

3GPP2 C.S0020-0-2

Version 1.0

Date: 24 April 2001



**3RD GENERATION
PARTNERSHIP
PROJECT 2
"3GPP2"**

13K Vocoder TTY/TDD Extension

Appendix B: TTY/TDD Extension

1 INTRODUCTION

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

This annex supercedes the previous version of the TTY/TDD extension for 13K IS-733-1.

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

2 OVERVIEW

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

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

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

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

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

3 TTY/TDD EXTENSION

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

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

3.1 TTY Onset Procedure

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

3.1.1 Encoder TTY Onset Procedure

When the TTY encoder processing initially detects that Baudot tones are present, the encoder shall label each frame as TTY_SILENCE until it buffers enough frames to detect the character being sent. The TTY_SILENCE message shall be sent to the decoder according to the method described below. Because of the delay caused by the buffering in the encoder and decoder to detect TTY characters, it is necessary to alert the decoder to mute its output when Baudot tones are first detected. This prevents the Baudot tones from getting through the speech path before the TTY decoder processing is able to detect the TTY characters and regenerate the tones. The TTY_SILENCE message shall be sent to the decoder within 2 frames after the PCM containing the Baudot tones initially enters the encoder. However, in order to reduce the risk of false alarms, the TTY encoder may delay sending the TTY_SILENCE message for the very first character in a call. The TTY_SILENCE message shall be sent for a minimum of 4 frames and shall continue to be sent until a TTY character is detected, or until a NON_TTY frame is detected.

3.1.2 Decoder TTY Onset Procedure

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

3.2 TTY_MODE PROCESSING

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

3.2.1 TTY_SILENCE Processing

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

3.3 TTY Header and Character Format

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

Table B-1: TTY Header and Character Fields

3.4 Transporting the TTY Information in the Speech Packet.

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

The TTY information replaces the pitch lag bits in the first subframe only. The pitch lag is set to zero in the remaining subframes. The ACB gain is set to zero for each subframe. The 7 bits are used to convey the TTY information. The 5 least significant bits of the pitch bits are used for the 5-bit Baudot code. Two additional bits are used for header information. The TTY information is assigned to the pitch lag bits 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

Table B-2: TTY Bit Assignment

3.4.1 Half Rate TTY Mode

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

4 TTY/TDD PROCESSING RECOMMENDATION

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

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

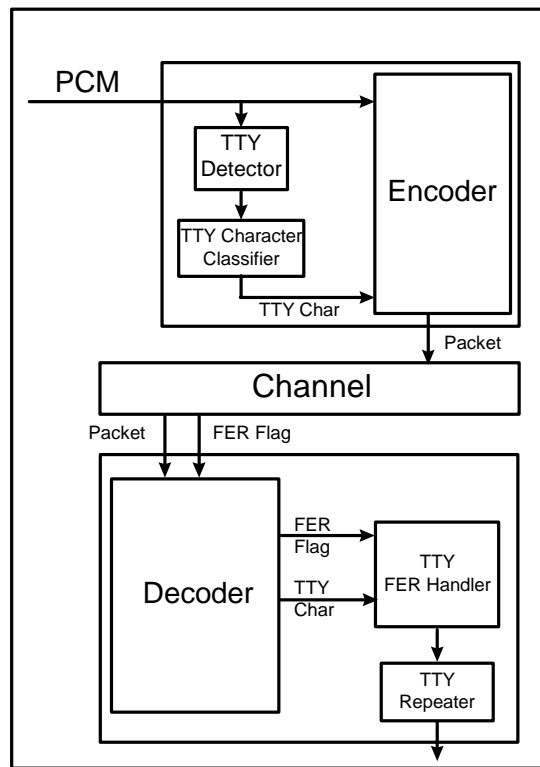


Figure B-1: TTY/TDD Processing Block Diagram

4.1 TTY/TDD Encoder Processing

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

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

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

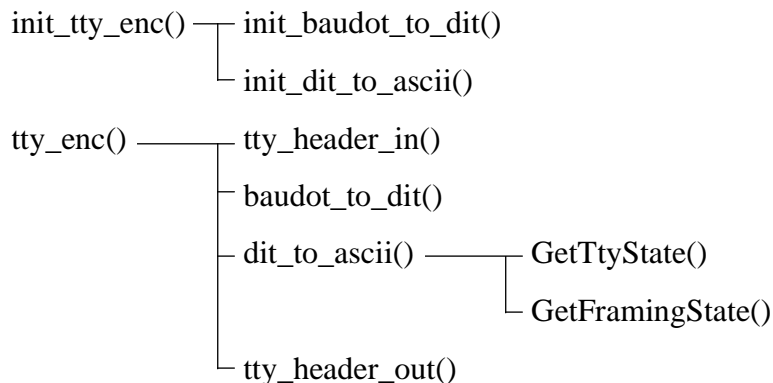


Figure B-2: TTY/TDD Encoder Processing Subroutines

The next level of detection is to determine if there are enough LOGIC_0 or LOGIC_1 dits in a row to justify transitioning from NON_TTY_MODE to TTY_ONSET. The TTY_ONSET window is $2\frac{1}{2}$ TTY bits long until the first character in a call is detected and $1\frac{1}{2}$ TTY bits long thereafter. This allows the encoder to start sending the TTY_SILENCE message to the decoder as soon as possible, even before a character is decoded (see Section 3.1).

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

4.1.1 tty_enc()

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

4.1.2 tty_header_in()

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

4.1.3 baudot_to_dit()

The routine `baudot_to_dit()` converts the 160 samples from the current frame into 10 dits. For every block of 16 samples, it computes the spectral energy at the mark and space frequencies using a 16-point DFT at 1400 Hz. and 1800 Hz. with a rectangular window. The total energy in the 16-sample dit is also computed. The ratio of the maximum energy between the mark and space energy and the total energy is compared to a threshold, as follows:

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

If that threshold is exceeded, the dit is labeled as either a mark or a space, whichever has the most energy. If not, then the dit is labeled as `TTY_SILENCE` if the total energy is below a silence threshold. If none of the thresholds are met, the dit is labeled as `NON_TTY`.

4.1.4 dit_to_ascii()

The routine `dit_to_ascii()` is responsible for transitioning the TTY encoder processing from one state to the other. When the vocoder is in `NON_TTY_MODE`, `dit_to_ascii()` calls `GetTtyState()` to transition into `TTY_ONSET` and then into `TTY_MODE`. If `GetTtyState()` returns `NON_TTY_MODE`, `dit_to_ascii()` labels the frame `NON_TTY` and returns. If `GetTtyState()` returns `TTY_ONSET`, `dit_to_ascii()` labels the frame `TTY_ONSET` so that the `TTY_SILENCE` message can be sent to the decoder. If `GetTtyState()` returns `TTY_MODE`, `dit_to_ascii()` enters the `NOT_FRAMED` state, meaning that it is in `TTY_MODE` but it has not decoded a character. While in the `NOT_FRAMED` state, `dit_to_ascii()` calls `GetFramingState()`. Its input is a dit buffer the length of a TTY character plus 2 bits of lookback (see Table B-3). `GetFramingState()` checks to see if the dit buffer is centered over a TTY character. If a character is not found, `dit_to_ascii()` shifts this buffer by one dit and the search for a character is repeated. This process continues until a character is framed or until `GetTtyState()` changes the state to `NON_TTY_MODE`. During the time when the encoder is in the `NOT_FRAMED` state, `dit_to_ascii()` labels the frame `TTY_ONSET` in order to send the `TTY_SILENCE` message to the decoder. A minimum of 4 frames is labeled as `TTY_ONSET` so that a minimum of 4 `TTY_SILENCE` messages is sent to the decoder before the character information is sent.

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

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	

Table B-3: Dit Buffer

4.1.5 GetTtyState()

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

TTY_ONSET state is determined by looking at a window of the most recent dit detections. The window is initially $2\frac{1}{2}$ TTY bits long so that the transition from a “0” to a “1” can be detected before declaring TTY_ONSET. This strict rule for declaring TTY_ONSET is enforced only for the first TTY character in order to avoid false alarms, but runs the risk of not sending the TTY_SILENCE message to the decoder quickly enough to mute the Baudot tones coming through the speech path. After the first TTY character is detected, the onset window is reduced to $1\frac{1}{2}$ TTY bits and the rule for declaring TTY_ONSET is relaxed so that only “0” or a “1” needs to be detected, not both. The largest number of consecutive LOGIC_0 dits and the largest number of consecutive LOGIC_1 dits are counted and compared to a threshold. If either one exceeds the threshold, TTY_ONSET is declared. The TTY_ONSET test is only performed in NON_TTY_MODE or TTY_ONSET state.

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

4.1.6 GetFramingState()

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

- **Memory Bits:** Because a character ends with a stop and begins with a start bit, it is illegal for a start bit to be preceded by a start bit. Therefore, the dits prior to the start bit are checked for LOGIC_0s. If their number exceeds a threshold, the framing test fails.
- **Start Bits:** The number of LOGIC_0s are counted in the dits corresponding to the start bit. If there are not a sufficient number of them, the test fails.

- **Data Bits:** Although it is not known whether each bit will be a “0” or a “1”, each bit should have mostly one value or the other. Each of the data bits is checked for consistent dit detections of either LOGIC_0 or LOGIC_1. The test fails when any one of the bits does not have consistent dit detections. If the test passes, the character is decoded based on the information from this test. The character is not considered framed, however, until the remaining tests are passed.
- **Stop Bits:** The number of LOGIC_1s is counted in the dits corresponding to the stop bit. If there are not a sufficient number of them, the test fails.
- **Silence Test:** Because there should not be any silence in the middle of a TTY character transmission, the number of SILENCE dits is counted. If they exceed a threshold, the framing test fails.

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

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.

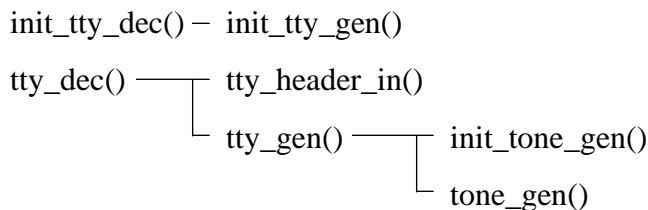


Figure B-3: TTY/TDD Decoder Subroutines

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.

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

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

The TTY decoder processing can reliably regenerate the TTY characters despite channel impairments because the character information is transmitted a minimum of 7 times from the encoder. Errors are corrected by a voting process. The TTY information in a 10-frame window, starting with the lookback frame, is compared to see which header and character appears most often. Errors are replaced with the winner of the vote. The voting is conducted every time the current frame is different from the lookback frame, that is to say, every time the decision for the current frame is different from the decision in the previous frame.

Although the TTY processing does not introduce additional delay when speech is present, the TTY history buffers in the encoder and decoder causes a delay from the time the TTY tones first arrive in the encoder and when the tones get regenerated by the decoder. During this delay, which is roughly 2½ TTY characters, the Baudot tones will be processed by the speech path, generating added characters every time there is a transition from NON_TTY_MODE to TTY_MODE. In order to prevent this, it is necessary to mute the decoder's output as soon as possible. This is accomplished by counting the number of silence frames inside the decoder's TTY history buffer when it is in NON_TTY_MODE. When this number exceeds a threshold, the current frame and the lookahead is converted to TTY_SILENCE, which will mute the decoder's output sooner than if it waited for the TTY_SILENCE frame to shift into the current frame.

Conversely, the decoder may start muting its output as the result of a false alarm. In order to minimize the time that a false alarm causes muting, if a NON_TTY message is received while the decoder is muting, the TTY_SILENCE messages are removed from tty_dec() history buffer and replaced with NON_TTY, forcing the decoder to start processing the packet through the speech packet.

After the voting, the decision for the current frame is complete, and `tty_dec()` calls `tty_gen()` to generate the appropriate PCM samples.

4.2.2 `tty_gen()`

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

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

In the event that the next character arrives before the regeneration of the current one is complete, the stop bit is shortened from 1½ stop bits to 1 stop bit in order to avoid falling behind.

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

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

4.2.3 `tone_gen()`

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

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

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