

Link to Github Repo:

<https://github.com/qailahb/3096-Pracs>

### Submitted Code:

```
/* 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 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, 636, 661, 684, 708, 731, 753, 775, 796,
817, 837, 856, 874, 891, 908, 923, 938, 951, 964, 975, 985, 994, 1002, 1009, 1014, 1018,
1022, 1023, 1024, 1023, 1022, 1018, 1014, 1009, 1002, 994, 985, 975, 964, 951, 938, 923,
908, 891, 874, 856, 837, 817, 796, 775, 753, 731, 708, 684, 661, 636, 612, 587, 562,
537, 512, 487, 462, 437, 412, 388, 363, 340, 316, 293, 271, 249, 228, 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, 228, 249, 271, 293, 316, 340,
363, 388, 412, 437, 462, 487};
```

```
uint32_t saw_LUT[NS] = {0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120,
128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248,
256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376,
384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504,
512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632,
640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760,
768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888,
896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008,
1016};
```

```
uint32_t triangle_LUT[NS] = {0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208,
224, 240,
256, 272, 288, 304, 320, 336, 352, 368, 384, 400, 416, 432, 448, 464, 480, 496,
512, 528, 544, 560, 576, 592, 608, 624, 640, 656, 672, 688, 704, 720, 736, 752,
768, 784, 800, 816, 832, 848, 864, 880, 896, 912, 928, 944, 960, 976, 992, 1008,
1024, 1008, 992, 976, 960, 944, 928, 912, 896, 880, 864, 848, 832, 816, 800, 784,
768, 752, 736, 720, 704, 688, 672, 656, 640, 624, 608, 592, 576, 560, 544, 528,
512, 496, 480, 464, 448, 432, 416, 400, 384, 368, 352, 336, 320, 304, 288, 272,
256, 240, 224, 208, 192, 176, 160, 144, 128, 112, 96, 80, 64, 48, 32, 16};
```

```
// TODO: Equation to calculate TIM2_Ticks
```

```
uint32_t TIM2_Ticks = (uint32_t) (TIM2CLK/(NS*F_SIGNAL)); // Calculation of TIM2_Ticks
uint32_t DestinationAddress = (uint32_t) &(TIM3->CCR3); // Writes new LUT values
```

```
/* 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 Destination is TIM3->CCR3;
start with Sine LUT
HAL_DMA_Start_IT(&hdma_tim2_ch1,(uint32_t)Sin_LUT,DestinationAddress,NS);

// TODO: Write current waveform to LCD ("Sine")
delay(3000);
lcd_command(CLEAR); //clear LCD
lcd_putstr("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;
    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)
{
    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 */
uint32_t buttonPushed = 0;
unsigned long debounceDelay = 200;
uint32_t lastTick = 0;

// TODO:
void EXTI0_1_IRQHandler(void) {
    // TODO: Debounce using HAL_GetTick()

```

```

uint32_t tick = HAL_GetTick(); // Debouncing

if (tick - lastTick > debounceDelay) {
    // Debounce the button press

// 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
    // Stop the DMA transfer
    __HAL_TIM_DISABLE_DMA(&htim2, TIM_DMA_CC1); // Disable DMA transfer
    HAL_DMA_Abort_IT(&hdma_tim2_ch1);           // Abort IT

    // Changes the source address of the DMA based on the current waveform
    switch (buttonPushed) {
        case 0:
            // Switch to Sawtooth wave
            HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)saw_LUT, DestinationAddress, NS);
            lcd_command(CLEAR); // Clears LCD screen
            lcd_putstring("Sawtooth"); // Prints to LCD screen
            buttonPushed = 1; // Sets parameter value to the switch
            break;
        case 1:
            // Switch to Triangle wave
            HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)triangle_LUT, DestinationAddress, NS);
            lcd_command(CLEAR);
            lcd_putstring("Triangle");
            buttonPushed = 2;
            break;
        case 2:
        default:
            // Switch back to Sine wave
            HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)Sin_LUT, DestinationAddress, NS);
            lcd_command(CLEAR);
            lcd_putstring("Sine");
            buttonPushed = 0;
            break;
    }

    // Re-enable the DMA transfer
    __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

    lastTick = tick;
}

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 */

```

```
#ifndef 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 */
```