Application Note

Software Defined Glass LCD Solution Based on MSPM0 MCUs

Jace Hall

ABSTRACT

This document presents a method of implementing a segment LCD controller via a software method. This method can be utilized to bring segment LCD capabilities to devices that do not have hardware modules available for controlling segment LCDs. This document explains the software method used in detail; however, details on how segment LCDs operate and the different implementation available are outside the scope of this document.

The software associated with this document can be found under the "Demos" folder for a given MSPM0 MCU in the MSPM0 SDK.

Table of Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Glass Segment LCD Basics

A glass segment LCD typically has two glass plates joined together with a small gap in between the plates. This gap contains a liquid crystal fluid, which is transparent under normal conditions. However, if an electric field is applied through a voltage differential, the molecules arrange themselves to align with the field direction and a black segment is visible on a clear background. The potential difference is applied to the segment LCD through an alternating, and repeating signal through the segment (*SEGy*) and common (*COMx*) lines attached to the display. *COMx* and *SEGy* lines are driven in opposite polarity in order to create said voltage differential. The voltage applied between *COMx* and *SEGy* lines is typically alternated to limit DC offset levels, thus extending LCD lifetime. Consult the segment LCD's manufacturer's specifications for max DC offset allowed.

The optical contrast, or how dark a segment is compared to the background, is controlled by how large of a differential between the RMS voltage of an ON segment, and an OFF segment. Since a repeating signal is used, the frequency of the signal has a visual effect as well. If the frequency used is too slow, then one would perceive a “flickering” of the screen. Faster frequencies do not have this issue, but consume more power. Most segment LCDs have a typical driving frequency between 30 Hz to 200Hz, which is well within the range of a software based solution.

1.1 Driving a Segment LCD

Inside the LCD assembly, the *COMx* and *SEGy* lines are arranged in a grid pattern. The COM lines are typically referred to as the back-plane of the LCD. This software solution implements a quadruplex back-plane, which means there are four *COMx* lines that are driven in sequence. To turn a particular segment on, the particular coordinate is activated by ensuring the associated *COMx* and *SEGy* lines are driven opposite of each other, and then alternated as shown in Figure 1-1. To keep a segment off, the appropriate *SEGy* line is driven in the same pattern as the *COMx* line that is currently being cycled through.

From an MCU perspective, the appropriate pins need to be driven high, low, or floating in a repeated pattern to display an image or message. An example of one LCD update cycle is shown in Figure 1-1. When a *COMx* line is active, it is first driven low for an update period, then high the following period. When inactive, the pin is set to floating and thus driven to VDD/2 via the external resistors described in Section 2. To turn a *SEGy* on, it is driven high, then low over two periods, and follows the same pattern as the *COMx* signal to keep it off.

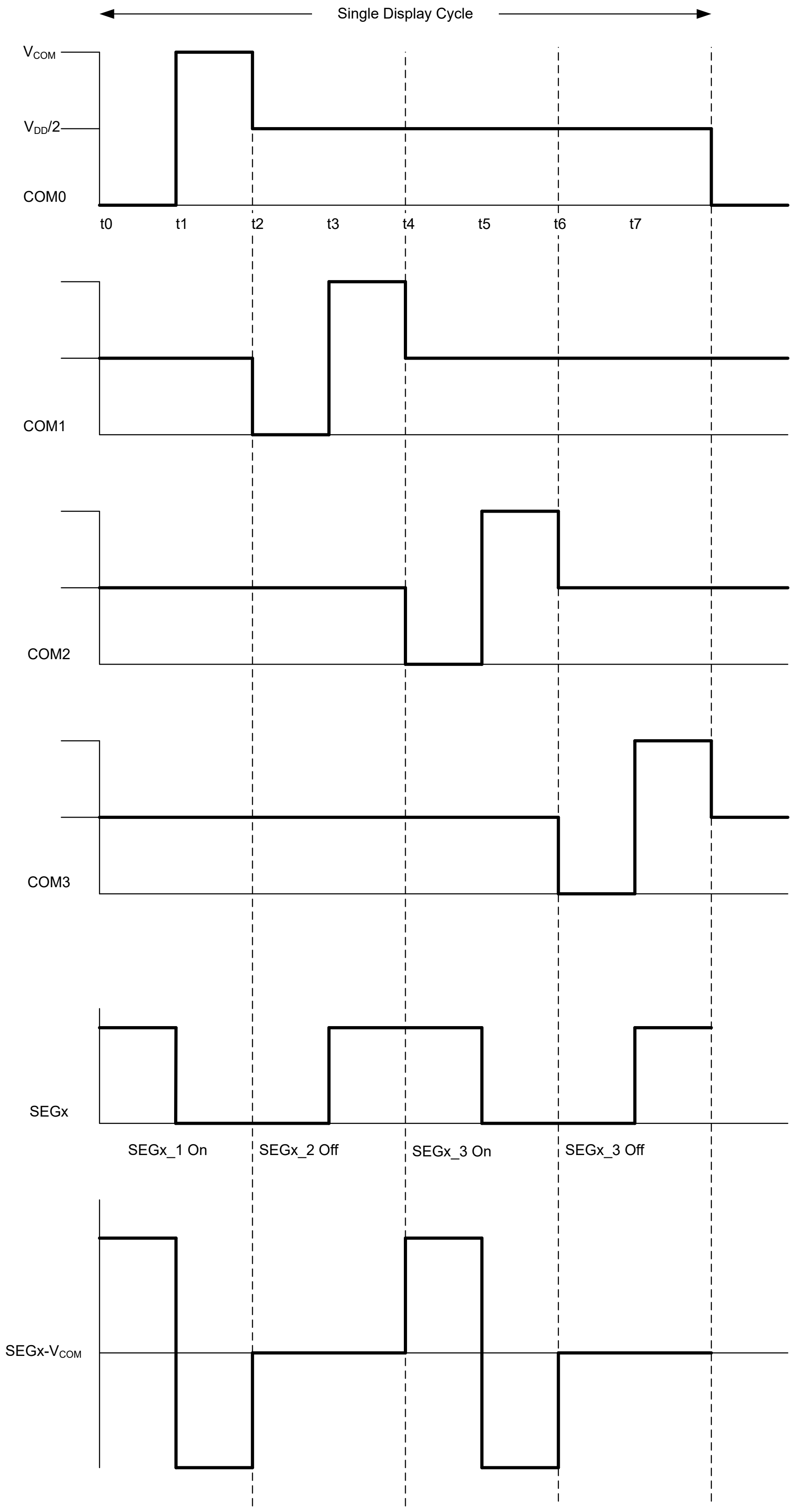
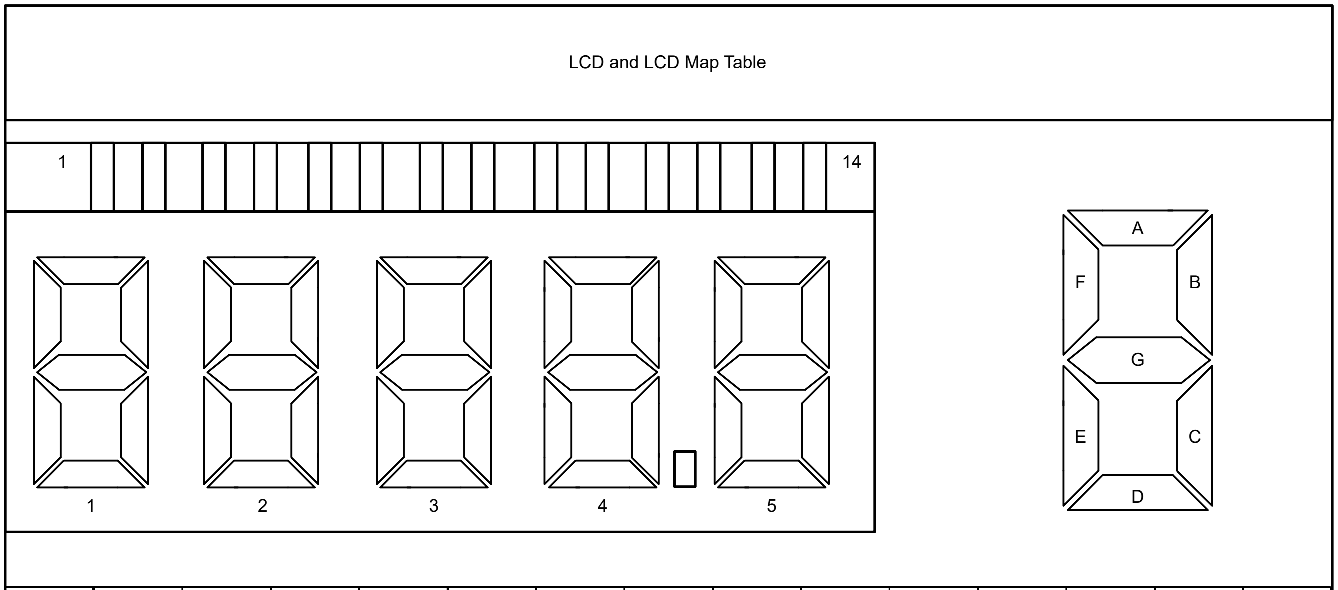


Figure 1-1. Segment LCD Cycle Diagram

1.2 LCD Mapping

With every LCD, documentation is provided as to the mapping of LCD to different *COMx* and *SEGy* lines. A pair of segment lines is combined with four *COMx* lines to generate one numeric or alphabetic (English) character. Special symbols are given specific *COMx* + *SEGy* crosses, or utilized in the spare cross from the typical character mapping. An example of an LCD map can be seen in Figure 1-2.



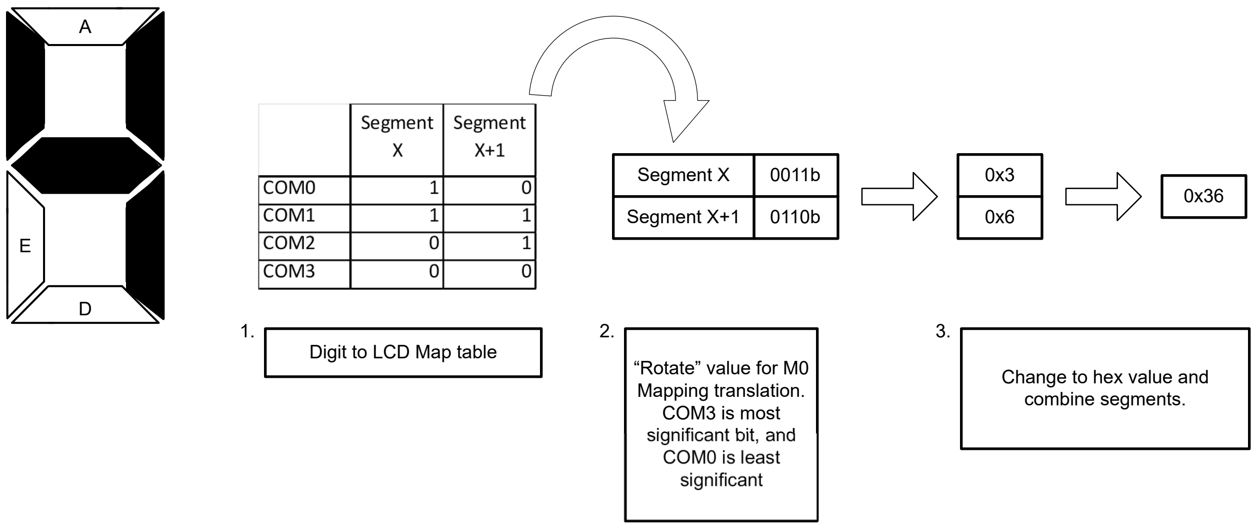
PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14
COM0				COM0	1F	1A	2F	2A	3F	3A	4F	4A	5F	5A
COM1			COM1		1G	1B	2G	2B	3G	3B	4G	4B	5G	5B
COM2		COM2			1E	1C	2E	2C	3E	3C	4E	4C	5E	5C
COM3	COM3					1D		2D		3D		4D	DOT1	5D
Segment	-	-	-	-	0	1	2	3	4	5	6	7	8	9

Figure 1-2. Segment LCD Map Example

From a software perspective, a character needs to be translated from the LCD map, to a numerical representation that can be utilized to determine which segments, and thus pins, needs to be turned on or off at each *COMx* stage. This translation would need to be done for each character one would want to display, and for each special symbol on the LCD. For this solution, perform the following steps to do a translation:

1. Determine character to be displayed.
2. Build out a table of the four *COMx* lines and two adjacent *SEGy* lines, placing a “1” corresponding to where a character needs a segment on, and a “0” for where a segment needs to be off.
3. Rotate this table so the highest *COMx* line is the most significant bit of the binary number created. This is how the crosses will be represented in the MCU memory.
4. For better understanding, change the binary value to hex representation, and combine the segments into a single 8-bit character, where the most significant bits are the representation of the lowest segment.

Figure 1-3 walks through an example of this translation with the character “4”. This translation will be stored in a lookup table described in Table 3-1.


A
E
D
Segment X
Segment X+1
COM0 1 0
COM1 1 1
COM2 0 1
COM3 0 0
1.
Digit to LCD Map table
Segment X 0011b
Segment X+1 0110b
2.
“Rotate” value for M0 Mapping translation. COM3 is most significant bit, and COM0 is least significant
0x3
0x6
0x36
3.
Change to hex value and combine segments.

	Segment X	Segment X+1
COM0	1	0
COM1	1	1
COM2	0	1
COM3	0	0

Figure 1-3. Mapping Characters

2 Hardware

Segment LCDs are commonly referenced to how many segments it is capable of displaying, following by the arrangement of COM by SEGMENT pins. For example, a 96-segment LCD could be in an arrangement of 4 x 24 or 3 x 32, COM pins by SEGMENT pins.

From a connection standpoint, you would connect the Segment LCD module to the MSPM0 as described in Figure 2-1. External resistors, configured as a voltage divider at half supply, are needed for the *COMx* lines in order to bias the voltage to a “neutral” state when inactive. The exact biasing needed can vary from different LCD manufacturers.

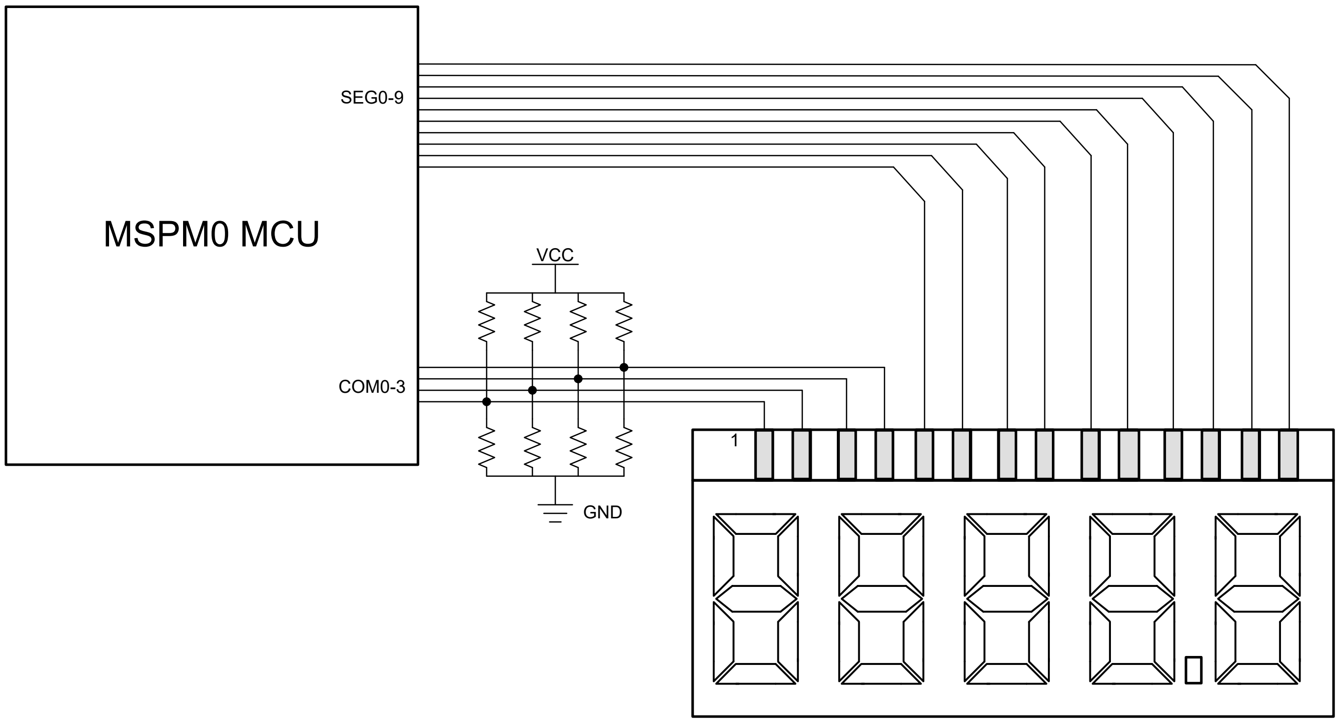


Figure 2-1. MSPM0 to Segment LCD Connections

2.1 Choosing Pins to Connect to LCD Module

Several factors come into play when choosing which pins to utilize for the MSPM0 software LCD implementation in order to simplify layout and software, while also allowing for other functionality that is muxed with the pins.

On the subject of layout, ideally LCD pins are chosen to be grouped together, in order, and on the same side of the chip as the LCD module. These factors minimize signal crossing, and keep LCD pins away from pins used to condition or sample analog signals.

From a software perspective, this solution utilizes the fact that pins can be addressed as a whole port of up to 32 pins at the same time. It also requires that every two segments are adjacent to each other from a port perspective. For example, if you had two *SEGy* lines, SEG0 and SEG1, they could be assigned to pins PA2 and PA3, respectively, but could not be assigned to PA2 and PA4 as they are not adjacent in the register. The software also assumes that if LCD assigned pins span more than one port, (e.g. PAx + PBx), that there is a clear delineation of *SEGy* lines between the ports. For more examples, see Figure 2-2. *COMx* lines from a software perspective, only require all of the *COMx* line to be on the same port.

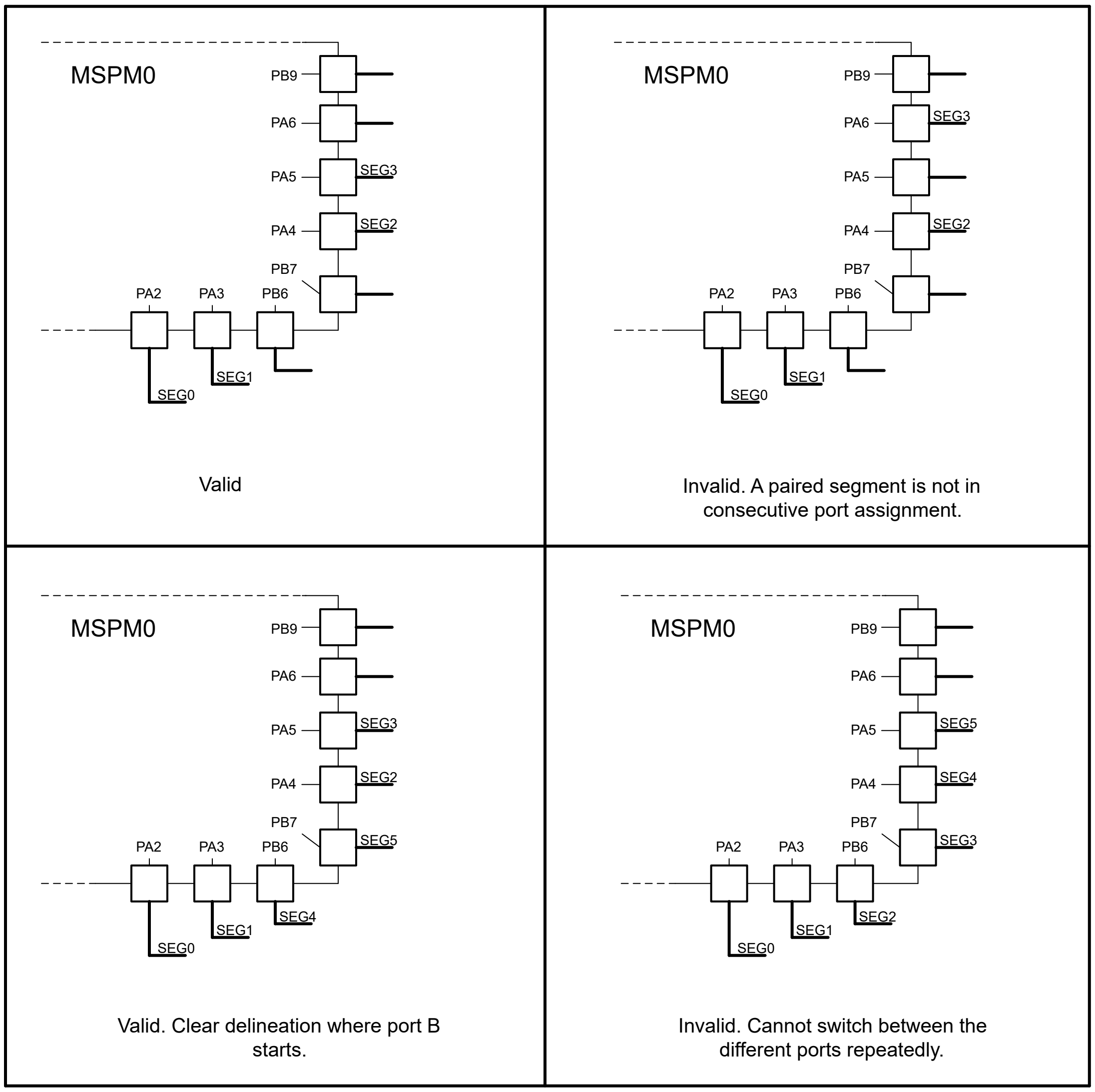


Figure 2-2. Pin Assignment Examples

From an application perspective, you must also balance which pins to use depending on the functionality you need coming out of the device, such as available communication interfaces, PWMs, or analog features. TI's Sysconfig tool can be utilized to make this job easier.

These factors can make choosing pins for SW LCD usage a challenge on certain packages within the MSPM0 devices, and trade offs may need to be made depending on your application. Please also keep in mind that pins defined as next to each other from a port register perspective (PA2 and PA3) may not be located physically beside each other on a package. On devices with multiple ports, it is possible to have pins physically beside each other be addressed by alternating ports along the package.

3 Software Implementation

Two different projects are provided for the MSPM0 software LCD controller. One focused on MSPM0 devices with a single GPIO port available, “MSPM0_SW_LCD_Single_Port”, and the other project targeted to MSPM0 devices that support more than one GPIO port, “MSPM0_SW_LCD_Multi_Port”.

Note

For the duration of this document, the “MSPM0_SW_LCD_Multi_Port” is discussed as it contains the same procedures as the single port version, but with added steps to setup and determine which port the *SEGy* lines are on.

3.1 Customizing the Software LCD Code

Sysconfig is utilized to select and name the needed GPIO pins and timer resource needed by the software LCD solution. The provided Sysconfig file already has the appropriate naming provided for a working solution. Add, remove, or modify the provided pin-out to suit your application needs, while observing the recommendations provided in the Section 2 section. Sysconfig is also where you can change timer clock settings, and thus the LCD refresh rate. If you need to hit a specific refresh rate, keep in mind that the LCD_Update_Clock described in Section 3.2, needs eight update cycles to refresh the entire LCD screen. This means you will need to be servicing the timer interrupt at eight times the refresh rate. For example, if one wanted an LCD refresh rate of 30Hz, then the timer must be setup to interrupt the device at a rate of 240Hz to meet that refresh rate.

Table 3-1 showcases the code definitions and maps that need to be modified to fit your solution. This includes customizing to your chosen pins and the characters to LCD mapping as discussed in Section 1.2. These definitions are found at the beginning of the principle .c file for each project.

Table 3-1. Important Code Definitions and Maps

Type	Name	Details
#define	COM_LINE_TOTAL	Number of COM lines used - default 4
#define	SEGMENT_LINE_TOTAL	Number of Segment lines used
#define	SEGMENT_LINE_LOWER_PORT_MASK	Port mask of used pins for segments in lower Port A
#define	SEGMENT_LINE_UPPER_PORT_MASK	Port mask of used pins for segments in upper Port B
#define	SEGMENT_LINE_PORT_SPLIT	First segment where an upper port pin is assigned. Corresponds to first upper port assignment in gSegmentOffsetMap
uint8_t array	gNumberMap[]	Array that maps different characters to LCD map.
uint8_t array	gSegmentOffsetMap[]	Map of different segments lines across a port. SEGMENT_LINE_LOWER_PORT_MASK must map these segments to port. All Port A segments (lower) must be listed before Port B segments (upper).

3.2 Software Flow

Figure 3-1 shows the software flow diagram for the two main pieces of code of this solution: the LCD_WriteCharacter function and the interrupt handler for the LCD_Update_Clock.

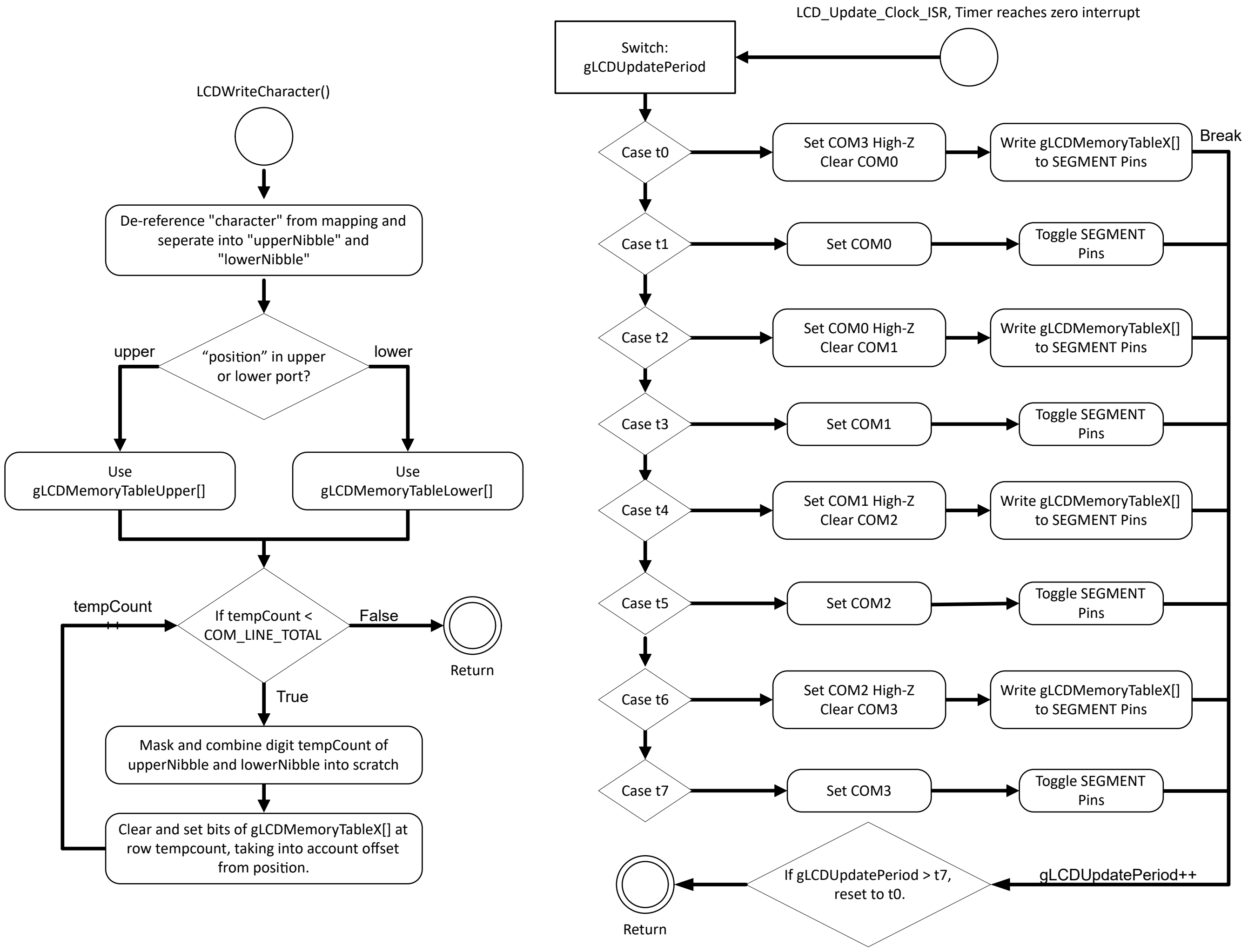

LCDWriteCharacter()
De-reference "character" from mapping and seperate into "upperNibble" and "lowerNibble"
"position" in upper or lower port?
upper
lower
Use gLCDMemoryTableUpper[]
Use gLCDMemoryTableLower[]
tempCount ++
If tempCount < COM_LINE_TOTAL
False
Return
True
Mask and combine digit tempCount of upperNibble and lowerNibble into scratch
Clear and set bits of gLCDMemoryTableX[] at row tempcount, taking into account offset from position.
LCD_Update_Clock_ISR, Timer reaches zero interrupt
Switch: gLCDUpdatePeriod
Case t0
Set COM3 High-Z Clear COM0
Write gLCDMemoryTableX[] to SEGMENT Pins
Break
Case t1
Set COM0
Toggle SEGMENT Pins
Case t2
Set COM0 High-Z Clear COM1
Write gLCDMemoryTableX[] to SEGMENT Pins
Case t3
Set COM1
Toggle SEGMENT Pins
Case t4
Set COM1 High-Z Clear COM2
Write gLCDMemoryTableX[] to SEGMENT Pins
Case t5
Set COM2
Toggle SEGMENT Pins
Case t6
Set COM2 High-Z Clear COM3
Write gLCDMemoryTableX[] to SEGMENT Pins
Case t7
Set COM3
Toggle SEGMENT Pins
If gLCDUpdatePeriod > t7, reset to t0.
gLCDUpdatePeriod++
Return

Figure 3-1. Software Flow Diagram

The LCD_WriteCharacter() takes in index to the character map array (gNumberMap) that corresponds to the alpha-numeric character you want to display on the LCD, as well as an index to the segment map (gSegmentOffsetMap) that corresponds to the character position on the LCD. The function parses the character and writes it to the appropriate LCD memory table (gLCD_MemoryTable_Lower or gLCD_MemoryTable_Upper). Due to this structure, the LCD_WriteCharacter() can be called at any point of the application, and update the LCD memory table asynchronously to the LCD update clock.

The LCD_Update_Clock interrupt handler is the main driver of the SW LCD solution. It includes a state machine that replicates the waveform described in Figure 1-1 earlier, where each case represents one of the time instances shown, and the following actions are taken.

For even cases:

- *COMx*-1 is placed in high-z sate
- *COMx* is set low
- All *SEGy* lines are set or cleared according to the LCD memory table

For odd cases:

- *COMx* is set high
- All *SEGy* lines are toggled

3.3 Integrating Software LCD Solution Within an Application

When utilizing this solution within an application, there are a few key care abouts with this solution. Since the LCD memory tables can be updated asynchronously, visual anomalies on the LCD can occur if the memory tables are updated too frequently in relation to the LCD refresh rate. These anomalies can be reduced or avoided by increasing the LCD refresh rate, only updating the LCD memory tables at the beginning/end of an update LCD update cycle, ensuring a character is displayed a minimum number of LCD refresh cycles before being changed, or a combination of these practices.

As with most software based solutions, timing is critical to the success of the solution. Care must be taken to service the LCD_Update_Clock interrupt handler in a timely manner to maintain solution stability. Frequent or long interrupt service routines outside this solution, can prevent proper operation of the software LCD solution. Applications can mitigate this by using a timer instance with higher interrupt priority than others in the system, increasing the interrupt priority of the timer instance used, keeping other application interrupts short, or using a software flag-loop method for other interrupts in the application.

4 Additional Resources

- Texas Instruments: *MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual*
- Texas Instruments: *MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual*
- MSPM0 Software Development Kit
- TI Sysconfig Tool
- MSPM0 SW LCD Muilt-Port Software
- MSPM0 SW LCD Single-Port Software
- Texas Instruments: *Software Glass LCD Driver With MSP430*
- Texas Instruments: *Designing With MSP430™ MCUs and Segment LCDs*