Driving to FRDM-KL43Z LCD Display

Nov 18, 2018

Purpose:

This document is intended to be a help to students in EE342 in using the small 4 character LCD display that is found on the FRDM-KL43Z microcontroller demo boards being used for the class. The KL-43Z microcontroller family is fairly unique in including an LCD driver. That is, the device directly drives an LCD display, generating the needed high Voltages on chip, and driving the various segments and digits of the display. Indeed 64 pins (aside from the Voltage supplies) are used to drive the "front" and "back" planes of signals. (The point is, there is no need for a separate LCD driver chip or sophisticated displays using such drivers if you have a KL43Z.) However, using the LCD is tricky. The Kinetis Softare Development Kit (SDK) driver example is useful, but does not go as far as needed to use the display for practical purposes. This document is intended to fill that gap.

Needed files and initialization:

To use the LCD display, a project needs to include (in the drivers folder) the fsl_slcd.h and fsl_slcd.c files. To get to the functions, a "# include "fsl_slcd.h" is needed. In my example to come, that is in the file containing the main program, where other includes are. This gives access to several structures and functions that are needed.

Initialization is complicated. The clock used to drive the LCD display needs to be selected (The internal 32KHz clock is the default), voltage levels and multiplexing modes and pin setups are needed. Fortunately for us, all that is taken care of by some code that we can borrow from the LCD driver example, which proves to be sufficient for our needs.

So, there are some essential global variables that are needed (though some of these are locals to main() in the LCD driver example). These are shown below:

```
/* LCD variables */
    slcd_config_t lcdConfig;
    slcd_clock_config_t clkConfig =
    {
        kSLCD_AlternateClk1,
        kSLCD_AltClkDivFactor256,
        kSLCD_ClkPrescaler01,
        false
    };
```

As with many other features, there is a configuration structure called "slcd_config_t". We will need one of those (I named it lcdConfig). There is also a special LCD clock configuration structure needed to initialize the LCD.

The initialization needs to be done from main() before entering the main loop (or doing anything else with the LCD display). The code fragment (mostly copied from the driver example with minor modifications) is shown below. This would be a good candidate to put in a separate initialization function. You can see the configuration structure "lcdConfig" being used for most of this.

```
/* Enable the MCGIRCLK (for LCD display)*/
MCG->C1 |= MCG_C1_IRCLKEN_MASK;
```

```
    /* SLCD get default configure. */
    /*
     * config.displayMode = kSLCD_NormalMode;
     * config.powerSupply = kSLCD_InternalVll3UseChargePump;
     * config.voltageTrim = kSLCD_RegulatedVolatgeTrim08;
     * config.lowPowerBehavior = kSLCD_EnabledInWaitStop;
     * config.frameFreqIntEnable = false;
     * config.faultConfig = NULL;
     */
    SLCD_GetDefaultConfig(&lcdConfig);
    lcdConfig.clkConfig = &clkConfig;
    lcdConfig.loadAdjust = kSLCD_HighLoadOrSlowestClkSrc;
    lcdConfig.dutyCycle = kSLCD_1Div4DutyCycle;
    lcdConfig.slcdLowPinEnabled = 0x0d10c000U;  /* LCD_P27/26/24/20 -> b27/26/24/20 =
1. */
    lcdConfig.slcdHighPinEnabled = 0x18001d00U; /* LCD_P44/43/42/40 -> b12/11/10/8 =
1. */
    lcdConfig.backPlaneLowPin = 0x0000c000U;    /* LCD_P15/P14 -> b15/b14 = 1. */
    lcdConfig.backPlaneHighPin = 0x18000000U;   /* LCD_P60/P59 -> b28/27 = 1. */
    lcdConfig.faultConfig = NULL;
    /* SLCD Initialize. */
    SLCD_Init(LCD, &lcdConfig);
```

There is actually a lot going on in this. We have a 4 digit display that needs to be driven with a ¼ duty cycle. The pins for the "backplane" (corresponding to the particular digits) need to be enabled. This is code that I have not actually parsed and understood given the hardware design, but it does work, and just needs to be done this once upon initialization. After setting all the needed data in the LCD control structure, the LCD is actually initialized by the call to SLCD_Init(LCD, &lcdConfig) which sets the LCD display at LCD (the address of the display in the memory map) to the configuration selected, which is designed for the FRDM-KL43Z board, and turns on the right signals and makes the right settings for the board we have.

Making it Say Something:

Now that the LCD display is initialized, characters can be displayed by writing "waveforms" to registers corresponding to the pins driving the "front plane" of the display, that is, the characters. Because our LCD display has 4 way multiplexing, a pin is needed for each 4 segments. The mapping is as follows:

Pin 20: segments f,g,e,and d of character 1 (leftmost)
Pin 24: segments a,b,c,and h of character 1
Pin 26: segments f,g,e,and d of character 2
Pin 27: segments a,b,c,and h of character 2
Pin 40: segments f,g,e,and d of character 3
Pin 41: segments a,b,c,and h of character 3
Pin 43: segments f,g,e,and d of character 4
Pin 44: segments a,b,c,and h of character 4 (rightmost)

Within each 8 bits of "waveform data" for a given pin, the bits are ordered (left to right) as indicated above, either f,g,e,d or a,b,d,h (decimal), in the low order 4 bits of the data. (It may be better to put the same data in both 4 bit halves. Some examples in the manual seemed to show that. I couldn't tell that it mattered.) To turn on the backplane (digits) and set the waveforms, the following code example sets the display to "HELP".

```
/* Set SLCD front plane phase to show: “HELP” . */
/* Set SLCD back plane phase. */
SLCD_SetBackPlanePhase(LCD, 59, kSLCD_PhaseAActivate);/* SLCD COM1 --- LCD_P59. */
SLCD_SetBackPlanePhase(LCD, 60, kSLCD_PhaseBActivate);/* SLCD COM2 --- LCD_P60. */
SLCD_SetBackPlanePhase(LCD, 14, kSLCD_PhaseCActivate);/* SLCD COM3 --- LCD_P14. */
SLCD_SetBackPlanePhase(LCD, 15, kSLCD_PhaseDActivate);/* SLCD COM4 --- LCD_P15. */
/* Set SLCD front plane phase to show. */
SLCD_SetFrontPlaneSegments(LCD, 20,/*waveForm*/0xEE); /* SLCD P05 --- LCD_P20. */
SLCD_SetFrontPlaneSegments(LCD, 24,/*waveForm*/0x66); /* SLCD P06 --- LCD_P24. */
SLCD_SetFrontPlaneSegments(LCD, 26,/*waveForm*/0xFF); /* SLCD P07 --- LCD_P26. */
SLCD_SetFrontPlaneSegments(LCD, 27,/*waveForm*/0x88); /* SLCD P08 --- LCD_P27. */
SLCD_SetFrontPlaneSegments(LCD, 40,/*waveForm*/0xBB); /* SLCD P09 --- LCD_P40. */
SLCD_SetFrontPlaneSegments(LCD, 42,/*waveForm*/0x00); /* SLCD P10 --- LCD_P42. */
SLCD_SetFrontPlaneSegments(LCD, 43,/*waveForm*/0xEE); /* SLCD P11 --- LCD_P43. */
SLCD_SetFrontPlaneSegments(LCD, 44,/*waveForm*/0xCC); /* SLCD P12 --- LCD_P44. */
/* Starts SLCD display. */
SLCD_StartDisplay(LCD);
PRINTF("\r\nSLCD Display HELP\r\n");
```

The character codes to display ‘H’ are hexadecimal E and 6; for ‘E’ they are F and 8, and so forth for ‘L’ and ‘P’. After initialization and executing these function calls the display indeed shows “HELP”. Note that a “1” turns a segment on, a “0” turns it off. The LCD display does NOT need for the programmer to do the multiplexing or refreshing registers to just let the display show a fixed message; that’s all done by the LCD display driver hardware (which is not simple).

Displaying Numbers:

You may have various things you might want the LCD display to show, but this example makes use of what I have already functional: the clock. The trick is to take an integer and convert it into the data to drive the 8 pins as shown above. To aid in doing that, I figured out the 7 segment codes for numerals 0 to 9, ordered as needed for the pairs of pins. These codes were put in a pair of global arrays. (Someone could do this for the whole ASCII character set. I didn’t.) I used a binary representation, and only used the lower half of each 8 bits of waveform.

```
const char sevSegFGED[]={0b1011,0b0000,0b0111,0b0101,0b1100,0b1101,0b1111,0b0000,0b1111,0b1100};
const char sevSegABCH[]={0b1110,0b0110,0b1100,0b1110,0b0110,0b1010,0b1010,0b1110,0b1110,0b1110};
```

So, here is how I displayed the time inside the main loop of the program:

```
/* LCD show time mm.ss*/
RTC_GetDatetime(RTC, &datE);
/* Set SLCD front plane phase to show time. */
SLCD_StopDisplay(LCD);
j=datE.minute/10;
SLCD_SetFrontPlaneSegments(LCD,20,sevSegFGED[j]);/* SLCD P05 --- LCD_P20. */
SLCD_SetFrontPlaneSegments(LCD,24,sevSegABCH[j]);/* SLCD P06 --- LCD_P24. */
j=datE.minute-j*10;
SLCD_SetFrontPlaneSegments(LCD,26,sevSegFGED[j]);/* SLCD P07 --- LCD_P26. */
SLCD_SetFrontPlaneSegments(LCD,27,sevSegABCH[j]);/* SLCD P08 --- LCD_P27. */
j=datE.second/10;
SLCD_SetFrontPlaneSegments(LCD,40,sevSegFGED[j]);/* SLCD P09 --- LCD_P40. */
SLCD_SetFrontPlaneSegments(LCD,42,sevSegABCH[j]);/* SLCD P10 --- LCD_P42. */
j=datE.second-j*10;
SLCD_SetFrontPlaneSegments(LCD,43,sevSegFGED[j]);/* SLCD P11 --- LCD_P43. */
SLCD_SetFrontPlaneSegments(LCD,44,sevSegABCH[j]);/* SLCD P12 --- LCD_P44. */
/* Starts SLCD display. */
SLCD_StartDisplay(LCD);
```

As the main program is right now, the "blocking" version of getchar() inside the terminal_process function means that the time is updated only when a character comes in from the terminal. A better place to do this is perhaps as a function called from the interrupt service routine, so that it will update every second. (It would probably be a good idea to turn on the decimal (segment h) between the two characters for minutes and those for seconds, too.

Uses:

The LCD display is small enough that it is probably not what you want to display critical operational data like track state; for that LED's are easier to interpret instantly. It may be useful though to convey other data about the operation of your system. For example, it could indicate automatic mode, or messages received from adjacent computers. Or, you could have state data that now comes out in the terminal sent to the LCD display instead (such as track Voltage or optical sensor data).

Conclusion:

This should be enough to help you get started using the LCD display. OIt really isn't too hard once the initialization is done, and all you need to do for each 4 character message is a set of 8 function calls. (Hmm, how about a function that takes a 4 character string and puts it into the LCD display? That would be handy intellectual property in the context of our class! Maybe someone would pay a chocolate bar for it.)