Keywords: LCD,liquid crystal display,DS89C450

APPLICATION NOTE 4039

Using the DS89C450 as a Static LCD Display Controller

May 22, 2007

Abstract: Many of the Company's microcontrollers integrate controllers for LCD displays, which are implemented in hardware. Some microcontrollers like the DS89C450 do not offer this functionality, but one can implement a simple display controller in software. This application note describes how to drive a static LCD panel with 7-segment digits using the DS89C450 ultra-high-speed flash microcontroller.

Overview

Liquid crystal display (LCD) panels are used in a variety of modern electronic equipment like calculators, handheld blood glucose meters, gas station pumps, and television sets. Because of their lower power consumption and easy viewing in direct light, LCDs have replaced older LED displays in many applications. A range of microcontrollers (such as the MAXQ2000) integrate LCD controllers capable of driving LCD panels at up to a ¼-muxed duty cycle. But in some instances, the ideal microcontroller for a particular application may not integrate an LCD controller. For these situations it is possible to implement a display controller in software by using the microcontroller's port pins to drive the display.

This application note describes how to implement a display controller for a simple, static LCD panel with 7-segment digits by using the DS89C450 ultra-high-speed flash microcontroller. Because no features specific to the DS89C450 are used, this example code can easily be ported to any 8051-compatible microcontroller, as long as that microcontroller has a sufficient number of port pins to drive the LCD panel used in the application.

Example code for this application note is available for download (ASM).

Selecting an LCD Panel

When selecting an LCD panel for an application, take care to match the LCD with a compatible microcontroller or LCD display controller. The following questions should be considered when making this decision.

- What is the operating voltage range for the LCD? Since the DS89C450 is a 5V microcontroller and its port pins operate at 5V levels, we must select a 5V LCD panel. Note that many microcontrollers which integrate LCD controllers use a dedicated supply input (VLCD) to set the voltage range used by that LCD controller.
- What is the LCD's duty cycle? **Static** LCD panels connect each segment in the display to a dedicated drive line. This means that the number of segment drivers must equal the number of LCD segments to be driven. **Multiplexed** LCD panels, however, drive more than one LCD segment

with each segment drive line (SEG). These panels use multiple common backplane (COM) outputs and drive multiple levels between VLCD and GND on the SEG and COM lines, depending on the duty cycle being used. Because the DS89C450, our example 8051 microcontroller, can only drive its port pin lines to 5V and GND, our example is limited to a static LCD. For additional information on driving a multiplexed LCD, refer to the following documents:
    - Application note 3548, "Using an LCD with MAXQ Microcontrollers"
    - The MAXQ® Family User Guide: MAXQ2000 Supplement
- How many segment and common drivers are required to operate the LCD panel? When controlling a static LCD panel, one drive line (port pin) is needed for each segment to be driven, plus an additional port pin for the common (COM) backplane line.

For this application note, the Lumex® LCD-S401C52TR display was selected. This LCD is a 5V static display panel with four 7-segment digits and three annunicator segments (a colon and three decimal points). Each of the numeric digits on the LCD consists of seven segments, as illustrated in **Figure 1** where the A, B, G, E, and D segments are turned on to display a "2" digit.

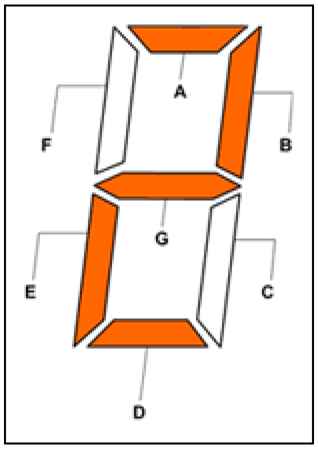

Figure 1. Seven-segment LCD display digit.

The LCD-S401C52TR display has a single COM backplane (connected to two pins) and 32 display segments, each connected to a segment drive pin. For this example, we will only use three of the 7-segment digits, which means that the DS89C450 will need to drive 21 SEG lines (seven segments for each of the three digits) and one COM line, thus requiring a total of 22 port pins. When not operating in the expanded memory bus configuration, the DS89C450 provides 24 push-pull port pins. The mircocontroller, therefore, has sufficient I/O capacity for this task. (An additional eight port pins are available on Port 0. However, these pins are open-drain and require additional pullup resistors to be usable as general-purpose I/O).

Hardware Setup

The hardware setup for this example was based on the DS89C450 Evaluation (EV) Kit (Rev B) with the memory interface CPLD (U5) and both external memory chips (U6 and U7) removed. This modification frees a number of additional port pins for use by our application, pins that would otherwise be used to implement the expanded memory bus, specifically Port 0 (all eight lines), Port 2 (all eight lines), Ports 3.6 and 3.7. See **Table 1**. (**Note**: Port 0 is not used in this example application.) The DS89C450 includes 64kB of internal code space and 1kB of internal data SRAM, which will be more than sufficient for this example.

The segment and common lines on the LCD-S401C52TR display were connected to port pins on the

DS89C450 by using the J4 header adjacent to the prototyping area. The segment lines were connected to the port pins through 1kΩ resistors instead of being connected to the port pins directly. This latter step was done because the DS89C450's port pins have higher drive capacity (strong pulldown for the 0 state and a one-shot, strong pullup followed by a weak pullup for the 1 state) than would normally be used by LCD panel drive lines. Because the COM line has a larger capacitance and requires a stronger driver, it was connected to its port pin directly. This application does not, however, recommend that the segment lines be driven directly by port pins. A problem occurs in that configuration: the capacitive coupling through the LCD display between the segment and common planes tends to pull the COM line away from its intended state as more and more segments turn on. (This problem happens because an active segment will always be the opposite voltage from the common plane.) As a result, segments which should be off, turn partially on. So connecting the port pins through the resistors to reduce their drive strength eliminates this issue.

Table 1. LCD Panel and Port Pin Connections

DS89C450 Port Pin	J4 Header Pin	LCD Pin(s)	LCD Signal	Notes
P1.0	1	21	4A	Through 1kΩ
P1.1	2	20	4B	Through 1kΩ
P1.2	3	19	4C	Through 1kΩ
P1.3	4	18	4D	Through 1kΩ
P1.4	5	17	4E	Through 1kΩ
P1.5	6	22	4F	Through 1kΩ
P1.6	7	23	4G	Through 1kΩ
P1.7	8	1, 40	COM	Connect directly
P2.0	21	25	3A	Through 1kΩ
P2.1	22	24	3B	Through 1kΩ
P2.2	23	15	3C	Through 1kΩ
P2.3	24	14	3D	Through 1kΩ
P2.4	25	13	3E	Through 1kΩ
P2.5	26	26	3F	Through 1kΩ
P2.6	27	27	3G	Through 1kΩ
P3.0	10	30	2A	Through 1kΩ
P3.1	11	29	2B	Through 1kΩ
P3.2	12	11	2C	Through 1kΩ
P3.3	13	10	2D	Through 1kΩ
P3.4	14	9	2E	Through 1kΩ
P3.5	15	31	2F	Through 1kΩ
P3.6	16	32	2G	Through 1kΩ

There are a few additional items to note about the hardware setup:

- A standard 16.384MHz crystal (inserted at Y1) was used to provide the clock for the DS89C450.
- When running the application, DIP switches SW1.1 and SW4.2 should be ON; all others should be OFF.
- When loading the application (using the MAXQ Microcontroller Tool Kit (MTK) or another development tool), DIP switches SW1.1, SW1.2, SW1.3, SW4.1, and SW4.2 should be ON; all others should be OFF.

- When the LCD display is running, activity from Port 1 will also be seen on the LED bar-graph display, U10. This is normal and, since the LCD display is buffered, does not affect the application.
- P3.0 and P3.1 are also used for the Tx/Rx lines of serial-port 0. Therefore, when the application is loading (using the serial-port bootloader), one or two segments on the LCD can flicker due to activity on these lines. This is normal. When the application is running, DIP switches SW1.2 and SW1.3 should be turned off to disable the serial-port function.
- Any unused segments on the LCD display should be driven *explicitly* to the OFF state and not allowed to float. This task can be done either by connecting one or more unused segments to a port pin that is driven to the OFF state (the same voltage waveform as COM), or by connecting unused segments to COM directly.

Driving LCD Segments

The default state of LCD segments is OFF (i.e., clear); when no voltage is applied, the segments become transparent and are invisible against the background in the LCD panel. In addition, when the *same* voltage is applied to both a segment line (SEG) and the common backplane (COM), the segment remains off. The segment will only switch to its ON (i.e., opaque) state when a voltage difference is applied between the SEG pin for that segment and the COM plane. As this voltage passes a particular level, known as the threshold voltage, the segment darkens and finally becomes completely opaque. The threshold voltage, which is a percentage of the specified operating voltage of the LCD panel, varies from one LCD to the next.

The polarity of the voltage differential does not matter for driving the LCD segments. A controller, for example, that drives an LCD with a 3V threshold voltage can switch on segment *n* either by setting COM to ground and SEG*n* to 3V, or by setting COM to 3V and SEG*n* to ground. This fact is important because, if a static DC voltage is left across an LCD segment for a long period of time, the segment can become damaged and will no longer switch properly. To avoid this problem, LCD segments are always driven with alternating waveforms to ensure that the overall DC voltage across each segment is always zero, whether the segment is in the ON or OFF state (**Figure 2**).

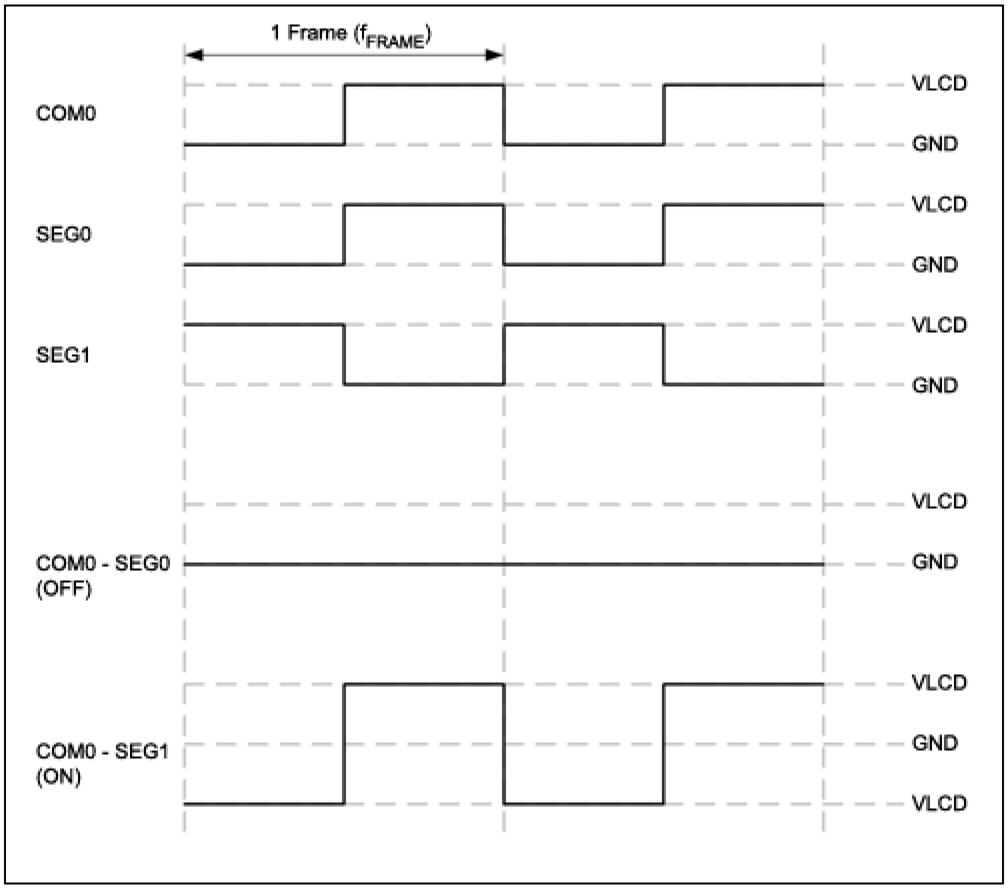

Figure 2. Alternating drive waveforms for static LCD segments.

As Figure 2 shows, the COM pin on a static display is constantly driven by a 50% duty cycle square wave between VLCD (5V for our setup) and GND. Each segment line is then driven by one of two patterns.

- To switch the segment OFF, it should be driven by a waveform **identical** to the one used to drive the COM pin. This will ensure that the DC voltage across the SEG/COM pair is always zero, which means that the segment will remain off.
- To switch the segment to ON, it should be driven by the **inverse** of the COM waveform. This means that half the time the segment will be driven by a positive voltage, and by a negative voltage the other half of the time. These two states have the same visual appearance, so the segment appears to be on constantly. Since the average DC value of the voltage difference is zero, no static DC bias remains which could damage the LCD glass.

The frequency at which the LCD is driven (known as the frame frequency) varies from one LCD panel to another. The proper value for a given application is usually derived by experimentation on a specific hardware setup. Since the rate at which an LCD segment can change state is limited by the overall capacitance of the segment, the LCD will operate properly only in a specific range of frame frequencies. Generally, this range runs from 20Hz to 200Hz. The example code for this application note runs the LCD at approximately 30Hz. Frame rates too high or too low for a particular display will cause the LCD segments to flicker or visually dim.

The main loop for the example application which drives the LCD segments is shown below.

```
Main:
   mov    IE, #080h           ; Disable timer 0 interrupt temporarily
   mov    R2, DigitP1         ; Grab local copies of digit variables
   mov    R3, DigitP2
   mov    R4, DigitP3
   mov    IE, #082h           ; Re-enable timer 0 interrupt

   mov    A, R2
   call   getDigit            ; Calculate segment pattern for ones digit
   anl    A, #01111111b       ; Ensure that COM (P1.7) is driven low
   mov    P1, A

   mov    A, R3
   call   getDigit            ; Calculate segment pattern for tens digit
   mov    P2, A

   mov    A, R4
   call   getDigit            ; Calculate segment pattern for hundreds digit
   mov    P3, A

;;;;  Delay loop  ;;;;

   mov    R0, #0FFh
L1A:
   mov    R1, #0FFh
L1B:
   djnz   R1, L1B
   djnz   R0, L1A

;;;;;;;;;;;;;;;;;;;;;;

   mov    A, R2
   call   getDigit            ; Calculate segment pattern for ones digit
   cpl    A                   ; Inverse of the pattern driven on the first
frame half
   orl    A, #10000000b       ; Ensure that COM (P1.7) is driven high
   mov    P1, A


   mov    A, R3
   call   getDigit            ; Calculate segment pattern for tens digit
   cpl    A                   ; Inverse of the pattern driven on the first
frame half
   mov    P2, A

   mov    A, R4
   call   getDigit            ; Calculate segment pattern for hundreds digit
   cpl    A                   ; Inverse of the pattern driven on the first
frame half
   mov    P3, A

;;;;  Delay loop  ;;;;

   mov    R0, #0FFh
L2A:
   mov    R1, #0FFh
L2B:
   djnz   R1, L2B
   djnz   R0, L2A

;;;;;;;;;;;;;;;;;;;;;;

   ljmp   Main                ; Go back for another frame cycle (endless loop)
```

Note that the COM line (connected to P1.7) is always driven with the same waveform: low for the first half of the frame, and high for the second half. For the segment lines, the pattern driven in the first half of the frame is inverted for the second half. Each of the three digits is connected in the same manner to each of the three ports, so that segment A is always connected to Px.0, segment B to Px.1, and so on. This configuration allows the example code to use the **`getDigit`** routine to calculate the segment pattern for each of the three LCD panel digits.

```
;*****************************************************************************
;*
;*  getDigit
;*
;*  Returns an LCD segment pattern (in Acc) for the decimal digit (0 to 9)
;*  input (also in Acc)
;*

getDigit:
   cjne   A, #0, getDigit_not0
;              xgfedcba
   mov    A, #00111111b         ; Zero
   ret
getDigit_not0:
   cjne   A, #1, getDigit_not1
;              xgfedcba
   mov    A, #00000110b         ; One
   ret
getDigit_not1:
   cjne   A, #2, getDigit_not2
;              xgfedcba
   mov    A, #01011011b         ; Two
   ret
getDigit_not2:
   cjne   A, #3, getDigit_not3
;              xgfedcba
   mov    A, #01001111b         ; Three
   ret

getDigit_not3:
   cjne   A, #4, getDigit_not4
;              xgfedcba
   mov    A, #01100110b         ; Four
   ret
getDigit_not4:
   cjne   A, #5, getDigit_not5
;              xgfedcba
   mov    A, #01101101b         ; Five
   ret
getDigit_not5:
   cjne   A, #6, getDigit_not6
;              xgfedcba
   mov    A, #01111101b         ; Six
   ret
getDigit_not6:
   cjne   A, #7, getDigit_not7
;              xgfedcba
   mov    A, #00000111b         ; Seven
   ret
getDigit_not7:
   cjne   A, #8, getDigit_not8
;              xgfedcba
   mov    A, #01111111b         ; Eight
   ret
getDigit_not8:
   cjne   A, #9, getDigit_not9
;              xgfedcba
   mov    A, #01101111b         ; Nine
   ret
getDigit_not9:
   mov    A, #0
   ret
```

Running the Counter

The pattern displayed on the LCD by the example code is a 3-digit decimal counter, which starts at 000 on power-up and increments to 001, 002, etc., until it reaches 999 and rolls over. Since the main loop of the program drives the LCD segment and common patterns, we must find another method to periodically increment the counter value. One solution is to periodically trigger an interrupt using Timer 0.

```
   mov    TMOD, #021h         ; Timer 1: 8-bit autoreload from TH1
                              ; Timer 0: 16-bit
   mov    TCON, #050h         ; Enable timers 0 and 1
   mov    CKMOD, #038h        ; Use system clock for all timer inputs

   mov    IE, #082h           ; Enable timer 0 interrupt
```

Each time the timer interrupt occurs, a delay counter in register memory is decremented. When this delay counter reaches zero, the LCD 3-digit counter value is incremented by one (with each digit rolling over as needed); the delay counter reinitializes to its maximum value. Since Timer 0 is 16 bits in width and since the example code sets the delay counter to 20, the 3-digit counter will increment approximately every (1/16.384MHz) × (2^{16}) × 20 = 0.08s, or about 12 times per second.

```
   org     000Bh              ; Timer 0 interrupt
   ljmp    IntTimer0


;*****************************************************************************
;*
;*  IntTimer0 (INTT0)
;*
;*  Timer interrupt service routine
;*

IntTimer0:
   push    ACC                ; Save off accumulator and R0
   push    R00

   mov     R0, Count          ; Only increment LCD digits every [CountMax]
                              ; interrupt cycles
   djnz    R0, INTT0_Done

   inc     DigitP1            ; Increment ones digit on display
   mov     A, DigitP1
   cjne    A, #10, INTT0_Continue      ; Check for rollover

   mov     DigitP1, #0
   inc     DigitP2            ; Increment tens digit on display
   mov     A, DigitP2
   cjne    A, #10, INTT0_Continue      ; Check for rollover

   mov     DigitP2, #0
   inc     DigitP3            ; Increment hundreds digit on display
   mov     A, DigitP3
   cjne    A, #10, INTT0_Continue      ; Check for rollover

   mov     DigitP3, #0

INTT0_Continue:
   mov     R0, CountMax       ; Reset to starting cycle count
INTT0_Done:
   mov     Count, R0          ; Update cycle counter
   pop     R00
   pop     ACC                ; Restore accumulator and R0
   reti
```

Conclusion

As with many dedicated digital peripherals on a microcontroller, a static or multiplexed LCD display controller can be implemented in software if necessary. The simplicity of a static display makes this implementation particularly straightforward. The standard, general-purpose I/O functionality of an 8051 microcontroller like the DS89C450 can be used to drive the SEG and COM waveforms on the LCD. Using the high-performance DS89C450 ensures that plenty of processing power remains for the bulk of the application even after the LCD controller is implemented in software.

Lumex is a registered trademark of Illinois Tool Works Inc.
MAXQ is a registered trademark of Maxim Integrated Products, Inc.

Related Parts		
DS89C430	Ultra-High-Speed Flash Microcontrollers	Free Samples
DS89C450	Ultra-High-Speed Flash Microcontrollers	Free Samples
DS89C450-K00	Evaluation Kit for the DS89C450	