APPLICATION NOTE

# SPEEX CODEC IMPLEMENTATION ON THE RPB RX210


Author: César MAKAMONA MBUMBA SHÉALTIEL

**January 2013**

# Abstract

Codec is a compound word derived from **Co**der-**dec**oder or **Co**mpressor-**dec**ompressor. A codec is software, a program, or hardware, a device, process used to encode/compress and decode/decompress data stream and signal. In one hand, it encodes data stream or signal for transmission, storage or encryption. In the other hand, it decodes for editing or playback.

Depending on the application aim, bitrate, time computing, audio quality … different algorithms or diagrams are used. A Codec can be used in media transmission on Internet, telephony, videoconferencing …

This application note describes how to build a Vocoder application by implementing the Speex codec on the RX210 microcontrollers.

# Contents

# List of figures

# List of tables

# I. Speex Codec

Speex is an open source, patent and royalty-free software codec designed and optimized for speech. It based on the CELP (Code Excited Linear Prediction, based itself in Linear Prediction Code) and was designed in particular for voice over IP. So it can encode voice at bitrates ranging from 2 to 44 kbps.

Here a summary Speex features:

- Resampler
- Fixed point implementation
- Intensity Stereo encoding
- Encoding/Decoding sampling rate : Narrowband (8 kHz), Wideband (16 kHz) and Ultra-wideband (32 kHz)
- Voice activity detection (VAD)
- Discontinuous transmission (DTX)
- Variable bitrate operation (VBR)
- Packet loss concealment
- Noise suppression

Speex has some features that are absent in some speech codecs, such as VBR, DTX and multiple sampling rates in the same bitstream and it designed to run on different platforms, software and hardware. For further details about Speex codec, please refer to the Speex website, **www.speex.org**, and Speex manual. 1.2rc1 is the release used for this application note.

# II. Hardware overview

To build a typical vocoder application, two stages are needed: speech processing and audio stream input/output. In this application note, the Speex codec is the first one and the RPB RX210 board is the second one.

The RPB RX210 microcontroller integrates the RX210 CPU, which is a 32 bits CPU without a Floating Point Unit (FPU). It can process MAC operations on 48 bits and 78 DMIPS at 50 MHz (maximum operating frequency) and has several peripherals as shown on figure 1.

In this application note, the embedded 12-bits resolution ADC, input audio stream, and 10-bits DAC, output audio stream, are used, as well as 8-bits Timer and DMAC.

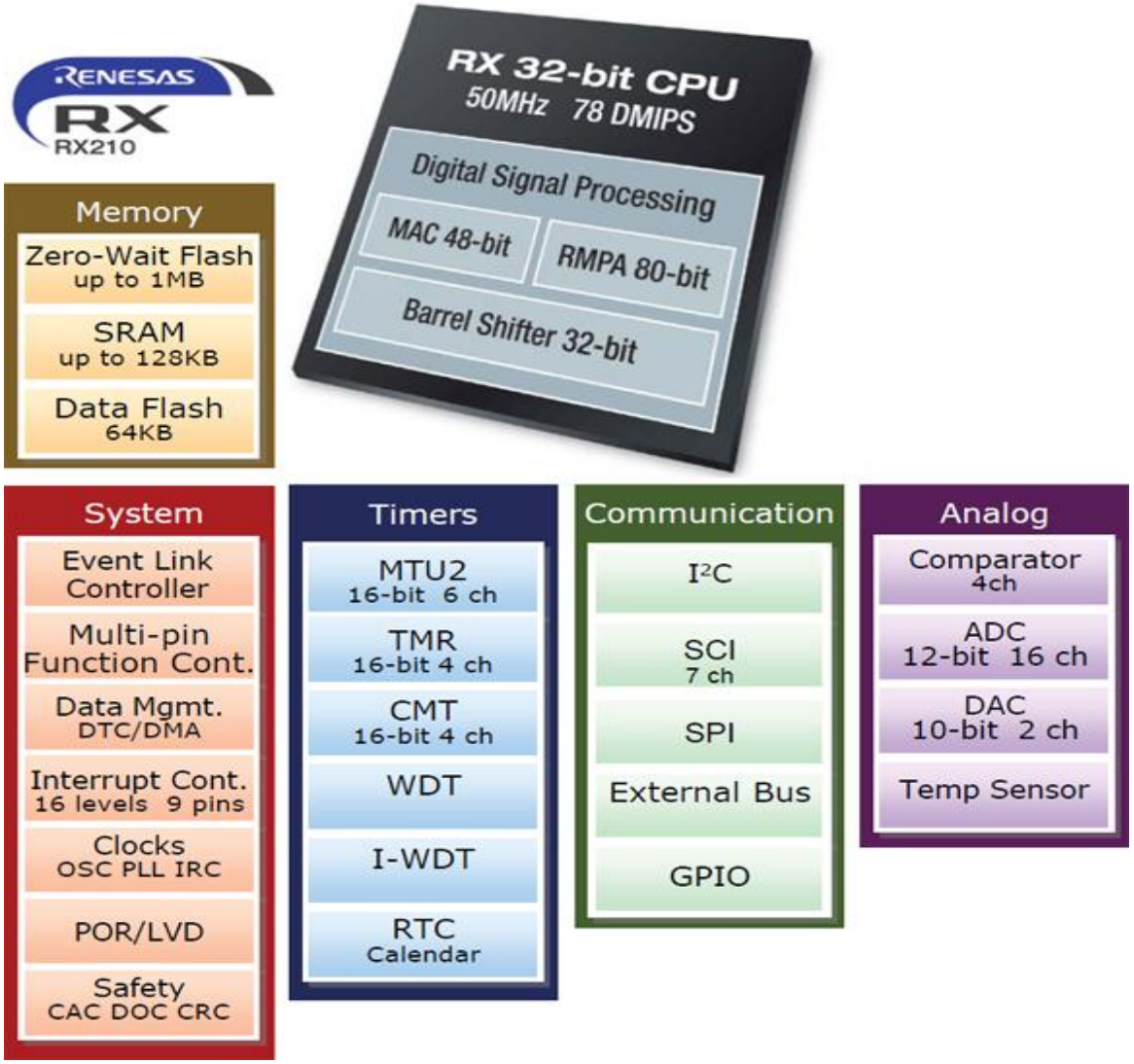


**Figure 1 : RX210 features**

For more information about the RX210 microcontrollers, please refer to the Renesas website, http://www.renesas.eu/products/mpumcu/rx/rx200/rx210/index.jsp, and RX210 hardware manual.

# III. Speex implementation

This part describes necessary modifications to operate on Speex codec library, to port Speex codec on platforms that were not designed for. However, these modifications depend on the application needs: compression, CPU load, audio quality …

In this application note, the porting is based on Fixed-point, CPU load and audio quality. In light of these needs, the Narrowband mode is used to meet the lesser CPU load, since it requires less data processed than others modes.

The Ogg library is not provided with Speex library. So, get the Ogg library on www.xiph.org. The release used in this application note is 1.3.0.

## 1. Encoding

To get Speex operating on Fixed-point, include in **arch.h** file, following macros:

- #define FIXED_POINT
- #define USE_KISS_FFT
- #define DISABLE_FLOAT_API
- #define DISABLE_VBR

Running on Fixed-point implies to disable VBR.

| Mode | Quality | Bit-rate (bps) | mflops | Quality/description |
|---|---|---|---|---|
| 0 | - | 250 | 0 | No transmission (DTX) |
| 1 | 0 | 2,150 | 6 | Vocoder (mostly for comfort noise) |
| 2 | 2 | 5,950 | 9 | Very noticeable artifacts/noise, good intelligibility |
| 3 | 3-4 | 8,000 | 10 | Artifacts/noise sometimes noticeable |
| 4 | 5-6 | 11,000 | 14 | Artifacts usually noticeable only with headphones |
| 5 | 7-8 | 15,000 | 11 | Need good headphones to tell the difference |
| 6 | 9 | 18,200 | 17.5 | Hard to tell the difference even with good headphones |
| 7 | 10 | 24,600 | 14.5 | Completely transparent for voice, good quality music |
| 8 | 1 | 3,950 | 10.5 | Very noticeable artifacts/noise, good intelligibility |

**Figure 2 : Narrowband mode, Quality vs Bitrate**

Quality is a parameter used to control bitrate. Referring to figure 2, in order to get a low CPU load with understandable audio output quality, quality is set to 4. In the same way, the complexity parameter is set to 1. With this setting, the ratio compression is 16:1.

Then all features that are unused for the application are removed, that implies removing relative files, functions and variables:

- VBR
- Stereo
- VAD
- DTX
- Preprocessor
- Resampler
- Average Bitrate (ABR)
- Perceptual enhancement (used only by decoder)
- Acoustic Echo Canceller

As shown below, Speex is designed to operate with audio files, specially, raw, .wav and .spx ones. To run in real time on the board, files are replaced by buffers.

Algorithm (simplified) of Speex source code

| Encoder | Decoder |
|---|---|
| • Reading the input file : wav or raw | • Reading the input file : .spx |
| • Encoding | • Decoding |
| • Writing the output file : spx | • Writing the output file : wav or raw |

So, reading/writing the input/output file is modified accordingly: **read_samples** and **oe_write_page** functions are main ones. At this step, make sure to modify all others relative functions and remove functions such as **exit**, **printf**, **fprintf** … Also remove Windows execution options relatives. Redefine Ogg and Speex types according to the platform used (Table 1), replace **srand** and **rand** functions by **speex_rand** function and replace **fread** and **fwrite** functions by **memcpy** one.

| | | | |
|---|---|---|---|
| **typedef** _SWORD | ogg_int16_t; | **typedef signed char** | spx_int8_t; |
| **typedef** _UWORD | ogg_uint16_t; | **typedef unsigned char** | spx_uint8_t; |
| **typedef** _SINT | ogg_int32_t; | **typedef signed short** | spx_int16_t; |
| **typedef** _UINT | ogg_uint32_t; | **typedef unsigned short** | spx_uint16_t; |
| **typedef** _SQWORD | ogg_int64_t; | **typedef signed int** | spx_int32_t; |
| **typedef** _UQWORD | ogg_uint64_t; | **typedef unsigned int** | spx_uint32_t; |

**Table 1 : Redefinition of Speex/Ogg types**

These functions must be modified:

- speex_encoder_ctl
- speex_encode_int (the main encoding function)
- speex_encoder_destroy
- speex_bits_destroy
- ogg_stream_clear

To avoid strange behaviors, especially in temporary memory allocation, modify the following functions:

- **speex_alloc**
- **speex_free**
- **speex_realloc**
- **speex_alloc_scratch**

## 2. Decoding

Fixed-point is managed as in encoding. Quality, complexity and Bitrate parameters are not set here. Do not remove perceptual enhancement and when removing Stereo relatives, pay attention, it quiet complicate in decoding.

These functions must be modified:

- process_header
- speex_decoder_ctl
- speex_decode_int (the main decoding function)
- speex_decoder_destroy
- speex_bits_destroy
- ogg_sync_clear

Remain actions, modifications and suppressions, are made as in encoding.

# 3. Optimization

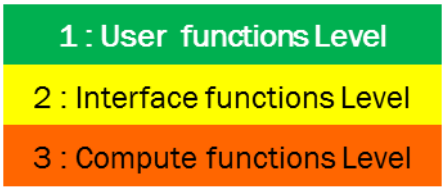


**Table 2 : Speex program architecture**

In both encoding and decoding, in some cases, it better to omit the interface level in order to optimize CPU load. Function such as **nb_encode** (level 3) can be directly called, by replacing **speex_encode_int**, in leve 1.

Always in reducing CUP load, ideally, these functions should be written in assembly:

- filter_mem16()
- iir_mem16()
- vq_nbest()
- pitch_xcorr()
- interp_pitch()

Remind that only one mode, Narrowband, is used. So, files, functions, structures, tables and variables relative to Wideband and Ultra-wideband can be removed to optimize code size. In the same way, include **#define VAR_ARRAYS** or **USE_ALLOCA** in arch.h file.

If the encoder and decoder are two separated modules, functions, tables, structures and variables which are used by decoder should be removed in encoder and vice versa.

# IV. Vocoder application

The vocoder is a speech processing application that provides human voice treatment such as encoding, decoding, filtering and amplifying. In this application note, the vocoder application is the **Encoder/Player**.

## 1. Application requirements

To buil this application, it requires to get:

- RPB RX210 board which integrate a Segger debugger (J-Link 4.38) www.renesas.eu/products/tools/introductory_evaluation_tools/renesas_promotional_boards/RPBRX210/rpbrx210.jsp
- Installation CD, delivered with the board, which contains debugger drivers and HEW installation
- High-performance Embedded Workshop (HEW), a RENESAS IDE for configuring, loading and debugging Renesas microcontrollers. The release used is 4.09.00.007.
- KPIT GNURX 12.02, an open-source and free-royalty compiler. For further details refer to www.kpitgnutools.com
- Audacity 2.0.2, an open-source and free-royalty audio recording and editing software. It is used, in the vocoder application, to record audio stream in correct data format. Downloadable at http://audacity.sourceforge.net/

In the project creation on HEW, using GNURX compiler, do not check the option surrounds on figure 3.

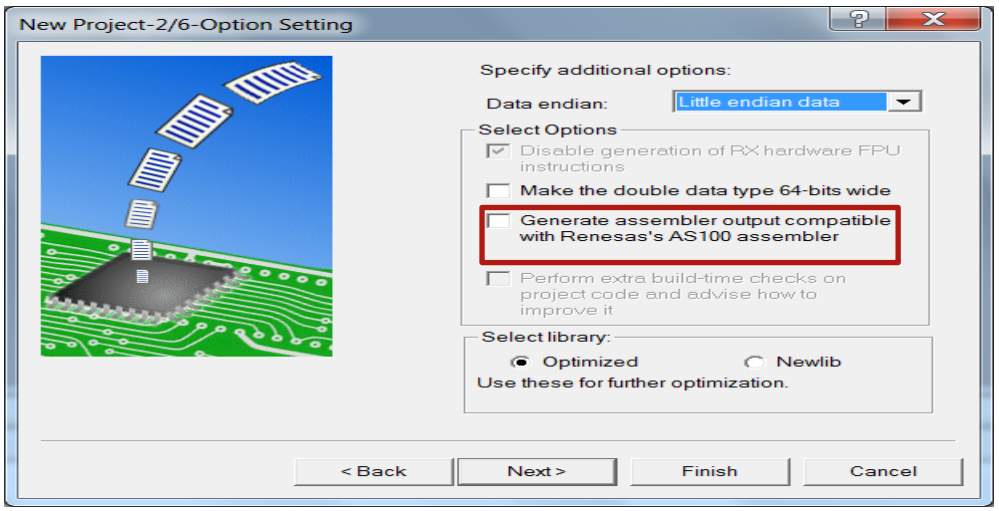


**Figure 3 : Project creation on HEW, with GNURX compiler**

Then copy the content of **rx200.h** file from C:\Program Files\Renesas\Hew\System\Pg\KPIT GNURX-ELF\GNURX_Info\Generate\iodefine to the **iodefine.h** generated in the project directory.

Make sure to have **Segger_JLink.hsf** or/and **JLinkOB.hsf** files, which permit to debug and write the On-chip memory respectively, in the project directory. Run demonstration projects to get these files. Steps below indicate how to run demonstration projects and to add session for debugging mode or write On-chip mode.

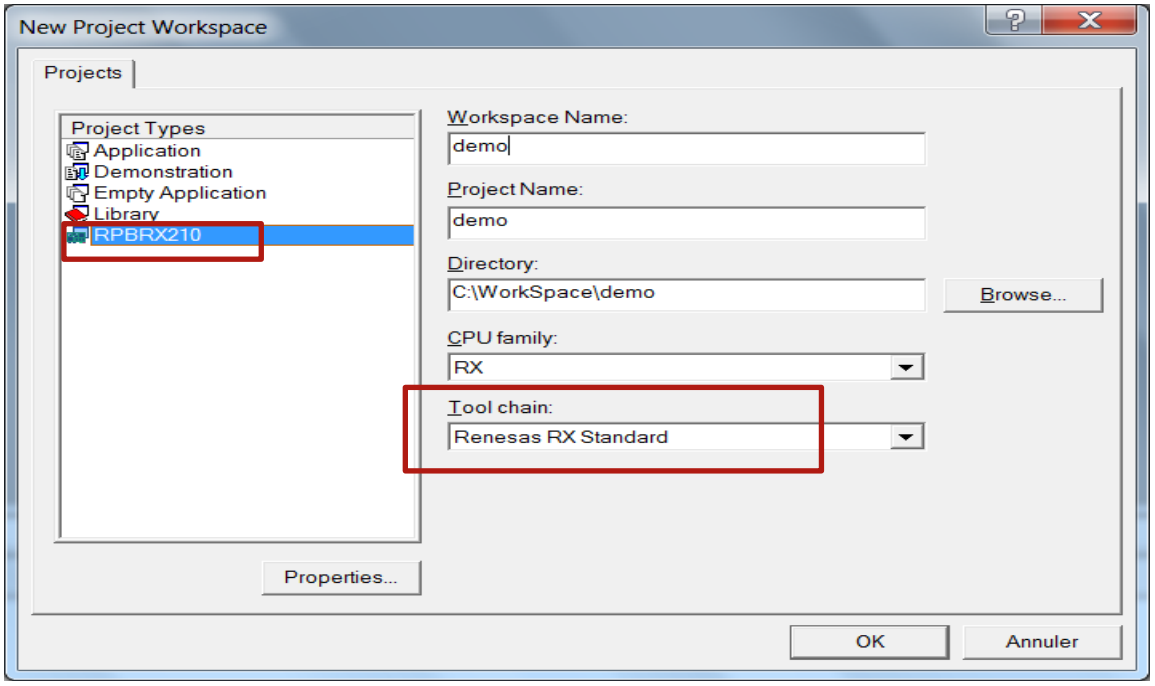


**Figure 4 : Running demonstration projects (1)**

Select Renesas tool chain and RPBR210 then press OK.

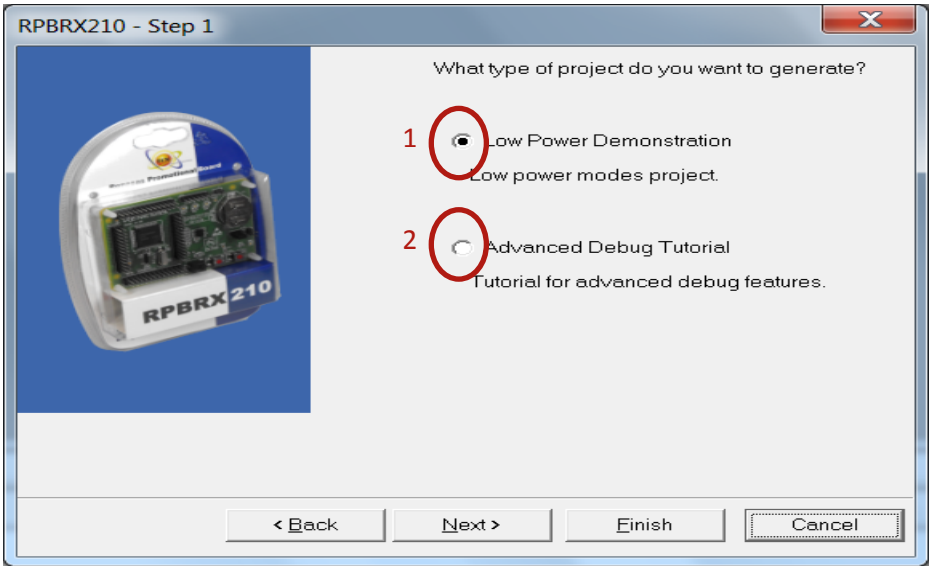


**Figure 5 : Running demonstration projects (2)**

Select 1 to get write On-chip mode file and 2 the debugging mode one.

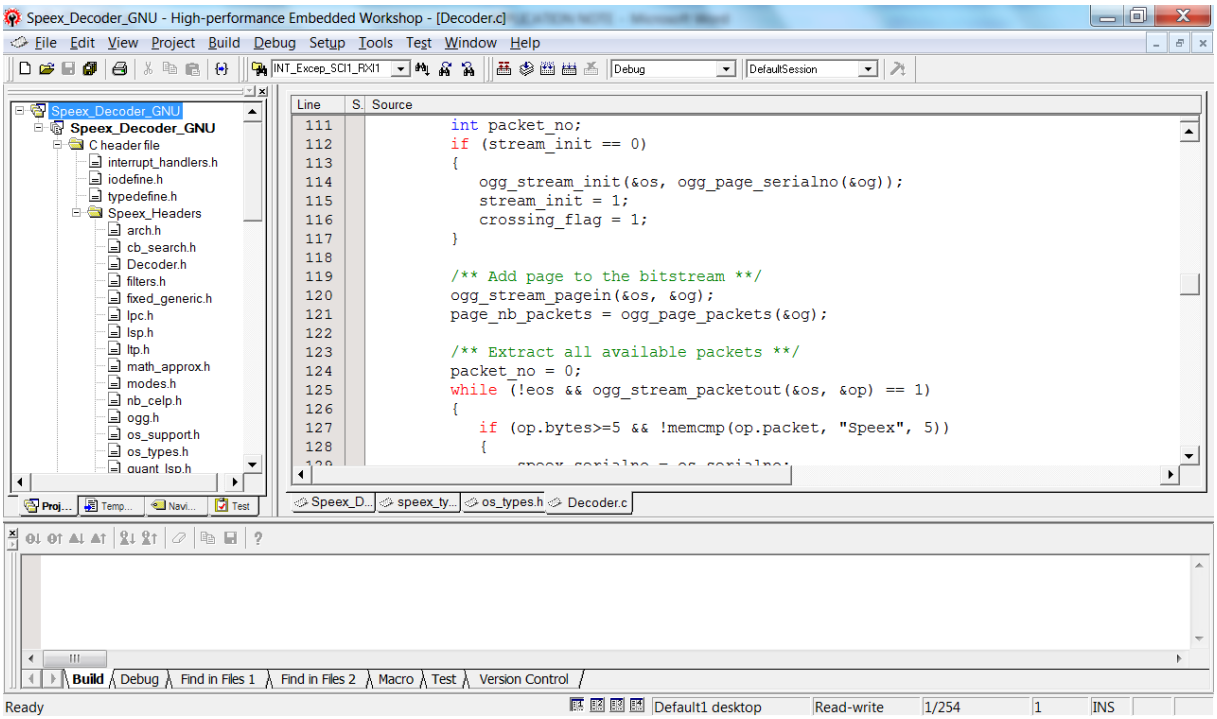


**Figure 6 : Adding debugging mode session (1)**

Notice that it is impossible to connect the board with default session

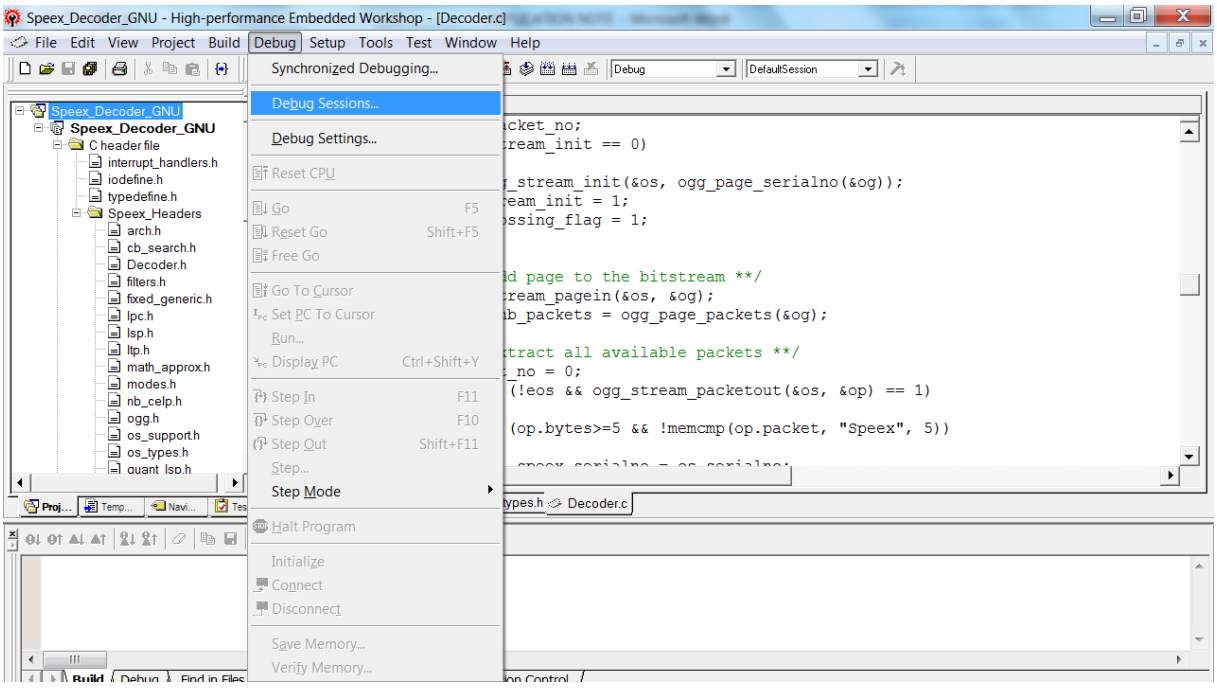


**Figure 7 : Adding debugging mode session (2)**

Select Debug →Debug Sessions, browse and add Segger_JLink.hsf. Then change the current session to Segger_JLink.

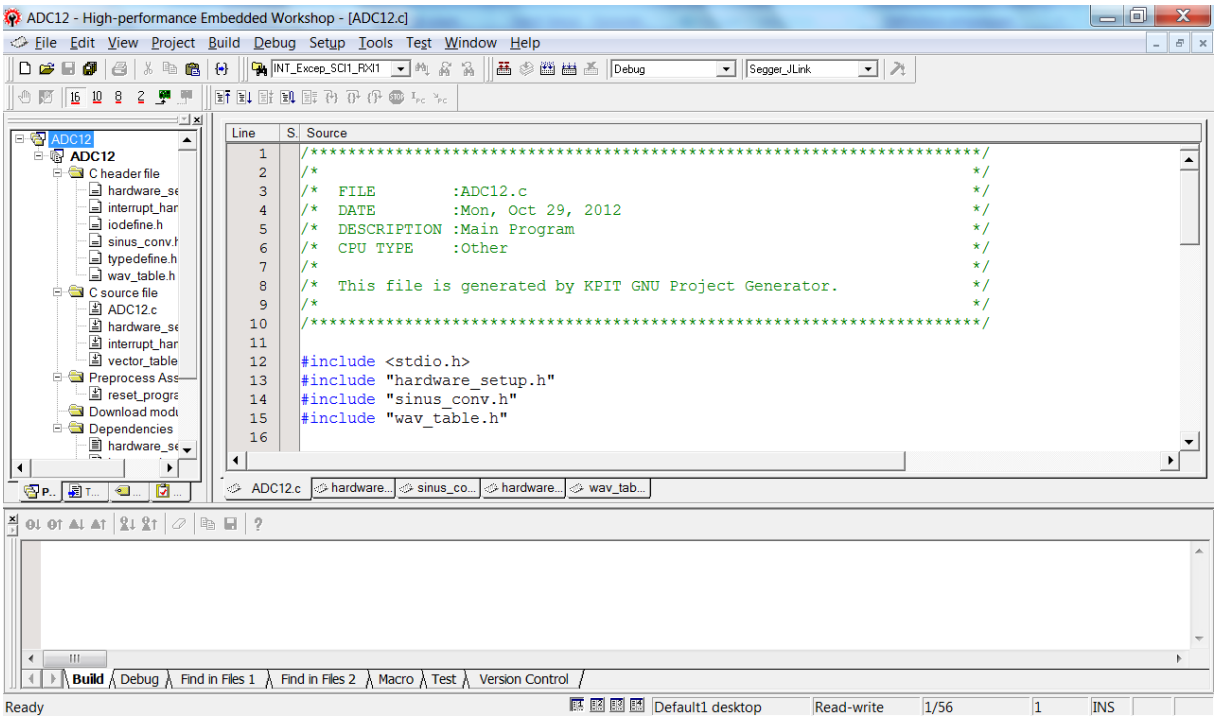


**Figure 8 : Adding debugging mode session (3)**

Now the program can be load on the board.

# 2. Speex Encoder/Player

For the following, it is supposed that Speex source code was already modified and tested, to be ported on the target. So, only hardware configuration will be detailed. This application is composed of two modules: Encoder and Player, as shown on figure 4.

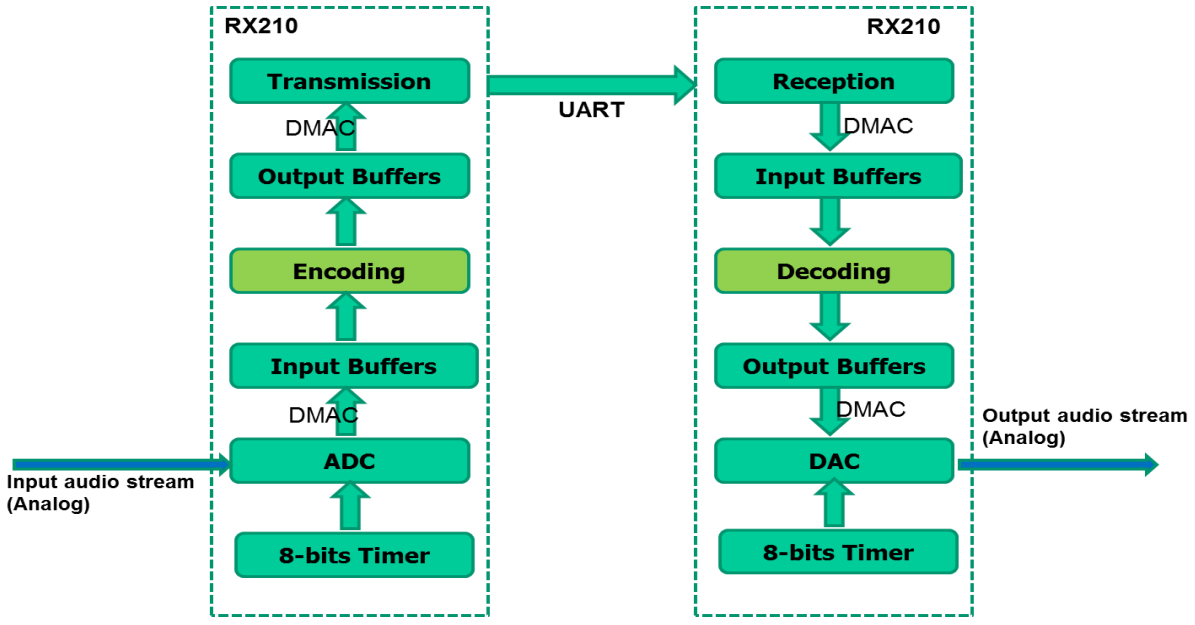


**Figure 9 : Speex Encoder/Player flow**

The Encoder is composed of ADC, 8-bits Timer, DMAC, UART and Speex encoding process. The Timer triggers the DMAC at 8 kHz (Narrowband), which triggers the ADC to transfer data in Input buffers. The DMAC launch encoding process when buffer is full. Two buffers of 16 bits x 160, according to the Speex settings describes above, are used as input cyclic buffer and one frame represents 20 ms (160 samples x 125 µs) of audio stream.

One frame is compressed to 20 Bytes (ratio compression 16:1) and data is packed before being sent. Also, two buffers of 8 bits x 300, to meet the Player configuration, are used as output cyclic buffer. Another DMAC channel is used to lead data to the UART module.

The same configuration is met in the Player module but in reverse, with input and output buffers sizes, 8 bits x 300 and 16 bits x 300, respectively.

Important to mention that RPB RX210 has not embedded pre-amplification stage, to capture input audio stream, and amplification stage to play back output audio stream. So, related board/stage or PC connectivity can be used. The table 3 shows the Speex implementation requirements.

| Functions | ROM kBytes | RAM kBytes | CPU load 50 MHz |
|---|---|---|---|
| Encoding | 68 | 5 | 50 % |
| Decoding | 57 | 3 | 10 % |
| Encoding + Décoding | 88 | | |
| | | | |

**Table 3 : Speex implementation requirements**

The memory footprints are calculated with hardware configuration and the CPU load with an approximate method as shown on figure 10. When Encoding and Decoding processes are in the same module, the RAM footprint and CPU load depend on the type of application.

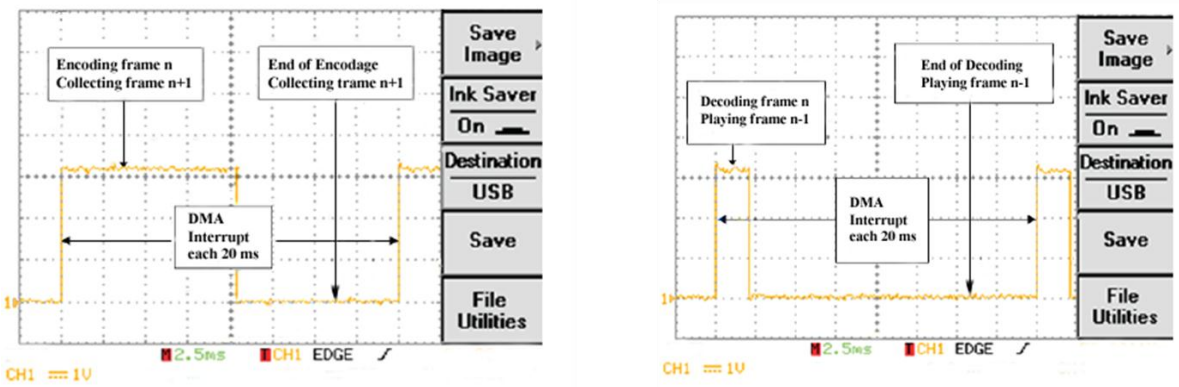


**Figure 10 : CPU load, Encoder (left) and Player (right)**

# Conclusion

Dedicated to speech encoding and decoding, Speex is a free audio codec which provides specific features, absent in others speech codecs, and a good quality of sound with a high level of compression. That ranks it as a high-performance solution for application using voice recorder or message playback, such as building and home safety systems, intercoms, answering machines, smart appliances, voice recorders or walkie-talkies.