

8ビットマイコン78K0S/Kx1+を使った モールス送受信機

大阪府
東條 陽介様

はじめに



8ビットマイコン78K0S/Kx1+を使ったモールスコードの光データ送受信機

今回はLEDとフォトセンサを使った、マイコン間のモールスコードによる文字データの光通信を紹介します。小規模なマイコン間シリアルデータ通信として一般的なものは、PCのCOMポートに代表される非同期シリアル通信や、SPIやI²Cバスなどの同期型シリアル通信です。また、光を使ったデータ通信としては、IrDAや赤外線リモコンなどの通信方式が馴染み深いでしょう。

これらのデータ通信は、基本的には電圧の高低や赤外線のON/OFFという2状態を使ってデータ伝送を行っていますが、マイコンはおろかコンピュータさえ無い頃からデータ通信用に使われてきたのが、いわゆるトン・ツーで文字伝送を行う電信です。この電信で利用されてきた文字コードがモールスコードです。モールスコードを利用した通信は当初有線(電線で直接つなぐ)で、その後は短波通信などが広く利用されました。特に短波は電離層のおかげで遠くまで届く(ちなみに、これを発見したのはアマチュア無線家でした)という利点から、船舶の通信をはじめ、長い間利用されてきました。

悪天候に遭遇した船の通信室で通信士が必死にSOS(・・・ ―― ・・・)を打電していたり、サーチライトの前にシャッターを付けたようなものを操作して船と船の間で光通信をしているシーンを映画などで見た方も多いことでしょう。

このように広く利用されていたモールスコードによる通信にも時代の波が押し寄せます。衛星中継によるデジタルデータ通信の発達によって、安定した長距離通信路が確保できるようになったことから、短波帯でのモールスコードによる通信の利用は急減し、現在ではアマチュア無線などごく限られた範囲で利用される程度になりました。

すでに実社会では過去のものとなっているモールスコードですが、コード化の仕組みなどを考えてみる素材として面白いと思います。



モールスコード

モールスコードは短点(トン)と長点(ツー)の組み合わせで文字をコード化しています。**表1**は、和文(イロハ・・・)と欧文(ABC・・・)のモールスコードを整理したものです。短点(トン)は「・」で、長点(ツー)は「―」で示しています。

モールスコードで短点や長点、間隔(スペース)は次のように決められています。

- ・長点は短点の3倍の長さ
- ・点と点の間隔は短点1個分
- ・文字と文字の間隔は短点3個分
- ・単語と単語の間隔は短点7個分

図1は「THIS IS A」と送信したときの例です。

モールスコードは一見トンとツーだけでなので2進コードのようにも思えますが、実際にはスペース(無音)期間も意味を持っているところがポイントです。

たとえば、「トンツー」とつながっていれば「A」ですが、「トン・・・ツー」と3短点分の時間で分かれていると「ET」という2文字の単語になり、「トン・・・・・・・・・・ツー」と7短点分の時間なら「E」と「T」という具合に分かれた2単語となるのです。

●文字コード

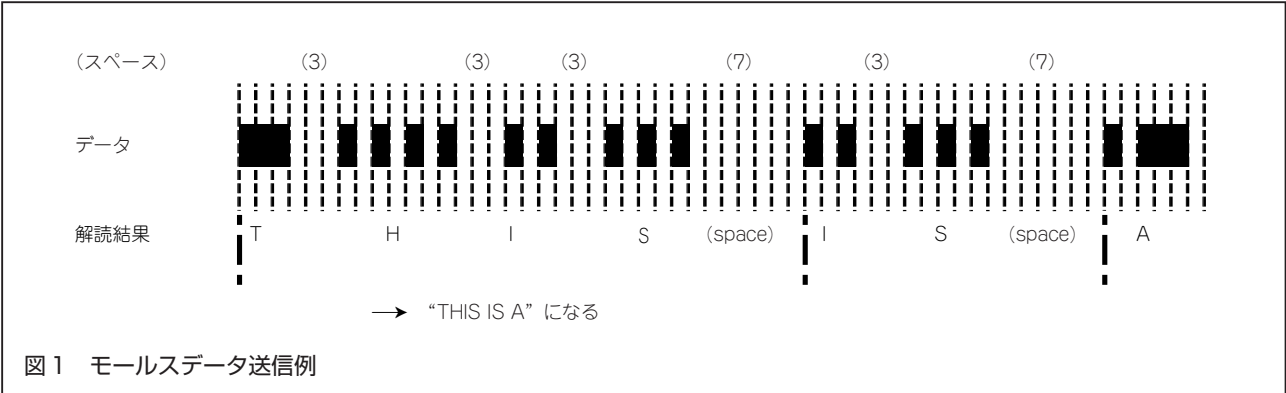
一般的に文字の伝送を行う場合、それぞれの文字に同じビット数の文字コードを割り付けるという方法が利用され

表1 和文/欧文モールスコード表

和	欧	符号	和	欧	符号
イ	A	·-·	ユ		-·-·-·
ロ		·-·-·	メ		-·-·-·
ハ	B	-·-·-·	ミ		·-·-·-·
ニ	C	-·-·-·	シ		-·-·-·
ホ	D	-·-·	エ		·-·-·-·
ヘ	E	·-·-·	ヒ		-·-·-·
ト		·-·-·-·	モ		-·-·-·
チ	F	·-·-·	セ		·-·-·-·
リ	G	-·-·	ス		-·-·-·
ヌ	H	·-·-·	ン		·-·-·-·
ル		-·-·-·	0		-·-·-·
濁音	I	·-·	1		·-·-·-·
ヲ	J	·-·-·-·	2		·-·-·-·
ワ	K	-·-·	3		·-·-·-·
カ	L	·-·-·	4		·-·-·-·
ヨ	M	-·-·	5		·-·-·-·
タ	N	·-·	6		-·-·-·
レ	O	-·-·	7		-·-·-·
ソ		-·-·-·	8		-·-·-·
ツ	P	·-·-·	9		-·-·-·
ネ	Q	-·-·-·	.	(ピリオド)	·-·-·-·
ナ	R	·-·-·	.	(カンマ)	-·-·-·

和	欧	符号	和	欧	符号	
ヲ	S	·-·-·		?	·-·-·-·	
ム	T	-		' (アポストロフ)	·-·-·-·	
ウ	U	·-·-·		!	-·-·-·	
キ		·-·-·-·		/	-·-·-·	
ノ		·-·-·-·		(-·-·-·	
オ		·-·-·-·	へ)	-·-·-·	
ク	V	·-·-·		:	(コロン)	-·-·-·
ヤ	W	·-·-·		;	(セミコロン)	-·-·-·
マ	X	-·-·-·		=	-·-·-·	
ケ	Y	-·-·-·		+	·-·-·-·	
フ	Z	-·-·-·		-	-·-·-·	
コ		-·-·-·		_	(アンダースコア)	·-·-·-·
エ		-·-·-·		"	·-·-·-·	
テ		·-·-·-·		\$	·-·-·-·	
ア		-·-·-·		@	·-·-·-·	
サ		-·-·-·	半濁音		·-·-·-·	
キ		-·-·-·	ゝ		·-·-·-·	

※長点の長さは点3つ分
 ※点(長点、短点とも)間は短点1つ分
 ※文字間は短点3つ分
 ※単語間は短点7つ分



ています。マイコンなどでも良く利用されているのは7ビット分のデータに英数字などを割り振ったASCIIコードです。このほかにも、漢字などを2バイトコードにして表すJISコード/シフトJISコードや、世界各国の文字を统一的に扱うUNICODE(ユニコード)などもあります。

一方、モールスコードはもともと人間が手作業で文字の送受信を行うことを前提として作成されたものであり、コードの長さやトン・ツーの組み合わせの数は文字コードによって異なります。いわば可変長データコードであると言えるでしょう。

モールスコードは、効率良く伝送を行えるように出現頻度の高いコードには短いコードを割り振るように設計されていると言われていますが、実際にどのようなになっているのかを調べてみました。

表2はGNU Free Documentation License Version 1.2, November 2002 (<http://www.gnu.org/licenses/fdl.html>)と、μPD78P9014のデータシート(<http://www.necel.com/nedis/image/U10912EJ1V0DS00.pdf>)から英文字(数字・記号等は除く)を取り出して、出現比率を計算したものです。

多少の順位の入れ替わりはありますが、E、T、I、A、Nなどが上位に、X、Q、J、Zなどが下位にきている点など、同じ傾向がみられます。

一方、モールスコードがどのように割り付けられているのかをツリー状に整理したのが図2です。二分岐している部分では左側が短点、右が長点です。たとえば、「R」ならば「·-·」なので、左、右、左と降りていった先にあります。更にLは「·-·-·」なのでRの先に更に短点がきたところに配置されています。大まかに言えば、このツリーで上にあるほど、人が電鍵(でんけん)をON/OFFする回数が少なく済み、早く送りやすいコードであるとも言えるでしょう。

表2 英文ドキュメントの文字別構成比率の例

GUN Free Documentation License			μPD78P9014データシート			GUN Free Documentation License			μPD78P9014データシート		
文字	出現数	比率	文字	出現数	比率	文字	出現数	比率	文字	出現数	比率
E	1884	12.512%	E	3336	10.009%	F	367	2.437%	H	889	2.667%
T	1525	10.128%	T	3099	9.298%	P	329	2.185%	V	556	1.668%
I	1269	8.428%	I	2488	7.465%	Y	294	1.953%	F	544	1.632%
O	1267	8.415%	A	2370	7.111%	G	203	1.348%	G	541	1.623%
N	1126	7.478%	O	2346	7.039%	V	196	1.302%	B	514	1.542%
A	1001	6.648%	R	2239	6.718%	B	189	1.255%	Y	468	1.404%
S	990	6.575%	S	2191	6.574%	W	154	1.023%	W	301	0.903%
R	894	5.937%	N	2092	6.277%	K	58	0.385%	K	272	0.816%
C	610	4.051%	D	1871	5.614%	Z	58	0.385%	X	231	0.693%
H	601	3.991%	P	1707	5.122%	Q	28	0.186%	J	59	0.177%
D	597	3.965%	C	1681	5.004%	J	9	0.060%	Q	52	0.156%
L	547	3.633%	M	1313	3.940%	Z	8	0.053%	Z	37	0.111%
U	439	2.916%	L	1150	3.450%	合計	15057	100.000%	合計	33329	100.000%
M	414	2.750%	U	982	2.946%						

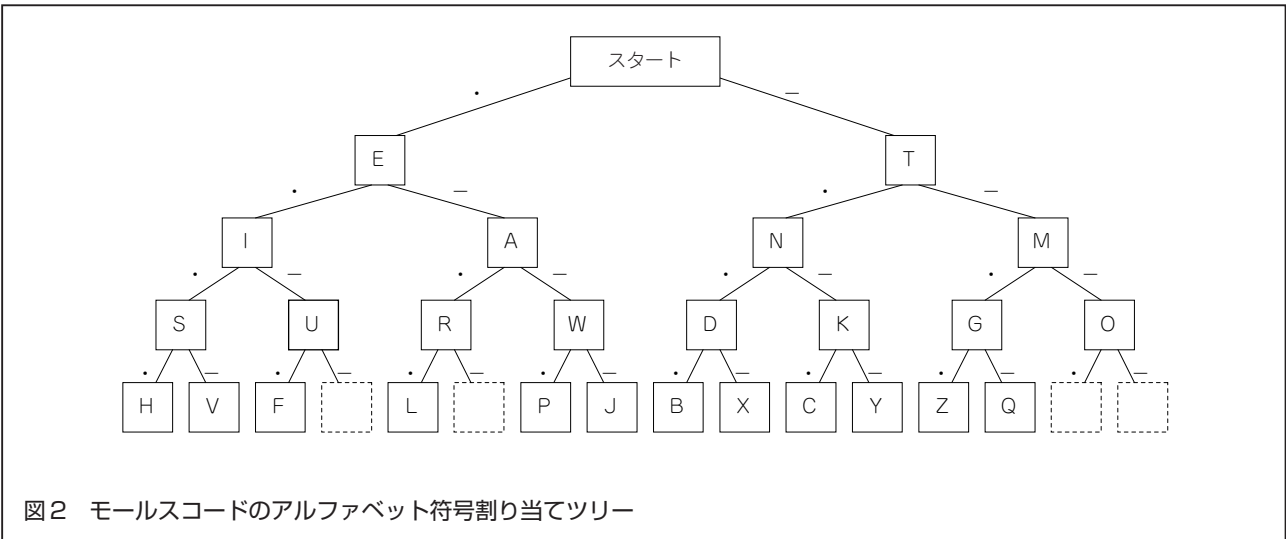


図2 モールスコードのアルファベット符号割り当てツリー

先ほどの出現比率とこのツリーを見比べてみると、確かに出現率が最も高いEが「·」に、二番目のTが「—」、三番目のIが「· ·」という具合に比較的短いコードを割り付けられており、逆にZ(— — · ·)やJ(· — — —)、Q(— — — —)などはツリーの最下層に配置されていることがわかります。やはり、モールスコード出現頻度の高い順に短いコード、送りやすいコードを割り振っているのです。

現在のように数万字あるテキストファイルから数秒で文字数をカウントできるわけでもなく、手作業でカウントし、コードを割り振っていたことを考えると、その労力も大変だったと思います。

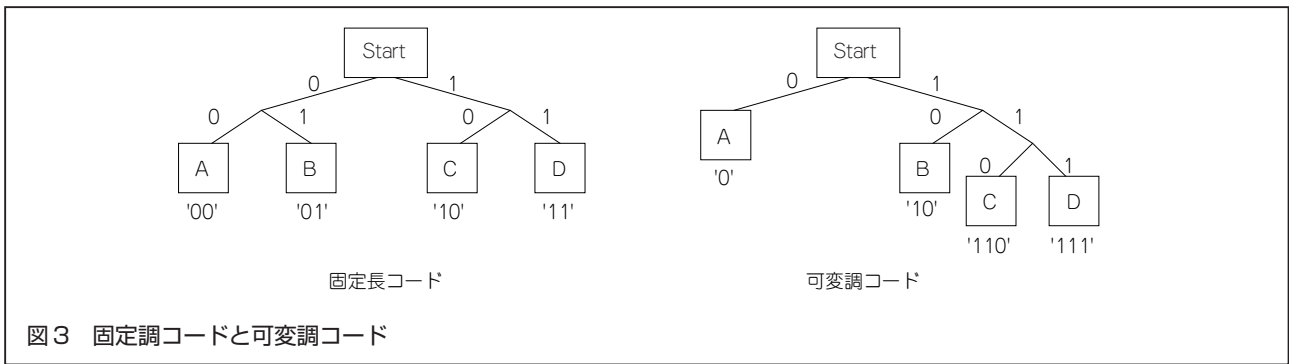
もっとも、これは欧文の場合で、和文の場合にはモールスコード表からわかるとおり、ABCの順に単純にイロハと割り振っていき、足りない部分は途中や最後にコードを追加するという具合で、出現頻度とはあまり関係なくコードを割り振っています。

●可変長コードの考え方

モールスコードのように、使用頻度に応じてコード長を変える可変長コードは、データ圧縮の分野でも利用されている基本的なテクニックです。簡単な例を使って考えてみることにしましょう。

送受信したいデータが“ABCD”という四つの文字の組み合わせで構成されていたとします。このとき次のようにすべてを2ビットで表現する方法がまず考えられます。

- A:00
- B:01
- C:10
- D:11



これが固定長コードの考え方です。一方、このビット数を文字によって変えるのが可変長コードの考え方です。たとえば次のように、1～3ビット長に割り付けてみます。

A:0
B:10
C:110
D:111

これを先ほどのモルスコードと同じようにツリー状にして表現したのが**図3**です。モルスのときの図と似ていますが、この場合はスペースによって文字間を区切ることができないので、分岐点には文字コードを割り振れず、末端部分が文字コードになります。

この例の場合に“ABC”という文字を表すと、固定長ならばそれぞれ2ビットなので6ビット、可変長のほうは1+2+3となり、やはり6ビットとなります。もし、長いコードを割り付けられているCやDのコードが多ければ固定長よりも長くなり、短いコードが多ければ固定長よりも短くなります。

出現頻度が違う場合にはどうなるのか考えてみましょう。A、B、C、Dの文字がそれぞれ50%、25%、13%、12%という頻度で現れたとします。わかりやすくするために100文字あるとして、A、B、C、Dが、それぞれ50、25、13、12文字現れたとして考えます。

この場合、固定長データならば、総データ長は100文字×2ビットで、200ビットです。一方、可変長データにすると、 $50 \times 1 + 25 \times 2 + 13 \times 3 + 12 \times 3$ で、計175ビットになります。つまり、12%ほどデータ量が減ったわけです。もちろん、データ量は減ってもきちんと元の文字列を再現できるので、情報自体はまったく失われていません。このように、データの出現頻度によってコード長を変更することで総データ量を減らすことができます。

この例ではデータを1～3ビット長に割り付けましたが、図の枝葉の分かれた先はすべて違うコードとして判別可能なので、最小コード長を1ビットに限定する必要はありません。

たとえば3文字(E、F、G)のコードを割り振りたいときはA、B、Dの下を分岐させ、次のような2～4ビット長のデータとしても良いのです。

00 : A
01 : E
100 : B
101 : F
110 : C
1110 : D
1111 : G

この場合には、7種類の文字を表現することになるので、固定長の考え方ならば3ビットが必要です。“ABCDEFG”を送るのに

固定長： $3 \times 7 = 21$ ビット

可変長： $2 \times 2 + 3 \times 3 + 4 \times 2 = 21$ ビット

となり、どちらでも同じデータ長なので、すべての文字がランダムに現れるならばどちらもほぼ同じデータ量になります。文字‘A’、‘E’の出現頻度が‘D’や‘G’よりも高ければ、可変長のデータ量は固定長よりも少なくなります。

ツリー図でわかるように、1つの文字のコードを他の文字コードの下に移動させると、配置される側と移動する文

字の2文字分のコードが1ビットずつ増えるので、コードの割り振り方には配慮が必要でしょう。

コードによるビット数の差が大きい場合、短いコードの出現率が高ければ、可変長コードは高い圧縮率が期待できますが、長いコードの総数が増えるので、ランダムに現れる場合にはやや不利です。逆にビット数の幅が小さければ、ランダムに文字が現れる場合には有利ですが、圧縮率はあまりあがらないということになります。

エラー検出と再同期

メモリ上のデータのように、データの受け渡し時のエラー発生を考えなくて良い場合には、先ほどの可変長データの例のようにデータを密に並べてもかまいません。たとえば“ABCDEFGH”であれば、0010011011100110111111という具合にしても順にデコードしていけば正しく元の文字を再現することができます。

しかし、実際のデータ伝送を行う場合には、必ず伝送エラーが発生することに配慮しなくてはなりません。上記の例では最初の‘0’が‘1’と誤って判断されるだけでも先頭の2文字がFAになり、パターンによっては延々とつじつまが合わなくなることもありえます。

また、モジュール通信の場合、コードは人の手で電鍵を操作して送るので、伝送速度は人によって大きく異なります。入門者で1分間あたり数十文字程度ですが、ベテランになると1分あたり100文字を軽く超える速度で送受信しています。データ通信の世界ではこれほど大きく変動することはあまりありませんが、送信側、受信側とも独立したクロック信号で動いている以上、送信側と受信側の基準周波数のずれは必ずあります。もしもずれたクロックに基づいてデータを取り込み続けていると、差分が蓄積されていき、同じデータを二回読んでしまうなどの問題が発生します。

このような問題に対処するため、通常データ伝送の場合には1バイト毎や1パケット毎に送受信双方のつじつまを合わせる(同期を取る)仕組みが組み込んであります。何らかの原因でつじつまが合わなくなっても、再び正しい位置からデータ受信が再開できるようになっています。

●非同期シリアル(調歩同期)の通信フォーマット

ここで、このような再同期の仕組みが実際にどのように実現されているのかを見てみましょう。身近なデータ通信としてマイコンのシリアルポートなどで良く利用されている非同期シリアル通信を取り上げます。図4を参照してください。

非同期シリアル通信で採用している方法は“調歩同期”と呼ばれます。‘1’の状態をマーク状態、‘0’の状態をスペース状態とも呼び、通常はマーク状態になっています。

データ送信が始まる時には、送信側はまずデータラインを1ビット分、スペース状態にします。これをスタートビットと呼びます。1ビット分の時間は送信ビットレートの逆数になるので、たとえば9600bpsならば、約104 μ s(1秒/9600)、1200bpsならば約833 μ sになります。

スタートビットに続いて送信したいデータを上位ビットから順に送ります。‘1’ならば1ビット期間中だけマーク状態に、‘0’ならばスペース状態にするだけです。この例では8ビットデータを送るので、ビット7(b7)から送っていますが、たとえば半角英数字だけなら7ビットあれば足りるため、7ビット長が使われることもあります。

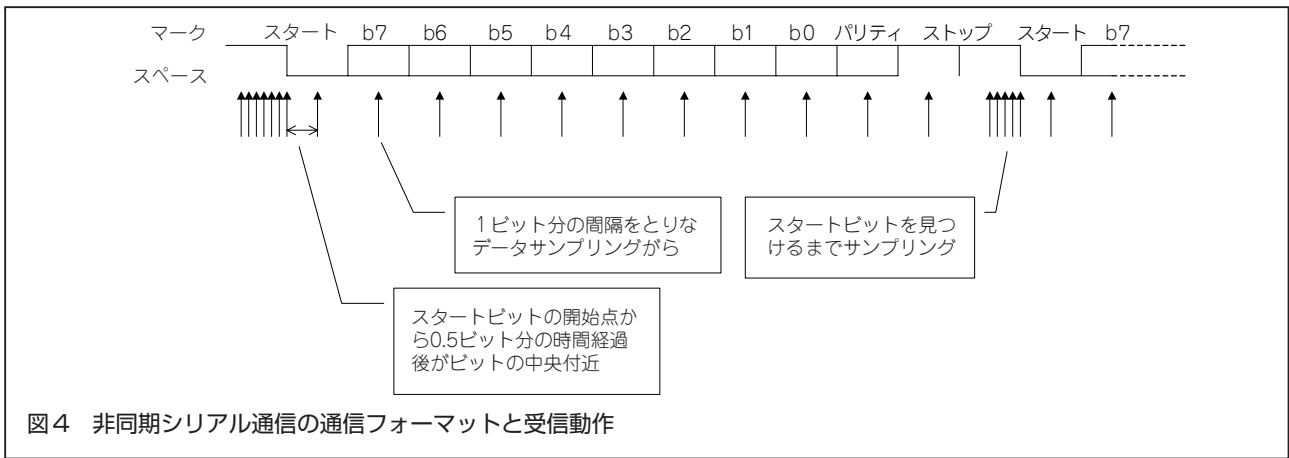
この例では、最後にエラーチェック用としてパリティビットを付加しています。パリティはデータ(今回の例ならばパリティ分をあわせて9ビットデータ)の中の‘1’の数を偶数、または奇数になるように調整するもので、偶数個にするのを偶数パリティ、奇数個にするのを奇数パリティと呼んでいます。たとえばデータが11111000(F8h)ならば1の数が5個あるので、偶数パリティを採用していれば、パリティビットは‘1’、奇数パリティならばパリティビットは‘0’になります。

パリティビットまで送った後はデータラインをマーク状態にします。連続してデータ伝送が行われる場合でも、必ず次のスタートビットの前に一定時間マーク状態にします。これをストップビットと言い、通常1ビット長、1.5ビット長、2ビット長のいずれかになっています。

伝送ビットレートやデータ長、パリティの有無や種別、ストップビット長などはあらかじめ送受信双方で同じ設定にしておきます。

●非同期シリアル(調歩同期)の受信側の動作

受信側ではどのようにしてこれを受信するのでしょうか。送信側からデータと共に取り込みタイミングを示すクロック信号が送られてくれば簡単ですが、実際にはデータラインしかつながっていません。そこで、非同期シリアル



通信では受信側はスタートビットを使って送信側にタイミングを合わせるようにしています。図4の上向き矢印は受信側によるデータラインのサンプリングのタイミングを表したものです。

受信側では伝送ビットレートよりも速いタイミング(通常8倍から16倍程度)でデータラインをサンプリングして、スタートビットが来るのを待ちます。スタートビットが検出された時点から0.5ビット分待つと、ビットの中央付近になります。たとえばビットレートの16倍の速度でサンプリングしているのであれば、8クロック分だけ待つとビットの中央付近になります。このあとは1ビット分(16倍サンプリングしていたなら16クロック分)の時間だけ待ってデータを取り込むという動作を繰り返せば、ビットのほぼ中央の安定したデータを取り込むことができます。最後のストップビットまで受信できたら、再び8倍なり16倍のサンプリングを開始してスタートビットがくるのを待ちます。

調歩同期式ではこのように1バイトデータごとにスタートビットで同期を取り直すので、ビットレートの変動や送受信間でのずれには比較的強くなっています。たとえば16倍でサンプリングしている場合、ワーストケースでスタートビットの開始位置から1クロック分ずれたところでサンプリングが始まったとします。このあと、受信側のビットレートが遅く、サンプリングするポイントが徐々に図4の左にずれたとします。もし、最後のパリティビットのサンプリングするところがストップビットの位置までずれてしまうと、パリティエラーになる可能性があります。また、2個目のストップビットを取り込むポイントが次のスタートビットの位置までできてしまうと、次のデータが取り込めなくなります。

仮にパリティビットまで取ればよいとしてみましょう。ストップビットを取り込むまでに7クロック分(16÷2-1)ずれなければ良いので、パリティまでは9.5ビット分、つまり9.5×16=152クロックあります。取り込みタイミングで1ビット分ずれるとすると、151クロック分ですので7÷151×100≒4.6%となります。余裕を見ても双方が±2%程度の誤差の範囲に入っていれば大丈夫でしょう。

●モールス通信での同期の取り方

モールス通信では、このような同期を取り直す仕組みとして短点と長点の長さやスペースの量を利用しています。最も同期がとりにくい場合として、送信が行われている途中から聞き始めたとします。このとき、いきなり聞いた「ピー」という音が短点なのか長点なのかはすぐにはわかりません。しかし、通常の記事を送っている場合に、短点や長点ばかり続いているということはないので、いずれ短点と長点の両方が現れ、点間や文字間の区切りが登場します。いわばこの段階がビットレートの同期段階です。続いて短点の長さを基準にしてスペースの種類(1、3、7短点分)が識別できます。これで文字の区切り、単語の区切りがわかるわけです。これらの判定によって文字単位、単語単位でのデータ(文字情報)の同期がとれます。

このようにモールス通信ではスペースが重要な役割を担っています。モールスコードを受信したとき、人は無意識のうちにこの作業を行っているのです。

伝送速度の変化にも自動的に追従してデコード結果をターミナルに表示することができるモールスコードの自動受信装置もありますが、今回はこのような高度な速度追従機能はもたせず、送受信ともあらかじめ同じ伝送レート(1短点が100ms)に設定して、文字間の3短点の区切りを調歩同期式のストップビットのように利用して、1文字ごとに同期を取り直すようにしました。

モールスコードによる光送受信機の製作

今回はモールスコードによる送受信の実験ということで、特に長距離伝送は考えず、LEDで送信し、フォトセンサで受信するようにしました。

写真1、**写真2**は今回作成したインターフェース部分の写真です。写真のとおりごく単純なものですので、製作も容易でしょう。

●回路図

図5が今回作成したモールス通信の回路図です。78K0S/KA1+のUARTとPCを接続して、シリアル通信を行います。光送受信に使うのはP4.5で、'1'を書き込むとLEDが点灯、'0'で消灯です。受信側は光を受けるとフォトダイオードの抵抗値が下がりP4.5が'1'になります。

●プログラム

送信側、受信側のデータ通信の流れを図にしたのが**図6**です。これは'A' (・ー)を送ったときの例です。

今回は'0'のときにLEDをON、'1'のときにOFFにしています。通常は'0'状態なので、LEDは点灯状態になり、'1'を送信するときに消灯します。受信側ではOFFのときに'0'、ONのときに'1'と読めます。つまり、送信側で'1'を書くと受信側で'0'として読めることになります。ただし、受信側のプログラムリストでは'1'をLEDON、'1'をLEDOFFと表記しています。

送信側、受信側とも1短点時間ごとにTM00を使って割り込みを発生させます。TM00のクロックはシステムク

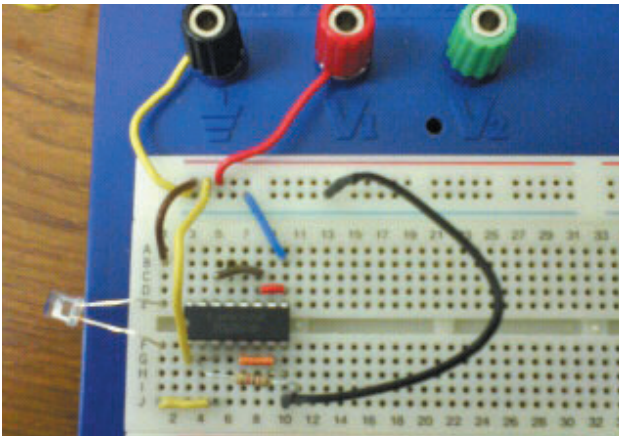


写真1

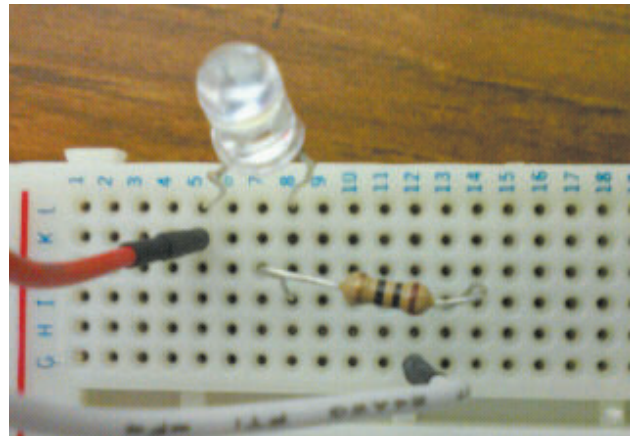
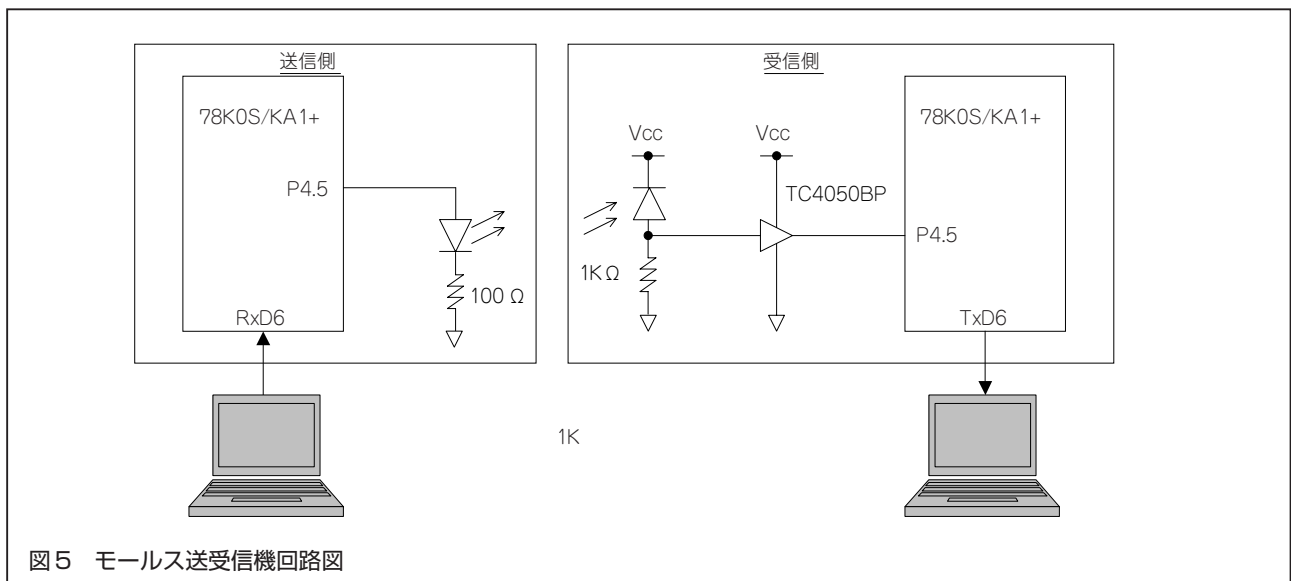


写真2



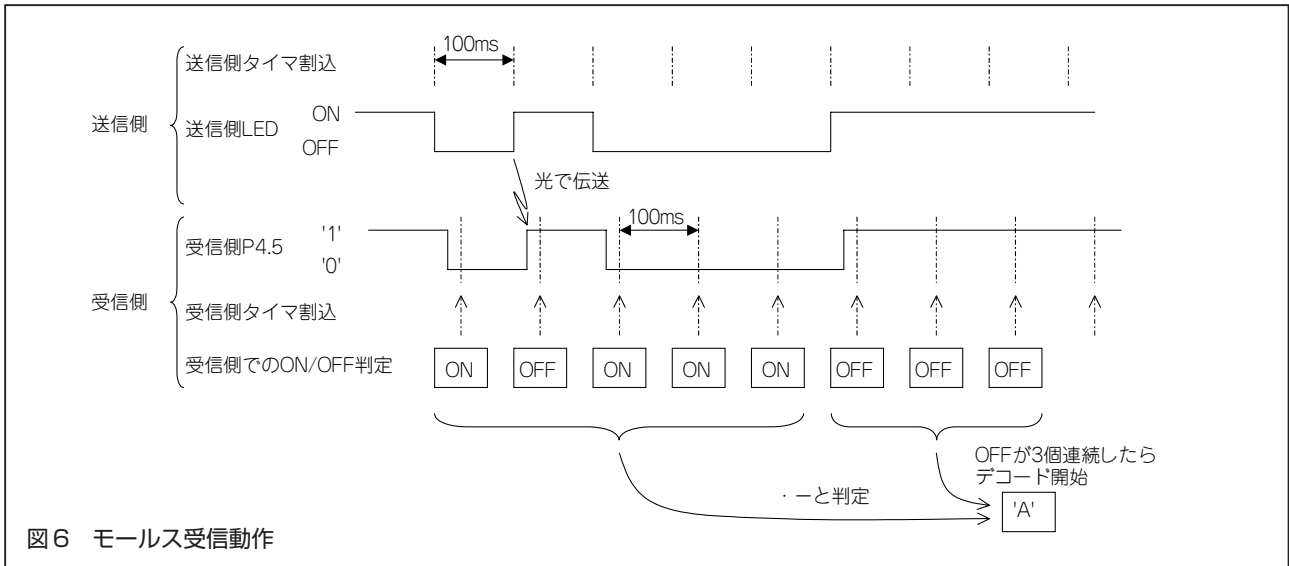


図6 モールス受信動作

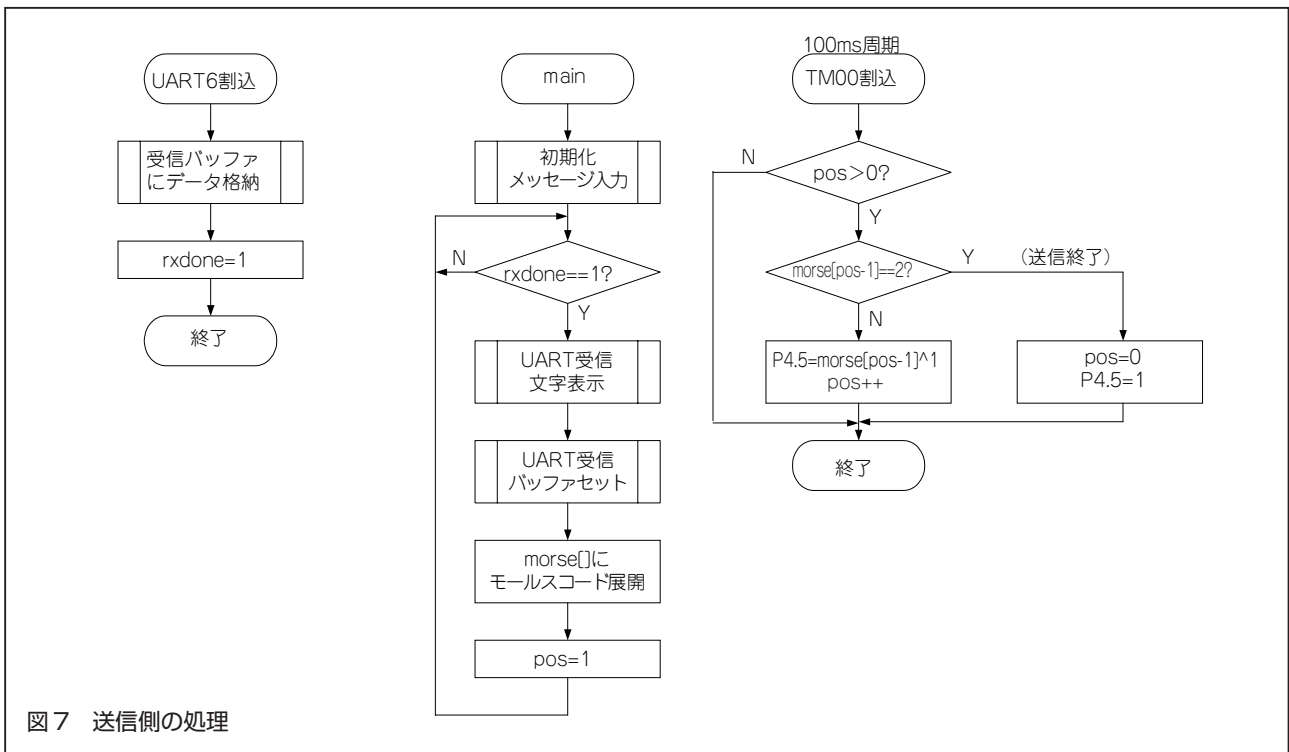


図7 送信側の処理

ロック(8MHzの高速内蔵オシレータ)を256分の1にした31.25KHzを利用して、更にこれを3125分周して10Hz(100ms)周期の割り込みを発生させます。これを1短点の時間としました。

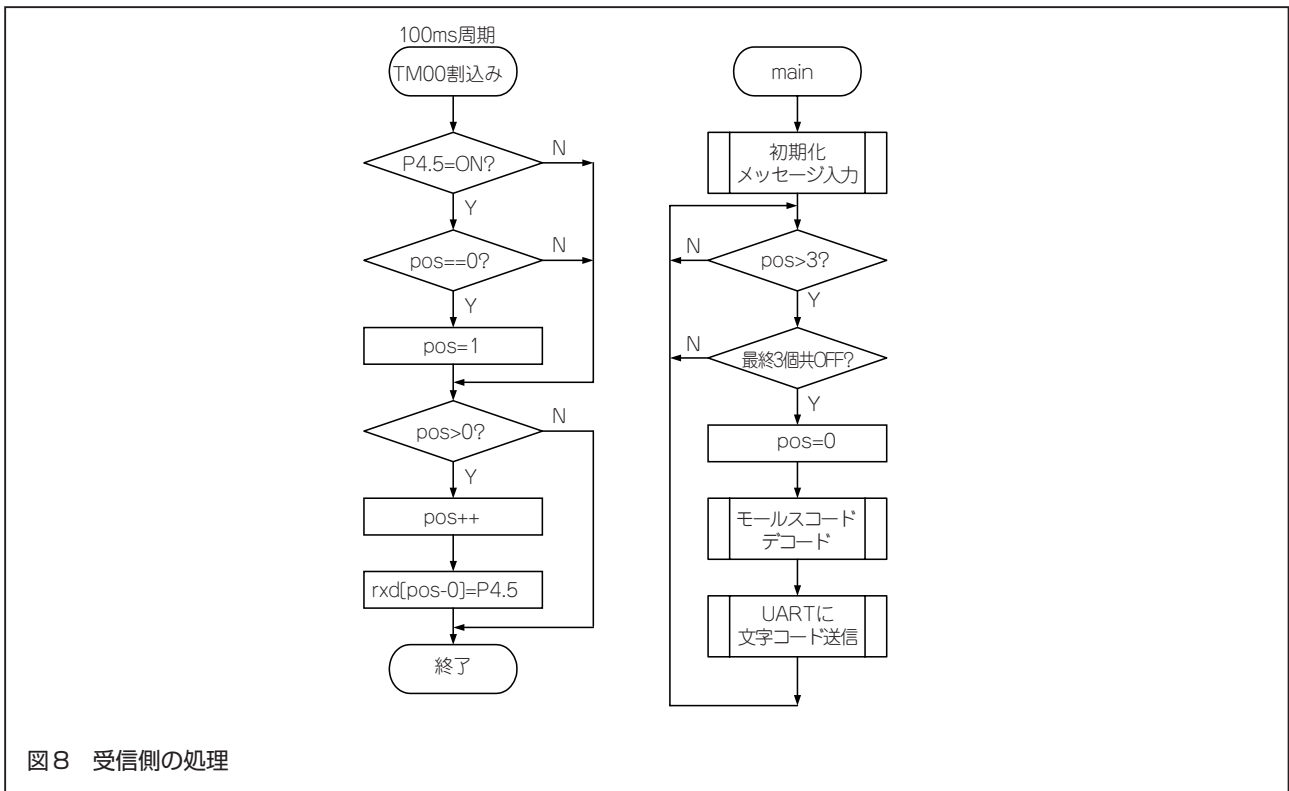
送信側ではこの100ms周期のタイマ割り込みを使って、データを送出し、受信側では同じように100ms周期のタイマ割り込みでデータをサンプリングします。もう少し正確に行うならば、調歩同期の時のように細かくサンプリングするほうが良いでしょう。

サンプリングされたデータは順次配列に格納していき、'1'状態(送信側では'0'状態)が3ビット続くと文字の終わりだと判断してデコードを行います。

●送信側のプログラム

図7が送信側のプログラムフローです。PCのターミナルソフトから入力された文字は78K0S/KA1+の割り込み処理の中で受け取られます。受信データはバッファに格納され、rxdoneビットを'1'にします。

メインルーチンではrxdoneフラグが立つのを待っています。フラグが立ってデータが受け取られると、シリアルポート経由でデータをエコーバックするとともに、送信バッファ(morse[])にモールスコードを展開します。



morse[]に展開されたコードは100msごとのTM00割込みの中でチェックされ、展開されていれば、TM00割込みのたびに順次送ります。

送出データはmorse[]データを反転しています。つまり、データが'0'ならば'1'を書き込み、'1'ならば'0'を書き込みます。

● 受信側のプログラム

図8が受信側のプログラムフローです。受信側では送信側と同じように100ms周期の割込みの中で受信ポートをチェックしています。受信データはrxid[]配列に順次格納されていきます。

rxid[]配列の内容はメインルーチン側でチェックしており、最後の3つがLEDOFF(実際には点灯)になっていると文字の区切りがきたものとみなして、それよりも前の部分をモールスコードとしてデコードし、UART経由でPCに送ります。

おわりに

今回はモールスコードをマイコン間通信に利用した工作事例です。短点1つ分の時間が100msもあり、アルファベットの「O」(---)を送るだけでも1秒程度の時間がかかる計算ですので、実用性はあまりありませんが、点滅する光を見てマイコンと一緒にモールスのデコードを行いながら、電信が通信の主役であった時代に思いを馳せてみるというのも楽しいことではないでしょうか。

編集 桑野 雅彦