

Simple Conversational Amateur Messaging Protocol (SCAMP)

by Daniel Marks, KW4TI

Draft v0.4 2021-11-22

Abstract

Modern digital modes for amateur radio tend to require complicated transceiver architectures. For example, most need upper sideband modulation with a very stable local oscillator. Furthermore, methods such as multiple FSK can be susceptible to effects such as intermodulation from strong adjacent stations. Earlier modulations such as CW, which uses on-off keying (OOK), and RTTY45, which uses 2 frequency-shift-keying (FSK) were usable on transceivers with much less demanding requirements. Recently, there has been a trend towards sending data with a very low symbol rate (e.g. FT8 and QRSS) in order to achieve communication with very low transmitter power and/or with poor receiving conditions. This is a proposal for a simple digital mode, Simple Conversational Amateur Messaging Protocol (SCAMP), that can be implemented with OOK or 2FSK for conversational or amateur radio contest QSOs, can be implemented with relatively crude transceivers, and on simple 8-bit microprocessors such as the ATMEGA328P used for the Arduino Uno or Nano. It has both conversational and data transfer modes, but is tailored more to low bit rate text connections.

Introduction

There is now an abundance of amateur radio data modes. Early digital modes used FSK, for example, RTTY45, AMTOR, and its successors in packet radio. As personal computers became widely available with increasing processing power, as well as receivers with stable local oscillators, phase sensitive modes such as PSK31 and methods with small frequency shifts such as multiple frequency-shift keying (MFSK) (Olivia/Contestia) became possible, adding increased resistance to noise. Protocols specializing in very short messages, synchronized with a global time standard such as JT65/FT8/FT4 were designed for making QSOs with low power and in poor conditions. While there have been remarkable improvements in amateur radio data modes, with FT8 especially becoming quite popular, these require transceivers with very stable local oscillators that are resistant to intermodulation and other distortions. Furthermore, the computation required to decode some of these protocols requires a computer with powerful digital signal processing capability. There are instances when simple transmitters and receivers are desirable, that for example may only be capable of OOK (such as is used for CW transmission) or FSK (by selecting a PLL frequency or modulating a crystal oscillator). It would be further desirable if the processing ability of a microcontroller (for example, ATMEGA328P) was sufficient to encode and decode the data mode modulation. QRP transceivers, for example, often use simplified hardware, based on a PLL synthesizer or crystal oscillator with a direct conversion

receiver. Controlled by a microprocessor, such QRP radios can be solar powered, are highly portable, and can serve as message relays, for telemetry, or for emergency use.

A de-facto specification for an efficient, reliable, portable HF communication protocol called Automatic Link Establishment exists, as specified in MIL-STD-188/141A. This protocol uses MFSK, forward error correction (FEC), frame interleaving, and automatic repeat request (ARQ), among other methods. This protocol has been very successful for its intended use, however, it requires the kind of complex transceivers and modems that this method wishes to avoid. However, there are aspects of such methods that can be adapted. In particular, few amateur radio modes combine simple modulation and forward error correction. Simple forms of forward error correction can be implemented on 8-bit processors, and other techniques like interleaving may also be used. Most forms of FEC require significant processing power to decode and use either dedicated hardware or high speed processors to implement methods like the Viterbi or Berlekamp-Massey algorithm. The extended (24,12,8) Golay code has been used in ALE and is a happy medium which achieves a $\frac{1}{2}$ code rate and is able to correct 3 of 24 bits. Its primary disadvantage is that its short code length requires that the data is interleaved to be resistant to long error bursts. This is problematic for conversational modes for which long latency is an issue, especially for contesting. The Walsh/Hadamard codes used in Olivia/Contestia have this problem in particular because of their low code rate. The venerable extended (24,12,8) Golay code is a compromise solution that can be decoded by an 8-bit microcontroller, has reasonable latency, good code rate, and can correct up to 12.5% bit errors.

Modulation layer

The modulation layer is of one of two types:

On Off Keying (OOK)– The carrier alternates between full power transmission and no transmission as a non-return to zero (NRZ) line code. The mark condition (or one bit) is transmitting and the space condition (or zero bit) is no transmission. Care should be taken that the transmitter does not significantly chirp the carrier when initiating a transmission. Envelope modulation may be built into the keying circuitry to prevent keyclick. The interval of each bit (transmitting or not transmitting) is identical and is given by the reciprocal of the baud rate.

2 Frequency Shift Keying (FSK) – When transmitting, the carrier alternates between two frequencies as a return-to-zero (RZ) line code. The mark condition corresponds to transmitting at one frequency and the space condition is transmitting on the other frequency. The mark frequency may be higher or lower than the space frequency. The separation between the two is determined by the baud rate. Ideally the separation is a multiple of $\frac{1}{2}$ the baud rate, with a multiple of one corresponding to minimum shift keying (MSK). However, it is not expected that the transmitter can achieve a perfect separation frequency, nor can the receiver perfectly coherently decode a MSK signal. Therefore a separation equaling a multiple of the baud rate is used, so that for 100 mark or space intervals per second, these would be separated by 100 Hz. This nominally retains the orthogonality of the mark and space conditions but increases the tolerance to error in the separation frequency or local oscillator phase.

Digital Encoding

Messages are sent as 30 bit data frames, in order of most significant bit to least significant bit (left to right as shown here). The format of each frame is

C XXXX C XXXX C XXXX C XXXX C XXXX C XXXX
MSB LSB

The 24 “X” are 24 data bits to be sent in the frame which are a Golay code word. Each “C” is the complement of the bit immediately following it. This ensures there are no more than five consecutive mark or spaces (ones or zeros) in a valid codeword or consecutive codewords. The transition between the mark and space condition is used to aid in clock recovery and to allow an initial or resynchronization phase to be recognized.

Golay code word

Each Golay code word consists of two halves.

PPPP PPPP PPPP XXXX XXXX XXXX
MSB LSB

The Golay code word is a 24 bit code word that contains 12 bits of data payload to be sent, represented by “X” and 12 bits of parity, represented by “P”. The parity bits are calculated from the data bits using the (24,12,8) extended Golay encoding algorithm as given in the Appendix.

Golay code word types

The Golay code word types are the 12-bit payload to be sent in the Golay code word. There are two Golay code word types.

Data code word type

1111 XXXX XXXX
MSB LSB

This encodes 8-bit raw binary data XXXX XXXX in order of MSB to LSB. This message is intended to be used for exchanging data for file transfer protocols and other uses left up to the users. It is not an efficient encoding of this data but is included so that the connection may be used for purposes other than text exchange.

Text code word type

YYYYYY XXXXXX
MSB LSB

These encode two 6 bit symbols XXXXXX and YYYYYY. The symbol XXXXXX precedes that of YYYYYY in the data stream, that is, when considered as part of a message, the symbol corresponding to XXXXXX precedes YYYYYY. The symbols encode characters as specified in a table in the Appendix. The 6-bit code corresponding to 000000 indicates “No symbol” so that no character should be decoded in the message for this 6-bit code. If only one code is to be sent in the code word, then XXXXXX should be the code, and YYYYYY should be 000000. A code word with both XXXXXX and YYYYYY being 000000 is valid and should be considered as two no symbols. This symbol may be sent continuously during a connection one wishes to send a real-time typing message, or alternatively one can initiate a resynchronization.

For additional redundancy, the same text code word may be sent multiple times in a row. If the receiver decodes the same code multiple times before receiving a different code, it should discard the redundant decodes of the code word. If the same code word needs to be sent multiple times and not have its redundant copies discarded, at least one no symbol code (000000 000000) should be sent between the code word and its next copy so that the receiver decodes a different code. Redundant copies should be sent in immediate succession, that is, there should be no delay between sending the redundant copies of the code word. This enables the copies of the code word to be coherently summed by the receiver. Redundantly sent data code words (as opposed to text code words) should not be discarded.

Reserved code word type

XXXXYY 1111YY
MSB LSB

where XXXX is not 1111. These 240 codes are reserved for future use. Some possibilities are codes to indicate the beginning or end of transmission, request retransmission, and codes to indicate the symbol set (for example Latin, Cyrillic, Arabic, Hebrew, Kana, or Greek) at the beginning of the transmission.

Frame Synchronization

One of the aspects of a protocol that is most susceptible to corruption is desynchronization of the bit stream, that is, incorrectly starting a 30-bit frame at the wrong point in the bit stream. Therefore a synchronization protocol is necessary that can synchronize the beginning of a 30-bit frame. A fixed 30-bit synchronization frame is used that the receiver can recognize to indicate when a full frame has been received, with the next 30-bit data frame starting immediately afterwards. To not be mistaken as a data frame, the synchronization codeword must not be likely to be received as a valid data frame, even with added noise corruption. As the 30-bit data frame has complement bits inserted into it, the synchronization code word should not have complement bits at the same positions for as few shifts of the code word as possible so that the synchronization and data frames are unlikely to be confused with

added noise. Furthermore, the synchronization frame should have low autocorrelation sidelobes so that it is unlikely that a shift of the frame in the presence of noise triggers a synchronization.

The synchronization frame is given by the 30 bit number

11111 01101 00011 00111 01000 11110 or 0x3ED19D1E

which is sent MSB to LSB, or in order of left to right as written here. This codeword has a maximum of two complemented bits for any shift and an autocorrelation, given a bipolar encoding such that 1 is +1 and 0 is -1:

30 3 -2 -3 0 -1 -2 3 0 -3 -2 -1 4 1 2 -1 -4 1 2 3 -4 1 -2 -1 4 3 2 1 0 -1

so that the maximum off-peak autocorrelation without noise has a magnitude of 4 for a noise rejection capability of $20 \log_{10} (30/4) = 17.5$ dB. The receiver should continuously cross-correlate the incoming bit stream with this code word and reset the clock edge timer so that the first bit of the subsequent data code word is sampled at one bit interval after the peak of the cross-correlation occurs, or the synchronization frame is approximately matched by the last 30 bits. The codeword may be resent periodically to aid in resynchronization.

When first keying up a transmission, a receiver must be given time to start receiving. Furthermore, if the FSK mode is used, the receiver must determine which of the two FSK frequencies corresponds to the mark condition. To do this, a sequence of 20 or more mark intervals are sent at the beginning of the transmission, followed by 3 or 4 space intervals, then two mark intervals, and finally the synchronization code word. This pattern may be restarted at any time during the transmission to resynchronize the receiver with the transmitter. So in full, at the beginning of the transmission the following is sent:

11 1111 1111 1111 1111 1111 1100 0011
11 1110 1101 0001 1001 1101 0001 1110

The initial 24 one bit intervals provides sufficient time for the receiver to begin to receive, and if the FSK mode is used, the receiver uses the received frequency to denote the one bit.

When a data frame is received without error, six pairs of complement bits should be received corresponding to the bit pairs 1 and 2, 6 and 7, 11 and 12, 16 and 17, 21 and 22, and 26 and 27. If six pairs of complement bits are not received, there are several possibilities as to what caused the error. Firstly, one of the bits may have been inverted, so for example only five rather than six pairs of complement bits are observed. However, another possibility is that an extra bit transition was observed, or a bit transition was missed. With the erroneous bit transition, it is unlikely that the six complemented bits will be observed in their proper positions. If an extra bit transition occurs in error, then the thirty bits of a frame occurs one bit sooner than it should for the uncorrupted frame. In this case, if the number of complement bits are counted for the frame with the frame being advanced one bit, ideally five complement bits should be counted, as this indicates that waiting for another bit transition would add the sixth complement bit pair and therefore complete the frame, though with bit errors (which might be corrected using FEC). If a bit transition is missed, then by examining the frame delayed by one bit, one should ideally count six complement bits, and this indicates that one counted an extra bit during the frame, and that the frame is the thirty bits that are delayed by one bit from the current bit. Furthermore, the most recently received bit is the first bit of the next frame. Importantly,

by examining the number of complement bits in frames delayed or advanced by one bit, one can detect frame synchronization errors and possibly correct them by advancing or delaying the bit stream by one bit to compensate for the error.

Appendices

Six-bit symbol encoding

The following is a table of the six bit code. One or two of these codes form a 12-bit Golay code word which can send one or two character symbols. The six bit code is as follows:

	000	001	010	011	100	101	110	111
000xxx	No symbol	Backspace	End of Line	(space)	! (exclamation mark)	“ (double quote)	‘ (single quote)	(left parenthesis
001xxx) right parenthesis	* (asterisk)	+ (plus)	, (comma)	- (minus)	. (period)	/ (slash)	0
010xxx	1	2	3	4	5	6	7	8
011xxx	9	: (colon)	; (semi colon)	= (equal)	? (question mark)	@ (at)	A	B
100xxx	C	D	E	F	G	H	I	J
101xxx	K	L	M	N	O	P	Q	R
110xxx	S	T	U	V	W	X	Y	Z
111xxx	\ (backslash)	^ (carat)	` (grave)	~ (tilde)	INVALID	INVALID	INVALID	INVALID

The codes 111100, 111101, 111110, and 111111 are invalid and are never to be used.

For languages that have diacritics, the single quote, carat, grave, tilde, and backslash may be interpreted as diacritics, with the diacritic applying to the previously sent character. The backslash may be interpreted as an umlaut. It is highly recommended to send the character and the diacritic in the same Golay codeword so that these are decoded in the same word. A “No symbol” (000000) can be placed in the previous codeword to ensure that the next symbol is included with its diacritic.

For other characters representable by 8-bit bytes, for example UTF-8 encoding could be used. However, as UTF-8 has variable length symbols, missed bytes could result in sustained decoding failures, which is a serious drawback for amateur communications unless there is a redundancy check and an automatic retransmit request mechanism.

MSB 1111 XXXX XXXX LSB

where XXXX XXXX is the byte to be sent. For example, to send lower case a, the 12-bit word 111101100001 or 0xF61 would be sent. However, no differentiation is made between these 8-bit symbols sent as text, and those sent as data, for example, as part of a file transfer protocol.

Golay Matrix and Encoding/Decoding

The Golay matrix is for the extended (24,12,8) Golay code. The code is implemented using the perfect (23,11,7) Golay code with an extra parity bit included. This can be implemented by multiplying the 12-bit code word with the Golay matrix to obtain the 12-bit Golay parity check word. The Golay matrix is its own inverse, so that applying the matrix to a 12-bit word, and then again to the result, yields the original word. The implementation of the Golay coding is in the code Appendix.

The encoded word is simply the 12-bit word to be send with the 12-bit parity code prepended to it. The Golay code can correct up to 3 bit errors, and so to decode every possible error up to 3 bits, this code performs the following:

1. Check to see if all three error bits are in the sent parity bits by calculating the parity code of the sent 12-bit word and seeing if there are three or fewer difference bits between the sent parity and the calculated parity. If so, the sent 12-bit word is correct.
2. Check to see if all three errors are in the sent 12-bit word. Calculate the 12-bit word that would be obtained with the given parity code and see if there are three or fewer difference bits between the sent word and the calculated word. If so, the calculated word is correct.
3. Try flipping every bit in the sent 12-bit word and see if there are 2 or fewer errors between the calculated parity and the sent parity. If so, we know which bit of the 12-bit word is wrong and it is corrected.
4. Try flipping every bit in the sent parity code and see if there are 2 or fewer errors between the calculated 12-bit word and the sent word. If so, we know which bits are wrong in the sent word and it is corrected.
5. Otherwise, the error is uncorrectable or undetectable.

Implementation of the Finite Impulse Response (FIR) filters

The following describes a method of implementing the FIR filters that is suitable for small microcontrollers such as the ATMEGA328P. 16- and 32-bit multiplication are computationally intensive operations on an 8-bit microcontroller and are generally avoided. Because of this, these are avoided, and the operation of OOK and FSK modulations can be designed to mostly require additions, subtractions, comparisons, and bit shifts, all of which tend to be economical on small, energy efficient processors.

The following approach is used. An analog anti-aliasing filter can be applied before the signal from the mixer is sampled by an analog-to-digital converter (ADC). This removes harmonics that do not need to be filtered in software, making the application of a quadrature demodulator and low-pass filter relatively easy. In particular, a quadrature demodulator can be implemented by using adds and subtracts from a counter oscillator, with the adds and subtracts for the in phase quadrature phase subtracts being delayed a quarter cycle from the in-phase add or subtract. A low-pass filter can be implemented using a moving average FIR filter, with the length of the filter designed to place the frequency nulls of the filter in an adjacent frequency channel to be rejected.

The filters are driven by a master sample clock which is typically the sample rate of the ADC. This frequency is denoted here by f and the corresponding sampling period $T=1/f$. Each period T , the ADC is sampled, usually as the first operation in an interrupt service routine to apply the filter. A quadrature filter to be applied must have a period that is a multiple of $4T$ so that the number of samples per period is $4N$, where N is an integer. A counter tracks the current phase of the quadrature demodulator, counting from 0 to $4N-1$ before resetting to 0 again. There are accumulators for the in-phase and quadrature components of the filter.

If the counter is between 0 and $2N-1$, the current sample is subtracted from the in-phase component, and if the counter is between $2N$ and $4N-1$ the in-phase component is added. If the counter is between N and $3N-1$ the current sample is subtracted from the quadrature-phase component, and if the counter is between 0 and $N-1$ or $3N$ and $4N-1$, it is added.

Furthermore, to implement low pass filtering, a moving average filter is applied to the samples. A circular buffer is maintained with the samples. Each frequency with period $4N$ is averaged over M cycles, so that the total FIR filter length is $4NM$. The sample in the buffer that is lagged $4NM$ samples behind the current sample is applied to the current in-phase and quadrature-phase components with the opposite sign as the current sample.

As an example, we consider a typical case of $f=2000$ Hz and $T=0.5$ ms. Typical frequencies to be filtered correspond to periods of 8 ($N=2$), 12 ($N=3$), 16 ($N=4$), 20 ($N=5$), and 24 ($N=6$) samples, or 250 Hz, 167 Hz, 125 Hz, 100 Hz, and 83 Hz respectively.

For OOK operation, the center frequency on which data is received could be chosen to be the $N=3$ channel, for example. The periods 8, 12, and 16 have a least common multiple of 48. The moving average filter corresponding to this 48 period interval corresponds to:

```

+++++-----+++++-----+++++-----+++++-----+++++----- (M=6, N=8, in-phase)
---+++++-----+++++-----+++++-----+++++-----+++++----- (M=6, N=8, quadrature-phase)
++++++-----++++++-----++++++-----++++++-----++++++----- (M=4, N=12, in-phase)
---++++++-----++++++-----++++++-----++++++-----++++++----- (M=4, N=12, quadrature-phase)
++++++-----++++++-----++++++-----++++++-----++++++----- (M=3, N=16, in-phase)
-----++++++-----++++++-----++++++-----++++++-----++++++----- (M=3, N=16, quadrature-phase)

```

These FIR filters are all orthogonal over the interval and therefore achieve the best discrimination between frequency channels for their length. The frequency nulls of each of these channels correspond to the two other channels. This is maintained for any interval that is a multiple of 48 periods.

For FSK operation, the two frequencies on which data is received could correspond to the N=3 and N=5 channels. Because these are lengths 12 and 20, respectively, their least common multiple is 60. The FIR filters corresponding to this are:

```

+++++-----+++++-----+++++-----+++++-----+++++----- (M=5, N=12, in-phase)
-----+++++-----+++++-----+++++-----+++++-----+++++----- (M=5, N=12, quadrature-phase)
++++++-----++++++-----++++++-----++++++-----++++++----- (M=3, N=20, in-phase)
-----++++++-----++++++-----++++++-----++++++-----++++++----- (M=3, N=20, quadrature-phase)

```

This FSK operation corresponds to a modulation similar to MSK, however, unlike MSK, it is insensitive to the phase of the carrier at the beginning of the interval. MSK modulation can be seen as alternating between modulating the carrier wave a half cycle or a full cycle over the 60 cycle interval. The orthogonality of the half and full cycle modulations requires that the phase of the half or full cycle be controlled at the beginning of the interval. The FSK modulation used here corresponds to one or two cycles modulated on the carrier over the 60 cycle interval. The orthogonality of one or two cycles over the interval does not depend on the initial phase of the carrier at the beginning of the interval. This is a much simpler to achieve with crude FSK modulation hardware where there could be phase discontinuities occurring when switching between the two FSK frequencies. The disadvantage is that only one quadrature signal of the two that could be used for sending separate data is being used at a time, halving the spectral efficiency and consequently reducing the signal-to-noise ratio.

The code example shows how to implement these FIR filters with the function `dsp_new_sample()` being called by an interrupt routine with a new sample to update the filters.

Other candidate synchronization code words

During the search for code words, these other candidate 30-bit code words were discovered. The chosen code word was decided by having the maximum number of preceding ones so it would be part of a constant carrier signal. These code words all have a maximum of two complement bits for any shift of the word and a correlation magnitude of 4. These were found by exhaustive computer search.

```
syncword: 1,0,0,0,0,1,1,1,0,1,0,0,0,1,1,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0
autocor: 30 3 -2 -3 0 -1 -2 3 0 -3 -2 -1 4 1 2 -1 -4 1 2 3 -4 1 -2 -1 4 3 2 1 0 -1

syncword: 1,0,0,0,0,1,1,1,1,0,1,1,1,1,0,0,1,0,1,0,0,1,1,1,0,1,1,0,0
autocor: 30 3 -2 -1 -4 3 -2 1 -4 -3 -4 -1 4 3 4 -3 2 3 -2 3 -2 -3 -2 -3 0 -1 2 3 0 -1

syncword: 1,0,0,0,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,0,1,1,0,0,1,0,0,1,0,0
autocor: 30 3 -2 -1 2 -3 0 1 4 3 -2 -3 -2 1 0 1 -2 -3 0 -3 -4 -1 2 -3 2 1 0 3 0 -1

syncword: 1,0,0,0,0,1,1,1,1,1,0,1,1,1,1,0,0,1,1,0,1,1,0,1,0,0,0,1,0
autocor: 30 3 -4 3 -2 -3 4 3 -4 -1 -2 1 4 -1 0 1 -2 -1 -2 -3 -4 -1 -2 3 2 1 0 -1 2 -1

syncword: 1,0,1,1,1,0,0,1,0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,1,1,1,1,0
autocor: 30 3 -4 3 -2 -3 4 3 -4 -1 -2 1 4 -1 0 1 -2 -1 -2 -3 -4 -1 -2 3 2 1 0 -1 2 -1

syncword: 1,0,1,1,1,0,1,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0
autocor: 30 3 -2 -3 0 3 2 3 2 1 -2 -1 4 -1 -2 -1 4 -1 -2 -1 -4 1 0 3 0 -1 -2 1 0 -1

syncword: 1,1,0,0,0,1,1,1,1,0,1,0,0,0,1,0,0,0,0,1,1,0,0,1,0,0,0,0,0
autocor: 30 3 -4 -1 2 -1 0 1 4 1 -4 3 4 3 2 -1 -4 3 2 -1 -2 -3 -2 -3 2 3 0 -1 -2 -1

syncword: 1,1,0,0,0,1,1,1,1,1,0,0,1,1,1,0,1,1,1,0,1,1,0,1,0,0,0,1,0
autocor: 30 3 -2 -3 0 3 2 3 2 1 -2 -1 4 -1 -2 -1 4 -1 -2 -1 -4 1 0 3 0 -1 -2 1 0 -1

syncword: 1,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0,0,0,0,0,1,0,0,0,0,1,1,1,1,0
autocor: 30 3 -2 -1 -4 3 -2 1 -4 -3 -4 -1 4 3 4 -3 2 3 -2 3 -2 -3 -2 -3 0 -1 2 3 0 -1

syncword: 1,1,0,1,1,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0
autocor: 30 3 -2 -1 2 -3 0 1 4 3 -2 -3 -2 1 0 1 -2 -3 0 -3 -4 -1 2 -3 2 1 0 3 0 -1

syncword: 1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1,0,1,1,1,1,0,1,0,0,0,0
autocor: 30 3 0 -1 -4 3 -2 -3 0 3 -2 1 2 3 4 -3 -2 3 0 1 -4 -1 2 -3 0 -1 -2 -1 -2 -1

syncword: 1,1,1,0,0,1,1,0,1,0,1,0,1,1,0,1,1,1,1,0,1,1,1,1,0,0,0,0,0
autocor: 30 3 4 1 0 3 2 1 -2 -3 -4 3 -4 -1 -2 -1 4 3 2 -1 -2 1 4 1 0 -1 -2 -3 -2 -1

syncword: 1,1,1,1,0,1,0,0,0,0,1,0,1,1,1,0,0,0,0,1,1,0,0,0,0,1,0,0,1,0,0
autocor: 30 3 0 -1 -4 3 -2 -3 0 3 -2 1 2 3 4 -3 -2 3 0 1 -4 -1 2 -3 0 -1 -2 -1 -2 -1

syncword: 1,1,1,1,0,1,1,0,1,1,0,0,1,1,1,0,1,1,1,0,1,0,0,0,0,1,1,1,0,0
autocor: 30 3 -4 -1 2 -1 0 1 4 1 -4 3 4 3 2 -1 -4 3 2 -1 -2 -3 -2 -3 2 3 0 -1 -2 -1

syncword: 1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,1,0,0,0
autocor: 30 3 4 1 0 3 2 1 -2 -3 -4 3 -4 -1 -2 -1 4 3 2 -1 -2 1 4 1 0 -1 -2 -3 -2 -1

syncword: 1,1,1,1,1,0,1,1,0,1,0,0,0,1,1,0,0,1,1,0,1,0,0,0,1,1,1,1,0
autocor: 30 3 -2 -3 0 -1 -2 3 0 -3 -2 -1 4 1 2 -1 -4 1 2 3 -4 1 -2 -1 4 3 2 1 0 -1
```

Software License for Code

The code examples in this document are licensed under the zlib license, the text of which is included below.

```
/*
 * Copyright (c) 2021 Daniel Marks

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.
*/
```

Code Appendix

xxx