Freescale Semiconductor
Application Note

AN3379
Rev. 0, 07/2007

Using the CEA709 eTPU Function

by: Michal Princ
System Application Engineer, Roznov Czech System Center

1 Introduction

The CEA709 Enhanced Time Processor Unit (eTPU) function serves as a 78.125 kbit/s ANSI/EIA/CEA 709.1 Medium Access Control (MAC) Layer. This function is intended to be used with the ANSI/EIA/CEA 709.1 protocol software developed by Domologic and targeted at Freescale's 32-bit ColdFire controller MCF5235. Then, the MCF5235 can serve as an ASNI/EIA/CEA 709.1 LonTalk® compatible communication node. This application note provides simple C interface routines to the CEA709 eTPU function. These routines are targeted at the MCF523*x* family of devices, but they could be easily used with any device that has an eTPU.

Contents

2 Function Overview

The CEA709 eTPU function set five functions:

- CEA709 RECEIVE (CEA709_RX) function uses one input channel to decode differential Manchester encoded signals from a LON® (Local Operating Network) transceiver, check the CRC (Cyclic Redundancy Check) at the end of the received packet, and maintain a receive buffer.
- CEA709 TRANSMIT (CEA709_TX) function uses one output channel to generate differential Manchester coded signals (packets), calculate and generate the CRC at the end of the transmitted packet, and maintain two transmit buffers.
- CEA709 DMA CONTROL (CEA709_DMA_CONTROL) function uses one output channel for generating a DMA transfer trigger signal.
- CEA709 TRANSMIT ENABLE (CEA709_TXEN) function uses one output channel for transmit enable signal generation. It is an optional channel.
- CEA709 COLLISION DETECTION (CEA709_CD) function uses one input channel that manages the collision detection signal. It is an optional channel.

3 Function Description

The CEA709 eTPU function, working together with the Domologic protocol stack, serves as an ANSI/EIA/CEA-709 communication node. The CEA709 eTPU function covers layer two of the protocol (see Figure 1), managing the following tasks:

- Receiving physical protocol data unit (PPDU) frames from physical transceiver and their decoding
- Generating and sending PPDU frames to physical transceiver (differential Manchester encoded signal)
- Establishing and maintaining buffers allowing data to be transferred between the CPU and the eTPU/MAC layer
- Generating interrupt requests and DMA requests to manage buffers
- Receiving MAC protocol data unit (MPDU) frames from the CPU and generating PPDU frames for transmission to the transceiver
- Extracting MPDUs from received PPDU data frames and transferring them to the CPU
- Backlog calculation and update
- CRC generation and checking

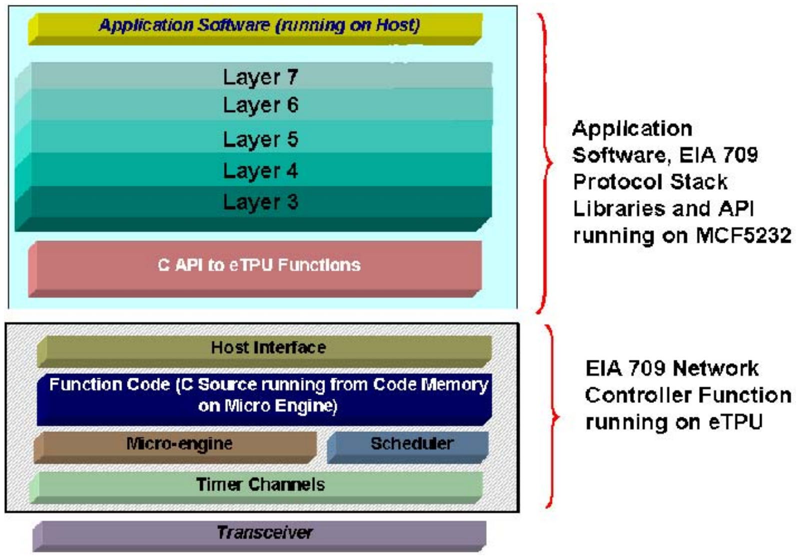
Application Software (running on Host)
Layer 7
Layer 6
Layer 5
Layer 4
Layer 3
C API to eTPU Functions
Application Software, EIA 709 Protocol Stack Libraries and API running on MCF5232
Host Interface
Function Code (C Source running from Code Memory on Micro Engine)
Micro-engine
Scheduler
Timer Channels
EIA 709 Network Controller Function running on eTPU
Transceiver

Figure 1. OSI Reference Model

The CEA709 eTPU function is an equivalent of the Neuron® Chip communications port configured to operate in direct mode. The CEA709 eTPU function encodes transmitted data and decodes received data using Differential Manchester coding (also known as bi-phase space coding). This scheme provides a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. The 0/1 data is indicated by the presence or absence of a second transition halfway between clock transitions. A mid-cell transition indicates a zero. Lack of a mid-cell transition indicates a one. Differential Manchester coding is polarity-insensitive. Therefore, reversal of polarity in the communication link does not affect data reception.

The transmitter transmits a preamble at the beginning of a packet to allow the other nodes to synchronize their receiver clocks. The preamble consists of a series of Differential Manchester ones. Its duration is at least 6 bits long and is selectable. The preamble ends with a single byte-sync bit, which marks the start of byte boundaries on the following bit. The byte-sync bit is a Differential Manchester 0.

The CEA709 eTPU function terminates the packet by forcing a Differential Manchester code violation. After sending the last CRC bit, it holds the data output transitionless for 3 bit periods.

The CEA709 communication bit rate equals 78.125 kbit/s, i.e. one bit period takes 12.8µs. Figure 2 shows a typical packet, where T is the bit period, equal to 1/(bit rate).

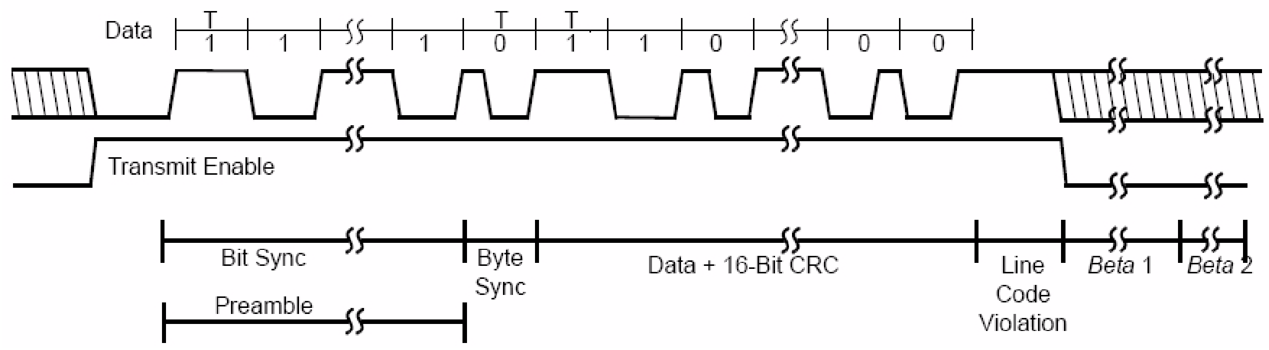

Figure 2. CEA709 Protocol Single-Ended Mode Data Format

3.1 RX Channel

The CEA709 RX channel manages a 16-byte long RX buffer located in the eTPU DATA RAM, see Figure 3. After the RX buffer is full, a DMA transfer is generated to empty the buffer. This transfers newly received data from the eTPU RX buffer to the defined memory place in the CPU. Apart from the DMA request, interrupt requests are generated to the CPU. An interrupt may be requested from the eTPU under the following conditions:

- When a LCV (line code violation = end of frame) has been received to re-initialize the DMA registers (source and destination addresses).
- When an error occurred
- When the backlog value overflows

3.2 TX Channel

The CEA709 TX channel manages two TX buffers. It deals with a tx_priority buffer and a tx_non-priority buffer, each occupying 16 bytes of the eTPU DATA RAM, see Figure 3. When a frame has to be sent, the CPU copies the first 16 bytes of the message to the eTPU shared memory and then initiates a send request (for non-priority or priority messages) using the defined API function. When the communication media has been granted to the eTPU, the eTPU TX channel requests an interrupt to the CPU. The CPU has to re-configure the DMA channel for sending (normally, the DMA channel used is configured for receiving). Because it is clear now if a priority or non-priority message has to be sent, the CPU sets the DMA source pointer accordingly. The eTPU fetches the data from the CPU memory using DMA requests. When sending, an interrupt may be requested from the eTPU under the following conditions:

- When the transmission begins and the CPU has to configure the DMA controller for sending
- When the message has been transmitted completely
- When an error occurred
- When the backlog value overflows
- When a collision is detected during the preamble or at the end of a packet

The CEA709 eTPU function parameter status reflects the actual status of the eTPU based MAC Layer. This parameter is accessible from the eTPU and the CPU. The CPU can read, set, or clear particular bits using the defined API functions. It deals with the following status data:

```
FS_ETPU_CEA709_STATUS_RX_FRAME_END               0x000001 // End of frame = line code violation
FS_ETPU_CEA709_STATUS_RX_ERROR                   0x000002 // Error occurred when receiving
FS_ETPU_CEA709_STATUS_TX_PRI_BUFFER_FULL         0x000004 // TX priority buffer contains
                                                             new data to be transmitted
FS_ETPU_CEA709_STATUS_TX_NONPRI_BUFFER_FULL      0x000008 // TX non-priority buffer contains new data to
                                                             be transmitted
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_BEGINNING  0x000010 // Transmission begins
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_SUCCESSFUL 0x000020 // Data transmitted successfully
FS_ETPU_CEA709_STATUS_TX_ERROR                   0x000040 // Error occurred when transmitting
FS_ETPU_CEA709_STATUS_BACKLOG_OVERFLOW           0x000080 // Backlog overflow
FS_ETPU_CEA709_STATUS_COLLISION_DETECTED         0x000100 // Collision detected during the preamble or at
                                                             the end of a packet
```

3.3 DMA Control Channel

The CEA709 DMA control channel generates the DMA transfer trigger signal. This function is optional and used if the DMA transfers cannot be triggered internally. This can happen when two or more CEA709 instances are running on one eTPU and the DMA module is not able to recognize which of the eTPU channels has asserted the DMA request. To deal with this, the CEA709_DMA_CONTROL channel has to be initialized and its output pin has to be connected with the applicable DREQ pin. The DMA triggering signal polarity is selectable, the trigger edge can be low-high or high-low.

Figure 3 and Figure 4 describes the linkage between the eTPU, the DMA, and the CPU. The operation of one eTPU CEA709 instance is described in Figure 3. Figure 4 shows the operation of two eTPU CEA709 instances.

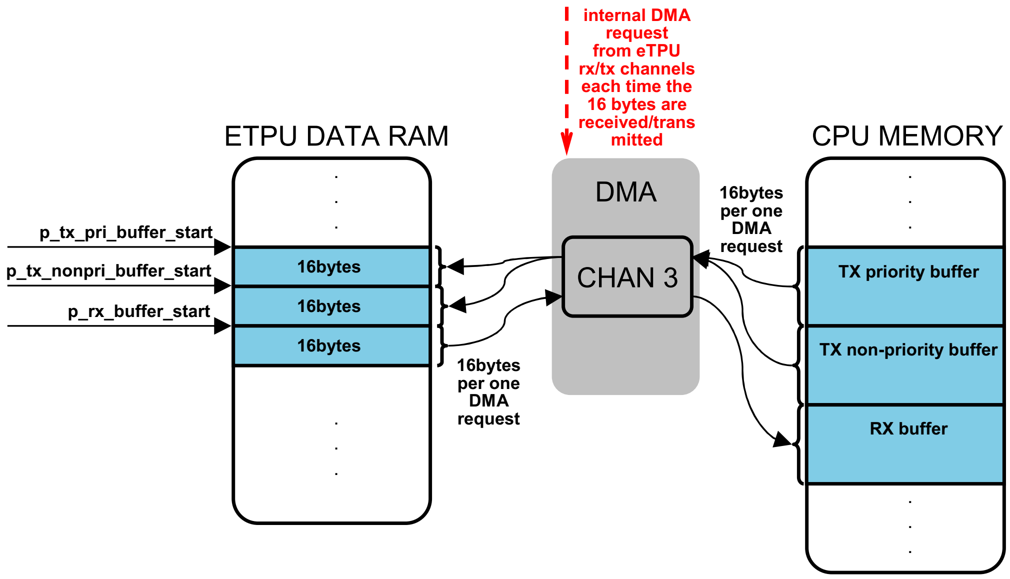

Figure 3. Operation of One eTPU CEA709 Instance – Internal Triggering of the DMA Transfers

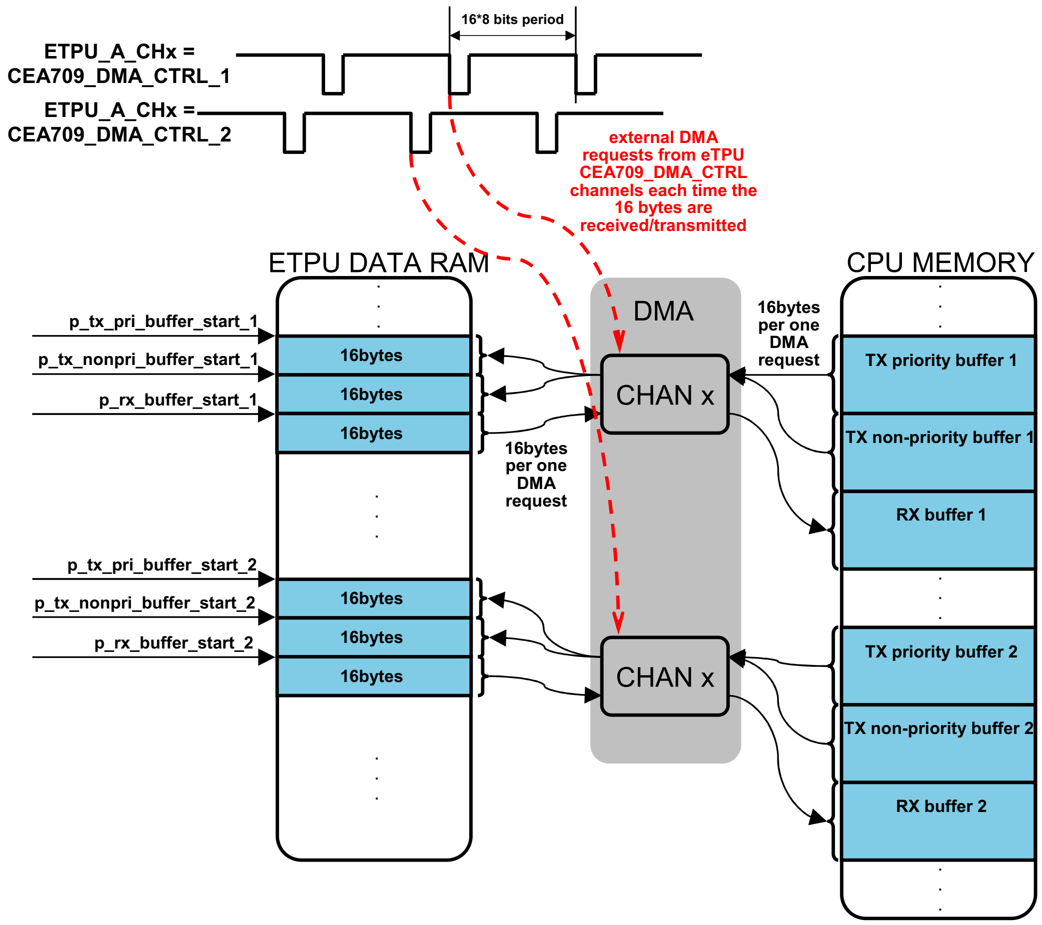

Figure 4. Operation of Two eTPU CEA709 Instances – External Triggering of the DMA Transfers

3.4 TX Enable Channel

The CEA709 TX enable channel is one of the optional channels. The transmit enable signal on the TX enable channel determines the time the TX channel transmits (see Figure 2). Before beginning to transmit the packet, the transmit enable pin is driven high. At the end of the packet after a Differential Manchester code violation, the transmit enable pin is driven low, indicating the end of transmission. Usage of the TX enable channel depends on the type of transceiver.

3.5 Collision Detection Channel

As an option, the CEA709 eTPU MAC unit accepts an active-low Collision Detect (CD) input from the transceiver. If collision detection is enabled and the CD signal goes low for at least one sixty-fourth of a bit period (200 ns) during transmission, the CEA709 eTPU MAC is signalled that a collision has or is occurring. The collision detect flag is checked optionally at the end of the preamble and at the end of the packet. When a collision is detected, the applicable bit in the status parameter is set to one (FS_ETPU_CEA709_STATUS_COLLISION_DETECTED), and the CD eTPU channel generates an interrupt to the CPU, indicating that the message must be sent again.

If the node does not use collision detection, the only way it can determine that a message has not been received is to request an acknowledgment.

3.6 Noise Immunity

When receiving, the CEA709 function uses an acceptance windows, defined in time, within which the next transition on the RX channel is expected. It deals with the receiver jitter tolerance windows. The window opening and closing times are calculated based on the last window end and the given coefficients (jitter1, jitter2). Two windows are set up for each bit period, T. The first window is set at T/2 and determines if a 0 is being received. The second window is at T and defines a 1 being received. If no transition occurs, a Manchester code violation is detected and the packet is assumed to have ended. If a transition falls outside of either window, it is not detected. Timing instability of the transitions, known as jitter, may be caused by changes in the communications medium or instability in the transmitting or receiving node's input clocks.

Apart from receiver jitter tolerance windows, another noise immunity technique is used on the CEA709 RX channel. When the RX transitions are serviced, it is checked to see whether the input pin has changed since the transition time. If the input pin has changed (a very narrow pulse has occurred), the transition is evaluated as a noise transition and the eTPU waits for a valid transition.

3.7 Interrupts

This paragraph summarizes the conditions under which interrupts from the eTPU CEA709 channels are requested:

- RX channel
  - A LCV (line code violation = end of frame) has been received
  - An error occurred
  - The backlog value overflows
- TX channel
  - The transmission begins and the CPU has to configure the DMA controller for sending
  - The message has been transmitted completely
  - An error occurred
  - The backlog value overflows
  - A collision is detected during the preamble or at the end of the packet
- DMA Control channel - does not generate any interrupt
- TX Enable channel - does not generate any interrupt
- Collision Detection channel - does not generate any interrupt

Apart from these interrupts, the applicable DMA channel has to be configured to generate an interrupt request to the CPU each time a successful DMA transfer finishes (see Figure 3). The interrupt handler then re-configures the DMA channel registers (source and destination addresses, BCR) to be ready for the next DMA transfer. This must be done by the time the next DMA transfer is requested. With regards to 78.125 kbit/s communication, the time between two consecutive DMA transfer requests is 16*8/(78.125 * 1024) = 1.6ms.

3.8 DMA requests

As described in previous paragraphs, the DMA requests can be generated internally or externally, depending on the application needs. The CEA709 RX channel generates the DMA transfer itself or through the DMA control channel each time one of eTPU RX channels is full and there is a need to empty the applicable RX buffer. The CEA709 TX channel generates the DMA transfer itself or through the DMA control channel each time the data from the applicable eTPU TX buffer was transmitted and there is a need to fetch new data from the CPU memory. The TX enable channel and the collision detection channel do not generate any DMA requests.

3.9 Performance

Like all the eTPU functions, the CEA709 function performance in an application is to some extent dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the scheduler.

The influence of the CEA709 function on the overall eTPU performance can be expressed by the following parameter:

- Maximum eTPU busy-time per one bit period

  This value, compared to the bit period value (1/bit rate), determines the proportional load on the eTPU engine caused by the CEA709 function.

Table 1 lists the maximum eTPU busy-times per bit period in eTPU cycles that depend on whether the CEA709 function is transmitting or receiving

Table 1. Maximum eTPU Busy-times

Communication States	Maximum eTPU busy-time per one bit period [eTPU cycles]
Receiving of packets	267
Transmitting of packets	141

The eTPU module clock is equal to the peripheral clock, which is half the CPU clock, on the MCF523x devices. For example, the eTPU module clock is 75 MHz on a 150 MHz MCF5235 and one eTPU cycle takes 13.33ns. Considering a 78.125 kbit/s communication rate, the maximum percentage eTPU load is 267*100*78.125/75000 = 27.8% when receiving, and 175*100*78.125/75000 = 14.7% when transmitting.

The performance is influenced by compiler efficiency. The above numbers, measured on code compiled by the eTPU compiler version 1.0.7, are given for guidance only and are subject to change. For up to date information, refer to the information provided in the particular eTPU function set release available from Freescale.

4 C Level Functions API

The following routines provide easy access, for the application developer, to the CEA709 function. Use of these functions eliminates the need to directly control the eTPU registers. There are 17 functions added to the application programming interface (API). The routines can be found in the etpu_CEA709.h and etpu_CEA709.c files, which should be included in the link file along with the top level development file(s). Figure 5 shows the CEA709 API state flow and lists the API functions that can be used in each of its states.

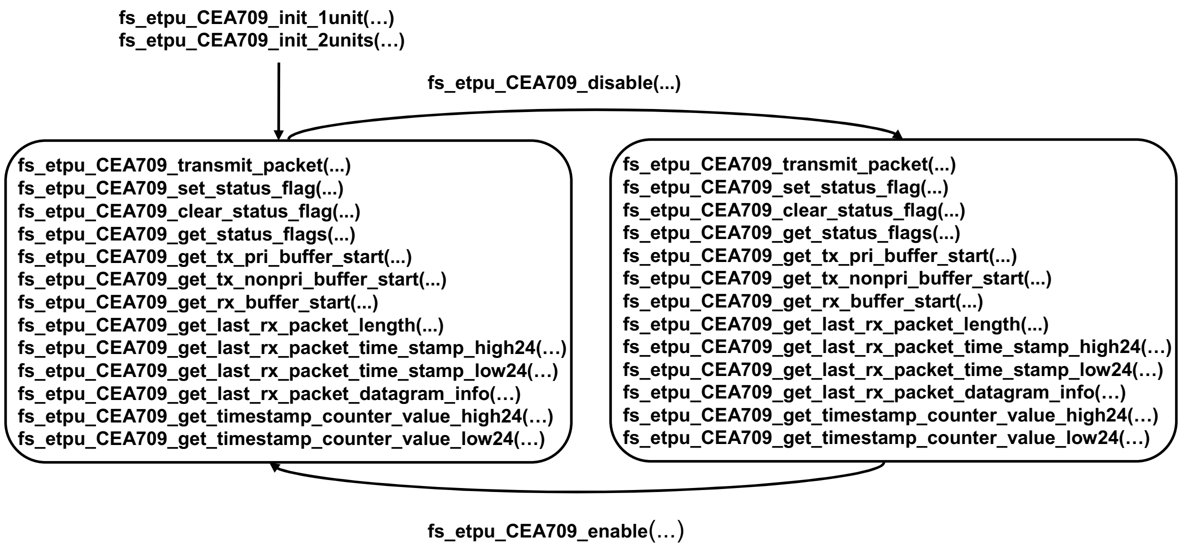
fs_etpu_CEA709_init_1unit(...)
fs_etpu_CEA709_init_2units(...)
fs_etpu_CEA709_disable(...)
fs_etpu_CEA709_transmit_packet(...)
fs_etpu_CEA709_set_status_flag(...)
fs_etpu_CEA709_clear_status_flag(...)
fs_etpu_CEA709_get_status_flags(...)
fs_etpu_CEA709_get_tx_pri_buffer_start(...)
fs_etpu_CEA709_get_tx_nonpri_buffer_start(...)
fs_etpu_CEA709_get_rx_buffer_start(...)
fs_etpu_CEA709_get_last_rx_packet_length(...)
fs_etpu_CEA709_get_last_rx_packet_time_stamp_high24(...)
fs_etpu_CEA709_get_last_rx_packet_time_stamp_low24(...)
fs_etpu_CEA709_get_last_rx_packet_datagram_info(...)
fs_etpu_CEA709_get_timestamp_counter_value_high24(...)
fs_etpu_CEA709_get_timestamp_counter_value_low24(...)
fs_etpu_CEA709_transmit_packet(...)
fs_etpu_CEA709_set_status_flag(...)
fs_etpu_CEA709_clear_status_flag(...)
fs_etpu_CEA709_get_status_flags(...)
fs_etpu_CEA709_get_tx_pri_buffer_start(...)
fs_etpu_CEA709_get_tx_nonpri_buffer_start(...)
fs_etpu_CEA709_get_rx_buffer_start(...)
fs_etpu_CEA709_get_last_rx_packet_length(...)
fs_etpu_CEA709_get_last_rx_packet_time_stamp_high24(...)
fs_etpu_CEA709_get_last_rx_packet_time_stamp_low24(...)
fs_etpu_CEA709_get_last_rx_packet_datagram_info(...)
fs_etpu_CEA709_get_timestamp_counter_value_high24(...)
fs_etpu_CEA709_get_timestamp_counter_value_low24(...)
fs_etpu_CEA709_enable(...)

Figure 5. CEA709 API State Flow

All CEA709 API routines are described in order and listed below:

- Initialization Functions:

```
int32_t fs_etpu_CEA709_init_1unit( uint8_t   rx_channel,
                                   uint8_t   tx_channel,
                                   uint8_t   preamble_length,
                                   uint24_t  packet_cycle,
                                   uint24_t  beta2_control,
                                   uint24_t  xmit_interpacket,
                                   uint24_t  receive_interpacket,
                                   uint8_t   channel_priorities,
                                   uint8_t   node_priority,
                                   uint8_t   bit_sync_threshold,
                                   uint8_t   dma_control_channel,
                                   uint8_t   dma_trigger_mode,
                                   uint8_t   tx_en_channel,
                                   uint8_t   cd_channel,
                                   uint8_t   cd_preamble,
                                   uint8_t   cd_tail,
                                   uint24_t  cd_to_end_packet)
int32_t fs_etpu_CEA709_init_2units( uint8_t   rx_channel_1,
                                    uint8_t   tx_channel_1,
                                    uint8_t   preamble_length_1,
                                    uint24_t  packet_cycle_1,
                                    uint24_t  beta2_control_1,
                                    uint24_t  xmit_interpacket_1,
                                    uint24_t  receive_interpacket_1,
                                    uint8_t   channel_priorities_1,
                                    uint8_t   node_priority_1,
                                    uint8_t   bit_sync_threshold_1,
                                    uint8_t   dma_control_channel_1,
                                    uint8_t   dma_trigger_mode_1,
                                    uint8_t   tx_en_channel_1,
                                    uint8_t   cd_channel_1,
                                    uint8_t   cd_preamble_1,
                                    uint8_t   cd_tail_1,
                                    uint24_t  cd_to_end_packet_1,
                                    uint8_t   rx_channel_2,
                                    uint8_t   tx_channel_2,
                                    uint8_t   preamble_length_2,
                                    uint24_t  packet_cycle_2,
                                    uint24_t  beta2_control_2,
                                    uint24_t  xmit_interpacket_2,
                                    uint24_t  receive_interpacket_2,
                                    uint8_t   channel_priorities_2,
                                    uint8_t   node_priority_2,
                                    uint8_t   bit_sync_threshold_2,
                                    uint8_t   dma_control_channel_2,
                                    uint8_t   dma_trigger_mode_2,
                                    uint8_t   tx_en_channel_2,
                                    uint8_t   cd_channel_2,
                                    uint8_t   cd_preamble_2,
                                    uint8_t   cd_tail_2,
                                        uint24_t  cd_to_end_packet_2)
```

- Change Operation Functions:

```
uint32_t fs_etpu_CEA709_enable(uint8_t   rx_channel,
                                uint8_t   tx_channel,
                                uint8_t   dma_control_channel,
                                uint8_t   tx_en_channel,
                                uint8_t   cd_channel)
uint32_t fs_etpu_CEA709_disable(uint8_t   rx_channel,
                                 uint8_t   tx_channel,
                                 uint8_t   dma_control_channel,
                                 uint8_t   tx_en_channel,
                                 uint8_t   cd_channel)
int32_t  fs_etpu_CEA709_transmit_packet(uint8_t rx_channel)
uint32_t fs_etpu_CEA709_set_status_flag(uint8_t rx_channel, uint32_t mask)
uint32_t fs_etpu_CEA709_clear_status_flag(uint8_t rx_channel, uint32_t mask)
```

- Value Return Functions:

```
uint32_t fs_etpu_CEA709_get_tx_pri_buffer_start(uint8_t rx_channel)
uint32_t fs_etpu_CEA709_get_tx_nonpri_buffer_start(uint8_t rx_channel)
uint32_t fs_etpu_CEA709_get_rx_buffer_start(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_status_flags(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_last_rx_packet_length(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_last_rx_packet_time_stamp_high24(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_last_rx_packet_time_stamp_low24(uint8_t rx_channel)
uint8_t fs_etpu_CEA709_get_last_rx_packet_datagram_info(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_timestamp_counter_value_high24(uint8_t rx_channel)
uint24_t fs_etpu_CEA709_get_timestamp_counter_value_low24(uint8_t rx_channel)
```

4.1 Initialization Function

4.1.1 Initialize One MAC Unit

The int32_t fs_etpu_CEA709_init_1unit routine initializes one CEA709 MAC unit. This function has the following parameters:

- rx_channel (uint8_t) – RX channel number (CP0)
- tx_channel (uint8_t) – TX channel number (CP1)
- preamble_length (uint8_t) – length of preamble for direct mode (in number of bits)
- packet_cycle (uint24_t) – packet cycle duration (in number of bits)
- beta2_control (UINT24_T) – beta2 slots width (in number of bits)
- xmit_interpacket (uint24_t) – interpacket padding after transmitting (in number of bits)
- receive_interpacket (uint24_t) – interpacket padding after receiving (in number of bits)
- channel_priorities (uint8_t) – number of priority slots on the channel
- node_priority (uint8_t) – priority slot used by the node when sending priority messages on the channel (1 – 255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is 0.
- bit_sync_threshold (uint8_t) – number of sync bits
- dma_control_channel (uint8_t) – DMA control channel number necessary for external DMA request generation
- dma_trigger_mode (uint8_t) – this parameter defines the polarity on DMA CONTROL channel and it should be assigned a value of:
  FS_ETPU_CEA709_DMA_TRIGGER_RISING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_FALLING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_INTERNAL.
- tx_en_channel (uint8_t) – transmit enable channel number (CP2,if required); If not applicable then set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_channel (uint8_t) – collision detection channel number (CP4,if required); If not applicable then set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_preamble (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet preamble; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDPREAMBLE_NO or
  FS_ETPU_CEA709_CDPREAMBLE_YES
- cd_tail (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDTAIL_NO or
  FS_ETPU_CEA709_CDTAIL_YES
- cd_to_end_packet (uint24_t) – how close to the end of the packet the CD signal is checked (in number of bits)

4.1.2 Initialize Two MAC Units

The int32_t fs_etpu_CEA709_init_2units routine initializes two CEA709 MAC units. This function has the following parameters:

- rx_channel_1 (uint8_t) – RX channel number (CP0) of unit 1
- tx_channel_1 (uint8_t) – TX channel number (CP1) of unit 1
- preamble_length_1 (uint8_t) – length of preamble for direct mode (in number of bits) of unit 1
- packet_cycle_1 (uint24_t) – packet cycle duration (in number of bits) of unit 1
- beta2_control_1 (uint24_t) – beta2 slots width (in number of bits) of unit 1
- xmit_interpacket_1 (uint24_t) – interpacket padding after transmitting (in number of bits) on unit 1
- receive_interpacket_1 (uint24_t) – interpacket padding after receiving (in number of bits) on unit 1
- channel_priorities_1 (uint8_t) – number of priority slots on the channel of unit 1
- node_priority_1 (uint8_t) – unit 1 priority slot used by the node when sending priority messages on the channel (1 255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is 0
- bit_sync_threshold_1 (uint8_t) – number of sync bits of unit 1
- dma_control_channel_1 (uint8_t) – DMA control channel number necessary for external DMA request generation for unit 1
- dma_trigger_mode_1 (uint8_t) – This parameter defines the polarity on the DMA CONTROL channel of unit 1. This parameter should be assigned a value of:
  FS_ETPU_CEA709_DMA_TRIGGER_RISING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_FALLING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_INTERNAL.
- tx_en_channel_1 (uint8_t) – transmit enable channel number (CP2,if required) of unit 1; If not applicable then set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_channel_1 (uint8_t) – collision detection channel number (CP4,if required) of unit 1; If not applicable then set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_preamble_1 (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet preamble; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDPREAMBLE_NO or
  FS_ETPU_CEA709_CDPREAMBLE_YES
- cd_tail_1 (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDTAIL_NO or
  FS_ETPU_CEA709_CDTAIL_YES
- cd_to_end_packet_1 (uint24_t) – how close to the end of the packet the CD signal is checked (in number of bits)
- rx_channel_2 (uint8_t) – RX channel number (CP0) of unit 2
- tx_channel_2 (uint8_t) – TX channel number (CP1) of unit 2
- preamble_length_2 (uint8_t) – length of preamble for direct mode (in number of bits) of unit 2
- packet_cycle_2 (uint24_t) – packet cycle duration (in number of bits) of unit 2

- beta2_control_2 (uint24_t) – beta2 slots width (in number of bits) of unit 2
- xmit_interpacket_2 (uint24_t) – interpacket padding after transmitting (in number of bits) on unit 2
- receive_interpacket_2 (uint24_t) – interpacket padding after receiving (in number of bits) on unit 2
- channel_priorities_2 (uint8_t) – number of priority slots on the channel of unit 2
- node_priority_2 (uint8_t) – unit 2 priority slot used by the node when sending priority messages on the channel (1 255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is 0.
- bit_sync_threshold_2 (uint8_t) – number of sync bits of unit 2
- dma_control_channel_2 (uint8_t) – DMA control channel number necessary for external DMA request generation for unit 2
- dma_trigger_mode_2 (uint8_t) – This parameter defines the polarity on the DMA CONTROL channel of unit 2. This parameter should be assigned a value of:
  FS_ETPU_CEA709_DMA_TRIGGER_RISING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_FALLING_EDGE or
  FS_ETPU_CEA709_DMA_TRIGGER_INTERNAL.
- tx_en_channel_2 (uint8_t) – transmit enable channel number (CP2, if required) of unit 2; If not applicable, set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_channel_2 (uint8_t) – collision detection channel number (CP4, if required) of unit 2; If not applicable, set to FS_ETPU_CEA709_CHAN_NOT_USED.
- cd_preamble_2 (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet preamble; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDPREAMBLE_NO or
  FS_ETPU_CEA709_CDPREAMBLE_YES
- cd_tail_2 (uint8_t) – this parameter determines whether the CD signal is checked at the end of the packet; this parameter should be assigned a value of:
  FS_ETPU_CEA709_CDTAIL_NO or
  FS_ETPU_CEA709_CDTAIL_YES
- cd_to_end_packet_2 (uint24_t) – how close to the end of the packet the cd signal is checked (in number of bits)

4.2 Change Operation Functions

The uint32_t fs_etpu_CEA709_enable function enables the CEA709 eTPU channels. This function has the following parameters:

- rx_channel (uint8_t) – RX channel number (CP0)
- tx_channel (uint8_t) – TX channel number (CP1)
- dma_control_channel (uint8_t) – DMA control channel number necessary for external DMA request generation
- tx_en_channel (uint8_t) – transmit enable channel number (CP2, if required)
- cd_channel (uint8_t) – collision detection channel number (CP4, if required)

The uint32_t fs_etpu_CEA709_disable function disables the CEA709 eTPU channels. This function has the following parameters:

- rx_channel (uint8_t) – RX channel number (CP0)
- tx_channel (uint8_t) – TX channel number (CP1)
- dma_control_channel (uint8_t) – DMA control channel number necessary for external DMA request generation
- tx_en_channel (uint8_t) – transmit enable channel number (CP2, if required)
- cd_channel (uint8_t) – collision detection channel number (CP4, if required)

The CPU requests transmission of the next packet by calling the int32_t fs_etpu_CEA709_transmit_packet. Each time the CPU has new data to transmit, it writes the data to the applicable TX buffer, sets the applicable status flags, and calls this function. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The uint32_t fs_etpu_CEA709_set_status_flag function sets the appropriate status flag. This function has the following parameters:

- rx_channel (uint8_t) – RX channel number (CP0)
- mask (uint32_t) – determines which status flag should be set. This parameter should be assigned a value of:

```
FS_ETPU_CEA709_STATUS_RX_FRAME_END             0x000001 // End of frame equals line code
                                                           violation
FS_ETPU_CEA709_STATUS_RX_ERROR                 0x000002 // Error occurred when receiving
FS_ETPU_CEA709_STATUS_TX_PRI_BUFFER_FULL       0x000004 // TX priority buffer contains new
                                                           data to be transmitted
FS_ETPU_CEA709_STATUS_TX_NONPRI_BUFFER_FULL    0x000008 // TX non-priority buffer contains new
                                                           data to be transmitted
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_BEGINNING  0x000010 // Transmission begins
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_SUCCESSFUL 0x000020 // Data transmitted successfully
FS_ETPU_CEA709_STATUS_TX_ERROR                 0x000040 // Error occurred when transmitting
FS_ETPU_CEA709_STATUS_BACKLOG_OVERFLOW         0x000080 // Backlog overflow
FS_ETPU_CEA709_STATUS_COLLISION_DETECTED       0x000100 // Collision detected during the
                                                          preamble or at the end of the packet
```

The uint32_t fs_etpu_CEA709_clear_status_flag function enables the clearing of the appropriate status flag. This function has the following parameters:

- rx_channel (uint8_t) – RX channel number (CP0)
- mask (uint32_t) – determines which status flag should be cleared. This parameter should be assigned a value of:

```
FS_ETPU_CEA709_STATUS_RX_FRAME_END              0x000001 // End of frame equals line code
                                                           violation
FS_ETPU_CEA709_STATUS_RX_ERROR                  0x000002 // Error occurred when receiving
FS_ETPU_CEA709_STATUS_TX_PRI_BUFFER_FULL        0x000004 // TX priority buffer contains new
                                                           data to be transmitted
FS_ETPU_CEA709_STATUS_TX_NONPRI_BUFFER_FULL     0x000008 // TX non-priority buffer contains new
                                                           data to be transmitted
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_BEGINNING 0x000010 // Transmission begins
FS_ETPU_CEA709_STATUS_TX_TRANSMISSION_SUCCESSFUL 0x000020 // Data transmitted successfully
FS_ETPU_CEA709_STATUS_TX_ERROR                  0x000040 // Error occurred when transmitting
FS_ETPU_CEA709_STATUS_BACKLOG_OVERFLOW          0x000080 // Backlog overflow
FS_ETPU_CEA709_STATUS_COLLISION_DETECTED        0x000100 // Collision detected during the
                                                preamble or at the end of the packet
```

4.3 Value Return Functions

The uint32_t fs_etpu_CEA709_get_tx_pri_buffer_start function gets the pointer to the beginning of the transmit priority buffer. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The pointer to the beginning of the transmit priority buffer is returned as an uint32_t.

The uint32_t fs_etpu_CEA709_get_tx_nonpri_buffer_start function gets the pointer to the beginning of the transmit non-priority buffer. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The pointer to the beginning of the transmit non-priority buffer is returned as an uint32_t.

This uint32_t fs_etpu_CEA709_get_rx_buffer_start function gets the pointer to the beginning of the receive buffer. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The pointer to the beginning of the receive buffer is returned as an uint32_t.

This uint24_t fs_etpu_CEA709_get_status_flags function gets the actual MAC/Link layer status flags for the callback function. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The actual MAC/Link layer status flags are returned as an uint24_t.

This uint24_t fs_etpu_CEA709_get_last_rx_packet_length function gets the length of the last received packet. This function has the following parameter:

- RX_CHANNEL (UINT8_T) – RX channel number (CP0)

The length of the last received packet is returned as an uint24_t.

This uint24_t fs_etpu_CEA709_get_last_rx_packet_time_stamp_high24 function gets the upper 24 bits of the last received packet timestamp. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The length of the last received packet is returned as an uint24_t.

This uint24_t fs_etpu_CEA709_get_last_rx_packet_time_stamp_low24 function gets the lower 24 bits of the last received packet timestamp. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The length of the last received packet is returned as an uint24_t.

This uint8_t fs_etpu_CEA709_get_last_rx_packet_datagram_info function gets the additional datagram info of the last received packet. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The length of the last received packet is returned as an uint8_t.

This uint24_t fs_etpu_CEA709_get_timestamp_counter_value_high24 function gets the upper 24 bits of the timestamp counter value. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The length of the last received packet is returned as an uint24_t.

This uint24_t fs_etpu_CEA709_get_timestamp_counter_value_low24 function gets the lower 24 bits of the timestamp counter value. This function has the following parameter:

- rx_channel (uint8_t) – RX channel number (CP0)

The length of the last received packet is returned as an uint24_t.

5 Use of Function Example

5.1 Running One CEA709 Instance

The following pieces of code serve as examples of using the CEA709 eTPU function API. The CEA709_DMA_CONTROL channel is initialized but the DMA triggering is performed internally. When using Echelon FTT-10A free topology twisted pair transceiver, the CEA709_TXEN and CEA709_CD channels do not have to be initialized (FTT-10A does not support these functions). The initialization of the CEA709 eTPU function should be as follows:

```
/* initialize one CEA709 function */
err_code = fs_etpu_CEA709_init_1unit(
  CEA709_RX_CHAN, /* eTPU channel 0 */
  CEA709_TX_CHAN, /* eTPU channel 1*/
  23, /* preamble_length */
  322, /* packet_cycle */
  13, /* beta2_control */
  70, /* xmit_interpacket */
  72, /* receive_interpacket */
  2,  /* channel_priorities */
  2,  /* node_priority */
  4,  /* bit_sync_threshold */
  CEA709_DMA_CONTROL_CHAN, /* dma_control_channel */
  FS_ETPU_CEA709_DMA_TRIGGER_INTERNAL, /* dma_trigger_mode */
  FS_ETPU_CEA709_CHAN_NOT_USED, /* tx_en_channel */
  FS_ETPU_CEA709_CHAN_NOT_USED, /* cd_channel */
  FS_ETPU_CEA709_CDPREAMBLE_NO, /* cd_preamble */
  FS_ETPU_CEA709_CDTAIL_NO, /* cd_tail */
  0); /* cd_to_end_packet */
```

It is necessary to enable packet reception/transmission by calling the fs_etpu_CEA709_enable() function after the CEA709 function initialization. When the CEA709 channels have been disabled, use the same function to restart.

```
/* enable CEA709 channels */
fs_etpu_CEA709_enable( CEA709_RX_CHAN,
                        CEA709_TX_CHAN,
                        CEA709_DMA_CONTROL_CHAN,
                        FS_ETPU_CEA709_CHAN_NOT_USED,
                        FS_ETPU_CEA709_CHAN_NOT_USED);
```

To obtain addresses of the eTPU RX and TX buffers, call the following functions. This is essential for further DMA channel configuration settings.

```
/* read addresses of the eTPU RX/TX buffers */
CEA709_rx_buffer_start_addr =
  fs_etpu_CEA709_get_rx_buffer_start(CEA709_RX_CHAN);
CEA709_tx_pri_buffer_start_addr =
  fs_etpu_CEA709_get_tx_pri_buffer_start(CEA709_RX_CHAN);
CEA709_tx_nonpri_buffer_start_addr =
  fs_etpu_CEA709_get_tx_nonpri_buffer_start(CEA709_RX_CHAN);
```

To transmit a new packet through the eTPU CEA709 function, the following actions have to be done. Fill the CPU priority/non-priority buffer by the MPDU and copy the first 16 bytes from the CPU priority/non-priority buffer to the applicable eTPU TX buffers. Then set the applicable status flags and call the fs_etpu_CEA709_transmit_packet() function as follows:

```
/* Set flags indicating that the priority and nonpriority TX eTPU buffers are full */
fs_etpu_CEA709_set_status_flag( CEA709_RX_CHAN,
                                 FS_ETPU_CEA709_STATUS_TX_NONPRI_BUFFER_FULL);
fs_etpu_CEA709_set_status_flag( CEA709_RX_CHAN,
                                 FS_ETPU_CEA709_STATUS_TX_PRI_BUFFER_FULL);

/* Trigger the next packet transmission */
hsrr = 1;
while(hsrr!=0)
{
   hsrr = fs_etpu_CEA709_transmit_packet(CEA709_RX_CHAN);
}
```

5.2 Running Two CEA709 Instances

The following piece of code demonstrates how to configure the eTPU for two CEA709 instances. As the DMA module on MCF523x is not able to recognize which of the eTPU channels asserted the DMA request, it is necessary to trigger the DMA transfer using an external DMA request. The CEA709_DMA_CONTROL channel has to be initialized and its output pin has to be connected with the applicable DREQ pin. The initialization of the CEA709 eTPU functions should be as follows:

```
/* initialize the first CEA709 instance */
err_code = fs_etpu_CEA709_init_2units(
  CEA7091_RX_CHAN, /* eTPU channel 0 */
  CEA7091_TX_CHAN, /* eTPU channel 1 */
  23, /* preamble_length */
  322, /* packet_cycle */
  13, /* beta2_control */
  70, /* xmit_interpacket */
  72, /* receive_interpacket */
  2,  /* channel_priorities */
  2,  /* node_priority */
  4,  /* bit_sync_threshold */
  CEA7091_DMA_CONTROL_CHAN, /* dma_control_channel (2) */
  FS_ETPU_CEA709_DMA_TRIGGER_FALLING_EDGE, /* dma_trigger_mode */
  CEA7091_TXEN_CHAN, /* tx_en_channel (3) */
  CEA7091_CD_CHAN, /* cd_channel (4) */
  FS_ETPU_CEA709_CDPREAMBLE_YES, /* cd_preamble */
  FS_ETPU_CEA709_CDTAIL_YES, /* cd_tail */
  10, /* cd_to_end_packet */
  CEA7092_RX_CHAN, /* eTPU channel 5 */
  CEA7092_TX_CHAN, /* eTPU channel 6 */
  23, /* preamble_length */
  322, /* packet_cycle */
  13, /* beta2_control */
  70, /* xmit_interpacket */
  72, /* receive_interpacket */
  2,  /* channel_priorities */
  2,  /* node_priority */
  4,  /* bit_sync_threshold */
  CEA7092_DMA_CONTROL_CHAN, /* dma_control_channel (7) */
  FS_ETPU_CEA709_DMA_TRIGGER_FALLING_EDGE, /* dma_trigger_mode */
  CEA7092_TXEN_CHAN, /* tx_en_channel (8) */
  CEA7092_CD_CHAN, /* cd_channel (9) */
  FS_ETPU_CEA709_CDPREAMBLE_YES, /* cd_preamble */
  FS_ETPU_CEA709_CDTAIL_YES, /* cd_tail */
  10); /* cd_to_end_packet */
```

Determine which CEA709 instance the called API function refers to each time the API functions are used. This is done by assigning to the rx_channel parameter of the API functions the applicable CEA709 instance RX channel number.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3379
Rev. 0
07/2007