# RoE mappers

[///Editor’s note: This clause defines one or more mappers to/from existing radio framing formats to/from RoE native transport encapsulation format.]

## Overview

[///Editor’s note: This subclause defines a mapper to/from CPRI v6.1 framing to/from RoE transport. It captures both structure agnostic and structure aware cases. Proposal to handle 8B/10B and 64B/66B CPRI PHYs as separate mappers. ]

## Mapper definition for constructing RoE data packet payload

[///Editor’s note: Need define scope here. I believe this is for both struture aware and agnostic. but not for tunnelling or native?]

This sub-clause defines a simple mapper description language that is used to describe the content of the RoE packet payload field, how the content of the CPRI packet is constructed to and from basic frames, and how the CPRI control words are separated from the data payload. The description language is specific to structure agnostic and structure aware mappers. Similar type description language is also provided for processing the CPRI control words in a separate sub-clause.

The content of a CPRI basic frame is divided into **RoE.numContainers** bit fields (i.e., containers). The container description always fully describe a complete CPRI basic frame. The RoE packet payload is then constructed based on the data extracted from the CPRI basic frames. that again can be repeated **RoE.numSegment** times. A single **RoE.container[0..RoE.numContainers-1]** array entry is described as below:

**{**

**.flow\_id**

**.ctrl**

**.lenSkip**

**.lenContainer**

**.modulo**

**.index**

**}**

Multiple CPRI basic frames are grouped together using a segment construction. The segment is described using **RoE.Segment** that has a similar content as the **RoE. container**:

**{**

**.num**

**.lenStuff**

**.lenSample**

**.interleave**

**}**

The container and the segment descriptions work for both extracting data from CPRI stream and describing the construction of the RoE payload field as well. The container and the segment constructs/variables have both have the same “**RoE**” root that binds the mapping description together. Table 7 lists the top level variables under the mapper root “**RoE**” variable.

’

Table 7- Variables under “RoE" branch

|  |  |  |
| --- | --- | --- |
| **Variable** | **Bits** | **Description** |
| .numContainers | 8 | The size of the **.container** array. |
| .container | -- | An array of **.container** structures. |

### Container definition

The “**.flow\_id**” identifies to which RoE flow or group of flows this container belongs to. Typically the “**.flow\_id**” equals to an antenna carrier that is placed into separate RoE packets/flows. The “**.ctrl**” defines whether this contrainer is about control (1) or data payload (0). This selection can be used, for example, output data selectively to control or data RoE packets flows. In a case “**.ctrl**” is set to 1 the “**.flow\_id**” should be set to NIL flow\_id.

The “**.lenSkip**” describes the number of unused bits and the “**.lenContainer**” the number of actual payload bits per each container. Note that the “**.lenSkip**” bits are only affective when extracting/storing data from/to some other source than RoE payload field such as the CPRI basic frame. When containers are stored into or read from the RoE payload field “skip bits” are not written or read.

The “**.modulo**” allows skipping containers and skipped containers are handled in a same way as “**.lenSkip**” bits. The modulo operation is applied to a sequence of input data that is counted from 0 to CPRI basic frame index. The container to select is matched comparing the “**.index**” to the output of the modulo operation. For example, to skip every second input container set the modulo to 2 and the index accordingly. To turn off modulo logic set the modulo to 0 (the same effect is also achieved by setting the “**.modulo**” to 1 and “**.index**” to 0).

The above scheme allows constructing rather compex payload fields out of a CPRI basic frame as well as very simple ones. The container definitions are per direction i.e. there may be different values for transmit and receive directions. A CPRI equivalent for a single container is basically an AxC Container. A series of container definitions can also be used to describe roughly equivalents of an AxC Group and an AxC Container Group. Table 8 summarizes the container variables and Figures 9 to 13 in sub-clause 7.2.4 show examples of RoE payload contruction in a case of mapped CPRI. The container definitions shall always describe one complete CPRI basic frame.

Table 8 - Variables under "RoE.Container" branch

|  |  |  |
| --- | --- | --- |
| **Variable** | **Bits** | **Description** |
| .flow\_id | 8 | The flow\_id this container belongs to. For flows that do not belong to a specific flow but are processed separately, like CPRI control words, shall set the value to NIL flow\_id. |
| .ctrl | 1 | Type of the container: 0 for data payload and 1 for control. |
| .lenSkip | 8 | The number of bits of reserved bits in front of the extracted segment. |
| .lenContainer | 10 | The size of the container in bits:   * NAxC – for CPRI maping method #1. * NAxC – 2 \* M (or M’) – for CPRI mapping method #3. |
| .modulo | 9 | Values from 0 to 256. Value 0 turns of the modulo logic. |
| .index | 8 | Index to match after the modulo operation. Valid values from 0 to **.modulo**-1. |

### Segment definition

[///Editor’s note: Need to describe uni/bi directionality]

The “**RoE.Segment.num**” implicitly defines the amount of data collected before starting to construct one or more RoE packets or CPRI basic frames. The “**RoE.Segment.num**” defines the number of collected samples (i.e., containers containing sample data), basic frames or bits depending on the used mapper. The “**.lenStuff**” defines the number of padding bits at the head of a collected sample data (i.e., a CPRI Container Block) and is only meant for container definitions for CPRI mapping method #1 (see CPRI v6.1 Section 4.2.7.2.5.) and shall be set to 0 otherwise.

The “**.lenSample**” describes the size of one sample. In a case of I/Q samples the total size of the I/Q sample is twice the “**.lenSample**”. The CPRI equivalent for “**.lenSample**” is M (or M’). In essence if the mapper is interested in individual samples within a CPRI AxC Container Block then “**.lenSample**” shall be non-zero and set to zero (0) otherwise.

Samples in a RoE packet follow the bit level ordering described in sub-clause 4.13.5 meaning the bit interleaved samples in CPRI AxC Containers have to be translated to RoE bit ordering and vice versa. The “**.interleave**” variable controls the I/Q sample bit interleraving translation. The interleaving translation is disabled when the variable is set to zero (0) and set to one (1) when translation between CPRI I/Q sample bit interleaving and RoE sample bit ordering is enabled. Table 9 summarizes the segment variables,

Table 9 - Variables under "RoE.Segment" branch

|  |  |  |
| --- | --- | --- |
| **Variable** | **Bits** | **Description** |
| .num | 12 | The number of samples, containers or CPRI basic frames needed for one or more RoE packets and vice versa. |
| .lenStuff | 32 | The number of stuffing bits in a container block. |
| .lenSample | 5 | The size of one sample in bits. The value 0 means the “**.lenSample**” information is not used. |
| .interleave | 1 | Controls the I/Q sample bit interleaving translation. The value 0 means the translation is disabled and the value 1 means the translation is enabled. |

### Other mapper definitions and variables

[//JiK comment: do we need to define offset values for mappers? Those would allow describing where exactly in the RoE flow the radio sample stream starts? The offset values would be used only at the very beginning of the flow setup]

### Payload and mapper examples

Figure 9 shows an example of a CPRI basic frame description, and how RoE container definitions are used to describe a “Packed Position” AxC Container mapping and how to separate the control words from sample data containing containers. 

Figure 9 – Packed Position AxC Container mapping in the IQ data block

Figure 10 shows an example of a CPRI basic frame, and how RoE container definitions are used to describe a “Flexible Position” AxC Container mapping and how to separate the control words from sample data containing containers. It should be noted that a RoE container “data” length can be 0 while the reserved bits can be non-zero.



Figure 10 – Flexible Position AxC Container mapping in the IQ data block

Figure 11 shows an example of using the RoE container definitions to describe a CPRI mapping method #1. It should be noted that in this case there should only be one RoE container for a CPRI AxC Container Block independent of the number of samples within the AxC Container Block.



Figure 11 - Example of CPRI mapping method #1

Figure 12 shows an example of using the RoE container definitions to describe a CPRI mapping method #3 (with an assumption there are no stuffing samples). Depending on the RoE container definitions each CPRI AxC Container can be treated as an individual RoE flow or grouped into one or more RoE flows.



Figure 12 - Example of CPRI mapping method #3

Figure 13 shows an example of using a structure aware mapper with a modulo rule. The CPRI flow has AxC0 and AxC1 at rate R and AxC2 and AxC3 at rate 0.5\*R (i.e., half rate AxCs). Using the modulo rule for the “half rate” AxCs it is possible to construct CPRI basic frames where the “half rate” AxCs occupy every other CPRI basic frame. This allows transporting four AxC flows but only use three AxC Containers per CPRI basic frame. When transported over RoE there would be two RoE flows at rate R (for AxC0 and AxC1) and another two RoE flows at rate 0.5\*R (for AxC2 and AxC3). Total K\*3 samples (or rather containers with sample data) are buffered/collected before assembling/disassembling a RoE packet.



Figure 13 - Example of using modulo rules and the structure aware mapper

## Simple tunneling mapper

This document names simple tunneling mapper as “**CPRI00**”.

The simple tunneling mapper is very simple. The entire serial CPRI data stream is simply encapsulated in the payload.

### Use of sequence number

Since all frame timing, including K28.5, HFN and BFN are preserved within the fully encapsulated CPRI stream in the payload, the sequence number is only useful to detect dropped packets.

### Use of control packets

The simple tunneling mapper “CPRI00” does not use any RoE control packets.

## Structure agnostic mapper

### RoE data packet (000010b)

This document names the structure agnostic CPRI mapper as “**CPRI10**”. The mapper shall remove the 8B/10B line coding used by CPRI for line rate options 1 to 7. The mapper shall be aware of the start of the radio frame. In the context of this mapper and CPRI v6.1 specification the radio frame is the 10ms frame number, which for UTRA-FDD would be aligned with NodeB Frame Number (BFN).

The mapper extracts/stores **CPRI10.lenBasicFrame** octets from/to the CPRI stream i.e. an individual CPRI Basic Frame (BF). **CPRI10.numBasicFramesPerPacket** are stored/extracted to/from RoE packets.

Other RoE configuration parameters shall be set as follows:

1. **RoE. Segment.num**=**CPRI10.numBasicFramesPerPacket**
2. **RoE.segment.lenStuff=0**
3. **RoE.segment.lenSample=0**
4. **RoE.segment.interleave=0**
5. **RoE.numContainer**=1
6. **RoE.container[0].lenSkip**=0
7. **RoE.container[0].lenContainer**=**CPRI10.lenBasicFrame**\*8
8. **RoE.container[0].flow\_id**=a number between 1 and 255
9. **RoE.container[0].ctrl**=0
10. **RoE.container[0].modulo**=0
11. **seqNumPMax=150\*256/CPRI10.numBasicFramesPerPacket-1**
12. **seqNumPInc=1**
13. **seqNumPIncProp=1**
14. **seqNumQMax=4095**
15. **seqNumPInc=1**
16. **seqNumPIncProp=1**

[///Editor’s note: Draw example figure here.]

### Use of sequence number

The sequence number is incremented by one (1) for each sent RoE data packet and the sequence number p-counter wraps around every 256\*150/**CPRI10.numBasicFramesPerPacket** sent packets (e.g., if there are 8 BFs per RoE packet the **seqNumPMax** is 4799). When the p-counter wraps the sequence number q-counter gets incrementd by one (1). The q-counter will wrap after 4096 increments i.e., being able to cover 12 bit CPRI BFN.

### Use of RoE control packets

There are no associated control packets for the “CPRI10” mapper.

## Structure aware mapper

### RoE data Packet (000011b)

This document names the structure-aware CPRI mapper as “**CPRI11**”.

[///Editor’s note: This is what we called the “better” mapper.]

[///Editor’s note: Proposal to require that GSM and other “non-UMTS Chip” antenna carriers are already resampled to some integer divisible UMTS Chip rate within CPRI traffic before the mapper is applied.]

This subclause defines a structure-aware CPRI to RoE mapper that looks into the CPRI frame and is able to further divide its content into different components. The mapper packetizes a number of CPRI Basic Frames worth of I/Q samples/AxC Containers for one AxC into a RoE data packet payload. The mapper is equivalent to CPRI mapping method #3 without stuffing samples.

The mapper shall remove the 8B/10B line coding used by CPRI for line rate options 1 to 7. The mapper shall be aware of the start of the radio frame. In the context of this mapper and CPRI v6.1 specification the radio frame is the 10ms frame number, which for UTRA-FDD would be aligned with NodeB Frame Number (BFN).

[///Editor’s note: This is rather under specified and assumes that everything complies to some integer fraction of UMTS Chip rate.]

The mapper extracts/stores **CPRI11.lenBasicFrame** octets from/to the CPRI stream i.e. an individual CPRI Basic Frame (BF). The mapper buffers **CPRI11.numBasicFramesForRoEPacket** worth of CPRI BFs and then stores/extracts individual AxC containers to/from one or more RoE packets.

The below RoE configuration parameter example is for CPRI line rate option 3 (assuming 20MHz LTE and 2x2 MIMO) and CPRI mapping method 3 without any stuffing samples. The CPRI AxC Group size is 2 (i.e., NA=2) and the AxC Container Block contains 16 samples (i.e., NC=NA\*8) The AxC Group AxC0 will have flow\_id 1 and the AxC1 will have flow\_id 2 i.e., there will be two RoE data packet flows.The I/Q sample size is 15 bits per component. One RoE data packet will contain 64 I/Q samples i.e. 8 BFs worth of samples.

The RoE configuration parameters shall be set as follows:

1. **CPRI11.numBasicFramesForRoEPacket=8**
2. **RoE.segment.interleave=1**
3. **RoE. Segment.num**=**CPRI11.numBasicFramesForRoEPacket\*8**
4. **RoE.segment.lenStuff=0**
5. **RoE.segment.lenSample=15**
6. **RoE.numContainer**=17
7. **RoE.container[0].lenSkip**=0
8. **RoE.container[0].lenContainer**=**32 // the control word**
9. **RoE.container[0].ctrl**=**1**
10. **RoE.container[0].flow\_id=0**
11. **RoE.container[0].modulo**=**0**
12. **RoE.container[2,4,6,8,10,12,14,16].lenSkip**=0
13. **RoE.container[2,4,6,8,10,12,14,16].lenContainer**=**2\*RoE.segment.lenSample**
14. **RoE.container[2,4,6,8,10,12,14,16].flow\_id**=2 // AxC1
15. **RoE.container[2,4,6,8,10,12,14,16].ctrl**=0
16. **RoE.container[2,4,6,8,10,12,14,16].modulo**=0
17. **RoE.container[1,3,5,7,9,11,13,15].lenSkip**=0
18. **RoE.container[1,3,5,7,9,11,13,15].lenContainer**=**2\*RoE.segment.lenSample**
19. **RoE.container[1,3,5,7,9,11,13,15].flow\_id**=1 // AxC0
20. **RoE.container[1,3,5,7,9,11,13,15].ctrl**=0
21. **RoE.container[1,3,5,7,9,11,13,15].modulo**=0
22. **seqNumPMax=150\*256/CPRI11.numBasicFramesPerPacket-1**
23. **seqNumPInc=1**
24. **seqNumPIncProp=1**
25. **seqNumQMax=4095**
26. **seqNumPInc=1**
27. **seqNumPIncProp=1**

### Use of sequence numbers for RoE pkt\_type 000011b

The sequence number is incremented by one (1) for each sent RoE data packet and the sequence number *p-counter* wraps around every 256\*150/**CPRI11.numBasicFramesPerPacket** sent packets (e.g. if there are 8 BFs per RoE packet the **seqNumPMax** is 4799). When the *p-counter* wraps the sequence number *q-counter* gets incrementd by one (1). The *q-counter* will wrap after 4096 increments i.e., being able to cover 12 bit CPRI BFN.