ポートモードレジスタ 1(PMR1)

ビット	ビット名	初期値	R/W	説 明
7	IRQ3	0	R/W	P17/$\overline{IRQ3}$/TRGV 端子の機能を選択します。 0：汎用入出力ポート 1：$\overline{IRQ3}$ および TRGV 入力端子
6	IRQ2	0	R/W	P16/$\overline{IRQ2}$ の機能を選択します。 0：汎用入出力ポート 1：$\overline{IRQ2}$ 入力端子
5	IRQ1	0	R/W	P15/$\overline{IRQ1}$ の機能を選択します。 0：汎用入出力ポート 1：$\overline{IRQ1}$ 入力端子
4	IRQ0	0	R/W	P14/$\overline{IRQ0}$ の機能を選択します。 0：汎用入出力ポート 1：$\overline{IRQ0}$ 入力端子
3 2	- -	1 1	- -	リザーブビットです。リードすると常に 1 が読み出されます。
1	TXD	0	R/W	P22/TXD 端子の機能を選択します。 0：汎用入出力ポート 1：TXD 出力端子
0	TMOW	0	R/W	P10/TMOW 端子の機能を選択します。 0：汎用入出力ポート 1：TMOW 出力端子

割り込み要求

SCI3 が生成する割り込み要求には、送信終了、送信データエンプティ、受信データフルおよび受信エラー（オーバランエラー、フレーミングエラー、パリティエラー）の計 6 種類があります。

表 11.7 に各割り込み要求の内容を示します。

表 11.7 SCI3 の割り込み要求

割り込み要求	略称	割り込み要因
受信データフル	RXI	SSR の RDRF のセット
送信データエンプティ	TXI	SSR の TDRE のセット
送信終了	TEI	SSR の TEND のセット
受信エラー	ERI	SSR の OER、FER、PER のセット

受信データフルの割込み要因である、SSR の RDRF のセットは、SRS から RDR に受信データが転送されたときにセットされます (P.265 表 11.5、P.261 の図 11.6 参照)。

SCI3 のレジスタ定義 (iodefine.h の一部を抜粋)

HEW のレジスタ定義ファイルのうち、関連する部分を載せておきます。

SCI3 のレジスタ定義 (iodefine.h の一部を抜粋)

```
struct st_sci3 {                                        /* struct SCI3  */
               union {                                  /* SMR          */
                     unsigned char BYTE;                /*  Byte Access */
                     struct {                           /*  Bit  Access */
                            unsigned char COM :1;       /*    COM       */
                            unsigned char CHR :1;       /*    CHR       */
                            unsigned char PE  :1;       /*    PE        */
                            unsigned char PM  :1;       /*    PM        */
                            unsigned char STOP:1;       /*    STOP      */
                            unsigned char MP  :1;       /*    MP        */
                            unsigned char CKS :2;       /*    CKS       */
                            }      BIT;                 /*              */
                     }          SMR;                    /*              */
               unsigned char    BRR;                    /* BRR          */
               union {                                  /* SCR3         */
                     unsigned char BYTE;                /*  Byte Access */
                     struct {                           /*  Bit  Access */
                            unsigned char TIE :1;       /*    TIE       */
                            unsigned char RIE :1;       /*    RIE       */
                            unsigned char TE  :1;       /*    TE        */
                            unsigned char RE  :1;       /*    RE        */
                            unsigned char MPIE:1;       /*    MPIE      */
                            unsigned char TEIE:1;       /*    TEIE      */
                            unsigned char CKE :2;       /*    CKE       */
                            }      BIT;                 /*              */
                     }          SCR3;                   /*              */
               unsigned char    TDR;                    /* TDR          */
               union {                                  /* SSR          */
                     unsigned char BYTE;                /*  Byte Access */
                     struct {                           /*  Bit  Access */
                            unsigned char TDRE:1;       /*    TDRE      */
                            unsigned char RDRF:1;       /*    RDRF      */
                            unsigned char OER :1;       /*    OER       */
                            unsigned char FER :1;       /*    FER       */
                            unsigned char PER :1;       /*    PER       */
                            unsigned char TEND:1;       /*    TEND      */
                            unsigned char MPBR:1;       /*    MPBR      */
                            unsigned char MPBT:1;       /*    MPBT      */
                            }      BIT;                 /*              */
                     }          SSR;                    /*              */
               unsigned char    RDR;                    /* RDR          */
};                                                      /*              */

......

#define SCI3    (*(volatile struct st_sci3  *)0xFFA8)   /* SCI3  Address*/
```

11.1.17 基本的なプログラム (割込みなし)

まずは最も基本的な、割り込みなしの送受信プログラムを作ってみましょう。

08_SCI01.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :08_SCI01.c                                             */
 4  /*  DESCRIPTION :シリアル通信 (基本形)                                  */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  RxD P21                                                             */
 8  /*  TxD P22                                                             */
 9  /*                                                                      */
10  /************************************************************************/
11
12  #include "iodefine.h"
13
14  typedef enum{
15    br19200=32,
16    br38400=15,
17    br57600=10
18  }BaudRate;
19
20  /*********************************
21     シリアルポートを初期化 :Sci1Init
22  *********************************/
23  void Sci1Init(BaudRate b)
24  {
25      SCI3.SCR3.BYTE = 0x00; /* TE、RE クリア   CKE 初期化 */
26      SCI3.SMR.BYTE  = 0x00; /* 調歩同期、8 ビット、パリティなし、ストップ 1 ビット n=0 */
27      SCI3.BRR  = b;
28      IO.PMR1.BIT.TXD = 1; /* TXD ポートイネーブル */
29      SCI3.SCR3.BYTE = 0x30; /* TE RE */
30  }
31
32  /*******************************
33     1 バイト送信関数 :Sci1Write
34  *******************************/
35  void Sci1Write(char c)
36  {
37    while(!SCI3.SSR.BIT.TDRE){ /* 送信データ書き込み可能になるまで待つ */
38      ;
39    }
40    SCI3.TDR=c;           /* 送信データを格納 */
41  }
42
43  /*******************************
44     1 バイト受信関数 :Sci1Read
45  *******************************/
46  char Sci1Read(void)
47  {
48    char read_char;
49
50    while(!SCI3.SSR.BIT.RDRF){ /* データを受信するまで待つ */
51      ;
52    }
53    read_char=SCI3.RDR;       /* データを変数へ */
54    SCI3.SSR.BIT.RDRF=0;
55
56    return(read_char);
57  }
58
59  /*******************************
60     main 関数 :
61  *******************************/
62
63  void main(void)
64  {
65
66    Sci1Init(br19200);
67    while(1){
68      Sci1Write('>');
69      Sci1Write(Sci1Read());
70      Sci1Write('\r');
71      Sci1Write('\n');
72    }
73  }
```

End Of List

CHECK ☞

- (hterm などの) ターミナルソフトを起動する。
- ターミナルソフトの設定は 19200bps、8 ビット、パリティなし、ストップビット 1 にする。
- マイコンのプログラムを実行する。

実行結果

図 11.7　実行結果

ターミナルソフトに入力した文字が表示される (エコーバック)。

プログラム解説 (08_SCI01.c)

```
14 typedef enum{
15   br19200=32,
16   br38400=15,
17   br57600=10
18 }BaudRate;
```

シリアル通信のボーレート (ビットレート、通信速度) を設定するための列挙体です。

シリアル通信の速度は、いくつか選択可能なのですが、通信速度と設定する数値の対応は複雑で、一目見ただけでは簡単には分かりません。コメントを付けてわかるようにしておく方法もあるのですが、いろいろと思考錯誤しているうちに、コメントというのは嘘になっていくことが往々にしてあります。

このような場合に、名前と定数を対応させておくのに便利なのが、列挙型です。ここの例では、typedef を使って BaudRate という型を新たに定義しています。

BaudRate b; b=br19200;

というふうに利用します。ここでは、BaudRate 型の変数 b を定義し、br19200 を代入しています。このとき、b の値は 32 になります (15 行目で定義)。

BaudRate 型の変数に代入できるのは、型の宣言でおこなった br19200,br38400,br57600 に限られます。各値は 32、15、10 となります。

```
23  void Sci1Init(BaudRate b)
24  {
25      SCI3.SCR3.BYTE = 0x00;  /* TE、RE クリア CKE 初期化 */
26      SCI3.SMR.BYTE  = 0x00;  /* 調歩同期、8 ビット、パリティなし、ストップ 1 ビット n=0 */
27      SCI3.BRR  = b;
28      IO.PMR1.BIT.TXD = 1;    /* TXD ポートイネーブル */
29      SCI3.SCR3.BYTE = 0x30;  /* TE RE */
30  }
```

23 行目から 30 行目はシリアルポートを初期化する関数です。ここで定義した関数は、後ほど別ファイルに分割し、仕様をまとめておきたいと思います。

25 行目は P.264 の表 11.4 から、送信、受信を停止し、クロックは内部クロックを使う設定をしていることが分かります。

26 行目は P.263 の表 11.3 から、調歩同期式、8 ビット、パリティなし、分周なし (Φ) であることが分かります。

27 行目では、ボーレートを設定しています。ボーレートは引数で与えられ、Sci1Init(br19200) のように指定します。14 行目から 18 行目を説明した時に述べたように、このとき SCI3.BRR には 32 が設定されます。

28 行目では、P.266 の表 11.6 から分かるように、P22/TXD 端子を TXD 出力端子に設定しています。

29 行目では、25 行目で停止していた送受信を利用可能にしています。

```
35  void Sci1Write(char c)
36  {
37    while(!SCI3.SSR.BIT.TDRE){ /* 送信データ書き込み可能になるまで待つ */
38      ;
39    }
40    SCI3.TDR=c;           /* 送信データを格納 */
41  }
```

1 文字を送信する関数です。割り込みは利用していません。

P.265 の表 11.5 から、シリアルステータスレジスタ (SSR) の TDRE ビット (ビット 7) が 1 の時にトランスミットデータレジスタ (TDR) が空であることが分かります。そのため、37 行目から 39 行目にかけて、このフラグが 0 である間は while ループを抜けないようにしています。送信が可能になったらループを抜けます。

40 行目では、関数 Sci1Write に渡された文字をトランスミットデータレジスタ (TDR) に書き込んでいます。これによってデータが送信されます。

```
46  char Sci1Read(void)
```

```
47  {
48    char read_char;
49
50    while(!SCI3.SSR.BIT.RDRF){ /* データを受信するまで待つ */
51      ;
52    }
53    read_char=SCI3.RDR;       /* データを変数へ */
54    SCI3.SSR.BIT.RDRF=0;
55
56    return(read_char);
57  }
```

46 行目から 57 行目までは 1 文字を受信する関数です。関数を実行すると、受信した 1 文字を戻り値として返します。

P.265 の表 11.5 から、シリアルステータスレジスタ (SSR) の RDRF ビット (ビット 6) が 1 の時にレシーブデータレジスタ (RDR) にデータがあることが分かります。そのため、50 行目から 52 行目にかけて、このフラグが 0 である間は while ループを抜けないようにしています。データが受信されたらループを抜けます。

53 行目では、受信したデータを変数 read_char に代入しています。

54 行目では、次のデータを受信するために、RDRF ビットを 0 にクリアしています。

56 行目では、受信したデータを関数の戻り値として返しています。

```
63  void main(void)
64  {
65
66    Sci1Init(br19200);
67    while(1){
68      Sci1Write('>');
69      Sci1Write(Sci1Read());
70      Sci1Write('\r');
71      Sci1Write('\n');
72    }
73  }
```

main 関数です。

今まで説明した関数を用いてエコーバックのプログラムを作りました。

66 行目で、関数 Sci1Init に引数 br19200 を渡して実行しています。シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定されます。

68 行目ではプロンプトとして「>」を表示しています。

69 行目では、関数 Sci1Read を実行して、受信した 1 文字を、関数 Sci1Write に渡すことで、受信した 1 文字を返しています。

このプログラムは、unsigned char 型の変数を使うことで、2 行で書くこともできます。

unsigned char 型の変数を tmp とすると、

tmp=Sci1Read();

Sci1Write(tmp);

のようになります。

70 行目と 71 行目は復帰と改行を送信しています。改行してコマンドを行の先頭に戻します。

67 行目から 72 行目を無限ループさせていますので、エコーバックを無限に繰り返すことになります。

✍ 課題 11.1.1 (提出) 入力された文字とアスキーコードを+1 した文字を表示する

ターミナルソフトから入力された文字とそのアスキーコードを+1 した文字を表示するプログラムを作ってください。アルファベットなら次の文字 (A を入力したら B、a を入力したら b)、数字なら一つ大きな値 (0 を入力したら 1) が表示されることになります。

シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。

プロジェクト：e08_SCI01

11.1.18 バッファ

バッファ(buffer) は、処理が早い CPU と処理が遅い外部機器の間でデータをやり取りする際に、その時間差を埋めるためにデータを一時的にためておく記憶装置や記憶領域のことです。

P.269 の「基本的なプログラム (割込みなし)」では、パソコンからデータが送られてくるまでレジスタのチェックをして待っていました。しかし、この間は他の処理ができません。

このような場合には割り込みを使うと、データが送られてきたら必要な処理を実行することができます。受信割り込みを使うことで、問題は解決するわけです。

しかし、データを貯めておくところがない場合、受け取ったデータを処理する前に次のデータが入ってきたらどうでしょう。新しいデータを受け取ってしまうと、前のデータは上書きされ消えてしまいます。

このような場合に、バッファがあるとそのバッファの分だけデータを貯めておくことができるわけです。

ここでは配列を用いてバッファを実現してみましょう。

11.1.19 1 文字送受信 (割込みあり)

前回と同じエコーバックのプログラムを受信割込みを用いて作ってみましょう。今回は初期化、受信、送信などを関数化したうえに、関数のプロトタイプ宣言などを sci.h に、関数本体を sci.c に記述し、lib というフォルダに保存しておくことにします。

まずは関数の仕様を決めましょう。

今回の関数の仕様は以下の通りです。

■ファイル (sci.c)、ヘッダファイル (sci.h) ■

```
-------------------------------------------------------
void Sci1Init(BaudRate b)
形式 :
      #include "sci.h"
       void Sci1Init(BaudRate b);
機能 : シリアルポートの初期化を行う関数。
シリアルポートを用いる場合には、必ず実行する。
引数 : BaudRate b(ボーレートを指定、br19200、br38400、br57600)
返却値 : 無し

-------------------------------------------------------
void Rx1BuffClear(void)
形式 :
      #include "sci.h"
       void Rx1BuffClear(void);
機能 : 受信バッファクリア関数 (割り込み有り)。
受信バッファをクリアする関数。
引数 : 無し
返却値 : 無し

-------------------------------------------------------
int Sci1Write(char c)
形式 :
      #include "sci.h"
       int Sci1Write(char c);
機能 : 1 文字送信関数。
       0x200000 回送信を行って送信できない場合には関数を抜ける。
引数 : char c(送信する文字)
返却値 : 送信成功なら 0、失敗なら-1 を返す。

-------------------------------------------------------
char Sci1Read(void)
形式 :
      #include "sci.h"
       char Sci1Read(void);
機能 : 1 文字受信関数 (割り込み、バッファ有り)。
引数 : 無し
返却値 : 受信した 1 文字を返す。

-------------------------------------------------------
```

sci.c を利用する方法は、第 6 章 (P.151) を参考にしてください。

以下に手順をまとめておきます。

- 新しいプロジェクトを作る。
- フォルダ lib の下に sci.h,sci.c を作る。
- プロジェクトの中の iodefine.h は削る。
- プロジェクトに sci.c を追加。
- インクルードファイルディレクトリを設定する。

なお、受信割込みを使うために、intprg.c の INT_SCI3 をコメントアウトしておいてください。ここで宣言されている関数は、sci.c に記述しました。

intprg.c の INT_SCI3 をコメントアウト

```
// vector 23 SCI3
//__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
```

08_SCI02.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :08_SCI02.c                                             */
/*  DESCRIPTION :シリアル通信 (1 文字送受信、割込みあり)                */
/*  CPU TYPE    :H8/3694F                                               */
/*                                                                      */
/*  RxD P21                                                             */
/*  TxD P22                                                             */
/*                                                                      */
/************************************************************************/

#include "sci.h"

/*******************************
   main 関数 :
*******************************/
void main(void)
{

  Sci1Init(br19200);
  while(1){
    Sci1Write('>');
    Sci1Write(Sci1Read());
    Sci1Write('\r');
    Sci1Write('\n');
  }
}
```

End Of List

sci.h

```
#ifndef _SCI_H_
#define _SCI_H_

/* レジスタ定義 */
#define TXD1F SCI3.SSR.BIT.TDRE //送信データフラグ
#define RXD1F SCI3.SSR.BIT.RDRF //受信データフラグ
#define TD1D SCI3.TDR           //送信データ
#define RD1D SCI3.RDR           //受信データ

#define SCI_RX_BUFF_SIZE 32
#define SCI_RX_BUFF_MASK (SCI_RX_BUFF_SIZE-1)

typedef enum{
  br19200=32,
  br38400=15,
  br57600=10
}BaudRate;

typedef struct{
  unsigned short start_index;
  unsigned short data_count;
  char data_buff[SCI_RX_BUFF_SIZE];
}SciRxBuffer;

void Sci1Init(BaudRate b);
int Sci1Write(char c);
char Sci1Read(void);

void Rx1BuffClear(void);

#endif
```

End Of List

sci.c

```
#include <machine.h>
#include "iodefine.h"
#include "sci.h"
```

```
 4
 5  #define CHECK_LOOP 0x200000
 6  #define CR 0x0D
 7  #define LF 0x0A
 8
 9  static SciRxBuffer rx1_buff;
10
11  /*********************************
12     シリアルポートを初期化 :Sci1Init
13  *********************************/
14  void Sci1Init(BaudRate b)
15  {
16    set_imask_ccr(1);   /* 割込み不可 */
17    SCI3.SCR3.BYTE = 0x00;    /* TE、RE クリア  CKE 初期化 */
18    SCI3.SMR.BYTE  = 0x00;    /* 調歩同期、8 ビット、パリティなし、ストップ 1 ビット n=0 */
19    SCI3.BRR  = b;
20    IO.PMR1.BIT.TXD = 1;  /* TXD ポートイネーブル */
21    SCI3.SCR3.BYTE = 0x70;    /* TE RE 受信割込み */
22    set_imask_ccr(0);   /* 割込み許可 */
23  }
24
25  /*******************************
26     1 バイト送信関数 :Sci1Write
27  *******************************/
28  int Sci1Write(char c)
29  {
30    unsigned long i;
31    unsigned short write_flag=0;    /* 送信データ書き込みフラグ */
32
33    for(i=0; i<CHECK_LOOP; i++){
34      if(TXD1F){ /* 送信データ書き込み可能になるまで待つ */
35        write_flag=1;
36        break;
37      }
38    }
39    if(write_flag){
40      TD1D=c;           /* 送信データを格納 */
41      return(0);
42    }
43    return(-1);
44  }
45
46  /*******************************
47     1 バイト受信関数 (割り込み有り)
48  *******************************/
49  __interrupt(vect=23) void INT_SCI3(void)
50  {
51    char msg;
52    int index;
53
54    if(RXD1F){
55      msg=RD1D;
56      if(rx1_buff.data_count < SCI_RX_BUFF_SIZE){
57        index=(rx1_buff.start_index+rx1_buff.data_count) & SCI_RX_BUFF_MASK;
58        rx1_buff.data_buff[index]=msg;
59        rx1_buff.data_count++;
60      }
61    }
62  }
63
64  int Sci1Rx1(char *msg)
65  {
66    if(rx1_buff.data_count > 0){
67      *msg=rx1_buff.data_buff[rx1_buff.start_index];
68      rx1_buff.start_index=(rx1_buff.start_index+1) & SCI_RX_BUFF_MASK;
69      rx1_buff.data_count--;
70      return(0);
71    }
72    *msg=0;
73    return(-1);
74  }
75
76  /***********************************************
77  *    1 文字受信 : Sci1Read
78  ***********************************************/
79  char Sci1Read(void)
80  {
81    char c;
82    while(Sci1Rx1(&c)){
83      ;
84    }
85    return(c);
86  }
87
88  /*************************************
89     受信バッファクリア関数 :Rx1BuffClear
90  *************************************/
91  void Rx1BuffClear(void)
92  {
93    rx1_buff.start_index=0;
94    rx1_buff.data_count=0;
95  }
```

End Of List

実行結果

図 11.8 実行結果

ターミナルソフトに入力した文字が表示される (エコーバック)。前のサンプルと動作は同じですが、今回は割込み受信を行っています。

プログラム解説 (08_SCI02.c)

```
12  #include "sci.h"
```

シリアル通信の関数は sci.c に記述し、プロトタイプ宣言などを記述するヘッダファイルは sci.h としました。ここでは、シリアル通信の関数などを使うために、sci.h を読み込んでいます。

```
20    Sci1Init(br19200);
21    while(1){
22      Sci1Write('>');
23      Sci1Write(Sci1Read());
24      Sci1Write('\r');
25      Sci1Write('\n');
26    }
```

P.272 で説明した main 関数と同じものです。

プロンプト「>」を表示し、受け取った 1 文字を返します。その後、復帰と改行を送信します。この動作を無限に繰り返します。

プログラム解説 (sci.h)

```
 1  #ifndef _SCI_H_
 2  #define _SCI_H_

   ......

31  #endif
```

P.126 の「プログラム解説 (led.h)」で説明したように、2 回目以降は読み込まれないようにするための設定です。

```
5  #define TXD1F SCI3.SSR.BIT.TDRE //送信データフラグ
6  #define RXD1F SCI3.SSR.BIT.RDRF //受信データフラグ
7  #define TD1D SCI3.TDR           //送信データ
8  #define RD1D SCI3.RDR           //受信データ
```

P.162 の時と同様に、移植性を考慮してレジスタに別名を付けました。

```
10  #define SCI_RX_BUFF_SIZE 32
11  #define SCI_RX_BUFF_MASK (SCI_RX_BUFF_SIZE-1)
```

10 行目はバッファのサイズです。32 バイト分を確保することにしました。

11 行目は 5 ビット分マスクするための値です。

SCI_RX_BUFF_SIZE-1=31 ですから、2 進数で表すと 0x11111 です。この値と論理積 (AND) を取ることにより、0 から 31 までの値に制限することができます。

詳細は sci.c のところで説明します。

```
13  typedef enum{
14    br19200=32,
15    br38400=15,
16    br57600=10
17  }BaudRate;
```

P.270 で説明した列挙体です。説明はそちらを参照してください。

```
19  typedef struct{
20    unsigned short start_index;
21    unsigned short data_count;
22    char data_buff[SCI_RX_BUFF_SIZE];
23  }SciRxBuffer;
```

19 行目から 23 行目までは、バッファ用の構造体の定義です。構造体の名前は 23 行目にある SciRxBuffer です。

20 行目のメンバ start_index は、データの始まりを表わす配列の要素番号です。start_index=10 の場合、データの先頭は data_buff[10] ということになります。

21 行目のメンバ data_count は読みだされていないデータの数です。data_count=5 の場合、読みだされていないデータが 5 個 (5 バイト) あるということです。start_index=10,data_count=5 の場合、data_buff[10] から data_buff[14] までに読みだされていないデータが入っているということになります。

データがない状態は start_index=0,data_count=0 です。バッファをクリアする関数 Rx1BuffClear では、この操作を行います。

22 行目は受信データを格納する配列です。32 バイト分確保してあります。

```
25  void Sci1Init(BaudRate b);
26  int Sci1Write(char c);
27  char Sci1Read(void);
28
29  void Rx1BuffClear(void);
```

25 行目から 29 行目までは関数のプロトタイプ宣言です。

他から利用する関数だけを宣言しています。

プログラム解説 (sci.c)

```
5  #define CHECK_LOOP 0x200000
6  #define CR 0x0D
7  #define LF 0x0A
```

5 行目は送信時の最大待ち回数です。

6 行目は carriage return(CR、復帰) のアスキーコードを CR と再定義しています。CR は行の先頭に戻す操作です。

7 行目は line feed(LF、狭義の改行) のアスキーコードを CR と再定義しています。ここでいう LF は次の行に行くことで、先頭には戻らず一行下に移動する動作を言います。

CR と LF で改行して行頭に戻る動作を表します。

今回のプログラムでは、CR と LF は使いませんでした。

```
9  static SciRxBuffer rx1_buff;
```

受信バッファの構造体変数の宣言です。

rx1_buff には、メンバ start_index 、data_count、data_buff があります。

```
1  #include <machine.h>
……
16    set_imask_ccr(1);   /* 割込み不可 */
……
23    set_imask_ccr(0);   /* 割込み許可 */
```

割込みのための設定です。

set_imask_ccr については、P.222 を確認してください。

```
14  void Sci1Init(BaudRate b)
15  {
16    set_imask_ccr(1);   /* 割込み不可 */
17    SCI3.SCR3.BYTE = 0x00;    /* TE、RE クリア   CKE 初期化 */
18    SCI3.SMR.BYTE  = 0x00;    /* 調歩同期、8 ビット、パリティなし、ストップ 1 ビット n=0 */
19    SCI3.BRR  = b;
20    IO.PMR1.BIT.TXD = 1;  /* TXD ポートイネーブル */
21    SCI3.SCR3.BYTE = 0x70;    /* TE RE 受信割込み */
22    set_imask_ccr(0);   /* 割込み許可 */
23  }
```

関数 Sci1Init はシリアルポートの初期化関数です。ほとんど P.271 の関数 Sci1Init と同じですが、今回は割込み受信を設定しています。

割込み要求については、P.267 の表 11.7 を参照してください。ここでは、21 行目で、RXI および ERI 割込み要求可能に設定しています。

```
29  int Sci1Write(char c)
30  {
31    unsigned long i;
32    unsigned short write_flag=0;    /* 送信データ書き込みフラグ */
33
34    for(i=0; i<CHECK_LOOP; i++){
35      if(TXD1F){ /* 送信データ書き込み可能になるまで待つ */
36        write_flag=1;
37        break;
38      }
39    }
40    if(write_flag){
41      TD1D=c;          /* 送信データを格納 */
42      return(0);
43    }
44    return(-1);
45  }
```

前回のサンプルでは、1 文字送信関数 Sci1Write は送信可能になるまで無限に待つようにプログラムしましたが、今回は 0x200000(2097152) 回送信を失敗したら関数を終了するという仕様にしました。実際には無限に待っても動作が固まることはありませんが、こちらのほうが信頼性が高いと思います。

32 行目の変数 write_flag は送信を行ったら 1 に設定し、送信できなかった場合には 0 のままになるようにしました。

34 行目から 39 行目までが送信可能性のチェックです。CHECK_LOOP 回 (0x200000) 送信できなければ write_flag は 0 のままループを抜けます。

35 行目から 38 行目までが送信可能な場合です。TXD1F は sci.h で受信データフラグ SCI3.SSR.BIT.RDRF の別名として定義されています。このフラグが 1 の場合には送信可能ですので、36 行目で write_flag を 1 に設定し、37 行目の break でループを抜けます (40 行目へ飛ぶ)。

40 行目から 43 行目までは、write_flag が 1 の時の動作です。送信可能ですので、トランスミットデータレジスタ (SCI3.TDR) にデータを入れます。TD1D は sci.h で SCI3.TDR の別名として定義されています。データを送信した場合には、関数は 0 を返します。

44 行目はデータを送信できなかった場合です。関数は-1 を返します。

```
49  __interrupt(vect=23) void INT_SCI3(void)
50  {
51    char msg;
52    int index;
53
54    if(RXD1F){
55      msg=RD1D;
56      if(rx1_buff.data_count < SCI_RX_BUFF_SIZE){
57        index=(rx1_buff.start_index+rx1_buff.data_count) & SCI_RX_BUFF_MASK;
58        rx1_buff.data_buff[index]=msg;
59        rx1_buff.data_count++;
60      }
61    }
62  }
```

49 行目から 62 行目までは割込み受信関数です。今の場合、RXI および ERI 割込みのみを可能にしていることに注意してください。

51 行目の変数 msg は受信データを格納する変数、52 行目の変数 index はデータの最後の場所を格納する変数です。

54 行目でレシーブデータフル (RDRF) ビットを調べて、受信データがある場合には 56 行目から 60 行目までを実行します。

55 行目ではレシーブデータレジスタ (RDR) の値を変数 msg に代入しています。

56 行目では (読みだしていない) 受信データ数 (rx1_buff.data_count) がバッファサイズ (SCI_RX_BUFF_SIZE) より小さいかのチェックです。小さかったら 57 行目から 59 行目の処理 (バッファへのデータ書き込み) を行います。

57 行目ではデータの最後を計算しています。データの先頭 (rx1_buff.start_index) に (読みだしていない) 受信データ数 (rx1_buff.data_count) を足せばデータの最後尾が分かります。ただし、配列の要素番号が 32 になったら要素番号を 0 に戻します。この操作は 32 で割った余りを求めることと同等ですから、SCI_RX_BUFF_MAS(31:0x11111) と論理積 (AND) を取ればよいことが分かります。

58 行目では、送信データをバッファの最後に追加しています。

59 行目では、データ数が 1 増えたので、(読みだしていない) 受信データ数 (rx1_buff.data_count) に 1 を足しています。

```
64  int Sci1Rx1(char *msg)
65  {
66    if(rx1_buff.data_count > 0){
67      *msg=rx1_buff.data_buff[rx1_buff.start_index];
68      rx1_buff.start_index=(rx1_buff.start_index+1) & SCI_RX_BUFF_MASK;
69      rx1_buff.data_count--;
70      return(0);
71    }
72    *msg=0;
73    return(-1);
74  }
```

64 行目から 74 行目は 1 文字読み出し関数です。この関数は直接外部から呼び出さず、これを用いた関数 Sci1Read を使うようにしています。

関数 Sci1Read はデータが受信されるまで待つようにプログラムしました。

関数 Sci1Rx1 の引数が char 型のポインターであることに注意してください。

66 行目では、バッファにデータがあるかをチェックしています。変数 rx1_buff.data_count の値が (読みだしていない) 受信データ数なので、0 より大きな値であればデータがあるということです。このとき 71 行目までを実行します。

67 行目では先頭のデータを読みだしています。rx1_buff.start_index がデータの先頭位置ですから、rx1_buff.data_buff[rx1_buff.start_index] がバッファにあるデータの先頭ということになります。この値を変数 msg に代入しています。

68 行目は、データの先頭位置を一つずらしています。配列の最後の要素数 31 まで行った場合には、値を 0 に戻す必要があります。そのために SCI_RX_BUFF_MAS(31:0x11111) と論理積 (AND) を取っています。

69 行目では、データ数 (rx1_buff.data_count) を 1 減らしています。

70 行目は、データの呼び出しに成功したら 0 を返すという仕様にしたので、0 を返しています。

72 行目、73 行目はバッファにデータがなかった場合です。msg の中身を 0 にして、-1 を返しています。

```
79  char Sci1Read(void)
80  {
81    char c;
82    while(Sci1Rx1(&c)){
83      ;
84    }
85    return(c);
86  }
```

79 行目から 86 行目までは一文字受信関数です。すでに上で説明したように、関数 Sci1Rx1 は読みだしに成功しても失敗しても 1 回で実行を終わりますが、関数 Sci1Read はデータが受信されるまで待つようにしています。回数を決めて、読みこめない時には関数を抜けるという仕様にしても良いのですが、受信を何秒も待つ場面も珍しくありません。マイコンの処理速度は速いので、これを実装しようとすると何千万回も何億回も待つようにしなくてはなりません。実用上無限回にしてしまったほうが便利なようなので、ここではそのようにしました。

関数 Sci1Rx1 は引数を受け取らないことに注意してください。

82 行目から 84 行目までがデータを読みこむ操作です。関数 Sci1Rx1(&c) を実行してその結果を while ループの条件にしています。関数 Sci1Rx1 はデータを読みだしたらその値は引数として渡した変数 c に代入され、値 0 が返されます。このとき while ループを抜けることになります。読みだせなかったときには、c には 0 が代入され、値-1 が返されるので、ループを繰り返すことになります。このため、データが読みだされるまで無限に待つことになるわけです。

85 行目では、読みだしたデータを返しています。

```
91  void Rx1BuffClear(void)
92  {
93    rx1_buff.start_index=0;
94    rx1_buff.data_count=0;
95  }
```

バッファを初期化する関数です。

すでに書いたように、データがない状態は start_index=0,data_count=0 です。ここではこの設定を行っています。データの中身を消す必要がないことに注意してください。

✍ 課題 11.1.2 (提出) 大文字小文字変換

ターミナルソフトから入力された文字がアルファベットの小文字だったら大文字に、大文字だったら小文字に変換して表示する (送り返す) プログラムを作ってください。アルファベット以外の文字についてはそのまま表示してください。

シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。

プロジェクト：e08_SCI02

11.1.20 1 行送受信 (割込みあり)

ここまでは、1 文字の送受信関数を作ってきました。しかし、実際には文字列の送受信を行うことのほうが多いかと思います。ここでは、1 文字送受信の関数を利用して、1 行送受信する関数を実装してみましょう。

まずは関数の仕様を決めましょう。

ここで作る文字列送受信関数は、パソコン側から受け取った復帰 (CR) は改行 (\n) に変換し、パソコンにデータを送るときには、改行 (\n) を CR+LF に変換することにしました。

前回のサンプルも含めて、関数の仕様は以下の通りです。

■ファイル (sci.c)、ヘッダファイル (sci.h) ■

```
------------------------------------------------------
void Sci1Init(BaudRate b)
形式 :
      #include "sci.h"
      void Sci1Init(BaudRate b);
機能 : シリアルポートの初期化を行う関数。
       シリアルポートを用いる場合には、必ず実行する。
引数 : BaudRate b(ボーレートを指定、br19200、br38400、br57600)
返却値 : 無し

------------------------------------------------------
void Rx1BuffClear(void)
形式 :
      #include "sci.h"
      void Rx1BuffClear(void);
機能 : 受信バッファクリア関数 (割り込み有り)。
       受信バッファをクリアする関数。
引数 : 無し
返却値 : 無し

------------------------------------------------------
int Sci1Write(char c)
形式 :
      #include "sci.h"
      int Sci1Write(char c);
機能 : 1 文字送信関数。
       0x200000 回送信を行って送信できない場合には関数を抜ける。
引数 : char c(送信する文字)
返却値 : 送信成功なら 0、失敗なら-1 を返す。

------------------------------------------------------
char Sci1Read(void)
形式 :
      #include "sci.h"
      char Sci1Read(void);
機能 : 1 文字受信関数 (割り込み、バッファ有り)。
引数 : 無し
返却値 : 受信した 1 文字を返す。

------------------------------------------------------
int Sci1Putchar(char c)
形式 :
      #include "sci.h"
      int Sci1Putchar(char c);
機能 : 1 文字送信関数。
       改行 (\n) は CR+LF に変換。
引数 : char c(送信する文字。改行 (\n) は CR+LF に変換)
返却値 : 送信成功なら 0、失敗なら-1 を返す。

------------------------------------------------------
int Sci1Puts(char *str)
形式 :
      #include "sci.h"
      int Sci1Puts(char *str);
機能 : 文字列送信関数。
       終端文字 (\0) まで文字列を送信。
       改行 (\n) は CR+LF に変換。
引数 : char *str(送信する文字列。最後は終端文字 (\0)。改行 (\n) は CR+LF に変換)
返却値 : 送信失敗なら 0、成功なら送信した文字数を返す。
```

■ファイル (sci.c)、ヘッダファイル (sci.h) ■

```
-------------------------------------------------------
char Sci1Getchar(void)
形式 :
      #include "sci.h"
      char Sci1Getchar(void);
機能 : 1 文字受信関数 (割り込み有り)。
      1 文字を受信する関数。
      CR は改行 (\n) に変換。
引数 : 無し
返却値 : 受信した 1 文字を返す。

-------------------------------------------------------
void Sci1Gets(char *str)
形式 :
      #include "sci.h"
      void Sci1Gets(char *str);
機能 : 文字列受信関数 (割り込み有り)。
      改行 (CR) まで文字列を受信する関数。
      CR は改行 (\n) に変換。
引数 : char *str(受信する文字列。改行 (CR) まで受信)
返却値 : 無し。

-------------------------------------------------------
```

以下では、新しい関数のみ説明することにします。

sci.c を利用する方法は、第 6 章 (P.151) を参考にしてください。

以下に手順をまとめておきます。

- 新しいプロジェクトを作る。
- フォルダ lib の下に sci.h,sci.c を作る。
- プロジェクトの中の iodefine.h は削る。
- プロジェクトに sci.c を追加。
- インクルードファイルディレクトリを設定する。
- ntprg.c の INT_SCI3 をコメントアウトする。

08_SCI03.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :08_SCI03.c                                             */
 4  /*  DESCRIPTION :シリアル通信 (1 行送受信、割込みあり)                  */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  RxD P21                                                             */
 8  /*  TxD P22                                                             */
 9  /*                                                                      */
10  /************************************************************************/
11
12  #include "iodefine.h"
13  #include "sci.h"
14
15  /*******************************
16     main 関数 :
17  *******************************/
18  void main(void)
19  {
20    char str[32];
21    Sci1Init(br19200);
22    Sci1Puts("******** H8/3694F ********\n");
23    while(1){
```

```
24      Sci1Gets(str);
25      Sci1Puts(str);
26      Sci1Putchar('\n');
27    }
28  }
```

End Of List

sci.h

```
1  #ifndef _SCI_H_
2  #define _SCI_H_
3
4  /* レジスタ定義 */
5  #define TXD1F SCI3.SSR.BIT.TDRE //送信データフラグ
6  #define RXD1F SCI3.SSR.BIT.RDRF //受信データフラグ
7  #define TD1D SCI3.TDR           //送信データ
8  #define RD1D SCI3.RDR           //受信データ
9
10 #define SCI_RX_BUFF_SIZE 32
11 #define SCI_RX_BUFF_MASK (SCI_RX_BUFF_SIZE-1)
12
13 typedef enum{
14   br19200=32,
15   br38400=15,
16   br57600=10
17 }BaudRate;
18
19 typedef struct{
20   unsigned short start_index;
21   unsigned short data_count;
22   char data_buff[SCI_RX_BUFF_SIZE];
23 }SciRxBuffer;
24
25 void Sci1Init(BaudRate b);
26 void Rx1BuffClear(void);
27
28 int Sci1Write(char c);
29 char Sci1Read(void);
30
31 int Sci1Putchar(char c);
32 int Sci1Puts(char *str);
33
34 char Sci1Getchar(void);
35 void Sci1Gets(char *str);
36
37 #endif
```

End Of List

sci.c

```
1  #include <machine.h>
2  #include "iodefine.h"
3  #include "sci.h"
4
5  #define CHECK_LOOP 0x200000
6  #define CR 0x0D
7  #define LF 0x0A
8
9  static SciRxBuffer rx1_buff;
10
11 /*********************************
12    シリアルポートを初期化 :Sci1Init
13 *********************************/
14 void Sci1Init(BaudRate b)
15 {
16   set_imask_ccr(1);   /* 割込み不可 */
17   SCI3.SCR3.BYTE = 0x00; /* TE、RE クリア   CKE 初期化 */
18   SCI3.SMR.BYTE  = 0x00; /* 調歩同期,8 ビット, パリティなし, ストップ 1 ビット n=0 */
19   SCI3.BRR  = b;
20   IO.PMR1.BIT.TXD = 1;  /* TXD ポートイネーブル */
21   SCI3.SCR3.BYTE = 0x70;    /* TE RE 受信割込み */
22   set_imask_ccr(0);   /* 割込み許可 */
23 }
24
25 /*******************************
26    1 バイト送信関数 :Sci1Write
27 *******************************/
28 int Sci1Write(char c)
29 {
30   unsigned long i;
31   unsigned short write_flag=0;    /* 送信データ書き込みフラグ */
32
33   for(i=0; i<CHECK_LOOP; i++){
34     if(TXD1F){ /* 送信データ書き込み可能になるまで待つ */
```

```
35        TXD1F=0;
36        write_flag=1;
37        break;
38      }
39    }
40    if(write_flag){
41      TD1D=c;           /* 送信データを格納 */
42      return(0);
43    }
44    return(-1);
45  }
46
47  /*******************************
48     1 バイト受信関数 (割り込み有り)
49  *******************************/
50  __interrupt(vect=23) void INT_SCI3(void)
51  {
52    char msg;
53    int index;
54
55    if(RXD1F){
56      RXD1F=0;
57      msg=RD1D;
58      if(rx1_buff.data_count < SCI_RX_BUFF_SIZE){
59        index=(rx1_buff.start_index+rx1_buff.data_count) & SCI_RX_BUFF_MASK;
60        rx1_buff.data_buff[index]=msg;
61        rx1_buff.data_count++;
62      }
63    }
64  }
65
66  int Sci1Rx1(char *msg)
67  {
68    if(rx1_buff.data_count > 0){
69      *msg=rx1_buff.data_buff[rx1_buff.start_index];
70      rx1_buff.start_index=(rx1_buff.start_index+1) & SCI_RX_BUFF_MASK;
71      rx1_buff.data_count--;
72      return(0);
73    }
74    *msg=0;
75    return(-1);
76  }
77
78  /***********************************************
79  *    1 文字受信 : Sci1Read
80  ***********************************************/
81  char Sci1Read(void)
82  {
83    char c;
84    while(Sci1Rx1(&c)){
85      ;
86    }
87    return(c);
88  }
89
90  /*************************************
91     受信バッファクリア関数 :Rx1BuffClear
92  *************************************/
93  void Rx1BuffClear(void)
94  {
95    rx1_buff.start_index=0;
96    rx1_buff.data_count=0;
97  }
98
99  /*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
100    汎用関数
101 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
102
103 /*********************************************************
104    1 バイト送信関数 (改行 (\n) は CR+LF に変換) :Sci1Putchar
105 *********************************************************/
106
107 int Sci1Putchar(char c)
108 {
109   /* 改行 (\n) は CR+LF に変換して送信 */
110   if(c=='\n'){
111     if(Sci1Write(CR)){
112       return(-1);
113     }
114     if(Sci1Write(LF)){
115       return(-1);
116     }
117   }else{  /* それ以外はそのまま送信 */
118     if(Sci1Write(c)){
119       return(-1);
120     }
121   }
122   return(0);
123 }
124
125 /*******************************************************
126 *     文字列送信 (改行 (\n) は CR+LF に変換) :Sci1Puts
127 *******************************************************/
```

```
128  int Sci1Puts(char *str)
129  {
130    char *msg;
131
132    /* 終端文字まで 1 文字ずつ表示 */
133    for(msg=str; *msg!='\0'; msg++){
134      if(Sci1Putchar(*msg)){
135        return(0);
136      }
137    }
138    return(msg-str);  /* 表示文字数を返す */
139  }
140
141  /************************************************
142  *    1 文字受信 (CR は改行 (\n) に変換) : Sci1Getchar
143  ************************************************/
144  char Sci1Getchar(void)
145  {
146    char c;
147    c=Sci1Read();
148    if(c==CR){
149      c='\n';
150    }
151    return(c);
152  }
153
154  /************************************************
155  *    文字列受信 (CR は改行 (\n) に変換) : Sci1Gets
156  ************************************************/
157  void Sci1Gets(char *str)
158  {
159    int i=0;
160    char s;
161
162    while((s=Sci1Getchar()) != '\n'){
163      str[i]=s;
164      i++;
165    }
166    str[i]='\0';
167  }
```

End Of List

実行結果

図 11.9　実行結果

ターミナルソフトに文字列を入力し、エンターを入力した時点でそれまでの文字列が表示される。

プログラム解説 (08_SCI03.c)

```
20    char str[32];
```

受信した文字列を格納するための配列です。

格納した文字列は、そのまま送信文字列として利用しています。

```
22    Sci1Puts("******** H8/3694F ********\n");
```

文字列送信関数を利用して、文字列を送信しています。文字列をダブルコーテーション (") で囲うことによって、文字列の最後に NULL 文字が追加されます。

\n は送信時に CR+LF に変換されます。

```
24    Sci1Gets(str);
25    Sci1Puts(str);
26    Sci1Putchar('\n');
```

受信した文字列を返すプログラムです。

24 行目で改行までの文字列を受信しています。受信した文字列は、配列 str に格納されます。

25 行目では、配列 str に格納された文字列を送信しています。この文字列には改行が入っていないので、26 行目で 1 文字送信関数を使って改行 (\n) を送っています。

プログラム解説 (sci.h)

新しく追加した部分について説明します。関数のプロトタイプ宣言を変更しただけです。

```
31  int Sci1Putchar(char c);
32  int Sci1Puts(char *str);
33
34  char Sci1Getchar(void);
35  void Sci1Gets(char *str);
```

1 文字送信関数 Sci1Putchar、文字列送信関数 Sci1Puts、1 文字受信関数 Sci1Getchar、文字列受信関数 Sci1Gets のプロトタイプ宣言を記述しました。

プログラム解説 (sci.c)

新しく追加した部分について説明します。追加した関数は、107 行目以降の「汎用関数」です。これらの関数にはハードウェア依存部がないことに注意してください (レジスタを直接にはいじっていない)。

```
107  int Sci1Putchar(char c)
108  {
109    /* 改行 (\n) は CR+LF に変換して送信 */
110    if(c=='\n'){
111      if(Sci1Write(CR)){
112        return(-1);
```

```
113      }
114      if(Sci1Write(LF)){
115        return(-1);
116      }
117    }else{  /* それ以外はそのまま送信 */
118      if(Sci1Write(c)){
119        return(-1);
120      }
121    }
122    return(0);
123  }
```

107 行目から 123 行目までは、1 文字送信関数です。\n は送信時に CR+LF に変換されます。

110 行目から 117 行目までが、\n を CR+LF に変換する部分です。110 行目で送信文字が\n であるかを調べています。\n だった場合、111 行目でまず CR を送信します。Sci1Write は送信に失敗すると-1 を返しますので、112 行目で-1 を返して関数を終了します。

111 行目で CR の送信に成功すると、114 行目で LF を送信します。この場合にも、送信に失敗すると、115 行目で-1 を返して関数を終了します。

\n 以外の場合には、117 行目以降が実行されます。118 行目で渡された 1 文字を送信しています。送信に失敗すると、119 行目で-1 を返して関数を終了します。

送信に成功した場合には、112 行目が実行され、0 を返して関数を抜けます。

```
128  int Sci1Puts(char *str)
129  {
130    char *msg;
131
132    /* 終端文字まで 1 文字ずつ表示 */
133    for(msg=str; *msg!='\0'; msg++){
134      if(Sci1Putchar(*msg)){
135        return(0);
136      }
137    }
138    return(msg-str);  /* 表示文字数を返す */
139  }
```

128 行目から 139 行目までは、文字列送信関数です。

Sci1Putchar を使って 1 文字ずつ NULL 文字 (\0) まで送信します。

130 行目では char 型のポインタ msg を宣言しています。これは引数として渡されたポインタの値をコピーし、この値を変更していきます。元の str の値はそのままにしておき、最終的に何文字送信したかが分かるようにします。

133 行目から 137 行目が文字列送信部分です。133 行目の for 文で、文字列の先頭のアドレスから、NULL 文字 (\0) まで、1 文字ずつ送信して言います。

134 行目で現在のアドレスの中身を送信し、失敗した場合には 135 行目で 0 を返して関数を終了します。

文字列の送信に成功したら、138 行目で msg と str のアドレスの差を取ることによって、送信文字数を計算します。この値を返して関数を終了します。

```
144  char Sci1Getchar(void)
145  {
```

```
146    char c;
147    c=Sci1Read();
148    if(c==CR){
149      c='\n';
150    }
151    return(c);
152  }
```

144 行目から 152 行目までは 1 文字受信関数です。

147 行目で、関数 Sci1Read を用いて受信バッファから 1 文字取りだします。

148 行目では、その文字が CR であった場合には、149 行目で\n に変換しています。ターミナルソフトでは、デフォルトでは改行として CR を送信しているようですのでこのようにしました。

151 行目で、受信した 1 文字を返しています。

```
157  void Sci1Gets(char *str)
158  {
159    int i=0;
160    char s;
161
162    while((s=Sci1Getchar()) != '\n'){
163      str[i]=s;
164      i++;
165    }
166    str[i]='\0';
167  }
```

157 行目から 167 行目までは文字列受信関数です。改行までを受信します。

この関数の中では Sci1Getchar を用いているので、改行は\n に変換されています。

157 行目で char 型のポインタを引数として受け取っていますが、1 行のデータが入るようにメモリを確保する必要があります。配列の先頭アドレスを渡すと良いでしょう。

159 行目の変数 i は何文字目かをカウントするためのものです。

162 行目から 165 行目までが文字列の受信です。関数 Sci1Getchar を用いて、\n まで受信しています。受信した文字は 163 行目で所定のアドレス (配列の要素) に書込みます。164 行目では i の値を 1 増やして次の書込みのためのアドレスを計算します。

166 行目は、文字列の最後に NULL 文字 (\0) をつけ足しています。

演習 11.1- 1

サンプルプログラムで、ターミナルソフトに表示するコマンドプロンプトを「H8/3694F>」に変更してみてください。また、入力された文字列を表示する際に、「You input:」と表示してから表示するようにしてください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e08_SCI03

11.1.21 数値と文字コード

ターミナルソフトからは、全ての文字が ASCII コードで送られてきます。

当然数字もそれぞれに対応した ASCII コードとして送られてくるわけです。

しかし、場合によっては数字を文字としてではなく数値として扱いたいこともあります。

たとえば、シリアル通信を用いて、0 を入力したら LED がすべて点灯、1 を入力したら LED がすべて消灯するプログラムを作るとします。ターミナルソフトからは 0 や 1 は文字として送信されます。アスキーコードでは文字 0 は 48、文字 1 は 49 です。受け取った文字を判定して、LED の点滅を指定すればよいでしょう。

しかし、これを数値に変換して、数値として扱えるようになれば、 0 なら全て消灯、1 なら LED 1 のみ点灯、2 なら LED2 のみ点灯などと 2 進数に対応した LED の点滅を作る際にも便利になります。

逆に、マイコン側でスイッチの値などを取得した場合には数値ですから、ターミナルソフトに表示するには対応した ASCII コードに変換する必要があります。

ここでは、数値を対応する数字の ASCII コードに変換したり、その逆をおこなう関数を作ってみましょう。

そのために、まずは 10 進数文字の判定を行う関数から作っていきましょう。

なお、数値と対応する数字の ASCII コードの変換は、シリアル通信以外にも利用することがありますので、新しく str.h と str.c というファイルに記述することにしました。

関数の仕様は以下の通りです。

■ファイル (str.c)、ヘッダファイル (str.h) ■

```
-------------------------------------------------------
int IsDigit(char c)
形式 :

      #include "srt.h"

       int IsDigit(char c);

機能 : 10 進数文字かチェックを行う関数。

引数 : char c(10 進数文字かチェックを行う 1 文字)

返却値 : 10 進数文字なら 1、それ以外なら 0 を返す。

-------------------------------------------------------
int IsHex(char c)
形式 :

      #include "srt.h"

       int IsHex(char c);

機能 : 16 進数文字かチェックを行う関数。

引数 : char c(16 進数文字かチェックを行う 1 文字)

返却値 : 16 進数文字なら 1、それ以外なら 0 を返す。

-------------------------------------------------------
int IsEos(char c)
形式 :

      #include "srt.h"

       int IsEos(char c);

機能 :「\n」あるいは「\0」文字かのチェックを行う関数。

引数 : char c(「\n」あるいは「\0」文字かチェックを行う 1 文字)

返却値 :「\n」あるいは「\0」文字なら 1、それ以外なら 0 を返す。
```

11.1.22 10 進数文字を判定

ここでは、ターミナルソフトから 1 文字読み込んで、それが 10 進数文字かどうかを判定し、結果を返すプログラムを作りましょう。

プロジェクトは、今まで同様に作ってください。今回は sci.c、str.c を追加する必要があります。

以下のソースリストには sci.h、sci.c を掲載しませんが、必要ですので追加するようにしてください。

08_SCI04.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :08_SCI04.c                                          */
 4  /*  DESCRIPTION :シリアル通信 (10 進数文字を判定)                    */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*  RxD P21                                                          */
 8  /*  TxD P22                                                          */
 9  /*                                                                   */
10  /*********************************************************************/
11
12  #include "iodefine.h"
13  #include "sci.h"
14  #include "str.h"
```

```
15
16 /*******************************
17    main 関数 :
18 *******************************/
19 void main(void)
20 {
21   char a;
22
23   Sci1Init(br19200);
24   while(1){
25     Sci1Putchar('>');
26     a=Sci1Getchar();
27     if(IsDigit(a)){
28       Sci1Putchar(a);
29       Sci1Puts(":Desit\n");
30     }else{
31       Sci1Putchar(a);
32       Sci1Puts(":Not Desit\n");
33     }
34     Sci1Putchar('\n');
35   }
36 }
```

End Of List

str.h

```
1 #ifndef _STR_H_
2 #define _STR_H_
3
4 int IsDigit(char);
5 int IsHex(char);
6 int IsEos(signed char c);
7
8 #endif
```

End Of List

str.c

```
1  #include "str.h"
2
3  /*******************************
4     10 進数文字かチェック
5  *******************************/
6  int IsDigit(char c)
7  {
8    if(c>='0' && c<='9'){
9      return(1);
10   }
11   return(0);
12 }
13
14 /*******************************
15    16 進数文字かチェック
16 *******************************/
17 int IsHex(char c)
18 {
19   if(c>='0' && c<='9'){
20     return(1);
21   }else if(c>='a' && c<='f'){
22     return(1);
23   }else if(c>='A' && c<='F'){
24     return(1);
25   }
26   return(0);
27 }
28
29 /*******************************
30    「\n」あるいは「\0」文字かのチェック
31 *******************************/
32 int IsEos(char c)
33 {
34   if(c=='\n' || c=='\0'){
35     return(1);
36   }
37   return(0);
38 }
```

End Of List

実行結果

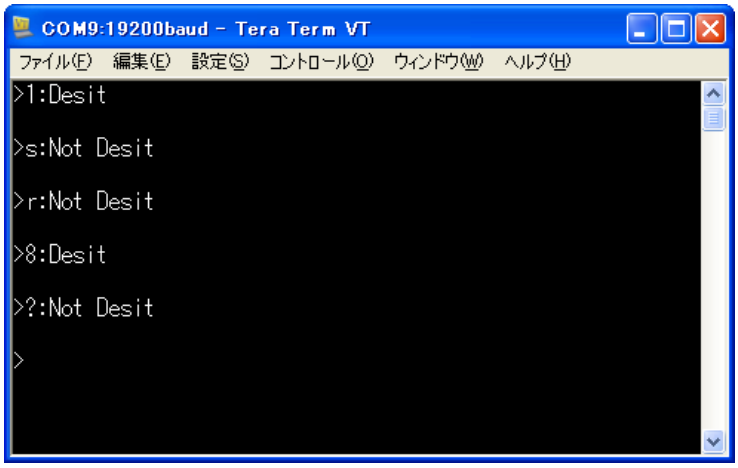


図 11.10 実行結果

ターミナルソフトに入力した文字が 10 進数文字かどうかを判定する。

プログラム解説 (08_SCI04.c)

```
14  #include "str.h"
```

今回利用する関数のプロトタイプ宣言が str.h に記述されているために、このファイルを読み込みます。

```
21    char a;
```

この変数は、受け取った 1 文字のデータを格納するのに利用します。

```
25    Sci1Putchar('>');
26    a=Sci1Getchar();
```

25 行目はプロンプトの表示です。「>」を表示します。

26 行目は Sci1Getchar を用いて 1 文字受信し、変数 a に格納しています。関数 Sci1Getchar は、受信した文字が CR であった場合には、\n に変換することに注意してください。

```
27    if(IsDigit(a)){
28      Sci1Putchar(a);
29      Sci1Puts(":Desit\n");
30    }else{
```

27 行目では、関数 IsDigit に受け取った 1 文字を渡しています。渡された文字が 10 進数文字の場合、IsDigit は 1 を返しますので、28 行目と 29 行目の処理を行います。

28 行目では、受け取った 1 文字を送信しています。

29 行目では、「:Desit」を送信して改行しています。

```
30    }else{
31      Sci1Putchar(a);
32      Sci1Puts(":Not Desit\n");
33    }
```

受け取った 1 文字が 10 進数文字ではなかったときの処理です。

31 行目で、受け取った 1 文字を送信しています。この部分は、28 行目と同じ処理なので、if 文の外に出してしまうこともできるでしょう。

32 行目では、「:Not Desit」を送信して改行しています。

```
34    Sci1Putchar('\n');
```

最後に改行を出力しています。

プログラム解説 (str.h)

このファイルは、関数のプロトタイプ宣言だけです。

プログラム解説 (str.c)

```
 6  int IsDigit(char c)
 7  {
 8    if(c>='0' && c<='9'){
 9      return(1);
10    }
11    return(0);
12  }
```

6 行目から 12 行目までは 10 進数文字かどうかを判定する関数です。

10 進数文字のアスキーコードは、「0」が 48、「1」が 49、「2」が 50、と連続した値です。「9」が 57 です。したがって、アスキーコードの値が「0」(48) から「9」(57) の間であれば 10 進数文字であることが分かります。この判定をしているのが 8 行目です。'0' は数字「0」のアスキーコードです。

受け取った文字が 10 進数文字である場合には、9 行目で 1 を返して関数を終了します。

そうでない場合には 11 行目で 0 を返して関数を終了します。

```
17  int IsHex(char c)
18  {
19    if(c>='0' && c<='9'){
20      return(1);
21    }else if(c>='a' && c<='f'){
22      return(1);
23    }else if(c>='A' && c<='F'){
24      return(1);
25    }
26    return(0);
27  }
```

17 行目から 27 行目までは 16 進数文字かどうかを判定する関数です。今回のプログラムでは利用していません。

19 行目から 21 行目までが 10 進数文字かどうかの判定です。

16 進数文字では、0 から 9 まで以外に、A から F までを使います。ここでは大文字でも小文字でも 16 進数文字として判定するようにプログラムしました。

21 行目から 23 行目までが小文字の a から f までの場合です。23 行目から 25 行目までが大文字の A から F までの場合の処理です。いずれも 1 を返して関数を終了します。

26 行目では 16 進数文字ではない場合です。0 を返して終了します。

```
32  int IsEos(char c)
33  {
34    if(c=='\n' || c=='\0'){
35      return(1);
36    }
37    return(0);
38  }
```

改行 (\n) と NULL 文字 (\0) を判定する関数です。

この関数も、今回のプログラムでは使っていません。

34 行目から 36 行目まででは、改行 (\n) か NULL 文字 (\0) であれば、1 を返すようにしています。

37 行目では、どちらでもない場合で、0 を返して関数を終了します。

演習 11.1- 2
16 進数文字の場合もプログラムしてみてください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e08_SCI04

11.1.23 数字を数値に変換 (LED の点灯)

10 進数文字、16 進数文字を判定する関数ができたので、これらを数値に変換する関数を作りましょう。

関数の仕様は以下の通りです。

■ファイル (str.c)、ヘッダファイル (str.h) ■

```
-------------------------------------------------------
int StrDecToLong(char *str, long *n)
形式 :

      #include "srt.h"

       int StrDecToLong(char *str, long *n);

機能 : 10 進数文字から数値に変換を行う関数。

引数 : char *str(10 進数文字列), long *n(long 型の数値へのポインタ)

返却値 :「\n」あるいは「\0」文字まで達していない場合には 0、達している場合には 1 を返す。

-------------------------------------------------------
int StrHexToLong(char *str, long *n)
形式 :

      #include "srt.h"

       int StrHexToLong(char *str, long *n);

機能 : 16 進数文字から数値に変換を行う関数。

引数 : char *str(16 進数文字列), long *n(long 型の数値へのポインタ)

返却値 :「\n」あるいは「\0」文字まで達していない場合には 0、達している場合には 1 を返す。

-------------------------------------------------------
```

これらの関数を用いて、ターミナルソフトから入力した数値によって、点灯する LED を変更するプログラムを作ってみましょう。

ターミナルソフトからは 10 進数文字 (0 から 7 まで) として値が送られて来るものとし、これを数値に変換して、LED 点滅用関数 LedSet に渡すことにします。

なお、このサンプルのように、受け取る数字の範囲が狭い場合には、if 文などで場合分けすることで数値に変換しなくてもプログラムすることはできます。

しかし、数字の範囲が広くなると (例えば 0 から 1000 まで)if 文で場合分けすることは難しくなることに注意してください。

以下がサンプルプログラムです。

08_SCI05.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :08_SCI05.c                                             */
 4  /*  DESCRIPTION :LED の点灯                                             */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /************************************************************************/
 8
 9
10  #include "led.h"
11  #include "sci.h"
12  #include "str.h"
13
14  void main(void)
15  {
16    long n;
17    char str[3];
18    int ret;
19
20    LedInit();
21    Sci1Init(br19200);
22    while(1){
23      Sci1Puts("********************\n");
```

```
24      Sci1Puts("LED ON,OFF\n");
25      Sci1Puts("Input 0-7\n");
26      Sci1Puts("********************\n");
27      Sci1Putchar('>');
28      str[0]=Sci1Getchar();
29      str[1]='\n';
30      str[2]='\0';
31      ret=StrDecToLong(str, &n);
32      if(ret==1 && (n>=0 && n<8)){
33        Sci1Puts("Input Data:");
34        Sci1Puts(str);
35        LedSet(n);
36      }else{
37        Sci1Puts("Input Error:");
38        Sci1Puts(str);
39      }
40    }
41  }
```

End Of List

str.h

```
1  #ifndef _STR_H_
2  #define _STR_H_
3
4  int IsDigit(char);
5  int IsHex(char);
6  int IsEos(signed char c);
7  int StrDecToLong(char *str, long *n);
8  int StrHexToLong(char *str, long *n);
9
10 #endif
```

End Of List

str.c

```
1  #include "str.h"
2
3  /*******************************
4     10 進数文字かチェック
5  *******************************/
6  int IsDigit(char c)
7  {
8    if(c>='0' && c<='9'){
9      return(1);
10   }
11   return(0);
12 }
13
14 /*******************************
15    16 進数文字かチェック
16 *******************************/
17 int IsHex(char c)
18 {
19   if(c>='0' && c<='9'){
20     return(1);
21   }else if(c>='a' && c<='f'){
22     return(1);
23   }else if(c>='A' && c<='F'){
24     return(1);
25   }
26   return(0);
27 }
28
29 /*******************************
30     「\n」あるいは「\0」文字かのチェック
31 *******************************/
32 int IsEos(char c)
33 {
34   if(c=='\n' || c=='\0'){
35     return(1);
36   }
37   return(0);
38 }
39
40 /*******************************
41    10 進数文字から数値に変換
42 *******************************/
43 int StrDecToLong(char *str, long *n)
44 {
45   int minus=0;
46
```

```
47    /* マイナスかをチェック */
48    if(*str == '-'){
49      minus=1;
50      str++;
51    }
52
53    /* 10 進数文字を数値に変換 */
54    for(*n=0; IsDigit(*str); str++){
55      *n *= 10;
56      *n += (*str - '0');
57    }
58
59    /* マイナスの場合 */
60    if(minus){
61      *n = -1*(*n);
62    }
63
64    /* 終端文字まで達していない場合 */
65    if(!IsEos(*str)){
66      return(0);
67    }
68    return(1);
69  }
70
71  /*****************************
72     16 進数文字から数値に変換
73  *****************************/
74  int StrHexToLong(char *str, long *n)
75  {
76
77    /* 16 進数文字を数値に変換 */
78    for(*n=0; IsHex(*str); str++){
79      *n <<= 4;
80      if(*str >= '0' && *str <= '9'){
81        *n += (*str - '0');
82      }else if(*str >= 'a' && *str <= 'f'){
83        *n += (*str - 'a' + 10);
84      }else if(*str >= 'A' && *str <= 'F'){
85        *n += (*str - 'A' + 10);
86      }
87    }
88
89    /* 終端文字まで達していない場合 */
90    if(!IsEos(*str)){
91      return(0);
92    }
93    return(1);
94  }
95
```

End Of List

実行結果

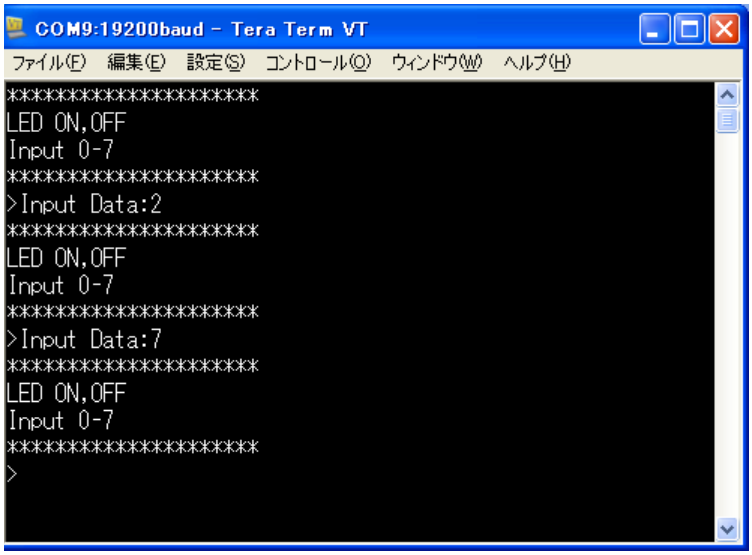


図 11.11 実行結果

ターミナルソフトに入力した値に応じて LED が点灯。

表 11.8 入力する数字と LED の点灯の関係

入力する数字	LED0	LED1	LED2
0	OFF	OFF	OFF
1	点灯	OFF	OFF
2	OFF	点灯	OFF
3	点灯	点灯	OFF
4	OFF	OFF	点灯
5	点灯	OFF	点灯
6	OFF	点灯	点灯
7	点灯	点灯	点灯

0 から 7 までの値意外を入れた場合には、エラーを表示する。

プログラム解説 (08_SCI05.c)

```
16    long n;
17    char str[3];
18    int ret;
```

16 行目の変数 n は受け取った 10 進数文字を数値に変換し、その値を格納する変数です。

17 行目の char 型の配列は、受け取った 10 進数文字をそのまま代入しておきます。返信するための改行と NULL 文字を追加するので、要素数は 3 としました。

18 行目の変数 ret は、関数 StrDecToLong を用いて 10 進数文字を数値に変換した結果 (変換に成功したら 1、最後まで変換できなかったら 0) を格納します。この値は、エラー処理に利用します。

```
23    Sci1Puts("**********************\n");
24    Sci1Puts("LED ON,OFF\n");
25    Sci1Puts("Input 0-7\n");
26    Sci1Puts("**********************\n");
27    Sci1Putchar('>');
```

ターミナルソフトに表示する画面のためのプログラムです。

図 11.11 と見比べてください。

```
28    str[0]=Sci1Getchar();
29    str[1]='\n';
30    str[2]='\0';
```

28 行目では、受け取った 1 文字を、配列 str(の先頭) に代入しています。

29 行目では、文字列を送り返して表示するために改行を入れています。10 進数文字を数値に変換する関数 StrDecToLong は、改行 (\n) もしくは NULL 文字 (\0) まで変換することにも注意してください。

30 行目では、文字列送信のために、最後に NULL 文字 (\0) を追加しています。

```
31    ret=StrDecToLong(str, &n);
```

10 進数文字を数値に変換する関数 StrDecToLong を実行しています。

str は 10 進数文字列 (今の場合は 1 文字) が入った配列 (の先頭のアドレス) です。変数 n のアドレスを渡すことで、変換の結果が n に代入されます。

ret には変換した結果 (変換に成功したら 1、最後まで変換できなかったら 0) が代入されます。

```
32    if(ret==1 && (n>=0 && n<8)){
33      Sci1Puts("Input Data:");
34      Sci1Puts(str);
35      LedSet(n);
36    }else{
```

10 進数文字を数値に変換でき、その値が 0 から 7 までであったら、33 行目から 35 行目までの処理を行います。

33 行目と 34 行目は、入力データ確認のために出力しています。図 11.11 と見比べてください。

35 行目で、入力データに対応する LED を点灯します。

```
36    }else{
37      Sci1Puts("Input Error:");
38      Sci1Puts(str);
39    }
```

受信文字が 0 から 7 まで出なかった場合の処理です。37 行目でエラーであることを表示し、38 行目で入力データを表示しています。

プログラム解説 (str.h)

このファイルは、関数のプロトタイプ宣言だけです。

プログラム解説 (str.c)

39 行目まではすでに説明した関数です。ここでは新たに付け加えた関数のみ説明します。

```
45  int minus=0;
```

43 行目から 69 行目までは、10 進数文字から数値に変換する関数 StrDecToLong です。

45 行目の変数 minus は入力された数字が正なら 0 に、負なら 1 に設定します。

```
48    if(*str == '-'){
49      minus=1;
50      str++;
51    }
```

受け取った 10 進数文字が負であるかどうかの判定です。最初が「-」であるときには負の 10 進数なので、48 行目でこれを判定しています。

負の数であった場合には、49 行目で変数 minus を 1 に設定し、50 行目で次の文字へと進んでいます。

```
54    for(*n=0; IsDigit(*str); str++){
55      *n *= 10;
56      *n += (*str - '0');
57    }
```

54 行目から 57 行目は、10 進数文字を数値に変換する部分です。

54 行目では、最初に数値を格納する変数の値を 0 にして、渡された文字列を改行 (\n) もしくは NULL 文字 (\0) まで 1 文字ずつ移動しています。

関数 IsDigit の戻り値は、10 進数文字なら 1 を、それ以外なら 0 を返します。条件文が真 (0 以外) のときはループは繰り返されます。偽 (0) なら for ループを抜けます。渡された文字列 (先頭のアドレス str) は、10 進数の後に改行 (\n) もしくは NULL 文字 (\0) が入っているので、そこまでループを繰り返すことになるわけです。

55 行目で、今の値を 10 倍し、56 行目で現在の 1 文字を数値に直しています。

文字「0」から「9」までのアスキーコードは連続しているので、それぞれのアスキーコードから「0」のアスキーコードを引けば数値に変換できます。

```
60    if(minus){
61      *n = -1*(*n);
62    }
```

変数 minus が 1 のとき、受け取った 10 進数文字は負ですので、-1 をかけます。

```
65    if(!IsEos(*str)){
66      return(0);
67    }
68    return(1);
```

改行 (\n) もしくは NULL 文字 (\0) まで達していない時には、変換が正しく行われなかったとして 0 を返します。

正しく行われた場合には 1 を返します。

```
79    *n <<= 4;
80    if(*str >= '0' && *str <= '9'){
81      *n += (*str - '0');
82    }else if(*str >= 'a' && *str <= 'f'){
83      *n += (*str - 'a' + 10);
84    }else if(*str >= 'A' && *str <= 'F'){
85      *n += (*str - 'A' + 10);
86    }
```

74 行目から 94 行目までは、16 進数文字を数値に変換する部分です。

79 行目では、それまでの値を 16 倍しています。4 ビット左にシフトすると 16 倍したことと同等であることに注意してください。

80 行目と 81 行目は、「0」から「9」までの場合です。10 進数文字の場合と同様に、「0」のアスキーコードを引けば対応する数値になります。

82 行目と 83 行目は「a」から「f」の場合です。文字「a」から「f」のアスキーコードも連続しています。「a」は 10 進数で 10 に対応しますので、入力された文字のアスキーコードから「a」のアスキーコードを引き、10 を足せば対応する数値になるわけです。

84 行目と 85 行目は、「A」から「F」までの場合です。小文字のときと同様の処理をしています。

演習 11.1- 3
1 から 3 までの値を入力したら、LED 三つが点滅するプログラムを作ってください。1 が一番点滅が早く、3 が一番遅いものとします。点滅の間隔 (点灯と消灯の間隔) は空のループを回して、1 が 0.5 秒程度、2 が 1 秒程度、3 が 2 秒程度になるようにしてください。
また、1 から 3 までの値以外はエラーを返すようにしてください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e08_SCI05

11.1.24 数値を 10 進数文字に変換 (レジスタの値を送信)

数字 (文字) を数値に変換することができましたので、ここでは逆に、数値を数字 (文字) に変換する関数を作りましょう。今回も 10 進数文字へ変換する関数と、16 進数文字へ変換する関数の両方を作ります。

16 進数文字は正の数に限り (エラー処理を行いませんでした)、「A」から「F」までは、大文字にするか小文字にするかを指定できるようにしました。

ここで使う文字列操作関数の仕様は以下の通りです。

■ファイル (str.c)、ヘッダファイル (str.h) ■

```
-------------------------------------------------------
int LongToStrDec(long n, char *str)
形式 :

      #include "srt.h"

       int LongToStrDec(long n, char *str);

機能 : 数値から 10 進数文字に変換を行う関数。

引数 : long n(long 型の数値), char *str(10 進数文字列)

返却値 : 10 進数文字列の文字数。
-------------------------------------------------------
int LongToStrHex(long n, char *str)
形式 :

      #include "srt.h"

       int LongToStrHex(long n, int upper, char *str);

機能 : 数値から 16 進数文字に変換を行う関数。

引数 : long n(long 型の数値), int upper(大文字小文字の指定), char *str(16 進数文字列)

返却値 : 16 進数文字列の文字数。
-------------------------------------------------------
```

08_SCI06.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :08_SCI06.c                                          */
 4  /*  DESCRIPTION :レジスタの値を送信                                  */
 5  /*  CPU TYPE    :H8/3694F                                            */
 6  /*                                                                   */
 7  /*********************************************************************/
 8
 9
10  #include "iodefine.h"
11  #include "sci.h"
12  #include "str.h"
13
14  void main(void)
15  {
16     char str[4];
17
18     Sci1Init(br19200);
19
20     Sci1Puts("********************\n");
21     Sci1Puts("BaudRate SCI3.BRR\n");
22     Sci1Puts("********************\n");
23     LongToStrDec(SCI3.BRR, str);
24     Sci1Puts(str);
25     while(1){
26       ;
27     }
28   }
```

End Of List

str.h

```
1  #ifndef _STR_H_
2  #define _STR_H_
3
4  int IsDigit(char);
5  int IsHex(char);
```

```
 6  int IsEos(signed char c);
 7  int StrDecToLong(char *str, long *n);
 8  int StrHexToLong(char *str, long *n);
 9  int LongToStrDec(long n, char *buf);
10  int LongToStrHex(long n, int upper, char *buf);
11
12  #endif
```

End Of List

str.c

```
 1  #include "str.h"
 2
 3  /*******************************
 4     10 進数文字かチェック
 5  *******************************/
 6  int IsDigit(char c)
 7  {
 8    if(c>='0' && c<='9'){
 9      return(1);
10    }
11    return(0);
12  }
13
14  /*******************************
15     16 進数文字かチェック
16  *******************************/
17  int IsHex(char c)
18  {
19    if(c>='0' && c<='9'){
20      return(1);
21    }else if(c>='a' && c<='f'){
22      return(1);
23    }else if(c>='A' && c<='F'){
24      return(1);
25    }
26    return(0);
27  }
28
29  /*******************************
30     「\n」あるいは「\0」文字かのチェック
31  *******************************/
32  int IsEos(char c)
33  {
34    if(c=='\n' || c=='\0'){
35      return(1);
36    }
37    return(0);
38  }
39
40  /*******************************
41     10 進数文字から数値に変換
42  *******************************/
43  int StrDecToLong(char *str, long *n)
44  {
45    int minus=0;
46
47    /* マイナスかをチェック */
48    if(*str == '-'){
49      minus=1;
50      str++;
51    }
52
53    /* 10 進数文字を数値に変換 */
54    for(*n=0; IsDigit(*str); str++){
55      *n *= 10;
56      *n += (*str - '0');
57    }
58
59    /* マイナスの場合 */
60    if(minus){
61      *n = -1*(*n);
62    }
63
64    /* 終端文字まで達していない場合 */
65    if(!IsEos(*str)){
66      return(0);
67    }
68    return(1);
69  }
70
71  /*******************************
72     16 進数文字から数値に変換
73  *******************************/
74  int StrHexToLong(char *str, long *n)
75  {
76
77    /* 16 進数文字を数値に変換 */
```

```
 78    for(*n=0; IsHex(*str); str++){
 79      *n <<= 4;
 80      if(*str >= '0' && *str <= '9'){
 81        *n += (*str - '0');
 82      }else if(*str >= 'a' && *str <= 'f'){
 83        *n += (*str - 'a' + 10);
 84      }else if(*str >= 'A' && *str <= 'F'){
 85        *n += (*str - 'A' + 10);
 86      }
 87    }
 88
 89    /* 終端文字まで達していない場合 */
 90    if(!IsEos(*str)){
 91      return(0);
 92    }
 93    return(1);
 94  }
 95
 96  /*********************************
 97    数値を 10 進数文字変換
 98   *********************************/
 99  int LongToStrDec(long n, char *buf)
100  {
101    char c;
102    int len = 0;
103    long i, half;
104    int minus=0;
105
106    /* マイナスかをチェック */
107    if(n<0){
108      minus=1;
109      n=-n;
110    }
111
112    /* 10 進文字列へ変換し文字数をカウント */
113    do{
114      if(n == 0){
115        i = 0;
116      }else{
117        i = n % 10;
118      }
119      buf[len] = (char)(i + '0');
120      len++;
121      n /= 10;
122    }while(n != 0);
123    if(minus==1){
124      buf[len]='-';
125      len++;
126    }
127
128    /* 文字の並び順を直す */
129    half = len >> 1;
130    for(i=0; i < half; i++){
131      c = buf[i];
132      buf[i] = buf[(len-1)-i];
133      buf[(len-1)-i] = c;
134    }
135
136    /* 終端文字列の挿入 */
137    buf[len]='\0';
138    return(len);
139  }
140
141  /*********************************
142    数値を 16 進数文字変換
143   *********************************/
144  int LongToStrHex(long n, int upper, char *buf)
145  {
146    char c;
147    char a = 'a';
148    int len = 0;
149    long i, half;
150
151    /* 大文字/小文字の設定 */
152    if(upper){
153      a = 'A';
154    }
155
156    /* 16 進文字列へ変換し文字数をカウント */
157    do{
158      i = n & 0x0F;
159      if(i > 9){
160        buf[len] = (char)(i + a - 10);
161      }else{
162        buf[len] = (char)(i + '0');
163      }
164      len++;
165      n >>= 4;
166    }while(n != 0);
167
168    /* 文字の並び順を直す */
169    half = len >> 1;
170    for(i=0; i < half; i++){
```

```
171       c = buf[i];
172       buf[i] = buf[(len-1)-i];
173       buf[(len-1)-i] = c;
174     }
175
176     /* 終端文字列の挿入 */
177     buf[len]='\0';
178     return(len);
179   }
```

End Of List

実行結果

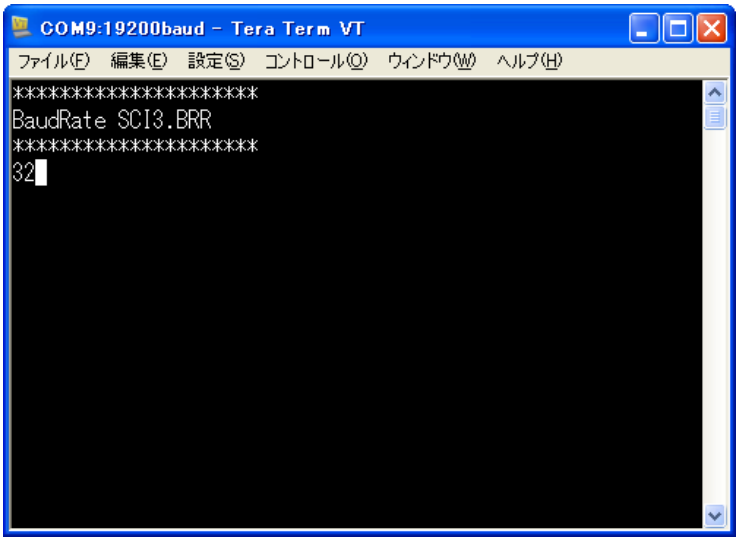


図 11.12　実行結果

レジスタ SCI3.BRR の値を表示。

プログラム解説 (08_SCI06.c)

```
10  #include "iodefine.h"
```

今回はレジスタの値を読みだすために、iodefine.h を読みこむようにしました。

```
16    char str[4];
```

レジスタの値を 10 進数文字に変換した結果を格納する配列です。

ビットレートを設定しているビットレートレジスタ (BRR) は 8 ビットのレジスタなので、最大でも 3 桁です。NULL 文字を最後に追加するので、配列の要素は 4 にしました。

```
20   Sci1Puts("**********************\n");
21   Sci1Puts("BaudRate SCI3.BRR\n");
22   Sci1Puts("**********************\n");
```

ターミナルソフトに表示する画面のためのプログラムです。

図 11.12 と見比べてください。

```
23   LongToStrDec(SCI3.BRR, str);
```

レジスタの値を 10 進数文字に変換します。SCI3.BRR がビットレートレジスタ (BRR) です。今の場合、18 行目で 19200bps に設定しています。対応する数値は 32 です。

変換後の 10 進数文字は配列 str に代入されます。

```
24   Sci1Puts(str);
```

ビットレートレジスタ (BRR) の値を 10 進数文字に変換した結果を送信しています。

プログラム解説 (str.h)

このファイルは、関数のプロトタイプ宣言だけです。

プログラム解説 (str.c)

```
101    char c;
102    int len = 0;
103    long i, half;
104    int minus=0;
```

99 行目から 139 行目までが、数値を 10 進数に変換する関数 LongToStrDec です。

変数 c は文字列の並び替えの際に、作業用に利用します。

変数 len は文字列の長さを代入します。

変数 i は for ループのカウンタなどに利用します。

変数 half は文字列の半分の長さを格納します。文字列を並べ替えるのに使います。

変数 minus は、数値が正である場合には 0 を、負である場合には 1 を代入します。

```
107    if(n<0){
108      minus=1;
109      n=-n;
110    }
```

変換する数値 n が負だった場合には、変数 minus を 1 にし、n に-1 をかけて正の値にします。

```
113    do{
114      if(n == 0){
115        i = 0;
116      }else{
117        i = n % 10;
118      }
119      buf[len] = (char)(i + '0');
120      len++;
121      n /= 10;
122    }while(n != 0);
```

数値を 10 進数文字に変換しています。

do、while ループは n の値が 0 になるまで、次の操作を繰り返します。

114 行目は変換する値 n が 0 の場合の処理です。ここは、最初に渡された n が 0 の場合のみ処理が行われます。このとき、115 行目で i に 0 を代入します。ここで i には最下位の桁から 1 桁分の数値を代入するのに利用しています。

n が 0 でなければ (1 以上であれば)、10 で割った余りを求めて、最下位の桁の数値を割り出します。結果を変数 i に代入します (117 行目)。

119 行目では、1 桁の数値を 10 進数文字に変換しています。「0」のアスキーコードを足せば 10 進数文字になるわけです。ここで、10 進数文字に変換した結果は、下位ビットから格納されていることに注意してください。これらは最終的に並べ替えを行う必要があるわけです。

120 行目では、文字列の長さをカウントしている変数 len に 1 を足しています。

121 行目では、n を 10 で割って最下位の桁を削っています。たとえば、n の値が 283 だとすると、10 で割った結果は 28 になります。結果が整数になることに注意してください。

122 行目では、上記のように、n が 0 になるまでループを繰り返しているわけです。

```
123    if(minus==1){
124      buf[len]='-';
125      len++;
126    }
```

変換する数値 n が負の値だったときの処理です。

107 行目から 110 行目までで、n の値が負だったら変数 minus を 1 にしましたので、minus が 1 だった場合には、文字列の最後に文字「-」を追加します。文字列の順番は逆になっていることに注意してください。

125 行目では、文字数をカウントする変数 len に 1 を足しています。

```
129    half = len >> 1;
130    for(i=0; i < half; i++){
131      c = buf[i];
132      buf[i] = buf[(len-1)-i];
133      buf[(len-1)-i] = c;
134    }
```

129 行目から 134 行目までは、逆順に入っている 10 進数文字列を正しい順番に入れなおす処理です。

入れ替えは、最初の文字と最後の文字、最初から二番目の文字と最後から二番目の文字という順番で行っていきます。最後は、文字数が奇数である場合には、真ん中の文字の前後を入れ替えて終了になります。偶数の場合には、真ん中の二文字を入れ替えて終了です。いずれも入れ替えの回数は文字数の半分 (2 で割った整数部) です。これは 1 ビット右シフトすることで得られます。

129 行目で、この入れ替え回数の計算を行っています。

130 行目から 134 行目までが入れ替え操作です。入れ替えるにあたっては、値を一時避難させておく変数 c を用いています。

```
137    buf[len]='\0';
138    return(len);
```

137 行目は、文字列の最後に終端文字として NULL 文字を追加しています。これは文字数に入れていません (len に 1 加えない)。

138 行目で、文字列の長さ len を返しています。

144 行目から 179 行目までは、数値を 16 進数文字列に変換する関数 LongToStrHex です。

基本的には、数値を 10 進数文字列に変換する関数 LongToStrDec と一緒です。ここでは異なる部分だけ説明しておきます。

```
147    char a = 'a';
```

16 進数文字は「a」から「f」までを大文字でも小文字でも表すことができます。どちらの表示にするのかを引数 upper で指定します (大文字なら 1、小文字なら 0)。

小文字を使う場合には「a」のアスキーコードを利用し、大文字を使う場合には「A」のアスキーコードを使います。ここでは、このアスキーコードを格納しておくために変数 a を使うことにしました。

```
152    if(upper){
153      a = 'A';
154    }
```

引数 upper が 1 の場合には、「A」のアスキーコードを利用します。ここではこの書き換えを行っています。

```
157    do{
158      i = n & 0x0F;
159      if(i > 9){
160        buf[len] = (char)(i + a - 10);
161      }else{
162        buf[len] = (char)(i + '0');
163        len++;
164        n >>= 4;
165      }
166    }while(n != 0);
```

10 進数の最下位 1 桁を取り出すには、10 で割ったあまりを求めましたが、16 進数の場合には&0x0F と論理積を取ることで計算できます。%を用いるより、ビット演算を行ったほうが処理が早いようです。

159 行目と 160 行目では、10 以上の値の場合に処理をしています。10 以上の場合には、10 を引いて「a」の (大文字で表示したい場合には「A」の) アスキーコードを足せば 16 進数文字に変換できます。

10 進数の場合には最下位一桁を削るのに、10 で割り算をしました。16 進数の場合には、4 ビット右シフトすることで同様の操作 (16 で割る) が実現できます。164 行目ではこれを行っています。

これ以降のプログラムは関数 LongToStrDec と同じです。

演習 11.1- 4
入力された文字のアスキーコードを返すプログラムを作ってください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e08_SCI06

演習 11.1- 5
ロータリスイッチの値を返すプログラムを作ってください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e08_SCI06_2

12 A/D変換の利用

12.1 A/D 変換

ここまでは、入力としてスイッチとシリアル通信を利用してきました。

スイッチは値が 0 もしくは 1 のどちらかで、1 ビットで格納できる値でした。

シリアルからの入力は文字列で、1 文字は 1 バイト (8 ビット) であらわされるアスキーコードであることを確認しました。これは 0,1,2,3…と続く整数値で、離散的 (とびとび) な値です。いわゆるデジタル量なわけです。

自然界の物理量というのは、ほとんどが連続した値です。体重ですと、70Kg と 71Kg の間には 70.1Kg とか 70.234297Kg とかいくらでも値が存在する連続量です。いわゆるアナログ量というものです。

マイコンは多くの場合周辺機器の制御に利用されます。このとき、外部の状況を取得して、目標の状態になるように制御器を動かすことになります。

たとえば、部屋を暖めようとした時には、現在の部屋の温度を取得して、必要に応じてヒータを発熱します。部屋の温度が 20 度で、21 度にしたいならば発熱量を抑えるでしょうし、部屋の温度が 5 度でしたら発熱量を多くすることになるでしょう。

このような外部環境を把握するためには、必要な情報を取得するための電子部品が必要になります。いわゆるセンサというものを利用するのです。

温度を測るのには温度センサ、距離を測るのには距離センサ、加速度を測るのには加速度センサを利用します。

これらのセンサの多くはその出力データがデジタル量ではなくアナログ量です。たとえば電圧で測定結果を出力します。アナログ量 (連続的な量) はコンピュータではそのまま扱うことができません。そこで、この出力されたアナログ量をデジタル量に変換する必要があるります。

この、アナログ量をデジタル量に変換する機能を実現しているのが、A/D 変換器です。

ここでは、A/D 変換器の使い方について学習しましょう。

12.1.1 H8/3694F の A/D 変換

H8/3694F には 8 チャンネルの A/D 変換器が備わっています。アナログの入力値はマイコンの A/D 変換器機能があるピンにつなぎます。つまり、最大でセンサ 8 個の値を取り込むことができるわけです。いずれも分解能は 10 ビットです。つまり、変換後のデジタル値は 2 進数 10 桁で表示されるわけです。変換の結果を格納するデータレジスタは 4 本あります。どのピンにつないだアナログ値がどのレジスタに格納されるかという説明は後ほどします。

12.1.2 入出力端子

A/D 変換器に関連する端子を表 12.1 に示します。データレジスタが 4 本であるために、8 本のアナログ入力端子は 2 つのグループに分割されています。アナログ入力端子 0〜3（AN0〜AN3）がグループ 0、アナログ入力端子 4〜7（AN4〜AN7）がグループ 1 です。AVcc 端子は A/D 変換器内のアナログ部の電源です。現在利用しているボード MB-RS10 では、アナログ部の電源端子は Vcc に接続されています。

表 12.1 A/D 変換器関連端子

端子名	略称	入出力	機能
アナログ電源端子	AVcc	入力	アナログ部の電源端子
アナログ入力端子 0	AN0	入力	グループ 0 のアナログ入力端子
アナログ入力端子 1	AN1	入力	
アナログ入力端子 2	AN2	入力	
アナログ入力端子 3	AN3	入力	
アナログ入力端子 4	AN4	入力	グループ 1 のアナログ入力端子
アナログ入力端子 5	AN5	入力	
アナログ入力端子 6	AN6	入力	
アナログ入力端子 7	AN7	入力	
A/D 外部トリガ入力端子	$\overline{ADTRG}$	入力	A/D 変換開始のためのトリガ入力端子

12.1.3 関連レジスタ

A/D 変換には以下のレジスタがあります。

- A/D データレジスタ A（ADDRA）
- A/D データレジスタ B（ADDRB）
- A/D データレジスタ C（ADDRC）
- A/D データレジスタ D（ADDRD）
- A/D コントロール／ステータスレジスタ（ADCSR）
- A/D コントロールレジスタ（ADCR）

A/D コントロールレジスタ（ADCR）は A/D 外部トリガ入力を利用する際に用います。今回はこの機能を使いませんので、それ以外のレジスタについてのみ説明します。上記のレジスタは次のアドレスに割り振られています。

表 12.2 A/D 変換関連レジスタのアドレス

レジスタ名	アドレス
A/D データレジスタ A（ADDRA）	H'FFB0
A/D データレジスタ B（ADDRB）	H'FFB2
A/D データレジスタ C（ADDRC）	H'FFB4
A/D データレジスタ D（ADDRD）	H'FFB6
A/D コントロール／ステータスレジスタ（ADCSR）	H'FFB8

A/D データレジスタ A～D（ADDRA～D）

A/D データレジスタは A/D 変換結果を格納するための 16 ビットのリード専用レジスタです。ADDRA ～ADDRD の 4 本あります。各アナログ入力チャネルの変換結果が格納される A/D データレジスタは表 12.3 のとおりです。

表 12.3 アナログ入力チャネルと A/D データレジスタの対応

アナログ入力チャンネル		変換結果が格納される
グループ 0	グループ 1	A/D データレジスタ
AN0	AN4	ADDRA
AN1	AN5	ADDRB
AN2	AN6	ADDRC
AN3	AN7	ADDRD

10 ビットの変換データは A/D データレジスタのビット 15 からビット 6 に格納されます (図 12.1)。下位 6 ビットの読み出し値は常に 0 です。

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	–	–	–	–	–	–
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

図 12.1 A/D データレジスタ A～D（ADDRA～D）

A/D コントロール／ステータスレジスタ（ADCSR）

ADCSR は A/D 変換器の終了フラグ、割込み許可、スキャンモード、チャンネルセレクトビットなどで構成されています。

表 12.4 A/D コントロール／ステータスレジスタ（ADCSR）

ビット	ビット名	初期値	R/W	説 明
7	ADF	0	R/W	A/D エンドフラグ [セット条件] ・単一モードで A/D 変換が終了したとき ・スキャンモードで選択されたすべてのチャネルの変換が 1 回終了したとき [クリア条件] ・1 の状態をリードした後、0 をライトしたとき
6	ADIE	0	R/W	A/D インタラプトイネーブル このビットを 1 にセットすると ADF による A/D 変換終了割り込み要求（ADI）が イネーブル (有効) になります。
5	ADST	0	R/W	A/D スタート このビットを 1 にセットすると A/D 変換を開始します。 単一モードでは A/D 変換を終了すると自動的にクリアされます。 スキャンモードではソフトウェア、リセット、 またはスタンバイモードによってクリアされるまで 選択されたチャネルを順次連続変換します。
4	SCAN	0	R/W	スキャンモード (P.316 の 12.1.4 参照) A/D 変換のモードを選択します。 0：単一モード 1：スキャンモード
3	CKS	0	R/W	クロックセレクト A/D 変換時間の設定を行います。 0：変換時間＝ 134 ステート（max） 1：変換時間＝ 70 ステート（max） 変換時間の切換えは、ADST ＝ 0 の状態で行ってください。
2 1 0	CH2 CH1 CH0	0 0 0	R/W R/W R/W	チャネルセレクト 2〜0 アナログ入力チャネルを選択します。 SCAN ＝ 0 のとき 000：AN0 001：AN1 010：AN2 011：AN3 100：AN4 101：AN5 110：AN6 111：AN7 SCAN ＝ 1 のとき 000：AN0 001：AN0〜AN1 010：AN0〜AN2 011：AN0〜AN3 100：AN4 101：AN4〜AN5 110：AN4〜AN6 111：AN4〜AN7

12.1.4 単一モードとスキャンモード

A/D 変換器の動作モードには単一モードとスキャンモードがあります。

動作モードやアナログ入力チャネルの切換えは、ADCSR の ADST ビットが 0 の状態で行ってください。動作モードモードやアナログ入力チャネルの変更と ADST ビットのセットは同時に行うことができます。

単一モードは指定された 1 チャネルのアナログ入力を 1 回 A/D 変換します (P.316 表 12.4 参照)。

スキャンモードは指定された最大 4 チャネルのアナログ入力を順次連続して A/D 変換します (P.316 表 12.4 参照)。

SCI3 のレジスタ定義 (iodefine.h の一部を抜粋)

HEW のレジスタ定義ファイルのうち、関連する部分を載せておきます。

AD 変換のレジスタ定義 (iodefine.h の一部を抜粋)

```
struct st_ad {                                          /* struct A/D   */
              unsigned int       ADDRA;                 /* ADDRA        */
              unsigned int       ADDRB;                 /* ADDRB        */
              unsigned int       ADDRC;                 /* ADDRC        */
              unsigned int       ADDRD;                 /* ADDRD        */
              union {                                   /* ADCSR        */
                    unsigned char BYTE;                 /*  Byte Access */
                    struct {                            /*  Bit  Access */
                           unsigned char ADF :1;        /*    ADF       */
                           unsigned char ADIE:1;        /*    ADIE      */
                           unsigned char ADST:1;        /*    ADST      */
                           unsigned char SCAN:1;        /*    SCAN      */
                           unsigned char CKS :1;        /*    CKS       */
                           unsigned char CH  :3;        /*    CH        */
                           }      BIT;                  /*              */
                    }             ADCSR;                /*              */
              union {                                   /* ADCR         */
                    unsigned char BYTE;                 /*  Byte Access */
                    struct {                            /*  Bit  Access */
                           unsigned char TRGE:1;        /*    TRGE      */
                           }      BIT;                  /*              */
                    }             ADCR;                 /*              */
};                                                      /*              */

......

#define AD      (*(volatile struct st_ad    *)0xFFB0)   /* A/D   Address*/
```

今回の関数の仕様は以下の通りです。

■ファイル (ad.c)、ヘッダファイル (ad.h) ■

```
------------------------------------------------------
void AdIinit(void)
形式：
      #include "ad.h"
      void AdIinit(void);
機能：A/D 変換の端子の初期化を行う関数。
      シングルモード、AN0 に限定
      A/D 変換を用いる場合には、必ず実行すること。
      デフォルトでは A/D 変換は停止状態。
引数：  無し
返却値：無し

------------------------------------------------------
void AdStart(void)
形式：
      #include "ad.h"
      void AdStart(void);
機能：A/D 変換を開始する関数。
引数：無し
返却値：無し

------------------------------------------------------
void AdStop(void)
形式：
      #include "ad.h"
      void AdStop(void);
機能：A/D 変換を停止する関数。
引数：無し
返却値：無し

------------------------------------------------------
unsigned short AdRead(void)
形式：
      #include "ad.h"
      unsigned short AdRead(void);
機能：A/D 変換の結果を取得する関数。
      AN0 の値のみ。
引数：無し
返却値：unsigned short(A/D 変換の結果 10 ビット)
------------------------------------------------------
```

12.1.5 A/D 変換の結果をシリアルで表示

まずは可変抵抗を AN0 に接続して、単一モードで値を A/D 変換し、その結果をシリアル通信を用いてターミナルソフトに表示してみましょう。

今プログラムは、AN0 につなぐものが可変抵抗でなくてもそのまま使えることに注意してください。5V 以下の出力でしたら、どのようなセンサをつないでもかまいません。

可変抵抗

半固定抵抗は、3 本の足があり、両端の足をプラス側と GND に繋ぎます。上部の可変部分をドライバで回すことによって、真中の足の抵抗が変えられます。

回路図

今回は AN0 に可変抵抗の出力をつなぎます。

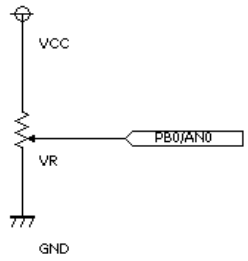


図 12.2 A/D 変換の回路図

今回は A/D 変換以外に、シリアル通信と数値を 10 進数に変換する関数を使いますので、sci.h、sci.c、str.h、str.c などが必要になります。

これらのファイルを利用する方法は、第 6 章 (P.151) を参考にしてください。

以下に手順をまとめておきます。

- 新しいプロジェクトを作る。
- プロジェクトの中の iodefine.h は削る。
- フォルダ lib の下の sci.c、str.c、ad.c をプロジェクトに追加。
- インクルードファイルディレクトリを設定する。

なお、受信割込みを使うために、intprg.c の INT_SCI3 をコメントアウトしておいてください。ここで宣言されている関数は、sci.c に記述しました。

intprg.c の INT_SCI3 をコメントアウト

```
// vector 23 SCI3
//__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
// vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
```

今回のプログラムでは、A/D 変換の結果は、4 回測定して平均をとるようにしました。このようにすると、値のばらつきが少なくなります。

では、以下にサンプルプログラムを示します。

09_ADC01.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :09_ADC01.c                                             */
/*  DESCRIPTION :AN0 の値を 4 回平均してシリアルで表示                  */
/*  CPU TYPE    :H8/3694F                                               */
/*                                                                      */
/*  PB0/AN0   41                                                        */
/*                                                                      */
/************************************************************************/

#include "ad.h"
#include "sci.h"
#include "str.h"

#define WAIT_COUNTER 0x000FFFFFL /* ループ回数 */

void main(void)
{
  char a[7];
  unsigned short i, j;
  unsigned int avr;
  unsigned long k;
  int count, avr_c;

  Sci1Init(br19200);
  Rx1BuffClear();
  AdInit();

  Sci1Puts("***** H8/3694F *****\n");

  while(1){
    avr=0;
    /* 4 回 A/D 変換をして平均を取る */
    for(avr_c=0; avr_c<4; avr_c++){
      AdStart();
      i=AdRead();
      avr=avr+i;
    }
    i=avr>>2;  /*  4 で割る */
    /*  数値を文字列に変換  */
    j=LongToStrDec((long)i,a);
    /* 5 桁未満の場合には 0 で埋める */
    for(count=0; count<5-j; count++){
      Sci1Putchar('0');
    }
    Sci1Puts(a);
    Sci1Putchar('\r');  /*  行頭に戻す */
    /*  ウェイトループ  */
    for(k=0; k<WAIT_COUNTER; k++){
      ;
    }
  }
}
```

End Of List

ad.h

```
1  #ifndef _AD_H_
2  #define _AD_H_
3
4  void AdIinit(int scan, int channel);
5  void AdStart(void);
6  void AdStop(void);
7  unsigned short AdRead(void);
8
9  #endif
```

End Of List

ad.c

```
1  /************************************************************************/
2  /*                                                                      */
3  /*  FILE        :ad.c                                                   */
4  /*  DESCRIPTION :A/D 変換用関数                                         */
5  /*  CPU TYPE    :h8/3694                                                */
6  /*                                                                      */
7  /*  AD01 AN0                                                            */
8  /*                                                                      */
9  /************************************************************************/
10
11 #include "iodefine.h"
12 #include "ad.h"
13
14 void AdInit(void)
15 {
16   AD.ADCSR.BIT.ADST = 0;        /* A/D 変換停止 */
17   AD.ADCSR.BIT.SCAN = 0;    /* モード */
18   AD.ADCSR.BIT.CKS  = 1;        /* 高速変換 */
19   AD.ADCSR.BIT.CH   = 0; /* チャンネル */
20 }
21
22 /*******************************
23    AD 変換開始
24 *******************************/
25
26 void AdStart(void)
27 {
28   AD.ADCSR.BIT.ADST = 1;      /* A/D 変換スタート */
29 }
30
31 /*******************************
32    AD 変換中止
33 *******************************/
34
35 void AdStop(void)
36 {
37   AD.ADCSR.BIT.ADST = 0;
38 }
39
40 /*******************************
41    AD 変換読み取り
42 *******************************/
43
44 unsigned short AdRead(void)
45 {
46   unsigned short ad_r;
47
48   while(!AD.ADCSR.BIT.ADF){
49     ;
50   }
51   ad_r=(AD.ADDRA>>6) ;
52   AD.ADCSR.BIT.ADF=0;
53   return(ad_r);
54 }
```

End Of List

実行結果

図 12.3 実行結果

ターミナルソフトに A/D 変換の結果が 0-1023 の値で表示される。

12.1.6 A/D 変換の結果と電圧の関係

ここでは、A/D 変換の結果と、出力電圧の関係を計算しておきましょう。

A/D 変換の結果が 0-1023 であり、1023 は 5V に対応しています。これから、出力結果が 1 増えると、電圧が、

$$\frac{5}{1023} = 0.0048876V$$

だけ増えることになります。

つまり、出力値に 0.0048876V をかけると出力電圧が得られるわけです。

プログラム解説 (09_ADC01.c)

```
11  #include "ad.h"
12  #include "sci.h"
13  #include "str.h"
```

このプログラムでは、ad.c、sci.c、str.c の関数が必要になります。そのためのヘッダファイルを読み込んでいます。

```
19    char a[7];
20    unsigned short i, j;
21    unsigned int avr;
22    unsigned long k;
23    int count, avr_c;
```

変数の宣言です。

19 行目の配列 a は、A/D 変換の結果を数字 (文字列) に変換した結果を格納します。今回は最大 5 桁 (1023 まで) で、最後に NULL 文字 (\0) を挿入するので、最大 6 文字分が必要です。ここでは、7 文字分を確保しています。

20 行目の変数 i は、A/D 変換の結果を代入したり、4 回の平均を代入したりします。

j は、A/D 変換の結果を文字列に変換した際の文字数を入れます。

21 行目の変数 avr は A/D 変換の 4 回の和を入れるのに利用します。

22 行目の変数 k は、ウェイトループのカウントのために利用します。

23 行目の count は、変換した文字列が 5 桁未満の場合に、上の桁を 0 で埋めるためのループカウンタとして使います。

avr_c は、A/D 変換の 4 回の和を計算する際のルーをプカウンタとして使います。

```
27    AdInit();
```

A/D 変換の初期化関数です。

シングルモードで AN0 に設定します。

```
34    for(avr_c=0; avr_c<4; avr_c++){
35      AdStart();
36      i=AdRead();
37      avr=avr+i;
38    }
39    i=avr>>2;  /*  4 で割る */
```

34 行目から 38 行目までで、A/D 変換を 4 回行い、その和を計算しています。

39 行目で、合計を右に 2 ビットシフトして、4 で割っています。

```
41    j=LongToStrDec((long)i,a);
```

A/D 変換の結果を、文字列に変換しています。

変数 j には、変換した文字数が代入されます。

```
43    for(count=0; count<5-j; count++){
44      Sci1Putchar('0');
45    }
46    Sci1Puts(a);
47    Sci1Putchar('\r');  /*  行頭に戻す */
```

43 行目から 45 行目までは、文字列が 5 桁未満の場合には上位の桁を 0 で埋めるるために、文字「0」を出力しています。表示桁が常に一定になるようにしています。

46 行目では、変換した文字列を送信しています。

47 行目ではターミナルに表示されるプロンプトを、行頭に戻しています。改行を行っていないので、表示が上書きされます。

プログラム解説 (ad.h)

関数のプロトタイプ宣言を行っています。

プログラム解説 (ad.c)

```
14    void AdInit(void)
15    {
16      AD.ADCSR.BIT.ADST = 0;        /* A/D 変換停止 */
17      AD.ADCSR.BIT.SCAN = 0;     /* モード */
18      AD.ADCSR.BIT.CKS  = 1;        /* 高速変換 */
19      AD.ADCSR.BIT.CH   = 0; /* チャンネル */
20    }
```

14 行目から 20 行目までは、A/D 変換の初期化関数です。

A/D コントロール／ステータスレジスタ（ADCSR）(P.316 の表 12.4) を設定しています。

16 行目は A/D 変換を停止しています。

17 行目は、単一モードとスキャンモードの選択です。0 に設定しているので、スキャンモードになります。

18 行目は、A/D 変換時間の設定です。70 ステートを選択しました。

19 行目は、チャンネルの選択です。AN0 を変換します。

```
26    void AdStart(void)
27    {
28      AD.ADCSR.BIT.ADST = 1;       /* A/D 変換スタート */
29    }
```

A/D 変換をスタートさせる関数です。

A/D 変換の初期化関数 AdInit では、A/D 変換を

スタートさせないことに注意してください。

```
35    void AdStop(void)
36    {
37      AD.ADCSR.BIT.ADST = 0;
38    }
```

A/D 変換をストップさせる関数です。

```
44    unsigned short AdRead(void)
45    {
46      unsigned short ad_r;
47
48      while(!AD.ADCSR.BIT.ADF){
49        ;
50      }
51      ad_r=(AD.ADDRA>>6);
52      AD.ADCSR.BIT.ADF=0;
53      return(ad_r);
54    }
```

A/D 変換の読み取り関数です。

46 行目の変数 ad_r は、AN0 の値を格納します。

48 行目から 50 行目までは、A/D 変換終了を待つループです。

51 行目では、A/D データレジスタ A(ADDRA) の値を読み込んでいます。下位 6 ビットは 0 ですので、6 ビット右シフトしています。

52 行目は、A/D エンドフラグをクリアしています。

53 行目で、A/D 変換の結果 (10 ビット) を返しています。

✎ 課題 12.1.1 (提出) A/D 変換の結果を LED に反映

可変抵抗器の電圧を A/D 変換した結果の上位 3 ビットを、LED 三つに反映するプログラムを作ってください。

1 が点灯 0 が消灯にしましょう。

プロジェクト名：e09_ADC01

✎ 課題 12.1.2 (提出) A/D 変換の結果を電圧で表示

可変抵抗器の電圧をターミナルソフトに出力するプログラムを作ってください。

プロジェクト名：e09_ADC02

12.1.7 PSD センサの値をシリアルで表示

12.1.5 の「A/D 変換の結果をシリアルで表示」のサンプルプログラムを使って、PSD センサの値を表示してみましょう。

今回利用する、PSD センサ GP2Y0A21YK0F の外観を図 12.4 に示します。ケーブルの色が VCC が黒、GND が赤、信号線が白ですので注意してください。

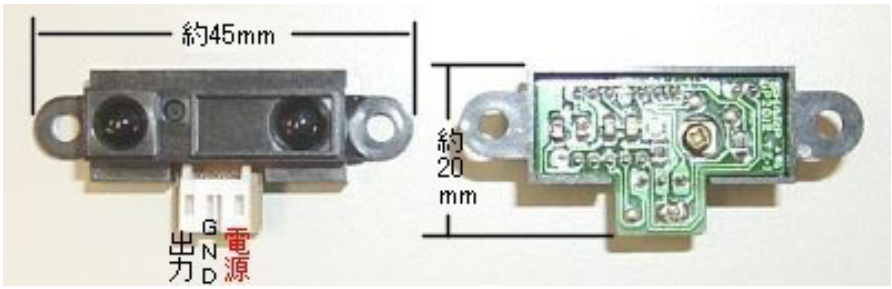


図 12.4　PSD センサ GP2Y0A21YK0F の外観

PSD センサは以下のような原理で物体までの距離を計測します (図 12.5、図 12.6)。

- スポット光が当たると電荷が発生し両極まで到達します。
- 発生する電荷の量は光の当たった位置から電極までの距離に「反」比例します。
- 両極の電流量を比較することで、スポット光が当たった場所を特定できます。

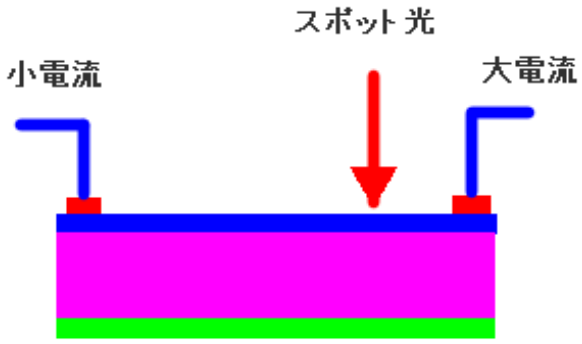


図 12.5　PSD センサの原理

- 障害物までの距離によって、反射光が返ってくる位置は違います。
- 反射光がどこに返ってきたかで、障害物までの距離が分かります。

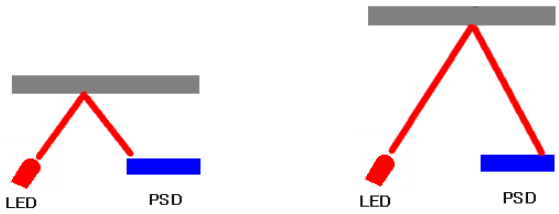


図 12.6　障害物が近いと左側に、遠いと右側にスポット光が当たる

障害物までの距離と PSD センサの出力の関係は以下の通りです (図 12.7)。計測できるのは、マニュアルによると、10cm から 80cm 程度です。電圧の最大値は 6cm あたりで 3V ちょっとのようです。

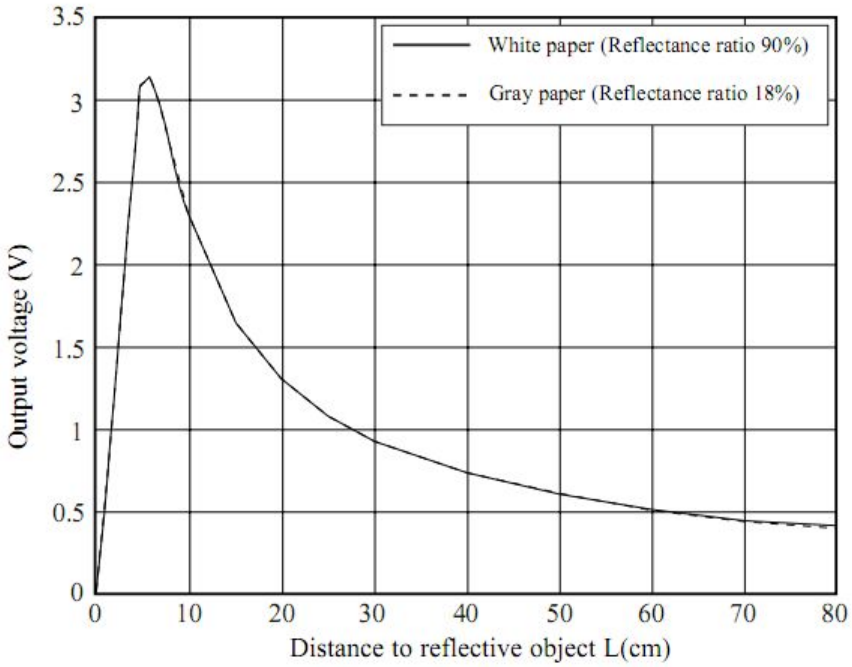


図 12.7　障害物までの距離と PSD センサの出力の関係 (マニュアルから抜粋)

今回の基板の A/D 変換は最大 5V ですので、PSD センサの出力を 1.5 倍に増幅すると精度良く計れそうです。以下のように OP アンプ LMC660 を使って増幅してみましょう。

今回はブレッドボード上に組むことにします。

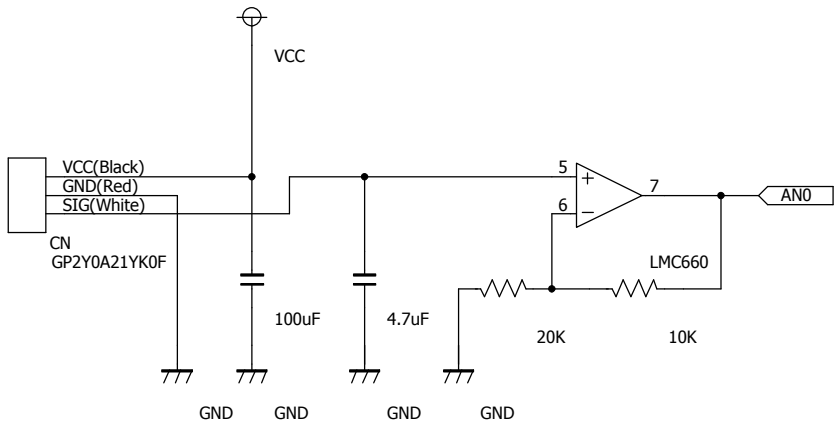


図 12.8　PSD センサの値を 1.5 倍に増幅する回路

今回のプログラムは、前回の 09_ADC01.c と同じ物を使います。

表示結果も P.322 の図 12.3 と同じです。

演習 12.1- 1
回路ができたら、実際にマイコンにつないでターミナルソフトに表示してみてください。距離と表示される値の関係をグラフ化して、図 12.7 のようになるか確認してみてください。
シリアル通信の設定は 19200bps、8 ビット、パリティなし、ストップビット 1 に設定してください。
プロジェクト：e09_ADC01_2

課題 12.1. 3　(提出) 距離を表示

ターミナルソフトに PSD センサから物体までの距離が表示されるようにしてください。

プロジェクト名：e09_ADC01_3

12.1.8 3 軸加速度センサを利用する

3 軸加速度センサを利用してみましょう。

ここで利用する加速度センサは、Kionix 社製 3 軸加速度センサ KXP84-2050(測定レンジ：±2g) です。これをモジュール化した KXP84 モジュールを使うことにしましょう (図 12.9)。

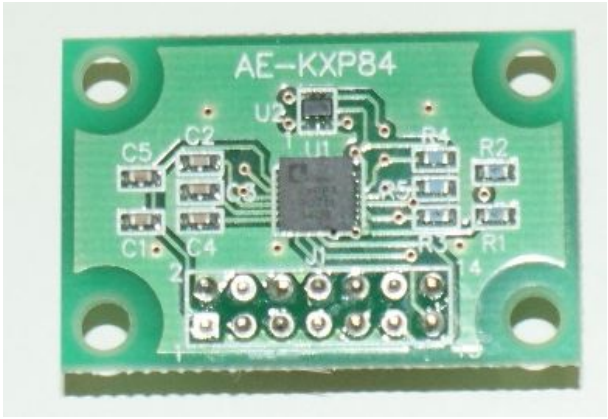


図 12.9 3 軸加速度センサ KXP84

KXP84 モジュールのピン配置をいかに示します (表 12.5)。

表 12.5 KXP84 モジュールのピン配置

ピン番号	記号	機能
1	VDD	電源
2	GND	グランド
3	MOT	Motion interrupt(割込み)
4	FF	Free-fall interrupt(割込み)
5	SCL/SCLK	シリアル通信クロック入力
6	IO_VDD	ディジタル入出力回路の電源
7	SDA_SD0	シリアル通信
8	RESET	リセット
9	ADDRO/SDI	スレーブアドレスの LSB 設定
10	CS	SPI 通信用選択端子
11	X_OUT	X 軸アナログ出力
12	Y_OUT	Y 軸アナログ出力
13	Z_OUT	Z 軸アナログ出力
14		接続しない

今回は、アナログ出力を利用するため、1,2 番ピンと 11 番ピンから 13 番ピンまでを利用します。

KXP84 モジュールの電源電圧は 3.3V ですが、最大定格が 5.25V ですので、マイコンの 5V 電源をそのまま使っています。

加速度センサは重力加速度も測定しますので、多くの場合 (重力の方向に関しての) 傾きを測定することになります。

たとえば、2 足歩行ロボットが直立しているか、うつぶせに倒れているか、仰向けに倒れているかなどを判定するのに使えます。

このテキストでは、後ほどロボットのコントローラに利用したいと思います。コントローラを傾けた方向にロボットは歩くというわけです。

KXP84 モジュールの傾きと、出力電圧の関係を測定してみたところ、図 12.10 ようになりました (電源電圧 5V)。

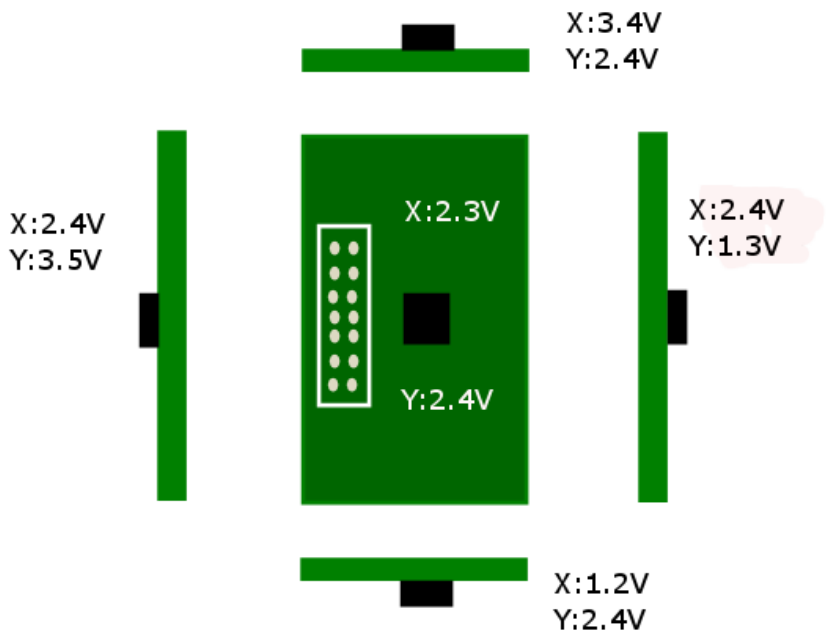


図 12.10 KXP84 の傾きと出力電圧

回路図

今回は、後ほどコントローラに使うことを考えて、X 軸と Y 軸のアナログ出力のみを計測することにしました。興味のある人は Z 軸についても測定してみてください。

以下に回路図を示します (図 12.11)。

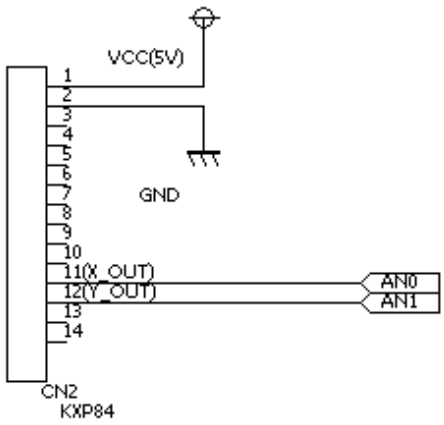


図 12.11 KXP84 の回路図

以下に示すのは、3 軸加速度センサの X 軸と Y 軸の出力を 10 ビットの範囲で表示するプログラムです。5 桁で表示し、上位のあいた桁には 0 を挿入しました。

09_ADC02.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :09_ADC02.c                                             */
/*  DESCRIPTION :AN0-1 の値を 4 回平均してシリアルで表示                */
/*  CPU TYPE    :H8/3694F                                               */
/*                                                                      */
/*  AD01  CN550-12                                                      */
/*                                                                      */
/************************************************************************/

#include "iodefine.h"
#include "sci.h"
#include "str.h"

#define WAIT_COUNTER 0x000FFFFFL /* ループ回数 */

#define WAIT_LOOP 0x06FFFFFL /* ループ回数 */

/*******************************
  wait 関数
*******************************/
void WaitLoop(void)
{
  unsigned long i;

  for(i=0; i<WAIT_LOOP; i++){
    ;
  }
}

/*******************************
   AD のつながった端子を初期化
*******************************/
void AdInit(void)
{
  AD.ADCSR.BIT.ADST = 0;      /* A/D 変換停止 */
  AD.ADCSR.BIT.SCAN = 1;      /* スキャンモード */
  AD.ADCSR.BIT.CKS  = 1;      /* 高速変換 */
  AD.ADCSR.BIT.CH   = 1;      /* AN0-1 */
}

/*******************************
   AD 変換開始
*******************************/
void AdStart(void)
{
  AD.ADCSR.BIT.ADST = 1;      /* A/D 変換スタート */
```

```
 48  }
 49
 50  /*******************************
 51     AD 変換中止
 52  *******************************/
 53  void AdStop(void)
 54  {
 55    AD.ADCSR.BIT.ADST = 0;
 56  }
 57
 58  /*******************************
 59     AD 変換読み取り
 60  *******************************/
 61  unsigned short AdRead(int a)
 62  {
 63    unsigned short ad_r;
 64
 65    while(!AD.ADCSR.BIT.ADF){
 66      ;
 67    }
 68    if(a==0){
 69      ad_r=(AD.ADDRA>>6) ;
 70    }else if(a==1){
 71      ad_r=(AD.ADDRB>>6) ;
 72    }
 73    AD.ADCSR.BIT.ADF=0;
 74    return(ad_r);
 75  }
 76
 77  /*******************************
 78     main 関数
 79  *******************************/
 80  void main(void)
 81  {
 82    char a[7];
 83    unsigned short i, j;
 84    unsigned int avr;
 85    unsigned long k;
 86    int count, avr_c;
 87
 88    WaitLoop();
 89
 90    Sci1Init(br19200);
 91    Rx1BuffClear();
 92    AdInit();
 93
 94    Sci1Puts("***** H8/3694F *****\n");
 95
 96    while(1){
 97      avr=0;
 98      /* 4 回 A/D 変換をして平均を取る */
 99      for(avr_c=0; avr_c<4; avr_c++){
100        AdStart();
101        i=AdRead(0);
102        avr=avr+i;
103      }
104      i=avr>>2;  /*  4 で割る */
105      Sci1Puts("X:");
106      /*  数値を文字列に変換  */
107      j=LongToStrDec((long)i,a);
108      /* 5 桁未満の場合には 0 で埋める */
109      for(count=0; count<5-j; count++){
110        Sci1Putchar('0');
111      }
112      Sci1Puts(a);
113      Sci1Puts("  Y:");
114      avr=0;
115      /* 4 回 A/D 変換をして平均を取る */
116      for(avr_c=0; avr_c<4; avr_c++){
117        AdStart();
118        i=AdRead(1);
119        avr=avr+i;
120      }
121      i=avr>>2;  /*  4 で割る */
122      /*  数値を文字列に変換  */
123      j=LongToStrDec((long)i,a);
124      /* 5 桁未満の場合には 0 で埋める */
125      for(count=0; count<5-j; count++){
126        Sci1Putchar('0');
127      }
128      Sci1Puts(a);
129      Sci1Putchar('\r');
130      /*  ウェイトループ  */
131      for(k=0; k<WAIT_COUNTER; k++){
132        ;
133      }
134    }
135  }
```

End Of List

実行結果

3 軸加速度センサの X 軸と Y 軸の出力結果が表示される。

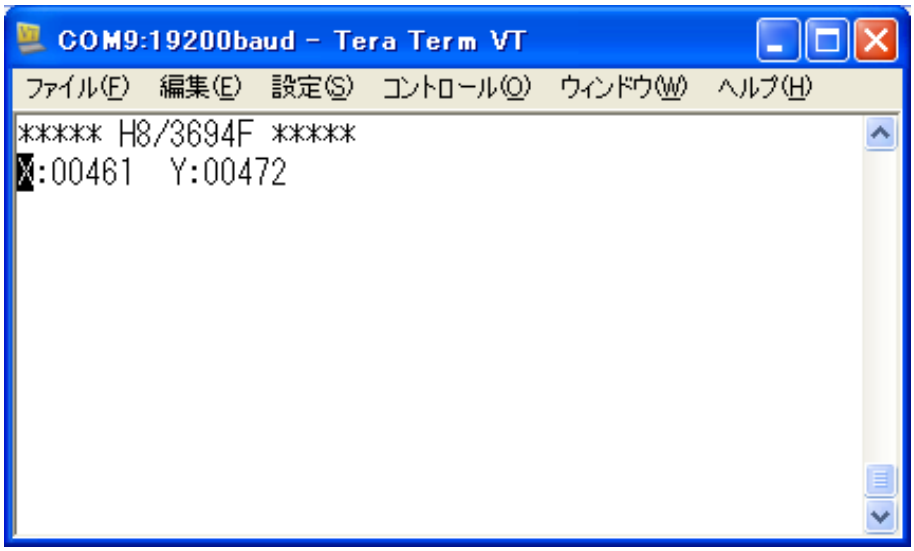


図 12.12　KXP84 の出力結果

課題 12.1.4　(提出) 電圧で表示

出力結果を電圧で表示してみましょう。

プロジェクト名：e09_ADC02

13 タイマWの利用(サーボモータ)

13.1 タイマW

タイマ W は 16 ビットタイマです。

外部イベントのカウントが可能です。また、タイマカウンタと 4 本のジェネラルレジスタのコンペアマッチ信号による任意のデューティ比のパルス出力もできます。この他、多機能タイマとして種々の応用が可能です。

ここでは、タイマ W を用いて歩行ロボット用サーボモータを駆動してみましょう。利用する機能は、タイマカウンタとジェネラルレジスタのコンペアマッチなどです。

まずは、ロボット用モータの説明をします。

13.1.1 ロボット用モータ

現在発売されているロボット用モータの制御方法は大きく二通りに大別されます。

ひとつはパルスによる位置指定であり、もうひとつはシリアルデータによる位置指定です。

パルスによる位置指定は、おおむね 20mS の周期で ON と OFF を繰り返し、 ON の幅が 0.5mS から 2.5mS 程度の範囲で位置の指定ができます。

このように、周期を一定にして ON の時間を変化させることによって (この比をデューティー比といいます) 情報を伝える方式を PWM(Pulse Width Modulation) といいます。

各社のモータでパルスと回転角度の対応は多少異なります。

シリアルデータによる位置指定は、シリアル通信を用いて位置データを送信するものです。このタイプのモータの特徴としては、位置だけではなく電流や温度などといった多種類の情報を送受信でき、可動範囲やトルクなどモータの設定をデータ送信によって変更することもできます。

二足歩行ロボットなどに利用されるモータ KRS788HV は、パルスにより位置指定をおこないます。

今回はこのモータを利用することにします。

図 13.1 に KRS788HV のおおよその軸位置とパルスの関係をあげておきます。このパルスを生成するた

めに、タイマ W を利用するわけです。

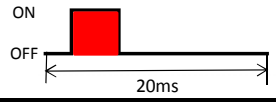
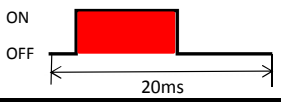
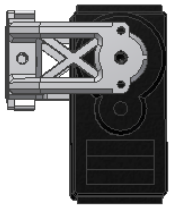
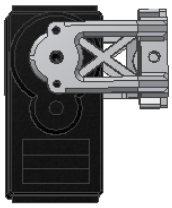
	反時計回りに約90度	中心	時計回りに約90度
パルスのON時間	約0.7mS	約1.5mS	約2.3mS
パルス波形			
モータの軸位置			

図 13.1　KRS788HV のパルスと軸位置の関係 (周期約 20mS)

13.1.2 関連レジスタ

タイマ W には以下のレジスタがあります。

- タイマモードレジスタ W（TMRW）
- タイマコントロールレジスタ W（TCRW）
- タイマインタラプトイネーブルレジスタ W（TIERW）
- タイマステータスレジスタ W（TSRW）
- タイマ I/O コントロールレジスタ 0（TIOR0）
- タイマ I/O コントロールレジスタ 1（TIOR1）
- タイマカウンタ（TCNT）
- ジェネラルレジスタ A（GRA）
- ジェネラルレジスタ B（GRB）
- ジェネラルレジスタ C（GRC）
- ジェネラルレジスタ D（GRD）

上記のレジスタは次のアドレスに割り振られています。

表 13.1 タイマ W 関連レジスタのアドレス

レジスタ名	アドレス
タイマモードレジスタ W（TMRW）	H'FF80
タイマコントロールレジスタ W（TCRW）	H'FF81
タイマインタラプトイネーブルレジスタ W（TIERW）	H'FF82
タイマステータスレジスタ W（TSRW）	H'FF83
タイマ I/O コントロールレジスタ 0（TIOR0）	H'FF84
タイマ I/O コントロールレジスタ 1（TIOR1）	H'FF85
タイマカウンタ（TCNT）	H'FF86
ジェネラルレジスタ A（GRA）	H'FF88
ジェネラルレジスタ B（GRB）	H'FF8A
ジェネラルレジスタ C（GRC）	H'FF8C
ジェネラルレジスタ D（GRD）	H'FF8E

タイマモードレジスタ W（TMRW）

TMRW はジェネラルレジスタの機能やタイマの出力モードなどを選択します。

表 13.2 タイマモードレジスタ W（TMRW）

ビット	ビット名	初期値	R/W	説 明
7	CTS	0	R/W	カウントスタート このビットが 0 のとき TCNT はカウント動作を停止し、 1 のときカウント動作を行います。
6	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
5	BUFEB	0	R/W	バッファ動作 B GRD の機能を選択します。 0 : インプットキャプチャ／アウトプットコンペアレジスタとして動作 1 : GRB のバッファレジスタとして動作。
4	BUFEA	0	R/W	バッファ動作 A GRC の機能を選択します。 0 : インプットキャプチャ／アウトプットコンペアレジスタとして動作 1 : GRA のバッファレジスタとして動作。
3	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
2	PWMD	0	R/W	PWM モード D FTIOD 端子の出力モードを選択します。 0 : 通常のアウトプットコンペア出力 1 : PWM 出力。
1	PWMC	0	R/W	PWM モード C FTIOC 端子の出力モードを選択します。 0 : 通常のアウトプットコンペア出力 1 : PWM 出力。
0	PWMB	0	R/W	PWM モード B FTIOB 端子の出力モードを選択します。 0 : 通常のアウトプットコンペア出力 1 : PWM 出力。

タイマコントロールレジスタ W（TCRW）

TCRW は TCNT のカウンタクロックの選択、カウンタのクリア条件やタイマの出力レベルの設定を選択します。

表 13.3　タイマコントロールレジスタ W（TCRW）

ビット	ビット名	初期値	R/W	説 明
7	CCRL	0	R/W	カウンタクリア このビットが 1 のときコンペアマッチ A によって TCNT がクリアされます。 0 のときは TCNT はフリーランニングカウンタとして動作します。
6 5 4	CKS2 CKS1 CKS0	0 0 0	R/W R/W R/W	クロックセレクト 2〜0 TCNT に入力するクロックを選択します。 000：内部クロック φ をカウント 001：内部クロック φ／2 をカウント 010：内部クロック φ／4 をカウント 011：内部クロック φ／8 をカウント 1XX 外部イベント (FTCI) の立ち上がりエッジをカウント 内部クロック φ を選択した場合、サブアクティブ、サブスリープモードで サブクロックをカウントします。
3	TOD	0	R/W	タイマ出力レベルセット D 最初のコンペアマッチ D が発生するまでの FTIOD 端子の出力値を設定します。 0：出力値 0* 1：出力値 1*
2	TOC	0	R/W	タイマ出力レベルセット C 最初のコンペアマッチ C が発生するまでの FTIOC 端子の出力値を設定します。 0：出力値 0* 1：出力値 1*
1	TOB	0	R/W	タイマ出力レベルセット B 最初のコンペアマッチ B が発生するまでの FTIOB 端子の出力値を設定します。 0：出力値 0* 1：出力値 1*
0	TOA	0	R/W	タイマ出力レベルセット A 最初のコンペアマッチ A が発生するまでの FTIOA 端子の出力値を設定します。 0：出力値 0* 1：出力値 1*

【注】 X：任意の値（0 でも 1 でも良い）
* 出力値は変更した時点で反映されます。

タイマインタラプトイネーブルレジスタ W（TIERW）

TIERW はタイマWの割り込み要求を制御します。

「イネーブルになる」とは「有効になる」という意味です。

表 13.4 タイマインタラプトイネーブルレジスタ W（TIERW）

ビット	ビット名	初期値	R/W	説 明
7	OVIE	0	R/W	タイマオーバーフロー割り込みイネーブル このビットが 1 のとき TSRW の OVF フラグによる割り込み要求 (FOVI) がイネーブルになります。
6	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
5	-	1	-	
4	-	1	-	
3	IMIED	0	R/W	インプットキャプチャ／コンペアマッチ割り込みイネーブル D このビットが 1 のとき TSRW の IMFD による割り込み要求（IMID）がイネーブルになります。
2	IMIEC	0	R/W	インプットキャプチャ／コンペアマッチ割り込みイネーブル C このビットが 1 のとき TSRW の IMFC による割り込み要求（IMIC）がイネーブルになります。
1	IMIEB	0	R/W	インプットキャプチャ／コンペアマッチ割り込みイネーブル B このビットが 1 のとき TSRW の IMFB による割り込み要求（IMIB）がイネーブルになります。
0	IMIEA	0	R/W	インプットキャプチャ／コンペアマッチ割り込みイネーブル A このビットが 1 のとき TSRW の IMFA による割り込み要求（IMIA）がイネーブルになります。

タイマステータスレジスタ W（TSRW）

TSRW は割り込み要求ステータスを表示します。

表 13.5　タイマステータスレジスタ W（TSRW）

ビット	ビット名	初期値	R/W	説 明
7	OVF	0	R/W	タイマオーバーフロー ［セット条件］ ・ TCNT が H'FFFF から H'0000 にオーバーフローしたとき ［クリア条件］ ・ 1 の状態をリードした後、0 をライトしたとき
6	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
5	-	1	-	
4	-	1	-	
3	IMFD	0	R/W	インプットキャプチャ／コンペアマッチフラグ D ［セット条件］ ・ GRD がアウトプットコンペアレジスタとして機能していて、 TCNT と一致したとき ・ GRD がインプットキャプチャレジスタとして機能していて、 インプットキャプチャ信号により TCNT の値が GRD に転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
2	IMFC	0	R/W	インプットキャプチャ／コンペアマッチフラグ C ［セット条件］ ・ GRC がアウトプットコンペアレジスタとして機能していて、 TCNT と一致したとき ・ GRC がインプットキャプチャレジスタとして機能していて、 インプットキャプチャ信号により TCNT の値が GRC に転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
1	IMFB	0	R/W	インプットキャプチャ／コンペアマッチフラグ B ［セット条件］ ・ GRB がアウトプットコンペアレジスタとして機能していて、 TCNT と一致したとき ・ GRB がインプットキャプチャレジスタとして機能していて、 インプットキャプチャ信号により TCNT の値が GRB に転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき
0	IMFA	0	R/W	インプットキャプチャ／コンペアマッチフラグ A ［セット条件］ ・ GRA がアウトプットコンペアレジスタとして機能していて、 TCNT と一致したとき ・ GRA がインプットキャプチャレジスタとして機能していて、 インプットキャプチャ信号により TCNT の値が GRA に転送されたとき ［クリア条件］ ・1 の状態をリードした後、0 をライトしたとき

タイマ I/O コントロールレジスタ 0（TIOR0）

TIOR0 は GRA、GRB および FTIOA、FTIOB 端子の機能を選択します。

表 13.6　タイマ I/O コントロールレジスタ 0（TIOR0）

ビット	ビット名	初期値	R/W	説 明
7	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
6	IOB2	0	R/W	I/O コントロール B2 GRB の機能を選択します。 0 : アウトプットコンペアレジスタとして機能 1 : インプットキャプチャレジスタとして機能
5	IOB1	0	R/W	I/O コントロール B1〜0 IOB2 = 0 のとき 00 : コンペアマッチによる端子出力禁止 01 : GRB のコンペアマッチで FTIOB 端子へ 0 出力 10 : GRB のコンペアマッチで FTIOB 端子へ 1 出力 11 : GRB のコンペアマッチで FTIOB 端子へトグル出力 IOB2 = 1 のとき 00 : FTIOB 端子の立ち上がりエッジで GRB へインプットキャプチャ 01 : FTIOB 端子の立ち下がりエッジで GRB へインプットキャプチャ 1X : FTIOB 端子の両エッジで GRB へインプットキャプチャ
4	IOB0	0	R/W	
3	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
2	IOA2	0	R/W	I/O コントロール A2 GRA の機能を選択します。 0 : アウトプットコンペアレジスタとして機能 1 : インプットキャプチャレジスタとして機能
1	IOA1	0	R/W	I/O コントロール A1〜0 IOA2 = 0 のとき 00 : コンペアマッチによる端子出力禁止 01 : GRA のコンペアマッチで FTIOA 端子へ 0 出力 10 : GRA のコンペアマッチで FTIOA 端子へ 1 出力 11 : GRA のコンペアマッチで FTIOA 端子へトグル出力 IOA2 = 1 のとき 00 : FTIOA 端子の立ち上がりエッジで GRA へインプットキャプチャ 01 : FTIOA 端子の立ち下がりエッジで GRA へインプットキャプチャ 1X : FTIOA 端子の両エッジで GRA へインプットキャプチャ
0	IOA0	0	R/W	

【注】 X : 任意の値（0 でも 1 でも良い）

タイマ I/O コントロールレジスタ 1（TIOR1）

TIOR1 は GRC、GRD および FTIOC、FTIOD 端子の機能を選択します。

表 13.7　タイマ I/O コントロールレジスタ 1（TIOR1）

ビット	ビット名	初期値	R/W	説 明
7	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
6	IOD2	0	R/W	I/O コントロール B2 GRD の機能を選択します。 0 : アウトプットコンペアレジスタとして機能 1 : インプットキャプチャレジスタとして機能
5	IOD1	0	R/W	I/O コントロール D1〜0 IOD2 = 0 のとき 00 : コンペアマッチによる端子出力禁止 01 : GRD のコンペアマッチで FTIOD 端子へ 0 出力 10 : GRD のコンペアマッチで FTIOD 端子へ 1 出力 11 : GRD のコンペアマッチで FTIOD 端子へトグル出力 IOD2 = 1 のとき 00 : FTIOD 端子の立ち上がりエッジで GRD へインプットキャプチャ 01 : FTIOD 端子の立ち下がりエッジで GRD へインプットキャプチャ 1X : FTIOD 端子の両エッジで GRD へインプットキャプチャ
4	IOD0	0	R/W	
3	-	1	-	リザーブビットです。読み出すと常に 1 が読み出されます。
2	IOC2	0	R/W	I/O コントロール C2 GRC の機能を選択します。 0 : アウトプットコンペアレジスタとして機能 1 : インプットキャプチャレジスタとして機能
1	IOC1	0	R/W	I/O コントロール C1〜0 IOA2 = 0 のとき 00 : コンペアマッチによる端子出力禁止 01 : GRC のコンペアマッチで FTIOC 端子へ 0 出力 10 : GRC のコンペアマッチで FTIOC 端子へ 1 出力 11 : GRC のコンペアマッチで FTIOC 端子へトグル出力 IOC2 = 1 のとき 00 : FTIOC 端子の立ち上がりエッジで GRC へインプットキャプチャ 01 : FTIOC 端子の立ち下がりエッジで GRC へインプットキャプチャ 1X : FTIOC 端子の両エッジで GRC へインプットキャプチャ
0	IOC0	0	R/W	

【注】 X : 任意の値（0 でも 1 でも良い）

タイマカウンタ（TCNT）

TCNT は 16 ビットのリード／ライト可能なアップカウンタです。入力クロックは TCRW の CKS2〜CKS0 のビットにより選択します。TCRW の CCLR の設定により GRA とのコンペアマッチにより H'0000 にクリアすることができます。TCNT が H'FFFF から H'0000 にオーバフローすると、TSRW の OVF が 1 にセットされます。このとき TIERW の OVIE がセットされていると割り込み要求を発生します。TCNT は 8 ビット単位のアクセスはできません。常に 16 ビット単位でアクセスしてください。TCNT の初期値は H'0000 です。

ジェネラルレジスタ A、B、C、D（GRA、GRB、GRC、GRD）

インプットキャプチャレジスタとしても使用できます。機能の切り替えは、TIOR0、TIOR1 により行います。

アウトプットコンペアレジスタに設定されたジェネラルレジスタの値は TCNT の値と常に比較されます。両者が一致（コンペアマッチ）すると、TSRW の IMFA～IMFD フラグが 1 にセットされます。このとき TIERW の IMIEA～IMIED がセットされていると割り込み要求を発生します。また TIOR によりコンペアマッチ出力を設定することができます。

GRA～GRD は 8 ビット単位のアクセスはできません。常に 16 ビット単位でアクセスしてください。GRA～GRD の初期値は H'FFFF です。

タイマ W のレジスタ定義 (iodefine.h の一部を抜粋)

以下にタイマ W のレジスタ定義を示します。

タイマ W のレジスタ定義 (iodefine.h の一部を抜粋)

```
struct st_tw {                                          /* struct TW    */
             union {                                    /* TMRW         */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char CTS  :1;        /*    CTS       */
                          unsigned char      :1;        /*              */
                          unsigned char BUFEB:1;        /*    BUFEB     */
                          unsigned char BUFEA:1;        /*    BUFEA     */
                          unsigned char      :1;        /*              */
                          unsigned char PWMD :1;        /*    PWMD      */
                          unsigned char PWMC :1;        /*    PWMC      */
                          unsigned char PWMB :1;        /*    PWMB      */
                          }      BIT;                   /*              */
                   }             TMRW;                  /*              */
             union {                                    /* TCRW         */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char CCLR:1;         /*    CCLR      */
                          unsigned char CKS :3;         /*    CKS       */
                          unsigned char TOD :1;         /*    TOD       */
                          unsigned char TOC :1;         /*    TOC       */
                          unsigned char TOB :1;         /*    TOB       */
                          unsigned char TOA :1;         /*    TOA       */
                          }      BIT;                   /*              */
                   }             TCRW;                  /*              */
             union {                                    /* TIERW        */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char OVIE :1;        /*    OVIE      */
                          unsigned char      :3;        /*              */
                          unsigned char IMIED:1;        /*    IMIED     */
                          unsigned char IMIEC:1;        /*    IMIEC     */
                          unsigned char IMIEB:1;        /*    IMIEB     */
                          unsigned char IMIEA:1;        /*    IMIEA     */
                          }      BIT;                   /*              */
                   }             TIERW;                 /*              */
             union {                                    /* TSRW         */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char OVF :1;         /*    OVF       */
                          unsigned char     :3;         /*              */
                          unsigned char IMFD:1;         /*    IMFD      */
                          unsigned char IMFC:1;         /*    IMFC      */
                          unsigned char IMFB:1;         /*    IMFB      */
                          unsigned char IMFA:1;         /*    IMFA      */
                          }      BIT;                   /*              */
                   }             TSRW;                  /*              */
             union {                                    /* TIOR0        */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char    :1;          /*              */
                          unsigned char IOB:3;          /*    IOB       */
                          unsigned char    :1;          /*              */
                          unsigned char IOA:3;          /*    IOA       */
                          }      BIT;                   /*              */
                   }             TIOR0;                 /*              */
             union {                                    /* TIOR1        */
                   unsigned char BYTE;                  /*  Byte Access */
                   struct {                             /*  Bit  Access */
                          unsigned char    :1;          /*              */
                          unsigned char IOD:3;          /*    IOD       */
                          unsigned char    :1;          /*              */
                          unsigned char IOC:3;          /*    IOC       */
                          }      BIT;                   /*              */
                   }             TIOR1;                 /*              */
             unsigned int        TCNT;                  /* TCNT         */
             unsigned int        GRA;                   /* GRA          */
             unsigned int        GRB;                   /* GRB          */
             unsigned int        GRC;                   /* GRC          */
             unsigned int        GRD;                   /* GRD          */
};                                                      /*              */

.....

#define TW      (*(volatile struct st_tw    *)0xFF80)   /* TW    Address*/
```

13.1.3 基本的なプログラム (PWM モード、割込みなし)

まずは、モータを稼動範囲のほぼ中央に動かすプログラムを作りましょう。

タイマ W を用いて PWM を出力する方法はいくつかありますが、PWM モードを用いて FTIOB 端子に出力することにします。この方法では、タイマの設定を行えばあとは自動的に PWM を出力してくれるので、プログラムが短くてすみます。

回路図

今回のサンプルプログラムのための回路を図 13.2 に示します。

モータ KRS788HV には 3 本のコードが出ています。

黒はグランドに (図 13.2 では、CN-3P の 1 番ピン)、赤は電源に (CN-3P の 2 番ピン)、白は PWM の信号を出力する端子 (CN-3P の 3 番ピン) につなぎます。今回は白いコードを P82/FTIOB につなぎます。

モータ KRS788HV は電源電圧が 9V から 12V です。マイコンとは別の電源を用意してください。

なお、モータとマイコンのグランドは必ずつないでください。忘れると正しく動作しません。

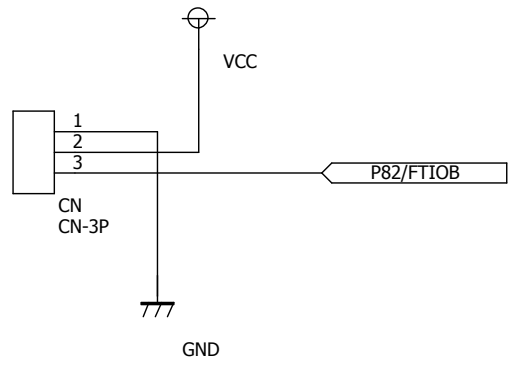


図 13.2 モータ回路図

クロックの分周とジェネラルレジスタ A、B（GRA、GRB）に設定する値を決めなくてはなりません。

タイマ W は 16 ビットカウンタなので、分周無しだと約 3.2768ms カウントできます。20ms をカウントするには、8 分周以上分周する必要があります。

8 分周でで 26.2144ms までカウント可能です。

前と同様な計算方法でまとめておくと、以下のようになります。

ジェネラルレジスタの値を計算 (20MHz を 8 分周したときに、20m 秒、1.5m 秒をカウント)

20MHz を 8 分周すると、

$$\frac{20000000}{8}(Hz)$$

になります。
この時のパルスの周期は、周波数の逆数を取って、

$$\frac{8}{20000000}(s)$$

です。
20m 秒カウントするためのジェネラルレジスタの値を x とすると、

$$\frac{8}{20000000} \times x = 0.02$$

となる x を求めればよいわけです。
結果は、

$$x = 50000$$

となります。
同様にして、1.5m 秒カウントするためのジェネラルレジスタの値を x とすると、

$$\frac{8}{20000000} \times x = 0.0015$$

となる x を求めればよいわけです。
結果は、

$$x = 3750$$

となります。

今回のプログラムでは、ジェネラルレジスタ A(GRA) を周期の設定に、ジェネラルレジスタ B(GRB) を ON 時間の設定に利用することにします。

では、モータの角度を稼動範囲のほぼ中央にするサンプルプログラムを示します。

07_TMRW01.c

```
 1  /*********************************************************************/
 2  /*                                                                   */
 3  /*  FILE        :07_TMRW01.c                                         */
 4  /*  DATE        :Sat, Feb 15, 2014                                   */
 5  /*  DESCRIPTION :PWM 出力 (PB2/FTIOB)                                 */
 6  /*  CPU TYPE    :H8/3694F                                            */
 7  /*                                                                   */
 8  /*  PB2/FTIOB                                                        */
 9  /*                                                                   */
10  /*********************************************************************/
11
12  #include"iodefine.h"
13
14  void main(void)
15  {
16    TW.TMRW.BYTE=0x49;  /* FTIOB 端子を PWM 出力に設定 */
17    TW.TCRW.BYTE=0xB2;  /* 8 分周 FTIOB の初期値は 1 */
18    TW.TIERW.BYTE=0x70; /* 割り込みを利用しない */
```

```
19    TW.TCNT=0x0000;      /* TCN の初期化 */
20    TW.GRA=50000;        /* 20mS */
21    TW.GRB=3750;         /* 1.5mS */
22    TW.TMRW.BIT.CTS=1;   /* TCNT カウンタスタート */
23    while(1){
24      ;
25    }
26  }
```

End Of List

実行結果

オシロスコープで測定して図 13.3 のように、周期が約 20ms(周波数約 50Hz)、ON の時間が約 1.5ms になっていることを確かめましょう。

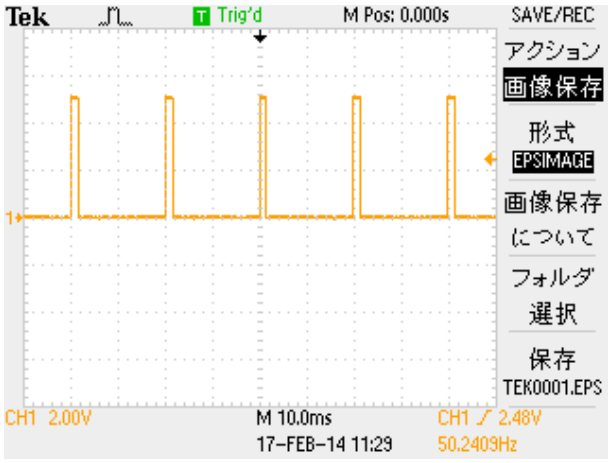


図 13.3 パルス出力結果

また、モータをつないで、稼動範囲の中間あたりまで回転することを確認してください。マイコンの電源を切り、モータを回して、どのような角度にしても、プログラムを動かすと、稼動範囲の中間あたりまで回転することも確認してください。

プログラム解説 (07_TMRW01.c)

```
16    TW.TMRW.BYTE=0x49;  /* FTIOB 端子を PWM 出力に設定 */
```

16 行目は、タイマモードレジスタ W（TMRW）の設定です。

P.337 の表 13.2 から分かるように、ここでは PWM モード B のみを設定しています。これにより FTIOB 端子を PWM 出力に設定することになります。

```
17    TW.TCRW.BYTE=0xB2;  /* 8 分周 FTIOB の初期値は 1 */
```

16 行目は、タイマコントロールレジスタ W（TCRW）の設定です。

P.338 の表 13.3 から分かるように、コンペアマッチ A により TCNT がクリアされます。

分周は内部クロックを 8 分周する設定です。

また、最初のコンペアマッチ B が発生するまで、FTIOB 端子の出力は 1 に設定しました。

```
18    TW.TIERW.BYTE=0x70; /* 割り込みを利用しない */
```

18 行目はタイマインタラプトイネーブルレジスタ W（TIERW）の設定です。

リザーブビット以外は 0 に設定しています。

割込みを設定していません。

```
19    TW.TCNT=0x0000;     /* TCN の初期化 */
20    TW.GRA=50000;       /* 20mS */
21    TW.GRB=3750;        /* 1.5mS */
```

19 行目ではタイマカウンタ（TCNT）を 0 に初期化しています。この操作は、なくてもかまいません。

20 行目では、ジェネラルレジスタ A(GRA) を周期の設定用いています。上で計算したように、50000 を設定すると 20ms になります。

21 行目は、ジェネラルレジスタ B(GRB) の設定です。ON 時間の設定になります。3750 で 1.5ms になります。

```
22    TW.TMRW.BIT.CTS=1;  /* TCNT カウンタスタート */
```

タイマ W をスタートさせています。

ここまでの設定だけで、FTIOB 端子から PWM が出力されることに注意してください。

✍ 課題 13.1.1　(提出) モータの角度を変更してみる

上のサンプルでは PWM の ON 時間は 1.5ms でしたが、これを変更して、ON 時間が 1.0ms の PWM を出力するプログラムを作ってみましょう。

さらに、ON 時間が 2.0ms の PWM を出力するように変更してください。

プロジェクト名：e07_TMRW01

13.1.4 モータを動かす (PWM モード、割込みなし)

前のサンプルでは、モータを指定の角度に動かしました。

ここでは、2 秒ごとに角度を変えるプログラムを作ってみましょう。

PWM の ON の時間を変えるには、ジェネラルレジスタ B(GRB) の値を変更すればよいのです。

また、コンペアマッチ A は 20m 秒ごとに起こるので、これを 100 回カウントすれば 2 秒になります。

回路図

今回のサンプルプログラムのための回路は図 13.2 と同じです。

以下に、このアルゴリズムでプログラムしたサンプルを示します。

07_TMRW02.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :07_TMRW02.c                                            */
 4  /*  DESCRIPTION :PWM 出力 (PB2/FTIOB) モータを動かす (1.0ms,1.5ms,2.0ms)   */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*                                                                      */
 7  /*  PB2/FTIOB                                                           */
 8  /*                                                                      */
 9  /************************************************************************/
10
11  #include"iodefine.h"
12
13  void main(void)
14  {
15    short duty[3]={2500, 3750, 5000}, i=0, j=0;
16
17    TW.TMRW.BYTE=0x49;       /* FTIOB 端子を PWM 出力に設定 */
18    TW.TCRW.BYTE=0xB2;       /* 8 分周 FTIOB の初期値は 1 */
19    TW.TIERW.BYTE=0x70;      /* 割り込みを利用しない */
20    TW.TCNT=0x0000;          /* TCN の初期化 */
21    TW.GRA=50000;            /* 20mS */
22    TW.GRB=duty[j];
23    TW.TMRW.BIT.CTS=1;       /* TCNT カウンタスタート */
24
25    while(1){
26      while(TW.TSRW.BIT.IMFA!=1){
27        ;
28      }
29      TW.TSRW.BIT.IMFA=0;
30      i++;
31      if(i>100){   /* 2s をカウント */
32        i=0;
33        j++;
34        if(j>2){
35          j=0;
36        }
37        TW.GRB=duty[j];
38      }
39    }
40  }
```

End Of List

実行結果

2 秒ごとにモータの角度が変わる。

プログラム解説 (07_TMRW02.c)

ここでは、前のサンプルと異なる部分だけ説明します。

```
15    short duty[3]={2500, 3750, 5000}, i=0, j=0;
```

配列 duty は 2 秒毎のジェネラルレジスタ B(GRB) の値を格納します。今回は ON 時間が 1.0ms、1.5ms、2.0ms になるように設定しました。

変数 i はコンペアマッチ A の回数を数えるのに用います。

変数 j は何番目のデータを利用しているかを格納します。

```
22    TW.GRB=duty[j];
```

ジェネラルレジスタ B(GRB) に値を代入しています。ここでは j=0 ですので、TW.GRB=duty[0] と書くのと同じです。

TW.GRB には 2500 が代入され、PWM の ON の時間は 1.0ms になります。

```
26    while(TW.TSRW.BIT.IMFA!=1){
27      ;
28    }
29    TW.TSRW.BIT.IMFA=0;
```

TW.TSRW.BIT.IMFA はタイマステータスレジスタ W（TSRW）のインプットキャプチャ／コンペアマッチフラグ A です (P.340 表 13.5 参照)。

GRA がアウトプットコンペアレジスタとして機能していて、TCNT と一致したときに、この値が 1 になります。

したがって、26 行目から 28 行目まではコンペアマッチが起こるのを待っています。コンペアマッチが起こると、TW.TSRW.BIT.IMFA は 1 になり、while ループを抜けて 29 行目に移ります。ここでは、インプットキャプチャ／コンペアマッチフラグ A を 0 に戻しています。次のコンペアマッチを確認するためです。

```
30    i++;
```

コンペアマッチのカウントを 1 増やしています。20ms 毎にコンペアマッチ A は発生することに注意してください。

```
31    if(i>100){   /* 2s をカウント */
32      i=0;
33      j++;
34      if(j>2){
35        j=0;
36      }
37      TW.GRB=duty[j];
38    }
```

31 行目から 38 行目までは、コンペアマッチ A が 100 回行われた場合の処理です。20ms のコンペアマッチ A が 100 回で 2 秒になります。

32 行目で i を 0 にクリアし、次の 100 回をカウントするための設定をします。

33 行目は、変数 j の値を 1 増やしています。j は配列 duty の要素番号を表しているので、0 から 2 までです。2 より大きくなったら 0 に戻します (34 行目から 36 行目まで)。

37 行目で、ジェネラルレジスタ B(GRB) の値を更新しています。

これによって、2 秒ごとに ON 時間が 1.0ms、1.5ms、2.0ms の PWM が出力されます。

✍ 課題 13.1.2 (提出) 1 秒ごとにモータの角度を変える

上記のプログラムでは、2 秒ごとに ON 時間が 1.0ms、1.5ms、2.0ms と変化し、再度 1.0ms に戻りました。

これを変更して、1 秒ごとに ON 時間が 1.0ms、1.5ms、2.0ms と変化し、また 1.5ms、1.0ms と戻り、1.5ms、2.0ms と繰り返すようにプログラムしてください。

プロジェクト名：e07_TMRW02

13.1.5 モータを 3 個動かす (PWM モード、割込みなし)

H8/3694F には、4 本のジェネラルレジスタがあり、PWM モードを用いて、3 本の独立した PWM を出力することができます。

周期をジェネラルレジスタ A(GRA) で設定し、三つの PWM の ON 時間をジェネラルレジスタ B(GRB)、ジェネラルレジスタ C(GRC)、ジェネラルレジスタ D(GRD) で設定します。

このとき、PWM は FTIOB、FTIOC、FTIOD のそれぞれの端子から出力されます。

回路図

今回のサンプルプログラムのための回路を図 13.4 に示します。

三つのモータの信号線は、FTIOB、FTIOC、FTIOD につないでください。

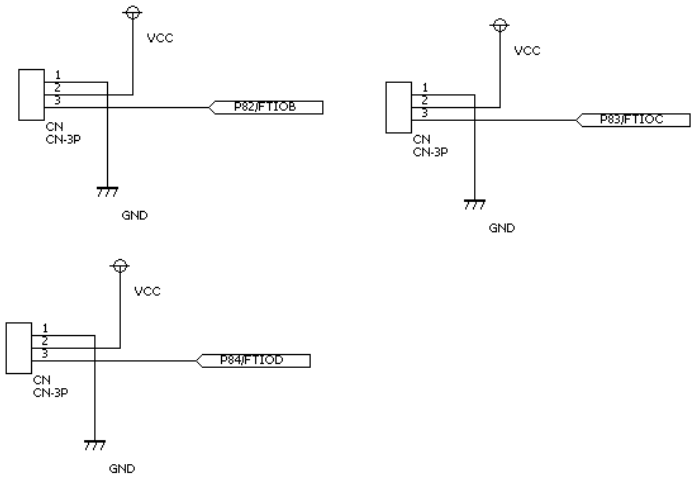


図 13.4　モータ回路図

以下に、PWM モードを用いて、三つのサーボモータを駆動するプログラムを示します。

07_TMRW03.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :07_TMRW03.c                                            */
/*  DESCRIPTION :モータを 3 個動かす                                    */
/*  CPU TYPE    :H8/3694F                                               */
/*                                                                      */
/*  PB2/FTIOB                                                           */
/*  PB3/FTIOC                                                           */
/*  PB4/FTIOD                                                           */
/*                                                                      */
/************************************************************************/

#include"iodefine.h"

#define SRV_CH_NUM 3 /* モータ数 */
#define MOTION_NUM 4 /* モーション数 */

void main(void)
{
  short duty[MOTION_NUM][SRV_CH_NUM]=
    {{2500, 3750, 5000},
     {3750, 5000, 3750},
     {5000, 3750, 2500},
     {3750, 2500, 3750}};
  int i=0, j=0;

  TW.TMRW.BYTE=0x4F;      /* FTIOB-D 端子を PWM 出力に設定 */
  TW.TCRW.BYTE=0xBE;      /* 8 分周 FTIOB の初期値は 1 */
  TW.TIERW.BYTE=0x70;     /* 割り込みを利用しない */
  TW.TCNT=0x0000;         /* TCN の初期化 */
  TW.GRA=50000;            /* 20mS */
  TW.GRB=duty[j][0];
  TW.GRC=duty[j][1];
  TW.GRD=duty[j][2];
  TW.TMRW.BIT.CTS=1;      /* TCNT カウンタスタート */

  while(1){
    while(TW.TSRW.BIT.IMFA!=1){
      ;
    }
    TW.TSRW.BIT.IMFA=0;
    i++;
    if(i>100){
      i=0;
      j++;
      if(j>=MOTION_NUM){
        j=0;
```

```
48          }
49          TW.GRB=duty[j][0];
50          TW.GRC=duty[j][1];
51          TW.GRD=duty[j][2];
52        }
53      }
54  }
```

End Of List

実行結果

FTIOB、FTIOC、FTIOD に接続した三つのモータの角度が 2 秒ごとに変わる。

プログラム解説 (07_TMRW03.c)

今回も、前のサンプルと異なる部分だけ説明します。

```
15    #define SRV_CH_NUM 3 /* モータ数 */
16    #define MOTION_NUM 4 /* モーション数 */
```

15 行目ではモータの数に、16 行目では各モータのデータの数に名前をつけています。

```
20    short duty[MOTION_NUM][SRV_CH_NUM]=
21      {{2500, 3750, 5000},
22       {3750, 5000, 3750},
23       {5000, 3750, 2500},
24       {3750, 2500, 3750}};
```

三つのモータの角度データは、二次元配列に格納しました。

21 行目から 24 行目までは、各行が三つのモータに対するデータです。たとえば、21 行目のデータは、2500 が FTIOB 端子につないだモータのデータ、3750 が FTIOC 端子につないだモータのデータ、5000 が FTIOD 端子につないだモータのデータです。21 行目は、2 秒後の書くモータのデータになります。

今回のプログラムとは異なる順番でデータを扱うことも可能です。たとえば duty[SRV_CH_NUM][MOTION_NUM] と書くことで、各行がひとつのモータに対するデータになるような書き方もできるのです。

しかし、歩行ロボットのモーションを扱うには、ポーズごとのデータをひとまとめにしておいたほうが、データが扱いやすいためこのようにしました。

なお、21 行目の三つのデータは、duty[0][0]、duty[0][1]、duty[0][2] で指定できることに注意してください。

```
27    TW.TMRW.BYTE=0x4F;      /* FTIOB-D 端子を PWM 出力に設定 */
```

FTIOB-D 端子を PWM 出力に設定しています。

タイマモードレジスタ W（TMRW）に関しては、P.337 の表 13.2 を確認してください。

```
32    TW.GRB=duty[j][0];
33    TW.GRC=duty[j][1];
34    TW.GRD=duty[j][2];
```

ジェネラルレジスタ B(GRB)、ジェネラルレジスタ C(GRC)、ジェネラルレジスタ D(GRD) に値を代入しています。

ここでは、j は 0 ですが、j の値を変更することで、データを変更することができることに注意してください。

49 行目から 51 行目では、j が 0 以外のときも含めた書き換えを行っています。

✍ 課題 13.1.3　(提出) モータが順番に動作する

三つのモータが順番に動作するプログラムを作ってみましょう。

初期状態は三つのモータは、中心にあるとします。

最初に FTIOB 端子につないだモータが左右に振れて、中心に戻ります。次に、FTIOC 端子につないだモータが左右に振れて、中心に戻ります。次に、FTIOD 端子につないだモータが左右に振れて、中心に戻ります。FTIOB 端子につないだモータの動作に戻ります。

プロジェクト名：e07_TMRW03

13.1.6 ソフト的に割り振り (8 個駆動)

前回は、PWM モードを用いて、3 本の独立した PWM を出力しました。

これによって、サーボモータを三つ制御できることを確認しました。

しかし、歩行ロボットなどを作ろうとしたら、サーボモータは 3 個では足りません。

四足歩行ロボットでしたら、8 個は必要でしょうし、二足歩行ロボットでしたら、20 個近くは制御したいところです。

これだけ多くのサーボモータを制御するにはどうしたらよいのでしょうか。

幸いなことに、ロボット用モータを動かすための PWM というのは、ON の時間が最大でも 2.3ms で、周期 20ms の大半は OFF の状態です。

端子への出力は一度設定してしまえば、書き変えない限り保持されたままです。

そこで、ある端子に 1 を出力し、モータの角度に対応した時間で 0 に変更します。その後 2.5ms で出力先を変更して、次の端子でも同様に 1 をある時間出力して、0 に切り替えます。その後、2.5ms 経過した時点で出力先を変更します。こうして八つの端子に次々に必要な時間だけ 1 を出力していくことによって、20ms の周期をもつ PWM が 8 本作りさせることになります (図 13.5 参照)。

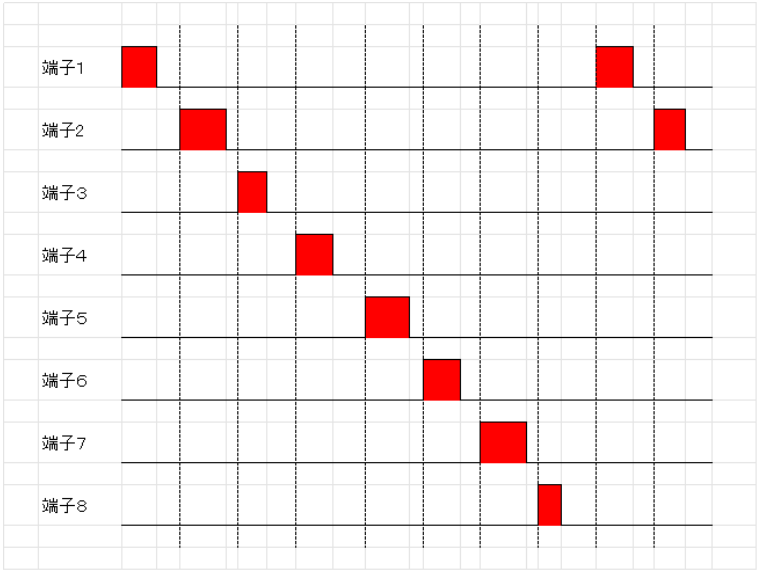


図 13.5 PWM を 8 本生成する

切り替え時間に必要なのは、 最悪で 0.2mS になります。

0.2mS 以内に出力ポートを切り替えられれば、 8 個のモータが駆動できるというわけです。

なお、出力の切り替えを、出力が 0 の時に行っていることに注意してください。

ここでは、この割り振りをプログラムでやることにしましょう。

コンペアマッチによる割込み関数を用いることにします。今回の割込み関数は、2.5ms ごとに実行する必要があることに注意してください。

回路図

今回のサンプルプログラムのための回路を図 13.6 に示します。

8 個のモータの信号線は、P50 から P57 につないでください。

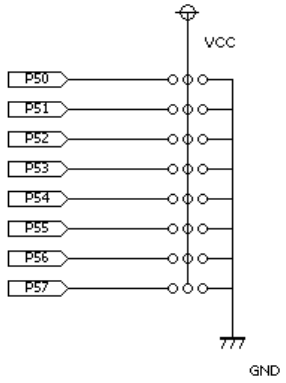


図 13.6　モータ回路図

今回はタイマ W のコンペアマッチ割込みを利用します。割込み関数は、main 関数と同じファイル (07_TMRW04.c) に記述しますので、intprg.c の INT_TimerW をコメントアウトしてください。

intprg.c の INT_TimerW をコメントアウト

```
// vector 21 Timer W
//__interrupt(vect=21) void INT_TimerW(void) {/* sleep(); */}
// vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {/* sleep(); */}
```

以下にサンプルプログラムを示します。今回の PWM は P50 から P57 までの 8 端子に出力します。

07_TMRW04.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :07_TMRW04.c                                            */
 4  /*  DESCRIPTION :ソフト的に割り振り (8 個駆動)                          */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*  モータ 0 P50                                                        */
 7  /*  モータ 1 P51                                                        */
 8  /*  モータ 2 P52                                                        */
 9  /*  モータ 3 P53                                                        */
10  /*  モータ 4 P54                                                        */
11  /*  モータ 5 P55                                                        */
12  /*  モータ 6 P56                                                        */
13  /*  モータ 7 P57                                                        */
14  /************************************************************************/
15
16  #include "iodefine.h"
17  #include<machine.h>
18
19  #define PWM_MAX_DUTY 12000-1 //2.4mS
20  #define PWM_MIN_DUTY  2000-1 //0.4mS
21  #define PWM_PERIOD   12500-1 //2.5mS
22
23  #define SRV_CH_NUM 8 /* モータ数 */
24  #define MOTION_NUM 4 /* モーション数 */
25
26  int srv_ch;     /* PWM の出力先 */
27  int srv_flg;    /* 20ms をカウント */
28  unsigned short pwm_duty[SRV_CH_NUM]; /* PWM のデューティのデータ */
29
30  /*******************************
31     PWM 出力端子を初期化
```

```
 32  ******************************/
 33  void PwmInit(void)
 34  {
 35    srv_ch=0;
 36    srv_flg=0;
 37    set_imask_ccr(1);
 38    TW.TMRW.BYTE=0x48;  /* FTIO 端子は使用しない */
 39    TW.TCRW.BYTE=0xA0;  /* 内部クロックの 1/4, 出力なし */
 40    TW.TIERW.BYTE=0x73; /* A,B の割り込みを可能にする */
 41    TW.TSRW.BYTE=0x70;  /* 割り込みフラグをクリア */
 42    TW.TCNT=0x0000;     /* TCN の初期化 */
 43    TW.GRA=PWM_PERIOD;  /* 周期 (2.5mS) */
 44    TW.GRB=7500;        /* 1.5mS */
 45    TW.TMRW.BIT.CTS=1;  /* TCNT カウンタスタート */
 46    set_imask_ccr(0);
 47
 48    IO.PCR5=0xFF;
 49    IO.PDR5.BYTE=0x00;
 50  }
 51
 52  /*********************************
 53    サーボ動作開始
 54   *********************************/
 55  void PwmStart(void)
 56  {
 57    TW.TMRW.BIT.CTS=1;
 58  }
 59
 60  /*********************************
 61    サーボ動作停止
 62   *********************************/
 63  void PwmStop(void)
 64  {
 65    TW.TMRW.BIT.CTS=0;
 66  }
 67
 68  /*********************************
 69    デューティの設定
 70   *********************************/
 71  void PwmSetDuty(int ch, unsigned short duty)
 72  {
 73    /* サーボチャンネルが範囲外なら無視 */
 74    if(ch >= SRV_CH_NUM){
 75      return ;
 76    }
 77
 78    if(duty < PWM_MIN_DUTY){          /* デューティを最小値以上に限定 */
 79      duty = PWM_MIN_DUTY;
 80    }else if(duty > PWM_MAX_DUTY){    /* デューティを最大値以下に限定 */
 81      duty = PWM_MAX_DUTY;
 82    }
 83    /* デューティを設定 */
 84    pwm_duty[ch] = duty;
 85  }
 86
 87  void main(void)
 88  {
 89    int i, j=0;
 90  /* PWM のデューティのデータ */
 91  unsigned short pwm_data[MOTION_NUM][SRV_CH_NUM]=
 92      {{ 5000,  7500, 10000,  7500, 7500, 10000,  7500,  5000},
 93       { 7500,  5000,  7500, 10000, 5000,  7500, 10000,  7500},
 94       {10000,  7500,  5000,  7500, 7500,  5000,  7500, 10000},
 95       { 5000, 10000,  7500,  5000,10000,  7500,  5000,  7500}};
 96
 97    PwmInit(); /* PWM の初期化 */
 98
 99    while(1){
100      /* 次のモーションデータを設定 */
101      for(i=0; i<SRV_CH_NUM; i++){
102        PwmSetDuty(i, pwm_data[j][i]);
103      }
104      while(srv_flg<100){
105        ;
106      }
107      srv_flg=0;
108      j++;
109      if(j>=MOTION_NUM){
110        j=0;
111      }
112    }
113  }
114
115  /*********************************
116    割り込み関数
117   *********************************/
118  __interrupt(vect=21) void INT_TimerW(void)
119  {
120    /* デューティのコンペアマッチ */
121    if(TW.TSRW.BIT.IMFB==1){
122      switch(srv_ch){
```

```
123         case 0:
124           IO.PDR5.BIT.B0=0;
125           break;
126         case 1:
127           IO.PDR5.BIT.B1=0;
128           break;
129         case 2:
130           IO.PDR5.BIT.B2=0;
131           break;
132         case 3:
133           IO.PDR5.BIT.B3=0;
134           break;
135         case 4:
136           IO.PDR5.BIT.B4=0;
137           break;
138         case 5:
139           IO.PDR5.BIT.B5=0;
140           break;
141         case 6:
142           IO.PDR5.BIT.B6=0;
143           break;
144         case 7:
145           IO.PDR5.BIT.B7=0;
146           break;
147       }
148       TW.TSRW.BIT.IMFB=0;
149     }
150     /* 2.5mS ごとのコンペアマッチ */
151     if(TW.TSRW.BIT.IMFA==1){
152       switch(srv_ch){
153         case 0:
154           IO.PDR5.BIT.B1=1;
155           break;
156         case 1:
157           IO.PDR5.BIT.B2=1;
158           break;
159         case 2:
160           IO.PDR5.BIT.B3=1;
161           break;
162         case 3:
163           IO.PDR5.BIT.B4=1;
164           break;
165         case 4:
166           IO.PDR5.BIT.B5=1;
167           break;
168         case 5:
169           IO.PDR5.BIT.B6=1;
170           break;
171         case 6:
172           IO.PDR5.BIT.B7=1;
173           break;
174         case 7:
175           IO.PDR5.BIT.B0=1;
176           break;
177       }
178         TW.TSRW.BIT.IMFA=0;
179         srv_ch++;
180         if(srv_ch>= SRV_CH_NUM){
181           srv_ch=0; /* モータ出力先を 0 に戻す */
182           srv_flg++;
183         }
184         TW.GRB=pwm_duty[srv_ch];
185     }
186   }
```

End Of List

実行結果

P50 から P57 に接続した 8 個のモータの角度が 2 秒ごとに変わる。

13.1.7 サーボモータを 8 個動かす (IC で割り振り)

前回は、割込み関数を用いて 8 本の独立した PWM を出力しました。

これによって、サーボモータを 8 個制御できることを確認しました。

しかし、8 個のロボット用モータを制御するために、8 本の端子が必要でした。20 個のモータを制御しようとしたら 20 本の端子が必要になります。

また、ソフト的に割り振りをしていたのでは処理スピードが問題になることもあります。後ほど紹介する「モーション作成ソフト」を用いた場合には、ソフト的に割り振っていたのでは処理が間に合わないこともあるのです。

一般に、ソフトウェアで処理するよりもハードウェア的に処理するほうが早いようです。また、回路によってはマイコンの出力端子を少なくすることも可能です。

ここでは、TC4051B を用いて、1 本の PWM を 8 本に振り分けるということをしてみましょう。

タイマ W の PWM モードを用いて FTIOB 端子に PWM を出力します。これを 2.5ms ごとに 8 本に振り分けるわけです。

図示すると、以下のようになります。

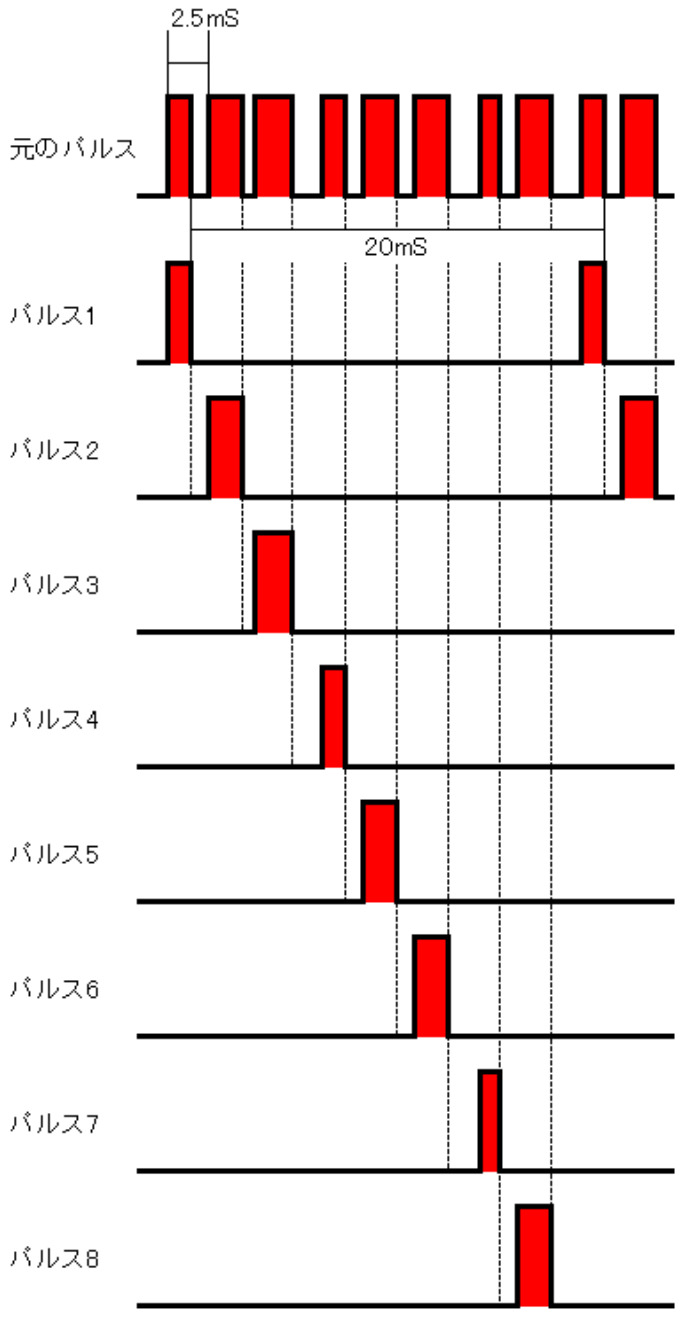


図 13.7　1 本の PWM を 8 本に振り分ける

図 13.7 で、 最初のパルスは 2.5mS の周期で ON 時間を変更しながらパルスを作っています。

もととなるパルス (FTIOB 端子に出力) を 2.5mS ごとに出力先を切り替えることで、ロボット用モータを 8 本出力することが可能になるわけです。

このとき、 切り替え時間に必要なのは、 最悪で 0.2mS になります。

0.2mS 以内に出力ポートを切り替えられれば、 8 個のモータが駆動できるのです。

なお、出力の切り替えを、出力が 0 の時に行っていることに注意してください。

TC4051B

上で説明した、パルスの振り分けをおこなうのが TC4051B という IC です。

TC4051B のピン接続図と真理値表を載せておきます。

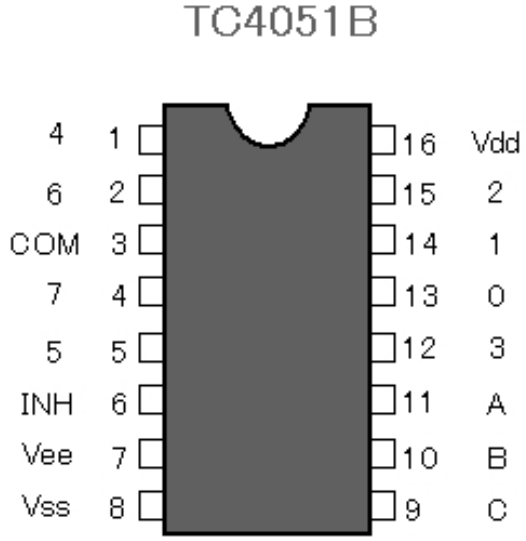


図 13.8 TC4051B のピン接続図

表 13.8 TC4051B の真理値表

コントロールチャンネル				出力チャンネル
INH	A	B	C	TC4051B
0	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	0	3
0	0	0	0	4
0	0	0	0	5
0	0	0	0	6
0	0	0	0	7
1	*	*	*	None

上の説明に対応させるならば、 COM にもととなるパルスを入力し、 A,B,C の値を変えることによって、 出力先を 0 から 7 までに切り替えることができるわけです。

表 13.8 から分かるように、 A=0,B=0,C=0 の時には COM から入力されたパルスは端子 0 から出力さ

れます。

A=1,B=0,C=0 の時にはパルスは端子 1 から、 A=0,B=1,C=0 の時にはパルスは端子 2 からというふうになります。

COM にたとえば FTIOB 端子をつなぎ、0 から 7 までの端子にサーボモータをつなぐことで、8 個のサーボモータを制御できます。

このとき、マイコンにつなぐ端子は COM と A、B、C の 4 本ですみます。

FTIOC 端子、FTIOD 端子にも TC4051BTF をつなげば、最大 24 本の PWM を 6 本の端子で制御できることになります (A、B、C は FTIOB 端子、FTIOC 端子、FTIOD 端子で共通にできます)。

回路図

今回のサンプルプログラムのための回路を図 13.9 に示します。

8 個のモータの信号線は、TC4051BP の 8 本の出力ピンにつないでください。

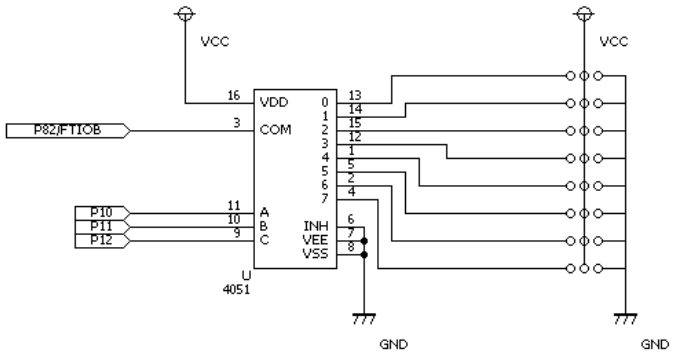


図 13.9　モータ回路図

今回のサンプルプログラムは、8 個のサーボモータを 2 秒ごとに動かすというものです。

07_TMRW05.c

```
 1  /************************************************************************/
 2  /*                                                                      */
 3  /*  FILE        :07_TMRW05.c                                            */
 4  /*  DESCRIPTION :Main Program                                           */
 5  /*  CPU TYPE    :H8/3694F                                               */
 6  /*  IC(TC4051B) を用いる                                                */
 7  /*                                                                      */
 8  /*  COM P82/FTIOB                                                       */
 9  /*  A P10                                                               */
10  /*  B P11                                                               */
11  /*  C P12                                                               */
12  /************************************************************************/
13
14  #include "iodefine.h"
15  #include<machine.h>
16
```

```
17  #define P1_MASK 0x07
18
19  #define PWM_MAX_DUTY 12000-1 //2.4mS
20  #define PWM_MIN_DUTY  2000-1 //0.4mS
21  #define PWM_PERIOD   12500-1 //2.5mS
22
23  #define SRV_CH_NUM 8 /* モータ数 */
24  #define MOTION_NUM 4 /* モーション数 */
25
26  int srv_ch;     /* PWM の出力先 */
27  int srv_flg;    /* 20ms をカウント */
28  unsigned short pwm_duty[SRV_CH_NUM]; /* PWM のデューティのデータ */
29
30  /*******************************
31     PWM 出力端子を初期化
32  *******************************/
33  void PwmInit(void)
34  {
35    srv_ch=0;
36    srv_flg=0;
37    IO.PCR1|=0x07;
38    IO.PDR1.BYTE|=(srv_ch & P1_MASK);
39    IO.PDR1.BYTE&=(srv_ch | (~P1_MASK));
40    set_imask_ccr(1);
41    TW.TMRW.BYTE=0x49;  /* FTIOB 端子を PWM 出力に設定 */
42    TW.TCRW.BYTE=0xA0;  /* 内部クロックの 1/4, コンペアマッチ B で 1 出力 */
43    TW.TIERW.BYTE=0x71; /* A の割り込みを可能にする */
44    TW.TSRW.BYTE=0x70;  /* 割り込みフラグをクリア */
45    TW.TCNT=0x0000;     /* TCN の初期化 */
46    TW.GRA=PWM_PERIOD;  /* 周期 (2.5mS) */
47    TW.GRB=PWM_PERIOD-7500;        /* 1.5mS */
48    TW.TMRW.BIT.CTS=1;  /* TCNT カウンタスタート */
49    set_imask_ccr(0);
50
51  }
52
53  /*********************************
54    サーボ動作開始
55   *********************************/
56  void PwmStart(void)
57  {
58    TW.TMRW.BIT.CTS=1;
59  }
60
61  /*********************************
62    サーボ動作停止
63   *********************************/
64  void PwmStop(void)
65  {
66    TW.TMRW.BIT.CTS=0;
67  }
68
69  /*********************************
70    デューティの設定
71   *********************************/
72  void PwmSetDuty(int ch, unsigned short duty)
73  {
74    /* サーボチャンネルが範囲外なら無視 */
75    if(ch >= SRV_CH_NUM){
76      return ;
77    }
78
79    if(duty < PWM_MIN_DUTY){          /* デューティを最小値以上に限定 */
80      duty = PWM_MIN_DUTY;
81    }else if(duty > PWM_MAX_DUTY){    /* デューティを最大値以下に限定 */
82      duty = PWM_MAX_DUTY;
83    }
84    /* デューティを設定 */
85    pwm_duty[ch] = duty;
86  }
87
88  void main(void)
89  {
90    int i, j=0;
91  /* PWM のデューティのデータ */
92  unsigned short pwm_data[MOTION_NUM][SRV_CH_NUM]=
93      {{ 5000,  7500, 10000,  7500, 7500, 10000,  7500,  5000},
94       { 7500,  5000,  7500, 10000, 5000,  7500, 10000,  7500},
95       { 7500, 10000,  7500,  5000, 7500,  5000,  7500, 10000},
96       { 5000,  7500, 10000,  7500, 7500,  5000,  7500, 10000}};
97
98    PwmInit(); /* PWM の初期化 */
99
100   while(1){
101     /* 次のモーションデータを設定 */
102     for(i=0; i<SRV_CH_NUM; i++){
103       PwmSetDuty(i, pwm_data[j][i]);
104     }
105     while(srv_flg<80){
106       ;
107     }
```

```
108       srv_flg=0;
109       j++;
110       if(j>=MOTION_NUM){
111         j=0;
112       }
113     }
114   }
115
116   /*********************************
117     割り込み関数
118    *********************************/
119   __interrupt(vect=21) void INT_TimerW(void)
120   {
121     /* 2.5mS ごとのコンペアマッチ */
122         TW.TSRW.BIT.IMFA=0;
123         srv_ch++;
124         if(srv_ch>= SRV_CH_NUM){
125           srv_ch=0; /* モータ出力先を 0 に戻す */
126           srv_flg++;
127         }
128         IO.PDR1.BYTE|=(srv_ch & P1_MASK);
129         IO.PDR1.BYTE&=(srv_ch | (~P1_MASK));
130         TW.GRB=PWM_PERIOD-pwm_duty[srv_ch];
131   }
132
```

End Of List

13.1.8 シリアルからサーボモータを動かす (IC で割り振り)

ここまでは、サーボモータに出力する PWM のデータをあらかじめプログラムに書き込んで実行してきました。

しかし、この方法でロボットに思ったポーズをとらせるのは意外と大変です。プログラムにデータを書き込み、ビルドしてマイコンに転送、実行してみて思ったポーズと異なるところを修正する。再度ビルドして実行ファイルをマイコンに転送、実行して確認する。この繰り返しです。これでうまく歩かせようとしたら、かなりの忍耐力を要します。

すでにシリアル通信のプログラムを学習しましたので、ターミナルソフトからサーボモータに送るパルスを変更するプログラムを作ってみましょう。このプログラムを用いれば、ロボットに思ったポーズを取らせることも比較的楽にできるようになります。

まずは、プログラムの仕様を考えておきましょう。

シリアルからサーボモータを動かすプログラム操作仕様

- 起動時は、すべてのモータが中心位置 (ON 時間 1.5ms)。操作対象モータは 0。
- 操作対象のモータ番号とそのパルスの ON 時間を表示 (図 13.10 参照)。
- + を入力するとパルスの ON 時間が 10us 増加する。
- - を入力するとパルスの ON 時間が 10us 減少する。
- u を入力するとパルスの ON 時間が 100us 増加する。
- d を入力するとパルスの ON 時間が 100us 減少する。
- > を入力すると操作対象のモータ番号が 1 増える。
- < を入力すると操作対象のモータ番号が 1 減る。
- s を入力するとパルス出力スタート (起動時はパルスは出力設定)。
- e を入力するとパルス出力ストップ。

以上の仕様をもとに、サンプルプログラムを作ってみました。参考にしてください。

07_TMRW06.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :07_TMRW06.c                                            */
/*  DESCRIPTION :シリアルから操作                                       */
/*  CPU TYPE    :H8/3694F                                               */
/*                                                                      */
/*  COM P82/FTIOB                                                       */
/*  A P10                                                               */
/*  B P11                                                               */
/*  C P12                                                               */
/************************************************************************/

#include "iodefine.h"
#include<machine.h>
#include "sci.h"
#include "str.h"

#define COEFF 5   /* 1us 単位 */

#define P1_MASK 0x07

#define PWM_MAX_DUTY 11500 //2.3mS
#define PWM_MIN_DUTY  3500 //0.7mS
#define PWM_MID_DUTY ((PWM_MAX_DUTY+PWM_MIN_DUTY)>>1) //1.5mS
#define PWM_PERIOD   12500-1 //2.5mS

#define SRV_CH_NUM 8 /* モータ数 */
#define MOTION_NUM 4 /* モーション数 */

char buf[6];
unsigned short n;
int c;
int srv_curr_index=0;
int srv_ch;     /* PWM の出力先 */
int srv_flg;    /* 20ms をカウント */
unsigned short pwm_duty[SRV_CH_NUM]; /* PWM のデューティのデータ */

/*******************************
   PWM 出力端子を初期化
*******************************/
void PwmInit(void)
{
  srv_ch=0;
  srv_flg=0;
  IO.PCR1|=0x07;
  IO.PDR1.BYTE|=(srv_ch & P1_MASK);
  IO.PDR1.BYTE&=(srv_ch | (~P1_MASK));
  set_imask_ccr(1);
  TW.TMRW.BYTE=0x49;  /* FTIOB 端子を PWM 出力に設定 */
  TW.TCRW.BYTE=0xA0;  /* 内部クロックの 1/4, コンペアマッチ B で 1 出力 */
```

```
51    TW.TIERW.BYTE=0x71; /* A の割り込みを可能にする */
52    TW.TSRW.BYTE=0x70;  /* 割り込みフラグをクリア */
53    TW.TCNT=0x0000;     /* TCN の初期化 */
54    TW.GRA=PWM_PERIOD;  /* 周期 (2.5mS) */
55    TW.GRB=PWM_PERIOD-PWM_MID_DUTY; /* 2.5ms-1.5ms */
56    TW.TMRW.BIT.CTS=1;  /* TCNT カウンタスタート */
57    set_imask_ccr(0);
58
59  }
60
61  /*********************************
62    サーボ動作開始
63   *********************************/
64  void PwmStart(void)
65  {
66    TW.TMRW.BIT.CTS=1;
67  }
68
69  /*********************************
70    サーボ動作停止
71   *********************************/
72  void PwmStop(void)
73  {
74    TW.TMRW.BIT.CTS=0;
75  }
76
77  /*********************************
78    デューティの設定
79   *********************************/
80  void PwmSetDuty(int ch, unsigned short duty)
81  {
82    /* サーボチャンネルが範囲外なら無視 */
83    if(ch >= SRV_CH_NUM){
84      return ;
85    }
86
87    if(duty < PWM_MIN_DUTY){          /* デューティを最小値以上に限定 */
88      duty = PWM_MIN_DUTY;
89    }else if(duty > PWM_MAX_DUTY){    /* デューティを最大値以下に限定 */
90      duty = PWM_MAX_DUTY;
91    }
92    /* デューティを設定 */
93    pwm_duty[ch] = duty;
94  }
95
96
97  /*********************************
98    デューティの取得（パラメータから）
99   *********************************/
100 unsigned short PwmGetDuty(unsigned short ch)
101 {
102   /* サーボチャンネルが範囲外なら無視 */
103   if(ch >= SRV_CH_NUM){
104     return (0);
105   }
106
107   /* デューティを返す */
108   return (pwm_duty[ch]);
109 }
110
111 void main(void)
112 {
113   int a;
114   Sci1Init(br19200);
115   PwmInit();
116   for(a=0; a<SRV_CH_NUM; a++){
117     PwmSetDuty(a, PWM_MID_DUTY);
118   }
119   PwmStart();
120   while(1){
121     Sci1Puts("*************************\n");
122     Sci1Puts("パルス増減\n");
123     LongToStrDec(srv_curr_index, buf);
124     Sci1Puts(buf);
125     Sci1Puts("のパルス ON 時間:約");
126     n=PwmGetDuty(srv_curr_index);
127     n=(n+1)/COEFF;
128     LongToStrDec(n, buf);
129     Sci1Puts(buf);
130     Sci1Puts("us\n");
131     Sci1Puts("--------------------------\n");
132     Sci1Puts("+:  +10us\n");
133     Sci1Puts("-:  -10us\n");
134     Sci1Puts("u: +100us\n");
135     Sci1Puts("d: -100us\n");
136     Sci1Puts(">:次のモータ\n");
137     Sci1Puts("<:前のモータ\n");
138     Sci1Puts("s:パルススタート\n");
139     Sci1Puts("e:パルスストップ\n");
140
```

```
141       c=Sci1Getchar();
142       switch(c){
143       case '+':
144         PwmSetDuty(srv_curr_index, PwmGetDuty(srv_curr_index)+10*COEFF);
145         break;
146       case '-':
147         PwmSetDuty(srv_curr_index, PwmGetDuty(srv_curr_index)-10*COEFF);
148         break;
149       case 'u':
150         PwmSetDuty(srv_curr_index, PwmGetDuty(srv_curr_index)+100*COEFF);
151         break;
152       case 'd':
153         PwmSetDuty(srv_curr_index, PwmGetDuty(srv_curr_index)-100*COEFF);
154         break;
155       case '>':
156         if(srv_curr_index < SRV_CH_NUM-1){
157           srv_curr_index++;
158         }
159         break;
160       case '<':
161         if(srv_curr_index > 0){
162           srv_curr_index--;
163         }
164         break;
165       case 's':
166         PwmStart();
167         break;
168       case 'e':
169         PwmStop();
170         break;
171       }
172     }
173   }
174
175   /*********************************
176     割り込み関数
177    *********************************/
178   __interrupt(vect=21) void INT_TimerW(void)
179   {
180     /* 2.5mS ごとのコンペアマッチ */
181         TW.TSRW.BIT.IMFA=0;
182         srv_ch++;
183         if(srv_ch>= SRV_CH_NUM){
184           srv_ch=0; /* モータ出力先を 0 に戻す */
185           srv_flg++;
186         }
187         IO.PDR1.BYTE|=(srv_ch & P1_MASK);
188         IO.PDR1.BYTE&=(srv_ch | (~P1_MASK));
189         TW.GRB=PWM_PERIOD-pwm_duty[srv_ch];
190   }
```

End Of List

実行結果

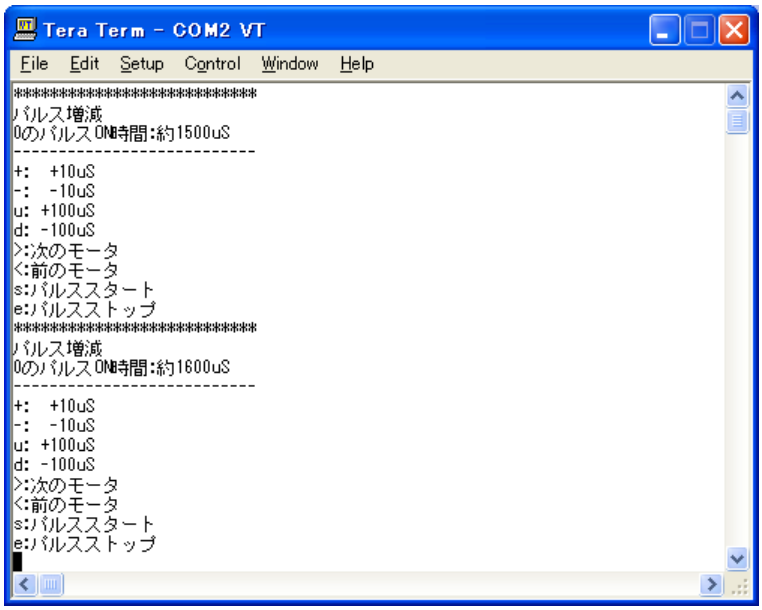


図 13.10　シリアル通信でサーボモータを動かす

課題 13.1.4　(提出) ソフトで割り振るサンプルをシリアル通信に対応させる

P.354 の 13.1.6「ソフト的に割り振り (8 個駆動)」で示したサンプルプログラム 07_TMRW04.c をシリアルからサーボモータが動かせるように変更してください。

このプログラムは、P.378 の 14.4 で、4 軸アザラシ型ロボットを動かすときに使います。

プロジェクト名：e07_TMRW06