Cyberon DSpotter Sample Code for Renesas RA MCU

Version: 1.2
Date of issue: 2021/2/5

Cyberon Corporation

Software solution provider for embedded system

http://www.cyberon.com.tw/

Contents

History：

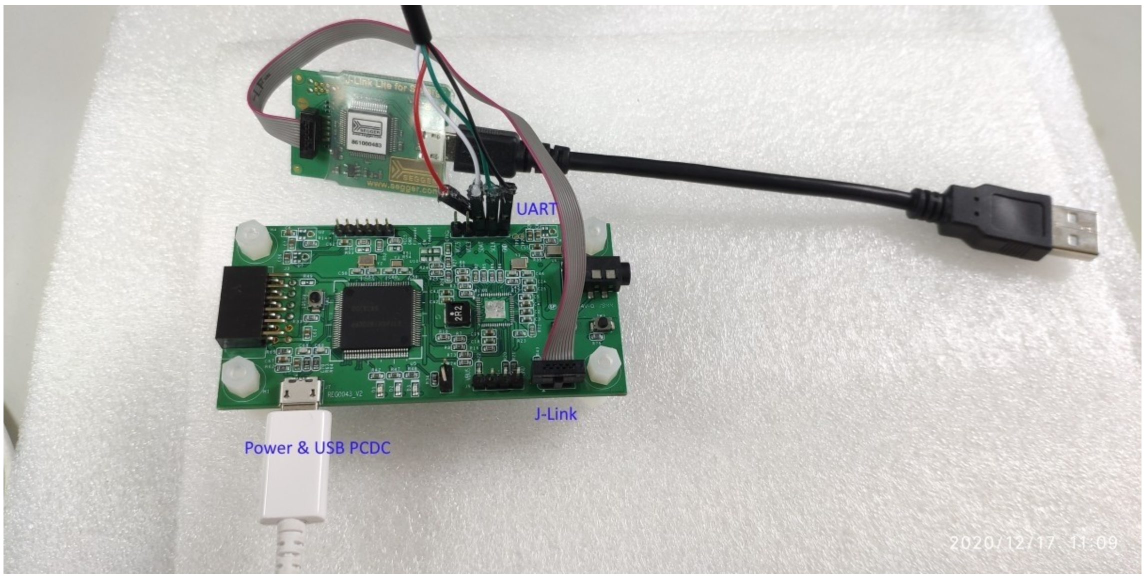
Ver.	Data	Update
1.0	2021-01-02	First release.
1.1	2021-01-22	Add RA6M1 Voice Recognition demo board V2.
1.2	2021-02-05	Add RA4M2 VC2. EK-RA6M4: Change UART9 to UART0, so we change the UART pin connection. Add the description of voice record button.

1. Develop Environment

1.1. Hardware Demo Board

Cyberon DSpotter is porting to Renesas RA MCU(Cortex M4 or M33). The DSpotter sample code use UART to log text and dump audio data. The setting of UART is 460800 bps, 8 bits data, 1 stop bit, no parity, no flow control. Please use terminal program(Ex. Putty, ExtraPutty, TeraTerm…) to show the text log. Please use CybSerialRecorder.exe to receive audio record data and save to wave file. This sample code is verified on the following RA demo board.

1. RA6M1 Voice Recognition demo board V1:

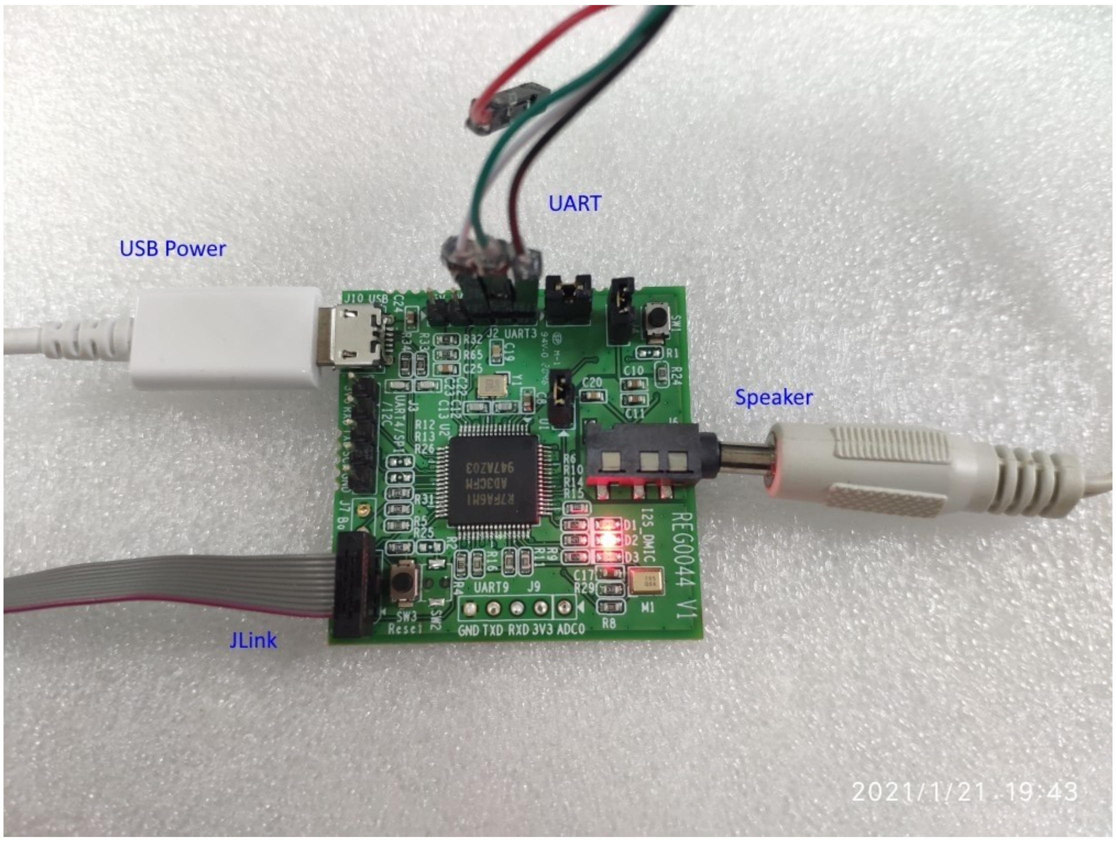


There is a I2S microphone(U6) on the demo board. We can find the microphone input hole on the background of demo board. We use four hexagonal prism to float the demo board for better recognition performance.
Please connect UART Tx to J5 SCLK, Rx to J5 SDAT, GND.
The voice tag record button is SW2.

The sample project: /SampleCode/ra6m1_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

PS. The XTAL frequency of this demo board is 24MHz. The XTAL frequency of EK-RA6M1 is 12MHz. To run EK-RA6M1's sample code on this demo board, please modify the clock page of configuration.xml.

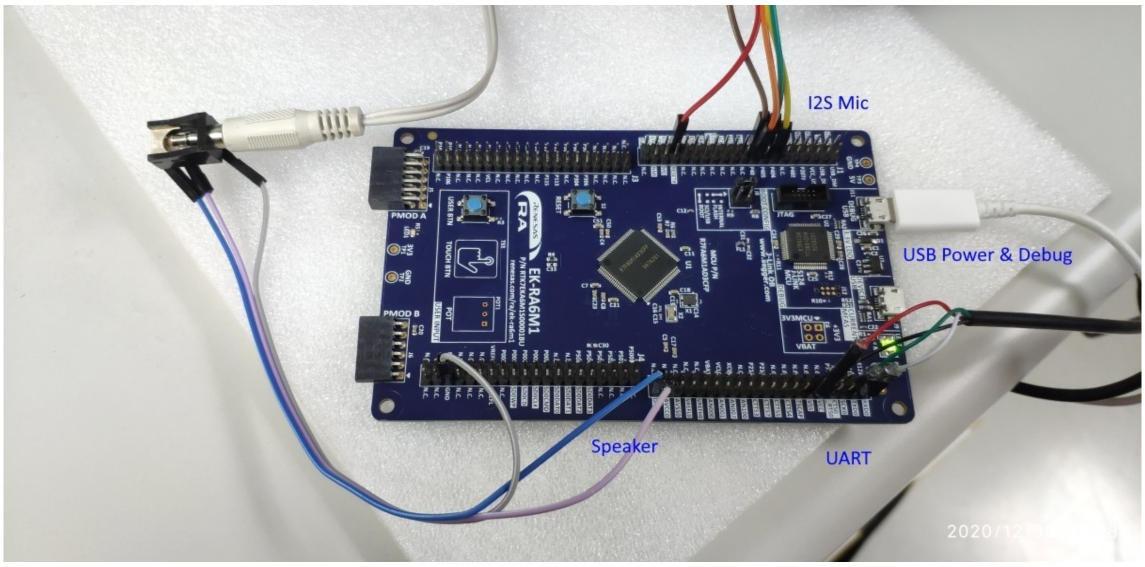
2. RA6M1 Voice Recognition demo board V2:



There is no I2S interface on 64 pin RA6M1, so the I2S microphone need to connect through SPI interface. We can find the microphone input hole on the background of demo board. Please flip the demo board when test voice command for better recognition performance.
Please connect UART to J2 UART3.
The voice tag record button is SW1.

The sample project: /SampleCode/ra6m1_vc2_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

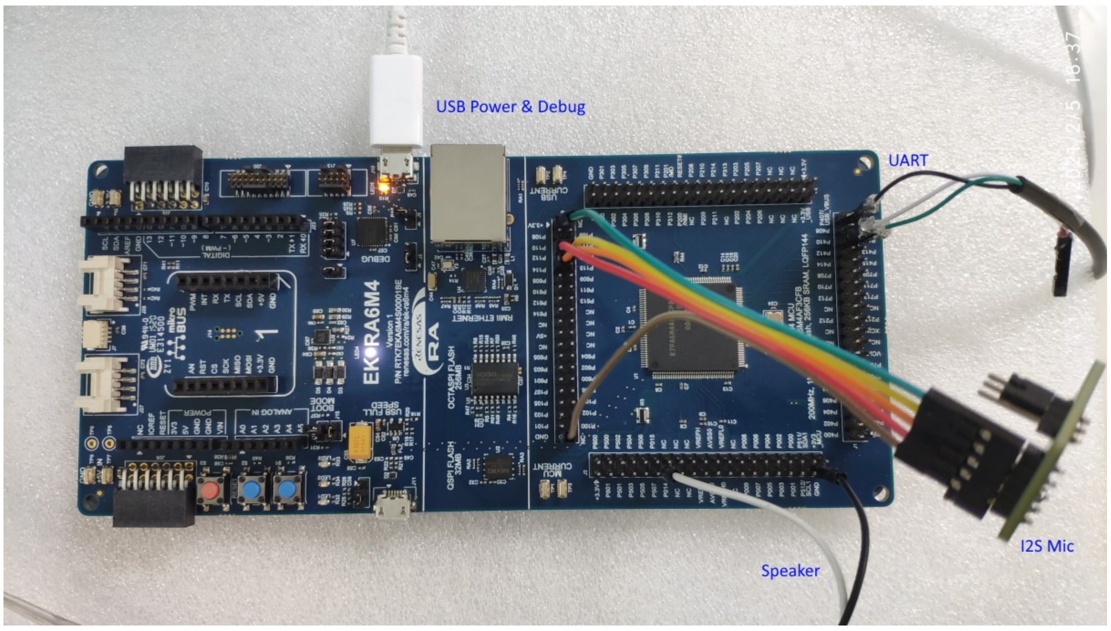
3. EK-RA6M1:



Please connect UART Tx to P101(TXD0), Rx to P100(RXD0), GND.
Please connect I2S microphone module to J1:
3.3V, LRC to P404, DATA to P406, BCLK to P403(GTIOC3A), GND.
Please connect speaker to P014(DA0) and GND.
The voice tag record button is USER BTN.

The sample project: /SampleCode/ra6m1_ek_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

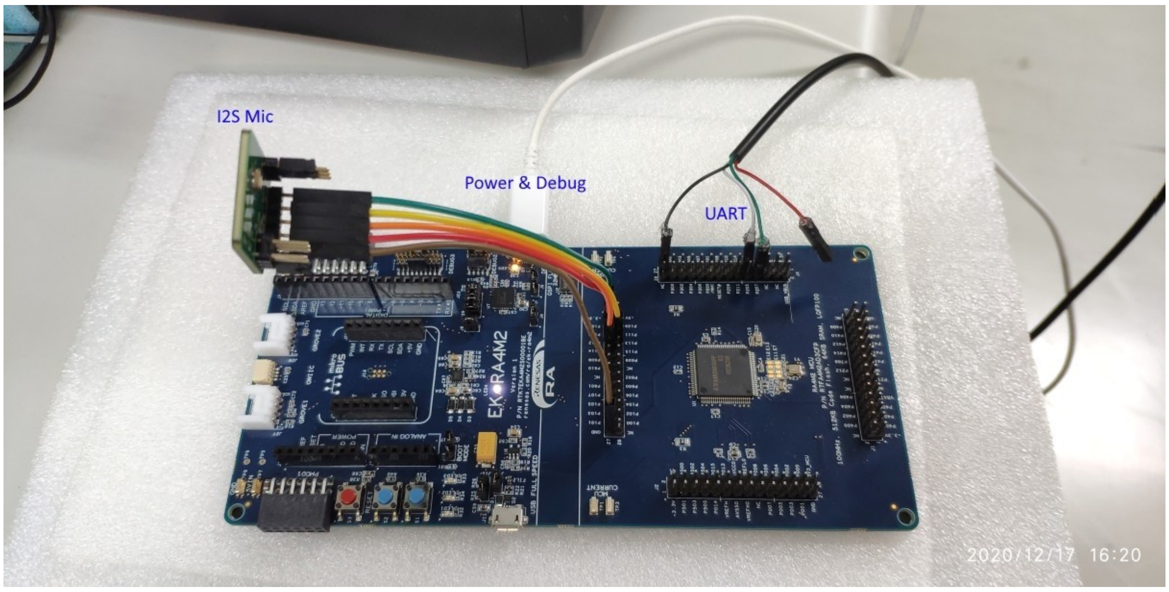
4. EK-RA6M4 (CM33):



Please connect UART Tx to J1 P411, Rx to J1 P410, GND.
Please connect I2S microphone module to J3:
  3.3V, LRC to P113, DATA to P114, BCLK to P112, GND.
Please connect speaker to P014 and GND.
The voice tag record button is S2.

The sample project: /SampleCode/ra6m4_ek_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

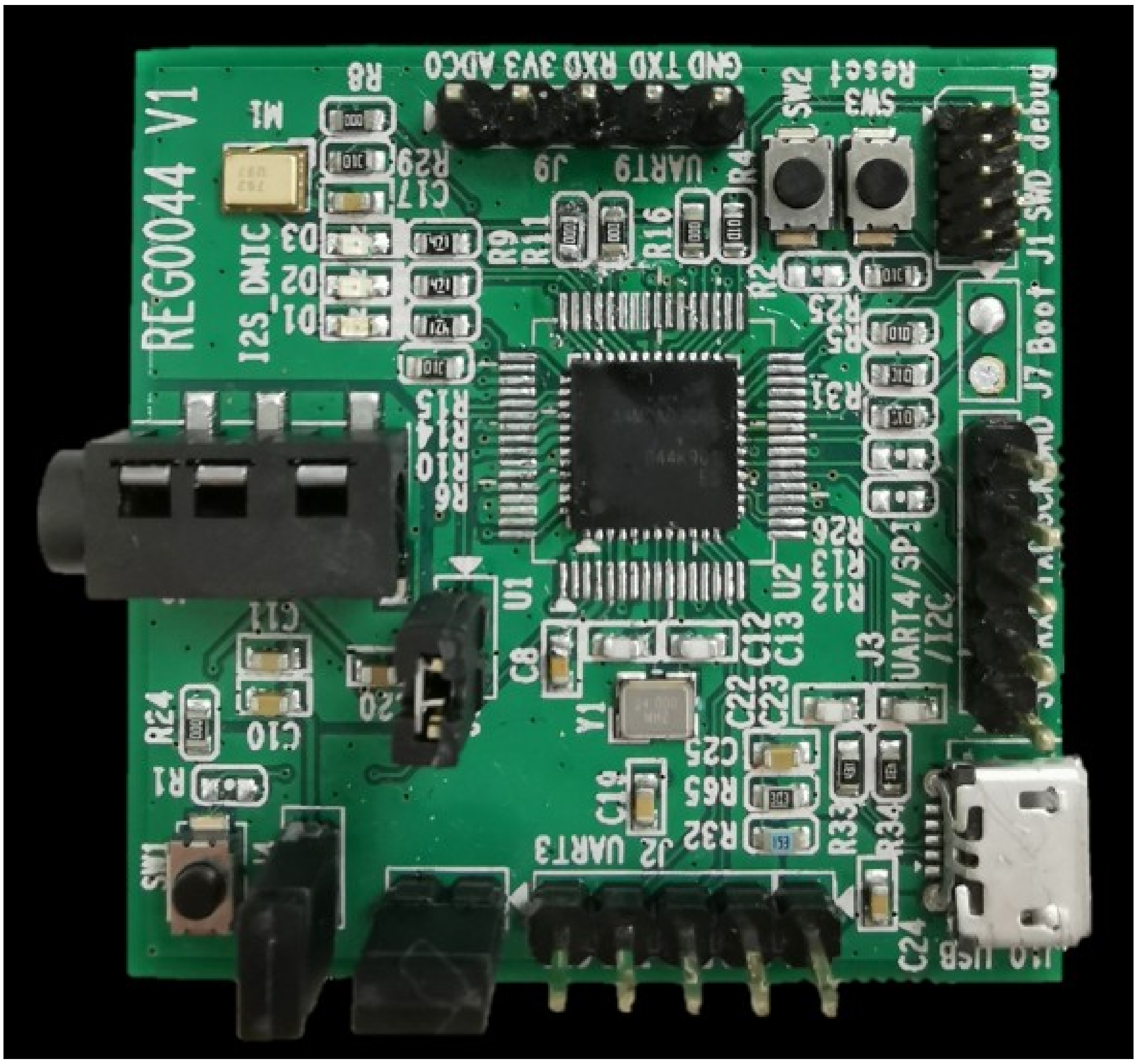
5. EK-RA4M2 (CM33):



Please connect UART Tx to J4 P207, Rx to J4 P206, GND.
Please connect I2S microphone module to J3:
    3.3V, LRC to P113, DATA to P114, BCLK to P112, GND.
Please connect speaker to P014 and GND.
The voice tag record button is S2.

The sample project: /SampleCode/ra4m2_ek_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

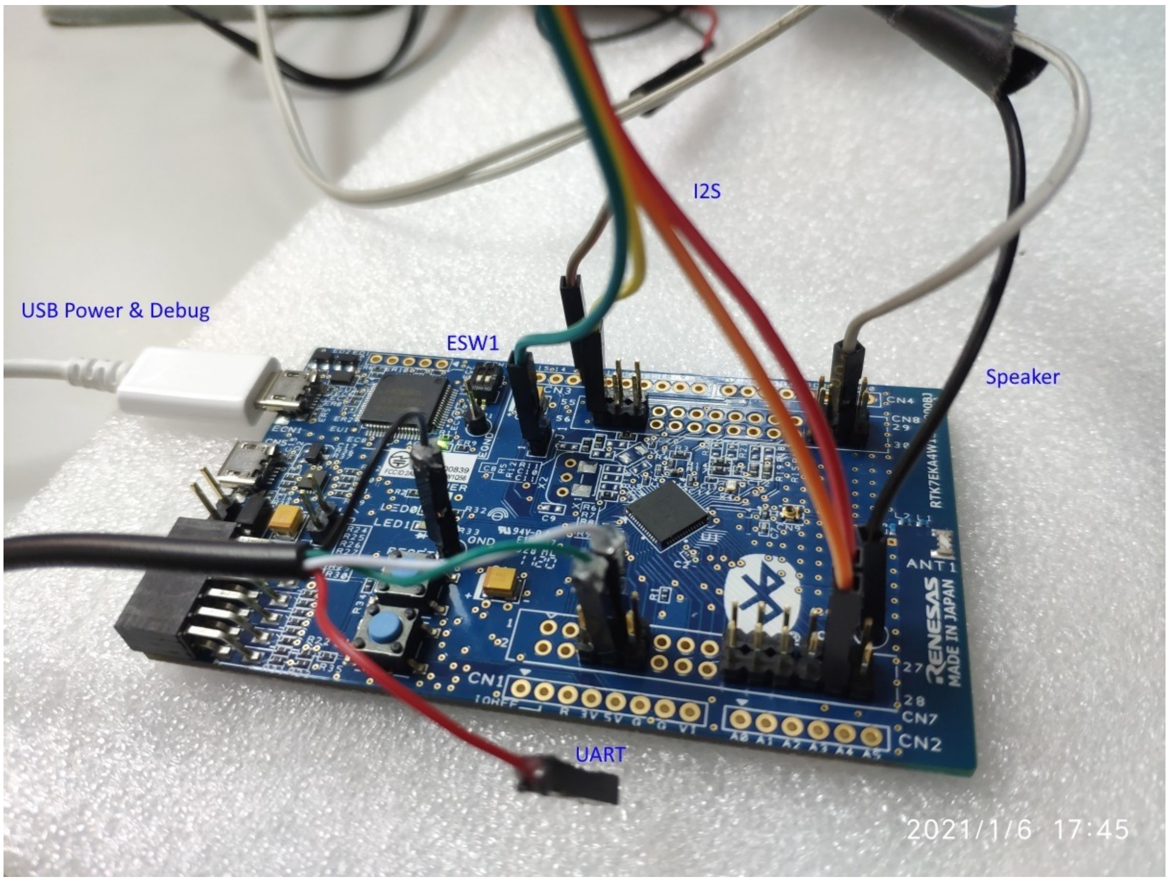
6. RA4M2 -VC2:



There is no I2S interface on 48 pin RA4M2, so the I2S microphone need to connect through SPI interface. The VC2 demo board support RA6M1 64 pins or RA4M2 48 pin MCU.
Please connect UART to J2 UART3.

The sample project: /SampleCode/ra4m2_vc2_voice_command_ds22x
Please copy "/SampleCode/src" directory to this path.

7. EK-RA4W1



There is no I2S interface on 64 pin RA4W1, so the I2S microphone need to connect through SPI interface. Please connect the following pins:
The pin group 1: P102, P104 and P111.
The pin group 2: P103 and P105

Please check the ESW1 is set to 1-ON 2-ON for USB debug.

Please connect UART Tx to P205, Rx to P206, GND.
Please connect I2S microphone module to:
3.3V, LRC to P409, DATA to P101, BCLK to P102, GND.
Please connect speaker to P014 and GND.

At current time, the sample code doesn't pass the test. It will be released in the future.

1.2. Software

1. IDE: e2studio V2021-1
2. FSP: 2.3
3. Windows license tool DSpotterLicenseTool.exe

   This tool is used to active the license of DSpotter. If the demo board is not licensed, please contact Dale (dale.lee.px@renesas.com) for this tool and authentication certificate file.
4. Windows tool CybSerialRecorder.exe

   Please download it at:
   https://drive.google.com/file/d/1pCU0_RkbFy7sERgD1iDgpglR1dt1Ud0j/view?usp=sharing

2. The Basic of Sample Code

2.1. The Projects of Sample Code

The DSpotter sample code is separated to seven projects for different hardware configuration:

1. ra6m1_voice_command_ds22x
   The sample project of RA6M1 Voice Recognition demo board V1.
2. ra6m1_voice_command_ds22x_60_15MHz
   The sample project of RA6M1 Voice Recognition demo board for demo VAD and power mode control. The clock date of normal mode is 60MHz when VAD detect the activity of human speech, and switch to 15MHz power saving mode when no speech detected.
3. ra6m1_vc2_voice_command_ds22x
   The sample project of RA6M1 Voice Recognition demo board V2.
4. ra6m1_ek_voice_command_ds22x
   The sample project of EK-RA6M1.
5. ra6m4_ek_voice_command_ds22x
   The sample project of EK-RA6M4 demo board.
6. ra4m2_ek_voice_command_ds22x
   The sample project of EK-RA4M2 demo board.
7. ra4m2_vc2_voice_command_ds22x
   The sample project of RA4M2 Voice Recognition demo board V2.

All sample code are shared with same source code located in SampleCode\src. These projects have different project property and compile option defined in voice_main.h for the different hardware configuration:

1. MCU type option
   The MCU_TYPE can be defined to one of following definition:
   MCU_TYPE_RA6M1
   MCU_TYPE_RA6M1_EK
   MCU_TYPE_RA6M1_VC2
   MCU_TYPE_RA6M4
   MCU_TYPE_RA4M2
   MCU_TYPE_RA4M2_VC2
   MCU_TYPE_RA4W1

Every MCU may have different flash address and size. Every demo board may have different LED IO pin.

2. Audio record option
   The AUDIO_RECORD can be defined to one of following definition:
   AUDIO_RECORD_I2S
   AUDIO_RECORD_SPI
   AUDIO_RECORD_AMIC
3. Power mode control option
   The POWER_MODE can be defined to one of following definition:
   POWER_MODE_NONE
   POWER_MODE_60_15_MHZ

2.2. Record Module - AudioRecord

The record module AudioRecord.c support audio recording from I2S or SPI interface. Please define AUDIO_RECORD to one of following value:

1. AUDIO_RECORD_I2S for recording from I2S microphone.
2. AUDIO_RECORD_SPI for recording from SPI interface.
3. AUDIO_RECORD_AMIC for recording from analog micrphone.

The record module use ring buffer to cache audio data, the ring buffer size is defined in AudioRecord.c:

```
#define RBUF_SIZE_RECORD    (320*9)      // Record ring buffer size, 90 ms.
```

When system callback new recording data by g_audio_cb(I2S) or g_spi_cb(SPI), the data will put into ring buffer. If ring buffer is full, this data will be skipped and the lost count will increase by one.

Please refer AudioRecord.h for the detail description of record function.

2.3. Power Mode Control Module - PowerMode

The power mode control module PowerMode.c has only two function:

1. EnterNormalPowerMode()
   Turn on green LED.
   In ra6m1_voice_command_ds22x_60_15MHz project, it will set system clock to 60 MHz and do nothing for other project.

2. EnterLowPowerMode()
   Turn off green LED.
   In ra6m1_voice_command_ds22x_60_15MHz project, it will set system clock to 15 MHz and do nothing for other project.

For ra6m1_voice_command_ds22x_60_15MHz project, please define:
"POWER_MODE = POWER_MODE_60_15_MHZ"

2.4. Integration of Recognition Model

The DSpotter Modeling Tool(DMST) is used to define command, fine tune, test and output the final model files. About the DSMT model files:

1. The basic model files are: CYBase.mod, Group_1.mod... Group_N.mod
2. The model file for voice tag: CYTrimap.mod
3. The command information file: Group_x.txt
   It contains the command text, map ID and saved in UTF-16 format.
4. DSpotter sample code treat Group_1 as wake up command group, and Group_2 as function command group.
5. DSpotter sample code need two type of packed model file.
   XXX_pack_withTxt.bin: (for normal usage)
     Pack with basic model files and command file.
   XXX_pack_withTxtAndTri.bin: (for support voice tag)
     Pack with basic model files, CYTrimap.mod and command file.
6. Please put packed model file to the sample code path /src/model/ and modify /src/model/CybModel1.asm to use this packed model file.
7. DSpotter sample code support to recognize two model at the same time. Please modify /src/model/CybModel2.asm if it is second packed model file.
8. Please clean and rebuild project after update the packed model file.

2.5. Packed Model Access Module - CybModelInfor

This module is used to handle packed model file:

1. Get the address of basic model for DSpotter initialization.
2. Get the address of CYTrimap.mod for DSpotter voice tag training.
3. Get the command information(text and ID) by group index and command index. The command text format maybe UTF16 or UTF8.

2.6. Flash Access Module - FlashMgr

This module is used to write data to data flash. DSpotter library will use this module to write license information to data flash. This module must put in sample code project. The main function of FlashMgr:

1. To erase blocks of data flash.
2. To check blank of data flash.
3. To write data to data flash.

2.7. Uart Access Module - UartMgr

This module is used to access data with Uart. DSpotter sample code use it to log text(please define UART_LOG in project setting) and dump audio data through Uart(please define SUPPORT_UART_DUMP_RECORD in voice_main.h). The DSpotter engine also use it to communicate with license tool. The main function of UartMgr:

1. Write data synchronously or asynchronously.
2. Read data asynchronously.
3. Write format string.

2.8. Auto Gain Control Module - AGCApi

For far field use case, we need to amplify the record data for better recognition rate. But it may clip data if we use in the near field with fixed gain. This module will adjust gain automatically to avoid data clipping.

2.9. Unpack/Decode Speex and Play Module - PlaySpeex

Speex is an audio codec, we use it to compress audio data to save storage. Please refer “Cyberon Speex Programming Guide” for the detail description.

The Windows tool WaveToSpeexPack.exe is used to compress a lot of wave files by Speex audio codec to one speex pack file. Please refer “WaveToSpeexPack User Guide” for this function.

The main function of PlaySpeex:

1. Unpack the speex pack file.
2. Decode the compressed data.

3. Play the audio data.

2.10. System Dependent Module - PortFunction

This module include the OS or MCU platform dependent function:

1. Software delay function. The clock rate is different for different MCU, so it is MCU platform dependent. We separate it from our engine library.
2. Memory management function.

3. Function Option

3.1. One Stage Recognition

It means every command is wake up and function command. Please design command and test it carefully at this condition. Too many commands may result in unacceptable false alarm rate.

In voice_main.h, please define:

```
#define RECOG_FLOW    RECOG_FLOW_ONE_STAGE
```

Every sample model file has two group, please define the test group index by:

```
#define ONE_STAGE_GROUP_INDEX    1 // or 0
```

3.2. Two Stage Recognition

The sample code will active the first group for wake up command recognition. After wake up command recognized, it will active the second group for function command recognition. If no function command recognized in 6 seconds, it will return to the first stage, otherwise it will reset the timing and keep in second stage.

In voice_main.h, please define:

```
#define RECOG_FLOW    RECOG_FLOW_TWO_STAGE
```

3.3. Play Response Audio After Recognition

After we use the Windows tool WaveToSpeexPack.exe to generate a speex pack file(*.spxpack), please put it to /src/model, and modify SpeexData.asm to use the file.

The PlaySpeex module will decode and play the audio by specify the command map ID.

In voice_main.h, please define SUPPORT_SPEEX_PLAY.

3.4. Recognize Two Model

It is used for some special case. For example, to recognize two language(like English and Japan) model at the same time. The model file size, RAM and CPU requirement are near to double of one model.

In voice_main.h, please define SUPPORT_RECOG_TWO_MODEL.

3.5. Voice Tag

DSpotter has a set of function(DSpotterSD_XXX) to record user's speech and train a voice tag command model. Please refer DSpotter SDK document for detail description. The initial API of voice tag need CYTrimap.mod, so we need the packed DSpotter model file: XXX_pack_withTxtAndTri.bin.

The operation of voice tag:

1. To start the voice tag training procedure, please click the S2 button.
2. The sample code will release DSpotter for save memory, initial DSpotterSD and check the human voice in recording data.
3. When it catch the human speech, it will train a voice tag model of the speech. The sample code save voice tag model to the memory:
   ```
   #define VOICE_TAG_MODEL_SIZE     2048
   g_byaSDModel[VOICE_TAG_MODEL_SIZE]
   ```
   It can store 4 ~ 8 voice tag.
4. After training success, the sample code will release DSpotterSD and initial DSpotter by multi-group configuration:
   The first group is the original group(wake up or function command group).
   The second group is the voice tag group.
5. The voice tag can be recognized both in wake up and function command stage.
6. To train the second voice tag, please click the S2 button again.

The limitation of voice tag:

1. There is no chance to fine tune it in advance. The false alarm rate may be higher than the pre-tuned command.
2. To reduce the false alarm rate, we recommend user to say five syllable at least when record voice tag.
3. To mapping the voice tag to the specific command, the device need a user interface.

In voice_main.h, please define SUPPORT_VOICE_TAG.

3.6. Voice Activity Detection (VAD)

The VAD is used to support power saving control. It will control system to enter low power mode when there is no human voice detected, and switch to normal power mode when it detect the human voice. Please refer “Cyberon VAD Programming Guide” for detail information.

Please note:

1. The recording and its quality can’t interrupt by power mode switch.
2. The voice start point will detect lately than the real start time. The sample code use another ring buffer (g_hCacheRingBuffer) to cache the recent 400 (VAD_CACHE_FRAME_COUNT = 40) ms recording data. After the voice start detect, DSpotter will get recording data from this ring buffer.
3. This VAD is designed for voice recognition, it is very sensitive for not to affect the performance of DSpotter.

In voice_main.h, please define SUPPORT_VAD.

4. The Definition

4.1. Global Definition

The global definition is defined in project properties: C/C++ Build => Settings => GNU ARM Cross C Compiler => Preprocessor. Please define the following global definition:

1. UART_LOG
   To log text to UART or transmit license through UART, please define it.
2. MCU_TYPE=MCU_TYPE_XXX
   Some settings is MCU dependent, so we must define it. Please refer to voice_main.h for MCU_TYPE_XXX.
3. AUDIO_RECORD=AUDIO_RECORD_XXX
   The same MCU platform may use different record source. Please refer to voice_main.h for AUDIO_RECORD_XXX. The default setting is AUDIO_RECORD_I2S if we don't define it.
4. POWER_MODE=POWER_MODE_XXX
   The power saving mode definition. The default setting is POWER_MODE_NONE if we don't define it.

4.2. MCU Platform Dependent Definition

We define MCU platform dependent definition in voice_main.h:

1. FLASH_HP_DF_SIZE
   The total size of data flash.
2. FLASH_HP_DF_BLOCK_SIZE
   The block size of data flash.
3. FLASH_HP_DF_BLOCK_BASE_ADDRESS
   The base address of data flash.
4. LED_R, LED_G, LED_B, LED_Y
   The IO pin number of LED.
5. ON, OFF
   The digit level (0 or 1) to light on/off the LED.

4.3. Application Definition

There are some application definition defined in voice_main.c:

1. DSPOTTER_MEM_SIZE_1
   The sample code declare a global array for this size and used by the first command group of DSpotter. The memory requirement can be got by API DSpotter_GetMemoryUsage_Multi(), so we can set this value by this API's return value.
2. DSPOTTER_MEM_SIZE_2
   The sample code declare a global array for this size and used by the second command group of DSpotter.
3. MAX_COMMAND_TIME
   The command must be spoke in MAX_COMMAND_TIME*10 ms. The larger the value, the more DSpotter memory is used.
4. COMMAND_RECOGNIZE_TIME
   The max recording time at the command recognizing stage. If no recognized result in this time, it will return to trigger recognizing stage.
5. VAD_FRAME_SIZE
   The algorithm frame size of VAD. It is 320(160 samples, 10 ms) for Cyberon's VAD. Please don't change it, except you replace other's VAD.
6. VAD_CACHE_FRAME_COUNT
   Because the detection delay of VAD, the sample code will use a ring buffer to cache the recent recording frames. The maximum detection delay of Cyberon's VAD is about 300 ms, we set this value to 40(40*10=400 ms) for safety.
7. VOLUME_SCALE_RECONG
   The AGC volume scale percentage for recognition. It depends on original microphone property. Please set to larger value for far field recognition.
8. VOLUME_SCALE_VOICE_TAG
   The AGC volume scale percentage for record voice tag. We usually record voice tag at near field, so this value is smaller.
9. COMMAND_STAGE_TIME_MIN
   When no result at command recognition stage, the minimum recording time in ms.
10. COMMAND_STAGE_TIME_MAX
    When no result at command recognition stage, the maximum recording time in ms.
11. VOICE_TAG_MAX_TIME
    The max speech time of a voice tag in ms.
12. VOICE_TAG_VOICE_SIZE
    The size of voice data buffer for voice tag training. It is dependent to

VOICE_TAG_MAX_TIME, about 16KB for 3 seconds voice tag.

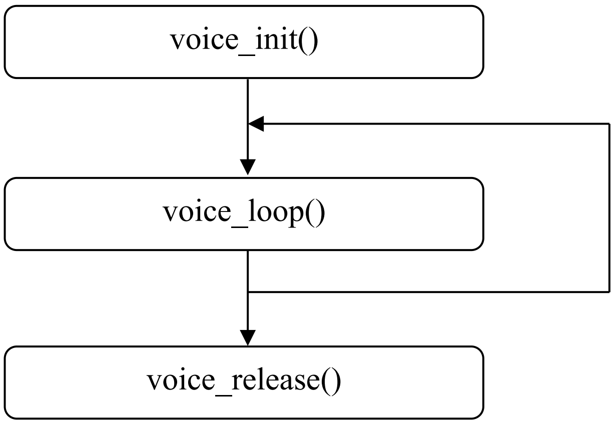
13. VOICE_TAG_MODEL_SIZE

    The size of temporary RAM buffer for saving voice tag model, it has 340 bytes header and need about 400B for each voice tag.

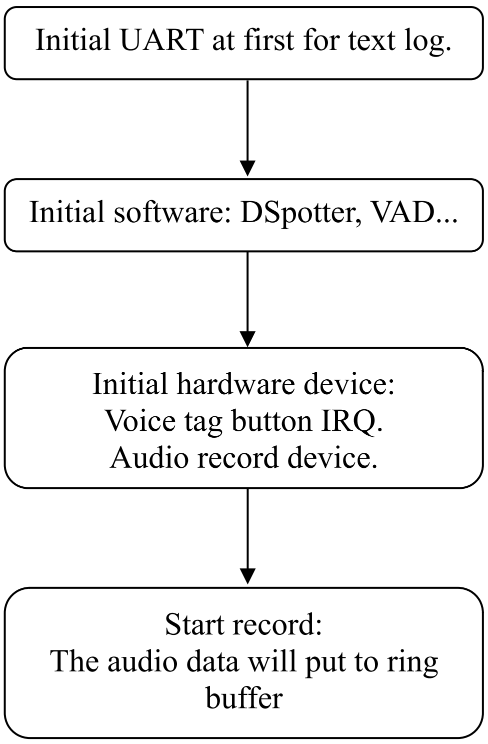
5. The Function Flow Chart

5.1. voice_main

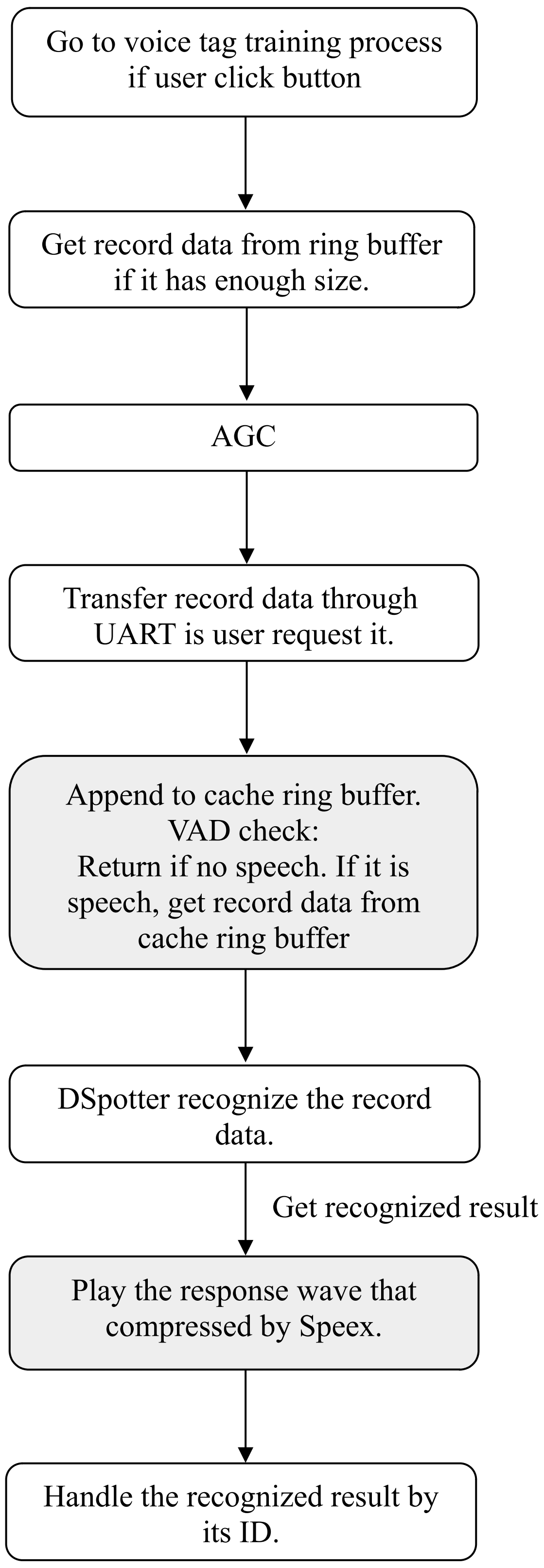
The voice_main() function is the entry function of sample code.



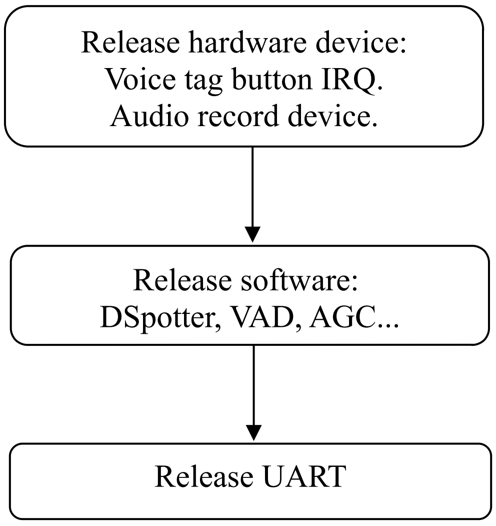
5.2. voice_init


Initial UART at first for text log.
Initial software: DSpotter, VAD...
Initial hardware device:
Voice tag button IRQ.
Audio record device.
Start record:
The audio data will put to ring buffer

5.3. voice_loop


Go to voice tag training process if user click button
Get record data from ring buffer if it has enough size.
AGC
Transfer record data through UART is user request it.
Append to cache ring buffer. VAD check: Return if no speech. If it is speech, get record data from cache ring buffer
DSpotter recognize the record data.
Get recognized result
Play the response wave that compressed by Speex.
Handle the recognized result by its ID.

5.4. voice_release


Release hardware device:
Voice tag button IRQ.
Audio record device.
Release software:
DSpotter, VAD, AGC...
Release UART