FREE SUPPLEMENT

HOW TO USE GRAPHICS LIQUID CRYSTAL DISPLAYS WITH PICS

JOHN BECKER

A step-by-step guide to understanding and using pixel-matrixed graphics l.c.d.s with your PIC microcontroller projects.

GRAPHICS liquid crystal displays have been available for several years. It would appear, though, that *EPE* readers have not successfully explored them. At least, that seems the logical conclusion since we have never been offered a design which uses them.

One reason may be that the prices of such devices have, in many instances, been somewhat expensive. Whilst many continue to be pricy for the average hobbyist, less expensive ones have been making their appearance.

Possibly the principal reason we have not been offered working designs is that readers have not been able to obtain, let alone interpret, the data sheets associated with them.

The latter stumbling block very much faced the author when he decided that he would like to know how to use graphics displays. Intermittently over several days, he scoured the Internet in search of their manufacturers and suppliers. As it turned out, there are quite a few around the globe, but when it came to obtaining data sheets – well, that was a totally different matter.

DATA DENIAL

Farnell appeared to have a selection of displays within a reasonable price range, but stated "no parametric data available". Attempting to Net-search for the manufacturer of these devices, Perdix, only revealed countless sites to do with *partridges* (a bird for which the Latin and Greek name is *perdix*!)

Whilst research had showed that RS Components supplies graphics displays, the only data sheet (RS 298-4613) available turned out to be specific to a development kit which uses them.

However, in the RS catalogue, the manufacturer of their displays is quoted as Powertip. Doing a Net-search, and eventually accessing Powertip's web site in Taiwan, rudimentary data on the devices was located. But, frustratingly, Powertip denies access to its data sheets by those who are not registered distributors.

Contacting the technical department at RS, the author was put in touch with an agent who imports from Powertip. This company sent Powertip's data sheet, a document which might, perhaps, be understandable to those already familiar with graphics l.c.d.s but is certainly not conducive to teaching those who do not. Its intelligibility is also marred by having been translated by someone inadequately familiar with English. Gross errors of fact were spotted as well.

To cut short a lengthy and convoluted tale, no manufacturer or supplier could be found who had adequate data for graphics displays available for download.

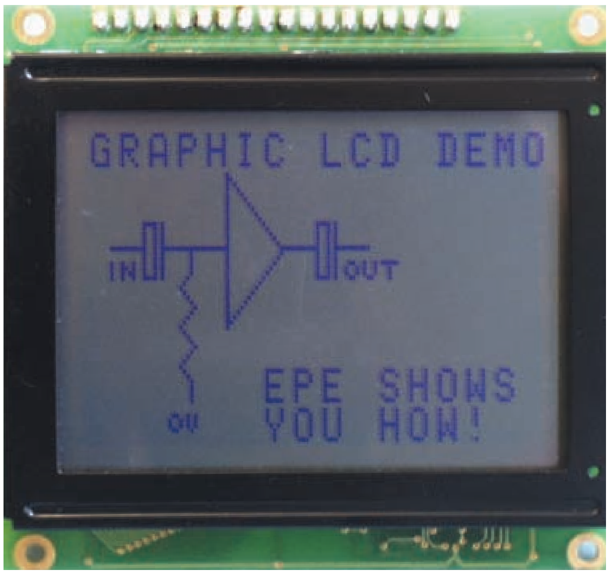


Photo 1. Graphics l.c.d. screen showing the display generated by the author's first demo program.

TOSHIBA T6963C

During the Net searches, however, various manufacturers had stated that their displays were controlled by the Toshiba T6963C chip, the same device as used by Powertip. Seemingly, then, the control architecture offered by the T6963C could be regarded as an "industry standard", and thus worth pursuing further through a Powertip display, the PG12864.

Toshiba state that the T6963C is an l.c.d. controller that has an 8-bit parallel data bus plus control lines for reading or writing through a microcontroller interface (such as a PIC).

It has a 128-word character generator ROM (read only memory) which can control an external display RAM (random access memory) of up to 64 kilobytes. It can be used in text, graphic and combination text-and-graphic modes, and includes various attribute functions.

Searching Toshiba's web site, the T6963C was eventually located (under Analogue & Peripherals/LCD Driver!), and its 46 page data sheet downloaded.

This data turned out to be the key to getting to grips with graphics l.c.d.s. Not so much because the data sheet was intelligible (which it did not become until much later), but because it gave programming examples of controlling the T6963C, albeit written in a microcontroller command language unknown to the author.

There were, though, sufficient commands whose structure appeared to be close to some other machine code dialects with which the author is familiar, for a translation to PIC microcontroller language to be attempted.

Photo 2. Toshiba's demo display.

Success was achieved when the display in Photo 2 appeared on the author's Powertip PG12864 64 × 128 pixel graphics l.c.d. screen – eventually!

POWERTIP PG12864

Whilst it is the Powertip PG12864 graphics display (RS 329-0329) used in the demos discussed, the commands are relevant to *any* graphics l.c.d. which uses the Toshiba T6963C controller, although the pin count/order may differ between display types. The PG12864 has the pinouts shown in Fig.1.

The PG12864 has a full dot-matrix l.c.d. structure consisting of a visible screen area having 128 dots × 64 dots (8192 dots in total). There are eight data lines, D0 to D7, and six control lines comprising WR, RD, CE, CD, RST and FS, names which will be clarified shortly.

The display has a single positive supply line (Vdd) at pin 3. The recommended working voltage is 5V, with an absolute maximum of 7V. There are two 0V connections, of which GND (pin 2) is the signal ground (Vss), and FG (pin 1) is the ground connection for the display's metal frame (bezel).

Display contrast is controlled via pin 4, named as CX in Fig.1 but can also be referred to as V0. This pin is normally connected to a negative voltage supply, of about –5V, via a contrast-adjusting preset potentiometer of typically 10kΩ to 25kΩ.

A summary of the pin functions is given in Table 1.

Which brings us to the first of the major discrepancies found in Powertip's own data sheet (be warned if you obtain it!):

1. Powertip quote data line D0 as being MSB. It's not. D0 is LSB (see Table 1).

2. Powertip also show an incorrect circuit diagram for the control of the screen contrast. The control pot is shown wrongly connected across pin 3 (+5V) and pin 9 (RST) with the wiper on pin 4 (CX). Furthermore, pin 9 is marked as V_{EE} (–VE). This configuration does not work.

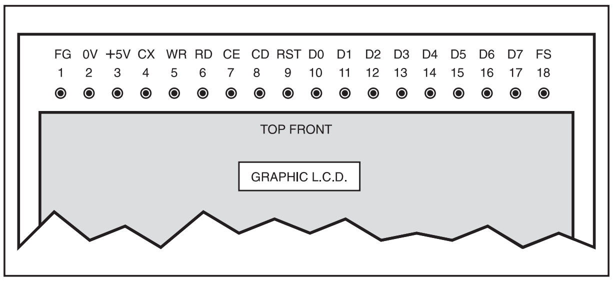


Fig.1. Pinouts of the Powertip PG12864 graphics l.c.d. module.

Table 1. Graphics l.c.d. pinout functions.

Pin	*Symbol*	*Function*
1	FG	Frame ground (connected to metal bezel)
2	GND	Signal ground supply (Vss)
3	+5V	Positive supply for logic (Vdd)
4	CX	Negative supply (V0) for l.c.d. contrast (–3·5V approx)
5	WR	Data write (active low)
6	RD	Data read (active low)
7	CE	Chip enable (active low)
8	CD	CD = 1, WR = 0: command write

		CD = 1, WR = 1: command read
		CD = 0, WR = 0: data write
		CD = 0, WR = 1: data read

9	RST	Module reset (active low)
10–17	D0–D7	Data bus (D0 = LSB, D7 = MSB)

18	FS	Font select:
		FS = 0: 8 × 8 dots font
		FS = 1 (or open–circuit): 6 × 8 dots font

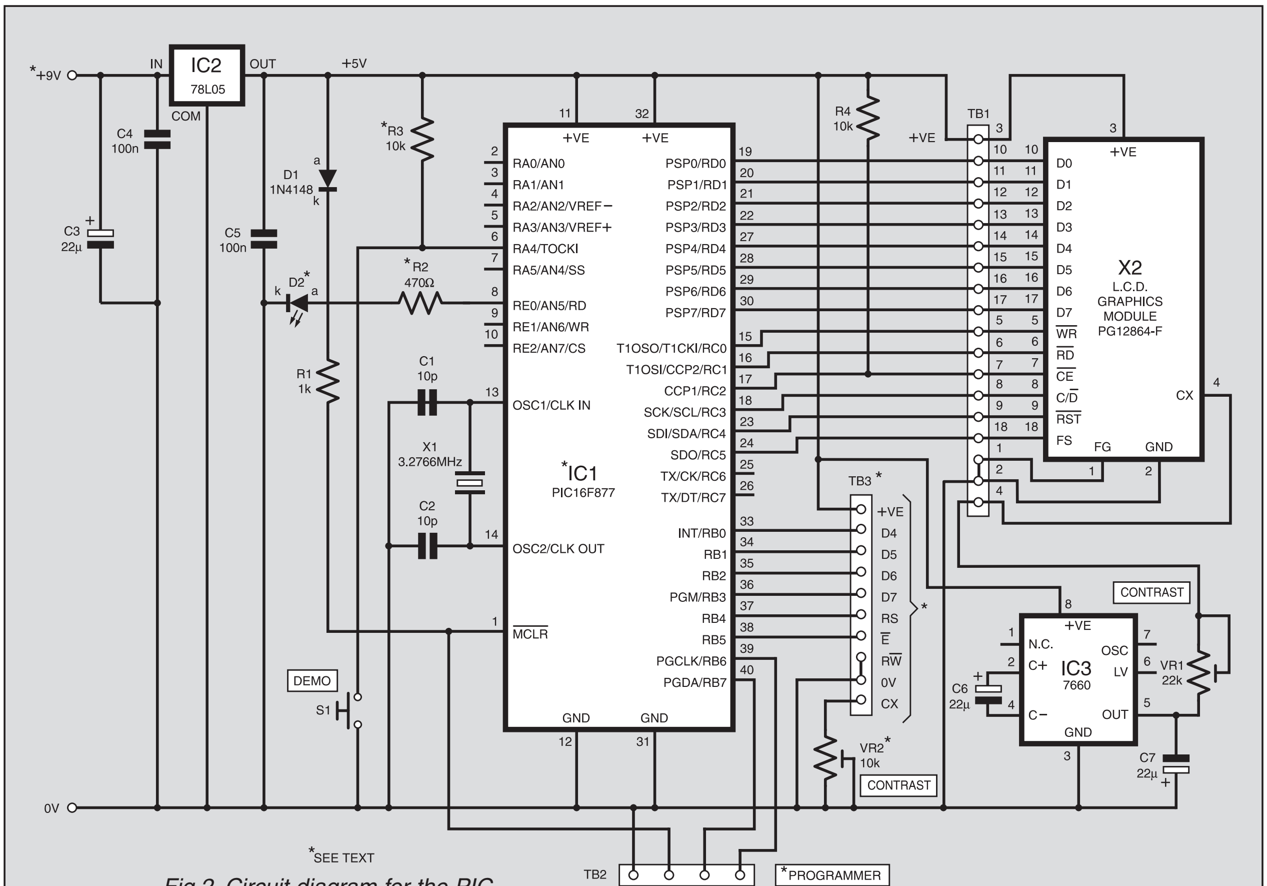
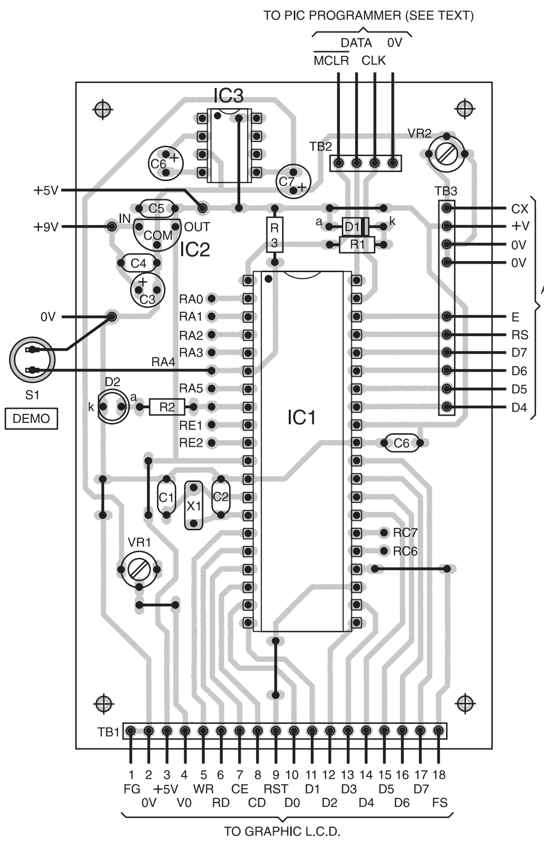


Fig.2. Circuit diagram for the PIC-controlled Graphics L.C.D. Demo.



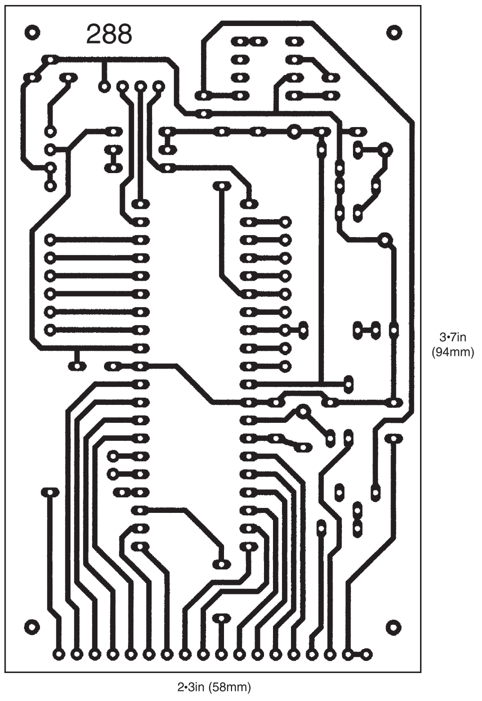


Fig.3. Component and full size copper foil track master pattern for the Graphics L.C.D. Demo printed circuit board.

Pin 9 (RST) does not go to –VE. It is held at logic 1 (+5V) for normal control chip operation and may be taken (briefly) to 0V to reset the chip, but it *must never be taken negative*, nor should one side of the control pot ever be connected to this pin.

Experimentation showed that the control pot's wiper *should* be connected to pin 4. One outer terminal of the pot then connects to a negative supply of, for example, –5V. The unused pot terminal is best connected to the wiper. This is illustrated in the demo circuit diagram (Fig.2.). The preset is adjusted until the desired contrast is observed.

The Toshiba data sheet does not discuss l.c.d. contrast setting, whilst that from RS is highly ambiguous on the subject.

DEMO CIRCUIT

The circuit diagram which the author used in his examination of the PG12864 display is shown in Fig.2. The circuit includes a PIC16F877 microcontroller (IC1), a crystal (X1) and its two capacitors (C1, C2), a 5V voltage regulator (IC2), a negative voltage inverter (IC3) with its two capacitors (C6, C7), the PG12864 l.c.d. display (X2) which is connected to the PIC via connector pins TB1, contrast controlling preset (VR1) and switch S1.

Components R1 and D1 are included so that the PIC may programmed on–board via connector TB2 and a suitable PIC programmer, such as the author's *PIC Toolkit Mk2* (May–June '99).

Resistor R2 and l.e.d. D2 were used by the author when originally translating the Toshiba demo program, the l.e.d. being set on or off at strategic points in the developing software. They have been left in and may be similarly used by readers when writing their own software. The control line is PORTE pin 0 (RE0).

Resistor R4 is included to keep the l.c.d.'s CE line high when the PIC is being programmed (so avoiding random displays appearing on the screen during that operation).

Resistor R3 biases high open-collector pin RA4 allowing demo-stepping switch S1 to operate correctly.

Components VR2 and TB3 are discussed in a moment.

A d.c. supply of between about 7V and 12V (nominally stated as +9V) can be used for powering the circuit via the 5V regulator, IC2.

Incidentally, the crystal clock frequency is not a critical matter and other frequencies could be used. If a crystal of greater than 4MHz is used, though, the PIC's configuration bit OS1

COMPONENTS

Approx. Cost Guidance Only **£50** *excluding case & batt.*

Resistors

R1	1k
R2	470Ω
R3, R4	10k (2 off)

All 0·25W 5% carbon film.

Potentiometers

VR1	22k (or 25k) min. preset, round
VR2	10k min. preset, round (see text)

See SHOP TALK page

Capacitors

C1, C2	10p ceramic, 5mm pitch (2 off)
C3, C6, C7	22µ radial elect. 16V (3 off)
C4, C5	100n ceramic, 5mm pitch (2 off)

Semiconductors

D1	1N4148 signal diode
D2	red l.e.d. (see text)
IC1	PIC16F877 microcontroller, pre-programmed (see text)
IC2	78L05 +5V 100mA regulator
IC3	7660 negative voltage converter

Miscellaneous

S1	min. s.p. push-to-make switch
X1	3·2768MHz crystal (see text)
X2	graphics l.c.d. module, Toshiba T6963C-based, 64 x 128 pixels, e.g. Powertip PG12864-F (see text)

Printed circuit board, available from the *EPE PCB Service,* code 288; 8-pin d.i.l. socket; 40-pin d.i.l. socket; terminal pin header strips and connectors (see text); 18-way ribbon cable (a few centimetres); p.c.b. supports (4 off); solder, etc.

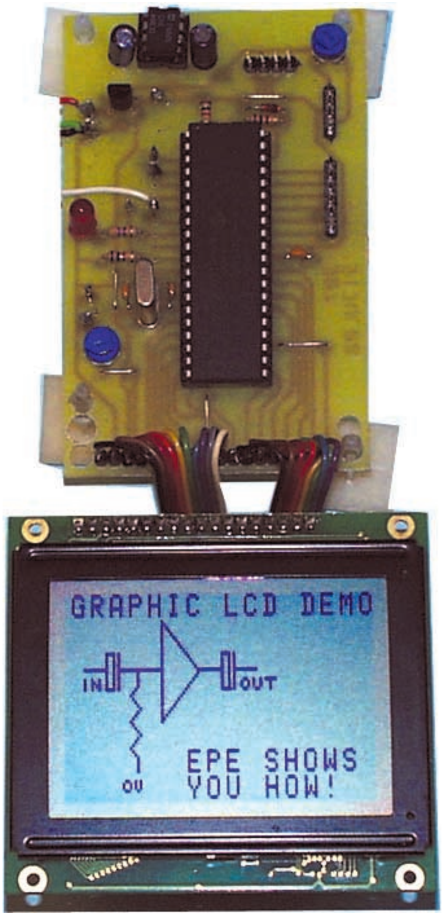


Photo 3. Demo printed circuit board assembly.

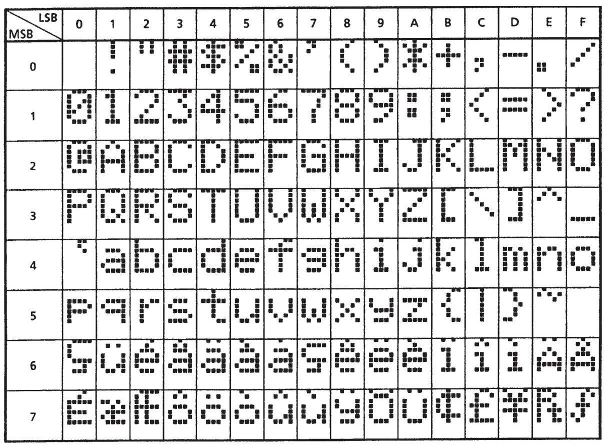


Fig.4. Character set for the Powertip PG12864 display (note that the last two lines may differ in some modules).

should be set high and OS0 set low (see later).

For his own demo board, the author in fact uses a 5MHz crystal, which does fractionally speed up Demo 8.

It may also be worth bearing in mind that the author's forthcoming follow–up constructional article, in which a PIC16F877 and the same graphics l.c.d. are used, also requires a 5MHz crystal.

PRINTED CIRCUIT BOARD

It is emphasised that unless you have a PIC programmer, there is no point in building this design since much of the discussion here concerns experimental software changes that you are recommended to try. These changes, once made, need the software to be re-assembled and downloaded to the PIC.

A printed circuit board design and its component layout are shown in Fig.3. This board is available from the *EPE PCB Service*, code 288.

You will observe that the p.c.b. includes components VR2 and TB3 towards the right. These are the points at which the author also included an ordinary alphanumeric l.c.d. so that various routines could be monitored during the development of the software translation from Toshiba to PIC.

The holes and tracks have been left intact so that the p.c.b. could be used as a future general-purpose development board with either type of l.c.d., and in conjunction with a PIC16F877.

A circuit diagram for the second l.c.d. is not included here, but the pin order and functions, and the controlling software subroutines, are the same as used in other recent "normal" l.c.d. projects designed by the author and published in *EPE*. The connections can be ascertained by studying any of those. In Fig.2, the connections are shown as TB3, with VR2 as the contrast control.

CONSTRUCTION

It is expected that all who build the circuit will be sufficiently experienced not to need constructional advice. It is recommended, though, that sockets are used for IC1 and IC3, and that pin header strips are used for the TB1 to TB3 connections.

Ideally, a graphics l.c.d. should be purchased that already has a suitable pin connector wired to it (see this month's *Shoptalk* page). The connections to the p.c.b. are in the "natural" order of the PG12864 l.c.d. used (as shown previously in Fig.1).

PIC PROGRAMMING

Having built the board and proved its workability, three lots of software need to be programmed into the PIC, one now and two later.

Before doing so, though, the PIC needs to be configured with the following settings (which are all *PIC Toolkit Mk2* default settings for a PIC16F877 running at 4MHz):

CP1	CP0	DBG	NIL	WRT	CPD	LVP
1	1	1	1	1	1	0
BOR	CP1	CP0	POR	WDT	OS1	OS0
0	1	1	0	0	0	1

Note that Logic 1 and Logic 0 in the settings do NOT necessarily mean that the function is on/off respectively – refer to the PIC '87 data sheet if you need to know more (also see earlier regarding the oscillator rate).

For the first part of the discussion that follows, you need the Toshiba demo program loaded into the PIC, GRAPHEPE.OBJ (source code name GRAPHEPE.ASM). The software is written in TASM, but *PIC Toolkit Mk2* can translate between TASM and MPASM if the latter is the programming language you are used to.

Two other program files will be loaded into the PIC later on.

The software is available free via the *EPE* web site, or on 3·5-inch disk (for which a nominal handling charge is made) from the Editorial office. See *Shoptalk* for details of both matters.

Having loaded the Toshiba demo, the display shown earlier in Photo 2 should be seen. It may be necessary for preset VR1 to be adjusted before the image is clearly visible.

It is this demo result which is first discussed before moving on to the author's demos, for which various exercises are suggested at some points.

DISPLAY STRUCTURE

Before starting to discuss programming detail, it is necessary to understand the physical arrangement of the ways in which data can be shown on the l.c.d. screen. There are three options, which can be summarised as:

1. **Alphanumeric text display** using the built–in character generator (as with any standard alphanumeric l.c.d.). 128 characters are available, as shown in Fig.4, and which are called by their own location numbers. In a very loose sense they can be regarded as the equivalent of ASCII characters. The addressing number order runs from zero to 127. Writing any of these values to the screen displays the "text" character associated with it. The characters in lines 7 and 8 may be slightly different in some variants of the PG12864 display.

2. **User–defined character generation and display**. Again this is similar to the facilities available on standard alphanumeric l.c.d.s, but the quantity of characters that can be simultaneously stored is far greater, 128 compared to the typical eight. The addressing order runs from 128 to 255. Writing any of these values to the screen displays the character that the user has created and allocated to the address value.

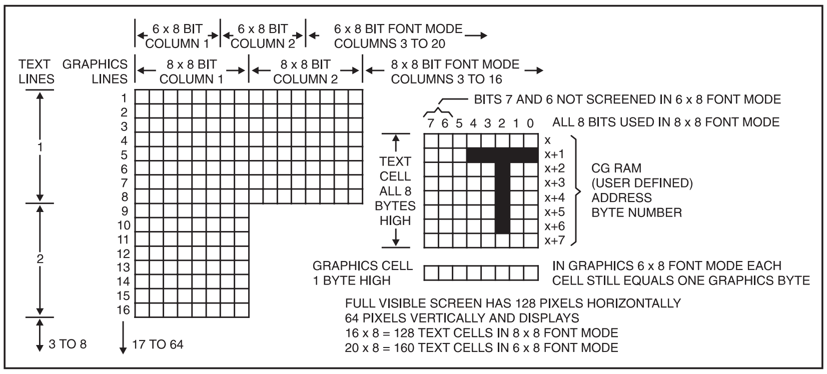

6 x 8 BIT COLUMN 1
6 x 8 BIT COLUMN 2
6 x 8 BIT FONT MODE COLUMNS 3 TO 20
TEXT LINES
GRAPHICS LINES
8 x 8 BIT COLUMN 1
8 x 8 BIT COLUMN 2
8 x 8 BIT FONT MODE COLUMNS 3 TO 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1
2
3 TO 8
17 TO 64
BITS 7 AND 6 NOT SCREENED IN 6 x 8 FONT MODE
7 6 5 4 3 2 1 0
ALL 8 BITS USED IN 8 x 8 FONT MODE
TEXT CELL ALL 8 BYTES HIGH
x x+1 x+2 x+3 x+4 x+5 x+6 x+7
CG RAM (USER DEFINED) ADDRESS BYTE NUMBER
GRAPHICS CELL 1 BYTE HIGH
IN GRAPHICS 6 x 8 FONT MODE EACH CELL STILL EQUALS ONE GRAPHICS BYTE
FULL VISIBLE SCREEN HAS 128 PIXELS HORIZONTALLY 64 PIXELS VERTICALLY AND DISPLAYS
16 x 8 = 128 TEXT CELLS IN 8 x 8 FONT MODE
20 x 8 = 160 TEXT CELLS IN 6 x 8 FONT MODE

Fig.5. Pixel, column and row distribution for the graphics l.c.d.

3. **User–defined graphics detail generation** in which the "character" size is one pixel high by eight pixels wide (i.e. a single byte). Any value between zero and 255 can be written to the screen and the setting of the binary bits that make up that value determines whether a screen pixel is turned on or off.

For both character modes, the characters are all eight pixels high, but the width can be specified as six or eight pixels. That is, the character generation (font) can be set for 6 × 8 or 8 × 8 format.

Line FS controls the font choice, FS = 0 for 8 × 8, FS = 1 for 6 × 8. The pin has an internal pull–up resistor that holds it high (FS = 1) and the pin may be left unconnected if the 6 × 8 font is required (also see later).

In 8 × 8 font mode (FS = 0), the screen can display 16 characters horizontally and 8 vertically. In 6 × 8 font mode (FS = 1), the display format is 20 horizontal × 8 vertical characters. It is conventional to refer to the character display in terms of lines (horizontal) and columns (vertical). See Fig.5.

For the graphics mode, 64 character locations can be written to vertically. Horizontally, the quantity is determined by the width mode set for the characters, i.e. 20 or 16.

There are two screen memory areas to which data is written, known as the Text screen and the Graphics screen. The Text screen displays the built–in and user-defined characters. The Graphics screen displays only graphics data.

The l.c.d. can be programmed to display Text only, Graphics only, or Text and Graphics combined.

There are many additional display attribute features that can be implemented, such as highlighting, blanking, flashing, panning etc.

COMING NEXT

The next several sections of this discussion relate to the T6963C l.c.d. controller, and how Toshiba's example programs are interpreted using the PG12864 graphics display and a PIC16F877 microcontroller, resulting in the display shown earlier in Photo 2.

Following this, the author's own examples of PIC-microcontrolling the l.c.d. in a variety of situations will be described. In a future issue, this same graphics l.c.d. will be the display used in a PIC-controlled audio frequency oscilloscope (currently having the working title of *PIC G-Scope*).

There is lot of information discussed from hereon, but it is illustrated with working program examples, and with many points at which you can experiment with various commands in the author's own demos.

As usual with this type of article, the author tries to lead you carefully from step to step.

CONTROL MATTERS

Those of you familiar with alphanumeric ("intelligent'') l.c.d. displays will be aware that they can be operated in either 4-bit or 8-bit data mode. They can also be controlled by just two control lines, RS and E, and rely on a predetermined delay between sending bytes or nibbles of data.

The same is not true of graphics displays. Those using the T6963C can only be operated in 8-bit mode. They use five control lines and *require status check routines to be performed before each action*. Timed delays are not used, nor can they *be* used between data transfers.

There are, though, delay requirements concerning the order in which the data and control lines are taken high or low, a matter which is discussed now. Status checking will be examined shortly.

CONTROL LINES

Data is written to or read from the l.c.d. via the eight data lines D0 to D7. Three control lines are used in most read or write situations:

CD: selection of data or command function, 0 = Data, 1 = Command

CE: chip enable, 0 = enabled, 1 = disabled

RD or WR: read or write functions. These are two separate lines and the relevant one is taken low to be active, with the other remaining high.

The timing characteristics for the setting of the data and control lines are shown in Fig.6 and Table 2.

Since the l.c.d. includes its own oscillator, the timings shown are independent of the clock rate controlling a PIC microcontroller.

PIC PORT SETTINGS

In the demo programs, PIC PORTC is used for setting the l.c.d. control lines, and PORTD for the data input/output lines.

Some microcontrollers and microprocessors have internal registers which allow the same data port to be used either for input or for output without the user having to specify the port's function, other than by the *write* or *read* command.

For example, the required port (e.g. parallel port connector on a PC computer) has two separate register addresses, one for inputting data, the other for outputting it. These would be equated as values at the head of the program, e.g. OUTPORT = &H378 (output register), INPORT = &H379

Table 2. Toshiba T6963C Timing Values

Item	*Symbol*	*Min*	*Max*	*Unit*
C/D Set-up Time	t_{CDS}	100	–	ns
C/D Hold Time	t_{CDH}	10	–	ns
$\overline{CE}$, $\overline{RD}$, $\overline{WR}$ Pulse Width	t_{CE}, t_{RD}, t_{WR}	80	–	ns
Data Set-Up Time	t_{DS}	80	–	ns
Data Hold Time	t_{DH}	40	–	ns
Access Time	t_{ACC}	–	150	ns
Output Hold Time	t_{OH}	10	50	ns

Test Conditions: V_{DD} = 5·0V ± 10%, V_{SS} = 0V, Ta = –20 to 75°C

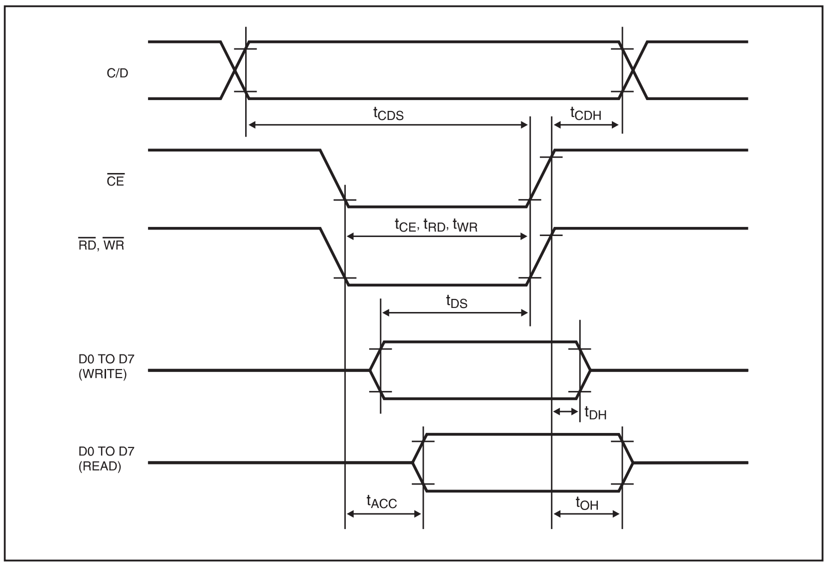


Fig.6. Timing waveforms for the Toshiba T6963C graphics l.c.d. controller, see also Table 2.

Table 3. Toshiba T6963C Status Register Settings

STA0 D0	Check command execution capability	0 : Disable 1 : Enable
STA1 D1	Check data read/write capability	0 : Disable 1 : Enable
STA2 D2	Check Auto mode data read capability	0 : Disable 1 : Enable
STA3 D3	Check Auto mode data write capability	0 : Disable 1 : Enable
STA4 D4	Not used	
STA5 D5	Check controller operation capability	0 : Disable 1 : Enable
STA6 D6	Error flag. Used for Screen Peek and Screen copy commands	0 : No error 1 : Error
STA7 D7	Check the blink condition	0 : Display off 1 : Normal display

(input register), both numbers referring to the same physical port.

To read data from the port, a command such as VALUE = INP(INPORT) would load the data present on the port connector and store it into the variable VALUE.

Similarly, to write data held in variable VALUE to the port, a command such as OUT(OUTPORT), VALUE would be used.

PIC microcontrollers, though, do not have this dual-function automatically available. A port's data direction register (DDR) has to have its input/output directions actively set from within the program prior to data input or output.

As you are no doubt aware, this is where PAGE and TRIS commands come into use in the PIC16x84, for example, setting the STATUS register Page (Bank) address through which the DDR is changed (STATUS bit 5 = high).

The PIC16F877, as used for this demo, has *two* STATUS register bits to be manipulated in order to enter the DDR setting mode, STATUS bits 5 and 6 (RP0 and RP1 – their full use will be discussed in a forthcoming *EPE* feature article). To set for Bank 1 (to access the DDR register), RP1 is set low, and RP0 is set high. To return to Bank 0 (for accessing the data port itself rather than its DDR), RP0 is returned low.

When Banks 2 and 3 are not used (as with this demo), it is convenient to define the setting of RP0 using the familiar PAGE commands, e.g.:

```
#DEFINE PAGE0 BCF STATUS,5
#DEFINE PAGE1 BSF STATUS,5
```

With RP1 held low, switching back and forth between DDR and data port addresses is simplified, and is the technique used in the demo programs.

STATUS CHECKS

Just as a PIC microcontroller has a STATUS register which informs of the results following various functions or commands (through bits C, DC and Z), so too does the T6963C. It is an 8-bit register of which seven bits are used, having the functions shown in Table 3.

In practice, there are only three forms of status check normally required, depending on the type of control function being used at that moment. There is an easy logic to status checks and examples of those used for different circumstances are illustrated in the author's own demo routines.

Let's take the most frequently used status check as a first example. It is used immediately prior to writing data (of any sort, display data or command data) to the l.c.d. It simply entails reading the STATUS register to check whether bits STA0 and STA1 are high (= 1), and if not, then waiting until they are. This is shown in the flow chart of Fig.7.

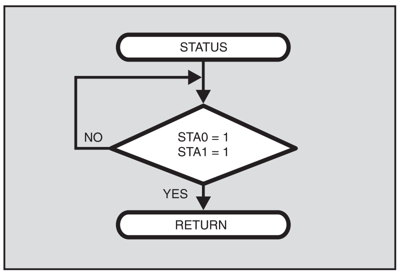


Fig.7. Flow chart for the most commonly used status check, which waits for bits STA0 and STA1 to become high (logic 1).

To explain the convention of the flow charts used in this discussion, the entry and exit points of the routine are indicated by the oval shapes enclosing, for example, the name of the routine and its end point.

Thus in Fig.7, the "working" aspect of the chart is simply that within the diamond shape. Here the question being represented is what status do STA0 and STA1 have. The question is repeated until they are both at logic 1, whereupon the routine ends.

An example of writing data to the l.c.d. is shown in the flow chart of Fig.8. It shows the status check, and then a rectangle stating the next action to be taken (write data) following the successful status check. After which the routine ends.

Immediately prior to reading the status register, command lines are set for CD and WR high, with CE and RD low:

CD	CE	RD	WR
1	0	0	1

This condition remains throughout the repeated checking of the status. Upon its successful conclusion, all four lines are returned high:

CD	CE	RD	WR
1	1	1	1

In the PIC program, the status checking flow chart becomes that in Fig.9.

Note that the square brackets statement [CALL CHECK3] indicates the command the PIC software issues in order to access the routine. Square brackets statements are used in other flow charts for a similar purpose.

The entry point address label of CHECK3 has been given because the routine checks if the value of input bits 0 and 1 is equal to 3 (binary 11, i.e. both high).

The PIC source coding involved is shown in Listing 1. Note that the RST bit is that which controls the reset of the T6963C (but does not perform such functions as

Listing 1. CHECK3 – Status check for PORTD RD0/RD1 (STA0/STA1) = 3. See also flow chart Fig.9.

```
CHECK3:
PAGE1               ; set for Bank 1 (DDR bank)
MOVLW 255
MOVWF TRISD         ; set PORTD for input
PAGE0               ; set for Bank 0 (Data port bank)
                    ; RST  CD  CE  RD WR
MOVLW %00011001     ;  1     1    0    0   1
MOVWF PORTC         ; set CE, RD low
NOP                 ; pause to allow port to stabilise
CK3:
BTFSS PORTD,0       ; PORTD bit 0 set?
GOTO CK3            ; no
CK3A:
BTFSS PORTD,1       ; PORTD bit 1 set?
GOTO CK3A           ; no
                    ; RST  CD  CE  RD  WR
MOVLW %00011111     ;  1     1     1   1     1
MOVWF PORTC         ; set controls high
NOP                 ; pause to allow port to stabilise
PAGE1
CLRF TRISD          ; set PORTD as outputs
PAGE0
RETURN
```

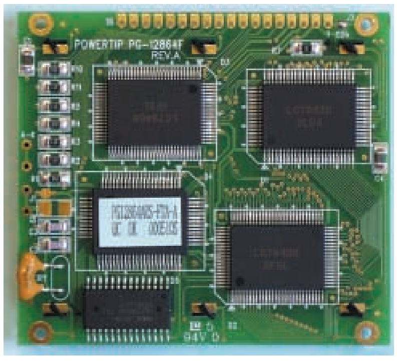


Photo 3. The minutely detailed rear view of the Powertip PG12864 display.

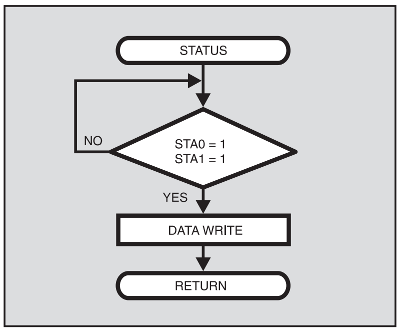


Fig.8. Flow chart for writing data to the l.c.d.

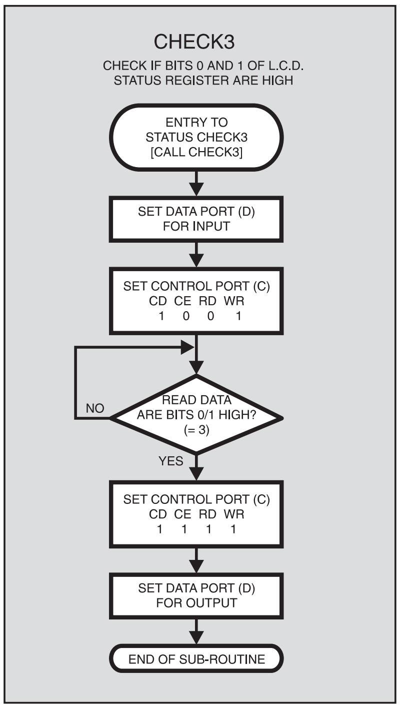

CHECK3
CHECK IF BITS 0 AND 1 OF L.C.D. STATUS REGISTER ARE HIGH
ENTRY TO STATUS CHECK3 [CALL CHECK3]
SET DATA PORT (D) FOR INPUT
SET CONTROL PORT (C) CD CE RD WR 1 0 0 1
READ DATA ARE BITS 0/1 HIGH? (= 3)
NO
YES
SET CONTROL PORT (C) CD CE RD WR 1 1 1 1
SET DATA PORT (D) FOR OUTPUT
END OF SUB-ROUTINE

Fig.9. Flow chart for status check named as CHECK3.

screen or memory clearing). It is held high throughout the normal use of the l.c.d., only being cleared briefly when the program commences.

In the Listings, note that the Labels are placed above the commands in order to conserve page space. In the full source code, they are to the left and the commands are indented as usual. In preparing this text for publication, other minor cosmetic changes have been made to some listings compared to the actual source code itself.

DATA WRITE

On entry to the data write routine (OUTDATA) the control lines are first set for data output in which line CD is taken low, with the other control lines set high:

CD	CE	RD	WR
0	1	1	1

The data to be sent is then placed on the data port output lines. Now, with CD remaining low, the CE and WR command lines are taken low, leaving RD high:

CD	CE	RD	WR
0	0	1	0

Next, and still leaving CD low, the CE and WR lines are again taken high:

CD	CE	RD	WR
0	1	1	1

After which the data write routine can be exited, leaving CD low. However, the author chooses to leave all control lines in the known setting of all high, so the previous routine is followed by returning CD high before exiting or performing the next required command:

CD	CE	RD	WR
1	1	1	1

The PIC source coding lines for the OUTDATA routine are shown in Listing 2 and associated flow chart in Fig.10.

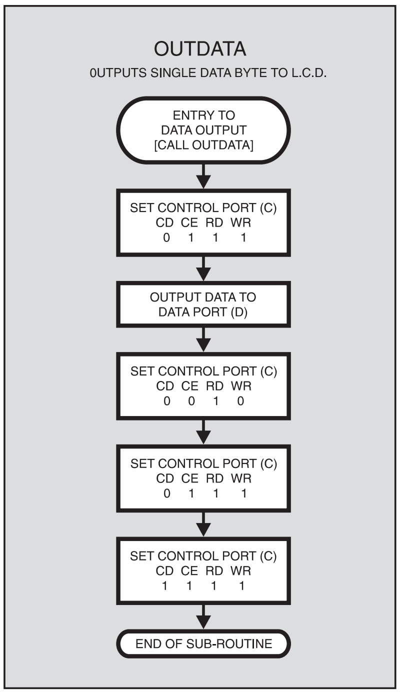

OUTDATA
0UTPUTS SINGLE DATA BYTE TO L.C.D.
ENTRY TO DATA OUTPUT [CALL OUTDATA]
SET CONTROL PORT (C) CD CE RD WR 0 1 1 1
OUTPUT DATA TO DATA PORT (D)
SET CONTROL PORT (C) CD CE RD WR 0 0 1 0
SET CONTROL PORT (C) CD CE RD WR 0 1 1 1
SET CONTROL PORT (C) CD CE RD WR 1 1 1 1
END OF SUB-ROUTINE

Fig.10. Flow chart for outputting data to the l.c.d.

Listing 2. OUTDATA – send data to l.c.d. routine. See also flow chart Fig.10.

```
OUTDATA:
MOVWF TEMPA        ; temp store val brought in on W
                   ; RST  CD  CE  RD  WR
MOVLW %00010111    ;  1    0   1   1   1
MOVWF PORTC        ; set CD low
MOVF TEMPA,W       ; get stored data
MOVWF PORTD        ; send data
NOP                ; pause to allow port to stabilise
                   ; RST  CD  CE  RD  WR
MOVLW %00010010    ;  1    0   0   1   0
MOVWF PORTC        ; set CD, CE, WR low
NOP                ; pause
                   ; RST  CD  CE  RD  WR
MOVLW %00010111    ;  1    0   1   1   1
MOVWF PORTC        ; set CE, WR high
NOP                ; pause
                   ;  RST  CD  CE  RD  WR
MOVLW %00011111    ;   1    1   1   1   1
MOVWF PORTC        ; set CD high
RETURN
```

TOSHIBA'S DEMO

Having set the first simple scenes, it seems best to illustrate the capabilities of a graphics l.c.d. controlled by a Toshiba T6963C by first discussing the sub-routines used in Toshiba's own demo program, the results of which you saw in Photo 2.

The program listings shown are the author's translations to PIC control language from the language used in Toshiba's original program (written for their microcontroller type TMPZ84C00P). Slight changes to Toshiba's program have been made apart from the translations.

The program is written for a 20 column × 8 line display, in 8 dots mode (font).

FIXED VARIABLES

As shown in our full program source code listing file, the PIC source code has its usual EQUates and #DEFINEs set at the beginning. Then follow fixed equates values for some specific commands, as specified by Toshiba, and shown in Listing 3.

Then, as said earlier, PORTC is set for control line output, and PORTD for data input/output.

It is worth noting that l.c.d. line FS (that which selects between font widths, FS = 0 = 8-bit, FS = 1 = 6-bit) is controlled by PORTC bit 5. The selection of which font mode is chosen is provided in the subroutine which sets the DDR registers for PORTC and PORTD. As said earlier, line FS has an internal pull-up resistor.

The choice of having FS high or low is then determined by the DDR setting of PORTC bit 5 (TRISC bit 5). With DDR bit 5 set for input (= 1), PORTC bit 5 presents a high impedance to line FS, which thus adopts the logic high status as set by the internal pull-up resistor.

With the DDR bit 5 set for output (= 0), line FS is thus controllable by the output value of PORTC bit 5. With Toshiba's demo, DDR bit is set high to use the 6 × 8 font. All the author's demo routines which write to PORTC have bit 5 set permanently low, which with DDR bit 5 set for output causes FS to be permanently set low, so selecting the 8-bit font.

Should you want 6-bit mode for another design, set DDR bit 5 for input, as with Toshiba's demo. Note that when programming the PIC *in situ*, the l.c.d. will show screen data in 6 × 8 font mode since PORTC is held in high impedance during programming.

DEMO SUBROUTINES

The PIC source code for calling each of Toshiba's demo routines to be discussed is shown in Listing 4. The first four routines are required to be run at the start of any program. They specify the address locations and column area of the Text and Graphics display memory areas.

The l.c.d.'s total available memory runs from addresses $0000 to $FFFF

Listing 3. Toshiba's fixed variables

```
TXHOME:    .EQU $40     ; text home (start) location
TXAREA:    .EQU $41     ; text area, i.e. number of active
                          columns
GRHOME:    .EQU $42     ; graphics home (start) location
GRAREA:    .EQU $43     ; graphics area, i.e. number of
                          active columns
AWRON:     .EQU $B0     ; autowrite on command
AWROFF:    .EQU $B2     ; autowrite off command
OFFSET:    .EQU $22     ; graphics offset
ADPSET:    .EQU $24     ; set address pointer command
PEEK:      .EQU $E0     ; read data from screen command
```

Listing 4. PIC source code for calling Toshiba's demo routines

```
CALL TEXTHOME     ; set Text Home address
CALL GRAPHHOME    ; set Graphic Home address
CALL TEXTAREA     ; set Text Area
CALL GRAPHAREA    ; set Graphic Area

CALL SETMODE      ; set Mode (e.g. OR mode, internal
                    Character Generator)
CALL SETOFFSET    ; set Offset register for character code
                    $80
CALL SETDISPLAY   ; set Display Mode (Text on, Graphics
                    and Cursor off)
CALL CLRTXT       ; clear Text Area
CALL WRITECG      ; write to external (user–defined)
                    character generator RAM
CALL WTDD         ; write Text Display data (internal CG
                    ROM)
CALL WTDD2        ; write Text Display data (external CG
                    RAM) upper part
```

Listing 5. TEXTHOME – set Text Home address. See also flow chart Fig.11.

```
CLRF ADRMSB       ; set Text Home address to $0000
CLRF ADRLSB
CALL CMDADR       ; send 2 bytes of address data
MOVLW TXHOME
CALL SENDCMD      ; send TXHOME command
RETURN
```

Listing 6. CMDADR – send command address to LCD. See also flow chart Fig.12.

```
CALL CHECK3              ; read status for DA0/DA1 = 3
MOVF ADRLSB,W            ; send address LSB
CALL OUTDATA
CALL CHECK3              ; read status for DA0/DA1 = 3
MOVF ADRMSB,W            ; send address MSB
CALL OUTDATA
RETURN
```

(64K bytes). The l.c.d.'s actual *visible screen* area, though, is 1024 bytes (1K) and so 64K of available memory area can be regarded as holding up to 64K/1K = 64 screen pages of data. As will be seen later, this allows for pages of data to be stored "behind the scenes'' and then called as required by simply changing the Text Home or Graphic Home addresses.

Toshiba state that Text data, Graphic data and user-defined CG RAM can be freely allocated to the full memory area, a matter on which they do not elaborate. It would seem logical, though, for the total area required for each data set to depend upon the total data required to be stored for that set.

Toshiba's demo, for example, has seven text letters, amounting to 7 × 8 = 56 bytes of data (each letter is eight bytes high) and eight user-created graphic symbols, making a further 8 × 8 = 64 bytes. Toshiba allocate the Text Home address to $0000 and the Graphics Home address to $0200, thus allocating a maximum of 512 bytes available for text use should the demo be expanded upon.

The routines which set these facts are TEXTHOME and GRAPHHOME. Referring to routine TEXTHOME in Listing 5 and Fig.11, the text address value ($0000) is set into the 2-byte address word consisting of bytes ADRMSB and ADRLSB, which in this instance are both cleared to zero.

The routine CMDADR (command address) is then called, in which the MSB/LSB address is sent to the l.c.d., as shown in Listing 6, and flow chart CMDADR in Fig.12.

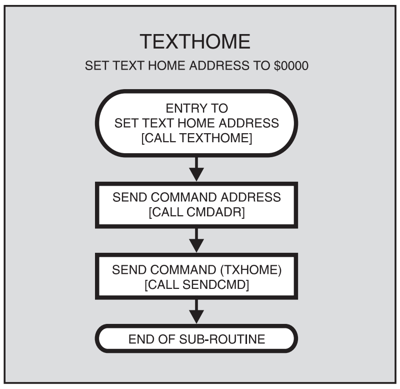

TEXTHOME
SET TEXT HOME ADDRESS TO $0000
ENTRY TO SET TEXT HOME ADDRESS [CALL TEXTHOME]
SEND COMMAND ADDRESS [CALL CMDADR]
SEND COMMAND (TXHOME) [CALL SENDCMD]
END OF SUB-ROUTINE

Fig.11. Flow chart for routine TEXTHOME.

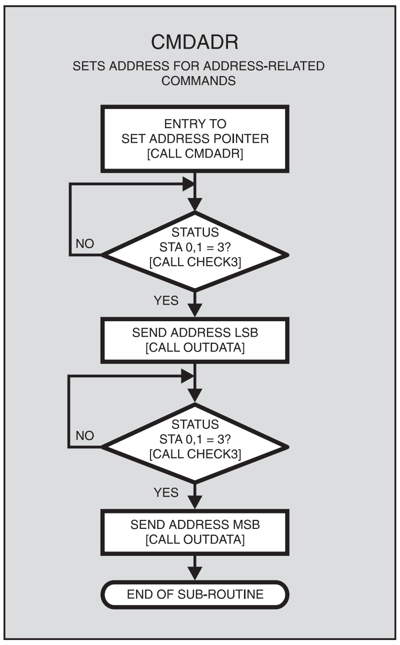

CMDADR
SETS ADDRESS FOR ADDRESS-RELATED COMMANDS
ENTRY TO SET ADDRESS POINTER [CALL CMDADR]
NO
STATUS STA 0,1 = 3? [CALL CHECK3]
YES
SEND ADDRESS LSB [CALL OUTDATA]
NO
STATUS STA 0,1 = 3? [CALL CHECK3]
YES
SEND ADDRESS MSB [CALL OUTDATA]
END OF SUB-ROUTINE

Fig.12. Flow chart for routine CMDADR.

As will be seen, the first action in CMDADR is to check the l.c.d. status via sub–routine CHECK3, as discussed earlier (Listing 1, Fig.9).

Next, the address LSB is sent to the l.c.d. via the OUTDATA routine, also discussed earlier (Listing 2, Fig.10). A further CHECK3 status check is made and the address MSB is sent, again via OUTDATA, and the routine is exited.

The address is now stored in the l.c.d. but has not been acted upon yet. It is brought into action at the end of Listing 5 by sending the TXHOME command to the l.c.d. via sub-routine SENDCMD.

The value of TXHOME, you will recall, was specified at the beginning of the program (see Listing 3) to be $40. This is a fixed command value that must be sent to the l.c.d. via routine SENDCMD each time the Text Home address is changed. Other commands are sent in other situations, as will become apparent as we discuss and illustrate them.

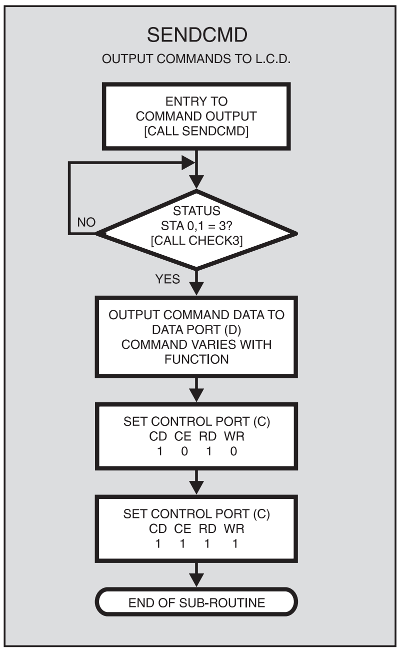

SENDCMD
OUTPUT COMMANDS TO L.C.D.
ENTRY TO COMMAND OUTPUT [CALL SENDCMD]
NO
STATUS STA 0,1 = 3? [CALL CHECK3]
YES
OUTPUT COMMAND DATA TO DATA PORT (D) COMMAND VARIES WITH FUNCTION
SET CONTROL PORT (C) CD CE RD WR 1 0 1 0
SET CONTROL PORT (C) CD CE RD WR 1 1 1 1
END OF SUB-ROUTINE

Fig.13. Flow chart for routine SENDCMD.

The SENDCMD routine is shown in Listing 7 and its flow chart Fig.13.

Having sent the TXHOME command, a RETURN is made to the calling routine.

From Listing 5, you will have seen that the command to be sent, TXHOME in

Listing 7. SENDCMD – send command instruction to LCD. See also flow chart Fig.13.

```
MOVWF TEMPA          ; temp store val brought in on W
CALL CHECK3          ; read status for DA0/DA1 = 3
MOVF TEMPA,W         ; recall stored data
MOVWF PORTD          ; send stored data
NOP                  ; RST  CD  CE  RD  WR
MOVLW %00011010      ;  1    1   0   1   0
MOVWF PORTC          ; set CE, WR low
NOP                  ; RST  CD  CE  RD  WR
MOVLW %00011111      ;  1    1   1   1   1
MOVWF PORTC          ; set all high
RETURN
```

LISTING 8. TEXTAREA – set Text Area. See also flow chart Fig.14.

```
CLRF ADRMSB       ; set Text area (columns)
MOVF COLUMN,W     ; column quantity (preset for 20 columns)
MOVWF ADRLSB
CALL CMDADR       ; send 2 bytes of address data
MOVLW TXAREA
CALL SENDCMD      ; send TXAREA command
RETURN
```

Listing 9. SETMODE – set INT/EXT/AND–OR–XOR mode. See Table 4.

```
MOVLW %10000000         ; External CG RAM, OR mode
CALL SENDCMD            ; send command
RETURN
```

Listing 10. SETDISPLAY – set text/graphics/cursor on/off. See Table 4.

```
MOVLW %10010100         ; text on, graphic off, cursor &
                          blink off  ($94)
CALL SENDCMD            ; send command
RETURN
```

Listing 11. SETOFFSET – Set Offset register.

```
CLRF ADRMSB          ; address value stated by Toshiba $0002
MOVLW $02
MOVWF ADRLSB
CALL CMDADR          ; send command address
MOVLW OFFSET
CALL SENDCMD         ; send OFFSET command
RETURN
```

this instance, is loaded into PIC register W (MOVLW TXHOME). On entry to SENDCMD (Listing 7), the value is stored in a temporary register, simply called TEMPA, although it could have any other name if preferred.

The now-familiar CHECK3 status check is made, after which the value stored in TEMPA is recalled and output to PORTD, the data input/output port, which is in its default state and set for output.

Following a one cycle pause (NOP) for stabilisation of the port, the command port, PORTC, sets CE and WR low, leaving the other bits high. Again a one cycle pause occurs to allow the l.c.d. to accept the data, and then CE and WR are taken high again, followed by a return to the calling routine. The l.c.d. will now have accepted the Text Home address of $0000.

In routine GRAPHHOME (set the Graphic Home address – not shown), the process is identical to that for TEXTHOME, this time sending $0200 as the address, and GRHOME as the actioning command.

AREA SETTING

Text and Graphics area setting is then performed respectively by routines TEXTAREA (Listing 8 and Fig.14) and the closely similar GRAPHAREA (not shown).

The business of Text and Graphics areas is somewhat subtle, and does not actually refer to the area of the display that is visible. It refers to the areas set aside for Text and Graphics data storage, and determines the way in which data is ultimately shown on screen. The area is specified by the number of columns it contains. A column, as said earlier, is specified as being one byte wide.

Later, a routine (AUTOWRITE) is demonstrated that allows automatic incrementing of addresses when data is repeatedly written to the l.c.d.

When Autowrite is on, addresses are incremented along the length of each allocated screen line right up to the end of the column count set through the relevant Area command. At the end of line, the address is incremented to the start of the next line. The process continues for as many increments as required.

If the Text Area, for example, has been set for 20 columns, the length of each line is 20 columns long. This means that if you start at the beginning of line 1 and write data to the l.c.d. 20 times, line 1 will be filled in incremental order. The 21st write, though, will place the next data byte at the start of line 2.

When the l.c.d. has been set for 6-bit mode, the actual screen area seen is also 20 columns wide, therefore you can repeatedly write text to the l.c.d. 20 (columns) × 8 (lines) = 160 times and the actual screen area will be filled with consecutive data along all 20 character positions through all eight text lines.

If, however, the data screen area has been set to 40 columns, for example, the same writing of 160 characters will have a different visual effect.

Line 1 will be filled up to position 20 and the screen will show the characters as before. The next 20 writes to the l.c.d., though, will be stored in the remaining 20 bytes of the column area allocated, which is "off–screen". These 20 bytes will not be seen. On the 41st write to the l.c.d., the first byte of the next line will be written to, which is once again "in-screen" and will thus be visible on the display.

So, in order to completely fill the actual screen area by writing consecutive data bytes in autowrite mode, 320 writes must be made, and only uneven–numbered groups of 20 characters (1, 3, 5 etc) will be seen. The evenly number groups of 20 characters (2, 4, 6 etc) will remain unseen.

The alternative, when in 40-column mode, is to write 20 bytes of data to line 1, reset the address for the start of line 2 and write another 20 bytes, and so on for the other visible lines.

Of course, using 40-column mode allows addresses to be set for displaying separately in the first 20 bytes of each line, and another batch of data stored separately and unseen into the last 20 bytes of each line. It is then possible to issue commands which cause either the first block to be displayed, or the second. In other words, to switch between l.c.d. blocks (pages) as referred to earlier.

In their demo, however, Toshiba do not illustrate this paging facility (although it is illustrated later in the author's own demo program).

Toshiba simply set the Text Area for 20 columns width, as is performed via subroutine TEXTAREA in Listing 8. At the start of the demo program, the author has allocated variable COLUMN as the store for the column width value, setting it for 20 ($14).

In the TEXTAREA routine, the Text Area is set into the same variables as were used earlier in the address setting routines, ADRMSB and ADRLSB. This double–byte value ($0014 = 20 decimal) is also sent to the l.c.d. via the same command address setting routine (CMDADR), followed by the command TXAREA being sent via the SENDCMD routine. (To set the area for 40 columns would require an address value of $0028 = 40 decimal to be sent.)

Routine GRAPHAREA (not shown) sets the graphics area in the same fashion, also for 20 columns, but with the actioning command becoming GRAREA instead of TXAREA.

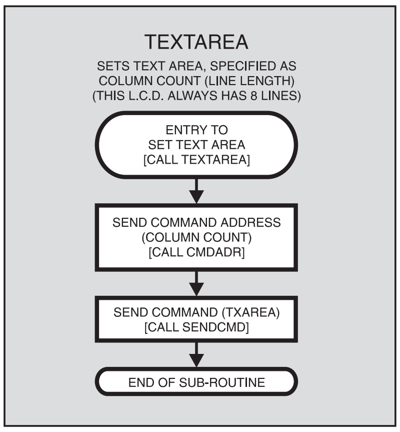

TEXTAREA
SETS TEXT AREA, SPECIFIED AS COLUMN COUNT (LINE LENGTH) (THIS L.C.D. ALWAYS HAS 8 LINES)
ENTRY TO SET TEXT AREA [CALL TEXTAREA]
SEND COMMAND ADDRESS (COLUMN COUNT) [CALL CMDADR]
SEND COMMAND (TXAREA) [CALL SENDCMD]
END OF SUB-ROUTINE

Fig.14. Flow chart for routine TEXTAREA.

MODE SETTING

There are two forms of mode defined by Toshiba which, regrettably, they only define as Mode and Display Mode (see Table 4). The various forms will be demonstrated more fully in the author's demos.

Table 4

Mode itself is subdivided into the following six sub-modes and codes (where X can be 0 or 1):

OR mode	1	0	0	0	X	0	0	0
XOR mode	1	0	0	0	X	0	0	1
AND mode	1	0	0	0	X	0	1	1
Text Attribute mode	1	0	0	0	X	1	0	0
Internal CG ROM mode	1	0	0	0	0	X	X	X
External CG RAM mode	1	0	0	0	1	X	X	X

Display Mode is also split into six sub-modes and codes:

Display off	1	0	0	1	0	0	0	0
Text on, graphic off	1	0	0	1	0	1	X	X
Text off, graphic on	1	0	0	1	1	0	X	X
Text on, graphic on	1	0	0	1	1	1	X	X
Cursor on, blink off	1	0	0	1	X	X	1	0
Cursor on, blink on	1	0	0	1	X	X	1	1

Listing 12. CLRTXT – clear text screen.

```
CLRTXT:
CLRF ADRMSB         ; set address ($0000)
CLRF ADRLSB
CALL SCREENADR      ; set screen write address
MOVLW AWRON         ; autowrite on
CALL SENDCMD        ; send AWRON command
MOVLW 8             ; number of lines to clear
MOVWF LOOPC
CLR2:
MOVF COLUMN,W       ; column length
MOVWF LOOPB
CLR3:
MOVLW 0             ; write 0
CALL AUTOWRITE      ; autowrite and increment
DECFSZ LOOPB,F
GOTO CLR3
DECFSZ LOOPC,F
GOTO CLR2
MOVLW AWROFF        ; autowrite off
CALL SENDCMD        ; send AWROFF command
RETURN
```

Listing 13. SCREENADR – send data read/write address to LCD.

```
CALL CHECK3         ; read status for DA0/DA1 = 3
MOVF ADRLSB,W       ; send address LSB
CALL OUTDATA
CALL CHECK3         ; read status for DA0/DA1 = 3
MOVF ADRMSB,W       ; send address MSB
CALL OUTDATA
MOVLW ADPSET        ; address pointer
CALL SENDCMD        ; send ADPSET command
RETURN
```

Listing 14. AUTOWRITE – autowrite data and increment address.

```
MOVWF TEMPA     ; temp store value brought in on W
CALL CHECK8     ; read status for DA3 = 8
MOVF TEMPA,W    ; recall stored value
CALL OUTDATA    ; output data
RETURN
```

OR mode is required in the Toshiba demo. This allows data to be written to the screen and ORed with any data already existing at the same location. The routine (SETMODE) is shown in Listing 9.

All that is involved is to send the command for OR mode (%10000000 or $80 or 128 decimal) to the l.c.d. via the SENDCMD routine which was discussed earlier.

The setting of the Display Mode (SETDISPLAY) has its routine shown in Listing 10, in which Text is turned on, and Graphics and the cursor are turned off. Again the only action required is to send the appropriate command (%10010100 or $94 or 148 decimal) to the l.c.d. via SENDCMD.

OFFSET SETTING

The Display Mode command, though, is shown in Toshiba's demo as following routine SETOFFSET, in which an offset register value command is issued. Listing 11 shows what is required.

Toshiba's explanation of the use of the offset register is not fully intelligible. The interpretation, however, appears to be that the offset register is used to determine the external (user-defined) character generator RAM area.

The T6963C assigns this generator so that when text character codes $80 to $FF are written to the l.c.d. they are treated in the same way as the "normal" text characters of the internal character generator RAM, which are called through codes $00 to $7F. That is, you write only one value to the screen to display the eight bytes of the character held in the CG RAM.

This is in contrast to writing *true* graphics data to the screen, in which eight bytes of data have to be individually sent.

Toshiba go on to state that setting the offset register to a value of 2 sets the CG RAM address to $1400, which then allows the user–defined characters to be called by their allocated code, between $80 and $FF. The implications of attempting to use different offset addresses have not been explored.

As with setting Home and Area values, and referring to Listing 11, the offset value is set into the 2–byte address as $0002 and the CMDADR routine called. This is followed by the OFFSET command being issued via SENDCMD.

The routine next in order of calling is SETDISPLAY, as discussed in the previous section.

INTO ACTION

This completes the basic initialisation of the l.c.d. and it is now ready to have real data sent to it for display on screen. The first data to be written, though, clears the screen of any previous data which might exist. At switch on, for example, random data could automatically (and unpredictably) be set into the screen and other areas.

Routine CLRTXT is that which clears the Text screen (there is no need to clear the Graphics screen since this has been deactivated in the Display Mode setting).

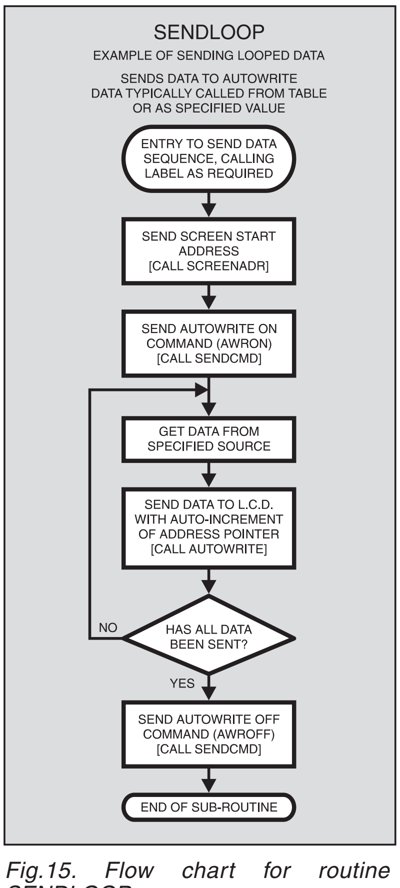

SENDLOOP
EXAMPLE OF SENDING LOOPED DATA
SENDS DATA TO AUTOWRITE DATA TYPICALLY CALLED FROM TABLE OR AS SPECIFIED VALUE
ENTRY TO SEND DATA SEQUENCE, CALLING LABEL AS REQUIRED
SEND SCREEN START ADDRESS [CALL SCREENADR]
SEND AUTOWRITE ON COMMAND (AWRON) [CALL SENDCMD]
GET DATA FROM SPECIFIED SOURCE
SEND DATA TO L.C.D. WITH AUTO-INCREMENT OF ADDRESS POINTER [CALL AUTOWRITE]
NO
HAS ALL DATA BEEN SENT?
YES
SEND AUTOWRITE OFF COMMAND (AWROFF) [CALL SENDCMD]
END OF SUB-ROUTINE

Fig.15. Flow chart for routine SENDLOOP.

The source code is shown in Listing 12, see also flow chart SENDLOOP in Fig.15.

The text address from which data is to be cleared is first set to $0000. Because it is *data* that is to be sent (as opposed to *commands* as with the previous routines), an Address Pointer command has to be sent as well, ADPSET.

A separate routine (SCREENADR) has been written that sets both the data address and the Address Pointer command. It is shown in Listing 13 and its flow chart is identical to the CMDADR routine (Listing 6) except that the sending of ADPSET is added at the end.

Following the call to SCREENADR (from Listing 12), Autowrite is set on by issuing the AWRON command. Next a LOOPC value is allocated, holding the number of lines to be cleared (eight). Then a LOOPB value, which holds the line length (COLUMN) involved, is set.

The subroutine CLR3 is then entered, in which the value of zero is repeatedly written to the l.c.d. for the duration of the nested decrementing loops.

At each write, the data is written to the screen via routine AUTOWRITE (Listing 14), in which the screen address is automatically incremented after each byte is written. As discussed earlier, all addresses are filled in order and in relation to the column value previously set in the initialisation routines. This will be more clearly seen later in the author's Demo 9.

On each entry to AUTOWRITE, the value brought in on W (in this instance zero) is temporarily stored in variable TEMPA. A status check is then called, but not the CHECK3 routine seen previously. This time, because we are in Autowrite mode, it is CHECK8 which is called, in which the status register is checked for the value of 8 (bit 3 high). The process is almost identical to that used in CHECK3 and is not listed here.

On conclusion of the check, the data in variable TEMPA is recalled and sent to the l.c.d. via the usual OUTDATA routine.

SYMBOL CREATION

Toshiba now illustrate the creation of characters (symbols) for storage in the external (user-defined) CG RAM. The data is specified in a table (EXTCG) that holds the 64 byte values that make up the eight component parts of the Japanese characters

Listing 15. WRITECG – write to external (user-defined) character generator.

```
WRITECG:
MOVLW $14            ; set CG RAM start address to
MOVWF ADRMSB           $1400
CLRF ADRLSB
CALL SCREENADR       ; send 2 bytes of address data +
                       address pointer
MOVLW AWRON          ; set autowrite on
CALL SENDCMD         ; send AWRON command
MOVLW 64             ; set loop for 8 sets of 8 bytes (= 64)
MOVWF STORE
CLRF LOOPB
EXCG:
MOVF LOOPB,W         ; get loop value
CALL EXTCG           ; get data from table position set by
                       loop val
CALL AUTOWRITE       ; autowrite and increment address
INCF LOOPB,F         ; increment loop counter
DECFSZ STORE,F       ; decrement counter, is it zero?
GOTO EXCG            ; no, so repeat
MOVLW AWROFF         ; yes, turn off autowrite
CALL SENDCMD         ; send AWROFF command
RETURN
```

seen earlier in Photo 2. The routine is illustrated in Listing 15 (WRITECG – see also flow chart SENDLOOP, Fig.15).

On entry to WRITECG, the CG RAM address at which the data writing commences is set at $1400. This is the value referred to earlier when Offsets were discussed. The Autowrite process is used and the address written to is automatically incremented on each data write.

Apart from data being called from a table, the routine is similar to that used for clearing the text screen (in which the written data had a value of 0).

The table is not illustrated here in full, but the following shows the data Toshiba specifies for creating the first user-defined character (the first column is the command and the second shows the value in binary):

```
RETLW $01    $01 = 00000001
RETLW $01    $01 = 00000001
RETLW $FF    $FF = 11111111
RETLW $01    $01 = 00000001
RETLW $3F    $3F = 00111111
RETLW $21    $21 = 00100001
RETLW $3F    $3F = 00111111
RETLW $21    $21 = 00100001
```

If you ignore the 0s in the binary code, concentrate on the 1s and look at Photo 2, you will see that the 1s represent the active pixels of the top left quadrant of the first Japanese symbol.

The remaining aspects of the symbols are similarly created.

For each block of eight data bytes read from the table and stored in CG RAM, a counter (internal to the T6963C) is automatically incremented, from $80 up to, in this table's instance, $87. These eight values are the addresses which are called in order to display the user-generated symbols on screen. In other words, to display the first symbol discussed, a value of $80 would be written to the l.c.d.'s text screen, a value of $81 for the second, etc.

Having been created, the symbols are called in the order specified in another table via routines WTDD2 and WTDD3. Both are very similar to that in Listing 15, relating to different addresses and calls to other tables. They are not shown here. The method is the same as that used during the creation of the symbols – a screen start address is specified and the symbols consecutively written to that and subsequent addresses. Accessing table EXPRT1, routine WTDD2 calls and writes the following data bytes to line 5 commencing at column 8 (address value $6C = 5 × 20 + 8 = 108 decimal):

```
RETLW $80
RETLW $81
RETLW $00
RETLW $00
RETLW $84
RETLW $85
```

Note the inclusion of zero bytes, specifying blank characters (from the text character CG ROM) to be written to the screen. Routine WTDD3 behaves similarly, accessing table EXPRT2 and writing data to the sixth line, again starting at column 8.

The table–specified order of writing can be changed if desired for other display circumstances. The data stored in the character generation table (EXTCG) can also be changed to suit the user's needs. This point will be amply illustrated in the author's later demos.

TEXT WRITING

Toshiba then go on to show how text itself is written to the screen, using the l.c.d.'s own internal character generator (CG ROM).

The following data bytes are accessed from table TXPRT and written to generate the word TOSHIBA, as shown in Photo 2.

```
RETLW $34  ; T
RETLW $00  ; blank
RETLW $2F  ; O
RETLW $00  ; blank
RETLW $33  ; S
RETLW $00  ; blank
RETLW $28  ; H
RETLW $00  ; blank
RETLW $29  ; I
RETLW $00  ; blank
RETLW $22  ; B
RETLW $00  ; blank
RETLW $21  ; A
```

For example, the data byte value of $34 specifies CG ROM location $34 at which the character for letter T is stored (as defined during the device's manufacture), and value $2F specifies letter O, and so on. Again note the inclusion of zero bytes for spaces.

Routine WTDD illustrates the text values being written to the screen. It is practically identical to the WTDD2 and WTDD3 routines except that a different data table (TXPRT) is accessed. It is not listed here.

TOSHIBA TO ASCII

It is reiterated that the values for the CG ROM characters held by the T6963C are not the same numbers as specified by ASCII codes. They are in fact ASCII values less 32. For instance, letter A in ASCII has a value of 65 decimal ($41). Deduct 32 from this and you obtain the value of 33 decimal ($21) for letter A in the Toshiba code, as shown at the end of the table in the previous section.

The author's later demo will illustrate a routine in which alphanumeric characters can be specified in the normal PIC fashion of enclosing the required ASCII character in single quotes (e.g. 'A') and for the routine to then automatically translate its ASCII value into a Toshiba value.

Following the completion of writing the characters to the screen, Toshiba's program ends and no further action occurs.

You might care, though, to change some of the data in Toshiba's demo and see what results occur. There will, however, be much more opportunity for such things in the author's demos, which we move on to now.

AUTHOR'S DEMOS

Before progressing, it is necessary to load two programs into the PIC, the author's demo routines, for which the first file is GEPE456.OBJ (source code GEPE456.ASM). It too is written in TASM.

Additionally, a text file needs to be loaded into the PIC's EEPROM data memory. The file is called DUCK08.MSG (for reasons that will become clear!) and is a Message file of the type recognised by *PIC Toolkit Mk2* for sending to the PIC's EEPROM. Beware that other programmers may not necessarily recognise the format (or be capable of directly accessing the EEPROM data memory).

DEMO CONTENT

The first matter examined is that of substituting different patterns for use as user–generated CG RAM symbols. The resulting display is shown in Photo 1, earlier.

After the PIC setup procedure, the column width (COLUMN) is then set at 34 (see label GRAPHIC) and the l.c.d. SETUP routine is called, in which all the subroutines discussed previously are actioned.

Two new routines, CLRGRAPH and CLRCG, are also sent, in which the graphics and user-defined CG RAM areas are cleared. The routines are closely similar to the CLRTXT routine and not listed here.

Then follow the author's 12 demos in which not only are routines covered that Toshiba did not illustrate, but you are given the opportunity to change various command codes and observe the results via your demo board.

A switch monitoring routine is placed within or between demos so that you can keep the results on screen until the switch (S1) is pressed to action the next demo. Labels are given to each demo call so that some demos can be *remmed* out, or jumps made from one demo to another further down. When Demo 12 has been finished with, a return to Demo 1 is made.

DEMO 1
Diagram and words

In Demo 1, a set of data held in table form (CGTABLE) is read and stored in the user-defined CG RAM, in a similar way to which Toshiba's Japanese symbols were created and shown. These symbols represent the component parts of a

Listing 16. Data for first character ($80) in table CGTABLE, representing the "amplifier" top left, plus first slope down.

```
RETLW %10000000
RETLW %11000000
RETLW %10100000
RETLW %10010000
RETLW %10001000
RETLW %10000100
RETLW %10000010
RETLW %10000001
```

simple electronic circuit, as shown earlier in Photo 1.

There are 15 characters created, each comprising eight bytes, making a total of 120 bytes. They are stored in consecutive CG RAM locations which are called with value codes of $80 to $8E (whereas Toshiba's occupied calling locations $80 to $87). The subroutine SETCG is used for this process. The routine is closely similar to that in Listing 15 and is not shown here. The first part of the symbol creation table is shown in Listing 16.

Unlike Toshiba's demo, the calling table (AMPLIFIER) does not insert blank character cells. Instead the data is written by several subroutines so that it is placed at the exact screen addresses required.

For example, the routine labelled CIRCUIT specifies that the location for the first symbol is to be placed at Column 5 Line 1, in which both lines and columns now start at zero (they started at 1 in Toshiba's demo).

The column required is specified by the value loaded into W. The selection of Line 1 is then specified by the call to LINE1. The LINE call is one of several in which the line number is specified by the called address, i.e. a call to LINE2 would specify that Line 2 was the required line (see the source code for the listing}.

Calculation of the screen address at which the first character is to be placed is in relation to the line number called, the number of columns specified for the display (column width), and with the column value as set into W prior to the call added to the total. Readers will no doubt be able to write a more compact routine than is used in the demo.

The routine which reads the table and sends data to screen is SHOWCG (not listed here but similar to Listing 15). Part of the listing for table AMPLIFIER is shown in Listing 17.

Note that the screen displays some small text characters as part of the circuit diagram. These have also been created as user-defined CG RAM symbols. The l.c.d. does not keep small characters as part of its fixed text symbol library.

Listing 17. First part of AMPLIFIER table.

```
RETLW $80      ; amp top left
RETLW $83      ; amp input
RETLW $86      ; cap top
RETLW $83      ; amp input
RETLW $83      ; amp input
RETLW $81      ; amp left
RETLW $82      ; amp slope down
RETLW $83      ; amp output
RETLW $86      ; cap top
RETLW $83      ; amp output
RETLW $8B      ; word IN
```

The normal size letters shown on the screen are created using the l.c.d.'s own internal text generator. The letters required are specified in a table (TABLE1) and called in order, with the l.c.d. screen address to which they are written being changed when needed.

The address setting and table calling routines (WORDS and SHWTXT) are similar to those used for sending the circuit diagram symbols to the screen and not shown here. Part of the text-holding TABLE1 called is shown in Listing 18.

Listing 18. First part of TABLE1.

```
RETLW 'G'
RETLW 'R'
RETLW 'A'
RETLW 'P'
RETLW 'H'
RETLW 'I'
RETLW 'C'
RETLW ' '
```

In routine SHWTXT, after TABLE1 has been called, 224 is added to the returned value. This is the equivalent of subtracting 32 from the ASCII value held in the table, so converting it to the Toshiba code value (as discussed earlier).

Exercise 1.1.

Experiment with changing the text content sent to the screen, and its positioning.

Exercise 1.2

Do the same with the graphics display, perhaps even completely creating your own replacement drawing.

DEMO 2
Bit setting and clearing

Pressing the switch now starts the second demo, which illustrates how individual pixel bits on the screen can be set or cleared (see Photo 4).

It uses the same illustration as in Demo 1 but to the right of it now draws a square and then clears it, followed by drawing it again, indefinitely. At the centre of the square a single pixel is shown constantly set.

The action uses a delay loop between the setting or clearing of each pixel so that it is in semi-slow motion.

For the first time in any of the demos so far, the Graphics screen itself is used for this action, superimposed on the circuit diagram, which you will recall makes use of the Text screen.

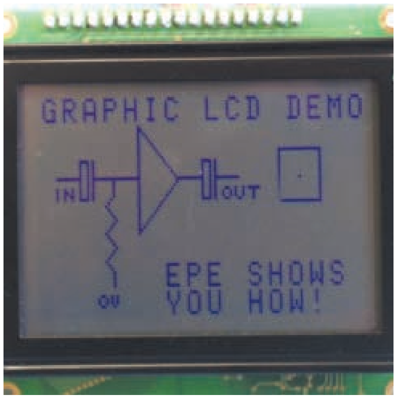


Photo 4. The square being drawn in Demo 2.

The first part of the demo routine is shown in Listing 19.

On entry to Demo 2, the first commands ensure that the display is placed on screen page 1 (more on this later). Both Text and Graphics screens are then activated by the commands:

```
MOVLW %10011100   ; text & graphic
                    on, cursor & blink off
CALL SENDCMD      ; send command
```

Until the display mode is changed, anything written to the Text or Graphic screens will be shown. Hence you continue to see the text-generated characters on the Text screen, plus the graphics-generated square data being drawn on the Graphics screen.

The Graphics screen is made up of 64 horizontal lines (raster lines) each of which contains the same number of columns as previously set during the initialisation. The address of any byte on screen is in relation to the line number and the column number. The pixel to be manipulated is one of the eight bits within the selected byte.

Table 5. Pixel bit setting codes

Code	*Function*
11110XXX	Bit Reset
11111XXX	Bit Set
1111X000	Bit 0 (LSB)
1111X001	Bit 1
1111X010	Bit 2
1111X011	Bit 3
1111X100	Bit 4
1111X101	Bit 5
1111X110	Bit 6
1111X111	Bit 7 (MSB)

To set a screen address, subroutine GLINE (graphics line – not shown) is told which column is required and on which line. It then calculates the address. For example, shortly after entry to Demo 2, column 12 line 23 is the required address. As shown in Listing 19, the column value (12) is loaded into the address LSB and W is then loaded with the line value (23). GLINE is then called.

GLINE performs a rudimentary multiplication routine, multiplying the line number by the number of columns specified in the initialisation. The actual column number required is then added to the total.

The selected pixel is turned on or off in relation to the value of bit 3 of a command byte (BITVAL) which is sent via a specific bit writing routine, BITWRITE. Bit 3 = 0 causes the screen pixel bit to be reset (cleared), and bit 3 = 1 sets the bit (turns it on). The bit itself is specified by the 3–bit code in bits 0 to 2.

Table 5 illustrates the logic. Listing 19 shows part of the routine, performing the setting of the square's single central bit.

In subroutine BITWRITE, the screen address is set from the calculated address value, and then the BITVAL byte is written to the screen, but as a command rather than actual screen data. See Listing 20.

In the square drawing routines, manipulation of the pixel-controlling bit within the byte at the selected address is done by an 8-value (0 to 7) incrementing counter (STORE1). The bit to be set or cleared is stated by the counter value.

Listing 19. First part of DEMO2 bit setting/clearing routine.

```
                        ; set single centre bit
MOVLW 12                ; set column number
MOVWF ADRLSB
MOVLW 23                ; set graph line
CALL GLINE              ; multiply by line length to get address
MOVLW %11111000         ; set bit 0
CALL BITWRITE
MOVLW %11111000         ; bits 0–2 indicate bit affected
MOVWF STORE1            ; bit 3 high = set, low = clear
```

Listing 20. BITWRITE – write single bit routine.

```
MOVWF BITVAL       ; temp store val brought in on W
CALL SCREENADR     ; set screen write address
MOVF BITVAL,W
CALL SENDCMD       ; send BITVAL command
RETURN
```

Listing 21. DEMO3 – fill selected graphics area with reverse attribute value, first section, for top line.

```
DEMO3:
MOVLW 2             ; set graphic address ($02xx)
MOVWF ADRMSB
MOVLW 0             ; set column
CALL LINE0          ; set cell number for line stated
MOVLW 16            ; character quantity to be affected
MOVWF LOOPC
MOVLW %00000101     ; attribute reverse
MOVWF ATTRIB
CALL SETATTR
```

There are several sub-routines within the square drawing demo (not shown here), which respectively cause the apparent movement of the square's perimeter.

During upwards or downwards drawing, the selected bit value remains constant. It is the address of the line/column which is changed, adding or subtracting the column width value depending on the direction of "travel".

In the full source code, note how bit 3 is toggled high or low at the end of each drawing of the square, so alternating between bit setting and bit clearing.

Exercise 2.1.
Set the display mode so that only the square being drawn is displayed.

Exercise 2.2.
Change the screen address at which the square is drawn, together with the centre active pixel.

Exercise 2.3.
Rewrite the square drawing program so that the drawing appears to take place in an anticlockwise fashion, instead of moving clockwise.

Exercise 2.4.
Is your logical thinking up to drawing a circle instead of a square?

Having finished Exercises 2.1 to 2.4, reinstate the Display Mode so that the circuit diagram and bit setting displays are both seen (text and graphics on).

Pressing the switch starts Demo 3.

DEMO 3
Text highlighting

Demo 3 illustrates text highlighting and flashing (see Photo 5). Part of the controlling routine is shown in Listing 21.

The action taken in Demo 3 is to highlight the words at the top of the screen (GRAPHIC LCD DEMO) by inverting them, causing clear letters to be shown on a dark background. Similarly with the other captions (EPE SHOWS YOU HOW), but with them flashing on and off within their dark background.

The control bytes which are responsible for these actions are written to the graphics screen area, consequently any graphics within the graphics screen at the affected locations are overwritten.

Referring to Table 6, the commands for inverting text characters against the background are:

```
MOVLW %00000101  ; attribute reverse
                   command
MOVWF ATTRIB     ; store it
CALL SETATTR     ; call set attribute
                   routine
```

The chosen command (%00000101 in this instance) is written to a temporary variable, ATTRIB, and then subroutine SETATTR is called. The routine is closely similar to other screen writing routines and is not listed here.

Table 6. Screen attribute codes

Code	*Function*
XXXX0000	Normal display
XXXX0101	Reverse display
XXXX0011	Inhibit display
XXXX1000	Blink of normal display
XXXX1101	Blink of reverse display
XXXX1011	Blink of inhibit display

Autowrite mode is used in SETATTR. The required address is first set, Autowrite is turned on and then a loop repeatedly sends the value held in ATTRIB to the required graphics screen area.

Any text character superimposed on that area via the text screen is affected by the ATTRIB value "beneath" it, in this case inverting it.

The blink-reverse command is %00001101, and is sent to the graphics screen area in the same way, having specified the required address and number of bytes involved:

```
MOVLW %00001101   ; blink reverse
MOVWF ATTRIB
CALL SETATTR
```

When all the attribute values have been sent to the required locations, the text has to be set for Attribute Mode, as is performed in Listing 22. The essential commands are:

```
MOVLW %10000100   ; text attribute
                    mode
CALL SENDCMD      ; send command
```

The following commands are also sent to ensure that the correct screen mode is set following any changes you may have made in the previous exercises:

```
MOVLW %10011100   ; text & graphic
                    on, cursor &
                    blink off
CALL SENDCMD      ; send command
```

Exercise 3.1.
Referring to the Attribute table (Table 6), experiment with using the other highlight options available. Also try putting the commands at other places on screen and observe what effect is produced.

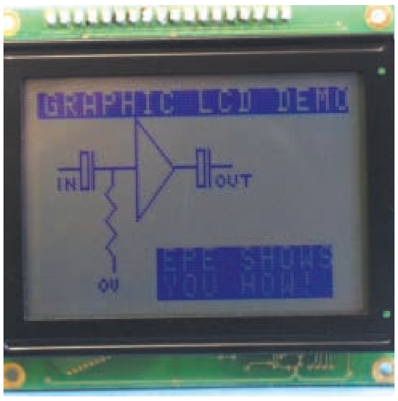


Photo 5. Screen as seen during part of Demo 3.

DEMO 4
Cursor Setting

Demo 4 illustrates how the cursor can be used. The basic routine is in Listing 23.

First it is necessary to specify the screen address at which the cursor is to be positioned, and issue the commands generated in subroutine CSRADR (see Listing 24):

```
CALL CMDADR      ; send command
                   address
MOVLW CSRPOS     ; cursor position
                   command
CALL SENDCMD     ; send command
```

Unlike other address setting calls, the cursor position is specified by the actual display screen line and column position. Thus to set the cursor for line 3 column 15, it is these two values which are sent as the address, rather than having the position calculated in relation to the first screen byte location and the column width set (as you saw occurring for placing text on screen).

Thus, for this line 3 column 15 example, the column position (15) is placed into the address LSB, and the line number (3)

Listing 22. Final section of attribute setting routine, setting for Text Attribute mode.

```
MOVLW %10000100    ; text attribute mode
CALL SENDCMD       ; send command
MOVLW %10011100    ; text & graphic on, cursor & blink off
CALL SENDCMD       ; send command
RETURN
```

Listing 23. DEMO4 – Cursor setting.

```
DEMO4:
MOVLW %10010111   ; text on, graphic off, cursor & blink on
CALL SENDCMD      ; send command
                  ; set cursor position + type
MOVLW 15          ; set column
MOVWF ADRLSB
MOVLW 3           ; set line
MOVWF ADRMSB
CALL CSRADR
CALL CSRTYP       ; set type (specified in sub-routine)
RETURN            ; (from 1 to 8 lines high)
```

Listing 24. CSRADR – Cursor position (address) setting.

```
CSR:
CALL CMDADR        ; send command address
MOVLW CSRPOS       ; cursor position command
CALL SENDCMD       ; send command
RETURN
```

Listing 25. CSRTYPE – set cursor type. See also flow chart Fig.16.

```
CSRTYP:
MOVLW %10100111    ; 8–line
CALL SENDCMD       ; send command
RETURN
```

placed into the MSB, following which the call to routine CSRADR is made, where the address is actioned:

```
MOVLW 15          ; set column
MOVWF ADRLSB
MOVLW 3           ; set line
MOVWF ADRMSB
CALL CSRADR
```

Cursors having eight different heights can be created, as illustrated in Fig.16. Listing 25 shows the command (%10100111) which generated the full 8-line cursor, "line" in this instance meaning a graphics line (of which there are 64, as stated earlier).

The cursor type command is issued in routine CSRTYP (Listing 25), which on this occasion is for an 8-line cursor.

As you will see from Fig.16, the cursor type is selected logically, with bits 0 to 2 holding the binary number whose decimal equivalent is the cursor line-count (height) value.

STATIC CURSOR

It is important to note that the cursor position remains in the same screen position to which it has been allocated. There is no facility for it be automatically incremented in position when writing text to the screen, unlike with standard alphanumeric l.c.d.s, where the cursor can be set to be "actively mobile".

You will also see from Demo 5, where switching between screen pages is performed, that the cursor position is always related to the actual location on the *visible* screen, rather than to the screen memory locations previously discussed.

When changing the cursor position on screen, it is only necessary to specify the address at which it is to be placed. Once the cursor type has been specified and actioned (via CSRTYP), it is not necessary to specify it again, unless you wish to change the type.

Note that the cursor can be set to flash or to remain static. The Display option controls its action (see Table 4).

On entry to Demo 4, the display mode is set for text on, graphic off, cursor and blink on, using the commands:

```
MOVLW %10010111   ; text on, graphic
                    off, cursor &
                    blink on
CALL SENDCMD      ; send command
```

Note that with the graphic screen now turned off, the attribute commands behind the top text line no longer cause the text to be inverted. However, the blinking text line continues to blink because the text attribute mode has not been cancelled.

Exercise 4.1.
Experiment with different addresses for the cursor to be placed, and also with the cursor type.

Exercise 4.2.
How would you stop the flashing of the bottom text lines, returning them to normal text display mode?

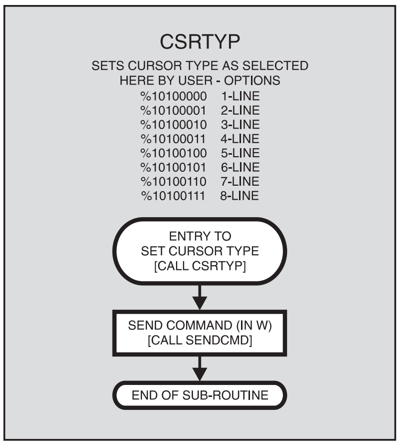


Fig.16. Flow chart for cursor type setting routine plus the cursor codes.

DEMO 5
Panning between pages

In Demo 5 panning between screen pages is illustrated. The principle of l.c.d. pages was discussed when describing how the column width setting determined how data was written to the display screen.

You would not have been aware of it at the time, but when writing text to the screen in Demo 1, text was also written into the region which we can call Page 2. That text simply says "THIS IS PAGE 2".

Panning occurs between the main page (page 1) and page 2. It has been slowed by the inclusion of a delay routine, but this can be omitted (or extended) if preferred in other applications. See the full source code for program details. It is based upon the TEXTHOME routine in Listing 5.

The Text Home address is initially set for page 1, line 1 column 1. A loop is then entered in which the address is repeatedly incremented, causing the display to shift by one column each time from right to left, each shift followed by a short pause.

As the display shifts left, so page 2 will gradually become revealed, with the progressive disappearance of page 1.

At the end of the preset loop count, the shifting is reversed so that it shifts from left to right, to again reveal page 1, with the disappearance of page 2.

The process repeats until such time that you press the switch to enter Demo 6.

If at the end of Demo 4 you had left the cursor and lower text lines flashing, you will see that the cursor does not shift in position while the panning occurs. Nor will the screen area which contains the instructions for text flashing. As the screen pans, the text beneath the flashing commands will shift away from those commands and cease to flash in the positions beyond them.

Note that the l.c.d.'s nature dictates that changes to the screen images do not instantly take effect. There is a short time that it takes for the display's liquid crystals to realign themselves following any change. This results in a brief "ghost" image of the display as it was prior to being shifted. The effect is especially noticeable in situations such as panning or switching between pages.

Exercise 5.1.
Experiment with changing the rate at which the screen pans.

Exercise 5.2.
Increase the column width setting and write text of your own invention to the area which can be regarded as page 3. Then set the panning loop lengths so that page 3 is revealed following page 2.

Exercise 5.3.
How many pages can you create, write to and pan through?

Exercise 5.4.
What happens if you cause the graphic screen to pan rather than the text screen?

Exercise 5.5.
What happens if you pan *both* screens?
Restore the original column width and panning loops to the original values before progressing to Demo 6.

DEMO 6
Switching between pages

Demo 6 is a variant of the action performed in Demo 5. Here the pages are switched between, rather than panned.

Exercise 6.1.
Perform the same experiments as in the exercises for Demo 5. Again ensure that the originally settings exist before you enter Demo 7.

DEMO 7
AND, OR, XOR

In Demo 7 (listing not shown), the AND, OR and XOR modes (Table 4) are demonstrated. Press the switch to enter the demo.

On entry, a return is made to page 1 with both Text and Graphics screens active. The previous text characters will be seen, plus probably the residual state of the square drawing demo on the graphics screen. The Text Attribute mode is also cleared. Initially, the OR display mode is active. These actions take place in the first few subroutines of Demo 7.

In the principal demo routine, an area of the graphics screen has a pattern written to it, the area of which spans part of the "circuit diagram". The process is similar to that in flow chart SENDLOOP.

The pattern is created by writing the binary value 10101010 to alternate lines within the area, and 01010101 to the other alternate lines.

This pattern is written as an Attribute, moving it into the variable ATTRIB and then writing this to the l.c.d. via the SETATTR routine which was demonstrated earlier. The key lines are as follows:

```
MOVLW %10101010  ; fill graphic with
                   val shown
BTFSS LOOPD,0
MOVLW %01010101  ; fill graphic with
                   val shown
MOVWF ATTRIB
CALL SETATTR     ; send value
```

When the pattern has been written to the designated area, the choice of whether it is ANDed, ORed or XORed with the text is available. As set, XOR mode is chosen. The choice of mode is actioned by sending that value as a command to the l.c.d. via the SENDCMD routine.

Following the sending of the selected command, the choice of which text and graphics screens are active is offered (refer back to Table 7). In the demo, the Text and Graphics On mode is selected, and the chosen value sent as a command via the SENDCMD routine.

Exercise 7.1
Experiment with choosing different options from the modes offered and note the results.

DEMO 8
Quackery!

Now we come to a complex example of creating a moving picture via the Graphics screen. Press the switch to enter Demo 8, and observe a bit of quackery (see Photo 6)!

You will see two creatures which might just be confused by some as being ducks! On the assumption that they might be, one of them opens its beak periodically and the word QUACK appears briefly on screen.

The ducks are also seen to be very sedately swimming slowly to the right. As the rightmost exits the screen area, another duck begins to appear on the left. A pattern of water is placed at the bottom of the screen, and a text message is at the top.

The water and text are created and displayed by the program in the manner discussed in previous routines. The water is a

Photo 6. The birds sedately "swim" across the screen in Demo 8.

pattern created by writing a series of 01010101 bytes to the graphics screen. The text is held in a table.

What is interesting, though, is that the data for the ducks themselves is not actually programmed into the source code, but has been programmed into the PIC's EEPROM data memory from a pattern held in a separate text file on disk (DUCK08.MSG).

This is the data you were asked to send to the PIC as part of its programming for the author's demos. Other data could be written instead and similarly downloaded to the EEPROM. The .MSG file can be read (and amended) through a normal text editor.

SIMPLE IN PRINCIPLE

A really complex set of programming commands is involved in creating this screen, and there is insufficient room to show it. The principles are simple, though:

First the water is displayed, plus the top line text. The ducks are identical and the pattern for just one is held in the EEPROM, where it is stored as a set of values for writing to the character generator (user-defined) RAM. The pattern is retrieved and written to the CG RAM. Detail of part of the bird's beak is also retrieved and stored as a separate CG RAM byte.

The data, except for the extra beak detail, is then written twice to the graphics screen, so that two ducks are shown.

The Graphics screen itself is now repeatedly read, line by line within the "Duck zone", and the bytes within each line are shifted (rotated) right, so that the LS bit of one byte shifts into the Carry register, which is then shifted into the next byte as the MS bit, and so on for all 16 bytes of each line.

The final shift right causes the final LS bit to be shifted into a holding register, from where in the next cycle it is shifted into the first byte of the line as the MS bit.

The same is performed for each of the 28 graphics lines involved. The effect is that of a line of ducks slowly swimming across the screen.

On every eighth cycle of shifting lines to the right, the separate beak section is called into action and placed so that one bird appears to have its beak open. Coincident with this, the word QUACK is written to the screen. A short pause follows in which no shifting occurs. Then the extra beak section is removed, and blanks written in place of QUACK.

Various experimental routines were written to see which method could create the fastest swimming effect. To be honest, the author was a bit disappointed that he could not get the ducks moving faster, but he still finds the display amusing!

It was decided, incidentally, not to use the panning technique illustrated earlier. This only shifts the screen by complete bytes. The ducks, though, are shifted by individual pixels, creating a smoother effect.

SCREEN READING

Whilst the full program cannot be included on these pages, it is pertinent to describe some of it in greater detail.

Each screen byte is read, shifted right to bring in the previous Carry from the left as the MS bit and to shift out the current LS bit into the Carry. The byte is then restored back on screen.

This occurs one line at a time, with a final "overflow" byte holding the Carry status of each line concluded, and which is shifted in from the left at the start of the reading procedure for the same line on the next batch round.

The routines involved, with two exceptions, are variants on those already discussed. The first exception is the EEPROM data read routine. There is nothing special about this that is worthy of discussion now. It was described in principle in the author's *PIC Tutorial* of March–May '98, again in the *PICTutor* CD ROM, and then expanded upon as a modification to suit PIC16F87x use, in *EPE* Oct '99 (*Mini PIC Tutorial*).

The other exception that is appropriate here, is the reading from the screen routine, DATAREAD, as shown in Flow Chart Fig.17.

In the full source code the command that instructs the l.c.d. to make a screen read is defined in PEEK, and it is this value which is written as the operative command.

Two sets of status checking are required for screen reads. CHECK6 is the first in which a wait occurs until status bit 6 is *low* – not *high* as in other status checks.

Then follows a CHECK3 status check, as has previously been discussed. However, in order to make the reading routine more efficient, neither check is made by calling the labelled subroutines. It is made *in-situ* within the read routine itself.

On conclusion of the CHECK3 equivalent, CD is taken low, and the byte presented by the l.c.d. to PORTD is read. It is this value which the l.c.d. has read from the addressed screen byte. The byte is temporarily stored in RDBYTE, the routine ended, and a return made to the calling routine. Now RDBYTE can be suitably dealt with as required.

There are no exercises suggested for Demo 8 (or the remainder yet to come). Think up something for yourself!

When you feel you are on the verge of "quacking-up" watching those darned ducks interminably crossing the screen (no, there's no shooting gallery program offered!), press the switch to enter Demo 9.

DEMO 9
Text Character Set

All that Demo 9 does is to display the full in-built text character set in order across the screen (see Fig.4 earlier).

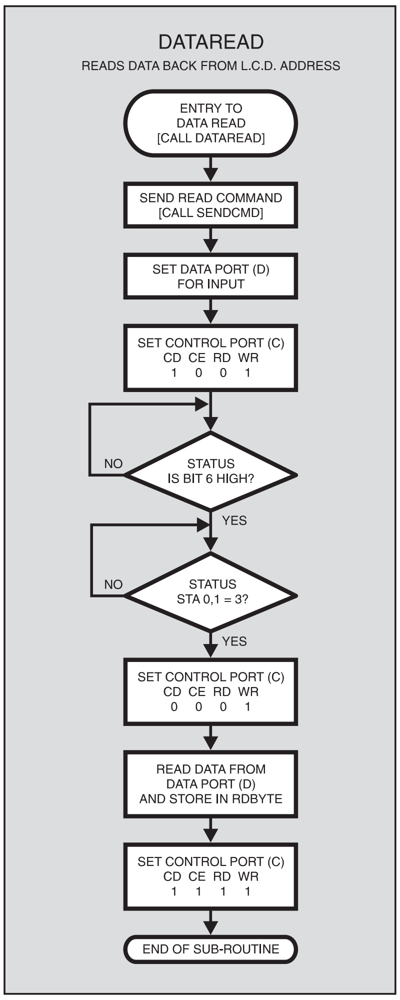

DATAREAD
READS DATA BACK FROM L.C.D. ADDRESS
ENTRY TO DATA READ [CALL DATAREAD]
SEND READ COMMAND [CALL SENDCMD]
SET DATA PORT (D) FOR INPUT
SET CONTROL PORT (C) CD CE RD WR 1 0 0 1
NO
STATUS IS BIT 6 HIGH?
YES
NO
STATUS STA 0,1 = 3?
YES
SET CONTROL PORT (C) CD CE RD WR 0 0 0 1
READ DATA FROM DATA PORT (D) AND STORE IN RDBYTE
SET CONTROL PORT (C) CD CE RD WR 1 1 1 1
END OF SUB-ROUTINE

Fig.17. Flow chart for DATAREAD routine.

DEMO 10
Graphics Set Used

Pressing the switch again enters Demo 10. This displays all the user-defined symbols that have been created and stored in the CG RAM. It includes those from the circuit diagram (those on the first line) and downloaded from the EEPROM as the duck detail.

Imagination is needed to interpret which symbol is which part of the demos. The beak, perhaps, is obvious as the penultimate symbol. So too are small sub-captions from the circuit diagram.

DEMO 11
Waveform (1)

Pressing the switch again to enter Demo 11 shows the results of the author's preparatory experiments with drawing waveforms on screen, with an ultimate view to designing the forthcoming *PIC G-Scope*.

In this example, the waveform is drawn horizontally and is seen shifting from top to bottom.

Note how the text is placed on the screen with the waveform shifting beneath it. A delay routine is included in the program and you can change the rate of shift. The number of waveforms seen can also be changed (determined by the rate of incrementing variable COUNT).

The routine shows another instance of writing individual bits to the screen, making use of a look-up table.

Whilst it is a complex program, ensuring that the waveform is not only created, but also erased on the next cycle, there is nothing special about its routines in terms of using the l.c.d.

DEMO 12
Waveform (2)

Demo 12 is similar to Demo 11, except that the waveform is created vertically in the traditional scope style, shifting horizontally (see Photo 6). It too was an experiment prior to designing the *PIC G-Scope*. Do as you like with it.

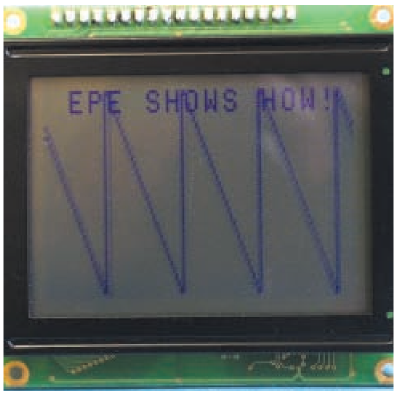


Photo 6. Waveform generated in Demo 12.

REPETITION

Pressing the switch again returns you to Demo 1.

When experimenting with the exercise suggestions, it is probably best if you remout the calls to those demos you do not need to see at that time, placing a semicolon in front of the respective CALL commands.

For each change that you make to the program, it must be reassembled from the source code to a format suitable for downloading (sending) to the PIC. The program is written in TASM, which requires it to be assembled as a .OBJ file for sending to the PIC via a programming tool such as *PIC Toolkit Mk2*.

If you prefer to work with the MPASM dialect, *Toolkit* can translate from TASM to MPASM. In which case any re–assembly would need to be to a .HEX file if you are then using an MPASM-type programmer.

When writing your own future programs you will find that many of the author's routines are ideally suited to use as library routines. You will also spot many ways in which you can deviate from the exactness of the routines while still retaining their essence.

Additionally, you will find that some routines can be "tightened–up" to become more efficient – changing bits, for example, instead of complete bytes. The LINE and GLINE commands, as another example, deserve attention to make them more efficiently programmed.

WINDING UP

Without the guidance of Toshiba's demo program, the author would have found it extremely difficult to get to know the operation of the T6963C l.c.d. controller.

It has to be said, though, that it would have been appreciated had Toshiba's examples gone further. There is much that was left to be discovered by logical deduction and by trial and error. Much of it has been achieved, as the author's demos illustrate. Nonetheless, there are still some questions unanswered, possibly more than are immediately apparent.

Should readers investigate beyond the regions explored through the author's demos, they might care to share their findings with others, submitting them for possible inclusion in *Readout*. You might even design a circuit based on a graphics l.c.d. that you would like to submit for possible inclusion as an *EPE* constructional project.

Give Editor Mike Kenward a call if you have an idea which you think might interest us and other readers.

DOWNLOAD SITES

The Toshiba T6963C data sheet was downloaded from **www.toshiba.com/taec/components/Datasheet/T6963CDS.PDF**. Note that the device is designed to control many different l.c.d. formats and some of the data is not relevant to the PG12864.

The Powertip data sheet is not available for download except by authorised Powertip agents (besides which, the author found it to be unhelpful and in some cases grossly inaccurate). Powertip's site is at **www.powertip.com.tw**.

The RS data sheet (RS 298.4613), which is also available from Electromail, was also found to be unhelpful (and of little relevance to the Powertip PG12864 as a separate entity).

Another graphics l.c.d. site is at **www.varitronix.com**. Some of their l.c.d.s are also controlled by the Toshiba T6963C, but adequate data sheets could not be located.

Kent Displays Inc of the USA have a site at **www.kentdisplays.com** which is interesting for its l.c.d.s that retain their image even after power has been switched off. They do not use the T6963C controller, however.

Microchip's site, from where a PIC16F877 data sheet can be downloaded, is at **www.microchip.com**.

The *EPE* web site, from where the software for this demo can be downloaded (and where lots of other matters of interest exist!), is at **www.epemag.wimborne.co.uk**.

Finally, if you are interested in Partridges (see earlier!) try doing a search on **Perdix**, via **www.google.com**. Google is an excellent search engine anyway and is well worth adding to your list of favourite sites. Thanks to friend Alan Winstanley for having told us about it! □