



# Reflex™ Graphic Display Module 128x32

## DATASHEET



### PRODUCT FEATURES

#### Display Assembly:

- 128 Columns x 32 Rows
- Approximate size: 1.3 x 0.47 x 0.06 Inches
- Available in Multiple Color Schemes
- Low Profile, Compact Design
- Superior Brightness
- Excellent Optical Properties
- Viewing Cone Comparable to Paper
- Indefinite “No Power” Memory

#### Driver ASIC:

- Serial Command and Data Interface
- Integrated Display Controller
- Integrated DC/DC Charge Pump

### PRODUCT DESCRIPTION

The Reflex™ display delivers maximum image information content flexibility in an extremely compact form factor. It is one of the most compact and cost efficient bi-stable graphic displays available today.

All Kent Displays Reflex™ products take advantage of the technology's unique “No Power” attribute without compromising superior optical performance even in direct sunlight.

The Chip-on-Glass (COG) driver IC also includes an LCD controller.

### TYPICAL APPLICATIONS

- Battery Powered Devices
- USB Powered Devices
- Inventory Tracking Displays
- Remote Control Displays





**CONTENTS**

<b>1</b>	<b>Overview .....</b>	<b>5</b>
<b>2</b>	<b>Block Diagram.....</b>	<b>5</b>
<b>3</b>	<b>Electrical Interface.....</b>	<b>6</b>
3.1	Display Assembly Interconnect .....	6
3.2	Pin Summary .....	6
3.3	Reference Schematic .....	8
<b>4</b>	<b>Operating Principles .....</b>	<b>9</b>
4.1	Bistability .....	9
4.2	Serial Interface .....	9
4.3	Image Data .....	9
<b>5</b>	<b>Specifications .....</b>	<b>11</b>
5.1	General.....	11
5.2	Absolute Maximum Ratings (ASIC).....	11
5.3	Electrical .....	11
5.3.1	Power Profile .....	12
5.3.2	Update Cycle Temperature Performance.....	13
5.4	Optical.....	14
5.5	Timing .....	15
5.5.1	Serial Interface.....	15
5.6	Mechanical.....	17
<b>6</b>	<b>Sample Code .....</b>	<b>20</b>
6.1	User-Defined Code.....	20
6.1.1	Type Definitions .....	20
6.1.2	Timing .....	20
6.1.3	BUSY Signal Monitoring.....	20
6.1.4	Command and Data Transmission .....	20
6.1.5	Interface Signals.....	20
6.1.6	Display Reset.....	21
6.1.7	Temperature Sensing.....	21
6.2	Standard Code.....	21
6.2.1	defines.h .....	21
6.2.2	display.c.....	23
6.2.3	system.c.....	30
6.2.4	LoadData.c.....	32
6.2.5	SampleImage.c .....	35
6.3	Main Code for User Program.....	37
6.4	Code Copyright Notice.....	37
<b>7</b>	<b>Ordering Information .....</b>	<b>39</b>
<b>8</b>	<b>Product Handling Precautions.....</b>	<b>40</b>



**FIGURES**

1: Block Diagram ..... 5  
2: Reference Schematic ..... 8  
3: Basic Serial Interface Timing..... 9  
4: Image Data to Pixel Mapping ..... 10  
5: Update Power vs. Time ..... 12  
6: Estimated Total Update Time ..... 13  
7: Spectral Reflectance Characteristics..... 14  
8: Contrast Ratio Polar Representation ..... 14  
9: Serial Timing ..... 15  
10: Assembly Drawing ..... 19  
11: Land Pattern Detail for 128x32 Pixel Display..... 19  
12: Display Assembly ..... 39

**TABLES**

1: Pin Summary ..... 6  
2: General Specifications (Display Assembly) ..... 11  
3: Absolute Maximum Ratings..... 11  
4: Electrical ..... 11  
5: Serial Interface Timing Parameter Values..... 15  
6: Ordering Information..... 39



# 1 Overview

The Reflex Graphic Display Module 128x32 is a passive matrix display ideally suited for battery/low voltage powered portable devices and display applications that require superior optical performance including wide viewing angle and sunlight readability. The display is a reflective cholesteric liquid crystal display (ChLCD) that takes full advantage of the technology's unique "No Power" image retention attribute. The embedded driver IC contains internal DC/DC conversion circuitry which requires only a small number of external capacitors for generating the LCD drive voltages. Image data and commands are transferred to the display from the host using a serial interface. The host system controls the image update through a simple sequence of commands.

# 2 Block Diagram

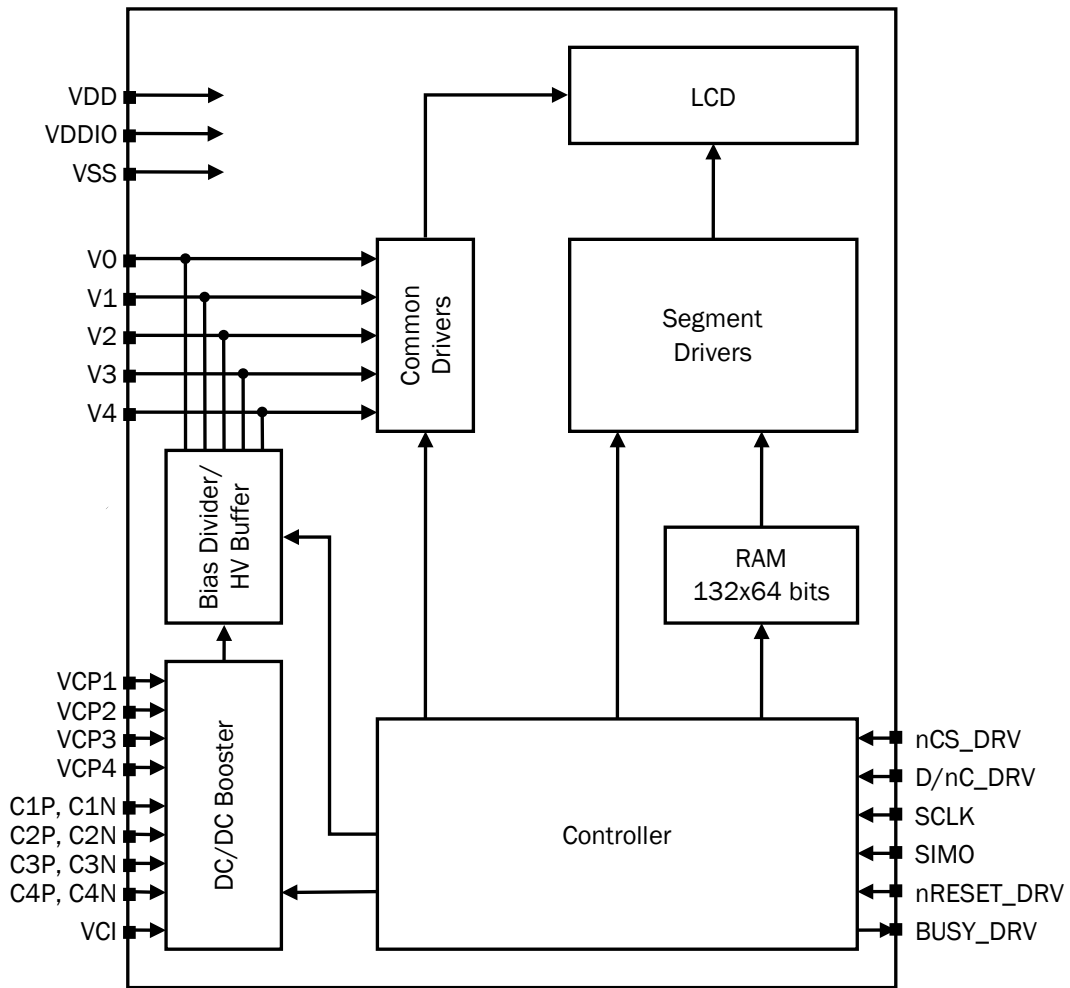


FIGURE 1: BLOCK DIAGRAM



## 3 Electrical Interface

### 3.1 Display Assembly Interconnect

The GDM interfaces to the host electronics through a 30-contact FPC (Flexible Printed Circuit). Refer to Figure 10 in Section 5.6 for FPC dimensions.

### 3.2 Pin Summary

**TABLE 1: PIN SUMMARY**

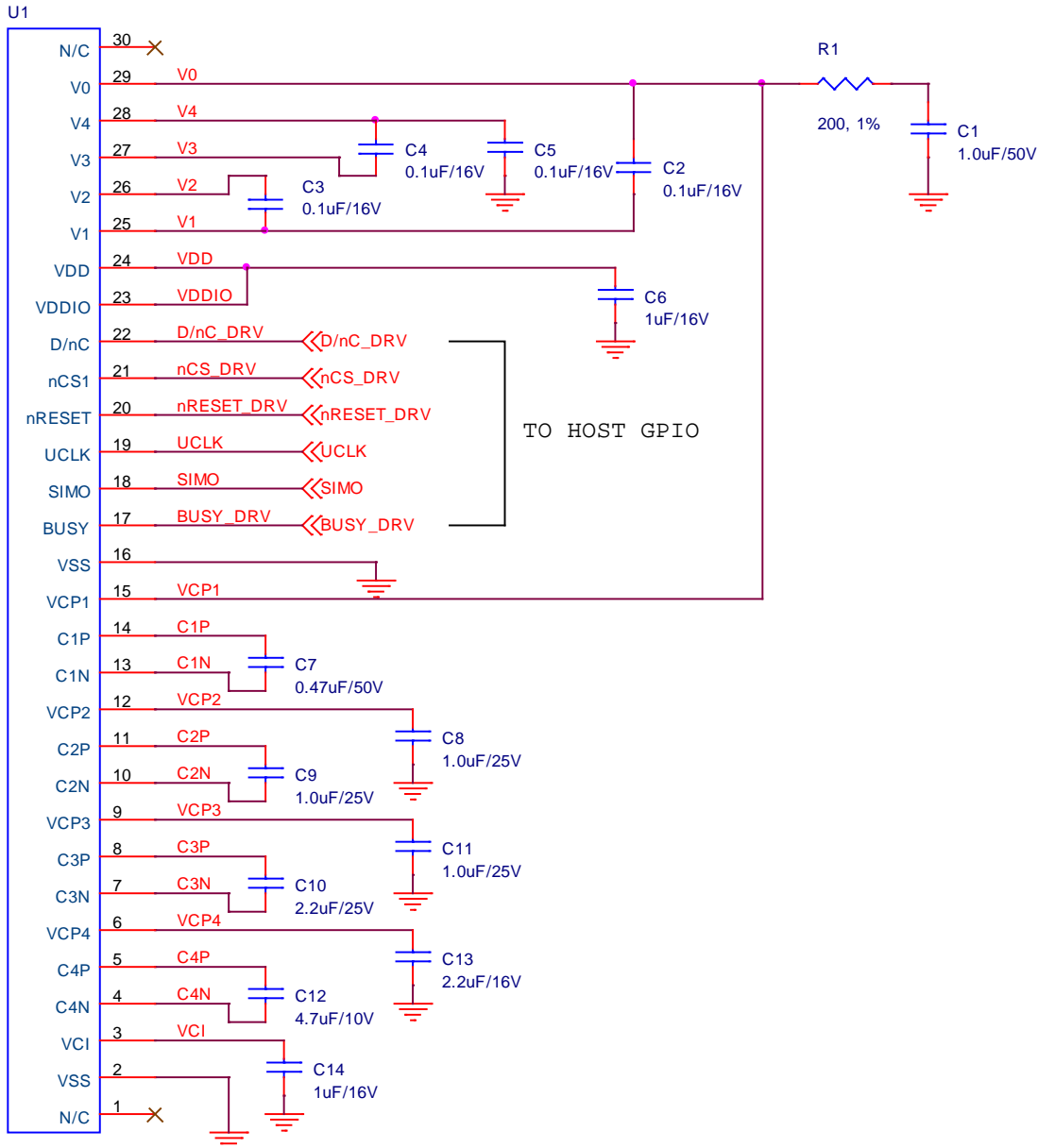
PIN	NAME	TYPE <sup>1</sup>	DESCRIPTION
1	NC	NC	No connection.
2	VSS	Supply	Ground.
3	VCI	Supply	Power Input for DC/DC converter. Connect to VDD.
4,5	C4N, C4P	DC/DC	DC/DC flying capacitor terminals. Connect a capacitor between pins.
6	VCP4	DC/DC	DC/DC intermediate output voltage. Connect with a capacitor to VSS.
7,8	C3N, C3P	DC/DC	DC/DC flying capacitor terminals. Connect a capacitor between pins.
9	VCP3	DC/DC	DC/DC intermediate output voltage. Connect with a capacitor to VSS.
10,11	C2N, C2P	DC/DC	DC/DC flying capacitor terminals. Connect a capacitor between pins.
12	VCP2	DC/DC	DC/DC intermediate output voltage. Connect with a capacitor to VSS.
13,14	C1N, C1P	DC/DC	DC/DC flying capacitor terminals. Connect a capacitor between pins.
15	VCP1	DC/DC	DC/DC output. Connect with a capacitor to VSS. Connect to V0 with a resistor.
16	VSS	Supply	Ground.
17	BUSY_DRV	Output	A high level indicates the driver is busy driving a waveform.
18	SIMO	Input	Serial data input.
19	SCLK	Input	Serial clock input.
20	nRESET_DRV	Input	Reset input. Set to VSS for $\geq 20 \mu\text{s}$ to initialize chip.
21	nCS_DRV	Input	Chip select input. Set to VSS to select the chip for serial communication.
22	D/nC_DRV	Input	Data/Command control pin. Set to VDD for data input and VSS for command input.
23	VDDIO	Supply	Interface logic supply. Tie to VDD.
24	VDD	Supply	Power supply.
25	V1	Power	Panel driving voltage. Connect with a capacitor to V2. Connect with a capacitor to V0.
26	V2	Power	Panel driving voltage. Connect with a capacitor to V1.
27	V3	Power	Panel driving voltage. Connect with a capacitor to V4.
28	V4	Power	Panel driving voltage. Connect with a capacitor to V3. Connect with a capacitor to VSS
29	V0	Power	Panel driving voltage. Connect to VCP1. Connect with a capacitor to VSS.
30	NC	NC	No Connection.

<sup>1</sup>Type:



- Supply - provide power to the IC
- Input - logic input
- Output - logic output
- DC/DC - used by DC/DC charge pump to generate LCD drive voltage
- Power - provide display drive voltages
- NC - no connection

### 3.3 Reference Schematic



\*Note: Depending upon the source impedance of the supply, additional capacitance may be required to stabilize VDD when running the internal DC/DC converter.

FIGURE 2: REFERENCE SCHEMATIC



## 4 Operating Principles

### 4.1 Bistability

The unique bistable property of Reflex™ display products means that an image placed on the display will remain indefinitely without the need for refreshing. This results in a different paradigm for managing image content from a traditional TN or STN type display. In particular, image data sent to the display controller RAM does not immediately appear on the display screen. The image data only appears on the display screen after commanding the display controller to perform an update.

### 4.2 Serial Interface

The display driver functions similar to an SPI slave device (see Figure 3). The host system (master) selects the display module for communication using the nCS\_DRV line and provides the clock signal (SCLK) used to clock in data. A byte transfer to the display begins with a high-to-low transition on nCS\_DRV and ends with a low-to-high transition on nCS\_DRV. New data (command or image data) for the display driver are placed on SIMO on falling edges of SCLK, and the display controller latches the data on the rising edge of SCLK. Transmission of each byte begins with the most significant bit (D7) and ends with the least significant bit (D0). The D/nC\_DRV signal is sampled with D0. The byte (D7 to D0) is treated as a command if D/nC\_DRV is sampled low and as image data if D/nC\_DRV is sampled high.

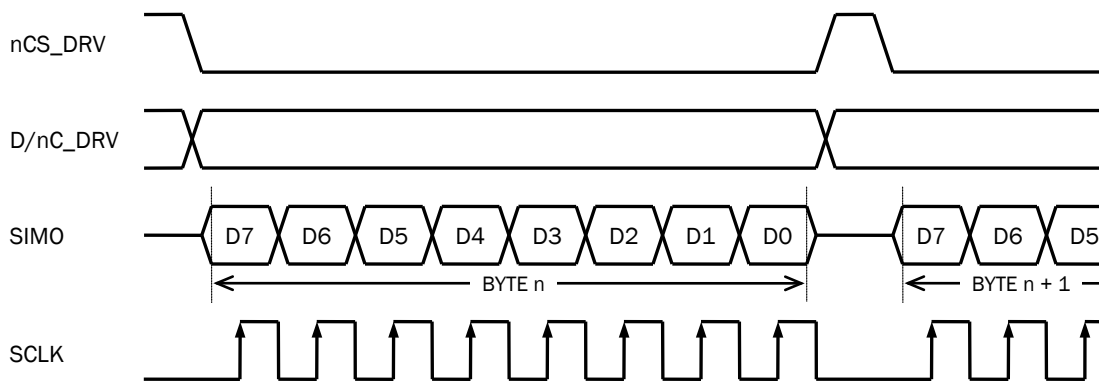


FIGURE 3: BASIC SERIAL INTERFACE TIMING

### 4.3 Image Data

The driver data space includes  $132 \times 64/8 = 1056$  bytes of memory divided into eight 132-byte pages. A single byte of display RAM defines the state of eight pixels placed vertically. Thus a page of display RAM defines an area of 132 columns by eight rows. The 128x32 display includes a full 512-byte image area surrounded by an active frame. The frame may be driven bright or dark. The 128x32 image and border are mapped into display RAM as shown in Figure 4.



PAGE	BIT	COLUMN ADDRESS												
		0	1	2	3	4	...	127	128	129	130	131		
0	0-7	DON'T CARE				...	DON'T CARE							
	1	DON'T CARE				...	DON'T CARE							
2	7	B	B	B	B	...	B	B	B	B	DON'T CARE			
	0	B	BYTE 0	BYTE 1	BYTE 2	...	BYTE 125	BYTE 126	BYTE 127	B				
	1	B												
	2	B												
	3	B												
	4	B												
	5	B												
	6	B												
7	B													
3	0	B	BYTE 128	BYTE 129	BYTE 130	...	BYTE 253	BYTE 254	BYTE 255	B	DON'T CARE			
	1	B												
	2	B												
	3	B												
	4	B												
	5	B												
	6	B												
	7	B												
4	0	B	BYTE 256	BYTE 257	BYTE 258	...	BYTE 381	BYTE 382	BYTE 383	B	DON'T CARE			
	1	B												
	2	B												
	3	B												
	4	B												
	5	B												
	6	B												
	7	B												
5	0	B	BYTE 384	BYTE 385	BYTE 386	...	BYTE 509	BYTE 510	BYTE 511	B	DON'T CARE			
	1	B												
	2	B												
	3	B												
	4	B												
	5	B												
	6	B												
	7	B												
6	0	B	B	B	B	...	B	B	B	B				
	1-7	DON'T CARE				...	DON'T CARE							
7	0-7	DON'T CARE				...	DON'T CARE							

**B** border  
**0** dark  
**1** bright

**FIGURE 4: IMAGE DATA TO PIXEL MAPPING**

The sample code in Section 6 illustrates how 512 bytes of image data ordered as in Figure 4 may be sent to the driver for display.



## 5 Specifications

### 5.1 General

**TABLE 2: GENERAL SPECIFICATIONS (DISPLAY ASSEMBLY)**

PARAMETER	DESCRIPTION
Display Type	Cholesteric Reflective LCD
Format	128 Columns x 32 Rows
Resolution	118 dots per inch (0.215mm pixel pitch, horizontal & vertical)
Viewing Area	1.08in x 0.27in (27.5mm x 6.9mm)
Display Assembly Weight	0.069 oz (2.15 grams)
Operating Temperature Range	0°C to +50°C
Storage Temperature Range	-30°C to +80°C
Display Update Time	1.7 seconds at 25 °C

### 5.2 Absolute Maximum Ratings (ASIC)

**TABLE 3: ABSOLUTE MAXIMUM RATINGS**

PARAMETER	SYMBOL	RATING	UNITS
IC Logic Supply	VDD	-0.3 to +3.6	V
Interface Logic Supply	VDDIO	-0.3 to min (VDD+0.5, +3.6)	V
DC/DC Supply	VCI	-0.3 to +3.6	V
Logic Input Voltage	Vin	-0.3 to (VDDIO + 0.3)	V

### 5.3 Electrical

**TABLE 4: ELECTRICAL**

PARAMETER	SYMBOL	MIN.	TYP.	MAX.	UNITS
IC Logic Supply	VDD	+2.4	+2.8	+3.5	V
Interface Logic Supply	VDDIO	+1.6	-	VDD	V
DC/DC Supply	VCL	+VDD	-	+3.5	V
Input Voltage	High	VIH	0.8 VDDIO	-	VDDIO



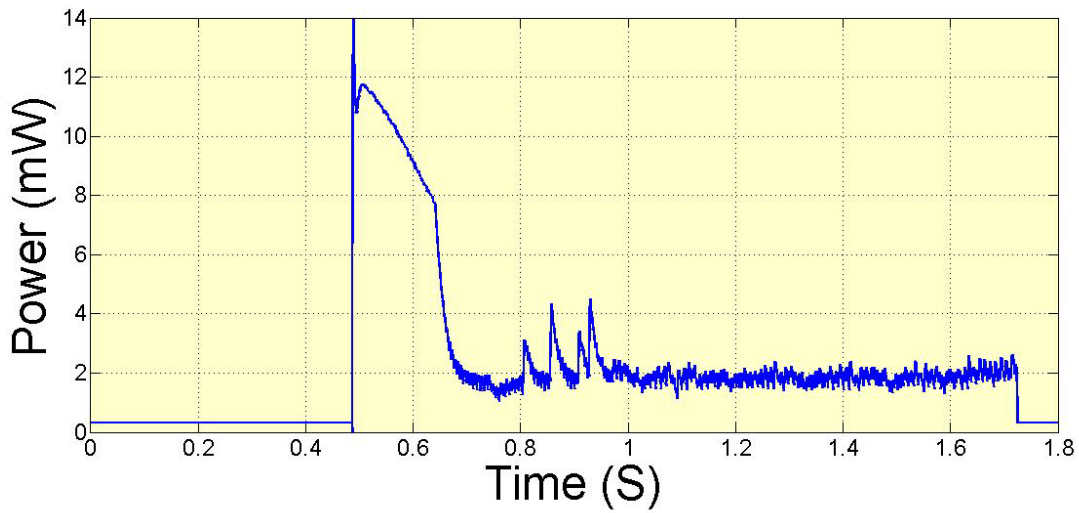
	Low	VIL	VSS	-	0.2 V <sub>DDIO</sub>	V
Output Voltage	High	VOH	0.9 V <sub>DDIO</sub>	-	V <sub>DDIO</sub>	V
	Low	VOL	VSS	-	0.1 V <sub>DDIO</sub>	V
Sleep Mode Current <sup>1</sup>		ISLP	-	+1	+5	μA

Conditions: T<sub>A</sub> = 25°C

<sup>1</sup> ISLP is the sum of VDD, VDDIO, and VCI currents. Power is measured with VDD = VDDIO = VCI = 3.5V.

\* Specifications are subject to change without prior notice.

### 5.3.1 Power Profile



Note: Power measurements based on VDD = 3.3V and T<sub>A</sub> = 25°C. The total energy for the update is 3.689mJ.

FIGURE 5: UPDATE POWER VS. TIME



### 5.3.2 Update Cycle Temperature Performance

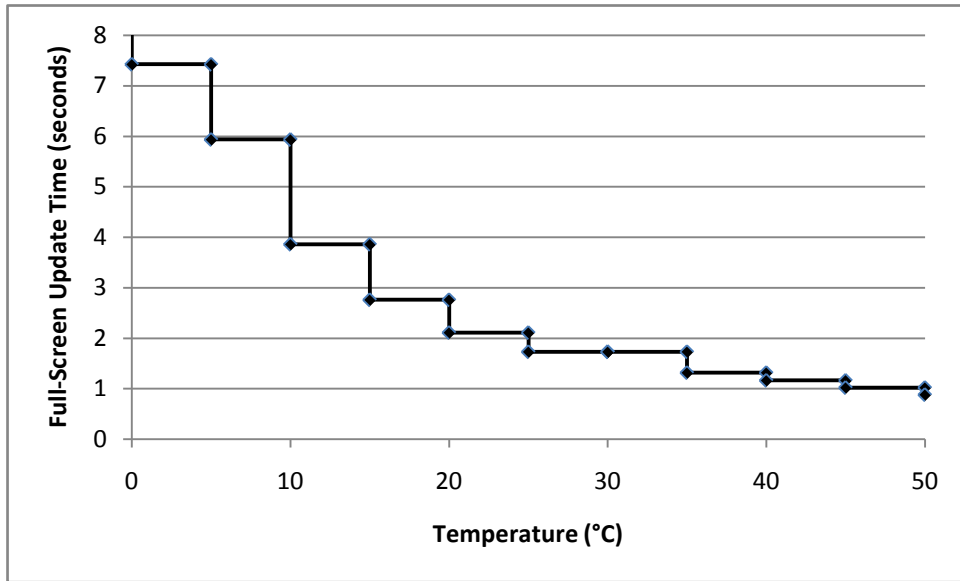
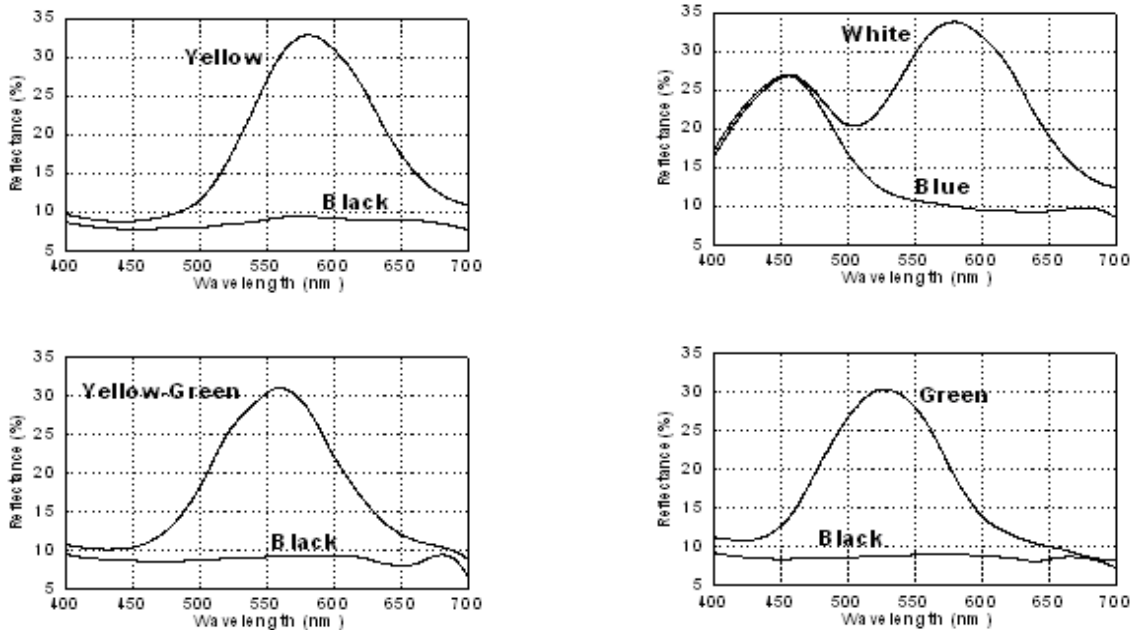


FIGURE 6: ESTIMATED TOTAL UPDATE TIME

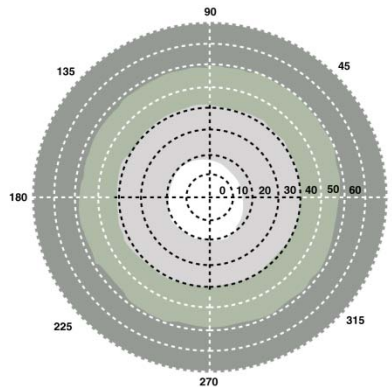


### 5.4 Optical



**FIGURE 7: SPECTRAL REFLECTANCE CHARACTERISTICS**

The graph in Figure 7 provides the spectral reflectance characteristics for a given display pixel when switched to either of the two possible stable states: reflective planar or transparent focal conic. The top line in the graph outlines the reflective characteristic of the planar state. The bottom line outlines the reflective characteristic of the focal conic state.



**FIGURE 8: CONTRAST RATIO POLAR REPRESENTATION**

As illustrated in Figure 8 for a standard Reflex™ display, all Kent Displays' Reflex™ display products have a 360 degree viewing cone. When measured normal to the plane of the display, the monochromatic contrast ratio is as high as 25:1 with a peak reflectivity approaching 35% of the incident light. The contrast ratio reduces as the viewing angle approaches the plane of the display but is still excellent at 11:1. Since no polarizers are used, display contrast reduces uniformly in all azimuthal directions when the viewing angle is increased.

## 5.5 Timing

### 5.5.1 Serial Interface

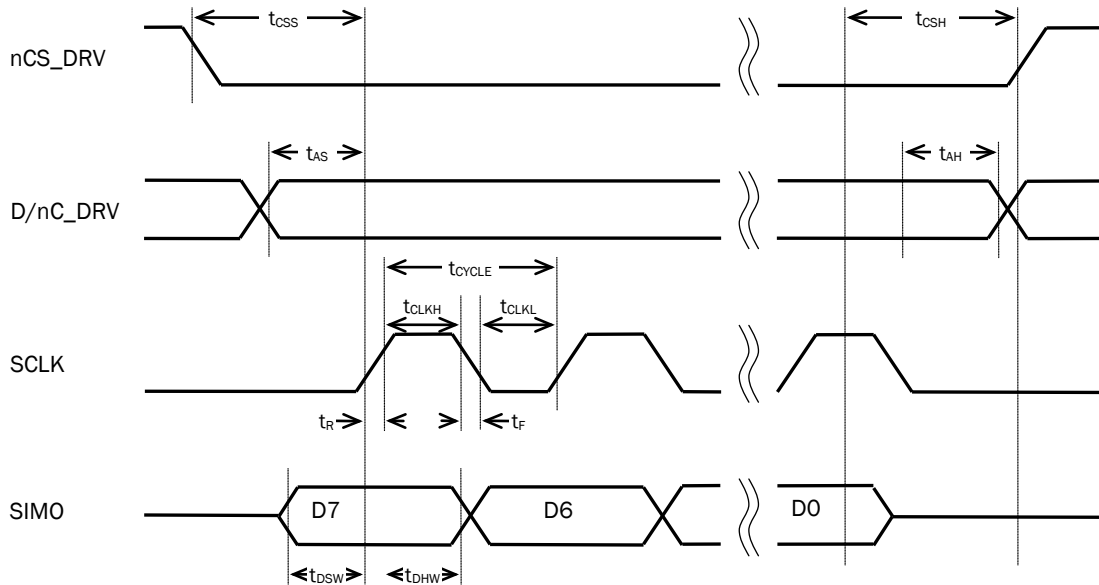


FIGURE 9: SERIAL TIMING

TABLE 5: SERIAL INTERFACE TIMING PARAMETER VALUES

SIGNAL	ITEM	SYMBOL	MIN.	TYP.	MAX.	UNITS
SCLK	Clock Cycle Time	$t_{\text{CYCLE}}$	60	-	-	ns
	Clock High Time	$t_{\text{CLKH}}$	30	-	-	
	Clock Low Time	$t_{\text{CLKL}}$	30	-	-	
	Rise Time	$t_{\text{R}}$	-	-	10	
	Fall Time	$t_{\text{F}}$	-	-	10	
D/nC_DRV	Address Setup Time	$t_{\text{AS}}$	10	-	-	ns
	Address Hold Time	$t_{\text{AH}}$	20	-	-	
SIMO	Write Data Setup Time	$t_{\text{DSW}}$	30	-	-	ns
	Write Data Hold Time	$t_{\text{DHW}}$	30	-	-	
nCS_DRV	Chip Select Setup Time	$t_{\text{CSS}}$	30	-	-	ns
	Chip Select Hold Time	$t_{\text{CSH}}$	30	-	-	

Conditions:  $T_A = -35$  to  $85^\circ\text{C}$ ,  $V_{\text{DD}} = V_{\text{CI}} = V_{\text{DDIO}} = 2.4\text{V}$  to  $3.5\text{V}$



5.6 Mechanical

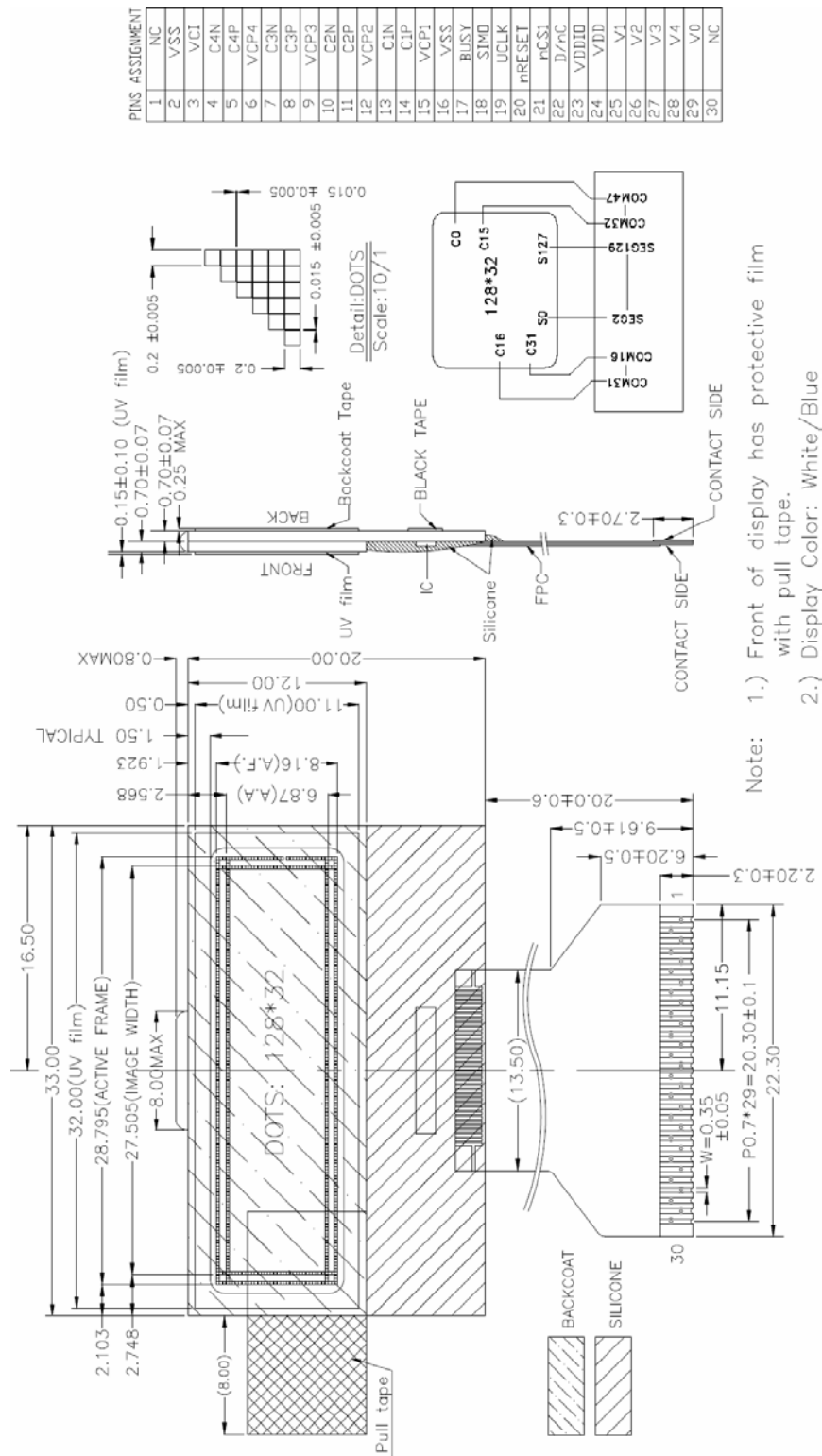
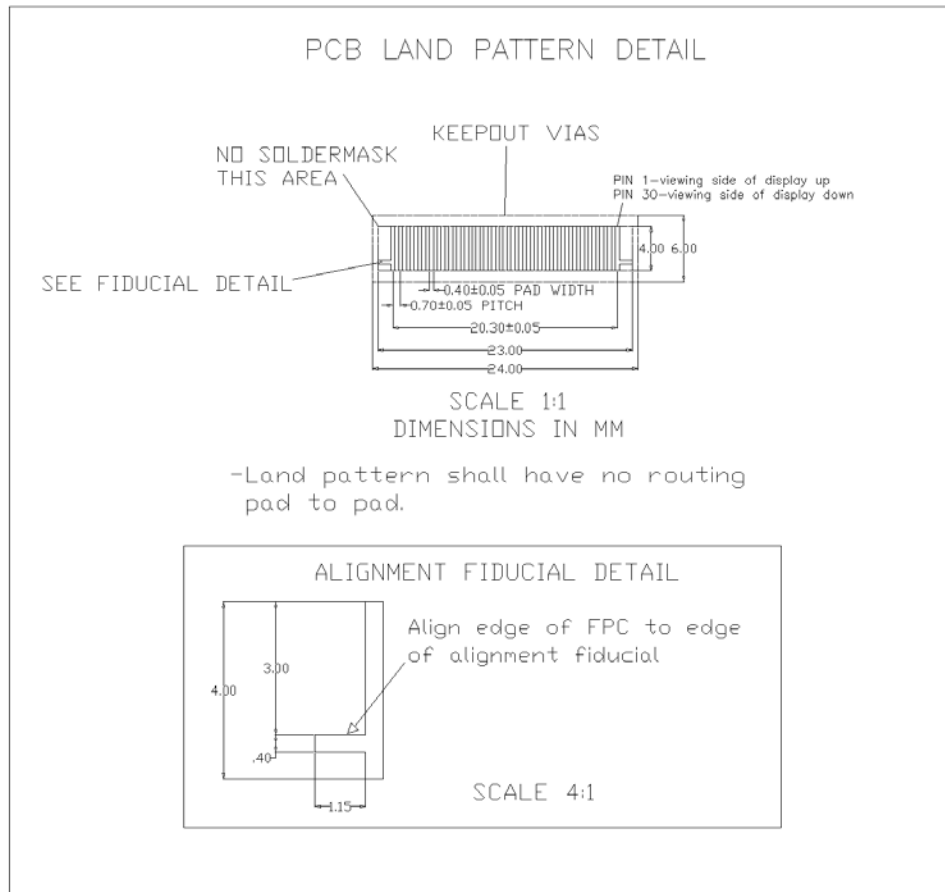


FIGURE 10: ASSEMBLY DRAWING



**FIGURE 11: LAND PATTERN DETAIL FOR 128X32 PIXEL DISPLAY**



## 6 Sample Code

Source code is provided in “C” for implementing a full-screen display update. A small number of user-defined functions and parameters specific to the host platform must be implemented independently.

### 6.1 User-Defined Code

#### 6.1.1 Type Definitions

The source code uses four data types: `int8_t`, `int16_t`, `uint8_t`, and `uint16_t`. These types are for signed and unsigned integers of the indicated number of bits. Compiler packages conforming to the C99 standard (ISO/IEC 9899:1999) define these types in the include file `stdint.h`. For compiler packages which do not include these definitions, the example definition below (which may vary by compiler) defines these types:

```
typedef signed char int8_t;           // signed 8-bit data type
typedef signed int  int16_t;          // signed 16-bit data type
typedef unsigned char uint8_t;        // unsigned 8-bit data type
typedef unsigned int uint16_t;        // unsigned 16-bit data type
```

#### 6.1.2 Timing

A function for implementing variable length delays is also required. An example function declaration, assuming the delay duration is specified as a number of ticks of a timer resource, is as follows:

```
void Delay(uint16_t ticks);
```

To accommodate differences in timer ticks among various platforms, the sample code uses the following definitions to convert milliseconds to timer ticks at compile time:

```
#define TICKS_PER_MS      (???)           // Platform-specific ticks per millisecond.
#define MS2TCK(msec)      (TICKS_PER_MS * (msec))
```

#### 6.1.3 BUSY Signal Monitoring

A function to monitor the `BUSY_DRV` signal from the driver is declared as follows:

```
void WaitNonBusy(void);
```

The implementation of this function must stall program execution for as long as the `BUSY_DRV` signal from the display controller remains high.

#### 6.1.4 Command and Data Transmission

Commands and data are transmitted serially to the display according to the serial protocol of Figure 9 and subject to the timing constraints of Table 5. The function, `PutCharSPI()`, must be implemented for sending commands and data. A reference implementation has been provided using general-purpose I/O for the case in which no additional delays are required in order to satisfy the timing constraints of Table 5.

#### 6.1.5 Interface Signals

The following functions must be implemented in order to set the interface signals to the designated values and/or to read them:

```
void Assert_nRESET_DRV(void);          // nRESET_DRV = 'L'
void Deassert_nRESET_DRV(void);        // nRESET_DRV = 'H'
uint8_t getBUSY_DRV(void);              // Read BUSY_DRV, returning 0 or 1
void SelectDisplay(void);               // nCS_DRV = 'L'
```



```
void DeselectDisplay(void);           // nCS_DRV = 'H'
uint8_t VDDEnabled(void);            // Read VDD status, 1 => on, 0 => off.
void VDDOn(void);                    // EN_VDD = 'H'
void VDDOff(void);                   // EN_VDD = 'L'
void Assert_D_DRV(void);             // D/nC_DRV = 'H'
void Assert_nC_DRV(void);           // D/nC_DRV = 'L'
void setSIMO(void);                  // SIMO = 'H'
void clearSIMO(void);                // SIMO = 'L'
void setSCLK(void);                  // SCLK = 'H'
void clearSCLK(void);                // SCLK = 'L'
void ZeroSPILines(void);             // SCLK, SIMO, D_nc_DRV = 'L'
void ZeroDriverLines(void);         // nCS_DRV, nRESET_DRV = 'L'
```

### 6.1.6 Display Reset

The display controller must be initialized after power is applied by pulsing nRESET\_DRV to VSS for  $\geq 20 \mu\text{s}$ . The sample code includes this reset.

### 6.1.7 Temperature Sensing

The drive parameters of the display are functions of temperature. A function to measure the temperature is declared as follows:

```
int16_t GetTemperature(void);
```

The implementation of this function must invoke a hardware resource to measure the temperature in degrees Celsius and return the temperature value as a 16-bit signed integer. If the conversion to Celsius demands too many platform resources, e.g. CPU cycles or code space, then the drive parameter table and associated look-up code may be transformed to use native platform units, such as A/D counts.

## 6.2 Standard Code

The following code should require minimal or no changes, once the user-defined code in `mydefines.h` is created.

### 6.2.1 defines.h

```
// Implementation-specific definitions are given in mydefines.h. This user-
// written file may include definitions and function prototypes for the user's
// program. It must define the integral data types, either by including
// <stdint.h> if it is available, or by explicitly defining the integral types
// as in the example below. This may vary by compiler:
//
// typedef signed char int8_t;        // 8-bit signed integer
// typedef signed int int16_t;       // 16-bit signed integer
// typedef unsigned char uint8_t;    // 8-bit unsigned integer
// typedef unsigned int uint16_t;    // 16-bit unsigned integer
//
// The file must also provide an appropriate definition for the following
// token to define the number of hardware timer ticks in a millisecond.
//
// TICKS_PER_MS
//
#include "mydefines.h"

// Prototypes for functions to abstract I/O and timer hardware. These control
```



```
// or read individual I/O lines at the display interface. The functions must
// be implemented in a user-written file.

// nRESET_DRV manipulation.
void Assert_nRESET_DRV(void);           // nRESET_DRV = 'L'
void Deassert_nRESET_DRV(void);        // nRESET_DRV = 'H'

// BUSY_DRV monitoring
uint8_t getBUSY_DRV(void);             // Read BUSY_DRV, returning 0 or 1

// Display selection/de-selection.
void SelectDisplay(void);              // nCS_DRV = 'L'
void DeselectDisplay(void);           // nCS_DRV = 'H'

// VDD Monitoring and control.
uint8_t VDDEnabled(void);             // Read VDD status, 1 => on, 0 => off.
void VDDOn(void);                     // EN_VDD = 'H'
void VDDOff(void);                   // EN_VDD = 'L'

// D_nC_DRV control.
void Assert_D_DRV(void);              // D/nC_DRV = 'H'
void Assert_nC_DRV(void);            // D/nC_DRV = 'L'

// SPI bit manipulations.
void setSIMO(void);                  // SIMO = 'H'
void clearSIMO(void);               // SIMO = 'L'
void setSCLK(void);                 // SCLK = 'H'
void clearSCLK(void);              // SCLK = 'L'

// Functions to zero outputs to the driver before powering it down.
void ZeroSPILines(void);             // SCLK, SIMO, D_nc_DRV = 'L'
void ZeroDriverLines(void);         // nCS_DRV, nRESET_DRV = 'L'

// Image characteristics (excluding frame).
#define N_ROWS          32
#define N_COLS          128
#define BYTES_PER_IMAGE (N_ROWS*N_COLS/8)
#define N_PAGES         (N_ROWS/8)
#define BYTES_PER_PAGE  (BYTES_PER_IMAGE/N_PAGES)

// Image remapping: A normal image's top is at the furthest edge from the
// flex cable. A remapped image's top is nearest the flex cable.
//
// REMAP_IMAGE      Image      Top of image.
// -----
// undefined        Normal     Away from flex.
// defined          Remapped    Near flex.
//
// #undef   REMAP_IMAGE      // Define or undefine this token for
//                          // desired image orientation.
//-----

// Conversion from ms to ticks for input to Delay().
```



```

#define MS2TCK(msec)      (TICKS_PER_MS * (msec))

// Function Prototypes

// display.c
void Display(uint8_t firstRow, uint8_t numRows);

// loaddata.c
void LoadData(uint16_t Index, uint16_t NumBytes, const uint8_t *pData);

// To use previous border when current one is unspecified.  For LoadBorder();
#define PREVIOUS_BORDER  0xFF

void LoadBorder(uint8_t border);

// system.c

// Tokens for selecting command byte or data byte.  For PutCharSPI().
#define CMD_MASK         0
#define DATA_MASK      1

void PutCharSPI(uint8_t data, uint8_t Data_nCmd);
void EnableVDD(void);
void DisableVDD(void);
void HardResetDisplay(void);
void Delay(uint16_t ticks);

// Temperature measurement.
int16_t GetTemperature(void);

```

## 6.2.2 display.c

```

#include "defines.h"

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// D e f i n i t i o n s
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Temperature compensation data element definition.
typedef struct
{
    int8_t Temp;
    uint8_t VAClearPW;
    uint8_t VAIdle;
    uint8_t AAClearPW;
    uint8_t AAIdle;
    uint8_t DrivePW;
    uint8_t Voltage;
} tcTableEntry;

#define TEMP_PTS (sizeof(tcTable)/sizeof(tcTable[0])) // # of table entries.
#define TEMP_MAX (tcTable[0].Temp+5) // Max update temp (C).

```



```

/////////////////////////////////////////////////////////////////
//
// I n t r a m o d u l a r   G l o b a l s
//
/////////////////////////////////////////////////////////////////

// Temp Comp Data
//
// White/Blue KLC-19.
//
const tcTableEntry tcTable[] =
{
  //
  // Temperatures are in degrees Celsius. All other values (in hex) are
  // encoded for the display driver. Within the comments, the values
  // are durations in milliseconds, except for voltage.
  //
  // Temp  VAClear  VAIdle  AAClear  AAIdle  Drive  Voltage
  // ----  -
  // 45    50      2       20      6       10     24
  { 45,   0x12,   0x05,   0x0D,   0x07,   0x09,   0x40},
  // ----  -
  // 40    60      2       25      6       12     24
  { 40,   0x13,   0x05,   0x0E,   0x07,   0x0A,   0x40},
  // ----  -
  // 35    80      2       30      6       14     24
  { 35,   0x14,   0x05,   0x0F,   0x07,   0x0B,   0x40},
  // ----  -
  // 30    100     2       40      6       20     24
  { 30,   0x15,   0x05,   0x11,   0x07,   0x0D,   0x40},
  // ----  -
  // 25    100     2       40      6       20     24
  { 25,   0x15,   0x05,   0x11,   0x07,   0x0D,   0x40},
  // ----  -
  // 20    150     2       50      6       25     24
  { 20,   0x16,   0x05,   0x12,   0x07,   0x0E,   0x40},
  // ----  -
  // 15    150     2       60      6       35     24
  { 15,   0x16,   0x05,   0x13,   0x07,   0x10,   0x40},
  // ----  -
  // 10    250     2       100     6       50     24
  { 10,   0x18,   0x05,   0x15,   0x07,   0x12,   0x40},
  // ----  -
  // 5     350     4       150    10      80     24
  { 5,    0x19,   0x06,   0x16,   0x09,   0x14,   0x40},
  // ----  -
  // 0     500     6       200    18     100    24
  { 0,    0x1A,   0x07,   0x17,   0x0C,   0x15,   0x40}
};
/////////////////////////////////////////////////////////////////

```



```
//
// P r i v a t e   F u n c t i o n s
//
//
//
// Function: DriveImage
// Purpose: Updates display with image data in driver RAM using
//          supplied drive parameters.
// Inputs:  firstRow  - first physical display row to update
//          numRows   - number of rows to update
//          VAClearPW - VA Clear Pulse width (driver encoding)
//          VAIdle    - VA Idle Duration (driver encoding)
//          AAClearPW - AA Clear Pulse width (driver encoding)
//          AAIdle    - AA Idle Duration (driver encoding)
//          DrivePW   - Scan Pulse width (driver encoding)
//          ClearV    - VA/AA Clear Voltage (driver encoding)
//          DriveV    - Scan Voltage (driver encoding)
//          VAClearRpt - Number of VA Clear Cycles
//          VAIdleRpt - Number of VA Idles
//          AAClearRpt - Number of AA Clear Cycles
//          AAIdleRpt - Number of AA Idles
//          DriveRpt  - Number of Driver Cycles
//          Bias      - Drive Voltage Bias (driver encoding)
// Outputs: None.
// Notes:  Full-screen updates include additional dummy rows (DUMMY_ROWS) to
//          let last image row see some non-select rows.
//          Driver must already be powered with image loaded.
//
//
void DriveImage(uint8_t firstRow, uint8_t numRows,
               uint8_t VAClearPW, uint8_t VAIdle,
               uint8_t AAClearPW, uint8_t AAIdle,
               uint8_t DrivePW, uint8_t ClearV, uint8_t DriveV,
               uint8_t VAClearRpt, uint8_t VAIdleRpt,
               uint8_t AAClearRpt, uint8_t AAIdleRpt,
               uint8_t DriveRpt,
               uint8_t Bias)
{
    uint8_t offset, startLineCmd;

    // Map first row to first driver COMMON.
    if ((firstRow == 0) && (numRows == 32)) // Full screen?
    {
        // Include top/bottom active frame in update area. Thus the image spans
        // 34 COM lines, COM15 to COM48.
        firstRow = 15;
        numRows = 34;
    }
    else // Partial screen.
    {
        // First image row is on the COM line that is 16 inward from the edge
        // COM16 or COM47.
        firstRow += 16;
    }
}
```



```
// Compute image location parameters.
#ifdef REMAP_IMAGE
    offset = 0x40 - firstRow;    // Location on display.
#else
    offset = firstRow + numRows; // Location on display.
    if (firstRow == 15)         // Offset needs special treatment when we
        offset += DUMMY_ROWS;  // do a full image in non-remapped mode.
#endif
startLineCmd = 0x40 + ((firstRow==15) ? 15 : 16); // Location in display RAM.

// Initialization code.
PutCharSPI( 0xA3, CMD_MASK ); // Enable band gap and other analog control.
PutCharSPI( 0x18, CMD_MASK );
PutCharSPI( 0xF6, CMD_MASK ); // Enable oscillator.
PutCharSPI( 0x40, CMD_MASK );
PutCharSPI( 0xAE, CMD_MASK ); // Set auto charge pump threshold value.
PutCharSPI( 0x00, CMD_MASK );

PutCharSPI( 0xA2, CMD_MASK ); // Set Bias level
PutCharSPI( Bias, CMD_MASK );

// Set drive parameters.
PutCharSPI( 0x80, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK );
PutCharSPI( VAClearPW, CMD_MASK ); // VA Clear Duration
PutCharSPI( VAIdle, CMD_MASK ); // VA Idle Duration. Stir time.
PutCharSPI( AAClearPW, CMD_MASK ); // AA Clear Duration
PutCharSPI( AAIdle, CMD_MASK ); // AA Idle Duration
PutCharSPI( DrivePW, CMD_MASK ); // Drive Duration
PutCharSPI( ClearV, CMD_MASK ); // Clear Voltage
PutCharSPI( DriveV, CMD_MASK ); // Drive Voltage

// Set dummy waveform for supply initialization.
PutCharSPI( 0x93, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK ); // Skip VA Clear.
PutCharSPI( 0x94, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK ); // Skip VA Idle.
PutCharSPI( 0x95, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK ); // Skip AA Clear.
PutCharSPI( 0x96, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK ); // Skip AA Idle.
PutCharSPI( 0x97, CMD_MASK );
PutCharSPI( 0x00, CMD_MASK ); // Skip Drive.

// Dummy update results in future supply initialization to Clear Voltage.
PutCharSPI( 0x31, CMD_MASK ); // Dummy update.
WaitNonBusy();

// More driver initialization code.
PutCharSPI( 0xA3, CMD_MASK ); // Enable other analog control.
PutCharSPI( 0x1A, CMD_MASK );
PutCharSPI( 0xE9, CMD_MASK ); // Enable bias driven.
```



```
PutCharSPI( 0x84, CMD_MASK);
PutCharSPI( 0x2F, CMD_MASK); // Enable booster and high voltage buffer.

// Delay to allow dc/dc reach voltage.
// Rise time to 25V with Vin = 2.4V is 280 msec.
// Rise time to 25V with Vin = 3.3V is 160 msec.
Delay(MS2TCK(300));

// Set up update.
PutCharSPI( 0x93, CMD_MASK );
PutCharSPI( VAClearRpt, CMD_MASK ); // VA Clear Repeats
PutCharSPI( 0x94, CMD_MASK );
PutCharSPI( VAIdleRpt, CMD_MASK ); // VA Idle Repeats
PutCharSPI( 0x95, CMD_MASK );
PutCharSPI( AAClearRpt, CMD_MASK ); // AA Clear Repeats
PutCharSPI( 0x96, CMD_MASK );
PutCharSPI( AAIdleRpt, CMD_MASK ); // AA Idle Repeats
PutCharSPI( 0x97, CMD_MASK );
PutCharSPI( DriveRpt, CMD_MASK ); // Drive Repeats

PutCharSPI( 0x32, CMD_MASK ); // Drive scheme:
PutCharSPI( 0x32, CMD_MASK ); // Clear to bright.

// Configure update area.
PutCharSPI( 0xa8, CMD_MASK );
if (numRows == 34)
    PutCharSPI( numRows+DUMMY_ROWS, CMD_MASK ); // Mux ratio, full screen.
else
    PutCharSPI( numRows, CMD_MASK ); // Mux ratio, partial screen.

PutCharSPI( 0xd3, CMD_MASK );
PutCharSPI( offset, CMD_MASK ); // Physical location on screen.
PutCharSPI( startLineCmd, CMD_MASK ); // Starting location in RAM.

// Set AA-idle-to-drive delay to 8 msec (coincidentally, 0x08 in driver
// encoding). Shorter values may cause image problems due to chip logic,
// especially at high operating frequencies. The problem manifests
// as one or more spurious bright lines during a partial update.
PutCharSPI( 0xAA, CMD_MASK );
PutCharSPI( 0x08, CMD_MASK );

// Set COMs to remap or not remap the image. Segs are remapped during
// display RAM loading.
#ifdef REMAP_IMAGE
    PutCharSPI( 0xC0, CMD_MASK ); // Scan COMs normally
#else
    PutCharSPI( 0xC8, CMD_MASK ); // Scan COMs in reverse.
#endif

// Perform Update.
PutCharSPI( 0x31, CMD_MASK);
WaitNonBusy();
```



```

// Put display to sleep.
PutCharSPI( 0x2A, CMD_MASK);
PutCharSPI( 0xE9, CMD_MASK);
PutCharSPI( 0x04, CMD_MASK);
PutCharSPI( 0xF6, CMD_MASK);
PutCharSPI( 0x00, CMD_MASK);
PutCharSPI( 0xA3, CMD_MASK);
PutCharSPI( 0x00, CMD_MASK);
PutCharSPI( 0xAB, CMD_MASK);
PutCharSPI( 0x00, CMD_MASK);

// Power down display/driver.
DisableVDD();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// P u b l i c   F u n c t i o n s
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function: Display
// Purpose: Updates display with image data in driver RAM using
//          temperature compensation table drive parameters.
// Inputs:  firstRow - first physical display row to update
//          numRows  - number of rows to update
// Outputs: None
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Display(uint8_t firstRow, uint8_t numRows)
{
    uint8_t VAClearPW, VAIdle, AAClearPW, AAIdle, DrivePW, ClearV;
    uint8_t DriveV, VAClearRpt, VAIdleRpt, AAClearRpt, AAIdleRpt, DriveRpt, Bias;
    uint8_t i;
    int16_t temperature;

    // Read the temperature in degrees C.
    temperature = GetTemperature();

    // Update display if the temperature is within limits.
    if ((temperature >= tcTable[TEMP_PTS-1].Temp) && (temperature <= TEMP_MAX))
    {
        // Search TC table to find the drive parameters.
        for (i = 0; i < TEMP_PTS; i++)
        {
            if ( temperature >= tcTable[i].Temp ) // Within range of table entry?
            {
                // Read parameters from table.
                DrivePW  = tcTable[i].DrivePW;
                DriveV   = tcTable[i].Voltage;
                VAClearPW = tcTable[i].VAClearPW;
                VAIdle   = tcTable[i].VAIdle;      // Stir time between VA & AA clear.
                AAClearPW = tcTable[i].AAClearPW;
            }
        }
    }
}

```



```

    AAIIdle    = tcTable[i].AAIdle;
    ClearV     = DriveV;

    // Set fixed parameters
    VAClearRpt = 1;
    VAIdleRpt  = 1;
    AAClearRpt = 1;
    AAIIdleRpt = 1;
    DriveRpt   = 1;
    Bias       = 0x02;                // 1/7

    // Display the image and break out of the search loop.
    DriveImage( firstRow, numRows,
                VAClearPW, VAIdle, AAClearPW, AAIIdle,
                DrivePW, ClearV, DriveV,
                VAClearRpt, VAIdleRpt,
                AAClearRpt, AAIIdleRpt,
                DriveRpt, Bias);

    break;
}
}
}

// Turn VDD off if not already done. This is done in DriveImage(),
// but that won't be called if the temperature is out of range.
if (VDDEnabled())
    DisableVDD();
}

```

### 6.2.3 system.c

The functions in `system.c` use the platform-specific I/O functions to provide an interface to the display hardware.

```

#include "defines.h"

////////////////////////////////////////////////////////////////
// Function:  PutCharSPI
// Purpose:   Sends a byte to the SPI bus.
// Inputs:    data - byte to send
//            Data_nCmd - command (0) or data (else) attribute of byte to send
// Outputs:   None.
////////////////////////////////////////////////////////////////
void PutCharSPI(uint8_t data, uint8_t Data_nCmd)
{
    uint8_t mask;

    // Assert chip select to display.
    // The driver seems to require a new chip select to start each byte.
    SelectDisplay();

    // Signal byte type to driver.
    if (Data_nCmd)

```



```
    Assert_D_Drv();
else
    Assert_nC_Drv();

// Bit-bang data out, MSB first. Data changes on falling edge and
// is latched on rising. Clock is inactive low.
for (mask = 0x80; mask != 0; mask = mask >> 1)
{
    if (data & mask)
        setSIMO();
    else
        clearSIMO();

    setSCLK();
    clearSCLK();
}

// De-assert chip select to display.
DeselectDisplay();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function: EnableVDD
// Purpose:  Enable power to the display.
// Inputs:   None.
// Outputs:  None.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void EnableVDD(void)
{
    // Enable power to driver and temperature circuit(s).
    VDDOn();

    // Delay ~2 msec for power to come on w/ reset still asserted low.
    Delay(MS2TCK(2));

    // Bring logic signals to driver to active levels and release reset.
    DeselectDisplay();
    Deassert_nRESET_Drv();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function: DisableVDD
// Purpose:  Remove power from the display.
// Inputs:   None.
// Outputs:  None.
// Notes:   The reference design removes power by switching off a FET,
//          essentially disconnecting the driver from its supply voltage.
//          So the code below sets all interface and control lines to the
//          driver to zero in order to avoid sourcing current through
//          the interface lines. In designs which do not remove power,
//          relying instead on the driver's lower power sleep mode, do not
//          assert the reset or chip select outputs, as doing so will cause
//          extra current demand.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```



```

//////////////////////////////////////////////////////////////////
void DisableVDD(void)
{
    // Set driver interface signals low.
    ZeroDriverLines();

    // Set driver SPI communications signals low.
    ZeroSPILines();

    // Cut power to driver and temp circuits.
    VDDOff();

    Delay(MS2TCK(25));
}

//////////////////////////////////////////////////////////////////
// Function:  HardResetDisplay
// Purpose:  Asserts the reset line to the display.  Systems which leave
//           the driver powered between updates should perform a hard reset
//           before a data load / update cycle.
// Inputs:   None.
// Outputs:  None.
//////////////////////////////////////////////////////////////////
void HardResetDisplay(void)
{
    // Assert reset line.
    Assert_nRESET_DRV();

    // Delay ~2 msec (only 20 uS req'd).
    Delay(MS2TCK(2));

    // De-assert reset line.
    Deassert_nRESET_DRV();
}

```

#### 6.2.4 LoadData.c

```

#include "defines.h"

//////////////////////////////////////////////////////////////////
//
// D e f i n i t i o n s
//
//////////////////////////////////////////////////////////////////
#define COL_LSN      0x00 // Set Lower Column Address
#define COL_MSN      0x10 // Set Higher Column Address
#define PAGE_ADD     0xB0 // Set Page Address

//////////////////////////////////////////////////////////////////
//
// P u b l i c   F u n c t i o n s
//
//////////////////////////////////////////////////////////////////

```



```
////////////////////////////////////
// Function: LoadData
// Purpose: Load a portion of display RAM with image data.
// Inputs: Index - starting position within display RAM for data (0 to
//           BYTES_PER_IMAGE-1)
//           NumBytes - number of bytes of image data to load
//           pData - pointer to the image data to load
// Outputs: None.
// Notes: Index is adjusted to properly map into display RAM.
//         If Vdd is off, this function turns it on and leaves it on,
//         presuming the display routine will turn it off later.
////////////////////////////////////
void LoadData(uint16_t Index, uint16_t NumBytes, const uint8_t *pData)
{
    uint8_t colAddr, page;
    uint16_t i;

    // Enable power to display and reset driver if it's not been done already.
    if (!VDDEnabled())
        EnableVDD();

    // Segment remapping must be done when loading data.
#ifdef REMAP_IMAGE
    PutCharSPI( 0xA1, CMD_MASK ); // Remap segs.
#else
    PutCharSPI( 0xA0, CMD_MASK ); // Don't remap segs.
#endif

    // Loop through the data to write.
    for (i = Index; i < (Index + NumBytes); i++)
    {
        colAddr = (i % N_COLS) + 2; // Add two to skip unused & frame.

        // Set page and column address as needed.
        if ((colAddr == 2) || (i == Index) )
        {
            page = (i/N_COLS) + 2; // Add two to skip unused & frame.
            PutCharSPI( (COL_MSN | (colAddr / 16)) , CMD_MASK );
            PutCharSPI( (COL_LSN | (colAddr % 16)) , CMD_MASK );
            PutCharSPI( (PAGE_ADD | page) , CMD_MASK );
        }

        // Write a data byte.
        PutCharSPI( *pData++, DATA_MASK );
    }

    // NOP command is required after writing data before chip select goes high.
    PutCharSPI( 0xE3, CMD_MASK );
}

////////////////////////////////////
// Function: LoadBorder
```



```
// Purpose: Sets the display RAM corresponding to the active frame to the
//          given value.
// Inputs:  border - 0: dark, 0xFF: previous, else: bright
// Outputs: None.
// Notes:   If Vdd is off, this function turns it on and leaves it on,
//          presuming the display routine will turn it off later.
//          ////////////////////////////////////////////////////////////////////
void LoadBorder(uint8_t border)
{
    static uint8_t Previous = 0;          // Previous border setting
    uint8_t data, i, colAddr, page;

    // Enable power to display and reset driver if it's not been done already.
    if (!VDDEnabled())
        EnableVDD();

    // Segment remapping must be done when loading data.
#ifdef REMAP_IMAGE
    PutCharSPI( 0xA1, CMD_MASK );        // Remap segs.
#else
    PutCharSPI( 0xA0, CMD_MASK );        // Don't remap segs.
#endif

    // Use previous border if specified. Else, save the given border as
    // the previous.
    if (border == PREVIOUS_BORDER)
        border = Previous;
    else
        Previous = border;

    // Initialize the image memory data value for the border.
    if (border == 0)
        data = 0;
    else
        data = 0xff;

    // Write the top border.
    colAddr = 1;
    page = 1;
    PutCharSPI( (COL_MSN | (colAddr / 16)) , CMD_MASK );
    PutCharSPI( (COL_LSN | (colAddr % 16)) , CMD_MASK );
    PutCharSPI( (PAGE_ADD | page) , CMD_MASK );
    for ( i = 0; i < N_COLS+2; i++)
        PutCharSPI( data, DATA_MASK );

    // Write the bottom border.
    colAddr = 1;
    page = N_PAGES + 2;
    PutCharSPI( (COL_MSN | (colAddr / 16)) , CMD_MASK );
    PutCharSPI( (COL_LSN | (colAddr % 16)) , CMD_MASK );
    PutCharSPI( (PAGE_ADD | page) , CMD_MASK );
    for ( i = 0; i < N_COLS+2; i++)
        PutCharSPI( data, DATA_MASK );
}
```



```
// Write the side borders for the 4 image pages.
for (page = 2; page < 2+N_PAGES; page++)
{
    // Write the left side border.
    colAddr = 1;
    PutCharSPI( (COL_MSN | (colAddr / 16)) , CMD_MASK );
    PutCharSPI( (COL_LSN | (colAddr % 16)) , CMD_MASK );
    PutCharSPI( (PAGE_ADD | page) , CMD_MASK );
    PutCharSPI( data, DATA_MASK );

    // Write the right side border.
    colAddr = N_COLS+2;
    PutCharSPI( (COL_MSN | (colAddr / 16)) , CMD_MASK );
    PutCharSPI( (COL_LSN | (colAddr % 16)) , CMD_MASK );
    PutCharSPI( data, DATA_MASK );
}
}
```

### 6.2.5 SampleImage.c

```
#include "defines.h"

// A pointer to this array may be passed to the Display() function for initial
// debugging. The array encodes the same image pictured on the displays in
// the datasheet. The data is otherwise not required.

const uint8_t sampleImage[512] = {
    0xFD, 0xFD, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
    0xF9, 0xF8, 0xFE, 0xFE, 0x7E, 0x3E, 0x9C, 0xC1,
    0xE3, 0xFF, 0xFF, 0xE3, 0x41, 0x1C, 0xBE, 0xBE,
    0xBE, 0x1C, 0x41, 0xE3, 0xFF, 0xFF, 0xFF, 0xFF,
    0xF7, 0xC7, 0x8F, 0x3F, 0x7F, 0x3F, 0x8F, 0xC7,
    0xF7, 0xFF, 0xFF, 0xFF, 0xFF, 0xF9, 0xF8, 0xFE,
    0xBE, 0xBE, 0xBE, 0x1C, 0x01, 0xE3, 0xFF, 0xFF,
    0xF9, 0xF8, 0xFE, 0xFE, 0x7E, 0x3E, 0x9C, 0xC1,
    0xE3, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xEF, 0xEF, 0xE0, 0xE0, 0x6F, 0x6F, 0xFF, 0xFF,
    0x67, 0x63, 0xE9, 0xEC, 0xEE, 0x6F, 0x6F, 0x6F,
    0xEF, 0xFF, 0xFF, 0xF8, 0xF0, 0xE7, 0xEF, 0xEF,
    0xEF, 0xE7, 0xF0, 0xF8, 0xFF, 0xFF, 0xFF, 0xFF,
    0xEF, 0xE3, 0xF1, 0xFC, 0xFE, 0xFC, 0xF1, 0xE3,
    0xEF, 0xFF, 0xFF, 0xFF, 0xFF, 0xF3, 0xE3, 0xEF,
    0xEF, 0xEF, 0xEF, 0xE7, 0xF0, 0xF8, 0xFF, 0xFF,
    0xE7, 0xE3, 0xE9, 0xEC, 0xEE, 0xEF, 0xEF, 0xEF,
    0xEF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```



```

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFF, 0xFF, 0xFF, 0xF8, 0xF7, 0xF7, 0xFF, 0x18,
0xF7, 0xF7, 0xF8, 0xFF, 0xF8, 0xD7, 0xD7, 0xE0,
0xFF, 0xFF, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD,
0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0x3D, 0xFD, 0xFD,
0xFD, 0xFD, 0x7D, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD,
0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0x3D, 0xFD,
0x7D, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD,
0xFD, 0xFD, 0xFD, 0x3D, 0xFD, 0xFD, 0xFD, 0xFD,
0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFD, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0xFF, 0xFF, 0xFF, 0xFF, 0xE3, 0x5D, 0x5D, 0x81,
0xFF, 0xC1, 0xFB, 0xED, 0xD5, 0xD5, 0xC3, 0xFF,
0x01, 0xDD, 0xDD, 0xE3, 0xFF, 0xC0, 0xFD, 0xFD,
0xC3, 0xFF, 0xC1, 0xFF, 0xE3, 0xDD, 0xDD, 0xFF,
0xFF, 0xFF, 0xFF, 0xE3, 0xDD, 0xDD, 0xC0, 0xFF,
0xC1, 0xFF, 0xD9, 0xD5, 0xCD, 0xFF, 0x01, 0xDD,
0xDD, 0xE3, 0xFF, 0xC0, 0xFF, 0xEF, 0xD5, 0xD5,
0xC3, 0xFF, 0xFD, 0x63, 0x9F, 0xE3, 0xFD, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

```

### 6.3 Main Code for User Program

The following code example shows how a user-defined `main()` function might be organized.

```

#include "defines.h"

extern const uint8_t sampleImage[];

void main(void)           // Program entry point.
{
    // Do host-specific initialization, such as setting up I/O ports, here. //
    EnableVDD();          // Power-up the display driver.
}

```



```

HardResetDisplay();      // Reset the driver

// Transfer the local image buffer to display RAM, setting the border bright.
LoadData(0, BYTES_PER_IMAGE, SampleImage);
LoadBorder(1);

// Perform a full-screen update of the display. This
// will power-down the display driver when finished.
Display(0, N_ROWS);

for (;;) {}             // Loop forever.
}

```

### 6.4 Code Copyright Notice

The code described in this chapter is copyrighted as described in the notice below:

```

/* *****
Copyright (c) 2009. Kent Displays, Inc. All Rights Reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose, without fee, and without a written
agreement, is hereby granted, provided that the above copyright notice,
this paragraph, and the following two paragraphs appear in all copies,
modifications, and distributions and is included with a Kent Displays,
Inc. product.

IN NO EVENT SHALL KENT DISPLAYS, INC. BE LIABLE TO ANY PARTY FOR DIRECT,
INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING
LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS
DOCUMENTATION, EVEN IF KENT DISPLAYS, INC. HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

KENT DISPLAYS, INC. SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING
DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS".
KENT DISPLAYS, INC. HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT,
UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
***** /

```



## 7 Ordering Information

Displays may be ordered as standalone modules. The release date for a development kit is yet to be determined.



FIGURE 12: DISPLAY ASSEMBLY

TABLE 6: ORDERING INFORMATION

PART #	DESCRIPTION
01594101208	Module, 128x32 White/Blue

Contact Kent Displays at [sales@kentdisplays.com](mailto:sales@kentdisplays.com) for additional color options, custom configurations, pricing, and additional information.



## 8 Product Handling Precautions

- 1) General
  - a) The liquid crystal device is made of glass. The display is sensitive to mechanical shock and scratching.
  - b) The display device can be damaged by flexing or bending. Caution should be taken to protect the display from delamination. If the display layers are separated, handle with care. Do not ingest the LCD fluid itself if it should leak from a damaged LCD module. The toxicity is extremely low but caution should be exercised at all times. If hands or clothing come in contact with LCD fluid, wash immediately with soap.
  - c) If the surface of the LCD is equipped with a UV protection film, it is necessary to guard against scratches as the film is a soft material.
  - d) Do not drive the LCD with DC voltage.
  - e) Avoid touching the viewing surface of the display to prevent deposition of oil and fats.
  - f) Use a power supply that is equipped with an over-current protection circuit. The display or display module is not provided with this protective feature.
- 2) Static Electricity
  - a) Ground your body and any electrical equipment you may be using when working with the module. Never touch or come into contact with any of the conductive parts such as the pads, the copper leads on the PCB and the interface terminals. The use of an antistatic mat to protect work tables from electrical shock is strongly recommended.
- b) Keep the display modules in antistatic bags or other containers resistant to static for storage purposes.
- c) Use soldering irons that are properly grounded.
- d) Avoid wearing work clothing composed of synthetic fibers. Instead, clothing made of cotton or other conductive fibers is recommended.
- e) Since dry air is inductive to statics, a relative humidity of 50 – 60% is recommended.
- f) Slowly and carefully remove the protective film from the LCD module to prevent the generation of static electricity.
- 3) Storage
  - a) LCD should be kept in sealed polyethylene bags while MDL should use antistatic bags. If properly sealed, there is no need for desiccant.
  - b) For long periods of storage, keep the temperature between 0 °C and 35 °C.
  - c) Protect the modules from high temperature and humidity.
  - d) Keep the modules out of direct sunlight or direct exposure to UV rays.
  - e) Protect the modules from excessive external forces.
- 4) Models which use flexible cable, heat seal, COF, or TAB:
  - a) Do not touch or hold by the connector area, as doing this may hinder reliability.
  - b) Avoid bending, pulling, or other excessive force, which can result in broken connections.

© 2009 Kent Displays, Inc., All rights reserved. Glass Display technology and products are protected by U.S. patents 5,251,048, 5,384,067, 5,437,811, 5,453,863, 5,493,430, 5,625,477, 5,636,044, 5,644,330, 5,695,682, 6,133,895, 6,172,720, 6,204,835, 6,268,839, 6,366,330, 7,023,409; pending patents include 11/464,698, 11/626,428. Other U.S. and foreign patents pending on products, technologies and services of Kent Display Systems, Inc., Kent Displays, Inc. and Kent State University. Foreign countries with PCT patent filings include: Canada, China, Europe, Israel, Japan, Korea, and Taiwan among others.