Technical Specification Group Services and System Aspects
Meeting #23, Phoenix, AZ, USA, 15-18 March 2004

TSGS#23(04)0077

3GPP TSG-SA4#30 meeting
February 23-27, 2004, Málaga, Spain

Tdoc S4-040153

Source: **TSG SA WG4** [STMicroelectronics][1]

Title: **Draft verification plan v. 1.0**

Agenda item: **7.4.3**

Document for: **Information**

---

1. Introduction

This document provides a verification plan for the SES codec selection. The SES candidate selected during SA4#30 (February 23-27, 2004) will be brought to TSG-SA for approval (TSG-SA#23, March 15-17, 2003). Some critical items (as listed in [1]) will be verified by volunteering organizations before the candidate is brought to TSG-SA.

The codecs under consideration are the AFE/X-AFE codec (Advanced DSR front-end and its extension, cf. [3,4]), the AMR-NB codec and the AMR-WB codec. In case of the AMR-NB and AMR-WB codecs are selected then the independent complexity assessment results that are already available from earlier standardisation efforts will be used to verify the complexity. In the case of the AFE/X-AFE codec the fixed-point implementation will be verified.

In the case that SA4 passes decision to TSG-SA because the performance falls in the "grey area" of the recommendation criteria (cf. [11]) and SA4 is unable to reach consensus then verification will also be performed before it is brought to TSG-SA.

2. Verification of bit-exactness

2.1 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. A 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 and the executables provided to the two test laboratories for selection testing. Output files from both versions are compared with respect to the bit-exactness.

2.2 Definition

The verification laboratories will make use of:

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

---

[1] **Stéphan Tassart**
STMicroelectronics,
Email: stephan.tassart@st.com

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).

Source codes will be provided to the verification laboratories. The executables compiled from the source code are the reference executables to be run at the different sampling rates (resp. B8 and B16).

Bit exactness will be checked with the VAD flag off since ASR vendors did not use VAD in their evaluations [section 2.3 of 10].

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

Acronym	Description	Duration	Bandwidth	Owner
A3I8	Aurora 3 Italian	8h	8kHz	Alcatel
A3I16	Aurora 3 Italian	8h	16kHz	Alcatel
MND8	Mandarin name dialling	5h	8kHz	Nokia

Table 1: complexity requirements for the SES candidate

2.3 Task

2.3.1 Narrowband verification

The verification laboratory tests the bit-exactness of the output bitstream of the candidate B8 vs. the output bitstream of the executable T8_linux or T8_AIX provided to the testing laboratories.

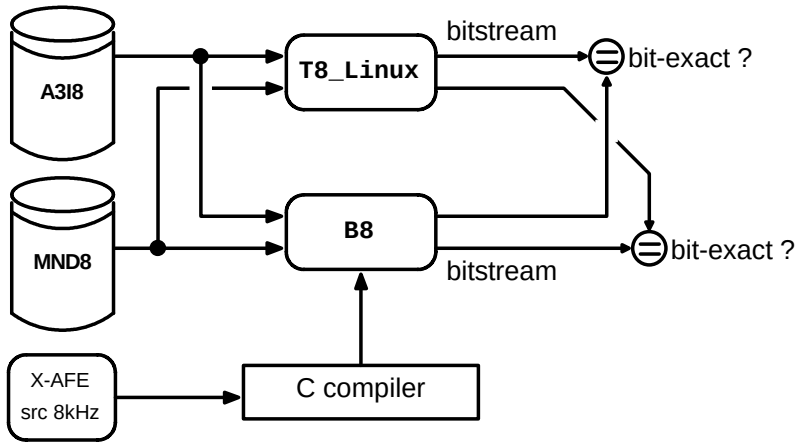

Figure 1: Verification of the bit-exactness of the narrowband candidate

The platform used for verifying the bit-exactness of the candidate is not relevant because the source code of the narrowband candidate is platform independent (i.e. bit-exact on any supported platform). The verification laboratories can use any supported platform for verifying T8_linux or T8_AIX, i.e. the executable used by the test laboratories.

2.3.2 Wideband verification

The verification laboratory tests the bit-exactness of the output bitstream of the candidate B16_linux vs. the output bitstream of the executable T16_linux provided to the testing laboratories.

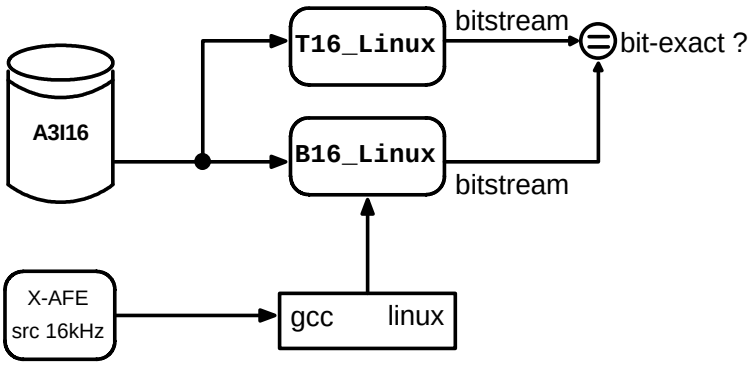

Figure 2: Verification of the bit-exactness of the wideband candidate

Motorola notified to the committee that 6 lines of the code delivered to the testing laboratories were incorrect. Only the wideband case (`T16_linux` and `T16_AIX`) is affected (cf. [5]). The code delivered to the testing laboratories contains a processing block using the floating-point arithmetic (cf. [6]). The verification laboratory checks that the compilation of the source code with the floating-point arithmetic mentioned by Motorola and the executables delivered to the testing laboratories generate identical bitstreams. However, since the IEEE floating-point arithmetic is not bit-exact (cf. [7,8]), the verification of the binaries can be conducted only on a similar platform (same hardware, same compiler, same compilation options).

3. WMOPS Complexity verification

3.1 Motivation

The compiled version of the fixed-point ANSI-C source code must meet the design constraints (cf. [9]). The WMOPS complexity of the candidate will be estimated in the framework of the worst observed frame on a subset of the samples used for the selection phase.

Bandwidth	WMOPS design constraint
narrowband	≤25 WMOPS
wideband	≤39 WMOPS

Table 2: complexity requirements for the SES candidate

3.2 Source-code verification

The source code is used to verify the complexity of the codec. The verification laboratory checks that the C-code has been correctly implemented with basic operators and that the C-code correctly implements the instrumentation that generates a maximum WMOPS score for each sample file.

3.3 Complexity verification

3.3.1 Task

The verification laboratories compile the C-code on one of the supported platforms (gcc on AIX, i386/linux RH7, Sun Solaris 8 or possibly VC++ on win32) and build an executable to be run at the different sampling rates (resp. A8 for the narrowband and A16 for the wideband) (Note: the versions A8 and B8 are identical).

The verification laboratories check that the complexity of the VAD processing is included in the WMOPS complexity verification as indicated in [9].

The executable generates a log file with the maximum observed WMOPS score for each sample file. The verification laboratories process all the files from the selected subset and evaluate the maximum observed WMOPS score. The maximum observed

WMOPS score is evaluated by selecting the maximum WMOPS score from every sample file. The obtained maximum observed WMOPS score is compared with the design constraints (cf. Table 2).

3.3.2 Database selection

The verification of the WMOPs complexity is made on a subset of the samples used for the selection phase (cf. Table 1).

3.3.3 Narrowband verification

The verification laboratory processes the selected databases (**A3I8** and **MND8**) through the A8 executable and produces the maximum WMOPS score.

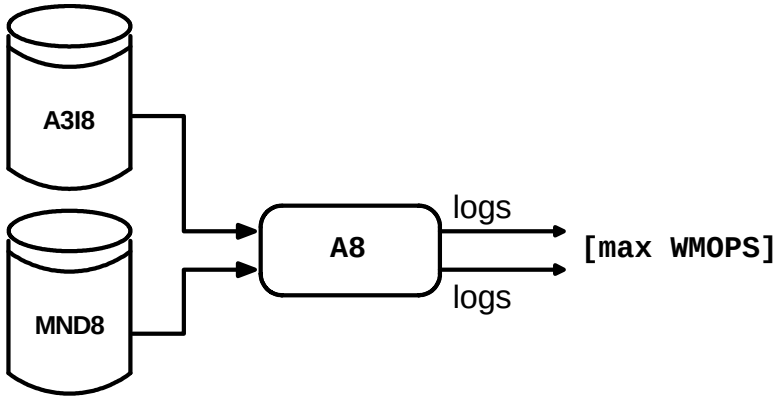

Figure 3: Verification of the complexity of the narrowband candidate

The platform used for the complexity verification is not relevant for the purpose of the WMOPS complexity verification.

3.3.4 Wideband verification

The verification laboratory processes the selected databases (**A3I16**) through the A16 executable and produces the maximum WMOPS score.

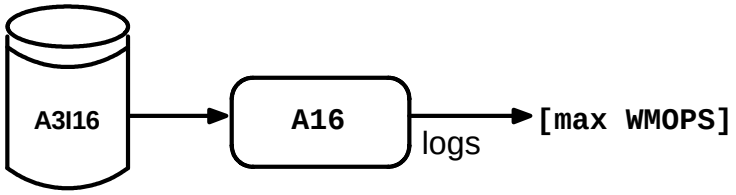

Figure 4: Verification of the complexity of the wideband candidate

The platform used for the complexity verification is not relevant for the purpose of the WMOPS complexity verification.

4. RAM and ROM Complexity verification

4.1 Motivation

The memory used by the fixed-point ANSI-C source code must meet the design constraints (cf. [9]). The memory complexity of the candidate will be estimated from the source code.

Bandwidth	ROM design constraint	RAM design constraint
narrowband	≤20 kwords	≤7 kwords
wideband	≤34 kwords	≤8 kwords

Table 3: memory requirements for the SES candidate (16-bit words)

4.2 Definition

The RAM memory used by the software is the sum of all the non-const arrays or variables defined with a global visibility, all the static arrays or variables (known as the static memory or permanent allocation) and the maximum amount of RAM required by the stack (known as the scratch memory).

The ROM memory used by the software is the sum of all the const arrays or variables (defined in a global or in local visibility). The ROM memory does not include the program ROM (cf. [9]).

The following sample source code explains how the RAM and the ROM memory are evaluated.

```
Word16        buff[16];
const Word32  tab[32];

Word16
func(void *state, Word16 a, const Word16 v[])
{
  Word16 ret;
  Word16 local_buff[8];
  static Word16 state=START;

  [...]

  return ret;
}
```

Code 1: Example of instrumented C-code

In this small example, the memory complexity would be evaluated as follow:

C instruction	Type of memory	Accounted for
`Word16 buff[16]`	static RAM	16
`const Word32 tab[32]`	ROM	64
`void *state`	stack	push 1
`Word16 a`	stack	push 1
`const Word16 v[]`	stack	push 1
`Word16 ret`	stack	push 1
`Word16 local_buff[8]`	stack	push 8
`static Word16 state`	static RAM	1
`Return`	stack	pop (-12)

Table 4: Example of memory assessment

4.3 Additional definitions

4.3.1 Static RAM array initialization

Arrays that are allocated and initialised in the static RAM are accounted simultaneously in static RAM and in ROM.

4.3.2 Stack array initialization

Arrays that are allocated and initialised in the stack are accounted only in static RAM. Furthermore, the code shall be instrumented with as many `move16()` (resp. `move32()`) basic operations than necessary in order to take into account the actual initialisation process. Here follows a small example:

```
Word16
func_proc(Word16 a, Word32 b)
{
  [...]
  Word16 autoBuff[4]={0x4000, 0x1400, 0xFC00, 0xAFF0};
  move16();move16();move16();move16();

  [...]

  return 0;
}
```

Code 2: Instrumented C-code initializing an array in the stack

Said differently, the process of initialising an array allocated in the stack is formally equivalent to the following C-code fragment:

```
Word16
func_proc(Word16 a, Word32 b)
{
  [...]
  Word16 autoBuff[4];

  autoBuff[0] = 0x4000; move16();
  autoBuff[1] = 0x1400; move16();
  autoBuff[2] = 0xFC00; move16();
  autoBuff[3] = 0xAFF0; move16();
  [...]

  return 0;
}
```

Code 3: Unambiguous equivalent C-code for initializing an array in the stack

4.3.3 Constant value usage

Most C compilers for DSP will inline Word16 and Word32 constant values directly in the assembly language code. Therefore, constant values (such as 0x00400000L and 25798L) will not be included in the data ROM; instead they are included in the program source code.

4.3.4 Summary

The following table sums up the different configurations considered for assessing the complexity and the memory usage regarding the usage of constant values in the reference C-code.

C instruction	Type of memory	Accounted for
`Word16 swRand[4]={…};`	ROM + static RAM	4 each
`Word16 autoBuff[4]={…};`	stack	push 4
`((Word16)0x(vvvv))`	program	transparent
`0x(hhhhllll)L`	program	transparent

Table 4: Memory assessment for initialization of arrays and constant value usage

4.3.5 Example C-code

This following imaginary sample code (which does nothing in particular) illustrates different cases that shall be taken into account for the memory assessment of the SES codec :

```
/* initialization counting for 4 words in the ROM */
Word16        swRand[4] = {8, 12, -4, -7};

Word16
func_proc(Word16 a, Word32 b)
{
  Word16 idx, idx2;

  /* constant value counting for 0 words ROM */
  Word32 enerLog = 0x00400000L;

  /* initialization counting for 0 word ROM */
  Word16 autoBuff[4] = {0x4000, 0x1400, 0xFC00, 0xAFF0};

  /* enerLog initialization */
  move32();

  /* autoBuff initialization */
  move16();move16();move16();move16();

  [...]
  /* loop preparation */
  idx2 = 0;   move16();
  for (idx=0;idx<4;idx++) {
    [...]
    autoBuff [idx] = swRand[idx2]; move16();
    swRand[idx2] = /* small constant 25798L counting 0 word ROM */
       extract_h(L_shr(L_add(25798L,
                             L_mult(swRand[idx2], 10037)),2));
    move16();
```

```
    [...]
  }

  [...]

  return 0;
```

Code 4: Sample instrumented C-code

4.4 ROM verification

The source code is used to evaluate the ROM complexity. The amount of ROM memory used by the candidate, as evaluated by the verification laboratories, is compared to the design constraints (cf. Table 3).

4.5 RAM verification

4.5.1 Permanent RAM verification

The source code is used to evaluate the RAM usage that is not related to the use of the stack. The verification laboratory enumerates all the array and variable definitions corresponding to a permanent allocation.

4.5.2 Stack verification

The source code is used to evaluate the stack usage. The verification laboratory builds the calling tree of the source code and evaluates the worst case for the stack usage.

4.5.3 Conclusion

The verification laboratory sums the amount of static RAM and the maximum amount of RAM required by the stack. The amount of RAM memory is compared to the design constraints (cf. Table 3).

5. Workplan

5.1 Verification laboratories

The verification will be performed by STMicroelectronics (contact is stephan.tassart@st.com) and IBM (contact is sorin@il.ibm.com).

Task	Company
bit-exactness verification, narrowband,linux (cf. 2.3.1)	ST
bit-exactness verification, wideband linux (cf. 2.3.2)	ST
bit-exactness verification, narrowband AIX (cf. 2.3.1)	IBM
source code verification (cf. 3.3.2)	ST
WMOPS verification, narrowband (cf. 3.3.3)	ST
WMOPS verification, wideband (cf. 3.3.4)	ST
RAM verification, narrowband (cf. 4.4)	ST
RAM verification, wideband (cf. 4.4)	ST
ROM verification, narrowband (cf. 4.4)	ST
ROM verification, wideband (cf. 4.4)	ST

5.2 Schedule

The workplan is organized as follow:

Date	Actions
19th Dec. 2003	Agree the verification plan by correspondence

16th Feb. 2004	Complete legal agreements with Alcatel for the A3I8 and A3I16 speech databases. Verification laboratories to obtain A3I8 and A3I16.
19th Feb. 2004	Complete legal agreements (NDA) with Motorola for the X-AFE source code.
5th Mar. 2004	Complete legal agreements with Nokia for MND8 speech database.
16th Feb. 2004	The I/O interface and the format of the log files of the X-AFE candidate are provided to the verification laboratories.
1st Mar. 2004	The testing laboratories to provide the executables (i.e. `T8_linux` and `T16_linux`) to the verification laboratories.
23rd –27th Feb.	Meeting SA4#30 – Malaga
1st Mar. 2004	DSR supporting companies to provide the source code to the verification laboratories. The verification laboratories compile the source code and obtain a binary (i.e. `B8` and `B16_linux`).
1st - 3rd Mar.	Bit-exactness verification: `B8` versus `T8_linux` on A3I8.
1st - 3rd Mar.	Verification of the source code instrumentation.
4th - 5th Mar.	Complexity wMOPs verification: `A8` on A3I8.
1st - 10th Mar.	Verification of the RAM and ROM figures.
8th Mar.-10th Mar.	Complexity wMOPs verification: `A8` on A3I16.
10th Mar. 2004	Conference call: discussion of partial verification results.
10th Mar. 2004	Verification laboratories to obtain MND8.
11th - 12th Mar.	Bit-exactness verification : `B16_linux` versus `T16_linux` on A3I16.
11th - 12th Mar.	Bit-exactness verification : `T8_AIX` versus `T8_linux` on A3I8.
15th Mar. 2004	Partial verification report completed: memory assessment completed, wMOPs assessment partially completed (A3I8, A3I16), bit-exactness verification partially completed (A3I8, A3I16)
15th - 17th Mar.	Meeting TSG SA4#23
15th - 17th Mar.	Bit-exactness verification : `B8_linux` versus `T8_linux` on MND8.
18th - 19th Mar.	Complexity wMOPs verification: `A8` on MND8.
26th Mar.	Verification report completed.

6. References

[1] S4-030745 “SES codec verification”

[2] S4-030852 “SES Workplan version 8.0”

[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] S4-030853 “Draft Report SQ and AUC ad-hoc sessions during SA4#29 plenary meeting”

[6] S4-030866 “Consideration of DSR executable code update to ASR vendors”

[7] IEEE 754-1985 Standard for Binary Floating-Point Arithmetic

[8] IEEE 854-1987 Standard for Radix-Independent Floating-Point Arithmetic

[9] S4-030248 “Design Constraints for default codec for speech enabled services (SES)”

[10] S4-030543 “Test and processing plan for default codec evaluation for speech enabled services (SES)”

[11] S4-030540 “Recommendation Criteria for default codec for speech enabled services (SES)”