Freescale Semiconductor
Application Note

Document Number: AN4372
Rev 0, July 2014

Building a Speex Dictaphone Application on the Freescale KwikStik

by: Martin Hrnčárek
Application Engineer, Freescale
Rožnov
Czech Republic

Contents

1 Introduction

This document uses a demo to explain how the user can use the Speex software codec on the Kinetis ARM Cortex™ M4-based MCU to build a Dictaphone application.

Speex is an open source patent-free audio codec specially developed for speech encoding and decoding[1]. The main benefit of Speex is that the high compression ratio still yields acceptable sound quality. Therefore it is an excellent solution for applications that use message playback or voice recording. The Dictaphone software is managed by the Freescale MQX Real-Time Operating System (RTOS).

The hardware of this demo is based on the Freescale KwikStik (version 5), an ultra low-cost, all-in-one development tool for evaluating, developing, and debugging Kinetis MCUs. It contains the K40X256VLQ100 (144LQFP) MCU which has these features:

- USB
- On-board J-Link USB programmer
- Capacitive touch sensing

1. Please note that as of the date of publication of this application note, the Speex codec is no longer being actively supported, and its website directs new users to the website of the Opus codec. However, since there are still many users of Speex who expect to continue using it for the foreseeable future, we are releasing this document, but expect to release a new application note in the future that will demonstrate use of the Opus codec.

- Segment LCD
- Microphone
- 3.5 mm audio output jack
- Micro SD card slot functionality

The Speex SW codec with all its useful features is suitable for the following applications:

- Voice recorders
- Answering machines
- Smart medical appliances
- VoIP telephony
- Safety systems
- Intercoms
- Walkie-talkies
- Electronic toys

2 Speex codec overview

The Speex codec is a patent- and royalty-free, open-source software targeted at human voice compression and decompression, based on CELP (code-excited linear prediction) and designed to compress speech at bit-rates from 2 to 44 kbps. The Speex codec is designed to be very flexible and support a wide range of speech qualities and bit-rates. Support for very good speech quality also means that Speex can encode wideband speech (16 kHz sampling rate) in addition to narrowband speech (telephone quality, 8 kHz sampling rate).

The Speex codec includes the following key features:

- 8 kHz (narrowband), 16k Hz (wideband), and 32 kHz (ultra-wideband) compression with the same bitstream
- Noise suppression
- Acoustic-echo canceller
- Fixed-point port
- Discontinuous transmission (DTX)
- Voice-activity detection (VAD)
- Variable bit-rate operation (VBR)
- Packet-loss concealment
- Intensity-stereo encoding

Note that Speex has a number of features that are not present in other codecs, such as intensity-stereo encoding, integration of multiple sampling rates in the same bitstream (embedded coding), and a VBR mode.

3 Speex Dictaphone functionality

The software of the demo application is based on the block diagram shown in Figure 1.

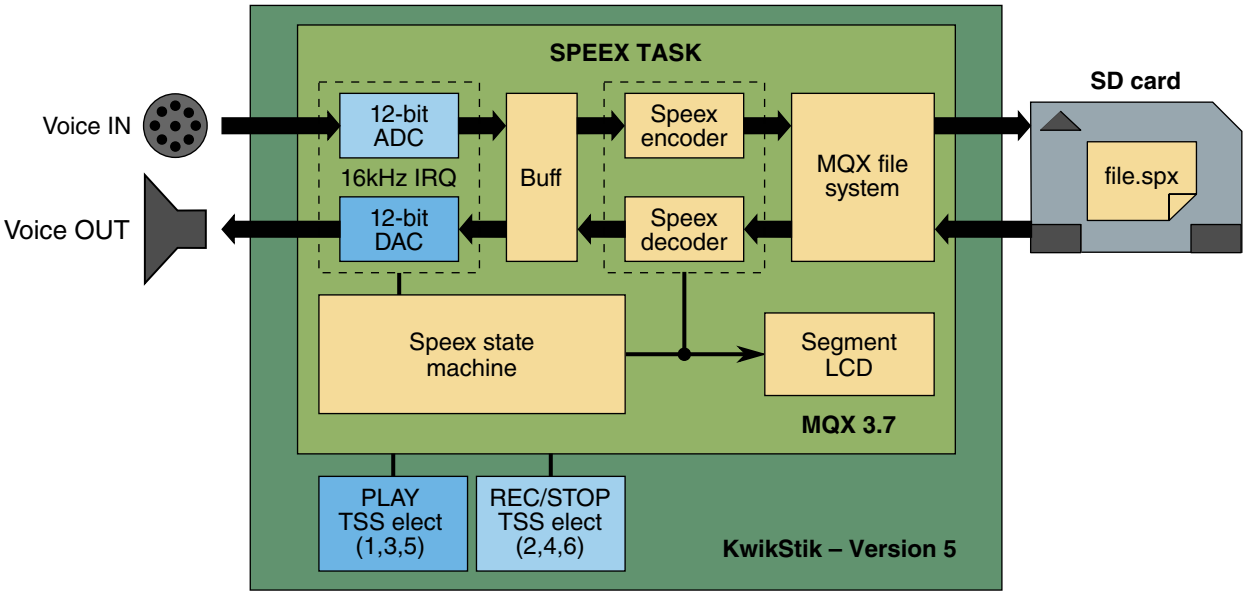

Figure 1. Speex Dictaphone block diagram

Figure 2 shows the KwikStik. The application is controlled by capacitive (TSS library) buttons located on both sides of the KwikStik. Touch buttons (E2, E4, E6) on the right side are used for the STOP/RECORD functions. The buttons on the other side are used for the PLAY function.

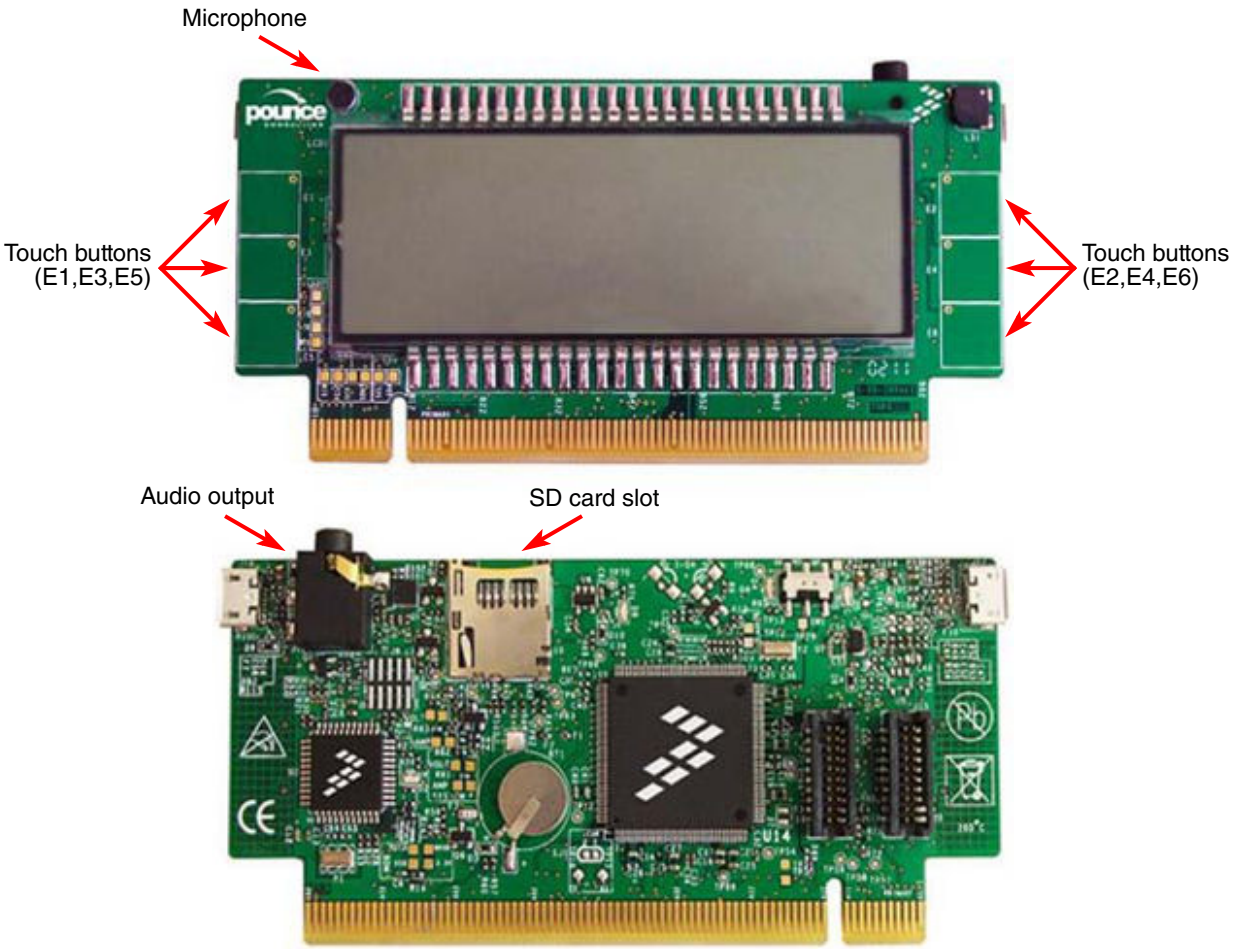

Figure 2. Top and bottom view of the KwikStik

Recording

Voice messages are recorded using the ADC and stored on an attached SD card after compression. After the recording process is complete, the message can be replayed. During recording, the application software fills one of the input audio buffers with audio samples acquired by the ADC in 12-bit mode. The application was designed to use two buffers for both decoding and encoding. When the input audio buffer is filled with raw audio samples, they are passed on to the Speex encoder for processing. The compressed audio data is stored on a file placed on the SD card.

Figure 3 shows the flow diagram for recording.

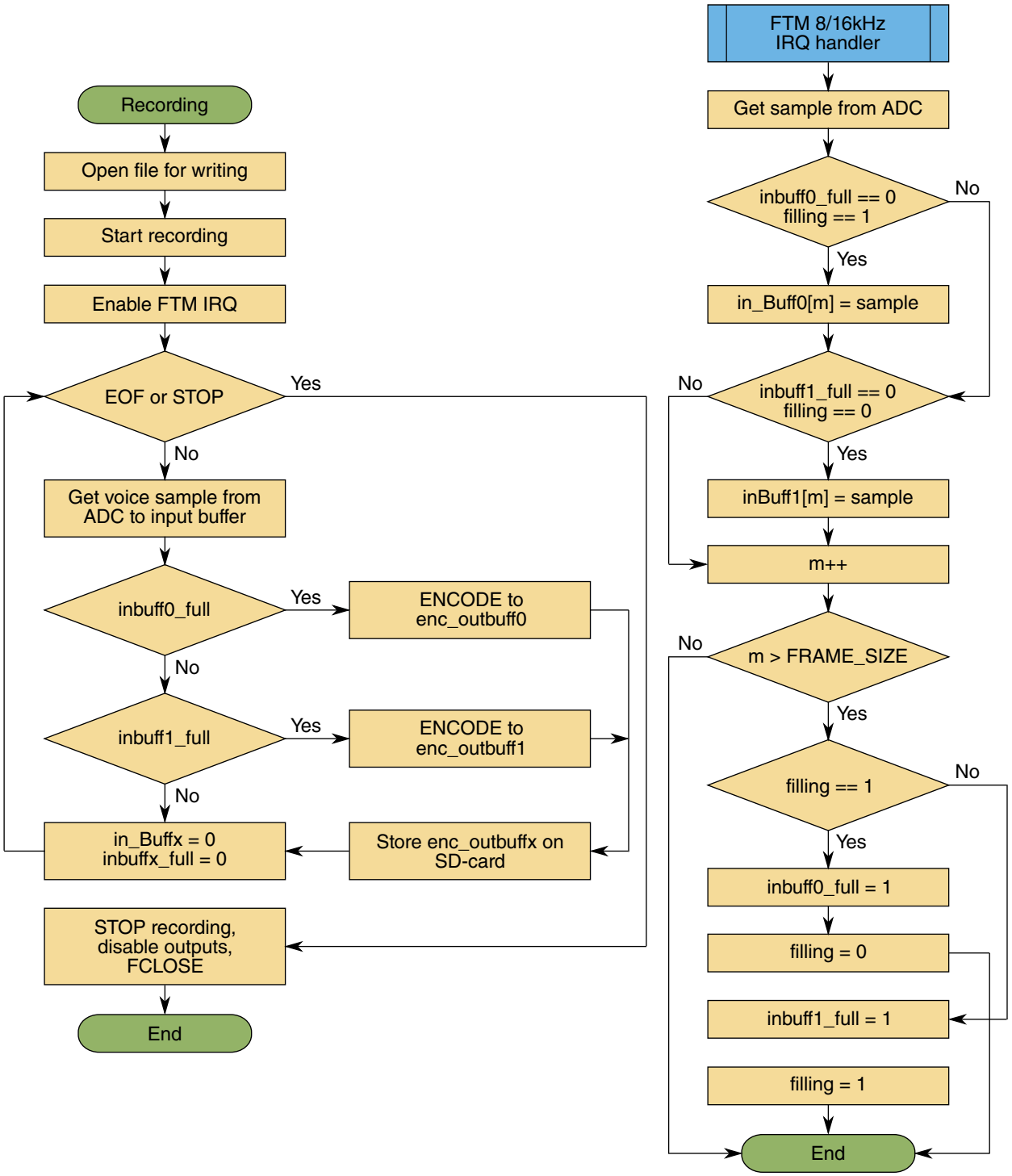

Figure 3. Recording block diagram

Playback

During message playback, the recording scenario is reversed. The compressed audio data is read from the SD card file and then passed on to the Speex decoder for processing, where the decompressed audio samples are reproduced using the internal 12-bit DAC. Audio data-paths and file I/O operations are double-buffered to avoid the loss of frames. Once the first buffer (containing the processed audio data) has been played or recorded, the application switches to the second buffer and reuses the first one to decode/encode another frame.

Control of speech input (ADC) and output (DAC) is driven via the FTM interrupt handler, which is periodically fired based on the Speex mode used (wideband 16 kHz, narrowband 8 kHz). The maximum length of a stored voice message is limited in code to 60 minutes or by the size of the SD card used.

The Freescale MQX real-time operating system (RTOS) provides real-time performance within a small, configurable footprint. The easy-to-use API and out-of-box experience ensure that first-time RTOS users can start developing their applications on the day the software is installed. The MQX RTOS is targeted mainly at embedded applications supporting most of the Freescale 32-bit MCUs. By using MQX, we can run several parallel tasks on a single CPU core. Moreover, MQX includes a complete file system (MFS) and peripheral drivers, such as the SD card used in this demo application.

MFS is an embedded FAT file system compatible with the Microsoft Windows and MS-DOS file systems. It can format, read, write, and exchange files with any operating systems running a FAT-12, FAT-16, or FAT-32 file system. It is fully reentrant and uses the Freescale MQX RTOS file device driver to access disk devices. See http://www.freescale.com/mqx for detailed information on MQX.

Figure 4 shows the flow diagram for playback.

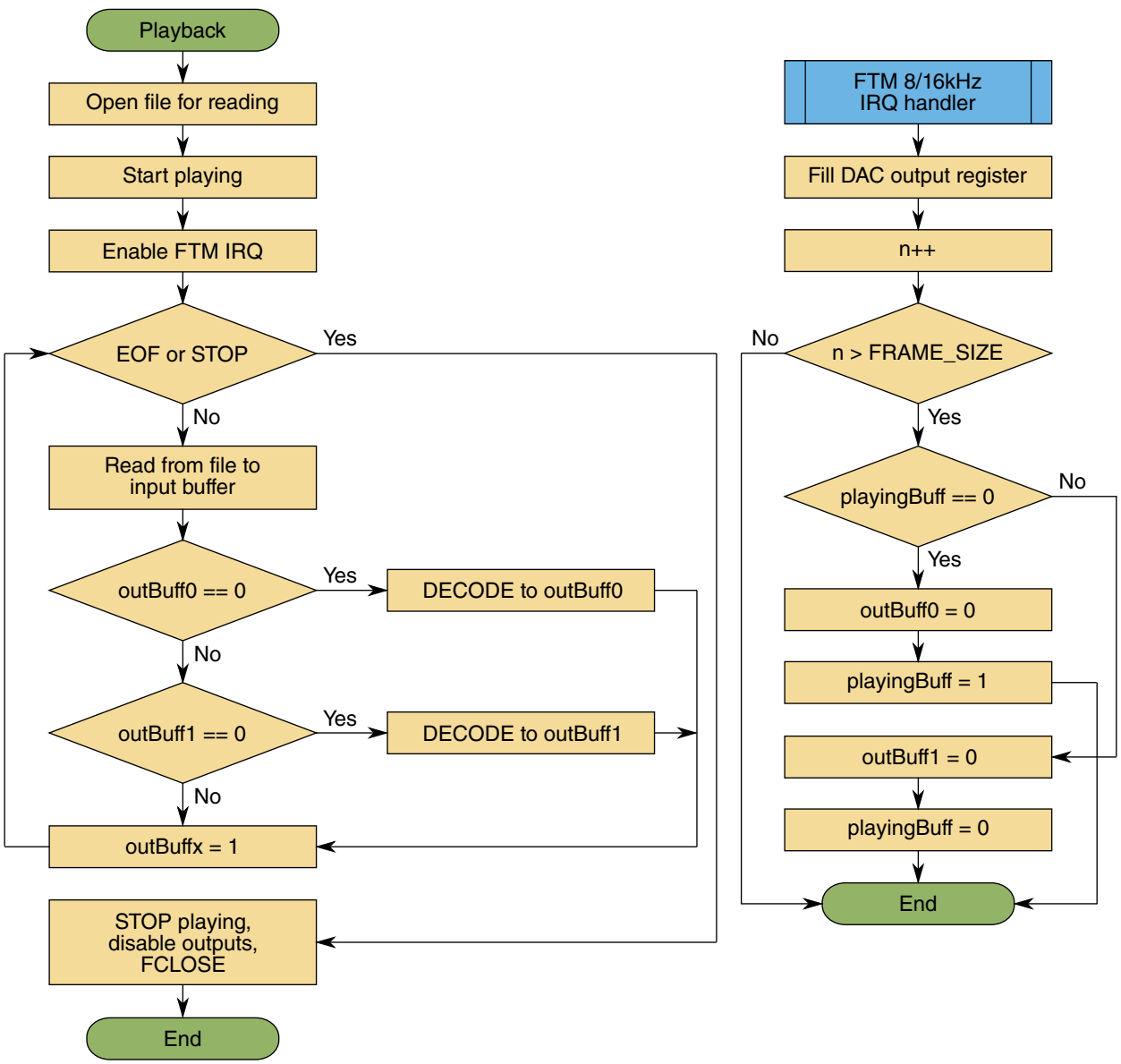

Figure 4. Playback flow diagram

4 Speex codec parameters and modes

The application supports the Speex narrowband mode (sampled at 8 kHz) as well as wideband mode (sampled at 16 kHz). The Speex codec has a few parameters that influence the quality of the reproduced sound that also influence the computing performance of the MCU. These parameters can be changed in the application with the macros shown in “Changing the default values,” but it is highly recommended to keep the default settings.

The following explanations of Speex parameters are quoted from *The Speex Codec Manual* Version 1.2 Beta 3.

- **Variable bit rate (VBR):** Variable bit-rate (VBR) allows a codec to change its bit-rate dynamically to adapt to the “difficulty” of the audio being encoded. In the example of Speex, sounds like vowels and high-energy transients require a higher bit-rate to achieve good quality, while fricatives (e.g. s,f sounds) can be coded adequately with less bits. For this reason, VBR can achive lower bit-rate for the same quality, or a better quality for a certain bit-rate. Despite its advantages, VBR has two main drawbacks: first, by only specifying quality, there’s no guaranty about the final average bit-rate. Second, for some real-time applications like voice over IP (VoIP), what counts is the maximum bit-rate, which must be low enough for the communication channel.
- **Quality (variable):** Speex is a lossy codec, which means that it achives compression at the expense of fidelity of the input speech signal. Unlike some other speech codecs, it is possible to control the tradeoff made between quality and bit-rate. The Speex encoding process is controlled most of the time by a quality parameter that ranges from 0 to 10. In constant bit-rate (CBR) operation, the quality parameter is an integer, while for variable bit-rate (VBR), the parameter is a float.
- **Complexity (variable):** With Speex, it is possible to vary the complexity allowed for the encoder. This is done by controlling how the search is performed with an integer ranging from 1 to 10 in a way that’s similar to the -1 to -9 options to gzip and bzip2 compression utilities. For normal use, the noise level at complexity 1 is between 1 and 2 dB higher than at complexity 10, but the CPU requirements for complexity 10 is about 5 times higher than for complexity 1. In practice, the best trade-off is between complexity 2 and 4, though higher settings are often useful when encoding non-speech sounds like DTMF tones.
- **Perceptual enhancement:** Perceptual enhancement is a part of the decoder which, when turned on, attempts to reduce the perception of the noise/distortion produced by the encoding/decoding process. In most cases, perceptual enhancement brings the sound further from the original objectively (e.g. considering only SNR), but in the end it still sounds better (subjective improvement).

Here is how to use the parameters.

Changing the default values

It is highly recommended to keep the default settings. Using wrong settings for these parameters can reduce or even ruin the quality of the sound processing.

Wideband

- `#define QUALITY 4`
- `#define COMPLEXITY 1`
- `#define VBR 0`
- `#define ENHANCEMENT 0`

Narrowband

- `#define QUALITY 3`
- `#define COMPLEXITY 2`
- `#define VBR 0`
- `#define ENHANCEMENT 0`

Because the VBR feature is implemented only for float arithmetic, it must be disabled through the following macro:

```
#define DISABLE_VBR
```

The Speex modes can be enabled through the following macros:

```
//#define NARROWBAND    // 8 kHz
#define WIDEBAND        // 16 kHz
```

Speex can be compiled for floating-point or fixed-point operation. KwikStik is based on the Kinetis K40 MCU without an FPU, so fixed-point arithmetic must be used. To disable the floating-point API and enable fixed-point, the following compiler macros must be defined:

```
#define DISABLE_FLOAT_API
#define FIXED_POINT
```

5 Optimizing the Speex codec

The computational requirement of Speex depends on the settings described in section 4. <<<verify numbering>>>The instruction set of the CPU used has a significant impact on the cycle count (execution time) of the processed code. Speex uses several math functions that consume most of the CPU cycles during encoding and decoding. The following functions are written in C and can be optimized for the chosen MCU architecture or the compiler used. This application was written on IAR 6.21. Thanks to high compiler optimization, optimal usage of the ARM Cortex™-M4 intrinsic functions are guaranteed. There is also the option to rewrite these functions in assembler or to replace with user functions. However, the same results must be achieved.

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

Memory requirements can be also optimized, mainly for usage in embedded systems. There are several ways to reduce Speex memory usage for code and data. Code size optimization is done by removing the following unused features:

- Wideband support
- Support for stereo (removing stereo.c)
- VBR support
- Static codebooks that are not needed for the bit rates that you are using (*_table.c files)

By redefining the NB_ENC_STACK and NB_DEC_STACK (or similar for wideband), it is possible to allocate memory only for a scenario that is known in advance. In this case, it is important to measure the amount of memory required for the specific sampling rate, bit rate, and complexity level being used. For more information, see the Speex manual [1].

6 Speex codec benchmark results

This section presents the results of the Speex audio codec performance using the Speex codec library on the ARM Cortex-M4 (K60N512 – 100 MHz). The library is written in pure C code.

How results were obtained

Voice input bit stream data[2] was encoded using the Speex codec algorithm. One frame of sampled data contained 160/320 PCM samples (20 ms of voice). The following parameters of the Speex codec for voice encoding were used:

- Quality = 3/4 (narrowband/wideband)
- Complexity = 1

Benchmark results

These parameters were used for this process:

2. Narrowband uses a 160 × 16-bit PCM data frame sampled at 8 kHz, input voice buffer consists of 10040 PCM samples; this means 1.255 s of voice. Wideband uses a 320 × 16-bit PCM data frame sampled at 16 kHz, input voice buffer consists of 91302 PCM samples; this means 5.7 s of voice.

- One frame of encoded data contained 20/25 data bytes.
- The resulting data compression was 1:16/1:26.
- This frame was consequently used for decoding.
- Decoded output data (160/320 PCM samples) can be reproduced by the PWM, DAC, TWR-AUDIO-SGTL (AN4369) modules.

Particular results of encoding and decoding of narrow/wideband using the Speex algorithm (such as memory size requirements[3], code performance) are shown in the following tables. These results were achieved using a high level[4] of C compiler optimization. Using this level of optimization, the compiler is able to use the DSP instruction-set effectively (an advantage of Cortex-M4 core).

Table 1. Speex codec requirements — narrowband

Process (20 ms voice data stream)	Program memory size (bytes)	RAM memory size (bytes)				CPU clock cycles	MIPS	CPU usage (% of 100 MHz)
		Stack	Heap	Main	Total			
Encoding	48776	1280	17664	2684	21628	585511	29, 3	29, 3
Decoding						99983	5, 0	5, 0

Table 2. Speex codec requirements — wideband

Process (20 ms voice data stream)	Program memory size (bytes)	RAM memory size (bytes)				CPU clock cycles	MIPS	CPU usage (% of 100 MHz)
		Stack	Heap	Main	Total			
Encoding	60424	1280	20480	3980	25740	844003	42, 2	42, 2
Decoding						215896	10, 8	10, 8

7 References

VALIN, Jean-Marc (2007, December 8). The Speex Codec Manual. Version 1.2 Beta 3. [downloaded 2011-11-01]. Retrieved from web site: http://www.speex.org/docs/manual/speex-manual.pdf

8 Revision history

This table summarizes changes to this document.

Revision number	Date	Description of changes
0	07/2014	Initial public release.

3. The input data buffer containing the voice samples was not included in the memory size estimation.
4. Level 4, speed optimized, loop unrolling must be disabled.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4372
Revision 0, July 2014