

Smart Sockets

Through the use of simple ASCII type instructions it is possible to use this device to produce arrays of many display devices, of similar and different types. The device features 8 built in numerical fonts, 2 special fonts for use with nixie tubes and bargraph displays, and 10 transition effects. The standard Burroughs alphanumeric font is pre-programmed and in addition to this there are 124 user definable characters. The industry standard 9600 Baud, 8 data bits, No parity, 1 stop bit, 8N1, protocol enables almost any computer, handheld PDA, or microcontroller to communicate with the display elements directly, allowing for rapid development, and enabling some complex display routines to be generated from very simple commands.

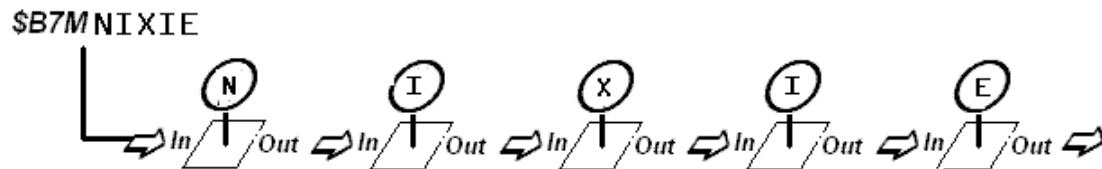
General Information

It is not unusual to want to generate complex display routines, only to find that you run out of memory, or have reached limits of performance of the controlling devices such that progress is hampered and compromises have to be made.

The idea behind this device came from a need to generate complex display routines through the use of a microcontroller. Having good programming skills is usually an advantage in such situations but even then it can become difficult to make a display which can respond to variable data in addition to the host performing the other functions which it has been assigned to do.

From a hardware point of view it is also convenient to mount all of the driving components on the same board as the B7971. With the exception of the power supply and data lines, there are no other connections required to make these boards produce the type of display which meets the modern expectations of complexity and visual interest. Each cathode has its own current limiting resistor fitted, in accordance with the original Burroughs recommended best practice. This means that a constant light output is guaranteed from each segment to make a uniform display regardless of how many segments are illuminated.

The 'smart' aspect of the device comes from the way in which each socket communicates with its neighbour in order to establish its own position in the display array. This means that if you send an ascii text string such as ' NIXIE ' to a display array of 5 sockets each character is displayed on the correct display with respect to it's position in the text string. So the first tube displays the 'N' , the next tube the 'I', and so on.....



A flexible data input format allows for the characters, fonts, effects and the speed of those effects to be changed in ways which suit different programming constructs and the different types of terminal programs which can be used to control the sockets.

The Instruction Set

Instructions are always sent to the first socket in an array. Each socket in turn resends the instructions to the following socket and extracts any useful information intended for itself. Each socket is aware of its position in an array such that it can calculate which character in a string of characters is to be interpreted

All instructions take the following form -

< 3 Character Header > + < Single Character Command > + < Command Values > + < ENTER >

The header and the command characters must all be in upper case. (Replace '<ENTER>' with ascii character 13 where necessary.)

The three character header is always = \$B7, which allows the sockets to be connected to an existing data bus with minimal risk of it being affected by an existing flow of data . Message validity is checked upon the receipt of the final <ENTER>.

COMMANDS

There are two types of commands. Display commands and resource commands.

DISPLAY COMMAND

M Group message string

RESOURCE COMMANDS

F Set individual fonts

E Set individual effects

S Set individual effect speeds

W Write a bit pattern to an eeprom location to create a user defined character

U Set the underscore segment on or off at any position in the array

Group Message String Command

The group message command is the most common message in most cases. With this command you are transmitting the actual display data to the array of display devices.

FORMAT = **\$B7Mxxxxxxx <ENTER>** (where xxxxxx is the string to display)

For example, if you wish to display the string ' NIXIE' across the array then send the following command: **\$B7MNIXIE <enter>** : Note there are no spaces or comma's.

The Null Character '!'

An important character is the exclamation mark " ! "

When this character is used in a command it is interpreted as 'no instruction' to the socket of the same relative position as the null character has in the string. This is useful in a message because you can send an instruction that only changes one character and leaves the existing display unchanged. For example, to set the character in the 4th position to the letter 'S' and leave the rest of the display untouched use the following command.

\$ B 7 M ! ! ! S ! ! ! ! ! ! <ENTER>

The null character can also be used in the resource commands for Font, Effect and Speed.

The Resource Command Structures

Resource commands are a useful way to set the fonts, effects and the speed of the effect used at the next transition. The current display is not altered during the programming of effects, fonts and speeds, which allows them to be sent 'silently' in the background. Even if a transition or effect is in progress the new resource commands will still be received and the new values stored for the next transition. A different font can be assigned to each position in the array, should it be desired. Also, different transition effects and effect speeds can be run in any character position

F Set individual fonts

FORMAT = **\$B7Fxxxxxx<ENTER>**

Where xxxxxx are the new fonts to be used, relative to their position in the display array. Valid font numbers are 0 to 9 and ' U '

EG: To change the fonts of the first and third tubes to font 3, and all other positions to font 1 the command to send is **\$B7F313111<ENTER>**

E Set individual effects

FORMAT = **\$B7Exxxxxxx<ENTER>**

Where xxxxxx are the numbers of the new effects, relative to their position in the display array. Valid effect numbers are 0 to 9

EG: To change the effects of the second and fourth positions to 5 and all other positions to effect 2 the command is **\$B7E252522<ENTER>**

S Set individual effect speeds

FORMAT = **\$B7Sxxxxxx<ENTER>** Where xxxxx are the new effect speeds, position dependant. Valid Speed Number values are 0 to 9

This command structure works in the same way as \$B7F and \$B7E above.

U Set the underscore segment on or off at any position in the array

\$B7U xxx1xx0xx<ENTER> Where 1 = underscore on in that position, 0 turns off the underscore for that position, and 'x' can be any other character. Only 0's and 1's are responded to when the underscore command has been sent.

Underscores are not turned on immediately but their requested status either on or off is remembered for the next time a character change command is sent to the same position. For example , sending \$B7U1!!!! will instruct the socket to turn on the underscore at position 1at the next character transition,

so afterwards if you send \$B7M8, then the first position will display the character '8' and the underscore will come on at the same time

The underscore inherits the transition effect in use for the characters.

Where there is a user defined character being displayed which makes use of the underscore, any instructions to set or clear the underscore are ignored.

The null character can be used with fonts and effects just as in the same way as it is used in character messages. In this way a new command can be transmitted by the host controller without it needing to have remembered the current settings in order to overwrite them correctly.

EG, to change the font in position four to font number eight, leaving all other fonts unchanged.....

\$ B 7 F ! ! ! 8 ! ! ! ! ! <ENTER>

W Write a bit pattern to eeprom memory to create a user defined character, (UDC)

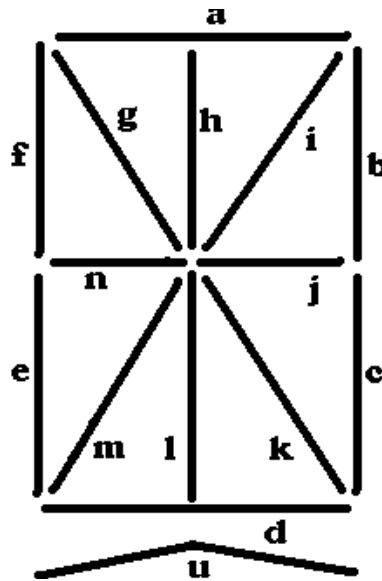
The characters which can be redefined are stored in an area of eeprom reserved specifically for the user, in no way can making changes to the UDC's affect program operation or affect the built in fonts, which makes dealing with UDC's a very safe practice. The use of eeprom memory means that the UDC's are retained after a power failure.

.

The characters which can be redefined are assigned to ascii characters which are easily referenced by using their character symbol, either directly from the keyboard of your terminal device, or by sending the ascii character value from the controlling device.

FORMAT = \$B7Wcbbbbbbbbbbbbbbb<ENTER>

Where **c** = the ascii character to be reprogrammed and **bbbbbbbbbbbbbbb** is the bit pattern to use to make the new character.



The segments of the B7971 and ZM1350 are denoted by letters. The group of letters 'bbbbbbbbbbbbbb' in the command refers to the segments in the following order abcdefghijklmnu. (where 'u' is the underscore).

Once you have designed the new character, by entering the assignment of the segments into the corresponding position they occupy in the command, where a '1' denotes an on cathode and a '0' denotes an off cathode segment, the new character will be stored.

Example Showing The Writing of a UDC Bit Pattern

To produce a character which is made up from segments a,b,c,d and j.
And to store this new character in the location currently assigned to the character 'c', the command to produce this change is

\$B7Wc111100000100000<ENTER>

In order to make use of the new character include it into a display command string, referring to it by its original ascii definition, in this case 'c'. It is important to note that the font which is assigned to the user defined characters does not have a number, but a single letter, the letter 'U' (in uppercase). So for example, to display this new character on position 4 tube we first need to set the number 4 position to display the UDC

\$ B 7 F ! ! ! U ! ! ! ! <ENTER>

And now to display it

\$ B 7 M ! ! ! c ! ! ! ! ! <ENTER>

Where c= the character and U = user defined character font

The User Definable Characters

Characters which can be redefined are shown below.




















It is important to note that programming the UDCs does not affect the inbuilt text font, which remains available when any font number 0-9 is selected

With the exception of Char 13, and the “!” sign, all characters from 0-58 can also be reprogrammed.

AsciiCode	Character	AsciiCode	Character	AsciiCode	Character	AsciiCode	Character	AsciiCode	Character	AsciiCode	Character	AsciiCode	Character	AsciiCode	Character
58	:	59	;	60	<	61	-	62	>	63	?	64	@	65	A
66	B	67	C	68	D	69	E	70	F	71	G	72	H	73	I
74	J	75	K	76	L	77	M	78	N	79	O	80	P	81	Q
82	R	83	S	84	T	85	U	86	V	87	W	88	X	89	Y
90	Z	91	[92	\	93]	94	^	95	_	96	-	97	a
98	b	99	c	100	d	101	e	102	f	103	g	104	h	105	i
106	j	107	k	108	l	109	m	110	n	111	o	112	p	113	q
114	r	115	s	116	t	117	u	118	v	119	w	120	x	121	y
122	z	123	{	124		125	}	126	~						

Initial UDC Settings

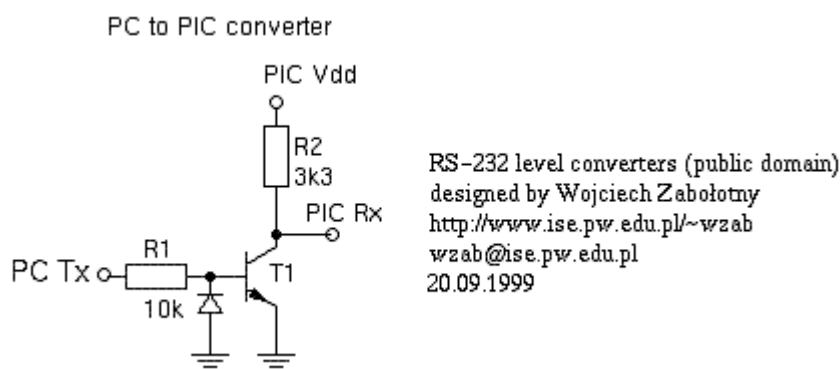
Some UDC Characters have already been programmed. These characters can be overwritten as required.

Code		Set	Code		Set	Code		Set	Code		Set	Code		Set
58	:		59	;		60	<		61	=		62	>	
63	?		64	@		65	A		66	B		67	C	
68	D		69	E		70	F		71	G		72	H	
73	I		74	J		75	K		76	L				

Data format information

Data Rate	Data Bits	Parity	Stop Bits	Flow Control
9600bps	8	N	1	None

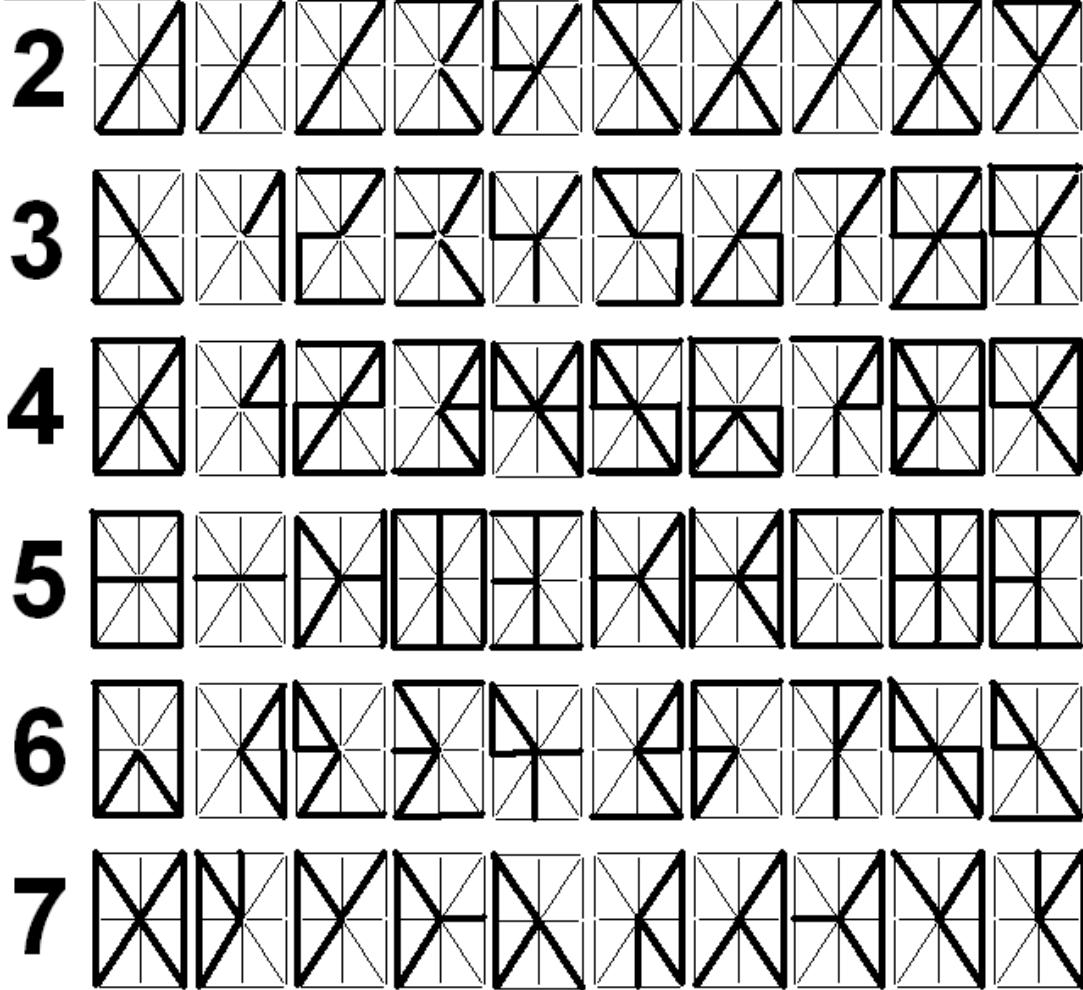
The default data level is TTL compatible and makes driving arrays of sockets with a microcontroller a very simple task. If you wish to use a PC, or any other device that transmits RS232 data levels the use of a level shifter is recommended. One of the most common is the MAX232 device. Many equivalents are also available. The MAX233 does not require the use of additional capacitors at all, which is a bonus, and the MAX232A requires you to use only 4 capacitors with a value of 0.1uF. You do not need to use a MAX232 or equivalent, a simple level shift arrangement can be achieved using the following circuit.



The Fonts

- 0 = Seven Segment – Standard Seven Segment Display Font
- 1 = Burroughs Fourteen Segment – Standard 14 Segment Display Font
- 2 = Leaning – Numbers an Italic Emphasis
- 3 = Formal – Digits Which Might Appear Somewhere Important
- 4 = Chamfer – A Font For Carpenters and Structural Engineers
- 5 = Secret – Lean Your Head Over To The Left
- 6 = Caitlin – My Daughter's Hard Work, Proving Fontmaking is Childsplay
- 7 = Clockalogue – When Incremented Sequentially The Time Ticks By
- 8 = Linear Bargraph – The Number of Lights Lit Increases
- 9 = Nixie – Standard 0 to 9 Independent Cathode Drive
- U = User Defined Characters

Font



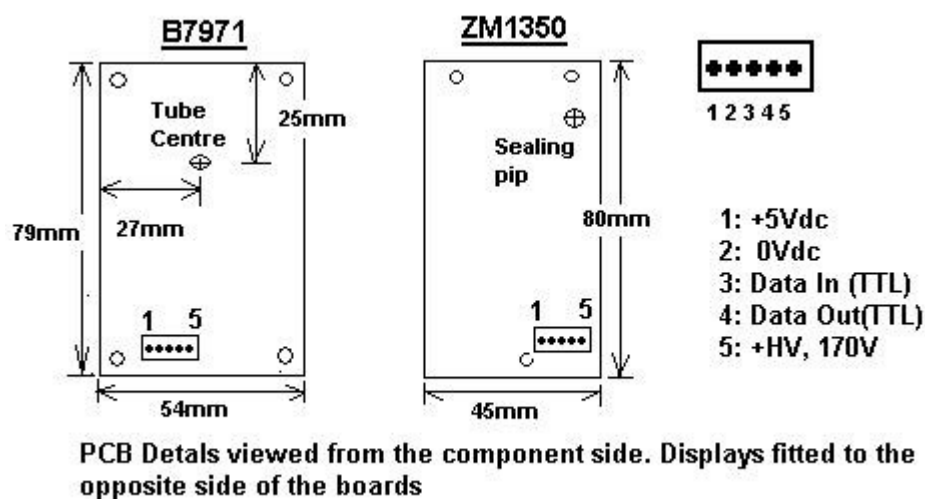
The Effects

- 0 = No effect, instant change
- 1 = Cross-fade
- 2 = Jump Fade (instant on, fade out)
- 3 = Fade out, fade in
- 4 = Zipper, bilateral downward wipe and refresh
- 5 = Shifter, characters wiped left to right *
- 6 = Segment deconstruction and rebuild
- 7 = Spin, half cycle
- 8 = Spin, full cycle
- 9 = Radar

*Used sequentially this can give the effect of a wipe across all tubes starting at the left and moving to the right

Advanced Use

With some clever programming, it would be possible for the host controller to properly parse the string being sent so that it will fit on the display correctly without overflowing. In order to do this the host will need to know how many display elements or smart sockets make up the display array. It is possible for the host to read the value being transmitted by the last socket in the array at startup. Doing this will provide a binary value of the number of sockets in the array. At startup the first socket assigns a value of 1 to itself and communicates with its downstream neighbour such that the next socket in line calls itself number 2...and so on down the line. This only happens once during each startup cycle, within 100mS of power being applied.



Details of a typical wiring harness layout

