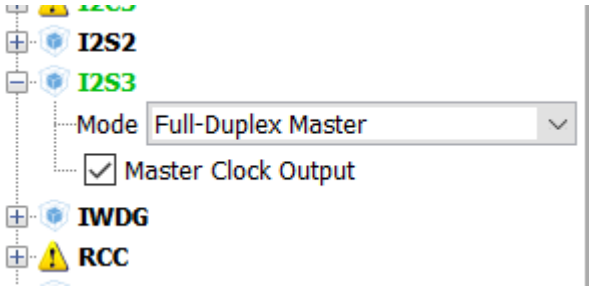
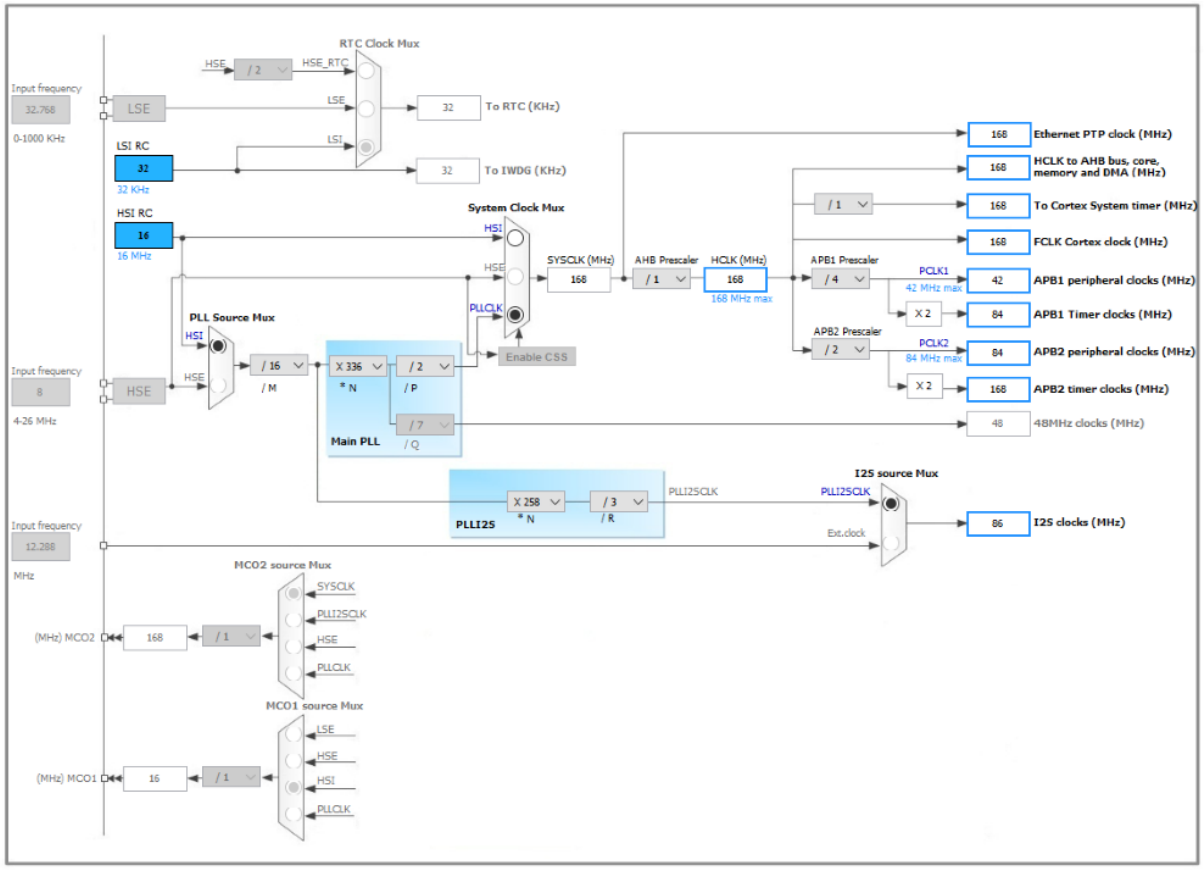
CS43L22: audio DAC

Set up Project configuration in STM32Cube MX

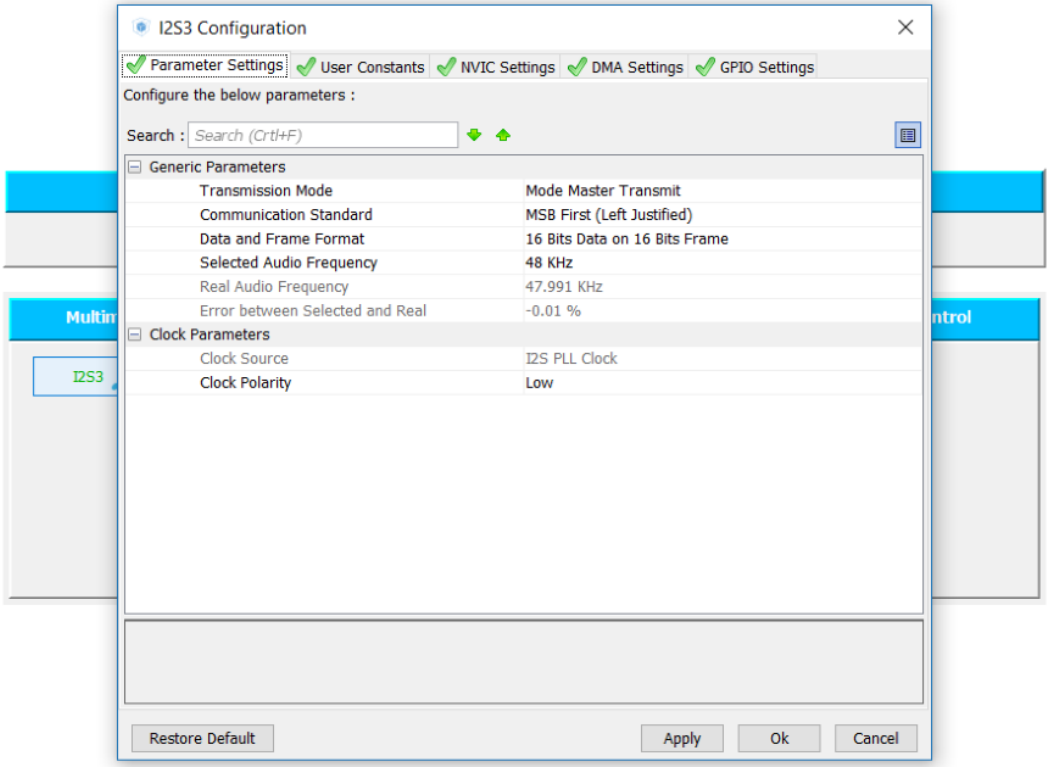
Open I2S Full-Duplex Master Mode with master clock output on.

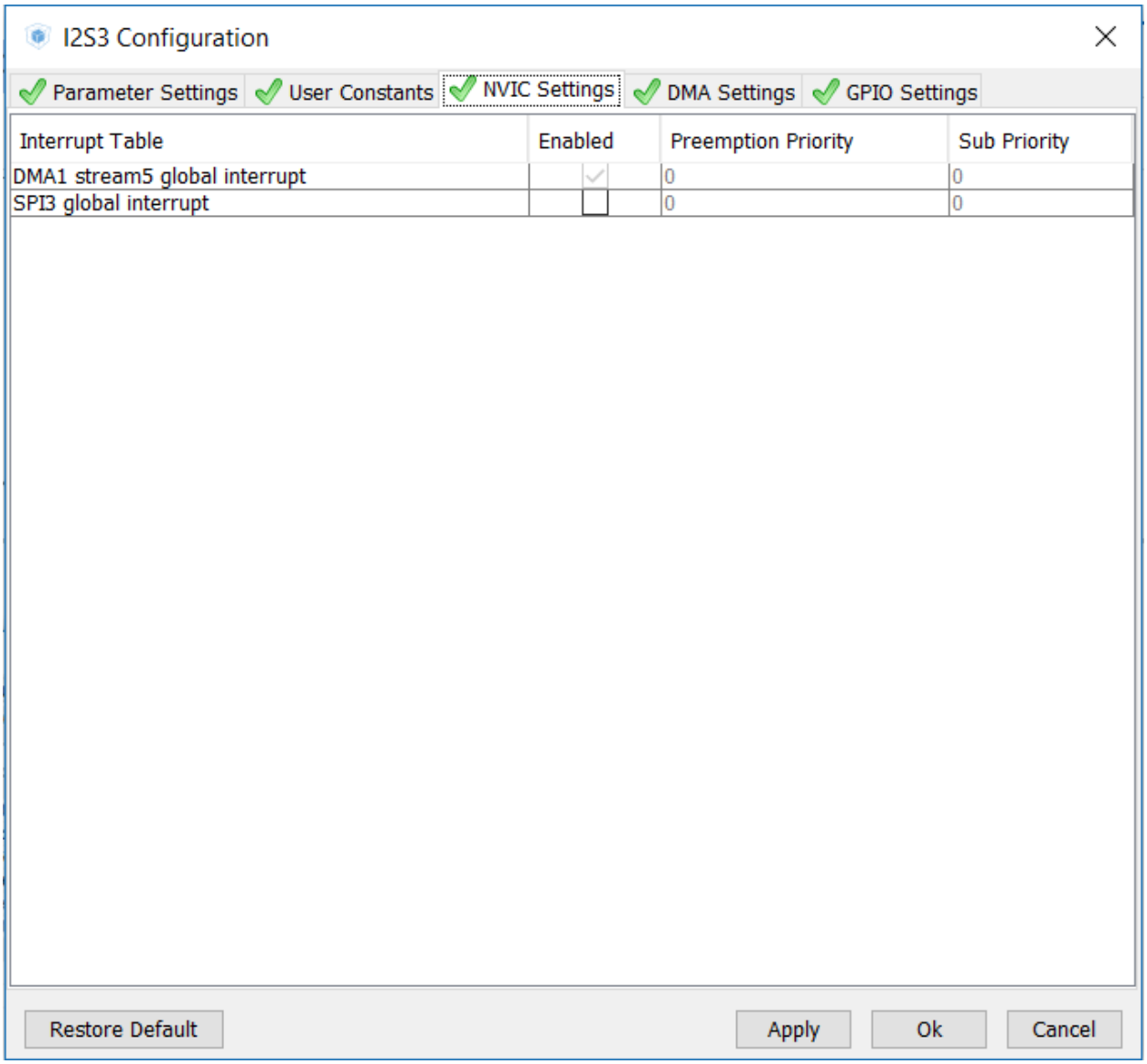


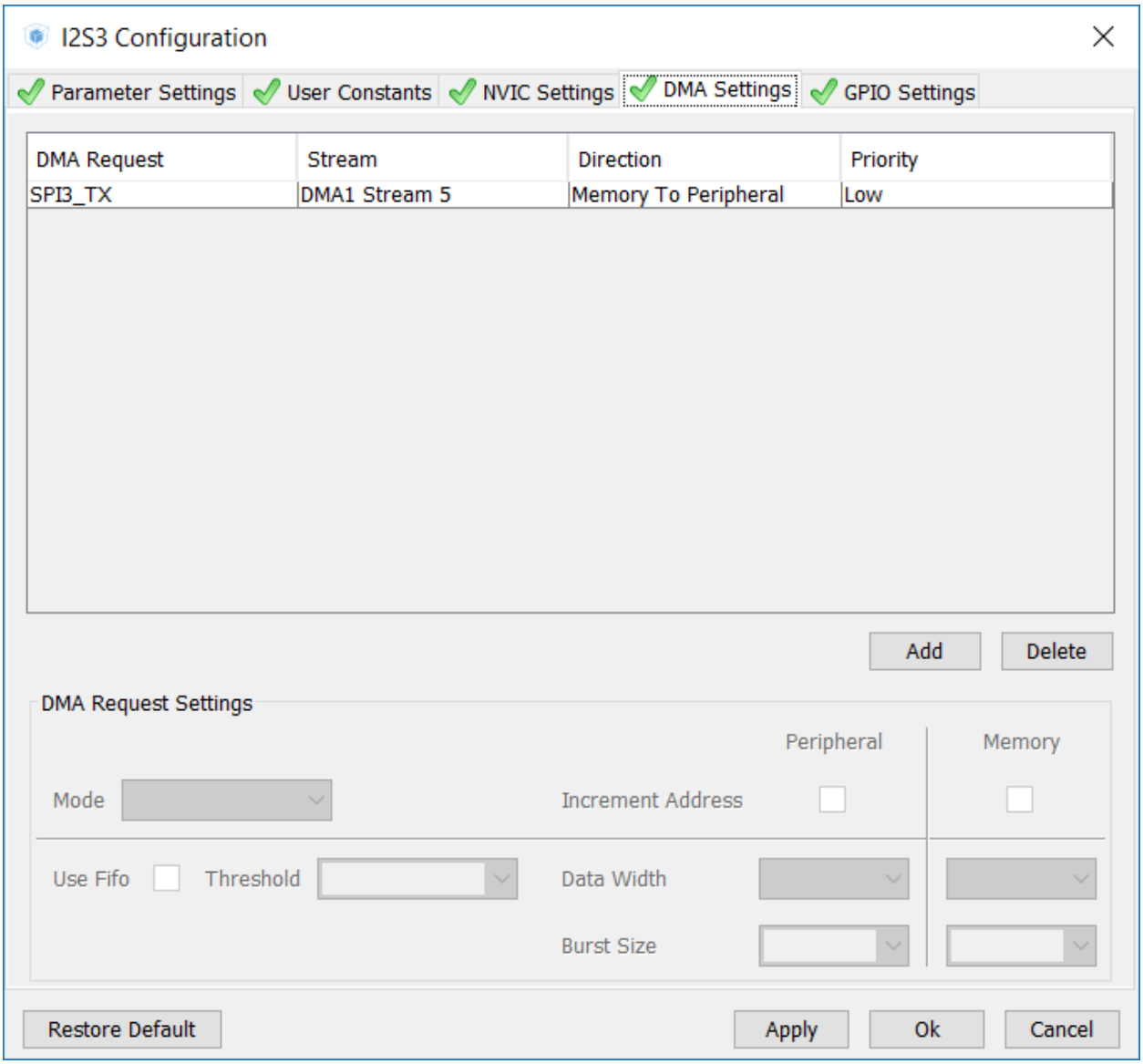
Set up clock for I2S



Setup I2S3 Configuration as image below.


I2S3 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Configure the below parameters :
Search : Search (Crtl+F)
Generic Parameters
Transmission Mode
Mode Master Transmit
Communication Standard
MSB First (Left Justified)
Data and Frame Format
16 Bits Data on 16 Bits Frame
Selected Audio Frequency
48 KHz
Real Audio Frequency
47.991 KHz
Error between Selected and Real
-0.01 %
Clock Parameters
Clock Source
I2S PLL Clock
Clock Polarity
Low
Restore Default
Apply
Ok
Cancel
I2S3


I2S3 Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Interrupt Table
Enabled
Preemption Priority
Sub Priority
DMA1 stream5 global interrupt
0
0
SPI3 global interrupt
0
0
Restore Default
Apply
Ok
Cancel



Code in main.c

Writing code for initialize CS43I22 (from Manual)

```
static void CS43I22_Init(void) {

  // 4.9 Recommended Power-Up Sequence (p. 31)

  // 1. Hold RESET low until the power supplies are stable.
  // HAL_Delay(100);

  // 2. Bring RESET high
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET);

  // 3. The default state of the Power Ctl. 1ï¿½ï¿½?ï¿½?ï¿½? register (0x02) is 0x01. Load the desired register settings while
  // keeping the  register set to 0x01.
  cmd[0] = 0x02;
  cmd[1] = 0x01;
  HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);

  // Load My Setting
  // 7.2 Power Control 1 (p.37)
  // cmd[0] = 0x02;
  // cmd[1] = 0x9E;
  // HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
  // 7.3 Power Control 2 (p.38)
  cmd[0] = 0x04;
  // cmd[1] = 0xAA; /* 1010 1010 ( Headphone channel is always ON. Speaker channel is always ON. )*/
  cmd[1] =  (2 << 6);  // PDN_HPB[0:1]  = 10 (HP-B always on)
  cmd[1] |= (2 << 4);  // PDN_HPA[0:1]  = 10 (HP-A always on)
  cmd[1] |= (3 << 2);  // PDN_SPKB[0:1] = 11 (Speaker B always off)
  cmd[1] |= (3 << 0);  // PDN_SPKA[0:1] = 11 (Speaker A always off)
  HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
  // 7.4 Clocking Control (p. 38)

cmd[0] = 0x05;
cmd[1] = 0x81; // 1000 0001 ( Auto-Detect, Double-Speed Mode (DSM - 50 kHz -100 kHz Fs), MCLK Divide By 2 )
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 7.5 Interface Control 1 (p. 40)
cmd[0] = 0x06;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 100);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1], 1, 100);
cmd[1] &= (1 << 5); // Clear all bits except bit 5 which is reserved
cmd[1] &= ~(1 << 7);  // Slave
cmd[1] &= ~(1 << 6);  // Clock polarity: Not inverted
cmd[1] &= ~(1 << 4);  // No DSP mode
cmd[1] &= ~(1 << 2);  // Left justified, up to 24 bit (default)
cmd[1] |=  (3 << 0);  // 16-bit audio word length for I2S interface
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
cmd[0] = 0x08;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 100);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1], 1, 100);
cmd[1] &= 0xF0;       // Bits [5-7] are reserved
cmd[1] |=  (1 << 0); // Use AIN1A as source for passthrough
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 7.7 Passthrough x Select: PassB (p. 42)
cmd[0] = 0x09;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 100);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1], 1, 100);
cmd[1] &= 0xF0;       // Bits [5-7] are reserved
cmd[1] |=  (1 << 0); // Use AIN1B as source for passthrough
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
```

```
// 7.11 Miscellaneous Controls (Address 0Eh) (p. 44)
cmd[0] = 0x0E;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 100);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1], 1, 100);
cmd[1] &= ~(1 << 7);   // Disable passthrough for AIN-A
cmd[1] &= ~(1 << 6);   // Disable passthrough for AIN-B
cmd[1] |=  (1 << 5);   // Mute passthrough on AIN-A
cmd[1] |=  (1 << 4);   // Mute passthrough on AIN-B
cmd[1] &= ~(1 << 3);   // Changed settings take affect immediately
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 100);
// 7.12 Playback Control 2 (Address 0Fh) (p. 45)
cmd[0] = 0x0F;
cmd[1] = 0x00;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 100);
// 7.23 Limiter Control 1, Min/Max Thresholds  (p. 53)
// cmd[0] = 0x27;
// cmd[1] = 0x00; /* 0000 0000 ( Limiter Maximum Threshold 0 dB, Limiter Cushion Threshold 0 dB ) */
// HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 7.14 PCMx Volume: PCMA (p. 47)
cmd[0] = 0x1A;
cmd[1] = 0x00;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 3, 1000);
// 7.14 PCMx Volume: PCMB (p. 47)
cmd[0] = 0x1B; // | 0x80;
cmd[1] = 0x00; // 0000 1010 ( PCM Channel B Volume )
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);

// 4. Load the required initialization settings listed in Section 4.11
// 4.1. Write 0x99 to register 0x00.
cmd[0] = 0x00;
cmd[1] = 0x99;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 4.2. Write 0x80 to register 0x47
cmd[0] = 0x47;
cmd[1] = 0x80;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 4.3. Write 1b to bit 7 in register 0x32.
cmd[0] = 0x32;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 1000);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1],1,1000);
cmd[1] |= 0x80;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 4.4. Write 0b to bit 7 in register 0x32.
cmd[0] = 0x32;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 1, 1000);
HAL_I2C_Master_Receive(&hi2c1, DAC_I2C_ADDR, &cmd[1],1,1000);
cmd[1] &= ~(0x80);
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
// 4.5. Write 0x00 to register 0x00.
cmd[0] = 0x00;
cmd[1] = 0x00;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);
```

```
// 5. Apply MCLK at the appropriate frequency, as discussed in Section 4.6. SCLK may be applied or set to
// master at any time; LRCK may only be applied or set to master while the PDN bit is set to 1.

// 6. Set the Power Ctl 1 register (0x02) to 0x9E
cmd[0] = 0x02;
cmd[1] = 0x9E;
HAL_I2C_Master_Transmit(&hi2c1, DAC_I2C_ADDR, cmd, 2, 1000);

// 7. Bring RESET low if the analog or digital supplies drop below the recommended operating condition to
// prevent power glitch related issues.
```

Set up variable for writing

```
int i = 0;
const int freq[] = { 183, 163, 145, 137, 122, 109, 97};
uint16_t wav[7][256];
uint8_t regValue = 0xFF;
uint8_t cmd[10];
uint8_t currentKey = 0;
```

freq *is array of Sample size / target frequency (Note C, D, E, F, G, A, B)*

wav *is array for storing sine wave value generate from freq for each note*

i *is left wave counter*

currentKey *is store the note that we're now playing*

Writing callback handler after transfer complete reduce wave counter and if wave counter is more than zero play another wave of current note

```
void HAL_I2S_TxCpltCallback(I2S_HandleTypeDef *hi2s3)
{
  i--;
  if (i > 0)
    HAL_I2S_Transmit_DMA(hi2s3, wav[currentKey], freq[currentKey]);
}
/* USER CODE END 4 */
```

In function main()

*Start calling CS43I22 initialize function and generate sin wave by expression sin(2 * PI / freq * current) * Amplitude*

```
CS43I22_Init();

for (j = 0; j < 7; j++) {
  for (i = 0; i < freq[ j ]; i++) {
    wav[j][i] = sin(2 * PI / freq[j] * i) * 32767;
  }
}
```

In while start polling receive data from UART as English alphabet A to G and start playing note as user input (A is Note C, B is Note D, C is Note E, so on).

```
while (1)
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

  uint8_t data;
  if (HAL_UART_Receive(&huart2, &data, 1, 5) == HAL_OK) {
    if (data - 'A' >= 0 && data - 'A' < 7) {
      HAL_UART_Transmit(&huart2, &data, 1, 1000);
      HAL_UART_Transmit(&huart2, "\n\r", 2, 1000);

      currentKey = data - 'A';
      i = freq[currentKey] * 2;
      HAL_I2S_Transmit_DMA(&hi2s3, wav[currentKey], freq[currentKey]);
    }
  }

}
```