**S****ource:** SQ-SWG[[1]](#footnote-2)

**Title:** SES verification report v1.0

# Agenda item: 14.4.1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introduction

According to the SES verification plan in [1], STMicroelectronics and IBM have conducted the verification of the SES codec selection. The codecs under consideration are the fixed-point implementations of the AFE/X-AFE codec (Advanced DSR front-end and its extension, cf. [3] and [4]).

The verification is split in different tasks:

* bitexactness verification
* WMOPS verification
* Memory verification

# Verification of bit-exactness

## Motivation

The motivation is to check that the executable used by the ASR vendors corresponds to the executable built from the source code of the selected candidate. The test of "bit-exactness" is used to verify the match of the output bitstreams of the compiled version of the source code of the selected candidate vs. the executables provided to the two test laboratories for selection testing. Output files from both versions are compared with respect to the bit-exactness.

## Definition

The verification laboratories have used:

1. Executables used for selection testing
2. Executables obtained by compiling the source code of the candidate
3. A subset of the samples used for the selection phase.

During the evaluation phase of the AFE/X-AFE algorithm conducted by the testing laboratories, two sampling rates were used, one for the narrowband case (T8) and one for the wideband case (T16). The binaries were delivered for two different platforms: I386/linux RH7.3 (resp. T8\_linux and T16\_linux) and AIX (resp. T8\_AIX and T16\_AIX). Furthermore, two binaries were delivered, respectively for the AFE algorithm (T8 and T16) and the X-AFE algorithm (XT8 and XT16) corresponding to AFE plus its extensions.

The source codes have been compiled on an I386/linux RH7.3 platform with the GNU C compiler, gcc. The executables compiled from the source code are referenced as the “ref” executables (e.g. T8\_ref, T16\_ref, XT8\_ref and XT16\_ref) whereas the executables binaries delivered to the testing laboratories are referenced with the suffix name of the laboratory (e.g. T8\_ibm, T8\_sw, XT8\_ibm and XT16\_ibm).

The bit-exactness verification is made on a subset of the samples used for the selection phase:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Acronym** | **Description** | **Duration** | **#Files** | **Bandwidth** | **Owner** |
| **A3I8** | Aurora 3 Italian | 14h16’ | 4260 | 8kHz | Alcatel |
| **sA3I8** | Subset Aurora 3 Italian | 23’ | 124 | 8kHz | Alcatel |
| **A3I16** | Aurora 3 Italian | 14h16’ | 4260 | 16kHz | Alcatel |
| **MND8** | Mandarin name dialling | 17h35’ | 10241 | 8kHz | Nokia |

Table 1: Databases used for the bit-exactness verification

Bit exactness is checked with the VAD flag off since ASR vendors did not use VAD in their evaluations (cf. section 2.3 of [5]).

The sources of the scripts used for the batches are available in attachment of this document.

## Task

The bitexactness verification is split in 16 batch processing, respectively:

|  |  |  |  |
| --- | --- | --- | --- |
| **Batch name** | **Binary name** | **Database** | **Laboratory** |
| **T8\_linux\_ref\_A3I8** | T8\_linux\_ref | A3I8 | ST |
| **T8\_linux\_sw\_A3I8** | T8\_linux\_sw | A3I8 | ST |
| **T8\_linux\_ibm\_sA3I8** | T8\_linux\_ibm | sA3I8 | IBM |
| **T8\_AIX\_ibm\_sA3I8** | T8\_AIX\_ibm | sA3I8 | IBM |
| **T16\_linux\_ref\_A3I16** | T16\_linux\_ref | A3I16 | ST |
| **T16\_linux\_sw\_A3I16** | T16\_linux\_ibm | A3I16 | ST |
| **XT8\_linux\_ref\_A3I8** | XT8\_linux\_ref | A3I8 | ST |
| **XT8\_linux\_ibm\_A3I8** | XT8\_linux\_ibm | A3I8 | ST |
| **XT8\_linux\_ibm\_sA3I8** | XT8\_linux\_ref | sA3I8 | IBM |
| **XT8\_AIX\_ibm\_sA3I8** | XT8\_AIX\_ibm | sA3I8 | IBM |
| **XT16\_linux\_ref\_A3I16** | XT16\_linux\_ref | A3I16 | ST |
| **XT16\_linux\_sw\_A3I16** | XT16\_linux\_ibm | A3I16 | ST |
| **T8\_linux\_ref\_MND8** | T8\_linux\_ref | MND8 | ST |
| **T8\_linux\_sw\_MND8** | T8\_linux\_sw | MND8 | ST |
| **XT8\_linux\_ref\_MND8** | XT8\_linux\_ref | MND8 | ST |
| **XT8\_linux\_ibm\_MND8** | XT8\_linux\_ibm | MND8 | ST |

Table 2: Batch used for the bit-exactness verification

## Results

The verification laboratory has checked that the binary executables T16\_linux\_ibm and T16\_linux\_sw were identical. The bitstreams generated by the batches are compared with the GNU "diff -x '\*.log' --binary -Nqr" instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Batch name A** | **Batch name B** | **Laboratory** | **Diff** |
| **T8\_linux\_sw\_A3I8** | **T8\_linux\_ref\_A3I8** | ST | none |
| **T8\_AIX\_ibm\_sA3I8** | **T8\_linux\_ibm\_sA3I8** | IBM | none |
| **T16\_linux\_sw\_A3I16** | **T16\_linux\_ref\_A3I16** | ST | none |
| **XT8\_linux\_ibm\_A3I8** | **XT8\_linux\_ref\_A3I8** | ST | none |
| **XT8\_AIX\_ibm\_sA3I8** | **XT8\_linux\_ibm\_sA3I8** | ST | none |
| **XT16\_linux\_sw\_A3I16** | **XT16\_linux\_ref\_A3I16** | IBM | none |
| **T8\_linux\_sw\_MND8** | **T8\_linux\_ref\_MND8** | ST | none |
| **XT8\_linux\_ibm\_MND8** | **XT8\_linux\_ref\_MND8** | ST | none |

Table 3: Bit-exactness results

The binary executables, the source code and the databases were provided in time. Based on the processed databases, the executables compiled by the verification laboratory from the source code gives a bit-exact output with the binary executables delivered to the testing laboratories.

The images of the processed databases will be archived on CD-ROM until TSG-SA#24.

# WMOPS Complexity verification

## Introduction

According to the verification plan, the verification laboratory has compiled the C-code with the extension (X-AFE) on one of the supported platforms (gcc on Sun Solaris 8). The compilation builds an executable to be run at the different sampling rates (resp. XA8 for the narrowband and XA16 for the wideband).

## Source-code verification

The verification laboratory has checked that the C-code correctly implements the basic operators and the source code instrumentation. The verification laboratory has sent to the X-AFE supporting companies a feedback report collecting the minor issues that could be cleaned from the original source code (XA8\_orig and XA16\_orig). The supporting companies have provided to the verification laboratory a cleaned version of the source code (XA8\_cln and XA16\_cln).

The modifications that occurred in the version XA8\_cln and XA16\_cln include:

* minor issues regarding instrumentation over-estimating or under-estimating the WMOPs. As an example, the function qsort\_be() was modified, as suggested in the intermediary report that was sent on the 11th of March, over the SA4 reflector, so that the qsort algorithm operates now directly on Word16 elements.
* A cleaning of the source code: French comments in the code are translated, some ROM arrays have been re-cast with the const keywords, and duplicated tables are deleted.

The verification laboratory has verified that the modifications that occurred in XA8\_cln and XA16\_cln do not impact the output bitstream. In order to do so, the verification laboratory has compared the output bitstream generated by both XA8\_orig and XA8\_cln (resp. XA16\_orig and XA16\_cln). This bit-exactness verification is split in 6 batch processing, respectively:

|  |  |  |
| --- | --- | --- |
| **Batch name** | **Binary name** | **Database** |
| **XA8\_orig\_A3I8** | XA8\_orig | A3I8 |
| **XA8\_orig\_MND8** | XA8\_orig | MND8 |
| **XA16\_orig\_A3I16** | XA16\_orig | A3I16 |
| **XA8\_cln\_A3I8** | XA8\_cln | A3I8 |
| **XA8\_cln\_MND8** | XA8\_cln | MND8 |
| **XA16\_cln\_A3I16** | XA16\_cln | A3I16 |

Table 4: Batch used for the verifying the cleaning process

The bitstreams generated by those 6 batches are compared with the GNU instruction "diff -x '\*.log' –x '\*.wmops' --binary -Nqr".

|  |  |  |
| --- | --- | --- |
| **Batch name A** | **Batch name B** | **Diff** |
| **XA8\_orig\_A3I8** | **XA8\_cln\_A3I8** | none |
| **XA8\_orig\_MND8** | **XA8\_cln\_MND8** | none |
| **XA16\_orig\_A3I16** | **XA16\_cln\_A3I16** | none |

Table 5: Bit-exactness results of the cleaning process

Based on the processed databases, XA8\_orig and XA8\_cln (resp. XA16\_orig and XA16\_cln) both give an identical (i.e. bit-exact) bitstream.

## Complexity results

The cleaned code provided by the candidates is instrumented in such a way that one line of log is generated for each frame, logging the current observed WMOPS score and the maximum observed WMOPS score. All the files from the selected databases (i.e. A3I8, MND8 or A3I16) were processed. The maximum observed WMOPS score is evaluated by selecting the maximum WMOPS score from every sample file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Executable** | **Database** | **Observed** | **Design constraint** |
| X-AFE + X-VQ 8kHz | A3I8 | 24.259 WMOPS (1) | 25 WMOPS |
| X-AFE + X-VQ 8kHz | MND8 | 24.216 WMOPS (2) | 25 WMOPS |
| X-AFE + X-VQ 16kHz | A3I16 | 30.948 WMOPS (3) | 39 WMOPS |

Table 6: WMOPs results

(1) was obtained with low\_speed\_rough\_road/climcontrol/ch0/v10631c5.it0.08

(2) was obtained with Male/taohb/taohb3/wang2jian4.o.a

(3) was obtained with low\_speed\_rough\_road/climcontrol/ch0/v10631c5.it0.16

The complexity is evaluated for both source codes (i.e. XA8\_orig and XA8\_cln; resp. XA16\_orig and XA16\_cln). The results are not significantly different.

# ROM Complexity verification

## Results

The ROM table is verified. Only constant tables and constant arrays are accounted. Constant variables are not counted. The amount of ROM necessary to implement the algorithm is summed up in the following tables:

|  |  |  |
| --- | --- | --- |
| **Executable** | **ROM** | **design constraint** |
| X-AFE 8kHz | 3150 words |  |
| X-VQ 8kHz | 1668 words |  |
| X-AFE + X-VQ 8kHz | 4818 words | 20 kwords |
| X-AFE 16kHz | 3531 words |  |
| X-VQ 16kHz | 1668 words |  |
| X-AFE + X-VQ 16kHz | 5199 words | 34 kwords |

Table 7: ROM usage of the X-AFE + X-VQ algorithm

Note 1: The table ROM\_astFrac is defined with 312 values but only 308 values are initialized.

## Supplementary information

It is noted that the tables used in X-AFE 8kHz are re-used in X-AFE 16kHz. The following table details the ROM usage when X-AFE is implemented with the support for both sampling rates:

|  |  |
| --- | --- |
| **Executable** | **ROM** |
| X-AFE 8/16kHz | 3275 words |
| X-VQ 8/16kHz | 2884 words |
| X-AFE + X-VQ 8/16kHz | 6159 words |

Table 8: ROM usage of the X-AFE+X-VQ algorithm at both sampling rates

# RAM Complexity verification

## Definition

The RAM usage was verified. The RAM usage in X-AFE is split in 3 forms:

* Static RAM
* Heap
* Stack

In order to evaluate the RAM usage of X-AFE, different databases were built. The database describes the memory usage for each function and also the calling tree structure. The format of the database is described in Annex C. A database was built for each of the algorithms, i.e. X-AFE 16kHz, X-AFE 8kHz, X-VQ 16kHz and X-VQ 8kHz. See respectively Annex D and Annex E for the special cases of qsort\_be() and X-VQ. The four databases used for assessing the memory usage (RAM and ROM) from the SES DSR codec are available as an attachment from this document.

## On the usage of structures

Some buffers allocated in the RAM, either from the heap or from the stack (i.e. in local buffers) are used for intermediate storing of complex structures. Those structures include sub-structures, contain pointers or mix relevant variables with function pointers or file handlers used only for interfacing the algorithm with the UNIX i/o or with unused options.

The verification laboratory notes that the size of the buffers allocated for storing such data must be modified on 32-bit platform such as the Unix or Windows platform compared to what is needed for a DSP platform using a 16-bit memory model:

* Variables that are wider than 16 bits are systematically aligned on a 32-bit boundary in 32-bit platforms. This causes a significant loss of memory when Word16 variables are mixed with other structure types inside a structure.
* Pointers are 32-bit wide on 32-bit platform.

In such condition, the memory model generated by the compiler does not match with the memory model used for a DSP; the buffers’ size used in the reference C-code for 32-bit platforms must be adapted in order to match the 16-bit memory model of the DSP.

The verification laboratory has taken those adaptations into account in order to estimate the amount of RAM necessary for supporting the SES-DSR algorithm.

## Static RAM

The static RAM corresponds to variables, tables or arrays that have a lifetime equivalent to the lifetime of the application. Those arrays are defined outside of the scope of a function block, or alternatively with the keyword static.

According to section 5.2, the size of the table scratchDoPitch[] is adapted from 1632 Words to 1090 Words (resp. scratchAdvProcess[], from 1100 Words to 825 Words).

## Heap

Memory from the heap is allocated during the initialization of the AFE/X-AFE. During the processing, the memory allocated in the heap is used like a static RAM memory. The functions malloc/calloc/free are never called during the frame processing.

The heap usage was determined by instrumenting the C-code and by tracing the malloc/calloc/free usage. Since the memory from the Heap is allocated during the initialization, it is independant from the processed file and can be determined alternatively at the compilation time.

The heap usage is accounted as static RAM.

As mentioned in section 5.2, the run-time analysis provides a value on 32-bit platform that over-estimates the amount of RAM memory necessary for DSP 16-bit platforms. See Annex F for the detail of the differences.

|  |  |  |  |
| --- | --- | --- | --- |
| **Executable** | **Run-time** | **Diff** | **Total** |
| X-AFE 8kHz | 3961 | 126 | 3835 |
| X-AFE 16kHz | 5304 | 139 | 5165 |

Table 9: Total heap usage

## Calling tree

The calling tree was verified in order to be able to evaluate the stack usage. The table 1, 2 and 3 gives the calling tree of the application. In 6, we produce the updated calling tree with some corrections (typo errors), the addition of the functions from MathFunc (Pow\_2, Log\_2, Sqrt\_2, Sqtr16\_2) or miscellaneous functions (UpDateDecal, ApplyDecal).

## Stack

The stack depth can be analyzed from the calling tree. Variables and values that can be determined at the compilation time (for instance FFTLength) are not taken into account. Variables that are duplication from already existing variables are not accounted in the stack if they are not used in the block as left-values (i.e. in write mode). According to the verification plan, the functions' arguments are accounted in the stack. The verification laboratory takes into account the sharing of the stack or the overlay of variables when the source code explicitly shows sub-blocks with local definition of variables.

At 8kHz and 16kHz the critical path for the stack usage is described in Table 10.

|  |  |  |
| --- | --- | --- |
| **Calling tree** | **stack depth (8kHz)** | **stack depth (16kHz)** |
| + main | 16 | 16 |
| + DoAdvProcess\_B | 849 | 850 |
| + DoPitchExtract\_B | 1973 | 1974 |
| + RVC\_MeasurePitch\_be | 2038 | 2039 |
| + FindPitchCandidates\_be | 2052 | 2053 |
| + CalcUtilityFunction\_be | 2074 | 2075 |
| + qsort\_be | 2172 | 2173 |
| + swap | 2181 | 2182 |

Table 10: Stack worst path

## Conclusion

The Table 11 details the RAM usage for both X-AFE + X-VQ at 8kHz and at 16kHz.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Executable** | **static RAM** | **stack RAM** | **heap RAM** | **total RAM** | **Design constraint** |
| X-AFE 8kHz | 446 | 2181 | 3835 | 6462 |  |
| X-VQ 8kHz | 7 | 38 | 0 | 45 |  |
| X-AFE + X-VQ 8kHz | 453 | 2181 | 3835 | 6469 | 7000 words |
| X-AFE 16kHz | 446 | 2182 | 5165 | 7793 |  |
| X-VQ 16kHz | 7 | 38 | 0 | 45 |  |
| X-AFE + X-VQ 16kHz | 453 | 2182 | 5165 | 7800 | 8000 words |

Table 11: Total RAM usage

# Bibliography

1. S4-040153 “Verification plan for SES DSR v1.0”, SA4#30
2. S4-040054 “Draft TS Software documentation for fixed-point DSR Extended Advanced Front-end”, SA4#30
3. ETSI standard ES 202 050 “Distributed Speech Recognition; Advanced Front-end Feature Extraction Algorithm; Compression Algorithms”, Oct 2002, <http://pda.etsi.org/PDA/home.asp?wki_id=yeZ1Qi@QwpOPXVVTO7wZ2>
4. ETSI standard ES 202 212 “Distributed Speech Recognition; Extended Advanced Front-end Feature Extraction Algorithm; Compression Algorithm”, Nov 2003, <http://pda.etsi.org/PDA/copy_file.asp?Action_type=&Action_Nb=&Profile_id=IugJxMadBBxgVRiTVU7weOO&Wki_id=yPyx-MSKzNpqwrsvVBZ_Z>
5. S4-030248 “Design Constraints for default codec for speech enabled services (SES)”, SA4#25bis
6. S4-030866 “Consideration of DSR executable code update to ASR vendors”, SA4#30
7. Updated Calling Tree
   1. XAFE 8kHz calling tree

main

AdvProcessInit\_B

DoNoiseSupInit\_B

DoWaveProcInit\_B

DoCompCepsInit\_B

DoPostProcInit\_B

DoVADInit\_F

BufIn32Alloc

AdvProcessAlloc\_B

DoNoiseSupAlloc\_B

DoWaveProcAlloc\_B

DoCompCepsAlloc\_B

DoPostProcAlloc\_B

DoVADAlloc\_F

FlushAdvProcess\_B

DoVADFlush\_F

CvFeatInt2Float

AdvProcessDelete\_B

DoNoiseSupDelete\_B

DoWaveProcDelete\_B

DoCompCepsDelete\_B

DoPostProcDelete\_B

DoVADDelete\_F

BufIn32Free

DoAdvProcess\_B

BufIn32ShiftToPut

DoNoiseSup\_B

VAD\_F

Log\_2

DoSigWindowing16\_F1

DoSigWindowing16\_F2

ff4NRFix32\_B

GetL15

GetH15

Mult16x32

Add\_Mult16x16\_16

Sub\_Mult16x16\_16

Permut

FFTtoPSD\_F

Square24d2\_B

Square24d2\_B

Square24\_B

PSDMean\_F

NoiseEstimation\_F1

Sqrt\_2

Sqrt16\_2

ADJUST\_SHFT

NoiseEstimation\_F2

Sqrt\_2

Sqrt16\_2

ADJUST\_SHFT

FilterCalc\_F

SpeechQVar

FilterBank16

SpeechQSpec

SpeechQMel

DoGainFact\_F1

Log\_2

DoGainFact\_F2

Log\_2

DoMelIDCT\_F16

ApplyWF

UpDateDecal

ApplyDecal

DCOffsetFil\_F

DoPitchExtract\_B

FilterBank

IsLowBandNoise

RVC\_MeasurePitch\_be

CalculateDoubleWindowDft\_be

ClearPitch\_be

DirichletInterpolation\_be

Finalize\_be

IsContinuousPitch\_be

Mpy\_lw\_sw

FindPitchCandidates\_be

CalcUtilityFunction\_be

AddSortedArrayOfPoints\_be

LinkArrayOfPoints\_be

Compare\_ARRAY\_OF\_XPOINTS\_be

ConvertLinkedListOfDiffPointsToUtilFunc\_be

CreatePieceWiseConstantFunction\_be

L\_Extract

Mpy\_32\_16

LinkArrayOfPoints\_be

qsort\_be\*

swap

ComparePitchFreqAscending\_be

FindDominantLocalMaximaInUtilityFunction\_be

Mpy\_lw\_sw

NormalizeAmplitudes\_be

SelectTopPitchCandidates\_be

Mpy\_lw\_sw

UtilityFunctionAtGivenPitchFreq\_be

compute\_pcorr\_be

Mpy\_lw\_sw

accumulate\_be

find\_most\_energetic\_window2\_be

find\_most\_energetic\_window\_be

interpolate\_be

Mpy\_lw\_lw

Mpy\_lw\_sw

sqrt\_l\_fix

qsort\_be\*

swap

IsLowLevelInput\_be

Mpy\_lw\_sw

PrepareSpectralPeaks\_be

CalcSpectrum\_be

Mpy\_lw\_sw

Mpy\_lw\_sw\_Add

qsort\_be\*

swap

CompareIpointAmp\_be

Final\_ScaleDownAmpsOfHighFreqPeaks\_be

FindPeaks\_be

Prelim\_ScaleDownAmpsOfHighFreqPeaks\_be

RefineSpectralPeaks\_be

sqrt\_l\_fix

Mpy\_lw\_sw

SelectFinalPitch\_be

BETTER\_be

CLOSELY\_LOCATED\_be

Mpy\_lw\_sw

ClearPitch\_be

qsort\_be\*

swap

ComparePitchFreqDescending\_be

GOOD\_ENOUGH\_be

IsContinuousPitch\_be

Mpy\_lw\_sw

ClearPitch\_be

classify\_frame

dsr\_afe\_vad

get\_vm

fnLog2

get\_zcm

pre\_process

iir\_d

iir\_s

BufIn32GetLast

DoWaveProc\_B

TeagerEng

GetTeagerFilter

GetMaximaPositions

DoCompCeps\_B

CepsCompute

Log\_2

PreEmphHamm

ff4NR16\_B

Permut

FilterBank

CosInv

DoPostProc\_B

DoVADProc\_F

focalpoint

CvFeatInt2Float

RVC\_ConstructPitchMeter\_be

Allocate\_InterpolatedDft\_be

RVC\_ResetPitchMeter\_be

RVC\_ConstructPitchRom\_be

RVC\_DestructPitchMeter\_be

Deallocate\_InterpolatedDft\_be

RVC\_DestructPitchRom\_be

* 1. XAFE 16kHz calling tree

main

AdvProcessInit\_B

DoNoiseSupInit\_B

DoWaveProcInit\_B

DoCompCepsInit\_B

DoPostProcInit\_B

DoVADInit\_F

Do16kProcInit\_B

QMF\_FIR\_Init\_B

fir\_initialization\_B

DP\_HP\_filters\_B

BufIn32Alloc

AdvProcessAlloc\_B

DoNoiseSupAlloc\_B

DoWaveProcAlloc\_B

DoCompCepsAlloc\_B

DoPostProcAlloc\_B

DoVADAlloc\_F

Do16kProcAlloc\_B

FlushAdvProcess\_B

DoVADFlush\_F

CvFeatInt2Float

AdvProcessDelete\_B

DoNoiseSupDelete\_B

DoWaveProcDelete\_B

DoCompCepsDelete\_B

DoPostProcDelete\_B

DoVADDelete\_F

BufIn32Free

DoAdvProcess\_B

BufIn32ShiftToPut

Do16kProcessing\_B

DoNoiseSup\_B

Get16k\_p\_bufferData16k\_B

Get16k\_bufData16kSize\_B

Get16k\_p\_BandsForCoding16k\_B

Get16k\_p\_CodeForBands16k\_B

VAD\_F

Log\_2

DoSigWindowing16\_F1

DoSigWindowing16\_F2

ff4NRFix32\_B

GetL15

GetH15

Mult16x32

Add\_Mult16x16\_16

Sub\_Mult16x16\_16

Permut

FFTtoPSD\_F

Square24d2\_B

Square24d2\_B

Square24\_B

Get16k\_BFC\_dec\_B

GetBandsForCoding16k\_B

Log\_2

PSDMean\_F

NoiseEstimation\_F1

Sqrt\_2

Sqrt16\_2

ADJUST\_SHFT

NoiseEstimation\_F2

Sqrt\_2

Sqrt16\_2

ADJUST\_SHFT

FilterCalc\_F

SpeechQVar

FilterBank16

SpeechQSpec

SpeechQMel

DoGainFact\_F1

Log\_2

DoGainFact\_F2

Log\_2

DoMelIDCT\_F16

ApplyWF

Get16k\_dec1

Get16k\_dec2

Get16k\_dec3

DoSigWindowing16\_F3

DoMelFB\_B

CodeBands16k\_B

DoSpecSub16k\_B

Log\_2

UpDateDecal

ApplyDecal

DCOffsetFil\_F

Get16k\_hpBandsSize\_B

Get16k\_p\_hpBands\_B

Get16k\_p\_bufferCodeForBands16k\_B

Get16k\_p\_CodeForBands16k\_B

Get16k\_p\_bufferCodeWeights\_B

Get16k\_p\_codeWeights\_B

Set16k\_hpBands\_dec\_B

DoPitchExtract\_B

FilterBank

IsLowBandNoise

RVC\_MeasurePitch\_be

CalculateDoubleWindowDft\_be

ClearPitch\_be

DirichletInterpolation\_be

Finalize\_be

IsContinuousPitch\_be

Mpy\_lw\_sw

FindPitchCandidates\_be

CalcUtilityFunction\_be

AddSortedArrayOfPoints\_be

LinkArrayOfPoints\_be

Compare\_ARRAY\_OF\_XPOINTS\_be

ConvertLinkedListOfDiffPointsToUtilFunc\_be

CreatePieceWiseConstantFunction\_be

L\_Extract

Mpy\_32\_16

LinkArrayOfPoints\_be

qsort\_be\*

swap

ComparePitchFreqAscending\_be

FindDominantLocalMaximaInUtilityFunction\_be

Mpy\_lw\_sw

NormalizeAmplitudes\_be

SelectTopPitchCandidates\_be

Mpy\_lw\_sw

UtilityFunctionAtGivenPitchFreq\_be

compute\_pcorr\_be

Mpy\_lw\_sw

accumulate\_be

find\_most\_energetic\_window2\_be

find\_most\_energetic\_window\_be

interpolate\_be

Mpy\_lw\_lw

Mpy\_lw\_sw

sqrt\_l\_fix

qsort\_be\*

swap

IsLowLevelInput\_be

Mpy\_lw\_sw

PrepareSpectralPeaks\_be

CalcSpectrum\_be

Mpy\_lw\_sw

Mpy\_lw\_sw\_Add

qsort\_be\*

swap

CompareIpointAmp\_be

Final\_ScaleDownAmpsOfHighFreqPeaks\_be

FindPeaks\_be

Prelim\_ScaleDownAmpsOfHighFreqPeaks\_be

RefineSpectralPeaks\_be

sqrt\_l\_fix

Mpy\_lw\_sw

SelectFinalPitch\_be

BETTER\_be

CLOSELY\_LOCATED\_be

Mpy\_lw\_sw

ClearPitch\_be

qsort\_be\*

swap

ComparePitchFreqDescending\_be

GOOD\_ENOUGH\_be

IsContinuousPitch\_be

Mpy\_lw\_sw

ClearPitch\_be

classify\_frame

dsr\_afe\_vad

get\_vm

fnLog2

get\_zcm

pre\_process

iir\_d

iir\_s

BufIn32GetLast

DoWaveProc\_B

TeagerEng

GetTeagerFilter

GetMaximaPositions

DoCompCeps\_B

CepsCompute

Get16k\_p\_bufferCodeWeights\_B

Get16k\_p\_bufferCodeForBands16k\_B

Log\_2

PreEmphHamm

ff4NR16\_B

Permut

GetBandsForDecoding16k\_B

DecodeBands16k\_B

FilterBank

Get16k\_hpBands\_dec\_B

Get16k\_p\_hpBands\_B

MergeSSandCoded\_B

CorrectEnergy\_B

Pow2

CosInv16Khz

DoPostProc\_B

DoVADProc\_F

focalpoint

CvFeatInt2Float

RVC\_ConstructPitchMeter\_be

Allocate\_InterpolatedDft\_be

RVC\_ResetPitchMeter\_be

RVC\_ConstructPitchRom\_be

RVC\_DestructPitchMeter\_be

Deallocate\_InterpolatedDft\_be

RVC\_DestructPitchRom\_be

1. Differences T8 vs. A8, XT16 vs. XA16
   1. XT16 vs. XA16
      1. Summary

STMicroelectronics has verified the differences between XT16 (the extended front-end C-code delivered to the testing laboratories) and XA16 (actually XA16\_orig, the extended front-end code, cf. section 3.2).

Two files were modified:

* ParmInterface\_B.c
* 16kHzProcessing\_B.c
  + 1. 16kHzProcessing\_B.c

The values from the ROM table LambdaNSEx2[] were stored initially in XT16 under the format 2\*(X\_INT16)((0x8000 - x)/2); in XA16, the same values are stored as x. In XA16, the values of this table are used through:

LambdaNSE32 = LambdaNSEx2[nbFrame];

whereas in XT16, they are used through:

LambdaNSE32 = (X\_INT16)((1<<15)-LambdaNSEx2[nbFrame]);

It is the understanding of STMicroelectronics that at least one additional significant bit is lost in the computing of LambdaNSE32 in the version from XT16.

Furthermore, in the fixed-point evaluation of frameEn32 and meanEn32, the scaling is different. From the two versions, XT16 is the version that brings the poorest accuracy. The version of XA16 is aligned with the correction presented in [6].

As a conclusion, it is the understanding of STMicroelectronics that the version of XT16 uses a poorer precision than the version of XA16 for the computing of lambdaNSE32, frameEn32 and meanEn32.

* + 1. ParmInterface\_B.c

In the file ParmInterface\_B.c, at the lines 328, 333, 338 and 343, the following block is present in XA16 and A16 whereas it is missing from XT16.

test();

if (curShft<0) {

curShft = add(32,curShft);

}

* 1. T16 vs. A16
     1. 16kHzProcessing\_B.c

The file 16kHzProcessing\_B.c is modified precisely as described in [6].

It is the understanding of STMicroelectronics that T16 is using the floating-point unit for computing lambdaNSE32 whereas A16 uses the fixed-point arithmetic for computing lambdaNSE32. The modification is made according to [6]. In [6], it is shown that the performances are not impacted by the loss of precision.

1. Database format

The memory usage was evaluated with the help of a database, which describes simultaneously the calling tree, the stack memory usage, the static RAM usage and the ROM usage.

The tools use two different interchange formats. Both are based on XML. The native format of the database is the FCT\_LIST format. In this format, each block of instruction and each function define a unique entry with a name, the description of its memory usage (static RAM, stack and ROM) and finally the list of its sub-functions (if any).

The alternative format of the database is the FCT\_TREE format. In this format, the structure of the calling tree is explicit. Each node in the tree corresponds to a node from the calling tree. The depth of the stack is available at each node. Other forms of memory (static RAM and ROM) are also available in this format but the detail of its usage.

The two formats are equivalent: it is possible to transform the database from the FCT\_LIST format into the FCT\_TREE format or from the FCT\_TREE format into the FCT\_LIST format. The native format of the database is FCT\_LIST.

Different tools are available for manipulating the database, for instance (and not only):

* computing the critical path for the stack usage,
* evaluating the static RAM usage,
* evaluating the ROM usage.

As an example, here is the section describing the pre\_process() function in the file preProc\_B.c.

<?xml version="1.0"?>

<fct\_list>

[...]

<fct name="pre\_process" file="preProc\_B.c">

<mem argin="6"/>

<ref\_list>

<ref id="#iir\_d"/>

<ref id="#iir\_s"/>

</ref\_list>

</fct>

<fct name="iir\_d" file="preProc\_B.c">

<mem argin="7" stack="6"/>

</fct>

<fct name="iir\_s" file="preProc\_B.c">

<mem argin="7" stack="6"/>

</fct>

</fct\_list>

Code 1: Example of description in FCT\_LIST format

1. Memory assessment for qsort\_be()

The function qsort\_be(), in the current implementation is a recursive implementation of the quick sort algorithm. The deepest recursion depth of the call is 31. The verification laboratory has found that 3 Words must be duplicated in the stack for each step of the algorithm, in addition to the stack usage from the original call. Therefore, the supplementary cost due to the recursion is 90 Words in the stack.

1. X-VQ

The SES DSR algorithm has been split in two separate binary executables that perform respectively the Mels-Frequency Cepstrum Coefficient (MFCC) computation and the Quantisation of the MFCC. Both algorithms operate on the same frame windows and share only a limited amount of data:

* 13 MFCC Coefficient (Word16)
* Pitch Information (Word32)
* Class Information (Word8)
* VAD (Boolean).

The interface of the two executable was kept compatible with the former floating point version available from the ETSI/Aurora group. The verification laboratory has checked that the floating-point values used in the file interface between X-AFE and X-VQ are used only for conveying the fixed-point values.

A IEEE floating point format is made from a 24-bit mantissa. A Word16 value can be stored entirely in the mantissa without loss or corruption of bits.

In theory, the transfer of a Word32 through the mantissa of a IEEE float value implies that only the 24 most significant bits are kept (the remaining bits are rounded). In the present situation, it was checked that the Pitch Information, stored in the algorithm in Word32 has a dynamic limited to a Word16 and that the transfer of data between X-AFE and X-VQ does not imply loss of precision.

In the source code provided by the candidate, X-VQ includes supplementary processing past the quantization process in order to store the quantized bitstream in an ETSI compatible format. The verification laboratory has limited the investigation and the memory assessment to the quantization process and has not taken into account the ETSI compatible bitstream generation.

The verification laboratory has also considered that the two algorithms (X-AFE and X-VQ) were both executed in sequence frame per frame. Therefore, the stack of the two binaries can be shared. The critical path being located in the X-AFE, the X-VQ can reuse at each frame all the stack freed by X-AFE after the frame processing and it is not necessary to add the stack usage of X-VQ in the total RAM usage.

1. Heap analysis

Due to structure alignment on 32-bit boundaries, pointers usages, function pointer usage, "int" type usage, the heap usage report available from run-time analysis must be adapted (cf. 5.2). The table below summarizes the differences.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table** | **Run-time** | **Diff** | **Total** | **File** |
| FEParamsX\_F | 112 | 93 | 19 | ParamInterface\_B.c |
| DoVADInit\_F (OutBuffer) | 14 | 7 | 7 | VAD\_F.c |
| NoiseSupStructX\_F |  | 7 |  | ExtNoiseSup\_B.c |
| CompCepsStructX\_F | 6 | 1 | 5 | CompCeps\_B.c |
| RVC\_PITCH\_ROM\_be | 18 | 5 | 13 | rvc\_pitch\_init\_B.c |
| RVC\_PITCH\_METER\_be | 40 | 13 | 27 | rvc\_pitch\_init\_B.c |
| DataFor16kProc\_B |  | 10 |  | 16kHzProcessing\_B.c |
| QMF\_FIR | 10 | 3 | 7 | 16kHzProcessing\_B.c |
| **Total (8kHz)** |  | 126 |  |  |
| **Total (16kHz)** |  | 139 |  |  |

Table 12: Detailed usage of the Heap

Note that (FEParamsX\_F \*) from ParamInterface\_B.c is allocated from the heap. This structure mixes buffers and variables necessary for the algorithm, with function pointers and file handlers, which should not be taken into account in the memory assessment, and also with pointers and "int" types, which are 32-bit wide on the platform used. Only 19 Words are used from the 112 allocated by the system. Therefore, the run-time analysis provides values that are over-estimated: 93 Words should not be taken into account.

In DoVADInit\_F, the memory allocated for OutBuffer is doubled because pointers are accounted for 32-bit data instead of 16-bits data. This is causing the waste of 7 words. The structure NoiseSupStructX\_F wastes 7 Word16 in 32-bit alignment. The structure RVC\_PITCH\_ROM\_be and RVC\_PITCH\_METER\_be waste respectively 5 and 13 Words in 32-bit alignment. The structure DataFor16kProc\_B, used at 16kHz, wastes also 10 Words due to the 32-bit alignment of data and pointers.

1. Editor: **Stéphan Tassart**

   STMicroelectronics,

   Email: stephan.tassart@st.com [↑](#footnote-ref-2)