... the world's most energy friendly microcontrollers

Speex Codec

AN0055 - Application Note

Introduction

Speex is a free audio codec which provides high level of compression with good sound quality for speech encoding and decoding. This application note demonstrates an energy efficient Speex solution on an EFM32GG for applications requiring voice recording or playback.

This application note includes:

- This PDF document
- Source files
  - Example C-code
  - Speex library files
  - Voice header files
  - IAR and Keil IDE projects
  - Pre-compiled binaries
- PC application to convert WAV file to Speex encoded format

1 Speex Overview

Speex is an open source and patent-free audio compression format designed for speech. Speex is based on CELP (Code-Excited Linear Prediction) and is designed to compress voice at bitrates ranging from 2 to 44 kbps. Some of Speex's features include:

- Narrowband (8 kHz), wideband (16 kHz) and ultra-wideband (32 kHz) compression
- Intensity stereo encoding
- Packet loss concealment
- Variable bitrate operation (VBR)
- Voice Activity Detection (VAD)
- Discontinuous Transmission (DTX)
- Fixed-point port
- Acoustic echo cancellation
- Noise suppression

This application note was developed with release 1.2rc1 of the Speex codec, implementing NB8K encoding and decoding, NB decoding, WB decoding and UWB decoding. For further details about the Speex codec, please refer to the Speex website: **www.speex.org**.

Acronyms used in this application note:

NB	Narrowband (8kHz)
NB8K	Narrowband 8 kbps only
WB	Wideband (16 kHz)
UWB	Ultra-wideband (32 kHz)

2 Speex Encoder

The Speex encoder consists of an audio input interface and speech encoding module. In this application note, the embedded 12 bit ADC in EFM32GG is used to collect audio samples whereas the Speex encoder software is used for speech encoding module.

2.1 Hardware

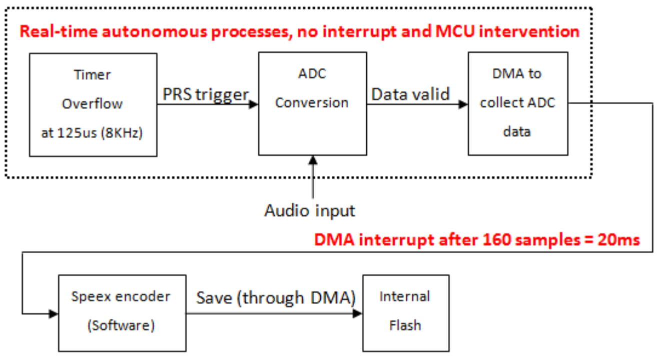
This application note was developed using the EFM32GG-DK3750 board which contains a line-in amplifier and filter (AUDIO IN jack) for audio input (see EFM32GG-DK3750 main board schematics).

A condenser microphone does not work with the AUDIO IN of EFM32GG-DK3750 board as this input requires line level signals.

2.2 Encoder Flow

The encoder is composed of embedded TIMER, PRS, ADC and DMA in the EFM32GG (Figure 2.1 (p. 3)). The TIMER triggers the ADC through PRS every 125us (8 kHz) and one frame (20ms) of ADC conversion data is collected by DMA for software Speex encoder. During the ADC data conversion phase, no MCU intervention is required and EFM32GG stays in EM1 (Sleep). Ping-pong mode DMA is used to ensure background ADC conversion during the Speex encoding process.

Figure 2.1. Hardware flow of encoder


Real-time autonomous processes, no interrupt and MCU intervention
Timer Overflow at 125us (8KHz)
PRS trigger
ADC Conversion
Data valid
DMA to collect ADC data
Audio input
DMA interrupt after 160 samples = 20ms
Speex encoder (Software)
Save (through DMA)
Internal Flash

2.3 Speex Encoder Library

In this application note the Speex encoder is implemented to support narrowband voice sampled at 8 kHz. The library also contains the 8 kbps decoder as described in Speex decoder section. The Speex encoder was modified to have a constant bitrate of 8 kbps with an aim to reduce memory footprint and use of the EFM32GG's resources. The Speex encoder library files are included in **\speex\libspeex** as described in Table 3.1 (p. 6) and the IDE settings for Speex encoder are described in Section 6.1 (p. 16) .

2.3.1 Clock Frequency

The EFM32 needs to run at 48 MHz clock with PREFETCH enabled to handle Speex encoding.

2.3.2 Bitrate and Compression Ratio

The Speex encoder has a parameter called “quality” that is used to control the bitrate. To set the bitrate to 8 kbps with 16:1 compression ratio, the quality parameter must be equal to 4 and kept constant (Table 2.1 (p. 4)). With the quality parameter equal to 4, the complexity of the encoder can only be set to a value lower than or equal to 2.

Table 2.1. Encoding quality vs bitrate and compression ratio

Quality	Complexity	NB Bitrate	Compression ratio
4	1 or 2	8 kbps	16:1

2.3.3 Memory Footprint

The FLASH and RAM usage of Speex codec is summarized in Table 2.2 (p. 4) (compiled with "Optimizations at High Speed" in IAR V6.40.2). These figures include 8 kbps encoder, decoder and hardware drivers.

Table 2.2. Memory footprint of Speex codec

	Flash (bytes)	RAM (bytes)
Codec NB8K	39,512	9,876

2.4 Speed vs. Memory Size

The memory size of a voice recording can be calculated as shown below:

Time interval between samples (8 kbps) = 1s/8000 = 125µs (16 bits per sample)

Frame length = 160 samples (16 bit) x 125µs = 20 ms

One frame will compress to 20 bytes (8 kbps). Time per byte = 20 ms / 20 bytes = 1 ms / byte

Example:

60 seconds 8 kbps Speex encoded voice data requires 60 / 1ms = 60000 bytes (58.6 KB)

Number of frames = 60s / 20ms = 3000 frames

- The 8 kbps encoded data is saved in memory space above 512 KB flash by read-while-write so only parts with more than 512 KB flash can run the Speex encoder demo.
- The start address of the encoded data must be greater than 512 KB with GG flash page size (4 KB) alignment.
- The 8 kbps encoded frame (20 bytes) is not a multiple of GG flash page size (4 KB) so the software skips the last 16 bytes (4096 – 204x20) on each flash page.
- For 512 KB flash, it can record 26,112 frames (204 x 128 pages), which is about 8.7 minutes of voice.

2.5 Current Consumption

The Speex codec is compiled with "Optimizations at High Speed" in IAR V6.40.2 and current consumption is measured with energyAware Profiler V1.01. Table 2.3 (p. 4) shows average current consumption of an EFM32GG when performing 8 kbps Speex encoding and decoding. The playback current consumption can be reduced further by dynamically prescaling the clock frequency from 48 MHz to 8 MHz. Note that these current numbers also include peripheral consumption from DACs etc.

Table 2.3. Average current consumption of 8kbps Speex codec

	NB8K record (Complexity = 1)	NB8K record (Complexity = 2)	NB8K playback
EFM32GG@48 MHz	15.5mA	16.6mA	7mA

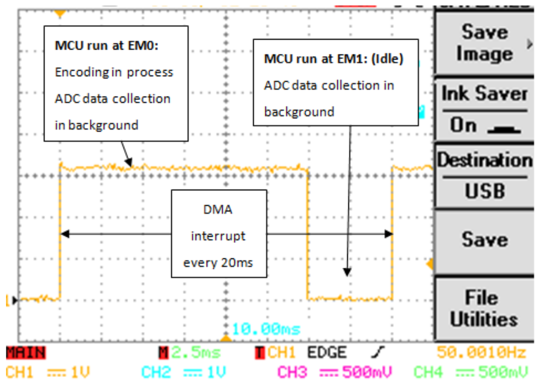
2.6 MCU Loading Analysis

MCU loading analysis is sometimes required to check whether the selected MCU clock frequency is fast enough to run the Speex codec. To investigate the MCU loading of Speex codec (NB8K), it needs to toggle one GPIO on start and end of encoding/decoding in speexfunc.c.

2.6.1 Speex Encoding (NB8K) with Complexity = 1

The MCU loading is ~75% with 48 MHz clock and PREFETCH enabled.

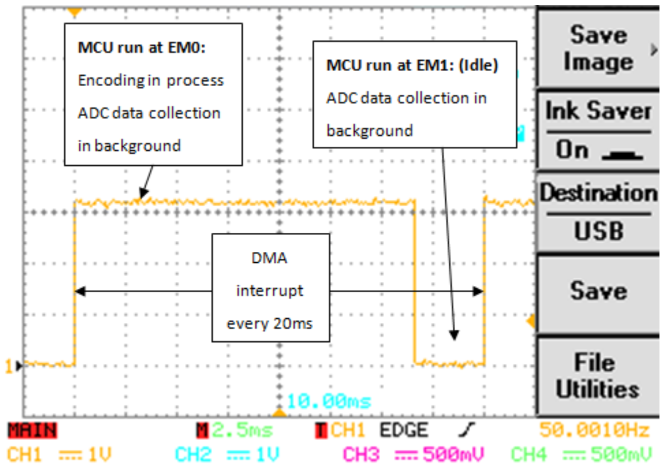
Figure 2.2. MCU loading at 8 kbps encoding (complexity = 1)



2.6.2 Speex Encoding (NB8K) with Complexity = 2

The MCU loading is ~82.5% with 48 MHz clock and PREFETCH enabled.

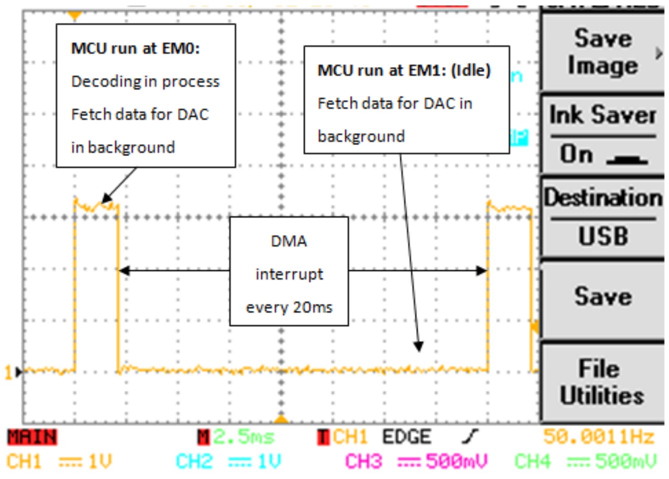
Figure 2.3. MCU loading at 8 kbps encoding (complexity = 2)



2.6.3 Speex Decoding (NB8K)

The MCU loading is ~10% with 48 MHz clock and PREFETCH disable.

Figure 2.4. MCU loading at 8 kbps decoding



3 Speex Decoder

The Speex decoder consists of an audio output interface and speech decoding module. In this application note, the embedded 12 bit DAC in EFM32GG uses for audio output interface whereas the Speex decoder software uses for speech decoding module.

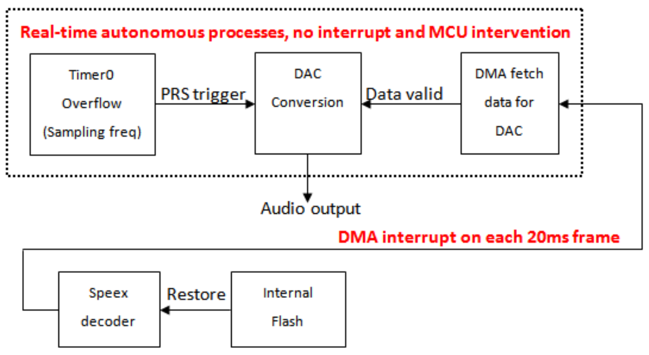
3.1 Hardware

This application note was developed using the EFM32GG-DK3750 board which contains a line out driver and filter (AUDIO OUT jack) for audio output (see EFM32GG-DK3750 main board schematics).

3.2 Decoder Flow

The decoder is composed of embedded TIMER, PRS, DAC and DMA in EFM32GG (Figure 3.1 (p. 6)). The TIMER triggers the DAC through PRS every sampling period and one frame (20ms) of data for DAC conversion is collected by the DMA from the software Speex decoder. During DAC data conversion phase, no MCU intervention is required and the EFM32GG stays in EM1 (Sleep). Ping-pong mode DMA is used to ensure background DAC conversion during the Speex decoding process.

Figure 3.1. Hardware flow of Speex decoder


Real-time autonomous processes, no interrupt and MCU intervention
Timer0 Overflow (Sampling freq)
PRS trigger
DAC Conversion
Data valid
DMA fetch data for DAC
Audio output
DMA interrupt on each 20ms frame
Speex decoder
Restore
Internal Flash

3.3 Speex Decoder Library

In this application note the Speex decoder is implemented to support narrowband (8 kHz), wideband (16 kHz) and ultra-wideband (32 kHz) voices at various bitrates with different compression ratios. Five projects are included to support different Speex modes on EFM32GG with an aim of reducing memory footprint and use of the EFM32GG's resources. The Speex library files (*.a for IAR and *.lib for Keil) are included in **\speex\libspeex** path (Table 3.1 (p. 6)).

Table 3.1. Projects and libraries summary

Project (IAR and Keil)	Speex Library	Encoder	Decoder
SpeexCodecNB8K	SpeexCodecNB8K	NB8K	NB8K
SpeexDecoderNB8K	SpeexDecoderNB8K	Not included	NB8K
SpeexDecoderNB	SpeexDecoderNB	Not included	NB
SpeexDecoderWB	SpeexDecoderWB	Not included	NB WB
SpeexDecoderUWB	SpeexDecoderUWB	Not included	NB WB UWB

3.3.1 Clock Frequency

The voice quality depends on the bitrate and sampling frequency, however, higher MCU clock frequency is required for higher sampling frequency. Table 3.2 (p. 7) provides a summary on clock frequency requirements.

Table 3.2. EFM32 clock frequency versus sampling frequency

	Narrowband (8 kHz)	Wideband (16 kHz)	Ultra-wideband (32 kHz)
EFM32 clock	8/16/32/48 MHz 8 MHz minimum	16/32/48 MHz 16 MHz minimum	48 MHz minimum

3.3.2 Bitrate and Compression Ratio

The Speex encoder has a parameter called “quality” that is used to control the bitrate. In this application note the Speex decoder supports Speex encoded data with quality 3 to 10 on narrowband, wideband and ultra-wideband as summarized in Table 3.3 (p. 7) (includes memory size for 1 minute voice).

Table 3.3. Quality vs bitrate and compression ratio

Quality	NB Bitrate	Compression ratio	WB Bitrate	Compression ratio	UWB Bitrate	Compression ratio
3	8,000	16:1 (1min-58.6KB)	9,800	26.12:1 (1min-58.6KB)	11,600	44.13:1 (1min-85KB)
4	8,000	16:1 (1min-58.6KB)	12,800	20:1 (1min-73.2KB)	14,600	35.06:1 (1min-108.4KB)
5	11,000	11.63:1 (1min-82KB)	16,800	15.23:1 (1min-123KB)	18,600	27.52:1 (1min-137.7KB)
6	11,000	11.63:1 (1min-82KB)	20,600	12.42:1 (1min-152.3KB)	22,400	22.85:1 (1min-164.1KB)
7	15,000	8.53:1 (1min-111.3KB)	23,800	10.75:1 (1min-175.8KB)	25,600	20:1 (1min-187.5KB)
8	15,000	8.53:1 (1min-111.3KB)	27,800	9.2:1 (1min-205KB)	29,600	17.29:1 (1min-216.8KB)
9	18,200	7.03:1 (1min-134.8KB)	34,200	7.48:1 (1min-252KB)	36,000	14.22:1 (1min-263.7KB)
10	24,600	5.2:1 (1min-181.6KB)	42,200	6.06:1 (1min-310.5KB)	44,000	12.8:1 (1min-325.2KB)

3.3.3 Memory Footprint

The FLASH and RAM usage depends on supported frequency band and is summarized in Table 3.4 (p. 7) (compiled with “Optimizations at High Speed” in IAR V6.40.2). These figures include Speex decoder and hardware drivers.

Table 3.4. Memory footprint of Speex decoder

Mode	Flash (bytes)	RAM (bytes)
NB8K	28,128	4,920
NB	34,332	4,924
NB & WB	43,728	6,592
NB & WB & UWB	43,960	9,920

3.4 Current Consumption

The Speex decoder is compiled with “Optimizations at High Speed” in IAR V6.40.2 and current consumption is measured with energyAware Profiler V1.01. Table 3.5 (p. 8) shows average current consumption of EFM32GG and EFM32G performing Speex decoding at different frequency bands. EFM32G is measured with 8 MHz and 16 MHz crystal whereas EFM32GG is measured with 48 MHz crystal (prescaling to 8 MHz and 16 MHz).The current consumption of EFM32GG should be close to EFM32G if 8 MHz or 16 MHz clock is not prescaled from 48 MHz but rather implemented with 8 MHz or 16 MHz crystals. Note that these current numbers also include peripheral consumption from DACs etc.

Table 3.5. Average current consumption on Speex decoding at different frequency bands

	NB	WB (play NB)	UWB (play NB)	WB	UWB (play WB)	UWB
MCU Clock	8 MHz	16 MHz	48 MHz	16 MHz	UWB play WB	UWB
EFM32G	2.2mA	2.9mA	-	4.9mA	-	-
EFM32GG	2.9mA	3.8mA	7mA	5.8mA	9.6mA	12.8mA

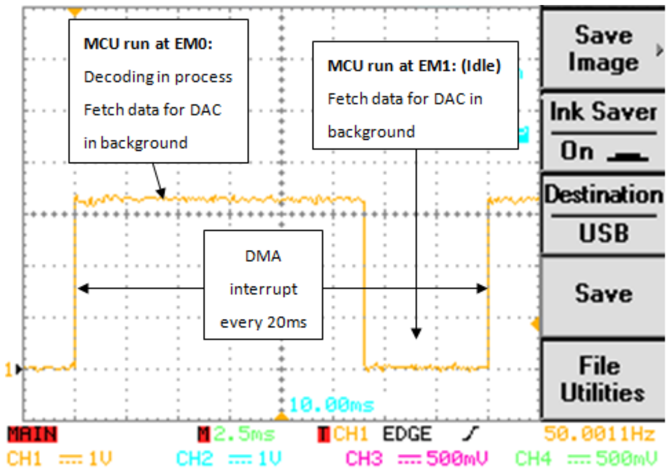
3.5 MCU Loading Analysis

MCU loading analysis is sometimes required to check whether the selected MCU clock frequency is fast enough for Speex decoding. To investigate the MCU loading of Speex decoder, it needs to toggle one GPIO on start and end of decoding in speexfunc.c.

3.5.1 Speex Decoding NB

The MCU loading is ~70% with 8 MHz clock and PREFETCH disabled.

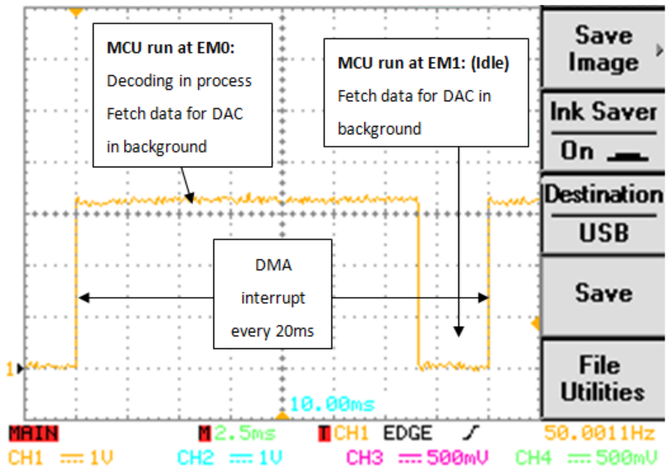
Figure 3.2. MCU loading at NB decoding



3.5.2 Speex Decoding WB

The MCU loading is ~82.5% with 16 MHz clock and PREFETCH disabled.

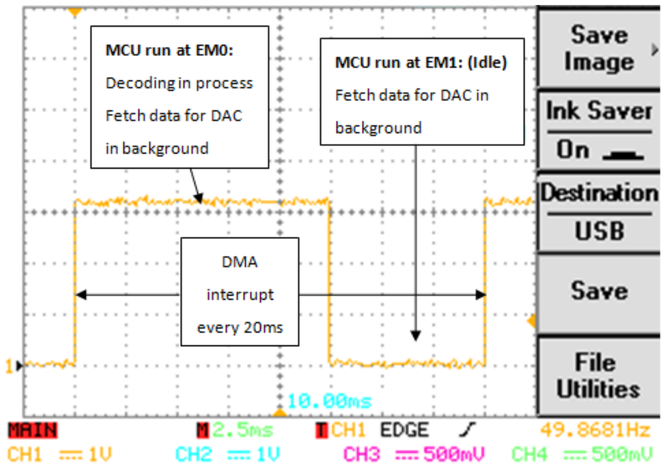
Figure 3.3. MCU loading at WB decoding



3.5.3 Speex Decoding UWB

The MCU loading is ~62.5% with 48 MHz clock and PREFETCH disabled.

Figure 3.4. MCU loading at UWB decoding



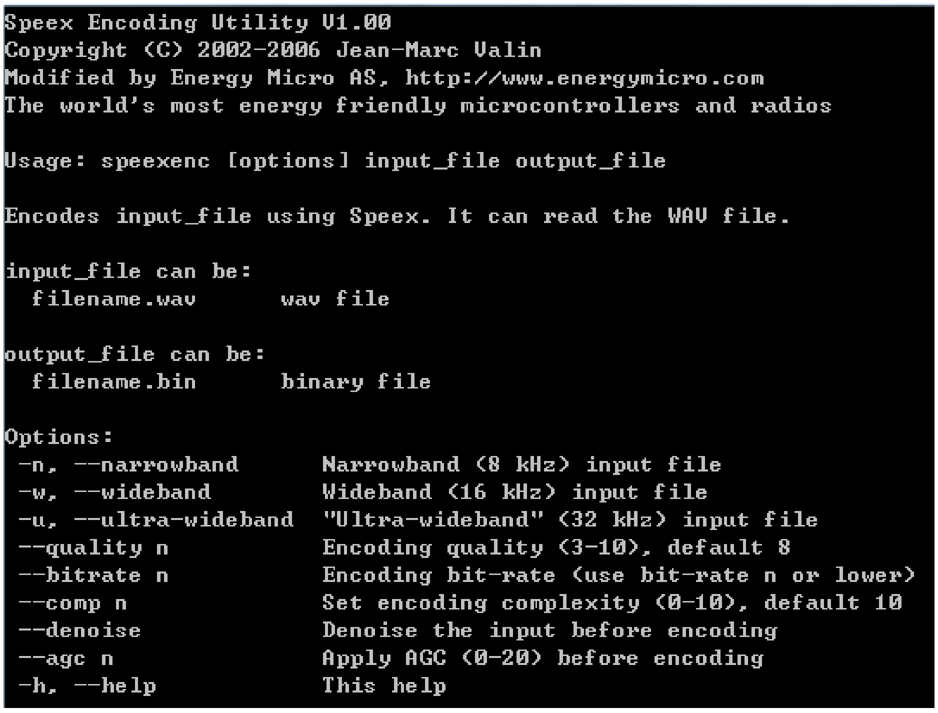
4 Speex Encoding Utility (PC Application)

In order to save Speex encoded data to internal flash for voice playback, a PC application is required to convert a WAV file into a header file for the compiler to work with. The **speexenc.exe** in “pc_software” folder is used for this purpose.

The Speex encoding utility is modified from speexenc.c that can be downloaded from **http://www.speex.org/downloads/**. This PC application will convert the input WAV file into Speex encoded format binary file and C header file in one step. The sampling frequency of input WAV file must be 8 kHz, 16 kHz or 32 kHz in MONO format.

To run the PC application, change the folder path to “pc_software” in DOS command line and then type speexenc.

Figure 4.1. Screenshot of Speex encoding utility



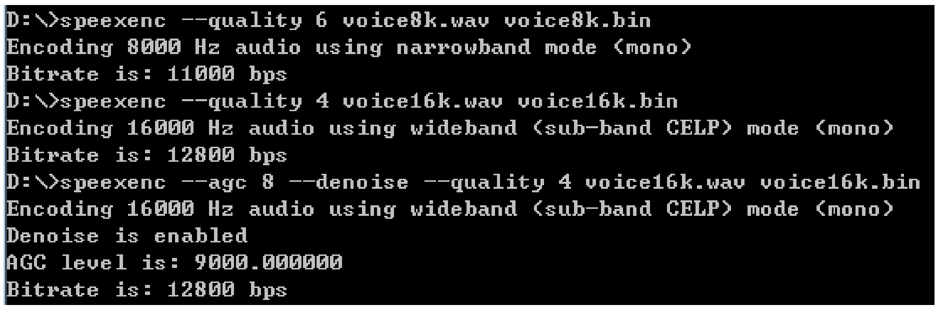
```
Speex Encoding Utility V1.00
Copyright (C) 2002-2006 Jean-Marc Valin
Modified by Energy Micro AS, http://www.energymicro.com
The world's most energy friendly microcontrollers and radios

Usage: speexenc [options] input_file output_file

Encodes input_file using Speex. It can read the WAV file.

input_file can be:
  filename.wav       wav file

output_file can be:
  filename.bin       binary file

Options:
 -n, --narrowband       Narrowband (8 kHz) input file
 -w, --wideband         Wideband (16 kHz) input file
 -u, --ultra-wideband   "Ultra-wideband" (32 kHz) input file
 --quality n            Encoding quality (3-10), default 8
 --bitrate n            Encoding bit-rate (use bit-rate n or lower)
 --comp n               Set encoding complexity (0-10), default 10
 --denoise              Denoise the input before encoding
 --agc n                Apply AGC (0-20) before encoding
 -h, --help             This help
```

The PC application automatically detects the sampling frequency of the input WAV file and then encodes it to Speex format based on the specified quality factor. In according to Table 3.3 (p. 7) , user can select desired bitrate with corresponding quality factor, normally only options --quality, --agc and --denoise are required.

Examples:

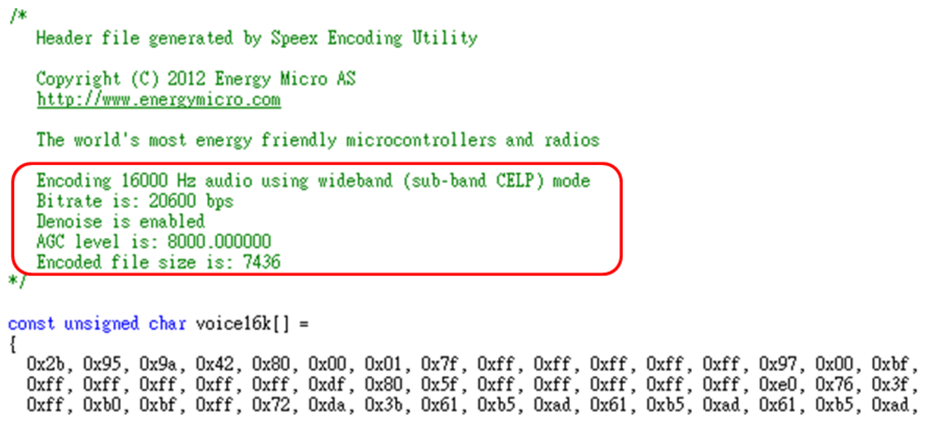
- Encode 8 kHz WAV file to 11 kbps (quality 6) Speex format C header file.
- Encode 16 kHz WAV file to 12.8 kbps (quality 4) Speex format C header file.
- Encode 16 kHz WAV file to 12.8 kbps (quality 4) Speex format C header file with AGC and Denoise enable.

Figure 4.2. Screenshot of examples



```
D:\>speexenc --quality 6 voice8k.wav voice8k.bin
Encoding 8000 Hz audio using narrowband mode (mono)
Bitrate is: 11000 bps
D:\>speexenc --quality 4 voice16k.wav voice16k.bin
Encoding 16000 Hz audio using wideband (sub-band CELP) mode (mono)
Bitrate is: 12800 bps
D:\>speexenc --agc 8 --denoise --quality 4 voice16k.wav voice16k.bin
Encoding 16000 Hz audio using wideband (sub-band CELP) mode (mono)
Denoise is enabled
AGC level is: 9000.000000
Bitrate is: 12800 bps
```

Figure 4.3. Example of Speex format C header file



```
/*
   Header file generated by Speex Encoding Utility

   Copyright (C) 2012 Energy Micro AS
   http://www.energymicro.com

   The world's most energy friendly microcontrollers and radios

   Encoding 16000 Hz audio using wideband (sub-band CELP) mode
   Bitrate is: 20600 bps
   Denoise is enabled
   AGC level is: 8000.000000
   Encoded file size is: 7436
*/

const unsigned char voice16k[] =
{
  0x2b, 0x95, 0x9a, 0x42, 0x80, 0x00, 0x01, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff, 0x97, 0x00, 0xbf,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xdf, 0x80, 0x5f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0, 0x76, 0x3f,
  0xff, 0xb0, 0xbf, 0xff, 0x72, 0xda, 0x3b, 0x61, 0xb5, 0xad, 0x61, 0xb5, 0xad, 0x61, 0xb5, 0xad,
```

A free WAV file sampling rate conversion software can be downloaded from:

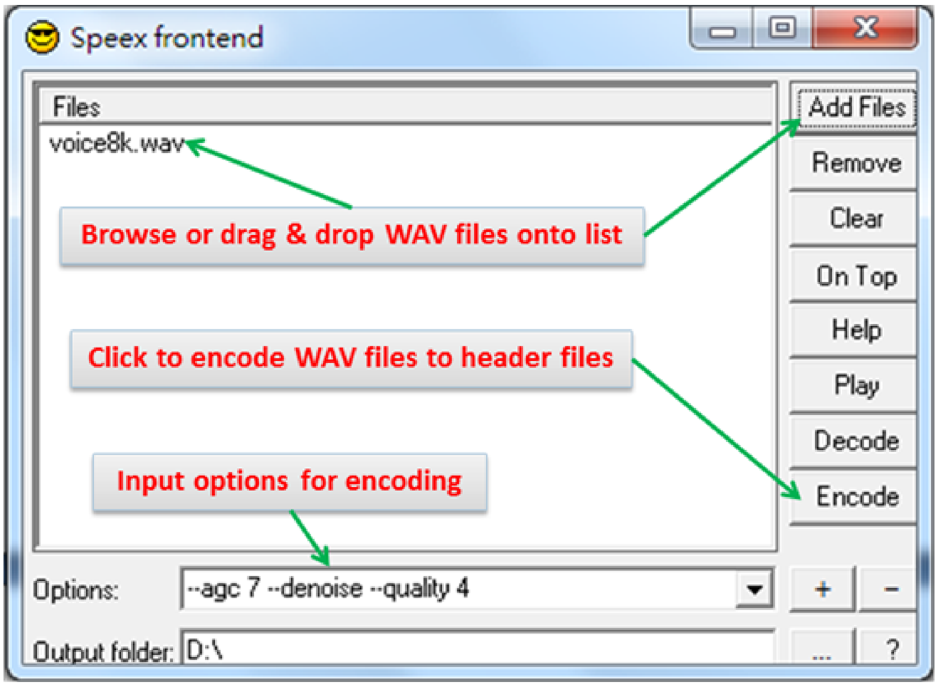
http://www.voxengo.com/product/r8brain/.

A free media (e.g. MP3 to WAV) and WAV file sampling rate converter can be downloaded from:

http://www.formatoz.com/.

In addition to DOS command line, the Speex encoding utility can run under Speex frontend as a Windows application. Put **speexenc.exe** to the folder that contains **Speex frontend.exe** and **win2dos.exe**; then double click **Speex frontend.exe** to start the application. The Speex frontend is a freeware and can be downloaded from: **http://members.home.nl/w.speek/speex.htm**.

Figure 4.4. Screenshot of Speex frontend with Speex encoding utility


Speex frontend
Files
voice8k.wav
Browse or drag & drop WAV files onto list
Click to encode WAV files to header files
Input options for encoding
Add Files
Remove
Clear
On Top
Help
Play
Decode
Encode
+
-
...
?

Options: --agc 7 --denoise --quality 4

Output folder: D:\

5 Speex API

The major APIs (in speexfunc.c) for this application note are used for Speex encoder and decoder as described below.

5.1 Speex Encoder

The function speexRecordFlash512() is used to record encoded Speex frames to the upper 512 KB flash section (EFM32GGxxxF1024).

void speexRecordFlash512(uint32_t complexity, uint32_t numOfFrames)

Parameters:

- complexity - Defines complexity of encoder (1 or 2)
- numOfFrames - Number of frames to record

Return:

- None

5.2 Speex Decoder

The function speexPlayBack() is used to play encoded Speex frames from flash (either recorded or pre-compiled).

void speexPlayBack(bool flashRecord, uint8_t freqBand, uint8_t encFrameSize, struct speexEncFile *ptrStruct)

Parameters:

- flashRecord - Playback from recording if true. Playback from pre-compiled voice if false.
- freqBand – Frequency band (NB or WB or UWB) of encoded data
- encFrameSize – Frame size of encoded data
- *ptrStruct – Pointer to structure of play list

Return:

- None

The list structure is defined in speexfunc.h. The frameStart variable is a pointer to encoded voice data for playback and the frameNum variable is the number of frames of the encoded voice data:

```
struct speexEncFile { char *frameStart; uint16_t frameNum; };
```

An array of speexEncFile structure (playList) is defined in main_xxxx.c so the speexPlayBack() function can play multiple encoded voice files.

```
struct speexEncFile playList[LIST_SIZE];
```

The frameStart variable is set to NULL at the end of the list so the maximum voice files that can be played is equal to LIST_SIZE minus 1. The encoded frame size of different sampling frequencies and bitrates are defined in config.h (FRAME_SIZE_XXXX). For recorded data, sampling frequency and bitrate are fixed at 8 kHz and 8 kbps. For pre-compiled data, sampling frequency and bitrate are described in the header file as in Figure 4.3 (p. 11) .

5.2.1 Examples

5.2.1.1 NB8K

Sample frequency and bitrate are constant and can only play 8 kbps NB data.

playList[0].frameStart = (char *)voice8k;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice8k)/FRAME_SIZE_8K;

speexPlayBack(0, NARROWBAND8K, FRAME_SIZE_8K, &playList[0]);

5.2.1.2 NB

Sample frequency is constant and bitrate is configurable. Can play different NB data by modifying the variables below:

playList[0].frameStart = (char *)voice11k;

playList[1].frameStart = NULL;

frameNum = sizeof(voice11k)/FRAME_SIZE_11K;

speexPlayBack(0, NARROWBAND, **FRAME_SIZE_11K**, &playList[0]);

5.2.1.3 WB

Sample frequency and bitrate is configurable. Can play different NB and WB data by modifying the variables below:

Wideband to play narrowband voice file:

playList[0].frameStart = (char *)voice11k;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice11k)/FRAME_SIZE_11K;

speexPlayBack(0, **NARROWBAND**, **FRAME_SIZE_11K**, &playList[0]);

Wideband to play wideband voice file:

playList[0].frameStart = (char *)voice9k8;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice9k8)/FRAME_SIZE_9K8;

speexPlayBack(0, **WIDEBAND**, **FRAME_SIZE_9K8**, &playList[0]);

5.2.1.4 UWB

Sample frequency and bitrate is configurable. Can play different NB, WB and UWB data by modifying the variables below:

Ultra-wideband to play narrowband voice file:

playList[0].frameStart = (char *)voice11k;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice11k)/FRAME_SIZE_11K;

speexPlayBack(0, **NARROWBAND, FRAME_SIZE_11K**, &playList[0]);

Ultra-wideband to play wideband voice file:

playList[0].frameStart = (char *)voice9k8;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice9k8)/FRAME_SIZE_9K8;

speexPlayBack(0, **WIDEBAND**, **FRAME_SIZE_9K8**, &playList[0]);

Ultra-wideband to play ultra-wideband voice file:

playList[0].frameStart = (char *)voice11k6;

playList[1].frameStart = NULL;

playList[0].frameNum = sizeof(voice11k6)/FRAME_SIZE_11K6;

speexPlayBack(0, **ULTRAWIDEBAND**, **FRAME_SIZE_11K6**, &playList[0]);

5.2.1.5 Play multiple files

Narrowband example:

encodedFrameSize = FRAME_SIZE_11K;

playList[0].frameStart = (char *)one8k;

playList[1].frameStart = (char *)two8k;

playList[2].frameStart = (char *)three8k;

playList[3].frameStart = (char *)four8k;

playList[4].frameStart = (char *)NULL;

playList[0].frameNum = sizeof(one8k)/FRAME_SIZE_8K;

playList[1].frameNum = sizeof(two8k)/FRAME_SIZE_8K;

playList[2].frameNum = sizeof(three8k)/FRAME_SIZE_8K;

playList[3].frameNum = sizeof(four8k)/FRAME_SIZE_8K;

speexPlayBack(0, **NARROWBAND**, **FRAME_SIZE_8K**, &playList[0];

6 Software Examples

This application note includes software examples that demonstrate how to use the Speex encoder and decoder with the EFM32GG-DK3750. It is possible to run this software on an EFM32GG-STK3700, by connecting additional hardware to support audio input and a speaker (including external amplifier) to play the output sound. The software examples includes support for the STK, which can be enabled by adding a defined symbol called GG_STK either at the start of main_xxxx.c or in the project defines. The audio output will be played on PB11 (EXP port pin 16) on GG STK, while the AUDIO_OUT jack will be used for the GG_DK. The software examples are also available as pre-compiled binaries in the "binaries" folder.

6.1 IDE Setup

The IDE settings as described below are required for IAR and Keil to compile and run the Speex encoder and decoder.

IAR:

- In the project options menu, select "C/C++ Compiler" and "Preprocessor". **Add HAVE_CONFIG_H in the “Defined symbols:” box**. This is used for Speex library to access config.h.
- In the project options menu, select "C/C++ Compiler" and "Diagnostics". **Add Pa050 in the “Suppress these diagnostics:” box**. This is used to suppress the compile warning on voice header files that is generated by PC encoding utility in section 4.
- In the project options menu, select "C/C++ Compiler" and "Optimizations". **Select High and Speed**. This is used to optimize the code for high speed. High speed optimization is set by default the Release build, but not in the Debug build in the supplied IAR projects.
- In the project options menu, select "Linker" and “Extra Options”. **Check “Use command line options” box and add --basic_heap in the “Command line options:” box**. This is used to keep compatibility of different IAR versions on heap usage. Note that this option is not set by default in the supplied IAR projects.

Keil:

- In the project options menu, select "C/C++". **Add HAVE_CONFIG_H in the “Preprocessor Symbols:” box**. This is used for Speex library to access config.h.
- **Check “Optimize for time” box and select Level 3 (-O3)**. This is used to optimize the code for high speed.

The linker files for IAR in “iar_linker_files” folder and startup files for Keil in “arm_startup_files” folder are used to define heap size that used on Speex encoder and decoder (Table 6.1 (p. 16)). The stack size is set to 0x600 for all projects.

Table 6.1. Heap size of IAR linker files and Keil startup files

Project	IAR linker file	Heap size	Keil startup file	Heap size
NB8K Codec	EFM32GG990F1024CODEC.icf	0x1600	startup_efm32ggCodec.s	0x2000
NB8K and NB Decoder	EFM32GG990F1024NB.icf	0x0C00	startup_efm32ggNBWB.s	0x1000
WB Decoder	EFM32GG990F1024WB.icf	0x1000	startup_efm32ggNBWB.s	0x1000
UWB Decoder	EFM32GG990F1024UWB.icf	0x1800	startup_efm32ggUWB.s	0x1800

6.2 Speex Encoder

The Speex encoder example records the audio signal from the AUDIO_IN jack, encodes into Speex format and saves it to internal flash. Operation details are shown in Table 6.2 (p. 17) and Table 6.3 (p. 17) .

Table 6.2. Operation of Speex encode example on EFM32GG-STK3700

Key	LED 1	LED 0	Input and Output	NB8K Codec
PB0	On	Off	PD6 (EXP port pin 16)	Record to flash then playback
PB1	Off	On	PB11 (EXP port pin 11)	Play recording from flash

Table 6.3. Operation of Speex encode example on EFM32GG-DK3750

Key	User LEDs	Input and Output	NB8K Codec
PB1	0 On	AUDIO IN	Record to flash then playback
PB2	8 On	AUDIO OUT	Play recording from flash
PB3	1 On	AUDIO OUT	Play 8 kbps voice

6.3 Customization Options for Encoder

Each project has its corresponding configuration file to setup the software and hardware environment. The Speex codec provides customizations on usage of ADC, crystal frequency, DAC, DMA, FLASH, PRS and TIMER in config.h. The config.h of Speex codec is placed in “config_codec_nb8k” folder.

Table 6.4 (p. 17) shows the ADC and FLASH parameters for Speex encoder, other parameters are described in Table 6.7 (p. 19) for 8 kbps Speex decoder. Parameters in Table 6.4 (p. 17) and Table 6.7 (p. 19) can be configured by the user; other parameters should be kept unchanged.

Table 6.4. Parameters for Speex encoder

Parameters	Usage	Default value
FLASHSTART	Set Flash record start address (>=512 KB aligned with 4 KB)	0x80000 (512 KB)
RECORD_FRAMES	Number of Speex frames to record (1 frame = 20ms)	600 (12s)
FRAME_MAXIMUM	Maximum frames depends on FLASHSTART	26112
ADC_CHANNEL	ADC channel for audio input	adcSingleInpCh6 (right channel of AUDIO IN)
ADC_PRS_CH ADC_PRS_SEL	PRS channel for ADC trigger (same channel as DAC)	PRS channel 0 adcPRSSELCh0
ADC_CLOCK	ADC clock frequency	12000000
ADC_REF_SELECT	0 – 1.25V as ADC reference 1 – 2.5V as ADC reference 2 – VDD as ADC reference	1
ADC_OVS_16	0 – Oversampling disabled 1 – Oversampling to 16-bit results	1

6.4 Speex Decoder

The Speex decoder examples (NB8K, NB, WB and UWB) fetch encoded Speex data from flash, decode it into a digital audio signal which is output to the AUDIO_OUT jack.

- The WAV file “Your blood glucose test result is below” is encoded to different bitrates with 8 kHz, 16 kHz and 32 kHz sampling frequency.
- The header files in “voice” folder are named by bitrate, for example, voice16k8.h means this header file is encoded at quality 5 - wideband 16.8 kbps (see Table 3.3 (p. 7)).
- PB0 in GG STK is used to select bitrate for playback, operation details are shown in Table 6.5 (p. 18) and Table 6.6 (p. 18) .

Table 6.5. Operation of Speex decoder examples on EFM32GG-STK3700

Key	LED 1	LED 0	NB8K	NB	WB	UWB
PB1	Off	Off	8 kbps	8 kbps	9.8 kbps	11.6 kbps
PB1	Off	On	-	11 kbps	12.8 kbps	14.6 kbps
PB1	On	Off	-	15 kbps	16.8 kbps	18.6 kbps
PB1	On	On	-	18.2 kbps	20.6 kbps	22.4 kbps

Table 6.6. Operation of Speex decoder examples on EFM32GG-DK3750

Key	LEDs	NB8K	NB	WB	UWB
PB1	0 On	8 kbps	8 kbps	9.8 kbps	11.6 kbps
PB2	1 On	-	11 kbps	12.8 kbps	14.6 kbps
PB3	2 On	-	15 kbps	16.8 kbps	18.6 kbps
PB4	3 On	-	18.2 kbps	20.6 kbps	22.4 kbps

WARNING

Do not attach or use headphones with AUDIO OUT. Exposure to loud noises from any source for extended periods of time may temporarily or permanently affect your hearing. The louder the volume sound level, the less time is required before your hearing could be affected. Hearing damage from loud noise is sometimes undetectable at first and can have accumulative effect.

6.5 Customization Options for Decoder

Each project has its corresponding configuration file to setup the software and hardware environment. The Speex decoder provides customizations on usage of DAC, crystal frequency, DMA, PRS and TIMER in config.h. The config.h is placed in “config_xxxxx” folder for different projects. Parameters in Table 6.7 (p. 19) can be configured by the user; other parameters should not be changed.

Table 6.7. Parameters for Speex decoder

Parameters	Usage	Default value
PRI_ADDRESS	DMA primary data structure address (must be at 0xXXXXXX00)	0x20001000
ALT_ADDRESS	DMA alternate data structure address (must be at 0xXXXXXX00 + 0x100)	0x20001100
SPEEX_HFXO_FREQ	EFM32GG crystal frequency	48000000
LIST_SIZE	Maximum voice header files that can play at one time	2 (Codec) or 10 (Decoder)
DAC_CHANNEL DAC_CHDATA DAC_CHDMA DAC_CHCTRL DAC_CHCTRLEN DAC_CHPRSEN	DAC channel for audio output	DAC0 channel 0 (right channel of AUDIO OUT)
DAC_PRS_CH DAC_PRS_SEL	PRS channel for DAC trigger (same channel as ADC)	PRS channel 0 dacPRSSELCh0
DAC_CLOCK	DAC clock frequency	1000000
DAC_REF_SELECT	DAC reference select 0 – 1.25V (lowest output) 1 – 2.5V 2 – VDD (highest output)	0
TIMER_USED TIMER_CLK TIMER_PRS TIMER_SRC	Select TIMER for PRS source	TIMER1
SAMPLE_8K SAMPLE_16K SAMPLE_32K	TIMER TOP value for 8/16/32 kHz sampling frequency	Band dependent
BASE_FREQ	TIMER base frequency to setup TOP value	Band dependent

6.6 Software Examples for Other EFM32s

Except for the Speex encoder and UWB decoder that require EFM32 run on 48 MHz, other Speex decoder examples (NB8K, NB and WB) can easily be ported to other EFM32s running at lower frequencies as described below:

- Change ALT_ADDRESS in config.h for new family (e.g. PRI_ADDRESS + 0x00000080 for Gecko)
- Change SPEEX_HFXO_FREQ in config.h to HFXO frequency of new family (e.g. 32000000)
- Migrate project settings from GG to new family
- Create IAR linker file and Keil startup file in according to current heap and stack size setting
- Modify main_xxxx.c and gpio.c for new family

7 Conclusion

The EFM32 Speex encoder and decoder are suitable for applications that require low power voice record and/or playback function. Due to the high compression ratio of the Speex encoder, the scalable flash size (up to 1 MB) of EFM32 can fulfill various lengths of voice record and playback. External SPI flash for voice playback data can be eliminated to reduce current consumption (typically 10-17mA during flash read) and simplify the programming (no need to program external SPI flash, only one time programming for application software and voice data in EFM32) and manufacturing process. The rich peripherals of EFM32 (USB, TFT, LCD, AES…) also provide expansion possibilities to many applications that include voice record and/or playback functions.

8 Revision History

8.1 Revision 1.06

2014-05-07

Updated example code to CMSIS 3.20.5

Changed source code license to Silicon Labs license

8.2 Revision 1.05

2013-10-14

New cover layout

8.3 Revision 1.04

2013-05-08

Increased stack size from 0x400 to 0x600.

Moved IDE setup and customization options sections to the Software Examples chapter.

8.4 Revision 1.03

2012-12-20

Added Windows support (through Speex frontend) to Speex encoding utility (V1.01).

8.5 Revision 1.02

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

8.6 Revision 1.01

2012-10-12

Fixed typo in document.

Fixed error with playlist pointer in NB decoder and WB decoder examples code.

8.7 Revision 1.00

2012-10-04

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

A.2 Trademark Information

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

List of Tables

silabs.com

ZERO
ARM Cortex-M0+

TINY
ARM Cortex-M3

GECKO
ARM Cortex-M3

LEOPARD
ARM Cortex-M3

GIANT
ARM Cortex-M3

WONDER
ARM Cortex-M4