3GPP TS 26.268 V16.0.0 (2020-07)

Technical Specification

3rd Generation Partnership Project;

Technical Specification Group Services and System Aspects;

eCall Data Transfer;

In-band modem solution;

ANSI-C reference code

(Release 16)



The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP.  
The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented.   
This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification.  
Specifications and reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices.

Keywords

GSM, UMTS, emergency, modem

***3GPP***

Postal address

3GPP support office address

650 Route des Lucioles - Sophia Antipolis

Valbonne - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet

http://www.3gpp.org

***Copyright Notification***

No part may be reproduced except as authorized by written permission.  
The copyright and the foregoing restriction extend to reproduction in all media.

© 2020, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).

All rights reserved.

UMTS™ is a Trade Mark of ETSI registered for the benefit of its members

3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  
LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners

GSM® and the GSM logo are registered and owned by the GSM Association

Contents

Foreword [4](#__RefHeading___Toc517358931)

1 Scope [5](#__RefHeading___Toc517358932)

2 References [5](#__RefHeading___Toc517358933)

3 Abbreviations [5](#__RefHeading___Toc517358934)

4 C code structure [5](#__RefHeading___Toc517358935)

4.1 Contents of the C source code [6](#__RefHeading___Toc517358936)

4.2 Program execution [7](#__RefHeading___Toc517358937)

4.3 Variables, constants and tables [9](#__RefHeading___Toc517358938)

4.3.1 Description of constants used in the C-code [9](#__RefHeading___Toc517358939)

4.3.2 Type Definitions [10](#__RefHeading___Toc517358940)

4.3.3 Description of fixed tables used in the C-code [14](#__RefHeading___Toc517358941)

4.3.4 Static variables used in the C-code [15](#__RefHeading___Toc517358942)

4.4 Functions of the C Code [15](#__RefHeading___Toc517358943)

4.4.1 Interface functions [16](#__RefHeading___Toc517358944)

4.4.2 IVS transmitter functions [18](#__RefHeading___Toc517358945)

4.4.3 PSAP receiver functions [19](#__RefHeading___Toc517358946)

4.4.4 PSAP transmitter functions [22](#__RefHeading___Toc517358947)

4.4.5 IVS receiver functions [22](#__RefHeading___Toc517358948)

4.4.6 Synchronization functions (IVS and PSAP) [22](#__RefHeading___Toc517358949)

4.4.7 Control link functions [24](#__RefHeading___Toc517358950)

4.4.8 Other utility functions (IVS and PSAP) [26](#__RefHeading___Toc517358951)

Annex A (informative): Change history [27](#__RefHeading___Toc517358952)

# Foreword

The present document has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

# 1 Scope

The present document contains an electronic copy of the ANSI‑C code for the eCall in-band modem solution for reliable transmission of MSD data from IVS to PSAP via the speech channel of cellular networks. The ANSI‑C code is necessary for a bit exact implementation of the IVS modem and PSAP modem described in 3GPP TS 26.267 [1].

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TS 26.267: "eCall Data Transfer; In-band modem solution; General description".

See also the references in 3GPP TS 26.267 [1].

# 3 Abbreviations

For the purpose of the present document, the following abbreviations apply:

ACK Acknowledgement

ANSI American National Standards Institute

CRC Cyclic Redundancy Check

FEC Forward Error Correction

GSM Global System for Mobile communications

HARQ Hybrid Automatic Repeat-reQuest

I/O Input/Output

IVS In-Vehicle System

MSD Minimum Set of Data

NACK Negative Acknowledgement

PCM Pulse Code Modulation

PSAP Public Safety Answering Point

RAM Random Access Memory

ROM Read Only Memory

RX Receive

TX Transmit

# 4 C code structure

This clause gives an overview of the structure of the bit‑exact C code and provides an overview of the contents and organization of the C code attached to the present document.

The C code has been verified on the following systems:

- Windows XP SP2 and Microsoft Visual Studio V8.0;

- Linux (Suse Linux) using the gcc v3.4.2 and v4.1.2 compilers.

## 4.1 Contents of the C source code

The distributed files with suffix "c" contain the source code and the files with suffix "h" are the header files.

Further explanation on the files is given in the Readme.txt file, which is reproduced in part here:

Package Contents

----------------

folder 'ecall':

Contains the complete eCall ANSI C fixed-point reference source code.

modem\_ivs.c : top-level modem implementation for IVS

modem\_psap.c : top-level modem implementation for PSAP

modemx.h : header file for both modem\_ivs.c and modem\_psap.c

ecall\_defines.h : compile time options and preprocessor constants

ecall\_control.h : header file control message handling

ecall\_fec.h : header file FEC encoder and decoder

ecall\_modem.h : header file modulator and demodulator

ecall\_sync.h : header file synchronization

ecall\_rom.h : header file ROM data

ecall\_control.c : control message handling

ecall\_fec.c : FEC encoder and decoder

ecall\_modem.c : modulator and demodulator

ecall\_sync.c : synchronization

ecall\_rom.c : ROM data

folder 'test\_setup':

Contains the eCall software simulation framework, to be compiled

and run on MS Windows systems.

folder 'test\_vec':

Contains binary PCM data (104 files) and receiver/transmitter port logs

in ASCII format (104 files) to test the eCall IVS and PSAP modems.

The PCM format is 16 bit signed, little endian, at 8 kHz sampling rate.

The data files reflect 26 test cases and were generated from the eCall

simulation framework.

campaign\_short.txt : configuration file for the 26 test cases

pcmdlout<index>.pcm : output PCM data of DL vocoder = input to IVS

pcmulout<index>.pcm : output PCM data of UL vocoder = input to PSAP

pcmdlin<index>.pcm : test vectors for PSAP modem output

pcmulin<index>.pcm : test vectors for IVS modem output

portivsrx<index>.txt : test vectors for IVS port logs (receiver)

portivstx<index>.txt : test vectors for IVS port logs (transmitter)

portpsaprx<index>.txt : test vectors for PSAP port logs (receiver)

portpsaptx<index>.txt : test vectors for PSAP port logs (transmitter)

standalone.c

main() wrapper to run the IVS or PSAP modem on prestored PCM files or

receiver/transmitter port logs. To get a list of command-line options,

invoke the corresponding executable with option '-h' (help).

standalone.h

header file for standalone.c

Makefile.win

NMAKE Makefile for Microsoft Visual Studio 2005 and above:

Builds 'standalone.exe' from standalone.c and the eCall sources,

build options are RELEASE and DEBUG.

Makefile.glx

GNU Linux Makefile using gcc

Builds 'standalone' from standalone.c and the eCall sources,

build options are RELEASE and DEBUG.

verify.bat

Windows batch file

Runs 'standalone.exe' in six different modem modes on the 26 test cases

contained in folder 'test\_vec' and performs a test vector comparison to

the respective output PCM and port log data.

verify.sh

Linux shell script

Runs 'standalone' in mode '-m ivs' and '-m psap' on 26 test cases

(folder 'pcm') and performs a test vector comparison to the respective

modem output PCM data.

## 4.2 Program execution

An explanation on code compilation and execution is given in the readme.txt file, which is reproduced in part here:

Getting Started

---------------

3GPP TS 26.268 provides the eCall modem source code, a software simulation

framework, and a standalone wrapper that allows to run the IVS or PSAP modem

on prestored reference data.

The following functions represent the eCall modem interface and invoke the

respective receiver and transmitter implementation of each modem:

\* void IvsReset(const Ord8 \*msd, int length);

\* void IvsProcess(Int16 \*pcm);

\* void IvsSendStart(void);

\* void PsapReset(void);

\* void PsapProcess(Int16 \*pcm);

\* void PsapSendStart(void);

\* void PsapSendHlack(const Ord8 data);

The external application must in addition implement the callback functions:

\* void IvsCatchEvent(IvsEvent ie);

\* void IvsReceiveAck(void);

\* void IvsReceiveHlack(const Ord8 data);

\* void PsapCatchEvent(PsapEvent pe);

\* void PsapReceiveMsd(const Ord8 \*msd, int length);

\* void Abort(const char \*format, ...);

\* void LogInfo(const char \*format, ...);

IvsCatchEvent and PsapCatchEvent inform about relevant modem events and can

be used to perform actions such as muting or unmuting of the voice channel.

The other callback functions indicate...

\* IvsReceiveAck : reception of the lower-layer ACK,

\* IvsReceiveHlack : reception of the HLACK message,

\* PsapReceiveMsd : successful MSD reception.

Abort and LogInfo should implement a variadic error and printlog handler,

respectively. See standalone.c for sample implementations of all callback

functions.

For a real-time simulation over 3GPP FR and AMR vocoders and to log PCM data

as input to the standalone wrapper, the eCall sources have to be integrated into a simulation framework; folder 'test\_setup' contains the one as used in

the 3GPP selection tests.

In order to compile and run the eCall modem code, follow the instructions

given below. For code testing, two batch files have been provided:

\* verify.bat : MS Windows systems

\* verify.sh : Linux systems

For each of the 26 test cases of campaign\_short.txt in folder 'test\_vec',

they run the standalone wrapper in six different modem modes (three IVS and

three PSAP modes). The resulting PCM and port log files in folder 'out' are

finally compared to the test vectors in folder 'test\_vec'.

In modes 'psap' and 'psaprx', you should see an MSD success message at the

end of each test case.

Code Compilation

----------------

MS Windows systems

Compilation assumes an installation of MS Visual Studio 2005 or above.

To set the environment variables for building, run 'vcvars32.bat' which

should be contained in the 'bin' subfolder of your VC installation.

To build standalone.exe from standalone.c and the eCall sources (or to

perform cleanup), run

nmake /f Makefile.win

nmake /f Makefile.win clean

The source code should compile without any errors or warnings.

Run 'verify.bat' to verify the executable against the test vectors.

GNU Linux systems

Compilation under Linux has been tested with

\* GNU Make version 3.81

\* gcc version 4.1.3 and 4.2.4

For building the executable 'standalone' and cleanup, use

make -f Makefile.glx

make -f Makefile.glx clean

On the platforms tested, the code compiled without errors or warnings.

Run 'verify.sh' to verify the executable against the test vectors.

Simulation Framework

--------------------

The eCall software simulation framework is provided in folder 'test\_setup'.

Important remarks:

\* See LICENSE.TXT and README.TXT for terms of usage!

\* The G.711 software is part of ITU-T Rec. G.191, (C) ITU 2000.

Distributed with the authorization of ITU as part of the test setup

software for 3GPP TS 26.268.

\* The framework must be compiled and run on MS Windows systems, as the

FR and AMR vocoders are attached to it in form of Windows executables

and via Windows specific API functions.

To build (or clean) the framework together with the eCall IVS and PSAP,

change to subfolder 'c' of 'test\_setup' and run (remember 'vcvars32.bat')

nmake /f makefile\_ecall

nmake /f makefile\_ecall clean

The framework has the five callback functions of above already implemented.

By default, the binaries (\*.exe \*.lib) are generated in the 'bin' subfolder.

You can use the following two batch files to run the executables:

demosim.bat : runs testsim\_demo.exe

demosock.bat : runs testlab.exe and modem\_demo.exe in socket mode

## 4.3 Variables, constants and tables

### 4.3.1 Description of constants used in the C-code

This clause contains a listing of all global constants defined in ecall\_defines.h., together with some explanatory comments.

**Constant Value Description**

#define MAX(a,b) ((a)>(b) ? (a) : (b))

#define MIN(a,b) ((a)<(b) ? (a) : (b))

#define ABS(a) ((a)<0 ? (-a) : (a))

#define SIGN(a) ((a)<0 ? (-1) : (1))

#define PCM\_LENGTH 160 length of PCM frame

#define MSD\_MAX\_LENGTH 140 length of MSD message (bytes)

/\* Synchronization \*/

#define SYNC\_BADCHECK (3) sync consecutive bad check

#define SYNC\_BADTRACK (4) sync consecutive bad track

#define SYNC\_IDXLEN (75) sync index length

#define SYNC\_THRESHOLD (10e6) sync threshold

#define LOCK\_RESYNC (2) messages to lock after sync loss

#define LOCK\_START\_UL (2) START messages to lock sync (UL)

#define LOCK\_START\_DL (3) START messages to lock sync (DL)

#define FAIL\_RESTART (3) number of START messages to restart

#define NRF\_WAKEUP (3) number of wakeup frames

#define NRF\_SYNC (13) length of sync in frames

#define NRF\_OBSERVE (10) number of sync observer frames

#define NRF\_RESYNC (60) resync frames after sync loss

#define NRS\_CP (2) number of samples next to peaks

#define NRS\_TRACK (240) number of samples to track

#define NRS\_CHECK (480) number of samples to check

#define PNSEQ\_OSF (22) "oversampling" rate of PN sequence

#define PEAK\_DIST\_PP (30\*PNSEQ\_OSF) distance outer positive peaks

#define PEAK\_DIST\_NN (54\*PNSEQ\_OSF) distance negative peaks

#define PEAK\_DIST\_PN (12\*PNSEQ\_OSF) distance positive to negative

/\* Uplink/Downlink format \*/

#define ARQ\_MAX (8) number of redundancy versions

#define NRB\_TAIL (3) number of encoder tail bits

#define NRB\_CRC (28) order of CRC polynomial

#define NRB\_INFO (8\*MSD\_MAX\_LENGTH)

#define NRB\_INFO\_CRC (8\*MSD\_MAX\_LENGTH + NRB\_CRC)

#define NRB\_CODE\_ARQ (1380)

#define NRB\_CODE\_BUFFER (3\*(8\*MSD\_MAX\_LENGTH + NRB\_CRC) + 4\*NRB\_TAIL)

#define SET\_LLMSG (16) set size lower-layer messages

#define SET\_HLMSG (16) set size higher-layer messages

#define NRF\_DLDATA (3) downlink data frames

#define NRF\_DLMUTE1LL (3) 1st muting lower-layer message

#define NRF\_DLMUTE1HL (1) 1st muting higher-layer message

#define NRF\_DLCHUNK (NRF\_SYNC + NRF\_DLMUTE1HL + 2\*NRF\_DLDATA)

/\* IVS/PSAP processing \*/

#define NRF\_MEMCTRL (7)

#define NRS\_MEMSYNC (508 + 38\*NRS\_CP)

#define IVS\_THRESHOLD (40000) threshold for control messages

#define IVS\_GOSTART (6) threshold for unreliable START

#define IVS\_TXFAST (10) fast modulator mode NACK condition

#define IVS\_TXINC (87) sample increment at restart

#define PSAP\_NUMSTART (500) number of START messages

#define PSAP\_NUMACK (5) number of ACK messages

#define PSAP\_NUMHLACK (5) number of PSAP HLACK messages

#define PSAP\_THRESHOLD (40) threshold for modulator type

#define FEC\_VAR (30206) variance: 1/4550000 in Q37

#define FEC\_MEAN (0xB9999A) mean: 5.8 in Q21

#define FEC\_ITERATIONS (8) number of decoder iterations

#define FEC\_STATES (8) number of decoder states

#define IntLLR Int16 size of soft bit buffer variables

#define LLR\_MAX ((Int32)(0x7fff-1))

#define LOGEXP\_RES (401) resolution of LOGEXP table

#define LOGEXP\_DELTA (-6) determines internal Q-factor

#define LOGEXP\_QIN (8) input Q-factor of LLR values

### 4.3.2 Type Definitions

The following type definitions have been used, which are defined in ecall\_defines.h, ecall\_modem.h, ecall\_sync.h, and modemx.h:

**Definition Description**

typedef enum { False, True } Bool; boolean variable

typedef enum { Minus = -1,

Zero,

Plus } Tern; ternary variable

typedef signed char Int8; 8 bit signed variable

typedef signed short int Int16; 16 bit signed variable

typedef signed int Int32; 32 bit signed variable

typedef unsigned char Ord1; binary symbol

typedef unsigned char Ord8; 8 bit unsigned variable

typedef unsigned short int Ord16; 16 bit unsigned variable

typedef unsigned int Ord32; 32 bit unsigned variable

typedef enum {

ModUndef,

Mod3bit4smp,

Mod3bit8smp

} ModType; modulator type for uplink transmission

typedef struct {

ModType type; identifies modulator type

Int16 bpsym; bits per symbol

Int16 spmf; samples per modulation frame

Int16 mfpf; modulation frames per frame = PCM\_LENGTH/spmf

Int16 decpos1; position 1st decoding trial

Int16 decpos2; position 2nd decoding trial

Int16 wutperiod; wakeup tone period in samples

Int16 nfmute1; number of muting frames 1st interval

Int16 nfmute4; number of muting frames 4th interval

Int16 nfmuteall; number of muting frames total

Int16 nfdata; number of data frames = NRB\_CODE\_ARQ/(mfpf\*bpsym)

const Int16 \*ulPulse;

const Int16 \*ulPulseMatch;

const Int16 \*mgTable;

const Int16 \*wakeupSin;

const Int16 \*wakeupCos;

} ModState; modulator state for uplink transmission

typedef struct {

Int32 \*mem; /\* memory for sync \*/

Int32 \*memWakeup; /\* memory for wakeup tone detector \*/

SyncSub syncPos; /\* regular sync (non-inverted) \*/

SyncSub syncNeg; /\* inverted sync \*/

Int32 amplitude[3]; /\* amplitudes (average, maximum, memory) \*/

Int32 shape[2\*NRS\_CP+1]; /\* shape of peak causing a sync event \*/

Int32 sign; /\* sync sign marker \*/

Bool flag; /\* indicates sync success \*/

Tern invert; /\* indicates sync inversion \*/

Bool resync; /\* indicates resynchronization event \*/

Int16 delay; /\* synchronization delay \*/

Int16 delayMem; /\* synchronization delay (memory) \*/

Int16 npeaks; /\* number of sync peaks detected \*/

Int16 npeaksMem; /\* number of sync peaks detected (memory) \*/

Int16 events; /\* number of subsequent equal sync events \*/

Tern check; /\* indicates sync check result (ternary variable) \*/

Int16 checkCnt; /\* counter for subsequent sync check failures \*/

Int16 index; /\* frame reference for sync evaluation \*/

} SyncState;

typedef struct {

Int32 amplitude[2]; /\* amplitudes (average, maximum) \*/

Int32 shape[2\*NRS\_CP+1]; /\* shape of peak causing a sync event \*/

Int32 sign; /\* sync sign marker \*/

Bool flag; /\* indicates sync success \*/

Int16 delay; /\* synchronization delay \*/

Int16 npeaks; /\* number of sync peaks detected \*/

Int16 npeaksChk; /\* number of sync peaks detected by sync check \*/

} SyncSub;

typedef enum {

DlMsgNoop = -2,

DlMsgReset,

DlMsgStart,

DlMsgNack,

DlMsgAck,

DlMsgHlack = SET\_LLMSG

} DlData; downlink message identifiers

typedef enum {

DlCntStart = -2,

DlCntWait,

DlCntNext

} DlCount; downlink message counter

typedef enum {

IVSEVENT\_IDLE,

IVSEVENT\_SENDINGSTART,

IVSEVENT\_SENDINGMSD,

IVSEVENT\_RESTARTMSD,

IVSEVENT\_CONTROLSYNC,

IVSEVENT\_CONTROLLOCK,

IVSEVENT\_LLACKRECEIVED,

IVSEVENT\_HLACKRECEIVED,

IVSEVENT\_TIMEOUT

} IvsEvent;

typedef enum {

IvsIdle,

IvsTrigger,

IvsStart,

IvsSendMsd,

IvsAck,

IvsHlack

} IvsState; IVS state identifiers

typedef struct {

IvsState state; IVS receiver state

CtrlRxData ctrl; IVS control struct

SyncState sync; IVS sync struct

Int16 dlData; downlink message symbol

Int16 dlIndex; donwlink frame counter

Int16 dlMsgCnt; downlink message counter

Int16 memCtrl[NRF\_MEMCTRL\*PCM\_LENGTH];

Int32 memSync[NRS\_MEMSYNC];

} IvsRxData;

typedef struct {

IvsState state; IVS transmitter state

CtrlTxData ctrl; IVS control struct

ModState mod; IVS modulator struct

Int16 delay; transmit offset in samples

Int16 rv; redundancy version

Int16 ulN; uplink number of frames

Int16 ulIndex; uplink frame counter

Int16 ulDelay; uplink transmit offset in samples

Int16 dlMsgOld; previous control message

Bool pendingStart; /\* indicates pending START message \*/

Int16 overallNack; /\* cumulative NACK count \*/

Int16 stateCnt[SET\_LLMSG + 1]; state counters

Int16 stateIgn[SET\_LLMSG + 1]; counter for unreliable messages

Ord1 memCode[NRB\_CODE\_BUFFER];

Int16 memDelay[2\*PCM\_LENGTH];

} IvsTxData;

typedef struct {

IvsRxData rx; IVS receiver struct

IvsTxData tx; IVS transmitter struct

} IvsData;

typedef enum {

PSAPEVENT\_IDLE,

PSAPEVENT\_SENDINGSTART,

PSAPEVENT\_SENDINGRESTART,

PSAPEVENT\_SENDINGLLACK,

PSAPEVENT\_SENDINGHLACK,

PSAPEVENT\_CONTROLSYNC,

PSAPEVENT\_CONTROLLOCK,

PSAPEVENT\_MSDSYNC,

PSAPEVENT\_MSDRECEIVED,

PSAPEVENT\_TIMEOUT

} PsapEvent;

typedef enum {

PsapIdle,

PsapTrigger,

PsapStart,

PsapNack,

PsapAck,

PsapHlack,

} PsapState; PSAP state identifiers

typedef struct {

PsapState state; PSAP receiver state

CtrlRxData ctrl; PSAP control struct

SyncState sync; PSAP sync struct

ModState mod; PSAP modulator struct

Int16 rv; redundancy version

Int16 ulN; uplink number of frames without muting

Int16 ulIndex; uplink frame counter

Int16 mgIndex; uplink position in muting gap table

Int16 ulTrials; uplink decoding trails

Int16 ulSyncTail; sync observation counter after sync success

Ord8 dlHlackData; downlink higher-layer message (4 bits)

Int16 dlData; downlink message symbol

Int16 dlIndex; donwlink frame counter

Int16 dlMsgCnt; downlink message counter

Ord8 \*msd; MSD in byte representation

Ord1 \*msdBin; MSD in binary representation

Int16 \*memCtrl; buffer for control and data demodulation

IntLLR \*memCode; soft bit buffer for decoding

char buffer[0

+ sizeof(IntLLR)\* NRB\_CODE\_ARQ

+ sizeof(Int16) \* NRF\_MEMCTRL\*PCM\_LENGTH

+ sizeof(Int32) \* NRS\_MEMSYNC

+ sizeof(Int32) \* 2\*(NRF\_SYNC+1)];

} PsapRxData;

typedef struct {

CtrlTxData ctrl; PSAP control struct

} PsapTxData;

typedef struct {

PsapRxData rx; PSAP receiver struct

PsapTxData tx; PSAP transmitter struct

Int16 msgCounter; message counter

} PsapData;

typedef enum {

CtrlRxIdle,

CtrlRxSync,

CtrlRxLock,

CtrlTxIdle,

CtrlTxSend

} PortState;

typedef struct {

PortState state; port state

Bool invert; port inversion flag

union {

CtrlTxPort tx; port control transmitter

CtrlRxPort rx; port control receiver

} u;

const char \*owner; port owner identification

} CtrlPort;

typedef struct {

Int16 dlData; message symbol

Int16 dlIndex; message frame counter

} CtrlTxPort;

typedef struct {

Int16 dlData; detected message symbol

Int16 dlMetric; receiver metric

} CtrlRxPort;

typedef struct {

CtrlPort port; port struct

} CtrlTxData;

typedef struct {

CtrlPort port; port struct

SyncState \*sync; pointer to sync struct

Int16 \*buffer; pointer to control receiver buffer

Ord8 dlHlackData; downlink higher-layer message (4 bits)

Tern dlRead; sync indication (ternary variable)

Int16 dlIndex; internal frame counter

Int16 dlSyncLock; number of sync events required

} CtrlRxData;

### 4.3.3 Description of fixed tables used in the C-code

This clause contains a listing of all fixed tables (ROM) defined in ecall\_rom.c.

**Type/Constant Dimension Description**

/\* Synchronization \*/

const Int16 wakeupSin500 [16] sine waveform at 500 Hz

const Int16 wakeupCos500 [16] cosine waveform at 500 Hz

const Int16 wakeupSin800 [10] sine waveform at 800 Hz

const Int16 wakeupCos800 [10] cosine waveform at 800 Hz

const Int16 syncPulseForm [5] sync pulse

const Int16 syncSequence [15] sync pulse sequence

const Int16 syncIndexPreamble [SYNC\_IDXLEN] sync pulse positions

const Int16 syncFrame [10\*PCM\_LENGTH] predefined synchronization signal

/\* Uplink/Downlink format \*/

const Int16 indexBits [24] bit positions for turbo decoder

// fast modulator mode:

const Int16 m4smp\_ulPulse [16] uplink waveform

const Int16 m4smp\_ulPulseMatch [64] matched filtered uplink waveform

const Int16 m4smp\_mgTable [66] table indicating muting gaps

// robust modulator mode:

const Int16 m8smp\_ulPulse [32] uplink waveform

const Int16 m8smp\_ulPulseMatch [128] matched filtered uplink waveform

const Int16 m8smp\_mgTable [116] table indicating muting gaps

const Int16 dlPcmData [4][NRF\_DLDATA\*PCM\_LENGTH] downlink transmit signal

const Int16 dlPcmDataMatch [4][NRF\_DLDATA\*PCM\_LENGTH] DL MF signal

/\* FEC encoder/decoder \*/

const Ord16 stateTransMat [8][2] FEC: state transitions

const Ord16 stateTrans [16] FEC: state transitions

const Ord16 revStateTransMat [8][2] FEC: reverse state transitions

const Ord16 revStateTrans [16] FEC: reverse state transitions

const Ord1 outputParityMat [8][2] FEC: output parity indicator

const Ord1 outputParity [16] FEC: output parity indicator

const Ord1 crcPolynomial [NRB\_CRC+1] coefficients of CRC polynomial

const Ord1 scramblingSeq [NRB\_INFO\_CRC] bit scrambling sequence

const Ord16 interleaverSeq [NRB\_INFO\_CRC] interleaver sequence

const Ord16 redVerIndex [8][NRB\_CODE\_ARQ] index vector for HARQ process

const IntLLR logExpTable [LOGEXP\_RES] lookup table (logExp function)

### 4.3.4 Static variables used in the C-code

This clause contains a listing of static variables (RAM) defined in source files.

**Definition Description**

IvsData ivs IVS static memory

PsapData psap PSAP static memory

IntLLR chLLRbuffer[NRB\_CODE\_BUFFER] soft bit buffer of turbo decoder

## 4.4 Functions of the C Code

This clause contains the headers of the employed IVS and PSAP functions. They correspond to a large extent to the functional description of the IVS and PSAP provided in 3GPP TS 26.267 [1].

Figure 1 gives an overview of the most important functions and their hierarchical relation.

Figure 1: Hierarchical function overview

### 4.4.1 Interface functions

/\*============================================================================\*/

/\* IVS implementation: IvsReset \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Reset of IVS before the reception of a new MSD \*/

/\* \*/

/\* In: const Ord8\* msd -> MSD to be transmitted \*/

/\* int length -> MSD length (equal to MSD\_MAX\_LENGTH) \*/

/\*----------------------------------------------------------------------------\*/

void IvsReset(const Ord8 \*msd, int length)

void IvsRxReset(void)

void IvsTxReset(const Ord8 \*msd, int length)

/\*============================================================================\*/

/\* IVS implementation: IvsProcess \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: IVS modem function that processes the PCM data \*/

/\* \*/

/\* InOut: Int16\* pcm <-> input and output frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void IvsProcess(Int16 \*pcm)

void IvsRxProcess(const Int16 \*pcm)

void IvsTxProcess(Int16 \*pcm)

/\*============================================================================\*/

/\* IVS implementation: IvsSendStart \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Initiates IVS to trigger the transmission of SEND messages \*/

/\*----------------------------------------------------------------------------\*/

void IvsSendStart(void);

/\*============================================================================\*/

/\* IVS implementation: IvsReceiveAck \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: callback function indicating a received ACK message \*/

/\*----------------------------------------------------------------------------\*/

void IvsReceiveAck(void);

/\*============================================================================\*/

/\* IVS implementation: IvsReceiveHlack \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: callback function indicating a received higher layer messages \*/

/\* \*/

/\* In: const Ord8 data -> data symbol identifierer \*/

/\*----------------------------------------------------------------------------\*/

void IvsReceiveHlack(const Ord8 data);

/\*============================================================================\*/

/\* PSAP implementation: PsapSendStart \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Initiates PSAP to trigger the transmission of an MSD \*/

/\*----------------------------------------------------------------------------\*/

void PsapSendStart(void)

/\*============================================================================\*/

/\* PSAP implementation: PsapSendHlack \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Initiates PSAP to send the higher layer messages \*/

/\* \*/

/\* In: const Ord8 data -> data symbol identifierer \*/

/\*----------------------------------------------------------------------------\*/

void PsapSendHlack(const Ord8 data);

/\*============================================================================\*/

/\* PSAP implementation: PsapReset \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Reset of PSAP before the reception of a new MSD \*/

/\*----------------------------------------------------------------------------\*/

void PsapReset(void)

void PsapRxReset(void)

void PsapTxReset(void)

/\*============================================================================\*/

/\* PSAP implementation: PsapProcess \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: PSAP modem function that processes the PCM data \*/

/\* \*/

/\* InOut: Int16\* pcm <-> input and output frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void PsapProcess(Int16 \*pcm)

void PsapRxProcess(const Int16 \*pcm)

void PsapTxProcess(Int16 \*pcm)

### 4.4.2 IVS transmitter functions

/\*============================================================================\*/

/\* IVS FUNCTION: IvsTransmitter \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: IVS transmitter function \*/

/\* \*/

/\* In: const ModState\* ms -> modulator struct \*/

/\* const Ord1\* buffer -> code bit buffer \*/

/\* Int16 rv -> redundancy version \*/

/\* Int16 index -> position within uplink frame \*/

/\* Out: Int16\* pcm <- output data \*/

/\*----------------------------------------------------------------------------\*/

void IvsTransmitter(const ModState \*ms, const Ord1 \*buffer, Int16 \*pcm,

Int16 rv, Int16 index)

/\*============================================================================\*/

/\* UTILITY FUNCTION: IvsTxState \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: IVS state machine evaluating feedback messages \*/

/\* \*/

/\* In: Int16 msg -> new control message symbol \*/

/\* Int16 metric -> receiver metric (-1: ignore symbol) \*/

/\* Bool syncLock -> indicates sync lock of control receiver \*/

/\*----------------------------------------------------------------------------\*/

void IvsTxState(Int16 msg, Int16 metric, Bool syncLock)

/\*============================================================================\*/

/\* IVS FUNCTION: SymbolMod \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: symbol modulator \*/

/\* \*/

/\* In: const ModState\* ms -> modulator struct \*/

/\* Int16 symbol -> symbol index \*/

/\* Out: Int16\* mPulse <- modulated output sequence \*/

/\*----------------------------------------------------------------------------\*/

void SymbolMod(const ModState \*ms, Int16 symbol, Int16 \*mPulse)

/\*============================================================================\*/

/\* IVS FUNCTION: Byte2Bit \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: conversion byte vector to bit vector \*/

/\* \*/

/\* In: Ord8\* in -> vector of input bytes \*/

/\* Int16 length -> length of input \*/

/\* Out: Ord1\* out <- vector of output bits \*/

/\*----------------------------------------------------------------------------\*/

void Byte2Bit(const Ord8 \*in, Ord1 \*out, Int16 length)

/\*============================================================================\*/

/\* ENCODER FUNCTION: FecEncode \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: encoding of MSD \*/

/\* \*/

/\* InOut: Ord1 \*buffer <-> takes info bits and returns coded bits \*/

/\*----------------------------------------------------------------------------\*/

void FecEncode(Ord1 \*buffer)

/\*============================================================================\*/

/\* ENCODER FUNCTION: AttachCrc \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: attaches CRC bits \*/

/\* \*/

/\* In: const Ord1\* infoBits -> input information bits \*/

/\* Out: Ord1\* infoWithCrc <- bits with CRC attached \*/

/\*----------------------------------------------------------------------------\*/

void AttachCrc(const Ord1 \*infoBits, Ord1 \*infoWithCrc)

/\*============================================================================\*/

/\* ENCODER FUNCTION: Scramble \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: bit scrambling \*/

/\* \*/

/\* In: const Ord1\* in -> non scrambled input bit sequence \*/

/\* Out: Ord1\* out <- scrambled output bit sequence \*/

/\*----------------------------------------------------------------------------\*/

void Scramble(const Ord1 \*in, Ord1 \*out)

/\*============================================================================\*/

/\* ENCODER FUNCTION: EncodeTwo \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: encoding of bit sequence \*/

/\* \*/

/\* InOut: Ord1\* codedBits <-> scrambled bits to coded bits \*/

/\*----------------------------------------------------------------------------\*/

void EncodeTwo(Ord1 \*codedBits)

/\*============================================================================\*/

/\* ENCODER FUNCTION: EncodeOne \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: convolutional encoding of each component \*/

/\* \*/

/\* In: Int16 encNr -> component number \*/

/\* InOut: Ord1\* codedBits <-> bits to be encoded \*/

/\*----------------------------------------------------------------------------\*/

void EncodeOne(Ord1 \*codedBits, Int16 encNr)

### 4.4.3 PSAP receiver functions

/\*============================================================================\*/

/\* UTILITY FUNCTION: PsapRxUplink \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: PSAP UL state machine, determines PSAP receiver operation \*/

/\* according to the state \*/

/\* \*/

/\* In: const Int16\* pcm -> input frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void PsapRxUplink(const Int16 \*pcm)

/\*============================================================================\*/

/\* UTILITY FUNCTION: PsapRxDownlink \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: PSAP DL state machine, determines PSAP transmitter operation \*/

/\* according to the state \*/

/\*----------------------------------------------------------------------------\*/

void PsapRxDownlink(void)

/\*============================================================================\*/

/\* PSAP FUNCTION: PsapReceiver \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: PSAP receiver function (decoding is done outside) \*/

/\* \*/

/\* In: const ModState\* ms -> modulator struct \*/

/\* const Int16\* pcm -> input data for demodulation \*/

/\* Out: IntLLR\* softBits <- demodulated soft bit sequence \*/

/\*----------------------------------------------------------------------------\*/

void PsapReceiver(const ModState \*ms, const Int16 \*pcm, IntLLR \*softBits)

/\*============================================================================\*/

/\* PSAP FUNCTION: SymbolDemod \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: symbol demodulator \*/

/\* \*/

/\* In: const ModState\* ms -> modulator struct \*/

/\* const Int16\* mPulse -> received pulse train \*/

/\* Out: IntLLR\* softBits <- demodulated soft bit sequence \*/

/\*----------------------------------------------------------------------------\*/

void SymbolDemod(const ModState \*ms, const Int16 \*mPulse, IntLLR \*softBits)

/\*============================================================================\*/

/\* PSAP FUNCTION: Bit2Byte \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: conversion bit vector to byte vector \*/

/\* \*/

/\* In: const Ord1\* in -> vector of input bits \*/

/\* Int16 length -> length of output \*/

/\* Out: Ord8\* out <- vector of output bytes \*/

/\*----------------------------------------------------------------------------\*/

void Bit2Byte(const Ord1 \*in, Ord8 \*out, Int16 length)

/\*============================================================================\*/

/\* PSAP FUNCTION: MpyLacc \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: multiply 32bit number with 16bit number (32bit result) \*/

/\* \*/

/\* In: Int32 var32 -> 32bit number \*/

/\* Int16 var16 -> 16bit number \*/

/\* Return: Int32 <- result \*/

/\*----------------------------------------------------------------------------\*/

Int32 MpyLacc(Int32 var32, Int16 var16)

/\*============================================================================\*/

/\* DECODER FUNCTION: FecDecode \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: decoding to find the MSD \*/

/\* \*/

/\* In: const IntLLR\* in -> received soft bits \*/

/\* Int16 rv -> redundancy version \*/

/\* Out: Ord1\* out <- decoded MSD in binary representation \*/

/\* Return: Bool <- result of CRC check \*/

/\*----------------------------------------------------------------------------\*/

Bool FecDecode(const IntLLR \*in, Int16 rv, Ord1 \*out)

/\*============================================================================\*/

/\* DECODER FUNCTION: UpdateBuffer \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: update channel LLR buffer with new soft bits \*/

/\* \*/

/\* In: const IntLLR\* softInBits -> received soft bits \*/

/\* Int16 rv -> redundancy version \*/

/\* InOut: IntLLR\* chLLRbuffer <-> decoder buffer \*/

/\*----------------------------------------------------------------------------\*/

void UpdateBuffer(IntLLR \*chLLRbuffer, const IntLLR \*softInBits, Int16 rv)

/\*============================================================================\*/

/\* DECODER FUNCTION: DecodeBuffer \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: decoding of LLR buffer \*/

/\* \*/

/\* In: const IntLLR\* syst1 -> RX systematic soft bits \*/

/\* const IntLLR\* syst2 -> interleaved RX systematic tail bits \*/

/\* const IntLLR\* parity1 -> RX parity soft bits \*/

/\* const IntLLR\* parity2 -> interleaved RX parity soft bits \*/

/\* Out: Ord1\* decBits <- decoded bits \*/

/\*----------------------------------------------------------------------------\*/

void DecodeBuffer(const IntLLR \*syst1, const IntLLR \*syst2,

const IntLLR \*parity1, const IntLLR \*parity2, Ord1 \*decBits)

/\*============================================================================\*/

/\* DECODER FUNCTION: Bcjr \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: BCJR algorithm \*/

/\* \*/

/\* In: const IntLLR\* parity -> received parity soft bits \*/

/\* InOut: IntLLR\* extrinsic <-> extrinsic information \*/

/\*----------------------------------------------------------------------------\*/

void Bcjr(const IntLLR \*parity, IntLLR \*extrinsic)

/\*============================================================================\*/

/\* DECODER FUNCTION: Interleave \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Turbo code interleaver \*/

/\* \*/

/\* In: const IntLLR\* in -> input sequence \*/

/\* Out: IntLLR\* out <- output sequence \*/

/\*----------------------------------------------------------------------------\*/

void Interleave(const IntLLR \*in, IntLLR \*out)

/\*============================================================================\*/

/\* DECODER FUNCTION: Deinterleave \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Turbo code deinterleaver \*/

/\* \*/

/\* InOut: IntLLR\* inout <-> input and deinterleaved output sequence \*/

/\*----------------------------------------------------------------------------\*/

void Deinterleave(IntLLR \*inout)

/\*============================================================================\*/

/\* DECODER FUNCTION: Descramble \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: descrambles decoded bits \*/

/\* \*/

/\* InOut: Ord1\* inout <-> input and output bit sequence \*/

/\*----------------------------------------------------------------------------\*/

void Descramble(Ord1 \*inout)

/\*============================================================================\*/

/\* DECODER FUNCTION: DecodeCrc \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: check CRC of decoded bits \*/

/\* \*/

/\* In: const Ord1\* codedBits -> decoded bit sequence to be checked \*/

/\* Return: Bool <- result of CRC check \*/

/\*----------------------------------------------------------------------------\*/

Bool DecodeCrc(const Ord1 \*codedBits)

/\*============================================================================\*/

/\* DECODER FUNCTION: GammaQ \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: compute gamma values for BCJR algorithm \*/

/\* \*/

/\* In: Int16 k -> bit position \*/

/\* Int16 l -> state \*/

/\* const IntLLR\* parity -> received parity bits \*/

/\* const IntLLR\* extrinsic -> sum of extrinsic and systematic bits \*/

/\* Return: IntLLR <- value of gamma(k,l) \*/

/\*----------------------------------------------------------------------------\*/

IntLLR GammaQ(Int16 k, Int16 l, const IntLLR \*parity, const IntLLR \*extrinsic)

/\*============================================================================\*/

/\* UTILITY FUNCTION: JacLog \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: Jacobian logarithm \*/

/\* \*/

/\* In: IntLLR a -> value one \*/

/\* IntLLR b -> value two \*/

/\* Return: IntLLR <- Jacobian logarithm \*/

/\*----------------------------------------------------------------------------\*/

IntLLR JacLog(Int32 a, Int32 b)

### 4.4.4 PSAP transmitter functions

See control link functions.

### 4.4.5 IVS receiver functions

See control link functions.

### 4.4.6 Synchronization functions (IVS and PSAP)

/\*============================================================================\*/

/\* FUNCTION: Sync \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: main synchronization function \*/

/\* \*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: const Int16\* pcm -> input frame \*/

/\* const char\* caller -> modem identification \*/

/\* Bool invert -> port inversion flag \*/

/\*----------------------------------------------------------------------------\*/

void Sync(SyncState \*sync, const Int16 \*pcm, const char \*caller, Bool invert)

/\*============================================================================\*/

/\* UTILITY FUNCTION: CtrlSync \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: control message sync function \*/

/\* \*/

/\* InOut: CtrlRxData\* control <-> control struct \*/

/\* In: const Int16\* pcm -> input frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void CtrlSync(CtrlRxData \*control, const Int16 \*pcm)

/\*============================================================================\*/

/\* UTILITY FUNCTION: SyncSubPut, SyncSubGet, SyncSubCpy \*/

/\*----------------------------------------------------------------------------\*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* InOut: SyncSub\* ssub <-> sync subsystem \*/

/\*----------------------------------------------------------------------------\*/

void SyncSubPut(SyncState \*sync, SyncSub \*ssub)

void SyncSubGet(SyncState \*sync, SyncSub \*ssub)

void SyncSubCpy(const SyncSub \*ssubIn, SyncSub \*ssubOut)

/\*============================================================================\*/

/\* UTILITY FUNCTION: SyncSubRun \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: sync peak evaluation \*/

/\* \*/

/\* InOut: SyncSub\* ssub <-> sync subsystem \*/

/\* In: const char\* caller -> modem identification \*/

/\* const Int32\* pPos -> positive peaks positions \*/

/\* const Int32\* pCorr -> positive peaks correlation values \*/

/\* const Int32\* nPos -> negative peaks positions \*/

/\* const Int32\* nCorr -> negative peaks correlation values \*/

/\*----------------------------------------------------------------------------\*/

void SyncSubRun(SyncSub \*ssub, const char \*caller,

const Int32 \*pPos, const Int32 \*pCorr,

const Int32 \*nPos, const Int32 \*nCorr)

/\*============================================================================\*/

/\* IVS FUNCTION: SyncCheck \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: check whether locked sync is still valid \*/

/\* \*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: const Int16\* pcm -> input frame \*/

/\* const char\* caller -> modem identification \*/

/\*----------------------------------------------------------------------------\*/

void SyncCheck(SyncState \*sync, const Int16 \*pcm, const char \*caller)

/\*============================================================================\*/

/\* IVS FUNCTION: SyncTrack \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: uplink sync tracker \*/

/\* \*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: Bool invert -> port inversion flag \*/

/\*----------------------------------------------------------------------------\*/

void SyncTrack(SyncState \*sync, Bool invert)

/\*============================================================================\*/

/\* FUNCTION: SyncFilter \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: sync filter implementation \*/

/\* \*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: const Int16\* pcm -> input frame \*/

/\* Bool invert -> port inversion flag \*/

/\*----------------------------------------------------------------------------\*/

void SyncFilter(SyncState \*sync, const Int16 \*pcm, Bool invert)

/\*============================================================================\*/

/\* UTILITY FUNCTION: ToneDetect \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: tone detection at 500 Hz or 800 Hz \*/

/\* \*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: const Int16\* pcm -> input frame \*/

/\*----------------------------------------------------------------------------\*/

void ToneDetect(SyncState \*sync, const Int16 \*pcm)

/\*============================================================================\*/

/\* UTILITY FUNCTION: PeakUpdate \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: update sync peak position \*/

/\* \*/

/\* In: const Int32\* pos -> vector of positions \*/

/\* const Int32\* corr -> vector of correlation values \*/

/\* Int16 dist -> distance to be checked \*/

/\* Return: Int16 <- updated peak position \*/

/\*----------------------------------------------------------------------------\*/

Int16 PeakUpdate(const Int32 \*pos, const Int32 \*corr, Int16 dist)

/\*============================================================================\*/

/\* UTILITY FUNCTION: PeakCheck \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: check sync peaks \*/

/\* \*/

/\* InOut: SyncSub\* ssub <-> sync subsystem \*/

/\* In: const char\* caller -> modem identification \*/

/\* const Bool\* pdet -> vector of peak detection flags \*/

/\* const Int16\* p -> vector of frame numbers \*/

/\* const Int32\* corr(X) -> vector of correlation values \*/

/\* Int16 pos1 -> peak position 1 \*/

/\* Int16 pos2 -> peak position 2 \*/

/\* Int16 npeaks -> number of detected peaks \*/

/\* Int16 delay -> target delay if sync successful \*/

/\*----------------------------------------------------------------------------\*/

void PeakCheck(SyncSub \*ssub,

const char \*caller, const Bool \*pdet, const Int16 \*p,

const Int32 \*corrP, const Int32 \*corrN, const Int32 \*corr,

Int16 pos1, Int16 pos2, Int16 npeaks, Int16 delay)

/\*============================================================================\*/

/\* UTILITY FUNCTION: SyncReset \*/

/\*----------------------------------------------------------------------------\*/

/\* InOut: SyncState\* sync <-> sync struct \*/

/\* In: Int32\* mem -> pointer to sync memory \*/

/\* Int32\* memWakeup -> pointer to sync wakeup memory \*/

/\*----------------------------------------------------------------------------\*/

void SyncReset(SyncState \*sync, Int32 \*mem, Int32 \*memWakeup)

/\*============================================================================\*/

/\* UTILITY FUNCTION: SyncSubReset \*/

/\*----------------------------------------------------------------------------\*/

/\* InOut: SyncSub\* ssub <-> sync subsystem \*/

/\*----------------------------------------------------------------------------\*/

void SyncSubReset(SyncSub \*ssub)

### 4.4.7 Control link functions

/\*============================================================================\*/

/\* UTILITY FUNCTION: CtrlPortName \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: converter for log output \*/

/\* \*/

/\* In: PortOwner owner -> modem identification \*/

/\* Return: const char\* <- port name as character string \*/

/\*----------------------------------------------------------------------------\*/

const char\* CtrlPortName(PortOwner owner)

/\*============================================================================\*/

/\* CONTROL FUNCTION: CtrlTxProcess \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: process function control transmitter \*/

/\* \*/

/\* InOut: CtrlTxData\* control <-> control struct \*/

/\* Int16\* pcm <-> frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void CtrlTxProcess(CtrlTxData \*control, Int16 \*pcm)

/\*============================================================================\*/

/\* UTILITY FUNCTION: CtrlTxMod \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: control message transmitter using prestored sequences \*/

/\* \*/

/\* In: Int16 symbol -> lower-layer or higher-layer message symbol \*/

/\* Int16 index -> position within message frame \*/

/\* Out: Int16\* pcm <- output data \*/

/\*----------------------------------------------------------------------------\*/

void CtrlTxMod(Int16 \*pcm, Int16 symbol, Int16 index)

/\*============================================================================\*/

/\* CONTROL FUNCTION: CtrlRxProcess \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: process function control receiver \*/

/\* \*/

/\* InOut: CtrlRxData\* control <-> control struct \*/

/\* In: const Int16\* pcm -> input frame of 16bit PCM samples \*/

/\*----------------------------------------------------------------------------\*/

void CtrlRxProcess(CtrlRxData \*control, const Int16 \*pcm)

/\*============================================================================\*/

/\* UTILITY FUNCTION: CtrlRxDemod \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: control message receiver \*/

/\* \*/

/\* In: const Int16\* pcm -> input PCM buffer \*/

/\* Out: Int16\* metric <- reliability factor (-1: skip) \*/

/\* Return: Int16 <- demodulated message \*/

/\*----------------------------------------------------------------------------\*/

Int16 CtrlRxDemod(const Int16 \*pcm, Int16 \*metric)

/\*============================================================================\*/

/\* CONTROL FUNCTION: CtrlTxReset \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: reset function control transmitter \*/

/\* \*/

/\* InOut: CtrlTxData\* control <-> control struct \*/

/\* In: const char\* owner -> modem identification \*/

/\*----------------------------------------------------------------------------\*/

void CtrlTxReset(CtrlTxData \*control, const char \*owner)

/\*============================================================================\*/

/\* CONTROL FUNCTION: CtrlRxReset \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: reset function control receiver \*/

/\* \*/

/\* InOut: CtrlRxData\* control <-> control struct \*/

/\* In: const char\* owner -> modem identification \*/

/\* SyncState\* sync -> pointer to sync struct \*/

/\* Int16\* buffer -> pointer to control receiver buffer \*/

/\* Int16 syncLock -> number of sync events required \*/

/\*----------------------------------------------------------------------------\*/

void CtrlRxReset(CtrlRxData \*control, const char \*owner,

SyncState \*sync, Int16 \*buffer, Int16 syncLock)

### 4.4.8 Other utility functions (IVS and PSAP)

/\*============================================================================\*/

/\* UTILITY FUNCTION: SetModState \*/

/\*----------------------------------------------------------------------------\*/

/\* Description: set the modulator state \*/

/\* \*/

/\* In: Int16 modType -> type of modulator to use \*/

/\* InOut: ModState\* ms <-> modulator struct \*/

/\*----------------------------------------------------------------------------\*/

void SetModState(ModState \*ms, ModType modType)

Annex A (informative):  
Change history

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | | | | |
| **Date** | **TSG SA#** | | **TSG Doc.** | | | **CR** | **Rev** | **Subject/Comment** | **Old** | **New** |
| 2009-03 | 43 | | SP-090201 | | |  |  | Approved at TSG SA#43 | 2.0.0 | 8.0.0 |
| 2009-06 | 44 | | SP-090251 | | | 0001 | 1 | Correction of a mismatch with 3GPP TS 26.267 concerning synchronization | 8.0.0 | 8.1.0 |
| 2009-06 | 44 | | SP-090251 | | | 0002 | 1 | Correction concerning modulator initialization | 8.0.0 | 8.1.0 |
| 2009-06 | 44 | | SP-090251 | | | 0003 | 1 | Correction of a mismatch with 3GPP TS 26.267 concerning ACK transmission | 8.0.0 | 8.1.0 |
| 2009-06 | 44 | | SP-090251 | | | 0004 | 1 | Extension of eCall test setup to allow conformance testing of ACK messages | 8.0.0 | 8.1.0 |
| 2009-06 | 44 | | SP-090251 | | | 0005 | 2 | Separation of IVS and PSAP transmitter and receiver functions in the C-code | 8.0.0 | 8.1.0 |
| 2009-09 | 45 | | SP-090565 | | | 0006 | 1 | Integration of higher-layer acknowledgement message | 8.1.0 | 8.2.0 |
| 2009-09 | 45 | | SP-090576 | | | 0007 |  | Integration of IVS-initiated signalling option | 8.1.0 | 8.2.0 |
| 2009-09 | 45 | | SP-090565 | | | 0008 |  | Parameter change in eCall test setup | 8.1.0 | 8.2.0 |
| 2009-09 | 45 | | SP-090565 | | | 0009 |  | Update of receiver-transmitter interfaces for conformance testing | 8.1.0 | 8.2.0 |
| 2009-09 | 45 | | SP-090565 | | | 0010 |  | Corrections and bugfixes of the eCall source code | 8.1.0 | 8.2.0 |
| 2009-12 | 46 | |  | | |  |  | Version for Release 9 | 8.2.0 | 9.0.0 |
| 2010-06 | | 48 | | SP-100297 | 0012 | | 1 | Correction of ACK detection conditions | 9.0.0 | 9.1.0 |
| 2010-06 | | 48 | | SP-100297 | 0014 | | 1 | Detector for handling PCM sample inversion in the network | 9.0.0 | 9.1.0 |
| 2010-06 | | 48 | | SP-100297 | 0016 | | 1 | Feedback signal modifications to increase robustness in the presence of network echo cancellers | 9.0.0 | 9.1.0 |
| 2010-09 | | 49 | | SP-100462 | 0018 | |  | Correction of some errors in the eCall reference code | 9.1.0 | 9.2.0 |
| 2010-09 | | 49 | | SP-100462 | 0020 | |  | Update of eCall test framework software to handle new in-band modem features | 9.1.0 | 9.2.0 |
| 2010-12 | | 50 | | SP-100783 | 0022 | | 1 | Correction of synchronization procedures in the eCall in-band modem | 9.2.0 | 9.3.0 |
| 2010-12 | | 50 | | SP-100783 | 0024 | | 1 | State machine corrections in the eCall in-band modem | 9.2.0 | 9.3.0 |
| 2010-12 | | 50 | | SP-100783 | 0026 | | 1 | Correction of the inversion detector | 9.2.0 | 9.3.0 |
| 2011-03 | | 51 | | SP-110033 | 0028 | |  | Correction of synchronization algorithm for the eCall in-band modem | 9.3.0 | 9.4.0 |
| 2011-03 | | 51 | | SP-110033 | 0030 | |  | Correction of signal inversion operation and callback message event set | 9.3.0 | 9.4.0 |
| 2011-03 | | 51 | |  |  | |  | Version for Release 10 | 9.4.0 | 10.0.0 |
| 2012-09 | | 57 | |  |  | |  | Version for Release 11 | 10.0.0 | 11.0.0 |
| 2014-09 | | 65 | | SP-140455 | 0032 | | 1 | Update to Codec Inversion Detection Algorithm | 11.0.0 | 12.0.0 |
| 2015-12 | | 70 | |  |  | |  | Version for Release 13 | 12.0.0 | 13.0.0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
| 2017-03 | 75 |  |  |  |  | Version for Release 14 | 14.0.0 |
| 2018-06 | 80 |  |  |  |  | Version for Release 15 | 15.0.0 |
| 2020-07 | - | - | - | - | - | Update to Rel-16 version (MCC) | **16.0.0** |