```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "stm32f0xx.h"
#include <lcd_stm32f0.c>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

// Definitions for SPI usage
#define MEM_SIZE 8192 // bytes
#define WREN 0b00000110 // enable writing
#define WRDI 0b00000100 // disable writing
#define RDSR 0b00000101 // read status register
#define WRSR 0b00000001 // write status register
#define READ 0b00000011
#define WRITE 0b00000010
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim16;
```

```
/* USER CODE BEGIN PV */

// TODO: Define input variables
uint32_t current_time = 0;
uint32_t prev_time = 0;
uint32_t delay_led = 500; //500ms delay

//array of 8-bit binary integers
uint8_t data[6] = {0b10101010, 0b01010101, 0b11001100, 0b00110011, 0b11110000,
0b00001111}; //Data array
uint16_t address = 0;//EEprom address
uint32_t adc_val;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM16_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
void TIM16_IRQHandler(void);
void writeLCD(char *char_in);

// ADC functions
uint32_t pollADC(void);
uint32_t ADCtoCCR(uint32_t adc_val);
```

```
// SPI functions
static void init_spi(void);
static void write_to_address(uint16_t address, uint8_t data);
static uint8_t read_from_address(uint16_t address);
static void spi_delay(uint32_t delay_in_us);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  init_spi();
  MX_GPIO_Init();
  MX_ADC_Init();
  MX_TIM3_Init();
  MX_TIM16_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */

  // Initialise LCD
  init_LCD();

  // Start timers
  HAL_TIM_Base_Start_IT(&htim6);
  HAL_TIM_Base_Start_IT(&htim16);

  // PWM setup
  uint32_t CCR = 0;
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3
  // TODO: Write bytes to EEPROM using "write_to_address"
  uint8_t index = 0;
  while(index < 6){
    write_to_address(address, data[index]);

    index++;
    spi_delay(100);
  }
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

  // TODO: Poll ADC
adc_val = pollADC();//read analogue adc value from potentiometer

  // TODO: Get CRR
  CCR = ADCtoCCR(adc_val);

  // Update PWM value   ( divide by 4  to make it turn off)
  __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR/4);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

```
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
  while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
  {
  }
  LL_RCC_HSI_Enable();

   /* Wait till HSI is ready */
  while(LL_RCC_HSI_IsReady() != 1)
  {

  }
  LL_RCC_HSI_SetCalibTrimming(16);
  LL_RCC_HSI14_Enable();

   /* Wait till HSI14 is ready */
  while(LL_RCC_HSI14_IsReady() != 1)
  {

  }
  LL_RCC_HSI14_SetCalibTrimming(16);
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);

   /* Wait till System clock is ready */
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
  {

  }
  LL_SetSystemCoreClock(8000000);

   /* Update the time base */
  if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
  {
    Error_Handler();
  }
  LL_RCC_HSI14_EnableADCControl();
}

/**
  * @brief ADC Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC_Init(void)
{

  /* USER CODE BEGIN ADC_Init 0 */
  /* USER CODE END ADC_Init 0 */
  }



  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC_Init 1 */

  /* USER CODE END ADC_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel to be converted.
  */
  sConfig.Channel = ADC_CHANNEL_6;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC_Init 2 */
  ADC1->CR |= ADC_CR_ADCAL;
  while(ADC1->CR & ADC_CR_ADCAL);     // Calibrate the ADC
  ADC1->CR |= (1 << 0);           // Enable ADC
  while((ADC1->ISR & (1 << 0)) == 0);   // Wait for ADC ready
  /* USER CODE END ADC_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */
```

```
}


TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};

/* USER CODE BEGIN TIM3_Init 1 */

/* USER CODE END TIM3_Init 1 */
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 47999;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */
  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 8000-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 500-1;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */
  NVIC_EnableIRQ(TIM6_IRQn);
  /* USER CODE END TIM6_Init 2 */

}
```

```
/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{

  /* USER CODE BEGIN TIM16_Init 0 */

  /* USER CODE END TIM16_Init 0 */

  /* USER CODE BEGIN TIM16_Init 1 */

  /* USER CODE END TIM16_Init 1 */
  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 8000-1;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim16.Init.Period = 1000-1;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM16_Init 2 */
  NVIC_EnableIRQ(TIM16_IRQn);
  /* USER CODE END TIM16_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);

  /**/
  LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);

  /**/
  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA,
LL_SYSCFG_EXTI_LINE0);

  /**/
  LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);

  /**/
  LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin,
LL_GPIO_MODE_INPUT);

  /**/
  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
  EXTI_InitStruct.LineCommand = ENABLE;
  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
  LL_EXTI_Init(&EXTI_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED7_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
  HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void EXTI0_1_IRQHandler(void)
```

```
{
  // TODO: Add code to switch LED7 delay frequency
  current_time = HAL_GetTick();

  //ensures unwanted noise within duration is not registered
  if((current_time - prev_time)> 200){
    if(delay_led = 500 ){ //if frequency of led is 2Hz
      delay_led = 1000;//toggle the frequency of LED by changing delay
      htim6.Init.Period = delay_led -1;
    }else if(delay_led = 1000){ //if frequency of led is 1Hz
      delay_led = 1000;
      htim6.Init.Period = delay_led -1;
    }

    //update TIM6 with the new period; ensure execution complete
        if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
        {
          Error_Handler();
        }

  }

  prev_time = current_time;//update the last time since click
  HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}

void TIM6_IRQHandler(void)
{
  // Acknowledge interrupt
  HAL_TIM_IRQHandler(&htim6);

  // Toggle LED7
  HAL_GPIO_TogglePin(GPIOB, LED7_Pin);
}

void TIM16_IRQHandler(void)
{
  // Acknowledge interrupt
  HAL_TIM_IRQHandler(&htim16);


  // TODO: Initialise a string to output second line on LC-D
  char charArray[16];//buffer

  // TODO: Change LED pattern; output 0x01 if the read SPI data is incorrect
  if (address > 5){

      address= 0;
    }

    //validate byte at address
  uint8_t num = read_from_address(address);
  spi_delay(100);
  snprintf(charArray, sizeof(charArray), "%d", read_from_address(address));
  writeLCD(charArray);


  //iterate address
  address++;
```

```
}

// TODO: Complete the writeLCD function
void writeLCD(char *char_in){
  delay(3000);
  lcd_command(CLEAR);
  lcd_putstring("EEPROM byte:");
  lcd_command(LINE_TWO);
  lcd_putstring(char_in);

}

// Get ADC value
uint32_t pollADC(void){
  HAL_ADC_Start(&hadc); // start the adc
  HAL_ADC_PollForConversion(&hadc, 100); // poll for conversion
  uint32_t val = HAL_ADC_GetValue(&hadc); // get the adc value
  HAL_ADC_Stop(&hadc); // stop adc
  return val;
}

// Calculate PWM CCR value
uint32_t ADCtoCCR(uint32_t adc_val){
  // TODO: Calculate CCR value (val) using an appropriate equation
  uint32_t val_ccr;

  val_ccr = (adc_val * 47999) / 4095;
  return val_ccr;
}

void ADC1_COMP_IRQHandler(void)
{
  adc_val = HAL_ADC_GetValue(&hadc); // read adc value
  HAL_ADC_IRQHandler(&hadc); //Clear flags
}

// Initialise SPI
static void init_spi(void) {

  // Clock to PB
  RCC->AHBENR |= RCC_AHBENR_GPIOBEN;  // Enable clock for SPI port

  // Set pin modes
  GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate Function
  GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
  GPIOB->BSRR |= GPIO_BSRR_BS_12;     // Pull CS high

  // Clock enable to SPI
  RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
  SPI2->CR1 |= SPI_CR1_BIDIOE;                  // Enable output
  SPI2->CR1 |= (SPI_CR1_BR_0 |  SPI_CR1_BR_1);         // Set Baud to fpclk / 16
  SPI2->CR1 |= SPI_CR1_MSTR;                  // Set to master mode
  SPI2->CR2 |= SPI_CR2_FRXTH;                   // Set RX threshold to be 8 bits
  SPI2->CR2 |= SPI_CR2_SSOE;                  // Enable slave output to work in master
mode
  SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);  // Set to 8-bit
mode
  SPI2->CR1 |= SPI_CR1_SPE;                  // Enable the SPI peripheral
}

// Implements a delay in microseconds
static void spi_delay(uint32_t delay_in_us) {
  volatile uint32_t counter = 0;
  delay_in_us *= 3;
  for(; counter < delay_in_us; counter++) {
    __asm("nop");
    __asm("nop");
  }
}

// Write to EEPROM address using SPI
static void write_to_address(uint16_t address, uint8_t data) {

  uint8_t dummy; // Junk from the DR

  // Set the Write Enable latch
  GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
  spi_delay(1);
  *((uint8_t*)(&SPI2->DR)) = WREN;
  while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
  spi_delay(5000);

  // Send write instruction
  GPIOB->BSRR |= GPIO_BSRR_BR_12;       // Pull CS low
  spi_delay(1);
  *((uint8_t*)(&SPI2->DR)) = WRITE;
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send 16-bit address
  *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  *((uint8_t*)(&SPI2->DR)) = (address);     // Address LSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send the data
  *((uint8_t*)(&SPI2->DR)) = data;
  while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
  spi_delay(5000);
}

// Read from EEPROM address using SPI
static uint8_t read_from_address(uint16_t address) {
  uint8_t dummy; // Junk from the DR

  // Send the read instruction
  GPIOB->BSRR |= GPIO_BSRR_BR_12;       // Pull CS low
  spi_delay(1);
  *((uint8_t*)(&SPI2->DR)) = READ;
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Send 16-bit address
  *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  *((uint8_t*)(&SPI2->DR)) = (address);     // Address LSB
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;

  // Clock in the data
  *((uint8_t*)(&SPI2->DR)) = 0x42;      // Clock out some junk data
  while ((SPI2->SR & SPI_SR_RXNE) == 0);    // Hang while RX is empty
  dummy = SPI2->DR;
  GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high
  spi_delay(5000);

  return dummy;               // Return read data
}
/* USER CODE END 4 */

/**
```

```
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#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 CODE BEGIN 6 */
  /* 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) */
  /* USER CODE END 6 */
}
```

```
#endif /* USE_FULL_ASSERT */
```