Miller Engineering Services, Inc.

Low Rate Data Modem (LRDM) System Product User Manual

Miller Engineering Services, Inc.
E-mail: sales@mesi.net
Web: http://www.mesi.net

1 Introduction

Low Rate Data Modem (LRDM) ties the MESi data pump components into a fully functional data modem. The suite is composed of the data pumps, AT command and response handlers, data handlers (V.42 or V.14), a sequencer and a hook for user functions.

The system is highly configurable and unneeded components can be conditionally removed with pound-defines at build time to save space and mips. The system supports multiple instances (channels) of modems and can be configured to operate on a sample by sample basis or collect a number of samples and then bulk process them.

2 Getting Started

2.1 Pre-built executable

The easiest way to get started with LRDM is to load and run the LRDMdemo.out file provided with the delivery. Connect the UART to the comm port on your pc and run your favorite terminal emulator program. Configure the terminal program for 115,200 baud, 8 data bits, 1 stop bit and no parity (8N1). Connect the Telco RJ-11 jack to a phone jack. Call the LRDM platform from another modem and the LRDM system will answer the phone. On the terminal emulator, you will see a 'Connect' message when the modem handshake is finished. That's it!

2.2 Building LRDM

To build the LRDM system, change directories to the LRDMdemo for the particular target you are using. For example, if you are build at Texas Instruments C54x, the directory would be \Mesi\products\systems\LRDM\c5400\LRDMdemo.

From the command line type:

make –D*TARGET*

where:

TARGET is the name of the target board. You have built a single channel LRDM system. This is the same LRDMdemo.out executable that ships in with your system.

3 Configurations

3.1 Sample Processing

As the codec samples become available in the Interrupt Service Routine (ISR), they can be processed in the ISR or simply stored in the LRDM's sample buffer memory for processing later. The selection of sample processing is done with the #define NUM_SAMPLES. Setting NUM_SAMPLES=1 selects sample by sample processing. Setting NUM_SAMPLES to any number other than one set the block size for processing. Setting NUM_SAMPLES also effects the size of the receive and transmit sample buffers.

3.1.1 Sample by Sample

A system that requires the minimum latency would allow processing in the ISR. The coded ISR in LRDM.c has calls to transmitter() and receiver() that generate and process one sample respectively. If for example, the sample rate is 8,000 samples/sec, then when the sample ISR is executed, one receive sample will be process and one transmit sample will be generated by the modem.

This does not affect the data handlers, sequencers and other modules; it only has function calls that effect sample processing. All other functions are done in the SYS_executive loop.

Depending on the target system, this may inhibit processing of other interrupts. A timing analysis needs to be done if the are other time critical processes running on the user's system.

3.1.2 Block Processing Samples

A more common approach is to allow the ISR to collect a number of samples and then allow the system to process them as a block. This lends itself easily to platforms that have operating systems. If run without an operating system in a stand alone system such as lrdmdemo, the receiver and transmitter functions wait until there are NUM_SAMPLES samples available before doing any processing. If used with an operating system, the OS would collect the samples and then
call the SYS_executive to process the samples.

3.2 Multiple Channel LRDM

LRDM can support one or multiple channels. Many of the DSP/EZKit demos are built under the single channel LRDM model. When stereo codecs are available, a special two channel version is available that takes advantage of this. This is primarily ISR support for the codec. The #define NUM_CHANNELS dictates the number of channels.

3.2.1 Single Channel

To build a single channel LRDM system, set NUM_CHANNELS to 1 in the LRDM.opt file and rebuild.

3.2.2 Multi Channel

To build a multi channel LRDM system, set NUM_CHANNELS to the desired number of channels in the LRDM.opt file and rebuild. The operation of a multi channel LRDM system is very similar to a single channel system with one exception. The command and response handlers are active for channel 0 only. Even though commands and responses for the remaining channels are used, there is only one central command/response handler which acts as the central clearing house for command processing.

4 Data Flow, Buffers and Pointers

The flow of samples, symbols, bits and bytes through the modems are logical, but sometimes overwhelming and confusion to the first time user. A brief explanation of the structure is given, followed by a graphical view of the pointers.

4.1 Transmitter

The transmitter is responsible for taking data from the user and producing samples that are modulated on the line.

There are two possible sources of data for the modulator. If the AT handler is used, the data comes from the AT command handler via the UART. The AT command handler produces an internal transmit command that is used by the command handler. The alternate method is to place the data directly into the Tx_DTE[] buffer. The Tx_DTE buffer where the transmit octets are stored is contained in the Data Handler (DH) block (see DH Block below). The following pointers are used to keep track of the data.

- *Tx_DTE_head* is the address of the pointer where the next octet will be written
- *Tx_DTE_tail* is the address of the pointer from which the next octet will be read
- *Tx_DTE_base* is the address of the pointer that points to the beginning of the buffer
- *Tx_DTE_len* is the address of an integer that contains the buffer length

Note that the head, tail and base are pointers to pointers.

Once the octets are received from the user, the next step is to frame them into symbols for the modulator. The framer is typically a V.14 (async-to-sync) or V.42 (HDLC) framer, although any custom framer can be implemented. In short, octets (bytes) are taken out of the DTE buffer, turned into symbols which are written to the DCE buffer for the modulator. Again, the pointers are contained in the DH block.

- *Tx_DCE_head* is the address of the pointer where the next symbol will be written
- *Tx_DCE_tail* is the address of the pointer from which the next symbol will be read
- *Tx_DCE_base* is the address of the pointer that points to the beginning of the buffer
- *Tx_DCE_len* is the address of an integer that contains the buffer length

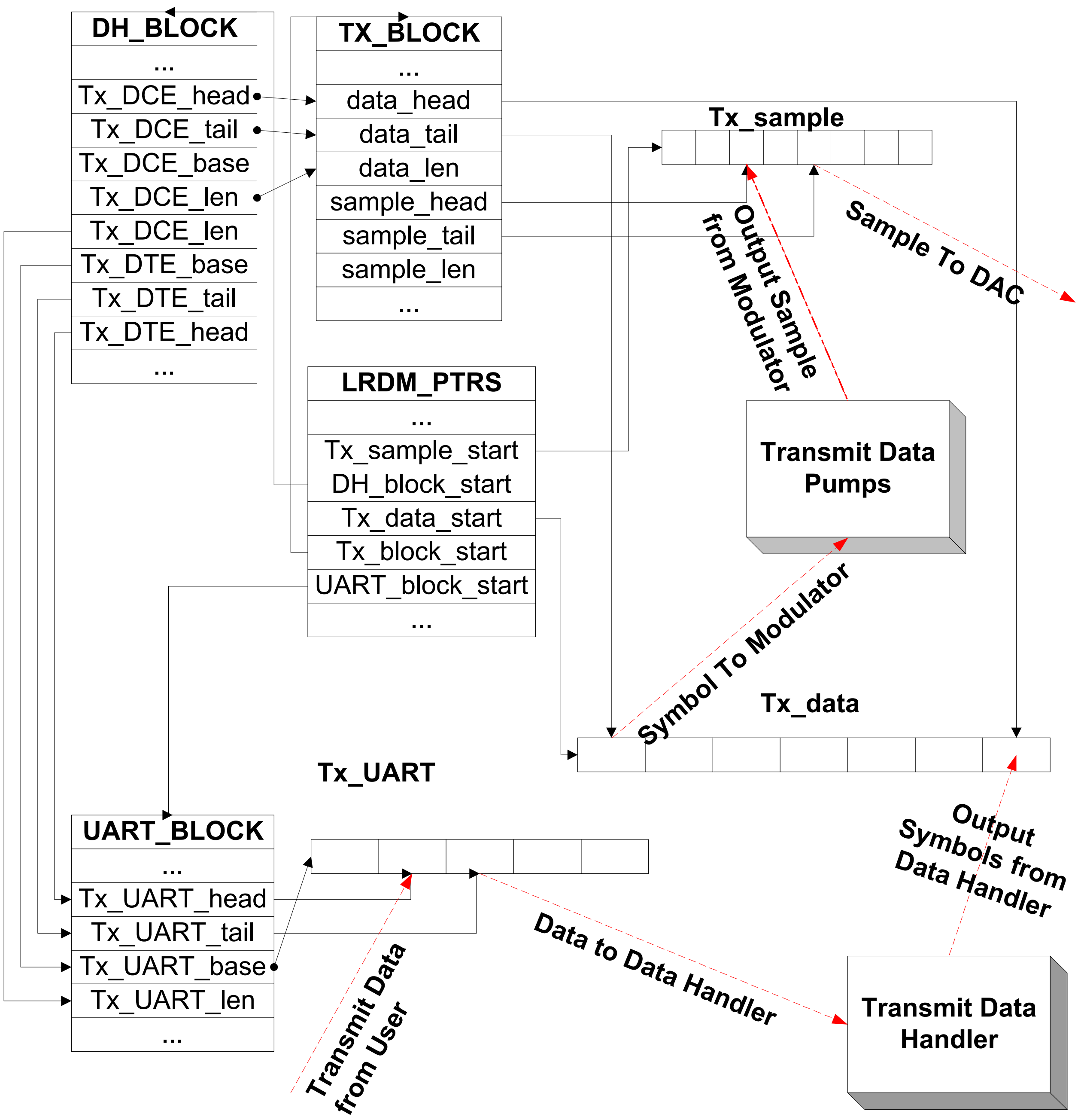
Note that the head, tail and base are pointers to pointers.

Finally, the modulator V.32, V.22, Bell 103, etc, takes the symbols out of the DCE buffer, modulates them and places the output sample in the transmit buffer. The function of the modulator is to convert the digital bits into an analog signal suitable for transmission over the phone line. The sample pointers are contained in the TX_BLOCK and start_ptrs blocks. The transmit block contains

- *sample_head* is the address where the next sample will be written
- *sample_tail* is the address the next sample to be sent to the DAC
- *sample_len* is the length of the sample buffer

and the start_ptrs structure contains

- *Tx_sample_start* is the address of the beginning of the sample buffer


DH_BLOCK
...
Tx_DCE_head
Tx_DCE_tail
Tx_DCE_base
Tx_DCE_len
Tx_DCE_len
Tx_DTE_base
Tx_DTE_tail
Tx_DTE_head
...
TX_BLOCK
...
data_head
data_tail
data_len
sample_head
sample_tail
sample_len
...
Tx_sample
Sample To DAC
Output Sample from Modulator
LRDM_PTRS
...
Tx_sample_start
DH_block_start
Tx_data_start
Tx_block_start
UART_block_start
...
Transmit Data Pumps
Symbol To Modulator
Tx_data
Tx_UART
UART_BLOCK
...
Tx_UART_head
Tx_UART_tail
Tx_UART_base
Tx_UART_len
...
Output Symbols from Data Handler
Data to Data Handler
Transmit Data from User
Transmit Data Handler

4.2 Receivers

The receiver is responsible for taking samples from the line and producing data; symbols or digits.

The demodulator V.32, V.22, Bell 103, etc, takes the samples out of the sample buffers and convert them into symbols

- *sample_head* is the address where the next sample will be written by the ADC
- *sample_tail* is the address the next sample to be processed by the demodulator
- *sample_len* is the length of the sample buffer

As symbols are generated by the demodulator, they are placed into the DCE buffer.

- *Rx_DCE_head* is the address of the pointer where the next symbol will be written
- *Rx_DCE_tail* is the address of the pointer from which the next symbol will be read
- *Rx_DCE_base* is the address of the pointer that points to the beginning of the buffer
- *Rx_DCE_len* is the address of an integer that contains the buffer length

Note that the head, tail and base are pointers to pointers.

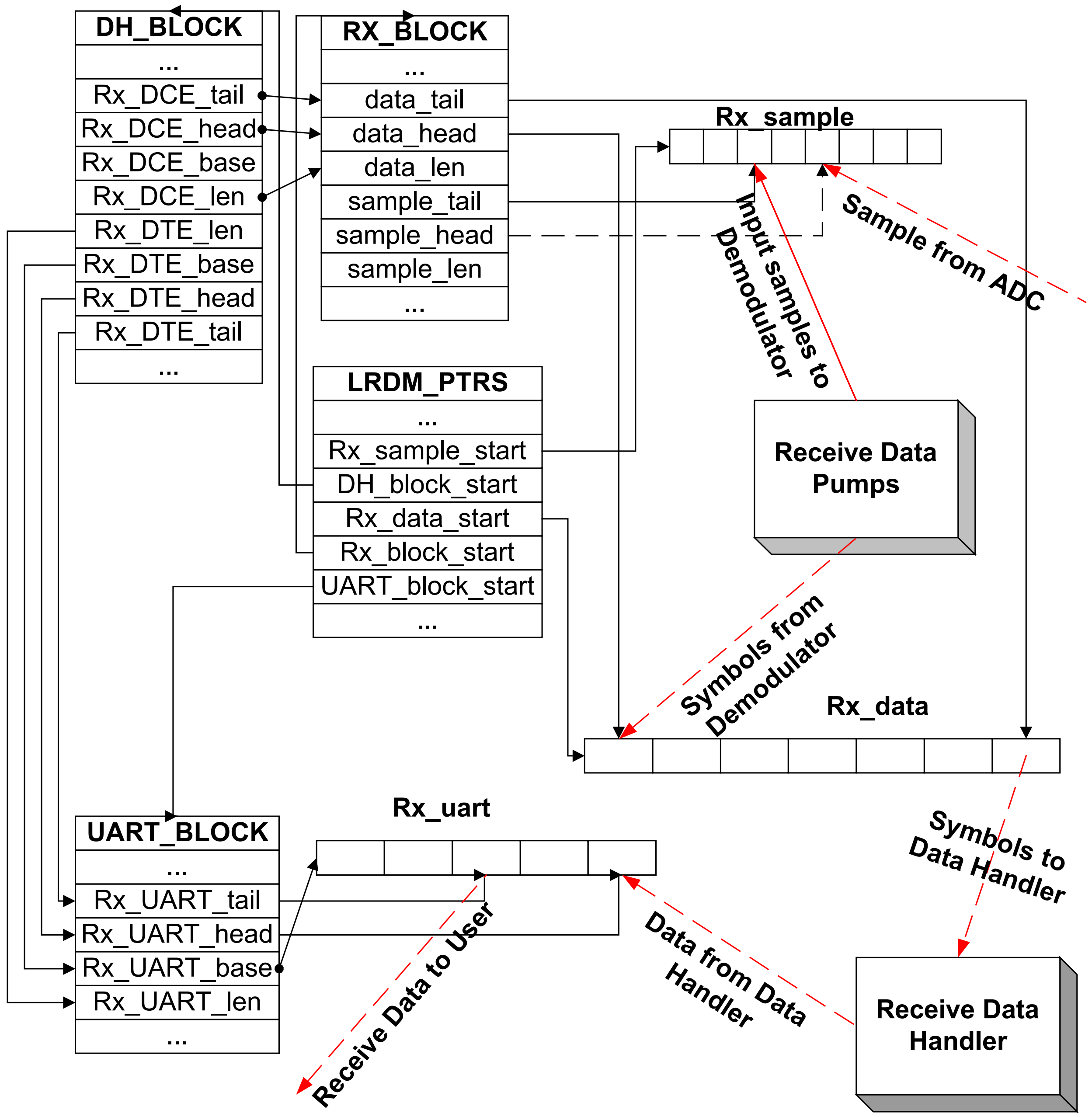
Next, some type of framer then takes the symbols and frames them into octets. The framer is typically a V.14 (async to sync) or V.42 (HDLC) framer, although any custom framer can be implemented. Again, the pointers are contained in the DH block.

- *Rx_DTE_head* is the address of the pointer where the next octet will be written
- *Rx_DTE_tail* is the address of the pointer from which the next octet will be read

- *Rx_DTE_base* is the address of the pointer that points to the beginning of the buffer
- *Rx_DTE_len* is the address of an integer that contains the buffer length

Note that the head, tail and base are pointers to pointers.

Finally, the data is delivered to the user. If the AT handler is running, it collect the data from the DTE buffer and sends a response message to the user. If the AT handler is not being used, the user directly reads the Rx_DTE_head and Rx_DTE_tail pointers. When they are not equal, data is available. The user reads the data from the Rx_DTE_tail and increments the Rx_DTE_tail pointer.


DH_BLOCK
...
Rx_DCE_tail
Rx_DCE_head
Rx_DCE_base
Rx_DCE_len
Rx_DTE_len
Rx_DTE_base
Rx_DTE_head
Rx_DTE_tail
...
RX_BLOCK
...
data_tail
data_head
data_len
sample_tail
sample_head
sample_len
...
Rx_sample
Input samples to Demodulator
Sample from ADC
LRDM_PTRS
...
Rx_sample_start
DH_block_start
Rx_data_start
Rx_block_start
UART_block_start
...
Receive Data Pumps
Symbols from Demodulator
Rx_data
UART_BLOCK
...
Rx_UART_tail
Rx_UART_head
Rx_UART_base
Rx_UART_len
...
Rx_uart
Symbols to Data Handler
Data from Data Handler
Receive Data to User
Receive Data Handler

5 LRDM Components

LRDM is a collection of building blocks that allows rapid prototyping of systems. The files lrdm.c and LRDMdemo.c tie these building blocks together and are intended to be used as templates for users to build systems. LRDM is highly configurable and is driven by an options file. The options file contains a series of define statements. The corresponding #ifdef's in the source code effect the build and system operation.

5.1 AT Command Handler

The AT command handler interfaces with the UART and processes AT commands from the user and issues AT responses. The AT handler has two modes of operation: command and online. Note: The UART must be enabled to use the AT handler.

5.1.1 Command Mode

In the command mode, the Rx AT handler takes data from the UART's Rx_UARTT[] buffer and interprets the data as AT commands. When a legal AT command is found, an internal LRDM command is generated and placed in the Command Handler buffer. As responses are generated by the Response Handler, they are placed in the Response Buffer. The AT handler extracts the response from the Response Buffer, generates a plain text response message and places that message in the Tx_UART[] buffer.

5.1.2 Online Mode

In online mode, the AT handler monitors the Rx_UART[] data buffer for the presence of the command escape sequence, normally three plus characters "+++" within a short period of time. If this sequence is found, then the AT handler switches back to Command Mode, otherwise the data is moved from the Rx_UART[] buffer to the Tx_DTE[] buffer. While in

Online Mode, data in Rx_DTE[] buffer is moved to the Tx_UART[] buffer.

5.2 Command Handler

The Command Handler monitors the command buffer for commands to execute. The commands can come from one of two sources. If the AT Handler is enabled, then the AT handler generates commands and places them in the command buffer. If the AT Handler is disabled, then the user can place commands directly into the command buffer. This is normally done by the use of dual ported memory or a host peripheral interface that allows visibility into the DSP memory. When the Command Handler detects a valid command, it interacts with the sequencer, response handler and data handlers to affect the command.

The counterpart of the command handler is the response handler. It responds to the commands issued. To selectively disable responses, see *Response Handler* and *Suppressing ACK Responses* sections below.

Communication between the HOST and DSP is via the HPI port of the DSP. The DSP will allocate two internal structures for this purpose, one for receiving commands from the HOST, and one for transmitting responses and data to the HOST. Each of these structures will contain a circular buffer used to hold the commands or responses. The structures also contain pointers to the head and tail positions of the buffers. In this discussion the head is the position in the buffer where the next write will occur, and the tail is the position where the next read will occur. It is the responsibility of the process performing the read from or the write to circularly modify and update the appropriate pointer. See *Modifying Internal Buffers* below. The following 7 rules define the command structure.

1. Commands to the DSP shall be formatted as a sequential array of bytes. The contents of the bytes are:

Offset	Contents

0	COMMAND_ID
1	CHANNEL_ID
2	PARAMETER_COUNT
3..N	COMMAND_PARAMETERS

2. Commands are placed into the DSPs command buffer by the HOST. The HOST is responsible for checking the command buffer head and tail pointers to determine if there is enough room in the buffer for the command.

3. The DSP polls the head and tail pointers to determine if a command is present in the buffer. If a command is present the DSP retrieves the command, modifies the tail pointer, and attempts to execute the command. If the command executes successfully, the DSP responds to the HOST with an acknowledgment response consisting of the COMMAND_ID and CHANNEL_ID associated with the command, and a PARAMETER_COUNT of zero. See *Suppressing ACK Responses* below to disable this feature. If the DSP cannot execute the command, it responds with a "negative acknowledge" (NACK) response. (See DSP to HOST Communication.) In this case the COMMAND_ID is logically OR'd with 0x0080 before being placed in the response buffer. The CHANNEL_ID and PARAMETER_COUNT for a NACK are as before, in the acknowledgment response.

   NOTE: It is the responsibility of the HOST to ensure commands and their parameters are valid before writing them to the HPI interface. The DSP does not perform exhaustive validation of command parameters.

4. The NACK command response may be returned due to an internal buffer full condition in the DSP, even though there is room in the command buffer. A command (and all its parameters) that is NACK'd is always discarded by the DSP. For example, if the HOST

attempts to send transmission data to the DSP (CMD_TRANSMIT_DATA) and the internal transmit data buffer is full, the command will be NACK'd. It is the responsibility of the HOST to monitor the ACK/NACK command responses and to re-issue commands as necessary. In this example, the HOST should re-issue the CMD_TRANSMIT_DATA command until it is ACK'd, or the HOST process times out. When enough data has been transmitted to free up room in the internal transmit data buffer and the CMD_TRANSMIT_DATA is received by the DSP, the command will be ACK'd.

5. The command structure is located at a base of 0x1000 in DSP memory (as specified in the linker command file). The command structure is specified in command.h. The HOST access to the command structure and the associated command buffer should be as follows. Note that the command buffer is circular, and the pointer modifications must be performed accordingly. All accesses are made though the HPI interface to the DSP.
    a. Read the head pointer from base+1
    b. Read the tail pointer from base+2
    c. Read the buffer start address from base+3
    d. Read the buffer length from base+4
    e. Determine the room available in the buffer
    f. room = buffer tail - buffer head
    g. if room<=0, room += buffer length
    h. room = room - 1

    If there is enough room for the command and its parameters, the command may be written to the command buffer. Do not write partial commands. The command write is as follows. Note that for the C54x devices the pointers are 16-bit values.

    i. Write the COMMAND_ID to the buffer head

j. Circularly increment the head pointer. Note that the macro CIRC_INCREMENT found in vmodem.h may be used for this purpose. The circular increment procedure is as follows:
k. head_pointer+=1
l. if (head_pointer>=start+length) head_pointer-=length
m. else if (head_pointer < start) head_pointer-=length
n. Write the CHANNEL_ID to the (new) buffer head
o. Circularly increment the head pointer
p. Write the PARAMETER_COUNT to the (new) buffer head
q. Circularly increment the head pointer
r. For PARAMETER_COUNT number of COMMAND_PARAMETERS
    i. Write the COMMAND_PARAMETER to the (new) buffer head
    ii. Circularly increment the head pointer
s. Write the new value of the head pointer to base+1

6. Valid CHANNEL_ID's are 0 to 127.

7. Valid COMMAND_ID's and their associated PARAMETER_COUNT's are shown in the table below.

Command Description (see command.h)	**Param_ Count**	**Parameter Range**	**Comment**
CMD_SOFT_RESET	0	-	
CMD_TRANSMIT_DATA	1 to N	0x00 to 0xff	N bytes data limited only by the size of the command buffer, cmd[].
CMD_AUTOANSWER_CONNECT	1	0 to 7	See note below
CMD_DTMF_CAL	0-21	none or	

L_CONNECT		0x00 to 0x0f	
CMD_ECHO_MODE	1	0x00 or 0x01	
CMD_HANG_UP	0	-	
CMD_ERROR_CONTROL_MODE	1	DH_V14_ASYNC_MODE, DH_SYNCHRONOUS_MODE, DH_V42_MODE	These parameters are defined in dh.h, and need to be verified.
CMD_MODEM_RATE	1	300, 1200, 2400, 4800, 7200, 9600, 12000, 14400	
CMD_RETURN_ONLINE	0	-	
CMD_RESULT_CODES_MODE	0	-	
CMD_VERBAL_MODE	0	-	
CMD_BELLCORE_MODE	1	0 or 1	Selects between ITU V.22 and Bell212a modem standards for rate=1200 bits/sec., and

			between ITU V.21 and Bell103 modem standard for rate=300 bits/sec.
			n=0 » enable ITU (v.22 and v.21) modem mode
			n=1 » enable Bellcore (Bell212a and Bell103) modem mode

CMD_SELECT_COMPANDING	1	0 = 16-bit linear (default) 1 = 8-bit A-law 2 = 8-bit Mu-law	Selects the companding for all channels
CMD_SELECT_DIGIT_DETECTOR	1	0 = none (default) 1 = DTMF 2 = MFC R1 3 = MFC R2 Forward 4 = MFC R2 Backward	Selects the detector type for each channel

Note: CMD_AUTOANSWER_CONNECT operation - If autoanswer is currently enabled, an ATA command with a parameter of 0 will disable autoanswer. If autoanswer is currently disabled, an ATA command with a parameter of zero will cause the modem to immediately go off hook and attempt to connect. Autoanswer remains disabled. An ATA command with a non-zero parameter sets the number of rings before going off hook in answer mode and enables autoanswer.

5.3 Response Handler

The Response Handler is the counterpart to the command handler. The response handler generates response messages based on transitions of modem state or external events and places responses in the response buffer. As with the command handler, the behavior is dependent on the existence of the AT Handler. If the AT handler is present, it will extract the response from the response buffer, translate it to an AT response and place the response in the Tx UART buffer. If the AT Handler is disabled, it is up to the user to process the responses.

1. The DSP shall use the following format for each response or data packet passed to the HOST. Each item below represents one (sequential) word in the response buffer.
    i. RESPONSE_ID
    ii. CHANNEL_ID
    iii. PARAMETER_COUNT
    iv. RESPONSE_PARAMETERS

2. The RESPONSE_ID is one of the values defined in table ?. The CHANNEL_ID identifies the channel (time slot) associated with the response. Valid CHANNEL_ID's are 0, 1,…, N-1 for N channels. The PARAMETER_COUNT specifies the number of RESPONSE_PARAMETERS associated with this response, as identified in table ?.

3. There are two types of responses: command responses and unsolicited responses. Command responses echo the COMMAND_ID and CHANNEL_ID received as a command packet, and have a PARAMETER_COUNT of zero. In addition, the COMMAND_ID is logically OR'd with 0x0080 to form a NACK (not acknowledged) RESPONSE_ID to identify commands that have not been accepted.

4. Responses are placed in the response buffer by the DSP. The DSP shall set the HPI interrupt when responses are buffered for the HOST. Note that the HOST is responsible for clearing this interrupt when it processes the responses.

5. The response structure is located at a base of 0x1040 in DSP memory (as specified in the linker command file). The response structure is specified in response.h. The HOST access to the response structure and the associated reponse buffer should be as follows. Note that the response buffer is circular, and the pointer modifications must be performed accordingly. All accesses are made though the HPI interface to the DSP.

   i. Read the head pointer from base+1
   ii. Read the tail pointer from base+2
   iii. Read the buffer start address from base+3
   iv. Read the buffer length from base+4
   v. Determine the number of words present in the buffer
      ```
      words=buffer head-buffer tail;
      if(words<0) words+=buffer length;
      ```
   vi. Read the RESPONSE_ID from the buffer tail
   vii. Circularly increment the tail pointer
   viii. Read the CHANNEL_ID from the (new) buffer tail
   ix. Circularly increment the tail pointer
   x. Read the PARAMETER_COUNT from the (new) buffer tail
   xi. Circularly increment the tail pointer

xii. For PARAMETER_COUNT number if RESPONSE_PARAMETERS
   1. Read the RESPONSE_PARAMETER from the (new) buffer tail
   2. Circularly increment the tail pointer.

xiii. Write the new value of the tail pointer to base+2

xiv. Clear the HPI interrupt

VALID CHANNEL_ID'S ARE 0 TO 127

VALID RESPONSE_ID'S AND THEIR ASSOCIATED PARAMETER_COUNT'S ARE SHOWN IN THE TABLE BELOW.

Response Description	**Param_ Count**	**Parameter Range**	**Comment**
CMD_SOFT_RESET	0	-	Command ack/nack
CMD_TRANSMIT_DATA	0	-	Command ack/nack
CMD_AUTOANSWER_CONNECT	0	-	Command ack/nack
CMD_DTMF_CALL_CONNECT	0	-	Command ack/nack
CMD_ECHO_MODE	0	-	Command ack/nack
CMD_HANG_UP	0	-	Command ack/nack
CMD_ERROR_CONTROL_MODE	0	-	Command ack/nack

CMD_MODEM_RATE	0	-	Command ack/nack
CMD_RETURN_ONLINE	0	-	Command ack/nack
CMD_RESULT_CODES_MODE	0	-	Command ack/nack
CMD_VERBAL_MODE	0	-	Command ack/nack
CMD_BELLCORE_MODE	0	-	Command ack/nack
RSP_RECEIVED_DATA	1-N	0x0000 to 0x00ff	Unsolicited, received data to pass to the HOST. The maximum number of bytes, N is limited only by the size of the response buffer, rsp[].
RSP_CONNECTED	3	see below	Unsolicited, connect message
RSP_NO_CARRIER	0		Unsolicited, no carrier message

NOTE: The following parameters are valid for RSP_CONNECTED:
param 1: (connect rate) 300, 1200, 2400, 4800, 7200, 9600, 12000, 14400
param 2: (modem type) (see sequence.h for definitions) 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040
param3: (error control mode) (see dh.h for definitions) 0x0000, 0x0001, 0x0002, 0x0004, 0x0008

5.4 Suppressing ACK Responses

To prevent LRDM from acknowledging a command, set the most significant bit of the command by oring the command with CMD_SUPPRESS_ACK. This allows selective enabling or suppressing of responses.

For example,

CMD_TRANSMIT_DATA | COMMAND_SUPPRESS_ACK

Negative acknowledgement cannot be suppressed as they indicate an error condition.

5.5 Sequencer

The sequencer controls the initial call establishment and monitors call progress indications for an orderly call startup.

The sequencer is responsible for interfacing with the DAA, ring detection and call cleardown. The sequencer monitors the call connection and provides fail downs to slower modem starndards when the initial connection cannot be provided. The default fallback algorithm is V.32 to V.22 to Bell 103.

Sequencer Start Up

The sequencer starts in the idle state

Seq_idle

Seq_idle	C A	Initial state for sequencer. If the Autoanswer mode bit is set in the sequencer mode register, the sequencer transitions to Seq_wait_for_ring.
Seq_wait_for_ring	A	The sequencer wait for the correct

		number of rings. When the correct number of rings is detected, the sequencer clears the CALL_MODE bit in the mode register, calls thego_off_hook function and then transitions to the Seq_wait_for_digit state.
Seq_billing_delay	C	The sequencer waits for the billing delay period and then transitions to Seq_wait_dialtone state.
Seq_wait_dialtone	C	The sequencer waits for either dialtone detection or a timeout. If dialtone is not detected, the sequencer reports No Dialtone and then transitions to Seq_idle. If dialtone is detected, or if WAIT_DIALTONE_DISABLE is selected in the options file, the sequencer transitions to Seq_dial_DTMF state.
Seq_dial_DTMF	C	The sequencer calls the DTMF generator for each word in the Tx DTE buffer. Each word in the DTE buffer represents one digit. When all of the digits have been dialed (tx->data_head == tx->data_tail), the sequencer transitions to Seq_wait_for_ANS state.
Seq_wait_for_ANS	C	The sequencer monitors the line for ringback, busy, reorder and answertone. If the sequencer times out waiting, it returns to Seq_idle state, otherwise it set up the sequencer for a call mode call and enters the Seq_wait_for_digit state.
Seq_wait_for_digit	C A	Using the sequencer mode bits, the sequencer calls the appropriate transmitter and receiver initialization routines and initializes the V.14 or V.42 data handler. The sequencer transitions

		to Seq_wait_for_carrier state.
Seq_wait_for_carrier		The sequencer waits for the receiver to indicate it is in the message state by monitoring the MESSAGE_ID bit in the receiver block. The sequencer transitions to the Seq_wait_for_data handler state in the message state or call Seq_terminate_connection on timeout.
Seq_wait_for_data_handler	C A	The sequencer waits for the data handler to indicate it is in the connected state by monitoring the DH_CONNECT_BIT in the data handler block. The sequencer transitions to the Seq_track_message state when connected or calls Seq_terminate_connection on timeout.
Seq_track_message	C A	The sequencer monitors the state of the call and calls Seq_terminate_connection when the call is completed or fails.
Function: Seq_terminate_connection		This functions is called when the sequencer needs to clear down a call and return to the idle state in an orderly fashion. The function go_on_hook() is called to place the hookswitch on hook, the transmitter is initialized to transmit silence, the receiver detectors are initialized, and the data handler is set to idle. The sequencer state is set to Seq_idle.

5.6 User Function

The user function is a convenient hook for the user to insert code into the modem. The supplied user funcion is an empty function called every cycle thru the system executive. The user can insert and modify this function as necessary.

5.7 Data Handler

The data handler process the raw data stream to/from the modems and frames them according to V.14 or V.42. V.14 provides synchronous to asynchrounous conversion and V.42 is an HDLC framer. The Data Handler takes symbols from the Rx_DCE[]buffer, processes them according to the current data handler mode (V.14, V.42, etc) and places the output in the Rx_DTE[]buffer. In the transmitting direction, the data handler takes bytes from the Tx_DTE[]buffer, processes them and places symbols in the Tx_DCE[]buffer. The data handler must ensure that enough symbols are available so the transmitter does not underrun and must remove the symbol from the receiver DCE buffer so the receiver does not overflow. If both the V.14 and V.42 data handlers are disabled, the data handler does nothing. It is up to the user to transfer data into and out of the DTE buffers.

5.8 Modems

The modems are heart of the system. In most cases, interfacing directly to the modems is not required. The sequencer, command and response handlers hide most of the details of the system. However, all of the structures are available for inspection and modification by the user. Refer to the *MESi Modem User's Manual* for modem specifics. The modems take symbols from the Tx_DCE[]buffer and modulate them onto the line and take demodulate symbols and place them in the Rx_DCE[]buffer.

5.9 UART

The UART is an optional component to the LRDM system. If the UART and AT Handler are enabled, communication via an RS-232 connection can be made with the LRDM. Modem data is passed between the UART and DTE as necessary. If the UART is disabled, the user must talk directly with the command and response handlers and must take/remove the DTE data from the DTE buffers.

6 Building Blocks

6.1 LRDM_ptrs

The LRDM_PTRS structure is a superset of the modem pointers. These is the handle needed by most LRDM functions. The structure definition is as follows:

```
struct LRDM_PTRS {
   TRANSMITTER_START_PTRS;
   RECEIVER_START_PTRS;
   int *DH_block_start;
   int *UART_block_start;
   int *AT_block_start;
   int *sequencer_block_start;
   int *command_block_start;
   int *response_block_start;
   int *user_block_start;
   };
```

where:

TRANSMITTER_START_PTRS is the start_ptrs of the transmitter
RECEIVER_START_PTRS is the start_ptrs of the receiver
DH_block_start is the pointer to the data handler block
UART_block_start is the pointer to the UART handler block
AT_block_start is the pointer to the AT handler block
sequencer_block_start is the pointer to the sequencer block
command_block_start is the pointer to the command handler block
response_block_start is the pointer to the response handler block
user_block_start is the pointer to the user block

6.2 Rx Block

The Rx_block structure contains a list of pointer to the various memory elements of the low rate data modem. See the *MESi Modem User's Manual* for the structure's contents.

6.3 Tx Block

The Tx_block structure contains a list of pointer to the various memory elements of the low rate data modem. See the *MESi Modem User's Manual* for the structure's contents.

6.4 DH Block

```
struct DH_START_PTRS *start_ptrs;
        int (*Tx_state)(struct DH_BLOCK *);
        int (*Rx_state)(struct DH_BLOCK *);
        int mode;
        int **Tx_DCE_head;
        int **Tx_DCE_tail;
        int *Tx_DCE_len;
        int **Tx_DCE_base;
        int **Rx_DCE_head;
        int **Rx_DCE_tail;
        int *Rx_DCE_len;
        int **Rx_DCE_base;
        int **Tx_DTE_head;
        int **Tx_DTE_tail;
        int *Tx_DTE_len;
        int **Tx_DTE_base;
        int **Rx_DTE_head;
        int **Rx_DTE_tail;
        int *Rx_DTE_len;
        int **Rx_DTE_base
```

where:

**Tx_state* is the action routine of the transmit data handler

**Rx_state* is the action routine of the receive data handler
mode is a bitmap field that has operation specific parameters for the data handlers. The definition of the bitfields are:

Neumonic	ordinal
DH_V14_ASYNC_MODE	0x0000
DH_SYNCHRONOUS_MODE	0x0001
DH_IDLE_MODE	0x0002
DH_V42_MODE	0x0004
DH_V42BIS_MODE	0x0008
DH_V14_SCAN_ENABLE_BIT	0x0100
DH_CALL_MODE	0x0200
DH_LOCAL_DIGITAL_LOOPBACK	0x0400
DH_OFFLINE_BIT	0x0800
DH_CONNECT_BIT	0x1000
V42_FAILDOWN_BIT	0x2000
V42BIS_FAILDOWN_BIT	0x4000
DH_V14_5_DATA_BITS	0X0050
DH_V14_6_DATA_BITS	0X0060
DH_V14_7_DATA_BITS	0X0070

***Tx_DCE_head* is the address of a pointer. The pointer contains the address of the location in the Tx_DCE[]buffer where the next symbol will be inserted by the data handler.
***Tx_DCE_tail* is the address of a pointer. The pointer contains the address of the location in the Tx_DCE[]buffer where the next symbol will be read by the modulator.
**Tx_DCE_len* is the address of the variable containing the length of the transmit DCE buffer length.
***Tx_DCE_base* is the address of a pointer. The pointer is the address of the variable that is the location of the beginning of the transmit DCE buffer.

***Rx_DCE_head* is the address of a pointer. The pointer contains the address of the location in the Rx_DCE[]buffer where the next symbol will be inserted by the demodulator.

***Rx_DCE_tail* is the address of a pointer. The pointer contains the address of the location in the Rx_DCE[]buffer where the next symbol will be read by the receive data handler.

**Rx_DCE_len* is the address of the variable containing the length of the transmit DCE buffer length.

***Rx_DCE_base* is the address of a pointer. The pointer is the address of the variable that is the location of the beginning of the receive DCE buffer.

***Tx_DTE_head* is the address of a pointer. The pointer contains the address of the location in the Tx_DTE[]buffer where the next octet will be inserted by the AT handler or user funcion.

***Tx_DTE_tail* is the address of a pointer. The pointer contains the address of the location in the Tx_DTE[]buffer where the next octet will be removed by the transmit data handler.

**Tx_DTE_len* is the address of the variable containing the length of the transmit DTE buffer length.

**Tx_DTE_base is the address of a pointer. The pointer is the address of the variable that is the location of the beginning of the transmit DTE buffer.

***Rx_DTE_head* is the address of a pointer. The pointer contains the address of the location in the Rx_DTE[]buffer where the next octet will be inserted by the data handler.

***Rx_DTE_tail* is the address of a pointer. The pointer contains the address of the location in the Rx_DTE[]buffer where the next octet will be removed by the AT handler or user function..

**Rx_DTE_len* is the address of the variable containing the length of the receive DTE buffer length.

***Rx_DTE_base* is the address of a pointer. The pointer is the address of the variable that is the location of the beginning of the receive DTE buffer.

6.5 Uart Block

6.6 AT Block

```
struct AT_BLOCK {
   struct LRDM_PTRS *start_ptrs;
   int (*Tx_state)(struct LRDM_PTRS *);
   int *Tx_UART_ptr;
   int channel;
   int mode;
   int *message_ptr;
   int message_len;
   int (*Rx_state)(struct LRDM_PTRS *);
   int *Rx_data_tail;
   int *Rx_data_head;
   int command;
   int cmd_counter;
   };
```

where:

start_ptrs is a pointer to the start_ptrs for channel 0
Tx_state is a pointer to the transmit routine for the AT handler
Tx_UART_ptr is a pointer to the transmit UART block
channel is the channel number
mode is the mode bits for the AT handler (see definition below)
message_ptr is
message_len is
Rx_state is
Rx_data_tail points to the tail of the UART??? Rx_data_buffer
Rx_data_head points to the input of the UART???? Rx_data_buffer
Command is
cmd_counter is

6.7 User Block

6.8 Sequencer Block

```
struct SEQUENCER_BLOCK {
   int (*state)(struct LRDM_PTRS *)
   int mode;
   int mode2;
   int bit_rate;
   int lineQuality;
   int event_counter;
   int old_event;
   int time_base;
   int time_base_low;
   };
```

where:

state is a pointer to the sequencer processing routine
mode is described by the table below

Neumonic	ordinal
SEQ_B103_MODE_BIT indicates Bell 103 is selected by the user or by the fallback logic. If enabled, the appropriate Bell 103 transmitters and receivers are run.	0x0001
SEQ_B202_MODE_BIT indicates Bell 202 is selected by the user or by the fallback logic. If enabled, the appropriate Bell 202 transmitters and receivers are run.	0x0002
SEQ_B212_MODE_BIT indicates Bell 212 is selected by the user or by the fallback logic. If enabled, the appropriate Bell 212 transmitters and receivers are run.	0x0004
SEQ_V21_MODE_BIT indicates V.21 is selected by the user or by the fallback logic. If enabled, the appropriate V.21 transmitters and receivers are run.	0x0008
SEQ_V22BIS_MODE_BIT indicates V.22bis is selected by the user or by the fallback logic. If	0x0010

enabled, the appropriate V.22bis transmitters and receivers are run.	
SEQ_V32BIS_MODE_BIT indicates V.32bis is selected by the user or by the fallback logic. If enabled, the appropriate V.32bis transmitters and receivers are run.	0x0020
SEQ_V34_MODE_BIT indicates V.34 is selected by the user. If enabled, the appropriate V.34transmitters and receivers are run.	0x0040
SEQ_BAUDOT_MODE_BIT indicates Baudot is selected by the user. If enabled, the Baudot transmitters and receivers are run.	0x0080
SEQ_BELLCORE_MODE_BIT	0x0100
SEQ_SYNCHRONOUS_MODE_BIT	0x0200
SEQ_V14_ASYNC_MODE_BIT is set when V.14 synchronous to asynchronous framing is used.	0x0400
SEQ_V42_MODE_BIT is set when V.42 HDLC framing is used.	0x0800
(reserved)	0x1000
SEQ_MESSAGE_BIT indicates that the sequencer is in the message state. The sequencer enter the message state after the datapump receiver sets the MESSAGE_ID bit and exits the message state when Seq_terminate_connection is called; usually in response to a timeout.	0x2000
SEQ_AUTOANSWER_MODE_BIT is set to force lrdm to answer an incoming call without user intervention.	0x4000
SEQ_CALL_MODE_BIT indicates that the sessions was originated the originator (bit set) or by in response to answering an incoming call (bit cleared).	0x8000

mode2 is described by the table below

Neumonic	ordinal

SEQ_MODE2_CHANNEL_MASK masks off the channel number in the mode2 register. Lrdm support 256 channels.	0x00ff
SEQ_MODE2_RING_S0_MASK masks off the number of rings required before LRDM will answer the incoming call.	0x0700
reserved	0x0800
SEQ_MODE2_DIGIT_DETECT indicates LRDM should detect DTMF digits.	0x1000

bit_rate indicates the bit rate in bits per second.
lineQuality indicates the signal quality (currently unused).
event_counter is used to count internal sequencer event.
old_event is used internally to debounce the ring detector.
time_base[_low] implements a 32 bit timer for use by the sequencer.

6.9 Command Block

```
struct COMMAND_BLOCK {
   int (*state)(struct COMMAND_BLOCK *, struct LRDM_PTRS *);
   CIRC *head;
   CIRC *tail;
   CIRC *start;
   int len;
   CIRC *last_command;
   CIRC command[COMMAND_BUF_LEN];
   };
```

where:

state is a pointer to the command processing routine
head is a pointer to the input pointer of the command buffer
tail is a pointer to the output pointer of the command buffer
start is a pointer to the beginning of the command buffer
len is the length of the command buffer in bytes

last_command is a pointer to the last command processed
command is an array of ints used to store the commands

6.10Response Block

The response buffer is the counterpart to the command buffer. It is used to transfer data and responses to commands from the LRDM to the user interface.

```
struct RESPONSE_BLOCK {
   int (*state)(struct RESPONSE_BLOCK *, struct LRDM_PTRS *);
   CIRC *head;
   CIRC *tail;
   CIRC *start;
   int len;
   int channel;
   CIRC response[RESPONSE_BUF_LEN];
   };
```

where:

state is a pointer to the response processing routine
head is a pointer to the input pointer of the response buffer
tail is a pointer to the output pointer of the response buffer
start is a pointer to the beginning of the response buffer
len is the length of the response buffer in bytes
channel is the channel id number
response is an array of ints used to store the responses

7 Accessing LRDM Memory

All of the LRDM memory is available to the user. The structure definitions are available in the header files as well as the components manual and this manual. The user can examine any element in the modem, but should only modify those pointers and data arrays associated with data flow to and from the user interface.

7.1 Circular buffers

Depending on the platform and the circular buffer capabilities of the platform, circular buffers are either on 2^N boundaries or implemented in software. Consequently, the user should always use the CIRC_INCREMENT macro for circular buffer accessing.

Normally, the user should only be writing data to the Tx DTE buffer, and reading data from the Rx DTE buffer. If the user is using the command and response buffers, then access to the Tx and Rx data is done thru the command and response buffers.

7.2 Example code

To write 10 bytes of data from the array UserMsg to the transmit buffer, the user would:

```
  struct DH_BLOCK  *d;
  char UserMsg[10];
  int I;
  /* The user needs to populate the array UserMsg and set
  * the pointer d to the address of the DH_BLOCK here
  */
  for(I=0;I<10;++I)
  {
        **d->Tx_DTE_head = UserMsg[i];   /* copy the byte to the buffer
*/
         CIRC_INCREMENT(                  /* increment the buffer */
             (*d->Tx_DTE_head),
             1,
             (*d->Tx_DTE_base),
             (*d->Tx_DTE_len));
     }
```

In a similar fashion, the user can read the data out of the receive buffer.

```
  struct DH_BLOCK  *d;
```

```
   char UserMsg[128];
   int I,count;
   /* The user needs to set the pointer d to the address
   *  of the DH_BLOCK here.
   */
   count=0;       /* set the count to 0 */

   while(   *d->Rx_DTE_head  != *d->Rx_DTE_tail )
   {
         UserMsg[count] = **d->Rx_DTE_tail;   /* copy the byte from the
buffer */
         ++count;                                   /* increment the count
          CIRC_INCREMENT(                           /* increment the buffer */
              (*d->Rx_DTE_tail),
              1,
              (*d->Rx_DTE_base),
              (*d->Rx_DTE_len));
      }
```

8 Symbol Definitions (#Defines)

To give LRDM its configurability, the system external build-time symbol definitions to override internal #defines. The following list of #defines may be found in the options files (LRDMdemo.opt). There are two main groups of defines; those used to control the LRDM build and general modem defines. At times, the line between system and data pumps blurs, so all defines are listed.

8.1 LRDM System Defines

AT_DISABLE disables the AT command handler. If this is done, the user is responsible monitoring the status bits and making the internal systems calls that are made by the AT handler.

ARROWLRDM_DEMO is defined to build LRDM on the Arrow Dsk board.

COMMAND_BUF_LEN is the length of the command buffer.

COMMAND_DISABLE This disables the inclusion of the command handler. Disabling the command handler means the LRDM will not process commands sent to it. The LRDM will generate response if the response handler is enabled (see RESPONSE_DISABLE).

COMMAND_NUM_CHANNELS is the number of command channels in the system.

DHBLOCKNEEDED determines if a DH block is built in memory. It is driven from V.14 and V.42 frames, but can be invoked for custom framers when the DH memory is needed and the V.14 and V.42 framers are not used.

DH_DISABLE - disables Data Handler code and memory. Used to eliminate Data Handler from LRDM builds

DH_NUM_CHANNELS is the number of data handler structures in the system.

LRDM_DEBUG is additional LRDM debugging software.

LRDM_DEMO builds the demo version of LRDM.

LRDM_PTRS_NUM_CHANNELS defines the number of channels of LRDM in the system.

RESPONSE_BUF_LEN is the size of the response buffer.

RESPONSE_DISABLE disables the response handler. Disabling the response handler means that the LRDM will not generate responses to system commands or system events.

RESPONSE_NUM_CHANNELS is the number of response channels in the system.

SEQUENCER_DISABLE The sequencer control call progress, call termination and control of hook switch (DAA). Disabling the hookswitch should only be used when the user has a specific requirement to operate the modems in a non-standard fashion and is going to provide the necessary modem startup commands.

SEQUENCER_NUM_CHANNELS is the number of sequencer channels in the system.

USER_NUM_CHANNELS is the number of user channels in the system.

USER_DISABLE disables the user code. If there is no user specific code, then the user function can be disabled. In a generic configuration, there is no need for the user function. The modem will function normally thru the AT handler.

8.2 Data Pump defines

OP_POINT_SHIFT - The shift corresponding to the power of two that OP_POINT is defined to be. For example if OP_POINT is 1/8
then OP_POINT_SHIFT is three.

ABORTMODS - is a temporary flag for HDLC to test the handling of the ABORT indication.

ACOUSTIC_ECHO_CANCELLER - enables acoustic echo cancellation module code segments. Defined in config.h

AD73311 - symbol to select Analog Devices AD73311 audio codec driver in target.asm

AEC_DEMO - symbol to select configuration profile for acoustic echo canceller demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

ALAW_COMPANDING - enables compilation of A-Law companding and expansion code in target.asm. A-Law companding is used for voice relays as well as fax intercept.

ALL_ASSEMBLY_SAMPLE_ISR - selects all assembly source code for 8 khz. interrupt routine in target.asm. This would be selected when the standard C source "sample_8khz_isr()" is not used for a sample transfer.

ALL_CSOURCE - symbol to select all C source code build rather than assembly source code build in makefiles.

ALT_DTMF - symbol to select configuration profile for the alternate DTMF detector version in config.h.

ALT_MF - symbol to select configuration profile for the alternate MF detector version in config.h.

ANALOG_IOMIC_SPKR_ANALOG_IO - enables drivers for microphone and speaker sample drivers in target.asm. This typically supports boards with seperate mic/ speaker and phone I/O.

ANALOG_LOOPBACK - ???

APSTUDIO_INVOKED - ???

APSTUDIO_READONLY_SYMBOLS - ???

ARROW5409 - selects the arrow5409 board type in target.asm.

ASSEMBLY_PTRS_INITIALIZATION - ???

ASYNC_RX_TX - ???

ATT_TWIST_SPEC - configures DTMF detection algorithm for compliance with AT&T performance specification in DTMF.C.

AT_DISABLE - disables the AT command handler. If this is done, the user is responsible monitoring the status bits and making the internal systems calls that are made by the AT handler.

AT_NUM_CHANNELS - specifies the number of AT command handler channels. The default is 1 and can be over- ridden externally.

AUTOBUF_DISABLE - ???

B42_MAX_CODEWORD - is the maximum code word size that is transmitted in V.42 bis. This parameter is determined during V.42 negotiations. B42_MAX_CODEWORD is the maximum codeword size that LRDM will support.
The final codeword size is the maximum size supported by both sides of the connection. The code word size is a minimum of 512 and is typically a power of 2. If you specify B42_MAX_CODEWORD larger than 2048, you can
not define V42BIS_COMPACT.

B42_MAX_STRING - is the size of longest string that can be encoded into a codeword. This parameter is determined during V.42 negotiations. B42_MAX_STRING is the maximum string size that LRDM will support. The final string size is the maximum size supported by both sides of the connection. The string word size is a minimum of 6.

BAUDOT_DEMO - symbol to select configuration profile for Baudot TDT modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over-ridden externally by a command- line symbol definition or from a *.opt options file.

BAUDOT_ENABLE - ???

BAUDOT_MODEM - ???

BELL103_DISABLE - symbol existence disables code and memory for BELL103 modem.

BELL_2225_TONE

BPF_FAST_MODE

BRAZIL_PSTN - attenuates Tx->scale1 by 2dB to produce tone twist required in Brazil.

C54CM_

C54XFAR

C54XFARC

CADENCE_CP sets cadence call progress. This is the default and is used in Europe. The counterpart is NORTH_AMERICAN_PSTN.

CALL_MODE is a VSIM define that shows the direction of the call (CALL or ANSWER).

CALL_PROGRESS

CAPTURE_TXSYMS

CED_TONE

CELLULAR sets the Crystal CS4216 to work thru the MIC/SPKR jacks.

CHANNEL_ECHO_LEN is a channel echo model parameter currently set to 2 seconds in config.h.

CHARACTER_GENERATOR

CIDDTMF_DEMO - symbol to select configuration profile for DTMF-based caller ID demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

CIDJAPAN_DEMO- symbol to select configuration profile for Japan caller ID demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

CID_DEBUG - changes the length of the caller ID bursts and the initial transmitter state for faster debugging.

CID_DEMO - symbol to select configuration profile for BELLCORE /ETSI caller ID demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

CIRC - is a typecast for circular buffers..

CIRCULAR_LOG - is used in V42 to provide a continuous circular log. The alternative is for a linear log that terminates logging when the end is reached.

CLEAN_SCREEN

CLIP_DEMO - symbol to select configuration profile for CLIP combination (BELCORE, DTMF, Japan) demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

CMD_DIGITAL_LOOPBACK

CNG_TONE

COMMON_MODEM

COMPANDING

COMPILER - is set in config.h to indicate if the compiler is used or if code is purely assembly code. The default is ENABLED.

CONSTELLATION_DISPLAY- is used in conjunction with VSIM to save the I and Q constellation points for further processing.

CPTD_DEMO - symbol to select configuration profile for call progress tone detection (dial tone, ringback, busy) demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

CREATE_VECTORS

DAA_CHANNEL_ASSIGNMENT

DECODER_BLOCK_LEN - is the size of the trellis decoder block and is set in config.h based on which decoder is implemented.

DECODER_BLOCK_NUM_CHANNELS - is the number of tcm decoder channels in the system.

DEFAULT_AUTOANSWER - enable automatic answer for incoming calls

DFTCOEF_IN_PROGRAM_MEMORY

DIGITAL_LB

DIGITAL_LOOPBACK

DISABLED - is a generic define set to 0.

DOS_GRAPHICS - is used in VSIM to link in the Borland graphics utilities to do DOS screen graphics.

DSK5402 - defines the target as a TEXAS INSTRUMENTS DSK5402.

DSK5416 - defines the target as a TEXAS INSTRUMENTS DSK5416.

DSK_V22

DSK_V32

DTE_LOOPBACK

DTMF_API_DEBUG

DTMF_DEMO - symbol to select configuration profile for DTMF generator/ detector demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

DTMF_DETECT

DUMP_BUFFER enables the logging of anything in modem by using dump_write(). The amount of data logged is dictated by DUMP_LEN. The dump buffer is a linear array and logging stops when the DUMP_LEN is reached.

DUMP_LEN is the length of the DUMP_BUFFER.

ECHO_CANCELLER

EC_COEF_LEN is the number of taps in the echo canceller and is set in config.h.

EC_COEF_NUM_CHANNELS is the number of echo canceller channels in the system and is defined in config.h.

ENABLED is a generic define set to '1'.

ENCODER_BLOCK_LEN is the size of the trellis encoder block and is driven by the selected trellis encoder.

ENCODER_BLOCK_NUM_CHANNELS is the number of encoder blocks in the symtem and is defined in config.h.

EQ_COEF_LEN is the number of taps in the equalizer and is defined in config.h.

EQ_COEF_NUM_CHANNELS is the number of equalizer channels in the system and is defined in config.h.

EQ_PLOT_MODE is used with VSIM graphics to plot either the equalized signal or the equalizer taps.

EVM541

EVM5510

EXTERNAL_DCE_BUFFERS Normally, the DCE data buffers are created along with other memory element. As symbols are transferred to/from the modems, they are placed into and taken from the DCE buffer. Since the data handler and modems act as an integral unit, there is no need to have external DCE buffers. As symbols are received by the data pumps, they are place in the Rx_Block's DCE buffer and the data handler accesses them from this buffer. In most cases when operating with a data handler, the EXTERNAL_DCE_BUFFERS should be enabled.

EXTERNAL_DTE_BUFFERS Normally, the DTE data buffers are created in the DH block along with the other memory elements. This means that as characters are transferred to/from the modems, they are placed into and taken from the DH DTE buffer. However, if LRDM is built with the UART enabled, memory is allocated in the UART block for the data and the data would have to be transferred from the UART block to the DH block. It is far more efficient to have the Data Handler's DTE receive buffer be the UART's transmit buffer and the DTE transmit buffer be the UART's receive buffer, eliminating the transfer of data and decreasing the amount of memory required. To do this, \#define EXTERNAL_DTE_BUFFERS in the options file. LRDM then creates only the UART's buffers and maps the Data Handler's buffers into the UART buffer.

EXTERNAL_UART_CLK is set when the UART clock is derived from an external source. The default is internal UART clock.

EZBF535 - set when the target is the BF535 Blackfin DSP.

EZ2181 - set when the target is the ADI 2181 EZ-Kit.

EZ2189 - set when the target is the ADI 2189 EZ-Kit.

EZ2191 - set when the target is the ADI 2191 EZ-Kit.

EZ_V22 is defined for the internet downloadable V22 demo program and has a subset of the capabilities of the modem.

EZ_V32 is defined for the internet downloadable V32 demo program and has a subset of the capabilities of the modem.

FAKE_NEEDED

FARC_MODE is defined when the FAR memory model calls are used for function calls from external sources to the modems and NEAR memory model calls for internal function calls. The default is for near memory model calls. See also FAR_MODE.

FAR_MODE is defined when FAR memory model calls are used for all function calls. The default is for near memory model calls. See also FARC_MODE.

FAX_DATA_DEMO - symbol to select configuration profile for FAX bundle (v17, v21, v27, v29, gendet)and data bundle (v22, v32)demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

FAX_DEMO - symbol to select configuration profile for FAX bundle (v17, v21, v27, v29, gendet) demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

FEC_COEF_LEN is the number of taps in the far echo canceller. See also EC_COEF_LEN and NEAR_EC_LEN.

FILE_IO is used when samples are taken from files instead of real time sample buffers.

FIR_SITE_INTERRUPTS_DISABLED

FOUR_WIRE

FPRINTF_BINS is used in VSIM to printf the goetzel energy bins in gendet.

FPRINT_VECTORS

FULL_DEMO

FULL_RATE

FUN_TRACE

GENDET_DEBUG allows analysis() to be called every sample during debug. Normally, analysis is called after RX_ANALYSIS_LEN samples have been collected.

GNU is used when there are differences between the GNU user interface and the windows one. It is applicable mainly to vsim for linux.

GOERTZEL_SAMPLE_RATE is the rate of the Goertzel transform and is set to 4000 Hz.

GUARD_TONE provides a guard tone for V22.

HARDWARE_UART allows the use of the onboard hardware UART. The hardware UART and software UARTs are mutually exclusive. If HARDWAREWARE_UART is defined, SOFTWARE_UART_DISABLE must also be defined.

HAWK81 - set when the target is the ADI HAWK 2181.

HW_SYSTEM_DELAY is the propagation delay that the modems compensate for when computing echo delay in the system.

INFO_DET_OVERSAMPLE_RATE is the oversampling rate of the INFO messages in V34.

INTERNAL_UART_CLK is set when the UART generates internal clocks. See also EXTERNAL_UART_CLK.

INTERP_DEC_EVAL is test code for a new interpolator and decimator evaluation.

INTLEN32 should be deprecated. The length of UINT_MAX in limits.h should be used instead.

IQ_DAC_WRITE enables the output constellation to be written to DACS for debugging. This option only applies to boards that have output DAC capabilities.

IQ_DUMP_WRITE enables the output constellation to be written to the debug buffer.

ISR_DEBUG

ISR_LOOPBACK loops back the transmit samaples to the receiver for loopback testing.

KTXNEEDED

L3_MODS

LOG_SAMPLES

LOGSIZE is the size of the V42LOG buffer.

LONG_TIMER is used to get a 32 bit timer resolution.

LOOPBACK

MAX_CHECK

MCBSP0

MCBSP1

MEASURE_RX_MIPS is set to measure the receiver mips.

MEASURE_TX_MIPS is set to measure the transmitter mips.

MESI_COMMENTS is a variant of MESI_INTERNAL.

MESI_INTERNAL is internal embedded documentation in the code that is meant to be stripped out before delivery.

MFDTMF_DEMO - symbol to select configuration profile for DTMF and MF generator/detector demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

MF_API_DEBUG

MF_DEMO - symbol to select configuration profile for MF (R1, R2f, R2b) generator/ forward detector demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

MF_DETECT

MIC_SPKR

MIPS_MEASURE

MULTICHANNEL_ISR

MULTIPLE_NAMED_CHANNEL_CREATION_TEMPLATE

NEC_COEF_LEN FEC_COEF_LEN is the number of taps in the near echo canceller. See also EC_COEF_LEN and FAR_EC_LEN.

NEED_REALTIME is used in the fax relay to signify that the realtime sample counter is needed in the sequencer.

NORTH_AMERICA_PSTN select NORTH_AMERICAN call progress detections as the default. The counterpart is CADENCE_CP, which is used in Europe.

NO_LO_ROTATOR removes the local oscillator phase rotator from the receiver.

NO_PRINT

NO_TELCO_INTERFACE

NUM_CHANNELS sets the number of channels. This effects the amount of system memory allocated as well as the number of channels called. There are three basic configurations to consider:
One - Used for single channel demo system. ISR supports one codec channel.
Two - Used for loopback demos. ISR supports two codec channels.
More than two - User systems with multi-channel support. No demo available.

NUM_SAMPLES dicates how many samples are processed each call. There are two cases of note:
One sample - data is processed in the ISR
More than 1 sample - data is processed in SYS_executive loop

NUM_SND_BUFS

ODIN548 - set when the target is the Odin TEXAS INSTRUMENTS C548 DSP.

ONBOARD_CPC5604_DAA

ON_CHIP_COEFFICIENTS is used in TEXAS INSTRUMENTS Dsp's to load the vcoefs data into program memory. This is needed to do efficent multiply and accumulate (MAC) operations in the dsp. To use this feature, the coefficients must be located in memory that is overlaid between program and data. The memory must also have the identical address. For example, on the TEXAS INSTRUMENTS C54x family, memory from 0x0000 to 0x7FFF is accessible as both program and data memory.

OP_POINT is the reference point used for the modem.

OVERLAY_MODE Now, when OVERLAY_MODE=ENABLED, the OVLY bit is enabled in PMST immediately upon entry to transmitter() and receiver() and stays in overlaid mode always. If OVERLAY_MODE=DISABLED then the device will still work if the "vcoefs" section is in physically overlaid memory area, and the chip was in overlay mode already. The ON_CHIP_COEFFICIENTS constant is unchanged, but now it really operates like a PSRAM/DSRAM switch for the vcoefs section. If ON_CHIP_COEFFICIENTS=ENABLED then the vcoefs section is assumed to be in DSRAM - not necessarily on-chip (hence the name is a mis-nomer). If ON_CHIP_COEFFICIENTS=DISABLED, then vcoefs is assumed to be in PSRAM and OVERLAY_MODE becomes relevant. However, now you can set OVERLAY_MODE = DISABLED and ON_CHIP_COEFFICIENTS = ENABLED, and force the linker to locate the "vcoefs" section in DSRAM. Assuming that the loader can initialize DSRAM, then the system will work without any need to alter PMST or to copy coefficients from PS to DS (xDAIS requirement) (03-14-00).

PAGED_218X_MEM is set when the memory on the ADI 281x target is paged.

PARALLEL_IO

PCM3002 is set to indicate that the PCM3002 codec is being used on the target platform.

PC_SIMULATION is for debugging V.42bis compression. With this defined, code words are output to STDIO.

PJ_RESONATOR_TYPE

POLL_ENABLED

PSTN_TYPE

RECEIVER is the main define for the receive side of the modems. It is set in config.h and must be enabled in order to do any receive signal processing.

REMOTE_DIGITAL_LOOPBACK enable loopback mode for received data.

RS232_RX_BUFF_SIZE

RTS_CTS_ON_MCBSP is defined for TEXAS INSTRUMENTS DSPs that has a UART on one of the multi channel buffered serial ports.

RTS_CTS_ON_MCBSP0 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 0.

RTS_CTS_ON_MCBSP1 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 1.

RTS_CTS_ON_MCBSP2 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 2.

RTS_CTS_ON_XF_BIO is defined in the Request to Send and Clear to Send functionality is provided on the XF pin. This is applicable only to Texas Instruments DSPs.

RXTX_DEMO - symbol to select configuration profile for basic reciever/ transmitter (silence and idle) demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

RX_BLOCK_LEN is the length of the receive structure. The size of the receive block depends on which components of the receiver are enabled.

RX_BLOCK_NUM_CHANNELS is the number of channels of receiver blocks the system can support.

RX_DATA_LEN is the length of the Rx_DCE[]buffer in symbols.

RX_DATA_NUM_CHANNELS is the number of channels of receive data that are support.

RX_DTMF enable DTMF detection (override config.inc)

RX_FIR_LEN is the size of the receive finite impulse filter. It is set in config.h, but for most applications is 128.

RX_FIR_NUM_CHANNELS is the number of channels of receive fir data.

Rx_FRAMESIZE is the size of the largest frame that can be sent from the Responder to the Initiator. This parameter is negotiated during V.42 XID. The default value is 128.

RX_FSK_MODEM is set in config.h to enable to fsk modem demodulation.

RX_MF is set in config.h to enable MF detection.

RX_NUM_CHANNELS is the number of receive channels in the system.

RX_NUM_SAMPLES is the number of samples that are processed at a time. The receiver waits until RX_NUM_SAMPLES are available in the sample buffer and then processes them. If receiver is called with less than RX_NUM_SAMPLES of data, it simply returns 0.

RX_SAMPLE_LEN is the length of the receive sample buffer. It is normally set in target.h as part of one of the DEMO configurations but can be manually set to any length. The minimum size is the larger of 2*NUM_SAMPLE+1 or NUM_SAMPLES+81.

RX_SAMPLE_NUM_CHANNELS

RX_DATA_LEN is the size of the receive symbol buffer (DCE). There must be enough symbols in the symbol buffer that correspond to NUM_SAMPLES of time, plus 1. For example, if NUM_SAMPLES is 80 (10 ms) and the baud rate is 2400, then there must be 2400*0.10 + 1= 25 symbols. If a fractional number results, it must be rounded up.

RX_UART_LEN is the length of the UART structure.

SCOPETRACES allocated memory for the windows version of vsim for the scopetraces. Two things must happen for a trace to be visible in windows. The first is that the bitfield in SCOPETRACES must be set. The second is that the particular function must be enabled at run time. The SCOPETRACES define builds the capabilities and the runtime option then selects it.

SERIAL_PORT_LOOPBACK

SHOW_GLOBAL

SI303X

SI3044

SILABS_30XXEVB_DAA

SIM_2191_WORKAROUND

SOFTWARE_UART_DISABLE disables the software UART (where applicable). Some DSP boards have both a hardware UART and a software UART. Only one UART can be enabled. If HARDWARE_UART is enabled, the SOFTWARE_UART_DISABLE must also be defined.

SPEED is a generic define chooses the faster alternative where size or speed can be gained at the expense of the other.

SPORT_DEFINED

SQUARE_ROOT_WHAT enables the computation and processing of the estimated signal point W - hat.

STAND_ALONE

START_PTRS_NUM_CHANNELS is the number of start pointer structures in the system.

STDIO is define in the modem sources to write debug data to the STDIO.

STU_III_DEMO - symbol to select configuration profile for demonstration STU III (v26, v32, gendet) in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

SUPPRESS_V22A_ANS disable answer tone

SUPPRESS_V22A_SILENCE disable answer mode silence prior transmitting answer tone. The silece period is typically used by the local telephone companies for transmitting billing information and is required for use on PSTN's.

TDK_DAUGHTER_CARD is defined when the codec on a TDK daughter card.

TESTBED is a build mode that generates a testbed V.32 debug platform.

TIGER5410 is defined when the target is a TEXAS INSTRUMENTS Tiger C5410 DSP.

TIGER542 is defined when the target is a TEXAS INSTRUMENTS Tiger C542 DSP.

TIGER549 is defined when the target is a TEXAS INSTRUMENTS Tiger C549 DSP.

TIGER54X is defined when the target is a TEXAS INSTRUMENTS Tiger C54x DSP.

TIMER_CONTROL_EVALUATION

TLC32AD50

TMP00

TONE_DEMO - symbol to select configuration profile for programmable tone generator and power measurement demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

TRACEBACK_LEN is the lookback size for the trellis decoder.

TRACEBACK_NUM_CHANNELS is the number of trellis decoder structures.

TRAIN_LOOPS_COARSE_TIMING_EVAL is eval code for training loops.

TRANSMITTER is the main define for the transmitter side of the modems. It is set in config.h and must be enabled in order to do any transmit signal generation.

TXD_RXD_ON_MCBSP is defined for TEXAS INSTRUMENTS DSPs that has a UART on one of the multi channel buffered serial ports.

TXD_RXD_ON_MCBSP0 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 0.

TXD_RXD_ON_MCBSP1 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 1.

TXD_RXD_ON_MCBSP2 is defined for TEXAS INSTRUMENTS DSPs that has a UART on multi channel buffered serial ports 2.

TX_DATA_LEN is the size of the transmit symbol buffer (DCE). There must be enough symbols in the symbol buffer that correspond to 2*NUM_SAMPLES of time, plus 1. For example, if NUM_SAMPLES is 80 (10 ms) and the baud rate is 2400, then there must be 2*2400*0.10 + 1= 49 symbols.

TX_DATA_NUM_CHANNELS is the number of channels of transmit dce buffers in the system.

TX_FIR_LEN is the length of the transmit fir filter, nominally 12.

TX_FIR_NUM_CHANNELS is the number of transmit fir channels in the system.

Tx_FRAMESIZE is the size of the largest frame that can be sent from the Initiator to the Responder. This parameter is negotiated during V.42 XID. The default value is 128.

TX_GAIN_RESCALING

TX_MODEM_LEN

TX_NUM_CHANNELS
TX_NUM_SAMPLES is the number of samples generated at a time in the transmitter. The transmit algorithm wait the number of samples in the transmit buffer is less than or equal to TX_SUM_SAMPLES and then calls the transmit routines.

TX_SAMPLE_LEN is the length of the transmit sample buffer. It is normally set in target.h as part of one of the DEMO configurations but can be manually set to any length. The size must at least 2*NUM_SAMPLE+1.

TX_SAMPLE_NUM_CHANNELS is the number of channels of transmitters in the system.

TX_UART_LEN is the size of the transmit UART structure.

UART_ASSEMBLY is defined when the UART is entirely in assembly.

UART_BAUD_RATE control the UART baud rate. Supported rates are 19,200 and 115,200 baud.

UART_BAUD_RATE115200 is defined when the baud rate is 115,200.

UART_BAUD_RATE19200 is defined when the baud rate is 19,200.

UART_CLK_100MHZ is defined on TEXAS INSTRUMENTS DSPs when the UART clock is 100 MHz. See also UART_CLK_92MHz.

UART_CLK_92MHZ is defined on TEXAS INSTRUMENTS DSPs when the UART clock is 92 MHz. See also UART_CLK_100MHz.

UART_CLK_DIVIDER

UART_CSRC_ISR is defined when the UART is in C. See also UART_ASSEMBLY.

UART_DISABLE is set when there is no UART. The UART is required if the AT Handler is enabled.

UART_NUM_CHANNELS is the number of UART channels in the system. Nominally, one UART channel is supported in the system.

ULAW_COMPANDING

UMD_TRETTER_PRECODER uses the Univeristy of Maryland's V.34 Precoder.

USE_BF is defined in the fax relay to use bit field in structures to conserve space.

USE_SEQ_NR is defined the fax relay to insert sequence numbers into the data packets.

V14_DISABLE disables the V.14 data handler. See ???? for the operation of the data handler.

V14_DISABLED

V21_DEMO - symbol to select configuration profile for v.21 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V22_DEMO - symbol to select configuration profile for v.22 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V22_DISABLE disables V.22 modems. If V.22 is disabled, an originate call will not generate the scrambled binary ones sequence for a V.22 call. The fallback origiate response behavior will depend on the status of BELL103_DISABLE. In answer mode, the system will generate ????? if Bell 103 is enabled, and will not do anything if Bell 103 is disabled.

V22_DISABLED

V22_ENABLE

V22_FAST decreases the training time for V22. TXA_SILENCE1_LEN decrease to 12.5 ms from 2.15 sec, TXC_SILENCE1_LEN decrease to 12.5 ms from 611 ms, TXA_ANSWER_TON_LEN decreases to 12.5 ms from 3.3 sec.

V23_DEMO - symbol to select configuration profile for v.23 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V26_DEMO - symbol to select configuration profile for v.26 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V27_DISABLE

V29_DEBUG set Terminal_count to 0 in Tx_init_v29().

V29_DEMO - symbol to select configuration profile for v.29 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V29_DISABLE

V32_DEMO - symbol to select configuration profile for v.32 modem demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V32_DISABLE disables V.32 modems. If V.32 is disabled, an originate call will not generate the AA sequence for a V.32 call. The origiate fallback response behavior will depend on the status of V22_DISABLE and BELL103_DISABLE. In answer mode, the system will ignore the AA sequence.

V32_FAST decreases the training time for V32. TXA_SILENCE1_LEN decrease to 0 from 1.8 sec, TXC_SILENCE1_LEN decrease to 0 from 300 ms, TXA_ANSWER_TON_LEN decreases to 100ms from 3.3 sec. All Tx traininging lengths are 1280 symbols and the threshold for EC convergence is decreased.

V32_NO_TCM forces the rate to 9600 unless the rate is 4800.

V32_STU_III

V34TCM defines the level of the trellis decoder for the V.34 modem. See V34TCM16, V34TCM32, and V34TCM64.

V34TCM16 defines the 16 state V.34 trellis decoder. The V.34 will only support the 16 state encoder and decoder.

V34TCM32 defines the 32 state V.34 trellis decoder. The V.34 will support 32 and 16 state encoders and decoders.

V34TCM64 defines the 64 state V.34 trellis decoder. The V.34 will support all three encoder and decoders.

V34_16STATE_TCM

V34_SR2400 is defined when there is support for 2400 baud in V.34

V34_SR2743 is defined when there is support for 2743 baud in V.34

V34_SR2800 is defined when there is support for 2800 baud in V.34

V34_SR3200 is defined when there is support for 3200 baud in V.34

V34_32STATE_TCM

V34_SR3429 is defined when there is support for 3429 baud in V.34

V34_64STATE_TCM

V34_COMMON_INTERP is defined to use an alternate interpolation filter in V.34 that replace the standard one in common.c.

V34_DEMO - symbol to select configuration profile for v.34 modem canceller demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V34_DEMO2 - symbol to select configuration profile for v.34, 2 channel modem canceller demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

V34_EQ_LEN is the length of the V.34 equalizer. The default is 63.

V34_FAST decrease the training times for V.34.

V42BIS enables V.42bis data compression. This parameter is negotiated during V.42 negotiation. V.42 must be enabled to run V.42bis. Both sides of the connection must support V.42 and V.42bis in order to run in compressed mode.

V42BIS_COMPACT packs the codeword entries to minimize the size of the table at the expense of speed. This can be enabled if B42_MAX_CODEWORD is less than 2048. For operation where ram is limited, this reduces the table size by B42_MAX_CODEWORD.

V42BIS_DISABLE disables V.42 bis. If V.42 bis is disabled, the V.42 options are disregarded. See V42BIS, V42BIS_COMPACT, B42MAXCODEWORD. Defining also inhibits V.42 bis during V.42 negotiations.

V42BIS_MAX_CODEWORD

V42BIS_MAX_STRING

V42DEBUG

V42LOG is a debugging tools that captures the V.42 Frames and state transitions in a V.42 session.
V42TRACE
V42_DISABLE disables V.42 hdlc framing. There are three data modes supported by the LRDM: V.42, V.14, and no framing.

V42BIS enables V.42bis data compression. This parameter is negotiated during V.42 negotiation. V.42 must be enabled to run V.42bis. Both sides of the connection must support V.42 and V.42bis in order to run in compressed mode.

V42BIS_DISABLE disables V.42 bis. If V.42 bis is disabled, the V.42 options are disregarded. See V42BIS, V42BIS_COMPACT, B42MAXCODEWORD. Defining also inhibits V.42 bis during V.42 negotiations.

V42BIS_COMPACT packs the codeword entries to minimize the size of the table at the expense of speed. This can be enabled if B42_MAX_CODEWORD is less than 2048. For operation where ram is limited, this reduces the table size by B42_MAX_CODEWORD.

V42_MIPS_MEASURE is defined to measure the mips required for V.42.

V8_DEBUG changes the training lengths and silence periods for debugging V.8.

VCOEF_IN_PROGRAM_MEMORY is set when the coefficients are placed in program memory.

VERBOSE_CHARS

VITERBI_TEST_OUTPUT is defined in V.34 to get the raw symbols from the decoder and write them to file to facilitate the V.34 trellis decoder debugging.

VMODEM_DEMO - symbol to select configuration profile for reduced feature v27/ tone canceller demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

VOCODER_ENABLED

VSIM is defined to enable VSIM specific sections of code and screen graphics.

VSIM_DEMO - symbol to select configuration profile for VSIM simulation canceller demonstration in config.h. This profile sets the buffer sizes and code module enables for the default build, which can each be over- ridden externally by a command- line symbol definition or from a *.opt options file.

WAIT_DIALTONE_DISABLE disable waiting for dialtone prior to dialing.

WHAT_NR_ROOTER_SCALE is the scale factor for V.34 NR rooter.

WINDOWS is defined for windows specific graphic code.

XDAIS_API define that Texas Instruments DAIS api.