- CO3009 -

- Embedded C Programming -

A Simple C program

```
void FSM();
void main(void){
    // initialize the device
    System_Initialization();
    while (1) {
            FSM();
   }
}
/**
  * @brief  This function handles TIM interrupt request.
  * @param  None
  * @retval None   */
void TIM3_IRQHandler(void){
      HAL_TIM_IRQHandler(&TimHandle);
}
```

```
/**
  * @brief  Period elapsed callback in non blocking mode
  * @param  htim : TIM handle
  * @retval None  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim) {
      SCH_Update();
}
```


A main() Function

- System initialization
  - Oscillator
  - Input/Output
  - Special peripherals/modules: Timers, CCP, ADC
  - Disable/Enable interrupt
- A super loop while(1)
- One or more finite state machine (FSM)

System Initialization

```c
enum InitState initState = HAL_INIT;

void System_Initialization(void)
{
    while(initState != MAX_INIT_STATE){
        switch (initState) {
        case HAL_INIT:
            HAL_Init();
            break;
        case SYSTEM_CLOCK_INIT:
            SystemClock_Config();
            break;
        case UART_INIT:
            UART3_Init();
            UART1_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"UART_INIT - Done \r\n"));
            break;
        case GPIO_INIT:
            MX_GPIO_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"GPIO_INIT - ADC_DMA_Init - Done \r\n"));
            break;
        case LED_DISPLAY_INIT:
            Led_Display_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"LED_DISPLAY_INIT - Done \r\n"));
            break;
        case RELAY_INIT:
            Relay_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"RELAY_INIT - Done \r\n"));
            break;
        case FLASH_INIT:
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"FLASH_INIT - Done \r\n"));
            break;
        case TIMER_INIT:
            Timer_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"TIMER_INIT - Done \r\n"));
            break;
        case SPI_INIT:
            SPI1_Init();
            SPI2_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"SPI_INIT - Done \r\n"));
            break;
        case SPI_25LCXXX_INIT:
            Eeprom_Initialize();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"SPI_25LCXXX_INIT - Done \r\n"));
            break;
        case I2C_INIT:
            I2C_Init();
            DEBUG_INIT(UART3_SendToHost((uint8_t*)"I2C_Init \r\n"));
            PCF_Init();
```

anhpham@hcmut.edu.vn

HAL_Init

- This function is used to initialize the HAL Library; it must be the first instruction to be executed in the main program (before to call any other HAL function), it performs the following:
  - Configure the Flash prefetch.
  - Configures the SysTick to generate an interrupt each 1 millisecond, which is clocked by the HSI (at this stage, the clock is not yet configured and thus the system is running from the internal HSI at 16 MHz).
  - Set NVIC Group Priority to 4.
  - Calls the **HAL_MspInit()** callback function defined in user file "**stm32f1xx_hal_msp.c**" to do the global low level hardware initialization

  - **@note** SysTick is used as time base for the **HAL_Delay()** function, the application need to ensure that the SysTick time base is always set to **01 millisecond** to have correct HAL operation.
  - **@retval** HAL status

HAL_Init

```
142 HAL_StatusTypeDef HAL_Init(void)
143 {
144   /* Configure Flash prefetch */
145 #if (PREFETCH_ENABLE != 0)
146 #if defined(STM32F101x6) || defined(STM32F101xB) || defined(STM32F101xE) || defined(STM32F101xG) || \
147     defined(STM32F102x6) || defined(STM32F102xB) || \
148     defined(STM32F103x6) || defined(STM32F103xB) || defined(STM32F103xE) || defined(STM32F103xG) || \
149     defined(STM32F105xC) || defined(STM32F107xC)
150
151   /* Prefetch buffer is not available on value line devices */
152   __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
153 #endif
154 #endif /* PREFETCH_ENABLE */
155
156   /* Set Interrupt Group Priority */
157   HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
158
159   /* Use systick as time base source and configure 1ms tick (default clock after Reset is HSI) */
160   HAL_InitTick(TICK_INT_PRIORITY);
161
162   /* Init the low level hardware */
163   HAL_MspInit();
164
165   /* Return function status */
166   return HAL_OK;
167 }
```


SystemClock_Config

```c
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
    /** Initializes the CPU, AHB and APB busses clocks
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
    RCC_OscInitStruct.HSEState        = RCC_HSE_OFF;
    RCC_OscInitStruct.LSEState        = RCC_LSE_OFF;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;

    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK){
        Error_Handler();
    }
    /** Initializes the CPU, AHB and APB busses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK){
        Error_Handler();
    }
    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
    PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK){
        Error_Handler();
    }
}
```


Clocks

Three different clock sources can be used to drive the system clock (SYSCLK):

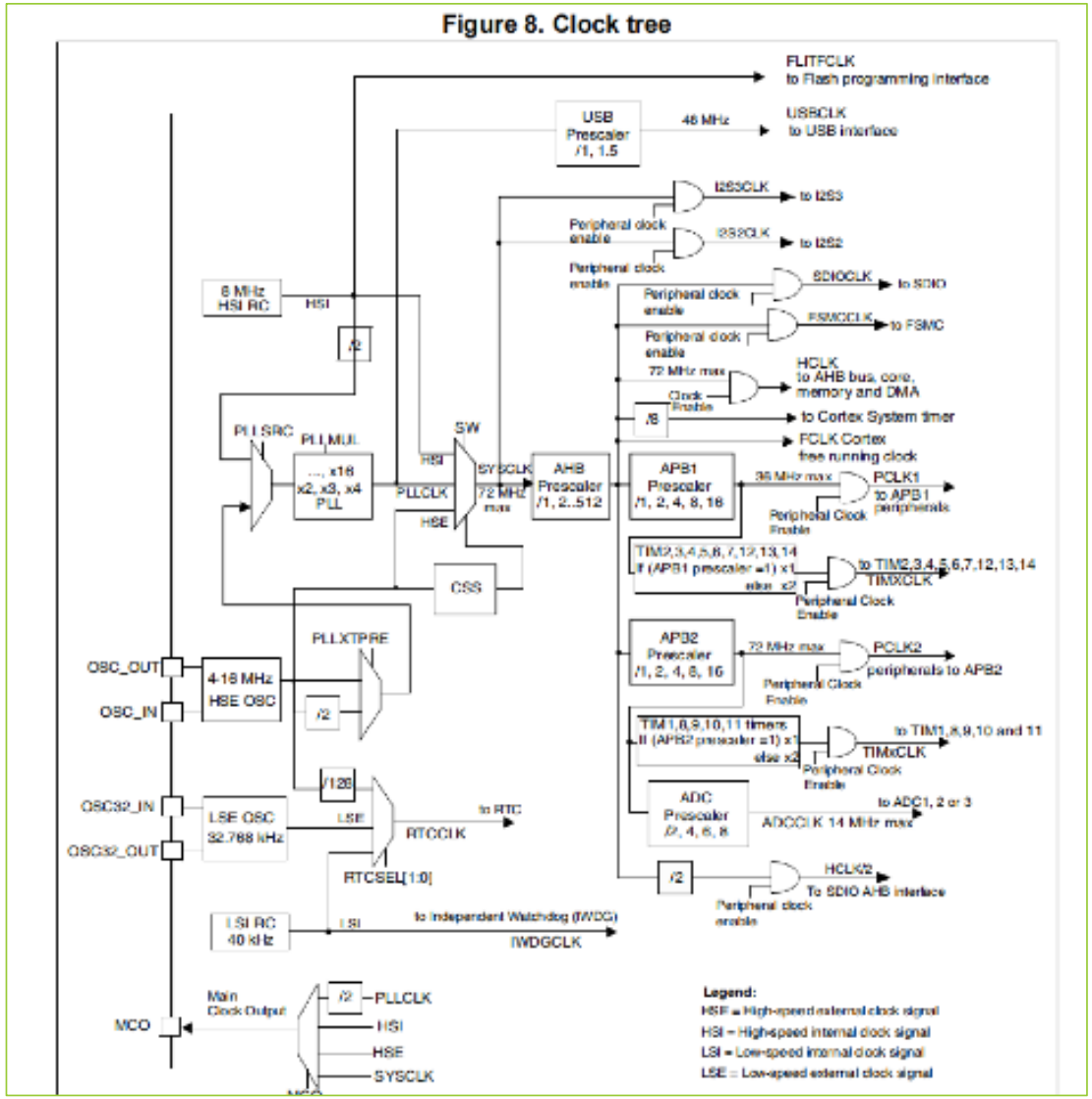
- HIS (High-Speed Internal) oscillator clock
- HSE (High-Speed External) oscillator clock
- PLL (Phase-Locked Loop) clock

The devices have the following two secondary clock sources:

- 40 kHz low speed internal RC (LSI RC), which drives the independent watchdog and optionally the RTC used for Auto-wakeup from Stop/Standby mode.
- 32.768 kHz low speed external crystal (LSE crystal), which optionally drives the real-time clock (RTCCLK)

Each clock source can be switched on or off independently when it is not used, to optimize power consumption.

STM32F103 Family Clockc Diagram



anhpham@hcmut.edu.vn

MX_GPIO_Init

```
14  /**
15    * @brief GPIO Initialization Function
16    * @param None
17    * @retval None
18    */
19  void MX_GPIO_Init(void)
20  {
21  //  GPIO_InitTypeDef GPIO_InitStruct = {0};
22
23      /* GPIO Ports Clock Enable */
24      __HAL_RCC_GPIOC_CLK_ENABLE();
25      __HAL_RCC_GPIOD_CLK_ENABLE();
26      __HAL_RCC_GPIOA_CLK_ENABLE();
27      __HAL_RCC_GPIOB_CLK_ENABLE();
28
29  //  /*Configure GPIO pin Output Level */
30  //  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
31  //
32      LED_Init();
33      GPIO_Relay_Init();
34      Buzzer_Init();
35      SPI_CS_Init();
36      ZeroPoint_Detection_Pin_Init();
37  }
```

anhpham@hcmut.edu.vn

Input Output Pins Initialization

```
38
39  void LED_Init(void){
40
41      GPIO_InitTypeDef GPIO_InitStruct = {0};
42
43      GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
44      GPIO_InitStruct.Pull  = GPIO_PULLUP;
45      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
46      GPIO_InitStruct.Pin = LED2_PIN;
47      HAL_GPIO_Init(LED2_GPIO_PORT, &GPIO_InitStruct);
48
49  }
50
```

```
102
103 void ZeroPoint_Detection_Pin_Init(void){
104     GPIO_InitTypeDef GPIO_InitStruct = {0};
105
106     GPIO_InitStruct.Pin = ZERO_POINT_DETECTION_PIN;
107     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
108     GPIO_InitStruct.Pull = GPIO_NOPULL;
109     HAL_GPIO_Init(ZERO_POINT_DETECTION_PORT, &GPIO_InitStruct);
110
111 #if(VERSION_EBOX == 2)
112     /* EXTI interrupt init*/
113       HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
114       HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
115 #else
116       HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
117       HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
118 #endif
119 }
120
```

```
54  void GPIO_Relay_Init(void){
55      GPIO_InitTypeDef GPIO_InitStruct = {0};
56
57      GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
58      GPIO_InitStruct.Pull  = GPIO_PULLUP;
59      GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
60
61      GPIO_InitStruct.Pin = RELAY_PIN_0;
62      HAL_GPIO_Init(RELAY_PORT_0, &GPIO_InitStruct);
63
64      GPIO_InitStruct.Pin = RELAY_PIN_1;
65      HAL_GPIO_Init(RELAY_PORT_1, &GPIO_InitStruct);
66      GPIO_InitStruct.Pin = RELAY_PIN_2;
67      HAL_GPIO_Init(RELAY_PORT_2, &GPIO_InitStruct);
68      GPIO_InitStruct.Pin = RELAY_PIN_3;
69      HAL_GPIO_Init(RELAY_PORT_3, &GPIO_InitStruct);
70      GPIO_InitStruct.Pin = RELAY_PIN_4;
71      HAL_GPIO_Init(RELAY_PORT_4, &GPIO_InitStruct);
72      GPIO_InitStruct.Pin = RELAY_PIN_5;
73      HAL_GPIO_Init(RELAY_PORT_5, &GPIO_InitStruct);
74      GPIO_InitStruct.Pin = RELAY_PIN_6;
75      HAL_GPIO_Init(RELAY_PORT_6, &GPIO_InitStruct);
76      GPIO_InitStruct.Pin = RELAY_PIN_7;
77      HAL_GPIO_Init(RELAY_PORT_7, &GPIO_InitStruct);
78      GPIO_InitStruct.Pin = RELAY_PIN_8;
79      HAL_GPIO_Init(RELAY_PORT_8, &GPIO_InitStruct);
80      GPIO_InitStruct.Pin = RELAY_PIN_9;
81      HAL_GPIO_Init(RELAY_PORT_9, &GPIO_InitStruct);
82  #if(VERSION_EBOX == 2)
83      GPIO_InitStruct.Pin = PD2_RELAY_ENABLE_PIN;
84      HAL_GPIO_Init(PD2_RELAY_ENABLE_PORT, &GPIO_InitStruct);
85  #endif
86  }
```


UART3_Init

```
78  /**
79    * @brief USART3 Initialization Function
80    * @param None
81    * @retval None
82    */
83  void UART3_Init(void)
84  {
85      Uart3Handle.Instance = USART3;
86      Uart3Handle.Init.BaudRate = 115200;
87      Uart3Handle.Init.WordLength = UART_WORDLENGTH_8B;
88      Uart3Handle.Init.StopBits = UART_STOPBITS_1;
89      Uart3Handle.Init.Parity = UART_PARITY_NONE;
90      Uart3Handle.Init.Mode = UART_MODE_TX_RX;
91      Uart3Handle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
92      Uart3Handle.Init.OverSampling = UART_OVERSAMPLING_16;
93    if (HAL_UART_Init(&Uart3Handle) != HAL_OK) {
94      Error_Handler();
95    }
96  }
```


Timer_Init

Configure the TIM peripheral

In this example TIM3 input clock (TIM3CLK) is set to APB1 clock (PCLK1) x2, since APB1 prescaler is set to 4 (0x100).

TIM3CLK = PCLK1*2

- PCLK1 = HCLK/2
- => TIM3CLK = PCLK1*2 = (HCLK/2)*2 = HCLK = SystemCoreClock
- To get TIM3 counter clock at 10 KHz, the Prescaler is computed as following:
- Prescaler = (TIM3CLK / TIM3 counter clock) - 1
- Prescaler = (SystemCoreClock /10 KHz) - 1

Note:

- SystemCoreClock variable holds HCLK frequency and is defined in **system_stm32f1xx.c** file.
- Each time the core clock (HCLK) changes, user had to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.
- This variable is updated in three ways:
  1) by calling CMSIS function SystemCoreClockUpdate()
  2) by calling HAL API function HAL_RCC_GetSysClockFreq()
  3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency

An FSM Function

```
enum {St_Intro, St_Voltmeter, St_Temperature, St_Clock,} State_Machine;
unsigned char State = St_Intro;
void FSM(){
	switch (State)
	{
		case St_Intro:
			Intro();
			break;
		case St_Voltmeter:
			Voltmeter();
			break;
		case St_Temperature:
			Temperature();
			break;
		case St_Clock:
			Clock();
			break;
	}
}
```