School of Electrical Engineering
Universiti Teknologi Malaysia (UTM)

Assignment: Work Procedure Documentation

Groupmates **:** Phuah Chai Ling
: Lim Calvin
: Nur Nadia binti Muslim
Lecturer's Name **:** Dr. Usman Ullah Sheikh

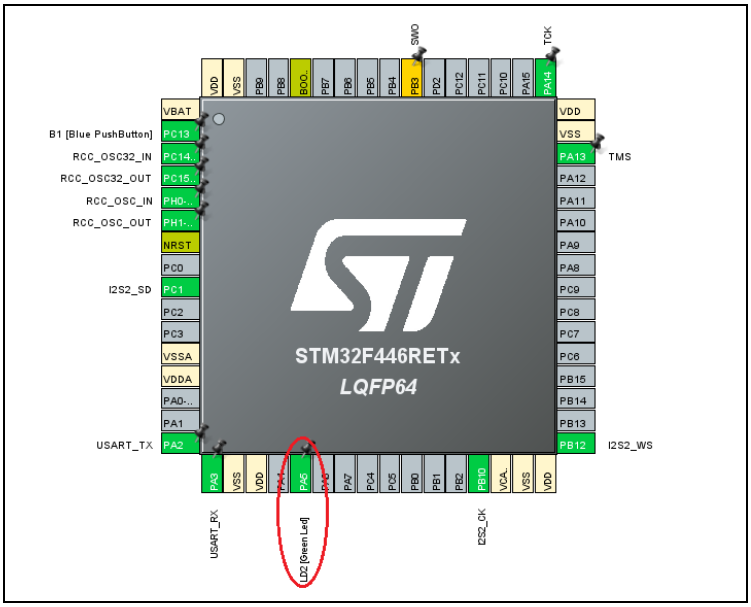
Table of Contents:-

LED (LD2) Blinking on NUCLEO-F446RE (STM32-Board)

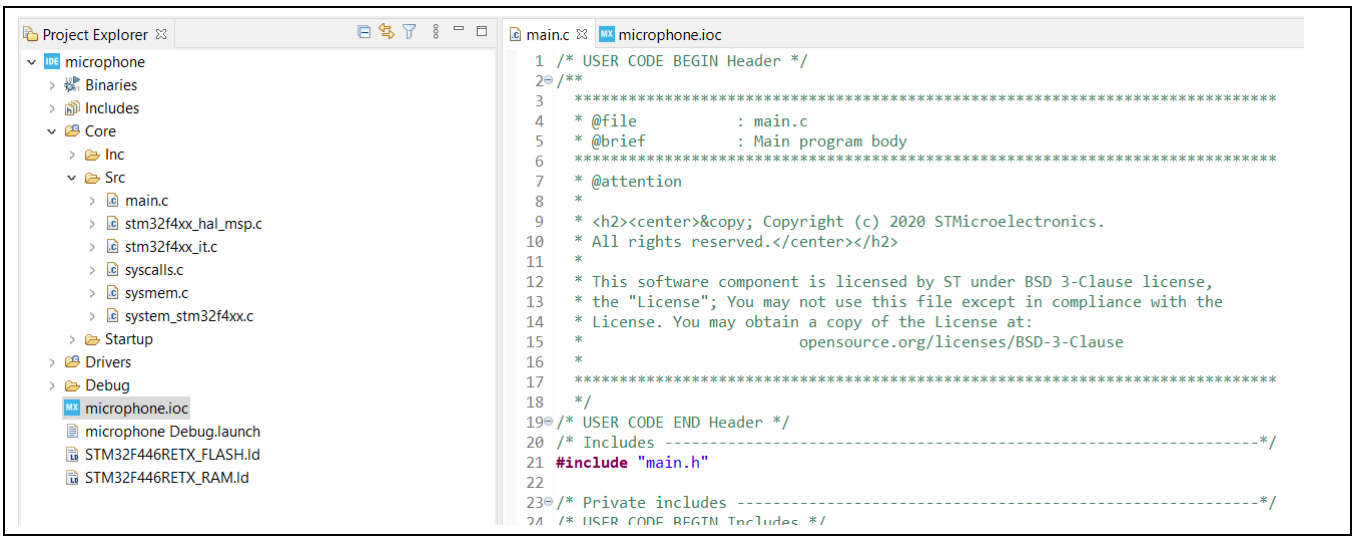
Created by Nur Nadia Muslim, Lim Calvin and Phuah Chai Ling

The peripherals of STM32CubeMX tool has been initialized by witnessing the LED blinking on STM32 board. The LED is set to pull up and down by adhering the 1 hertz(Hz) which is equivalent to 1 seconds(s). Therefore, the ***HAL_Delay*** is set to 1000. The perquisites of initializing the LED blinking at 1Hz has been interpreted based on the following steps according to https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Step2_Blink_LED .

1. The LED (LD2) pin is already assigned as shown below: -



2. Generate project, normally.
3. Save the projects and open it in STM32CubeIDE
4. In STM32CubeIDE interface: -



5. In main.c, add lines as shown below: -

```
5     while (1)
6     {
7         HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
8         HAL_Delay(1000);            // Blink at 1Hz = 1s
9       /* USER CODE END WHILE */
10
11      /* USER CODE BEGIN 3 */
12    }
13    /* USER CODE END 3 */
```

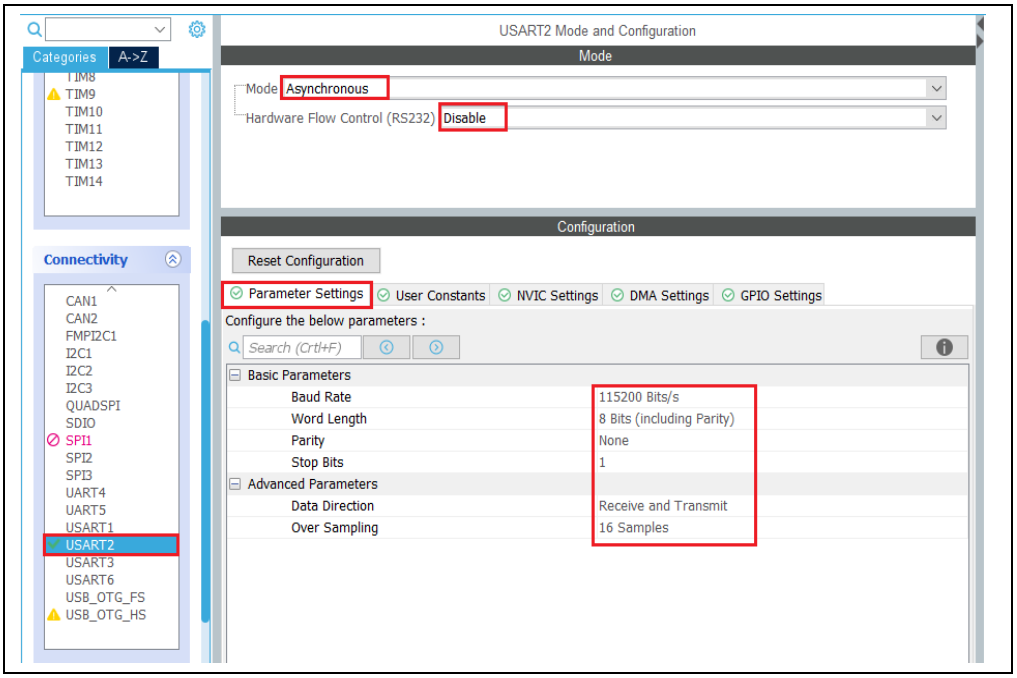
6. Build and run the program in Release mode.
7. Observe the LED blinking at 1Hz on the STM32 board itself.

UART from/to NUCLEO-F446RE (STM32-Board)

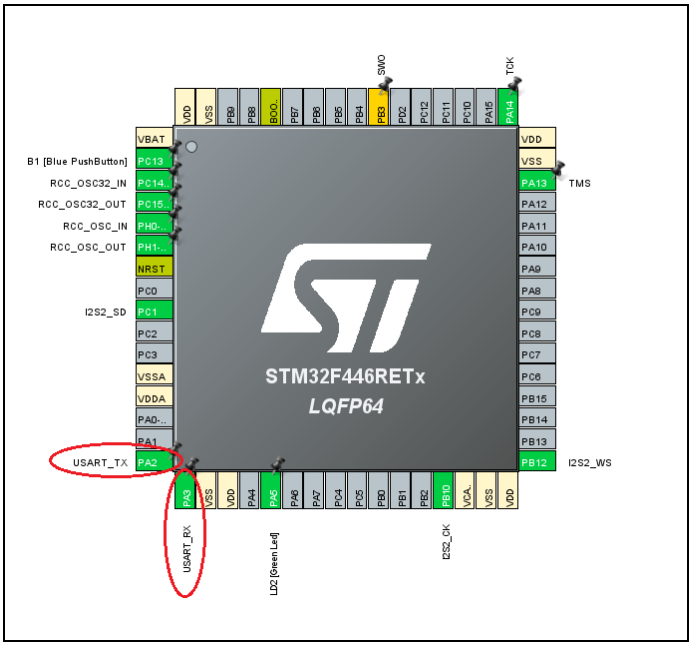
Created by Nur Nadia Muslim, Lim Calvin and Phuah Chai Ling

The basic serial communication of receiving and transmitting the asynchronous logs has been applied. The purpose of assigning USART port into STM32 board is to indicate the output logs shown according to the modified coding regarding on the desired output. The reference for implementing UART into our system is based on https://medium.com/vicara-hardware-university/a-guide-to-transmitting-structures-using-stm32-uart-and-python-56b67f806566 .

1. In STM32CubeMX, choose the settings as shown below:

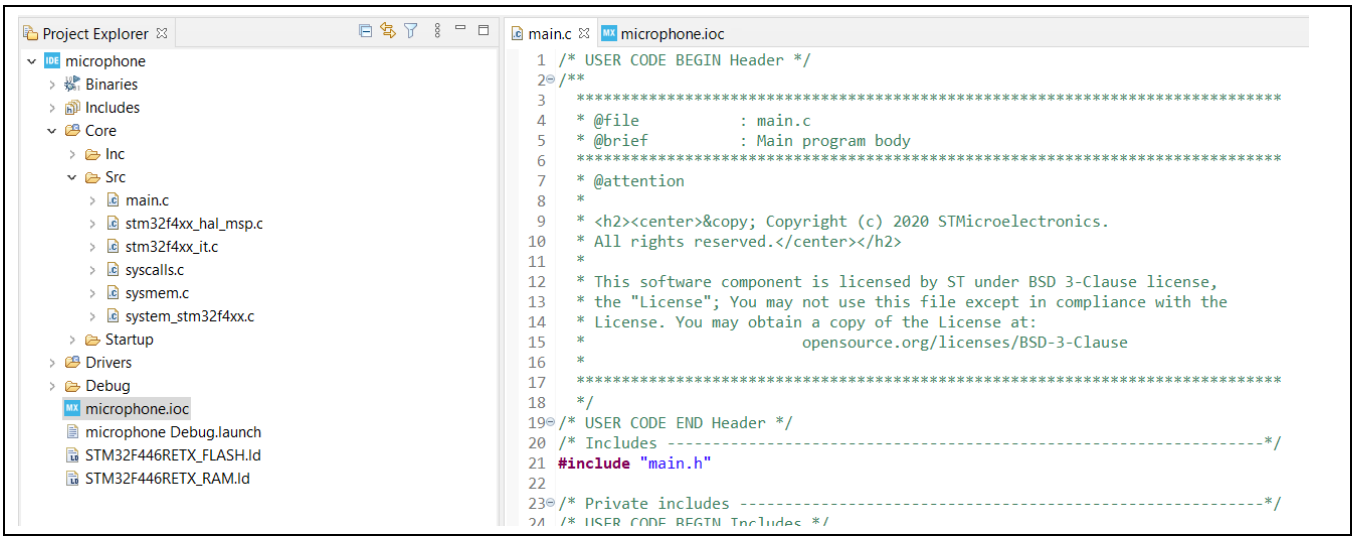


2. The UART pins will be automatically assigned as shown below:



3. Generate project, normally.
4. Save the projects and open it in STM32CubeIDE.

5. In STM32CubeIDE interface:



6. In main.c, add lines as shown below:

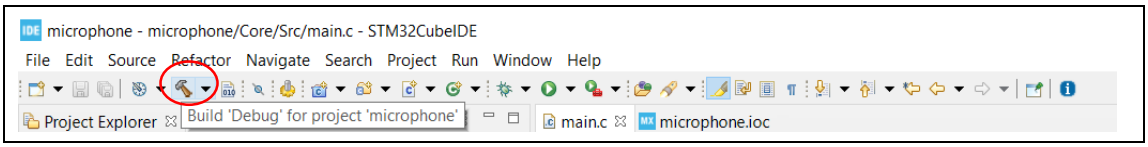
1st way: -

```
/* USER CODE BEGIN Init */
uint8_t myTxData[13] = "Hello World\n";
/* USER CODE END Init */


  while (1)
  {
      HAL_UART_Transmit(&huart2, myTxData, 13, 10);
      HAL_Delay(1000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

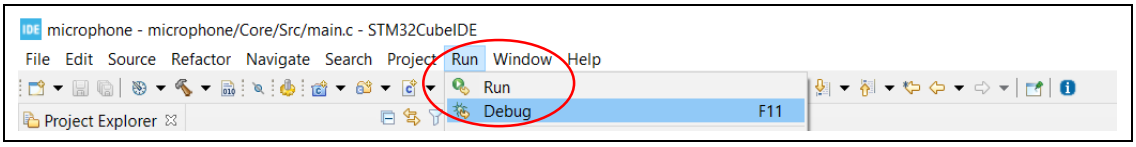
2nd way: -

```
/* USER CODE BEGIN Init */
uint8_t buf[12];
/* USER CODE END Init */

 while (1)
  {

      strcpy((char*)buf, "Hello!\r\n");
      HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);
      HAL_Delay(500);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

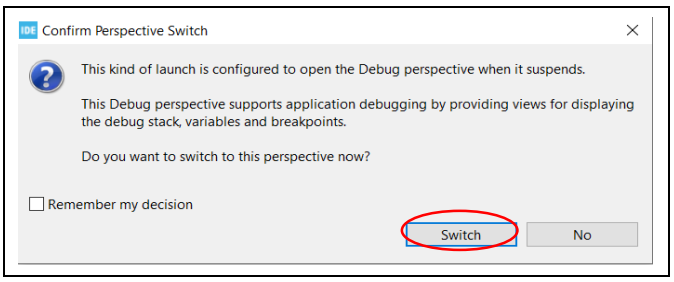
7. Build 'Debug' for project '<your_project_name>'



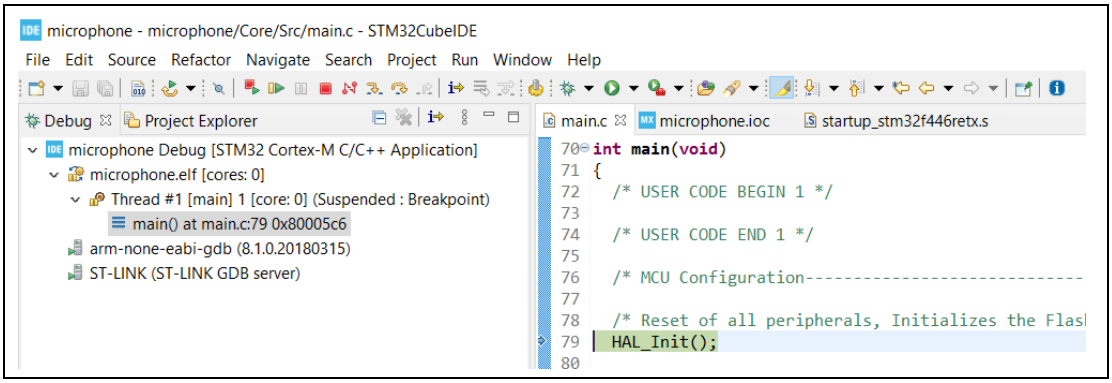
8. Run >> Debug >> Then wait for the program to compiled successfully



9. Click Switch to the debug perspective chosen previously.

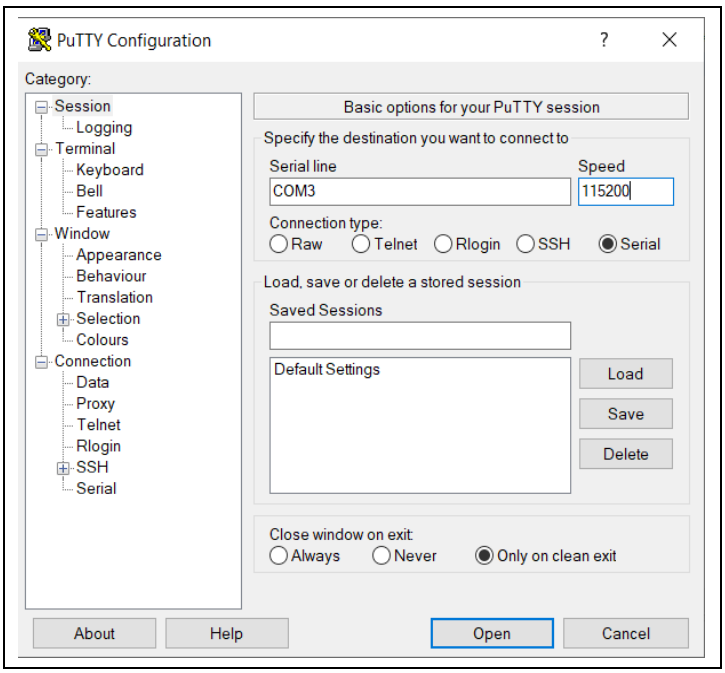


10. The lists interface will changed as shown below:

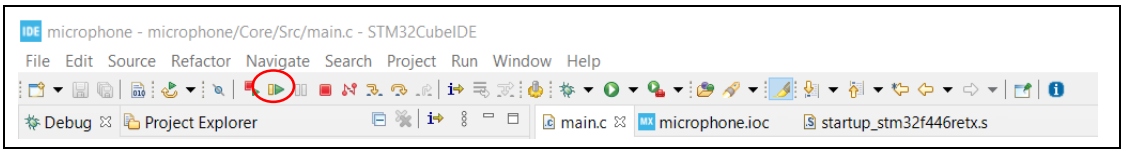


11. In puTTy, choose the setting as shown below.

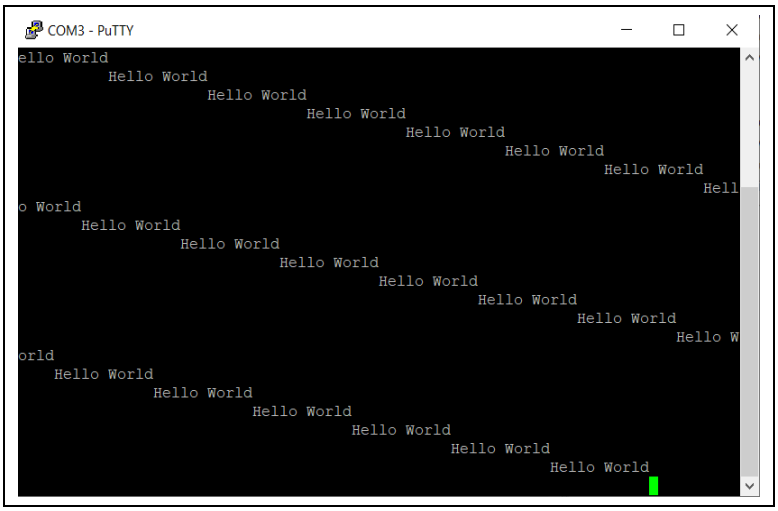
    - Example: COM3 is referred to the NUCLEO-F446RE ports connected to the PC/laptop.



12. Resume (F8) to continue running the program in debug mode.



13. In puTTy, the logs running as shown below:



I2S Integration from INMP441 (microphone) to NUCLEO-F446RE (STM32-board)

Created by Lim Calvin

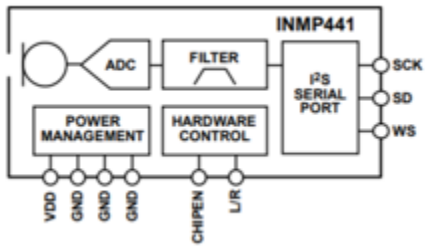
The microphone device used for our system is INMP441 model. It is an omnidirectional MEMS microphone that consists of MEMS sensor. It connects directly to digital processors through I2S interface. It is mentioned in the datasheet https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf . Hence, the I2S interface has been employed to connect to the MEMS microphone based on https://www.youtube.com/watch?v=m8LwPNXqK9o .The outcome is determined by feeding the data in the coding and witness it through the logs display. The crucial parts of I2S configuration are by setting the ***data and frame format*** into ***24bits data into 32bits frame*** and change the ***transmission mode*** as a ***receiver***. Otherwise, it's not working.

1. Microphone device and its connection:-

i. Microphone device: -

INMP441 model: -

ii. Schematic diagram of microphone device: -

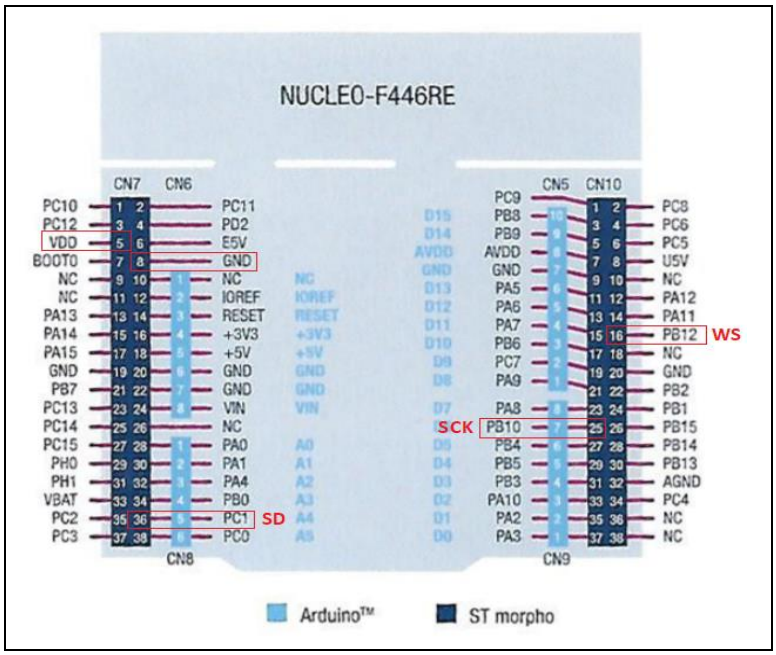


iii. Pins interconnect of microphone device to STM32 board: -

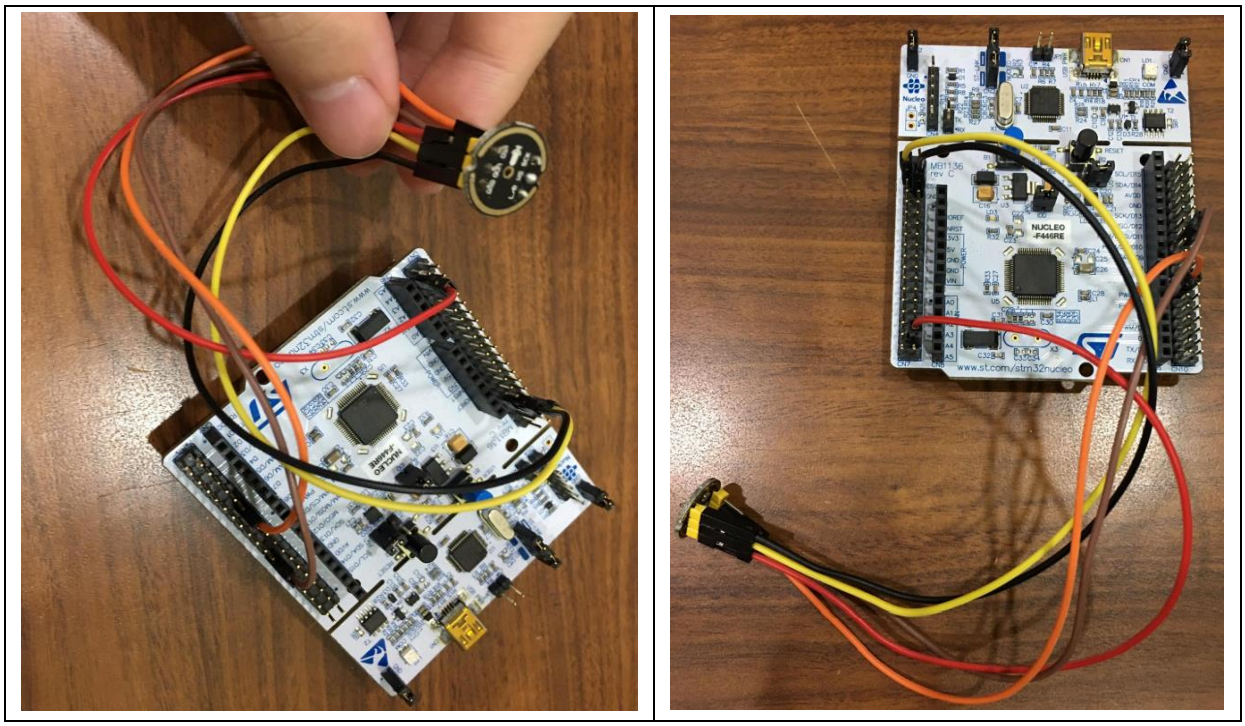
Table 1: Pins interconnect of microphone device to STM32 board

Pins on STM32 Board	**Pins on Microphone**
PC1 (port C pin1)	SD
PB10 (port B pin 10)	SCK
PB12 (port B pin 12)	WS
Vdd	Vdd
Gnd	Gnd

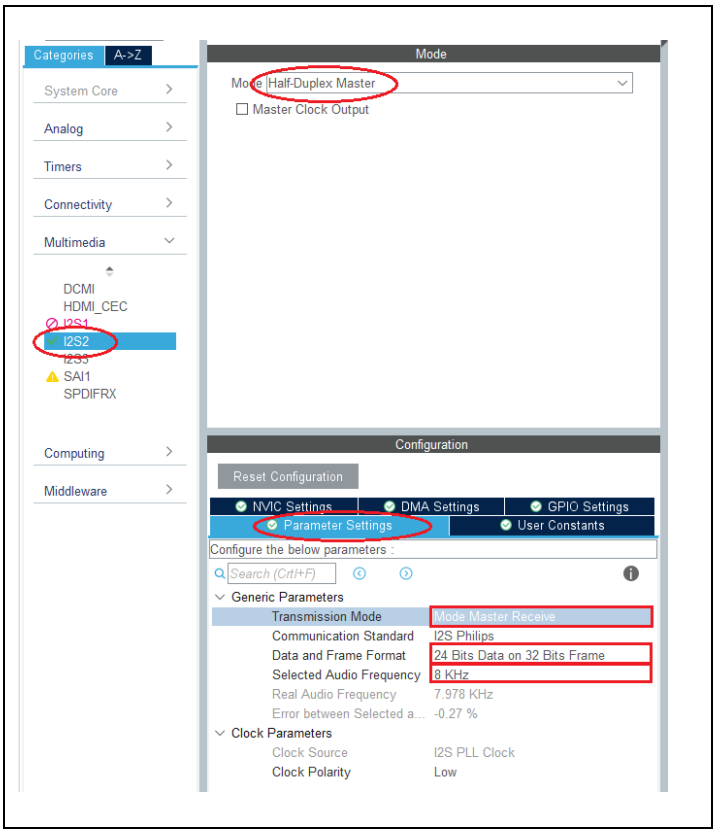
Schematic diagram of NUCLEO-F446RE board with I2S connections: -



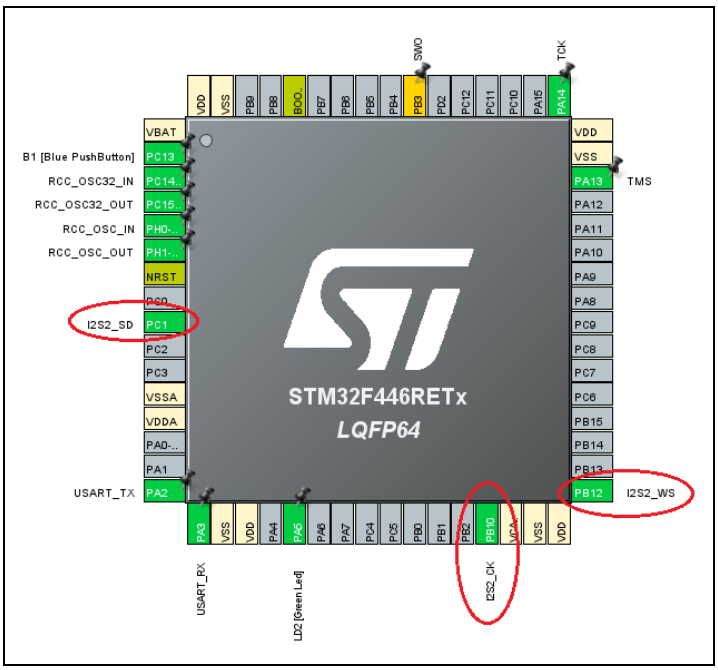
Real-time pins connection: -



2. In STM32CubeMX, choose the settings as shown below: -

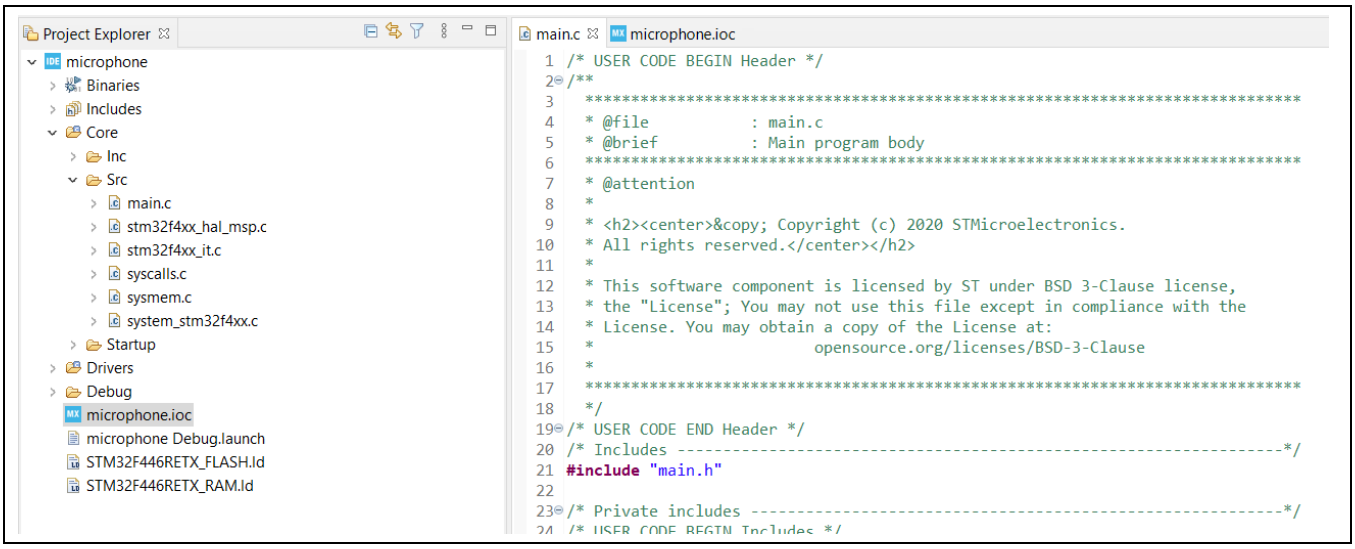


3. The I2S pins will be automatically assigned as shown below: -



4. Generate project, normally.
5. Save the projects and open it in STM32CubeIDE.

6. In STM32CubeIDE interface: -



7. In main.c, add lines as shown below: -

Declaration: -

```
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */



/* USER CODE BEGIN Init */
uint8_t buf[12];
uint16_t data_in[2];
/* USER CODE END Init */
```

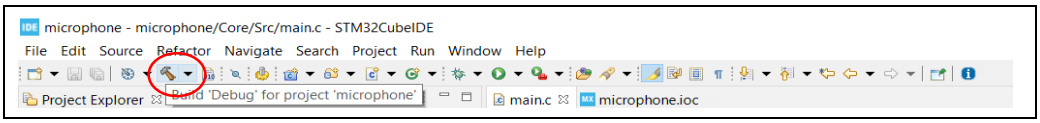
1st way: -

```
  while (1)
  {
      volatile HAL_StatusTypeDef result = HAL_I2S_Receive(&hi2s2, data_in, 2, 100);

      if (result == HAL_OK) {
          volatile int32_t data_full = (int32_t) data_in[0] << 16 | data_in[1];
          volatile int16_t data_short = (int16_t) data_in[0];
          sprintf((char*)buf, "0x%x \r\n", ((unsigned int)data_full));
          sprintf((char*)buf, "0x%x \r\n", ((unsigned int)data_short));
      }
      else {
          strcpy((char*)buf, "Error Rx\r\n");
      }
      //strcpy((char*)buf, "Hello!\r\n");
      HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
      //HAL_Delay(1000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

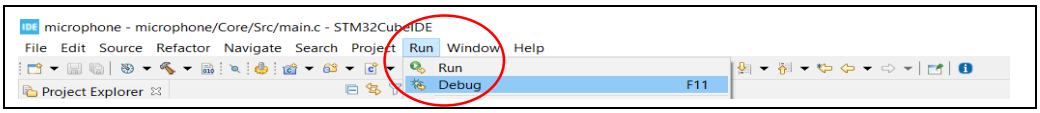
2nd way: -

```
  while (1)
  {
      volatile HAL_StatusTypeDef result = HAL_I2S_Receive(&hi2s2, data_in, 2, 100);

      if (result != HAL_OK) {
          strcpy((char*)buf, "Error Rx\r\n");
      }
      else {
          int32_t data_full = (int32_t) data_in[0] << 16 | data_in[1];
          //volatile int16_t data_short = (int16_t) data_in[0];
          sprintf((char*)buf, "0x%x \r\n", ((unsigned int)data_full));
      }
      //strcpy((char*)buf, "Hello!\r\n");
      HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);
      //HAL_Delay(500);
    /* USER CODE BEGIN 3 */
  }
```

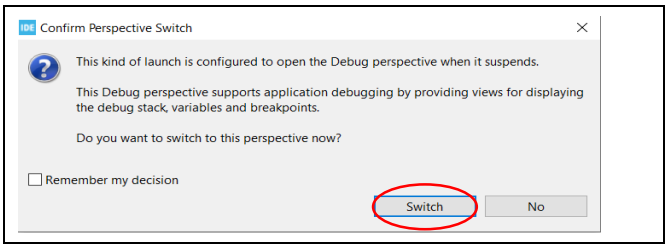
8. Build 'Debug' for project '<your_project_name>'



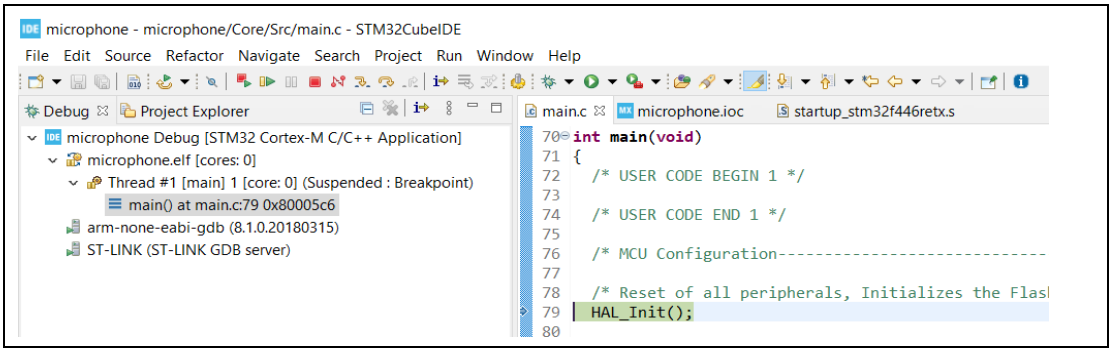
9. Run >> Debug >> Then wait for the program to compiled successfully.



10. Click Switch to the debug perspective chosen previously.

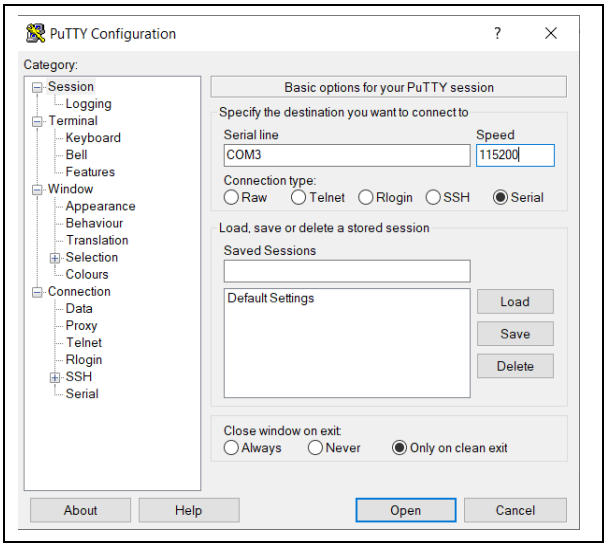


11. The lists interface will changed as shown below: -

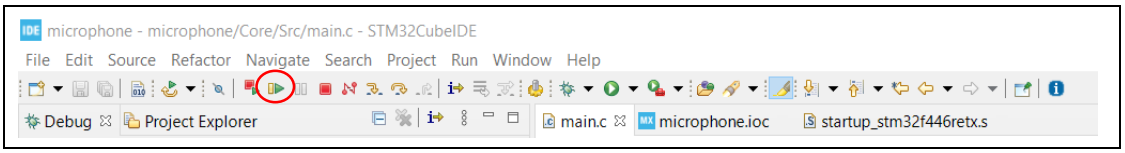


12. In puTTy, choose the setting as shown below.

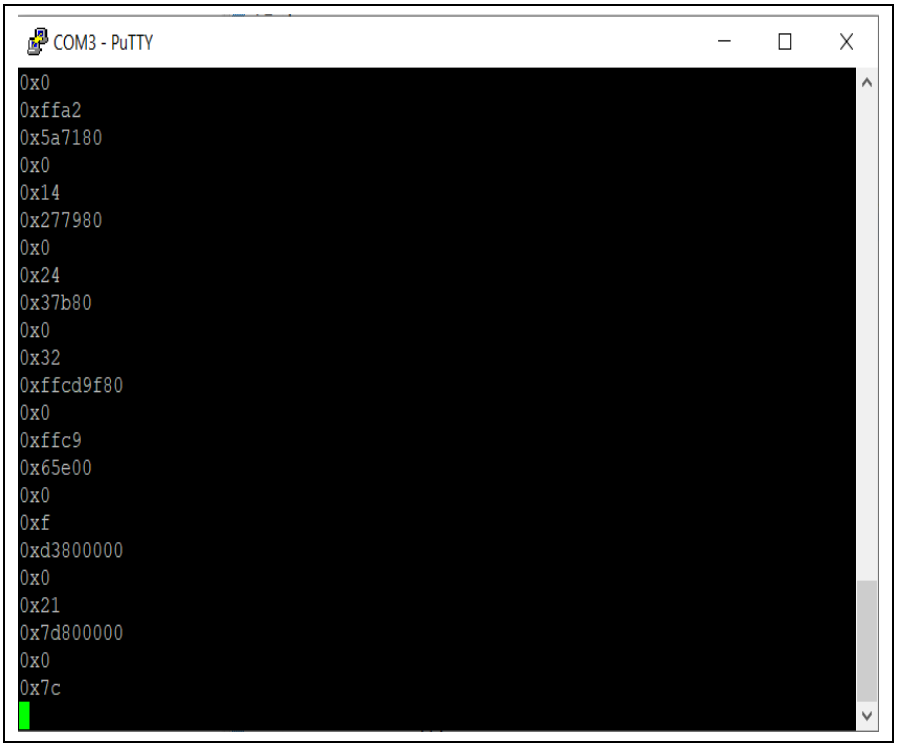
- Example: COM3 is referred to the NUCLEO-F446RE ports connected to the PC/laptop.



13. Resume (F8) to continue running the program in debug mode.



14. In puTTy, the logs running as shown below: -

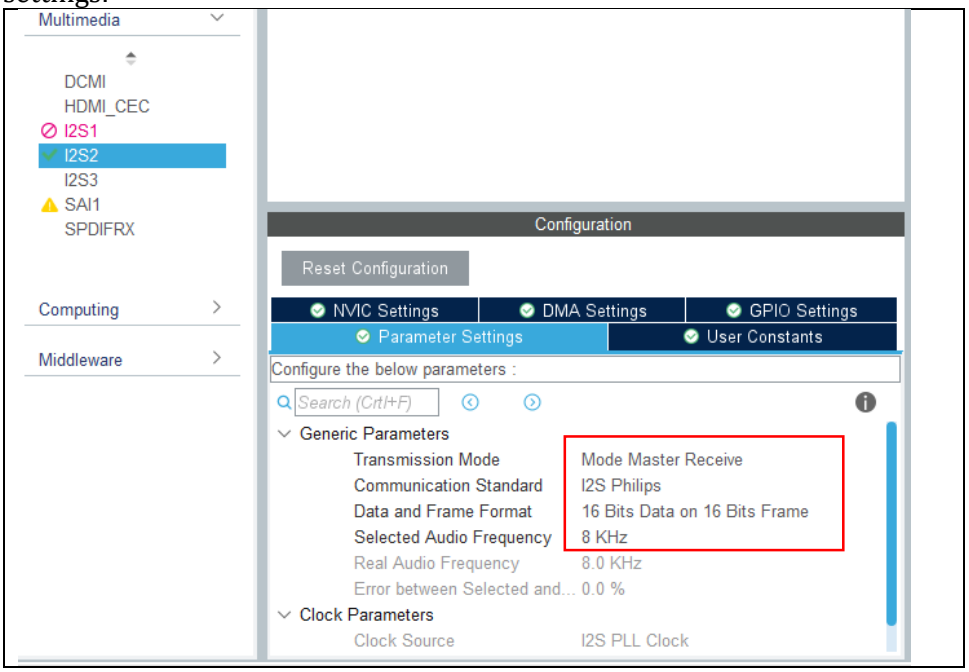


I2S Integration from INMP441 (microphone) to KWS Algorithm on NUCLEO-F446RE (STM32-board)

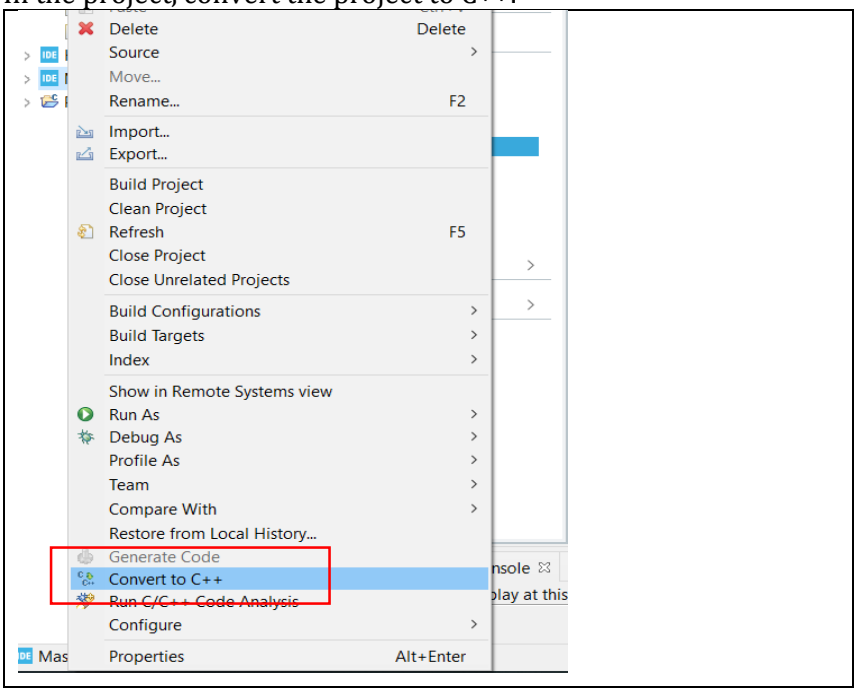
Created by Lim Calvin

Previous topic only covering the step to get the microphone inputs. In this topic, we will be covering the setting and the coding to correctly integrate the usage of the microphone, to capture audio sample to be fetched into the KWS algorithm to perform the correct keyword spotting. We will be collecting 16 Bits Data on 16 Bits Frame with 8KHz audio frequency, to fetch into the audio_buffer with size of 16000. This audio_buffer[16000] is the 1 second sound sample to be analyzed in the KWS algorithm. The project needs to be converted into C++ projects, and rename the main.c to main.cc in order to successfully integrate the microphone with KWS algorithm.

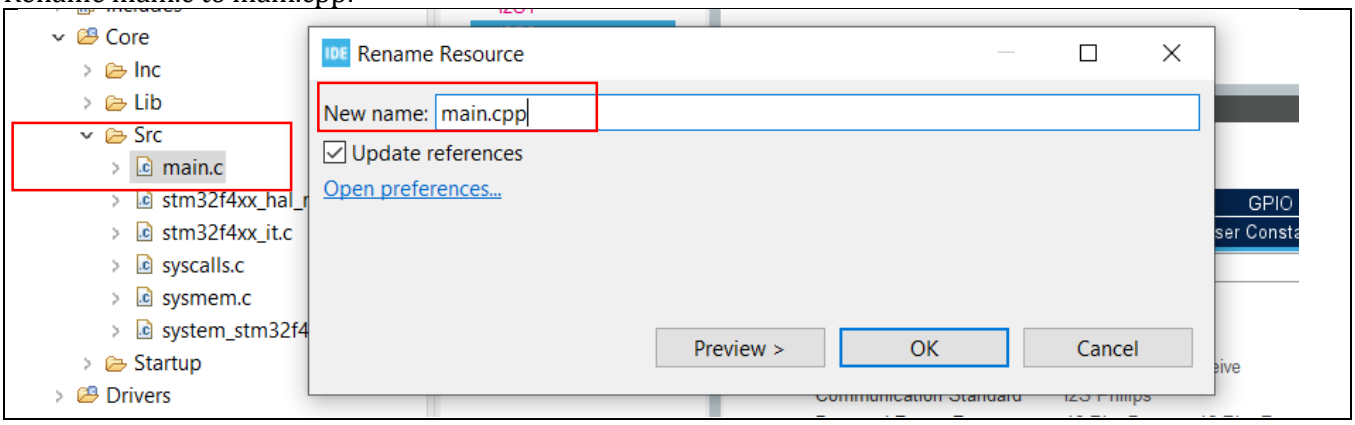
1. The hardware setup was the same from previous topic, but with the following change on the I2S parameter settings: -



2. In the project, convert the project to C++:



3. Rename main.c to main.cpp: -



4. Below are the coding modifications on the main.cpp: -

1st, Include header file .h: -

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "kws_ds_cnn.h"
#include "wav_data.h"
#include "stdio.h"

```

2nd, Declare variable: -

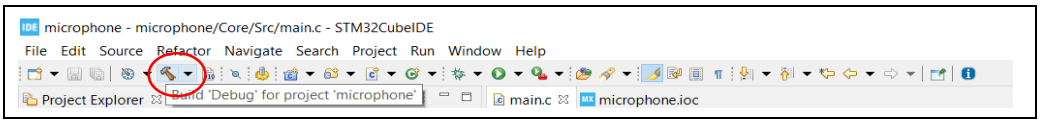
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t buf[12];
uint16_t data_in[2];
int16_t audio_buffer[16000];

```

3rd, Coding in while loop,

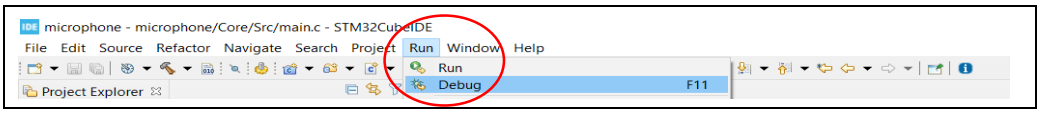
```
  while (1)
  {
    /* USER CODE END WHILE */
      HAL_Delay(500);
      HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);  //calvin

      for (int i=0;i<16000;i+=2){
          volatile HAL_StatusTypeDef result = HAL_I2S_Receive(&hi2s2, data_in, 2, 100);
          if ( result != HAL_OK ) {
              strcpy((char*)buf, "Error Rx\r\n");\
              }else{
                  audio_buffer[i]   = (int16_t)data_in[0];
                  audio_buffer[i+1] = (int16_t)data_in[1];
              }
          }

      HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);    //calvin

      char output_class[12][8] = {"Silence", "Unknown","yes","no","up","down","left","right","on","off","stop","go"};
      KWS_DS_CNN kws(audio_buffer);

      kws.extract_features(); //extract mfcc features
      kws.classify();     //classify using dnn

      int max_ind = kws.get_top_class(kws.output);

      char buffer [70];
      int buffer_output = 0;
      buffer_output = sprintf(buffer, "Detected %s (%d%%)\r\n",output_class[max_ind],((int)kws.output[max_ind]*100/128));
      HAL_UART_Transmit(&huart2, (uint8_t *)buffer, buffer_output, 100);

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

The code will first Turn on the LED as an indication for “start recording”. During the recording process, the MCU will try to sample the incoming audio data bits into an audio_buffer array with size of 16000. Once the audio_buffer was filled, the MCU turns off the LED, and proceed with the KWS algorithm. The output is ported to PuTTy terminal but can be modified to output to LCD Display.

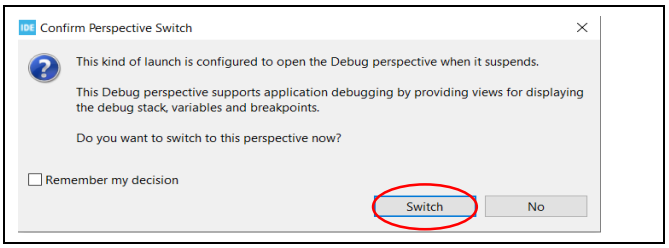
5. Build 'Debug' for project '<your_project_name>'



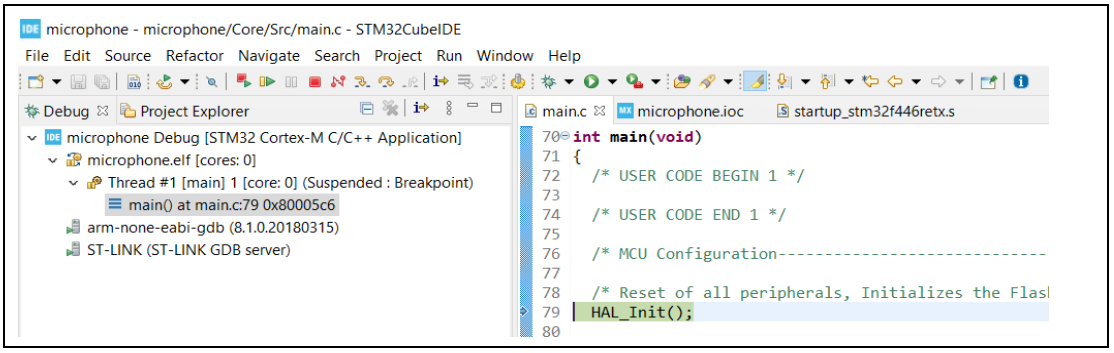
6. Run >> Debug >> Then wait for the program to compiled successfully.



7. Click Switch to the debug perspective chosen previously.

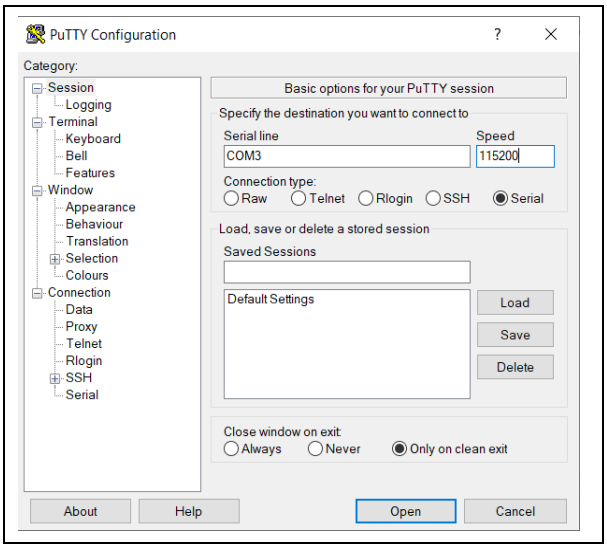


8. The lists interface will changed as shown below: -

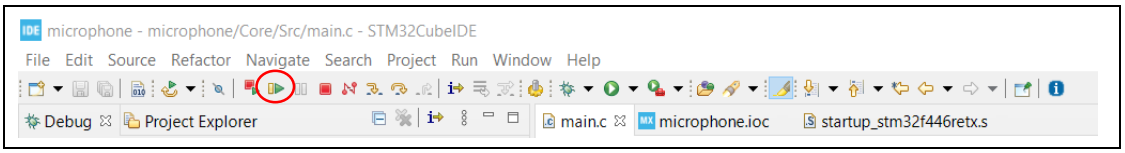


9. In puTTy, choose the setting as shown below.

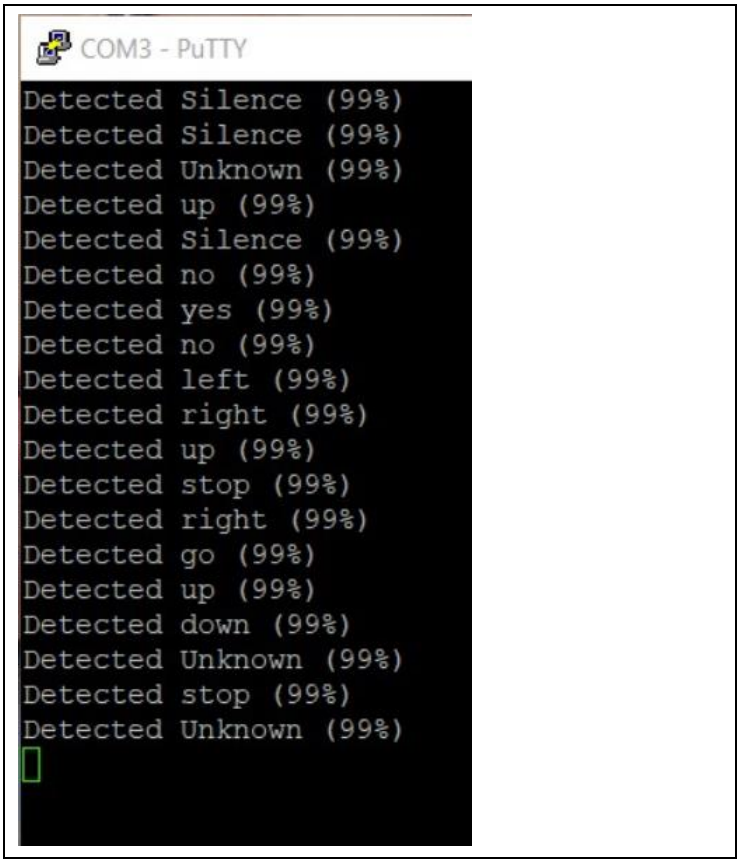
- Example: COM3 is referred to the NUCLEO-F446RE ports connected to the PC/laptop.



10. Resume (F8) to continue running the program in debug mode.



11. In puTTy, the logs running as shown below: -



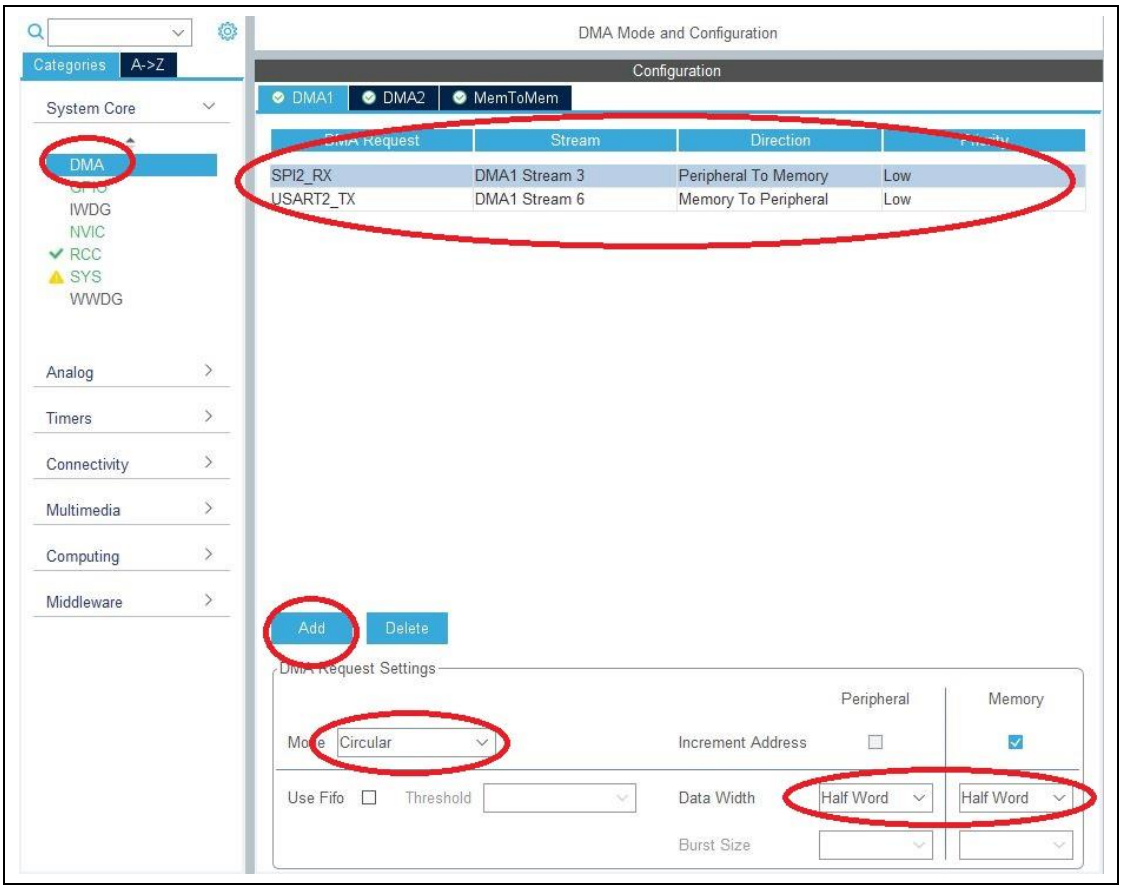
DMA Integration to NUCLEO-F446RE (STM32-board)

Created by Lim Calvin

In order to provide high speed data transfer between peripherals and memory as well as memory to memory, the direct access memory (DMA) is used. The DMA integration into STM32-board is executed based on

a. https://www.st.com/content/ccc/resource/training/technical/product_training/group0/ce/9e/fe/44/e5/62/45/34/STM32F7_System_DMA/files/STM32F7_System_DMA.pdf/jcr:content/translations/en.STM32F7_System_DMA.pdf
b. https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf
c. https://www.digikey.com/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c

1. Configurations in STM32CubeMX (*.ioc):-
   >> Click on **System Core**
   >> Double click on **DMA**
   >> Click on **Add**
   >> Choose DMA Request as **SPI2_RX** and **USART2_TX**
   >> Choose Model as **Circular** and Data Width as **Half Word**



2. Code modification in STM32CubeIDE (main.c): -

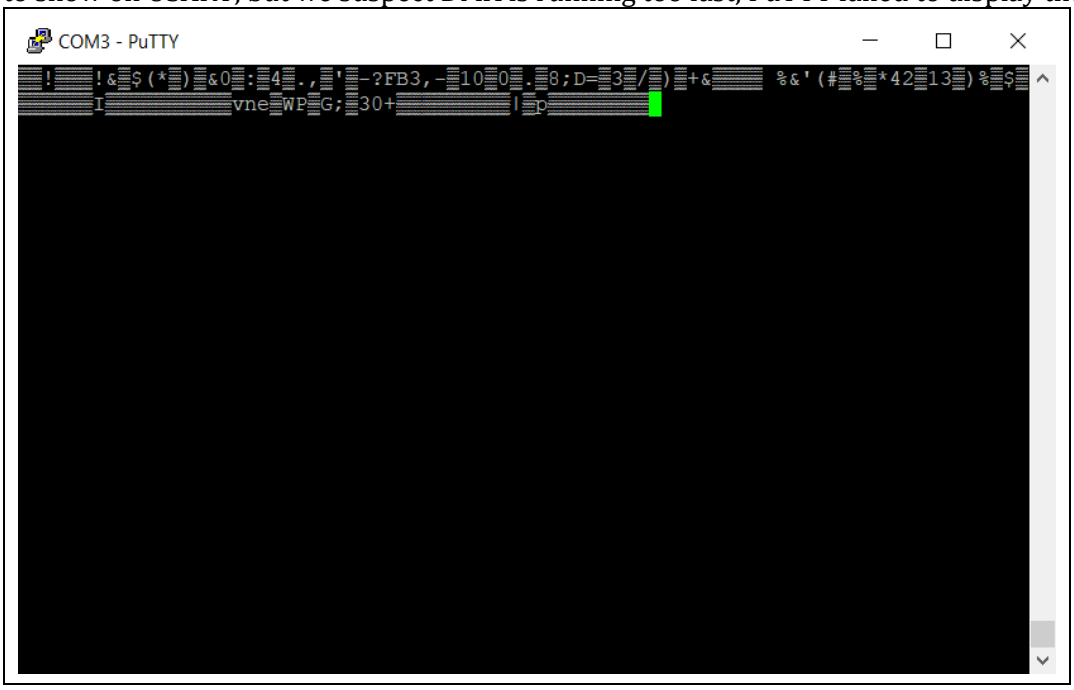
```
/* USER CODE BEGIN PD */
#define DMA_BUF_LEN 4096
/* USER CODE END PD */

/* USER CODE BEGIN Init */
  uint8_t buf[12];
  uint16_t data_in[2];
  uint16_t dma_buffer[DMA_BUF_LEN];
/* USER CODE END Init */


/* USER CODE BEGIN 2 */
  HAL_I2S_Receive_DMA(&hi2s2, dma_buffer, DMA_BUF_LEN);
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN 2 */
  HAL_I2S_Receive_DMA(&hi2s2, dma_buffer, DMA_BUF_LEN);
/* USER CODE END 2 */

while(1){
/*START - DMA*/
      huart2.Instance->CR3 |= USART_CR3_DMAT;
      HAL_DMA_Start_IT(&hdma_usart2_tx, (uint32_t)dma_buffer,
                       (uint32_t)&huart2.Instance->DR, sizeof(dma_buffer));
/*END - DMA*/
}

/* USER CODE BEGIN 4 */
void HAL_I2S_RxHalfCpltCallBack(I2S_HandleTypeDef* hi2s2){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
}
void HAL_I2S_RxCpltCallBack(I2S_HandleTypeDef* hi2s2){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
}
/* USER CODE END 4 */
```

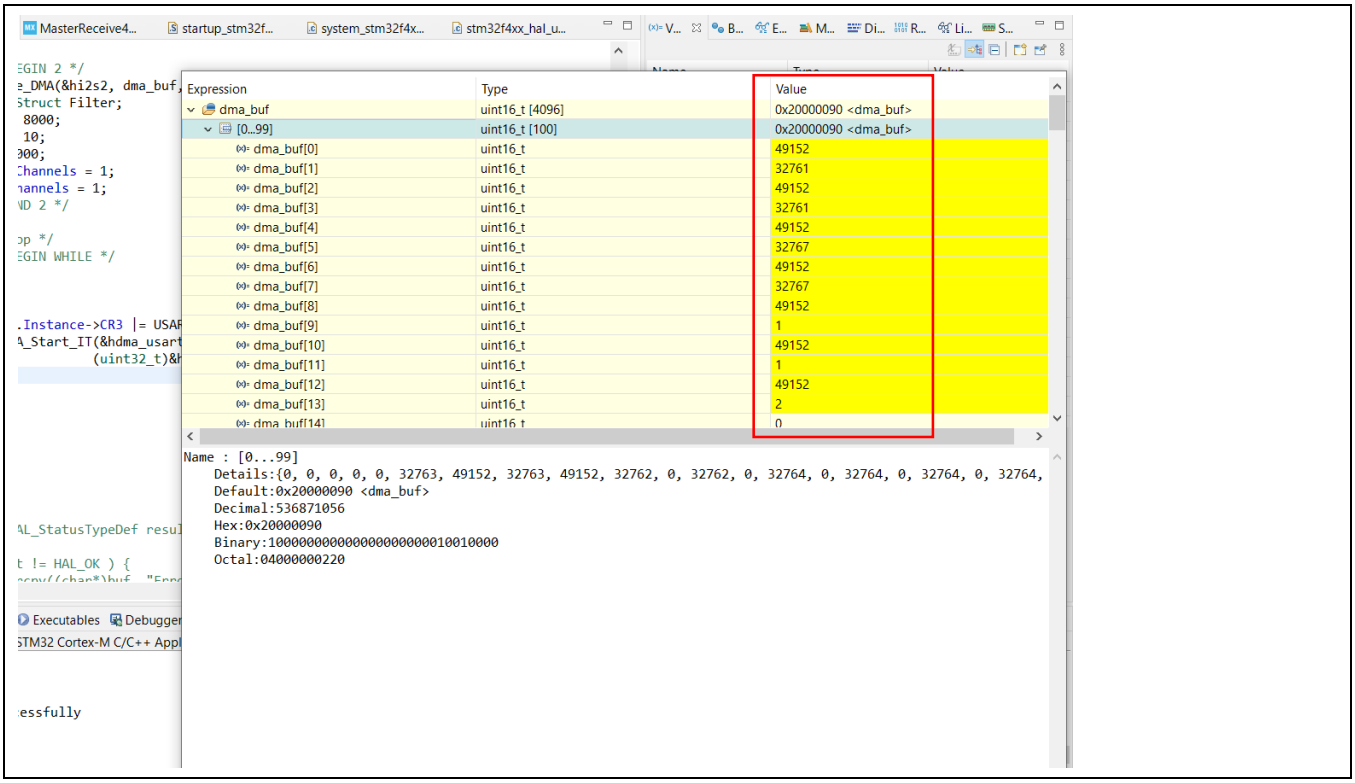
3. In coding, add on breakpoint at Call Back function for I2S, these 2 Call Back act as an indicator for the DMA buffer status: -

```

/* USER CODE BEGIN 4 */
void HAL_I2S_RxHalfCpltCallBack(I2S_HandleTypeDef* hi2s2){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
}
void HAL_I2S_RxCpltCallBack(I2S_HandleTypeDef* hi2s2){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
}
/* USER CODE END 4 */
```

4. Run the system in Debug mode.
   Display output in puTTy shown as below, the purpose is to identify whether the output is correctly transfer to show on USART, but we suspect DMA is running too fast, PuTTY failed to display the value:-



5. While the system is running in Debug mode, it will be halted at the breakpoint.
6. Hover the mouse to DMA_BUFFER, and the data details will be shown: -

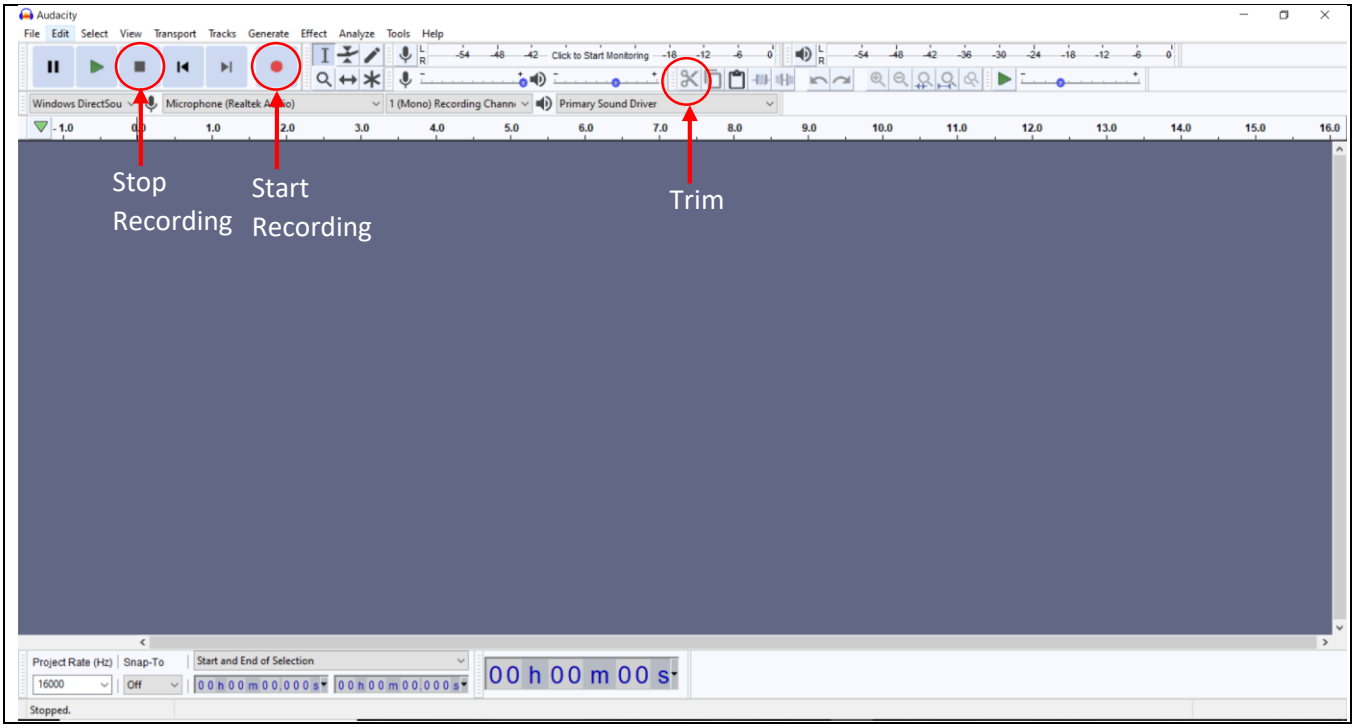


Audio Recording in WAV File (.wav) Format

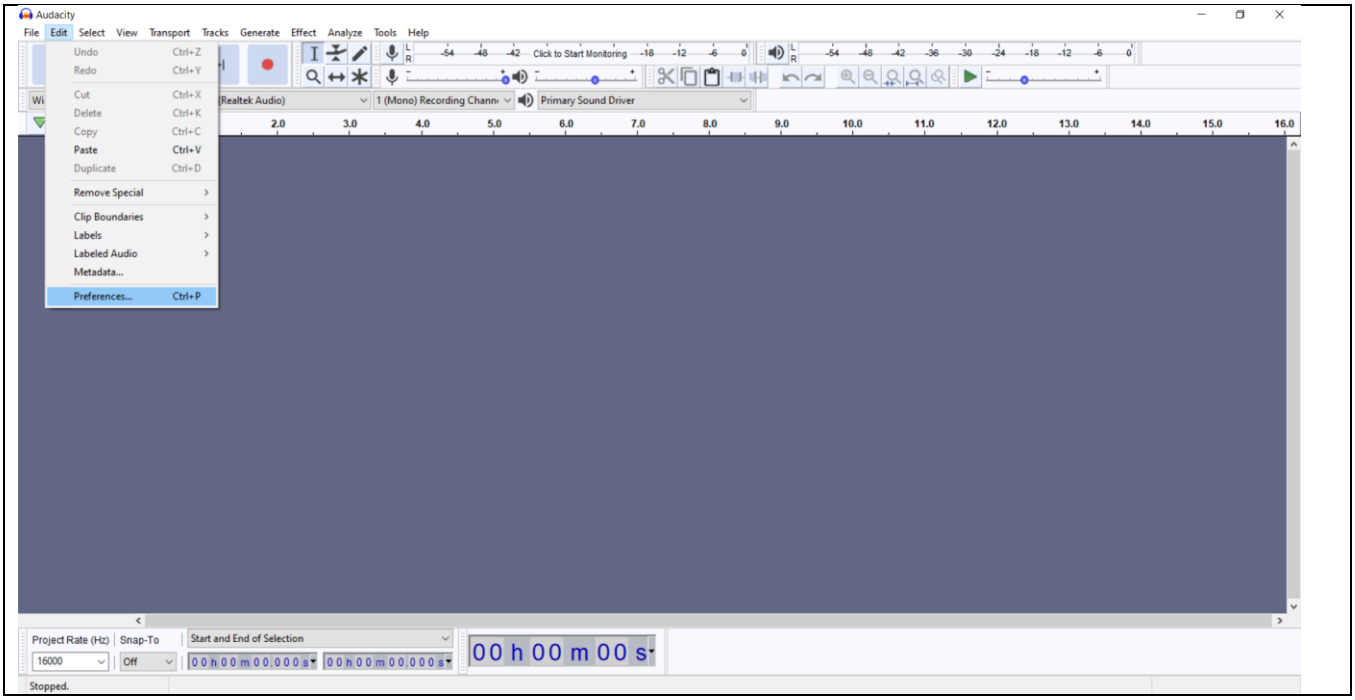
Created by Phuah Chai Ling

First of all, an audio clip is recorded by a microphone connected to a laptop via the software called "Audacity". Audacity is a free software that can be downloaded from https://www.audacityteam.org/download/. The setup for the software environment is as shown below:

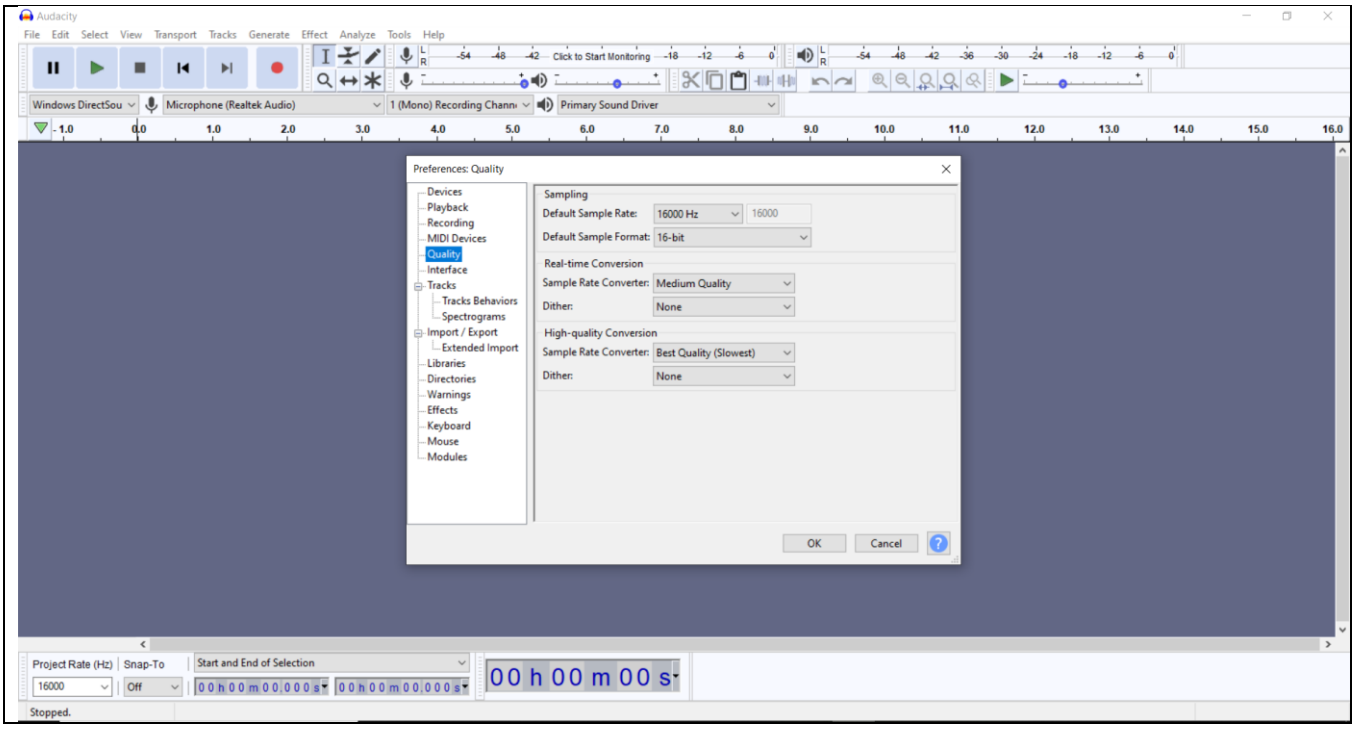
1. Launch the "Audacity" application.



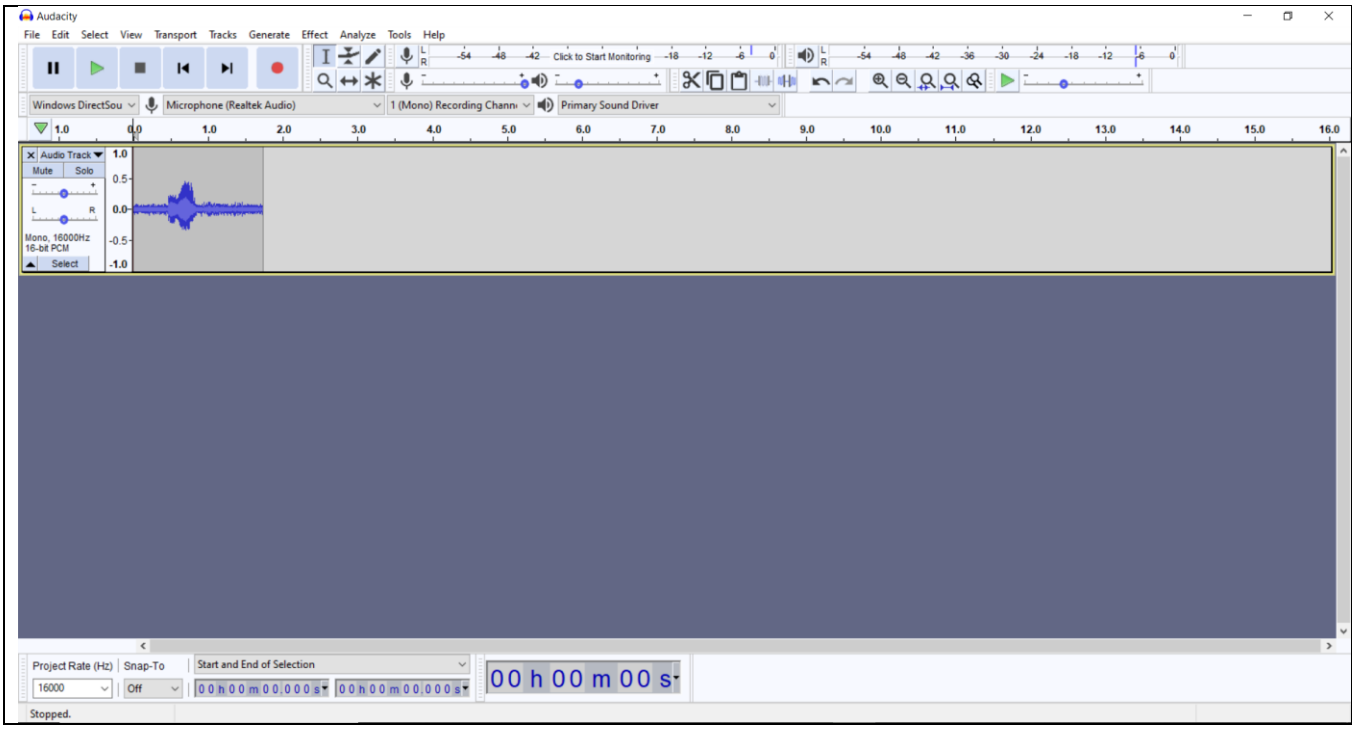
2. Go to Edit > Preferences…



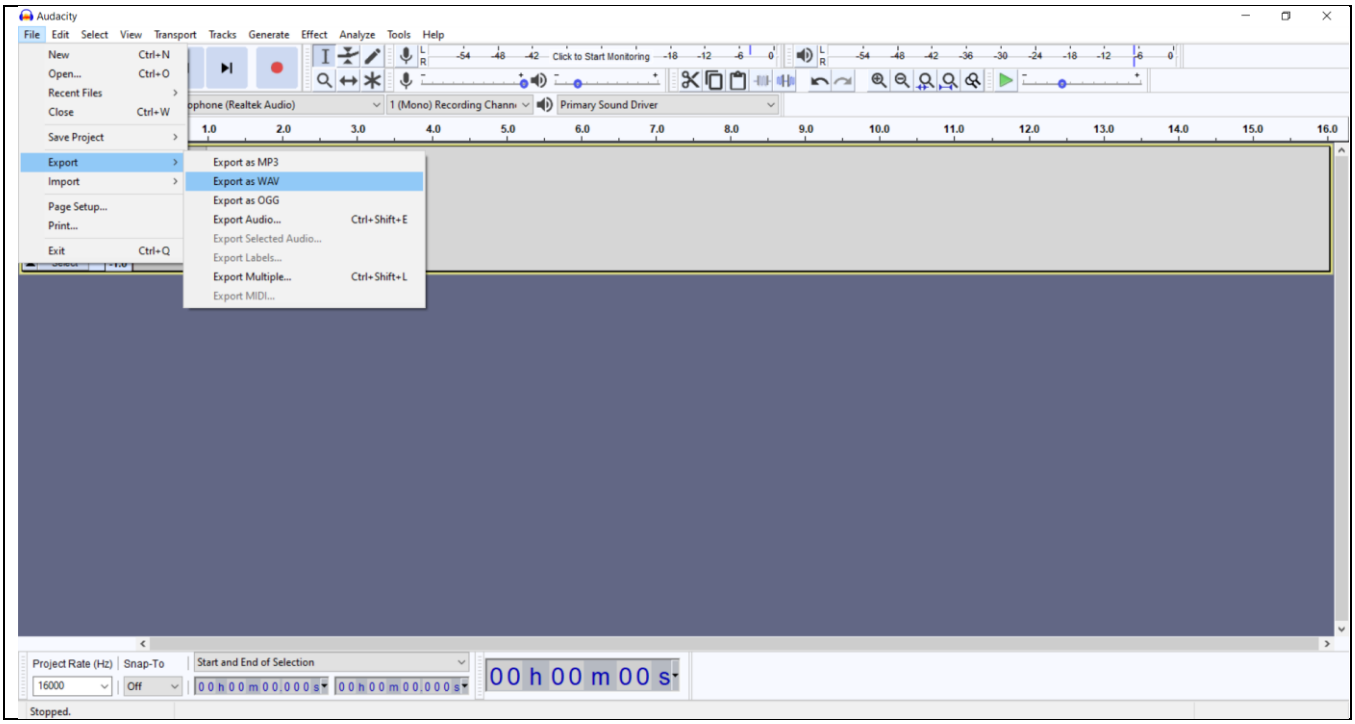
3. On the Preferences… tab, go to “Quality” and customize the settings accordingly. In this example, the sample rate and sample format were set as 16kHz and 16-bit respectively. It is important to set both the “Dither” tabs to “None” to prevent the Audacity software from converting the 16-bit sample format into 32-bit (due to the nature of the software) during the export of the audio clip. Click “OK” once everything is set.



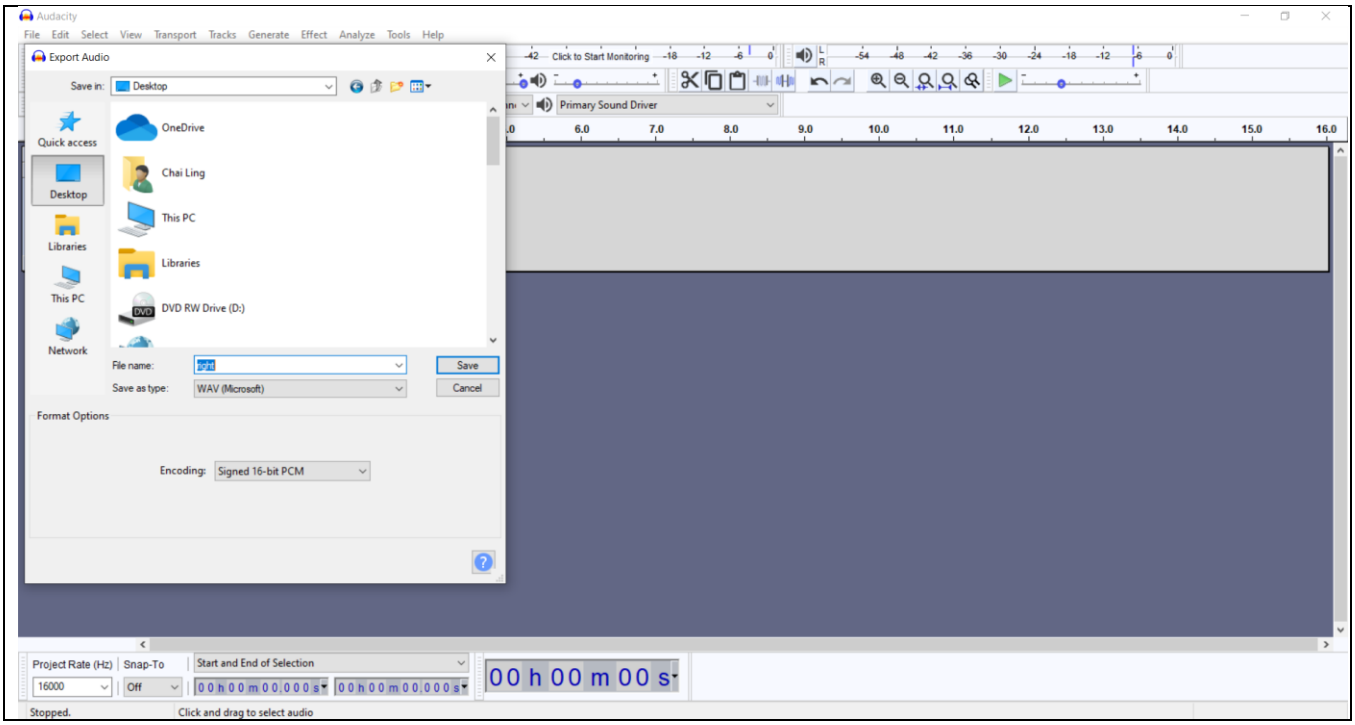
4. Start recording and trim the unwanted audio clip.



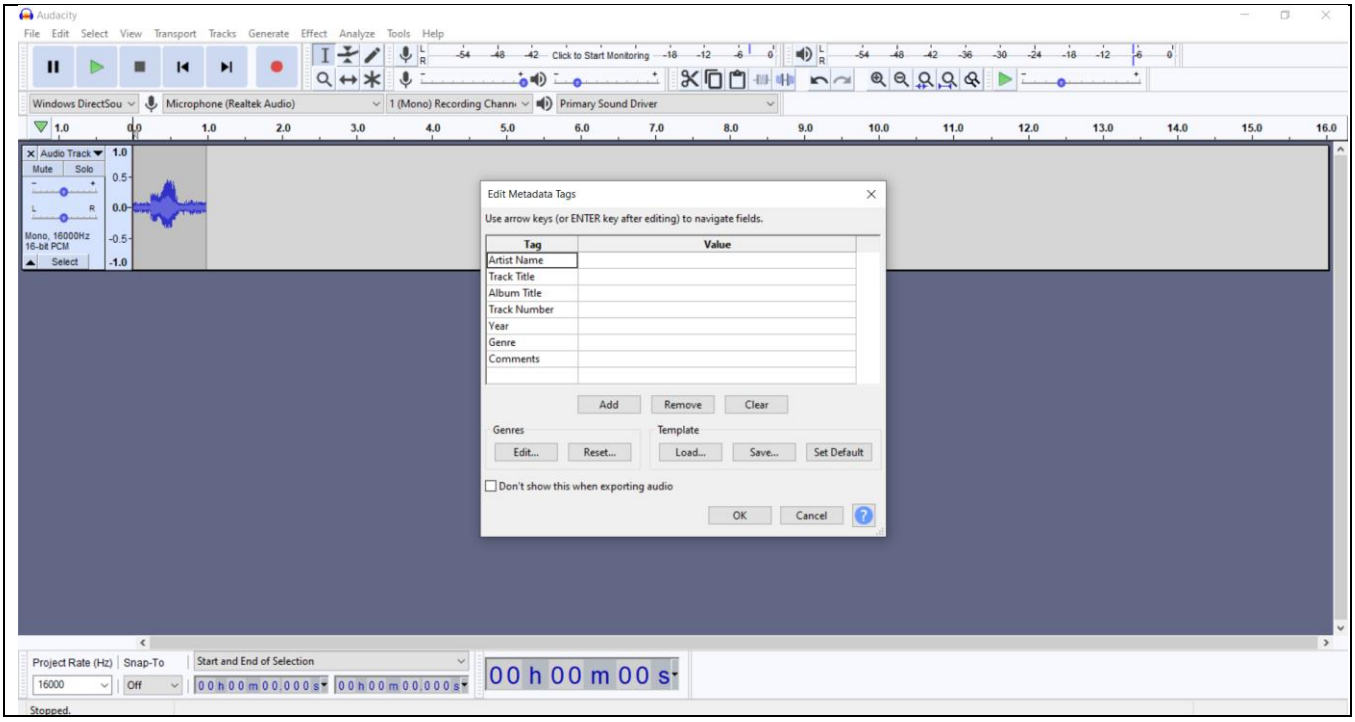
5. Go to File > Export > Export as WAV.



6. Save the file as .WAV type and encoded in Signed 16-bit PCM.



7. Just click “OK” if you do not wish to modify the metadata and exit the software once done.

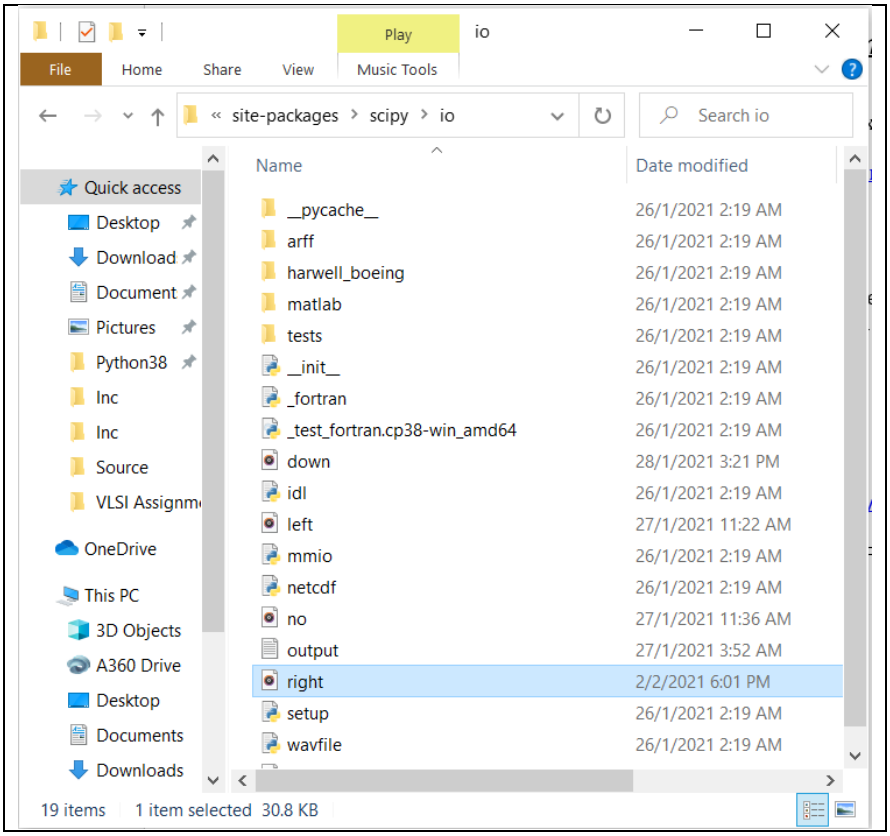


Extraction of the Audio Data Samples from the WAV File (.wav)

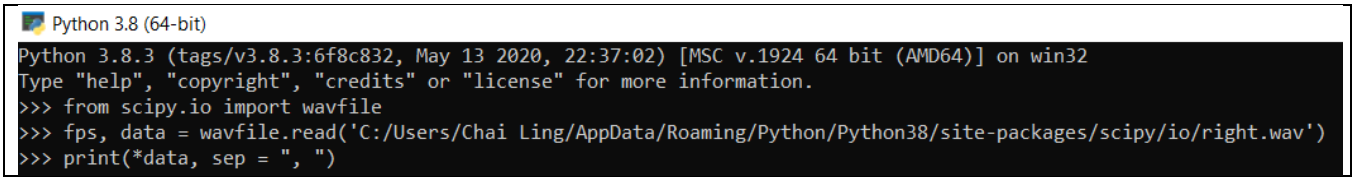
Created by Phuah Chai Ling

Once the audio clip is recorded, the audio data samples are then extracted from the .wav file by using the Python application.

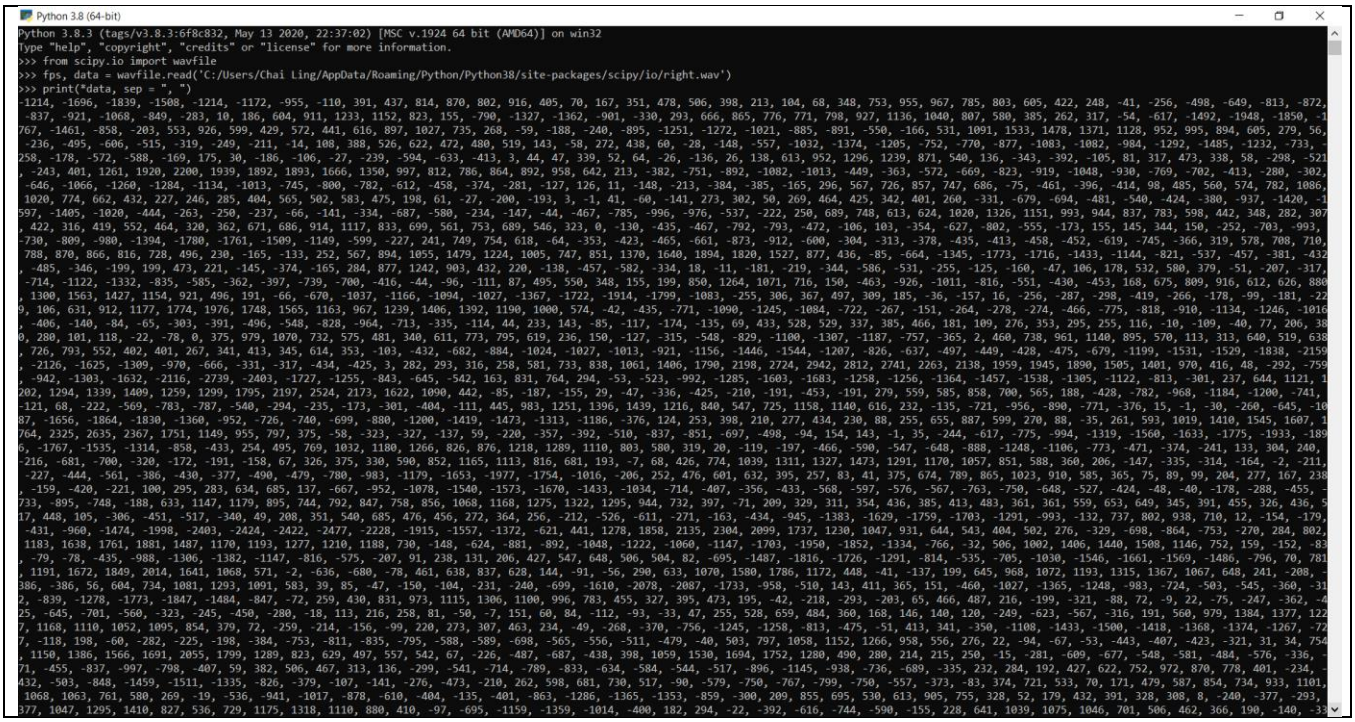
1. You may download the Python application from https://www.python.org/downloads/windows/.
2. Install the Python application and also the Python SciPy library.
3. Place the .wav file into the SciPy > io folder.



4. Launch the Python application, and type the following codes.



5. These are the audio data samples extracted from the .wav file.



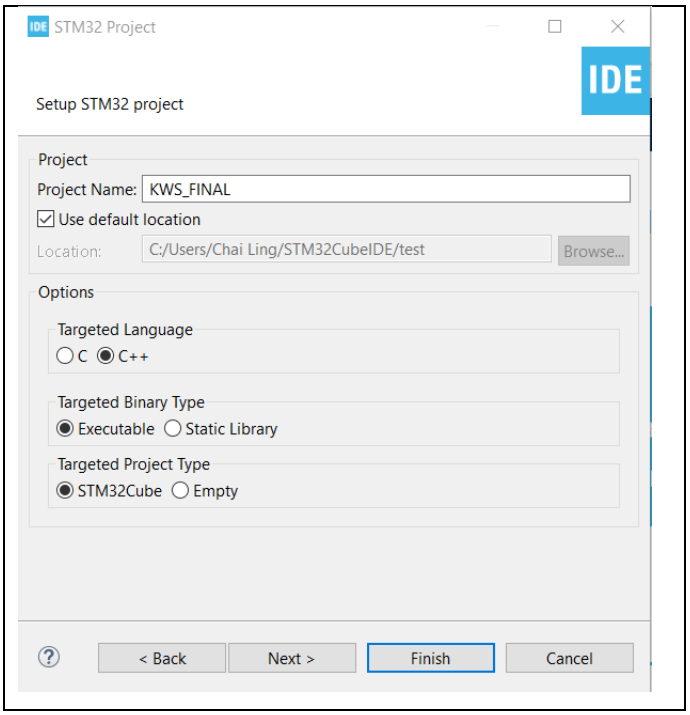
6. Save all the audio data samples extracted.

Integration of Keyword Spotting (KWS) Algorithm (Non-Real Time)

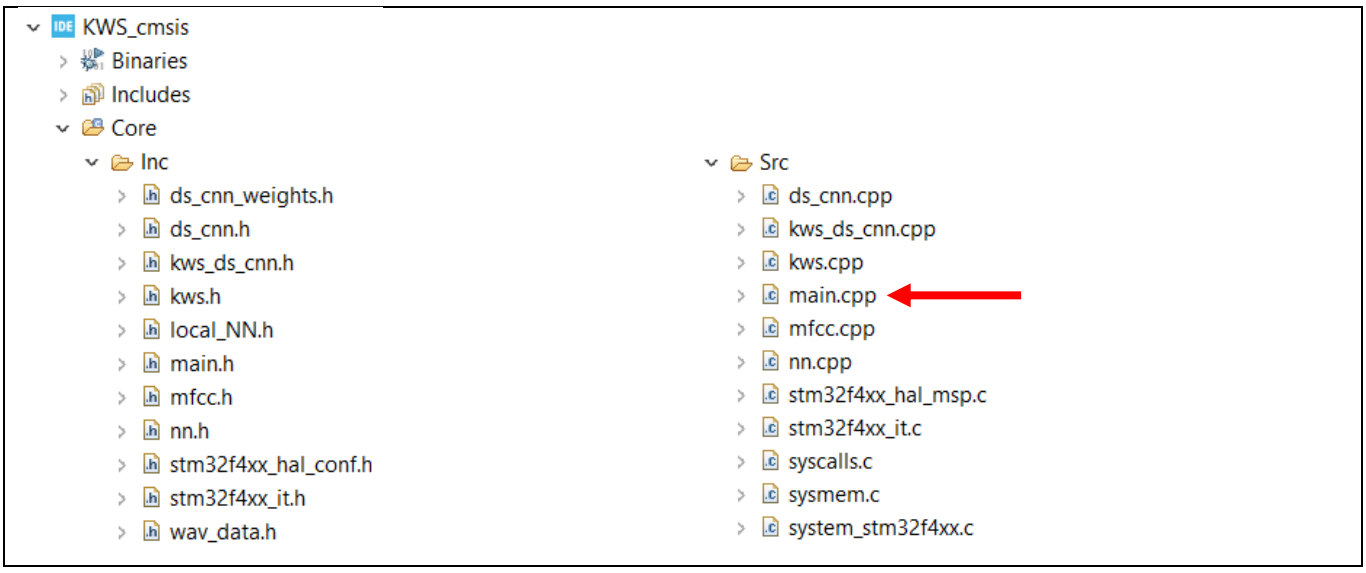
Created by Phuah Chai Ling

The implementation of the KWS algorithm in this project is based on the reference paper ***"Hello Edge: Keyword Spotting on Microcontrollers"*** or can be directly accessed at https://arxiv.org/pdf/1711.07128.pdf . There are 10 keywords that have been validated which are ***Up, Down, No, Yes, Left, Right, On, Off, Go, Stop, Unknown*** and ***Silence***. The detected keyword together with its detection accuracy (%) are displayed through the PuTTY serial console.

1. First of all, launch the STM32CubeIDE software and start a new STM32 project. Select "C++"as the targeted language and initialize all the peripherals needed.



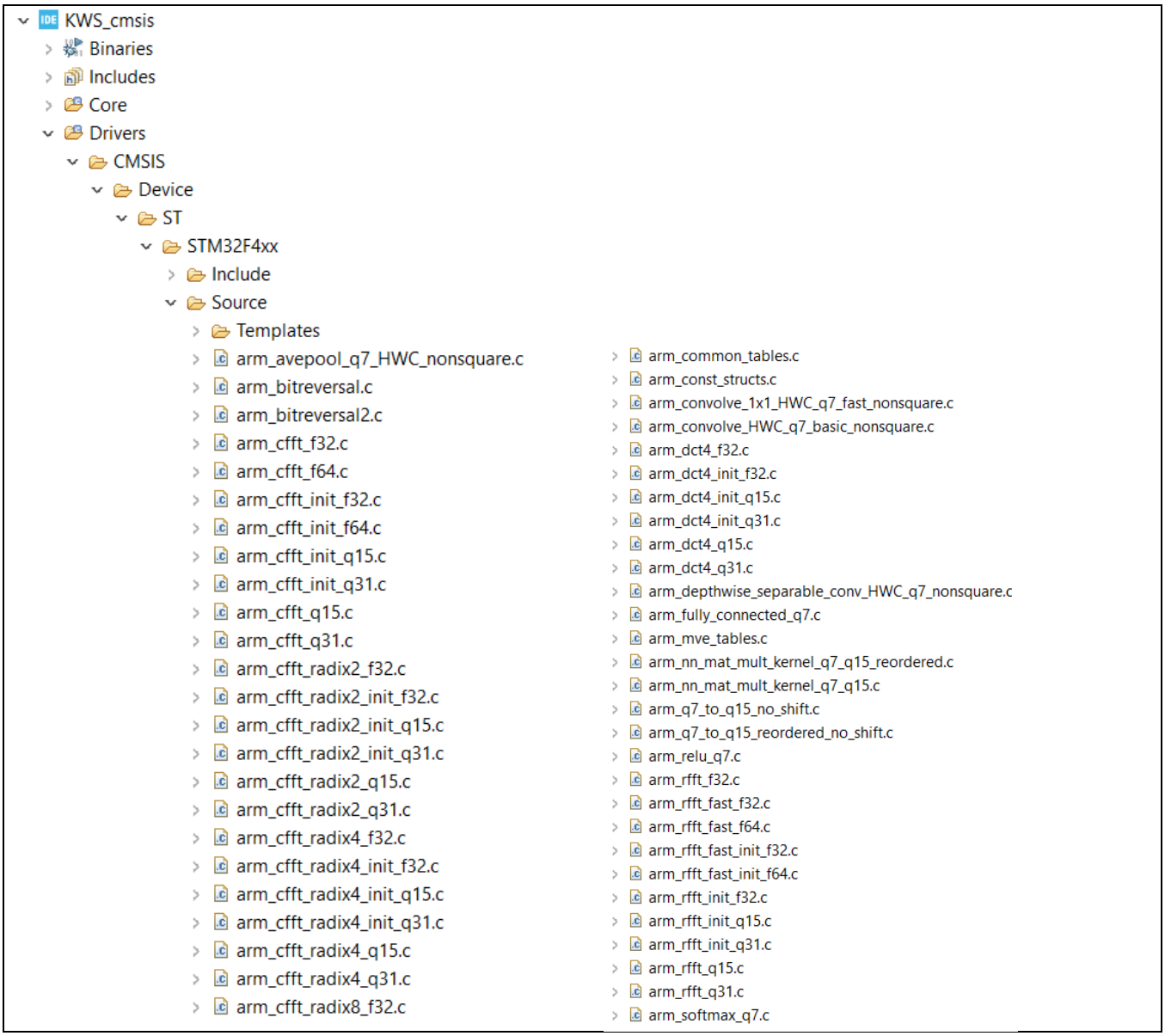
2. Once the new project has been created, import all the necessary files from the GitHub (Reference : https://github.com/ARM-software/ML-KWS-for-MCU) into the project directory. Also, make sure that the "main" source file is in .cpp format.



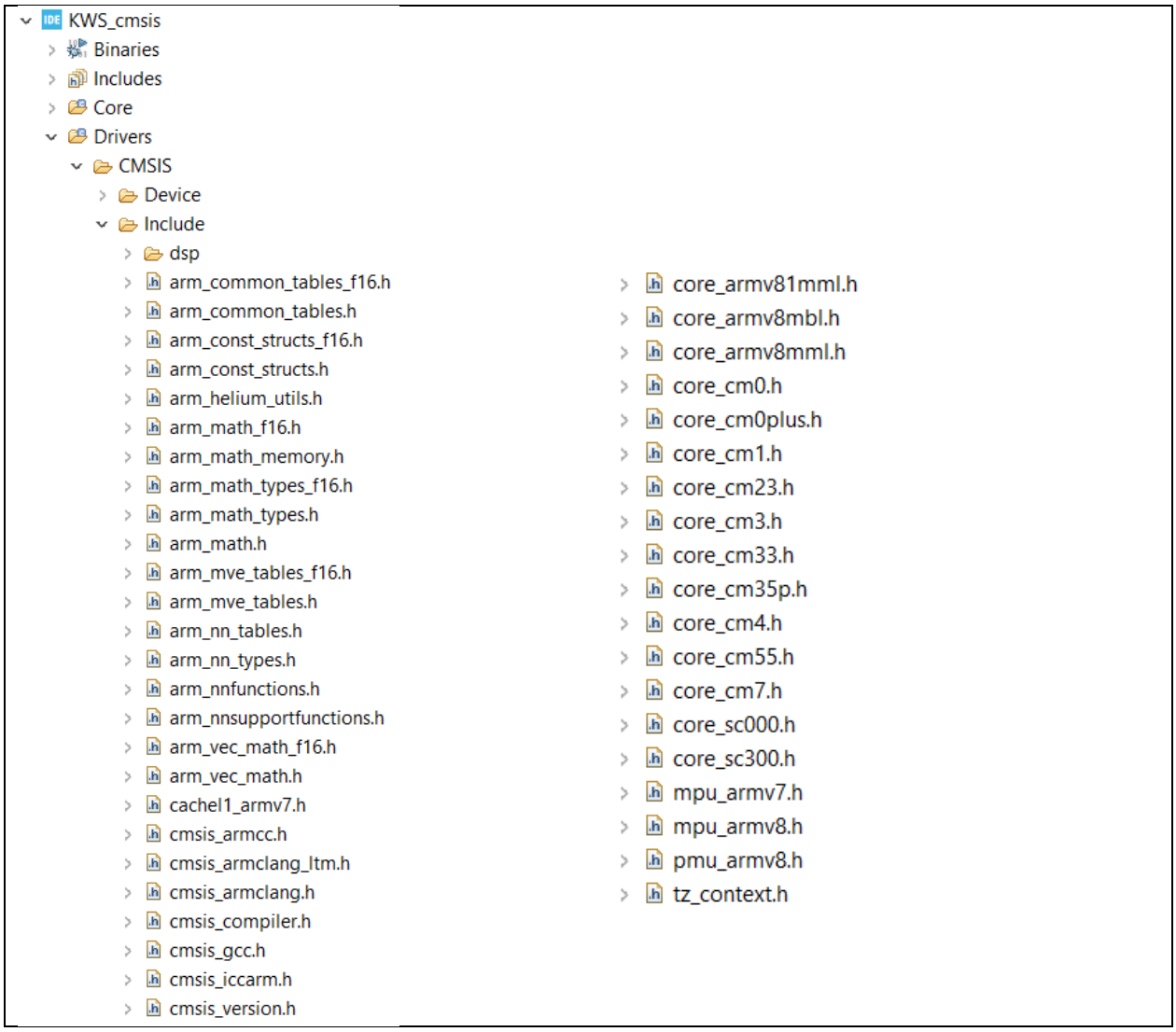
3. Download the “CMSIS_5-develop” library from this GitHub:
   http://arm-software.github.io/CMSIS_5/index.html

4. Import these files from the “CMSIS_5-develop” library into this project directory.

   I. NN source files



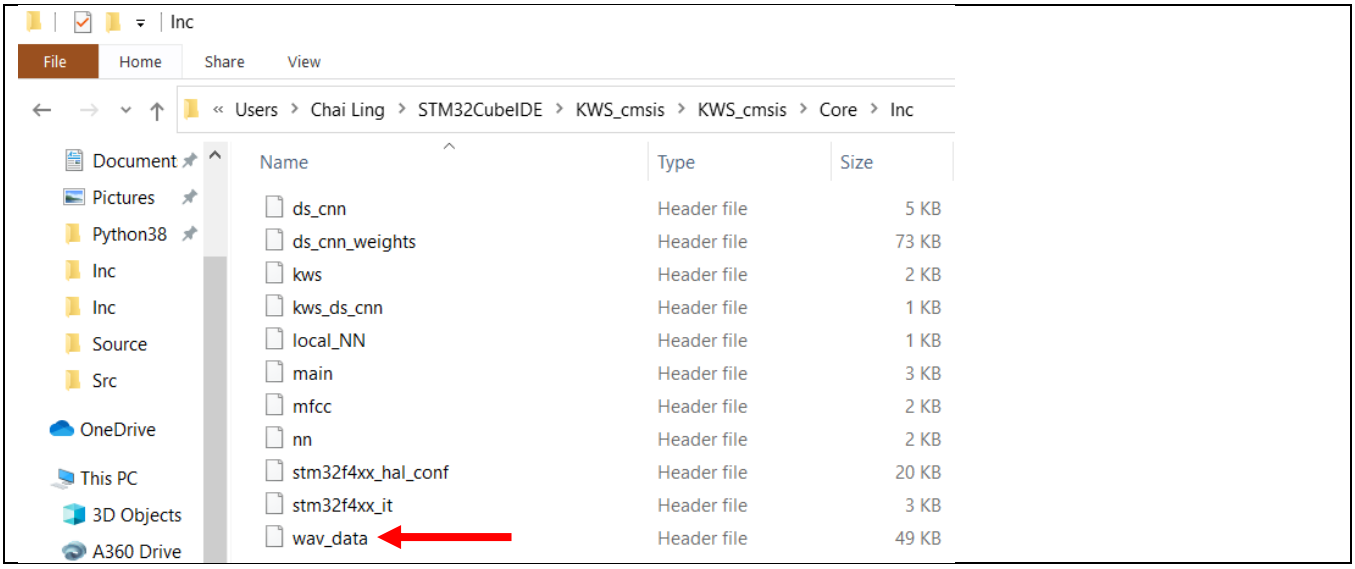
II. NN header files and DSP library



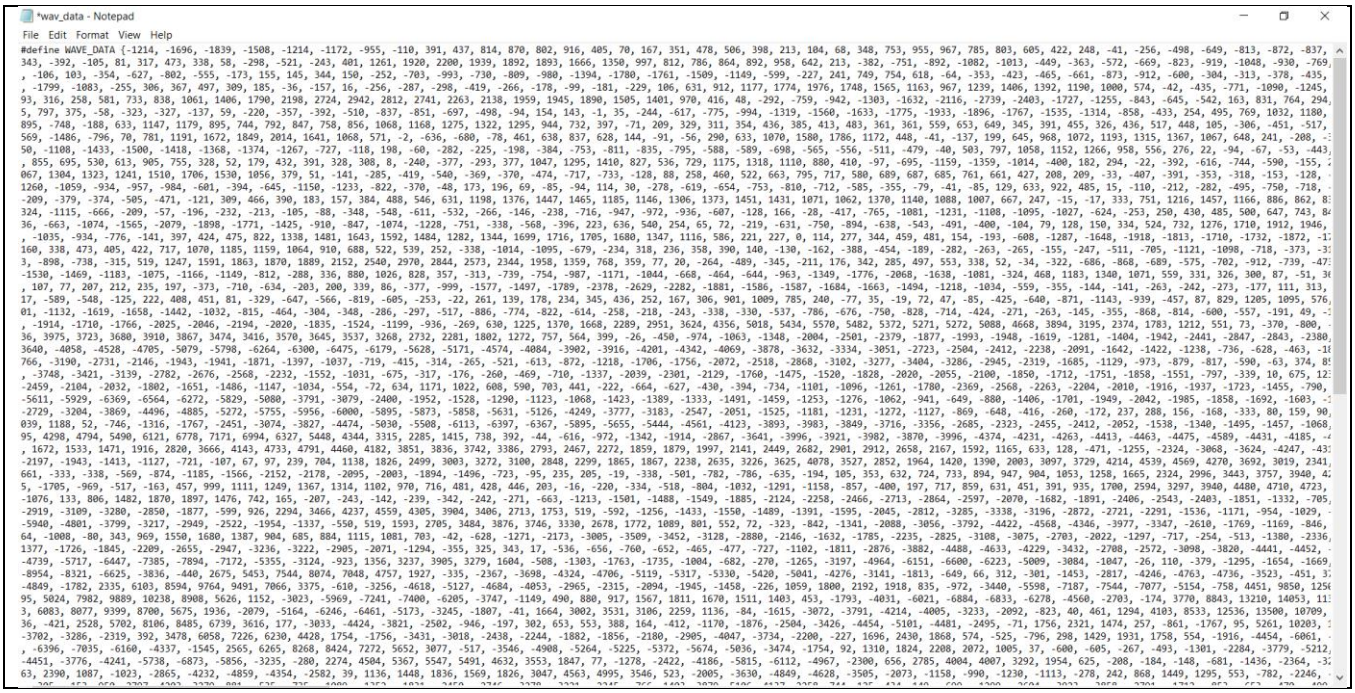
5. Modify the coding in main.cpp: -
   - Reference: https://github.com/ARM-software/ML-KWS-for-MCU/blob/master/Deployment/Examples/simple_test/main.cpp

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "kws_ds_cnn.h"
#include "wav_data.h"
#include "stdio.h"
/* USER CODE END Includes */

  .
  .
  .
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */


    /* USER CODE BEGIN 3 */
          int16_t audio_buffer[16000]=WAVE_DATA;
          char output_class[12][8] = {"Silence", "Unknown","yes","no","up","down","left","right","on","off","stop","go"};
          KWS_DS_CNN kws(audio_buffer);

          kws.extract_features(); //extract mfcc features
          kws.classify();     //classify using dnn

          int max_ind = kws.get_top_class(kws.output);

          char buffer [70];
          int buffer_output = 0;
          buffer_output = sprintf(buffer, "Detected %s (%d%%)\r\n",output_class[max_ind],((int)kws.output[max_ind]*100/128));
          HAL_UART_Transmit(&huart2, (uint8_t *)buffer, buffer_output, 100);
          HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }
  /* USER CODE END 3 */
}
```

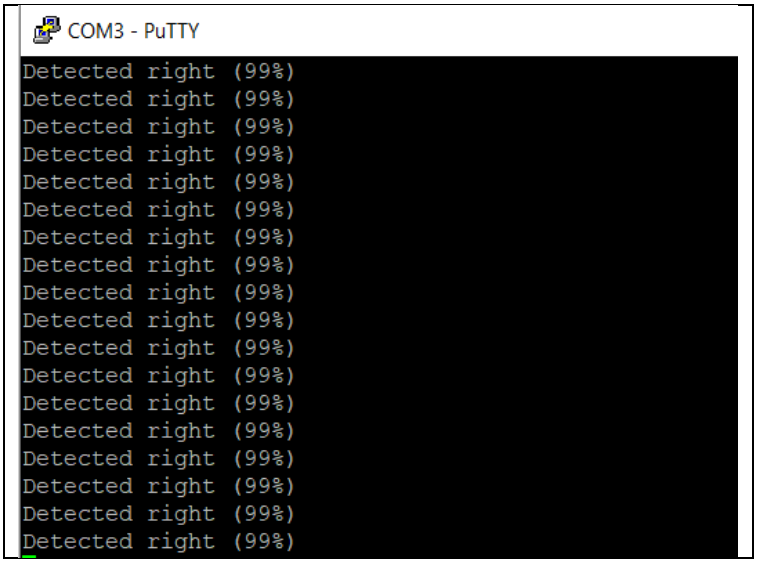
6. Open the wav_data.h file from the project folder.



7. Replace the data samples in the wav_data.h file with the audio data samples extracted from the previously recorded audio clip (Refer to the previous section). Save and close the file once done.



8. Build and run the project. Make sure the STM32 F446RE microcontroller is already connected to the laptop.
9. Open PuTTY and set the correct configurations. The output of the inference will then be displayed on PuTTY.



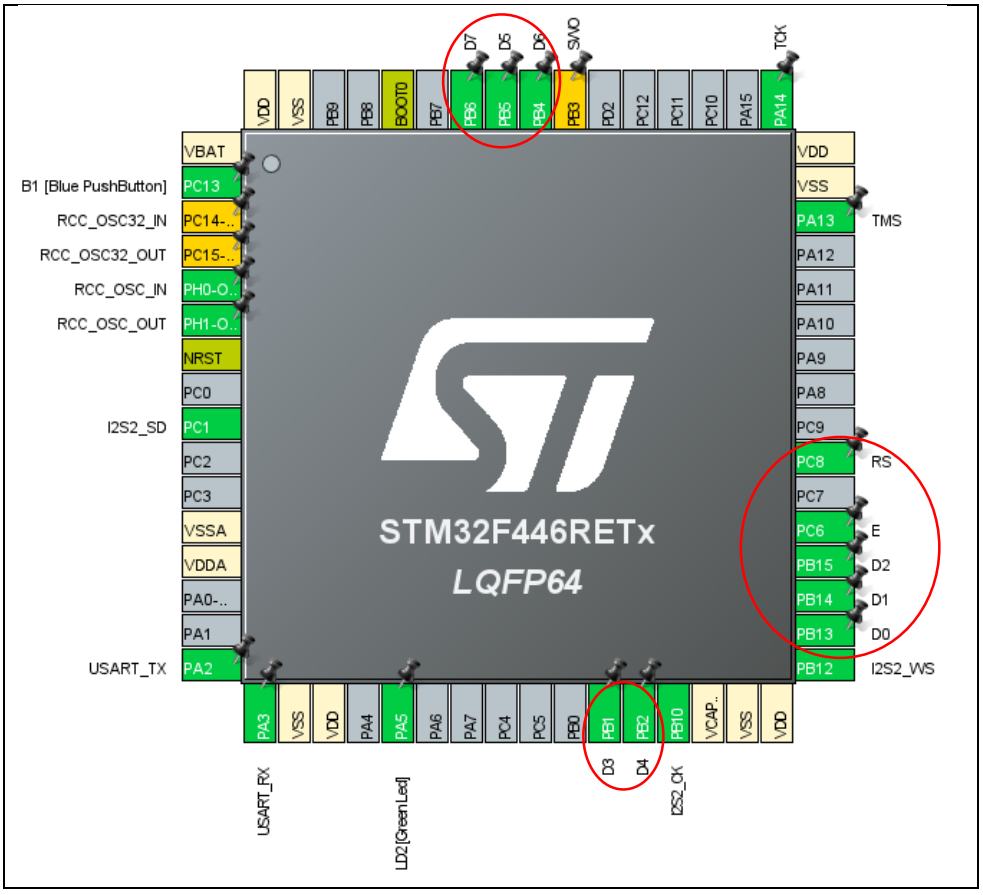
10. If you wish to test on the other keywords, simply update the wav_data.h file to the audio data samples of your desired audio clip and rebuild and run the project.

LCD Display for KWS Detection

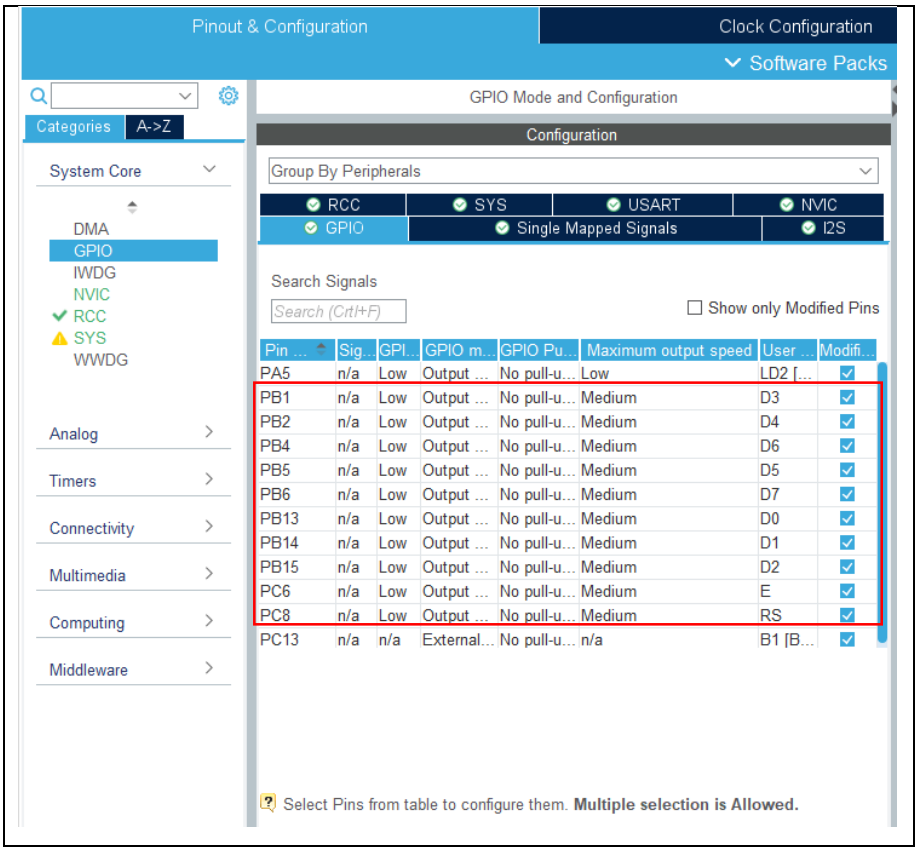
Created by Phuah Chai Ling

Instead of using the PuTTY serial console to display the outputs, an alternative is to use a LCD Display. A 20x4 LCD display is used in this project.

1. Connect the LCD to the microcontroller.
2. Create a new STM32CubeIDE project or open the previously built project. *Recommended to create a new project*
3. Initialize the peripherals. Set the pins to GPIO_outputs in order to connect the LCD display to the microcontroller.



4. Set the maximum output speed of those GPIO_output pins to “Medium”.



5. If a new project is being created, repeat the steps stated in the previous section to import all the required files to build the KWS algorithm.
6. Import the LCD source file and header file from this GitHub: https://github.com/MYaqoobEmbedded/STM32-Tutorials/tree/master/Tutorial%2011%20-%20LCD16x2

7. Modify the code in the main.cpp file.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "kws_ds_cnn.h"
#include "wav_data.h"
#include "stdio.h"
#include "lcd16x2.h"
/* USER CODE END Includes */
--
  .
  .
  .
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2S2_Init();
  /* USER CODE BEGIN 2 */
  lcd16x2_init_8bits(RS_GPIO_Port, RS_Pin, E_Pin,
                     D0_GPIO_Port, D0_Pin, D1_Pin, D2_Pin, D3_Pin,
                     D4_GPIO_Port, D4_Pin, D5_Pin, D6_Pin, D7_Pin);


  /* USER CODE END 2 */
  .
  .
  .
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      int16_t audio_buffer[20000]=WAVE_DATA;
      char output_class[12][8] = {"Silence", "Unknown","yes","no","up","down","left","right","on","off","stop","go"};
      KWS_DS_CNN kws(audio_buffer);

      kws.extract_features(); //extract mfcc features
      kws.classify();     //classify using dnn

      int max_ind = kws.get_top_class(kws.output);

      char buffer [70];
      int buffer_output = 0;
      buffer_output = sprintf(buffer, "Detected %s (%d%%)\r\n",output_class[max_ind],((int)kws.output[max_ind]*100/128));
      HAL_UART_Transmit(&huart2, (uint8_t *)buffer, buffer_output, 100);

      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

      lcd16x2_1stLine();
      lcd16x2_printf("Detected %s", output_class[max_ind]);
      lcd16x2_2ndLine();
      lcd16x2_printf("(%d%%)", ((int)kws.output[max_ind]*100/128 ));


  }
  /* USER CODE END 3 */
}
```

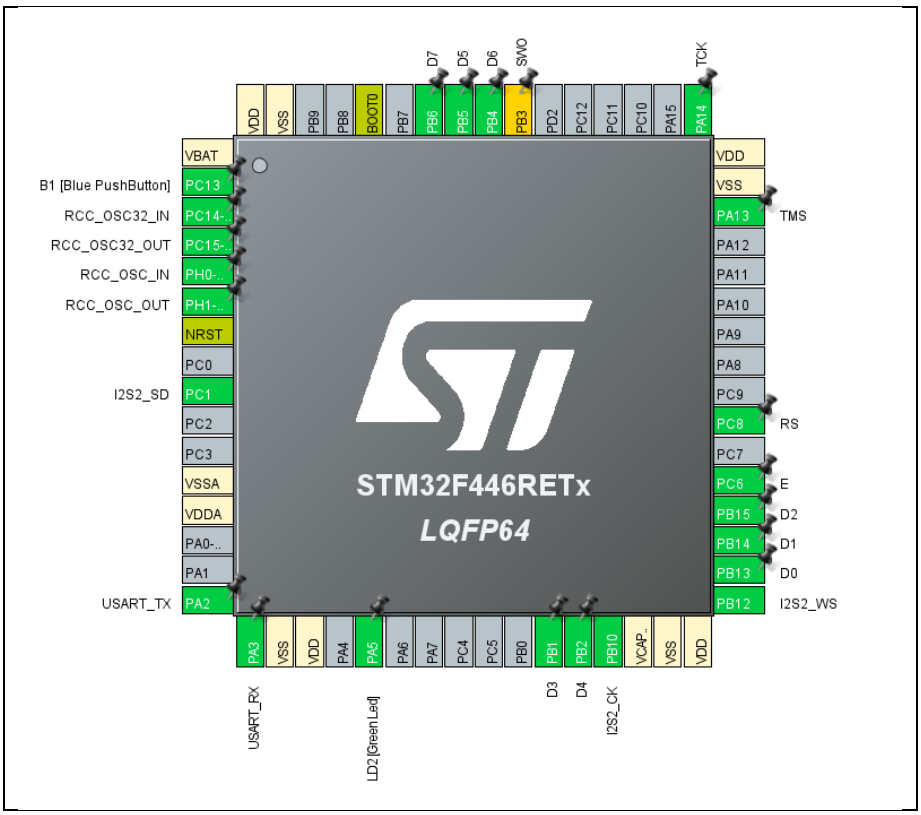
8. Ensure that the wav_data.h file contains the audio data samples that you wish to detect.
9. Build and run the project. Make sure the STM32 F446RE microcontroller is already connected to the laptop.
10. Example of output displayed on the LCD.

Keyword Spotting (KWS) Algorithm (Real Time Implementation)

Created by Phuah Chai Ling, Lim Calvin, Nur Nadia Muslim

This section basically shows the real-time implementation of the whole KWS system. Since the procedures of setting up the system have been elaborated part by part in detailed in the previous sections, this section will only showcase the final set-up of the peripherals as well as the final code.

1. The pin-out view of the peripherals set-up in STM32CubeIDE.



2. The final code snippets.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "kws_ds_cnn.h"
#include "stdio.h"
#include "lcd16x2.h"
/* USER CODE END Includes */

    .
    .
    .
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint8_t buf[12];
uint16_t data_in[2];
int16_t audio_buffer[16000];
/* USER CODE END 0 */

    .
    .
    .
```

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2S2_Init();
  /* USER CODE BEGIN 2 */
  lcd16x2_init_8bits(RS_GPIO_Port, RS_Pin, E_Pin,
                     D0_GPIO_Port, D0_Pin, D1_Pin, D2_Pin, D3_Pin,
                     D4_GPIO_Port, D4_Pin, D5_Pin, D6_Pin, D7_Pin);
  /* USER CODE END 2 */
.
.
.
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          HAL_Delay(100);
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);  //calvin

          for (int i=0;i<16000;i+=2){
              volatile HAL_StatusTypeDef result = HAL_I2S_Receive(&hi2s2, data_in, 2, 100);
              if ( result != HAL_OK ) {
                  strcpy((char*)buf, "Error Rx\r\n");\
                  }else{
                      audio_buffer[i]   = (int16_t)data_in[0];
                      audio_buffer[i+1] = (int16_t)data_in[1];
                  }
              }

          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);    //calvin


          char output_class[12][8] = {"Silence", "Unknown","yes","no","up","down","left","right","on","off","stop","go"};
          KWS_DS_CNN kws(audio_buffer);

          kws.extract_features(); //extract mfcc features
          kws.classify();     //classify using dnn

          int max_ind = kws.get_top_class(kws.output);

          char buffer [70];
          int buffer_output = 0;
          buffer_output = sprintf(buffer, "Detected %s (%d%%)\r\n",output_class[max_ind],((int)kws.output[max_ind]*100/128));
          HAL_UART_Transmit(&huart2, (uint8_t *)buffer, buffer_output, 100);



          lcd16x2_1stLine();
          lcd16x2_printf(" %s", output_class[max_ind]);
          lcd16x2_2ndLine();
          lcd16x2_printf("(%d%%)", ((int)kws.output[max_ind]*100/128 ));
          HAL_Delay(1000);
          lcd16x2_clear();

  }
  /* USER CODE END 3 */
}
```

(Optional) Neural Network (NN) Tensorflow Lite

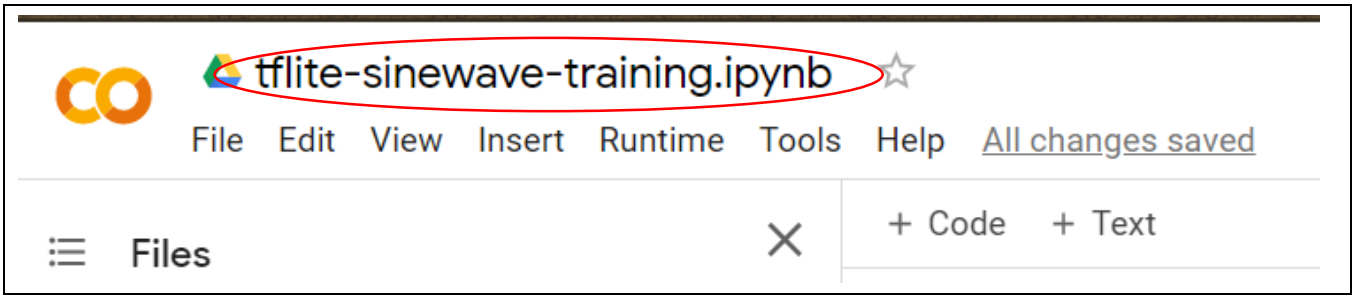
Created by Nur Nadia Muslim

The NN of Tensorflow Lite has been built to perform the Sine model interface. The NN inputs in raw bytes data had been successfully generated in X-CUBE-AI directory and these inputs can provide the sine details output in serial logs. Since the project we are currently focusing on is towards Keyword Spotting (KWS) algorithm. Therefore, this progress has been discontinued however, below are the recorded methods in case the progress resumes in the near future.

1. Sign up for a https://accounts.google.com/login and head to https://colab.research.google.com/

2. Generate TensorFlow model: -

   i. Rename it as ***tflite-sinewave-training.ipynb*** as shown below: -



   ii. Feed the code required: -

   - Source by github: https://gist.github.com/ShawnHymel/79237fe6aee5a3653c497d879f746c0c

```
%tensorflow_version 2.1

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import math
from tensorflow.keras import layers
```

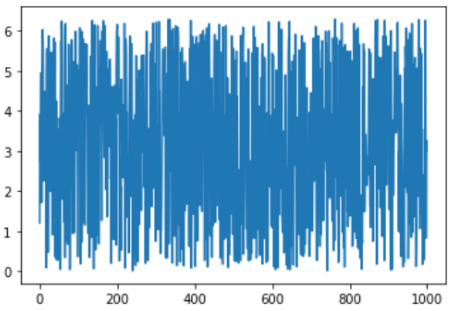
```
`%tensorflow_version` only switches the major version: 1.x or 2.x.
You set: `2.1`. This will be interpreted as: `2.x`.


TensorFlow 2.x selected.
```

```
[4] # Print versions
    !python --versions
    print('Numpy ' + np.__version__)
    print('TensorFlow ' + tf.__version__)
    print('Keras ' + tf.keras.__version__)
```

```
Unknown option: --
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try `python -h' for more information.
Numpy 1.19.5
TensorFlow 2.4.0
Keras 2.4.0
```

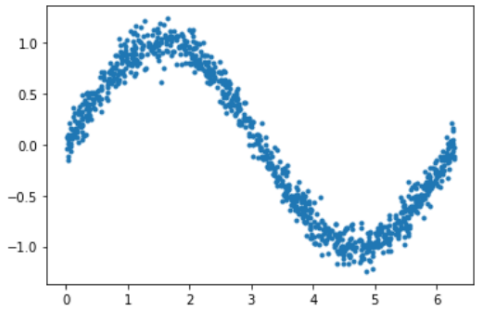
[6]
```
# Settings
nsamples = 1000      # Number of samples to use as a dataset
val_ratio = 0.2      # Percentage of samples that should be held for validation set
test_ratio = 0.2     # Percentage of samples that should be held for test set
tflite_model_name = 'sine_model'  # Will be given .tflite suffix
c_model_name = 'sine_model'       # Will be given .h suffix

# Generate some random samples
np.random.seed(1234)
x_values = np.random.uniform(low=0, high=(2 * math.pi), size=nsamples)
plt.plot(x_values)
```

```
[<matplotlib.lines.Line2D at 0x7fa00d6c5f28>]
```

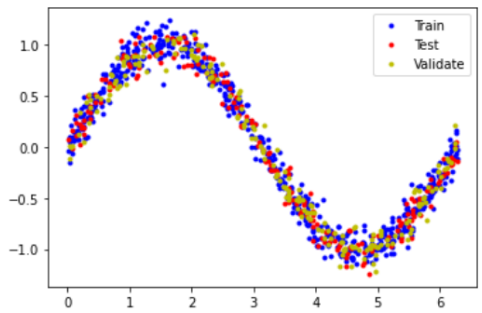


[7]
```
# Create a noisy sinewave with these values
y_values = np.sin(x_values) + (0.1 * np.random.randn(x_values.shape[0]))
plt.plot(x_values, y_values, '.')
```

```
[<matplotlib.lines.Line2D at 0x7fa00d19c630>]
```



[8]
```
# Plit the dataset into training, validation, and test sets
val_split = int(val_ratio * nsamples)
test_split = int(val_split + (test_ratio * nsamples))
x_val, x_test, x_train = np.split(x_values, [val_split, test_split])
y_val, y_test, y_train = np.split(y_values, [val_split, test_split])

# Check that our splits add up correctly
assert(x_train.size + x_val.size + x_test.size) == nsamples

# Plot the data in each partition in different colors:
plt.plot(x_train, y_train, 'b.', label="Train")
plt.plot(x_test, y_test, 'r.', label="Test")
plt.plot(x_val, y_val, 'y.', label="Validate")
plt.legend()
plt.show()
```



```
[9] # Create a model
    model = tf.keras.Sequential()
    model.add(layers.Dense(16, activation='relu', input_shape=(1,)))
    model.add(layers.Dense(16, activation='relu'))
    model.add(layers.Dense(1))

    # View model
    model.summary()
```

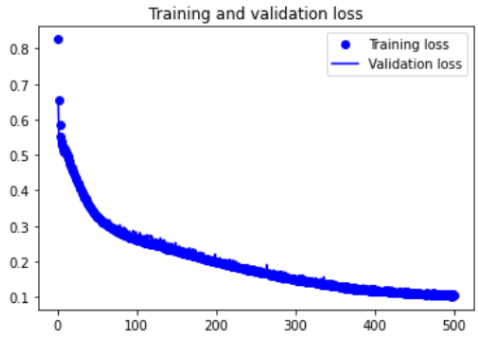
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                32
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
```

```
# Add optimizer, loss function, and metrics to model and compile it
model.compile(optimizer='rmsprop', loss='mae', metrics=['mae'])

# Train model
history = model.fit(x_train,
                    y_train,
                    epochs=500,
                    batch_size=100,
                    validation_data=(x_val, y_val))
```

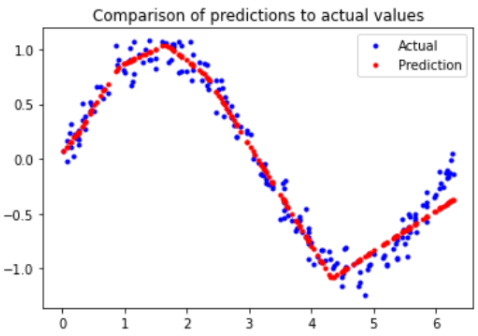
```
Epoch 471/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1040 - mae: 0.1040 - val_loss: 0.1096 - val_mae: 0.1096
Epoch 472/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1044 - mae: 0.1044 - val_loss: 0.1074 - val_mae: 0.1074
Epoch 473/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1050 - mae: 0.1050 - val_loss: 0.1090 - val_mae: 0.1090
Epoch 474/500
6/6 [==============================] - 0s 9ms/step - loss: 0.1027 - mae: 0.1027 - val_loss: 0.1120 - val_mae: 0.1120
Epoch 475/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1134 - mae: 0.1134 - val_loss: 0.1078 - val_mae: 0.1078
Epoch 476/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1053 - mae: 0.1053 - val_loss: 0.1087 - val_mae: 0.1087
Epoch 477/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1019 - mae: 0.1019 - val_loss: 0.1042 - val_mae: 0.1042
Epoch 478/500
6/6 [==============================] - 0s 11ms/step - loss: 0.0988 - mae: 0.0988 - val_loss: 0.1054 - val_mae: 0.1054
Epoch 479/500
6/6 [==============================] - 0s 33ms/step - loss: 0.1058 - mae: 0.1058 - val_loss: 0.1083 - val_mae: 0.1083
Epoch 480/500
Epoch 481/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1087 - mae: 0.1087 - val_loss: 0.1055 - val_mae: 0.1055
Epoch 482/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1053 - mae: 0.1053 - val_loss: 0.1079 - val_mae: 0.1079
Epoch 483/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1076 - mae: 0.1076 - val_loss: 0.1090 - val_mae: 0.1090
Epoch 484/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1064 - mae: 0.1064 - val_loss: 0.1043 - val_mae: 0.1043
Epoch 485/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1012 - mae: 0.1012 - val_loss: 0.1108 - val_mae: 0.1108
Epoch 486/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1074 - mae: 0.1074 - val_loss: 0.1097 - val_mae: 0.1097
Epoch 487/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1082 - mae: 0.1082 - val_loss: 0.1063 - val_mae: 0.1063
Epoch 488/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1032 - mae: 0.1032 - val_loss: 0.1045 - val_mae: 0.1045
Epoch 489/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1045 - mae: 0.1045 - val_loss: 0.1086 - val_mae: 0.1086
Epoch 490/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1100 - mae: 0.1100 - val_loss: 0.1081 - val_mae: 0.1081
Epoch 491/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1013 - mae: 0.1013 - val_loss: 0.1048 - val_mae: 0.1048
Epoch 492/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1068 - mae: 0.1068 - val_loss: 0.1075 - val_mae: 0.1075
Epoch 493/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1011 - mae: 0.1011 - val_loss: 0.1093 - val_mae: 0.1093
Epoch 494/500
```

```
6/6 [==============================] - 0s 10ms/step - loss: 0.1024 - mae: 0.1024 - val_loss: 0.1109 - val_mae: 0.1109
Epoch 495/500
6/6 [==============================] - 0s 11ms/step - loss: 0.1071 - mae: 0.1071 - val_loss: 0.1040 - val_mae: 0.1040
Epoch 496/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1089 - mae: 0.1089 - val_loss: 0.1050 - val_mae: 0.1050
Epoch 497/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1019 - mae: 0.1019 - val_loss: 0.1040 - val_mae: 0.1040
Epoch 498/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1026 - mae: 0.1026 - val_loss: 0.1036 - val_mae: 0.1036
Epoch 499/500
6/6 [==============================] - 0s 8ms/step - loss: 0.1065 - mae: 0.1065 - val_loss: 0.1011 - val_mae: 0.1011
Epoch 500/500
6/6 [==============================] - 0s 10ms/step - loss: 0.1024 - mae: 0.1024 - val_loss: 0.1023 - val_mae: 0.1023
```

```
# Plot the training history
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```




```
# Plot predictions against actual values
predictions = model.predict(x_test)

plt.clf()
plt.title("Comparison of predictions to actual values")
plt.plot(x_test, y_test, 'b.', label='Actual')
plt.plot(x_test, predictions, 'r.', label='Prediction')
plt.legend()
plt.show()
```

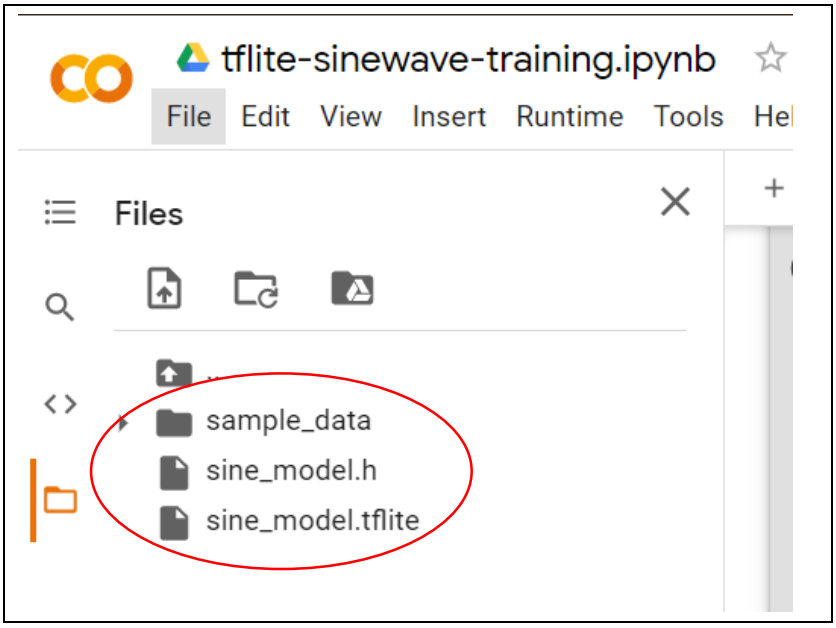



```
# Convert Keras model to a tflite model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()

open(tflite_model_name + '.tflite', 'wb').write(tflite_model)
```

```
INFO:tensorflow:Assets written to: /tmp/tmppmh8a0_5/assets
2960
```

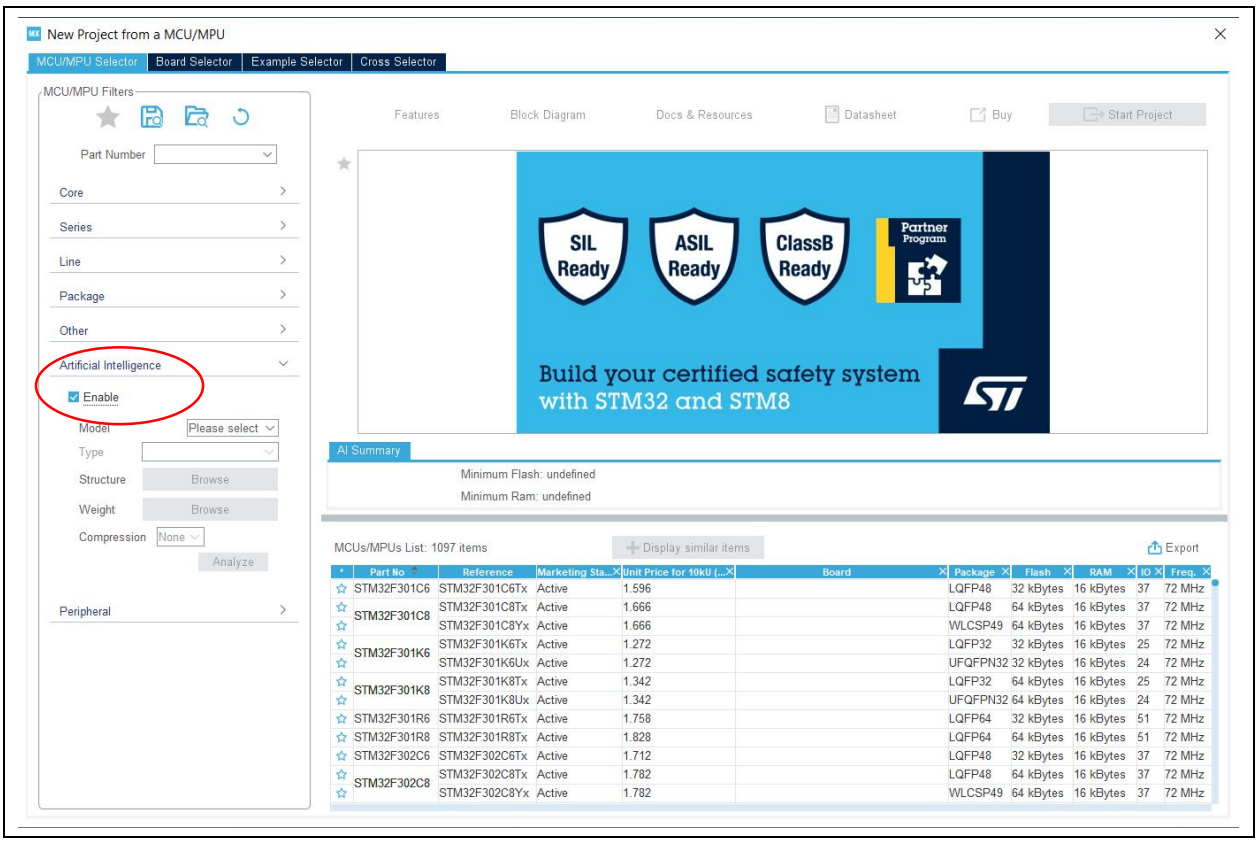
```
# Function: Convert some hex value into an array for C programming
def hex_to_c_array(hex_data, var_name):

  c_str = ''

  # Create header guard
  c_str += '#ifndef ' + var_name.upper() + '_H\n'
  c_str += '#define ' + var_name.upper() + '_H\n\n'

  # Add array length at top of file
  c_str += '\nunsigned int ' + var_name + '_len = ' + str(len(hex_data)) + ';\n'

  # Declare C variable
  c_str += 'unsigned char ' + var_name + '[] = {'
  hex_array = []
  for i, val in enumerate(hex_data) :

    # Construct string from hex
    hex_str = format(val, '#04x')

    # Add formatting so each line stays within 80 characters
    if (i + 1) < len(hex_data):
      hex_str += ','
    if (i + 1) % 12 == 0:
      hex_str += '\n '
    hex_array.append(hex_str)

  # Add closing brace
  c_str += '\n ' + format(' '.join(hex_array)) + '\n};\n\n'

  # Close out header guard
  c_str += '#endif //' + var_name.upper() + '_H'

  return c_str

  # Write TFLite model to a C source (or header) file
with open(c_model_name + '.h', 'w') as file:
  file.write(hex_to_c_array(tflite_model, c_model_name))
```

iii. The ***sine_model.h*** and ***sine_model.tflite*** is generated after running the last code: -

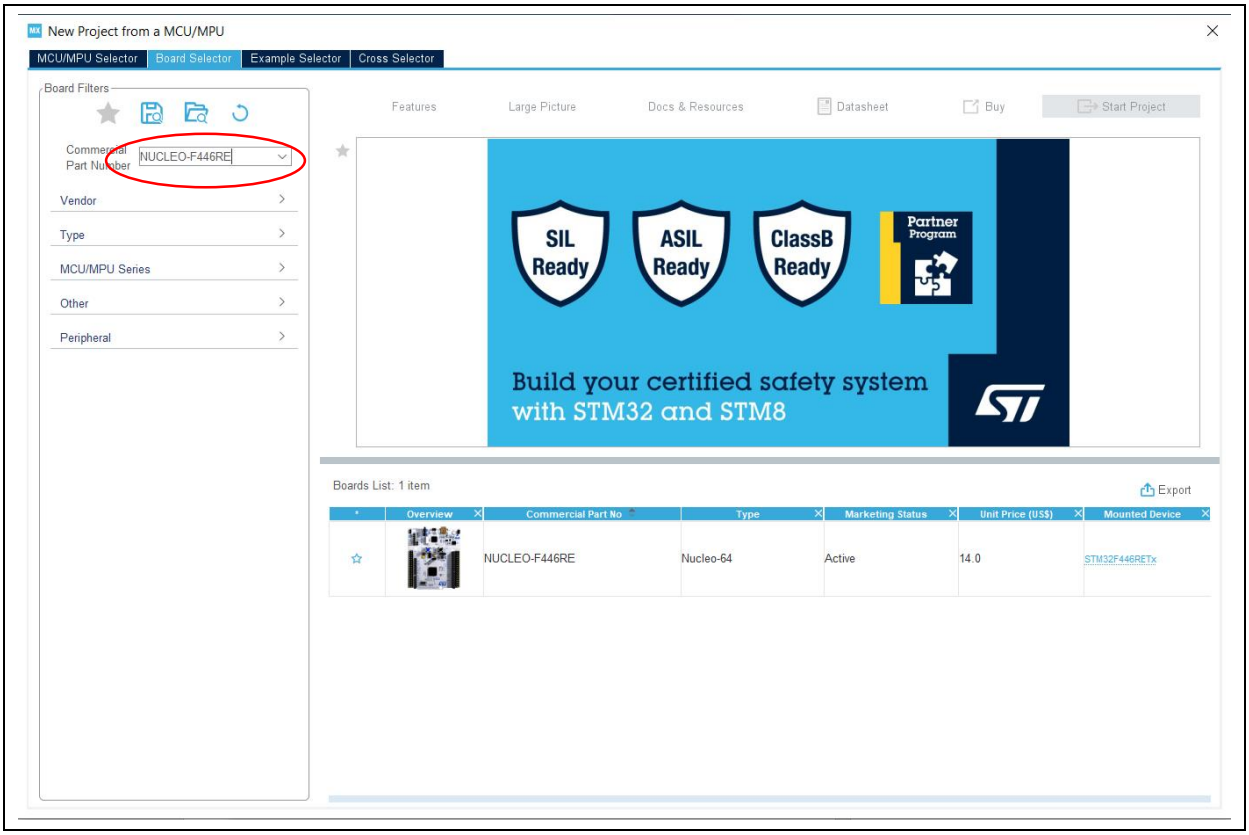


3. Settings in STM32CubeMX for generating Tensorflow: -

i. Enable the Artificial Intelligence: -

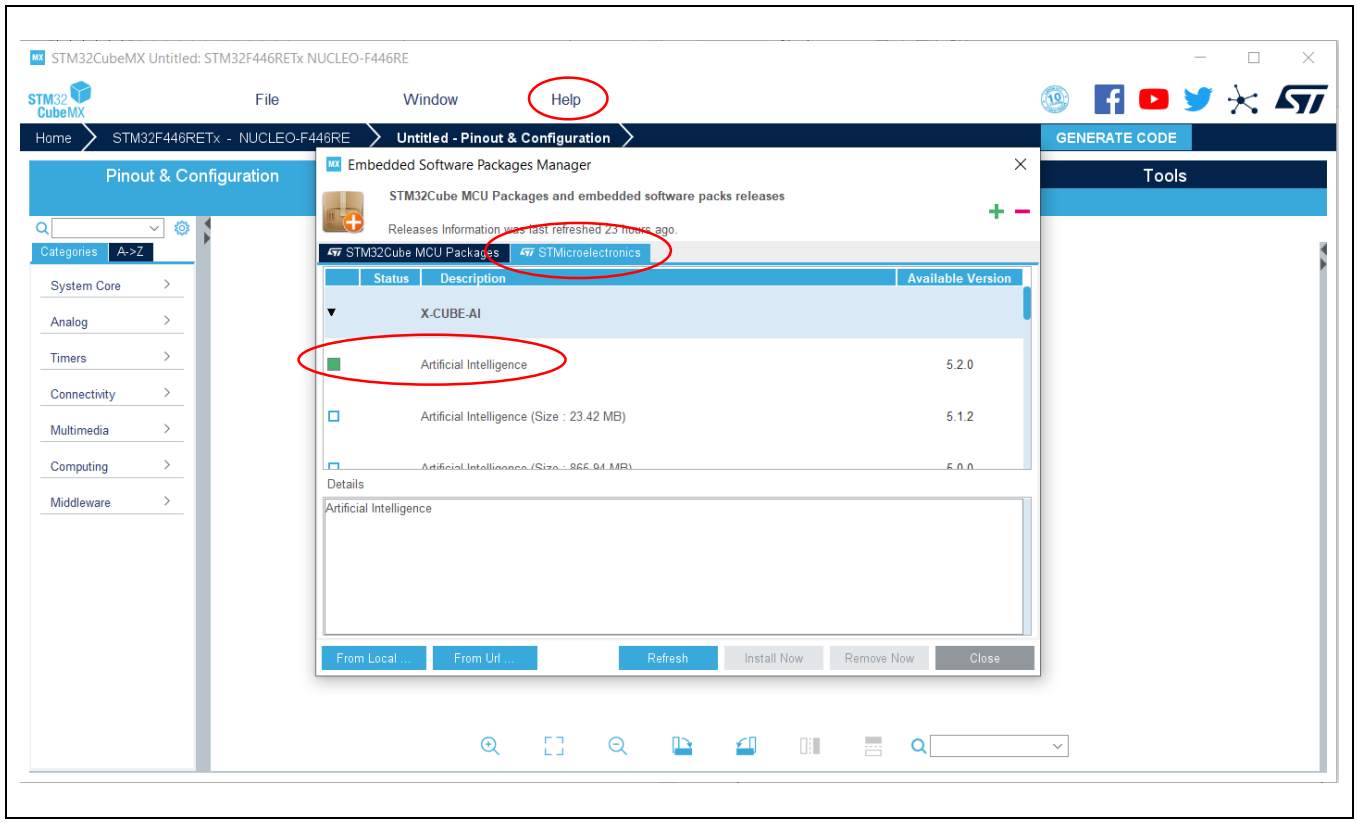


ii. Select the NUCLEO board required: -

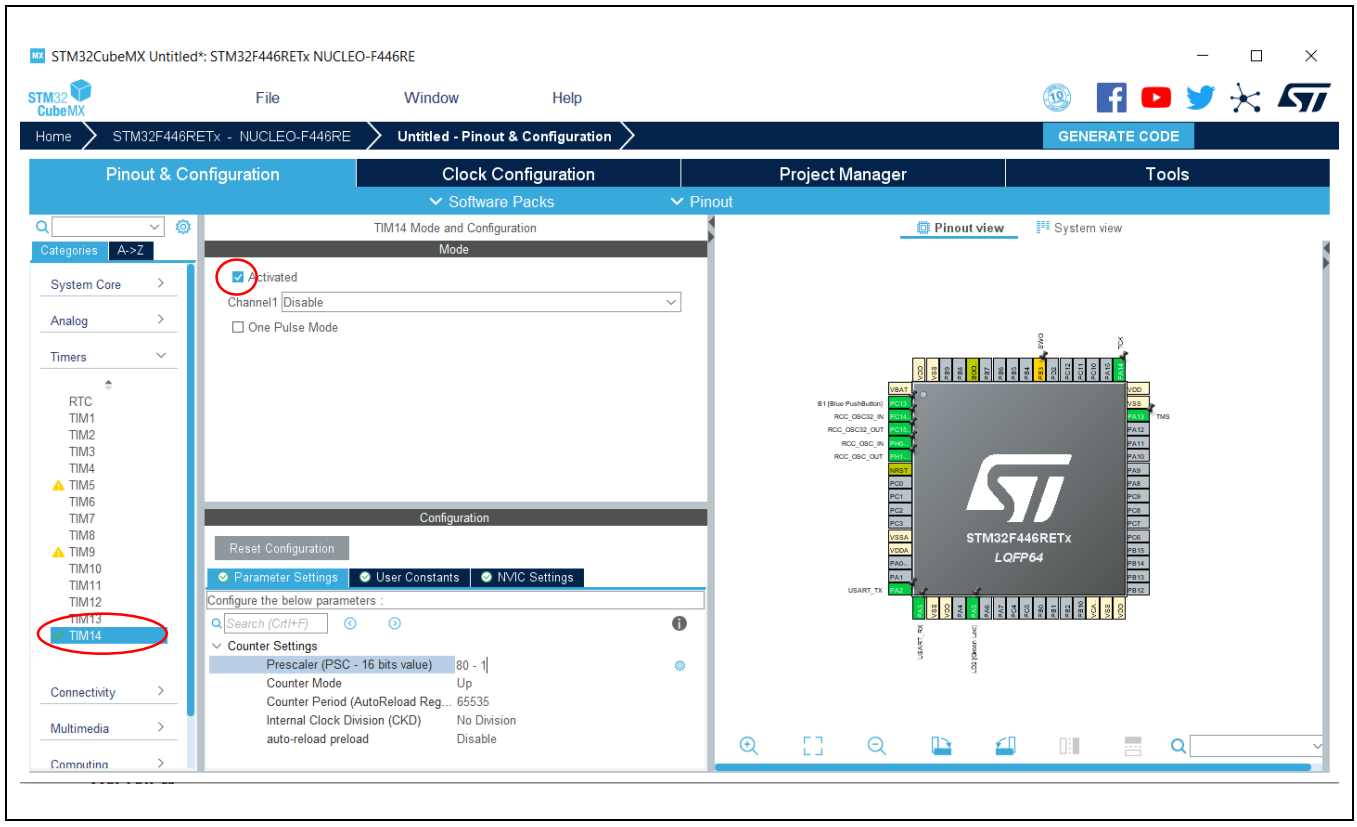


iii. Install X-CUBE-AI as ***Artifical Intelligence***: -

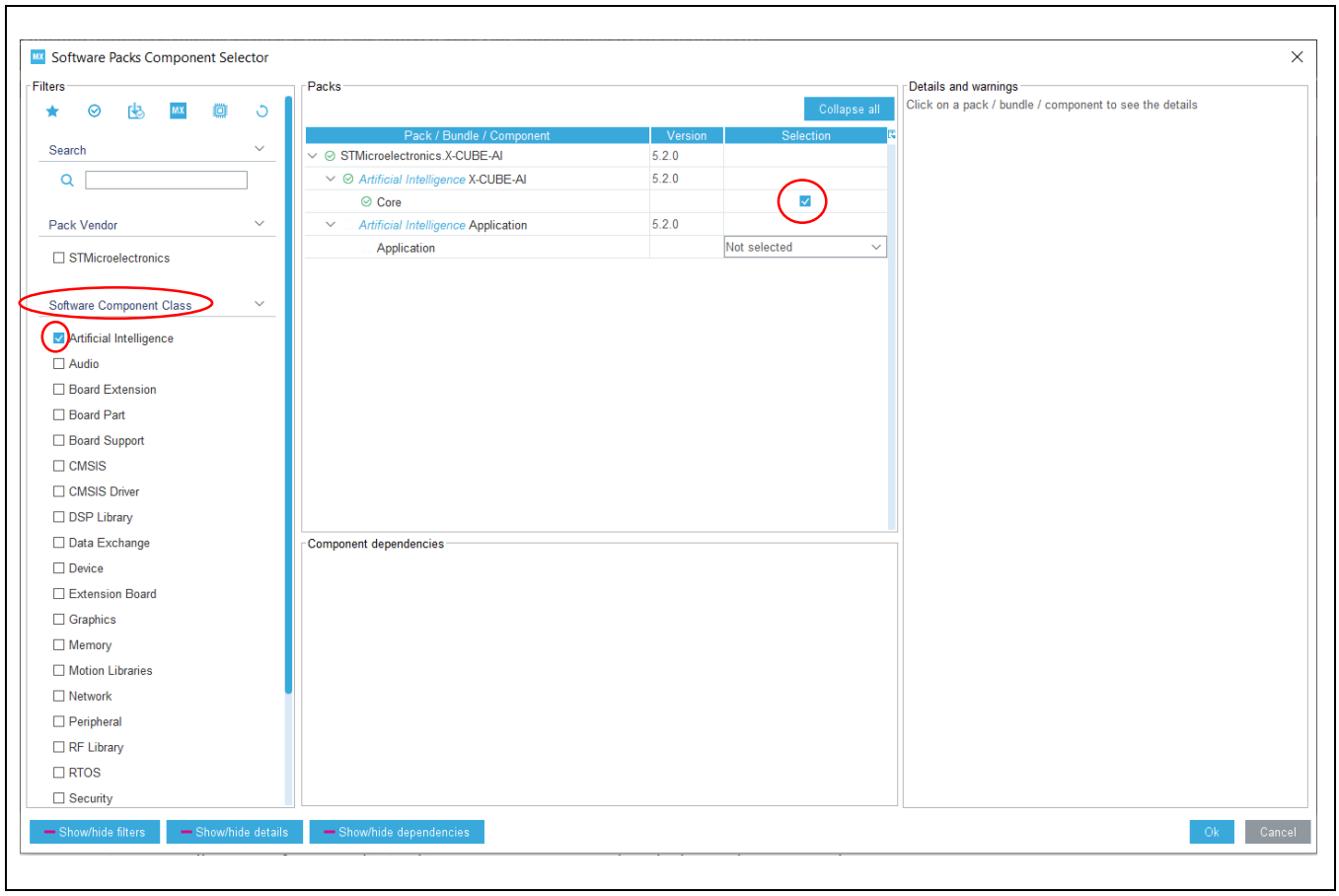
- Purpose: To generate the ***X-CUBE-AI*** folder under the project in STM32CubeIDE



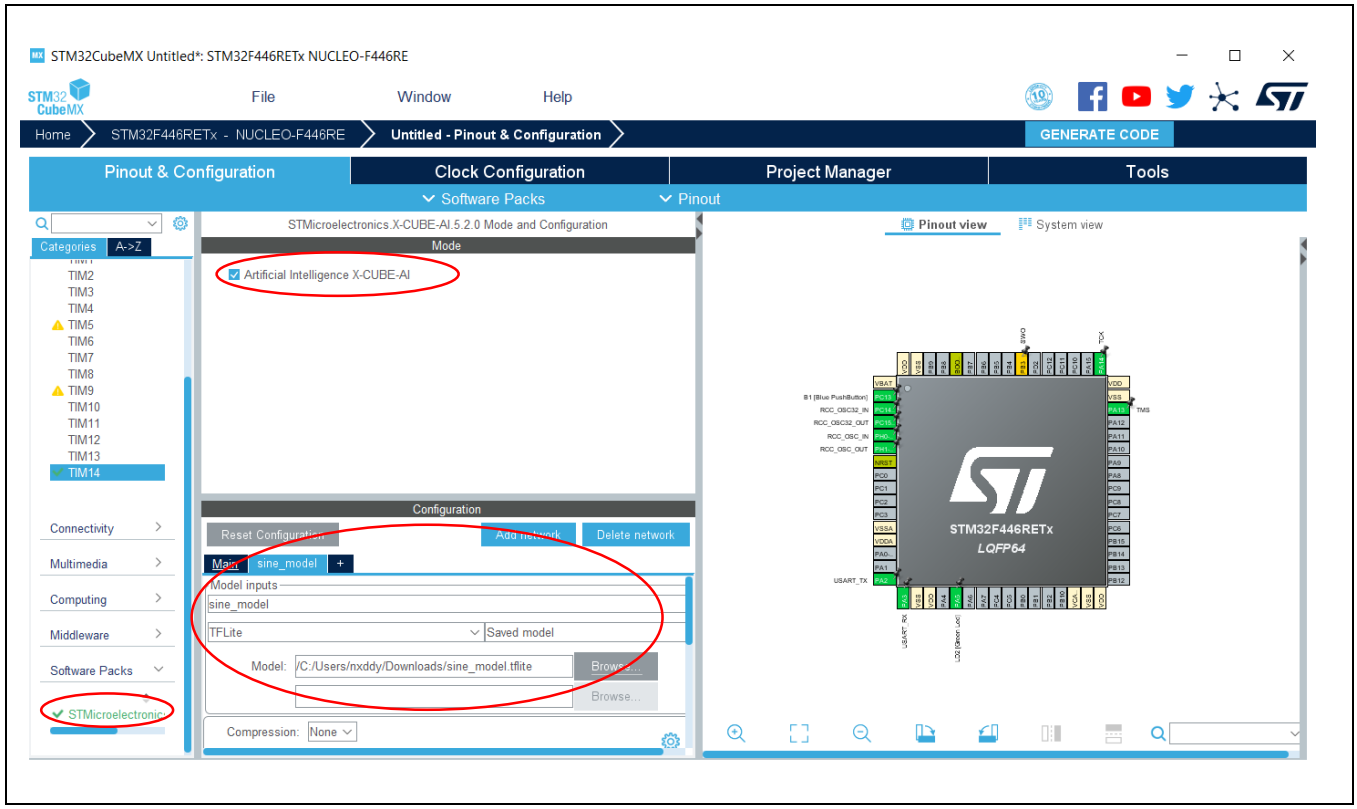
iv. Activated the Timer: -



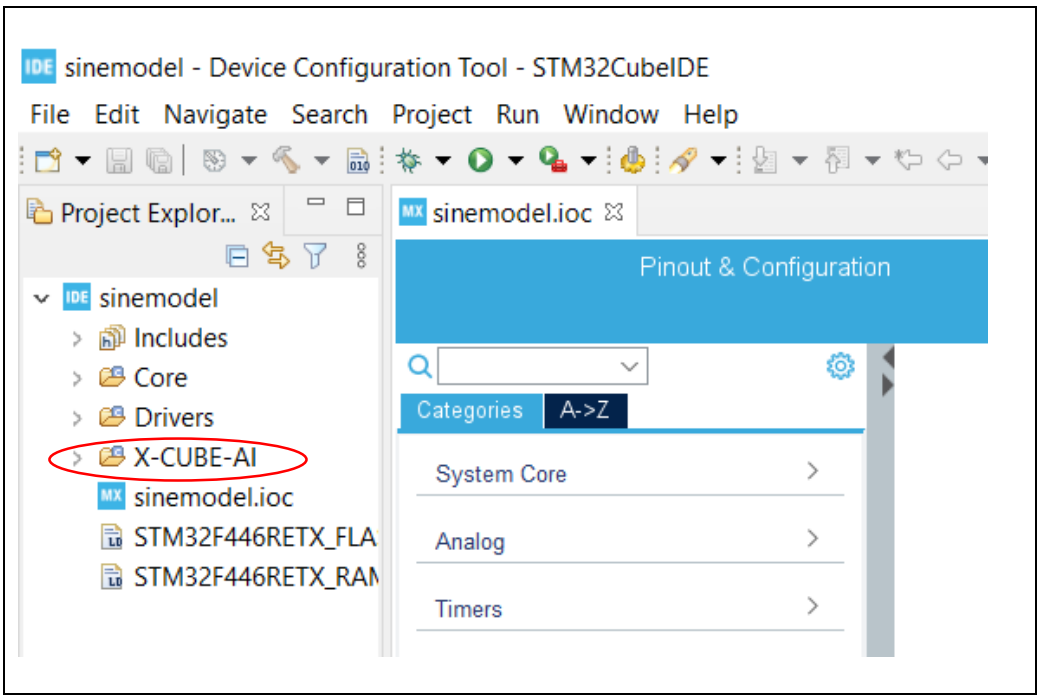
v. Select the software package as ***Artificial Intelligence*** for "Core": -



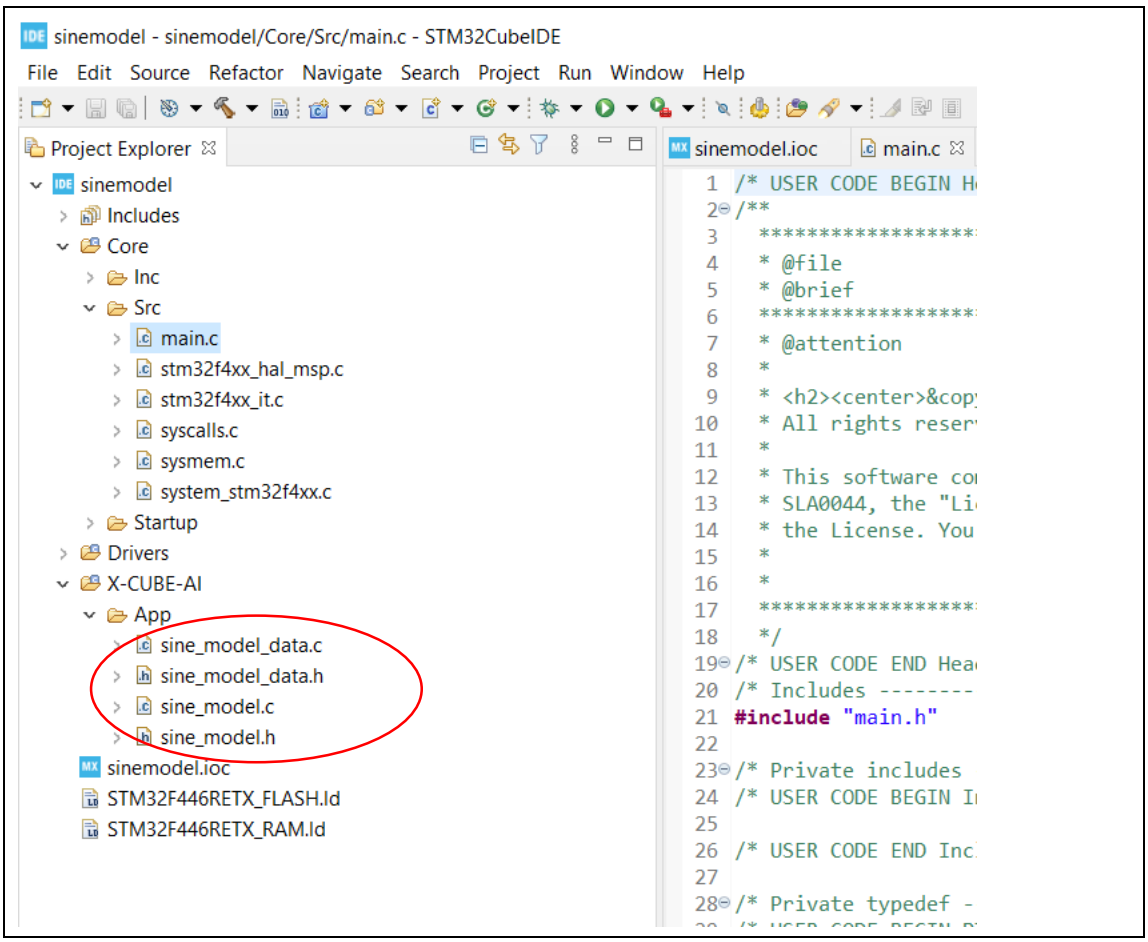
vi. Imported the generated ***sine_model.tflite*** created from ***google.colab***: -



4. Generate the project, normally.
5. In STM32CubeIDE, the ***X-CUBE-AI*** folder is listed under the project file: -



6. The ***X-CUBE-AI*** folder will contain the files below: -



7. Program code in main.c file: -

```
/* USER CODE BEGIN Includes */
#include <stdio.h>

#include "ai_datatypes_defines.h"
#include "ai_platform.h"
#include "sine_model.h"
#include "sine_model_data.h"
/* USER CODE END Includes */
```

.
.
.

```
int main(void)
{
  /* USER CODE BEGIN 1 */
    char buf[50];
      int buf_len = 0;
      ai_error ai_err;
      ai_i32 nbatch;
      uint32_t timestamp;
      float y_val;

      // Chunk of memory used to hold intermediate values for neural network
        AI_ALIGNED(4) ai_u8 activations[AI_SINE_MODEL_DATA_ACTIVATIONS_SIZE];

        // Buffers used to store input and output tensors
        AI_ALIGNED(4) ai_i8 in_data[AI_SINE_MODEL_IN_1_SIZE_BYTES];
        AI_ALIGNED(4) ai_i8 out_data[AI_SINE_MODEL_OUT_1_SIZE_BYTES];

        // Pointer to our model
        ai_handle sine_model = AI_HANDLE_NULL;

        // Initialize wrapper structs that hold pointers to data and info about the
        // data (tensor height, width, channels)
        ai_buffer ai_input[AI_SINE_MODEL_IN_NUM] = AI_SINE_MODEL_IN;
        ai_buffer ai_output[AI_SINE_MODEL_OUT_NUM] = AI_SINE_MODEL_OUT;

        // Set working memory and get weights/biases from model
        ai_network_params ai_params = {
          AI_SINE_MODEL_DATA_WEIGHTS(ai_sine_model_data_weights_get()),
          AI_SINE_MODEL_DATA_ACTIVATIONS(activations)
        };

        // Set pointers wrapper structs to our data buffers
        ai_input[0].n_batches = 1;
        ai_input[0].data = AI_HANDLE_PTR(in_data);
        ai_output[0].n_batches = 1;
        ai_output[0].data = AI_HANDLE_PTR(out_data);
  /* USER CODE END 1 */
```

.
.
.

```
137   // Start timer/counter
138     HAL_TIM_Base_Start(&htim14);
139
140     // Greetings!
141     buf_len = sprintf(buf, "\r\n\r\nSTM32 X-Cube-AI test\r\n");
142     HAL_UART_Transmit(&huart2, (uint8_t *)buf, buf_len, 100);
143
144     // Create instance of neural network
145     ai_err = ai_sine_model_create(&sine_model, AI_SINE_MODEL_DATA_CONFIG);
146     if (ai_err.type != AI_ERROR_NONE)
147     {
148       buf_len = sprintf(buf, "Error: could not create NN instance\r\n");
149       HAL_UART_Transmit(&huart2, (uint8_t *)buf, buf_len, 100);
150       while(1);
151     }
152
153     // Initialize neural network
154     if (!ai_sine_model_init(sine_model, &ai_params))
155     {
156       buf_len = sprintf(buf, "Error: could not initialize NN\r\n");
157       HAL_UART_Transmit(&huart2, (uint8_t *)buf, buf_len, 100);
158       while(1);
159     }
```

.
.
.

```
165   while (1)
166   {
167       // Fill input buffer (use test value)
168         for (uint32_t i = 0; i < AI_SINE_MODEL_IN_1_SIZE; i++)
169         {
170           ((ai_float *)in_data)[i] = (ai_float)2.0f;
171         }
172
173         // Get current timestamp
174         timestamp = htim14.Instance->CNT;
175
176         // Perform inference
177         nbatch = ai_sine_model_run(sine_model, &ai_input[0], &ai_output[0]);
178         if (nbatch != 1) {
179           buf_len = sprintf(buf, "Error: could not run inference\r\n");
180           HAL_UART_Transmit(&huart2, (uint8_t *)buf, buf_len, 100);
181         }
182
183         // Read output (predicted y) of neural network
184         y_val = ((float *)out_data)[0];
185
186         // Print output of neural network along with inference time (microseconds)
187         buf_len = sprintf(buf,
188                           "Output: %f | Duration: %lu\r\n",
189                           y_val,
190                           htim14.Instance->CNT - timestamp);
191         HAL_UART_Transmit(&huart2, (uint8_t *)buf, buf_len, 100);
192         // Start timer/counter
193
194         // Wait before doing it again
195         HAL_Delay(500);
196     /* USER CODE END WHILE */
```

8. In the main.c will get error after compiled and run the code. Hence modified the configurations as followed:
   >> Project
   >> Properties
   >> C/C++ Build
   >> Settings
   >> Tool Settings
   >> MCU GCC Linker
   >> Miscellaneous
   >> Enter value: -u_printf_float

9. The NN inputs as raw byte in ***sine_model_data.h*** under ***X-CUBE-AI*** folder: -

```
#include "sine_model_data.h"

ai_handle ai_sine_model_data_weights_get(void)
{

  AI_ALIGNED(4)
  static const ai_u8 s_sine_model_weights[ 1284 ] = {
    0x14, 0x07, 0xd7, 0xbe, 0x10, 0xf6, 0xb9, 0xbe, 0x32, 0x5e,
    0x74, 0xbe, 0xfc, 0x38, 0xa8, 0x3e, 0x53, 0xac, 0xcd, 0xbe,
    0xa5, 0x39, 0x10, 0xbe, 0x82, 0xd6, 0x7e, 0x3e, 0xd8, 0x68,
    0x86, 0xbd, 0x54, 0x5a, 0xcf, 0xbd, 0x07, 0x4c, 0xa8, 0x3e,
    0x5e, 0xfd, 0xb3, 0xbc, 0x1a, 0xb6, 0x5a, 0xbe, 0x2c, 0x88,
    0xd5, 0x3e, 0x80, 0x55, 0x63, 0xbd, 0x12, 0xfb, 0x05, 0xbf,
    0xa8, 0x30, 0x19, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe8, 0x0b, 0x86, 0x3e,
    0x00, 0x00, 0x00, 0x00, 0xb1, 0xc6, 0x67, 0x3f, 0x58, 0x61,
    0x13, 0xbf, 0xfa, 0x87, 0x93, 0x3f, 0x00, 0x00, 0x00, 0x00,
    0xcb, 0x53, 0x2e, 0xbe, 0x51, 0xa1, 0x81, 0x3f, 0x00, 0x00,
    0x00, 0x00, 0x17, 0x46, 0x60, 0xbe, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xb8, 0x0a, 0x41, 0xbe, 0xdc, 0xdc,
    0xf6, 0x3d, 0x0c, 0x1b, 0xd2, 0xbd, 0xef, 0x1b, 0xc6, 0xbe,
    0xe3, 0x46, 0xa5, 0xbe, 0x62, 0x16, 0x09, 0x3e, 0x3d, 0x42,
    0xea, 0x3e, 0xd4, 0x62, 0xd3, 0xbd, 0xb3, 0xaa, 0xe4, 0x3e,
    0x20, 0x5c, 0x56, 0x3d, 0x05, 0x52, 0x88, 0xbc, 0x72, 0xe8,
    0x07, 0x3f, 0x73, 0x9e, 0xc1, 0x3e, 0x66, 0xdb, 0xc8, 0xbe,
    0xa7, 0x40, 0xc5, 0x3e, 0xdc, 0x1c, 0xb4, 0x3d, 0x00, 0x47,
    0x9e, 0x3e, 0x9c, 0x6a, 0xb4, 0x3d, 0x53, 0x06, 0xaf, 0x3e,
    0xb4, 0xc6, 0x85, 0xbe, 0x21, 0xfd, 0xff, 0x3e, 0x19, 0x48,
    0xab, 0xbe, 0xe1, 0x53, 0x16, 0x3f, 0x21, 0x71, 0xc6, 0xbf,
    0x09, 0x36, 0x1b, 0x3f, 0xc0, 0x8f, 0x80, 0xbe, 0x2e, 0xd9,
    0xa2, 0xbe, 0x50, 0xc5, 0xb2, 0x3e, 0xfe, 0xcd, 0x53, 0x3e,
    0x73, 0xe0, 0xbf, 0xbe, 0x20, 0x74, 0x0b, 0x3c, 0xac, 0x7b,
    0xe4, 0x3d, 0x8d, 0x55, 0x95, 0xbe, 0xce, 0x51, 0x72, 0xbe,
    0x26, 0xa5, 0xd3, 0xbe, 0x8c, 0x8a, 0xbb, 0x3d, 0x0d, 0x50,
    0xcf, 0xbe, 0x46, 0x7f, 0x6f, 0x3e, 0x28, 0x8f, 0xd8, 0xbe,
    0xdc, 0x9f, 0xc5, 0x3d, 0x5c, 0x8b, 0x90, 0xbe, 0x46, 0xd3,
    0xcd, 0xbe, 0xde, 0x6d, 0x31, 0x3e, 0xfb, 0xd8, 0x8c, 0xbe,
    0x1e, 0xf5, 0x73, 0x3e, 0x67, 0xb5, 0xd7, 0x3e, 0x6a, 0xb9,
    0x53, 0x3e, 0x80, 0x6c, 0xb5, 0x3c, 0x11, 0x53, 0x93, 0xbe,
    0x70, 0xa7, 0x4e, 0x3d, 0xd9, 0x6e, 0xaa, 0x3e, 0x3b, 0x5b,
    0x7b, 0xbe, 0x7e, 0x7c, 0xae, 0x3e, 0xee, 0xa9, 0x1c, 0xbe,
    0xe3, 0xd7, 0x30, 0x3e, 0x55, 0xb0, 0xa4, 0xbe, 0x20, 0x8f,
    0xf3, 0x3e, 0x73, 0xcb, 0xc2, 0x3e, 0x8e, 0x51, 0x8f, 0xbd,
    0x8c, 0x95, 0xaf, 0x3d, 0xbc, 0xbe, 0xc7, 0x3d, 0x7d, 0x25,
```

10. Run the code in Debug mode.
11. Open puTTy, then observe the serial logs as displayed below: -