OmniXtend Specification

November 12th 2019

OmniXtend Specification

Copyright Notice

Release Information

Revision	Date	Notes
1.0.0-draft	August 12th, 2019	Initial draft version.
1.0.1-draft	September 16th, 2019	First pre-release version.
1.0.2-draft	October 4th, 2019	Second pre-release version.
1.0.3-draft	November 12th, 2019	Clarification of retransmit initialization and duplicate packet processing. *MaxStartOfMessageFlit* definition.

Table of Contents

List of Figures

List of Tables

1 Introduction

TileLink is an interconnect standard providing multiple masters with coherent memory-mapped access to a collection of memories and slave devices. TileLink has primarily been used to connect tightly coupled components on a single chip. The TileLink over Ethernet protocol (TLoE), a.k.a OmniXtend, defines a serialization of the coherent on-chip TileLink bus suitable for transport over Ethernet links.

TLoE defines the packet format used to carry TileLink messages inside Ethernet frames. Since Ethernet is not a lossless protocol, TLoE also defines retransmission and end-to-end flow control. The retransmission protocol supports automatic retransmission of lost Ethernet frames. The flow control protocol extends, end to end, the valid/ready handshake of each TileLink channel.

A TLoE adapter consists of a TLoE transmitter and a TLoE receiver. The TLoE transmitter maps TileLink messages (originated by a local master module on channels A/C/E, or by a local slave module on channels B/D) into Ethernet frames. The TLoE receiver demaps TileLink messages (originated by a remote master on channels A/C/E, or by a remote slave on channels B/D) from the received Ethernet frames.

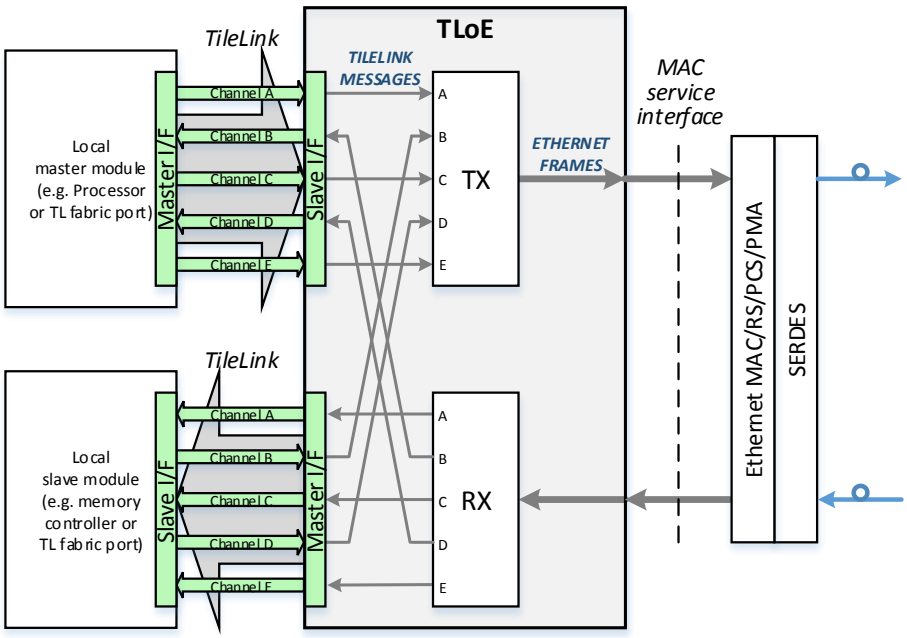

Figure 1 – TLoE adapter

The TLoE protocol is strictly point-to-point and serves to carry one TileLink connection, per direction, between a pair of TLoE endpoints. In other words, the protocol supports symmetric operation, so each TLoE adapter provides both a TileLink Slave interface (allowing local master modules to access remote slave modules), and a TileLink Master interface (allowing remote master modules to access local slave modules), as illustrated in Figure 2.

The TLoE protocol also supports asymmetric operation, as illustrated in Figure 3.

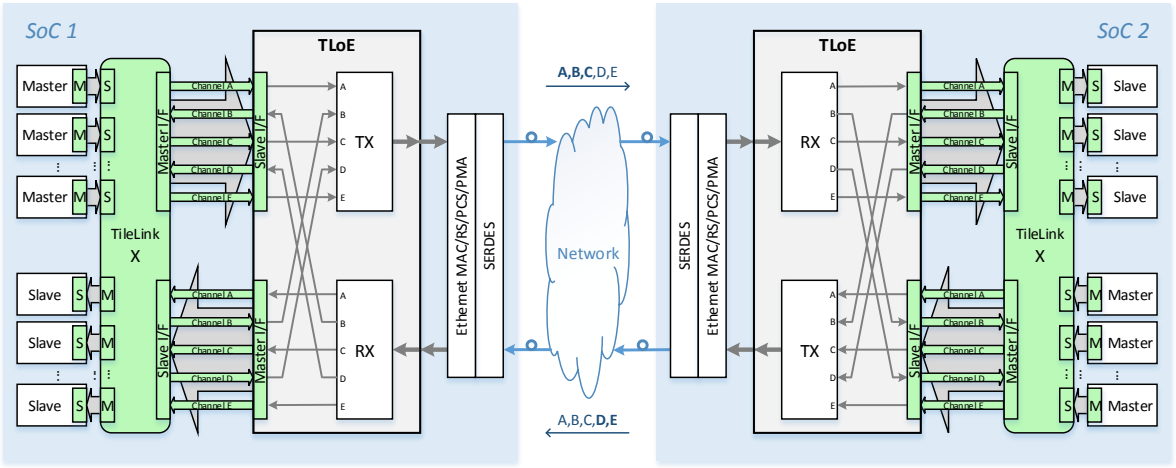

Figure 2 – Symmetric operation

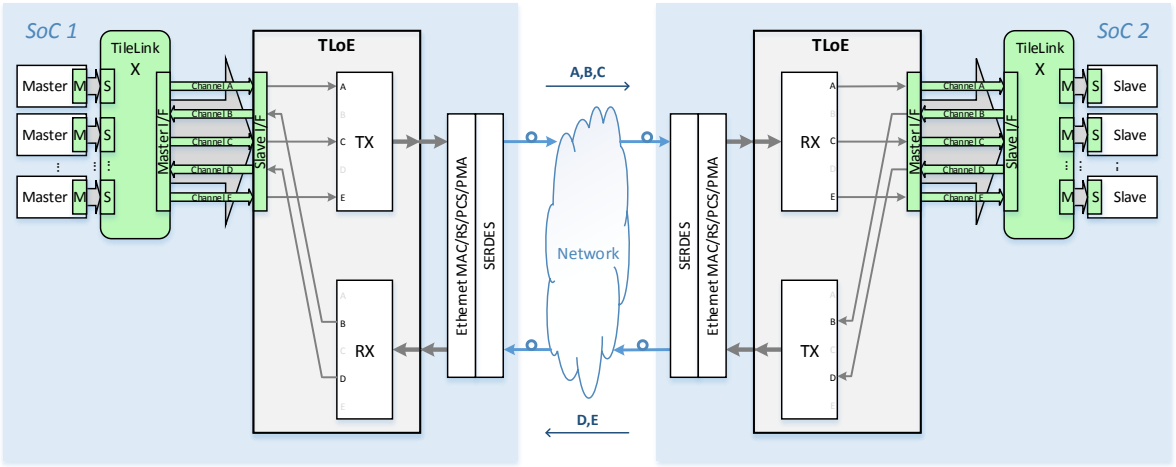

Figure 3 – Asymmetric operation

The TLoE adapter may share access to the Ethernet port with other TLoE adapters and with coexisting Ethernet traffic (e.g. TCP/IP). This requires the use of a multiplexing function, as shown in Figure 4; in this specific example, the non-TLoE traffic (e.g. TCP/IP) is terminated at a master/slave attached to the same TileLink fabric as the master(s)/slave(s) terminating TLoE traffic, but TCP/IP traffic could come from any other arbitrary place. The specific policies and methods used to share the Ethernet port bandwidth between the TLoE traffic and coexisting Ethernet traffic are out of the scope of the TLoE protocol specification.

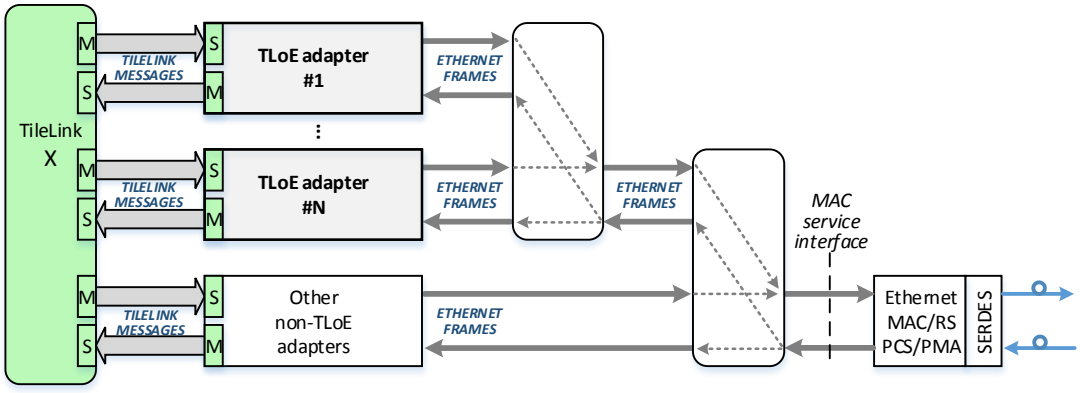

Figure 4 – Coexisting Ethernet traffic on the same Ethernet link

2 Use cases

Figure 5 shows the use of the TLoE protocol for the direct connection to a partner SoC. This is the simplest use case, with a single instance of the TLoE protocol over a dedicated Ethernet port. Each SoC could connect to multiple partner SoCs by using multiple instances of the TLoE protocol, each associated to a separate Ethernet port.

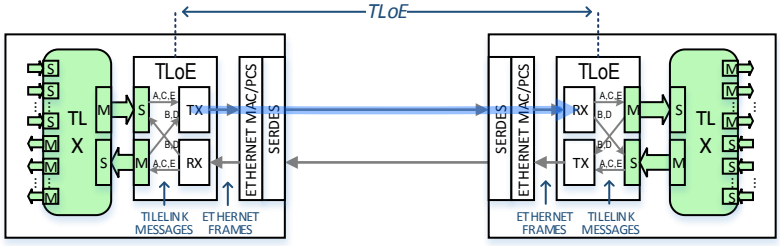

Figure 5 – Direct connection

Figure 6 shows the use of the TLoE protocol to interconnect multiple SoCs through a smart switch (e.g. a P4-programmable switch). The smart switch can be configured to terminate the TLoE protocol, demap the TileLink messages from each Ethernet Ingres port, switch the TileLink messages, and remap them into each Egress Ethernet port. The scope of the TLoE protocol is the Ethernet link. The interconnected SoCs implement a single instance of the TLoE protocol, mapping TileLink messages associated to different TileLink endpoints into the same Ethernet frame. The smart switching fabric can actively participate in the coherence protocol.

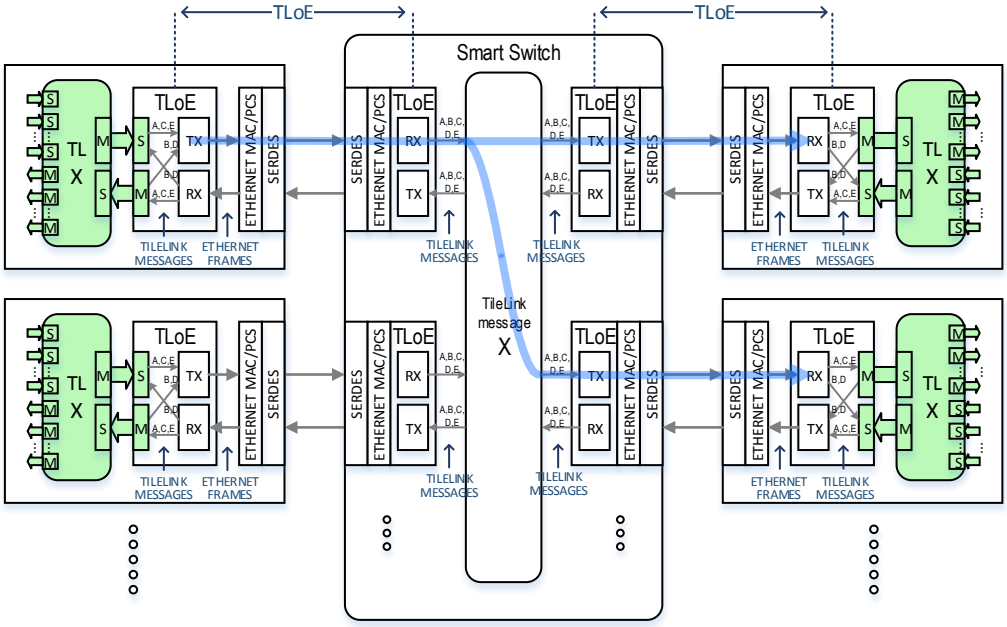

Figure 6 – Connection through a TLoE-aware switch

Figure 7 shows the use of the TLoE protocol to interconnect multiple SoCs through a non-programmable Ethernet switch. The Ethernet switch does not terminate the TLoE protocol, it relays the Ethernet frames from one TLoE endpoint to another, so the TLoE protocol is carried transparently end to end. An Ethernet frame carries TileLink messages that are associated to the same TLoE endpoint. Each interconnected SoC can implement multiple logical instances of the TLoE

protocol, one logical instance per connection (i.e. per partner SoC). The Ethernet switch cannot actively participate in the coherence protocol.

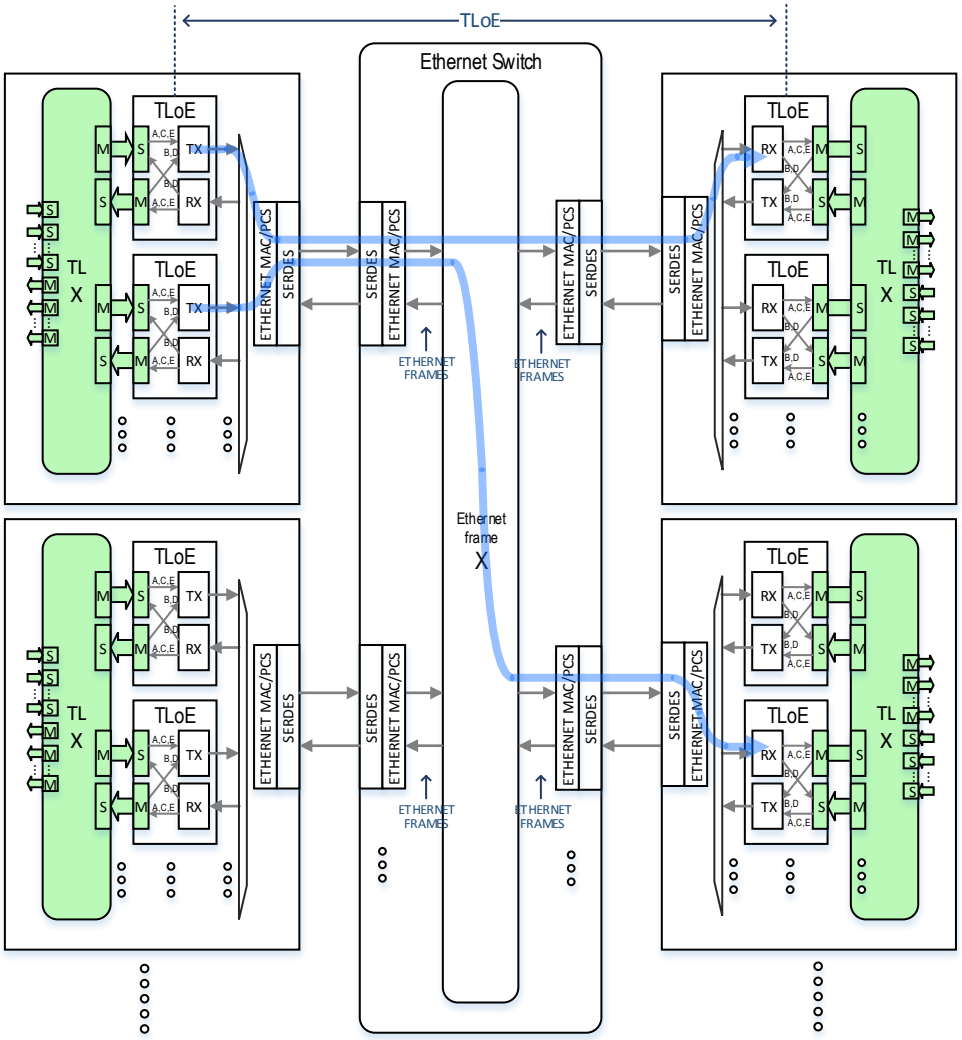

Figure 7 – Connection through a TLoE-unaware switch

3 TLoE frame encoding

The TLoE frame consists of the following areas:

- The Ethernet MAC header
- The TLoE frame header, providing support for retransmission and flow control.
- *m* TileLink messages (*m* = 0, 1, 2, …, 64).
- Optional padding before or after each TileLink message.
- A 64 bit mask indicating the starting location of each of the m messages.
- The Ethernet FCS.

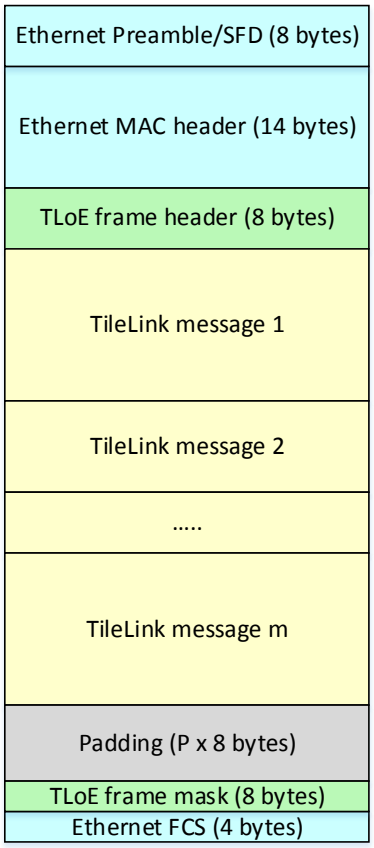

Figure 8 – TLoE frame structure

The fields comprising the MAC header (8-byte Preamble/SFD, 6-byte MAC Destination Address, 6-byte MAC Source Address, and 2-byte EtherType field) are not defined in this specification, except that the value in the EtherType field shall be TBD. The MAC Destination and Source address fields are the respective MAC addresses of the destination and source Ethernet ports.

VLAN tagging is permitted, but not required by TLoE. Use of VLAN tagging is out of the scope of the TLoE specification. The FCS field is shown for completeness.

The FCS field is not generated by the TLoE adapter, it is generated by the Ethernet MAC.

The different TLoE fields (TLoE header, TileLink messages, padding and TLoE frame mask) are encoded as a sequence of 64-bit double-words. The 64-bit encoding is intended to simplify hardware design.

3.1 TLoE frame header

Figure 9 shows the encoding of the TLoE frame header.

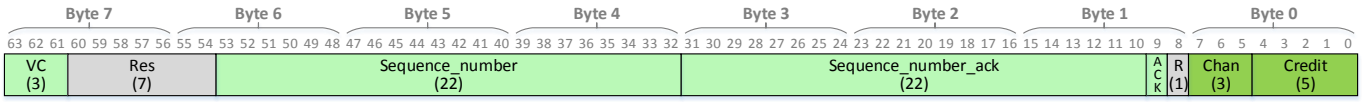

Figure 9 – TLoE frame header encoding

The *Virtual Channel* (VC) field allows the use of multiple instances of the TLoE protocol between two TLoE endpoints.

The *Sequence_number*/*Sequence_number_ack*/*Ack* fields are used for retransmission, and the *Credit*/*Chan* fields are used for flow control.

The *Sequence_number* field carries an incrementing sequence number and is used by the TLoE receiver to detect packets that are received out of order.

The *Sequence_number_ack* field is used by TLoE receiver to indicate (back to the TLoE transmitter) the sequence number of the last packet that was received in correct sequence order.

The *Ack* field is used by the TLoE receiver to indicate (back to the TLoE transmitter) a positive acknowledge (*Ack*=1) or a negative acknowledge (*Ack*=0):

- *Ack*=1 (ACK) indicates that the receiver has received packets in correct sequence order up to *Sequence_number_ack*.
- *Ack*=0 (NAK) indicates that the receiver has received packets with incorrect sequence order, so the transmitter needs to retransmit packets starting with sequence number *Sequence_number_ack* + 1.

The *Credit* field indicates the number of credits that are returned from the receiver back to the transmitter. The *Credit* field is encoded exponentially:

- *Credit*=0 indicates 2^0 credits
- *Credit*=1 indicates 2^1 credits
- …
- *Credit*=31 indicates 2^{31} credits

The *Channel* field identifies the specific channel returning the credits:

- *Channel*=0 indicates that the *Credit* field carried in this frame must be ignored.
- *Channel*=1 indicates that the *Credit* field is associated to TileLink channel A
- *Channel*=2 indicates that the *Credit* field is associated to TileLink channel B
- *Channel*=3 indicates that the *Credit* field is associated to TileLink channel C
- *Channel*=4 indicates that the *Credit* field is associated to TileLink channel D
- *Channel*=5 indicates that the *Credit* field is associated to TileLink channel E

3.2 TileLink messages

TileLink messages may be packed in any order in the frame, up to a maximum of 64 messages (due to the 64-bit TLoE frame mask), within the limit of frame size set during initialization/configuration.

Configuration parameter *MaxStartOfMessageFlit* indicates the last allowable flit position, after the TloE header, where a new TileLink message can start. *MaxStartOfMessageFlit* can be set between 1 and 64 during initial configuration. Setting *MaxStartOfMessageFlit* = 1 results on TLoE frames with a single TileLink message and zero padding bytes between the TLoE header and the TileLink message.

The utilization of the Ethernet link bandwidth can be improved by packing multiple TileLink messages into a single TLoE frame, so a TLoE transmitter may choose to delay the transmission of a TLoE frame and/or insert padding into that frame in order to wait for additional TileLink messages to pack into the frame. TileLink performance is impacted by increased latency, so a TLoE transmitter should not delay TileLink messages indefinitely. If the TileLink transmitter delays a TileLink message, the delay should be limited using a configurable maximum timeout period (e.g. 1/32th of the round-trip-time).

The format of each TileLink message depends on the TileLink channel (A, B, C, D or E), the Opcode, and the size of the data field. The maximum size of the Data field, *MaxTileLinkBurstSize*, is set during initialization/configuration. Figure 10 shows all the different TileLink message formats. The messages associated to TileLink channels A, B and C consist of a header of 16 bytes, followed by an optional Data field or a combination of Mask+Data. The messages associated to TileLink channel D consist of a header of 8 or 16 bytes, followed by an optional data field. The messages associated to TileLink channel E consist of a header of 8 bytes.

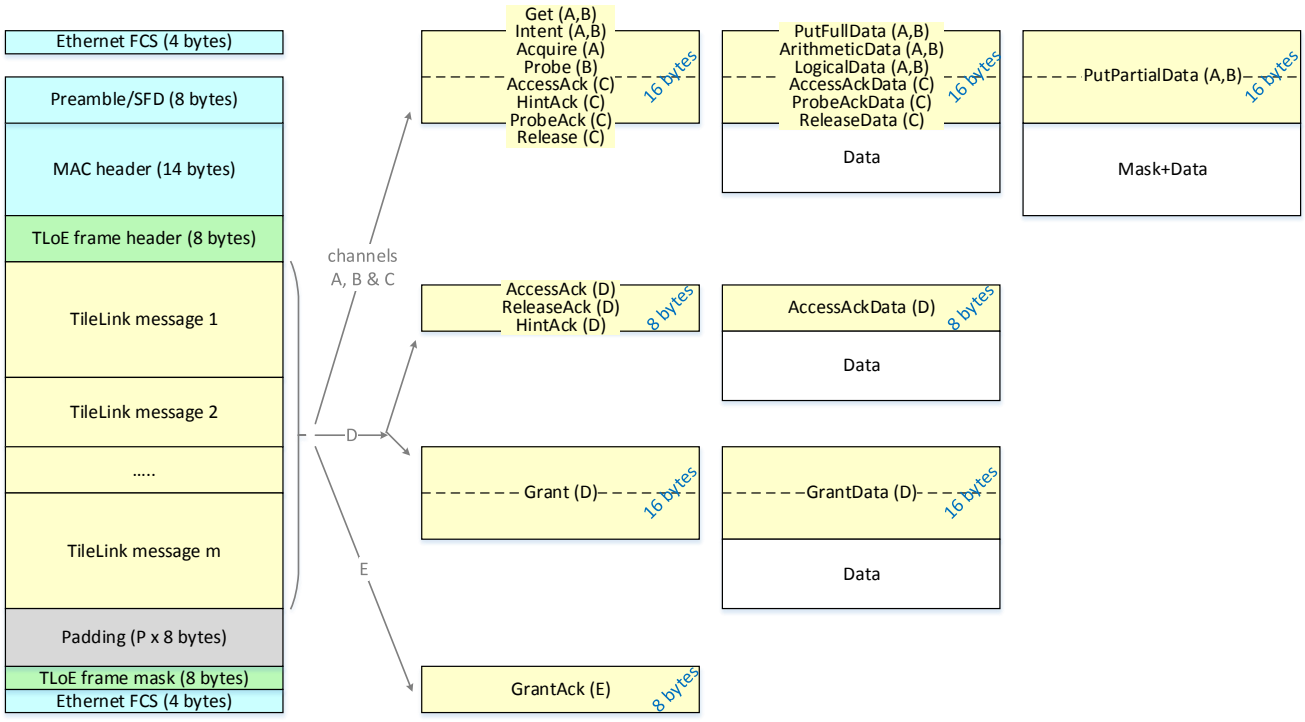

Figure 10 –TileLink message formatting

Figure 11 through Figure 14 show the detailed encoding of the TileLink messages, and Table 1 shows the association among each TLoE field and the different TileLink channel signals.

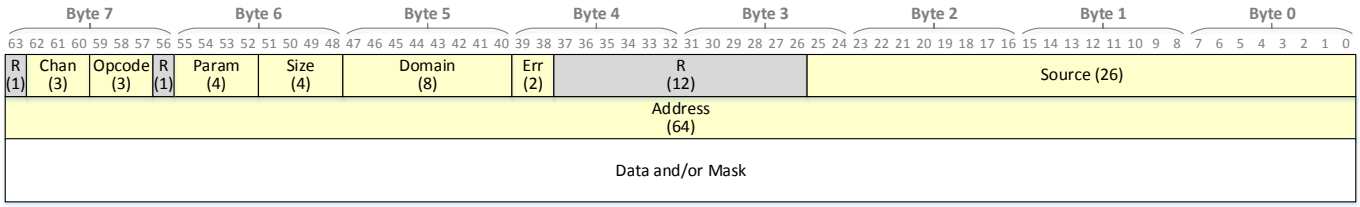

Figure 11 –TileLink message encoding for channels A, B and C

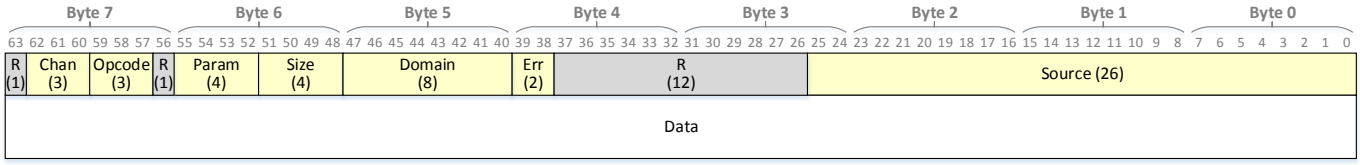

Figure 12 – TileLink message encoding for channel D – *AccessAck*, *ReleaseAck*, *HintAck* and *AccessAckData* messages

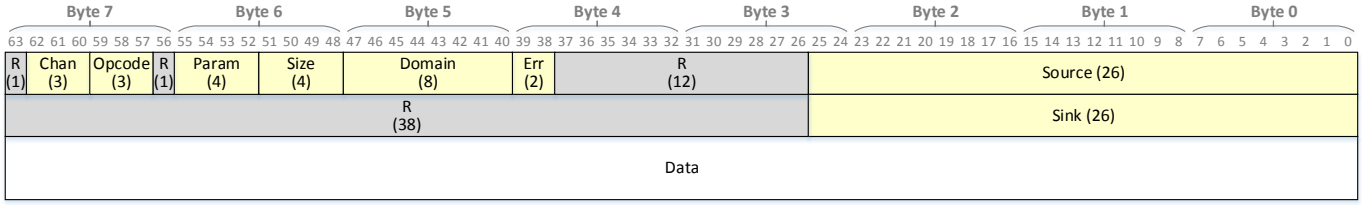

Figure 13 – TileLink message encoding for channel D – *Grant* and *GrantData* messages

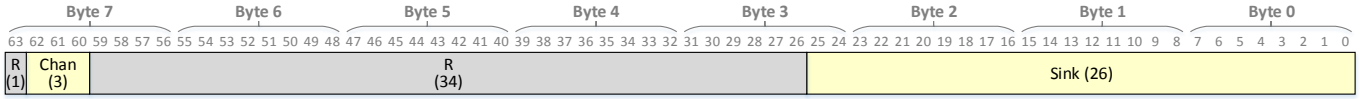

Figure 14 – TileLink message encoding for channel E

TLoE field	channel A	channel B	channel C	channel D	channel E	Description
Chan[2:0]	"001"	"010"	"011"	"100"	"101"	Associates each TLoE message to a specific TileLink channel.
Domain[7:0]	Domain ID	0	0	Domain ID		Ordering domain ID.
Opcode[2:0]	a_opcode	b_opcode	c_opcode	d_opcode		Each of these fields carries transparently the associated TileLink signal.
Param[3:0]	a_param	b_param	c_param	d_param		
Size[3:0]	a_size	b_size	c_size	d_size		
Err[0]	a_corrupt	b_corrupt	c_corrupt	d_corrupt		
Err[1]				d_denied		
Source[25:0]	a_source	b_source	c_source	d_source		
Sink[25:0]				d_sink	e_sink	
Address[63:0]	a_address	b_address	c_address			
Mask / Data	a_mask a_data	b_mask b_data	c_data	d_data		Carries the content of the TileLink *_mask and *_data signals, as described in Figure 15 and Figure 16.

Table 1 – Equivalence between TLoE fields and TileLink signals

The *Channel* field associates each message to a specific TileLink channel:

- 0: padding (64-bit, all-zero), see section 3.3
- 1-5: channels A-E
- 6-7: reserved

The Ordering Domain ID is used to provide message-ordering guarantees within different subsets of messages. A domain ID of 0 indicates that no ordering guarantees are required. A non-zero domain ID guarantees that the message is transmitted in FIFO order with respect to earlier messages with the same Domain ID. Note that the TLoE adapter delivers all the messages in order, and therefore the TLoE adapter does not need to use the Domain ID. The TLoE adapter simply carries the Ordering Domain ID end-to-end transparently.

The size of the Data and Mask fields corresponds to the value *n* indicated by the TileLink `a_size`, `b_size`, `c_size` and `d_size` signals.

Err[1] carries the TileLink `d_denied` signal, and Err[0] carries the TileLink `a_corrupt`, `b_corrupt`, `c_corrupt` or `d_corrupt` signal.

The Data and Mask fields are interleaved in groups of 64 mask bits followed by 64 data bytes, as shown in Figure 15. Figure 16 shows how each of the 64 mask bits is associated to each of the 64 data bytes.

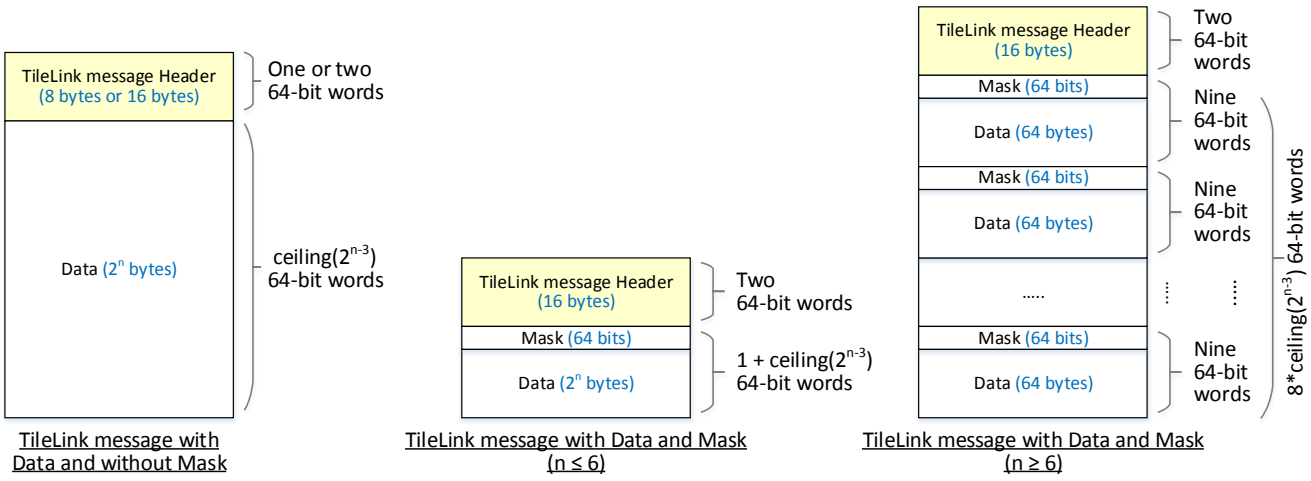

Figure 15 –TileLink message Data and Mask fields

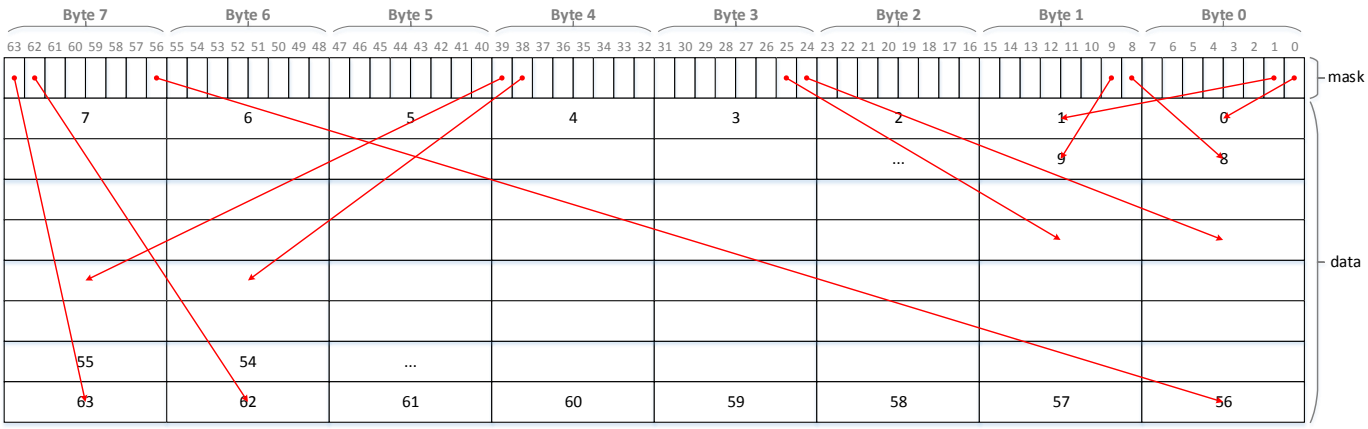

Figure 16 –Byte lane select association

3.3 Padding

Padding double-words are necessary if the TileLink information (TLoE header + messages + mask) comprises less than 46 bytes. Padding double-words may also inserted after each message in order to prevent underrun while optimizing packing of messages into frames. Padding double-words do not consume credits and are ignored by the TLoE receiver.

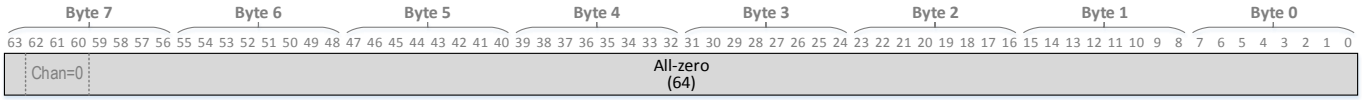

Figure 17 – Padding double-word (all zero)

3.4 TLoE frame mask

The 64-bit TLoE frame mask indicates the starting location of each of the *m* messages. The 64-bit mask can be used by the receiver to simplify the implementation by allowing parallel, single-cycle determination of message boundaries within the frame. A bit of the TLoE message mask is set to one to indicate the presence of the first double-word of message header at the corresponding 8-byte flit after the TLoE header. A bit of the TLoE frame mask is set to zero if a padding double-word is present at the corresponding 8-byte flit. The TLoE frame mask cannot represent the presence of a TileLink message header beyond the 64th flit (i.e. beyond the 512th byte). If a TileLink message has data that extends to the 64th flit (or beyond) then this message must be the last one on the TLoE frame.

The TLoE frame mask is always located right before the Ethernet FCS, regardless of the TLoE frame size. Therefore, if the number of bytes carried inside the Ethernet frame (TLoE frame header + TileLink messages + TLoE frame mask) is less than 46 then the TLoE adapter must stretch the TLoE frame by inserting padding double-words between the last TLoE message (message *m*) and the TLoE frame mask.

Figure 18 shows an example where five different TileLink messages (*m*=5) are packed on the same TLoE frame.

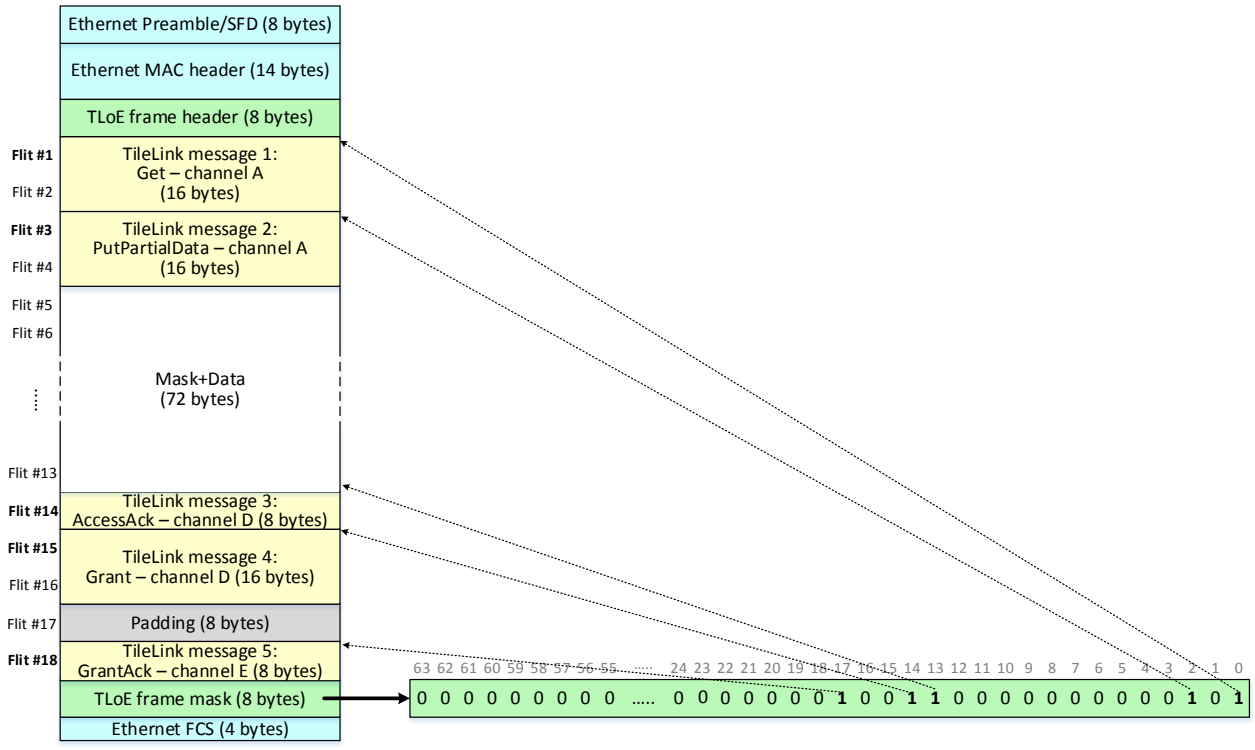

Figure 18 –TLoE frame mask example

4 Retransmission

TLoE retransmission is based on a sliding window protocol. The sliding window mechanism uses the *Sequence_number* field (see Figure 9) to identify and locate a packet in the order of transmission.

The TLoE transmitter increments the frame sequence number on each consecutive transmitted frame: *Sequence_number = (Sequence_number + 1) modulo* 2^{22}.

The TLoE receiver compares the received sequence number against the expected sequence number. If the received sequence number matches the expected sequence number, the received packet is accepted and processed. If the received sequence number does not match the expected sequence number, the received packet is dropped.

The TLoE receiver controls the operation of the retransmit protocol by sending ACK/NAK indications on the opposite direction.

If the TLoE frames are received in proper sequential order, the TLoE receiver sends an ACK indication inside each of the TLoE frames that are transmitted in the opposite direction. The ACK indication is encoded by setting the *Ack* field to 1 and the *Sequence_number_ack* field to the value of the last sequence number that was received in proper sequence order.

If a TLoE frame is received with an unexpected sequence number, the TLoE receiver sends a NAK indication inside the next TLoE frame that is transmitted in the opposite direction. The NAK indication is encoded by setting the *Ack* field to 0 and the *Sequence_number_ack* field to the value of the last sequence number that was received in proper sequence order.

The retransmission protocol is based on the Go-Back-N method. If the transmitter gets a NAK indication with *Sequence_number_ack*=N, and the transmitter has already transmitted the TLoE frame with *Sequence_number*=M (N ≤ M), the TLoE transmitter retransmits all the TLoE frames with sequence numbers N+1, …., M-1, M. The Go-Back-N method simplifies the design of the TLoE receiver buffers and has a low impact to performance if the links are not too unreliable.

The size of the sliding window limits how many packets can be in-flight. It allows the TLoE transmitter to transmit a number of TLoE packets in succession, before the first ACK indication is received on the opposite direction. To utilize the full bandwidth of the network, the sliding window must be at least twice as large as the maximum number of TLoE frames that can be transmitted before the first ACK is received, i.e. at least twice as large as the Bandwidth x Delay product (BDP). The size of the *Sequence_number* and *Sequence_number_ack* fields allows to support a BDP as large as 2^{22-1} * 512byte = 8Gbit.

During TLoE connection initialization, the transmitted and expected sequence numbers are initialized to zero. This is better described in terms of conceptual counters used by the TLoE transmitter and TLoE receiver; these conceptual counters do not imply nor require a particular implementation, but are useful to describe the TLoE retransmission operation:

- TLoE transmitter:
    - The TLoE transmitter operation is controlled using the following counters:
        - `NEXT_TX_SEQ` stores the *Sequence_number* value that will be used for the next transmitted frame. Set to 0 during initialization.
        - `ACKD_SEQ` stores the sequence number that was acknowledged by the most recently received ACK or NAK indication. Set to 2^{22}-1 during initialization.

    - The transmission of a new TLoE frame can be started if the retransmit buffer has sufficient space to store the new frame and (NEXT_TX_SEQ - ACKD_SEQ) mod 2^{22} < 2^{21}.
    - During the transmission of a TLoE frame, the NEXT_TX_SEQ counter is updated as follows: NEXT_TX_SEQ = (NEXT_TX_SEQ + 1) mod 2^{22}.
- TLoE receiver:
    - The TLoE receiver operation is controlled using the following counter:
        - NEXT_RX_SEQ stores the expected *Sequence_number* value of the next received TLoE frame. Set to 0 during initialization.
    - The TLoE receiver operates as follows:
        - If *Sequence_number* = NEXT_RX_SEQ then the TLoE frame is processed, NEXT_RX_SEQ is updated, and a positive acknowledge is sent using the received sequence number.
        - If (NEXT_RX_SEQ − *Sequence_number*) mod 2^{22} <= 2^{21} then the received TLoE frame is a duplicate. The received TLoE frame is dropped, the NEXT_RX_SEQ is not updated, and a positive acknowledge is sent using the received sequence number.
        - Otherwise, the received TLoE frame is out of sequence, indicating that one or more TLoE frames have been lost. The received TLoE frame is dropped, the NEXT_RX_SEQ is not updated, and a negative acknowledge is sent using the last sequence number that was received in proper sequence order.

Figure 19 shows a TLoE retransmission operation example:

- At time t0, during TLoE connection initialization, the TLoE transmitter and TLoE receiver counters are reset: NEXT_TX_SEQ=0, ACKD_REQ=2^{22}-1, NEXT_RX_SEQ=0.
- TLoE endpoint 1 transmits TLoE frames (blue arrows) in proper sequence order: Sequence_number=0, 1, 2, 3, 4, 5, etc.
- TLoE endpoint 2 sends ACK indications (green arrows) at times t1 (Sequence_number_ack=4, Ack=1) and t2 (Sequence_number_ack=7, Ack=1).
- At time t3, TLoE endpoint 2 detects that the frame with Sequence_number=9 has been lost, and sends a NAK indication with Sequence_number_ack=8 and Ack=0 (red arrow).
- At time t4, TLoE endpoint 1 receives the NAK indication and starts retransmission of TLoE frames starting with Sequence_number=9.
- At time t5, TLoE endpoint 1 restarts normal transmission.

Figure 20 shows a retransmit buffer operation example.

The transmission of an ACK indication (see section 3.1) can be delayed, acknowledgment of received packets need not be sent immediately. This helps improve the utilization of the Ethernet link bandwidth by reducing the number of “acknowledge only” frames, i.e. TLoE frames carrying zero TileLink messages and no credit updates. This is illustrated in Figure 19, where a single ACK indication acknowledges the reception of TLoE frames with sequence numbers 3 and 4, and another ACK indication acknowledges the reception of TLoE frames with sequence numbers 5, 6 and 7.

The transmission of an ACK indication should not be delayed indefinitely. If an implementation delays the transmission of an ACK indication (to combine it with further TileLink messages and/or flow control credit updates), the delay should be limited using a configurable maximum timeout period (e.g. 1/4th of the overall round-trip-time).

The reception of an “acknowledge only” TLoE frame should not be acknowledged by generating another “acknowledge only” TLoE frame.

Retransmission of TLoE frames is triggered in one of two different ways:

- When a negative acknowledge indication (NAK) is received, as shown in Figure 19.
- When the expected positive acknowledge indication (ACK) is not received during a configurable timeout period (e.g. 2 x round-trip-time).

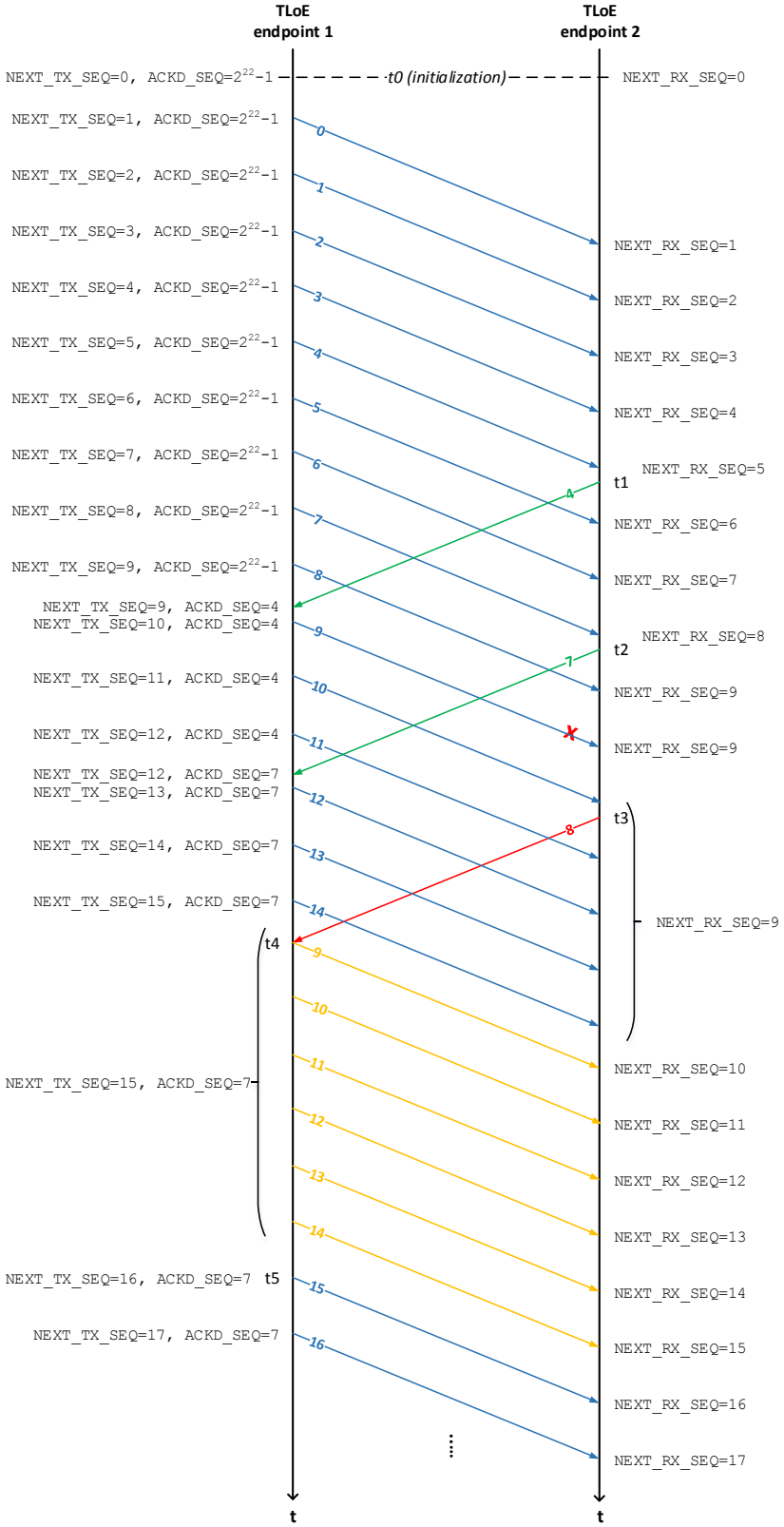

Figure 19 – Retransmission operation example

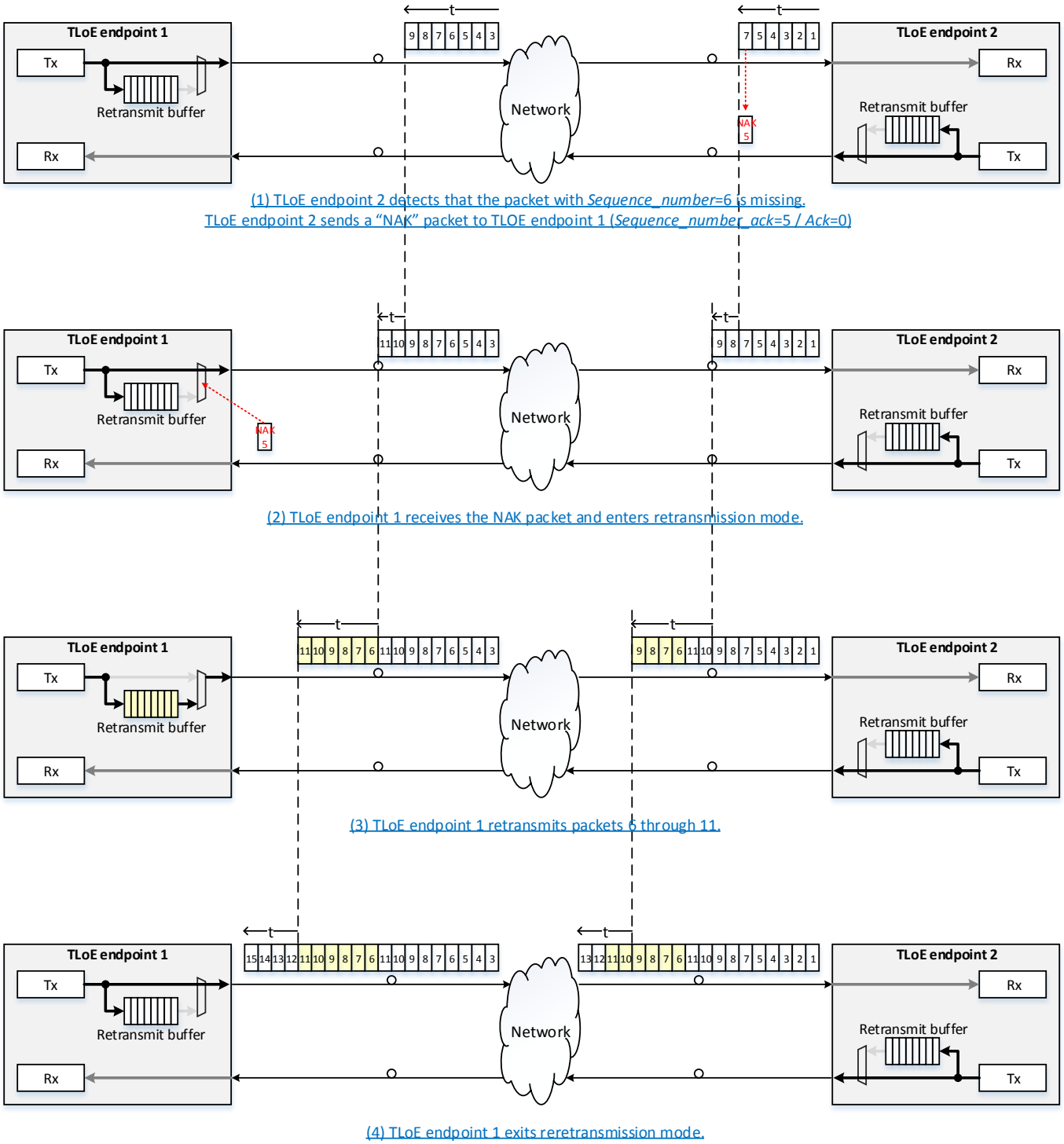

Figure 20 – Retransmit buffer operation example

5 Flow control

TLoE flow control is required to extend TileLink's valid/ready handshake end-to-end. To avoid receiver buffer overflows, the TLoE transmitter needs to obey the flow control indications generated by the TLoE receiver.

TLoE flow control is credit-based. A credit unit corresponds to 64 bits of data (i.e. a "flit").

The TLoE transmitter operation is controlled using a credit counter.

Every time the TLoE receiver reads a message from the receive buffer, the corresponding number of credits is returned to the remote TLoE transmitter. After initialization, the TLoE receiver has a number of credits, that corresponds to the size of its receive buffer, ready to be given to the TLoE transmitter.

The TLoE transmitter can transmit new TileLink messages if the credit counter value is larger than zero. Every time a TileLink message is transmitted, the credit counter value is decreased by the number of transmitted flits. The credit counter value is increased when the remote TLoE receiver returns credits to the TLoE transmitter. Note that the a TLoE frames that carries zero TileLink message do not consume credits.

Figure 21 illustrates TLoE flow control operation. The operation of the TLoE flow control protocol is symmetric: TLoE endpoint 1 returns credits to TLoE endpoint 2 (blue color), and TLoE endpoint 2 returns credits to TLoE endpoint 1 (green color). TLoE carries five independent TileLink channels per direction (A, B, C, D and E), so each TLoE receiver has five independent receive buffers and each TLoE transmitter has five independent credit counters.

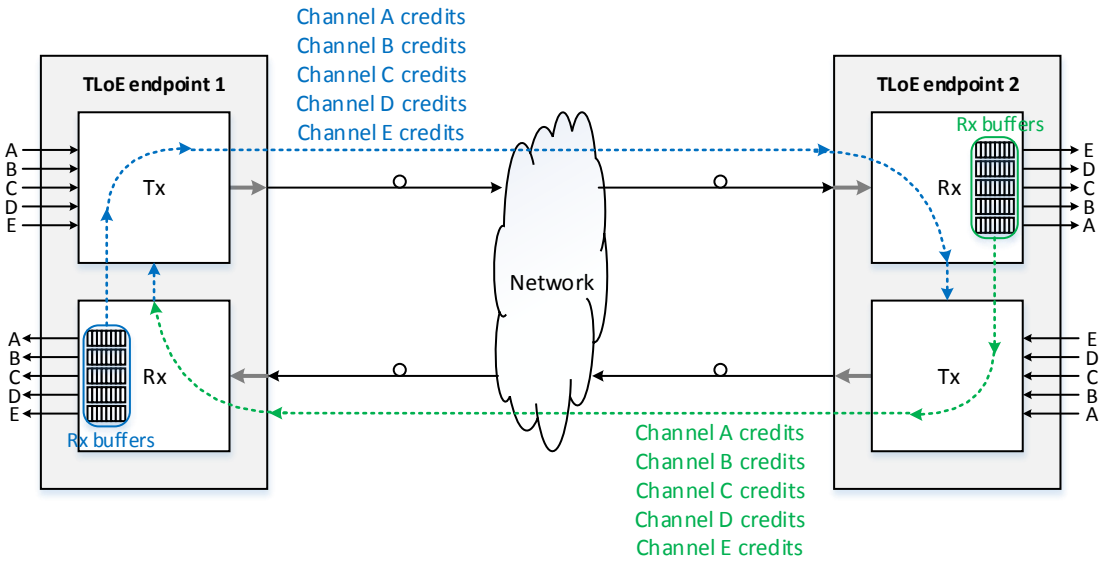

Figure 21 – Flow control operation

The credits are returned to the remote TLoE transmitter using the *Credit*/*Chan* fields described in section 3.1.

6 Transmission order

Figure 22 illustrates TLoE transmission order:

- Each TLoE 64-bit word (63:0) is transmitted using a big-endian scheme. The most significant byte is transmitted first (63:56) and the least significant byte is transmitted last (7:0).
- Each byte is transmitted as specified by the 802.3 standard. The least significant bit is transmitted first (bit 0), and the most significant bit is transmitted last (bit 7).

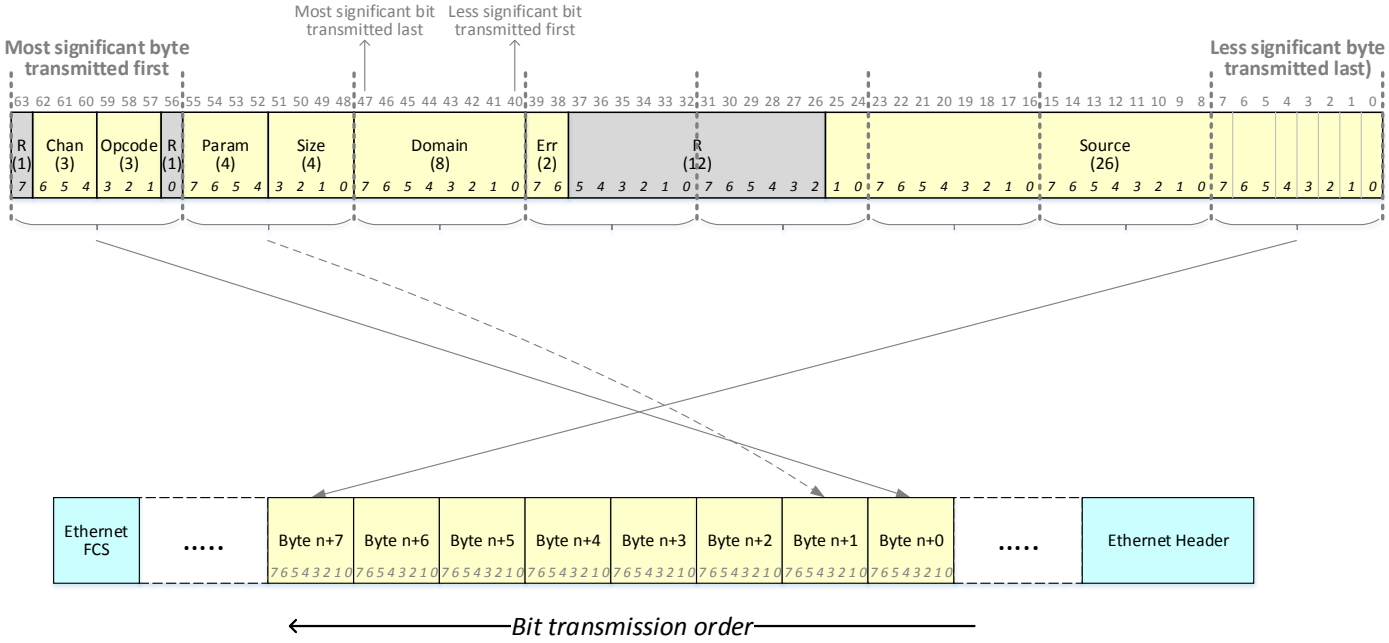

Figure 22 – TLoE transmission order

Table 2 shows the encoding of a TLoE sample packet. Bits are transmitted from right to left; bit 0 is transmitted first and bit 63 is transmitted last. The 64 bit words are transmitted from top to bottom.

63							7 6 5 4 3 2 1 0
D5	55	55	55	55	55	55	0x78
SA[39:32] 8b	SA[47:40] 0e	DA[7:0] 38	DA[15:8] 05	DA[23:16] 77	DA[31:24] 20	DA[39:32] 00	DA[47:40] 08=00001000
TLoE_hdr[55:48]	TLoE_hdr[63:56]	ET[7:0] 25	ET[15:8] 89	SA[7:0] 00	SA[15:8] 00	SA[23:16] 00	SA[31:24] 00
TLM_1[55:48]	TLM_1[63:56]	TLoE_hdr[7:0]	TLoE_hdr[15:8]	TLoE_hdr[23:16]	TLoE_hdr[31:24]	TLoE_hdr[39:32]	TLoE_hdr[47:40]
TLM_2[55:48]	TLM_2[63:56]	TLM_1[7:0]	TLM_1[15:8]	TLM_1[23:16]	TLM_1[31:24]	TLM_1[39:32]	TLM_1[47:40]
…	…	…	…	…	…	…	…
M[55:48]	M[63:56]	TLM_M[7:0]	TLM_M[15:8]	TLM_M[23:16]	TLM_M[31:24]	TLM_M[39:32]	TLM_M[47:40]
CRC[16:23]	CRC[24:31]	M[7:0]	M[15:8]	M[23:16]	M[31:24]	M[39:32]	M[47:40]
-	-	-	-	-	-	CRC[0:7]	CRC[8:15]

Table 2 – MAC header encoding example

The following sample values are shown in red color.

- DA[47:0] = 08 00 20 77 05 38
- SA[47:0] = 0e 8b 00 00 00 00
- EtherType[15:0] = 89 25

Note that the Ethertype value used to identify the TLoE protocol is not yet defined, the 0x8925 value is just an example.

This is a non-tagged Ethernet frame, so the size of the MAC header is fourteen bytes and the first byte of the TLoE frame header (TLoE_hdr[63:56]) appears on the seventh byte lane; the seventh byte lane corresponds to TXD/RXD<55:48> on the CGMII interface, and D_6 on a 66b block.

In case of an 802.1Q single VLAN tag, TLoE_hdr[63:56] would appear on the third byte lane. In case of an 802.1ad double-tagged frame, TLoE_hdr[63:56] would appear on the seventh byte lane.

The different fields are transmitted in the following order:

- Ethernet MAC header
    - DA[40] (underlined) is transmitted first, right after the last Preamble/SFD bit.
    - TypeLength[7] is transmitted last.
- TLoE frame header (section 3.1)
    - TLoE_hdr[56] (*Chan[0]*) is transmitted first, after TypeLength[7].
    - TLoE_hdr[7] (*Reserved* bit) is transmitted last.
- m TileLink messages (section TileLink messages)
    - TLM_1[56] (bit 56 of the first double word of the first TileLink message) is transmitted first, after TLoE_hdr[7]. This is *Source[18]* for channels A, B, C and D, and *Sink[18]* for channel E.
    - TLM_M[7] (bit 7 of the last double word of TileLink message m) is transmitted last.
- TLoE frame mask mask
    - M[56] is transmitted first, after TLM_M[7].
    - M[7] is transmitted last.

Annex A: **Encoding examples**

TLoE messages are encoded using one of the nine different formats shown in Figure 23.

Section A.1 shows encoding examples with a single TileLink message per TLoE frame. Each example corresponds to one of the nine different format templates shown in Figure 23.

Section A.2 shows encoding examples with multiple TileLink messages per TLoE frame.

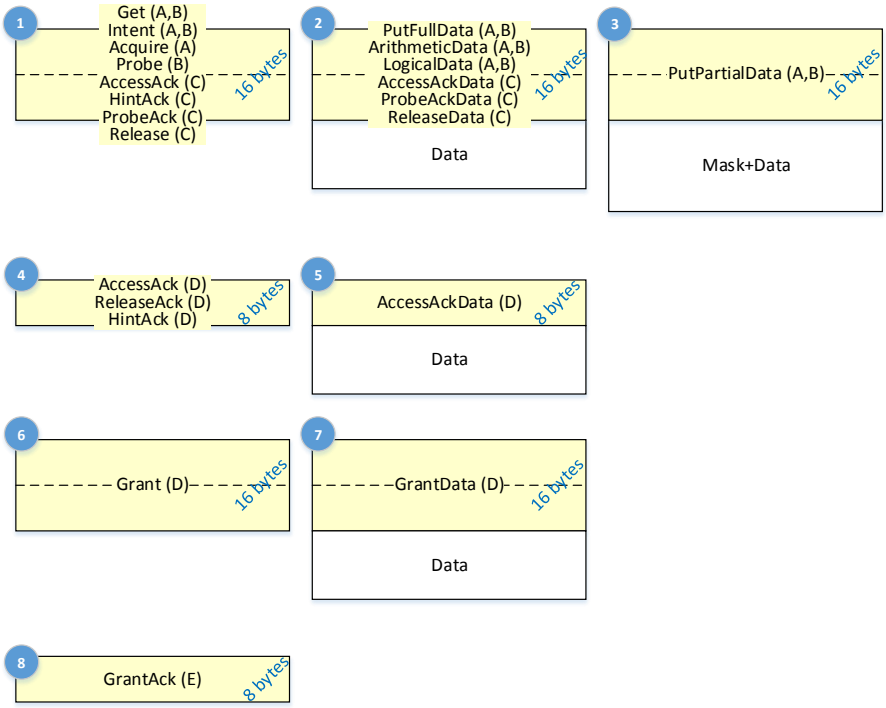

Figure 23 – TileLink messages formats

A.1 Single message per TLoE frame

A.1.1 Get on channel A

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=1* indicates that is a channel A message.
  - *Opcode=4* indicates that this is a Get message.
  - Size=5 indicates that this message reads 32 bytes of data.
  - Padding is inserted before the TLoE frame mask to ensure that the total number of bytes (48) is ≥46.

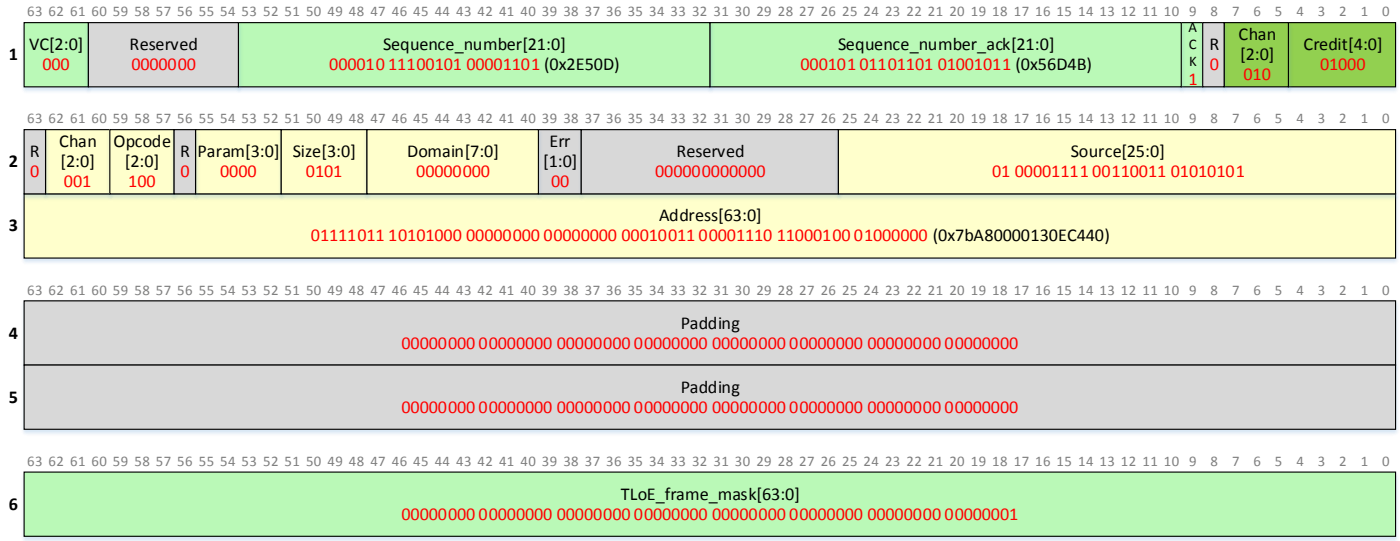

Figure 24 – Get on channel A

A.1.2 PutFullData on channel A

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=2/Chan=3* indicate that this frame returns 2^2=4 credits for channel C.
- TileLink message
  - *Chan=1* indicates that is a channel A message.
  - *Opcode=0* indicates that this is a PutFullData message.
  - Size=6 indicates that this message writes 64 bytes of data.
  - In this example, the value of each data byte (0x40, 0x41, etc.) matches the value of the less significant bits of the associated memory address.

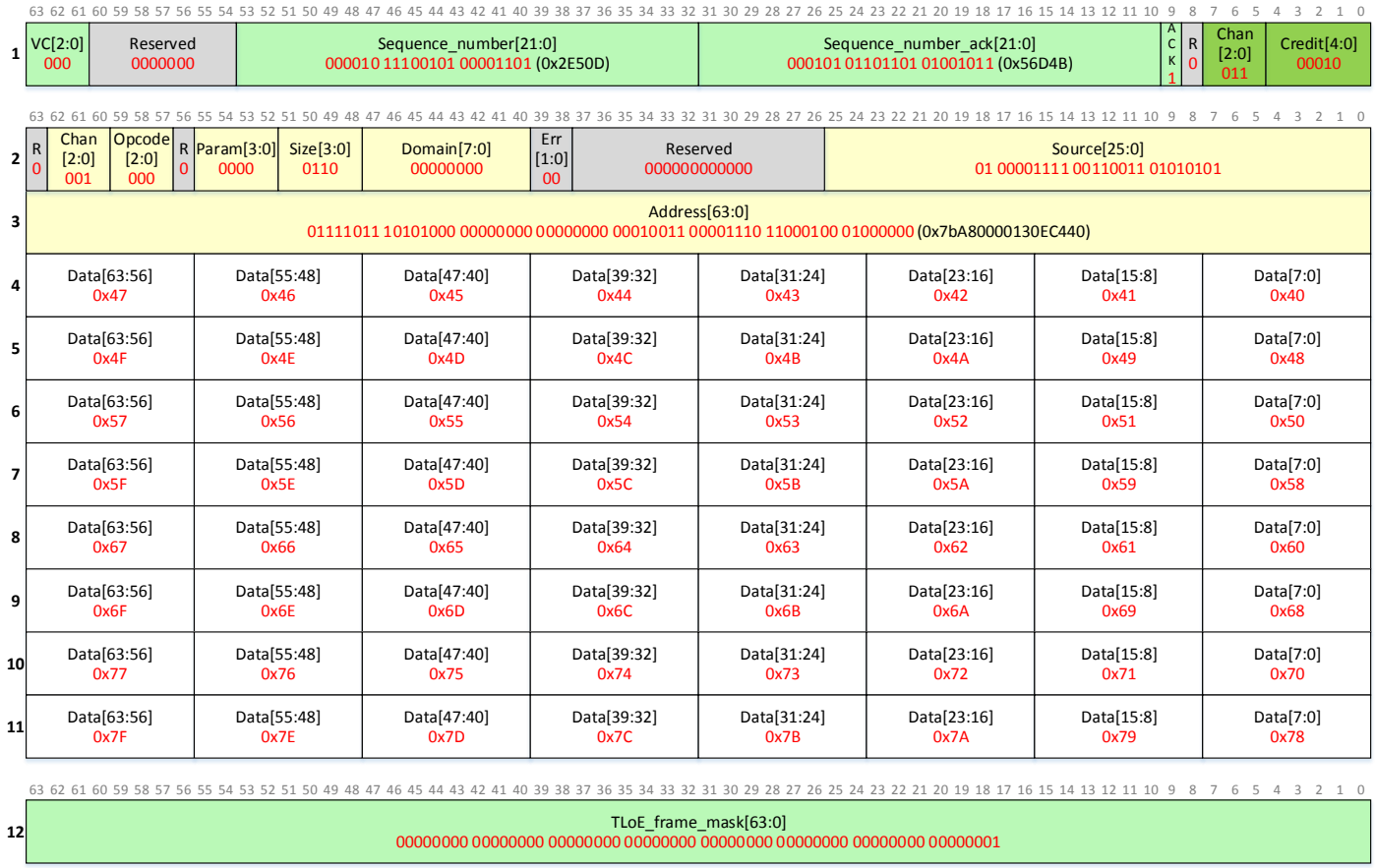

Figure 25 – PutFullData on channel A

A.1.3 PutPartialData on channel B

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=6/Chan=4* indicate that this frame returns 2^6=64 credits for channel D.
- TileLink message
  - *Chan=2* indicates that is a channel B message.
  - *Opcode=1* indicates that this is a PutPartialData message.
  - Size=4 indicates that this message writes up to 16 bytes of data.
  - In this example, the value of each data byte (0x40, 0x41, etc.) matches the value of the less significant bits of the associated memory address.
  - The *Mask* field indicates that the first two bytes (0x40 and 0x41) are not written.

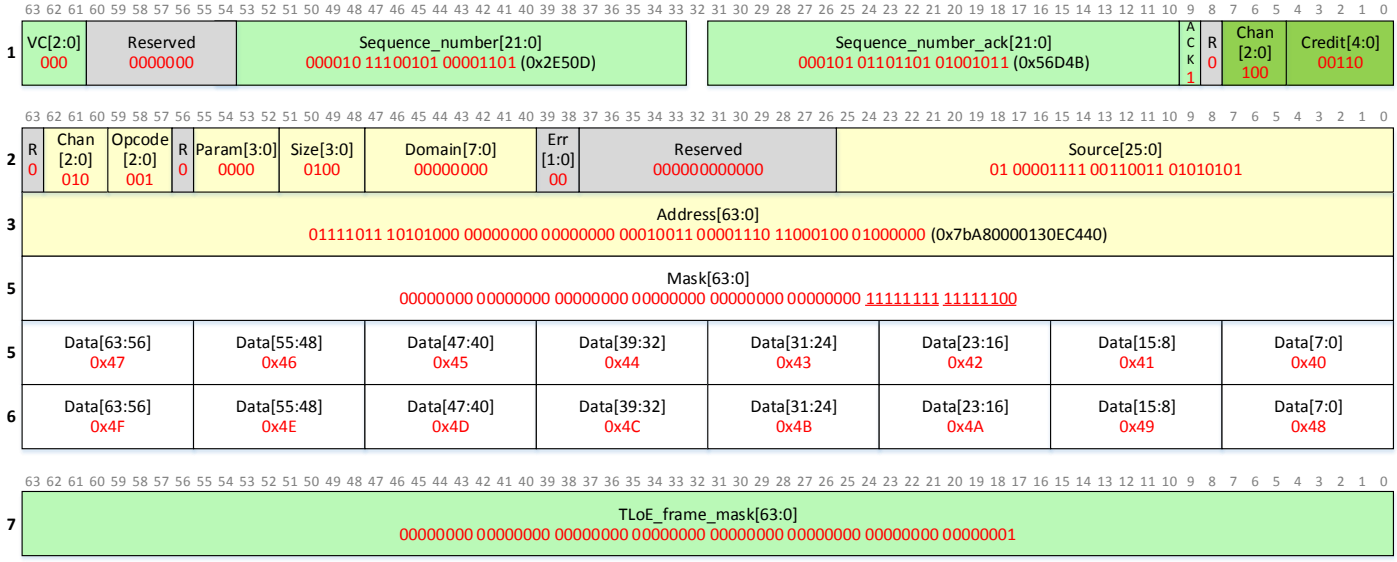

Figure 26 – PutPartialData on channel B

A.1.4 AccessAck on channel D

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=4* indicates that is a channel D message.
  - *Opcode=0* indicates that this is an AccessAck message.
  - Size=5 indicates that the slave accessed 32 bytes of data.
  - Err[1] carries the `d_denied` signal, and Err[0] is reserved and set to zero.
  - Padding is inserted before the TLoE frame mask to ensure that the total number of bytes (48) is ≥46.

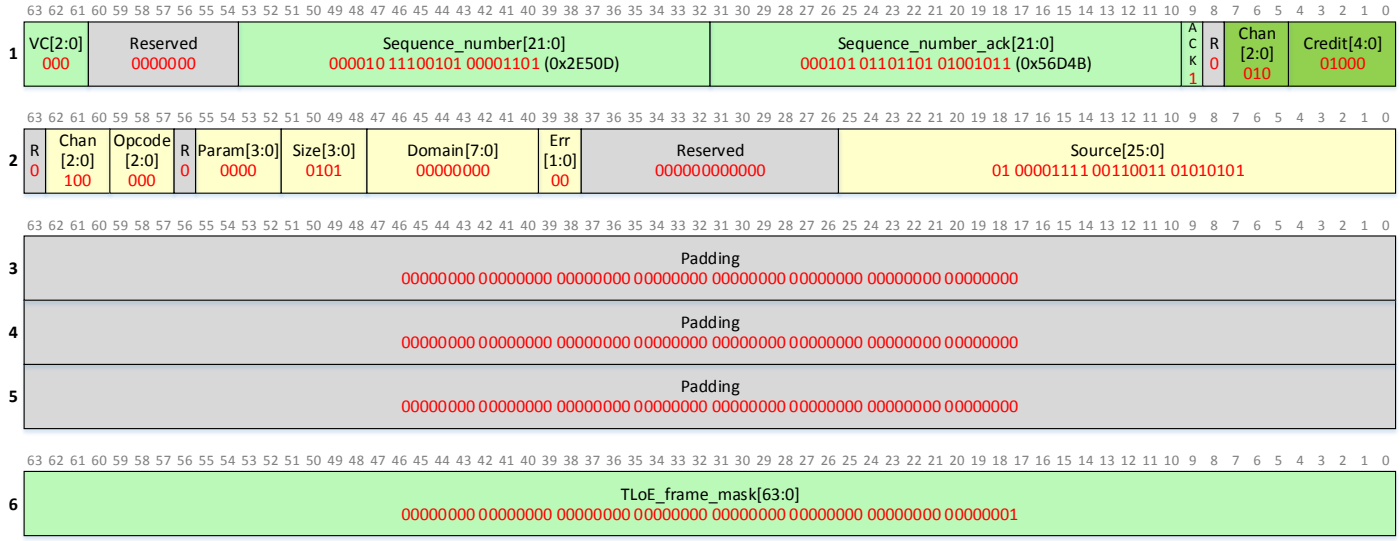

Figure 27 – AccessAck on channel D

A.1.5 AccessAckData on channel D

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=4* indicates that is a channel D message.
  - *Opcode=1* indicates that this is an AccessAckData message.
  - Size=5 indicates that the slave accessed 32 bytes of data.
  - Err[1] carries the `d_denied` signal, and Err[0] carries the `d_corrupt` signal.

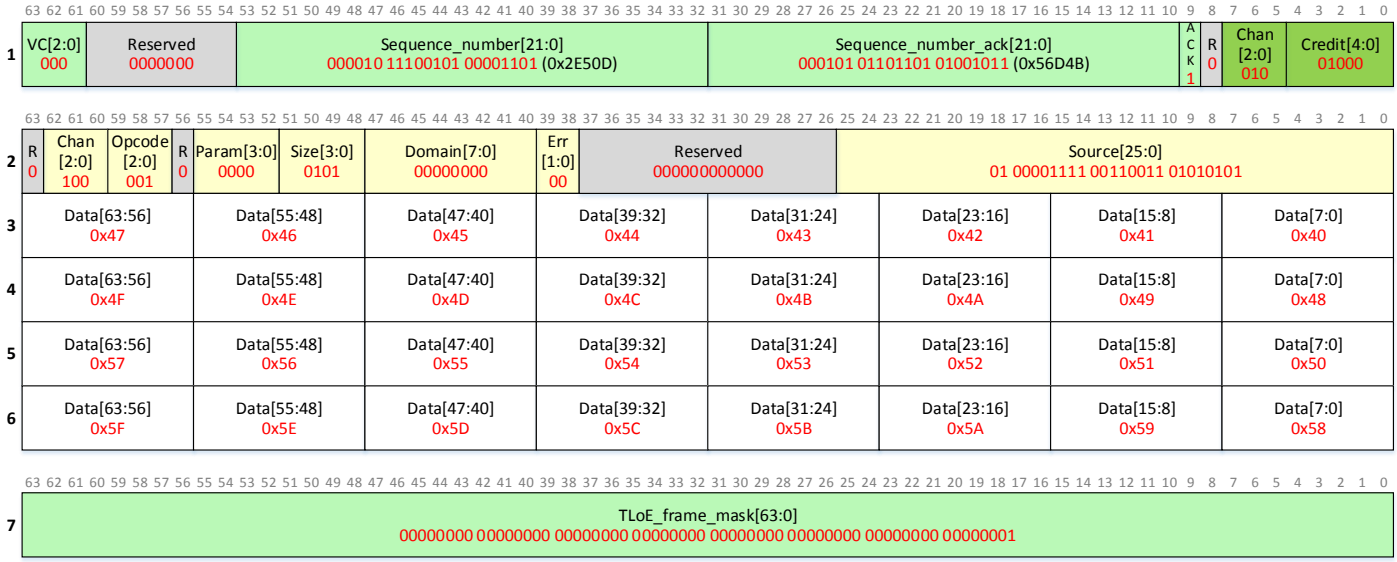

Figure 28 – AccessAckData on channel D

A.1.6 Grant on channel D

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=4* indicates that is a channel D message.
  - *Opcode=4* indicates that this is a Grant message.
  - *Param=0* indicates toT permission transfer.
  - Size=6 indicates that the slave accessed 64 bytes of data.
  - Err[1] carries the `d_denied` signal, and Err[0] is reserved and set to zero.
  - Padding is inserted before the TLoE frame mask to ensure that the total number of bytes (48) is ≥46.

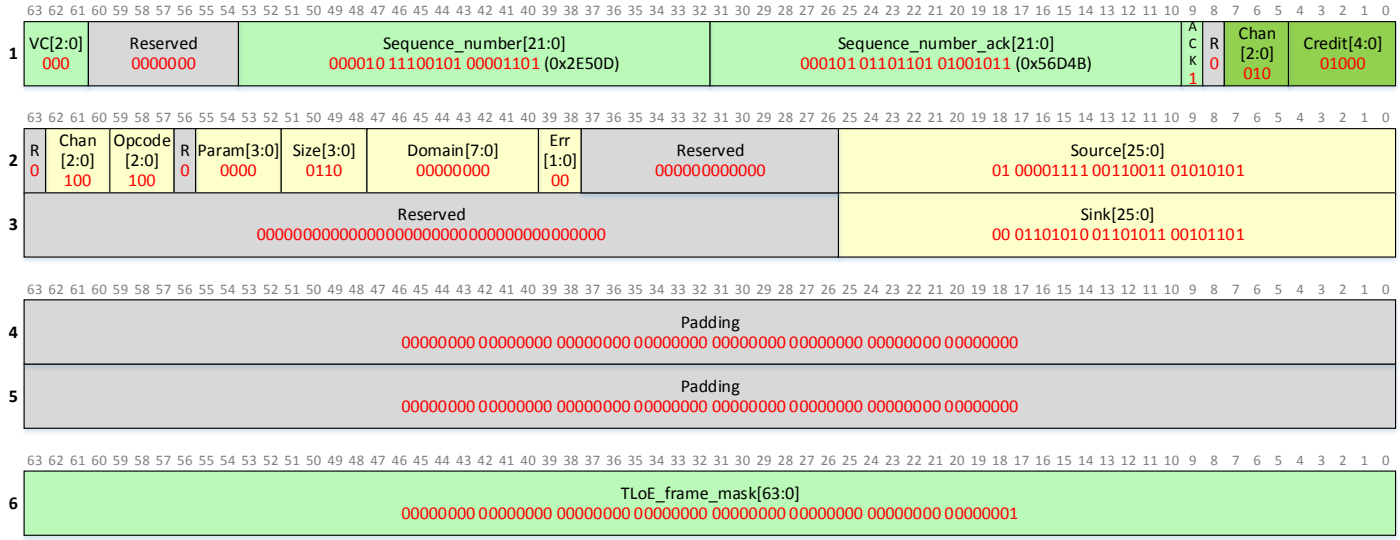

Figure 29 – Grant on channel D

A.1.7 GrantData on channel D

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=4* indicates that is a channel D message.
  - *Opcode=5* indicates that this is a GrantData message.
  - *Param=0* indicates toN permission transfer.
  - Size=6 indicates that the slave is transferring 64 bytes of data.
  - Err[1] carries the `d_denied` signal, and Err[0] carries the `d_corrupt` signal.

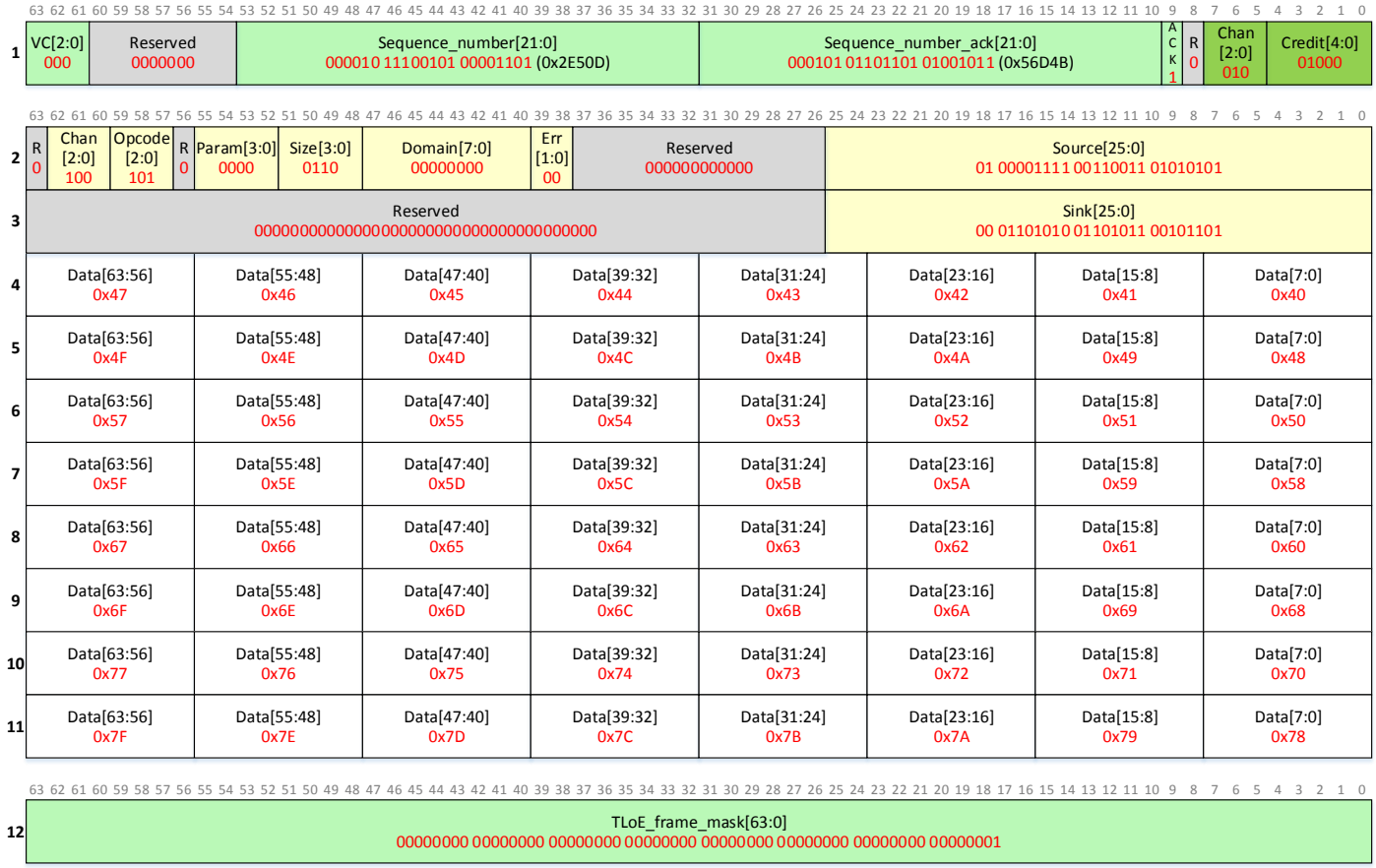

Figure 30 – GrantData on channel D

A.1.8 GrantAck on channel E

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=8/Chan=2* indicate that this frame returns 2^8=256 credits for channel B.
- TileLink message
  - *Chan=5* indicates that is a channel E message.

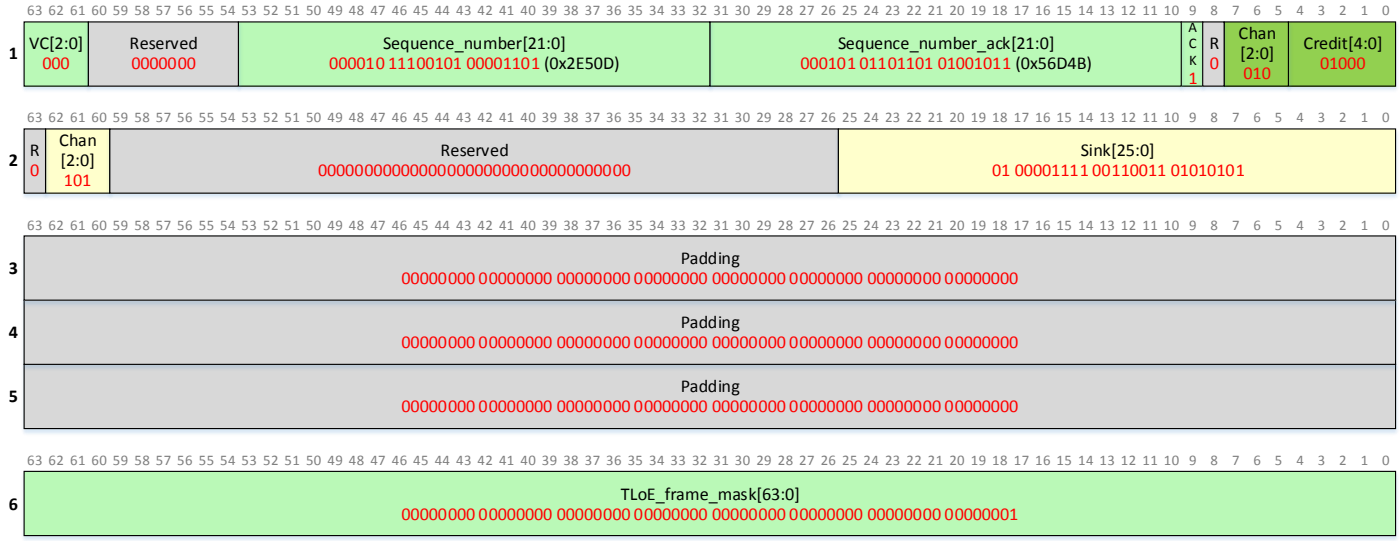

Figure 31 – GrantAck on channel E

A.2 Multiple messages per TLoE frame

Figure 32 shows an example with five different TileLink messages bundled inside the same TLoE frame: PutFullData on A + PutPartialData on B + Get on A + AccessAck on D + GrantAck on E.

- The TLoE frame header
  - *Ack=1* indicates a positive acknowledge (ACK).
  - *Credit=2/Chan=3* indicate that this frame returns 2^2=4 credits for channel C.
- First TileLink message (PutFullData on channel A)
  - *Chan=1* indicates that is a channel A message.
  - *Opcode=0* indicates that this is a PutFullData message.
  - Size=6 indicates that this message writes 64 bytes of data.
  - In this example, the value of each data byte (0x40, 0x41, etc.) matches the value of the less significant bits of the associated memory address.
- Second TileLink message (PutPartialData on channel B)
  - *Chan=2* indicates that is a channel B message.
  - *Opcode=1* indicates that this is a PutPartialData message.
  - Size=4 indicates that this message writes up to 16 bytes of data.
  - In this example, the value of each data byte (0x80, 0x81, etc.) matches the value of the less significant bits of the associated memory address.
  - The *Mask* field indicates that the first two bytes (0x80 and 0x81) are not written.
- Third TileLink message (Get on channel A)
  - *Chan=1* indicates that is a channel A message.
  - *Opcode=4* indicates that this is a Get message.
  - Size=5 indicates that this message reads 32 bytes of data.
- Fourth TileLink message (AccessAck on channel D)
  - *Chan=4* indicates that is a channel D message.
  - *Opcode=0* indicates that this is an AccessAck message.
  - Size=5 indicates that the slave accessed 32 bytes of data.
  - Err[1] carries the `d_denied` signal, and Err[0] is reserved and set to zero.
- Fifth TileLink message (GrantAck on channel E)
  - *Chan=5* indicates that is a channel E message.

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	VC[2:0] 000	Reserved 0000000	Sequence_number[21:0] 000010 11100101 00001101 (0x2E50D)	Sequence_number_ack[21:0] 000101 01101101 01001011 (0x56D4B)	ACK 1	R 0	Chan[2:0] 011	Credit[4:0] 00010

2	R 0	Chan[2:0] 001	Opcode[2:0] 000	R 0	Param[3:0] 0000	Size[3:0] 0110	Domain[7:0] 00000000	Err[1:0] 00	Reserved 000000	Source[25:0] 01 00001111 00110011 01010101	Reserved 000000
3	Address[63:0] 01111011 10101000 00000000 00000000 00010011 00001110 11000100 01000000 (0x7bA80000130EC440)										

4	Data[63:56] 0x47	Data[55:48] 0x46	Data[47:40] 0x45	Data[39:32] 0x44	Data[31:24] 0x43	Data[23:16] 0x42	Data[15:8] 0x41	Data[7:0] 0x40
5	Data[63:56] 0x4F	Data[55:48] 0x4E	Data[47:40] 0x4D	Data[39:32] 0x4C	Data[31:24] 0x4B	Data[23:16] 0x4A	Data[15:8] 0x49	Data[7:0] 0x48
6	Data[63:56] 0x57	Data[55:48] 0x56	Data[47:40] 0x55	Data[39:32] 0x54	Data[31:24] 0x53	Data[23:16] 0x52	Data[15:8] 0x51	Data[7:0] 0x50
7	Data[63:56] 0x5F	Data[55:48] 0x5E	Data[47:40] 0x5D	Data[39:32] 0x5C	Data[31:24] 0x5B	Data[23:16] 0x5A	Data[15:8] 0x59	Data[7:0] 0x58
8	Data[63:56] 0x67	Data[55:48] 0x66	Data[47:40] 0x65	Data[39:32] 0x64	Data[31:24] 0x63	Data[23:16] 0x62	Data[15:8] 0x61	Data[7:0] 0x60
9	Data[63:56] 0x6F	Data[55:48] 0x6E	Data[47:40] 0x6D	Data[39:32] 0x6C	Data[31:24] 0x6B	Data[23:16] 0x6A	Data[15:8] 0x69	Data[7:0] 0x68
10	Data[63:56] 0x77	Data[55:48] 0x76	Data[47:40] 0x75	Data[39:32] 0x74	Data[31:24] 0x73	Data[23:16] 0x72	Data[15:8] 0x71	Data[7:0] 0x70
11	Data[63:56] 0x7F	Data[55:48] 0x7E	Data[47:40] 0x7D	Data[39:32] 0x7C	Data[31:24] 0x7B	Data[23:16] 0x7A	Data[15:8] 0x79	Data[7:0] 0x78

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

12	R 0	Chan[2:0] 010	Opcode[2:0] 001	R 0	Param[3:0] 0000	Size[3:0] 0100	Domain[7:0] 00000000	Err[1:0] 00	Reserved 000000	Source[25:0] 00 01011010 00110011 01011101	Reserved 000000
13	Address[63:0] 01111011 10101000 00000000 00000000 10101010 10101010 10101010 10000000 (0x7bA80000130EC480)										
14	Mask[63:0] 00000000 00000000 00000000 00000000 00000000 00000000 <u>11111111</u> <u>11111100</u>										

15	Data[63:56] 0x87	Data[55:48] 0x86	Data[47:40] 0x85	Data[39:32] 0x84	Data[31:24] 0x83	Data[23:16] 0x82	Data[15:8] 0x81	Data[7:0] 0x80
16	Data[63:56] 0x8F	Data[55:48] 0x8E	Data[47:40] 0x8D	Data[39:32] 0x8C	Data[31:24] 0x8B	Data[23:16] 0x8A	Data[15:8] 0x89	Data[7:0] 0x88

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

17	R 0	Chan[2:0] 001	Opcode[2:0] 100	R 0	Param[3:0] 0000	Size[3:0] 0101	Domain[7:0] 00000000	Err[1:0] 00	Reserved 000000	Source[25:0] 01 00001111 00110011 01010110	Reserved 000000
18	Address[63:0] 01111011 10101000 00000000 00000000 00010011 00001110 11000100 10010000 (0x7bA80000130EC490)										
19	Mask[63:0] 00000000 00000000 00000000 00000000 <u>11111111</u> <u>11111111</u> <u>11111111</u> <u>11111111</u>										

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

20	R 0	Chan[2:0] 100	Opcode[2:0] 000	R 0	Param[3:0] 0000	Size[3:0] 0101	Domain[7:0] 00000000	Err[1:0] 00	Reserved 000000	Source[25:0] 00 00110011 01110101 01111000	Reserved 000000

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

21	R 0	Chan[2:0] 101	Reserved 0000000000000000000000000000	Sink[25:0] 00 00010001 10100101 00110110	Reserved 000000

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

22	TLoE_frame_mask[63:0] 00000000 00000000 00000000 00000000 00000000 00001100 10000100 00000001

Figure 32 – Multiple messages per TLoE frame example

A.3 Operation examples

Figure 33 and Figure 34 show operation examples where TLoE frames are transferred between a TileLink Master on the left side and a TileLink Slave on the right side. These examples pack a single TileLink message per TLoE frame. The blue arrows represent logical dependencies.

The example in Figure 33 shows Put and Get access operations (Get → PutPartialData → PutFullData → Get) that are initiated by the TileLink Master on two different address ranges, hAAAAAAAAAAAAxxxx and hBBBBBBBBBBBBxxxx. In this example, the TileLink Slave responds to independent requests within each address range in FIFO order.

The example in Figure 34 mixes access and transfer operations. Some operations are initiated by the TileLink Master (ReleaseData → Get → Acquire) and other operations are initiated by the TileLink Slave (Probe).

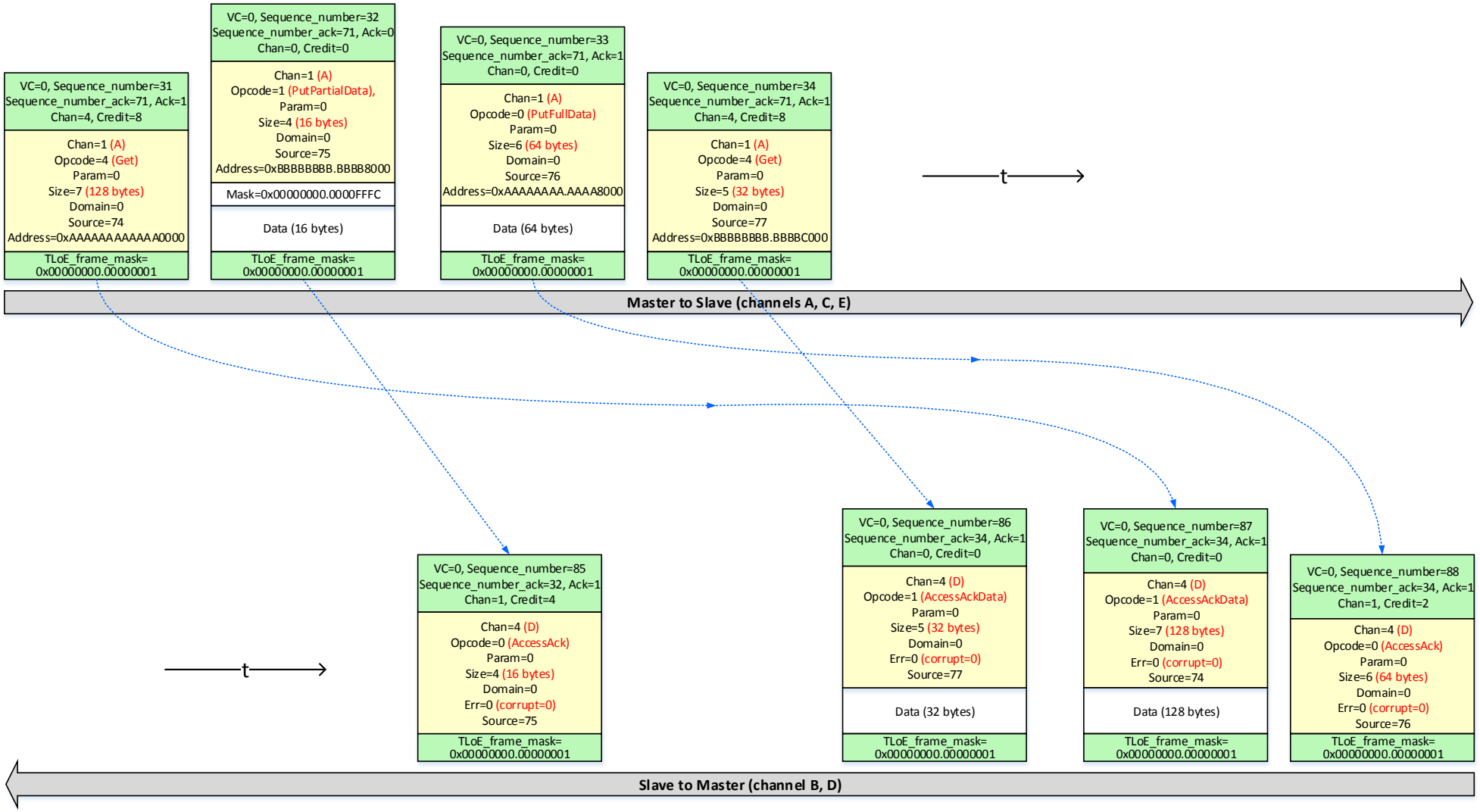

Figure 33 – Operation example 1: Access operations only (TL-UL, TL-UH)

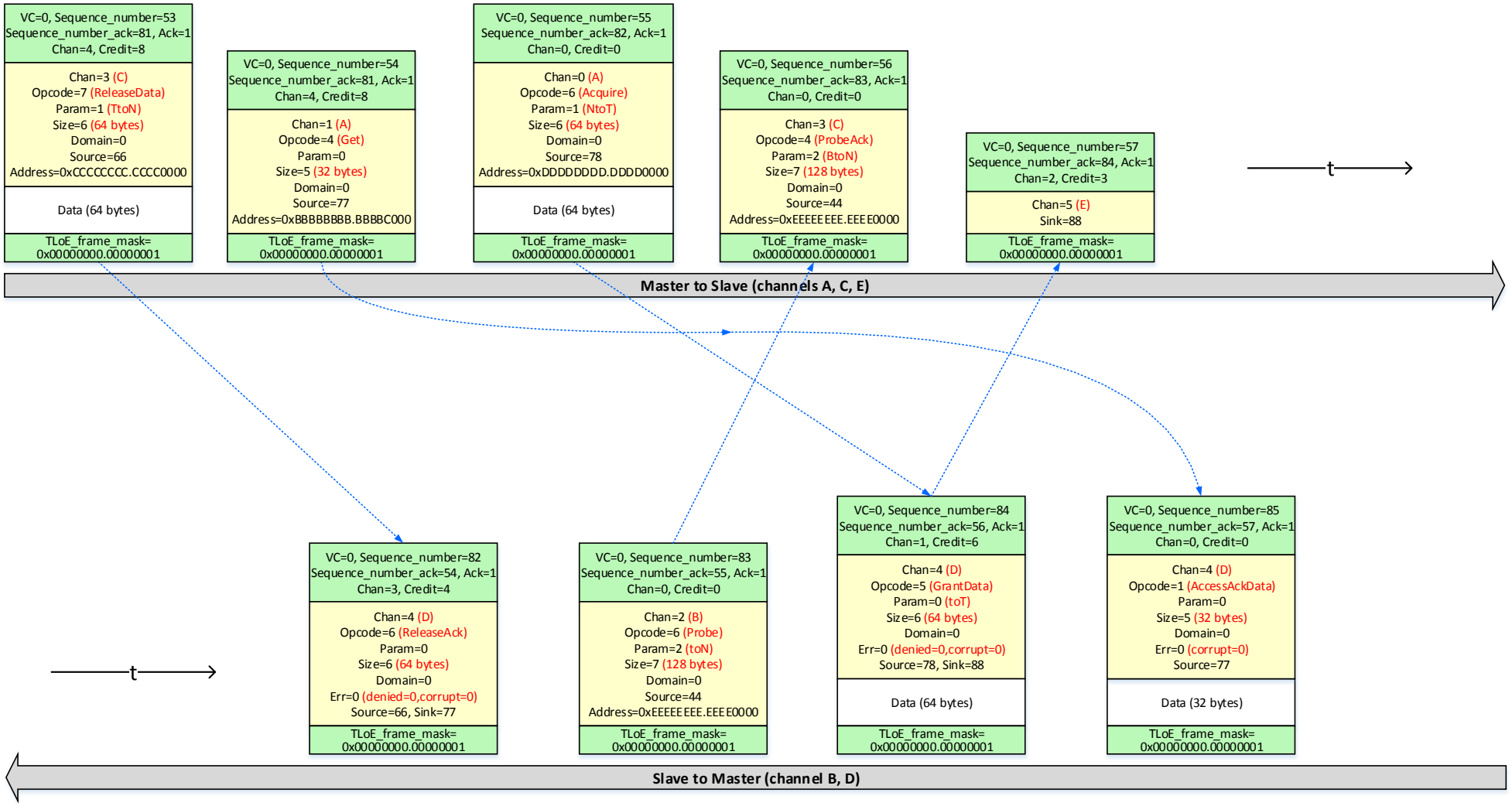

Figure 34 – Operation example 2: Access and Transfer operations (TL-C)

Annex B: **Implementation considerations**

Figure 35 shows the TLoE adapter functional elements used in the following examples.

The Retransmit buffer (mandatory) stores the packets that have been transmitted over a large window of time (~RTT).

The Transaction ID remap table (optional) allows for compression of the size of the transaction IDs carried on request messages.

The Outstanding requests table (optional) is used to avoid TileLink protocol violations; if a remote responder node is down (detected via timeout), the TLoE adapter must generate response messages for all the outstanding request messages. This table is only needed if the system needs to avoid bringing the local node down in case of a remote node failure.

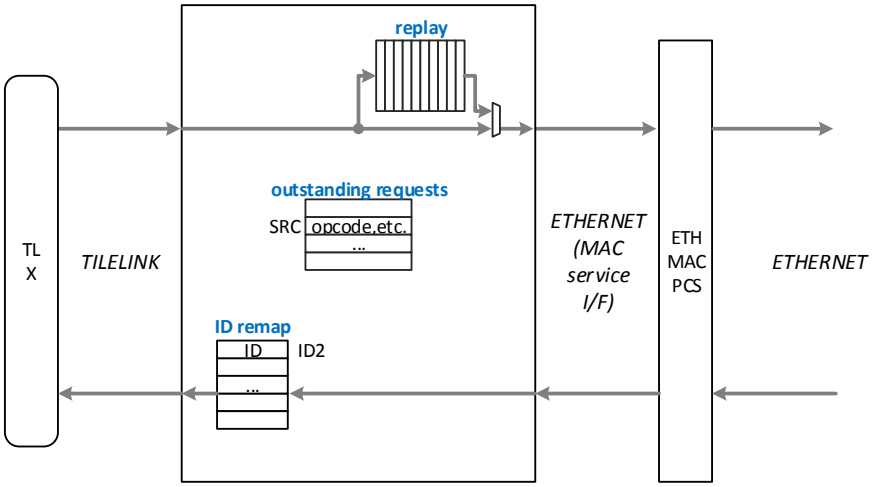

Figure 35 – TLoE adapter functional elements

B.1 Example 1: Get/Put (A→D) or Release (C→D) operation

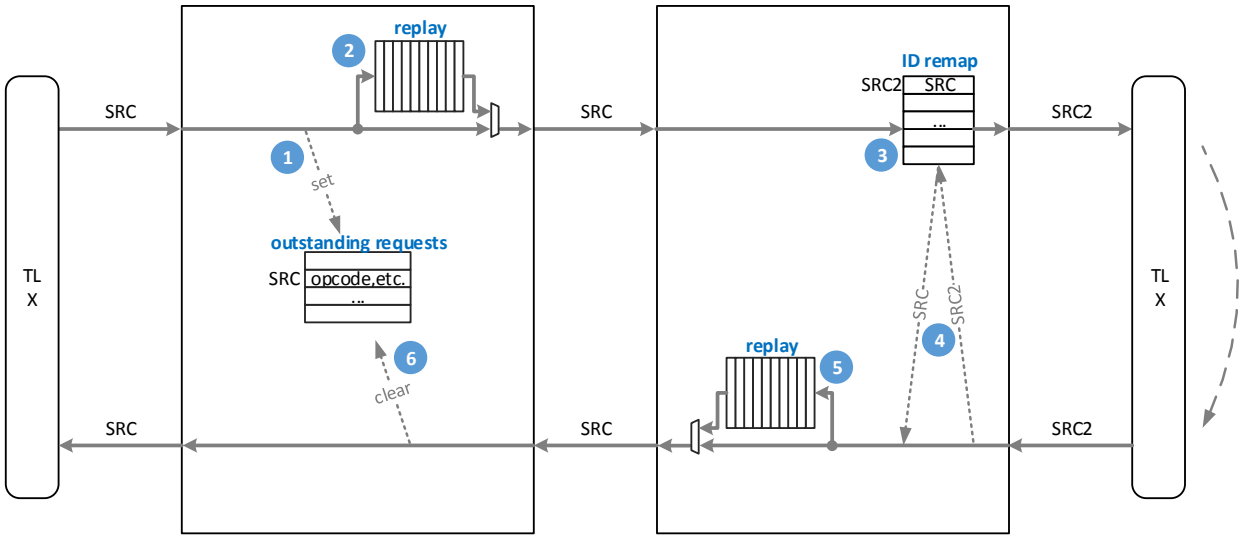

Figure 36 – Example 1: Get/Put (A→D) or Release (C→D) operation

1. The request is flagged as “outstanding”
2. The frame is stored into the retransmit buffer
3. The source ID is remapped
4. The source ID is remapped back to its original form
5. The frame is stored into the retransmit buffer
6. The outstanding flag associated to this operation is cleared.

Note that the Access (channel A) and Release (channel C) requests are “NAKed” by setting `d_denied` on the generated AccessAck / ReleaseAck messages (channel D).

A Transfer probe operation (B→C) operates on a similar way to the Access (A→D) and Transfer release (C→D) operations. The main differences are detailed below:

- The Probe request is routed by address (channel B) and the response is routed by ID, so normally the TLoE adapter would need to access the “outstanding requests” table using the address. However, for the TileLink over Ethernet application, the Slave can set `b_source`=0, so the TLoE adapter can overwrite `b_source` with non-zero values that can be used to access the “outstanding requests” table.

B.2 Example 2: Acquire (A→D→E) operation

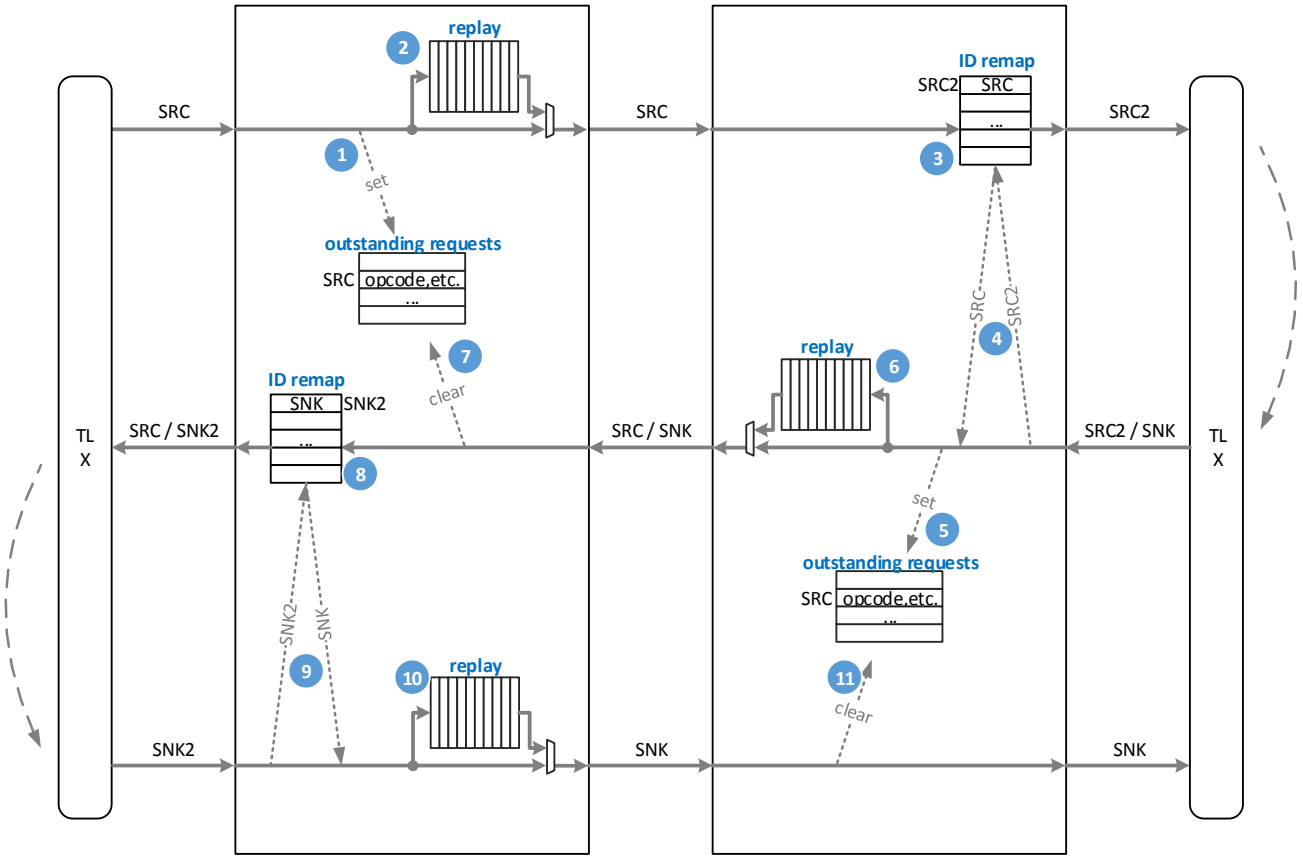

Figure 37 – Acquire (A→D→E) operation

1. The request is flagged as “outstanding”
2. The frame is stored into the retransmit buffer
3. The source ID is remapped
4. The source ID is remapped back to its original form
5. The D channel messages is flagged as “outstanding”
6. The frame is stored into the retransmit buffer
7. The outstanding flag associated to channel A’s request is cleared
8. The sink ID is remapped
9. The sink ID is remapped back to its original form
10. The frame is stored into the retransmit buffer
11. The outstanding flag associated to this operation is cleared.

Normally, messages are “NAKed” using the `*_denied` signal. For example, a NAK is generated for an outstanding Acquire request (channel A) by setting the `d_denied` signal on the generated Grant message (channel D).

Channel E does not have an `e_denied` signal. Instead of generating a NAK, a normal GrantAck message is generated.