```
/*
* DOG_LCD_C_driver.c
*
* Created: 3/19/2022 8:09:13 PM
* Author : jason
*/


/* modified p8_1.c:
*
* SERCOM1 is configured as SPI with hardware Slave Select.
* Clock rate is set to 2 MHz, half of the main clock.
* Polarity/Phase are 1, 1 to communicate with DOG LCD
*
* PA16  PAD0  MOSI
* PA17  PAD1  SCK
* PA18  PAD2  /SS       hardware controlled
* PA19  PAD3  MISO
*
* PB06  RS LCD  // Register select for LCD
*
* Tested with Atmel Studio 7
*/

#include <avr/io.h>

#include <stdio.h>

// Display buffer for DOG LCD using sprintf()
char dsp_buff1[17];
char dsp_buff2[17];
char dsp_buff3[17];

void lcd_spi_transmit_CMD (unsigned char cmd);
void lcd_spi_transmit_DATA (unsigned char cmd);
void init_spi_lcd (void);
void init_lcd_dog (void);
void delay_40mS(void);
void delay_30uS(void);
void update_lcd_dog(void);

//unsigned char* ARRAY_PORT_PINCFG0 = (unsigned char*)&SPI0_CTRLA;
//unsigned char* ARRAY_PORT_PMUX0 = (unsigned char*)&REG_PORT_PMUX0;

//////////////////////// Driver Functions /////////////////////////

void lcd_spi_transmit_CMD (unsigned char cmd) {
    //Poll until ready to send the command
    //while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT &= ~PIN0_bm;  //Clear PC0 = RS = 0 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

    //while(!(REG_SERCOM1_SPI_INTFLAG & 1)) {}    // wait until Tx ready
    //REG_PORT_OUTCLR1 = 0x00000040;    // RS = 0 for command
    //  REG_PORT_OUTCLR0 = 0x00040000;  //assert slave select, not needed when MSSEN =
       1
    //REG_SERCOM1_SPI_DATA = cmd;       //send command
    //while(!(REG_SERCOM1_SPI_INTFLAG & 1)) {}    // wait until Tx ready
    //  REG_PORT_OUTSET0 = 0x00040000;  //unassert slave select, not needed when MSSEN
       = 1
}

void lcd_spi_transmit_DATA (unsigned char cmd) {
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTC_OUT |= PIN0_bm;   //PC0 = RS = 1 = command
    PORTA_OUT &= ~PIN7_bm;  //clear PA7 = /SS = selected
    SPI0_DATA = cmd;
    //Poll until ready to send the command
    while(!(SPI0_INTFLAGS & SPI_IF_bm)){}
    PORTA_OUT |= PIN7_bm;   //clear PA7 = /SS = selected

    //while(!(REG_SERCOM1_SPI_INTFLAG & 1)) {}    // wait until Tx ready
    //REG_PORT_OUTSET1 = 0x00000040;    // RS = 1 for data
    //  REG_PORT_OUTCLR0 = 0x00040000;  //assert slave select, not needed when MSSEN =
       1
    //REG_SERCOM1_SPI_DATA = cmd;       //send command
    //while(!(REG_SERCOM1_SPI_INTFLAG & 1)) {}    // wait until Tx ready
    //  REG_PORT_OUTSET0 = 0x00040000;  //unassert slave select, not needed when MSSEN
       = 1
}

void init_spi_lcd (void) {
    PORTA_DIR |= PIN4_bm | PIN6_bm | PIN7_bm | ~(PIN5_bm);  //Set MOSI, SCK and //SS
      as output while MISO as input

    PORTC_DIR |= PIN0_bm;  //Set RS of LCD as output

    SPI0_CTRLA |= SPI_ENABLE_bm | SPI_MASTER_bm; //Enable the SPI and make it in the
      Master Mode

    SPI0_CTRLB |= SPI_SSD_bm | SPI_MODE1_bm | SPI_MODE0_bm;   //Put the SPI with slave
       select (/SS) to be enabled and be in SPI Mode 3 (CPOL = 1 and CPHA = 1)

    //Wait to clears the IF flag in the INTFLAG meaning there no serial data yet to be
       transferred
    //while(SPI0_INTFLAGS & SPI_IF_bm){}
    PORTC_OUT &= ~PIN0_bm;  //PC0 = RS = 0 = command

    //REG_MCLK_AHBMASK |= 0x00000004;   /* APBC bus clock enabled by default */
    //REG_MCLK_APBCMASK |= 0x00000002;  /* SERCOM1 APBC bus clock enabled by default 
      */
    // Generic clock generator 0, enabled at reset @ 4MHz, is used for peripheral 
      clock

    //REG_GCLK_PCHCTRL19 = 0x00000040;  /* SERCOM1 core clock not enabled by default 
      */

    //ARRAY_PORT_PINCFG0[16] |= 1;    /* allow pmux to set PA16 pin configuration */
    //ARRAY_PORT_PINCFG0[17] |= 1;    /* allow pmux to set PA17 pin configuration */
    //ARRAY_PORT_PINCFG0[18] |= 1;    /* allow pmux to set PA18 pin configuration */
    //ARRAY_PORT_PINCFG0[19] |= 1;    /* allow pmux to set PA19 pin configuration */
    //ARRAY_PORT_PMUX0[8] = 0x22;     /* PA16 = MOSI, PA17 = SCK */
    //ARRAY_PORT_PMUX0[9] = 0x22;     /* PA18 = SS,   PA19 = MISO */

    //REG_SERCOM1_SPI_CTRLA = 1;              /* reset SERCOM1 */
    //while (REG_SERCOM1_SPI_CTRLA & 1) {}    /* wait for reset to complete */
    // Msb first, CPOL = 1, CPHA = 1
    //REG_SERCOM1_SPI_CTRLA = 0x3030000C;     /* MISO-3, MOSI-0, SCK-1, SS-2, SPI 
      master */
    //REG_SERCOM1_SPI_CTRLB = 0x00002000;     /* Master SS, 8-bit */
    // BAUD = 4MHz/(2 * 3.125 MHz) - 1 = -0.36 = 0, giving 2MHz
    //REG_SERCOM1_SPI_BAUD = 0;               /* SPI clock is 4MHz/2 = 2MzHz */
    //REG_SERCOM1_SPI_CTRLA |= 2;             /* enable SERCOM1 */

    //REG_PORT_DIRSET1 = 0x00000040;    // PB06 is output for RS of LCD
    //REG_PORT_OUTCLR1 = 0x00000040;    // RS = 0 for command

}


void init_lcd_dog (void) {

    init_spi_lcd();     //Initialize mcu for LCD SPI

    //start_dly_40ms:
    delay_40mS();    //startup delay.


    //func_set1:
    lcd_spi_transmit_CMD(0x39);   // sedn function set #1
    delay_30uS();   //delay for command to be processed


    //func_set2:
    lcd_spi_transmit_CMD(0x39); //send fuction set #2
    delay_30uS();   //delay for command to be processed


    //bias_set:
    lcd_spi_transmit_CMD(0x1E); //set bias value.
    delay_30uS();   //delay for command to be processed
```

```c
    //power_ctrl:
    lcd_spi_transmit_CMD(0x55); //~ 0x50 nominal for 5V
    //~ 0x55 for 3.3V (delicate adjustment).
    delay_30uS();   //delay for command to be processed


    //follower_ctrl:
    lcd_spi_transmit_CMD(0x6C); //follower mode on...
    delay_40mS();   //delay for command to be processed


    //contrast_set:
    lcd_spi_transmit_CMD(0x7F); //~ 77 for 5V, ~ 7F for 3.3V
    delay_30uS();   //delay for command to be processed


    //display_on:
    lcd_spi_transmit_CMD(0x0c); //display on, cursor off, blink off
    delay_30uS();   //delay for command to be processed


    //clr_display:
    lcd_spi_transmit_CMD(0x01); //clear display, cursor home
    delay_30uS();   //delay for command to be processed


    //entry_mode:
    lcd_spi_transmit_CMD(0x06); //clear display, cursor home
    delay_30uS();   //delay for command to be processed
}


void delay_40mS(void) {
    int i;
    for (int n = 40; n > 0; n--)
    for (i = 0; i < 800; i++)
    __asm("nop");
}

void delay_30uS(void) {
    int i;
    for (int n = 1; n > 0; n--)
    for (i = 0; i < 2; i++)
    __asm("nop");
}


// Updates the LCD display lines 1, 2, and 3, using the
// contents of dsp_buff_1, dsp_buff_2, and dsp_buff_3, respectively.
void update_lcd_dog(void) {
    init_spi_lcd();     //init SPI port for LCD.

    // send line 1 to the LCD module.
    lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff1[i]);
        delay_30uS();
    }

    // send line 2 to the LCD module.
    lcd_spi_transmit_CMD(0x90); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff2[i]);
        delay_30uS();
    }

    // send line 3 to the LCD module.
    lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
    delay_30uS();
    for (int i = 0; i < 16; i++) {
        lcd_spi_transmit_DATA(dsp_buff3[i]);
        delay_30uS();
    }
}
```