**BBC Iterative Decoder**

**Overview**

A BBC message consists of M data bits (typically M = 210 = 1024) that has C checksum bits (typically C = 8) appended to the end. The hash function then maps successively longer prefixes of the augmented message to specific mark locations within the L-bit (typically L = 220 = 1,048,576) codeword. In addition, the first and last codeword bits are set (these are the “bookend” marks).

When decoding a packet, the objective is to extract all codewords covered by the packet. At the decoder level, a “packet” is a series of L bits representing whether the energy received corresponding to a particular mark location is above or below the detection threshold. In general, the packet represents a snapshot in time/frequency of a portion of the received signal.

The first step in decoding a packet is to verify the presence of both bookend marks by confirming that both the first and the last bit in the packet is set. If this criteria is met, the packet may contain codewords and is decoded as described below, otherwise it contains no codewords and processing can advance to the next packet.

**Decoder Variables**

To decode a packet in constant memory, the following arrays and variables are used:

P[L] – An array of bits containing the packet data (this can be a portion of a ring register in the receiver).

D[M+C] – An array of bits containing the message data and checksum bits..

n - In addition, an index, n, is used to track where within the message processing is currently happening.

Strictly speaking, the decoder can be implemented with D[] sized to only hold the M message bits, but more efficient processing can be achieved if the checksum bits are treated as part of the effective message.

**Conceptual Decoding Description**

The decoding process can be viewed as the repetitive application of a core decoding algorithm that begins after a candidate partial message has been created and ends with the creation of the next partial message candidate. What the algorithm does depends on how long the candidate partial message is. The two possibilities are that the current partial message consists solely of data bits or that it is long enough to contain one or more checksum bits. Which case applies is determined by the value of n.

If n < M: Decoding a message bit

If M <= n < M+C: Decoding a checksum bit

Whether decoding a message bit or a checksum bit, the first step is to determine if the present candidate message is covered by the packet by verifying that a mark exists in the packet at the location given by the hash of the candidate message. Using Glowworm, this is done by calling addBit(M[n]).

IF the mark IS found: // The new candidate message is valid and the tree is explored deeper.

If n == (M+C-1), the mark just found is the final checksum bit associated with a valid message. The current message is then written to the output buffer for further processing and processing of the packet then proceeds just as if this final mark had not been found.

If n < (M+C-1), the mark just found indicates that the candidate message has been been culled and decoding can continue by incrementing n and setting D[n] equal to 0. This ends this iteration of the decoding algorithm

IF the mark is NOT found: // The new candidate message is culled and the tree is backtracked.

WHILE n >= M: // The hash function must first be rewound to remove the checksum bits

Remove a 0 from the candidate message (delBit(0)).

Decrement n

WHILE n > 0 AND D[n] == 1: Backtrack removing 1s until the first 0 is encountered

Remove a 1 from the candidate message (delBit(1)).

Decrement n

IF n < 0: // The packet has been fully decoded

Break and proceed with the next pack

ELSE: // Move over to the 1 branch of the current search

Set D[n] = 1 to create next candidate message.