```c
/* Includes ------------------------------------------------------------------*/

/* Standard STM32L1xxx driver headers */
#include "misc.h"
#include "stm32l1xx_adc.h"
#include "stm32l1xx_lcd.h"
#include "stm32l1xx_rcc.h"
#include "stm32l1xx_rtc.h"
#include "stm32l1xx_exti.h"
#include "stm32l1xx_pwr.h"
#include "stm32l1xx_syscfg.h"
#include "stm32l1xx_dbgmcu.h"
#include <stdlib.h>

/* Touch sensing driver headers */
#include "stm32_tsl_api.h"
#include "stm32l15x_tsl_ct_acquisition.h"

/* discovery board and specific drivers headers*/
#include "discover_board.h"
#include "icc_measure.h"
#include "discover_functions.h"
#include "stm32l_discovery_lcd.h"
#include "stm32_tsl_timebase.h"
#include "stm32l1xx_usart.h"
#include "thermalprinter.h"

#include "stdint.h"
#include <stdio.h>
#include "string.h"

void print (void);
static volatile uint32_t TimingDelay;
extern unsigned char Bias_Current;    /* Bias Current stored in E²Prom used for
ICC mesurement precision */
extern uint8_t t_bar[2];              /* LCD bar graph: used for displaying
active function */
extern bool Auto_test;                /* Auto_test activation flag: set by
interrupt handler if user button is pressed for a few seconds */
extern bool Idd_WakeUP;               /* */
extern volatile bool KeyPressed;      /* */
extern bool UserButton;               /* Set by interrupt handler to indicate
that user button is pressed */
uint8_t state_machine;                /* Machine status used by main() wich
indicats the active function, set by user button in interrupt handler */
uint16_t Int_CurrentSTBY;

/*******************************************************************/
//command structure
typedef struct command
{
      char *name;
      void (*function)(void);
}cmd_lst;
/********************************************************************/
//command List,here we have only 1 command but we can add more commands here
directly.
const cmd_lst list[]={
            {"print",print}
};
/* print("data u want to print",mode)
this is the command format and mode selection u can see from thermal printer.h*/
```

```
/*****************************************************************************/
unsigned long int adr,d;
char p_cmd[20];
char temp1[50]={'\0'};
char temp2[50]={'\0'};
/*****************************************************************************/
int main(void)
{
  bool StanbyWakeUp ;
  float Current_STBY;
  char ch[50]={'\0'};
  uint8_t ind=0,done = 0;
 /*!< At this stage the microcontroller clock setting is already configured,
       this is done through SystemInit() function which is called from startup
       file (startup_stm32l1xx_md.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32l1xx.c file
     */

  /* store Standby Current*/
  Int_CurrentSTBY = Current_Measurement();

  /* Check if the StandBy flag is set */
  if (PWR_GetFlagStatus(PWR_FLAG_SB) != RESET)
  {
    /* System resumed from STANDBY mode */
    /* Clear StandBy flag */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR,ENABLE);
    PWR_ClearFlag(PWR_FLAG_SB);
    /* set StandbyWakeup indicator*/
    StanbyWakeUp = TRUE;
  } else
  {
    /* Reset StandbyWakeup indicator*/
    StanbyWakeUp = FALSE;
  }

  /* Configure Clocks for Application need */
  RCC_Configuration();

  /* Set internal voltage regulator to 1.8V */
  PWR_VoltageScalingConfig(PWR_VoltageScaling_Range1);

  /* Wait Until the Voltage Regulator is ready */
  while (PWR_GetFlagStatus(PWR_FLAG_VOS) != RESET) ;

  /* Init I/O ports */
  Init_GPIOs();

  /* Initializes ADC */
  ADC_Icc_Init();

  /* Enable General interrupts */
  enableInterrupts();

  /* Init Touch Sensing configuration */
  TSL_Init();

  /*  Multichanel Key settings*/
  sMCKeyInfo[0].Setting.b.IMPLEMENTED = 1;
  sMCKeyInfo[0].Setting.b.ENABLED = 1;
  sMCKeyInfo[0].DxSGroup = 0x00;
  /* Initializes the LCD glass */
  LCD_GLASS_Init();

  /* Reset Keypressed flag used in interrupt and Scrollsentence */
  KeyPressed = FALSE;

  /* user button actif */
  UserButton = TRUE;

  /* Check if User button press at Power ON  */
  if ((USERBUTTON_GPIO_PORT->IDR & USERBUTTON_GPIO_PIN) != 0x0)
  {
    /* Measure operational amplifier bias current and store value in E²Prom for
application need*/
    Bias_measurement();
  }

  /* Standard application startup */
  if ( !StanbyWakeUp )
  {
    /* Reset autotest flag stored in memory */
    AUTOTEST(FALSE) ;
    /* Display Welcome message */
    LCD_GLASS_ScrollSentence((uint8_t*)"      ** STM32L-DISCOVERY
**",1,SCROLL_SPEED);
    if (!KeyPressed)
    {
       /* if welcome message not skipped Display blinking message JP1 ON*/
      LCD_BlinkConfig(LCD_BlinkMode_AllSEG_AllCOM,LCD_BlinkFrequency_Div512);
      LCD_GLASS_DisplayString((uint8_t*)"JP1 ON");
      TEMPO;
      TEMPO;
      TEMPO;
      TEMPO;
      LCD_BlinkConfig(LCD_BlinkMode_Off,LCD_BlinkFrequency_Div32);
    }
  /* Wake up from Standby or autotest */
  }  else  {
    /*Check Autotest value stored in flash to get wakeup context*/
    if (Auto_test)
    {
      /* Wake UP: Return of RESET by Auto test */
      auto_test_part2();
    } else {
      /* Wake UP: Return of RESET by Current STAND BY measurement */
      LCD_GLASS_ScrollSentence((uint8_t*)"     STANDBY WAKEUP",1,SCROLL_SPEED);
      /* Substract bias current from operational amplifier*/
      if ( Int_CurrentSTBY > Bias_Current )
        Int_CurrentSTBY -= Bias_Current;
      Current_STBY = Int_CurrentSTBY * Vdd_appli()/ADC_CONV;
      Current_STBY *= 20L;
      display_MuAmp((uint32_t)Current_STBY);
      /* Wait for user button press to continue */
      while(!KeyPressed);
    }
  }
  /* Reset KeyPress Flag */
  KeyPressed = FALSE;
  /* Clear LCD bars */
  BAR0_OFF;
  BAR1_OFF;
  BAR2_OFF;
  BAR3_OFF;
  /* Switch off the leds*/
 // GPIO_HIGH(LD_GPIO_PORT,LD_GREEN_GPIO_PIN);
  GPIO_LOW(LD_GPIO_PORT,LD_BLUE_GPIO_PIN);
  /* Set application state machine to VREF state  */
  state_machine = STATE_VREF ;
  /*Until application reset*/
  Init_UART();
  while (1)
  {
  /* run autotest if requested by the user */
    if (Auto_test)
      auto_test();
    /* Perform Actions depending on current application State  */
    switch (state_machine)
    {
      case STATE_VREF:
            if (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) != 0)//check for
new ch. received
                        {
                              ch[ind] = USART_ReceiveData(USART1);// receive
character

                              if((ch[ind] == 10)|(ch[ind] == 13))// check for
CR/LF, terminate reception
                              {
                                    ch[ind]='\0';//terminate with null to make
it string
                                    parse(ch);//send to parse
                                    search_cmd(p_cmd);//search and execute
command
                                    ind=0;done=1;// set done and wait for next
command
                              }

                       if(!done)
                     {
                        if(ch[ind]==8)
                          { if(ind!=0)
                                 ind--; // for backspace
                          }
                        else ind++;
                        }
                            done = 0;
                        }

             break;

        /* for safe: normaly never reaches */
            default:
                  LCD_GLASS_Clear();
                  LCD_GLASS_DisplayString((uint8_t*)"ERROR");
            break;
       }
      /* Reset KeyPress flag*/
      KeyPressed = FALSE;
  }
}

/
********************************************************************************
```

```
******************************/


// Command parsing
void parse(char *ch)
{
      int i=0,j=0;
      /*********************************************************************
****/
      //Command parsing checks for '('.
      while((ch[i]!=40) && (ch[i]!='\0'))
      {
            p_cmd[i]=ch[i];i++;
      }
      p_cmd[i]='\0';i++;
      /*********************************************************************
****/
      //1st data argument parsing.
      j=0;i++;
      while((ch[i]!=34) && (ch[i]!='\0'))
      {
            temp1[j]=ch[i];j++;i++;
      }
      temp1[j]='\0';i++;

      /*********************************************************************
****/
      //2nd data argument parsing
      j=0;i++;
      while((ch[i]!=41) && (ch[i]!='\0'))
      {
            temp2[j]=ch[i];j++;i++;
      }
      temp2[j]='\0';

}
/
*******************************************************************************
*************/

void search_cmd(char* cmd)
{
      uint8_t i,j;
      uint16_t flag;
      i=0;
      for(i=0;i<2;i++)
      {
            flag=strcmp(list[i].name,cmd);
            if(flag==0)
            {
                  list[i].function();
                  return;
            }
      }

}

/*************************************************************/

void  Init_UART (void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART1_InitStructure;
  USART_ClockInitTypeDef USART_ClockInitStructure;


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 |GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType =GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_PinAFConfig(GPIOB , GPIO_PinSource6, GPIO_AF_USART1);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);



  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

  USART_StructInit(&USART1_InitStructure);
  USART_Init(USART1, &USART1_InitStructure);

  USART_Cmd(USART1, ENABLE);

}
/
********************************************************************************
*********************/

void print (void)
{int x;

x=temp2[0]-48;
tprint(temp1,x);

}


void RCC_Configuration(void)
{
  /* Enable HSI Clock */
  RCC_HSICmd(ENABLE);

  /*!< Wait till HSI is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);

  /* Set HSI as sys clock*/
  RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

  /* Set MSI clock range to ~4.194MHz*/
  RCC_MSIRangeConfig(RCC_MSIRange_6);

  /* Enable the GPIOs clocks */
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOB |
RCC_AHBPeriph_GPIOC| RCC_AHBPeriph_GPIOD| RCC_AHBPeriph_GPIOE|
RCC_AHBPeriph_GPIOH, ENABLE);

  /* Enable comparator, LCD and PWR mngt clocks */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_COMP | RCC_APB1Periph_LCD |
RCC_APB1Periph_PWR,ENABLE);

  /* Enable ADC & SYSCFG clocks */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_SYSCFG , ENABLE);

  /* Allow access to the RTC */
  PWR_RTCAccessCmd(ENABLE);

  /* Reset RTC Backup Domain */
  RCC_RTCResetCmd(ENABLE);
  RCC_RTCResetCmd(DISABLE);

  /* LSE Enable */
  RCC_LSEConfig(RCC_LSE_ON);

  /* Wait until LSE is ready */
  while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET);

   /* RTC Clock Source Selection */
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

  /* Enable the RTC */
  RCC_RTCCLKCmd(ENABLE);

  /*Disable HSE*/
  RCC_HSEConfig(RCC_HSE_OFF);
  if(RCC_GetFlagStatus(RCC_FLAG_HSERDY) != RESET )
  {
    /* Stay in infinite loop if HSE is not disabled*/
    while(1);
  }
}

/**
  * @brief  To initialize the I/O ports
  * @caller main
  * @param None
  * @retval None
  */
void  Init_GPIOs (void)
{
  /* GPIO, EXTI and NVIC Init structure declaration */
  GPIO_InitTypeDef GPIO_InitStructure;
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure User Button pin as input */
  GPIO_InitStructure.GPIO_Pin = USERBUTTON_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
  GPIO_Init(USERBUTTON_GPIO_PORT, &GPIO_InitStructure);

  /* Select User Button pin as input source for EXTI Line */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);

  /* Configure EXT1 Line 0 in interrupt mode trigged on Rising edge */
  EXTI_InitStructure.EXTI_Line = EXTI_Line0 ;  // PA0 for User button AND 
IDD_WakeUP
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set EXTI0 Interrupt to the lowest priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn ;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Configure the LED_pin as output push-pull for LD3 & LD4 usage*/
  GPIO_InitStructure.GPIO_Pin = LD_GREEN_GPIO_PIN | LD_BLUE_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(LD_GPIO_PORT, &GPIO_InitStructure);

  /* Force a low level on LEDs*/
  GPIO_LOW(LD_GPIO_PORT,LD_GREEN_GPIO_PIN);
  GPIO_LOW(LD_GPIO_PORT,LD_BLUE_GPIO_PIN);

/* Counter enable: GPIO set in output for enable the counter */
  GPIO_InitStructure.GPIO_Pin = CTN_CNTEN_GPIO_PIN;
  GPIO_Init( CTN_GPIO_PORT, &GPIO_InitStructure);

/* To prepare to start counter */
  GPIO_HIGH(CTN_GPIO_PORT,CTN_CNTEN_GPIO_PIN);

/* Configure Port A LCD Output pins as alternate function */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 |
GPIO_Pin_8 | GPIO_Pin_9 |GPIO_Pin_10 |GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOA, &GPIO_InitStructure);

/* Select LCD alternate function for Port A LCD Output pins */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource15,GPIO_AF_LCD) ;

  /* Configure Port B LCD Output pins as alternate function */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_8 | GPIO_Pin_9 \
                                 | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 |
GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOB, &GPIO_InitStructure);

  /* Select LCD alternate function for Port B LCD Output pins */
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource4,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource5,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource11,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource12,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource13,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource14,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource15,GPIO_AF_LCD) ;

  /* Configure Port C LCD Output pins as alternate function */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
GPIO_Pin_3 | GPIO_Pin_6 \
                                 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10 |GPIO_Pin_11 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_Init( GPIOC, &GPIO_InitStructure);

  /* Select LCD alternate function for Port B LCD Output pins */
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource0,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource1,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource2,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource3,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource10,GPIO_AF_LCD) ;
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11,GPIO_AF_LCD) ;

  /* Configure ADC (IDD_MEASURE) pin as Analogue */
  GPIO_InitStructure.GPIO_Pin = IDD_MEASURE  ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_Init( IDD_MEASURE_PORT, &GPIO_InitStructure);
}


/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;
  while(TimingDelay != 0);
}

/**
  * @brief  Decrements the TimingDelay variable.
  * @caller SysTick interrupt Handler
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
    TimingDelay--;
}





#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
```

```
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* Infinite loop */
      (void)file;
      (void)line;
  while (1);
}

#endif

/******************* (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```