

```

1/* USER CODE BEGIN Header */
2/**
3 *****
4 * @file      : main.c
5 * @brief     : Main program body
6 *****
7 * @attention
8 *
9 * Copyright (c) 2023 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 *****
17 */
18/* USER CODE END Header */
19/* Includes -----*/
20#include "main.h"
21//Github link
22//https://github.com/aimeesimons/NDXDAN019_SMNAIM002_EEE3096S.git
23/* Private includes -----*/
24/* USER CODE BEGIN Includes */
25#include <stdio.h>
26#include "stm32f0xx.h"
27#include <lcd_stm32f0.c>
28/* USER CODE END Includes */
29
30/* Private typedef -----*/
31/* USER CODE BEGIN PTD */
32
33/* USER CODE END PTD */
34
35/* Private define -----*/
36/* USER CODE BEGIN PD */
37// TODO: Add values for below variables
38#define NS 128 // Number of samples in LUT
39#define TIM2CLK 8000000 // STM Clock frequency
40#define F_SIGNAL 100 // Frequency of output analog signal
41/* USER CODE END PD */
42
43/* Private macro -----*/
44/* USER CODE BEGIN PM */
45
46/* USER CODE END PM */
47
48/* Private variables -----*/
49TIM_HandleTypeDef htim2;
50TIM_HandleTypeDef htim3;
51DMA_HandleTypeDef hdma_tim2_ch1;
52
53/* USER CODE BEGIN PV */
54// TODO: Add code for global variables, including LUTs
55
56uint32_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

```

```

76,986,995,1002,1009,1014,1018,1021,1023,1023,1022,1020,1016,1012,1006,999,990,981,970,959,946
,932,917,901,884,866,848,828,808,787,765,743,720,697,673,649,624,600,575,549,524,499,474,448,4
23,399,374,350,326,303,280,258,236,215,195,175,157,139,122,106,91,77,64,53,42,33,24,17,11,7,3,
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;
57
58uint32_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,
217,226,234,242,250,258,266,274,282,290,298,306,314,322,330,338,346,354,362,371,379,387,395,40
3,411,419,427,435,443,451,459,467,475,483,491,499,507,516,524,532,540,548,556,564,572,580,588,
596,604,612,620,628,636,644,652,661,669,677,685,693,701,709,717,725,733,741,749,757,765,773,78
1,789,797,806,814,822,830,838,846,854,862,870,878,886,894,902,910,918,926,934,942,951,959,967,
975,983,991,999,1007,1015,0;
59
60uint32_t triangle_LUT[NS] =
    0,16,32,48,64,81,97,113,129,145,161,177,193,209,226,242,258,274,290,306,322,338,354,371,387,4
03,419,435,451,467,483,499,516,532,548,564,580,596,612,628,644,661,677,693,709,725,741,757,773
,789,806,822,838,854,870,886,902,918,934,951,967,983,999,1015,1015,999,983,967,951,934,918,902
,886,870,854,838,822,806,789,773,757,741,725,709,693,677,661,644,628,612,596,580,564,548,532,5
16,499,483,467,451,435,419,403,387,371,354,338,322,306,290,274,258,242,226,209,193,177,161,145
,129,113,97,81,64,48,32,16,0;
61
62uint32_t start = 0;
63int count = 1;
64// TODO: Equation to calculate TIM2_Ticks
65uint32_t TIM2_Ticks = TIM2CLK/(F_SIGNAL*NS); // How often to write new LUT value
66uint32_t DestAddress = (uint32_t)&(TIM3->CCR3); // Write LUT TO TIM3->CCR3 to modify PWM duty
    cycle
67
68/* USER CODE END PV */
69
70/* Private function prototypes -----*/
71void SystemClock_Config(void);
72static void MX_GPIO_Init(void);
73static void MX_DMA_Init(void);
74static void MX_TIM2_Init(void);
75static void MX_TIM3_Init(void);
76
77/* USER CODE BEGIN PFP */
78void EXTI0_1_IRQHandler(void);
79/* USER CODE END PFP */
80
81/* Private user code -----*/
82/* USER CODE BEGIN 0 */
83
84/* USER CODE END 0 */
85
86/**
87 * @brief The application entry point.
88 * @retval int
89 */
90int main(void)
91{
92    /* USER CODE BEGIN 1 */
93    /* USER CODE END 1 */
94
95    /* MCU Configuration-----*/

```

```

96
97 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
98 HAL_Init();
99
100 /* USER CODE BEGIN Init */
101 init_LCD();
102 /* USER CODE END Init */
103
104 /* Configure the system clock */
105 SystemClock_Config();
106
107 /* USER CODE BEGIN SysInit */
108 /* USER CODE END SysInit */
109
110 /* Initialize all configured peripherals */
111 MX_GPIO_Init();
112 MX_DMA_Init();
113 MX_TIM2_Init();
114 MX_TIM3_Init();
115
116 /* USER CODE BEGIN 2 */
117 // TODO: Start TIM3 in PWM mode on channel 3
118 HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
119
120 // TODO: Start TIM2 in Output Compare (OC) mode on channel 1.
121 HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_1);
122
123 // TODO: Start DMA in IT mode on TIM2->CH1; Source is LUT and Dest is TIM3->CCR3; start with
Sine LUT
124
125 HAL_DMA_Start_IT(&hdma_tim2_ch1, Sin_LUT, DestAddress, NS);
126
127 // TODO: Write current waveform to LCD ("Sine")
128 lcd_putstring("Sine");
129 delay(3000);
130
131 // TODO: Enable DMA (start transfer from LUT to CCR)
132 __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
133
134 /* USER CODE END 2 */
135
136 /* Infinite loop */
137 /* USER CODE BEGIN WHILE */
138 while (1)
139 {
140     /* USER CODE END WHILE */
141
142     /* USER CODE BEGIN 3 */
143 }
144 /* USER CODE END 3 */
145 }
146
147 /**
148  * @brief System Clock Configuration
149  * @retval None
150  */
151 void SystemClock_Config(void)

```

```

152 {
153     LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
154     while LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0
155     {
156     }
157     LL_RCC_HSI_Enable();
158
159     /* Wait till HSI is ready */
160     while LL_RCC_HSI_IsReady() != 1
161     {
162     }
163
164     LL_RCC_HSI_SetCalibTrimming(16);
165     LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
166     LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
167     LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
168
169     /* Wait till System clock is ready */
170     while LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI
171     {
172     }
173
174     LL_SetSystemCoreClock(8000000);
175
176     /* Update the time base */
177     if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
178     {
179         Error_Handler();
180     }
181 }
182
183 /**
184  * @brief TIM2 Initialization Function
185  * @param None
186  * @retval None
187  */
188 static void MX_TIM2_Init(void)
189 {
190
191     /* USER CODE BEGIN TIM2_Init 0 */
192
193     /* USER CODE END TIM2_Init 0 */
194
195     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
196     TIM_MasterConfigTypeDef sMasterConfig = {0};
197     TIM_OC_InitTypeDef sConfigOC = {0};
198
199     /* USER CODE BEGIN TIM2_Init 1 */
200
201     /* USER CODE END TIM2_Init 1 */
202     htim2.Instance = TIM2;
203     htim2.Init.Prescaler = 0;
204     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
205     htim2.Init.Period = TIM2_Ticks - 1;
206     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
207     htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
208     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)

```

```

209 {
210     Error_Handler();
211 }
212 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
213 if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
214 {
215     Error_Handler();
216 }
217 if (HAL_TIM_OC_Init(&htim2) != HAL_OK)
218 {
219     Error_Handler();
220 }
221 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
222 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
223 if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
224 {
225     Error_Handler();
226 }
227 sConfigOC.OCMode = TIM_OCMODE_TIMING;
228 sConfigOC.Pulse = 0;
229 sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
230 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
231 if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
232 {
233     Error_Handler();
234 }
235 /* USER CODE BEGIN TIM2_Init 2 */
236
237 /* USER CODE END TIM2_Init 2 */
238
239 }
240
241 /**
242  * @brief TIM3 Initialization Function
243  * @param None
244  * @retval None
245  */
246 static void MX_TIM3_Init(void)
247 {
248
249     /* USER CODE BEGIN TIM3_Init 0 */
250
251     /* USER CODE END TIM3_Init 0 */
252
253     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
254     TIM_MasterConfigTypeDef sMasterConfig = {0};
255     TIM_OC_InitTypeDef sConfigOC = {0};
256
257     /* USER CODE BEGIN TIM3_Init 1 */
258
259     /* USER CODE END TIM3_Init 1 */
260     htim3.Instance = TIM3;
261     htim3.Init.Prescaler = 0;
262     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
263     htim3.Init.Period = 1023;
264     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
265     htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;

```

```

266 if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
267 {
268     Error_Handler();
269 }
270 sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
271 if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
272 {
273     Error_Handler();
274 }
275 if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
276 {
277     Error_Handler();
278 }
279 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
280 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
281 if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
282 {
283     Error_Handler();
284 }
285 sConfigOC.OCMode = TIM_OCMODE_PWM1;
286 sConfigOC.Pulse = 0;
287 sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
288 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
289 if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
290 {
291     Error_Handler();
292 }
293 /* USER CODE BEGIN TIM3_Init 2 */
294
295 /* USER CODE END TIM3_Init 2 */
296 HAL_TIM_MspPostInit(&htim3);
297
298
299
300 /**
301  * Enable DMA controller clock
302  */
303 static void MX_DMA_Init(void)
304 {
305
306     /* DMA controller clock enable */
307     HAL_RCC_DMA1_CLK_ENABLE();
308
309     /* DMA interrupt init */
310     /* DMA1_Channel4_5_IRQn interrupt configuration */
311     HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
312     HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);
313
314
315
316 /**
317  * @brief GPIO Initialization Function
318  * @param None
319  * @retval None
320  */
321 static void MX_GPIO_Init(void)
322 {

```

```

323 LL_EXTI_InitTypeDef EXTI_InitStructure = {0};
324 /* USER CODE BEGIN MX_GPIO_Init_1 */
325 /* USER CODE END MX_GPIO_Init_1 */
326
327 /* GPIO Ports Clock Enable */
328 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
329 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
330 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
331
332 /**/
333 LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
334
335 /**/
336 LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
337
338 /**/
339 LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
340
341 /**/
342 EXTI_InitStructure.Line_0_31 = LL_EXTI_LINE_0;
343 EXTI_InitStructure.LineCommand = ENABLE;
344 EXTI_InitStructure.Mode = LL_EXTI_MODE_IT;
345 EXTI_InitStructure.Trigger = LL_EXTI_TRIGGER_RISING;
346 LL_EXTI_Init(&EXTI_InitStructure);
347
348 /* USER CODE BEGIN MX_GPIO_Init_2 */
349 HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
350 HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
351 /* USER CODE END MX_GPIO_Init_2 */
352 }
353
354 /* USER CODE BEGIN 4 */
355 void EXTI0_1_IRQHandler(void)
356 {
357     // TODO: Debounce using HAL_GetTick()
358     if HAL_GetTick() - start > 1000 {
359         count += 1;
360         if (count == 4) {
361             count = 1;
362         }
363         // TODO: Disable DMA transfer and abort IT, then start DMA in IT mode with new LUT and re-
enable transfer
364         // HINT: Consider using C's "switch" function to handle LUT changes
365         __HAL_TIM_DISABLE_DMA(&htim2, TIM_DMA_CC1);
366         HAL_DMA_Abort_IT(&hdma_tim2_ch1);
367         switch (count) {
368
369             case 1:
370                 HAL_DMA_Start_IT(&hdma_tim2_ch1, Sin_LUT, DestAddress, NS);
371                 lcd_command CLEAR;
372                 lcd_putstring "Sine";
373                 delay(3000);
374                 __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
375                 break;
376             case 2:
377                 HAL_DMA_Start_IT(&hdma_tim2_ch1, saw_LUT, DestAddress, NS);
378                 lcd_command CLEAR;

```

```

379     lcd_putstring("Sawtooth");
380     delay(3000);
381     __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
382     break;
383     case 3:
384         HAL_DMA_Start_IT(&hdma_tim2_ch1, triangle_LUT, DestAddress, NS);
385         lcd_command CLEAR;
386         lcd_putstring("Triangular");
387         delay(3000);
388         __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
389         break;
390     }
391
392     start = HAL_GetTick();
393 }
394 HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
395
396 /* USER CODE END 4 */
397
398 /**
399  * @brief This function is executed in case of error occurrence.
400  * @retval None
401  */
402 void Error_Handler(void)
403 {
404     /* USER CODE BEGIN Error_Handler_Debug */
405     /* User can add his own implementation to report the HAL error return state */
406     __disable_irq();
407     while (1)
408     {
409     }
410     /* USER CODE END Error_Handler_Debug */
411 }
412
413 #ifndef USE_FULL_ASSERT
414 /**
415  * @brief Reports the name of the source file and the source line number
416  *        where the assert_param error has occurred.
417  * @param file: pointer to the source file name
418  * @param line: assert_param error line source number
419  * @retval None
420  */
421 void assert_failed(uint8_t *file, uint32_t line)
422 {
423     /* USER CODE BEGIN 6 */
424     /* User can add his own implementation to report the file name and line number,
425        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
426     /* USER CODE END 6 */
427 }
428 #endif /* USE_FULL_ASSERT */
429

```