```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);  //função para principal -
void AD9833reset(void);       // reseta o ad
void WriteRegister (uint16_t data,uint16_t data2); // SPI para os registradores do AD
void delay_us(u32 nus);
void AD9833setNote(uint16_t frequency); // recebe protocolo midi pela uart e aciona o ad com a frequencia
uint16_t selectWave1(void);  //seleciona forma de onda para o osc 01
uint16_t selectWave2(void);  //seleciona forma de onda para o osc 02
uint16_t  readLFO(void);  // LFO nao finalizado ate o momento mas com pwm ja configurado para posterior uso.

/* 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];
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 */


  /* MCU Configuration--------------------------------------------------------*/
```

```
122
123   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
124   HAL_Init();
125
126   /* USER CODE BEGIN Init */
127
128   /* USER CODE END Init */
129
130   /* Configure the system clock */
131   SystemClock_Config();
132
133   /* USER CODE BEGIN SysInit */
134
135   /* USER CODE END SysInit */
136
137   /* Initialize all configured peripherals */
138   MX_GPIO_Init();
139   MX_DMA_Init();
140   MX_USART1_UART_Init();
141   MX_SPI1_Init();
142   MX_SPI2_Init();
143   MX_TIM3_Init();
144   MX_ADC1_Init();
145   MX_TIM4_Init();
146   MX_ADC2_Init();
147   /* USER CODE BEGIN 2 */
148   HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
149
150
151 StartDefaultTask();
152
153  // HAL_UART_IRQHandler(&huart1);
154
155
156   /* USER CODE END 2 */
157
158   /* Infinite loop */
159   /* USER CODE BEGIN WHILE */
160
161   while (1)
162   {
163     /* USER CODE END WHILE */
164
165     /* USER CODE BEGIN 3 */
166      //   step = readLFO();
167
168
169
170
171     //   user_pwm_setvalue(step);
172
173
174     //  HAL_Delay(10);
175
176   }
177   /* USER CODE END 3 */
178 }
179
180 /**
181   * @brief System Clock Configuration
182   * @retval None
183   */
```

```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;
  hadc1.Init.ContinuousConvMode = DISABLE;
```

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

```c
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief SPI2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI2_Init(void)
{

  /* USER CODE BEGIN SPI2_Init 0 */

  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
```

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

```
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 1;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 9000-1;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &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(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */
  HAL_TIM_MspPostInit(&htim4);

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
```

```
494 {
495
496   /* USER CODE BEGIN USART1_Init 0 */
497                                                         //   baudRate  31250
498   /* USER CODE END USART1_Init 0 */
499
500   /* USER CODE BEGIN USART1_Init 1 */
501
502   /* USER CODE END USART1_Init 1 */
503   huart1.Instance = USART1;
504   huart1.Init.BaudRate = 31250;
505   huart1.Init.WordLength = UART_WORDLENGTH_8B;
506   huart1.Init.StopBits = UART_STOPBITS_1;
507   huart1.Init.Parity = UART_PARITY_NONE;
508   huart1.Init.Mode = UART_MODE_TX_RX;
509   huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
510   huart1.Init.OverSampling = UART_OVERSAMPLING_16;
511   if (HAL_UART_Init(&huart1) != HAL_OK)
512   {
513     Error_Handler();
514   }
515   /* USER CODE BEGIN USART1_Init 2 */
516
517
518   /* USER CODE END USART1_Init 2 */
519
520 }
521
522 /**
523   * Enable DMA controller clock
524   */
525 static void MX_DMA_Init(void)
526 {
527   /* DMA controller clock enable */
528   __HAL_RCC_DMA1_CLK_ENABLE();
529
530   /* DMA interrupt init */
531   /* DMA1_Channel4_IRQn interrupt configuration */
532   HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
533   HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
534   /* DMA1_Channel5_IRQn interrupt configuration */
535   HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);
536   HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);
537
538 }
539
540 /**
541   * @brief GPIO Initialization Function
542   * @param None
543   * @retval None
544   */
545 static void MX_GPIO_Init(void)
546 {
547   GPIO_InitTypeDef GPIO_InitStruct = {0};
548
549   /* GPIO Ports Clock Enable */
550   __HAL_RCC_GPIOD_CLK_ENABLE();
551   __HAL_RCC_GPIOA_CLK_ENABLE();
552   __HAL_RCC_GPIOB_CLK_ENABLE();
553
554   /*Configure GPIO pin Output Level */
555   HAL_GPIO_WritePin(GPIOA, LED_Pin|FSYNC_Pin|GPIO_PIN_12, GPIO_PIN_RESET);
```

```
556
557   /*Configure GPIO pin Output Level */
558   HAL_GPIO_WritePin(GPIOB, GT_NOTA_Pin|FSYNC2_Pin, GPIO_PIN_RESET);
559
560   /*Configure GPIO pin : WAVE1_Pin */
561   GPIO_InitStruct.Pin = WAVE1_Pin;
562   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
563   HAL_GPIO_Init(WAVE1_GPIO_Port, &GPIO_InitStruct);
564
565   /*Configure GPIO pins : LED_Pin FSYNC_Pin PA12 */
566   GPIO_InitStruct.Pin = LED_Pin|FSYNC_Pin|GPIO_PIN_12;
567   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
568   GPIO_InitStruct.Pull = GPIO_NOPULL;
569   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
570   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
571
572   /*Configure GPIO pins : GT_NOTA_Pin FSYNC2_Pin */
573   GPIO_InitStruct.Pin = GT_NOTA_Pin|FSYNC2_Pin;
574   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
575   GPIO_InitStruct.Pull = GPIO_NOPULL;
576   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
577   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
578
579   /*Configure GPIO pin : LFO_Pin */
580   GPIO_InitStruct.Pin = LFO_Pin;
581   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
582   HAL_GPIO_Init(LFO_GPIO_Port, &GPIO_InitStruct);
583
584   /*Configure GPIO pin : PA8 */
585   GPIO_InitStruct.Pin = GPIO_PIN_8;
586   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
587   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
588   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
589
590 }
591
592 /* USER CODE BEGIN 4 */
593
594 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
595 {
596     __NOP(); //check if we receive all data
597
598 }
599
600 void user_pwm_setvalue(uint16_t value)
601   {
602       TIM_OC_InitTypeDef sConfigOC;
603
604       sConfigOC.OCMode = TIM_OCMODE_PWM1;
605       sConfigOC.Pulse = value;
606       sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
607       sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
608       HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1);     // pwm
609       HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);                          //pwm
610   }
611
612
613
614 void delay_us(u32 nus)
615 {
616     u32 temp;
617     while(nus--)
```

```
618     {
619        for(temp=0;temp<48;temp++);
620 
621     }
622 }
623 
624 void StartDefaultTask(void)
625  {
626      HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_SET);
627      HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_SET);
628 
629 
630      AD9833reset();
631 
632      WriteRegister(0x2168,0x2168);
633      WriteRegister(0xC000,0xC000);
634 
635 //   WriteRegister(0x2000,0x2002);   // test descomentar
636 //   AD9833setNote(220);            // test
637 
    //-----------------------------------------------------COMENTAR ABAIXO  TESTE AD
638    for(;;)
639      {
640 
641       WriteRegister(selectWave1(),selectWave2());
642 
643 
644        HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
645 
646           if  (in[0] == 0x90)
647 
648          {
649 
650              HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
651 
652               AD9833setNote(notes[in[0]]);
653 
654               HAL_UART_Receive(&huart1, (uint8_t *)in, 1, 1000);
655             if (in[0] == 0x00)
656                 AD9833setNote(notes[0xF0]);
657          }
658   //        AD9833setNote(220);
659    }
660 
661 
    //-----------------------------------------------------COMENTAR ACIMA TEST AD
662 
663  }
664 
665 void AD9833reset(void){
666          uint16_t sdata = 0x100;
667          WriteRegister (sdata,sdata);
668          delay_us (10);
669 
670 
671 }
672 
673 
674 
675 void AD9833setNote(uint16_t frequency)
676 {
677 
```

```
678     long FreqWord = (frequency * pow(2, 28)) / refFreq;
679 
680          uint16_t MSB = ((FreqWord & 0xFFFC000) >> 14);    //Only lower 14 bits are used
    for data
681          uint16_t LSB = (FreqWord & 0x3FFF);
682 
683          //Set control bits 15 ande 14 to 0 and 1, respectively, for frequency register 0
684          LSB |= 0x4000;
685          MSB |= 0x4000;
686 
687          WriteRegister(LSB,LSB);                  // Write lower 16 bits to AD9833
    registers
688          WriteRegister(MSB,MSB);                  // Write upper 16 bits to AD9833
    registers.
689                                             // Exit & Reset to SINE, SQUARE or TRIANGLE
690 
691 
692 }
693 
694 void WriteRegister (uint16_t data, uint16_t data2){
695 
696 
697 
698    uint8_t   spi_tx_LB =  (uint8_t) data;  //Low Byte  ;
699    uint8_t   spi_tx_MB =   data >> 8 ;  //High Byte;
700 
701    uint8_t   spi_tx_LB2 =  (uint8_t) data2;  //Low Byte  ;
702    uint8_t   spi_tx_MB2 =   data2 >> 8 ;  //High Byte;
703 
704 
705 
706     HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_RESET);
707     HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_RESET);
708     delay_us (10);
709 
710 
711        HAL_SPI_Transmit(&hspi1, &spi_tx_MB, 1, 1000);
712        HAL_SPI_Transmit(&hspi1, &spi_tx_LB, 1, 1000);
713 
714        HAL_SPI_Transmit(&hspi2, &spi_tx_MB2, 1, 1000);
715        HAL_SPI_Transmit(&hspi2, &spi_tx_LB2, 1, 1000);
716 
717 
718        delay_us (10);
719 
720    HAL_GPIO_WritePin(FSYNC_GPIO_Port, FSYNC_Pin, GPIO_PIN_SET);
721   HAL_GPIO_WritePin(FSYNC2_GPIO_Port, FSYNC2_Pin, GPIO_PIN_SET);
722 
723 
724    ////////////////////////////////////////////////////////////////////////////
725 
726 
727 }
728 
729 uint16_t selectWave1(void){
730 
731     uint32_t analog_val_1 = 0;
732 
733     ////////////////////////////////ESCOLHE O TIPO DE ONDA OSCILADOR
    1//////////////////////////////////////////////////////
734 
735    for(;;)
```

```
736     {
737         HAL_ADC_Start(&hadc1);
738 
739         if (HAL_ADC_PollForConversion(&hadc1, 1000000) == HAL_OK)
740                {
741                     analog_val_1 = HAL_ADC_GetValue(&hadc1);
742                     if (analog_val_1 <= 1342)
743                         return 0x2000;             //SINE
744                     else{
745                         if (analog_val_1 <= 2684){
746                         return 0x2002;          // TRIANGLE
747                         }
748                         else{
749                             return 0x2028;        // SQUARE
750                         }
751                     }
752 
753                }
754 
755 
756 
757     }
758 }
759 
760 uint16_t selectWave2(void){
761 
762     uint32_t analog_val_1 = 0;
763 
764     ////////////////////////////////ESCOLHE O TIPO DE ONDA OSCILADOR
    2//////////////////////////////////////////////////////
765 
766    for(;;)
767     {
768         HAL_ADC_Start(&hadc2);
769 
770         if (HAL_ADC_PollForConversion(&hadc2, 1000000) == HAL_OK)
771                {
772                     analog_val_1 = HAL_ADC_GetValue(&hadc2);
773                     if (analog_val_1 <= 1342)
774                         return 0x2000;             //SINE
775                     else{
776                         if (analog_val_1 <= 2684){
777                             return 0x2002;         // TRIANGLE
778                         }
779                         else{
780                             return 0x2028;        // SQUARE
781                         }
782                     }
783 
784                }
785 
786     }
787 }
788 
789 
790 
791 
792 
793 /* USER CODE END 4 */
794 
795 /**
796   * @brief  This function is executed in case of error occurrence.
```

```
797   * @retval None
798   */
799 void Error_Handler(void)
800 {
801   /* USER CODE BEGIN Error_Handler_Debug */
802   /* User can add his own implementation to report the HAL error return state */
803
804   /* USER CODE END Error_Handler_Debug */
805 }
806
807 #ifdef  USE_FULL_ASSERT
808 /**
809   * @brief  Reports the name of the source file and the source line number
810   *         where the assert_param error has occurred.
811   * @param  file: pointer to the source file name
812   * @param  line: assert_param error line source number
813   * @retval None
814   */
815 void assert_failed(uint8_t *file, uint32_t line)
816 {
817   /* USER CODE BEGIN 6 */
818   /* User can add his own implementation to report the file name and line number,
819      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
820   /* USER CODE END 6 */
821 }
822 #endif /* USE_FULL_ASSERT */
823
824 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
825
```