```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "math.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_rx;
DMA_HandleTypeDef hdma_usart1_tx;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
```

```c
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM3_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC2_Init(void);
/* USER CODE BEGIN PFP */

typedef uint32_t u32;
const float refFreq = 25000000.0;

void StartDefaultTask(void);
void AD9833reset(void);
//void AD9833setFrequency(uint16_t frequency, uint16_t waveform);
void WriteRegister (uint16_t data,uint16_t data2);
void delay_us(u32 nus);
void AD9833setNote(uint16_t frequency);
uint16_t selectWave1(uint8_t w);
uint16_t selectWave2(uint8_t w);
uint16_t  readLFO(void);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int step = 0;
uint16_t pwm_value = 0;
void user_pwm_setvalue(uint16_t value);
uint8_t in[1];
uint8_t nt[1];    // note
uint8_t vl[1];    // velocity

char vt[4];
uint8_t tx_buff[] = {0,1,2,3};
uint8_t rx_buff[4];
uint16_t notes[] = {16,17,18,19,20,110,22,23,24,27,29,30,33,
                    34,36,39,41,43,46,48,51,55,58,65,69,73,
                    77,82,87,92,97,103,110,116,123,130,138,
                    146,155,168,174,185,196,208,220,260,280,300,
                    310,340,360,390,400,410,440,480,500,520,540          // frequências das notas
};



/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */


```

```
124   /* MCU Configuration--------------------------------------------------------*/
125
126   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
127   HAL_Init();
128
129   /* USER CODE BEGIN Init */
130
131   /* USER CODE END Init */
132
133   /* Configure the system clock */
134   SystemClock_Config();
135
136   /* USER CODE BEGIN SysInit */
137
138   /* USER CODE END SysInit */
139
140   /* Initialize all configured peripherals */
141   MX_GPIO_Init();
142   MX_DMA_Init();
143   MX_USART1_UART_Init();
144   MX_SPI1_Init();
145   MX_SPI2_Init();
146   MX_TIM3_Init();
147   MX_ADC1_Init();
148   MX_TIM4_Init();
149   MX_ADC2_Init();
150   /* USER CODE BEGIN 2 */
151   HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
152
153
154 StartDefaultTask();
155
156  // HAL_UART_IRQHandler(&huart1);
157
158
159   /* USER CODE END 2 */
160
161   /* Infinite loop */
162   /* USER CODE BEGIN WHILE */
163
164   while (1)
165   {
166     /* USER CODE END WHILE */
167
168     /* USER CODE BEGIN 3 */
169      //   step = readLFO();
170
171
172
173
174     //   user_pwm_setvalue(step);
175
176
177     //  HAL_Delay(10);
178
179   }
180   /* USER CODE END 3 */
181 }
182
183 /**
184   * @brief System Clock Configuration
185   * @retval None
```

```c
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  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_MUL2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses 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_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  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;
```

```
248   hadc1.Init.ContinuousConvMode = DISABLE;
249   hadc1.Init.DiscontinuousConvMode = DISABLE;
250   hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
251   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
252   hadc1.Init.NbrOfConversion = 1;
253   if (HAL_ADC_Init(&hadc1) != HAL_OK)
254   {
255     Error_Handler();
256   }
257   /** Configure Regular Channel
258   */
259   sConfig.Channel = ADC_CHANNEL_0;
260   sConfig.Rank = ADC_REGULAR_RANK_1;
261   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
262   if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
263   {
264     Error_Handler();
265   }
266   /* USER CODE BEGIN ADC1_Init 2 */
267
268   /* USER CODE END ADC1_Init 2 */
269
270 }
271
272 /**
273   * @brief ADC2 Initialization Function
274   * @param None
275   * @retval None
276   */
277 static void MX_ADC2_Init(void)
278 {
279
280   /* USER CODE BEGIN ADC2_Init 0 */
281
282   /* USER CODE END ADC2_Init 0 */
283
284   ADC_ChannelConfTypeDef sConfig = {0};
285
286   /* USER CODE BEGIN ADC2_Init 1 */
287
288   /* USER CODE END ADC2_Init 1 */
289   /** Common config
290   */
291   hadc2.Instance = ADC2;
292   hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
293   hadc2.Init.ContinuousConvMode = DISABLE;
294   hadc2.Init.DiscontinuousConvMode = DISABLE;
295   hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
296   hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
297   hadc2.Init.NbrOfConversion = 1;
298   if (HAL_ADC_Init(&hadc2) != HAL_OK)
299   {
300     Error_Handler();
301   }
302   /** Configure Regular Channel
303   */
304   sConfig.Channel = ADC_CHANNEL_2;
305   sConfig.Rank = ADC_REGULAR_RANK_1;
306   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
307   if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
308   {
309     Error_Handler();
```

```
310   }
311   /* USER CODE BEGIN ADC2_Init 2 */
312
313   /* USER CODE END ADC2_Init 2 */
314
315 }
316
317 /**
318   * @brief SPI1 Initialization Function
319   * @param None
320   * @retval None
321   */
322 static void MX_SPI1_Init(void)
323 {
324
325   /* USER CODE BEGIN SPI1_Init 0 */
326
327   /* USER CODE END SPI1_Init 0 */
328
329   /* USER CODE BEGIN SPI1_Init 1 */
330
331   /* USER CODE END SPI1_Init 1 */
332   /* SPI1 parameter configuration*/
333   hspi1.Instance = SPI1;
334   hspi1.Init.Mode = SPI_MODE_MASTER;
335   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
336   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
337   hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
338   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
339   hspi1.Init.NSS = SPI_NSS_SOFT;
340   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
341   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
342   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
343   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
344   hspi1.Init.CRCPolynomial = 10;
345   if (HAL_SPI_Init(&hspi1) != HAL_OK)
346   {
347     Error_Handler();
348   }
349   /* USER CODE BEGIN SPI1_Init 2 */
350
351   /* USER CODE END SPI1_Init 2 */
352
353 }
354
355 /**
356   * @brief SPI2 Initialization Function
357   * @param None
358   * @retval None
359   */
360 static void MX_SPI2_Init(void)
361 {
362
363   /* USER CODE BEGIN SPI2_Init 0 */
364
365   /* USER CODE END SPI2_Init 0 */
366
367   /* USER CODE BEGIN SPI2_Init 1 */
368
369   /* USER CODE END SPI2_Init 1 */
370   /* SPI2 parameter configuration*/
371   hspi2.Instance = SPI2;
```

```
372   hspi2.Init.Mode = SPI_MODE_MASTER;
373   hspi2.Init.Direction = SPI_DIRECTION_2LINES;
374   hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
375   hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;
376   hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
377   hspi2.Init.NSS = SPI_NSS_SOFT;
378   hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
379   hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
380   hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
381   hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
382   hspi2.Init.CRCPolynomial = 10;
383   if (HAL_SPI_Init(&hspi2) != HAL_OK)
384   {
385     Error_Handler();
386   }
387   /* USER CODE BEGIN SPI2_Init 2 */
388
389   /* USER CODE END SPI2_Init 2 */
390
391 }
392
393 /**
394   * @brief TIM3 Initialization Function
395   * @param None
396   * @retval None
397   */
398 static void MX_TIM3_Init(void)
399 {
400
401   /* USER CODE BEGIN TIM3_Init 0 */
402
403   /* USER CODE END TIM3_Init 0 */
404
405   TIM_MasterConfigTypeDef sMasterConfig = {0};
406   TIM_OC_InitTypeDef sConfigOC = {0};
407
408   /* USER CODE BEGIN TIM3_Init 1 */
409
410   /* USER CODE END TIM3_Init 1 */
411   htim3.Instance = TIM3;
412   htim3.Init.Prescaler = 1;
413   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
414   htim3.Init.Period = 9000-1;
415   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
416   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
417   if (HAL_TIM_OC_Init(&htim3) != HAL_OK)
418   {
419     Error_Handler();
420   }
421   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
422   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
423   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
424   {
425     Error_Handler();
426   }
427   sConfigOC.OCMode = TIM_OCMODE_TIMING;
428   sConfigOC.Pulse = 0;
429   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
430   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
431   if (HAL_TIM_OC_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
432   {
433     Error_Handler();
```

```
434   }
435   /* USER CODE BEGIN TIM3_Init 2 */
436
437   /* USER CODE END TIM3_Init 2 */
438   HAL_TIM_MspPostInit(&htim3);
439
440 }
441
442 /**
443   * @brief TIM4 Initialization Function
444   * @param None
445   * @retval None
446   */
447 static void MX_TIM4_Init(void)
448 {
449
450   /* USER CODE BEGIN TIM4_Init 0 */
451
452   /* USER CODE END TIM4_Init 0 */
453
454   TIM_MasterConfigTypeDef sMasterConfig = {0};
455   TIM_OC_InitTypeDef sConfigOC = {0};
456
457   /* USER CODE BEGIN TIM4_Init 1 */
458
459   /* USER CODE END TIM4_Init 1 */
460   htim4.Instance = TIM4;
461   htim4.Init.Prescaler = 1;
462   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
463   htim4.Init.Period = 9000-1;
464   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
465   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
466   if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
467   {
468     Error_Handler();
469   }
470   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
471   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
472   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
473   {
474     Error_Handler();
475   }
476   sConfigOC.OCMode = TIM_OCMODE_PWM1;
477   sConfigOC.Pulse = 0;
478   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
479   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
480   if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
481   {
482     Error_Handler();
483   }
484   /* USER CODE BEGIN TIM4_Init 2 */
485
486   /* USER CODE END TIM4_Init 2 */
487   HAL_TIM_MspPostInit(&htim4);
488
489 }
490
491 /**
492   * @brief USART1 Initialization Function
493   * @param None
494   * @retval None
495   */
```

```c
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */
                                                    //   baudRate  31250
  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 31250;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */


  /* USER CODE END USART1_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
  /* DMA1_Channel5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);

}

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

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
```

```
558   HAL_GPIO_WritePin(GPIOA, LED_Pin|FSYNC_Pin|GPIO_PIN_12, GPIO_PIN_RESET);
559
560   /*Configure GPIO pin Output Level */
561   HAL_GPIO_WritePin(GPIOB, GT_NOTA_Pin|FSYNC2_Pin, GPIO_PIN_RESET);
562
563   /*Configure GPIO pin : WAVE1_Pin */
564   GPIO_InitStruct.Pin = WAVE1_Pin;
565   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
566   HAL_GPIO_Init(WAVE1_GPIO_Port, &GPIO_InitStruct);
567
568   /*Configure GPIO pins : LED_Pin FSYNC_Pin PA12 */
569   GPIO_InitStruct.Pin = LED_Pin|FSYNC_Pin|GPIO_PIN_12;
570   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
571   GPIO_InitStruct.Pull = GPIO_NOPULL;
572   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
573   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
574
575   /*Configure GPIO pins : GT_NOTA_Pin FSYNC2_Pin */
576   GPIO_InitStruct.Pin = GT_NOTA_Pin|FSYNC2_Pin;
577   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
578   GPIO_InitStruct.Pull = GPIO_NOPULL;
579   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
580   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
581
582   /*Configure GPIO pin : LFO_Pin */
583   GPIO_InitStruct.Pin = LFO_Pin;
584   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
585   HAL_GPIO_Init(LFO_GPIO_Port, &GPIO_InitStruct);
586
587   /*Configure GPIO pin : PA8 */
588   GPIO_InitStruct.Pin = GPIO_PIN_8;
589   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
590   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
591   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
592
593 }
594
595 /* USER CODE BEGIN 4 */
596
597 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
598 {
599     __NOP(); //check if we receive all data
600
601 }
602
603 void user_pwm_setvalue(uint16_t value)
604   {
605       TIM_OC_InitTypeDef sConfigOC;
606
607       sConfigOC.OCMode = TIM_OCMODE_PWM1;
608       sConfigOC.Pulse = value;
609       sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
610       sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
611       HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1);     // pwm
612       HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);                          //pwm
613   }
614
615
616
617 void delay_us(u32 nus)
618 {
619     u32 temp;
```

```
620     while(nus--)
621     {
622        for(temp=0;temp<48;temp++);
623 
624     }
625 }
626 
627 void StartDefaultTask(void)
628  {
629      HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_SET);
630      HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_SET);
631 
632 
633      AD9833reset();
634 
635      WriteRegister(0x2168,0x2168);
636      WriteRegister(0xC000,0xC000);
637 
638      WriteRegister(0x2000,0x2002);   // test
639      AD9833setNote(220);            // test
640 
641 //   for(;;)
642 //     {
643 //
644 //    WriteRegister(selectWave2(1),selectWave2(1));   // Define AD9833's waveform register
    value.
645 //
646 //
647 //     HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
648 //
649 //          if  (in[0] == 0x90)
650 //
651 //       {
652 //
653 //           HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
654 //
655 //            AD9833setNote(notes[in[0]]);
656 //
657 //            HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
658 //          if (in[0] == 0x00)
659 //              AD9833setNote(notes[0xF0]);
660 //       }
661 //
662 //    }
663 
664 
665 
666  }
667 
668 void AD9833reset(void){
669         uint16_t sdata = 0x100;
670         WriteRegister (sdata,sdata);
671         delay_us (10);
672 
673 
674 }
675 
676 
677 
678 void AD9833setNote(uint16_t frequency)
679 {
680 
```

```
681     long FreqWord = (frequency * pow(2, 28)) / refFreq;
682 
683          uint16_t MSB = ((FreqWord & 0xFFFC000) >> 14);    //Only lower 14 bits are used
    for data
684          uint16_t LSB = (FreqWord & 0x3FFF);
685 
686          //Set control bits 15 ande 14 to 0 and 1, respectively, for frequency register 0
687          LSB |= 0x4000;
688          MSB |= 0x4000;
689 
690          WriteRegister(LSB,LSB);                  // Write lower 16 bits to AD9833
    registers
691          WriteRegister(MSB,MSB);                  // Write upper 16 bits to AD9833
    registers.
692                                             // Exit & Reset to SINE, SQUARE or TRIANGLE
693 
694 
695 }
696 
697 void WriteRegister (uint16_t data, uint16_t data2){
698 
699 
700 
701    uint8_t   spi_tx_LB =  (uint8_t) data;  //Low Byte  ;
702    uint8_t   spi_tx_MB =   data >> 8 ;  //High Byte;
703 
704    uint8_t   spi_tx_LB2 =  (uint8_t) data2;  //Low Byte  ;
705    uint8_t   spi_tx_MB2 =   data2 >> 8 ;  //High Byte;
706 
707 
708 
709     HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_RESET);
710     HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_RESET);
711     delay_us (10);
712 
713 
714        HAL_SPI_Transmit(&hspi1, &spi_tx_MB, 1, 1000);
715        HAL_SPI_Transmit(&hspi1, &spi_tx_LB, 1, 1000);
716 
717        HAL_SPI_Transmit(&hspi2, &spi_tx_MB2, 1, 1000);
718        HAL_SPI_Transmit(&hspi2, &spi_tx_LB2, 1, 1000);
719 
720 
721        delay_us (10);
722 
723    HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_SET);
724   HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_SET);
725 
726 
727    ////////////////////////////////////////////////////////////////////////////
728 
729 
730 }
731 
732 uint16_t selectWave1(uint8_t w){
733 
734     uint32_t analog_val_1 = 0;
735 
736     ////////////////////////////////ESCOLHE O TIPO DE ONDA OSCILADOR
    1//////////////////////////////////////////////////////
737 
738    for(;;)
```

```
739     {
740         HAL_ADC_Start(&hadc1);
741 
742         if (HAL_ADC_PollForConversion(&hadc1, 1000000) == HAL_OK)
743                {
744                     analog_val_1 = HAL_ADC_GetValue(&hadc1);
745                     if (analog_val_1 <= 1342)
746                         return 0x2000;             //SINE
747                     else{
748                         if (analog_val_1 <= 2684){
749                             return 0x2002;         // TRIANGLE
750                         }
751                         else{
752                             return 0x2028;        // SQUARE
753                         }
754                     }
755 
756                }
757 
758 
759 
760     }
761 }
762 
763 uint16_t selectWave2(uint8_t w){
764 
765     uint32_t analog_val_1 = 0;
766 
767     ////////////////////////////////ESCOLHE O TIPO DE ONDA OSCILADOR
    2//////////////////////////////////////////////////////
768 
769    for(;;)
770     {
771         HAL_ADC_Start(&hadc2);
772 
773         if (HAL_ADC_PollForConversion(&hadc2, 1000000) == HAL_OK)
774                {
775                     analog_val_1 = HAL_ADC_GetValue(&hadc2);
776                     if (analog_val_1 <= 1342)
777                         return 0x2000;             //SINE
778                     else{
779                         if (analog_val_1 <= 2684){
780                             return 0x2002;         // TRIANGLE
781                         }
782                         else{
783                             return 0x2028;        // SQUARE
784                         }
785                     }
786 
787                }
788 
789     }
790 }
791 
792 
793 
794 
795 
796 /* USER CODE END 4 */
797 
798 /**
799   * @brief  This function is executed in case of error occurrence.
```

```
800   * @retval None
801   */
802 void Error_Handler(void)
803 {
804   /* USER CODE BEGIN Error_Handler_Debug */
805   /* User can add his own implementation to report the HAL error return state */
806
807   /* USER CODE END Error_Handler_Debug */
808 }
809
810 #ifdef  USE_FULL_ASSERT
811 /**
812   * @brief  Reports the name of the source file and the source line number
813   *         where the assert_param error has occurred.
814   * @param  file: pointer to the source file name
815   * @param  line: assert_param error line source number
816   * @retval None
817   */
818 void assert_failed(uint8_t *file, uint32_t line)
819 {
820   /* USER CODE BEGIN 6 */
821   /* User can add his own implementation to report the file name and line number,
822      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
823   /* USER CODE END 6 */
824 }
825 #endif /* USE_FULL_ASSERT */
826
827 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
828
```