

Document: C.S0020-0-1

Version:

Date: December 1999



3RD GENERATION
PARTNERSHIP
PROJECT 2
"3GPP2"

High Rate Speech Service Option 17 for Wideband Spread Spectrum Communications Systems - Addendum

COPYRIGHT

3GPP2 and its Organizational Partners claim copyright in this document and individual Organizational Partners may copyright and issue documents or standards publications in individual Organizational Partner's name based on this document. Requests for reproduction of this document should be directed to the 3GPP2 Secretariat at secretariat@3gpp2.org. Requests to reproduce individual Organizational Partner's documents should be directed to that Organizational Partner. See www.3gpp2.org for more information.

TIA/EIA INTERIM STANDARD

High Rate Speech Service Option 17 for Wideband Spread Spectrum Communications Systems

Addendum 1

TIA/EIA/IS-733-1

SEPTEMBER 1999

TELECOMMUNICATIONS INDUSTRY ASSOCIATION



Representing the telecommunications industry in
association with the Electronic Industries Alliance



NOTICE

TIA/EIA Engineering Standards and Publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining with minimum delay the proper product for his particular need. Existence of such Standards and Publications shall not in any respect preclude any member or nonmember of TIA/EIA from manufacturing or selling products not conforming to such Standards and Publications, nor shall the existence of such Standards and Publications preclude their voluntary use by those other than TIA/EIA members, whether the standard is to be used either domestically or internationally.

Standards and Publications are adopted by TIA/EIA in accordance with the American National Standards Institute (ANSI) patent policy. By such action, TIA/EIA does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the Standard or Publication.

TIA/EIA INTERIM STANDARDS

TIA/EIA Interim Standards contain information deemed to be of technical value to the industry, and are published at the request of the originating Committee without necessarily following the rigorous public review and resolution of comments which is a procedural part of the development of a TIA/EIA Standard.

TIA/EIA Interim Standards should be reviewed on an annual basis by the formulating Committee and a decision made on whether to proceed to develop a TIA/EIA Standard on this subject. TIA/EIA Interim Standards must be cancelled by the Committee and removed from the TIA/EIA Standards Catalog before the end of their third year of existence.

Publication of this TIA/EIA Interim Standard for trial use and comment has been approved by the Telecommunications Industry Association. Distribution of this TIA/EIA Interim Standard for comment shall not continue beyond 36 months from the date of publication. It is expected that following this 36 month period, this TIA/EIA Interim Standard, revised as necessary, will be submitted to the American National Standards Institute for approval as an American National Standard. Suggestions for revision should be directed to: Standards & Technology Department, Telecommunications Industry Association, 2500 Wilson Boulevard, Arlington, VA 22201.

(From Project No. 4538, formulated under the cognizance of the TIA TR-45.5 Subcommittee on Spread Spectrum Digital Technology.)

Published by

©TELECOMMUNICATIONS INDUSTRY ASSOCIATION 1999
Standards & Technology Department
2500 Wilson Boulevard
Arlington, VA 22201

**PRICE: Please refer to current Catalog of
EIA ELECTRONIC INDUSTRIES ALLIANCE STANDARDS and ENGINEERING
PUBLICATIONS or call Global Engineering Documents, USA and Canada
(1-800-854-7179) International (303-397-7956)**

All rights reserved
Printed in U.S.A.

PLEASE!

**DON'T VIOLATE
THE
LAW!**

This document is copyrighted by the TIA and may not be reproduced without permission.

Organizations may obtain permission to reproduce a limited number of copies through entering into a license agreement. For information, contact:

Global Engineering Documents
15 Inverness Way East
Englewood, CO 80112-5704 or call
U.S.A. and Canada 1-800-854-7179, International (303) 397-7956

1 **Appendix B: TTY/TDD Extension**

2 **B.1. INTRODUCTION:**

3 This annex provides an option for modifying the current IS-733 standard to reliably
4 transport the TTY/TDD 45.45 bps Baudot code, making digital wireless technology
5 accessible to TTY/TDD users. The annex is separated into two major components.
6 Section B.3 describes the aspects of this annex that are required in order to be
7 compliant with this extension. Specifically, it describes the new interface between the
8 encoder and the decoder for transporting the TTY information. Section B.4 is a
9 description of the TTY/TDD software simulation of this annex, and is offered only as a
10 recommendation for implementation. However, in the event of ambiguous or
11 contradictory information, the software simulation shall be used to resolve any
12 conflicts.

13 This annex uses the following verbal forms: "Shall" and "shall not" identify
14 requirements to be followed strictly to conform to the standard and from which no
15 deviation is permitted. "Should" and "should not" indicate that one of several
16 possibilities is recommended as particularly suitable, without mentioning or excluding
17 others; that a certain course of action is preferred but not necessarily required; or that
18 (in the negative form) a certain possibility or course of action is discouraged but not
19 prohibited. "May" and "need not" indicate a course of action permissible within the
20 limits of the standard. "Can" and "cannot" are used for statements of possibility and
21 capability, whether material, physical, or causal.

22 **B.2. OVERVIEW:**

23 The following provides a method for reliably transporting the 45.45 bps Baudot code in
24 the audio path, making digital wireless telephony accessible to TTY/TDD users. The
25 following extension is robust to frame and bit errors and is completely interoperable
26 with the pre-existing IS-733 standard. The extension is also passive, requiring no
27 external interaction on the part of the user, nor on any other part of the network. The
28 solution supports voice carryover/hearing carryover (VCO/HCO). VCO allows a
29 TTY/TDD user to switch between receiving TTY and talking into the phone. Similarly,
30 HCO allows a user to switch between transmitting TTY characters and picking up the
31 phone to listen. When Baudot tones are not present, the vocoder operates as usual;
32 there is no modification or added delay to the voice path when speech is present.

33 The TTY/TDD audio solution transports Baudot signals through the vocoder by
34 detecting the characters that are being transmitted by the TTY/TDD in the encoder
35 and conveying those characters to the decoder. Because one Baudot character spans
36 at least 8 speech processing frames, the character being transmitted shall be sent a
37 minimum of 7 times to the decoder, allowing the decoder to correctly regenerate the
38 character despite frame errors and random bit errors in the speech packet.

1 The TTY characters are concealed in the speech packet in a way that interoperates
2 with legacy vocoders that have not been modified. This is made possible because,
3 when Baudot tones are present, the TTY information replaces the pitch lag bits for the
4 adaptive codebook (ACB) and the ACB gain is set to zero so that an unmodified decoder
5 will ignore the TTY information. The rest of the bits in the speech packet contain
6 information for an unmodified decoder to reconstruct the Baudot signal with the fixed
7 codebook and the linear prediction (LPC) filter at least as well as if the encoder was not
8 modified. Furthermore, if there is any noise suppression, the encoder shall disable
9 noise suppression, and the rate shall be set to full rate when the Baudot tones are
10 present. This further enhances the system's performance when a modified encoder is
11 interoperating with an unmodified decoder.

12 A decoder modified with this extension maintains a history buffer to monitor the ACB
13 gain and pitch lag in the speech packets. When the decoder detects that the ACB gain
14 has been set to zero, and the pitch lag contains information consistent with TTY, the
15 decoder stops decoding speech and begins regenerating the Baudot tones. When the
16 decoder stops detecting TTY information, it resumes processing speech.

17 When Baudot tones are not present, the modified vocoder operates on speech in exactly
18 the same way as the unmodified vocoder. The TTY processing does not add any
19 additional delay to the speech path.

20 **B.3. TTY/TDD EXTENSION:**

21 The TTY processing in the encoder shall process the RX PCM one frame at a time and
22 label each frame as NON_TTY, or as TTY_SILENCE, or as a TTY character. The vocoder
23 will be in one of two states: TTY_MODE or NON_TTY_MODE. In the absence of Baudot
24 tones, the encoder and decoder shall be in the NON_TTY_MODE, and the encoder and
25 decoder shall process the frame as speech. When Baudot tones are present, the
26 encoder and decoder shall enter TTY_MODE and process the TTY information as
27 described below.

28 There shall exist a mechanism to disable the TTY/TDD extension in the vocoder,
29 reverting the vocoder to its unmodified state.

30 **B.3.1 TTY Onset Procedure**

31 The TTY Onset Procedure describes the process by which the vocoder shall transition
32 from the speech mode to the TTY mode.

1 B.3.1.1 Encoder TTY Onset Procedure

2 When the TTY encoder processing initially detects that Baudot tones are present, the
3 encoder shall label each frame as TTY_SILENCE until it buffers enough frames to
4 detect the character being sent. The TTY_SILENCE message shall be sent to the
5 decoder according to the method described below. Because of the delay caused by the
6 buffering in the encoder and decoder to detect TTY characters, it is necessary to alert
7 the decoder to mute its output when Baudot tones are first detected. This prevents the
8 Baudot tones from getting through the speech path before the TTY decoder processing
9 is able to detect the TTY characters and regenerate the tones. The TTY_SILENCE
10 message shall be sent to the decoder within 2 frames after the PCM containing the
11 Baudot tones initially enters the encoder, and the message shall continue to be sent
12 until a TTY character is detected, or until a NON_TTY frame is detected.

13 B.3.1.2 Decoder TTY Onset Procedure

14 When the decoder is in NON_TTY_MODE, the packet shall be decoded in the usual
15 manner for speech. Because there are no bits in the packet to switch the decoder's
16 state, the decoder shall infer the presence of TTY information from the ACB gain and
17 pitch information. The decoder shall recognize when TTY_SILENCE messages are
18 being sent in the packets and transition from NON_TTY_MODE to TTY_MODE before
19 the decoder's speech path reconstructs a TTY character from the audio information in
20 the speech packets. When the decoder makes the transition to TTY_MODE, it shall
21 mute its output until it detects TTY characters or until it transitions back to
22 NON_TTY_MODE. Refer to the implementation recommendation in Section B.4 for an
23 example of the TTY decoder processing.

24 **B.3.2 TTY_MODE PROCESSING:**

25 The format of the 45.45 bps Baudot code can be found in ITU-T Recommendation V.18.
26 The Baudot code is a carrierless, binary FSK signaling scheme. A 1400 Hz ($\pm 5\%$) tone
27 is used to signal a logical "1" and an 1800 Hz ($\pm 5\%$) tone is used to signal a logical "0".
28 A TTY bit has a duration of 22 ± 0.4 ms and a character consists of 1 start bit, 5 data
29 bits, and 1.5 – 2 stop bits. When a character is not being transmitted, silence, or a
30 noisy equivalent, is transmitted. Hence, a TTY character spans a minimum of 8
31 speech processing frames. When the TTY encoder processing detects a character, it
32 shall send the character and its header (see Section B.3.3 for a description of the
33 header) to the decoder over a minimum of 7 consecutive frames and a maximum of 16
34 frames. Because channel impairments cause frame errors and bit errors, the decoder
35 may not receive all of the packets sent by the encoder. The decoder shall use the
36 redundancy to correct any corrupted TTY information. Once the decoder recognizes the
37 TTY character being sent, the decoder's TTY repeater shall regenerate the Baudot
38 tones corresponding to that character. It is recommended that the repeater generate
39 the Baudot tones for the shortest possible character duration described in V.18, e.g., a
40 bit duration of 21.6 ms and a stop bit length of 1.5 bits.

1 B.3.2.1 TTY_SILENCE Processing

2 In order to reduce the average data rate of a TTY call, the TTY processing shall be
 3 capable of transmitting 1/8 rate packets to the decoder when the encoder is processing
 4 the silence periods between characters. Since no TTY information is in the 1/8
 5 packet, the decoder shall infer TTY_SILENCE from an 1/8 rate packet when it is in
 6 TTY_MODE. The TTY_SILENCE message may also be sent to the decoder using a full
 7 rate frame, as described in Section B.3.3. When setting the rate to accommodate the
 8 TTY information, care must be taken so that a full rate frame is not immediately
 9 followed by an 1/8 rate frame. This is an illegal rate transition according to IS-733,
 10 and will force an unmodified decoder to declare a frame erasure.

11 **B.3.3 TTY Header and Character Format**

12 The TTY information put into the speech packet contains header and character
 13 information. When the encoder is transmitting a TTY character, the header shall
 14 contain a sequence number to distinguish that character from its preceding and
 15 following neighbors. The same header and character information shall be transmitted
 16 for each instance of a character for a minimum of 7 frames and a maximum of 16
 17 frames. The header shall cycle through its range of valid values, one value for each
 18 instance of a character. The header and character field shall be assigned a value to
 19 correspond to the TTY_SILENCE message. TTY_SILENCE may also be conveyed by 1/8
 20 rate packets (see Section B.3.2.1). The valid values for the TTY header and character
 21 fields are specified in Table B-4.

Description	Range	
	Header (2 bits)	Character (5 bits)
TTY_SILENCE	0	4
Reserved	0	0 - 3, 5 - 31
TTY Character	1 - 3	0 - 31

22 **Table B-1: TTY Header and Character Fields**

23 **B.3.4 Transporting the TTY Information in the Speech Packet.**

24 In full rate and half rate, there are 8 bits per subframe assigned to the pitch lag. Both
 25 half rate and full rate packets are capable of transporting TTY information. In order to
 26 improve interoperability between a modified encoder and an unmodified decoder, it is
 27 recommended to transport the TTY information in a full rate packet; however, a
 28 modified decoder shall be capable of detecting TTY information in both full rate and half
 29 rate packets.

1 The TTY information replaces the pitch lag bits in the first subframe only. The pitch
 2 lag is set to zero in the remaining subframes. The ACB gain is set to zero for each
 3 subframe. Seven bits are used to convey the TTY information. The five least
 4 significant bits of the pitch bits are used for the 5-bit Baudot code. Two additional bits
 5 are used for header information. The TTY information is assigned to the pitch lag bits
 6 according to Table B-2.

PITCH LAG BIT ASSIGNMENT							
MSB							LSB
7	6	5	4	3	2	1	0
	TTY HEADER		5 BIT BAUDOT CODE				
	MSB	LSB	MSB				LSB
Reserved	1	0	4	3	2	1	0

8 **Table B-2: TTY Bit Assignment**

9 B.3.4.1 Half Rate TTY Mode

10 In the case where the encoder and decoder are both modified for TTY, it is possible to
 11 reduce the average data rate by using half rate packets to transport TTY information.
 12 Half rate packets should only be used to transport TTY information when it is
 13 determined that both the near-end and far-end vocoders are TTY capable. When the
 14 near-end (far-end) decoder recognizes that the far-end (near-end) encoder is sending
 15 TTY information, the near-end (far-end) encoder may be notified by the near-end
 16 (far-end) decoder to send TTY information in half rate packets. When the near-end (far-
 17 end) decoder receives a NON_TTY packet, the near-end (far-end) encoder should exit
 18 half rate TTY mode. This preserves interoperability in the event of a hard handoff from
 19 a modified vocoder to an unmodified vocoder.

20 **B.4. TTY/TDD PROCESSING RECOMMENDATION**

21 The following sections describes the software simulation of this annex. It is intended
 22 as a recommendation only and is not part of the standard. However, the software shall
 23 be used to resolve ambiguous or incomplete statements that may exist in the sections
 24 above.

25 The TTY/TDD processing is divided into 2 major components, encoder processing and
 26 decoder processing. The TTY encoder process detects the presence of Baudot tones and
 27 decodes the TTY character being transmitted. It then conveys that information to the
 28 decoder. The TTY decoder processing must detect the presence of TTY information and
 29 regenerate the Baudot tones corresponding to that character. Refer to Figure B-1 for a
 30 block diagram of the TTY processing.

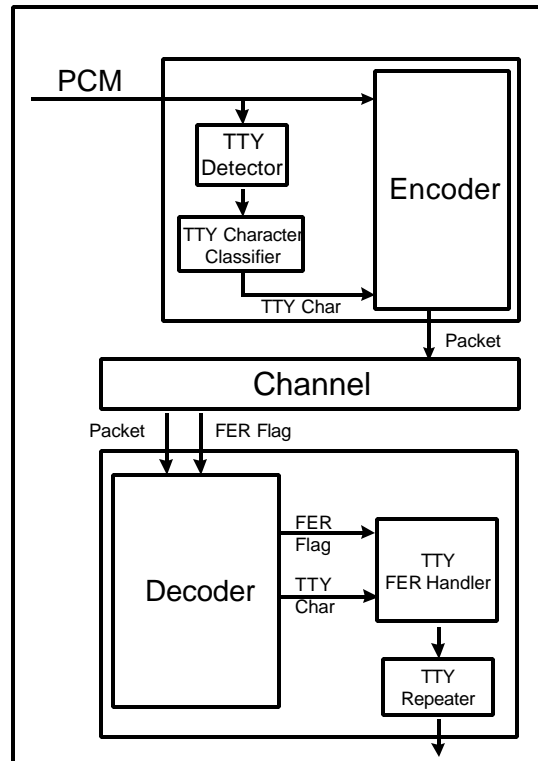


Figure B-1: TTY/TDD Processing Block Diagram

1 **B.4.1 TTY/TDD Encoder Processing**

2 The TTY/TDD encoder subroutines and their hierarchy are shown in Figure B-2. The
 3 initialization function `init_tty_enc()` initializes all of the state variables and static
 4 arrays for the TTY encoder processing.

5 The TTY encoder processing takes the larger task of detecting TTY characters and
 6 divides it into a series of smaller tasks, creating different levels of detection. It is
 7 through this divide and conquer approach that the `tty_enc()` routine has low complexity
 8 in the absence of Baudot tones.

9 The first level of detection is to divide the 160 samples in the speech frame into 10
 10 blocks of 16 samples. These blocks are called detection intervals, or dits. Each dit is
 11 classified as `NON_TTY`, as `LOGIC_0`, as `LOGIC_1`, or as `TTY_SILENCE`

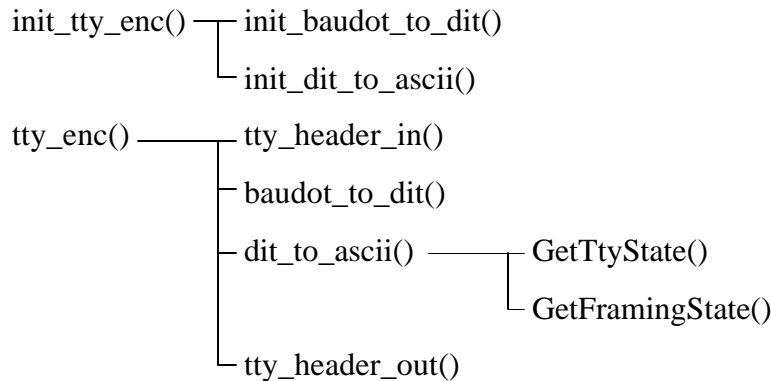


Figure B-2: TTY/TDD Encoder Processing Subroutines

1 The next level of detection is to determine if there are enough LOGIC_0 or LOGIC_1
 2 dits in a row to justify transitioning from NON_TTY_MODE to TTY_ONSET. The
 3 TTY_ONSET window is 1½ TTY bits long. This allows the encoder to start sending the
 4 TTY_SILENCE message to the decoder as soon as possible, even before a character is
 5 decoded (see Section B.3.1).

6 The next level of detection is to count the number of LOGIC_0 and LOGIC_1 dits that
 7 are inside a window the length of a TTY character. Once the count exceeds a
 8 threshold, the encoder will transition from TTY_ONSET to TTY_MODE and begin the
 9 search for a character.

10 B.4.1.1 tty_enc()

11 The routine `tty_enc()` is the top-level TTY encoder routine. It takes the RX PCM as its
 12 input and calls the lower level routines. If the frame is labeled as NON_TTY, it returns
 13 a zero. If the encoder gets into TTY_ONSET or TTY_MODE, it sets the flag to non-zero
 14 and outputs the appropriate TTY header and character information according to
 15 Section B.3.3.

16 B.4.1.2 tty_header_in()

17 The routine `tty_header_in()` converts the header information from the format that is
 18 sent to in the packets to the TTY processing's internal format. This internal format is
 19 chosen to simplify DSP programming. The TTY processing requires many compound `if()`
 20 statements, e.g. `if(header == TTY_SILENCE || header == NON_TTY)`. The internal
 21 format uses a different bit for each constant, so that the above `if()` statement can be
 22 efficiently implemented with a single conditional. If `TTY_SILENCE = 0x01` and
 23 `NON_TTY = 0x20`, then the above statement can be implemented by
 24 `if((header & TTY_SILENCE & NON_TTY) != 0)`.

1 B.4.1.3 baudot_to_dit()

2 The routine baudot_to_dit() converts the 160 samples from the current frame into 10
 3 dits. For every block of 16 samples, it computes the spectral energy at the mark and
 4 space frequencies using a 16-point DFT at 1400 Hz. and 1800 Hz. with a rectangular
 5 window. The total energy in the 16 sample dit is also computed. If the total energy is
 6 below a threshold, the dit is labeled as TTY_SILENCE. If not, the ratio of the maximum
 7 energy between the mark and space energy and the total energy is compared to a
 8 threshold, as follows:

$$\max(\text{mark_energy}, \text{space_energy})/\text{total_energy} > \text{THRESH.}$$

10 If that threshold is exceeded, the dit is labeled as either a mark or a space, whichever
 11 has the most energy. If none of the thresholds are met, the dit is labeled as NON_TTY.

12 B.4.1.4 dit_to_ascii()

13 The routine dit_to_ascii() is responsible for transitioning the TTY encoder processing
 14 from one state to the other. When the vocoder is in NON_TTY_MODE, dit_to_ascii()
 15 calls GetTtyState() to transition into TTY_ONSET and then into TTY_MODE. If
 16 GetTtyState() returns NON_TTY_MODE, dit_to_ascii() labels the frame NON_TTY and
 17 returns. If GetTtyState() returns TTY_ONSET, dit_to_ascii() labels the frame
 18 TTY_ONSET so that the TTY_SILENCE message can be sent to the decoder. If
 19 GetTtyState() returns TTY_MODE, dit_to_ascii() enters the NOT_FRAMED state,
 20 meaning that it is in TTY_MODE but it has not decoded a character. While in the
 21 NOT_FRAMED state, dit_to_ascii() calls GetFramingState(). Its input is a dit buffer the
 22 length of a TTY character plus 2 bits of lookback (see Table B-3). GetFramingState()
 23 checks to see if the dit buffer centered over a TTY character. If a character is not
 24 found, dit_to_ascii() shifts this buffer by one dit and the search for a character is
 25 repeated. This process continues until a character is framed or until GetTtyState()
 26 changes the state to NON_TTY_MODE.

27 Once a character is framed, the character and its header are sent to the decoder over a
 28 minimum of 7 frames and a maximum of 16 frames. The constant
 29 FRAMING_HANGOVER dictates the maximum number of times the information for the
 30 same character is sent. If a new character is framed before FRAMING_HANGOVER is
 31 reached, the information for the old character is terminated and the new information
 32 is sent to the decoder. However, after a character is framed, the search for the next
 33 character is not begun until most of the dit detections from this character are flushed
 34 from the dit buffer.

35

Dit #:	0-10	11-21	22-32	33-43	44-54	55-65	66-76	77-87	88-98	99-109
	Memory Bits		Start Bit	Data Bits					Stop Bit	

36 **Table B-3: Dit Buffer**

1 B.4.1.5 GetTtyState()

2 GetTtyState() is responsible for changing the state of TTY encoder processing. There
3 are 3 states: NON_TTY_MODE, TTY_ONSET, and TTY_MODE.

4 TTY_ONSET state is determined by looking at 16 of the most recent dit detections.
5 This corresponds to 1½ TTY bits. The largest number of consecutive LOGIC_0 dits and
6 the largest number of consecutive LOGIC_1 dits are counted and compared to a
7 threshold. If either one exceeds the threshold, TTY_ONSET is declared. The
8 TTY_ONSET test is only performed in NON_TTY_MODE or TTY_ONSET state.

9 Once TTY_ONSET is declared, GetTtyState then tests for TTY_MODE by looking at a dit
10 buffer the length of a TTY character plus 2 bits (see Table B-3). If the TTY processing is
11 not in TTY_MODE, TTY_MODE is declared when the total number of LOGIC_0 and
12 LOGIC_1 dits exceeds a threshold. If the TTY processing is already in TTY_MODE, then
13 the number of LOGIC_0, LOGIC_1, and TTY_SILENCE dits are counted and compared to
14 the same threshold. If the threshold is exceeded, then the TTY processing stays in
15 TTY_MODE; otherwise the state is changed to NON_TTY_MODE.

16 B.4.1.6 GetFramingState()

17 GetFramingState() is the routine that decodes the TTY character. The dit buffer
18 (described in Table B-3) is tested to see if the dits are consistent with a TTY character.
19 In order for the dit buffer to be centered over a character, the following rules are
20 applied:

- 21 • **Memory Bits:** Because a character ends with a stop and begins with a start bit, it
22 is illegal for a start bit to be preceded by a start bit. Therefore, the dits prior to the
23 start bit are checked for LOGIC_0s. If their number exceeds a threshold, the
24 framing test fails.
- 25 • **Start Bits:** The number of LOGIC_0s are counted in the dits corresponding to the
26 start bit. If there are not a sufficient number of them, the test fails.
- 27 • **Data Bits:** Although it is not known whether each bit will be a “0” or a “1”, each bit
28 should have mostly one value or the other. Each of the data bits is checked for
29 consistent dit detections of either LOGIC_0 or LOGIC_1. The test fails when any
30 one of the bits does not have consistent dit detections. If the test passes, the
31 character is decoded based on the information from this test. The character is not
32 considered framed, however, until the remaining tests are passed.
- 33 • **Stop Bits:** The number of LOGIC_1s are counted in the dits corresponding to the
34 stop bit. If there are not a sufficient number of them, the test fails.
- 35 • **Silence Test:** Because there should not be any silence in the middle of a TTY
36 character transmission, the number of SILENCE dits is counted. If they exceed a
37 threshold, the framing test fails.

38 If all of the above tests pass, then a flag is set and the decoded character is returned.

B.4.2 TTY/TDD Decoder Processing

The TTY/TDD decoder subroutines and their hierarchy are shown in Figure B-3. The initialization function `init_tty_dec()` initializes all of the state variables and static arrays for the TTY decoder processing.

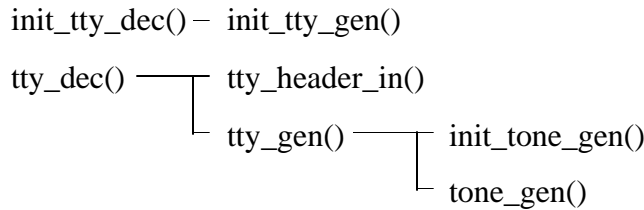


Figure B-3: TTY/TDD Decoder Subroutines

The TTY decoder processing shall recognize when TTY/TDD information is in the packet, shall recover from channel impairments to decode the TTY character being sent, and shall regenerate the Baudot tones corresponding to that character.

B.4.2.1 tty_dec()

The task of detecting the presence of TTY information, recovering from FERs, and decoding the TTY character is performed in `tty_dec()`. From the top level, the input to this routine is the pitch, the ACB gain information and the frame erasure flag. The outputs are a flag and a PCM buffer. If the routine detects that TTY information is being sent, the `tty_dec()` flag shall be set to non-zero and the PCM buffer shall be filled with the appropriate Baudot tones. If TTY is not detected, the flag shall be set to zero and the PCM buffer shall be returned unmodified.

The routine labels each frame as `NON_TTY`, as `TTY_SILENCE`, as `FER`, `EIGHTH_RATE`, or as a TTY character and maintains a history buffer of these classifications for 11 frames: 9 frames of lookahead, 1 current frame, and 1 frame of lookback (see Table B-4). The most recent packet enters the buffer at location 0, but the decision for the current frame is based on the contents of location 9. The buffer is updated at the end of each frame, shifting its contents to the right. The buffer is initialized to `NON_TTY`.

Frame:	0	1	2	3	4	5	6	7	8	9	10
Description:	Lookahead									Current Frame	Look-back

Table B-4: tty_dec() History Buffer

1 At the start of each frame, the most recent information is checked to see if it is
 2 consistent with TTY information. If the ACB gain is non-zero for any of the subframes,
 3 the frame is immediately labeled as NON_TTY. Likewise, if the frame erasure flag is
 4 set, the frame is labeled FER; otherwise the pitch information is scrutinized, checking
 5 to see if the pitch lag, when interpreted as TTY information, falls within the allowed
 6 range of values for the header and TTY character fields (see Table B-2). If all of the
 7 tests pass, Frame 0 is labeled with the TTY character in the history buffer.

8 The TTY decoder processing can reliably regenerate the TTY characters despite
 9 channel impairments because the character information is transmitted a minimum of
 10 7 times from the encoder. Errors are corrected by a voting process. The TTY
 11 information in a 9 frame window, starting with the current frame, is compared to see
 12 which header and character appears most often. Errors are replaced with the winner of
 13 the vote. The voting is conducted under the following conditions:

- 14 • Any time the current frame is labeled as a FER.
- 15 • Every time the current frame contains information for the start of a new character.
 16 Because a new character must contain a minimum of 7 frames of the same
 17 information, a vote is taken to verify that the information is present before it will
 18 generate the tones for that character. Once a character wins the vote, any FERs,
 19 BERs, or other inconsistencies are corrected in the 8 frame window where the
 20 character information is expected; i.e. the current frame and the adjacent 7 frames
 21 of lookahead will contain the same header and character information.
- 22 • Any time the current frame contains TTY_SILENCE or a TTY character, and the
 23 frame of lookback contains NON_TTY. This makes it harder for the decoder to
 24 erroneously go into TTY_MODE, thus preventing false alarms when speech is
 25 present.

26 Although the TTY processing does not introduce additional delay when speech is
 27 present, the TTY history buffers in the encoder and decoder causes a delay from the
 28 time the TTY tones first arrive in the encoder and when the tones get regenerated by
 29 the decoder. During this delay, which is roughly $2\frac{1}{2}$ TTY characters, the Baudot tones
 30 will get processed by the speech path, generating added characters every time there is
 31 a transition from NON_TTY_MODE to TTY_MODE. In order to prevent this, it is
 32 necessary to mute the decoder's output as soon as possible. This is accomplished by
 33 counting the number of silence frames inside the decoder's TTY history buffer when it
 34 is in NON_TTY_MODE. When this number exceeds a threshold, the current frame and
 35 the lookahead is converted to TTY_SILENCE, which will mute the decoder's output
 36 sooner than if it waited for the TTY_SILENCE frame to shift into the current frame.

37 At this point, the decision for the current frame is complete, and `tty_dec()` calls
 38 `tty_gen()` to generate the appropriate PCM samples.

39 B.4.2.2 `tty_gen()`

40 Once the current frame is labeled by `tty_dec()`, `tty_gen()` is called to generate the PCM
 41 buffer with the appropriate PCM samples. In the case of NON_TTY, the PCM buffer is
 42 returned unmodified. In the case of TTY_SILENCE, the PCM is muted.

1 Generating TTY characters is more involved, because one character spans many
2 frames, so `tty_gen()` must generate the Baudot tones one subframe at a time. When a
3 TTY character needs to be regenerated, `tty_gen()` puts a subframe's worth of samples in
4 the PCM buffer. It keeps track of which bit it is in the middle of generating and the
5 number of samples left to generate for that bit, so that the next time it is called, it can
6 pick up where it left off. Once `tty_gen()` begins to generate a character, it will generate
7 the entire character before it will generate the next character. This is done so that
8 the repeater will only generate valid TTY characters.

9 There exists a provision in V.18 for the TTY/TDD device to extend its stop bit in order
10 to prevent a TTY/TDD device from detecting its own echo. This routine will extend the
11 stop bit a maximum of 300 ms. if a TTY character is followed by silence. If a new
12 character arrives before 300 ms. has elapsed, the extended stop bit is terminated and
13 the new character is generated immediately.

14 The tones themselves are generated by `tone_gen()`. Before `tty_gen()` returns, it updates
15 the decoder's lookback field in the TTY history buffer with the information
16 corresponding to the last samples generated. For example, if `tty_gen()` finished
17 generating a character in the middle of the subframe and started generating silence,
18 the lookback field is updated with `TTY_SILENCE`. If `tty_gen()` is extending the stop bit of
19 a character, the lookback field is updated with the information of that character.

20 B.4.2.3 `tone_gen()`

21 The routine `tone_gen()` is a sine wave generator. Given a frequency and the number of
22 samples, it will generate the PCM samples by using a 2 tap marginally stable IIR filter.
23 The filter implements the trigonometric identity

$$24 \quad \cos(wk) = 2 \cdot \cos(w) \cdot \cos(w(k-1)) - \cos(w(k-2)).$$

25 It is a zero excitation filter, using only its past 2 samples and the cosine of the
26 frequency to be generated, to produce the next sample. This algorithm was chosen
27 because it is easily implemented using fixed point arithmetic.

