/**
 * @file   MKL43Z256xxx4_terminal.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL43Z4.h"
#include "fsl_debug_console.h"

```
#include "hcc_terminal.h"
#include "fsl_gpio.h"
#include "fsl_port.h"
#include "fsl_rtc.h"
#include "fsl_slcd.h"

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

/* Global time variables*/
    rtc_datetime_t datE;
    volatile bool myAlarm;
    rtc_config_t rtcConfig;

/* command to get the time */
static void cmd_time(char *param)
{
    param++;
    /* Get date time */
    RTC_GetDatetime(RTC, &datE);
    /* print default time */
    PRINTF("Current datetime: %d-%d-%d %2d:%2d:%2d\r\n", datE.year, datE.month,
            datE.day, datE.hour, datE.minute, datE.second);
}
/* the command to print the time */
static const command_t time_cmd = {
  "time", cmd_time, "Print the time"};

/* command to set the time; Format hh:mm:ss dd-mm-yy*/
static void cmd_settime(char *param)
{
    if(param[2]!=':'||param[5]!=':'){
        print("set time format error");
        print(param);
        print("\n\r");
        return;
    }
```

```
    datE.hour=(param[0]-'0')*10+param[1]-'0';
    datE.minute=(param[3]-'0')*10+param[4]-'0';
    datE.second=(param[6]-'0')*10+param[7]-'0';
    if(param[9]>='0'&&param[9]<='9')datE.day=(param[9]-'0')*10+param[10]-'0';
    if(param[11]=='-')datE.month=(param[12]-'0')*10+(param[13]-'0');
    if(param[14]=='-')datE.year=(param[15]-'0')*10+(param[16]-'0')+2000;
    RTC_StopTimer(RTC);
    RTC_SetDatetime(RTC, &datE);
    RTC_StartTimer(RTC);
}
/* the command to set the time */
static const command_t settime_cmd = {
  "settime", cmd_settime, "Set the time hh:mm:ss dd-mm-yy"};

/* command to set alarm */
static void cmd_alarm(char *param)
{
    uint32_t alarmtime;  /* alarm time in seconds from now*/
    uint32_t currSeconds;/* current time, becomes alarm time */
    if(param[0]=='0'){
        print("no parameter.\r\n");
        return;
    }
    alarmtime=param[0]-'0';
    if(param[1]!='\0'){
        alarmtime=alarmtime*10+(param[1]-'0');
    }
    if(param[2]!='\0'){
        alarmtime=alarmtime*10+(param[2]-'0');
    }
    currSeconds = RTC->TSR;
    /* Add alarm seconds to current time */
    currSeconds = currSeconds+alarmtime;
    /* Set alarm time in seconds */
    RTC->TAR = currSeconds;
    /* Get alarm time */
    RTC_GetAlarm(RTC, &datE);
    /* Print alarm time */
    PRINTF("Alarm will occur at: %d-%hd-%d %2d:%2d:%2d\r\n", datE.year,
            datE.month, datE.day, datE.hour, datE.minute, datE.second);
}
/* the command to set the alarm */
static const command_t alarm_cmd = {
  "alarm", cmd_alarm, "Set the alarm time for x seconds (1-3 digits)"};

/* command called function to modify LED state */
static void cmd_led(char *param)
{
    static char ledr=0,ledg=0;
    if(param[0]=='r'||param[0]=='R'){
        if(ledr==1){
```

```
            GPIO_WritePinOutput (GPIOE, 31, 1);
            //LED_RED_OFF();
            ledr=0;}
        else{
            GPIO_WritePinOutput(GPIOE, 31, 0);
            //LED_RED_ON();
            ledr=1;}
    }
    else if(param[0]=='g'||param[0]=='G'){
        if(ledg==1){
            //GPIO_WritePinOutput (GPIOD, 5, 1);
            LED_GREEN_OFF();
            ledg=0;}
        else{
            //GPIO_WritePinOutput(GPIOD, 5, 0);
            LED_GREEN_ON();
            ledg=1;}
    }
    else print("Invalid LED\r\n");
}

static void cmd_sayyes(char *param)
{
  param++;
  print("Yes\n\r");
}

/* the command to modify the LED state */
static const command_t led_cmd = {
  "led", cmd_led, "Toggle led R or G"};

static const command_t sayyes_cmd = {
  "yes", cmd_sayyes, "Says Yes"};

//New JBG dummy kbhit to initiate call to blocking PUTCH
int kbhit(void){
    return 1;}

/*
 * @brief   Application entry point.
 */
int main(void) {
    /* Force the counter to be placed into memory. */
    volatile static int i = 0 ;
    int j;
    /* LCD variables */
    uint8_t waveForm = 0;/* all segments on*/
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
```

```
/* Init FSL debug console. */
BOARD_InitDebugConsole();
BOARD_InitLEDs();
/* 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);
/* Set SLCD front plane phase to show: all segments on . */
waveForm = (kSLCD_PhaseAActivate | kSLCD_PhaseBActivate |
 kSLCD_PhaseCActivate | kSLCD_PhaseDActivate);
/* 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");

/* Init RTC */
/*
 * rtcConfig.wakeupSelect = false;
 * rtcConfig.updateMode = false;
 * rtcConfig.supervisorAccess = false;
 * rtcConfig.compensationInterval = 0;
 * rtcConfig.compensationTime = 0;
 */
RTC_GetDefaultConfig(&rtcConfig);
RTC_Init(RTC, &rtcConfig);
/* Select RTC clock source */
RTC_SetClockSource(RTC);
/* Set a start date time and start RT */
datE.year = 2018U;
datE.month = 11U;
datE.day = 13U;
datE.hour = 11U;
datE.minute = 0;
datE.second = 0;
/* RTC time counter has to be stopped before setting the date & time in
 the TSR register */
RTC_StopTimer(RTC);
/* Set RTC time to default */
RTC_SetDatetime(RTC, &datE);
/* Enable RTC alarm interrupt */
RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);
/* Enable at the NVIC */
EnableIRQ(RTC_IRQn);
/* Start the RTC time counter */
RTC_StartTimer(RTC);

PRINTF("Hello World\n");
PRINTF("Current datetime: %d-%d-%d %2d:%2d:%2d\r\n", datE.year, datE.month,
        datE.day, datE.hour, datE.minute, datE.second);
terminal_init(PUTCHAR, GETCHAR, kbhit);
(void)terminal_add_cmd((command_t*)&sayyes_cmd);
(void)terminal_add_cmd((command_t*)&led_cmd);
(void)terminal_add_cmd((command_t*)&time_cmd);
(void)terminal_add_cmd((command_t*)&settime_cmd);
```

```
    (void)terminal_add_cmd((command_t*)&alarm_cmd);
    /* Enter an infinite loop */
    while(1) {
        terminal_process();
        i++ ;
        /* Blink mode Display. */
        //if(i&1==0)SLCD_StartBlinkMode(LCD, kSLCD_BlankDisplayBlink,
         kSLCD_BlinkRate01);
        //else SLCD_StopBlinkMode(LCD);
        /* 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);

        if(myAlarm){
            print("ALARM went off after last character\n\r!");
            RTC_GetDatetime(RTC, &datE);
            PRINTF(" at %2d:%2d:%2d\r\n", datE.hour, datE.minute, datE.second);
            myAlarm=false;
        }
    }
    return 0 ;
}

/*!
* @brief ISR for Alarm interrupt
*
* This function changes the state of busyWait.
*/
void RTC_IRQHandler(void)
```

```
{
    uint32_t status = RTC_GetStatusFlags(RTC);

    if (status & kRTC_AlarmFlag)
    {
        myAlarm=true;

        /* Clear alarm flag */
        RTC_ClearStatusFlags(RTC, kRTC_AlarmInterruptEnable);
    }
    else if (status & kRTC_TimeInvalidFlag)
    {
        /* Clear timer invalid flag */
        RTC_ClearStatusFlags(RTC, kRTC_TimeInvalidFlag);
    }
    else
    {
    }
    /* Add for ARM errata 838869, affects Cortex-M4, Cortex-M4F Store
     immediate overlapping
      exception return operation might vector to incorrect interrupt */
#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
```