```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 */

// TODO: Add values for below variables

#define NS 128        // Number of samples in LUT

#define TIM2CLK 8000000  // STM Clock frequency

#define F_SIGNAL 1000 // 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] =
{512,537,562,587,611,636,660,684,707,730,753,774,796,816,836,855,873,890,90
7,922,937,950,

      963,974,984,993,1001,1008,1013,1017,1021,1022,1023,1022,1021,1017,101
3,1008,1001,993,984,974,963,950,937,

      922,907,890,873,855,836,816,796,774,753,730,707,684,660,636,611,587,5
62,537,512,486,461,436,412,387,363,

      339,316,293,270,249,227,207,187,168,150,133,116,101,86,73,60,49,39,30
,22,15,10,6,2,1,0,1,2,6,10,15,22,30

      ,39,49,60,73,86,101,116,133,150,168,187,207,227,249,270,293,316,339,3
63,387,412,436,461,486};



uint32_t saw_LUT[NS] =
{0,8,16,24,32,40,48,56,64,72,81,89,97,105,113,121,129,137,145,153,161,169,1
77,185,193,201,

      209,217,226,234,242,250,258,266,274,282,290,298,306,314,322,330,338,3
46,354,362,371,379,387,395,403,411,

      419,427,435,443,451,459,467,475,483,491,499,507,516,524,532,540,548,5
56,564,572,580,588,596,604,612,620,

      628,636,644,652,661,669,677,685,693,701,709,717,725,733,741,749,757,7
65,773,781,789,797,806,814,822,830,

      838,846,854,862,870,878,886,894,902,910,918,926,934,942,951,959,967,9
75,983,991,999,1007,1015,1023};



uint32_t triangle_LUT[NS] =
{0,8,16,24,32,41,49,57,65,73,81,89,97,106,114,122,130,138,146,154,162,171,1
79,187,195,

      203,211,219,227,235,244,252,260,268,276,284,292,300,309,317,325,333,3
41,349,357,365,373,382,390,398,406,414,

      422,430,438,447,455,463,471,479,487,495,503,512,512,503,495,487,479,4
71,463,455,447,438,430,422,414,406,398,

      390,382,373,365,357,349,341,333,325,317,309,300,292,284,276,268,260,2
52,244,235,227,219,211,203,195,187,179,

      171,162,154,146,138,130,122,114,106,97,89,81,73,65,57,49,41,32,24,16,
8,0};


// TODO: Equation to calculate TIM2_Ticks

uint32_t TIM2_Ticks = TIM2CLK / (F_SIGNAL * NS); // How often to write TO
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, (uint32_t)sin_LUT, DestAddress, NS);

  // TODO: Write current waveform to LCD ("Sine")
  delay(3000);
  lcd_command(CLEAR);
  lcd_putstring("SINE :( ");
  // 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)
{
  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;
```

```c
  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)
```

```c
  {
    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)
{
  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 */

int i;

void EXTI0_1_IRQHandler(void)

{

        // TODO: Debounce using HAL_GetTick()

        static uint32_t lastDebounceTime = 0;

            static uint32_t debounceDelay = 75;  // Adjust this value for
debouncing

            uint32_t currentMillis = HAL_GetTick();
   // Check if the button press is debounced

            if (currentMillis - lastDebounceTime > debounceDelay)

            {

                // Debounce time has passed, process the button press

                // Disable DMA transfer and abort it

                __HAL_TIM_DISABLE_DMA(&htim2, TIM_DMA_CC1);

                HAL_DMA_Abort_IT(&hdma_tim2_ch1);


                // Re-enable DMA with the new source address

                HAL_DMA_Start_IT(&hdma_tim2_ch1, DestAddress,
(uint32_t)&(TIM3->CCR3), NS);

                // Update the debounce time
```

```
                    lastDebounceTime = currentMillis;

                }


        // TODO: Disable DMA transfer and abort IT, then start DMA in IT
mode with new LUT and re-enable transfer

            //HAL_DMA_DISABLE(&hdma_tim2_ch1, (uint32_t)sin_LUT,
DestAddress, NS);

        __HAL_TIM_DISABLE_DMA(&htim2, TIM_DMA_CC1);

        HAL_DMA_Abort_IT(&hdma_tim2_ch1);

       switch (j){

    case 0:

    lcd_command(CLEAR);

        lcd_putstring("SAW WAVE ;)");

    // Change to the next waveform (e.g., Sawtooth)

//DestAddress = (uint32_t)saw_LUT;

   // TODO: Start DMA in& IT mode with the new LUT

   __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

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

      j=1;

        break;

  case 1:

          lcd_command(CLEAR);

    lcd_putstring("TRIANGLE WAVE :|");

        // Change to the next waveform (e.g., Sawtooth)

        //DestAddress = (uint32_t)triangle_LUT;

           // TODO: Start DMA in IT mode with the new LUT

        __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

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

     j=2;

      break;
     case 2:

             lcd_command(CLEAR);

      lcd_putstring("SINE WAVE :)");

            // Change to the next waveform (e.g., Sawtooth)

           //DestAddress = (uint32_t)sin_LUT;

          // TODO: Start DMA in IT mode with the new LUT

        __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

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

     j=0;
 break;

                           // Add more cases if needed

                       }

        // HINT: Consider using C's "switch" function to handle LUT changes


        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 */
```

```
}
```

```c
#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 */
```