

**MKEL 1123 ADVANCED MICROPROCESSOR SYSTEM**

**VOICE CONTROL FOR LIGHTING USING KEYWORD SPOTTING PROCEDURE DOCUMENTATION**

**Group 1**

**Dr. Mohd Afzan Othman**

The development of this project can be divided into 4 stages:

1. Voice Control Module (VC-02) Integration for Speech Detection
2. Audio Signal Acquisition and Processing using i2s INMP441 and CMSIS DSP
3. AI Model Development for Keyword Spotting
4. External LED Control for Indication of Keyword Spotted

List of materials

1. STM32F446RE Microcontroller
2. Voice Control Module (VC-02)
3. i2s INMP441
4. CMSIS DSP  
   5. LEDs
5. **Voice Control Module (VC-02)**

The VC-02 module is a low-cost pure offline speech recognition module developed by Shenzhen, Ai-Thinker Technology Co., Ltd. The module uses the voice chip US516P6 launched by Unisound, and continuously optimizes and innovates algorithms in speech recognition technology. The VC02 module is designed to recognize specific voice commands and trigger actions based on those commands. It is commonly used in home automation, robotics, and other applications where hands-free control is beneficial. The module typically includes a microphone for capturing voice input, a microcontroller for processing the voice commands, a speaker to produce an output response voice based on the command given, and an interface for communicating with other devices, such as microcontrollers or microprocessors. The VC02 module can communicate with the STM32 board through a UART pin such as RX1 and TX1, and GPIO pins like IOB pins can be used to connect the peripheral input or output for various functions. For protocol pins such as SCL and SDA can be used for I2C communication while TCK and TMS can used for JTAG interface, which typically for debugging. Meanwhile, for DAC\_L and DAC\_R, these pins are responsible for the left and right audio channel’s digital-to-analog conversion. It outputs the analog signal for the left stereo component. Figure 1 below shows the pin diagram of VC02 module that can be leveraged as intended for this project.

**A close up of a computer chip

Description automatically generated**

Figure 1: VC02 module pin diagram.

1. **Procedure**

The VC02 module can be directly configured through online Firmware built from the developer, [http://voice.ai-thinker.com/#/](http://voice.ai-thinker.com/" \l "/). Once configuration generation is completed, user can download its Firmware and then burn it into the module. The below step entailed the flow of configuring the VC02 module.

1. **Firmware Built**
2. Go to this page, [http://voice.ai-thinker.com/#/](http://voice.ai-thinker.com/" \l "/)
3. First, make sure to register the account and log in using the account registered.
4. Then, select *“Create Product”*

A screenshot of a website

Description automatically generated

1. Select product category to demonstrate your output. In this case, can select *“Ceiling Lamp”* as the output for LED.

A screenshot of a phone

Description automatically generated

1. After that, select scene as *“Offline”* , because this project only support offline voice recognition.

A screenshot of a computer

Description automatically generated

1. Then, select module as *“VC-02”* since this is the model of the voice control module we will be using in this project.

A close-up of a computer screen

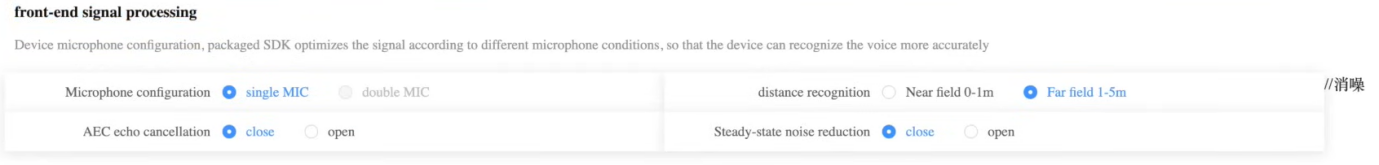
Description automatically generated

1. For the product information, you can name your product name to any name and the select language, you can either choose *“chinese"* or *“english"*. But for this project we select *“english"*  as our language*.*

A white background with black lines

Description automatically generated

1. Click save.
2. For front-end signal processing let it remain as default.



1. For pin configuration, set all pin as *“GPIO\_XX”* pin, for simple LED control. But, if we want to send the signal to another microcontroller for processing, we need to select “UART\_XX” where it will send the data in hex format.
2. In wake-up customization, we can set any wake-up word so that the module is ON and ready to receive the command. At the same time, it is recommended to set the wake-up sensitivity to the highest possible.

A screenshot of a computer

Description automatically generated

A close up of a button

Description automatically generated

1. For reply for wake-up, we can set any word as a reply to wake-up word we set before.

A screenshot of a chat

Description automatically generated

1. In customize offline command words and response words, the behavior section user can put any wording but avoid spacing. For turning ON and OFF LED, we need two behavior. While in command word section, it is the section where the module recognizes certain words and response to it, based on defined reply word. Multiple command can be defined at one behavior with ‘|’ separated.

A screenshot of a computer

Description automatically generated

1. After that got to control details tab then add control to the behavior we set previously. For behavior RedLightOn to turn ON LED , the control type is where we set the output pin, action is either we want the output in form of “pulse” or “setting” while parameter is where we set the high or low trigger. Set the same configuration to RedLightOff behavior except the parameter set to *“low level trigger”* to turn OFF the LED. For recognition sensitivity set it to high.

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated

1. Then, for wake-up free command words, we can select up to 10 command words that already defined in step 13 where it allow module to recognize these particular commands without being wake-up first.

A screenshot of a computer

Description automatically generated

1. In the boot broadcast, we can set any broadcast words for the module to say once it got connected to power supply. And for timeout, we can set the timer for how long it will stay before going to sleep mode, if it remain in idle state, and what its words before going to sleep.

A screenshot of a computer

Description automatically generated

1. Or we can directly give command for the module to go to sleep with quit command and with its reply.

A screenshot of a computer

Description automatically generated

1. Click *“make new version”* and then add any description to your project.
2. Click *“build SDK”* and then wait for 10-30 minutes for it to generated the Firmware. Refresh the page after 10-30 minutes and SDK status will show *‘built’* which mean your Firmware is ready to be downloaded. Click “download firmware” and also “download uni\_hb\_m\_solution”.
3. **Firmware Burn**
4. Extract both downloaded file.
5. Then go to **downloads**\uni\_hb\_m\_solution-162904-20240610.tar\uni\_hb\_m\_solution-162904-20240610\uni\_hb\_m\_solution\image\_demo\Hummingbird-M-Update-Tool, then execute the “*UniOneUpdateTool.exe”*
6. Make sure to connect your VC-02 module to PC’s via USB, then it able to detect your connected module. After that, load new “*uni\_app\_release\_update.bin”* from ‘output.tar.gz’ folder you downloaded.

A computer screen shot of a computer

Description automatically generated

1. Once loaded new “*uni\_app\_release\_update.bin”.* Press the reset button on VC-02 module and the module starts flashing the Firmware. Wait until it completed 100%.

A screenshot of a computer

Description automatically generated

1. Once it completed flashing, the connected port will change from yellow to green showing the Firmware is successfully flashed.
2. **C code for LED control on STM32 board**

This code mainly to demonstrate the LED control on STM32 board after command recognition from VC-02 module. GPIO pin, PC13 was used as an input that received output from VC-02 module after signal processing and then GPIO pin, PA5 was used as output to turn ON or OFF the LED on STM32 board. The demonstration video can be found at <https://www.youtube.com/watch?v=7jqv3_vvTLQ> in the minute of 3:27.

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

int main(void)

{

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* Configure the system clock \*/

SystemClock\_Config();

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

while (1)

{

if (HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_13) == GPIO\_PIN\_SET)

{

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_SET);

HAL\_Delay(2000);

}

else

{

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET);

HAL\_Delay(2000);

}

}

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE3);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PC13 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_13;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\*Configure GPIO pin : PA5 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_5;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

}

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

\_\_disable\_irq();

while (1)

{

}

}

1. **Audio Signal Acquisition and Processing using i2s INMP441 and CMSIS DSP.**

**Section 1: Instantiating CMSIS DSP**

1. To use the CMSIS DSP functions in STM32F4, go to software packs > select components.

A blue and white rectangle with red text

Description automatically generated

1. Select DSP library on the left panel, tick the selection box decollapsing all, and click selection, then press ok

A screenshot of a computer

Description automatically generated

1. Save the project to generate the C code first

A screenshot of a computer

Description automatically generated

1. Right click on the project and select properties > C/C++ General > Paths and Symbols

A screenshot of a computer

Description automatically generated

In the symbols tab, click on Add and insert symbol ARM\_MATH\_CM4 (because the board we are using is STM32F446RE. The F4 necessitates the CM to be set to 4.

A screenshot of a computer

Description automatically generated

1. In the main.c program, insert
   1. #include “arm\_math.h”

**Section 2: IOC Hardware configuration**

1. Open the .ioc file, go to pinout & configuration -> Multimedia -> I2S2

A screenshot of a computer

Description automatically generated

1. In the mode, select mode as: **Half-Duplex Master**
2. In the Parameters settings, select these options:
   1. Transmission Mode: Mode Master Receive
   2. Communication Standard: i2s Phillips
   3. Data and Frame Format: 24 bits data on 32 bits frame
   4. Selected Audio Frequency: 48kHz
3. In DMA settings, select Mode as Circular and the Data Width as Byte.

A screenshot of a computer

Description automatically generated

1. Go to Clock Configuration and check if I2S\_APB1\_CLK is set as 48MHz as define in step 3).

A computer screen shot of a computer

Description automatically generated

1. Go to <http://t-filter.engineerjs.com/> to design obtain the coefficient for the FIR filter.

A screen shot of a graph

Description automatically generated

Section 3: Code

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2024 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

**#define** **ARM\_MATH\_CM4**

**#include** "main.h"

**#include** "arm\_math.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

/\* 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 ---------------------------------------------------------\*/

**I2S\_HandleTypeDef** hi2s2;

**DMA\_HandleTypeDef** hdma\_spi2\_rx;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_DMA\_Init**(**void**);

**static** **void** **MX\_I2S2\_Init**(**void**);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

**#define** **FILTER\_TAP\_NUM** 17 // Tap number

**#define** **BLOCK\_SIZE\_FLOAT** 512 //

**#define** **BLOCK\_SIZE\_U16** 2048 //

**static** **float** filter\_taps[FILTER\_TAP\_NUM] = {

-0.0021834891907904987,

0.023133081888390004,

0.03440125360693663,

0.054016706019288735,

0.07610902012650608,

0.09772535709704201,

0.11593264129629442,

0.12810228628568973,

0.13238343618749146,

0.12810228628568973,

0.11593264129629442,

0.09772535709704201,

0.07610902012650608,

0.054016706019288735,

0.03440125360693663,

0.023133081888390004,

-0.0021834891907904987

}; //filter coefficient

**arm\_fir\_instance\_f32** firsettings;

**float** fir\_state [BLOCK\_SIZE\_FLOAT + FILTER\_TAP\_NUM - 1];

**uint16\_t** rxBuf[BLOCK\_SIZE\_U16\*2]; //input buffer

**uint16\_t** txBuf[BLOCK\_SIZE\_U16\*2]; //output buffer

**float** buf\_in [BLOCK\_SIZE\_FLOAT\*2];

**float** buf\_out [BLOCK\_SIZE\_FLOAT\*2];

**uint8\_t** callback\_state = 0;

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

**HAL\_I2S\_Receive\_DMA**(&hi2s2, rxBuf, BLOCK\_SIZE\_U16); //HAL function to enable DMA for I2S receiver

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

**HAL\_Init**();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

**SystemClock\_Config**();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

**MX\_GPIO\_Init**();

**MX\_DMA\_Init**();

**MX\_I2S2\_Init**();

**arm\_fir\_init\_f32**(&firsettings, FILTER\_TAP\_NUM, &filter\_taps[0], &fir\_state[0], BLOCK\_SIZE\_FLOAT); //arm cmsis fir function

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

**int** offset\_r\_ptr; //read pointer

**int** offset\_w\_ptr, w\_ptr; //write pointer

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* USER CODE END WHILE \*/

**if** (callback\_state != 0) {

//decide if it was half or cplt callback

**if** (callback\_state == 1) {

offset\_r\_ptr = 0; //

offset\_w\_ptr = 0;

w\_ptr = 0;

}

**else** **if** (callback\_state == 2) {

offset\_r\_ptr = BLOCK\_SIZE\_U16;

offset\_w\_ptr = BLOCK\_SIZE\_FLOAT;

w\_ptr = BLOCK\_SIZE\_FLOAT;

}

//restore input sample buffer to float array

**for** (**int** i=offset\_r\_ptr; i<offset\_r\_ptr+BLOCK\_SIZE\_U16; i=i+4) {

buf\_in[w\_ptr] = (**float**) ((**int**) (rxBuf[i]<<16)|rxBuf[i+1]);

w\_ptr++;

}

//process FIR

**arm\_fir\_f32** (&firsettings, &buf\_in[offset\_w\_ptr], &buf\_out[offset\_w\_ptr],BLOCK\_SIZE\_FLOAT);

w\_ptr = offset\_w\_ptr;

**for** (**int** i=offset\_r\_ptr; i<offset\_r\_ptr+BLOCK\_SIZE\_U16; i=i+4) {

txBuf[i] = (((**int**)buf\_out[w\_ptr])>>16)&0xFFFF;

txBuf[i+1] = ((**int**)buf\_out[w\_ptr])&0xFFFF;

w\_ptr++;

}

callback\_state = 0;

//restore processed float-array to output sample-buffer

callback\_state = 0;

}

}

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

**void** **HAL\_I2S\_RxHalfCpltCallback**(**I2S\_HandleTypeDef** \*hi2s){

callback\_state = 1; //half complete callback

}

**void** **HAL\_I2S\_RxCpltCallback**(**I2S\_HandleTypeDef** \*hi2s){

callback\_state = 2; //complete callback

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

**RCC\_OscInitTypeDef** RCC\_OscInitStruct = {0};

**RCC\_ClkInitTypeDef** RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE3);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 16;

RCC\_OscInitStruct.PLL.PLLN = 336;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV4;

RCC\_OscInitStruct.PLL.PLLQ = 2;

RCC\_OscInitStruct.PLL.PLLR = 2;

**if** (**HAL\_RCC\_OscConfig**(&RCC\_OscInitStruct) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\*\* Initializes the CPU, AHB and APB buses 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\_2) != *HAL\_OK*)

{

**Error\_Handler**();

}

}

/\*\*

\* @brief I2S2 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_I2S2\_Init**(**void**)

{

/\* USER CODE BEGIN I2S2\_Init 0 \*/

/\* USER CODE END I2S2\_Init 0 \*/

/\* USER CODE BEGIN I2S2\_Init 1 \*/

/\* USER CODE END I2S2\_Init 1 \*/

hi2s2.Instance = SPI2;

hi2s2.Init.Mode = I2S\_MODE\_MASTER\_RX;

hi2s2.Init.Standard = I2S\_STANDARD\_PHILIPS;

hi2s2.Init.DataFormat = I2S\_DATAFORMAT\_24B;

hi2s2.Init.MCLKOutput = I2S\_MCLKOUTPUT\_DISABLE;

hi2s2.Init.AudioFreq = I2S\_AUDIOFREQ\_48K;

hi2s2.Init.CPOL = I2S\_CPOL\_LOW;

hi2s2.Init.ClockSource = I2S\_CLOCK\_PLL;

hi2s2.Init.FullDuplexMode = I2S\_FULLDUPLEXMODE\_DISABLE;

**if** (**HAL\_I2S\_Init**(&hi2s2) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN I2S2\_Init 2 \*/

/\* USER CODE END I2S2\_Init 2 \*/

}

/\*\*

\* Enable DMA controller clock

\*/

**static** **void** **MX\_DMA\_Init**(**void**)

{

/\* DMA controller clock enable \*/

\_\_HAL\_RCC\_DMA1\_CLK\_ENABLE();

/\* DMA interrupt init \*/

/\* DMA1\_Stream3\_IRQn interrupt configuration \*/

**HAL\_NVIC\_SetPriority**(*DMA1\_Stream3\_IRQn*, 0, 0);

**HAL\_NVIC\_EnableIRQ**(*DMA1\_Stream3\_IRQn*);

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

**void** **Error\_Handler**(**void**)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

**\_\_disable\_irq**();

**while** (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

**#ifdef** USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

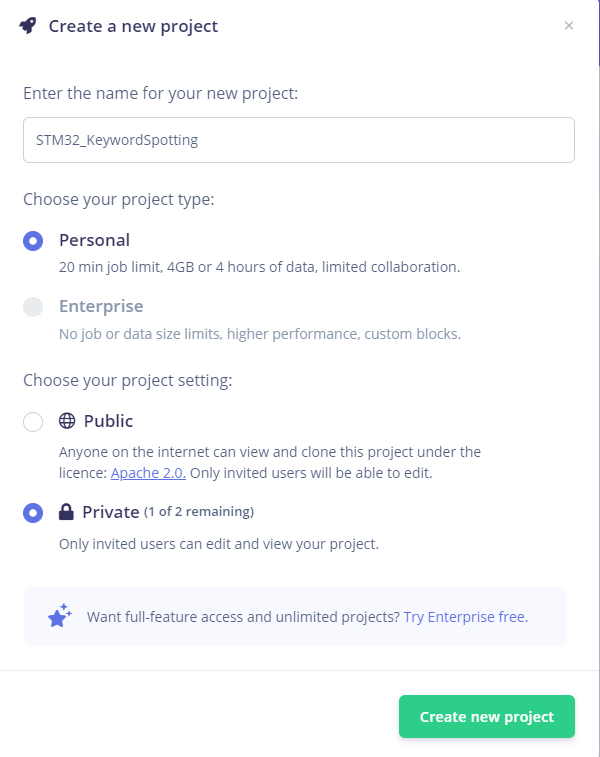
**#endif** /\* USE\_FULL\_ASSERT \*/

1. **AI Model for Keyword Spotting**

For generating the AI model for keyword spotting, Edge Impulse is chosen as the platform given that its compatibility with STM32 model and easiness. Edge Impulse allows the export of the Impulse (or AI model) as Cube.MX CMSIS-PACK and generation of source code as C++ for deployment on STM32.

**FIRMWARE IMPLEMENTATION**

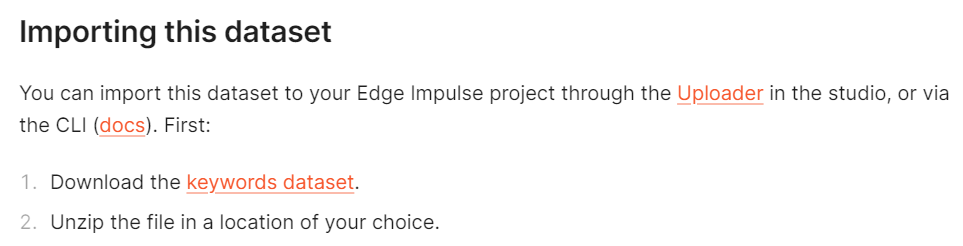
1. Register an account in edgeimpulse.com. Create new project and fill in the details.



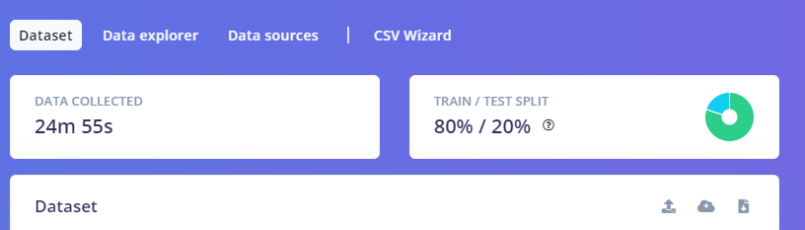
1. Obtain dataset from

<https://docs.edgeimpulse.com/docs/pre-built-datasets/keyword-spotting>

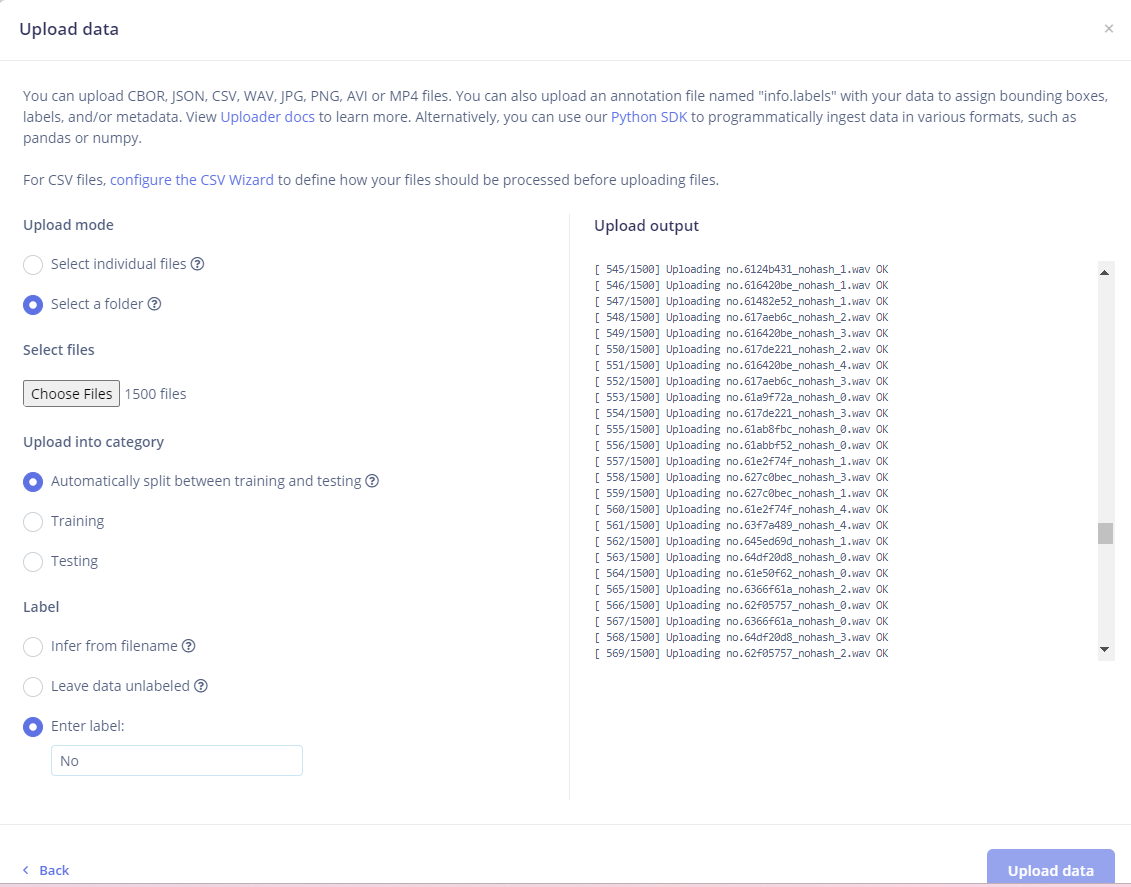
Download and unzip the keywords dataset.



In the project, navigate to dataset page and click on the upload button.

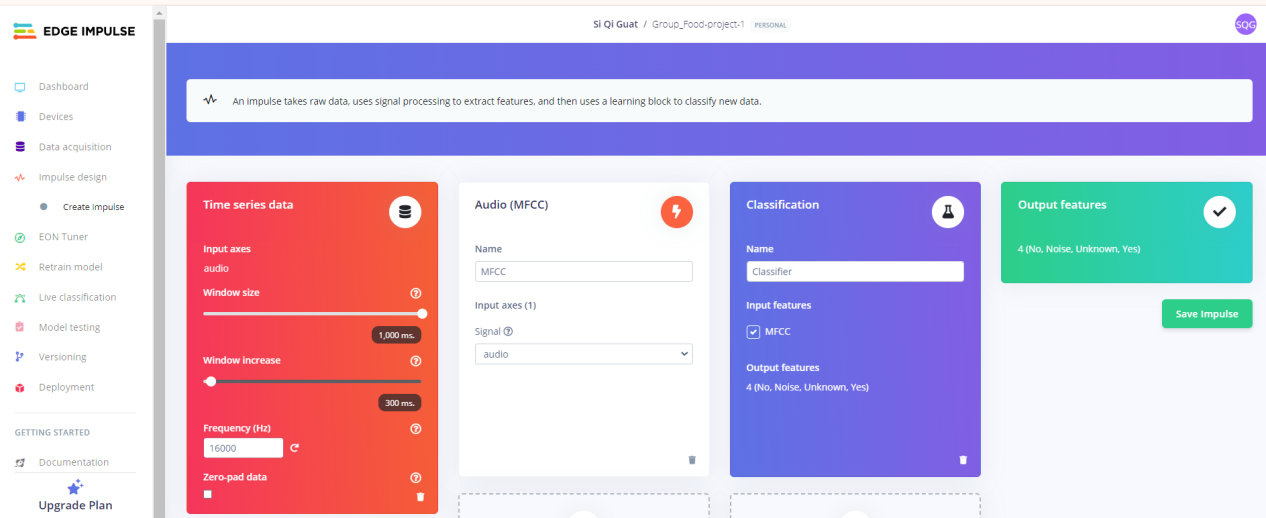


1. Set up the options as shown below. Upload each class folder by folder and fill in the label. Upload all 4 classes: Noise, Unknown, Yes, and No accordingly.

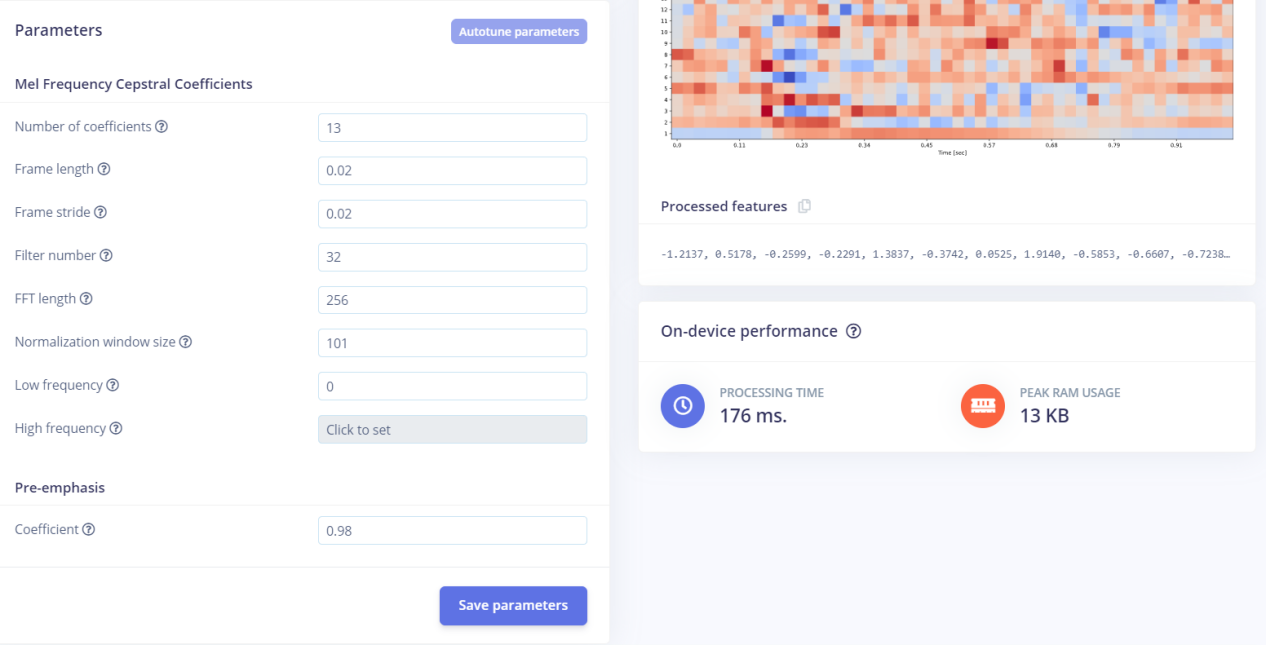
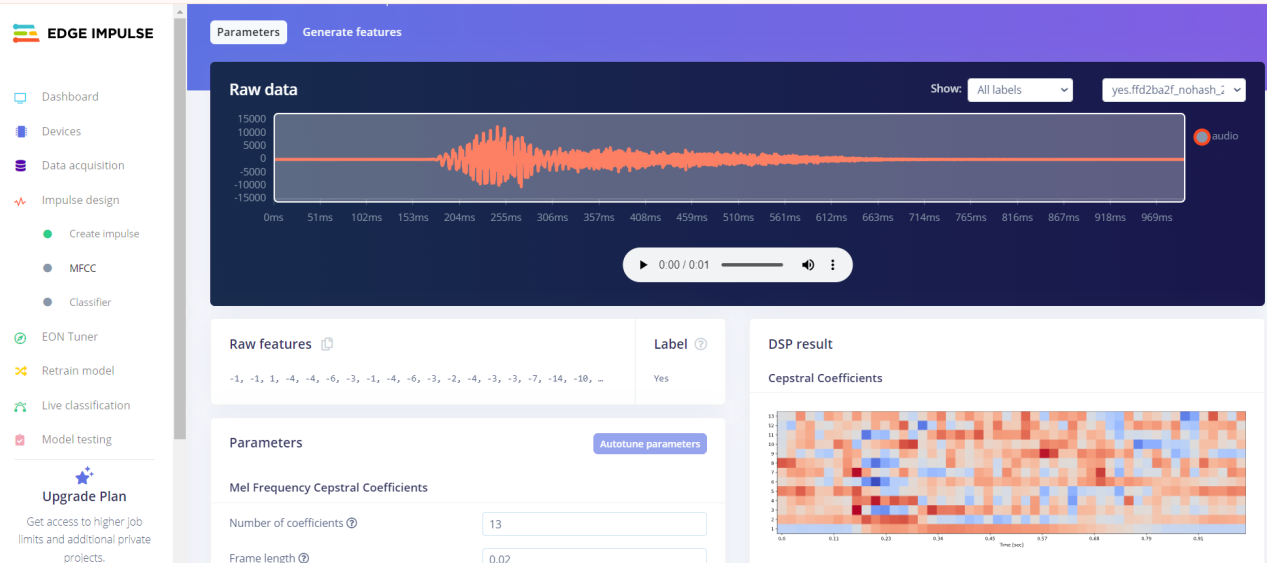


1. Navigate to ‘Create impulse’ page and fill in the details accordingly.

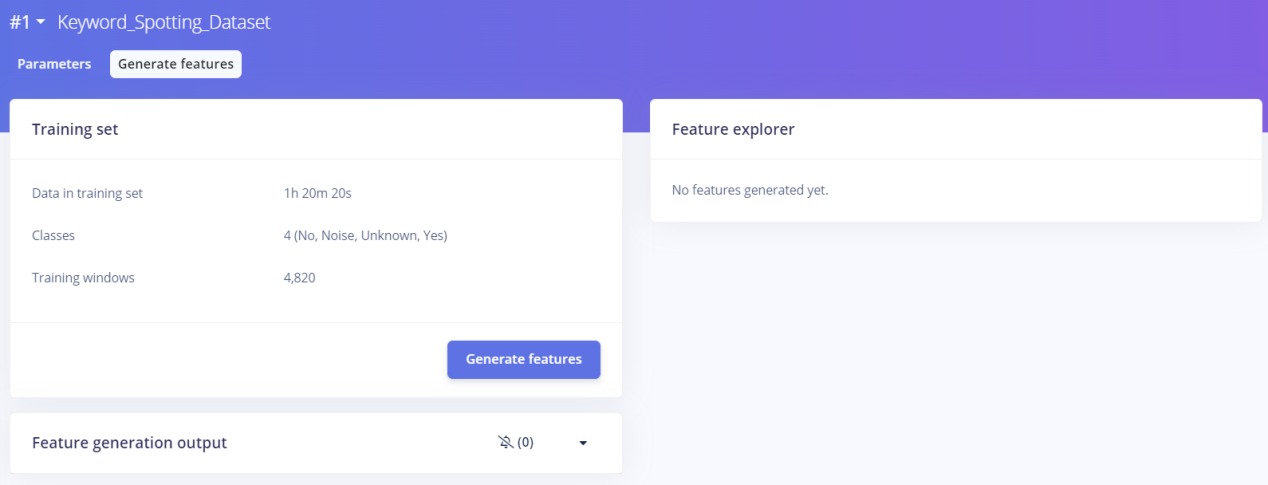




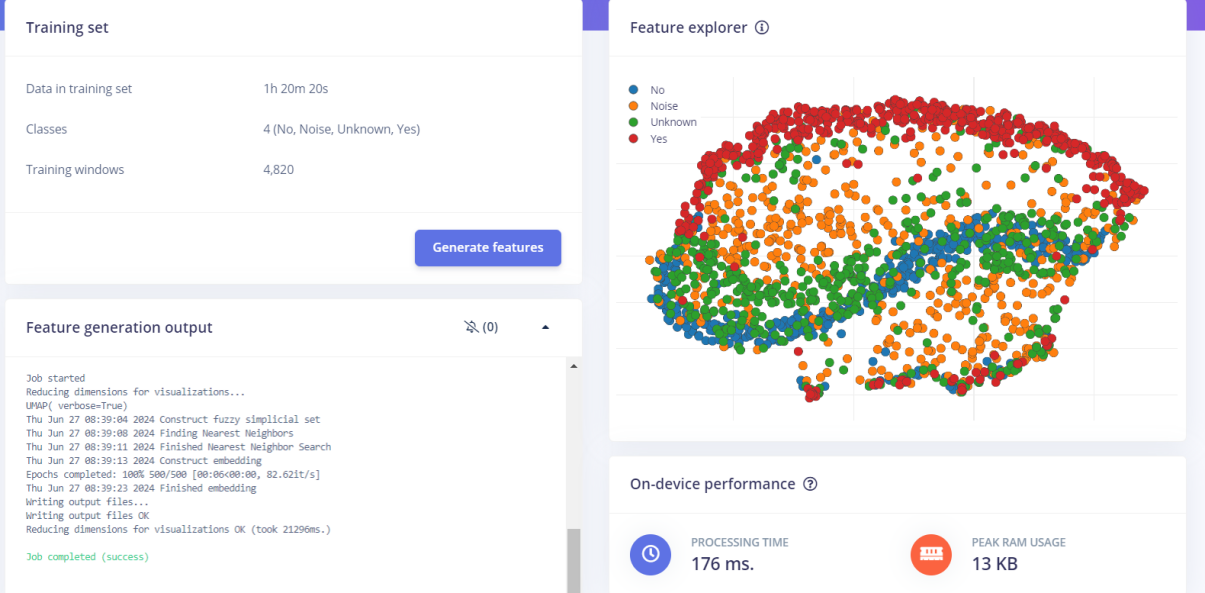
1. Navigate to MFCC tab and set up the parameters as shown below.



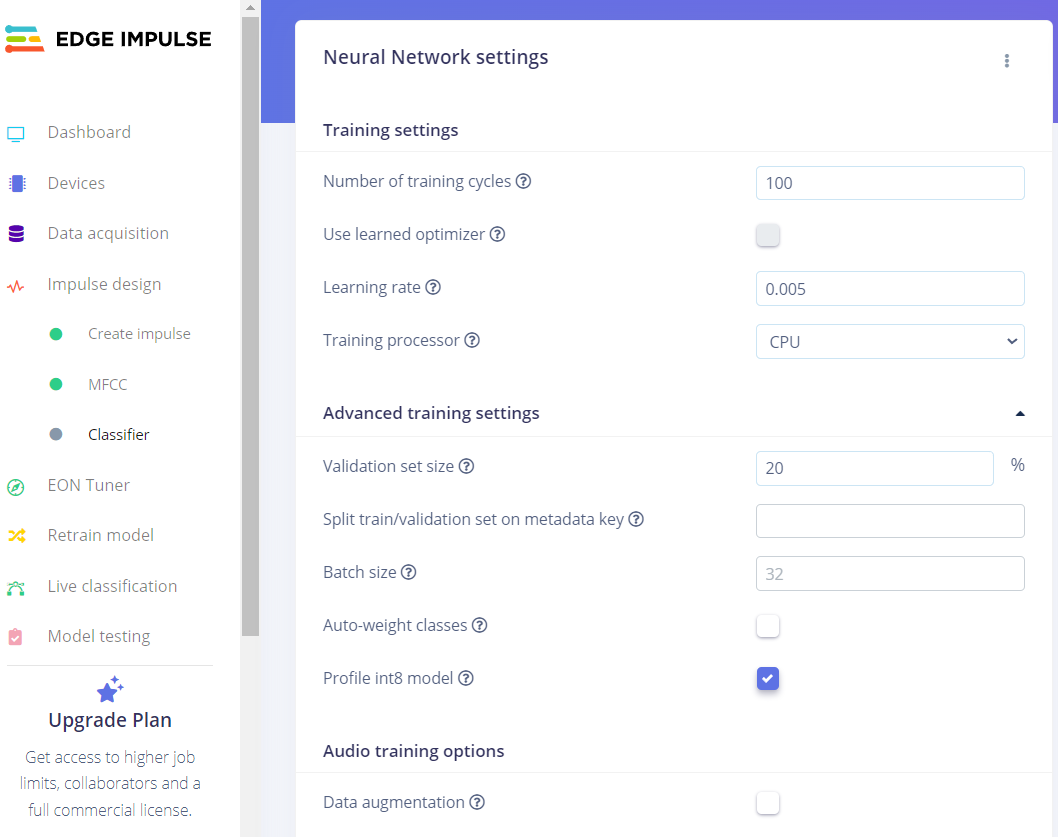
1. Navigate to Generate features tab. Generate MFE block by clicking on ‘Generate features’.



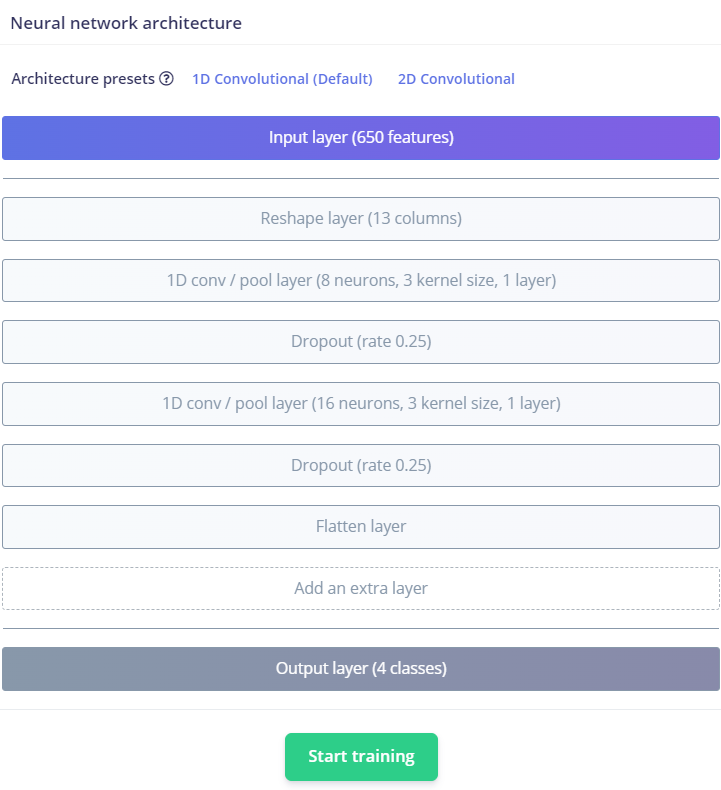
Results is as shown below.

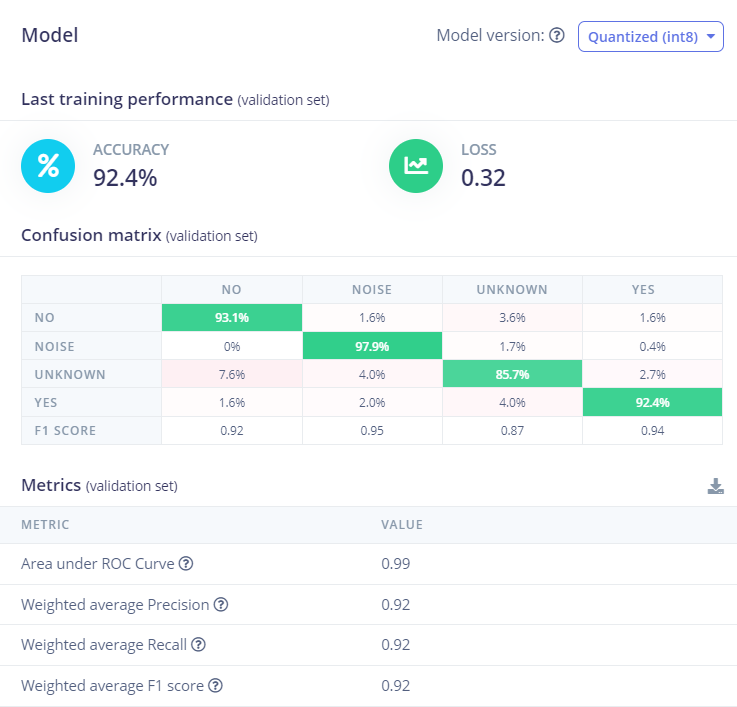


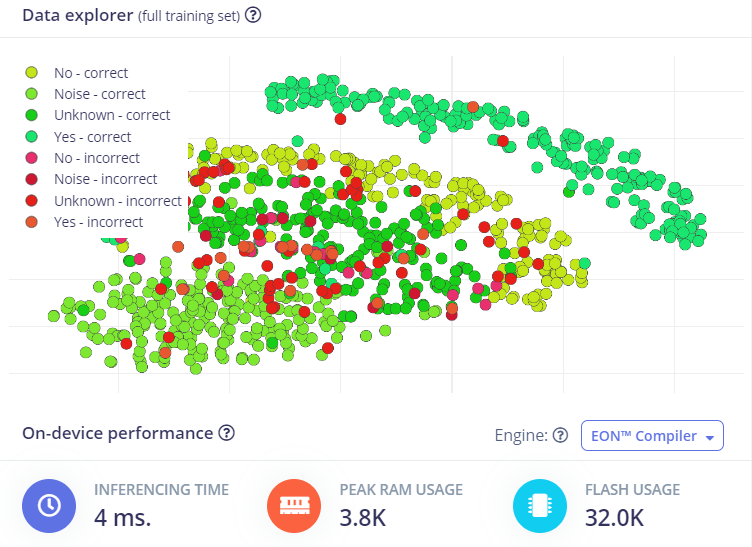
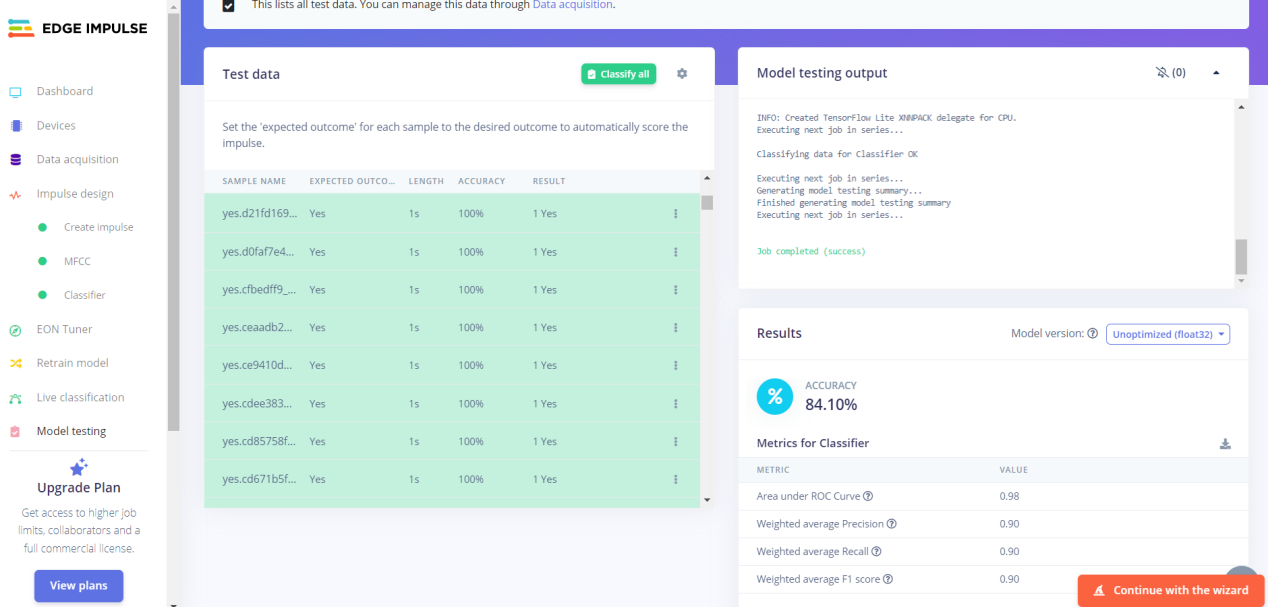
1. Navigate to classifier page. Set up the neural network settings as shown below.

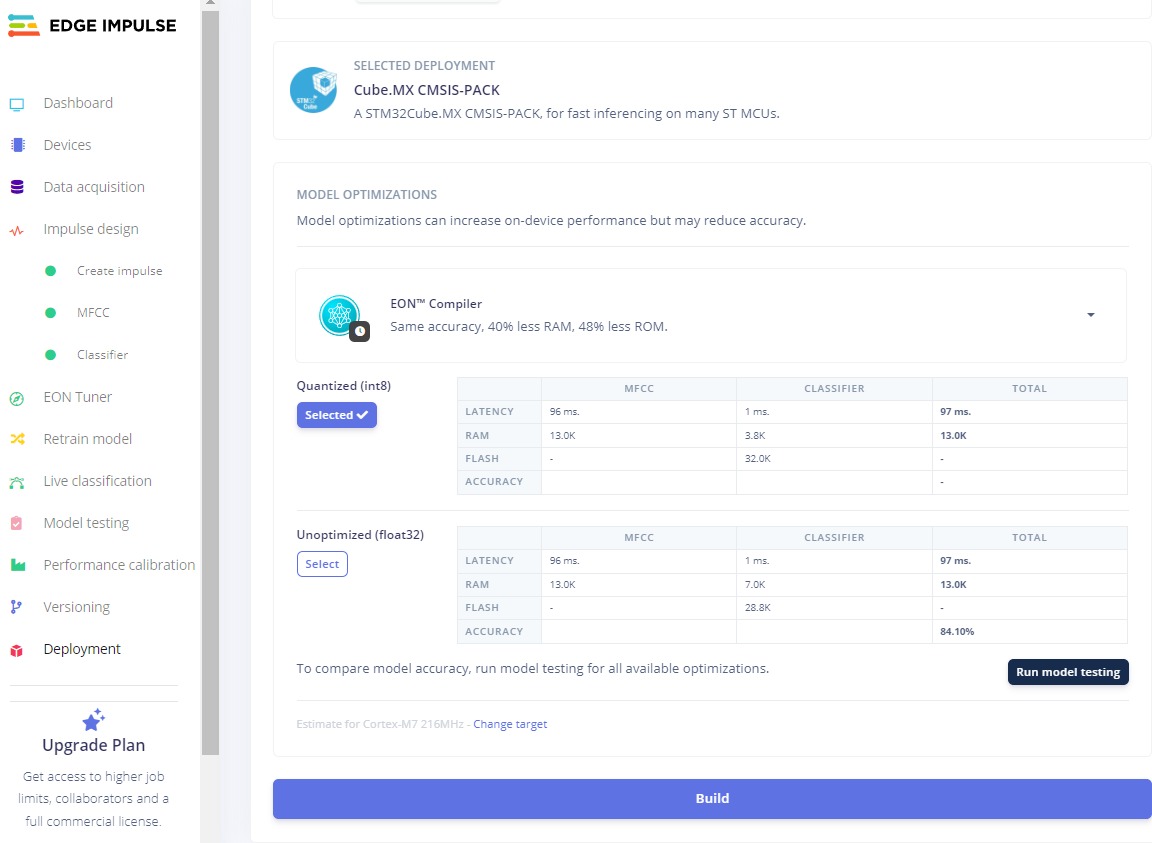
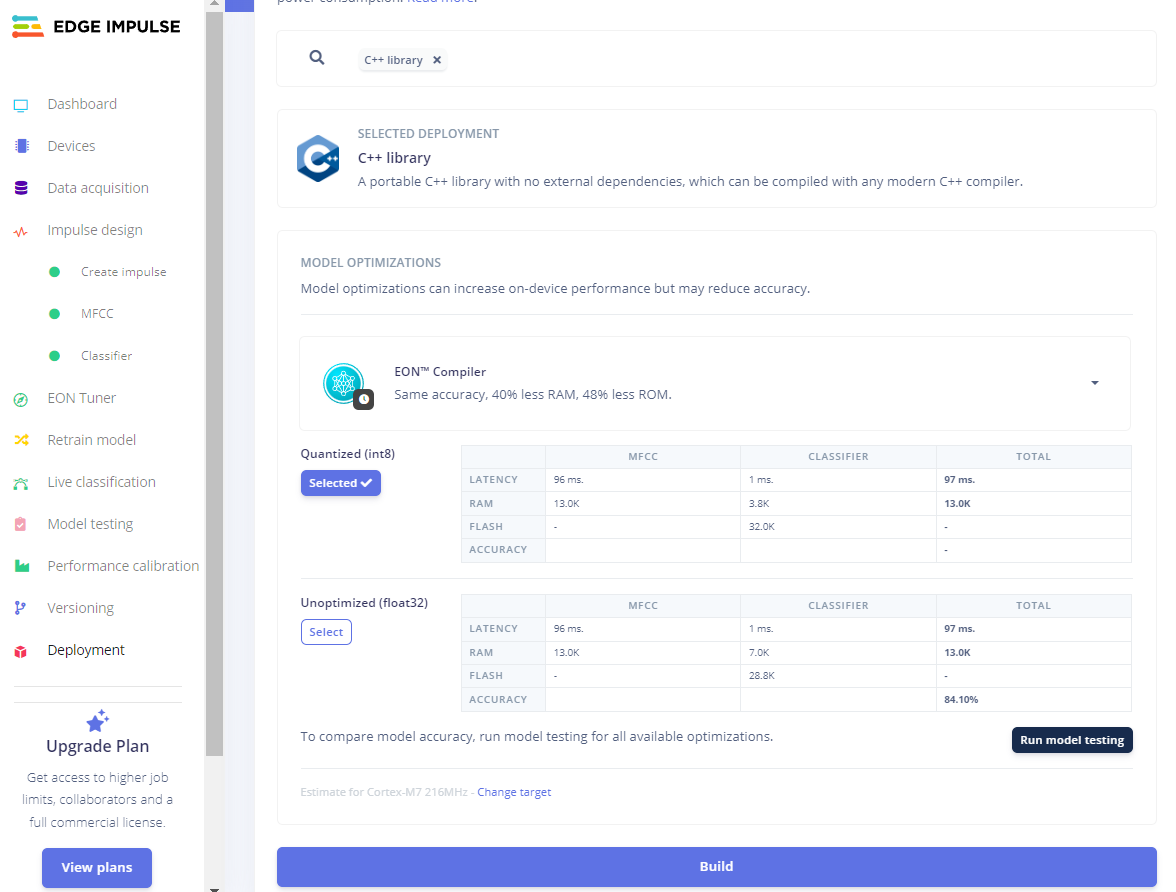


1. The neural network architecture is as shown below. Click on ‘Start Training’.

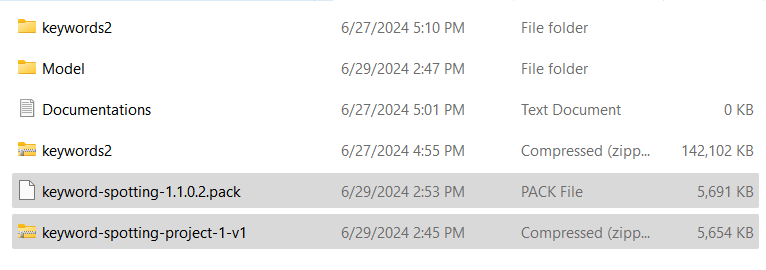


1. The training results of the model is as shown below. 

10. Navigate to Model Testing page, click on ‘Classify all’. 

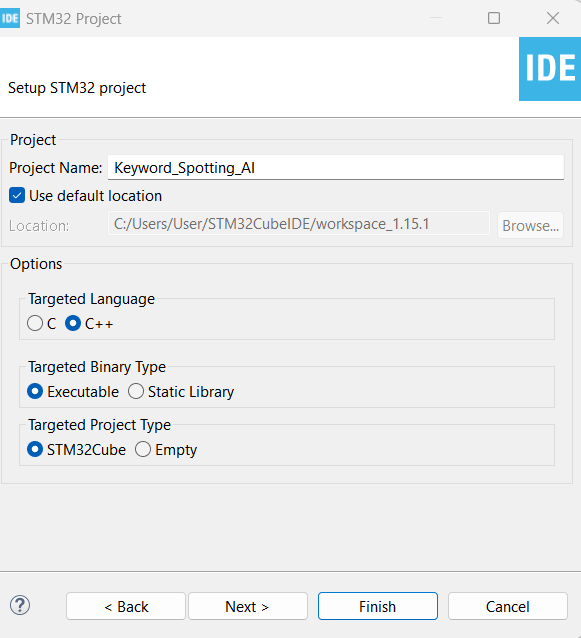
11 Navigate to Deployment page. Export the code as C++ and Cube MX CMSIS-PACK for integration. 

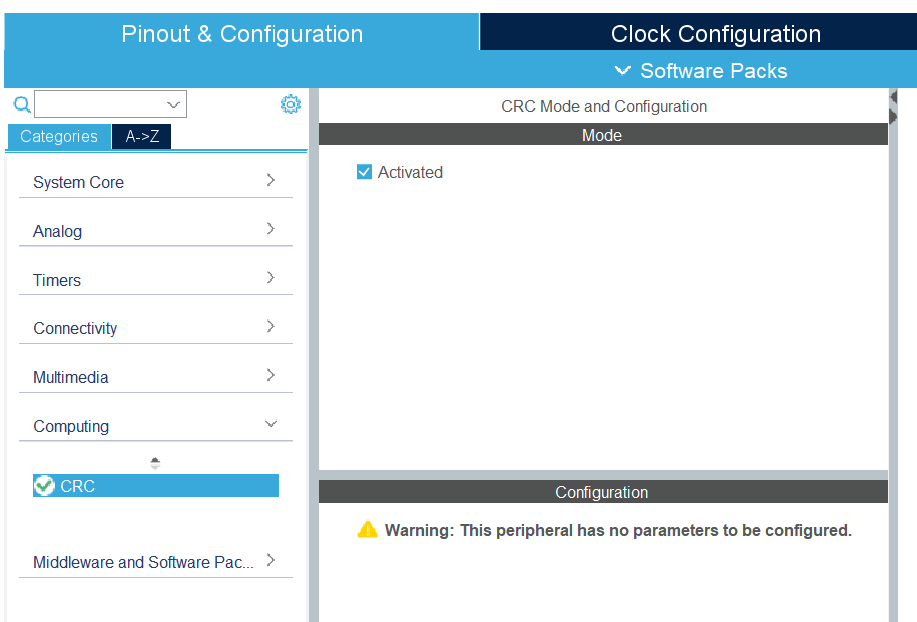
The generated code files is as shown below.



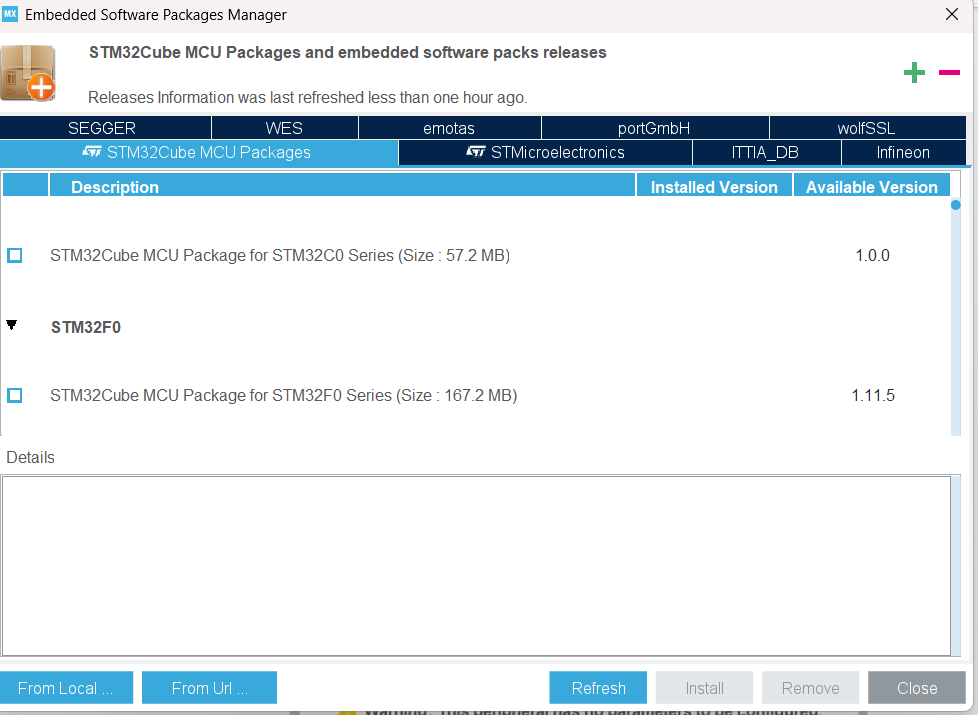
**HARDWARE IMPLEMENTATION**

1. Create STM32 project with C++ targeted language, as shown below.

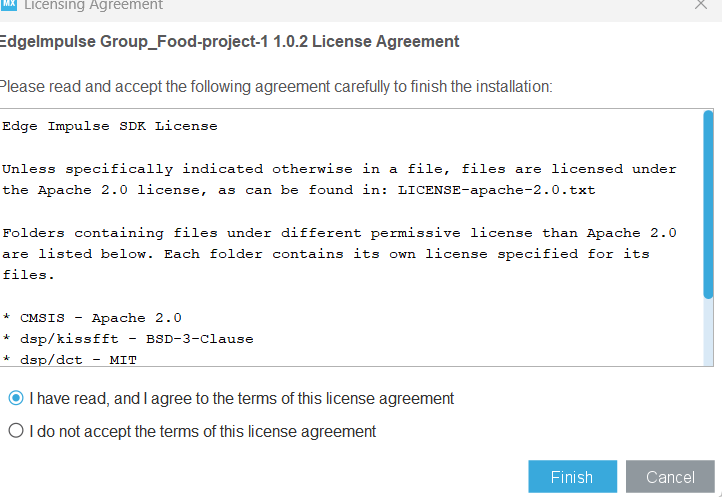




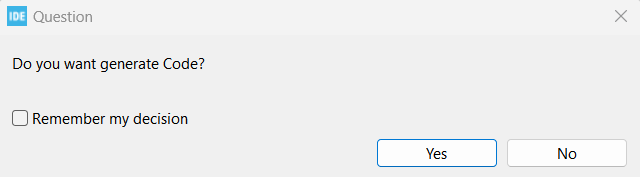
1. Help > Manage Embedded Software Packages Manager > ‘From Local’



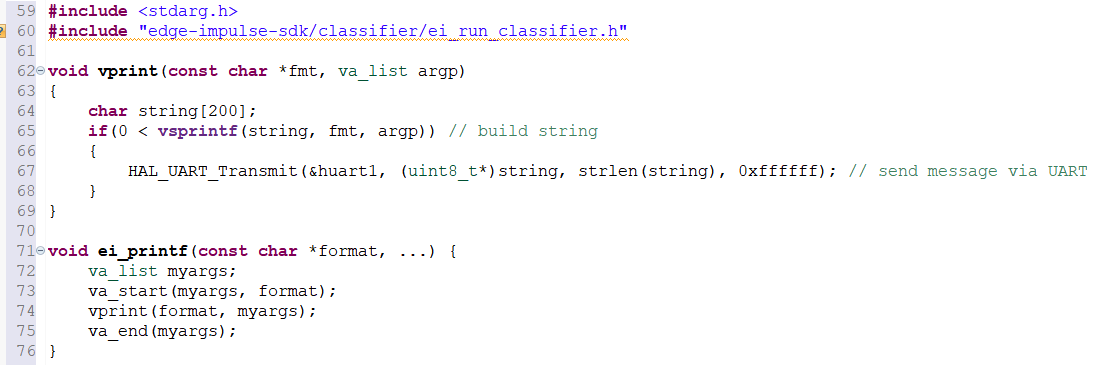
Choose ‘I have read, and I agree to the terms of this license agreement’



1. File > Save all > Yes to generate the code for the model.



1. To log debug information the CMSIS-PACK uses the (weak defined) ei\_printf function. This is done by overriding this function in your main.cpp (if you only have a main.c rename it) to log to your UART port. Under USER CODE BEGIN 0 add:



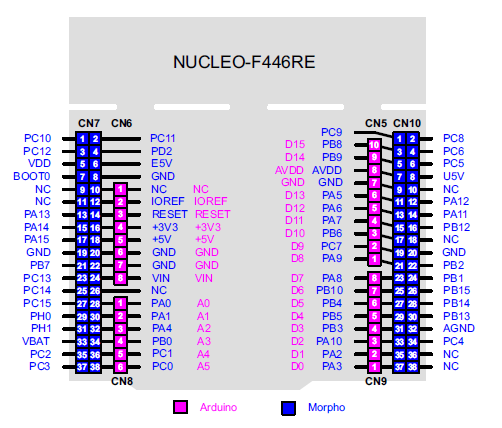
Remaining procedures for hardware implementation to be followed (Postponed for later integration):

[Cube.MX CMSIS-PACK | Edge Impulse Documentation](https://docs.edgeimpulse.com/docs/run-inference/using-cubeai)

**4 External LED Control**

In NUCLEO-F446RE board, the available LEDs are limited. To extend more LEDs, external LEDs can be used by connecting to GPIO\_Output pins of the board.

**Section 1: IOC Hardware Configuration**



Pins and STM32 Naming of NUCLEO-F446RE

1. To use D4 to D7 pins, the pins need to be set as GPIO\_Output. The STM32 naming of D4 to D7 pins are equivalent to PB5, PB4, PB10 and PA8 respectively.

Reference: <https://www.youtube.com/watch?v=TCDB39hDvRA>

A computer screen shot of a computer chip

Description automatically generated

1. Configure PB5, PB4, PB10 and PA8 as GPIO\_Output.

A screenshot of a computer

Description automatically generated

1. PB5, PB4, PB10 and PA8 are labelled as LD5, LD6, LD7, and LD8.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

1. The pins are initialized based on configuration during startup.

A screenshot of a computer

Description automatically generated

1. The exact pins are defined in header file.

A screenshot of a computer code

Description automatically generated

1. To switch ON/OFF the external LEDs, HAL\_GPIO\_WritePin() can be used to set HIGH/LOW in the main() function.

List of website references:

1. <https://docs.edgeimpulse.com/docs/pre-built-datasets/keyword-spotting>
2. [Cube.MX CMSIS-PACK | Edge Impulse Documentation](https://docs.edgeimpulse.com/docs/run-inference/using-cubeai)
3. <https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/audio-classification>
4. <https://www.youtube.com/watch?v=TCDB39hDvRA>
5. [http://voice.ai-thinker.com/#/](http://voice.ai-thinker.com/" \l "/)