```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"
//Github link
//https://github.com/aimeesimons/NDXDAN019_SMNAIM002_EEE3096S.git
/* 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 */
// TODO: Add values for below variables
#define NS      128      // Number of samples in LUT
#define TIM2CLK 8000000  // STM Clock frequency
#define F_SIGNAL 100 // Frequency of output analog signal
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
DMA_HandleTypeDef hdma_tim2_ch1;

/* USER CODE BEGIN PV */
// TODO: Add code for global variables, including LUTs

uint32_t Sin_LUT[NS] =
{512,537,562,587,612,637,661,685,709,732,754,776,798,818,838,857,875,893,909,925,939,952,965,9
```

```c
76,986,995,1002,1009,1014,1018,1021,1023,1023,1022,1020,1016,1012,1006,999,990,981,970,959,946,932,917,901,884,866,848,828,808,787,765,743,720,697,673,649,624,600,575,549,524,499,474,448,423,399,374,350,326,303,280,258,236,215,195,175,157,139,122,106,91,77,64,53,42,33,24,17,11,7,3,1,0,0,2,5,9,14,21,28,37,47,58,71,84,98,114,130,148,166,185,205,225,247,269,291,314,338,362,386,411,436,461,486,511};

uint32_t saw_LUT[NS] = {0,8,16,24,32,40,48,56,64,72,81,89,97,105,113,121,129,137,145,153,161,169,177,185,193,201,209,217,226,234,242,250,258,266,274,282,290,298,306,314,322,330,338,346,354,362,371,379,387,395,403,411,419,427,435,443,451,459,467,475,483,491,499,507,516,524,532,540,548,556,564,572,580,588,596,604,612,620,628,636,644,652,661,669,677,685,693,701,709,717,725,733,741,749,757,765,773,781,789,797,806,814,822,830,838,846,854,862,870,878,886,894,902,910,918,926,934,942,951,959,967,975,983,991,999,1007,1015,0};

uint32_t triangle_LUT[NS] = {0,16,32,48,64,81,97,113,129,145,161,177,193,209,226,242,258,274,290,306,322,338,354,371,387,403,419,435,451,467,483,499,516,532,548,564,580,596,612,628,644,661,677,693,709,725,741,757,773,789,806,822,838,854,870,886,902,918,934,951,967,983,999,1015,1015,999,983,967,951,934,918,902,886,870,854,838,822,806,789,773,757,741,725,709,693,677,661,644,628,612,596,580,564,548,532,516,499,483,467,451,435,419,403,387,371,354,338,322,306,290,274,258,242,226,209,193,177,161,145,129,113,97,81,64,48,32,16,0};

uint32_t start = 0;
int count = 1;
// TODO: Equation to calculate TIM2_Ticks
uint32_t TIM2_Ticks = TIM2CLK/(F_SIGNAL*NS); // How often to write new LUT value
uint32_t DestAddress = (uint32_t)&(TIM3->CCR3); // Write LUT TO TIM3->CCR3 to modify PWM duty cycle

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);

/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
/* 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 */
  init_LCD();
  /* USER CODE END Init */

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

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

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();

  /* USER CODE BEGIN 2 */
  // TODO: Start TIM3 in PWM mode on channel 3
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);

  // TODO: Start TIM2 in Output Compare (OC) mode on channel 1.
    HAL_TIM_OC_Start(&htim2,TIM_CHANNEL_1);

  // TODO: Start DMA in IT mode on TIM2->CH1; Source is LUT and Dest is TIM3->CCR3; start with Sine LUT

    HAL_DMA_Start_IT(&hdma_tim2_ch1, Sin_LUT, DestAddress, NS);

  // TODO: Write current waveform to LCD ("Sine")
    lcd_putstring("Sine");
  delay(3000);

  // TODO: Enable DMA (start transfer from LUT to CCR)
     __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

  /* USER CODE END 2 */

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

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

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
```

```c
{
  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_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();
  }
}

/**
  * @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_ClockConfigTypeDef sClockSourceConfig = {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 = TIM2_Ticks - 1;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_OC_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_TIMING;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_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 = 1023;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  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);

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel4_5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
```

```c
  LL_EXTI_InitTypeDef EXTI_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_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);

/* 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: Debounce using HAL_GetTick()
    if(HAL_GetTick()-start>1000){
        count +=1;
        if (count==4){
            count = 1;
        }
    // TODO: Disable DMA transfer and abort IT, then start DMA in IT mode with new LUT and re-enable transfer
    // HINT: Consider using C's "switch" function to handle LUT changes
    __HAL_TIM_DISABLE_DMA(&htim2,TIM_DMA_CC1);
    HAL_DMA_Abort_IT(&hdma_tim2_ch1);
    switch(count){

    case 1:
        HAL_DMA_Start_IT(&hdma_tim2_ch1, Sin_LUT, DestAddress, NS);
        lcd_command(CLEAR);
        lcd_putstring("Sine");
        delay(3000);
        __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
        break;
    case 2:
        HAL_DMA_Start_IT(&hdma_tim2_ch1, saw_LUT, DestAddress, NS);
        lcd_command(CLEAR);
        lcd_putstring("Sawtooth");
        delay(3000);
        __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
        break;
    case 3:
        HAL_DMA_Start_IT(&hdma_tim2_ch1, triangle_LUT, DestAddress, NS);
        lcd_command(CLEAR);
        lcd_putstring("Triangular");
        delay(3000);
        __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
        break;
    }

    start = HAL_GetTick();
  }
    HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}
/* 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 */

```