Computer Systems, Inc.
MERCURY

FM3TR Packetizer Component

FM3TR Waveform Reference Implementation

SDR Forum Contract

March 23, 2007

Revision 1.0

Table of Contents

1 Component Name

FM3TR Packetizer (FM3TR_WaveformPacketizer)

2 Component Processing Summary

The FM3TR packetizer assembles frames of data by adding additional control information for the receiver. This information includes the operational mode (voice/data) as well as the number of hops per frame.

3 Where used

Both voice and data waveforms.

4 Data Input and Output Ports

The FM3TR packetizer has one provides ("FM3TR_WaveformPacketizerIn") and one uses ("FM3TR_WaveformPacketizerOut") port. The input port accepts blocks of data from either the CVSDEncoder or RsHopEncoder components, adds the appropriate packetizing overhead and pushes to the output data port. Both ports operate on char sequences.

5 Control Interfaces

The FM3TR Packetizer inherits the control interfaces from CF::Resource. Additionally, the component contains a control interface, "LLC_control_input," which is used for pausing data flow to components.

6 Component SCA Properties

Aside from the DLL execparams, the FM3TR packetizer has just one SCA property which determines in which mode the component should run. The property, "Operational_Mode," can take on values "data" or "voice."

7 Component Attributes/Key Variables

Below is a list of several key variables to the FM3TR packetizer with a brief description of their purpose.

`m_a_code`	This is a binary sequence (±1) that has good auto- and cross-correlation properties. The variable itself is actually an array of chars, 32-elements long. The "a" code is used only once and at the beginning of the frame so that the receiver to know when to start logging data.
`m_s_code`	Similar to `m_a_code`, `m_s_code` is a 32-element

	binary sequence with good auto- and cross-correlation properties. The "s" code is used many times by the packetizer to convey control information, including operational mode and hop rate. `m_inv_s_code` is simply the inverse of `m_s_code`.
`m_llc_enable`	Boolean variable which governs whether or not the Run() method should continue

8 Processing Details

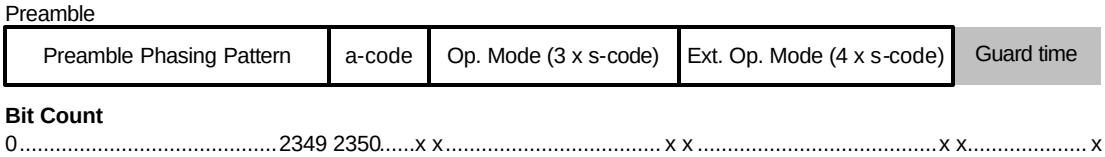
Processing behind the FM3TR packetizer is fairly straighforward. Regardless of the operational mode, the packetizer encodes exactly 320 bits of data per frame divided equally into four data hops. A single synchronization hop at the end of the frame is used for control information, thus totalling 5 hops per frame. Each data hop totals 80 information bits with a few rise- and transition-time bits as buffers between each hop. These bits are ignored by the receiver and are only used as spacers such that the RF hardware has time to switch frequencies. These bits are assembled in the `m_output_frame` buffer by the EncodeFrame() method.

8.1 Method: SendPreamble()

According to the FM3TR specifications, the preamble consists of four major parts:

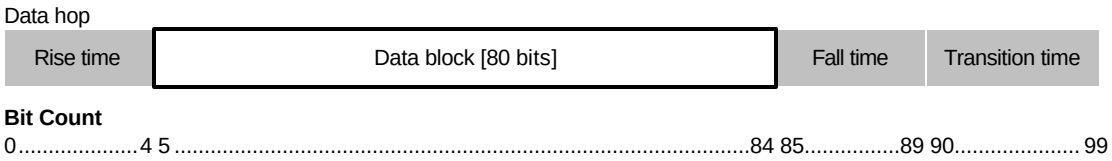
- A synchronization phasing pattern (sequence of `101010`… 2,350 bits long)
- Synchronization a-code
- Operational mode (3 × s-code)
- Expansion mode (4 × s-code)

The entire preamble is transmitted at run time, but is not pushed as one entire block, but rather broken up into more manageable pieces. Because the phasing pattern is so long, it it sent in 512-element blocks. The figure below, although not to scale, demonstrates the packet configuration of the entire preamble.

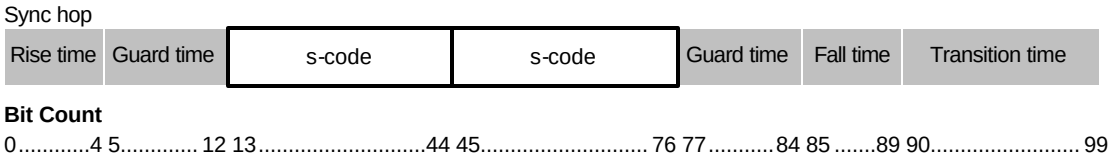


8.2 Method: EncodeFrame()

The only difference between voice and data modes in the EncodeFrame() method is the way the incoming data are packed. In voice mode, the data arrive directly from the CVSD encoder which regards each byte of data as a single bit. This CORBA sequence is exactly 320 elements long. However, in data mode, the data arrive from the RsBlockEncoder component packed into 64 char elements, each with 5 significant bits. This translates into 320 bits as well, but the EncodeFrame() method must first unpack the data. Once unpacked, the data are packetized the same way as with voice mode.
A frame, as stated above, consists of exactly five hops, each 100 bits in length. The first four hops contain application data while the fifth hop is for synchronization. The data hop contains 80 bits of relevant data; the remaining 20 bits are used as rise/fall/transition times between hops, viz.



The synchronization hop includes a special code that informs the receiver an end-of-message (EOM) marker. This is achieved by adding two inverse s-codes. If not at EOM, the packetizer uses two non-inverted s- codes. A bit-wise diagram of the sync hop is depicted below.



8.3 Method: StartStream()

With the MAC and LLC extensions, the FM3TR waveform has the capability of conditionally pausing the flow of data. The StartStream() method allows the Run() method to be invoked by setting the flag `m_llc_enable` to "true." The StartStream() method is called by the FileInput_MAC_LLC component through the "LLC_control_input" control port.

8.4 Method: StopStream()

Execution of the FM3TR packetizer can be paused if the StopStream() method is invoked. Calling this method set the `m_llc_enable` flag to "false" which prevents the Run() method from being invoked until StartStream() is called.