THIS SPEC IS OBSOLETE

Spec No: 002-04433

Spec Title: AN204433 FM3 MICROCONTROLLER - VOCODER APPLICATION USING SPEEX CODEC

Replaced by: None

AN204433

FM3 Microcontroller - VOCODER Application Using SPEEX CODEC

Associated Part Family:	Series Name	Product Number
	MB9B510A	MB9BF514A/515A/516A

A vocoder application digitizes an analog audio signal (in our case the human voice) and compresses the data via a codec. In this application the open-source Speex codec is used. The compressed data can be directly decompressed or decoded and output via DAC or PWM in the so-called “through mode” of a single system.

Contents

1 Introduction

1.1 About Document

A vocoder application digitizes an analog audio signal (in our case the human voice) and compresses the data via a codec. In this application the open-source Speex codec is used. The compressed data can be directly decompressed or decoded and output via DAC or PWM in the so-called “through mode” of a single system.

Another application is to transfer the compressed data from one system to another. If the remote system is built in the same way some kind of intercom can be set up, if the compressed data is sent from one system to another in a duplex way. This system works in a so-called “intercommunication mode” or “intercom mode”.

This application note describes a single “through mode” system first and later a “intercom” system.

2 The Speex Codec

The Speex Codec was primarily designed for Voice-Over-IP applications and is highly adapted for the human voice and speech. It uses the Code Excited Linear Prediction (CELP) method.

The Speex Codec consists of the following components:

1. Encoder
2. Decoder
3. Echo Cancellation
4. Jitter Buffer
5. Resampler

In this application note the items 1 and 2 are used and discussed.

Although Speex is able to offer much more features for more powerful systems than an embedded MCU, this application note uses a minimum feature set, but it produces very good sound results.

The application note's software package uses the Speex Codec version 1.2 Beta 3. The parameter, which are used are:

1. 8 kBit/s sampling and processing (encode and decode)
2. 160 samples per coding package (package duration: 20 ms)
3. No variable bit rate
4. Fixed-point calculation (Q15)

The Speex codec offers some preprocessor definitions to override some of the filter function, so that the programmer is able to adapt these functions for his processor system and thus to optimize the calculation time for speed reasons. The vocoder software explained here uses own optimized code for the following functions, which override the Speex codec functions by Speex preprocessor definitions:

1. filter_mem16()
2. iir_mem16()
3. vq_nbest()
4. inner_prod()
5. pitch_xcorr()

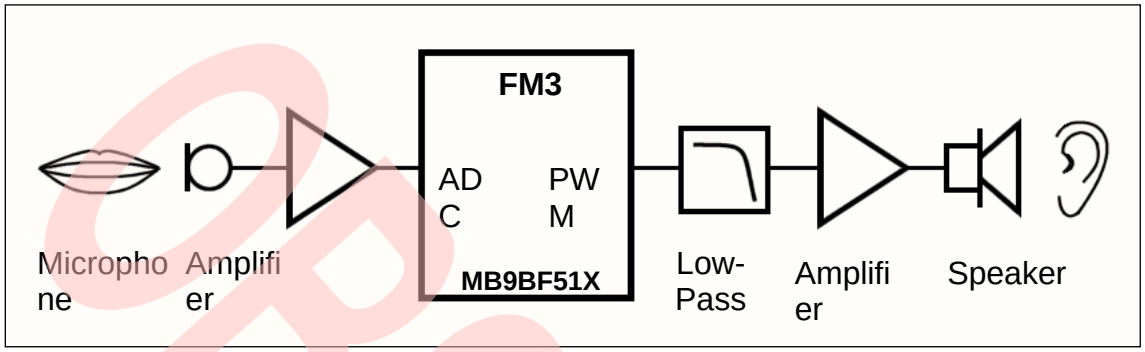
The optimizations were done by manual loop-unrolling and using the CORTEX-M3's native saturation instruction.

These optimizations saved about 1.5 ms per coding package of 160 samples.

3 Hardware

3.1 Through Mode Hardware

For using a single vocoder system the block schematic looks like the following illustration:

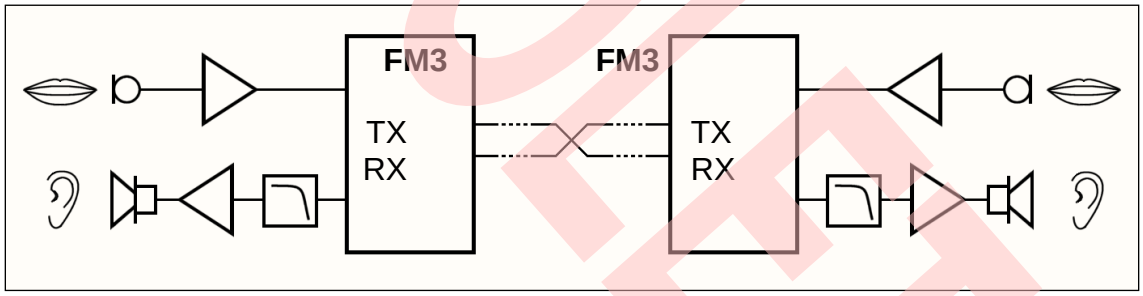


The amplifier should gain the speech signal between Vss and AVcc to use the full range of the ADC input. The low pass filter should have a cut-off frequency of 8 kHz to reduce the PWM carrier frequency sound.

The application itself samples the sound data via an ADC channel and performs a Speex encode. This encoded data is decoded and output via a PWM channel.

3.2 Intercom Mode Hardware

For using a two devices vocoder system the block schematic looks like the following illustration (Note that not described parts are the same as above):



In this configuration the encoded sound data is not transferred to the decoder but send via a communication device to a remote system and vice versa. Both systems must have the same software configuration and peripheral settings and the software of one system may be exactly the same, if the same peripherals and their input/output pins are used.

The application note's software uses a standard 8N1 UART protocol with 115.2 kBit/s and the MCUs are connected via a cross cable.

4 Software (Through Mode)

The software consists of the following parts

1. Low Level Library functions (L3, partly modified)
2. Speex Library
3. Application Files

The first part uses the original L3 modules for ADC (Analog Digital Converter), BT (Base Timer), and MFS (Multi-Function Serial Interface). The modules `interrupts.c/.h` and `l3.c/.h` are modified for the vocoder application.

The Speex Library was not modified. All necessary files for 8 kBit/s sampling are integrated as the original C code.

The third part is the application itself. The following list shows each module an explains the functionality briefly.

Module Name	Functionality
`cortex_filters.c`	Collection of M3 core optimized code for replacement of certain Speex filter functions
`init.c`	Initialization functions for the used peripherals
`main.c`	Main module, just contains a call to vocoder()
`vocoder.c`	Main application functions including interrupt callbacks from L3 functions.
`vocoder.h`	User settings for the application and L3 definitions
`config.h`	Speex user configuration file

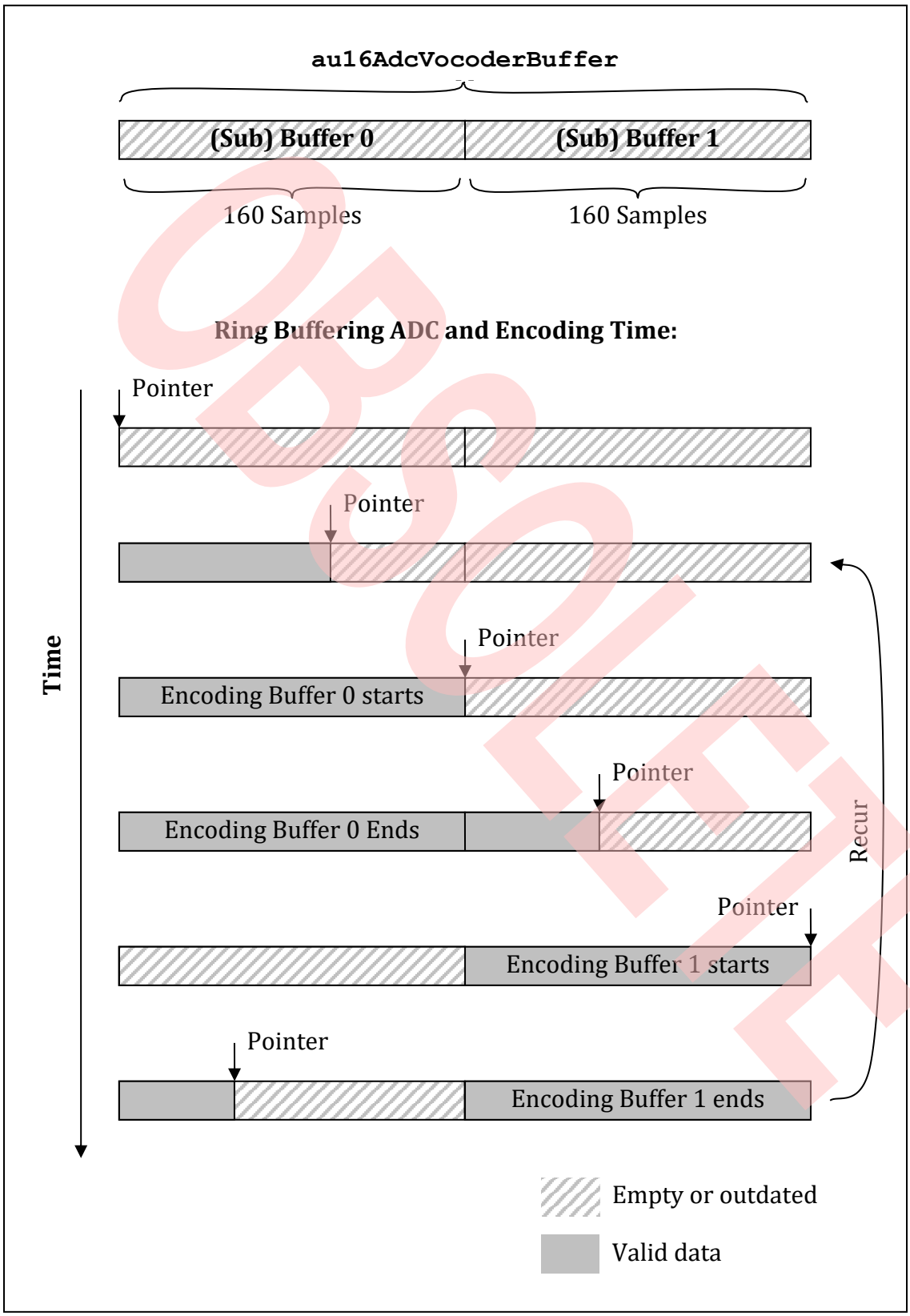
4.1 Sample Part

At the initialization time the ADC is initialized to approximately 1 µs conversion time for the bus clock frequency of 72 MHz. For sampling the voice data channel 0 (AN00) is selected.

The ADC itself is continuously triggered by the Base Timer 0 (BT0) with the sample rate of 8 kHz.

During runtime the sample data is stored via the continuous ring-buffering mode in the global array `au16AdcVocoderBuffer[VOCODER_ADC_BUFFER_SIZE]`, where `VOCODER_ADC_BUFFER_SIZE` is the double size of the Speex definition `FRAME_SIZE`, which is `160` for 8 kBit/s. The buffer size is doubled, so that after the first 160 samples, the encoding can be started and the rest of the buffer is filled consecutively to the end after it is reset to 0, so that the 2nd part can be encoded.

The following illustration shows this concept:


au16AdcVocoderBuffer
(Sub) Buffer 0
(Sub) Buffer 1
160 Samples
160 Samples
Ring Buffering ADC and Encoding Time:
Pointer
Pointer
Pointer
Encoding Buffer 0 starts
Pointer
Encoding Buffer 0 Ends
Pointer
Encoding Buffer 1 starts
Pointer
Encoding Buffer 1 ends
Time
Recur
Empty or outdated
Valid data

4.2 Sound Output

The Base Timer 2 (BT2) is initialized as PWM and uses the same base frequency like the BT0 but does not trigger any other peripheral. The interrupt callback of the L3's BT module is used to update the pulse width taken from the buffer `au16VocoderOutputBuffer[VOCODER_ADC_BUFFER_SIZE]` (which is filled by the decoder). This buffer is also split into two sub buffer parts, where one sub buffer is used for voice output and the other for the encoder output.

If the buffer counter reaches `0` or the half of its size (which corresponds to 160 samples) a notification for decoding is set via `bRequestEncodeBuffer0` and `bRequestEncodeBuffer1` respectively.

This encoding must be done in thread mode to release the CPU from handler mode, so that the interrupt cannel of the BT2 is not blocked. The function `Vocoder()` checks the notification and starts the encoding for the corresponding buffer.

4.3 Encoding

As shown in the illustration above the encoding of one buffer is done while the other buffer is being filled by ADC data. The encoding time is much more less than the filling time of a buffer, so that there will never be a collision or overlapping.

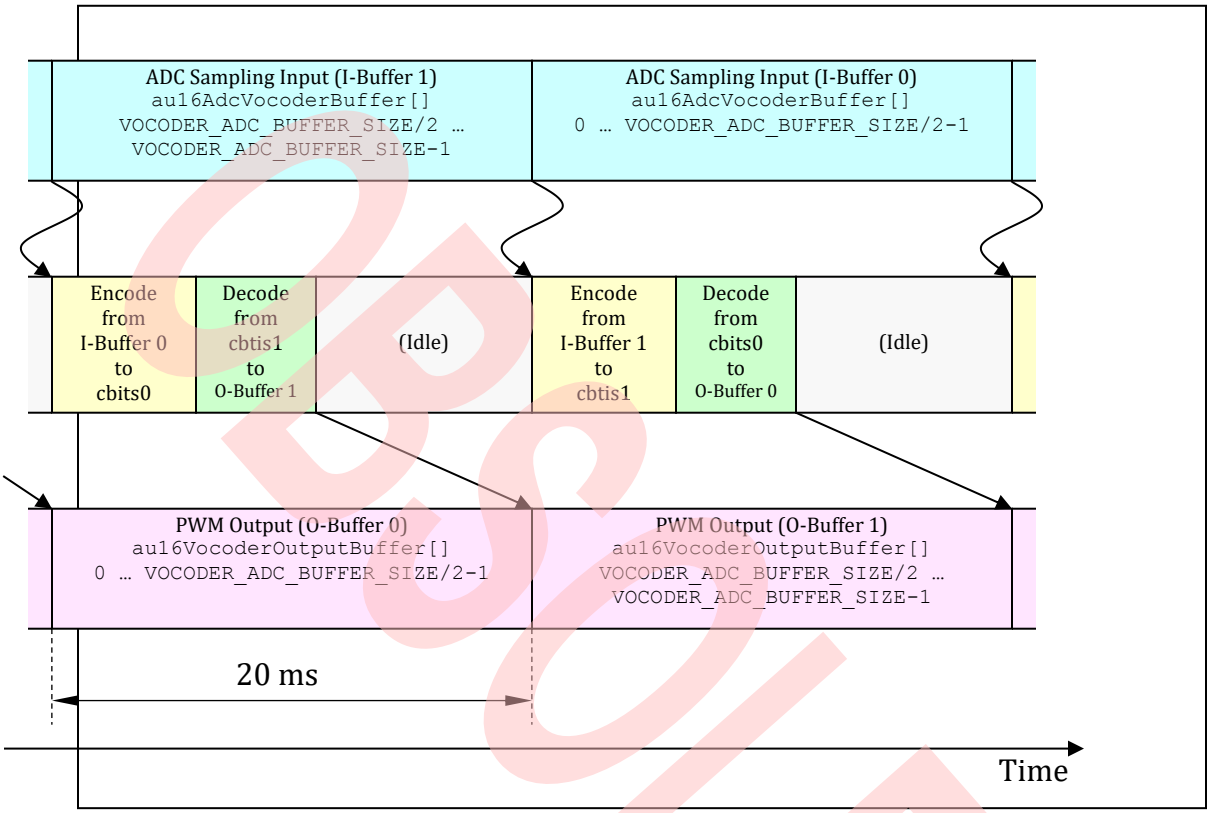
The encoded data is stored in the arrays `cbits0[VOCODER_MAX_ENCODED_BYTES]` and `cbits1[VOCODER_MAX_ENCODED_BYTES]`, where `VOCODER_MAX_ENCODED_BYTES` is the maximum allowed encoded data size (`201`).

4.4 Decoding

The decoding is also done in the function `Vocoder()` directly after encoding the other buffer. The output voice data result decoded from the `cbitsn[]` buffers is stored in the array `au16VocoderOutputBuffer[VOCODER_ADC_BUFFER_SIZE]` corresponding to the sub buffer 0 or 1 location.

4.5 Timing Diagram

The following diagram shows the task parts of the through mode software:



Running the core at 144 MHz the time of the encoding process is about 6.24 ms and for decoding 2.08 ms. These times jitter a bit depending on the voice data. Because a 160 sample frame takes 20 ms, the idle time is 20 ms – 6.24 ms – 2.08 ms = 11.68 ms, so that the CPU load for en- and decoding takes about 41%.

4.6 Memory Usage

4.6.1 Stacks and Heap

Because there is not a very deep sub routine usage on the whole application, the stack size can be remain at `0x800` bytes like in the FM3 template projects.

The Speex Library uses dynamical memory management. Therefore the heap must be increased to `0x8000` bytes in the linker definitions. This value is a save size for all `alloc()` and `malloc()` functions.

4.6.2 RAM Sizes of Project

The Speex Library only uses local (stack) variables and dynamic memory management in the HEAP RAM area. In the following table the RAM usage is listed, where `CSTACK` and `HEAP` sections are added to the C-Lib size.

Project Part	Size in KBytes
Speex Library	-
Vocoder Application	1.9
Low Level Library	0.3
C-Lib, Stacks, Heap	34.0
Total	36.2

4.6.3 ROM Sizes of Project

The following table shows the approximately ROM consumption of the project parts. The data was taken from the IAR linker output with the compiler optimization high/balanced.

Project Part	Size in KBytes
Speex Library	24.0
Vocoder Application	1.8
Low Level Library	2.9
C-Lib, Vectors, Constants	1.8
Total	30.6

5 Software (Intercom Mode)

The principle of the intercom mode is the same like the through mode, with the difference, that the encoded data is sent out via UART and the data to be decoded received by UART.

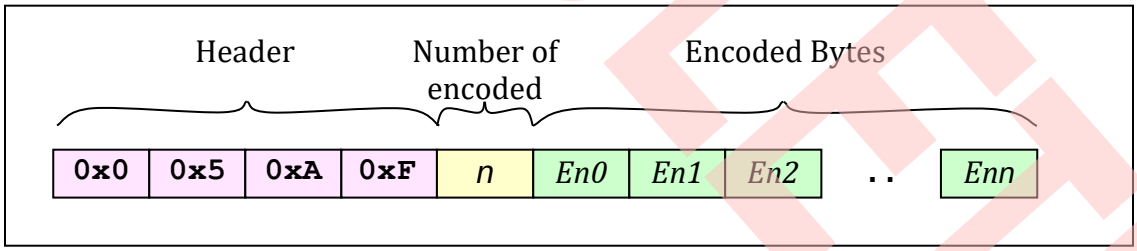
5.1 UART communication

Because of the fact that encoded data are divided in encapsulated frames with a defined start and end, the communication needs some synchronization mechanism to detect the beginning of a frame.

For this reason a header is sent before the frame data. This header consists of the byte pattern `0x00`, `0x55`, `0xAA`, and `0xFF` – a combination, which never occurs in the encoded data itself.

After this header the number of the encoded bytes are sent followed by the encoded data itself.

A protocol frame looks like the following illustration:



For the ease of the UART transmission also the header is being put to the `cbit0[]` and `cbit1[]` arrays. Therefore the `cbit` encoded bytes-start begins at position 4 (number of bytes, `nbBytes`).

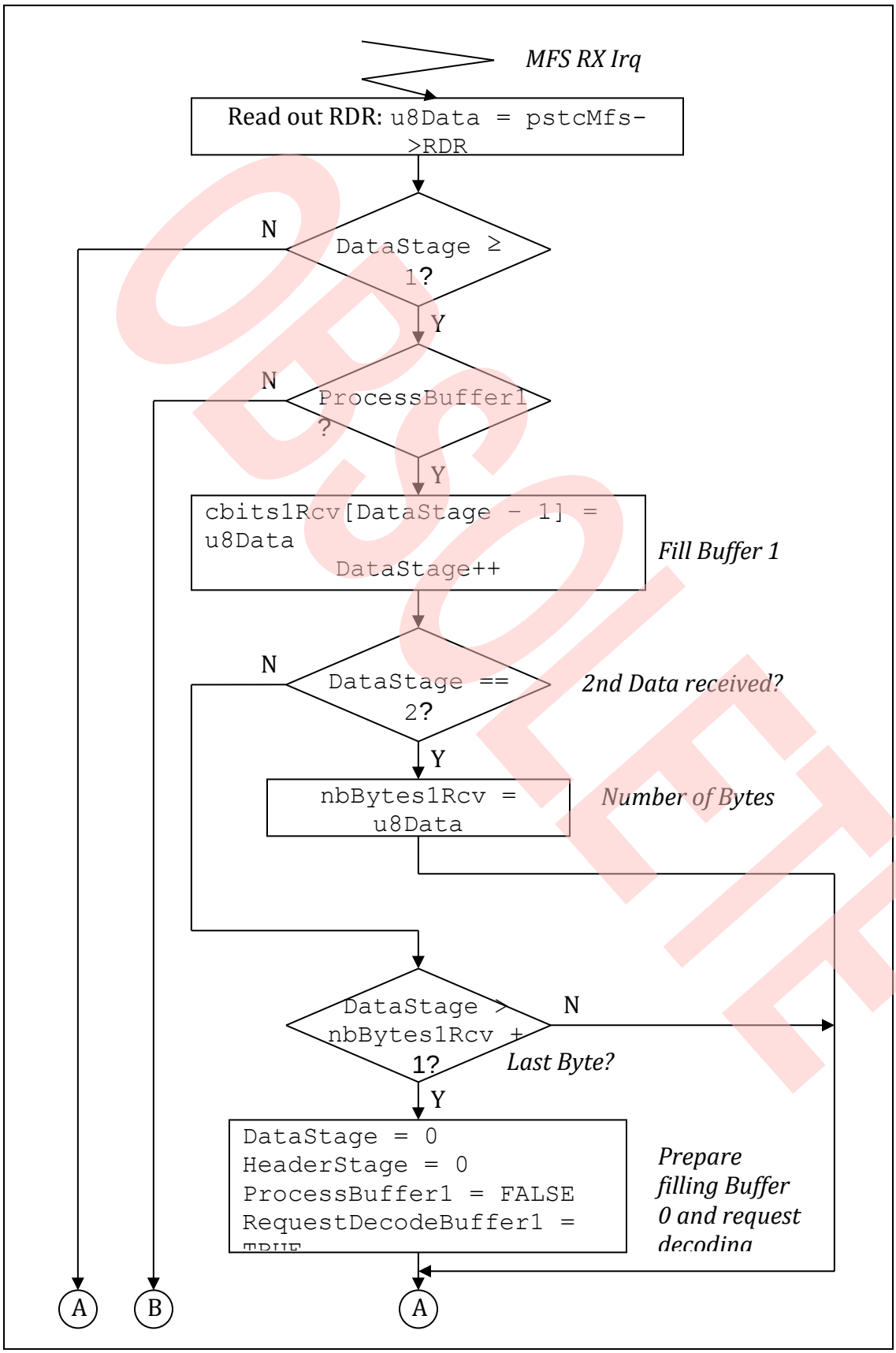
For the UART protocol the intercom application uses 8N1 format at 115200 Bits/s.

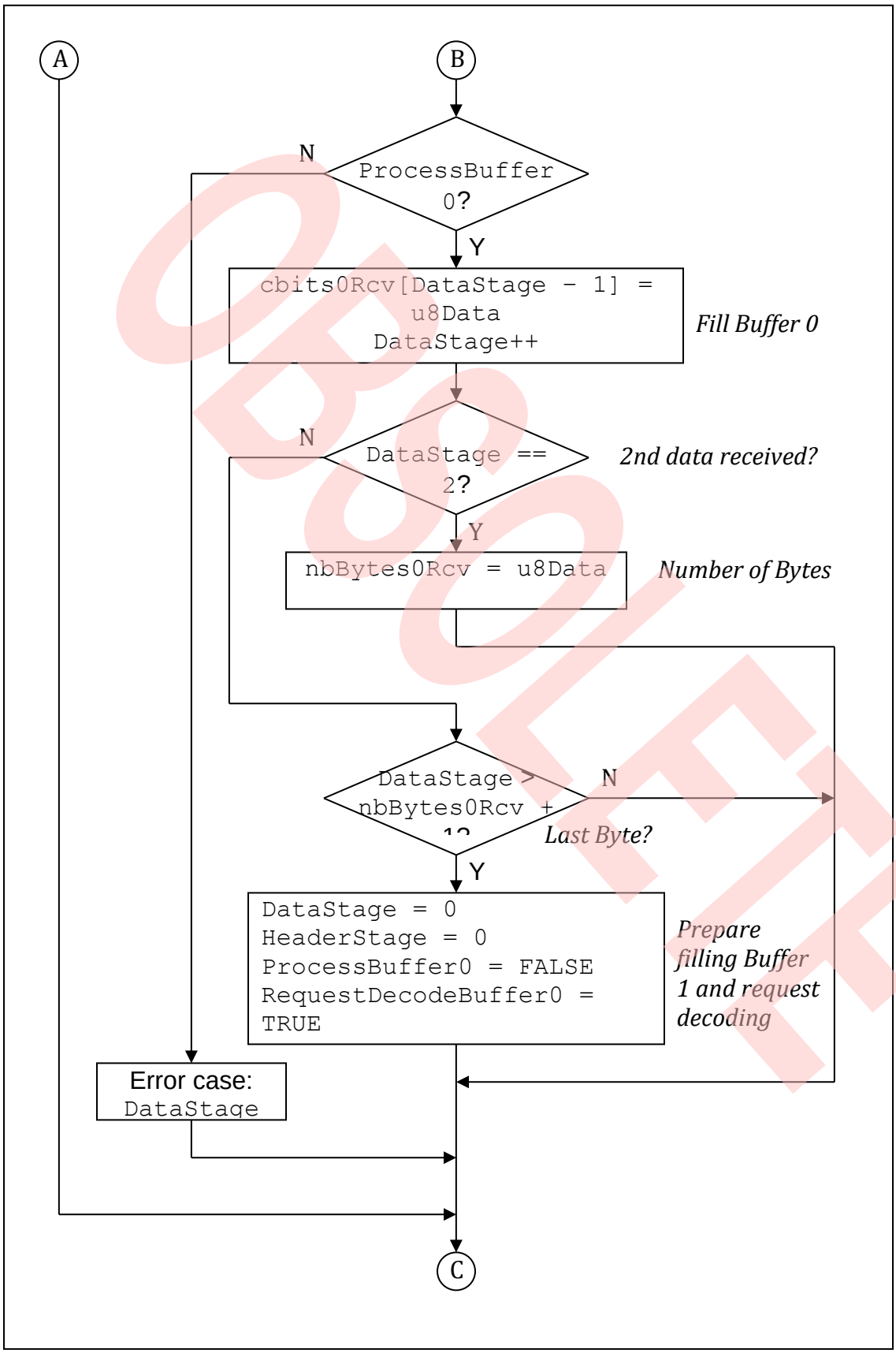
The number of encoded bytes is around 20, and thus we get a frame length of 4 header bytes + 1 number byte + 20 encoded bytes = 25 bytes. For 8N1 format we have an overall bit length of 10 * 25 = 250 bits. At 115200 Bits/s the communication time is 2.17 ms. This more than fits into the idle phase of 11.68 ms from the calculation in chapter 4.5.

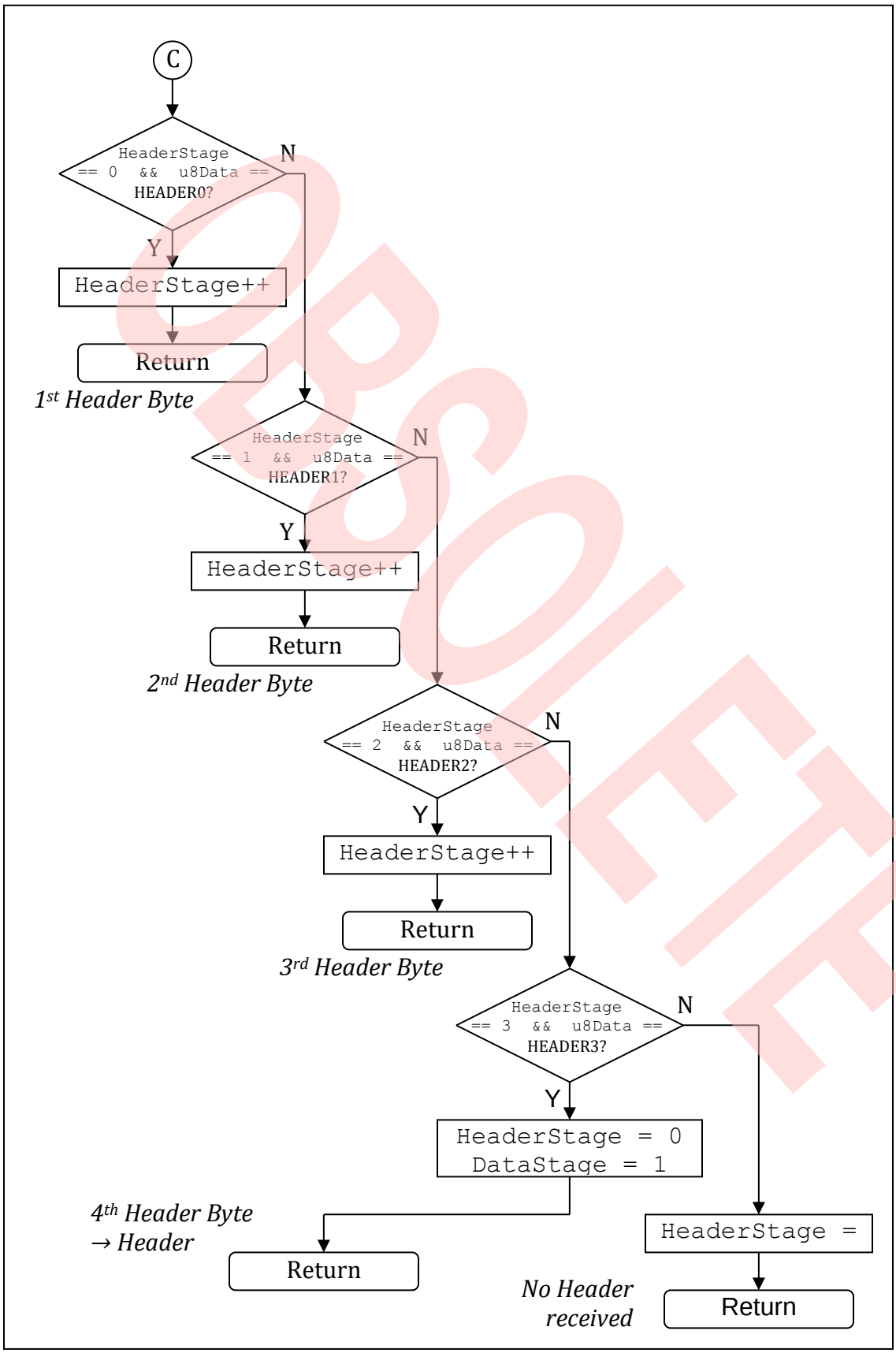
Transmitting a frame is done by the standard `Mfs_Write()` function of the L3.

Because of the use of the header a special interrupt handler was used instead of the standard UART read function of the L3.

The following flow chart shows the UART RX-ISR `Vocoder_MfsnRxIrq()`:


MFS RX Irq
Read out RDR: u8Data = pstcMfs->RDR
DataStage ≥ 1?
N
Y
ProcessBuffer1 ?
N
Y
cbits1Rcv[DataStage - 1] = u8Data
DataStage++
Fill Buffer 1
DataStage == 2?
N
Y
2nd Data received?
nbBytes1Rcv = u8Data
Number of Bytes
DataStage > nbBytes1Rcv + 1?
N
Y
Last Byte?
DataStage = 0
HeaderStage = 0
ProcessBuffer1 = FALSE
RequestDecodeBuffer1 = TRUE
Prepare filling Buffer 0 and request decoding
A
B
A

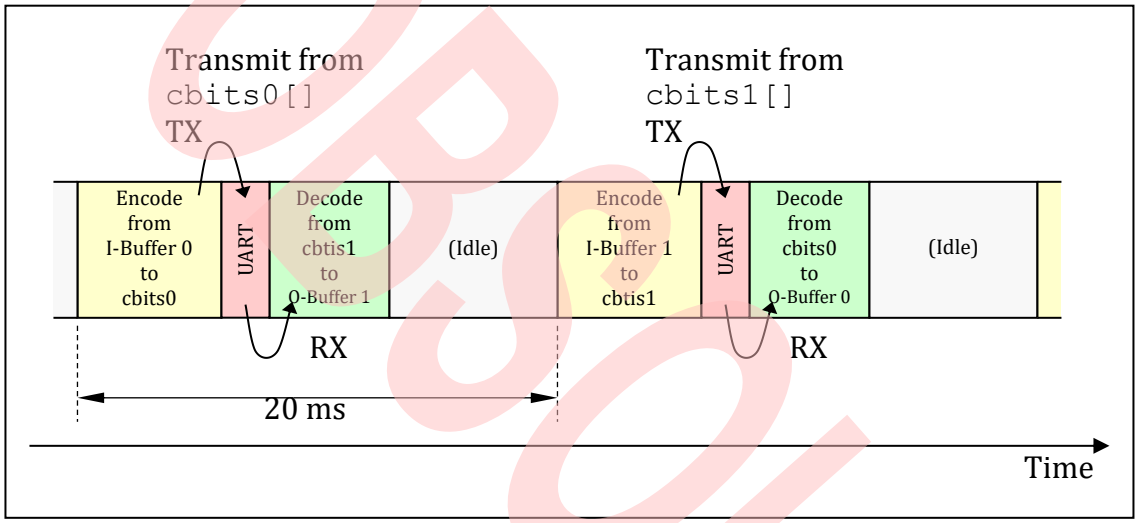

A
B
ProcessBuffer 0?
N
Y
cbits0Rcv[DataStage - 1] = u8Data
DataStage++
Fill Buffer 0
DataStage == 2?
N
Y
2nd data received?
nbBytes0Rcv = u8Data
Number of Bytes
DataStage > nbBytes0Rcv +
Last Byte?
N
Y
DataStage = 0
HeaderStage = 0
ProcessBuffer0 = FALSE
RequestDecodeBuffer0 = TRUE
Prepare filling Buffer 1 and request decoding
Error case:
DataStage
C


C
HeaderStage == 0 && u8Data == HEADER0?
N
Y
HeaderStage++
Return
1st Header Byte
HeaderStage == 1 && u8Data == HEADER1?
N
Y
HeaderStage++
Return
2nd Header Byte
HeaderStage == 2 && u8Data == HEADER2?
N
Y
HeaderStage++
Return
3rd Header Byte
HeaderStage == 3 && u8Data == HEADER3?
N
Y
HeaderStage = 0
DataStage = 1
4th Header Byte
→ Header
Return
HeaderStage =
No Header received
Return

Note that the header check is always performed although the `cbitsn` array may be filled in parallel. At the time the 4th header byte is successfully received, the data count for `cbtisn` is initialized to 1, so that it gets filled in the following data reception and the protocol is synchronized.

5.2 Timing Diagram

The following diagram shows the task parts of the intercom mode software:



Running the core at 144 MHz the time of the encoding process is about 6.24 ms and for decoding 2.08 ms. The UART communication takes 2.17 ms as calculated above. The remaining idle time is 20 ms – 6.24 ms – 2.08 ms – 2.17 ms = 9.51 ms, so that the CPU load for encoding, decoding and communication takes about 52%.

5.3 Memory Usage

5.3.1 Stacks and Heap

Because there is not a very deep sub routine usage on the whole application, the stack size can be remain at `0x800` bytes like in the FM3 template projects.

The Speex Library uses dynamical memory management. Therefore the heap must be increased to `0xE000` bytes in the linker definitions. This value is a save size for all `alloc()` and `malloc()` functions.

5.3.2 RAM Sizes of Project

The Speex Library only uses local (stack) variables and dynamic memory management in the HEAP RAM area. In the following table the RAM usage is listed, where `CSTACK` and `HEAP` sections are added to the C-Lib size.

Project Part	Size in KBytes
Speex Library	-
Vocoder Application	2.3
Low Level Library	0.9
C-Lib, Stacks, Heap	58.5
Total	61.7

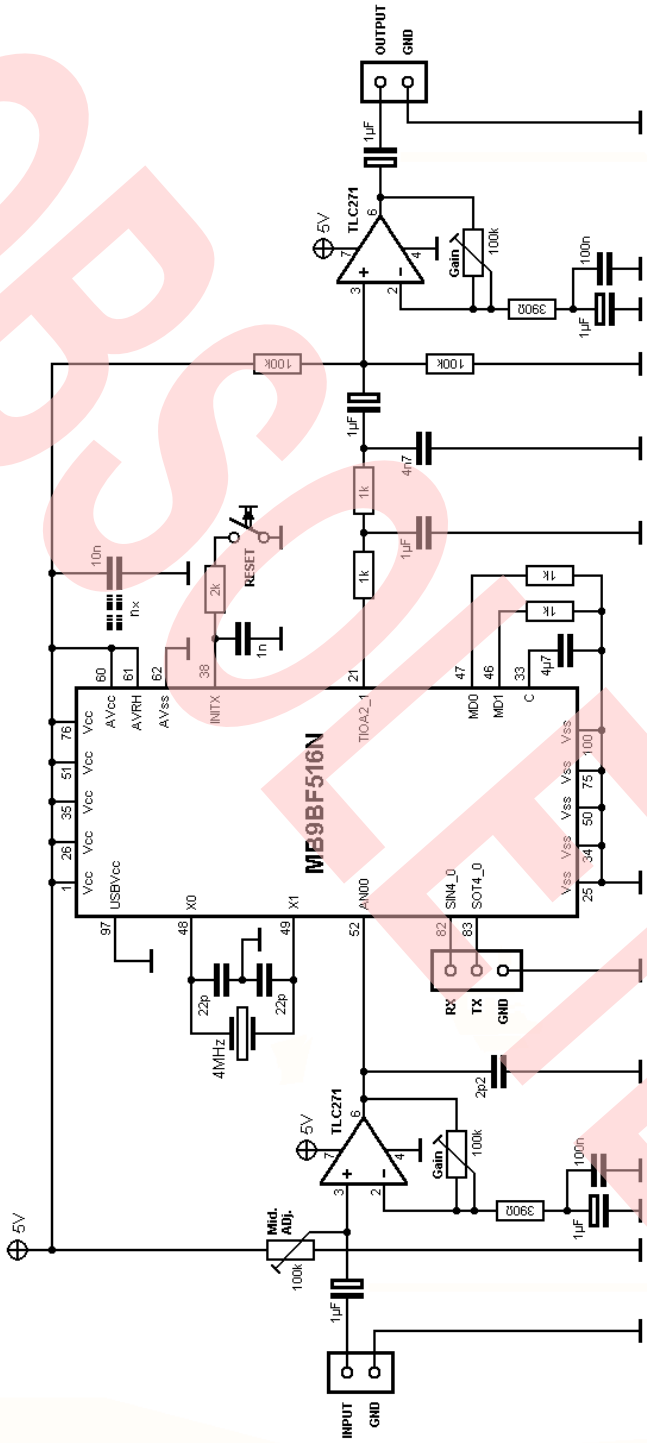
5.3.3 ROM Sizes of Project

The following table shows the approximately ROM consumption of the project parts. The data was taken from the IAR linker output with the compiler optimization high/balanced.

Project Part	Size in KBytes
Speex Library	23.9
Vocoder Application	2.8
Low Level Library	5.1
C-Lib, Vectors, Constants	6.8
Total	38.6

6 Schematic

The following schematic shows a minimum vocoder system for intercom mode. For through-mode the UART RX/TX connector can be skipped.



The “Mid Adjust” trimming resistor just before the first op-amp is calibrated, when the pulses at TIOA2_1 are symmetrical.

The gain trimming resistors can be used for optimum amplification of the signals.

Note that the 2-times low pass filter is only a tiny solution. For better voice output a low pass filter with more than 3rd order should be used. Additionally to suppress the carrier frequency of 8 kHz a Notch filter with this resonance frequency can be used.

7 Further Improvements

7.1 Software

The current software is based on two boards with the same clock source.

In the field the two boards do not have a common clock source, and they will differ according e.g. the crystal frequency. Also different ambient temperatures may desynchronize the communication.

Because the algorithm's base time is the 20 ms audio slot, there will be some communication conflicts.

Further improvements should consider lost communication packages and overrun packages. Lost packages may be solved by repeating the latest received package and overrun may skip the incoming package.

There might be some very short audible "clicking noises" or "buzzing sounds", but this will not decrease the intelligibility.

7.2 Hardware

Because the carrier frequency of the output leads to the need to use a high-order low pass filter.

A very simple solution may be using an FM3 (or FM4) device with an analog output. The resulting alias frequency then can be damped by a low-cost passive low pass filter.

8 Document History

Document Title: AN204433 - FM3 Microcontroller - VOCODER Application Using SPEEX CODEC

Document Number: 002-04433

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	—	MAWI	03/06/2014	Initial release
*A	5034066	MAWI	12/02/2015	Converted Spansion Application Note “FM3_AN706-00083-1v0-E” to Cypress format
*B	**6508191**	WOFR	03/12/2019	Obsolete the document, as it is no longer needed.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless
Spansion Products	spansion.com/products

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com