# Sharp Memory LCDs: Theory, Interfacing, and Programming

*Ken Green, Sharp Microelectronics of the Americas*

## INTRODUCTION

Sharp's Memory LCDs represent a step forward in power-saving monochrome displays. The modules have an SPI interface and do not require the usual LCD controller. Most of the panels have a 10-conductor 0.5 mm pitch FFC that contains all of the signals needed, including power.

This Application Note will touch briefly upon the theory of operation, as well as explain some general steps to programming the modules, and give some practical examples.

## THEORY OF OPERATION

Sharp's Memory LCDs derive their name from the way the LCD pixel array is used to store image data. Each pixel in the LCD array forms a 1-bit write-only memory; therefore the frame buffer is internal to the panel. This feature relieves the processor and bus from the over-head of continuous data transfers to refresh the image, as the image only needs to be written once to be retained. The image is retained indefinitely as long as there is sufficient current to do so. The LCDs also do not exhibit image retention problems, so there is no worry about keeping a single image displayed for a very long time.

When a change to the displayed image is desired, there is no need to clear and rewrite the entire existing image; only the lines that are to be changed need to be written. The displays have a minimum addressable unit of one line.

Since the displays are write-only, there is no way to read back any information; so if part of a line is to be changed, a copy of it will have to be maintained else-where, such as in local memory. This is because the new data for the line must be merged into the other, existing data for that line; since the minimum address-able unit is one line. After the data has been inserted, the line to be changed can be sent to the LCD. If major parts of the entire image are to be changed, then the entire frame should be duplicated in the processor memory.

Using two panels as an example for how to compute the necessary memory requirements, we get:

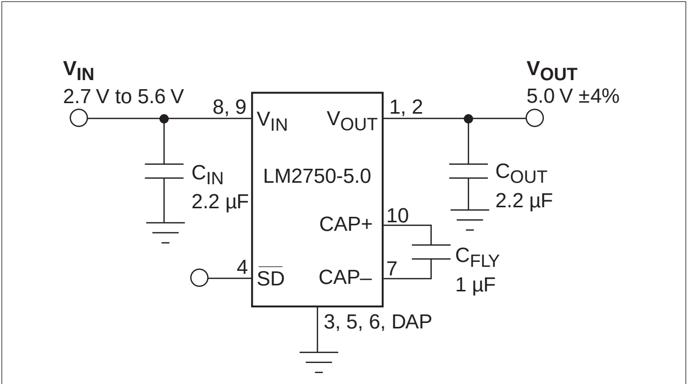
- 1.28-inch (128 × 128 pixels) = 128 × 128 / 8 = 2048 bytes
- 2.7-inch (400 × 240 pixels) = 400 × 240 / 8 = 12000 bytes

## INTERFACE

The interface to most of the panels is through a 10-conductor FFC with 0.5 mm pitch. A sample con-nector is the Hirose FH12-10S-0.5SH(05). The signals are compatible with 3.3 V logic levels. The SPI clock can run up to 2 MHz for 5 V parts; up to 1 MHz for 3 V parts.

## USING 5V PANELS IN 3V SYSTEMS

Some of the Memory LCDs require 5 V (because of their size), and some require 3 V. However, because memory LCD panels are such low power, it is still easy to use panels that require 5 V in systems that only have 3 V available. A simple high efficiency charge pump circuit can be used to boost the 3 V to 5 V as shown in Figure 1.



NOTES:
$I_{OUT}$ up to 120 mA, ($V_{IN} \geq 2.9$ V)
$I_{OUT}$ up to 40 mA, ($V_{IN} \geq 2.7$ V)
Capacitors: 1.0 µF - TDK C1608X5R1A105K
2.2 µF - TDK C2012X7R1A225K
LCD45-11

**Figure 1. Charge Pump Voltage Boost Circuit**

## VCOM

When designing with a memory LCD panel, a decision must be made as to how VCOM will be generated. VCOM is an alternating signal that prevents a DC bias from being built up within the panel. Memory LCDs do not generate this signal internally. It can be supplied using one of two methods: software, or an external clock. The method is selected by the EXTMODE pin on the interface.

If external clock is selected [EXTMODE = H], a clock must be supplied to the EXTCOMM pin. See Figure 2. The clock is a pulse sent to the panel at a constant rate. Every time a pulse is sent, the internal level of VCOM toggles (much like a flip-flop). The rate is spec-ified in each part's corresponding Specifications.

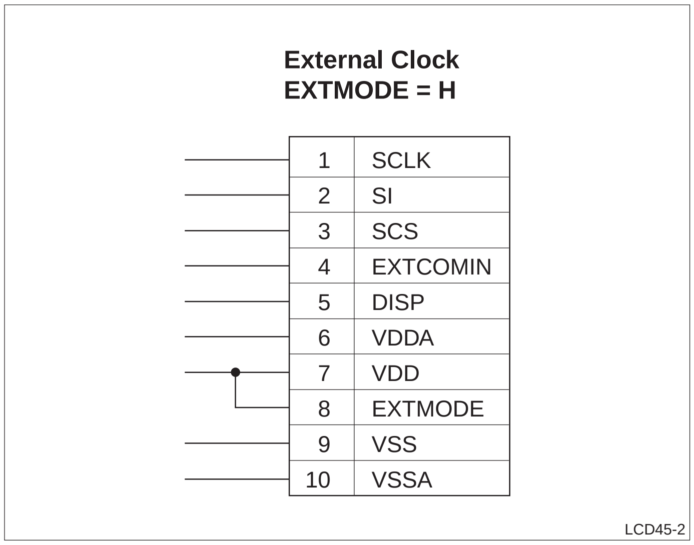


**Figure 2. Selecting External Clock**

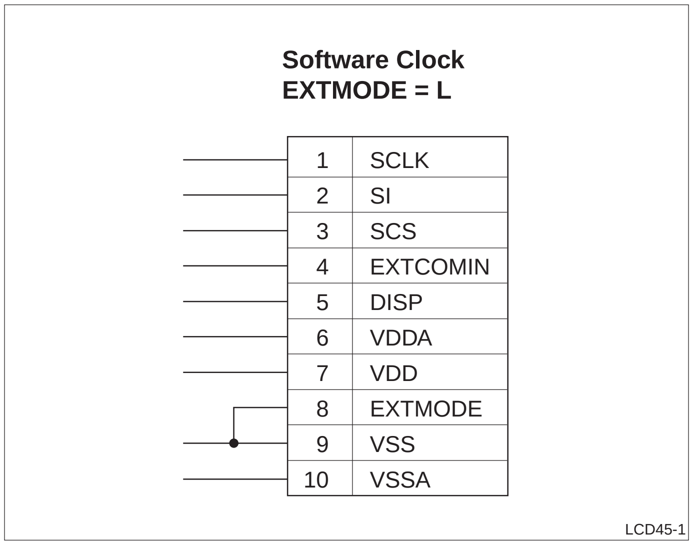


**Figure 3. Selecting Internal Clock**

In general, for panels that require a 3 V power supply, the frequency of EXTCOMM is 60 Hz. For 5 V panels the frequency is slower and more lenient, 1 – 20 Hz; again, check the Specifications. Higher frequency gives better contrast; lower frequency saves power.

When the software clock is selected [EXTMODE = L], bit V of the command bit string reflects the actual state of VCOM. See Figure 3. This bit must change states (by writing to the panel) at the frequency listed in the corresponding Specifications.

Any command can be used to change the state of the VCOM bit. If no update to the pixel memory is needed when it is time to change VCOM, the Change VCOM command can be used to change the state. It is important that the time intervals between VCOM level changes be as symmetrical as possible. In the descriptions below, the V bit represents the desired VCOM state if software VCOM control is selected. If external clock is selected, then the V-bit state doesn't matter.

## LCD COMMANDS

There are four commands that can be sent to a memory LCD panel:

- Write Line
- Write Multiple Lines
- Change VCOM
- Clear Screen

The commands are explained below. The illustrations are structured to show the first bits being sent are on the left. The very first bits to be sent out are the mode bits. A common mistake made is sending the bits in the wrong order.

Some SPI modules can only send data MSB first. The data order for these modules will have to be reversed before being sent.

## WRITE LINE

The minimum amount of data that can be written to the panel is one line. The actual number of data bytes written depends on the horizontal resolution of the panel itself. Therefore, a panel with a resolution of 400 × 240 will require a 400/8 = 50 bytes of data (plus overhead). See Figure 4.

The command structure for Write Line is as follows:

- Command: 8 bits (see above regarding V-bit)
- Line address: 8 bits
- Data bits: leftmost pixel first, with data width depending on the resolution of the panel. For instance, a panel 400 pixels wide will require 400 data bits.
- Trailer: 16 bits. These clocks allow the panel time to transfer the data from the incoming latch to the pixel memory.

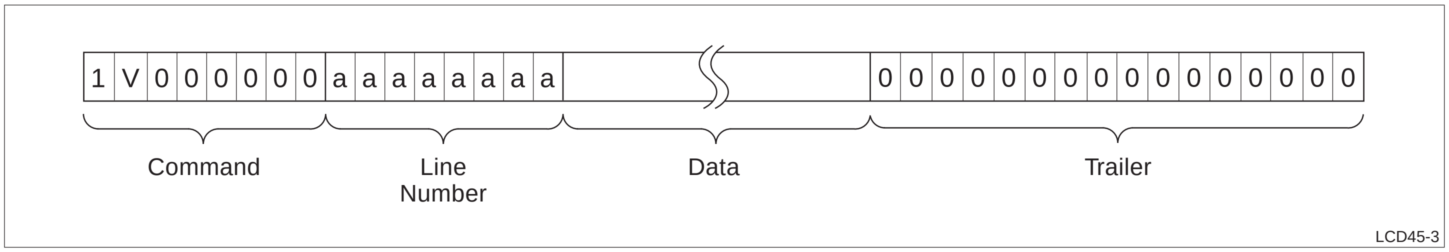


**Figure 4. Write Line Data String**

## WRITE MULTIPLE LINES

Multiple lines can be updated quickly with this command. A line address is still used so the lines do not have to be successive.

This command begins the same as the Write Line command; using the same Write Line command bits and data bits. See Figure 5.

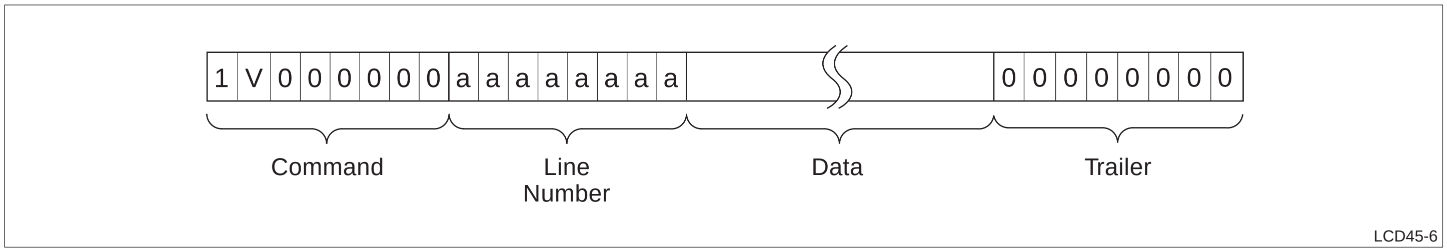


**Figure 5. Write Multiple Lines Data String, First Line**

After the data bits follow 8 trailer bits (instead of 16), then the address of the next line (8 bits) to be written, followed by the data bits for that line, and so on until all of the desired lines are written. See Figure 6.

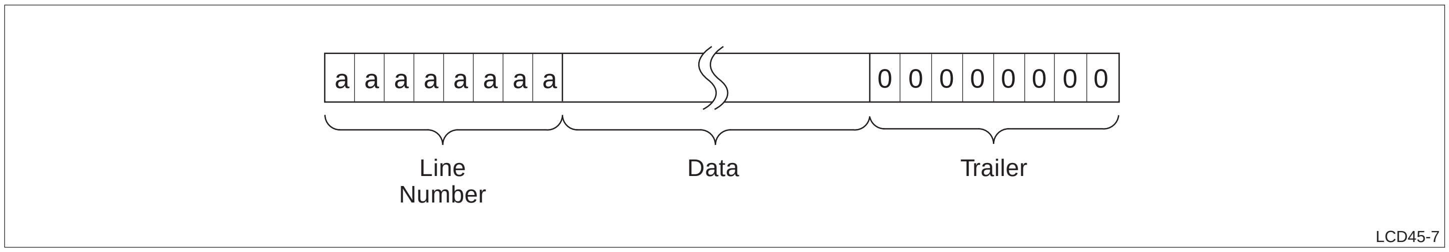


**Figure 6. Write Multiple Lines Data String, Intermediate Line**

For the last line to be written, use 16 trailer bits (see Figure 7) instead of 8 bits and a line address.

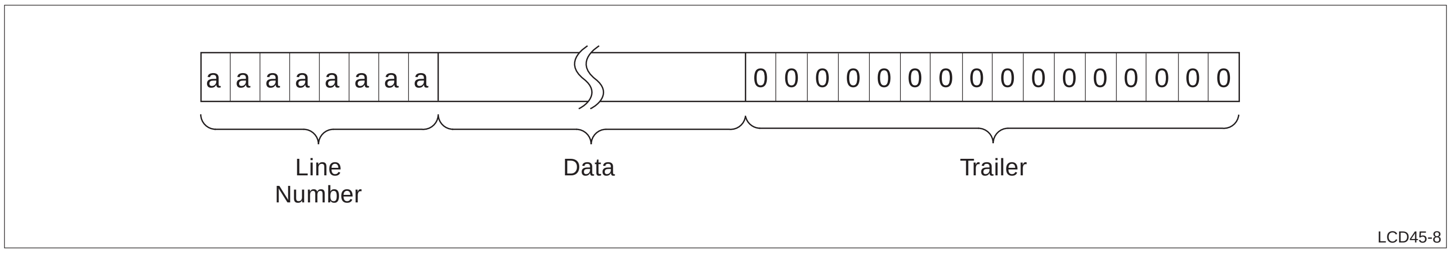


**Figure 7. Write Multiple Lines Data String, Last Line**

## CLEAR SCREEN

This command clears the screen to all white by writing 0's to all of the memory locations in the frame buffer. See Figure 8. The command structure is as follows:

- Command: 8 bits (including V-bit)
- Trailer: 8 bits (allows for latch transfer time)

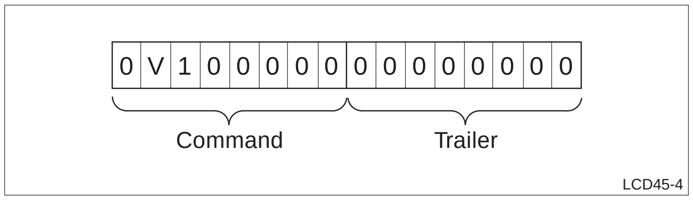


**Figure 8. Clear Screen Data String**

## CHANGE VCOM

This command is only used if [EXTMODE = L]. It is used to change the state of VCOM if no other command is sent within the appropriate amount of time to maintain proper VCOM frequency. See Figure 9. Note that all commands have the ability to change the state of VCOM. The command structure is:

- Command: 8 bits
- Trailer: 8 bits (allows for latch transfer time)

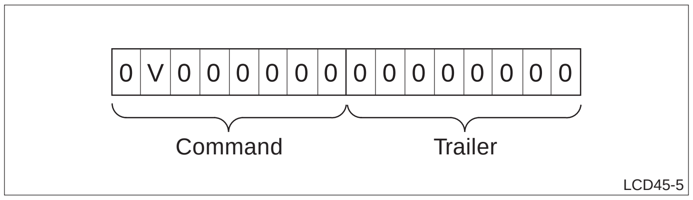


**Figure 9. Toggle VCOM Data String**

## SOFTWARE IMPLEMENTATION

The SPI port should be set up so that, in the idle state, the serial chip select (SCS) and the serial clock (SC) are LOW. Data is clocked on the rising edge of SC. Care should be taken to meet all of the timing such as setup and hold times and SC clock speed as given in the specifications. The SCS pin should go HIGH before any clocks on the SC line occur. The SCS pin should stay HIGH while the command and all data bytes are sent, and drop LOW after the trailer bytes are sent.

Sample code for interfacing with the panel is shown below. When sending a line of data to the panel, the first pixel data byte sent contains the first 8 pixels for that line with the first bit sent being the data for the left most pixel. Most graphics software packages organize the frame buffer with the upper left-most pixel being the LSB of the first byte of the buffer. This means that the SPI module must shift out the LSB (bit 0) first. If this is not the case for your system, you may have to swap the order of the bits.

As mentioned before, the minimum addressable item is one line. Even if only one bit in a line is changing, all the data for the line must be sent. Pixels that have not changed can either be recreated or stored. Storing all of the data for all lines is more memory intensive, but can be easier to implement. If the images are more text based, and maybe only a few lines change at a time, recreating a few lines of text will require less processor memory. The memory will of course need to be RAM memory.

If creating a full frame buffer is desired, the best way to use it is to pre-format it with the required overhead bytes. The software routine MLCD_InitFrameBuf below does this. Location 0 of the frame buffer starts with a write command plus a line address of 1 for the first line. The routine shown below does not format the data bytes, but it could be added if desired. The required data bytes are skipped over and the trailer bytes are formatted. In order to perform a continuous write, the trailer bytes should be a write command and the address of the next line, in this case line 2. Then again the data bytes are skipped over followed by two more trailer bytes containing the write command and line address (now line 3) of the next line and so on until the end of the buffer. Using this method, the MLCD_WriteLines command can be used to write any number of lines starting at any line.

If using the LS032B7DD01 panel, the symbol MLCD_EXTADDR should be defined.

```
//////////////////////////////////////////////////////////////////////
//
// Memory LCD driver  (MLCD Driver.c)
//
// Provides basic interface with any of the memory LCD modules produced by
// Sharp.
//
//
//////////////////////////////////////////////////////////////////////

/*********************************************************************
 * SHARP MICROELECTRONICS OF THE AMERICAS MAKES NO REPRESENTATION
 * OR WARRANTIES WITH RESPECT TO THE PERFORMANCE OF THIS SOFTWARE,
 * AND SPECIFICALLY DISCLAIMS ANY RESPONSIBILITY FOR ANY DAMAGES,
 * SPECIAL OR CONSEQUENTIAL, CONNECTED WITH THE USE OF THIS SOFTWARE.
 *
 * SHARP MICROELECTRONICS OF THE AMERICAS PROVIDES THIS SOFTWARE SOLELY
 * FOR THE PURPOSE OF AN EXAMPLE INCORPORATING THE USE OF A
 * SHARP LCD PRODUCT. USE OF THIS SOURCE FILE IMPLIES ACCEPTANCE OF
 * THESE CONDITIONS.
 *
 * COPYRIGHT (C) 2016 SHARP MICROELECTRONICS OF THE AMERICAS, INC.
 *     CAMAS, WA
 *********************************************************************/


#include <stdint.h>
#include <stdbool.h>
#include “MLCD Driver.h”
```

```
// Global definitions
#ifndef HIGH
#define HIGH 1
#endif
#ifndef LOW
#define LOW  0
#endif


#define MLCD_WR      0x80
#define MLCD_VCOM    0x40
#define MLCD_CLR     0x20

//Uncomment the line below if using the LS032B7DD01
//#define MLCD_EXTADDR

#define MLCD_DAT         uint_8


// Global variables


// User supplied functions
// The user must supply these routines which are implementation dependent
extern void MLCD_Init_SPI(void);    //initialize the SPI port
extern void MLCD_Init_VCOM(void);   //set up hardware VCOM if used
extern void MLCD_Xfer_Enable(uint8_t *addr, uint16_t cnt);   //start a data transfer
                                                             //to the MLCD


// Local functions


// Local variables
static const uint8_t clear[2] = {MLCD_CLR,0};   //clear screen command




void MLCD_Init(void)
```

```
{
   // Initalize the SPI port that talks to the MLCD (user supplied).
   MLCD_Init_SPI();

   // Start up the external COM signal (user supplied).
   MLCD_Init_VCOM();

   // Blank the screen
   MLCD_BlankScreen();

   return;
}




// Clear (blank) the screen by sending “clear screen” command
void MLCD_BlankScreen(void)
{
   // Send MLCD_CLR_SCREEN command
   MLCD_Xfer_Enable((uint8_t *) (&clear[0]), 2);   //start the xfer

   return;
}




// Initialize the frame buffer
// The frame buffer holds a whole frame of data plus memory LCD overhead bytes
// such that the whole frame (or any part) can be sent to the panel in one
// dma transfer.
void MLCD_InitFrameBuf(uint8_t *bufptr)
{
   // Format the buffer with line numbers and dummy data
#ifdef MLCD_EXTADDR
   uint16_t line_num, cnt;
   union
   {
      uint16_t val;
      struct
      {
         uint8_t LB;
         uint8_t HB;
      } comp;
   } ext_addr;


   ext_addr.val = 0x40;

   for(cnt = 1; cnt <= MLCD_VERT_RES; cnt++)
   {
      *bufptr++ = ext_addr.comp.LB | MLCD_WR;   //b0, b1 of addr + write cmd
      *bufptr++ = ext_addr.comp.HB;             //b2-b9 of addr
      bufptr += BYTES_LINE;
      ext_addr.val += 0x40;                     //inc line addr
   }
#else
   uint16_t line_num;


   for(line_num = 1; line_num <= MLCD_VERT_RES; line_num++)
   {
      *bufptr++ = MLCD_WR;            //first byte is write command
      *bufptr++ = line_num;           //line number
      bufptr += MLCD_BYTES_LINE;      //skip to end of data
   }
#endif

   return;
}




// Write lines from to the panel
// This routine assumes that “buffer” is formatted with the overhead bytes and
// starts with line 1.
//   linnum -  starting line number (first line is 1)
//   numlins - number of lines to write
void MLCD_WriteLines(uint8_t *buffer, uint16_t linnum, uint16_t numlins)
{
   // Sanity checks
   if(linnum > MLCD_VERT_RES) return;
   if((linnum + linecnt) > (MLCD_VERT_RES + 1)) linecnt = MLCD_VERT_RES - linnum + 1;

   // Start the transfer
   MLCD_Xfer_Enable(buffer + ((MLCD_BYTES_LINE + 2) * (linnum - 1)),   //starting addr
               (numlins * (MLCD_BYTES_LINE + 2)) + 2);   //byte count

   return;
}
```

```
// Set/Clear numlines of buffer starting with line linnum
// This function assumes that a line of pixel data is a multiple of 16 bits (for
// speed reasons) and of course the buffer is in RAM.
// data - 0: set all bits to 0; 1: set all bits to 1
void MLCD_ClearLines(uint8_t *buffer, uint16_t linnum, uint16_t linecnt, bool data)
{
   uint16_t *fbptr;
   uint16_t i, j;
   uint16_t mod_dat;


   // Sanity checks
   if(linnum > MLCD_VERT_RES) return;
   if((linnum + linecnt) > (MLCD_VERT_RES + 1)) linecnt = MLCD_VERT_RES - linnum + 1;

   if(data)
      mod_dat = 0xffff;   //assume dat = 0xff
   else
      mod_dat = 0;      //data = 0

   // Get pointer to first line
   fbptr = (uint16_t *)(buffer + ((linnum -1) * (MLCD_BYTES_LINE + 2)) + 2);
   for(i = 0; i < linecnt; i++) {
      for(j = 0; j < MLCD_BYTES_LINE / 2; j++) {
         *fbptr++ = mod_dat;
      }
      fbptr++;   //skip over overhead bytes
   }

   return;
}


//////////////////////////////////////////////////////////////////////
//
// Memory LCD driver Header File  (MLCD Driver.h)
//
// Provides basic interface with any of the memory LCD modules produced by
// Sharp.
//
//
//////////////////////////////////////////////////////////////////////

/*********************************************************************
 * SHARP MICROELECTRONICS OF THE AMERICAS MAKES NO REPRESENTATION
 * OR WARRANTIES WITH RESPECT TO THE PERFORMANCE OF THIS SOFTWARE,
 * AND SPECIFICALLY DISCLAIMS ANY RESPONSIBILITY FOR ANY DAMAGES,
 * SPECIAL OR CONSEQUENTIAL, CONNECTED WITH THE USE OF THIS SOFTWARE.
 *
 * SHARP MICROELECTRONICS OF THE AMERICAS PROVIDES THIS SOFTWARE SOLELY
 * FOR THE PURPOSE OF AN EXAMPLE INCORPORATING THE USE OF A
 * SHARP LCD PRODUCT. USE OF THIS SOURCE FILE IMPLIES ACCEPTANCE OF
 * THESE CONDITIONS.
 *
 * COPYRIGHT (C) 2016 SHARP MICROELECTRONICS OF THE AMERICAS, INC.
 *     CAMAS, WA
 *********************************************************************/
#include <stdint.h>
#include <stdbool.h>

#ifndef _MLCD_DRIVER_
#define _MLCD_DRIVER_
```

```
// If using the LS032B7DD01 panel the line below should be uncommented
//#define   MLCD_EXTADDR


// Define Memory LCD panel resolution
// Note: MLCD_HORZ_RES must be defined somewhere and included here
#define MLCD_BYTES_LINE  MLCD_HORZ_RES / 8


// Function definitions
void MLCD_Init(void);
void MLCD_BlankScreen(void);
void MLCD_InitFrameBuf(uint8_t *);
void MLCD_WriteLines(uint8_t *, uint16_t, uint16_t);
void MLCD_ClearLines(uint8_t *, uint16_t, uint16_t, bool);


#endif
```

Sample program using an ST Micro STM32F072 processor.

```
//*****************************************************************
//
// File Name: MLCD User Functions.c
// Author:    Ken Green
// Date       10/29/15
//
// Collection of routines that depend on which processor is used and the design
// of the system.
//
//*****************************************************************


/**********************************************************************
 * SHARP MICROELECTRONICS OF THE AMERICAS MAKES NO REPRESENTATION
 * OR WARRANTIES WITH RESPECT TO THE PERFORMANCE OF THIS SOFTWARE,
 * AND SPECIFICALLY DISCLAIMS ANY RESPONSIBILITY FOR ANY DAMAGES,
 * SPECIAL OR CONSEQUENTIAL, CONNECTED WITH THE USE OF THIS SOFTWARE.
 *
 * SHARP MICROELECTRONICS OF THE AMERICAS PROVIDES THIS SOFTWARE SOLELY
 * FOR THE PURPOSE OF AN EXAMPLE INCORPORATING THE USE OF A
 * SHARP LCD PRODUCT. USE OF THIS SOURCE FILE IMPLIES ACCEPTANCE OF
 * THESE CONDITIONS.
 *
 * COPYRIGHT (C) 2016 SHARP MICROELECTRONICS OF THE AMERICAS, INC.
 *     CAMAS, WA
 ***********************************************************************/
#include “MLCD User Functions.h”
#include “MLCD Driver.h”
#include “stm32f072xb.h”

// Note: Both the stm32f0xx_hal_spi.c and stm32f0xx_hal_dma.c routines were
//       streamlined for this application. These routines may be obtained
//       by contacting your local Sharp SMA office.
#include “stm32f072_sma_spi.h”
#include “stm32f0xx_sma_dma.h”



static SPI_HandleTypeDef spi_blk;
static SPI_IBLK_TypeDef  spi_iblk;


void MLCD_Init_SPI(void)
{
   // Set up the SPI blocks so the SPI can be configured
   spi_iblk.SPI_CR1_INIT  = SPI_SPEED;    //set master mode, LSB first, SPI speed
   spi_iblk.SPI_PORT_PINS = 0x4075;       //port pin mapping
   spi_iblk.SPI_ALTFUNC   = 0x0000;       //alternate functions
   spi_iblk.SPI_GPIO_EN   = 0x00020000;   //I/O port enables
   spi_iblk.SPI_PORT_SCK  = GPIOA;        //pointers to GPIO register set
   spi_iblk.SPI_PORT_MOSI = GPIOA;
   spi_iblk.SPI_PORT_MISO = 0;            //(not used)
   spi_iblk.SPI_PORT_SS   = GPIOA;

   spi_blk.Instance = SPI1;
   spi_blk.Init = &spi_iblk;              //ptr to Init Block
   SPI_Init(&spi_blk);

   return;
}





void MLCD_Init_VCOM(void)
{
   // Port pin PA6 is used to drive the EXTCOM pin on the MLCD
   // At this point the clock for the GPIO port used to drive the COM signal
   // should be enabled. Since it is the same port as the SPI, it has already
   // been enabled.
   RCC_TypeDef  *rccptr = RCC;


   // Enable clock to Timer 3
   rccptr->APB1ENR  |= RCC_APB1ENR_TIM3EN;
   rccptr->APB1RSTR |= RCC_APB1RSTR_TIM3RST;   //reset timer 3
   rccptr->APB1RSTR &= ~(RCC_APB1RSTR_TIM3RST);

   // Configure the pin as timer 3 counter 1 output
   GPIOA->AFR[0] |= 0x01000000;   //pin PA6 - alternate function 1 (TIM3_CH1)
   GPIOA->MODER  |= 0x00002000;   //set PA6 to “alternate function”
   GPIOA->OSPEEDR &= ~(0x00003000);  //set pin speed

   // Configure timer 3 channel 1
   TIM3->CCMR1  = 0x0068;   //PWM mode 1; output; preload enable
   TIM3->CCER   = 0x0001;   //CC1 is output to pin

   TIM3->PSC    = 23 - 1;      //was 8
   TIM3->ARR    = 100000 - 1;  //was 100000
   TIM3->CCR1   = 3000;

   TIM3->CR1   |= 0x0001;   //start the counter

   return;
}
```

```
void MLCD_Xfer_Enable(uint8_t *addr, uint16_t cnt)
{
   // Set up the SPI block
   spi_blk.pTxBuffPtr = addr;   //starting addr
   spi_blk.TxXferSize = cnt;    //xfer size

   // Start the transfer
   SPI_SendData(&spi_blk);

   return;
}



//*****************************************************************
//
// File Name: MLCD User Functions.c
// Author:    Ken Green
// Date       10/29/15
//
// Collection of routines that depend on which processor is used and the design
// of the system.
//
//*****************************************************************

/**********************************************************************
 * SHARP MICROELECTRONICS OF THE AMERICAS MAKES NO REPRESENTATION
 * OR WARRANTIES WITH RESPECT TO THE PERFORMANCE OF THIS SOFTWARE,
 * AND SPECIFICALLY DISCLAIMS ANY RESPONSIBILITY FOR ANY DAMAGES,
 * SPECIAL OR CONSEQUENTIAL, CONNECTED WITH THE USE OF THIS SOFTWARE.
 *
 * SHARP MICROELECTRONICS OF THE AMERICAS PROVIDES THIS SOFTWARE SOLELY
 * FOR THE PURPOSE OF AN EXAMPLE INCORPORATING THE USE OF A
 * SHARP LCD PRODUCT. USE OF THIS SOURCE FILE IMPLIES ACCEPTANCE OF
 * THESE CONDITIONS.
 *
 * COPYRIGHT (C) 2016 SHARP MICROELECTRONICS OF THE AMERICAS, INC.
 *     CAMAS, WA
 *********************************************************************/

#include <stdint.h>
#include <stdbool.h>


#ifndef  _MLCD_USER_
#define  _MLCD_USER_



void MLCD_Init_SPI(void);
void MLCD_Init_VCOM(void);
void MLCD_Xfer_Enable(uint8_t *, uint16_t);


#endif
```

## SUMMARY

Sharp's Memory LCDs have a serial interface that makes them simple to program. The most challenging tasks for the programmer will be to ensure that VCOM is toggled periodically to maintain the lack of DC bias on the display; and that data is sent to the panel in the correct order.

It is the responsibility of the user to validate Sharp's technical materials and information for your particular application; as well as to be mindful of intellectual property issues related to their use. This includes, but is not limited to, our technical materials such as Application Notes, White Papers, Reference Designs, and verbal assistance. Please see our *Important Restrictions* page on http://www.sharpsma.com/ for more details regarding liability.

## NORTH AMERICA

SHARP Microelectronics of the Americas
5700 NW Pacific Rim Blvd.
Camas, WA 98607, U.S.A.

Phone: (1) 360-834-2500
Fax: (1) 360-834-8903

www.sharpsma.com

## TAIWAN

SHARP Electronic Components
(Taiwan) Corporation
8F-A, No. 16, Sec. 4, Nanking E. Rd.
Taipei, Taiwan, Republic of China

Phone: (886) 2-2577-7341
Fax: (886) 2-2577-7326/2-2577-7328

## CHINA

SHARP Microelectronics of China
(Shanghai) Co., Ltd.
28 Xin Jin Qiao Road King Tower 16F
Pudong Shanghai, 201206 P.R. China

Phone: (86) 21-5854-7710/21-5834-6056
Fax: (86) 21-5854-4340/21-5834-6057

**Head Office:**
No. 360, Bashen Road,
Xin Development Bldg. 22
Waigaoqiao Free Trade Zone Shanghai
200131 P.R. China

Email: smc@china.global.sharp.co.jp

## EUROPE

SHARP Devices Europe GmbH
Landsberger Straβe 398
Munich 81241, Germany

Phone: +49-89-5468-420
Fax: +49-89-5468-4250

www.sharpsde.com

## SINGAPORE

SHARP Electronics (Singapore) PTE., Ltd.
438A, Alexandra Road, #05-01/02
Alexandra Technopark,
Singapore 119967

Phone: (65) 271-3566
Fax: (65) 271-3855

## KOREA

SHARP Electronic Components
(Korea) Corporation
RM 501 Geosung B/D, 541
Dohwa-dong, Mapo-ku
Seoul 121-701, Korea

Phone: (82) 2-711-5813 ~ 8
Fax: (82) 2-711-5819

## JAPAN

SHARP Corporation
Electronic Components & Devices
22-22 Nagaike-cho, Abeno-Ku
Osaka 545-8522, Japan

Phone: (81) 6-6621-1221
Fax: (81) 6117-725300/6117-725301

www.sharp-world.com

## HONG KONG

SHARP-ROXY (Hong Kong) Ltd.
3rd Business Division,
17/F, Admiralty Centre, Tower 1
18 Harcourt Road, Hong Kong

Phone: (852) 28229311
Fax: (852) 28660779

www.sharp.com.hk

**Shenzhen Representative Office:**
Room 13B1, Tower C,
Electronics Science & Technology Building
Shen Nan Zhong Road
Shenzhen, P.R. China

Phone: (86) 755-3273731
Fax: (86) 755-3273735