Using Segmented Displays (LCD)

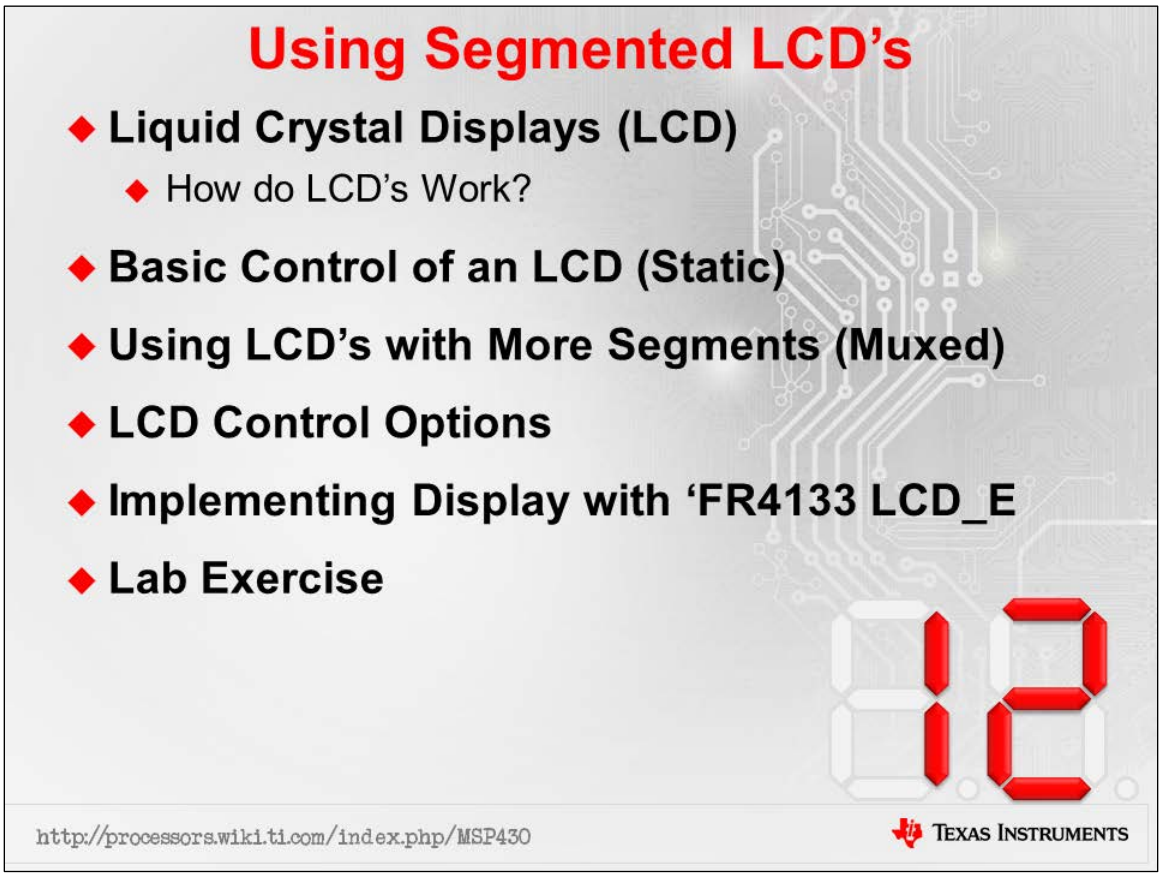
Introduction

This chapter introduces the segmented liquid crystal display (LCD). We begin with a quick introduction to LCD's and how they work.

Next we examine how they can be controlled and used within an embedded system.

Finally we learn how to implement designs with the LCD_E controller found on the MSP430FR4133 microcontroller. The 'FR4133 Launchpad – with its built-in LCD display – makes a great platform platform for LCD experimentation.

Learning Objectives



Chapter Topics

For More Information on LCD's

For More Information

- LCD app note: www.ti.com/lit/pdf/slaa654
- MSP430 Microcontroller Basics by John H. Davies, (ISBN-10 0750682760) Link↗

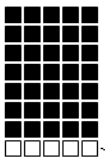
Liquid Crystal Displays (LCD)

Types of LCD's

Segmented Passive Displays

7-Segment Digit

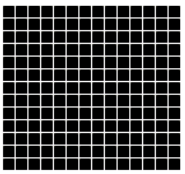
14 or 16-Segment Character



5 X 7 Alphanumeric Character

Cursor row

Graphic Displays (Passive & Active Matrix)



Monochrome

Color

- **Segmented Passive LCD's**
  - Cheaper than Active Matrix
  - Ideal low-power embedded applications
  - Easily controlled by microcontrollers
- **This chapter focuses on using segmented LCD's**

* http://www.pcmag.com/encyclopedia/term/60488/lcd-types

MSP430 + LCD Examples

Appliances

A/C remote control

Healthcare

Blood Glucose Meter

Consumer

One-Time-Password Token

Watches

Low-power LCD hand held

Industrial

Water Meter

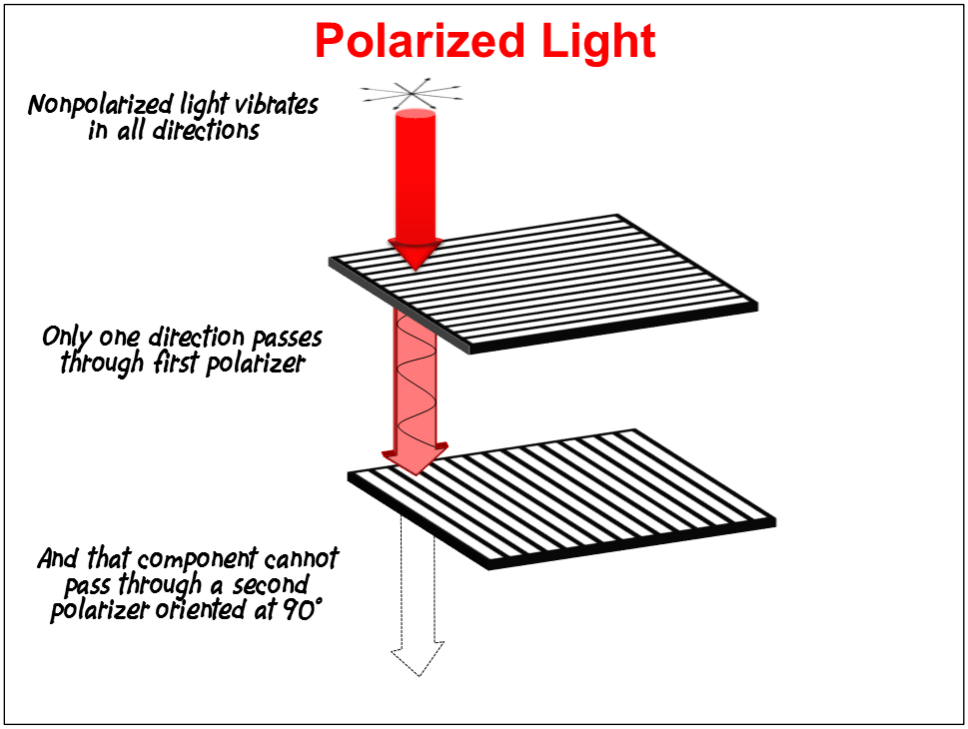
Commercial

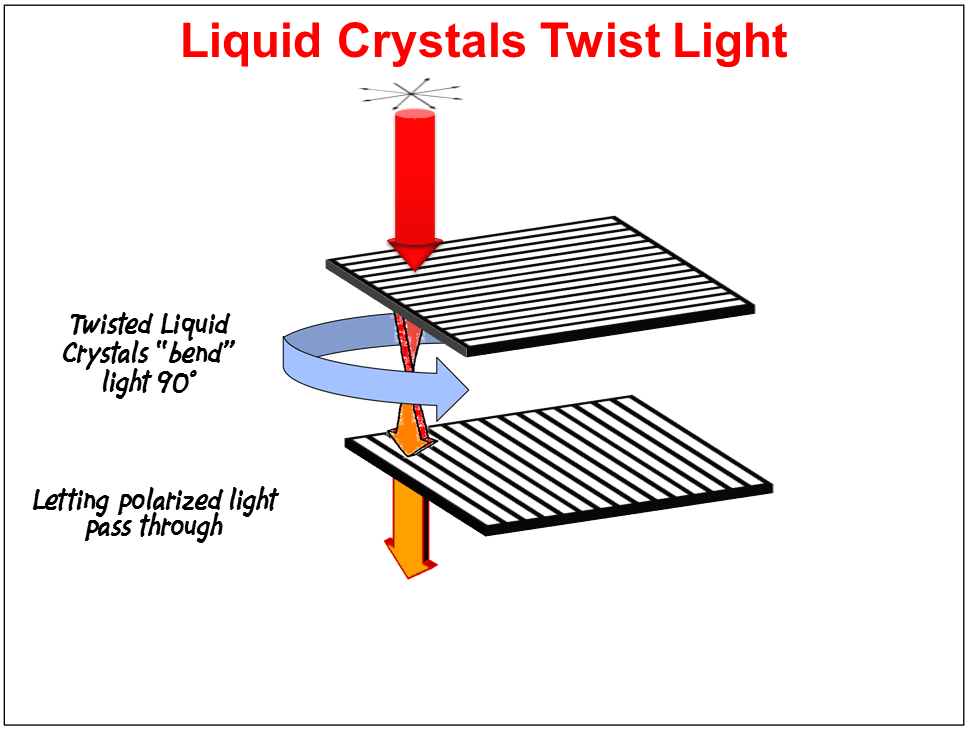
E-Shelf Label

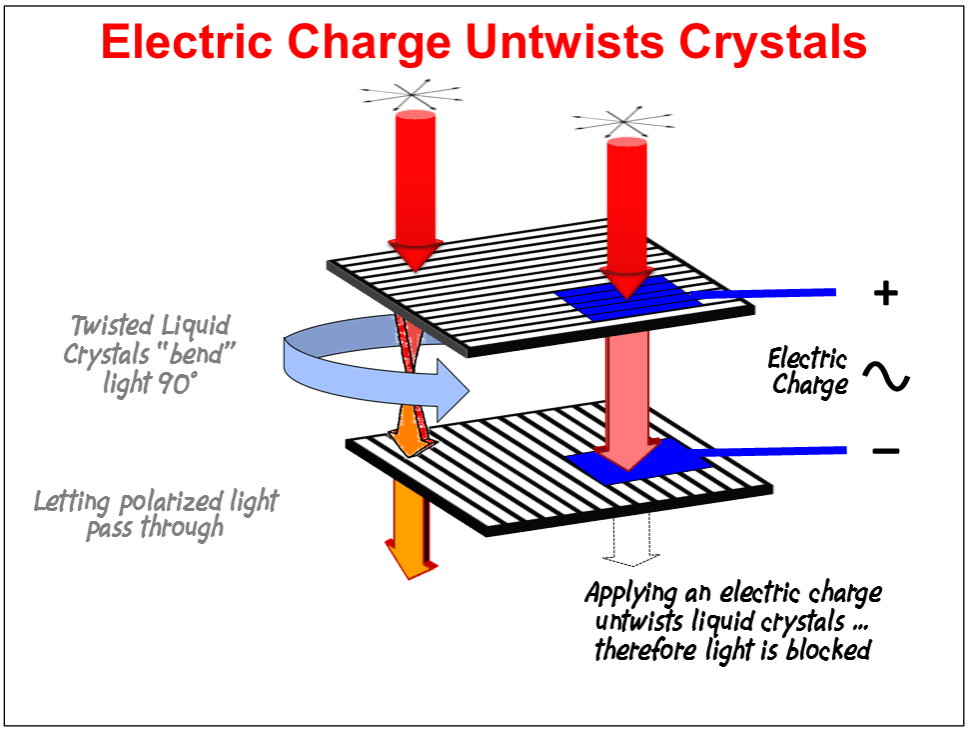
Simple control with LCD display

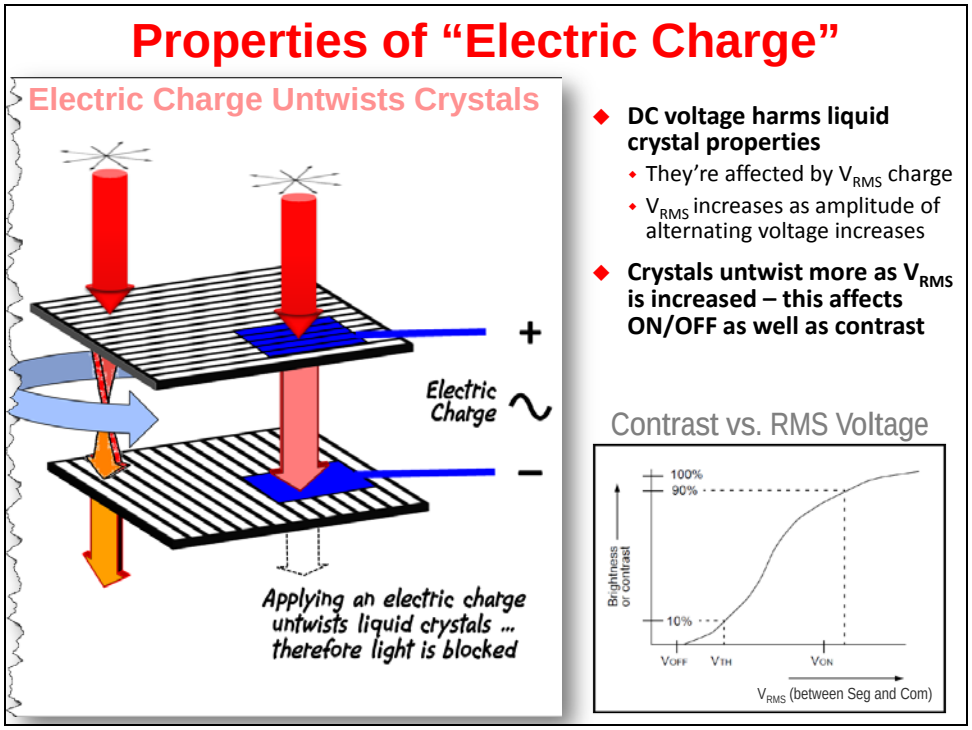
Remotes

How do LCD's Work?



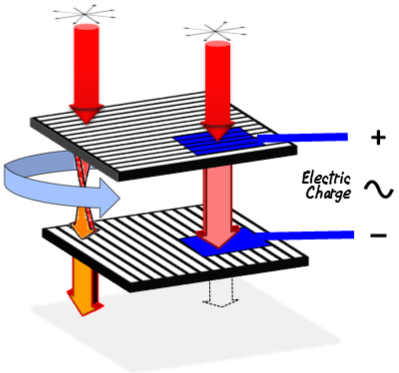



Electric Charge Untwists Crystals
Twisted Liquid Crystals "bend" light 90°
Letting polarized light pass through
+
Electric Charge
−
Applying an electric charge untwists liquid crystals ... therefore light is blocked


Properties of "Electric Charge"
Electric Charge Untwists Crystals
+
Electric Charge
−
Applying an electric charge untwists liquid crystals ... therefore light is blocked
DC voltage harms liquid crystal properties
They're affected by V_{RMS} charge
V_{RMS} increases as amplitude of alternating voltage increases
Crystals untwist more as V_{RMS} is increased – this affects ON/OFF as well as contrast
Contrast vs. RMS Voltage
100%
90%
10%
Brightness or contrast
VOFF
VTH
VON
V_{RMS} (between Seg and Com)

Review of LCD Basics

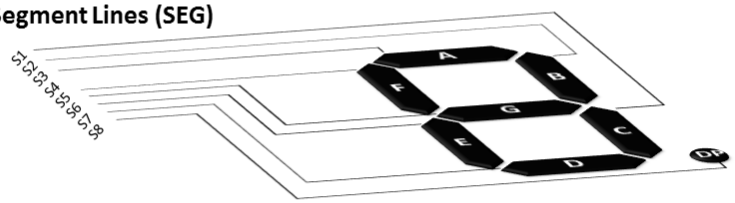
- Polarizers at 90 degree rotation from each other
- Reflective backing projects image back to user
- Segment "Off":
  - No electric charge applied
  - Liquid crystals twist the light so it goes through both polarizers
  - Looks grey
- Segment "On":
  - V_{RMS} electric charge untwists crystal molecules
  - Liquid crystals don't twist the light - so light can't get through both polarizers
  - Looks dark (e.g. black)



Basic Control of an LCD (Static)

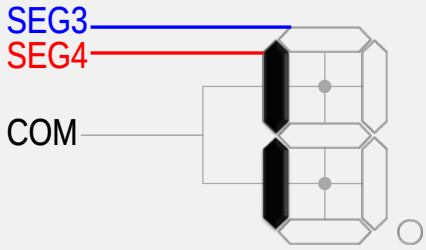
Simple Connection to Display

- LCD's are controlled with Segment (SEG) & Common (COM) lines
  - Each SEG line turns a segment ON/OFF
  - COM provides the 'ground' for all of the segments
  - Your application will require N+1 pins for N segments
    (The next section shows how to get more segments per pin)
  - SEG/COM pins could be GPIO or dedicated LCD controller pins
- Example:
  - Segment Lines (SEG)

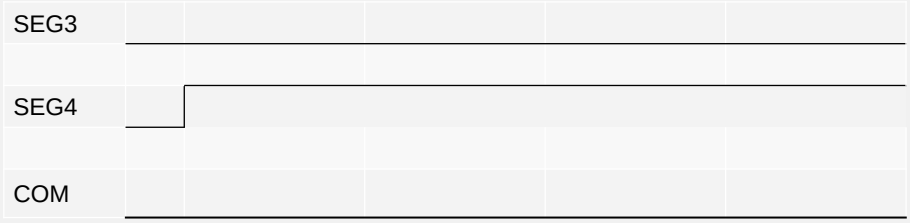


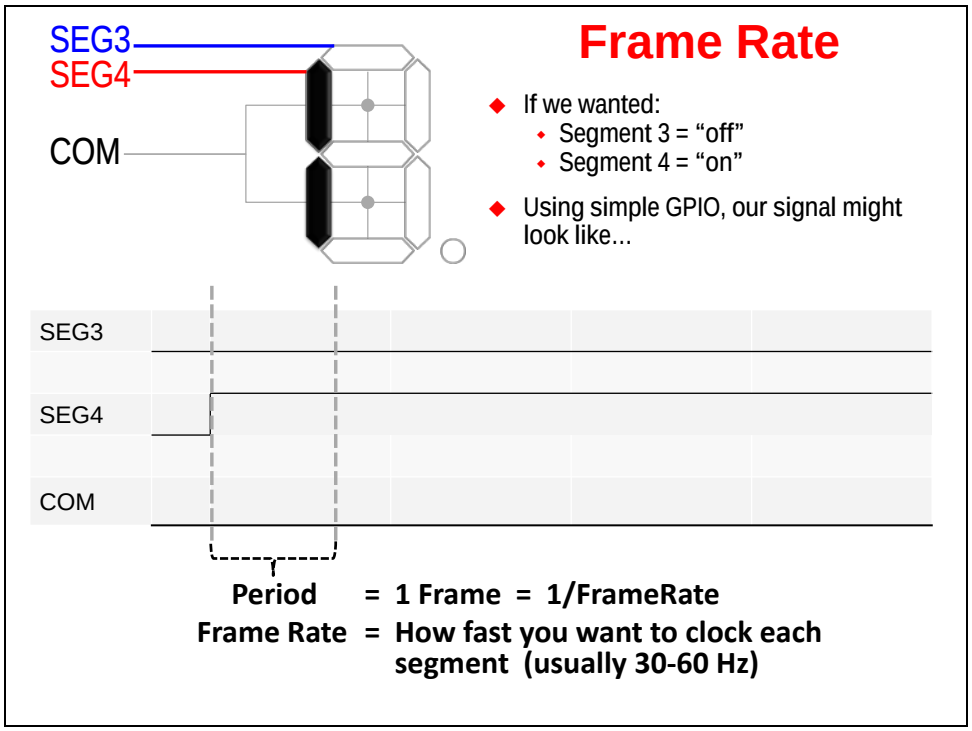
  - Common Line (COM)

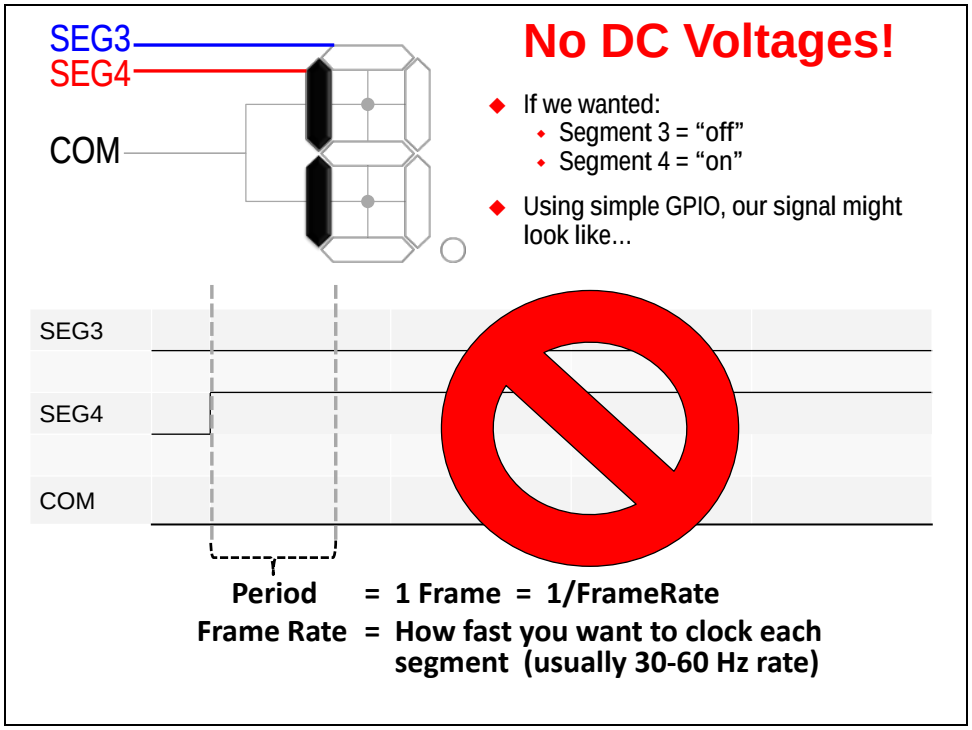
Driving SEG and COM

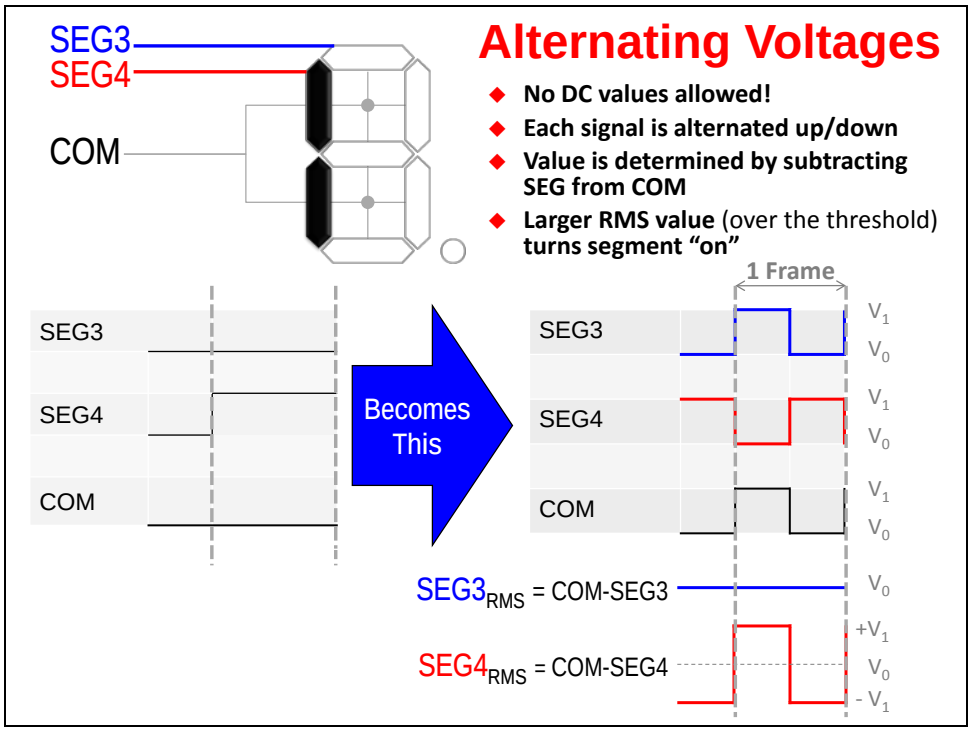


- If we wanted:
  - Segment 3 = "off"
  - Segment 4 = "on"
- Using simple GPIO, our signal might look like…




SEG3
SEG4
COM
Frame Rate
If we wanted:
Segment 3 = “off”
Segment 4 = “on”
Using simple GPIO, our signal might look like...
SEG3
SEG4
COM
Period = 1 Frame = 1/FrameRate
Frame Rate = How fast you want to clock each segment (usually 30-60 Hz)


SEG3
SEG4
COM
No DC Voltages!
If we wanted:
Segment 3 = “off”
Segment 4 = “on”
Using simple GPIO, our signal might look like...
SEG3
SEG4
COM
Period = 1 Frame = 1/FrameRate
Frame Rate = How fast you want to clock each segment (usually 30-60 Hz rate)

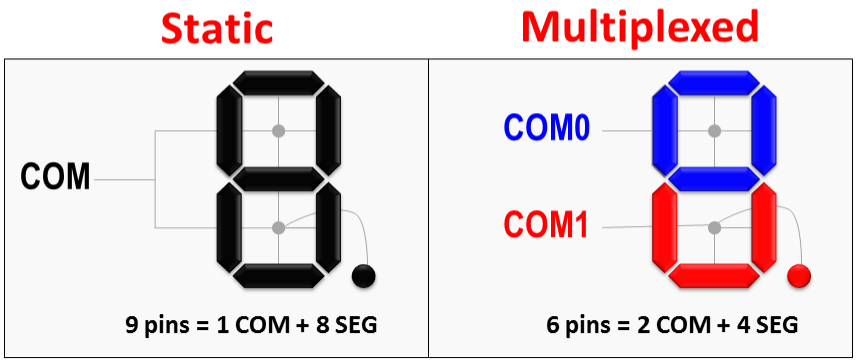

SEG3
SEG4
COM
Alternating Voltages
No DC values allowed!
Each signal is alternated up/down
Value is determined by subtracting SEG from COM
Larger RMS value (over the threshold) turns segment "on"
1 Frame
SEG3
SEG4
COM
Becomes This
SEG3
SEG4
COM
V1
V0
V1
V0
V1
V0
SEG3RMS = COM-SEG3
SEG4RMS = COM-SEG4
V0
+V1
V0
- V1

Using LCD's with More Segments (Muxed)

Static vs Muxed

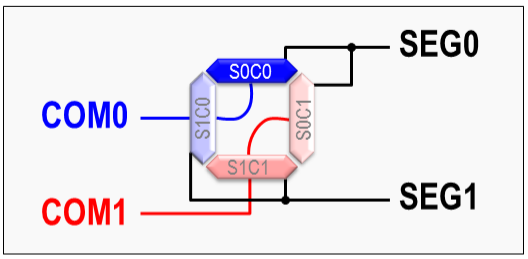
Multiplexed Connections

- "Static" designs have only one common for all the segments
- "Muxed" designs use multiple common lines
  - Multiplexing is a common embedded systems *trick*
  - Segment pins are reused over-and-over again
  - Allows many more segments can be controlled by same # of pins
- Look at this simple Static and 2-Mux example

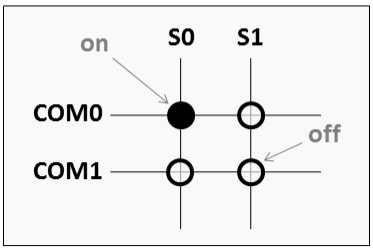


Let's examine a simple 2-mux example…

Simple 2-Mux Example

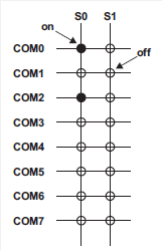


Here's a common shorthand nomenclature for expressing LCD connections



Muxing

- Segmented LCDs use Muxing to limit control pin count
- Notation: Static (no mux), 2-mux, 3-mux, 4-mux, etc
  - N-mux = each segment pin (Sx) drives N segments
  - This means there are N common (COMx) pins.
- Up to 8-mux on some MSP430s
  - Some MSP430s support up to 320 segments
- See device-specific datasheet



8-mux

Texas Instruments

Muxed Control Signals

Basic 2-Mux Example

- **Differences from Static Example**
  - Each COM is handled every frame in a TDM (time-division muxed) fashion
  - Frame contains 2*C phases (where C is the number of COM lines)
  - Per frame: COM0 alternates in first 2 phases, while COM1 in the 2nd two phases
  - Multiple bias voltages are needed to create multiplexed V_{RMS} values (e.g. V1, V3, V5)
- **Signals are combined to determine state of each segment**
  - As before larger "swings" (i.e. amplitude) enable the segment
- **Thankfully… LCD peripheral handles the details**
  - Waveforms generated automatically by MSP430 LCD module
  - Low-power LCD charge pump can generate all required bias voltages
  - LCD memory makes it easy to specify which segments should be 'on'

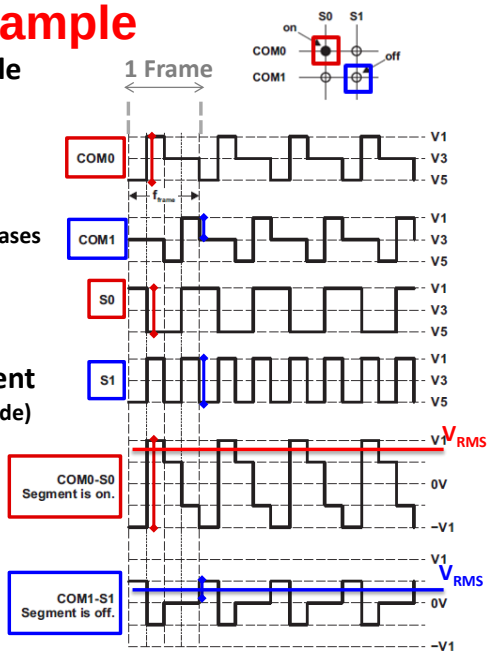
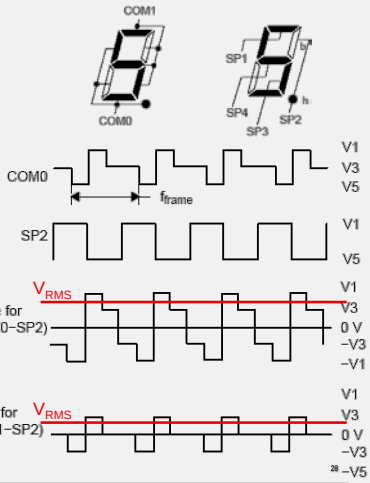


Figure 33-6. Example 2-Mux Waveforms

LCD Drive Basics

- AC signals only
  - DC can damage the LCD
- Automatic signal generation
- RMS voltage determines if segment is on/off
- <50mV DC voltage allowed



TI Confidential – NDA Restrictions 28

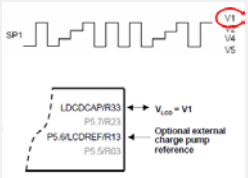
Texas Instruments

Bias Voltages

- Multiple biases needed to make it all work.... thankfully for us, the controller handles this
  - Contrast
  - Get this info from display datasheet?
- Voltage sources
  - External (resistor network)
  - Charge pump

Charge Pump

- V_{LCD} sets V1 (the highest LCD voltage level):
  - Avcc
  - Internal Charge Pump
  - External
- Most MSP430 LCD modules include a built-in charge pump
- Programmable voltage levels
- External reference sync voltage of multiple MSP430s
- Requires one external capacitor



TI Confidential – NDA Restrictions 30

Texas Instruments

Biasing

- Bias voltages can be generated internally or externally (V2 to V5)
  - Independent of V_{LCD} source
  - 1/2,1/3 bias options
- Internal = simpler
  - No external resistors needed
- External = lower power
  - Requires external resistor divider
  - Resistor size depends on display

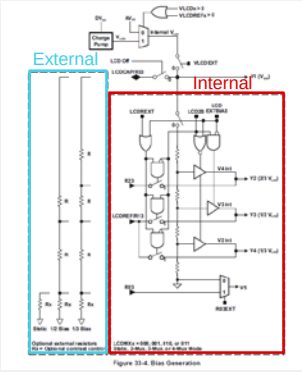


Figure 33-4. Bias Generation

TI Confidential – NDA Restrictions 31

Texas Instruments

Contrast Control

- Charge pump gives software selectable V_{LCD}
- Changing V_{LCD} automatically adjusts other LCD voltages (no matter internal/external bias)
- Different biasing modes (1/2, 1/3, etc) along with used LCD display also impact contrast ratio.
  - Tradeoff: 1/3 bias gives better contrast, but remember 1/3 requires higher V_{LCD}
- External contrast control also an option

Table 1-3. LCD Voltage and Biasing Characteristics

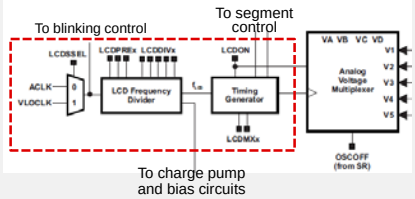
Mode	Bias Config	LCDMx	LCD2B	COM Lines	Voltage Levels	$V_{RMS,OFF}/V_{LCD}$	$V_{RMS,ON}/V_{LCD}$	Contrast Ratio $V_{RMS,ON}/V_{RMS,OFF}$
Static	Static	0000	X	1	V1, V5	0	1	1/0
2-mux	1/2 (1)	0001	1	2	V1, V3, V5	0.354	0.791	2.236
	1/3	0001	0	2	V1, V2, V4, V5	0.333	0.745	2.236
3-mux	1/2	0010	1	3	V1, V3, V5	0.408	0.707	1.732
	1/3 (1)	0010	0	3	V1, V2, V4, V5	0.333	0.638	1.915
4-mux	1/2	0011	1	4	V1, V3, V5	0.433	0.661	1.528
	1/3 (1)	0011	0	4	V1, V2, V4, V5	0.333	0.577	1.732
5-mux	1/3 (1)	0100	0	5	V1, V2, V4, V5	0.333	0.537	1.612
6-mux	1/3 (1)	0101	0	6	V1, V2, V4, V5	0.333	0.509	1.528
7-mux	1/3 (1)	0110	0	7	V1, V2, V4, V5	0.333	0.488	1.464
8-mux	1/3 (1)	0111	0	8	V1, V2, V4, V5	0.333	0.471	1.414

(1) Recommended setting to achieve best contrast

TI Confid 32

Texas Instruments

Timing



- Internal timing generation
- Clock can be prescaled and divided further within the module
- f_{LCD} generates timing for COMx and Sx lines
- f_{LCD} based on f_{FRAME} of LCD used
- Lowest frequency = lowest current
- Highest frequency = lowest flicker

$$F_{LCD} = 2 \times MUX \times F_{FRAME}$$

TI Confidential – NDA Restrictions 33

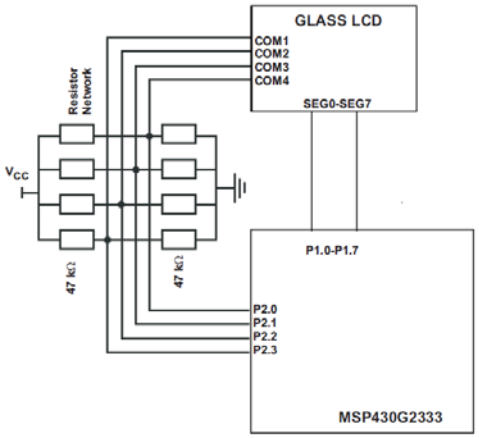
Texas Instruments

LCD Control Options

Bit Banging Display

Devices without LCD module

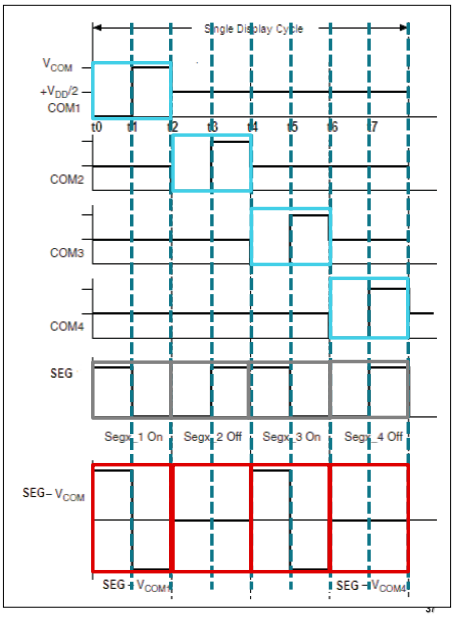
- **Two Options:**
  - **Use external LCD controller (connect via SPI, I2C)**
  - **Bit Bang - Use software and GPIO to drive the display**
- **App Note: Software Glass LCD Driver Based on MSP430 MCU**
  www.ti.com/lit/pdf/slaa516
  - **Use resistors to for bias voltage**
  - **Timer to do the frame timing**
  - **4-mux software example**
- **Tradeoffs:**
  - **+ More device options; Not required to find device with LCD**
  - **– Higher current consumption (wake 8 times per frame)**
  - **– Uses CPU cycles just to keep display "on"**
  - **– More external components req'd**
  - **– Code is quite a bit more complex**



SLAA516 App Note

- Example code for 4-mux included with app note
- Frame divided into 8 time slots –
  - 4 (1 for each COM)
  - each divided into two parts because no DC on LCD (must toggle)
- Timer used to generate the 8 slots
- Must wake on each slot and software set all COM & SEG lines
- SEG same as COM = off
- SEG opposite COM = on

Time Slots



TI Confidential – NDA Restrictions

Texas Instruments

Displays with Built-in Drivers

Displays with Built-in Drivers

- Dot-matrix LCD or e-paper displays
- Typically have built-in driver
- Typically controlled using SPI or I2C, so MSP430 with USI or USCI can easily control these
  - Some displays do not have read-back capability, so may need to stor current image in MSP430 RAM/FRAM
- Sharp LCD Boosterpack www.ti.com/tool/430boost-sharp96

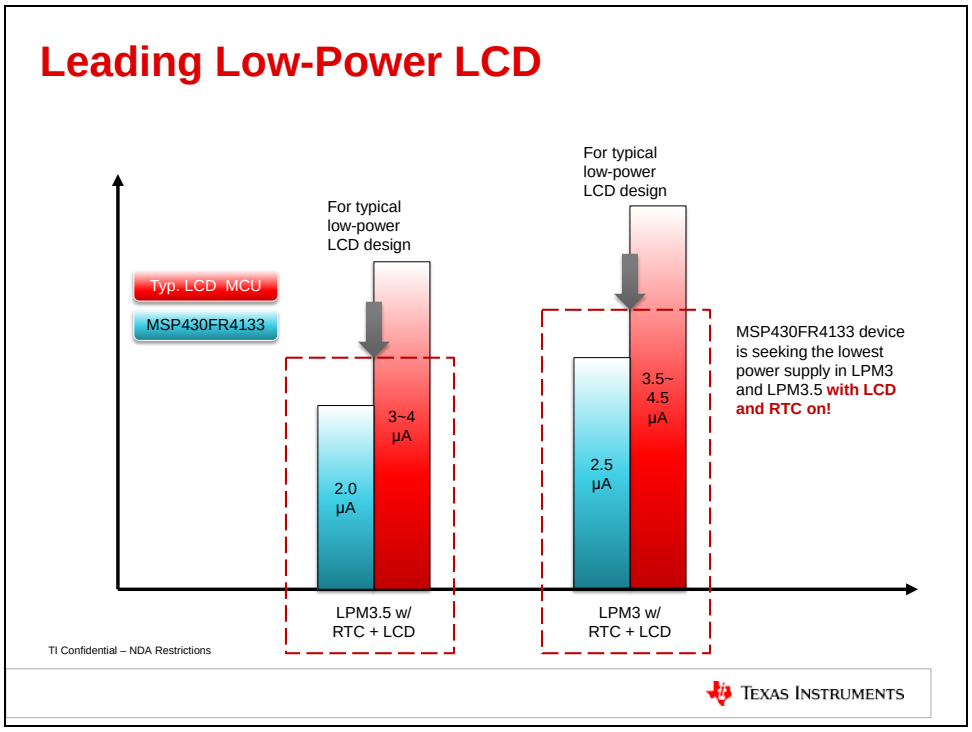
TI Confidential – NDA Restrictions 39

Texas Instruments

MSP430 LCD Peripherals

MSP430 LCD portfolio

Parameter	LCD	LCD_A	LCD_B	LCD_C	LDC_E
Number of Segments	128/4-MUX	160/4-MUX	160/4-MUX	320/8-MUX	448/8-MUX
Number of LCD Pins			up to 4x46	up to 4 x 50 or 8 x 46	up to 4 x 60 or 8 x 56
Segment functionality against port pin selection	Minimum is group of 16	Selection done in groups of 4 segments	Individual selection can be done	Individual selection can be done	Individual selection can be done
COM/SEG Pin Assignments	COM Fixed	COM Fixed	COM Fixed	COM Fixed	Any LCD pin
LCD Clock selection	ACLK	ACLK	ACLK, VLO	ACLK, VLO	XT1, ACLK, VLO
Interrupt capabilities	NO	NO	YES (4 sources)	YES (4 sources)	YES (3 sources)
Individual segment blinking capabilities	NO	NO	YES	YES	YES
Prog. blinking frequency	N/A	N/A	YES	YES	YES
Dual memory display	NO	NO	YES	YES	YES
Works in LPM3.5	No	No	No	No	Yes



Implementing Display with ‘FR4133 LCD_E

Choose Display and Pin Layout

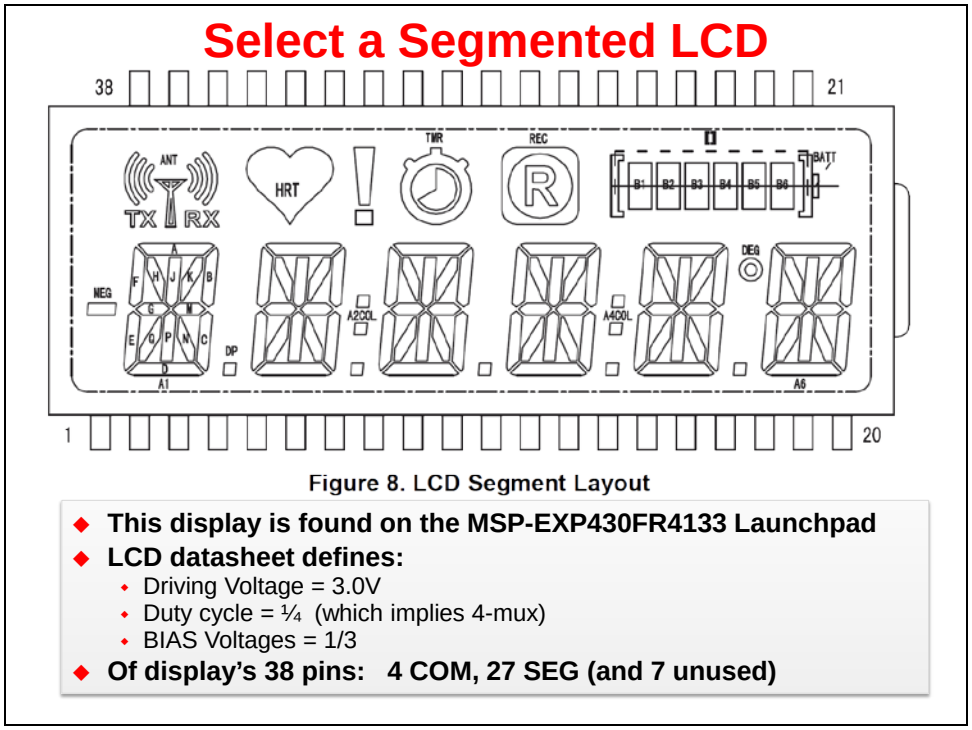
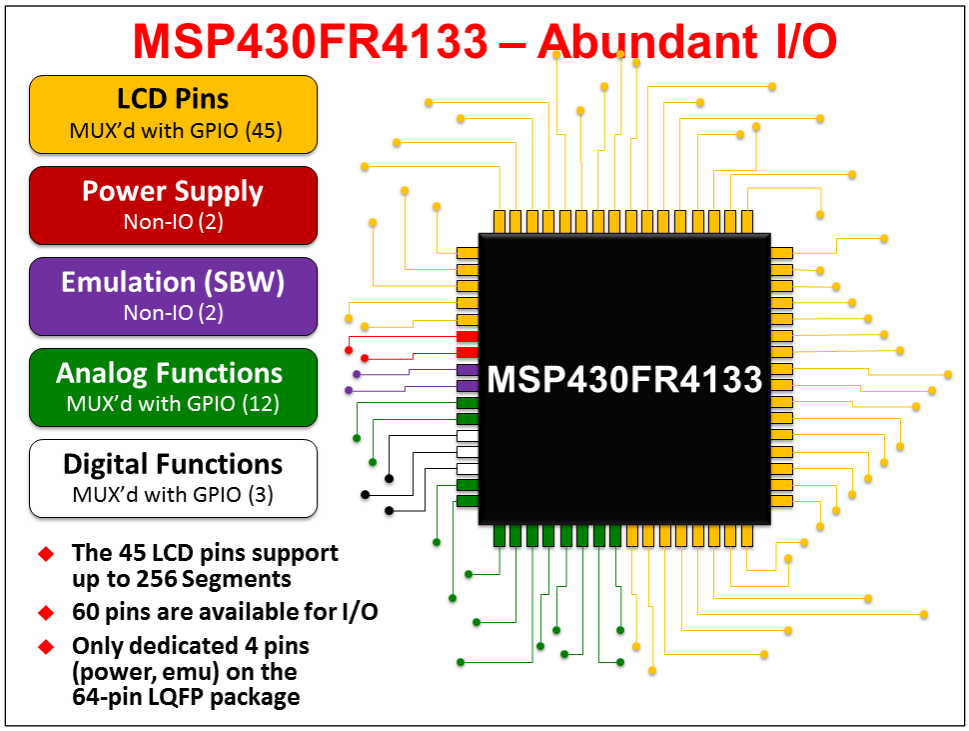
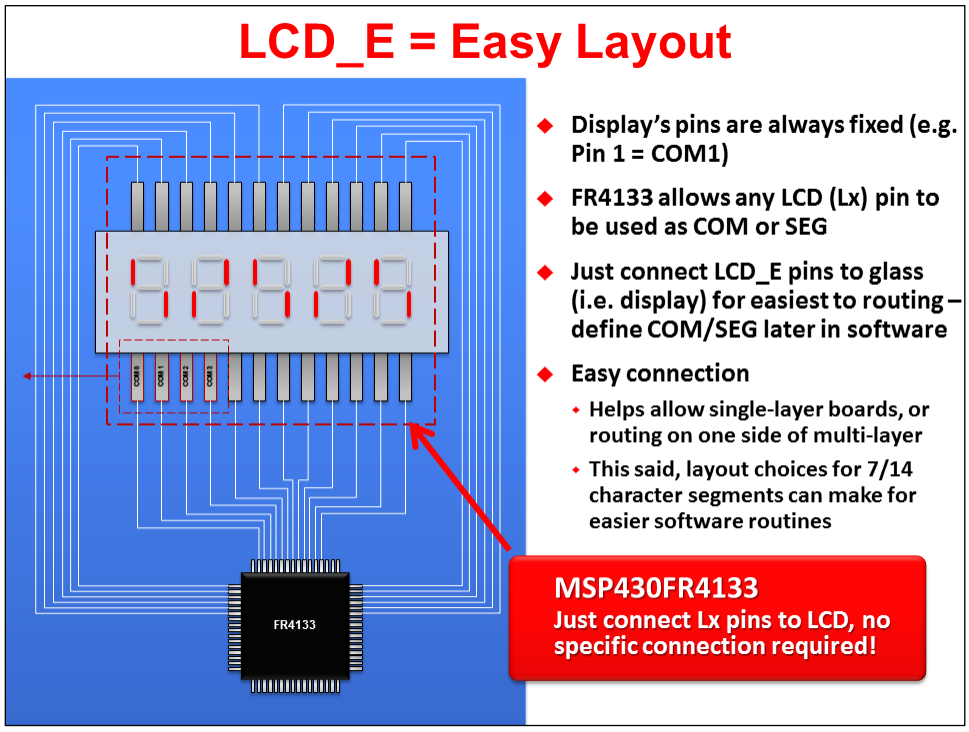


Figure 8. LCD Segment Layout

- **This display is found on the MSP-EXP430FR4133 Launchpad**
- **LCD datasheet defines:**
  - Driving Voltage = 3.0V
  - Duty cycle = ¼ (which implies 4-mux)
  - BIAS Voltages = 1/3
- **Of display’s 38 pins: 4 COM, 27 SEG (and 7 unused)**


MSP430FR4133 – Abundant I/O
LCD Pins
MUX'd with GPIO (45)
Power Supply
Non-IO (2)
Emulation (SBW)
Non-IO (2)
Analog Functions
MUX'd with GPIO (12)
Digital Functions
MUX'd with GPIO (3)
MSP430FR4133
The 45 LCD pins support up to 256 Segments
60 pins are available for I/O
Only dedicated 4 pins (power, emu) on the 64-pin LQFP package


LCD_E = Easy Layout
Display's pins are always fixed (e.g. Pin 1 = COM1)
FR4133 allows any LCD (Lx) pin to be used as COM or SEG
Just connect LCD_E pins to glass (i.e. display) for easiest to routing – define COM/SEG later in software
Easy connection
Helps allow single-layer boards, or routing on one side of multi-layer
This said, layout choices for 7/14 character segments can make for easier software routines
FR4133
MSP430FR4133
Just connect Lx pins to LCD, no specific connection required!

LCD Init Code

LCD Init – LCD Pins

- **Using the LCD datasheet's specs:**
  - Driving Voltage = 3.0V
  - Duty cycle = ¼ (which implies 4-mux)
  - BIAS Voltages = 1/3
- **Of display's 38 pins: 4 COM, 27 SEG (and 7 unused)**

```
void initLCD(void) {
 // Turn off LCD
 LCD_E_off(LCD_E_BASE);

 // Select range(s) of FR4133 LCD pins (Lx) to connect to LCD
 // Note: this means they won't be available for GPIO
 LCD_E_setPinAsLCDFunctionEx( );
 LCD_E_setPinAsLCDFunctionEx( );

 // Configure first 4 pins as COMMON lines (COM0 – COM3)
 LCD_E_setPinAsCOM( LCD_E_SEGMENT_LINE_0, LCD_E_MEMORY_COM0);
 LCD_E_setPinAsCOM( LCD_E_SEGMENT_LINE_1, LCD_E_MEMORY_COM1);
 LCD_E_setPinAsCOM( LCD_E_SEGMENT_LINE_2, LCD_E_MEMORY_COM2);
 LCD_E_setPinAsCOM( LCD_E_SEGMENT_LINE_3, LCD_E_MEMORY_COM3);
```

LCD Init – Configure Modes

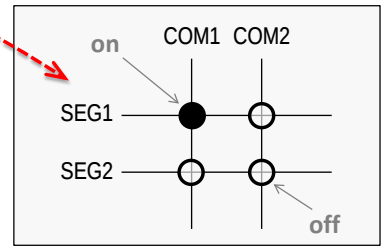
- **Using the LCD datasheet's specs:**
  - Driving Voltage = 3.0V
  - Duty cycle = ¼ (which implies 4-mux)
  - BIAS Voltages = 1/3
- **Of display's 38 pins: 4 COM, 27 SEG (and 7 unused)**

```
// Initialze LCD Clock and Mux mode
LCD_E_initParam initParams = LCD_E_INIT_PARAM;
initParams.clockDivider = CLOCKDIV_3;  // Set frame rate
initParams.muxRate      = LCD_E_4_MUX; // Select mux
LCD_E_init(LCD_E_BASE, &initParams);

// Configure Voltage Sources for the LCD Controller
LCD_E_setVLCDSource(INTERNAL_REF_VOLTAGE, EXTERNAL_SUPPLY_V
LCD_E_setVLCDVoltage(LCD_E_REFERENCE_VOLTAGE_2_96V);
LCD_E_enableChargePump(LCD_E_BASE);
LCD_E_setChargePumpFreq(LCD_E_BASE, LCD_E_CHARGEPUMP_FREQ_16);

// Turn off LCD
 LCD_E_on(LCD_E_BASE);
```

Controlling Segments

LCD Memory

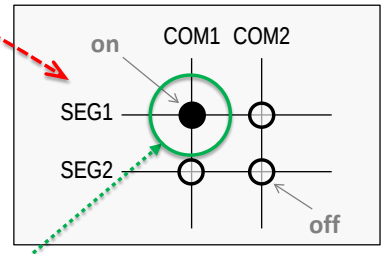
- Recall our earlier nomenclature for COM/SEG connections
- LCD_E include memory registers where each bit controls a segment
  - Remember, in 4-mux mode, each Lx pin controls 4 segments
  - Thus an 8-bit memory register can control up to 8 segments



Segments	COM1	COM2	COM3	COM4	COM1	COM2	COM3	COM4
LCDM19	LCD Pin (L39)				LCD Pin (L38)			
…								
LCDM1	LCD Pin (L3)				LCD Pin (L2)			
LCDM0	LCD Pin (L1)				LCD Pin (L0)			

LCD Memory

- Recall our earlier nomenclature for COM/SEG connections
- LCD_E include memory registers where each bit controls a segment
  - Remember, in 4-mux mode, each Lx pin controls 4 segments
  - Thus an 8-bit memory register can control up to 8 segments



Segments	COM1	COM2	COM3	COM4	COM1	COM2	COM3	COM4
LCDM19	LCD Pin (L39)				LCD Pin (L38)			
…	Setting a bit turns on a segment							
LCDM1	1	0	0	0	LCD Pin (L2)			
LCDM0	LCD Pin (L1)				LCD Pin (L0)			

FR4133 Launchpad LCD Connections

- Launchpad User's Guide nicely describes pin connections:
  - e.g. FR4133 pin (L1), Port Pin (P7.1), LCD's pin (23)
  - Along with which segment each LCDMEM bit enables

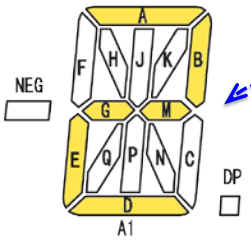
LCDMEM	Port Pin	FR4133 Pin	LCD Pin	COM3	COM2	COM1	COM0	Port Pin	FR4133 Pin	LCD Pin	COM3	COM2	COM1	COM0
LCDM19	P5.7	L39	38	A6H	A6J	A6K	A6P	P5.6	L38	37	A6Q	TX	A6N	RX
LCDM18	P5.5	L37	36	A6A	A6B	A6C	A6D	P5.4	L36	35	A6E	A6F	A6G	A6M
LCDM17	P5.3	L35						P5.2	L34					
LCDM16	P5.1	L33						P5.0	L32					
LCDM15	P2.7	L31						P2.6	L30					
LCDM14	P2.5	L29						P2.4	L28					
LCDM13	P2.3	L27						P2.2	L26	34	B5	B3	B1	[]
LCDM12	P2.1	L25	33	B6	B4	B2	BATT	P2.0	L24	32	TMR	HRT	REC	I
LCDM11	P6.7	L23	16	A4H	A4J	A4K	A4P	P6.6	L22	15	A4Q	A4COL	A4N	A4DP
LCDM10	P6.5	L21	14	A4A	A4B	A4C	A4D	P6.4	L20	13	A4R	A4F	A4G	A4M
LCDM9	P6.3	L19	12	A3H	A3J	A3K	A3P	P6.2	L18	11	A3Q	ANT	A3N	A3DP
LCDM8	P6.1	L17	10	A3A	A3B	A3C	A3D	P6.0	L16	9	A3R	A3F	A3G	A3M
LCDM7	P3.7	L15	8	A2H	A2J	A2K	A2P	P3.6	L14	7	A2Q	A2COL	A2N	A2DP
LCDM6	P3.5	L13	6	A2A	A2B	A2C	A2D	P3.4	L12	5	A2E	A2F	A2G	A2M
LCDM5	P3.3	L11	4	A1H	A1J	A1K	A1P	P3.2	L10	3	A1Q	NEG	A1N	A1DP
LCDM4	P3.1	L9	2	A1A	A1B	A1C	A1D	P3.0	L8	1	A1E	A1F	A1G	A1M
LCDM3	P7.7	L7	20	A5H	A5J	A5K	A5P	P7.6	L6	19	A5Q	DEG	A5N	A5DP
LCDM2	P7.5	L5	18	A5A	A5B	A5C	A5D	P7.4	L4	17	A5E	A5F	A5G	A5M
LCDM1	P7.3	L3	21	COM3	-	-	-	P7.2	L2	22	-	COM2	-	-
LCDM0	P7.1	L1	23	-	-	COM1	-	P7.0	L0	24	-	-	-	COM0

Table 4. LCD to MSP430 Connections

Choosing Layout for Easier Software

LCDM6	P3.5	L13	6	A2A	A2B	A2C	A2D	P3.4	L12	5	A2E	A2F	A2G	A2M
LCDM5	P3.3	L11	4	A1H	A1J	A1K	A1P	P3.2	L10	3	A1Q	NEG	A1N	A1DP
LCDM4	P3.1	L9	2	A1A	A1B	A1C	A1D	P3.0	L8	1	A1E	A1F	A1G	A1M
LCDM3	P7.7	L7	20	A5H	A5J	A5K	A5P	P7.6	L6	19	A5Q	DEG	A5N	A5DP
LCDM2	P7.5	L5	18	A5A	A5B	A5C	A5D	P7.4	L4	17	A5E	A5F	A5G	A5M
LCDM1	P7.3	L3	21	COM3	-	-	-	P7.2	L2	22	-	COM2	-	-
LCDM0	P7.1	L1	23	-	-	COM1	-	P7.0	L0	24	-	-	-	COM0

- **Notice how we can set an entire character (e.g. "2") by writing one or two consecutive LCD Memory locations (LCD4 and LCD5)**
- **Hence the comment: *"Choosing a good pin layout can ease software"***
- **Make programming easier by pre-defining segment values for digits; you can also define the alphabet in a similar fashion**



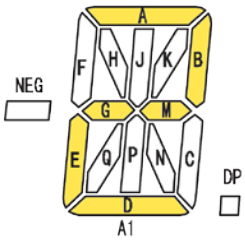

```
const char digit[10] =
{
    0xFC,   /* "0" */
    0x60,   /* "1" */
    0xDB,   /* "2" */
    0xF3,   /* "3" */
    0x67,   /* "4" */
    0xB7,   /* "5" */
    ...
```

Controlling LCD_E with DriverLib

```
#define pos1  4          // Position 1 (A1) is at LCDM4

LCD_E_setMemory( LCD_E_BASE, pos1, digit[2] );
```

LCD Memory location

Segment Mask




```
const char digit[10] =
{
    0xFC,   /* "0" */
    0x60,   /* "1" */
    0xDB,   /* "2" */
    0xF3,   /* "3" */
    0x67,   /* "4" */
    0xB7,   /* "5" */
    ...
```

Set/Clear Segments with DriverLib

```
#define pos1  4          // Position 1 (A1) is at LCDM4

LCD_E_setMemory( LCD_E_BASE, pos1, digit[2] );
```

Turn segments on/off:

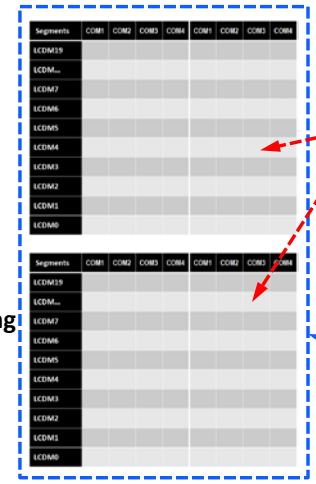
- LCD_E_setMemory() Overwrites LCDMx memory with provided value
- LCD_E_clearMemory() Clears the specified bits of LCDMx register
- LCD_E_toggleMemory() Toggles all 8-bits in bits in LCDMx register
- LCD_E_updateMemory() Sets the specified bits of LCDMx (LCDMx |= mask)

Dual Memories & Blinking

Two LCD Memories

LCD Memory (LCDM)

LCD Blinking Memory (LCDBM)



- **LCD_E has two similar memories:**
  - **LCDM**
  - **LCDBM**
- **In Static, Mux-2, and Mux-4 modes:**
  - Use as dual display memories, easily switching between them
  - Put a different image in each and let the LCD controller alternate between them – to create a custom blinking effect
- **In Mux-5 thru -8 modes:**
  - Since more than 4 common (COM) lines are required, both memories are needed to hold a single image
- **LCD_E is disabled**
  - You can use the LCD memory for other data

DriverLib Overview

```
#define pos1  4          // Position 1 (A1) is at LCDM4

LCD_E_setBlinkingMemory( LCD_E_BASE, pos1, digit[2] );
```

Turn Blinking Memory segments on/off:

- LCD_E_setBlinkingMemory() — Overwrites LCDBMx location
- LCD_E_clearBlinkingMemory() — Clears specified bits of LCDBMx
- LCD_E_toggleBlinkingMemory() — Toggles all bits of LCDBMx location
- LCD_E_updateBlinkingMemory() — Sets specified bits of LCDBMx location

Clear All segments:

- LCD_E_clearAllMemory() — Clears entire LCDM memory
- LCD_E_clearAllBlinkingMemory() — Clears entire LCDBM memory

Memory vs Blinking Memory:

- LCD_E_selectDisplayMemory() — Display either LCDM or LCDBM
- LCD_E_setBlinkingControl() — Sets blinking freq. and 1 of 4 blink modes
  1. Blinking is off
  2. Blink individual segments
  3. Blink all segments
  4. Alternate display between LCDM & LCDBM

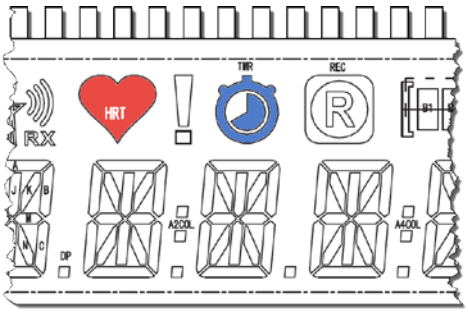
Notes:

Using Segmented Displays (LCD)

Lab 12 – Using an Segmented LCD

lab_12a_heart

- Initialize the LCD
- Explore turning on/off segments for the Heart (HRT) and Timer (TMR)
- Experiment with the blinking features of LCD_E



lab_12b_persistent

- Starting with the solution from Lab 9
- Lab 9 flashed the LED and used printf() to display the reset count to the CCS console
- In this exercise you'll add code to display the # of times the Launchpad is reset on the LCD

'FR4133 only

Chapter Topics

Lab 12a – A Launchpad with Heart

Using an LCD requires a few of steps:

- **Planning** – figuring out what LCD you need for your application; verifying the LCD controller can operate that display (often called ‘glass’); and, implementing the hardware design. For this exercise, we assume these steps have been completed and that you have a board – such as the ‘FR4133 Launchpad – that is ready for software.
- **Initialization** – like most other peripherals, we have to choose the proper modes of operation for our application and ready the device. Here are the basic initialization steps:
  - Turn off LCD_E
  - Set Lx I/O pins needed by controller
  - Setup the input clocking and frame rate (and enable the segment pins)
  - Configure voltage requirements – including enabling the built-in Charge Pump, if used.
  - Set COM pins (not required for all LCD controllers, but necessary for the ‘FR4133 since any LCD pin can be used as a COM line.
  - Finally, turn on the LCD_E controller
- **Runtime** – display the segements need for your application; changing them as necessary.

Initialization Worksheet

1. **From the MSP-EXP430FR4133 Launchpad User’s Guide, what ‘FR4133 LCD pins (Lx) need to be configured for use by the display. (Hint, look on page 15 of slau595.pdf.)**

   _____

   _____

2. **Complete the DriverLib function which sets these Lx pins as LCD pins.**

   (Hint: Look in the DriverLib User’s Guide for the proper syntax.)

```
LCD_E_setPinAsLCDFunctionEx( LCD_E_BASE,

        ____________________________________  //starting pin

        ____________________________________  //ending pin

)
LCD_E_setPinAsLCDFunctionEx( LCD_E_BASE,

        ____________________________________  //starting pin

        ____________________________________  //ending pin

)
```

3. How fast will the frame rate (F_{frame}) be given this initialization code?

This code is used to initialize the LCD controller.

```
LCD_E_initParam initParams = LCD_E_INIT_PARAM;
  initParams.clockDivider  = LCD_E_CLOCKDIVIDER_3;
  initParams.muxRate       = LCD_E_4_MUX;
  initParams.segments      = LCD_E_SEGMENTS_ENABLED;
LCD_E_init(LCD_E_BASE, &initParams);
```

Here's a brief line-by-line description of the code:

a) Creates an inititialization variable (initParam) and sets it to a set of default values. (The default values are specified in the DriverLib User's Guide.)

The remaining 3 lines of code alter these elements from their defaults. Other structure elements, such as *initParams.clockSource* is, left to its default = XT1CLK.

b) The clock divider alters the F_{lcd}, which in turn affects F_{frame}.

c) Static displays are the default, but the Launchpad use a 4-mux display.

d) By default all segments are left disabled. We want to leave them enabled.

e) The LCD_E_init() call applies the parameters to the LCD controller.

F_{LCD} = _____

F_{FRAME} = _____

Hints:

- The 'FR4133 User's Guide provides two formulas to help you calculate the frame rate.
  - As we discussed, the LCD frequency should be:

$$f_{LCD} = 2 \times mux \times f_{FRAME}$$

  - The LCD frequency can also be calculated with this expression:

$$f_{LCD} = \frac{f_{SOURCE}}{(LCDDIVx + 1) \times MUXDIVDER}$$

  - The code snippet in this lab step provides us the f_{source} and LCDDIVx values.
  - The trickiest part is figuring out the value of MUXDIVIDER. It isn't the "obvious" value, which would be "4". Rather, the value is specified in a table within the FR4133 User's Guide – look for it in the LCD_E section entitled *"LCD Timing Generation"*.

4. **Write two lines of code to clear all the LCD Memory.**

_____

_____

5. **Which bits are set by these 4 lines of code?**

```
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_0, LCD_E_MEMORY_COM0);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_1, LCD_E_MEMORY_COM1);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_2, LCD_E_MEMORY_COM2);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_3, LCD_E_MEMORY_COM3);
```

These functions tell LCD_E which (Lx) pins to use for the common (COM) signals. Where is this information stored? (That is, what gets altered by this code?)

_____

_____

Runtime Worksheet

6. **Which address/bit controls each of the following segments? Fill out the table.**

Just to get you started, we added the Antenna symbol to the table.

Symbol	Memory Location	Bit Location	Hex Value
Antenna (ANT)	LCDM9	2	0x04
Heart (HRT)			
Timer (TMR)			

7. **Write the line of code that sets (i.e. turns on) the "Heart" segment.**

```
LCD_E_______________________________( LCD_E_BASE,

            ______________________________,  // Location

            ______________________________   // Mask (hex)
);
```

8. **What happens if we set (turn on) the HRT symbol, then set the TMR symbol?**

   Will they both be enabled, or will the second one replace the first one?

   _____

   _____

9. **What function lets us clear one symbol without affecting another controlled by the same memory location (LDCMx)?**

   _____

   Complete the function to clear the Timer (TMR) symbol.

```
LCD_E_______________________________( LCD_E_BASE,
          LCD_E_MEMORY_BLINKINGMEMORY_12,  // Location
          0x8 );                           // Mask (hex)
```

10. **What's the greatest advantage to the automatic blinking features of the 'FR4133?**

    _____

    _____

    _____

11. **Finish the following line of code so that it enables the LCD controller's blinking feature – switching between both banks of memory.**

```
LCD_E_______________________________( LCD_E_BASE,
          LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64,

          ____________________________________________________ );
```

Lab File Management

12. Verify CCS is open and close any projects that are open in the workspace.

13. Import the `lab_12a_heart` project.

```
Project → Import CCS Projects…
C:\msp430_workshop\fr4133_fram\lab_12a_heart
Copy the project into your workspace
```

Note: For your reference, to created this project by copying/pasting lab_06a_timer and renaming it. We then deleted: main.c, timer.h, timer.c

Finally, we added the files: myLcd.h, myLcd.c, (and a new file called) main.c

---

Examine and Tweak LCD Files

We want to quickly introduce you to each of the three new files found in this project. Note, some will require a little bit of editing.

myLcd.h (No edits required)

Defines or declares a number of items that can be used in your programs. The three main categories are:

- Definitions for character positions – one for each character on the display (from Left→Right)
- Global variables that define values for numerical digits and the alphabet. With them, you can easily print a "3" or a ""B" to the LCD. They are defined in `mylcd.c`.
- Finally, the header includes prototypes for three functions defined in `mylcd.c`.

myLcd.h

```
11
12
13 //***** Defines ****************************************
14 /* LCD Symbol Bits */
15 #define LCD_HEART                  (0x0004)
16 #define LCD_TIMER                  (0x0008)
17
18 /* LCD memory locations for each character position on the d
19 #define pos1 4   /* Digit A1 - L4  */
20 #define pos2 6   /* Digit A2 - L6  */
21 #define pos3 8   /* Digit A3 - L8  */
22 #define pos4 10  /* Digit A4 - L10 */
23 #define pos5 2   /* Digit A5 - L2  */
24 #define pos6 18  /* Digit A6 - L18 */
25
26 /* Define word access definitions to LCD memories */
27 //#define LCDMEMW ((int*)LCDMEM)
28
29
30 //***** Global Variables ***********************************
31 extern const char digit[10][2];            // Segment values
32 extern const char alphabetBig[26][2];      // Segment values
33
34
35 //***** Prototypes *****************************************
36 void LCD_init(void);
37 void LCD_showChar( char, int );
38 void LCD_displayNumber( unsigned long );
39
```

:

myLcd.c (Some edits required)

In a nutshell, here's the things you'll find in this file. *(By the way, thanks to the MSP applications team as we borrowed quite a bit of code from their out-of-box demo application.*

- It begins with the initialization of the 'digit' and 'alphabet' arrays. Once again, this makes it easy to use symbols without needing sprinkle hex values all thoughout your code.
- LCD_init() function
  - Turn off LCD_E
  - Set Lx I/O pins needed by controller
  - Setup the input clocking and frame rate (and enable the segment pins)
  - Configure voltage requirements – including enabling the built-in Charge Pump, if used.
  - Set COM pins (not required for all LCD controllers, but necessary for the 'FR4133 since any LCD pin can be used as a COM line.
  - Finally, turn on the LCD_E controller
- LCD_showChar() function
  - This function displays a character given a character/digit and position within the display
- LCD_displayNumber()
  - This function takes a numerical value (using the long data type) and displays it on the LCD.
  - If the value isn't a number, the function displays "ERROR".
  - Also, the value is displayed in a right-justified fashion.
  - We developed this function for use in the `lab_12b_persistence` exercise.

Now, on to the edits for this file… we've left a few items for you to fill-in, based upon the earlier worksheet questions.

14. Fill in the details for the two functions which assign Lx pins to the LCD Controller.

Refer back question #2 (on page 12-26).

15. Write in the two functions needed to clear the LCD memory.

Refer back question #4 (on page 12-28).

16. What happens if you set the COM pins and cleared the memory in reverse order?

If you're not quite sure, Question #5 (on page 12-28) should help. That is, thinking about where the COM bits are stored .

_____

_____

main.c (Some edits required)

Main Edits

Only edits here are to fill in the details for three missing functions.

17. Fill in the function that sets the heart to display.

Refer back question #7 (on page 12-28).

18. How do we clear some LCD memory location? Fix that line of the file.

Refer back question #9 (on page 12-29).

19. Finally, complete the function which turns on blinking by switching back-and-forth between memories.

Refer back question #11 (on page 12-29).

20. Build your code and fix any typos and errors.

Step and Observe

21. Launch the debugger to load your code into the 'FR4133.

22. Set a breakpoint on the first line of code where we begin manipulating the display.

Up until this point, the code is the same as it was in Lab6a – the only difference being that we've initialized the LCD rather than a Timer.

```
31
32      // Initialize LCD
33      LCD_init();
34
35 // ------------------------------------------------------------------------------
36 // Single-step through this code to see how each line affects the LCD
37 // ------------------------------------------------------------------------------
38
39      // Set the LCD's "Heart" symbol
40      LCD_E_setMemory(LCD_E_BASE, LCD_E_MEMORY_BLINKINGMEMORY_12, LCD_HEART);      // "LC
41
42      // Set the LCD's "Timer" symbole
43      LCD_E_setMemory(LCD_E_BASE, LCD_E_MEMORY_BLINKINGMEMORY_12, LCD_TIMER);      // He
```

23. Stepping over the first LCD_E_setMemory() function, you should see the "Heart" appear on the LCD display.

Did the heart appear?_____

24. The next stepover should display the Timer symbol.

Did the Timer appear? _____

Is the Heart still there? _____

Refer back to Question #8 (on page 12-29). Was your prediction correct?

25. The next three Step-overs demonstrate the 'update', 'clear' and 'toggle' memory functions.

Verify they worked as expected.

26. The next step takes us back to the 'setMemory' function.

```
// Set both the "Heart" and "Timer" symbols
LCD_E_setMemory(LCD_E_BASE, LCD_E_MEMORY_BLINKINGMEMORY_12, LCD_HEART | LCD_TIMER);
```

What's different about this function this time? _____

Made you Blink

27. The next three function calls explore many of the blinking features.

```
// Let's explore the Blinking features

    // Blinks enaboled segments on the display (in this case, the Heart and Timer)
    // Notice how the blinking continues, even when the processor is halted at a breakpoint (or during single-stepping)
    // This is because the LCD controller is automatically handling the blinking -- no timers or interrupts are required!
    LCD_E_setBlinkingControl(LCD_E_BASE, LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64, LCD_E_BLINK_MODE_INDIVIDUAL_SEGMENTS);

    // Turn off the Timer symbol
    LCD_E_clearMemory(LCD_E_BASE, LCD_E_MEMORY_BLINKINGMEMORY_12, LCD_TIMER);              // Erases just the Timer symbol

    // Turns off the blinking feature
    LCD_E_setBlinkingControl(LCD_E_BASE, LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64, LCD_E_BLINK_MODE_DISABLED);
```

First we enable all the individual segments to flash. Does that mean every segment flashes? Or just the enabled segments?

_____

28. The final set of "blinking" function calls:

a) First has us populating – and using – the Blinking memory (LCDBM). This shows us how to switch back and forth (manually) between displaying either memory.

b) Next, we can start to see how to use these two memories to make custom (more complicated) blinking patterns. With the "Heart" enabled in one memory… and the "Timer" in the other…

Did the two icons alternate flashing? _____

Why would this solution be superior to using a timer ISR to tell you when to go switch what is being displayed?

_____

D.O.N.E.

29. Well, actually the last step in our program just spells DONE (not D.O.N.E.).

30. When you're all done playing and tweaking the code, please go ahead and close the project.

(Optional) Lab 12b – Displaying Persistent Data

We thought it would be fun to take `lab_09a_persistent` and write the count value to the LCD display, rather than just to the CCS console.

Go ahead and run this lab. You can probably tell right away that it's a mashup of `lab_09a_persistent` and `lab_12a_heart`.

Explore the code, build it and test it out.

If you're looking for more of a challenge, you could remove the printf() and/or LED feedback options. Alternatively, you could program the buttons (using the code from Lab 5) to reset the count value or increment it further.

Notes

Appendix

Initialization Worksheet

1. **From the MSP-EXP430FR4133 Launchpad User's Guide, what 'FR4133 LCD pins (Lx) need to be configured for use by the display. (Hint, look on page 15 of slau595.pdf.)**

   pins L0~L26 as well as

   pins L36~L39

2. **Complete the DriverLib function which sets these Lx pins as LCD pins.**

   (Hint: Look in the DriverLib User's Guide for the proper syntax.)

```
LCD_E_setPinAsLCDFunctionEx( LCD_E_BASE,
        LCD_E_SEGMENT_LINE_0          //starting pin
        LCD_E_SEGMENT_LINE_26         //ending pin
)
LCD_E_setPinAsLCDFunctionEx( LCD_E_BASE,
        LCD_E_SEGMENT_LINE_36         //starting pin
        LCD_E_SEGMENT_LINE_39         //ending pin
)
```

3. **How fast will the frame rate (F_{frame}) be given this initialization code?**

   This code is used to initialize the LCD controller.

```
LCD_E_initParam initParams = LCD_E_INIT_PARAM;
  initParams.clockDivider  = LCD_E_CLOCKDIVIDER_3;
  initParams.muxRate       = LCD_E_4_MUX;
  initParams.segments      = LCD_E_SEGMENTS_ENABLED;
LCD_E_init(LCD_E_BASE, &initParams);
```

F_{LCD} = 32768 / (3 + 1) * 16 = 512

F_{FRAME} = 512 / 8 = 64 Hz

Hints:

- The 'FR4133 User's Guide provides two formulas to help you calculate the frame rate.
  - As we discussed, the LCD frequency should be:

$$f_{LCD} = 2 \times mux \times f_{FRAME}$$

  - The LCD frequency can also be calculated with this expression:

$$f_{LCD} = \frac{f_{SOURCE}}{(LCDDIVx + 1) \times MUXDIVDER}$$

4. **Write two lines of code to clear all the LCD Memory.**

```
LCD_E_clearAllMemory(LCD_E_BASE);
LCD_E_clearAllBlinkingMemory(LCD_E_BASE);
```

5. **Which bits are set by these 4 lines of code?**

```
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_0, LCD_E_MEMORY_COM0);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_1, LCD_E_MEMORY_COM1);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_2, LCD_E_MEMORY_COM2);
LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_3, LCD_E_MEMORY_COM3);
```

These functions tell LCD_E which (Lx) pins to use for the common (COM) signals. Where is this information stored? (That is, what gets altered by this code?)

The COM pin assignments is stored in the LCD memory

Warning – if you clear the LCD, you erase these assignments

Runtime Worksheet

6. **Which address/bit controls each of the following segments? Fill out the table.**

Just to get you started, we added the Antenna symbol to the table.

Symbol	Memory Location	Bit Location	Hex Value
Antenna (ANT)	LCDM9	2	0x04
Heart (HRT)	LCDM12	2	0x04
Timer (TMR)	LCDM12	3	0x08

7. **Write the line of code that sets (i.e. turns on) the "Heart" segment.**

```
LCD_E_setMemory( LCD_E_BASE,
    LCD_E_MEMORY_BLINKINGMEMORY_12,   // Location
                 0x08                 // Mask (hex)
);
```

8. **What happens if we set (turn on) the HRT symbol, then set the TMR symbol?**

   Will they both be enabled, or will the second one replace the first one?

   Only one will be "on", as the "setMemory" function overwrites the memory location

9. **What function lets us clear one symbol without affecting another controlled by the same memory location (LDCMx)?**

   LCD_E_clearMemory

   Complete the function to clear the Timer (TMR) symbol.

```
LCD_E__clearMemory_____________( LCD_E_BASE,
            LCD_E_MEMORY_BLINKINGMEMORY_12,   // Location
            0x8 );                            // Mask (hex)
```

10. **What's the greatest advantage to the automatic blinking features of the 'FR4133?**

    While it's easy to use, the greatest advantage is Ultra Low-Power. You get these advantages, even when the device is sleeping in LPM3.5 mode!

11. **Finish the following line of code so that it enables the LCD controller's blinking feature – switching between both banks of memory.**

```
LCD_E__setBlinkingControl_____( LCD_E_BASE,
            LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64,
      LCD_E_BLINK_MODE_SWITCHING_BETWEEN_DISPLAY_CONTENTS );
```

16. **What happens if you set the COM pins and cleared the memory in reverse order?**

    If you're not quite sure, Question #5 (on page 12-28) should help. That is, thinking about where the COM bits are stored.

    Warning – if you clear the LCD memory, this will erase the COM pin assignments

23. Stepping over the first LCD_E_setMemory() function, you should see the "Heart" appear on the LCD display.

Did the heart appear? Yes

24. The next stepover should display the Timer symbol.

Did the Timer appear? Yes

Is the Heart still there? No

What's different about this function this time? Sets both segments at the same time

Made you Blink

27. The next three function calls explore many of the blinking features.

```
// Let's explore the Blinking features

    // Blinks enaboled segments on the display (in this case, the Heart and Timer)
    // Notice how the blinking continues, even when the processor is halted at a breakpoint (or during single-stepping)
    // This is because the LCD controller is automatically handling the blinking -- no timers or interrupts are required!
    LCD_E_setBlinkingControl(LCD_E_BASE, LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64, LCD_E_BLINK_MODE_INDIVIDUAL_SEGMENTS);

    // Turn off the Timer symbol
    LCD_E_clearMemory(LCD_E_BASE, LCD_E_MEMORY_BLINKINGMEMORY_12, LCD_TIMER);            // Erases just the Timer symbol

    // Turns off the blinking feature
    LCD_E_setBlinkingControl(LCD_E_BASE, LCD_E_BLINK_FREQ_CLOCK_PRESCALAR_64, LCD_E_BLINK_MODE_DISABLED);
```

First we enable all the individual segments to flash. Does that mean every segment flashes? Or just the enabled segments?

Just the enabled ones

28. The final set of "blinking" function calls:

a) First has us populating – and using – the Blinking memory (LCDBM). This shows us how to switch back and forth (manually) between displaying either memory.

b) Next, we can start to see how to use these two memories to make custom (more complicated) blinking patterns. With the "Heart" enabled in one memory… and the "Timer" in the other…

Did the two icons alternate flashing? Yes

Why would this solution be superior to using a timer ISR to tell you when to go switch what is being displayed?

Lower CPU overhead; more precise timing; MUCH, MUCH lower power