A Radioteletype Over-Sampling Software Decoder for Amateur Radio

Submitted To:
Proceedings of the ARRL and TAPR Digital Communications Conference
Submitted June 28, 2014

By:
Joseph J. Roby, Jr., KØJJR
2129 Bel Aire Avenue
Duluth, Minnesota 55803-1432
k0jjr@arrl.net

Abstract

A radioteletype (RTTY) over-sampling software decoder for amateur radio is described. Using computationally inexpensive software techniques, including over-sampling with the Goertzel algorithm, a reliable, self-synchronizing decoder can be created.

Key Words

Radioteletype, RTTY, Decoder, Baudot, Over-Sampling, Goertzel

Introduction

Amateur radio radioteletype (RTTY) is a digital radio mode for the exchange of text messages. Originally transacted with mechanical teleprinters, RTTY now is transacted with soundcard-equipped computers. The computer's software decodes the RTTY audio signal sent to the computer's soundcard by the radio receiver, so that the RTTY signal's text may be recovered.

This paper describes a software implementation of an over-sampling RTTY decoder. This decoder does not utilize the Fast Fourier Transform (FFT), any timing techniques that depend on clocks, any I/Q (in-phase, quadrature) techniques, or any multi-threading techniques. All tasks are performed with computationally inexpensive techniques, including the Goertzel algorithm, [1]-[4]. Text recovery is excellent, even for weak signals and in the presence of noise. The decoder is self-synchronizing with the received RTTY signal.

The decoder was programmed in Microsoft Visual Basic (primarily for the graphical user interface) and in Microsoft Visual C++ (primarily for the computations and soundcard functions). The Microsoft Windows WaveIn application programming interface (API) was used for communicating with the computer's soundcard. The decoder is incorporated into a complete RTTY contest and logging program written by the author.

I. RTTY Basics

RTTY is based on the five-bit Baudot code named for its inventor, Jean-Maurice-Emile Baudot, a French telegraph engineer, [5]. With five bits, up to 32 characters can be encoded, $2^5 = 32$.

But if two of the Baudot codes are dedicated to “shift” and “unshift” characters, up to 60 characters can be encoded, $(2^5 - 2) \times 2 = 60$. When the decoder encounters a shift character, all of the following Baudot codes will be treated as shifted Baudot codes, which correspond to the digits 0 through 9, plus a variety of other characters. When the decoder encounters an unshift character, all of the following Baudot codes will be treated as unshifted Baudot codes, which correspond to the letters A through Z (upper case only).

For example, Baudot shifted code 00011 is the hyphen character, and Baudot unshifted code 00011 is letter A. Null, carriage return, line feed, and the space characters are in both the shifted and unshifted sets of Baudot codes. Here is the table of the Baudot codes:

Binary	Hex	Unshifted Characters	Shifted Characters
00000	00	null	null
00001	01	E	3
00010	02	line feed	line feed
00011	03	A	-
00100	04	space	space
00101	05	S	bell
00110	06	I	8
00111	07	U	7
01000	08	carriage return	carriage return
01001	09	D	$
01010	0A	R	4
01011	0B	J	'
01100	0C	N	,
01101	0D	F	!
01110	0E	C	:
01111	0F	K	(
10000	10	T	5
10001	11	Z	"
10010	12	L	)
10011	13	W	2
10100	14	H	#
10101	15	Y	6
10110	16	P	0
10111	17	Q	1
11000	18	O	9
11001	19	B	?
11010	1A	G	&
11011	1B	shift	shift
11100	1C	M	.
11101	1D	X	/
11110	1E	V	;
11111	1F	unshift	unshift

When transmitted as an RTTY signal, each five-bit Baudot code is preceded by one start bit, which is always a zero, and is followed by two stop bits, which are always ones. Thus, a total of eight

bits comprise each RTTY Baudot code. The bits of an RTTY Baudot code are transmitted in this order: the start bit; then the right-most (least significant) Baudot bit; then, the second, third, fourth, and fifth Baudot bits moving from right to left within the five-bit Baudot code; then the two stop bits. For example, the eight bits of the RTTY Baudot code for the five-bit Baudot code 00011 are transmitted in this order: 0 1 1 0 0 0 1 1.

In RTTY parlance, an RTTY Baudot one bit is called a Mark and an RTTY Baudot zero bit is called a Space. Marks and Spaces are represented in an RTTY signal by audio tones of 2,125 Hz and 2,295 Hz, respectively. The 170 Hz separation between the Mark and Space audio tones is the standard used by amateur radio. Lower side band (LSB) is the preferred mode for RTTY communications. Note that in LSB mode, a Mark tone has higher a RF frequency than a Space tone, even though the Mark tone has a lower audio frequency than the Space tone. The Mark tone RF frequency is considered to be the operating frequency for logging purposes. For example, for an RTTY signal transmitted in LSB at 14.085 MHZ, the Mark tones will be at 14.082875 MHz and the Space tones will be at 14.082705 MHz. The operating frequency for logging purposes will be 14.082875 MHz.

Most amateur radio RTTY signals are transmitted at a baud rate (Marks and Spaces per second) of 45.45, which is meant to simulate approximately 60 words per minute, 45.45 ÷ 8 bits/char. ÷ 6 chars./word x 60 secs./min. = 56.81. Thus, each Mark and Space tone has a duration of 22 milliseconds, 1 ÷ 45.45 = 0.022) To a computer, 22 milliseconds is a long time – more than enough time to decode the RTTY signal through the use of over-sampling.

II. Soundcard and Radio Receiver Setup

The computer's soundcard takes in analog audio from the radio receiver and then digitizes the analog audio into a buffer of digital audio samples. The digital audio sample buffer is then processed by the decoder. While one digital audio sample buffer is being processed by the decoder, the soundcard synchronously fills another digital audio sample buffer to be ready for immediate processing when the decoder finishes decoding the prior digital audio sample buffer. This multi-buffering avoids clicks and gaps in the received audio.

The soundcard's digitizing parameters are programmable. For the decoder described in this paper, the soundcard is programmed at 44,100 digital audio samples per second. This is the rate used for compact disk recordings and provides more than sufficient fidelity for the decoder's purposes. The soundcard is also programmed for 8,192 digital audio samples per buffer. This is an arbitrary figure determined by trial and error. It has the benefits of reducing the frequency with which the soundcard must be accessed and of giving the decoder sufficient time to process a digital audio sample buffer as the next buffer is being synchronously filled by the soundcard. Finally, the soundcard is programmed for 16 bits per each digital audio sample. Sixteen bits, as opposed to eight, ensures that the full dynamic range of the analog audio is captured.

The radio receiver is tuned so that a reasonably narrow filter, say 500 Hz, is centered at 2,210 Hz below the receiver's frequency setting, which is halfway between the Mark and Space audio frequencies in LSB mode, 2,125 Hz + ((2,295 Hz - 2,125 Hz) ÷ 2) = 2,210 Hz. The radio receiver will then send analog audio to the computer's soundcard that includes any Mark and Space tones received relative to the radio receiver's frequency setting.

III. Band Pass Filtering

Ex. A shows the flow of data through the decoder. It starts when the computer's soundcard notifies the decoder that the soundcard has just finished filling a buffer with 8,192 digital audio samples taken from the analog audio passed to the soundcard by the radio receiver. The decoder needs two of those buffers, so that they can be processed in parallel, one for the detection of Mark tones and the other for the detection of Space tones. Therefore, the decoder makes a copy of the digital audio sample buffer's 8,192 samples.

One of those two identical buffers is passed through a digital band pass filter tuned to the Mark audio frequency, 2,125 Hz. The other is passed through a digital band pass filter tuned to the Space audio frequency, 2,295 Hz. This filtering suppresses unwanted audio tones in the respective buffers. Digital filtering source code is available for free on any number of web sites. The decoder described in this paper uses the free band pass filter source code made available on-line by Exstrom Laboratories, LLC, [6].

IV. Over-Sampling

Over-sampling takes multiple readings of a signal during the duration of the signal. From the multiple readings a decision can be made about the characteristics of the signal. For this decoder, the goal was to take multiple readings of the digital audio samples during the duration of a Mark or Space tone, 22 milliseconds. From those multiple readings, a decision would be made whether those digital audio samples include, among all the represented audio tones, a Mark or Space tone.

An over-sampling buffer of 128 digital audio samples works well for this purpose. That number of samples has a duration of 2.9 milliseconds, $128 \div 44{,}100 = 0.0029$. The duration of a Mark or Space tone, 22 milliseconds, can accommodate 7.6 over-sample readings, $22 \div 2.9 = 7.5862$.

Each of the Mark- and Space-filtered buffers of digital audio samples, 8,192 samples each, can be subdivided into 64 over-sampling buffers, each having 128 digital audio samples, $8{,}192 \div 128 = 64$. Each of the 64 over-sampling buffers is analyzed using the Goertzel algorithm.

V. The Goertzel Algorithm

The Goertzel algorithm performs single frequency audio tone detection within a buffer of digital audio samples. It does so much faster and using much less computational power than the FFT and other digital techniques, [1]-[4]. The Goertzel algorithm has been employed for the detection of dual-tone multi-frequency signaling (DTMF) tones on touch-tone telephones and for the detection of continuous tone coded squelch signals (CTCSS) in mobile radio transmissions.

The decoder uses an optimized form of the Goertzel algorithm described by Banks in [2]. The Goertzel algorithm is mathematically "tuned" to detect a specific audio tone. The algorithm then processes a buffer of digital audio samples and decides whether that specific audio tone is included among all the audio tones represented by the buffer's digital audio samples. The Goertzel algorithm returns a value representing the Goertzel relative magnitude of the specific audio tone. The greater the Goertzel relative magnitude, the greater the likelihood that the specific audio tone has been detected.

For purposes of decoding RTTY signals, the Goertzel algorithm is ideal because in each over-sampling buffer only one specific audio tone needs to be detected, either a Mark tone or a Space tone. To do so, the decoder uses two instances of the Goertzel algorithm, one tuned to detect the Mark audio tone, 2,125 Hz, and the other tuned to detect the Space audio tone, 2,295 Hz.

The decoder steps through the Mark- and Space-filtered digital audio sample buffers, 8,192 samples each, 128 samples at a time (the over-sampling buffer size). For each of the two corresponding over-sampling buffers, the decoder applies a "window function." A window function reduces the discontinuities and spectral "leakage" inherent in a finite set of digital audio samples, resulting in better frequency identification when the samples are examined to detect the audio tones they represent, [7]. This decoder uses a "Hamming" window, named after the mathematician who first proposed it, [7], [8].

After windowing, the decoder applies the Mark-tuned Goertzel algorithm to the Mark-filtered over-sampling buffer and the Space-tuned Goertzel algorithm to the corresponding Space-filtered over-sampling buffer. The resulting Mark-tuned Goertzel relative magnitude is compared to the resulting Space-tuned Goertzel relative magnitude. The result of the comparison is called a Goertzel bit. The Goertzel bit is given a value of one if the Mark-tuned Goertzel relative magnitude is greater than or equal to the Space-tuned Goertzel relative magnitude. The Goertzel bit is given a value of zero if the Space-tuned Goertzel relative magnitude is greater than the Mark-tuned Goertzel relative magnitude. A Goertzel one bit indicates detection of a Mark audio tone, and a Goertzel zero bit indicates detection of a Space audio tone.

VI. Timing

With an over-sampling rate of 7.6 Goertzel bits during the duration of a Mark or Space tone, accurate timing can be achieved by alternating between seven and eight Goertzel bits for each RTTY Baudot bit, so that the average is approximately 7.6:

One RTTY Baudot Bit 22 ms.	Number of Goertzel Bits 2.9 ms. ea.
start bit (Space)	8
1st - least significant - Baudot bit	7
2nd Baudot bit	8
3rd Baudot bit	7
4th Baudot bit	8
5th - most significant - Baudot bit	7
1st stop bit (Mark)	8
2nd stop bit (Mark)	7

The over-sampling technique may be summarized as follows, as shown on Ex. A:

- A Mark-filtered buffer of 8,192 16-bit digital audio samples and a Space-filtered buffer of equal size are examined . . .

- 128 digital audio samples at a time (the over-sampling buffer), and, . . .

- For each corresponding pair of Mark-filtered and Space-filtered over-sampling buffers, . . .
- A Hamming window is applied, and, . . .
- One Goertzel bit is recovered by applying the appropriately-tuned Goertzel algorithms and comparing the results, followed by, . . .
- The accumulation of seven or eight Goertzel bits, from which, . . .
- One RTTY Baudot bit is recovered, followed by, . . .
- The accumulation of eight RTTY Baudot bits, from which, . . .
- A five-bit Baudot code is recovered, which, . . .
- Is looked up in a table and the corresponding character is displayed on the computer screen.

VII. Synchronization

Please see the diagram marked as Ex. B. It shows how Goertzel bits are used to detect RTTY Baudot bits. The decoder counts the Goertzel bits as they are recovered. The Goertzel bit counter values are shown in the bottom two rows of the diagram. The decoder takes certain actions along the way, depending the value of the Goertzel bit counter, which is reset at the appropriate time. But first, the decoder must synchronize with the RTTY signal.

It would be an unpredictable coincidence for the decoder to have been started at precisely the moment when the RTTY signal was presenting an RTTY Baudot start bit, a Space. All that can be said with certainty when the decoder is started is that at some point within the next eight or fewer RTTY Baudot bits the RTTY signal will present an RTTY Baudot stop bit, a Mark, followed by an RTTY start bit, a Space, representing the transition from one RTTY Baudot code to the next. Although that pattern – Mark, Space – is necessary to indicate the ending of an RTTY Baudot code and the beginning of the next RTTY Baudot code, it is not sufficient. The pattern also appears within the five bits of many of the Baudot codes. Nonetheless, the pattern is a good starting point for the synchronization.

Therefore, the decoder begins synchronization by waiting for an RTTY Baudot stop bit, a Mark, followed by an RTTY Baudot start bit, a Space, both of which are determined by the Goertzel bits recovered from over-sampling.

When a Goertzel one bit is recovered, the decoder assumes (assumption 1) that it indicates an RTTY Baudot stop bit, a Mark. If assumption 1 is correct, a series of consecutive Goertzel one bits comprising portions or all of the two RTTY Baudot stop bits, both Marks, should be followed by a Goertzel zero bit indicating an RTTY Baudot start bit, a Space, for the next RTTY Baudot code. Therefore, the decoder, after having detected one or more Goertzel one bits, waits for a Goertzel zero bit.

When a Goertzel zero bit is recovered, the decoder assumes (assumption 2) that it is the beginning of an RTTY Baudot start bit, a Space. If assumption 2 is correct, it is then assumed (assumption 3) that the next seven Goertzel bits will also be Goertzel zero bits. To test assumption 3, the decoder looks at Goertzel bit counters 04 and 06. If either is a Goertzel one bit, then one or more of the three assumptions was incorrect, and the decoder begins anew with the synchronization. But if both Goertzel bit counters 04 and 06 are Goertzel zero bits, it is likely that the beginning of an RTTY Baudot code start bit, a Space, has been detected and the decoder proceeds accordingly.

This method of synchronization works well. Although it might seem to be a hit and miss proposition, it actually synchronizes with the received RTTY signal within a character or two. To assist with synchronization, it is good practice to preface a transmitted message with one, or a few, so-called "diddle" characters, which is the same thing as unshift character. Prefacing the message in this way helps the receiving station's decoder synchronize before any actual characters are encountered in the received RTTY signal.

After initial synchronization, the decoder keeps track the Goertzel bits as they are recovered. The decoder adds up the second, third, fourth, fifth, and sixth Goertzel bits of each RTTY Baudot bit. If those five Goertzel bits add up to zero, one, or two, the RTTY Baudot bit is declared to be a Space. If those five Goertzel bits add up to three, four, or five, the RTTY Baudot bit is declared to be a Mark. Examining multiple Goertzel bits in this way minimizes the adverse effects of drop out, fade, and other errors in the reception of the RTTY signal.

For example, Goertzel bit counters 10-16 are recovered during the duration of the first Baudot bit. Each of those Goertzel bits is either a zero or a one. Goertzel bits 11-15 are added, and the total will be zero to five. If the total is zero, one, or two, the first Baudot bit is declared to be a Space. If the total is three, four, or five, the first Baudot bit is declared to be a Mark. The same process is repeated for each of the following four Baudot bits. Although all seven or eight Goertzel bits recovered during the duration of an RTTY Baudot code could be added, it was thought that adding only five would be sufficient, and that has proved to be the case.

At Goertzel bit counters 47 and 48, the decoder will be into the first of the two RTTY Baudot stop bits (both Marks). At that point, the Goertzel bit counter is reset to one, which has the effect of restarting the synchronizing process. This resynchronization remedies any problems that might have been caused by variations in the received RTTY signal's Baudot bit duration, or if the received RTTY signal has only one and one-half RTTY Baudot stop bits (instead of two), which sometimes is the case. Also, if the decoder has been out of synchronization, this gives the decoder a new opportunity to become synchronized. In this fashion, the decoder synchronizes anew with every RTTY Baudot code and avoids the accumulation of any synchronization errors. No clock reference is needed.

There is one other synchronization consideration. Given that the duration of each RTTY Baudot bit is 22 milliseconds, and given that there are eight RTTY Baudot bits per RTTY Baudot code, each RTTY Baudot code has a duration of 176 milliseconds, 8 x 22 = 176. At 44,100 digital audio samples per second, and with a digital audio sample buffer of size 8,192, each digital audio sample buffer has a duration of 186 milliseconds, 8,192 ÷ 44,100 = 0.1858. Because the duration of an RTTY Baudot code, 176 milliseconds, is less than the duration of the digital audio sample buffer, 186 milliseconds, every digital audio sample buffer will always represent portions of at least two, if not three, RTTY Baudot codes. Any of the following scenarios is possible, and none of them can be predicted when the decoder synchronizes:

- An RTTY Baudot code might begin when the digital audio sample buffer begins. In this scenario, an entire RTTY Baudot code is at the front of the digital audio sample buffer, followed by the beginning portion of the next RTTY Baudot code.

- An entire RTTY Baudot code might fall somewhere in the middle of the digital audio sample buffer. In this scenario, that code will be preceded by the ending portion of the previous RTTY Baudot code, and will be followed by the beginning portion of the next RTTY Baudot code.

- An RTTY Baudot code might end when the digital audio sample buffer ends. In this scenario, that code will be preceded by the ending portion of the previous RTTY Baudot code.

- The digital audio sample buffer might contain, at its beginning, the ending portion of an RTTY Baudot code, and thereafter the beginning portion of the next RTTY Baudot code. In other words, the digital audio sample buffer spans portions of each of two RTTY Baudot codes.

The decoder must recover all RTTY Baudot codes, regardless of where in the digital audio sample buffer the RTTY Baudot code begins or ends, and even if the RTTY Baudot code spans the boundary between two digital audio sample buffers. The decoder accounts for these scenarios by keeping a running count of Goertzel bits and RTTY Baudot bits from one digital audio sample buffer to the next. The counts are reset at the appropriate times. The over-sampling technique used by the decoder minimizes if not eliminates loss of RTTY text from buffer to buffer.

Once the decoder recovers a five-bit Baudot code, it is a simple process to look up that Baudot code in a table to determine which character (digit, letter, space, or other) it represents, and, if displayable, to send the character to the computer video screen. The character also can be saved to a file to create a transcript of the RTTY session.

VIII. 75 Baud

As mentioned above, most amateur radio RTTY signals are transmitted at a baud rate of 45.45. There is, however, some activity at 75 baud, including contests sponsored by the British Amateur Radio Teledata Group and the Ukrainian Amateur Radio League. The decoder has been adapted for 75 baud simply by changing the timing relationship between the Goertzel bits and the RTTY Baudot bits. No changes needed to be made to the soundcard's digitizing parameters, the digital audio sample buffer size, the band pass filtering, the over-sampling buffer size, the Hamming Window, or the Mark- and Space-tuned Goertzel algorithms.

For 75 baud, each Mark and Space tone has a duration of 13.3 milliseconds, $1 \div 75 = 0.01333$. And, from above, the duration of an over-sampling buffer of 128 digital audio samples is 2.9 milliseconds. At 75 baud, the duration of a Mark or Space tone, 13.3 milliseconds, can accommodate 4.6 over-sample readings, $13.3 \div 2.9 = 4.5862$.

With an over-sampling rate of 4.6 Goertzel bits during the duration of a Mark or Space tone, accurate timing can be achieved at 75 baud by alternating between four and five Goertzel bits for each RTTY Baudot bit, so that the average is approximately 4.6:

One RTTY Baudot Bit	Number of Goertzel Bits
13.3 ms.	2.9 ms. ea.
start bit (Space)	5
1st - least significant - Baudot bit	4
2nd Baudot bit	5
3rd Baudot bit	4
4th Baudot bit	5
5th - most significant - Baudot bit	4
1st stop bit (Mark)	5
2nd stop bit (Mark)	4

Ex. B shows the resulting synchronization at 75 baud.

The counting of Goertzel bits is adjusted accordingly. For example, the decoder adds up the first, second, and third Goertzel bits of each RTTY Baudot bit. If those three Goertzel bits add up to zero or one, the Baudot bit is declared to be a Space. If those three Goertzel bits add up to two or three, the Baudot bit is declared to be a Mark.

Conclusion

This paper described an RTTY over-sampling software decoder for amateur radio. Using computationally inexpensive software techniques, including over-sampling with the Goertzel algorithm, a reliable, self-synchronizing decoder can be created.

References

[1] "Goertzel Algorithm," June 22, 2014. [Online]. Available: http://en.wikipedia.org/wiki/Goertzel_algorithm. [Accessed: June 23, 2014].

[2] K. Banks, "The Goertzel Algorithm," August 28, 2002. [Online]. Available: http://www.embedded.com/design/configurable-systems/4024443/The-Goertzel-Algorithm. [Accessed: June 23, 2014].

[3] G. Small, "Detecting CTCSS Tones With Goertzel's Algorithm," April 21, 2006. [Online]. Available: http://www.embedded.com/design/connectivity/4025660/Detecting-CTCSS-tones-with-Goertzel-s-algorithm. [Accessed: June 23, 2014].

[4] Microstar Laboratories, Inc., "Detecting a Single Frequency Efficiently," 2009. [Online]. Available: http://www.mstarlabs.com/dsp/goertzel/goertzel.html. [Accessed: June 23, 2014].

[5] "Baudot Code," June 10, 2014. [Online]. Available: http://en.wikipedia.org/wiki/Baudot_code. [Accessed: June 23, 2014].

[6] S. Hollos and J. Hollos, "Digital Signal Processing – Software," (2012). [Online]. Available: http://www.exstrom.com/journal/sigproc/. [Accessed: June 23, 2014].

[7] S. Smith, "Applications of the DFT," in *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1997. [Online]. Available: http://www.dspguide.com/ch9/1.htm. [Accessed: June 23, 2014].

[8] "Window Function," June 9, 2014. [Online]. Available: http://en.wikipedia.org/wiki/Hamming_window#Hamming_window. [Accessed: June 23, 2014].

Exhibit A - Flowchart

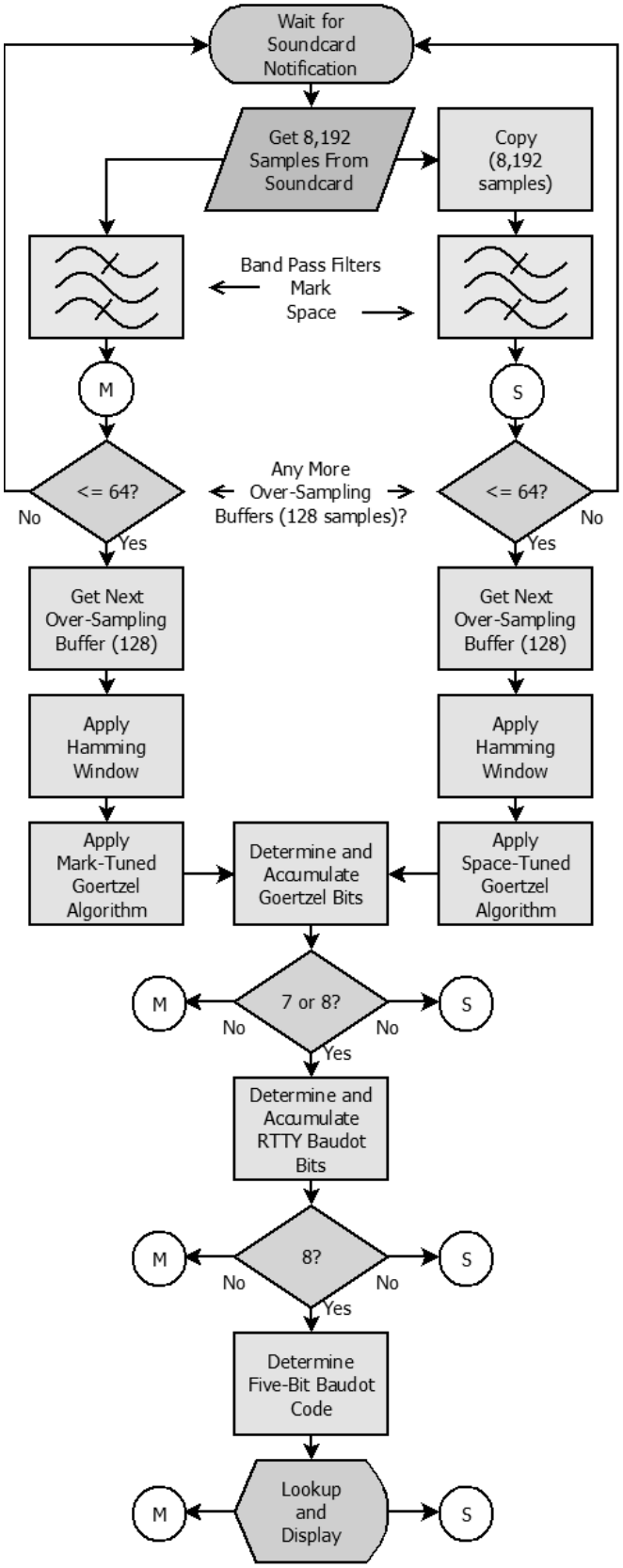

Exhibit B – Synchronization Between Goertzel Bits and RTTY Baudot Bits

45.45 Baud

	synchronize	RTTY Baudot start bit	1st Baudot bit	2nd Baudot bit	3rd Baudot bit	4th Baudot bit	5th Baudot bit	RTTY Baudot stop bit	RTTY Baudot stop bit	RTTY Baudot start bit	1st Baudot bit	
Goertzel bits:	0···000 1111···11	SSSSSSSS	???????	????????	???????	????????	???????	11111111	1111111	00000000	???????	etc.
Goertzel bit counter:	0···000 0000···00	00000000	1111111	11122222	2222233	33333333	4444444	44000000	0000000	00000000	1111111	
" " "	0···000 1111···11	23456789	0123456	78901234	5678901	23456789	0123456	78111111	1111111	23456789	0123456	

75 Baud

	synchronize	RTTY Baudot start bit	1st Baudot bit	2nd Baudot bit	3rd Baudot bit	4th Baudot bit	5th Baudot bit	RTTY Baudot stop bit	RTTY Baudot stop bit	RTTY Baudot start bit	1st Baudot bit	
Goertzel bits:	S···000 1111···11	00000	????	?????	????	?????	????	11111	1111	00000	????	etc.
Goertzel bit counter:	0···000 0000···00	00000	0001	11111	1111	22222	2222	23333	3333	33444	4444	
" " "	0···000 1111···11	23456	7890	12345	6789	01234	5678	90123	4567	89012	3456	