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 for LCD experimentation.

Learning Objectives

- Liquid Crystal Displays (LCD)
  - How do LCD’s Work?
- Basic Control of an LCD (Static)
- Using LCD’s with More Segments (Muxed)
- LCD Control Options
- Implementing Display with ‘FR4133 LCD_E
- Lab Exercise

Chapter Topics

Using Segmented Displays (LCD) ........................................................................................................... 12-1
   For More Information on LCD’s ........................................................................................................ 12-2
   Liquid Crystal Displays (LCD) ........................................................................................................... 12-3
      How do LCD’s Work? .................................................................................................................... 12-4
   Basic Control of an LCD (Static) ..................................................................................................... 12-7
      Using LCD’s with More Segments (Muxed) .................................................................................. 12-10
      Static vs Muxed ............................................................................................................................ 12-10
      Muxed Control Signals .................................................................................................................. 12-11
   LCD Control Options ....................................................................................................................... 12-14
      Bit Banging Display ....................................................................................................................... 12-14
      Displays with Built-in Drivers ..................................................................................................... 12-15
      MSP430 LCD Peripherals ............................................................................................................. 12-15
   Implementing Display with ‘FR4133 LCD_E ............................................................................... 12-16
      Choose Display and Pin Layout .................................................................................................... 12-16
      LCD Init Code ............................................................................................................................... 12-18
      Controlling Segments ................................................................................................................... 12-19
      Dual Memories & Blinking ............................................................................................................ 12-23
   Lab Exercise ........................................................................................................................................ 12-25

For More Information on LCD’s

For More Information

◆ LCD app note: www.ti.com/lit/pdf/slaa654
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](http://www.pcmag.com/encyclopedia/term/60488/lcd-types)

MSP430 + LCD Examples

- A/C remote control
- Low-power LCD hand held
- Healthcare
- Blood Glucose Meter
- Industrial
- Water Meter
- Consumer
- One-Time-Password Token
- Commercial
- E-Shelf Label
- Remotes

MSP430 Design Workshop - Using Segmented Displays (LCD) 12 - 3
How do LCD’s Work?

**Polarized Light**

Nonpolarized light vibrates in all directions.

Only one direction passes through first polarizer.

And that component cannot pass through a second polarizer oriented at 90°.

**Liquid Crystals Twist Light**

Twisted liquid crystals "bend" light 90°.

Letting polarized light pass through.
Liquid Crystal Displays (LCD)

**Electric Charge Untwists Crystals**

Twisted Liquid Crystals “bend” light 90°

Letting polarized light pass through

Applying an electric charge untwists liquid crystals ... therefore light is blocked

**Properties of “Electric Charge”**

- 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

- 10%
- 50%
- 90%

$V_{rms}$ (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”:
  - $V_{BMS}$ 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...

<table>
<thead>
<tr>
<th>SEG3</th>
<th>SEG4</th>
<th>COM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frame Rate

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

Period = 1 Frame = 1/FrameRate
Frame Rate = How fast you want to clock each segment (usually 30-60 Hz)

No DC Voltages!

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

Period = 1 Frame = 1/FrameRate
Frame Rate = How fast you want to clock each segment (usually 30-60 Hz rate)
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”

\[
\begin{align*}
\text{SEG}_3\text{RMS} &= \text{COM}-\text{SEG}_3 \\
\text{SEG}_4\text{RMS} &= \text{COM}-\text{SEG}_4
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>Static</th>
<th>Multiplexed</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM</td>
<td>COM0</td>
</tr>
<tr>
<td>8</td>
<td>COM1</td>
</tr>
<tr>
<td>9 pins = 1 COM + 8 SEG</td>
<td>6 pins = 2 COM + 4 SEG</td>
</tr>
</tbody>
</table>

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

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. $V_1$, $V_3$, $V_5$)
- 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’
Using LCD’s with More Segments (Muxed)

**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

**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
**Biasing**

- Bias voltages can be generated internally or externally (V2 to V5)
  - Independent of VLCD 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

**Contrast Control**

- Charge pump gives software selectable VLCD
- Changing VLCD 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 VLCD
- External contrast control also an option

**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

\[
F_{LCD} = 2 \times MUX \times F_{FRAME}
\]
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](http://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

(TI Confidential – NDA Restrictions)
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 store current image in MSP430 RAM/FRAM
- Sharp LCD Boosterpack
  [www.ti.com/tool/430boost-sharp96](http://www.ti.com/tool/430boost-sharp96)

MSP430 LCD Peripherals

<table>
<thead>
<tr>
<th>Parameter</th>
<th>LCD</th>
<th>LCD_A</th>
<th>LCD_B</th>
<th>LCD_C</th>
<th>LCD_E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Segments</td>
<td>128/4-MUX</td>
<td>160/4-MUX</td>
<td>160/4-MUX</td>
<td>320/8-MUX</td>
<td>448/8-MUX</td>
</tr>
<tr>
<td>Number of LCD Pins</td>
<td>up to 4x46</td>
<td>up to 4 x 50 or 8 x 46</td>
<td>up to 4 x 50 or 8 x 46</td>
<td>up to 4 x 50 or 8 x 56</td>
<td></td>
</tr>
<tr>
<td>Segment functionality against port pin selection</td>
<td>Minimum is group of 16</td>
<td>Selection done in groups of 4 segments</td>
<td>Individual selection can be done</td>
<td>Individual selection can be done</td>
<td>Individual selection can be done</td>
</tr>
<tr>
<td>COM/SEG Pin Assignments</td>
<td>COM Fixed</td>
<td>COM Fixed</td>
<td>COM Fixed</td>
<td>COM Fixed</td>
<td>Any LCD pin</td>
</tr>
<tr>
<td>LCD Clock selection</td>
<td>ACLK</td>
<td>ACLK</td>
<td>ACLK, VLO</td>
<td>ACLK, VLO</td>
<td>XT1, ACLK, VLO</td>
</tr>
<tr>
<td>Interrupt capabilities</td>
<td>NO</td>
<td>NO</td>
<td>YES (4 sources)</td>
<td>YES (4 sources)</td>
<td>YES (3 sources)</td>
</tr>
<tr>
<td>Individual segment blinking capabilities</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Prog. blinking frequency</td>
<td>N/A</td>
<td>N/A</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Dual memory display</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Works in LPM3.5</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Implementing Display with ‘FR4133 LCD_E

Choose Display and Pin Layout

Select a Segmented LCD

- This display is found on the MSP-EXP430FR4133 Launchpad
- LCD datasheet defines:
  - Driving Voltage = 3.0V
  - Duty cycle = \( \frac{1}{4} \) (which implies 4-mux)
  - BIAS Voltages = 1/3
- Of display’s 38 pins: 4 COM, 27 SEG (and 7 unused)
Implementing Display with 'FR4133 LCD_E

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)

- 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

MSP430FR4133
Just connect Lx pins to LCD, no specific connection required!
**Implementing Display with FR4133 LCD**

### 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)

```c
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)

```c
// Initialize 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

<table>
<thead>
<tr>
<th>Segments</th>
<th>COM1</th>
<th>COM2</th>
<th>COM3</th>
<th>COM4</th>
<th>COM1</th>
<th>COM2</th>
<th>COM3</th>
<th>COM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDM19</td>
<td>LCD Pin (L39)</td>
<td>LCD Pin (L38)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCDM1</td>
<td>LCD Pin (L3)</td>
<td>LCD Pin (L2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCDM0</td>
<td>LCD Pin (L1)</td>
<td>LCD Pin (L0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Implementing Display with 'FR4133 LCD_E**

- 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

---

**LCD Memory**

<table>
<thead>
<tr>
<th>Segments</th>
<th>COM1</th>
<th>COM2</th>
<th>COM3</th>
<th>COM4</th>
<th>COM1</th>
<th>COM2</th>
<th>COM3</th>
<th>COM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCDM19</td>
<td>LCD Pin (L39)</td>
<td>LCD Pin (L38)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCDM1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LCD Pin (L2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCDM0</td>
<td>LCD Pin (L1)</td>
<td>LCD Pin (L0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**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

---

**Table 4. LCD to MSP430 Connections**
Implementing Display with 'FR4133 LCD_E

Choosing Layout for Easier Software

- 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

```c
const char digit[10] = {
    0xFC,  /* "0" */
    0x60,  /* "1" */
    0xDB,  /* "2" */
    0xF3,  /* "3" */
    0x67,  /* "4" */
    0xB7,  /* "5" */
    ...
};
```
Implementing Display with ‘FR4133 LCD_E

- **Controlling LCD_E with DriverLib**
  ```
  #define pos1 4       // Position 1 (A1) is at LCDM4
  LCD_E_setMemory( LCD_E_BASE, pos1, digit[2] );
  ```

- **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 LCDMx register
  - `LCD_E_updateMemory()` Sets the specified bits of LCDMx (LCDMx |= mask)
Dual Memories & Blinking

Two LCD Memories

- 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

```c
#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

MSP430 Design Workshop - Using Segmented Displays (LCD)
Notes:
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

Chapter Topics

Using Segmented Displays (LCD) .............................................................. 12-25
Lab 12a – A Launchpad with Heart ............................................................. 12-26
  Initialization Worksheet ......................................................................... 12-26
  Runtime Worksheet ............................................................................... 12-28
  Lab File Management ........................................................................... 12-30
  Examine and Tweak LCD Files ............................................................... 12-31
(Optional) Lab 12b – Displaying Persistent Data ...................................... 12-35
  Appendix ............................................................................................. 12-36
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 segments 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\textsubscript{frame}) be given this initialization code?**

This code is used to initialize the LCD controller.

```c
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 initialization 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 $F_{\text{lcd}}$, which in turn affects $F_{\text{frame}}$.

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.

$$F_{\text{LCD}} = \phantom{\text{_______}}$$

$$F_{\text{FRAME}} = \phantom{\text{_______}}$$

**Hints:**

- The ‘FR4133 User’s Guide provides two formulas to help you calculate the frame rate.
  
  - As we discussed, the LCD frequency should be:
    
    $$f_{\text{LCD}} = 2 \times \text{mux} \times f_{\text{FRAME}}$$
    
  - The LCD frequency can also be calculated with this expression:
    
    $$f_{\text{LCD}} = \frac{f_{\text{SOURCE}}}{(\text{LCDDIVx} + 1) \times \text{MUXDIVIDER}}$$
    
  - The code snippet in this lab step provides us the $f_{\text{source}}$ 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?

   \begin{verbatim}
   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);
   \end{verbatim}

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?)

_________________________________________________________________________
_________________________________________________________________________

\section*{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.

\begin{verbatim}
Symbol  | Memory Location | Bit Location | Hex Value
\hline
Antenna (ANT) | LCDM9 | 2 | 0x04
Heart (HRT) | | | 
Timer (TMR) | | | 
\end{verbatim}

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

   \begin{verbatim}
   LCD_E____________________________( LCD_E_BASE,
                                          __________________________, // Location
                                          ___________________________ // Mask (hex)
   );
   \end{verbatim}
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.

```c
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.

```c
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`.

```c
#define LCD_HEART (0x0004)
#define LCD_TIMER (0x0008)
#define pos1 4 /* Digit A1 - L4 */
#define pos2 6 /* Digit A2 - L6 */
#define pos3 8 /* Digit A3 - L8 */
#define pos4 10 /* Digit A4 - L10 */
#define pos5 2 /* Digit A5 - L2 */
#define pos6 18 /* Digit A6 - L18 */
#define digit[10][2]; // Segment values
#define alphabetBig[26][2]; // Segment values

void LCD_init(void);
void LCD_showChar( char, int );
void LCD_displayNumber( unsigned long );
```
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 throughout 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.

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 enabled 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?

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 slau695.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)

   ```c
   void LCD_E_setPinAsLCDFunctionEx( LCD_E_BASE, LCD_E_SEGMENT_LINE_0, LCD_E_SEGMENT_LINE_26 );
   // 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:

   ```c
   LCD_E_initParam initParams = LCD_E_INIT_PARAM;
   initParams.clockDivider = LCD_E_CLOCKDIVIDER_5;
   initParams.muxRate = LCD_E_MUXRATE_4;
   initParams.segments = LCD_E_SEGMENTS_ENABLED;
   LCD_E_init( LCD_E_BASE, &initParams );
   ```

   \[ F_{\text{LCD}} = \frac{32768}{3 + 1} \times 16 = 512 \]

   \[ F_{\text{FRAME}} = \frac{512}{8} = 64 \text{ Hz} \]

   Hints:
   - The FR4133 User’s Guide provides two formulas to help you calculate the frame rate.
     - As we discussed, the LCD frequency should be:
       \[ f_{\text{LCD}} = 2 \times \text{mux} \times f_{\text{FRAME}} \]
     - The LCD frequency can also be calculated with this expression:
       \[ f_{\text{LCD}} = \frac{f_{\text{SOURCE}}}{(\text{LCDDIVx} + 1) \times \text{MUXDIVDER}} \]
Appendix

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

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

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

   ```c
   LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_0, LCD_E_MEMORY_COM);
   LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_1, LCD_E_MEMORY_COM);
   LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_2, LCD_E_MEMORY_COM);
   LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_3, LCD_E_MEMORY_COM);
   ```

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.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Memory Location</th>
<th>Bit Location</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antenna (ANT)</td>
<td>LCDM9</td>
<td>2</td>
<td>0x04</td>
</tr>
<tr>
<td>Heart (HRT)</td>
<td>LCDM12</td>
<td>2</td>
<td>0x04</td>
</tr>
<tr>
<td>Timer (TMR)</td>
<td>LCDM12</td>
<td>3</td>
<td>0x08</td>
</tr>
</tbody>
</table>

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

   ```c
   LCD_E_setMemory(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 (LCDs)?

**LCD_E_clearMemory**

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

```c
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.

```c
LCD_E_setBlinkingControl (LCD_E_BASE,
    LCD_E_BLINK_FREQ_CLOCK_FRESHSCALE_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 #8 (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.

```c
// let's explore the blinking features:

// Blink on enabled segments only on the display (in this case, the Heart and Timer)
// Note how the blinking continues, even when the processor is halted or is doing single-stepping
// This is because the LCD controller is automatically handling the blinking -- no timers or interrupts are required!

// Turn off the Timer symbol
LCD_E_setSymbol(LCD_E_SET_SEGMENT_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L);

// Turn off the blinking feature
LCD_E_setSymbol(LCD_E_SET_SEGMENT_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L, LCD_E_SET_SYMBOL_L);
```

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 populate -- 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 switch what is being displayed?

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