```
#include "stm32f1xx_hal.h"

/* Global Variables */
ADC_HandleTypeDef hadc1;
TIM_HandleTypeDef htim2;
uint8_t sound_pattern = 0;

/* Pin Definitions */
#define LED_RED_PIN    GPIO_PIN_12
#define LED_GREEN_PIN  GPIO_PIN_13
#define LED_BLUE_PIN   GPIO_PIN_14
#define LED_PORT       GPIOB
#define BUTTON_PIN     GPIO_PIN_0
#define BUTTON_PORT    GPIOB
#define BUZZER_PIN     GPIO_PIN_2  // TIM2_CH3 (PA2)

/* Threshold Values */
#define ADC_THRESH_HIGH  3000
#define ADC_THRESH_MID   1500

/* Frekuensi Buzzer - using uint32_t instead of uint16_t */
const uint32_t pwm_periods[] = {1000, 50000, 719999}; // 72MHz/freq - 1

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
void update_leds_and_buzzer(uint32_t adc_val, uint8_t btn_state); // Updated function signature
void change_sound_pattern(void);
void Error_Handler(void); // Explicit declaration
```

```
int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);

  HAL_ADC_Start(&hadc1);

  while (1) {
    static uint32_t last_adc_tick = 0;
    static uint32_t last_sound_change = 0;
    uint8_t button_state = HAL_GPIO_ReadPin(BUTTON_PORT, BUTTON_PIN);

    if (HAL_GetTick() - last_adc_tick > 200) {
      last_adc_tick = HAL_GetTick();
      HAL_ADC_Start(&hadc1);
      if (HAL_ADC_PollForConversion(&hadc1, 10) == HAL_OK) {
        update_leds_and_buzzer(HAL_ADC_GetValue(&hadc1), button_state);
      }
    }

    if (button_state == GPIO_PIN_RESET && (HAL_ADC_GetValue(&hadc1) < ADC_THRESH_MID)) {
      if (HAL_GetTick() - last_sound_change > 1000) {
        last_sound_change = HAL_GetTick();
        change_sound_pattern();
      }
    }
```

```
    HAL_Delay(10);
  }
}

void update_leds_and_buzzer(uint32_t adc_val, uint8_t btn_state) {
  HAL_GPIO_WritePin(LED_PORT, LED_RED_PIN|LED_GREEN_PIN|LED_BLUE_PIN, GPIO_PIN_RESET);

  if (adc_val >= ADC_THRESH_HIGH) {
    HAL_GPIO_WritePin(LED_PORT, LED_GREEN_PIN, GPIO_PIN_SET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);
  }
  else if (adc_val >= ADC_THRESH_MID) {
    HAL_GPIO_WritePin(LED_PORT, LED_BLUE_PIN, GPIO_PIN_SET);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);
  }
  else {
    HAL_GPIO_WritePin(LED_PORT, LED_RED_PIN, GPIO_PIN_SET);
    if (btn_state == GPIO_PIN_RESET) {
      __HAL_TIM_SET_AUTORELOAD(&htim2, pwm_periods[sound_pattern]);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, pwm_periods[sound_pattern]/2);
    } else {
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);
    }
  }
}

void change_sound_pattern(void) {
  sound_pattern = (sound_pattern + 1) % 3;
  if (HAL_ADC_GetValue(&hadc1) < ADC_THRESH_MID &&
      HAL_GPIO_ReadPin(BUTTON_PORT, BUTTON_PIN) == GPIO_PIN_SET) {
```

```
    __HAL_TIM_SET_AUTORELOAD(&htim2, pwm_periods[sound_pattern]);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, pwm_periods[sound_pattern]/2);
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses 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();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */
```

```
  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}
```

```
/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

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

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65535;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &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(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
/* GPIO Initialization */
static void MX_GPIO_Init(void) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
  /* LED Outputs */
  GPIO_InitStruct.Pin = LED_RED_PIN|LED_GREEN_PIN|LED_BLUE_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_PORT, &GPIO_InitStruct);

  /* Button Input */
  GPIO_InitStruct.Pin = BUTTON_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(BUTTON_PORT, &GPIO_InitStruct);
}

/* USER CODE BEGIN 4 */

/* 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 */
```