SC04 - 8x8 LED Dot Matrix Addon Card for SXI & SXII

The 8x8 LED dot matrix displays give an introduction into how to program things such as moving message displays (these same 8x8 matrix units are used in such displays) and LCD dot matrix displays.

This Kit allows the connection of one or two displays to be connected to the CN1 I/O connector of the Southern Cross computer. A wide range of program examples are provided on the software in **SC04.ZIP** which you can download from the software download page of

http://www.quasarelectronics.com/software.htm

Provision has been made on the PCB so that the display may be separated from the drive circuit and the two parts connected by flat ribbon cable. Extra rows of pads and holes are on the board for this purpose. For example, you may join two displays together and have a moving message scrolling across the two of them. In this case the circuitry can be located a few inches behind or near them.

Construction. Check all the components provided in the Kit against the Component Listing. There are **eleven** links to add to the board. The first thing to watch during construction is the way the 8x8 display is mounted. The white notch in the centre of one of the plastic sides on the back of the display goes to the top of the board. This notch is shown on the overlay. The writing on the side of the display module points to the IDC connector at the bottom of the board.

The second thing to take care about is the construction of the 16 pin IDC cable. When you press the components together to make the cable use a vice to deliver even pressure across the connector as you press the pieces together. When you finish inspect the pins closely to be sure that each pin is connected to the cable strand that it is supposed to go to. It is rather easy when doing hand construction of these cables to find one pin has gone in skew and is shorting between two adjacent V-pins. Make sure that pin 1 at one end of the cable goes to pin 1 at the other end, and not pin 16.

Components:	
10uF electrolytic capacitor	1
74HC273	2
UDN2981	1
ULN2803A	1
18 pin IC socket	2
20 pin IC socket	2
8 LED display	1
DPDT PCB-mounted switch	1
16 pin box header	1
16 pin IDC socket & holder	2
16 strand flat cable	10"
8x8 Display PCB, SC04	1
Hook-up wire	10"

To check that the board is working the Southern Cross monitor has a kaleidoscope program built into it. Put the switch in the up position. This will connect the two latches on the board to ports 80H & 82H. Press Function E. (To remind you - press the Reset key, then the Fn key then the 'E' key.) A pattern of randomly generated symmetric images should appear on the display. This will continue until Reset is pressed.

What to do if it does not work
Check the orientation of the ICs and electrolytic capacitor. Check the cable & cable connections. Is the display in the correct way, notch to the top

How it Works
Two latches IC1 and IC2 are connected to the Data Bus of the Southern Cross through the I/O port. Each latch can be addressed by different I/O addresses which are selected by the DPDT switch. There are a total of four port addresses to choose from. They are 80h,81h,82h and 83h. A PCB mounted DPDT switch switches the latches between two sets of port addresses (one for the x axis of the display and the other for the y axis.) One display uses ports 80h & 82h and the other uses ports 81h & 83h. This means that two LED display boards can be connected on the one 16 pin cable. On one board the switch is put into the up position. And on the other the switch is put in the down position.

The Y axis latch outputs (IC1, 74HC273) are connected to the common anodes of the LED matrix through a source driver (IC4, UDN2981A). Similarly the X axis latch (IC2, 74HC273) is connected to the common cathodes of the LED display through a sink driver (IC3, ULN2803A). The X latch is also connected to the system Reset to ensure LED's are not lit when the latches are first powered up or the system is Reset.

Background. Persistence of Vision. When you see the TV picture of a TV picture you must have noticed how badly the illustrated screen flickers. This is because a TV picture is never all 'on' at the same moment of time. It is being rewritten 50 times a second by a moving trace. The image you see on the TV screen persists on the retina of your eye so your brain think it is seeing a continuous image over the whole screen. When you see a TV screen of a TV screen the latter screen is not being scanned this way so it flickers badly. Another example is at the picture theatre; 24 pictures are flashed on the screen each second. Because each image stays on your retina for a time the gradually changing images blend into each other so you get the impression of moving pictures.

This same scanning effect is used in the 8x8 display. In this way complex patterns like moving messages can be displayed. In the kaleidoscope program each LED seems to be on all the time, but it is not. It is only turned on for only 15 micro-seconds every 500 micro-seconds (half a milli-second.) This is duty cycle of 3% (Pulse width/period x 100/1.) When the LED is on it draws 70 mA but the average current drawn is only 2 mA (I_{av} = (Pulse width x I_{peak})/Period.) This is called **multiplexing** the display. Multiplexing is also used to turn on the six displays on the Southern Cross. This was discussed in Chapter 7 of the Users Manual.) As the continuously rated current of the LEDs is only 30mA and

we are allowing up to 70mA to flow you risk destruction of the LEDs in the display if you allow the LEDs to be turned on continuously.

When we use the SCAND subroutine only one row of LED's is on at any one instant of time. Because the average current drawn by the LED's is only 2mA no current limiting resistors are required by the circuit.

Multiplexing techniques use the persistence of vision effect to allow the display of complex messages. There is no reason to turn on continuously the LEDs in the display when they only have to be turned on 3% of the time to achieve the exactly same effect.

Programming the 8x8. We could start out by turning one LED on continuously then expanding the examples to turn on several LEDs. However, such examples lead nowhere since the normal way to program the 8x8 displays is by multiplexing for the reasons just discussed. The first example displays the letter A using a system call 16 to the Monitor. (For a discussion of system calls see Ch. 7 of the Users Manual.)

Multiplexing the 8x8 can be done in several ways. The SKATE subroutine we will use now (the one in the Monitor) is one way. Another way is shown later. In SKATE one row of 8 LEDs is scanned at a time. The LEDs to be turned on in that row are given by the bit pattern of the 8 positions. A bit pattern of 10000001 (or 81h) will turn on the outer two LEDs. A pattern of 11111111 (FFh) will turn them all on.

To program this the byte representing the top row is stored in the register pair HL. HL+1 stores the byte for the second row from the top, HL+2 the byte for row 3 etc. See Figure 1 and the method will be immediately apparent. We can conveniently use system call 16 to scan the 8x8 display rather than reinvent the wheel and write our own code. An example will show this more clearly.

Using a piece of paper form the letter A of your choice using the 8x8 matrix. We decided on 18, 24, 42, 42, 42, 7E, 7E & 42 as follows:

```
00011000        =18h
00100100        =24h
01000010        =42h
01000010        =42h
01000010        =42h
01111110        =7Eh
01111110        =7Eh
01000010        =42h
```

Do you see the capital A outlined by the 1's and how to derive the hex byte representing the 0 & 1 pattern? Hand enter these bytes into locations 2000h to 2007h of the Southern Cross. Next enter the code in Table 1 on the next page at 2100h then do Fn 0.

You should have the letter A displayed on the 8x8. (If not move the switch to address the other pair of ports.)If you do not like its font then change it in the buffer at 2000h. Eg, change the first 2 bytes to 00,3C. That may look better to you.

Again this example has demonstrated how using the subroutines in the Monitor greatly simplifies code development and reduces time. Just 4 lines of code have put the contents of the 8 byte buffer on the display. Add some bit shift instructions, delays and a bigger message buffer and you can move a message across the screen. Or you can develop a maze game. Examples of each of these types of programs have been supplied on the floppy disk which accompanies this Kit. We present the programs here as further examples for you to study and to learn from. The complete code with comments for each program has been supplied. Download the hex file using the serial download method discussed in Chapter 8 of the Users Manual.

Maze Game. Maze is a simple maze program that should keep you amused for some time as you struggle to find the exit from this LED labyrinth. Best of all, once you have mastered it you can design your own to impress your friends! The program is written so that any size maze can be designed. All you need is the patience to code it.

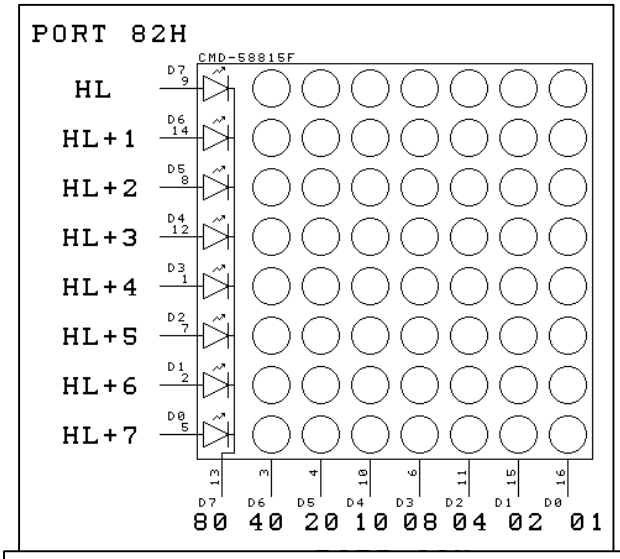

Figure 1. Port, Memory Buffer Location & Hex Coordinates.

Use the flashing LED 'cursor' to move around the maze in search of the flashing exit LED. The 2,5,7,A diamond will move the cursor. The Fn key will exit from the maze and take you back to the Monitor. Move your cursor anywhere there is not a LED 'wall'. As you move the cursor off the display another view is presented. If the cursor will not move off a view it means that there is a wall blocking you in the next view. Move the cursor onto the flashing exit LED to reveal your reward. Press any key to return to the monitor.

A description of how the maze game works and how you can design your own, bigger maze is contained in a text file on the floppy disk. Try to write the maze so that

scrolling is smooth rather than jumping an 8x8 unit at a time (one direction is harder to program than the other.)

```
2100    21 00 20    LD HL,2000h    ;point HL to buffer
2103    0E 16       LD C,16H       ;system call SKATE
2105    F7          RST 30H            ;call it
2106    C3 00 21    JP 2100h       ;repeat the loop
```

Table 1. System call to Output Display Routine.

Scrolling Message Display. Message is a 'running message display' program which will display a variable length message across two 8x8 units. System calls are not used but the displays are multiplexed. A different method of character encoding is used to that presented above.

Note a limitation of the z8t assembler has shown up in this example. The object code in the lookup table in the prn file output is only 4 bytes long. It does not show the fifth byte. It is in the hex output of course but not in the prn file output.

Characters are stored in a 5x7 character fonts matrix. All upper case letters, numerals and a range of special characters have already been coded and placed in the lookup table. It is left as an exercise for you to add the lower case letters to this table. Let us look how characters are stored in the table.

A 5x7 character matrix is used. Here is an example for the small letter a:

```
1 2 3 4 5    Byte number    1 2 3 4 5
 (0 0 0 0 0 ) This row all zeros
0 0 0 0 0                   0 1 1 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 0 1
0 1 1 1 1                   2 5 5 5 F
1 0 0 0 1                   h h h h h
0 1 1 1 1                     LSB
```

Character encoding is done in columns (since it is easier to program the scrolling movement.) To translate the above diagram into byte values mentally divide each column into two halves. The lower 4 bits make up the lower nibble and the upper 4 (with the extra row added) make up the higher nibble.

Using the binary to hex table on the right-hand side of the Southern cross keyboard find the corresponding hex digit of both halves. Do the top half first and write down the hex digit. Now do the bottom half and write down that hex digit to the right of the first. Repeat this process for all 5 columns. This process should be done from left to right and the resultant bytes recorded in the same order. The bytes are summarized on the right of the bit table above.

They are:
Byte 1 02H
Byte 2 15H
Byte 3 15H
Byte 4 15H
Byte 5 0FH

or '021515150F' when put into consecutive locations in the lookup table. Once you see how to do it it is easy.

Look at the last line in the lookup table at 2300h and you will see that 'a' is the last character entered in it. It is an exercise for you to add b, c, d ... z into the table after it.

Naturally the characters of any language or any symbols can be formed and stored in the above fashion. (An 8x8 character matrix may be better than the 5x7 one.)

Let us return to the running display program. The message to be displayed is converted into standard ASCII code either manually (that is, you do it) or a PC does it for you automatically. In the latter case all you do is enter the message in plain english into the .z8t file at 2400h and use the PC to assemble it for you. Of course you cannot enter symbols for which you have not defined the font! The first part of the program translates to ASCII codes of the message into groups of 5 bytes from the lookup table. All the bytes are transferred to a display buffer. The second part of the program scans the bytes onto the 8x8 displays and moves them across the displays. The method of moving the characters is interesting. Four pointers are set up and moved through the display buffer creating a new scanning starting point each time the display is to be shifted.

The program is fully commented. Try to understand the logic of how it works. It is very easy to see exactly how the program runs using the Southern Cross. Remember the hardware speed control? Move the jumper or switch to S)low. Adjust the potentiometer so the display is going slowest. You should be able to see how the program operates by looking at the displays. Each character is scanned repeatedly in a fixed position. Then all the characters move one LED position to the left. Move the pot to the fastest scanning and see what happens. Then move the switch to F)ast. The scanning of the characters in the fixed position is now so fast you do not see it. All you see are the letters moving across the screen. So by using the hardware speed control you have been able to see 'inside' the program, something you normally have to deduce logically by looking at the program flow chart.

Three 8x8 Displays. We have designed this 8x8 add-on board so that 1 or 2 boards may easily be attached to the CN1 I/O Port of the Southern Cross. The switch is up for one board, and down for the other. However, with minor hardware modifications three 8x8 displays may be attached. Port 80h, for example, may go the the Y latch of all three boards while ports 81h, 82h and 83h go to the X latches of each board respectively. The program 3_8x8.z8t on the floppy disk will run 3 8x8 displays. To drive more than three 8x8 displays you will have to use the 40 pin IDC socket CN3 and decode the port addresses

for yourself. The basis of a large, say 2x30 8x8 unit moving display has now been discussed in principle. Time and date information is available if you use the DS1216C smartWatch. Can you write a program to display this information on a moving display? There are many possibilities open to you

Conclusions. The 8x8 add-on board opens up the real world of programming. The essential interface output circuit between any computer and the outside world is found on the 8x8 Display Board. Whether you are driving a printer (magnetic print hammers), relay board (solenoids) or 8x8 display the basic circuit is taking a low power digital logic signal from an output Port of the computer and latching it (in the 74HC273) to a driver chip or transistor.

(Documentation September 11, 2002.)

- - - - - - - - - - - - -