VoATM-VoIP Interworking on C-5.
Architecture & Design Document

Version 1.0

Motorola India Electronics Pvt. Ltd .
"The Senate"
No. 33A, Ulsoor Road
Bangalore - 560042
INDIA

Revision History

Date	Revision	Description	Author
18 September 2001	0.1	Reviewed RTP/UDP Chapter added.	Vineet
04 October 2001	0.2	Chapter 1 Document Introduction, and Chapter 2 System Description and Architecture added.	Vineet
09 October 2001	0.3	Chapter 3 Existing ATM Layer and its Adaptation, Chapter 4 Existing AAL2 CPS Layer and its Adaptation, Chapter 5 Existing Ethernet and IP Layer and its Adaptation and Chapter 9 C-5 Executive Processor, Resources and Timers Added.	Murali
10 October 2001	0.4	To incorporate AAL2 adaptation Review Comments	Murali
11 October 2001	0.5	AAL2 SSCS Rx functionality and AAL2 to RTP Mapping module design chapter added.	Vineet
15 October 2001	0.6	AAL2 SSCS Rx functionality and AAL2 to RTP Mapping module design chapter modified after review.	Vineet
15 October 2001	0.7	ATM adaptation chapter added after review.	Murali
17 October 2001	0.8	1. Changes made in XP and resources chapter after review. 2. SONET message descriptor is added to the ATM chapter. 3. Table id field removed from the AAL2 Rx CP init descriptor from XP, in AAL2 chapter.	Murali
18 October 2001	0.9	AAL2 SSCS Tx functionality and RTP to AAL2 Mapping module design chapter added. 1. Existing Ethernet and IP Layer and its Adaptation chapter modified after review comments. 2. XP init descriptors for IP and AAL2 to RTP mapping CP are modified.	Vineet Murali
19 October 2001	1.0	1. AAL2 SSCS Tx functionality and RTP to AAL2 Mapping module design chapter modified after review. 2. System Description and Architecture chapter modified for review comments. 3. AAL2 SSCS Rx functionality and AAL2 to RTP Mapping module design chapter re-organized for heading numbers, and one testing of G.729B frame added. 4. UDP/RTP chapter re-organized for its font size, and its existing figure is removed. 5. callPresentFlag added in the HTK table for RTP to AAL2 mapping module. 6. Feature Requirements Trace Matrix cross checked and added. 7. This Version is baselined.	Vineet

Table of Contents

1. Introduction

This document is architecture and design document for the VoATM to VoIP Interworking Gateway reference application, which will be running on the Motorola C-Port's C-5 network processor.

1.1 Scope

This is the only single reference design document for the application, which will be used for the development purpose.

1.2 Reader Level

Readers of this document, who are the developers of the application, must be familiar with the relevant domain, standards, and the scope of the application.

1.3 Companion Documentation

Companion document for this design document is Feature Requirements and Architectural Specifications (FRAS) for the application, which outlines the functional specifications and broad requirements of the system.

1.4 Naming Convention

VoATM-VoIP Interworking Function (IWF), Media Gateway, and Gateway terms refer to the same system, unless specified otherwise.

1.5 References & Standards

- ITU-T I.361 B-ISDN ATM Layer specification.
- ITU-T I.363.2 B-ISDN ATM adaptation Layer specification: Type 2 AAL.
- ITU-T I.366.2 (11/2000) AAL Type-2 service specific convergence sub layer for narrow-band services.
- RFC 1889, RTP: A Transport Protocol for Real Time Applications.
- RFC 1890, RTP Profile for Audio and Video Conferences with Minimal Control.
- RFC 768, User Data gram Protocol.
- RFC 792, Internet Control Message Protocol.
- C-PORT, C-5 DCP Architecture Guide.
- af-vtoa-0113.000, ATM Forum - ATM Trunking using AAL2 for Narrowband services.
- af-vmoa-0145.000, ATM Forum - Voice and Multimedia Over ATM -Loop Emulation Service Using AAL2.
- VoATM-VoIP Interworking on C-5; Feature Requirements & Architectural Specifications.
- draft-ietf-avt-profile-new-11.txt. (RTP Profile for Audio and Video Conferences with Minimal Control), Expires: January 2002.
- draft-ietf-avt-rtp-cn-03.txt. (RTP Payload for Comfort Noise), July 2001.

1.6 Acronyms

AAL2	ATM Adaptation Layer 2
ADPCM	Adaptive Pulse Code Modulation
ATM	Asynchronous Transfer Mode
BMU	C-5 Buffer Management Unit
CDS	C-Port Development System
CID	Channel Id (AAL2 header field).
CP	C-5 Channel Processor

CPCI	Compact PCI.
CPS	Common Part Sub layer
CS-ACELP	Conjugate Structure – Algebraic Code Excited Linear Prediction
DSL	Digital Subscriber Line
DSP	Digital Signal Processing
DTMF	Dual Tone Multi Frequency
IAD	Integrated Access Device
ITU	International Telecommunication Union
Map-1	AAL2 to RTP Mapping Module in this document
Map-2	RTP to AAL2 Mapping Module in this document
MF	Multi frequency
OC-3c	Optical Carrier Level-3, 155Mbps
PCM	Pulse Code Modulation
PCMA	PCM-A Law
PCMU	PCM-µ Law
PVC	Permanent Virtual Circuits (ATM)
QMU	C-5 Queue Management Unit
RFC	Request For Comments.
RTCP	Real time Transport Control Protocol
RTP	Real time Transport Protocol
SDP	Session Description Protocol
SID	Silence Insertion Descriptor (I.366.2)
SONET	Synchronous Optical Network
SSCS	Service Specific Convergence Sub layer
TLU	C-5 Table Lookup Unit
UDP	User Data gram Protocol
UUI	User-User Indication
VAD	Voice Activity Detection
VC	Virtual Circuit
VCI	Virtual Channel Identifier
VoATM	Voice over ATM.
VoIP	Voice over IP.
VPI	Virtual Path Identifier
XP	C-5 Executive Processor

2. System Description & Architecture

2.1 Overview of the System

VoATM-VoIP Media Gateway is mechanism to provide call interworking between the subscribers who are connected to ATM and IP networks. ATM Adaptation Layer number 2 (AAL2) is selected for transport on the ATM network, while Real Time Transport Protocol (RTP) is the transport protocol on the IP network.

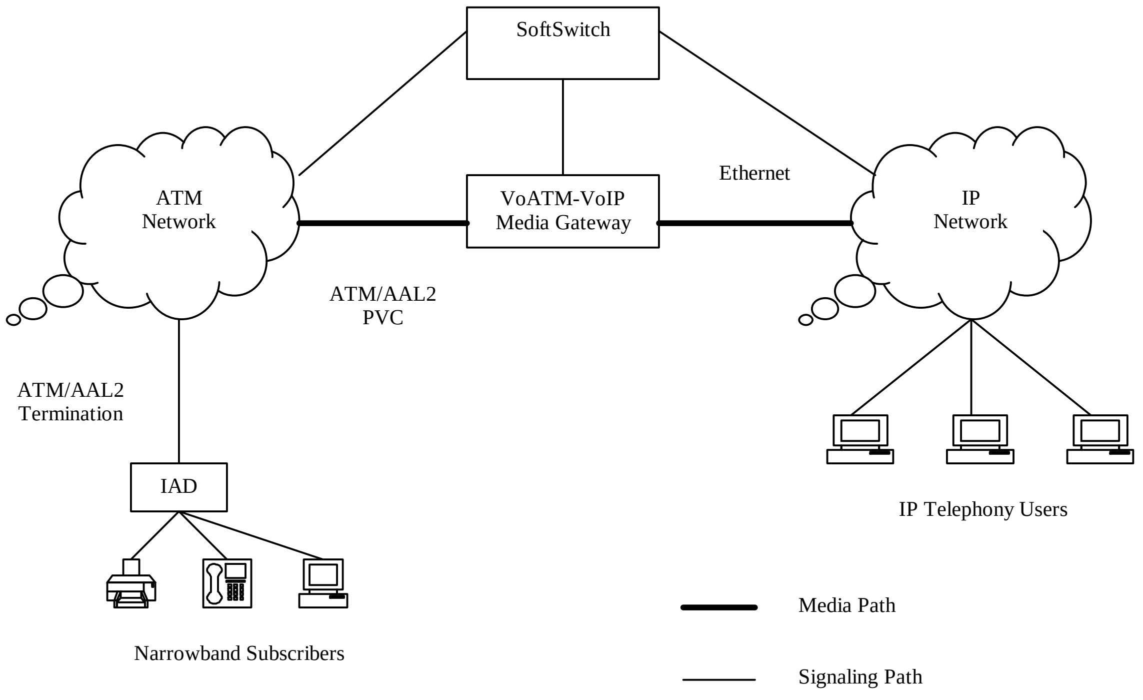

Figure 1 VoATM-VoIP Media Gateway

2.1.1 AAL2 Common Part and Service Specific Sub layers:

AAL2 CPS provides multiplexing of different users on to a single ATM connection. The packets from these users are multiplexed into the payload of ATM cell stream at the sending side and de-multiplexed at the receiving side. CPS packet has a header of 3 octets and a payload of variable length up to a maximum value of 64 octets (default maximum length is 45). The header consists of the following four fields:

Channel Identifier (CID, 8 bits): Identifies the AAL2 channel (CPS user). AAL2 channel is bi-directional (i.e. same CID is used for both directions). The value “0” is not used and values “1” to “7” are reserved.
Length Indicator (LI, 6 bits): Indicates the length of the CPS packet payload. Default maximum length is 45 octets and can be set to 64.

User-to-User Indication (UUI, 5 bits): It is used to convey specific information transparently between the CPS users (i.e., between SSCS entities or between Layer Management), and to distinguish between the SSCS entities and Layer Management users of the CPS. The 5-bit UUI field provides for 32 codepoints, “0” ... “31”. Codepoints “0” ... “27” are available for SSCS entities, codepoints "28"… 30" are reserved for future standardization, and code point "31" is reserved for OAM.

<u>Header Error Control (HEC, 5 bits)</u>: Detects bit errors in the header by a 5 bit CRC.

Packing/Multiplexing and Unpacking/De-multiplexing of CPS packets are done using Start Field (STF) as shown in figure 8. The STF consists of **-** 6 bit Offset Field (OSF), 1 bit Sequence Number (SN) and a Parity bit (P). The OSF points to the start of the first CPS packet (or to start of PAD field in the absence of any CPS packet start) in CPS PDU. The value 47 indicates there is no start (i.e. whole CPS PDU is part of a CPS packet). Values greater than 47 are not allowed. The SN is used to detect the lost ATM cell and the parity bit is used to detect errors in the STF.

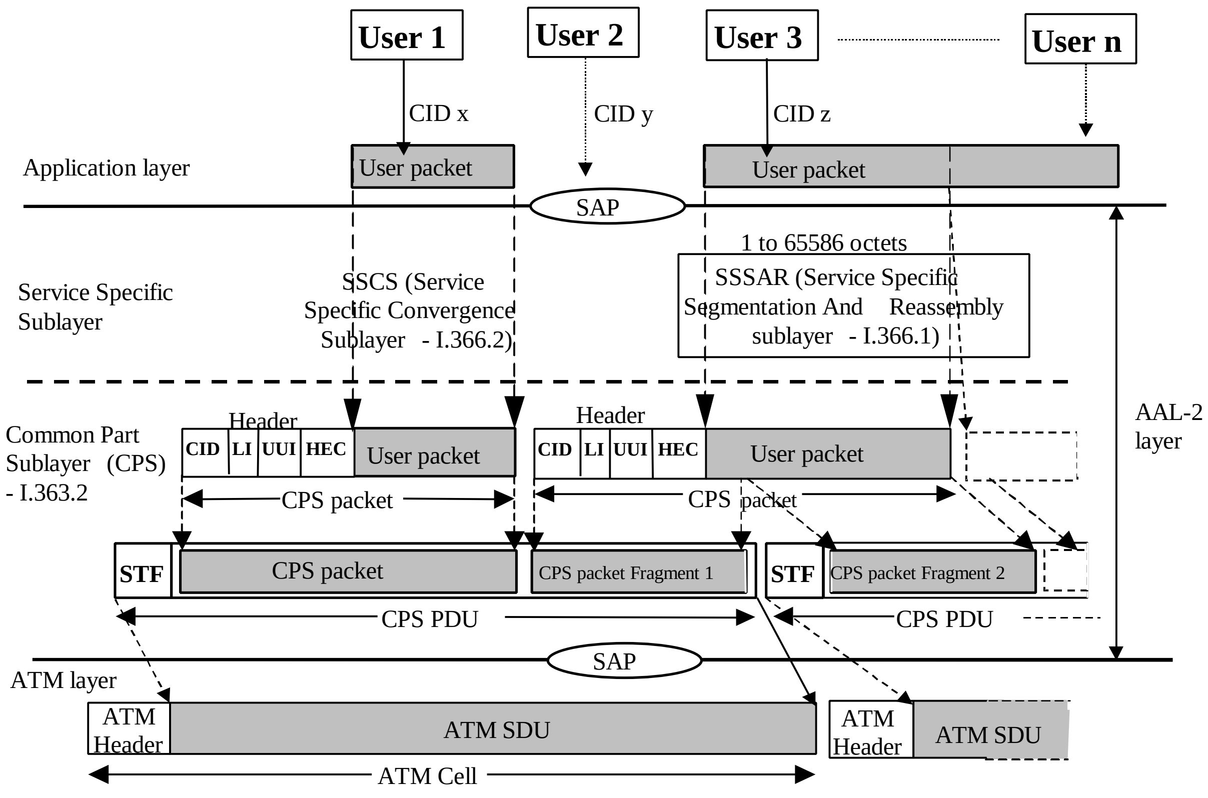

Figure 2 ATM/AAL2 Protocol structure

<u>AAL2 SSCS:</u> is used to carry the information content of one narrow band call over each AAL2 connection. As specified in the I.366.2, UUI codepoints from "0"…"15" is used for formatting the audio packets.

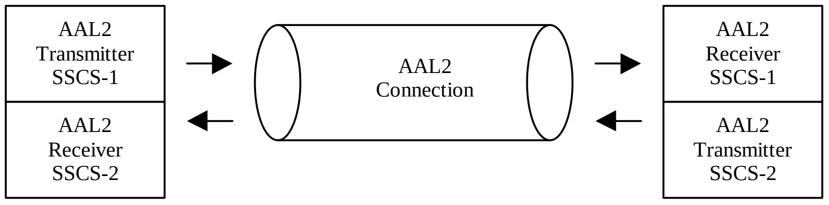

Figure 3 Receiver/Transmitter Connected to the AAL2 termination

I.366.2, Annexure P also specifies the predefined encoding format profiles, which are identified using a unique identifier.

Identifier	**Description of Profile**
0	Not used
1	PCM-64
2	PCM-64 and silence
3	ADPCM and silence
4	G.728 with higher efficiency
5	G.728 with lower delay
6	G.729 with higher efficiency and G.726 for voice band data
7	G.729 with lower delay
8	G.729 with lower delay and G.726-32 for voice band data at lower rates
9	G.729 with lower delay and G.726-40 for voice band data at higher rates
10	G.729 with full variable bit rates
11	AMR
12	G.723
13	PCM 64 kbits/s and ADPCM 32 kbits/s
14-255	Reserved for future ITU-T assignment

The above-mentioned identifiers will be referred to as profile identifier in the document. By making reference to the profile identifiers transmitter and the receiver can agree on one of the major operating parameters of the SSCS. According to I.366.2, an encoding profile is categorized within an AAL2 packet with the following characteristics:

(i) Profile entry index.
(ii) UUI code point range (0-15 for audio encoding).
(iii) Packet length in octets.
(iv) Description of the algorithm.
(v) Number of SDUs in the packet (M).
(vi) Packet time in ms.
(vii) Sequence number interval in ms.

Example #1: generic PCM (G.711 64Kbps) formatting for AAL2 SSCS (predefined profile identifier # 1) can be described using the above parameters as follows:

Profile entry index	UUI codepoint range	Packet length (octets)	Description of algorithm	M	Packet time (ms)	Sequence number interval (ms)
0	0-15	40	PCM, G.711-64, generic	1	5	5

Example #2: G.723.1 formatting for AAL2 SSCS (profile identifier # 12) can be described using the above parameters as follows:

Profile entry index	UUI codepoint range	Packet length (octets)	Description of algorithm	M	Packet time (ms)	Sequence number interval (ms)

0	0-15	24	G.723.1-6.4	1	30	5
1	0-15	20	G.723.1-5.3	1	30	5
2	0-15	4	G.723.1 SID	1	30	5

Packet length in octets is encoding specific. For example G.711 PCM coder generates one octet every 125 us, so 40 such octets are packed in one AAL2 packet to represent 5 ms of encoded audio data.

Sequence number interval specifies the minimum unit of UUI sequence value. For all the encoding types, except AMR it is 5 ms. For example, consecutive G.711 AAL2 packets will be having UUI codepoints like 0, 1, 2 ….15, because their packet time and sequence number intervals are same. But, consecutive G.723.1 AAL2 packets will be having UUI codepoints like 0, 6, 12, 2, 8 etc (modulo 16), because one G723.1 AAL2 packet is of 30 ms, which in turn embeds 6 sequence number intervals in one AAL2 packet.

Service data units (SDUs) for audio are defined in relation to the profile of encoding formats adopted on a given AAL type 2 connection. One SDU, depending upon the encoding type may contain one or more EDUs.

Audio Packet = M * SDU, SDU = N_1 * EDU, SDU time = N_2 * Sequence Number Interval.
There is no direct relationship between the EDU and the sequence number interval.
N1 and N2 are encoding specific constants defined in I.366.2 in the corresponding annexure.

2.1.2 Real Time Transport Protocol:

RTP packet along with header and its payload comes within a UDP datagram. The different fields of the RTP header are shown in the figure below:

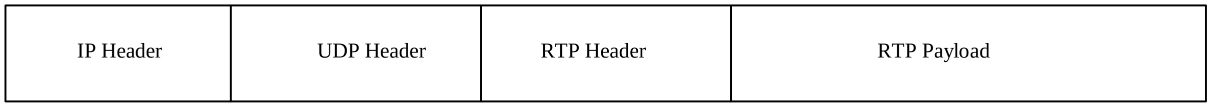

Figure 4 RTP Header & Payload as UDP Payload

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5 RTP Header (RFC 1889)

- RTP version 2 (V=2) will be used, as defined in RFC 1889.
- Padding bit (P) indicates padding added to the payload.
- Extension bit (X) indicates whether header extension is used or not.
- CSRC Count (CC) contains the contributing source count. This field is used while conferencing etc. to tell the number of contributors.

- Marker bit (M) will be normally set to 0, after silence, when audio packets start coming it will be 1, indicating starting of the media.
- Payload type (PT) will identify the encoding type carried by the RTP packet.
- Sequence number is used to sequence the media packets in the order of generation.
- Time stamp will be generated starting from any random value. In this context, for all the calls it will be generated starting from 0, and will incremented by 8 samples /ms * RTP packet time for the following encoding types: G.711, G.723.1, G.726, G.728, and G.729. For example: for 30 ms RTP packet, RTP time stamp will be increased with a value of 240 (8*30) every RTP packet time. (RFC 1890)
- SSRC identifier is a the synchronization source identifier and contains a 32 bit random value.
- CSRC identifier list represents contributing sources to the media session. Maximum 15 sources can be identified.

2.2 Media translation concept

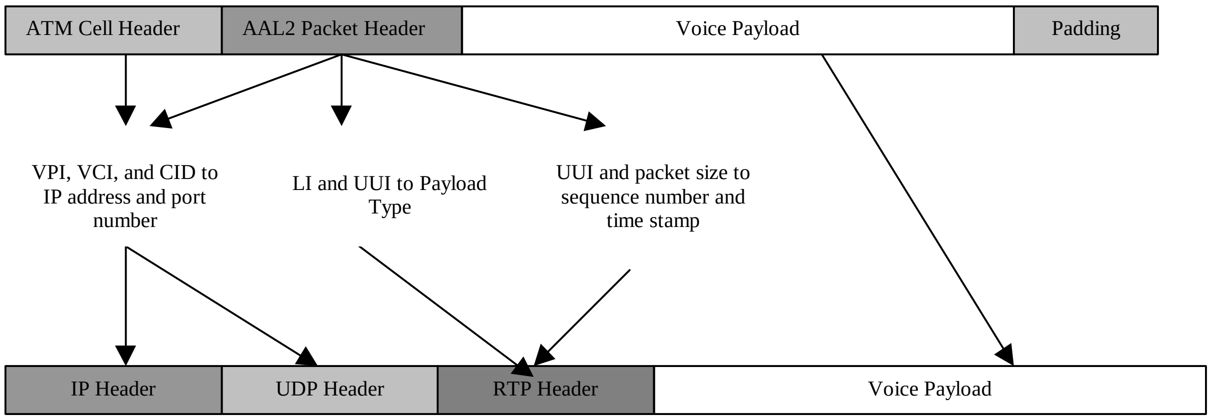
ATM Cell Header
AAL2 Packet Header
Voice Payload
Padding
VPI, VCI, and CID to IP address and port number
LI and UUI to Payload Type
UUI and packet size to sequence number and time stamp
IP Header
UDP Header
RTP Header
Voice Payload

Figure 6 Media Translation between AAL2 and RTP

VPI and VCI values in the ATM header, and the CID value in the AAL2 header uniquely identifies a narrowband subscriber connected to a ATM/AAL2 termination. On the other side, IP address in the IP header, and UDP port number in the UDP header uniquely identifies a RTP session on a IP telephony user terminal. These pairs are matched to identify the end users.

RTP header contains the payload type field, which identifies which encoding type data is being carried by RTP session, on the AAL2 side, UUI codepoints range and packet length defines the payload type.

RTP header contains the time stamp field which tells the RTP packet size ,on the other side, sequence number interval, as described in the above sections, and UUI range tells the AAL2 packet time.

2.3 Scope of the reference application

The reference application will support the following:

- One OC-3c interface, and two fast Ethernet (100Mbps) interfaces.
- Maximum of 1K AAL2 PVCs terminations.

- Maximum of 8K unique conversations. Maximum of 8K/3 conversations per 1K/3 PVCs.
- Audio services only.
- Support for the following encoding types: G.711, G.723.1, G.726, G.728, and G.729, which corresponds to the profile identifiers range 1 to 13, as in Annexure P of I.366.2.
- Only one audio session per call, unicast towards IP side, addressing one audio endpoint on ATM/AAL2 termination and vice-versa.
- Symmetric encoding types on the both the endpoints.
- Only data plane will be supported, that is conversion of AAL2 media format to RTP media format and vice-versa.
- Maximum receive capability of the gateways would be 30 ms of RTP packet for all the encoding types.

The reference application will not support the following:

- Support for frame mode data, circuit mode data, fax, CAS, dialed digit service etc.
- Handling of out of sequence IP packets. This issue will be addressed using Q-5 queue processor.
- AAL2 and IP telephony call and control signaling plane.
- Asymmetric encoding types on the both the endpoints using dynamic codec negotiation.
- Support for the following encoding types: G.722, G.727, and AMR.
- Handling of non AAL2 ATM cells. These cells will be dropped by the gateway.

The following assumptions are made:

- IP telephony users and AAL2 endpoint audio devices are capable for discontinuous transmission (DTX) and, VAD (Voice Activity Detection) and CNG (Comfort Noise Generation) algorithms are implemented for all the supported encoding types.

- Signaling specific information, which is required to control the data plane, is provided through TLU tables or through user configuration from XP/host.

- Gateway will not be able to find out the MAC destination address dynamically, because ARP is not be fully implemented. So, the requirements for its deployment would be:

    - In the field, both of the Ethernet ports of the gateway will be connected to the same and only one next hop router. MAC address of this router will be configured in the gateway before boot up.
    - For the system test lab setup, IP call generator, and the gateway will be on the same router's zone. So that the router cannot send ICMP re-direct message to gateway to find out the IP call generator.

2.4 C-5 CP Cluster Information

CP cluster information for the VoATM-VoIP gateway implementation on C-5 is shown in the figure below. The details of this cluster are written in the following table. The ordering of p, q, r, and s are physical board dependent, e.g. which physical ports are connected to which CPs:

Cluster #	CPs allocated to this cluster
Cluster-p	CP-p0: For ATM receiver and transmitter. CP-p1, p2, and p3: For AAL2 CPS transmitter.
Cluster-q	CP-q0, and q1: AAL2 CPS receiver. CP- q2, and q3: RTP to AAL2 direction Mapping module (Map-2).
Cluster-r	CP-r0: RTP to AAL2 direction mapping module (Map-2). CP-r1, r2, and r3: AAL2 to RTP direction mapping module

	(Map-1).
Cluster-s	CP-s0, and s1: RTP/UDP formatter/de-formatter. CP-s2, and s3: IP and Ethernet receiver and transmitter.

Figure 7 VoATM-VoIP Gateway CP Cluster Information

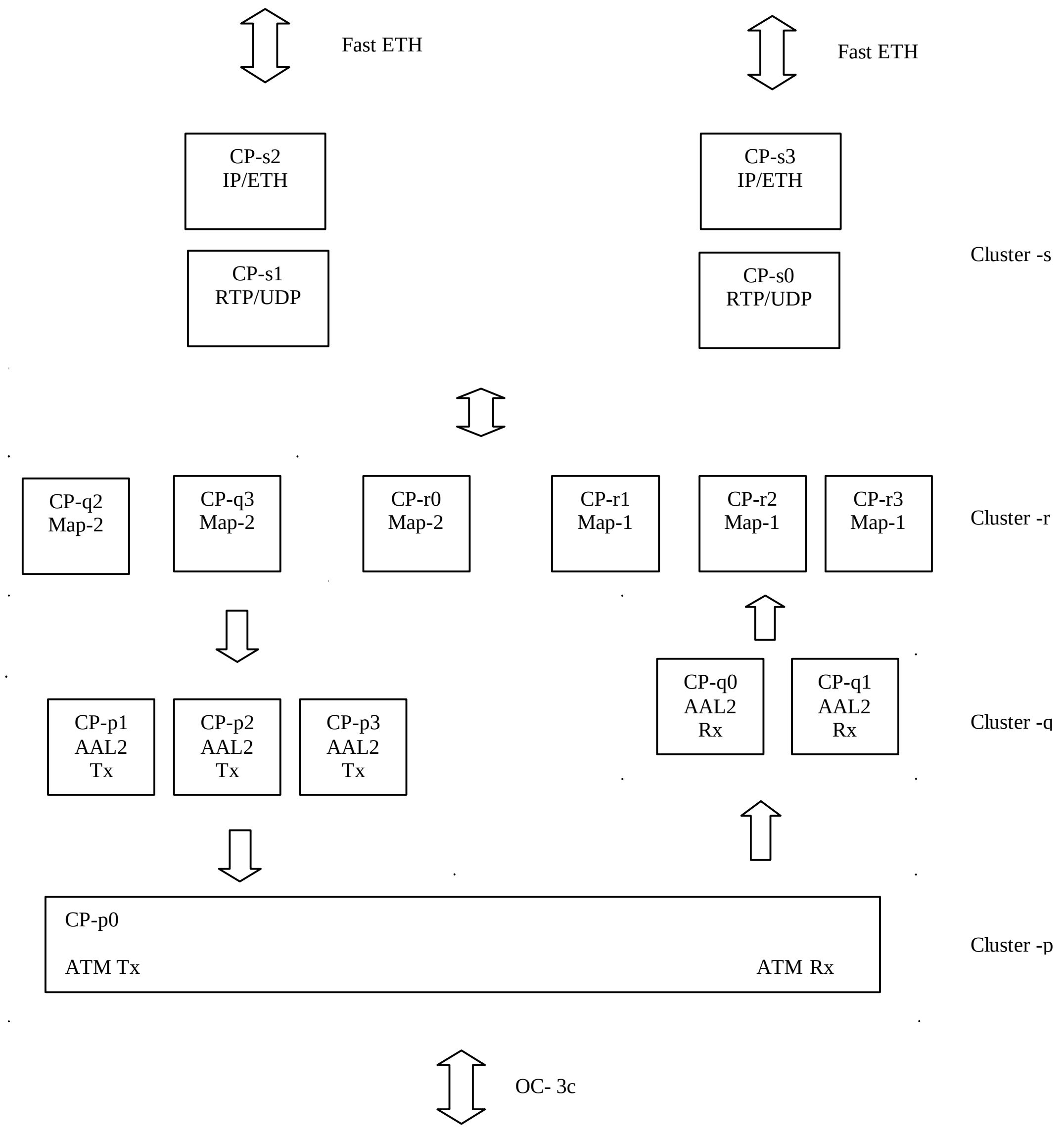

Figure 8 VoATM-VoIP Gateway CP Cluster Arrangement on C-5

2.5 CP: Functionality Decomposition

2.5.1 ATM Receiver and Transmitter:

For details, please refer to the ATM switch reference library documentation. Interface adaptation relevant to this application is described in the following chapters.

2.5.2 AAL2 CPS Receiver and Transmitter:

For details, please refer to the ATM/AAL2 switch reference library documentation. Interface adaptation relevant to this application is described in the following chapters.

2.5.3 IP and Ethernet:

For details, please refer to the Ethernet switch and IP forwarding reference library documentation. Interface adaptation and other changes relevant to this application are described in the following chapters.

2.5.4 AAL2 to RTP Mapping Module (Map-1):

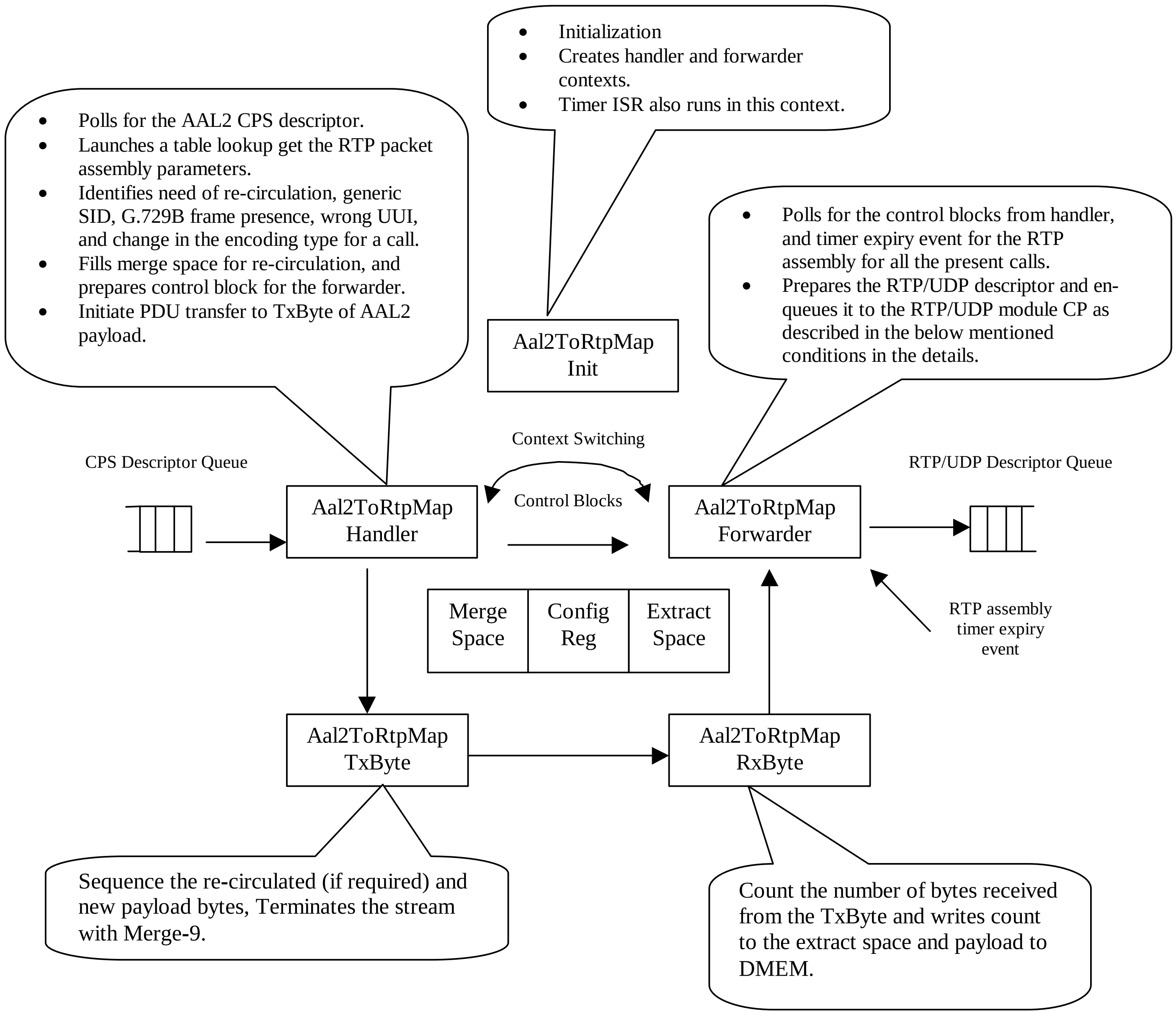
• Initialization
• Creates handler and forwarder contexts.
• Timer ISR also runs in this context.
• Polls for the AAL2 CPS descriptor.
• Launches a table lookup get the RTP packet assembly parameters.
• Identifies need of re-circulation, generic SID, G.729B frame presence, wrong UUI, and change in the encoding type for a call.
• Fills merge space for re-circulation, and prepares control block for the forwarder.
• Initiate PDU transfer to TxByte of AAL2 payload.
• Polls for the control blocks from handler, and timer expiry event for the RTP assembly for all the present calls.
• Prepares the RTP/UDP descriptor and en-queues it to the RTP/UDP module CP as described in the below mentioned conditions in the details.
Aal2ToRtpMap Init
Context Switching
CPS Descriptor Queue
RTP/UDP Descriptor Queue
Aal2ToRtpMap Handler
Control Blocks
Aal2ToRtpMap Forwarder
Merge Space
Config Reg
Extract Space
RTP assembly timer expiry event
Aal2ToRtpMap TxByte
Aal2ToRtpMap RxByte
Sequence the re-circulated (if required) and new payload bytes, Terminates the stream with Merge-9.
Count the number of bytes received from the TxByte and writes count to the extract space and payload to DMEM.

Figure 9 AAL2 to RTP Mapping Module Functionality

AAL2 To RTP direction mapping module is used to map the AAL2 incoming audio packets to RTP audio packets for its destination side of IP telephony users. There are three such channel processors for this mapping module, each of which processes 1K/3 VC connections for 8K/3 conversations. To conserve the IP bandwidth, assembly of the AAL2 packet will be done, which will be monitored by the RTP assembly timer. Each CP consists of the following contexts:

Aal2ToRtpMap Init Context:

- Initializes CP.
- Creates the other two Handler and Forwarder contexts.
- Timer Interrupt Service Routine (ISR) also runs in this context. This timer is used to monitor the assembly of RTP packets using one or more AAL2 packets.
- QMU interrupt handler also runs in this context.
- Configure the Tx and Rx Byte processor in re-circulation operating mode.

Aal2ToRtpMap Handler Context:

- Handler Context polls for the AAL2 CPS descriptor en-queued by the AAL2 Rx module.
- En-queues OAM-Alarm and CID 1…7 AAL2 packets to XP.
- Drops AAL2 packets other than audio packets (UUI 0...15).
- Based on VcIndex and CID, this context will launch a lookup to get the RTP parameters like:
    - RTP Time Stamp.
    - RTP Sequence Number.
    - Last Payload Type.
    - Expected AAL2 UUI value.
    - Buffer Handle of the previously stored AAL2 payload.
    - Number of bytes stored.
    - Count of AAL2 packet required filling a RTP packet.
    - IP address of the IP telephony user.
    - UDP port number being used by that IP telephony user.
    - Destination RTP/UDP queue id.
    - RTP SSRC identifier used for this call.
    - Call Id for this call session.

- The following things will be identified based on the CPS descriptor and lookup results:
    - Is there any change in the encoding type from the previous one?
    - Is this AAL2 packet a generic SID, or a G.729 SID frame?
    - Has AAL2 packet a wrong UUI number then the expected one? (AAL2 packets will not come out of sequence, but there could be a packet loss.)
    - Count of previously stored bytes was not a multiple of 16, so to append the new payload; re-circulation of the previous payload will be required?

- Based on the above decisions a control block will be prepared to co-ordinate the Forwarder's activity. These control blocks will be two in numbers, and will be protected by Handler/Forwarder read/write control lock.

- During assembly of the RTP payload, if re-circulation of the previous payload is required, the previous payload bytes will be copied to DMEM from SDRAM, and then will be filled in the merge space for TxByte processor to sequence them.

Aal2ToRtpMap TxByte Processor:

After owning the merge space, TxByte processor will first read the re-circulated bytes from the merge space (if any) then it will transmit them to RxByte processor, and then it will transmit the new AAL2 packet payload bytes. In the last byte of the payload, it will set the Merge-9 bit to indicate the end of frame.

Aal2ToRtpMap RxByte Processor:

If RxByte processor receives a valid data from the TxByte processor, it transfers those bytes to DMEM location set by the Forwarder context and write the number of bytes in the extract space, identifying the end of frame bit setting (Data-9).

Aal2ToRtpMap Forwarder Context:

There will be a RTP assembly timer for each call of N ms, to monitor the assembly. If x is the AAL2 packet time for that encoding. It means, after getting an AAL2 packet, it waits to get (N-x)/x more packets to come in an ideal case. AAL2 packets will not be fragmented to fill the RTP packet. For example, if RTP packet time is 30ms, and AAL2 packet is 20ms, One RTP packet will contain only one AAL2 packet, next AAL2 packet will not be fragmented to fill the RTP packet gap. Assembly is also subject to the IP telephony user's receiving capability. By default, one AAL2 packet will be converted to one RTP packet, and RTP assembly timer will not be started for any of the calls.

Forwarder polls on the two events:

(i) Availability of the control block prepared by the Handler.
(ii) RTP assembly timer expiry event for a particular call session.

When a control block prepared by the Handler becomes available to Forwarder, it will transfer the payload bytes received from the RxByte processor to the SDRAM location based on the following factors:

- If current AAL2 packet is a generic SID, or a G.729 SID frame?
- If encoding type has been changed, from the last type?
- Is there any discrepancy in the expected AAL2 UUI?
- If payload bytes are being re-circulated via Tx and RxByte processors?

Control block is processed in the following manner:

(1) If received AAL2 packet is a generic SID, then:

  - If there is payload accumulated in the buffer, prepare a descriptor and post it to the RTP/UDP module CP. Then en-queue a SID descriptor also with a time stamp based on the number of octets packed in the last RTP packet. New time stamp value for the next RTP packet will be adjusted based on the number of AAL2 packets this RTP packet contained + 5 ms for this generic SID.

  - If there is no previous payload, en-queue SID descriptor only. Increase the next RTP time stamp to 5 ms for this generic SID.

(2) If encoding type of the call has been changed from the previous one:

  - If there is payload accumulated in the buffer, prepare a descriptor and post it to the RTP/UDP module CP. New buffer will be allocated to store the new payload. Time stamp will be adjusted according to the last RTP packet content. Timer will be restarted, based on the assembly requirement. New encoding type will be updated in the table also.

   - If there is no previous payload accumulated, a fresh buffer will be allocated and the AAL2 payload will be stored there. Timer will be restarted, based on the assembly requirement.

(3) If AAL2 packet arrives with a wrong sequence number (AAL2 packets will not come out of sequence, but there could be a packet loss):

   - If there is payload accumulated in the buffer, prepare a descriptor and post it to the RTP/UDP module CP. New time stamp value for the next RTP packet will be adjusted based on the number of AAL2 packets this RTP packet contained + adjustment for the missed sequence numbers. This AAL2 payload will be stored in the new buffer.
   - If there is no previous payload, time stamp will be adjusted for the missed packets, and this AAL2 packet will be stored in the newly allocated buffer.

(4) If payload bytes are being re-circulated via Tx and RxByte processors, the new payload will be stored in the SDRAM location.

(5) It will be ensured that G.729 SID frame is being transmitted as an independent RTP packet or, is packed as a last frame in RTP packet after voice frames.

In all of the above cases, new table parameter values will be updated after each processing.

If there is an event of assembly timer expiry:

- If RTP assembly is over within this much time, i.e. accumulated data represents N ms of a RTP packet, prepare a descriptor and en-queue it to the RTP CP.
- If all the AAL2 packets do not arrive in N ms, which are required to fill an N ms RTP packet, a descriptor will be queued for the existing payload.
   - Time stamp will be adjusted for only accumulated AAL2 packets.
   - If required AAL2 packets which did not come earlier, arrive in sequence now, will be given to RTP/UDP module immediately without assembling them into a larger RTP packet. If they don’t arrive in a sequence, they will be assembled again from fresh and adjusting the time stamp value for the missed packets.

After the arrival of each AAL2 packet the count of AAL2 packets required will be decreased by one in the table, which will help to know the count of more AAL2 packets required at any moment.

2.5.5 RTP to AAL2 Mapping Module (Map-2):

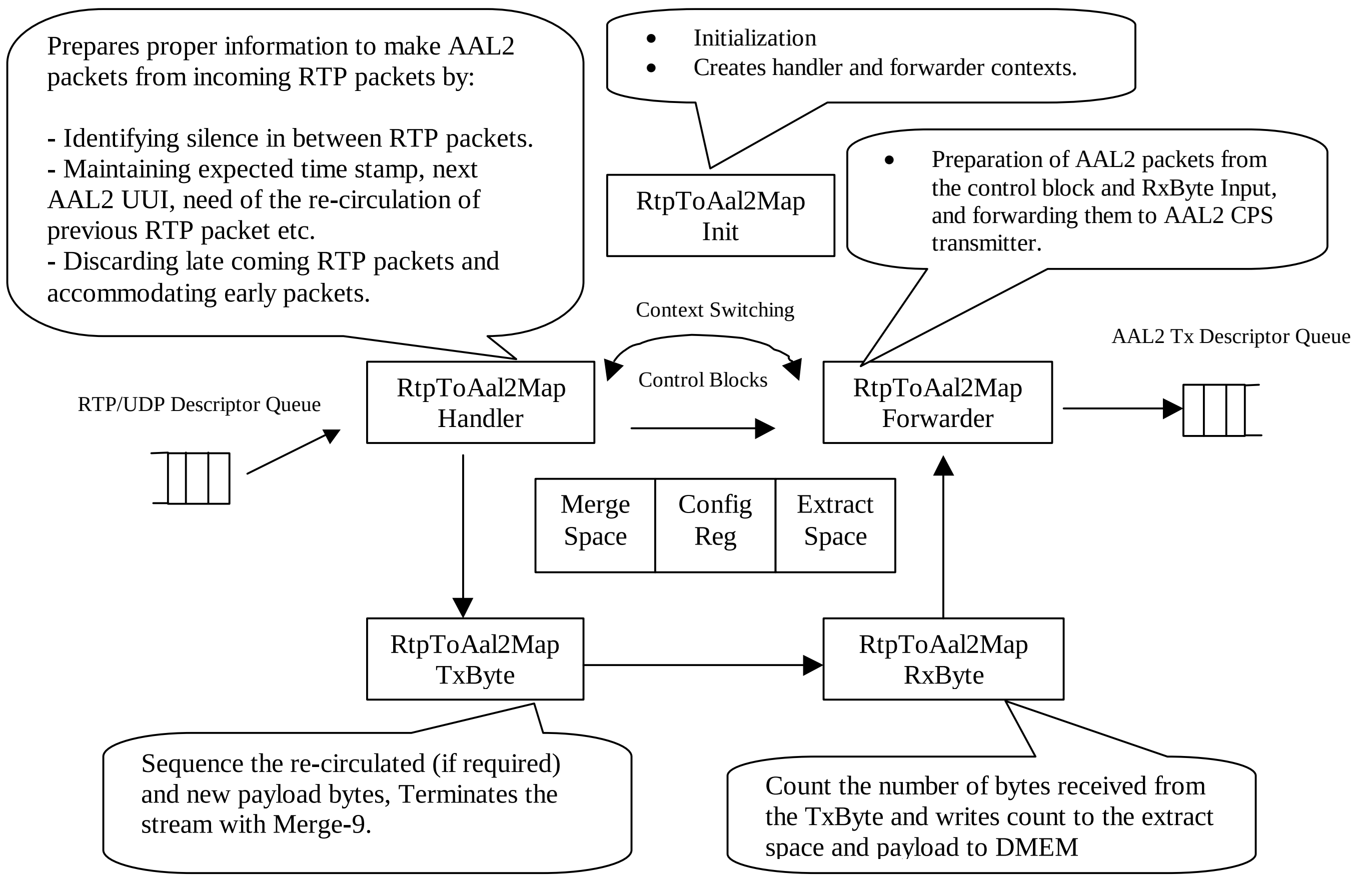
Prepares proper information to make AAL2 packets from incoming RTP packets by:
- Identifying silence in between RTP packets.
- Maintaining expected time stamp, next AAL2 UUI, need of the re-circulation of previous RTP packet etc.
- Discarding late coming RTP packets and accommodating early packets.
• Initialization
• Creates handler and forwarder contexts.
RtpToAal2Map Init
• Preparation of AAL2 packets from the control block and RxByte Input, and forwarding them to AAL2 CPS transmitter.
Context Switching
Control Blocks
AAL2 Tx Descriptor Queue
RTP/UDP Descriptor Queue
RtpToAal2Map Handler
RtpToAal2Map Forwarder
Merge Space
Config Reg
Extract Space
RtpToAal2Map TxByte
RtpToAal2Map RxByte
Sequence the re-circulated (if required) and new payload bytes, Terminates the stream with Merge-9.
Count the number of bytes received from the TxByte and writes count to the extract space and payload to DMEM

Figure 10 RTP to AAL2 Mapping Module Functionality

The following RTP parameter values will be used for the different encoding types, as mentioned in the table:

Sl. No.	Encoding Type	Rates Supported (Kbps)	Registered RTP Payload Types (PT)	Dynamically used Payload Type (PT) values @	Remarks
1.	G.711	64	0 or 8	#	PT=0 is for PCMU, 8 is for PCMA.
2.	G.723.1	5.3, 6.4	4	#	Payload itself indicates the rate.
3a.	G.726	40	*	96	
3b.	G.726	32	2	#	
3c.	G.726	24	*	97	
3d.	G.726	16	*	98	
4a.	G.728	16	15	#	
4b.	G.728	12.8	*	99	
4c.	G.728	9.6	*	100	
5a.	G.729A, AB	8	18	#	G.729 B frames are SID frames, when intermixed with voice frames shares the

5b.	G.729D, DB	6.4	*	101	
5c.	G.729E, EB	11.8 (Forward and Backward Adaptive Mode)	*	102	
6.	Generic SID (RTP payload for comfort noise)	N.A.	13	#	Will be used by G.711, G.726 and G.728 encoding types.

*: Not assigned by RTP, @: Used by the gateway, #: Not required, N.A.: Not Applicable

In the RTP to AAL2 mapping module, small size RTP packets will be accumulated, and big size RTP packet will be fragmented to make right size AAL2 packets, depending upon the encoding type selected.

RtpToAal2Map Init Context:

- Initializes CP.
- Creates the other two Handler and Forwarder contexts.
- Configure the Tx and Rx Byte processor in re-circulation operating mode.
- QMU interrupt handler is also registered in this context.

RtpToAal2Map Handler Context:

- Handler Context polls for the RTP descriptor en-queued by the RTP/UDP module. The parameters in the descriptor are:
    - VPI
    - VCI
    - CID
    - VC Index
    - Call Present Flag
    - Expected RTP Time Stamp value
    - Expected RTP Sequence Number of the RTP packet
    - Next UUI value for the AAL2 packet
    - Buffer handle for the fragmented RTP payload (fragmented for AAL2 packets)
    - Previous count of bytes stored in the fragmented RTP payload buffer
    - Last payload type
    - Buffer Handle of the new RTP packet (header + payload)
    - Count of bytes in the new RTP packet (header + payload)

- Based on the following factors it drops the RTP packet, and frees its buffer handle:

    1. RtpVersion is not equal to 2. No support for the old RTP versions.
    2. RTP Padding Bit is set to 1. No padded RTP payload is expected.
    3. RTP Header Extension Bit is set to 1. Header extension is not expected.
    4. If RTP CSRC count is not equal to 0. This application does not expect contributing sources other than the source itself.
    5. If RtpPayloadType is not supported based on the profile identifier currently selected, based on the I.366.2, Annexure P.

6. If RTP Payload Type, and RTP Packet Bytes Count represent a RTP packet containing more than 30 ms worth of audio data (gateway's maximum receive capability).

7. RTP Sequence Number is less than the expected RTP Sequence Number; it means it is a late arrived packet.

- Now, handler computes the following values:

  1. Length of the RTP payload
  2. Value of the next AAL2 UUI, based on the RTP time stamp, and sequence number.
  3. Size of each AAL2 packet going to be formed using this RTP payload.
  4. Presence of any SID in the payload.
  5. Number of SIDs.
  6. Location of SID in the payload.
  7. Number of complete and partial AAL2 packets that can be formed using this RTP payload.
  8. If previous accumulation of the AAL2 packet could not be completed, whether it can be completed now. Any need of re-circulation to complete the accumulation of the AAL2 packet?
  9. Next expected RTP time stamp and sequence number.

- Based on the above decisions a control block will be prepared to co-ordinate the Forwarder's activity. These control blocks will be two in numbers, and will be protected by Handler/Forwarder read/write control lock. Handler also fills the merge space accordingly.

RtpToAal2Map TxByte Processor:

TxByte processor sequence the AAL2 payload bytes after reading the merge space entries, and delimits the boundaries of AAL2 packet setting Merge-9 bit, and transfers the stream to RxByte processor.

RtpToAal2Map RxByte Processor:

RxByte processor stores the AAL2 payload delimited by the TxByte processor to SDRAM location, which is set by the forwarder, and writes the number of bytes in each AAL2 packet in extract space.

RtpToAal2Map Forwarder Context:

It polls for the availability of control block prepared by the handler context. Allocates the SDRAM buffers to store the AAL2 payload transferred by the RxByte processor, and en-queues the completed AAL2 packet descriptors to AAL2 CPS transmitter for multiplexing into the ATM cells. It also updates the table entries to process the next RTP packet.

2.5.6 RTP/ UDP Formatter/De-formatter module:

RTP/UDP Init Context:

- Initializes CP.
- Creates the other two Handler and Forwarder contexts.
- Configure the Tx and Rx Byte processor in re-circulation operating mode.

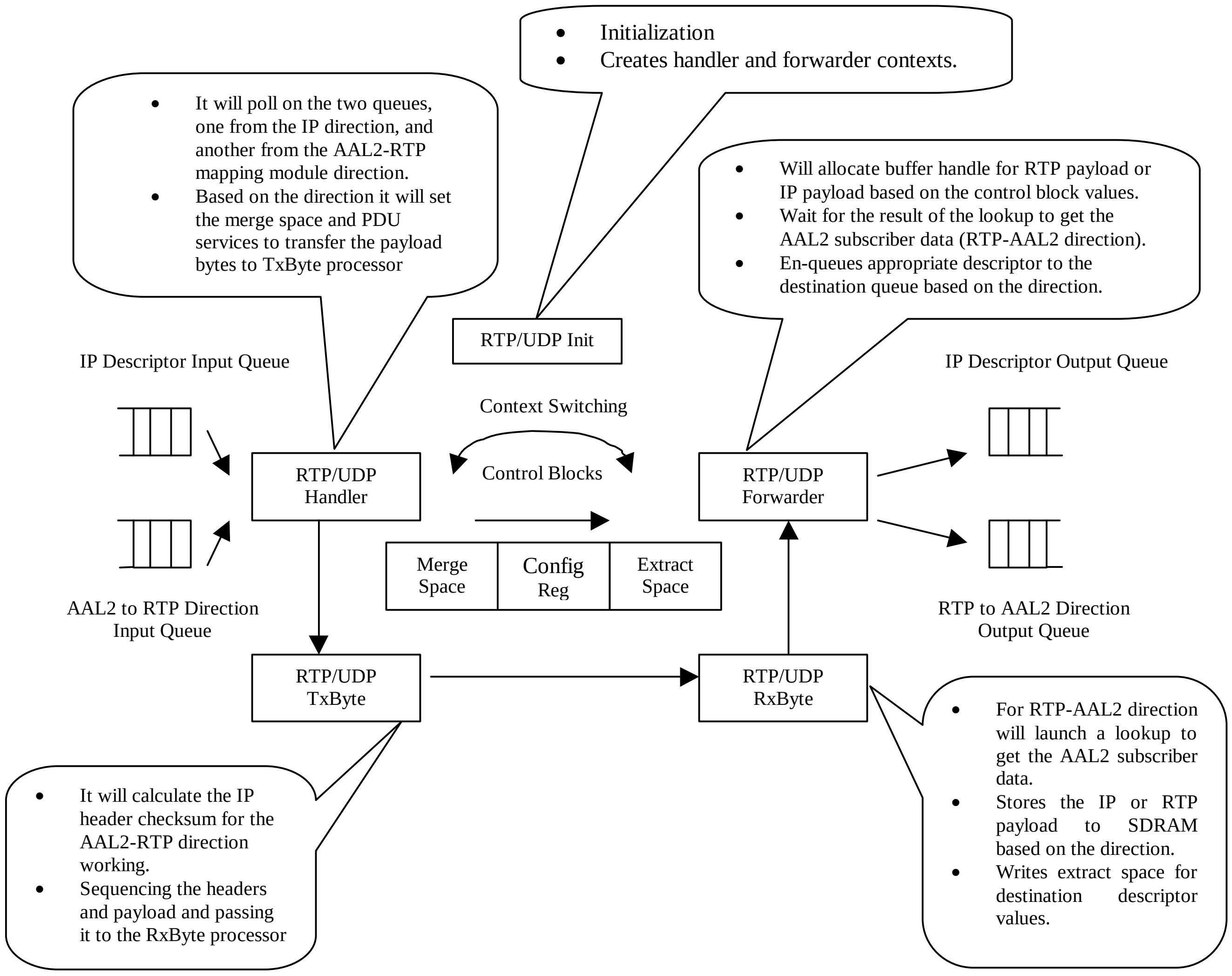
Initialization
Creates handler and forwarder contexts.
It will poll on the two queues, one from the IP direction, and another from the AAL2-RTP mapping module direction.
Based on the direction it will set the merge space and PDU services to transfer the payload bytes to TxByte processor
Will allocate buffer handle for RTP payload or IP payload based on the control block values.
Wait for the result of the lookup to get the AAL2 subscriber data (RTP-AAL2 direction).
En-queues appropriate descriptor to the destination queue based on the direction.
RTP/UDP Init
IP Descriptor Input Queue
IP Descriptor Output Queue
Context Switching
Control Blocks
RTP/UDP Handler
RTP/UDP Forwarder
Merge Space
Config Reg
Extract Space
AAL2 to RTP Direction Input Queue
RTP to AAL2 Direction Output Queue
RTP/UDP TxByte
RTP/UDP RxByte
For RTP-AAL2 direction will launch a lookup to get the AAL2 subscriber data.
Stores the IP or RTP payload to SDRAM based on the direction.
Writes extract space for destination descriptor values.
It will calculate the IP header checksum for the AAL2-RTP direction working.
Sequencing the headers and payload and passing it to the RxByte processor

Figure 11 RTP/UDP Formatter/De-formatter Module Functionality

<u>RTP/UDP Handler Context:</u>

RTP/UDP handler context will be receiving the input from two directions; one is from the IP side and another is from the AAL2 to RTP mapping module side. Based on the direction it will set the merge space and start transfer the payload from SDRAM to TxByte processor. It also prepares control block for the forwarder to convey the operating direction.

<u>RTP/UDP TxByte Processor:</u>

After owning the merge space, it will check the receiving direction set in the merge space. If the direction is towards IP side, then it will calculate IP header check sum and will sequence the IP, UDP and RTP header and RTP payload bytes for RxByte processor.

Otherwise, if the direction is from the IP side, it will only transfer the IP payload (UDP datagram) to RxByte processor.

In both the cases, it will also convey the direction to the RxByte processor.
RTP/UDP RxByte Processor:

After receiving the input from the TxByte processor, it will check the direction as received from the TxByte processor. If it is from the IP side, it will launch a lookup using IP source address and UDP source port number to get the AAL2 subscriber parameters. It also stores the IP payload (UDP datagram), or RTP payload after removing the header to the SDRAM location as directed by the Forwarder, based on the direction.

RTP/UDP Forwarder Context:

Forwarder only polls for the availability of control block prepared by the Handler. After receiving it, it allocates SDRAM buffer to store the IP or RTP payload coming through RxByte processor. If the operating direction is from the IP side, it also waits for the look up result, launched by the RxByte processor to get the AAL2 subscriber parameters. After getting the RTP payload, and forming its descriptor it en-queues it to the RTP to AAL2 mapping module CP, or after getting the IP payload as formed by the RxByte processor, and forming the IP descriptor, it en-queues it to the IP module.

2.6 XP: Functionality

1. **Initialization of system services**
   - Kernel Services
   - Buffer Services
   - Queue Services
   - Table Services

2. **Queue Creation & Configuration**

   (1) The following queues will be maintained for the XP:

   - Timer_CU start request message queue
   - SONET monitoring queue
   - ICMP message queue from IP.
   - Reserved CIDs (1...7), and AAL2 Alarms Message queue

   (2) The following queues will be maintained for the ATM CP (total 16 queues):

   - Base queue: for outgoing ATM cell descriptors.

   (3) The following queues will be maintained for the AAL2 Rx CP (total 16 queues):

   - Base queue: Incoming ATM cell descriptors from ATM.

   (4) The following queues will be maintained for the AAL2 Tx CP (total 16 queues):

   - Base queue: CPS packet descriptors from RTP-AAL2 map module.
   - Base queue+1: Timer_CU expiry message queue from XP.

   (5) The following queues will be maintained for the RTP/UDP CP (total 16 queues):

   - Base queue: To receive RTP descriptors from AAL2-RTP map module.
   - Base queue+1: To receive IP descriptors from IP CP.

(6) The following queues will be maintained for the AAL2 to RTP Map CP (total 16 queues):

- Base queue: To receive CPS packet descriptors from AAL2 Rx.

(7) The following queues will be maintained for the RTP to AAL2 Map CP (total 16 queues):

- Base queue: To receive RTP packet descriptors from RTP/UDP.

(8) The following queues will be maintained for the IP/Ethernet CP (total 16 queues):

- Base queue: To receive IP packet descriptors from UDP/RTP.

3. Buffer Pool Creation & Configuration

4. Tables Set-up

- Existing HTK table in the existing AAL2 Switch Application for the ATM module, which hashes on combination of (CpId, VPI, VCI). This table is indexed by CpId (4 bits), Partial VPI (12 bits), and Partial VCI (16 bits). The Hash Trie Key Table values shall be modified to have values of VcIndices from 0 to 1023for 1K VCs. ATM QoS parameters will be removed from this table.
- New HTK table for the module that maps from AAL2 to RTP, which hashes on the combination of (VcIndex, and CID). This table will be indexed by the CID (8bits), and VcIndex (16 bits).
- New HTK table for module that maps from RTP to AAL2, which hashes on the combination of the (IP Address, and UDP Port Number). Thus this table will be indexed by the UDP port number (16 bits), and IP Address (32 bits).
- Simple table indexed by call id (2 bytes) to receive VcIndex and CID. The RTP assembly timer processing will use it.

5. CP configuration and initialization

- SDP configuration of ATM CPs for OC-3c interface.
- SDP configuration of AAL2 Rx & Tx CPs for re-circulation.
- Package Loading.
- Initialization descriptor building with respective configuration parameters (Buffer Pool Ids, Table Ids, etc.) for CPs and passing them to CPs.
- Monitoring the input messages from different queues.

6. Timer_CU monitoring

For AAL2 CPS transmitter XP also manages Timer_CU, as it was used for the AAL2 switch application.

3. Existing ATM Layer and its Adaptation

3.1 Overview

The existing ATM Layer comprises of ATM Rx and ATM Tx functionality residing in a single CP. The SDPs used are RxBit, RxSync, RxByte and TxByte and TxBit processors. The RxByte Processor launches a lookup with Port Number, VPI and VCI combination as the key on the Hash-Tri Key Table and the CPRC module processes the response and sends it to the appropriate destination CP. Resource Management and OAM cells are sent to the XP ATM Control Queue. The CPRC ATM Tx module keeps on polling for cell descriptors from ATM QOS CP and on receipt of outgoing cell descriptor, prepares the merge space and uses PDU services for transmitting the cell payload from SDRAM to TxByte Processor. The existing AAL2 Switch application uses the ATM QOS Module and takes care of the following classes of traffic, Constant Bit Rate, real-time Variable Bit Rate, non real-time Variable Bit Rate, and Unspecified Bit Rate.

3.2 Interfaces

The following interfaces exist for ATM CP:

- Interface with AAL2 CPS Rx CP, based on the ATM Cell descriptor.
- ATM QOS CP Interface, based on the ATM Cell descriptor.
- XP Interface, based on the init descriptor.

3.3 Changes Required

- The ATM QOS Component is not going to be used. Hence, in this application, AAL2 CPS Tx CP will interface with ATM CP, based on the ATM Cell descriptor.

- The ATM Rx, ATM Tx Code shall be shared along with Aal2 Tx Code in same cluster.

- As number of queues allotted to each CP had changed from 4 to 16, hash defines for ATM_CONTROL_PLANE_QUEUE and ATM_SONET_MONITOR_QUEUE should be changed from 262 and 261 to 322 and 321 respectively.

3.4 Data Structures

```
typedef struct {
   BsBufHandle Bh;                /* Buffer Handle */
   int16u      length;           /* used by AAL5 payload type */
   int16u      VcIndex;           /* used for mCast but supplied for all */
   CellHeader  CellHeader;        /* txVpi on ucast cell, indicates rxVpi when encodedPti > 0 */
                              /* txVci on ucast cell, indicates rxVci when encodedPti > 0 */
   int8u       payloadType;       /* encoded information about cell type */
   Boolean     crc10Err;          /* if nonzero, indicates CRC-10 error on cell (ignore if not OAM
) */
   int16       MulticastFlag;     /* Pt-MPt indicator */
} CellDescriptor;
```

The above cell descrpitor data structure will be used from ATM Rx to AAL2 CPS Rx, and AAL2 CPS Tx to ATM Tx direction.

The SONET message descriptor en-queued to the XP by the ATM CP is as follows:

```
Typedef struct {
   PsSonetEvent       newEvents;
   PsSonetDefect      newStates;
   int8u              c2PathSignalLabel;
```

```
    int8u            traceErrorState; /* J0, bit 1, J1 bit 0 */
    int8u            portId;
    int8u            rdiPathType;
    int8u            S1;
} SonetMsg;
```

4. Existing AAL2 CPS Layer and its Adaptation

4.1 Overview

The existing AAL2 CPS Rx and AAL2 CPS Tx modules have been written for AAL2 CID switching. In the existing AAL2 Switch application, there are three AAL2 Rx CPs (each AAL2 Rx CP Supporting 512 VcIndices) and three AAL2 Tx CPs (each AAL2 Tx CP Supporting 512 VcIndices) connected to each ATM CP. In the AAL2 Switch application, there are two ATM CPs as the application switches between two OC-3c ports. So, there are total 6 AAL2 CPS Rx CPs, and total 6 AAL2 Tx CPs.

4.2 Interfaces

The following interfaces exist for AAL2 CPS Rx:

(1) Interface with ATM CP, based on the ATM Cell descriptor.
(2) AAL2 CPS Tx CP Interface, based on the AAL2 CPS packet descriptor.
(3) XP Interface, based on the init descriptor.

The following interfaces exist for AAL2 CPS Tx:

(1) Interface with AAL2 CPS Rx, based on the AAL2 CPS packet descriptor.
(2) XP Interface, based on the init descriptor.
(3) ATM QoS CP Interface, based on the ATM Cell descriptor.
(4) Timer_CU related interfaces with XP:
    - Interface from AAL2 Tx CP to XP for timer start request.
    - Interface from XP to AAL2 Tx CP for timer expiry notification.

4.3 Changes Required

- In this application, AAL2 Rx CP will en-queue the CPS packet descriptor to AAL2 to RTP mapping module CP, not to the AAL2 CPS Tx CP.

- AAL2 CPS Tx CP will receive AAL2 CPS packet descriptor from RTP to AAL2 mapping module CP, not from AAL2 CPS Rx CP.

- AAL2 CPS Tx CP will en-queue the ATM cell descriptor to ATM CP, because there is no ATM QoS CP in this application.

- Present implementation of CU Timer interface between XP and AAL2 Tx assumes each AAL2 Tx CP supports 512 Timers, but in this application, as each AAL2 Tx CP supports 1K/3 VcIndices, XP should ensure that if VcIndex is from 0 to 340, it sends the timer expiry descriptor to first Aal2 Tx CP, if VcIndex is from 341 to 681, it sends the timer expiry descriptor to the second Aal2 Tx CP and if VcIndex is from 682 to 1023, it sends the timer expiry descriptor to the third Aal2 Tx CP. These first, second and third AAL2 Tx CPs are based on the cluster, which they share.

- The AAL2 CID Switching related table lookup and the related table response processing code in AAL2 Rx module has to be removed.

- The Aal2 Rx Module has to find out the Map Queue Id, where it has to enqueue AAL2 CPS descriptor, based on the VcIndex. It compares the VcIndex with VcIndexLimit (from Init descriptor received from XP) and based on that it en-queues it to the corresponding Map Queue Id.

  DestId1and DestId2 are queue ids received through the init descriptor from XP. These are queue ids, where AAL2 Rx has to en-queue CPS packet descriptor.

  ```
  If (received VcIndex < VcIndexLimit)
  {
      QsMessageSend (DestId1, &mapDescMsg);
  ```

```
}
else
{
   QsMessageSend (DestId2, &mapDescMsg);
}
```

The VcIndex range corresponding to each CP is as shown in the figure below, cluster information and corresponding CP numbers are subject to change, based on the physical interface modules:

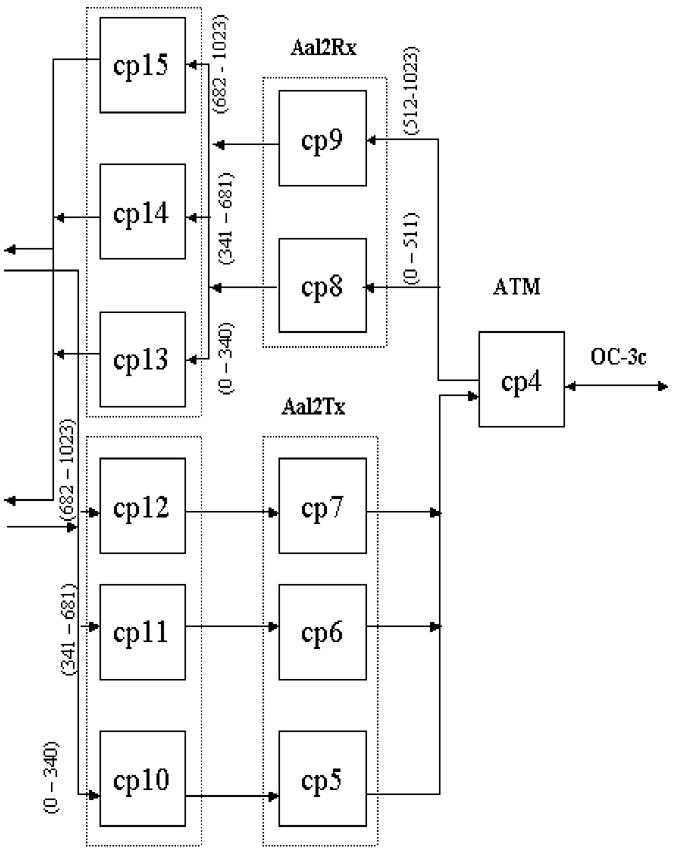

Figure 12 Interfaces for AAL2 CPS and ATM CPs.

- The SDP code in the existing AAL2 Switch application has SDPmain, which calls AAL2 Rx SDP or AAL2 Tx SDP Code based on CPId. This has to be removed and descriptor file shall be modified as below:

```
CP4-7 shared {
   CODE = $(ATMAAL2TXRCFILE);
   SDP4 = $(ATMSDPFILE);
   SDP5-7 = $(AAL2TXSDPFILE);
}
CP8-11 shared {
   CODE = $(AAL2RXMAPRCFILE);
   SDP8-9 = $(AAL2RXSDPFILE);
   SDP10-11 = $(RTPTOAAL2MAPSDPFILE);
}
```

The above packaging information may be changed, if cluster changes.

- The number of Aal2 Rx CPs is two, while the number of Aal2 Tx CPs are three. Hence, this has to be taken into account, during consideration of the Timers (for AAL2 Tx CP) and VcIndex table in DMEM, that is, for Aal2 Rx CP, the number of VcIndices supported is 512 per CP. As, the number of Aal2 Tx CPs is three, the number of VcIndices supported is 1024/3 = 342 per CP. The mask to get the proper DMEM index in AAL2 Tx CP should be calculated like this:

  Aal2TxDmemTableIndex = VcIndex – aal2TxNumber * 341;
  Where VcIndex can have values from 0 to 1023 and
  Aal2TxNumber is a parameter sent to AAL2 Tx CP from the XP and can have values 0, 1 or 2.

- The Aal2 Rx Module functionality of checking for reserved CIDs from 1 to 7 and enqueueing it to XP Queue should be taken care in AAL2 to RTP mapping module.

- The Aal2 Rx Module functionality of checking for UUIs from 16 to 30 and discarding should be taken care in AAL2 to RTP mapping module.

- The Aal2 Rx Module functionality of checking for UUI of value 31 and enqueueing it to XP Queue should be taken care in AAL2 to RTP mapping module.

4.4 Data Structures

Init Descriptor en-queued to the AAL2 Rx module by XP shall be modified as below:

```
Typedef struct
{
   BsPoolId    poolId;
   QsQueueId   destId1;
   QsQueueId   destId2;
   Int16u          VcIndexLimit;
} Aal2RxCpInitDesc;
```

Init Descriptor Enqueued to the AAL2 Tx module by XP shall be modified as below:

```
Typedef struct
{
   BsPoolId    poolId;
   QsQueueId   destId;
    Int8u          Aal2TxNumber;
} Aal2TxCpInitDesc;
```

The CPS Descriptor enqueued by AAL2 Rx CP to AAL2 to RTP Mapping module and also by RTP to AAL2 Mapping Module to AAL2 Tx CP is as below:

```
Typedef struct cpsDescriptor
{
   BsBufHandle cpsPckPayload;  /* Pointer to payload of the CPS Packet */
   Int16u      egressVcIndex;  /* Egress VC index   */
   Int8u       cid;            /* Channel ID */
   Int16u      li: 6;       /* Length Indicator */
   Int16u      uui: 5;       /* User-to-User Indication */
   Int16u      pad: 5;       /* Header Error Control */
   Int32u      atmEgressKey;   /* ATM Egress VPI <20:31>, VCI <4:19> and Port # <0:3> */
} Aal2CpsDesc;
```

The Timeout descriptor enqueued by XP to AAL2 Tx is as follows:

```
Typedef struct _TimeoutDesc
```

```
{
   Int8u        seqNo;
   Int16u        vcIndex;
} TimeoutDesc;
```

The Start Timer Descriptor structures used by AAL2 Tx CP to enqueue Timer Start command to XP is as follows :

```
Typedef struct _TimerStartData
{
   Int8u        seqNo;
   Int8u        cpId;
   Int16u       vcIndex;
} TimerStartData;

Typedef union _XpInQueueData
{
   TimerStartData timerStartCommand;
} XpInQueueData;

Typedef struct _XpInQueueDesc
{
   Int32u        command;
   XpInQueueData data;
} XpInQueueDesc;
```

The above C language constructs may not follow C-Port coding guideline and naming conventions, but it will be properly taken care of in the new development.

5. Existing Ethernet and IP Layer and its Adaptation

5.1 Overview

In the existing IP forwarding and Ethernet switching code, the IP SDP RxByte extracts the MAC DA value from received MAC frame, and finds out whether the received MAC frame is for MAC switching or IP forwarding. If it is meant for IP forwarding, the IP SDP RxByte, then parses the IP Header and does a table lookup based on the Destination IP Address. The IP RC module processes this table response and processes ICMP messages. On the transmit side, the TxByte assumes that the IP header has already checksum calculated and just decrements the TTL field in the IP header, and re-computes the delta checksum and TxByte transmits the MAC frame.

5.2 Interfaces

The following interfaces exist for IP CP:

- Interface with Packet over SONET (POS) and Gigabit Ethernet CPs, based on the IP descriptor.
- XP Interface, based on the init descriptor.

5.3 Changes Required

- In this application, IP CP will en-queue the IP descriptor to RTP/UDP CP. In the existing Ethernet switching application, IP Descriptors were en-queued to the POS and the gigabit Ethernet CPs.

- In this application, IP CP will receive IP descriptor from RTP/UDP CP. In the existing Ethernet switching application, IP Descriptors were en-queued by the POS and the gigabit Ethernet CPs.

- The table for MAC DA, IP SA and IP DA lookup will be removed and the corresponding code in CPRC module will be commented out. The destination queue Id, where the received IP Descriptor shall be en-queued, is received from the table lookup in the existing application. In the gateway application, the IP Descriptors shall be en-queued to the RTP/UDP CP queue ID, which is received from the init descriptor from XP.

- The IP Code shall en-queue an IP Descriptor to RTP/UDP CP, only if the Protocol ID is UDP. ICMP messages should be en-queued to XP. All other protocols IP packets will be dropped, and the buffers corresponding to them will be freed.

- The IP Header bytes shall not be sent to SDRAM and they shall be written only to extract space. In the existing Ethernet application, they are sent to SDRAM and stored along with IP Payload.

- The Buffer en-queued to IP Tx Module must have IP header checksum calculated, which will be done by the RTP/UDP CP. Also, the buffer shall comprise of other IP Header values and the IP Payload.

- Processing for ICMP message with type = 3(Destination Unreachable) and code =3(Port Unreachable) shall be taken care as follows: If this message arrives, then this has to be sent to XP, and XP should go ahead and reset the columns related to those call Ids. For all other ICMP messages, the buffer shall be freed by XP. Before en-queuing the ICMP descriptor to the XP, IP CP shall read the IP Payload into the DMEM, extract the UDP Port number information, prepare the ICMP message descriptor with IP and UDP information and then it will be en-queued to XP. This mechanism will be used to identify the call closure from the IP end point in the absence of call signaling.

- Both the Ethernet Ports will be connected to the same next hop router as shown in the figure below. So, MAC destination address for the outgoing Ethernet frames will be hard coded in the frame header. The reason for doing this is that full ARP support is not available in the existing IP/Ethernet code, so gateway cannot find the next frame destination address dynamically. But it will be supporting the ARP requests with proper ARP responses as the ports' MAC addresses, which it is receiving from XP in the init descriptor.

- The delta checksum calculation code in existing IP TxByte Processor shall be commented out, as this is not required in the gateway application.

ARP Modifications:

1. On receipt of Ethernet MAC frame with type value equal to 0x806, it shall be checked whether the ARP op-code is ARP Request. If the IP Address of the gateway is the same as the one in ARP request message, an ARP Reply message shall be composed with the MAC address having the gateway's corresponding Ethernet port's MAC address.

2. For all the Ethernet frames sent out of Ethernet ports shall have destination MAC address hard coded to the Next Hop Router's MAC address. In the existing Ethernet implementation, the destination MAC address is assigned from the IP Descriptor values en-queued to the IP Tx CP. Next hop router's MAC address will be provided by the XP in init descriptor.

In the field deployment scenario, the VoATM-VOIP gateway shall have only one IP router connecting it to the IP Network.

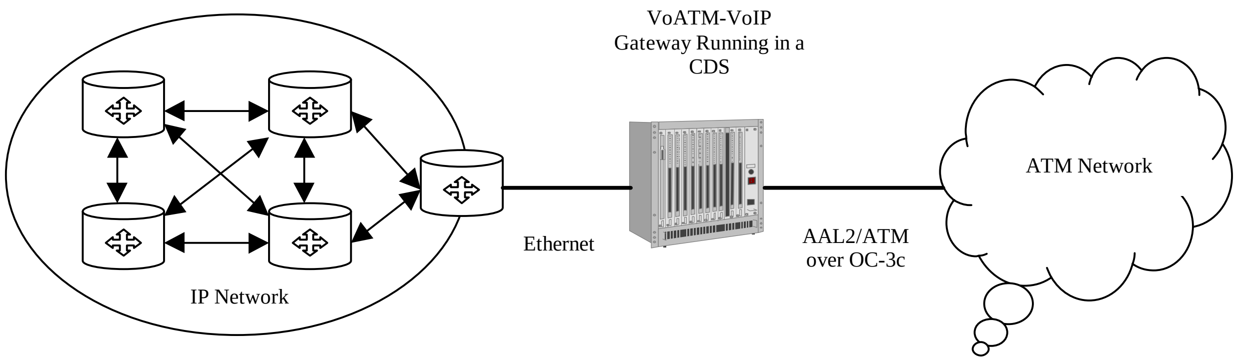

Figure 13 Field Deployment Scenario

In the system test lab, it should be ensured that the gateway application and the VOIP call generator are connected to the same router as shown below, because it will not support ICMP re-direct message to update the ARP cache table.

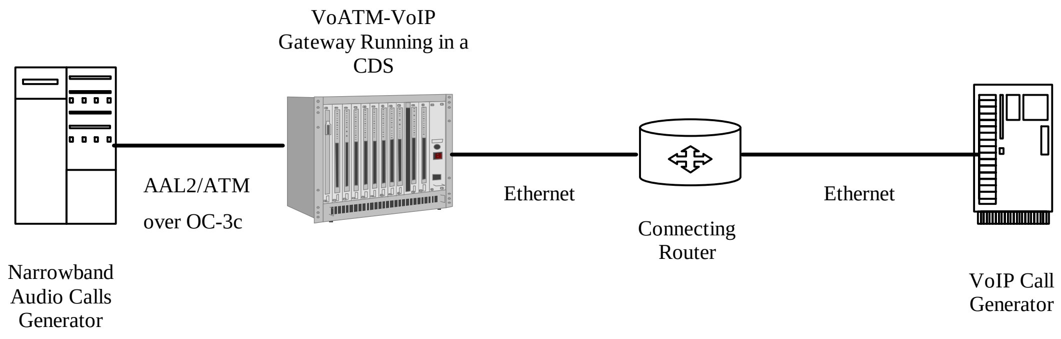

Figure 14 System Test Lab setup

5.4 Data Structures

The IP Descriptor en-queued to RTP/UDP CP has an additional field of source IP Address. The RTP/UDP CP for en-queuing to the IP CP uses this descriptor.

```
/* IP Descriptor */
Typedef struct {
    BsBufHandle   bufHandle;
    Int16u        length;
    Int16u        VNID_client;   /* 12 bit VNID + 4 bits for clientId. */
    Int32u        sourceAddr;   /* source address of the IP telephony user*/
    Int32u        appData1;
    Int16u        appData2;
    Int16u        appData3;
} IpDescriptor;
```

The ICMP Descriptor en-queued to the XP by the IP CP is as follows:

```
/* ICMP Descriptor */
typedef struct {
    BsBufHandle bufHandle;
    Int16u length;
    Int16u VNID_client;
    Int32u appData1; /* IP Address */
    Int16u appData2; /* UDP Port Number */
    Int16u appData3; /* ICMP Type and ICMP Code */
} IcmpDescriptor;
```

The Init Descriptor sent to the IP CP by the XP is as follows:

```
Typedef struct
{
    BsPoolId    PoolId;
    QsQueueId   rtpUdpQueueId;
    QsQueueId   icmpXpQueueId;  /* The queue Id of XP to which ICMP messages shall be sent to */
    Int32u      macInternalAddressHi4;
    Int16u      macInternalAddressLo2;
    Int32u      macDestAddressHi4; /* MAC Address for the next-hop router, */
    Int16u      macDestAddressLo2; /* which will be hard-coded for both of the Ethernet ports */
    Int8u       fabricEnabled;
    Int8u       fabricId;
} IpCpInitDesc;
```

6. RTP and UDP

Please refer to the figure RTP/UDP Formatter/De-formatter Module Functionality in the system description and architecture chapter.

6.1 Details

RTP/UDP Channel processor will be working in a dual mode. They will receive the incoming IP data grams from the IP module, as well as the RTP descriptor with RTP payload from the AAL2 to RTP direction mapping module.

There would be two CPRC threads working in this module. One is named as RTP/UDP Handler, which will handle the incoming data either from the IP side or from the AAL2 to RTP mapping module side. The second thread named as RTP/UDP Forwarder that will take care of the forwarding responsibility. It will forward the received data from the IP side to RTP to AAL2 mapping module, and the data received from the AAL2 to RTP mapping module to the IP side. The TxByte and RxByte processors will perform the intermediate sequencing of the payload and header bytes, which includes insertion and deletion of the required fields.

The direction will be controlled using the operating direction parameter, which will be decided on the basis of the input on the RTP/UDP handler side.

The detailed functioning of the RTP/UDP module is described below:

(1) RTP/UDP handler will be polling on two queues.

- (i) IP descriptor queue.
- (ii) RTP descriptor queue.

The first queue is to receive the descriptor en-queued by the IP module, while the second one is to receive the descriptor en-queued by the AAL2-RTP Mapping module.

(2) The parameters in the IP descriptor are:

- o IP Address of the IP telephony user.
- o Buffer handle of the UDP datagram (IP payload).
- o Others?

(3) The parameters in the RTP descriptor are:

- o IP Address of the IP telephony user
- o UDP Port Number of the IP telephony user.
- o Buffer handle of the RTP payload.
- o The following RTP header values:
  - o P: Padding bit (1 bit)
  - o X: Extension bit (1 bit)
  - o CC: CSRC Count (4 bits)
  - o M: Marker bit (1 bit)
  - o PT: Payload Type (7 bits)
  - o SN: Sequence Number (16 bits)
  - o TS: Time Stamp (32 bits)
  - o SSRC List (32 bits)
  - o CSRC List (15*32 bits) (May not be required, won't fit in the merge space with other parameters)

The following values will be constants, which are required to prepare an Ethernet frame from a RTP packet:

- o IP Address of the Gateway port.
- o UDP Port number used by the gateway (a dummy one).
- o Mac SA of the Gateway port.
- o Mac DA of the next hop router (for simulator testing: a dummy one).
- o Other IP/Ethernet header values.

(4) If RTP/UDP module has received something on the IP descriptor queue:

- o Descriptor will be de-queued from the queue.
- o Operating direction will be set to IP_TO_ATM.
- o In the merge space following values will be written:

  (i) Operating direction.
  (ii) IP Address of the IP telephony user.

- o Buffer handle of the UDP datagram will be set to transfer the datagram to the TxByte processor.

(5) If RTP/UDP module has received something on the RTP descriptor queue:

- o Descriptor will be de-queued from the queue.
- o Operating direction will be set to ATM_TO_IP.
- o In the merge space following values will be written:

  (i) Operating direction.
  (ii) Relevant RTP descriptor values to make an IP datagram. (Please note that RTP descriptor contains other parameters also other than RTP header parameter)

- o Buffer handle of the RTP Payload will be set to transfer the datagram to the TxByte processor.

(6) After owning the scope, TxByte processor will examine the operating direction set in the merge space by Handler thread.

<u>If the direction is IP_TO_ATM:</u>

(i) Operating direction will be passed to the RxByte processor.
(ii) IP Address of the IP telephony user will be passed to the RxByte processor.
(iii) Above fields will be followed by the UDP data gram.

<u>If the direction is ATM_TO_IP:</u>

(i) Operating direction will be passed to the RxByte processor.
(ii) The TxByte will compute the IP header checksum. IP header parameters, followed by UDP header, followed by RTP header, followed by the RTP payload will be sequenced by the TxByte processor and would be passed to the RxByte processor. TxByte will read the required constants to make and sequence the proper header values.

(UDP checksum from the pseudo header as per RFC 768 will not be calculated, as it is an optional field. It will be always set to 0 while transmitting. On the other direction, while receiving, this field will be ignored).

(7) After checking the input validity as valid, RxByte processor will examine the operating direction received as a first byte from TxByte processor to interpret the followed sequence of bytes.

<u>If the direction is IP_TO_ATM:</u>

(i) After operating direction, the next byte would be the IP address followed by the UDP data gram.
(ii) RxByte would launch a lookup using IP address and UDP source port number to get the values for the ATM-AAL2 subscriber parameters.
(iii) RxByte will extract out the RTP header values from the incoming stream and it will write it to the extract space along with the operating direction. The values written in the extract space will be used to prepare a descriptor for the RTP-AAL2 mapping module.
(iv) RxByte processor will also write the RTP payload (after chopping off the different headers) to the SDRAM as dictated by the RTP/UDP Forwarder module.

<u>If the direction is ATM_ TO_IP:</u>

(i) After operating direction, the next byte would be the IP header followed by the UDP data gram.
(ii) RxByte would extract out the IP header fields. It would write those fields, with the operating direction, in to the extract space and would transfer the UDP datagram to the SDRAM as set by the RTP/UDP Forwarder module. The values written in the extract space will be used to prepare a descriptor for the IP module.

(8) After owning the extract space, RTP/UDP forwarder module, will check the operating direction as written by the RxByte processor in the extract space.

<u>If the direction is IP_TO_ATM:</u>

(i) It will wait for the lookup response to come. The RxByte processor had launched this TLU lookup earlier.
(ii) After getting the ATM-AAL2 subscriber parameters via lookup results it will en-queue a descriptor to the RTP-AAL2 mapping module. Lookup result will also contain the destination queue id of one of the CPs, working as mapping modules, where this descriptor will be en-queued by this forwarder.
<u>(iii)</u> The above said descriptor will also contain the buffer handle of the RTP payload, which was filled by the RxByte processor.

<u>If the direction is ATM_TO_IP:</u>

(iv) It will make a descriptor for the IP module and will en-queue it to that module. This descriptor will also contain the buffer handle for the IP payload (UDP data gram), filled by the RxByte processor. There could be a load balancing policy involved, which will arbitrate the distribution between two output IP CPs.

(9) The co-ordination between RTP/UDP Handler and Forwarder threads:

- The co-ordination will be based on a control block concept. There will be two control blocks available for co-ordination.
- When handler will be preparing merge space entries, it will also prepare a control block for the Forwarder.
- Using this control block, Forwarder will be allocating a SDRAM buffer handle for the RxByte to transfer the payload.
- This control block may contain parameters like operating direction, semaphore control field for the Handler/Forwarder read/write control etc.

7. AAL2 SSCS Rx Functionality and AAL2 to RTP Mapping

AAL2 SSCS Rx functionality in this interworking application provides identification of the audio AAL2 packets. UUI range 0 to 15 is identified as audio AAL2 packets in I.366.2. These AAL2 packets will be processed in this mapping module. AAL2 packets having a UUI value 31 will be en-queued to XP, because they are OAM packets. Other UUI AAL2 packets are discarded and would not be processed further.

7.1 Details

Please refer to the Map-1 functionality figure shown in the System Description & Architecture chapter. The following description and pseudo code do not strictly follow any design methodology, coding and naming convention, code optimization and any programming language constructs; the only objective is to elaborate the design with suitable explanation and details.

Encoding profile identifier representing the profile-identifier as per I.366.2, Annexure P will be known as global parameter, which are configurable and given by the XP.

7.1.1 Aal2ToRtpMap Init Context:

It does the initialization work, receives initialization descriptor from the XP, and sets the global variables. It also configures Tx and Rx Byte processors to work in byte re-circulating mode. QMU interrupt handler, and RTP assembly timer ISR will also be registered in this context.

7.1.2 Aal2ToRtpMap Handler:

2. Polls for the CPS packet descriptor from AAL2 Common Part Sub-layer receiver (AAL2 CPS Rx). When becomes available, de-queues it from the queue and reads the following fields from the descriptor:

    - Aal2Cid
    - Aal2Li
    - Aal2Uui
    - Aal2PayloadBufHandle
    - VcIndex

3. AAL2 CPS packets for Aal2Cid values 1…7, will be en-queued to XP. These CIDs are used for AAL2 signaling.

4. If the Aal2Uui is 31, it means it is an OAM AAL2 packet; it will be en-queued to XP, with VC Index, and other CPS header values. Otherwise only 0…15 UUI values will be processed, other CPS packets will not be processed, and AAL2 CPS payload buffer handles will be freed for them.

5. Forms a key using Aal2Cid and VcIndex for TLU table lookup and launches a lookup. After launching a lookup, based on the EncodingProfileIdentifier, and the Aal2Li, it identifies the type of encoding that is NewEncodingType. Then till it gets the lookup result, switches the context to Aal2ToRtpMap Forwarder. If NewEncodingType is not valid, AAL2 CPS packet will be dropped, and its buffer handle will be freed.

6. If table lookup result fails, no further processing will be performed, it indicates that received AAL2 CPS packet has to be dropped, because no corresponding could be found in the table. In this case, AAL2 CPS payload buffer handle will be freed. In case of successful lookup response, the following parameters are expected from the lookup result:

    - PresentRtpTimeStamp
    - PresentRtpSeqNumber

- OldEncodingType
- ExpectedAal2UuiValue
- RtpPayloadBufHandle
- RtpPayloadBytesCount
- CountAal2PacketsNeeded
- IP Address of the IP Telephony user.
- UDP port number of the IP Telephony user.
- Destination RTP/UDP Queue Id.
- SSRC Identifier.
- CallId.

6. The following things will be identified from the lookup result and CPS packet descriptor:

   - NewEncodingType is not same as OldEncodingType?
   - Is NewEncodingType a generic SID?
   - Aal2Uui is not same as ExpectedAal2UuiValue?
   - RtpPayloadBytesCount is not an exact multiple of 16, i.e. re-circulation required, in case when encoding type has not changed from the last encoding type, ExpectedAal2UuiValue is same as Aal2Uui, CountAal2PacketsNeeded! =0, and assembly timer is not expired for this call?

7. If required, a static variable will be maintained to keep track of the current scope. After each AAL2 CPS packet processing in handler, this will be incremented in a module-2 fashion. This is not mandatory, because CPI library internally takes care of this.

8. Co-ordination between handler and forwarder contexts will be maintained using a control block, Aal2ToRtpControlBlock. These blocks will be two in numbers and will be used in a circular fashion. Next block availability will be checked, if next control block is ready to be written after forwarder processing of it, that is whether ForwarderProcDoneLock variable in the block has become TRUE.

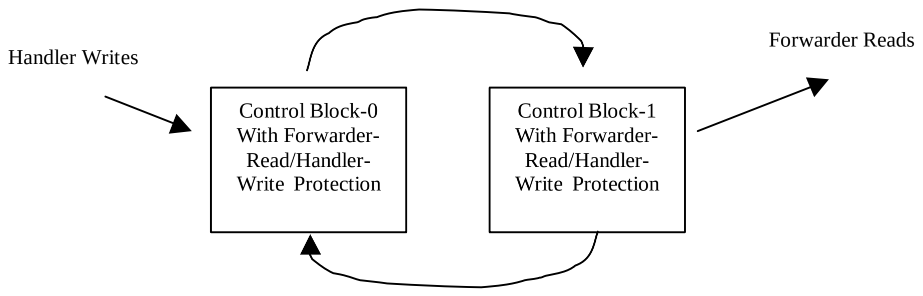
Handler Writes
Control Block-0 With Forwarder-Read/Handler-Write Protection
Control Block-1 With Forwarder-Read/Handler-Write Protection
Forwarder Reads

Figure 15 Aal2ToRtpMap Control Blocks

The following fields will be written in the Aal2ToRtpControlBlock, which are received from the table lookup and CPS descriptor:

- PresentRtpTimeStamp
- PresentRtpSeqNumber
- OldEncodingType
- NewEncodingType
- RtpPayloadBufHandle
- RtpPayloadBytesCount
- GenericSidPresentFlag
- G.729SidFramePresentFlag

- MissedAal2Uui = ExpectedAal2UuiValue - Aal2Uui
- Aal2Cid
- VcIndex
- ForwarderProcDoneLock
- IP Address of the IP Telephony user.
- UDP port number of the IP Telephony user.
- Destination RTP/UDP Queue Id.
- SSRC Identifier.
- CallId.

If re-circulation is required, RtpPayloadBytesCount variable will be decremented by the number of bytes being re-circulated, because these bytes will be stored again, and this field in the table will be updated again.

After filling the control block, it opens the protection lock, ForwarderProcDoneLock = FALSE, so that Forwarder can read this block. (Forwarder makes it TRUE, after processing it)

9. Owner ship of the merge space will be checked for the current scope, if it is available, and if re-circulation is required, handler copies the no-aligned previous payload bytes to DMEM from RtpPayloadBufHandle, where AAL2 packets are being assembled to make a bigger RTP packet and then, fills them into the merge space in the following way:

CountOfRe-circulatedBytes	Byte-1	Byte-2	Byte-3
Byte-4	Byte-5	Byte-6	Byte-7
Byte-8	Byte-9	Byte-10	Byte-11
Byte-12	Byte-13	Byte-14	Byte-15
CountOfNewAal2PayloadBytes			

Figure 16 Aal2ToRtpMap Merge Space Registers

If there are no re-circulated bytes then the corresponding count field of the merge space will be set to 0, and Byte-1 to Byte-15 merge space register locations will be ignored. So, maximum usage of the merge space is 17 bytes out of 64 bytes.

If merge space is not available for the present scope, handler keeps on switching the context for Forwarder, till it gets one.

7. After filling the merge space and control blocks, handler transfers the new AAL2 payload stored at Aal2PayloadBufHandle to TxByte processor using PDU services CPI API, and gives the merge space ownership to TxByte processor for sequencing.

8. Above processing cycle finishes the Aal2ToRtpMap Handler processing of one AAL2 CPS packet received from AAL2 CPS Rx.

7.1.3 Aal2ToRtpMap TxByte Processor:

1. It polls for the merge space availability, which will be made available by the CP RISC core (handler context).

2. After owning the merge space, TxByte processor checks for the CountOfRe-circulatedBytes field in the merge space, if 0, then it reads the CountOfNewPayloadBytes field from the merge space, and transfers those bytes to RxByte processor and puts the Merge-9 bit in the last byte to indicate the end of the stream.

3. If CountOfRe-circulatedBytes is not 0, then it transfers the re-circulated bytes from the merge space registers to RxByte processor, the it reads the CountOfNewPayloadBytes field from the merge space, transfers the new payload bytes to RxByte processor, and puts Merge-9 bit in the last byte to indicate the end of the stream.

4. After this much processing it gives the merge space ownership to CP RISC core (handler context), and polls for the next availability of merge space as it was doing in the beginning.

7.1.4 Aal2ToRtpMap RxByte Processor:

1. It polls for the extract space availability, which will be made available by the CP RISC core (forwarder context). After owning the extract space it loads the location address of the CountIncomingBytes variable of the extract space in the control register. After this, it checks the incoming stream availability in Large FIFO from TxByte processor.

2. When there is some bytes in the Large FIFO, it transfers those bytes to SDRAM location, which is already set by the forwarder context, and writes the count of the received bytes (identifying Data-9 bit) in the extract space location of CountIncomingBytes using control register. In the last it gives the ownership of the extract space to CP RISC core, and polls again for its ownership.

7.1.5 Aal2ToRtpMap Forwarder:

1. If required, a static variable will be maintained to keep track of the current scope. This will be incremented in a module-2 fashion. This is not mandatory, because CPI library internally takes care of this.

2. Forwarder polls on the two events:

   (1) Availability of the next control block as prepared by the handler as ForwarderProcDoneLock = FALSE.

   (2) 5 ms timer expiry event. There will be a global variable (TimerTableProcessed) for this, which will be set by the ISR. Whenever timer ISR will be invoked as a 5 ms time expiry, this variable will be set to FALSE. After Forwarder recognizes this event and completes it's processing over the table, it will set it to TRUE.

3. If the control block is ready to be read, the processing will be done as follows after reading its contents:

```
/* When incoming AAL2 packet is a generic SID */
If ((GenericSidPresentFlag == TRUE)
{
        First of all, stop the timer running for this call, using CallId as an index to the timer array.

        /* If something is stored already, first send it as a RTP packet, because SID will be transmitted
        separately */
        If (RtpPayloadBufHandle! = NULL)
        {
                /* Prepare the following RTP descriptor for the existing payload */

                /* Variables for RTP are updated based on the received control block contents */
                /* In the beginning PresentRtpTimeStamp and sequence number will be having some
                random value */
                RtpTimeStamp = PresentRtpTimeStamp;
                RtpSeqNumber = PresentRtpSeqNumber;
                RtpPayloadType = OldEncodingType;
                RtpBufHandle = RtpPayloadBufHandle;
                RtpPayloadBytes  = RtpPayloadBytesCount;
                IpDestAddr = IP Address of the IP Telephony user.
                UdpDestPort = UDP port number of the IP Telephony user.
                SsrcIdentifier = SSRC Identifier.
                MarkerBit = 0;
                En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP Queue Id;

                /* Update the parameters for the next RTP packet to be sent, which would be SID, in
                this case */
                PresentRtpTimeStamp += Adjusted time for the stored RTP payload;
                PresentRtpSeqNumber += 1;
                /* Other variables does not need any update, will be updated below */
        }

        /* Adjust the time stamp for the missed AAL2 packets, if any, otherwise MissedAal2Uui will
        be 0. MissedAal2Uui will be non-0, only if there are dropped AAL2 packets */

        /* AAL2_PACKET_SEQ_NUM_INTERVAL will be 5*8 =40. Unit of the time stamp is
        always samples in the RTP packet time as per RFC 1890 */

        PresentRtpTimeStamp += AAL2_PACKET_SEQ_NUM_INTERVAL * MissedAal2Uui;

        /* Prepare RTP descriptor for this SID */
        RtpTimeStamp = PresentRtpTimeStamp;
        RtpSeqNumber = PresentRtpSeqNumber;
        RtpPayloadType = GENERIC_SID;
        RtpBufHandle  = Allocate a new buffer to store SID byte;
        RtpPayloadBytes  = 1;
        IpDestAddr = IP Address of the IP Telephony user.
        UdpDestPort = UDP port number of the IP Telephony user.
        SsrcIdentifier = SSRC Identifier.
        MarkerBit = (MissedAal2Uui == 0)? 0:1;

        Transfer Payload from RxByte processor to DMEM and then to SDRAM location.

        En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP Queue Id;

        /* Update the parameters for the next RTP packet to be sent next time */
        PresentRtpTimeStamp += Packet time of generic SID;
        PresentRtpSeqNumber += 1;
        RtpPayloadBufHandle = NOT_ALLOCATED;
        RtpPayloadBytesCount = 0;
        OldEncodingType value will be not be changed;

        /* No more AAL2 packets are expected for the RTP assembly. It has to start from the fresh  */
        CountAal2PacketsNeeded = 0;
}


/* When the encoding type has been changed from the last type */
Else {
If ((OldEncodingType! = NewEncodingType)
{
        First of all, stop the timer running for this call, using CallId as an index to the timer array.
```

```
/* If something is stored already, first send it as a RTP packet */
If (RtpPayloadBufHandle! = NOT_ALLOCATED)
{
        /* Prepare the following RTP descriptor for the existing payload */

        RtpTimeStamp = PresentRtpTimeStamp;
        RtpSeqNumber = PresentRtpSeqNumber;
        RtpPayloadType = OldEncodingType;
        RtpBufHandle = RtpPayloadBufHandle;
        RtpPayloadBytes  = RtpPayloadBytesCount;
        IpDestAddr = IP Address of the IP Telephony user.
        UdpDestPort = UDP port number of the IP Telephony user.
        SsrcIdentifier = SSRC Identifier.
        MarkerBit = 0;

        En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP Queue Id;

        /* Update the parameters for the next RTP packet to be sent */
        PresentRtpTimeStamp += Adjusted time for this RTP packet;
        PresentRtpSeqNumber += 1;
}
/* If RtpPayloadBufHandle is NOT_ALLOCATED, no previous accumulation */
Else {

/* Adjust the time stamp for the missed AAL2 packets, if any, otherwise MissedAal2Uui will
be 0. MissedAal2Uui will be non-0, only if there are dropped AAL2 packets */

PresentRtpTimeStamp += AAL2_PACKET_SEQ_NUM_INTERVAL * MissedAal2Uui;

If (This AAL2 packet does not need any accumulation i.e. can be sent as a RTP packet,
because otherwise would need fragmentation of AAL2 packets to make a large RTP packet, | |
G.729SidFramePresentFlag == TRUE)
{
        /* Prepare RTP descriptor for this AAL2 packet */
        RtpTimeStamp = PresentRtpTimeStamp;
        RtpSeqNumber = PresentRtpSeqNumber;
        RtpPayloadType = NewEncodingType;
        RtpBufHandle  = Allocate a new buffer to store incoming payload stream;
        RtpPayloadBytes = Read it from extract space.
        IpDestAddr = IP Address of the IP Telephony user.
        UdpDestPort = UDP port number of the IP Telephony user.
        SsrcIdentifier = SSRC Identifier.
        MarkerBit = (MissedAal2Uui == 0)? 0:1;

        Transfer Payload from RxByte processor to DMEM, and then to SDRAM location.
        En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP Queue Id;

        /* Update the parameters for the next RTP packet to be sent */
        PresentRtpTimeStamp += Adjusted time for this RTP packet;
        PresentRtpSeqNumber += 1;
        OldEncodingType = NewEncodingType;
        RtpPayloadBufHandle = NOT_ALLOCATED;
        RtpPayloadBytesCount = 0;
        CountAal2PacketsNeeded = 0;
}
/* AAL2 packets can be accumulated to make a bigger RTP packet */
```

```
        Else
        {
                /* Time stamp and sequence numbers are already updated */
                OldEncodingType = NewEncodingType;

                /* Accumulate this AAL2 packet to the new location */
                RtpPayloadBufHandle  = Allocate a new buffer to store incoming payload stream;
                Transfer Payload from RxByte processor to DMEM and then SDRAM location.
                RtpPayloadBytesCount = Update based on the extract space entry;
                CountAal2PacketsNeeded = Update based on the encoding type and received AAL2
                packet size;

                Reload the timer count, e.g. if 5 more, 5 ms AAL2 packets are expected, then timer
                count should be loaded as 5, representing 25 ms, assuming timer granularity as 5 ms.
                One AAL2 packet is already stored, so target is to store 30 ms worth of RTP data.
        }}
}

/* New encoding type is same as the old one */
Else
{
        /* Adjust the time stamp for the missed AAL2 packets, if any, otherwise MissedAal2Uui will
        be 0. MissedAal2Uui will be non-0, only if there are dropped AAL2 packets */

        PresentRtpTimeStamp += AAL2_PACKET_SEQ_NUM_INTERVAL * MissedAal2Uui;

        /* There is already an allocated buffer for accumulation */
        If (RtpPayloadBufHandle! = NOT_ALLOCATED)
        {
                /* If there are no missed AAL2 packets in between  */
                If (MissedAal2Uui == 0)
                {
                        Transfer Payload from RxByte processor to this RtpPayloadBufHandle.
                        RtpPayloadBytesCount += Update based extract space entry;
                        Decrease CountAal2PacketsNeeded based on the packet time;
                }

                /* Descriptor can be en-queued to RTP/UDP in the following cases: accumulation is
                over (early then expected AAL2 packets), or there are few missed AAL2 packets in
                between, or there are no missed AAL2 packets, but this is a G.729B frame. */

                If ((CountAal2PacketsNeeded == 0) || (MissedAal2Uui! =0) || ((MissedAal2Uui ==
                0) && (G.729SidFramePresentFlag == TRUE)))
                {
                        First of all, stop the timer running for this call, using CallId as an index to
                        the timer array.

                        RtpTimeStamp = PresentRtpTimeStamp;
                        RtpSeqNumber = PresentRtpSeqNumber;
                        RtpPayloadType = OldEncodingType;
                        RtpBufHandle  = RtpPayloadBufHandle;
                        RtpPayloadBytes  = RtpPayloadBytesCount;
                        IpDestAddr = IP Address of the IP Telephony user.
                        UdpDestPort = UDP port number of the IP Telephony user.
                        SsrcIdentifier = SSRC Identifier.
                        MarkerBit = (MissedAal2Uui == 0)? 0:1;
                En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP
                Queue Id;

                /* Update the parameters for the next RTP packet to be sent */
                /* Old encoding type will remain same; that is why this piece is being
                executed */
                PresentRtpTimeStamp += Adjusted time for this RTP packet;
                PresentRtpSeqNumber += 1;
                RtpPayloadBufHandle = NOT_ALLOCATED;
                RtpPayloadBytesCount = 0;
                CountAal2PacketsNeeded = 0;
        }
}
/* If there is no previous allocation for accumulation*/
Else
{
        If (This AAL2 packet does not need any accumulation i.e. can be sent as a RTP
        packet, because otherwise would need fragmentation to make a large RTP packet | |
        G.729SidFramePresentFlag == TRUE | | CountAal2PacketsNeeded! = 0)
        {
                First of all, stop the timer running for this call, using CallId as an index to
                the timer array.

                /* Prepare RTP descriptor for this AAL2 packet */
                /* If there are missed AAL2 packets, time stamp is already adjusted above
                in the beginning */
                RtpTimeStamp = PresentRtpTimeStamp;
                RtpSeqNumber = PresentRtpSeqNumber;
                RtpPayloadType = NewEncodingType;
                RtpBufHandle  = Allocate a new buffer to store incoming payload stream;
                RtpPayloadBytes  = Read it from the extract space;
                IpDestAddr = IP Address of the IP Telephony user.
                UdpDestPort = UDP port number of the IP Telephony user.
                SsrcIdentifier = SSRC Identifier.
                MarkerBit = (MissedAal2Uui == 0)? 0:1;

                Transfer Payload from RxByte processor to DMEM and then to SDRAM
                location.
                En-queue this descriptor to the RTP/UDP CP using destination RTP/UDP
                Queue Id;

                /* Update the parameters for the next RTP packet to be sent */
                /* Old encoding type will remain same; that is why this piece is being
                executed */
                PresentRtpTimeStamp += Adjusted time for this RTP packet;
                PresentRtpSeqNumber += 1;
                RtpPayloadBufHandle = NOT_ALLOCATED;
                RtpPayloadBytesCount = 0;
                If (CountAal2PacketsNeeded)
                {
                        CountAal2PacketsNeeded = CountAal2PacketsNeeded - 1;
                }
        }

        /* Allocate new SDRAM buffer and start accumulation */
        Else
        {
```

```
                    /* Accumulate this AAL2 packet to the new location */
                    RtpPayloadBufHandle  = Allocate a new buffer to store incoming payload
                    stream;
                    Transfer Payload from RxByte processor to DMEM and then to  SDRAM
                    location.
                    RtpPayloadBytesCount = Read it from the extract space;
                    CountAal2PacketsNeeded = Update based on the encoding type and
                    received AAL2 packet time;

                    Reload the timer count, e.g. if 5 more, 5 ms AAL2 packets are expected,
                    then timer count should be loaded as 5, representing 25 ms, assuming timer
                    granularity as 5 ms. One AAL2 packet is already stored, so target is to store
                    30 ms worth of RTP data.
            }
        }
}}
```

ExpectedAal2UuiValue = Update based on the received UUI and encoding type;
Update the following values in TLU table, for the next AAL2 packet processing:

- PresentRtpTimeStamp
- PresentRtpSeqNumber
- OldEncodingType
- ExpectedAal2UuiValue
- RtpPayloadBufHandle
- RtpPayloadBytesCount
- CountAal2PacketsNeeded

After this control block processing, forwarder frees the control block by setting ForwarderProcDoneLock = TRUE.

4. Timer Expiry handling mechanism in forwarder:

Index on CallId

Timer Started (1 bit)	Timer Load Count, N (Representing N*5 ms) (7 bits)

Figure 17 Timer Table Entry

The above figure represents the timer table (stored in DMEM) for all the calls per AAL2 To RTP Mapping Module CP. This is an array of bytes, indexed on the CallIds. MSB represents if timer is started for that call, if set to 1, and call is using the timer mechanism for AAL2 packets assembly for RTP packets. Seven LSBs represent the timer load count, as depicted in the figure. Load value multiplied by 5 represents the time count in milliseconds, because granularity of the timer is 5 ms.

For timer ISR, DCP register will be filled by a value, which represent number of clock cycles equivalent to 5 ms, using ksTimerSet () API. When this count will become 0, ISR will be invoked. On invocation, five milliseconds timer ISR will traverse the timer table to see if timers are started for which of the CallIds. If this one bit field is set, it means timer is started for this particular CallId. Then for this entry if timer load count is not 0, it will decrease it by 1, then it will set the TimerTableProcessed variable to FALSE, after traversing the whole table.

Aal2ToRtpMap Forwarder will be also polling on the variable TimerTableProcessed. If it is set to FALSE by the ISR, it will scan the timer table in the following way:

```
Static int16u CallId = CallIdLowerLimit;

If (TimerTableProcessed == FALSE)
```

```
{
        While (TimerTable [CallId]! = 0x80)
        {
                FunctionIncrementCallId (CallId);
                If (TimerTableProcessed == TRUE)
                {
                        Break;
                }
        }

        /* Call is present with timer is expired */
        If  (TimerTableProcessed == FALSE)
        {
                • Launch a lookup using CallId to find out the VcIndex and CID
                  for this call. Using VcIndex and CID find out the other
                  parameters from the table, which is being used for the RTP
                  packet assembly.
                • Prepare a RTP descriptor and en-queue it to the RTP/UDP CP.
                • Update the entries of the RTP table.
                • Stop the timer for this call.
                • FunctionIncrementCallId (CallId)
        }
}
FunctionIncrementCallId (CallId)
{
        If (++CallId > CallIdUpperLimit)
        {
                CallId = CallIdLowerLimit;
                TimerTableProcessed = TRUE;
        }
}
```

This completes the processing of the Forwarder context.

8. AAL2 SSCS Tx Functionality and RTP to AAL2 Mapping

AAL2 SSCS Tx functionality in this interworking application provides encapsulation of the audio AAL2 packets. This application does not directly do any encoding of the digitized audio samples using any encoding algorithm, but it re-encapsulates the audio data coming from the IP endpoint using RTP as a transport mechanism, to AAL2 CPS packet format as defined in I366.2. Only audio RTP packets will be mapped to equivalent AAL2 packets, which use a UUI range of 0…15.

8.1 Details

Please refer to the Map-2 functionality diagram shown in the System Description & Architecture chapter. The following description does not strictly follow any design methodology, naming convention, and optimization; the only objective is to elaborate the design with suitable explanation and details.

Encoding profile identifier representing the profile-identifier as per I.366.2, Annexure P will be known as global parameter, which is configurable and given by the XP to this module in the initialization descriptor.

8.1.1 RtpToAal2Map Init Context:

It does the initialization work, receives initialization descriptor from the XP, and sets the global variables. It also configures Tx and Rx Byte processors to work in byte re-circulating mode. QMU interrupt handler will also be registered in this context.

8.1.2 RtpToAal2Map Handler:

1. Polls for the RTP packet descriptor from the RTP/UDP module. If not available, it switches the context for the Forwarder context. When becomes available, it de-queues it from the queue and reads the following fields from the descriptor:

    - VPI (2 bytes)
    - VCI (2 bytes)
    - CID (1 byte)
    - VcIndex (2 bytes)
    - CallPresentFlag (1 bytes)
    - ExpectedRtpTimeStamp (4 bytes)
    - ExpectedRtpSequenceNumber (2 bytes)
    - NextAal2UuiValue (1 byte)
    - FragAal2BufHandle (4 bytes)
    - FragAal2BytesCount (1 byte)
    - LastRtpPayloadType (1 byte)
    - RtpBufHandle (4 bytes)
    - RtpPacketBytesCount (2 bytes)

2. If all the cases mentioned below, CallPresentFlag will be checked while checking ExpectedRtpTimeStamp, and ExpectedRtpSequenceNumber fields. If it is set to FALSE, these fields will be ignored.

3. First 16 bytes of the RTP packet contents (12 bytes of the fixed RTP header, excluding the CSRC identifiers list) will be copied to the DMEM from SDRAM, and the following RTP header values will be read:

    - RtpVersion (2 bits)

- RtpPaddingBit (1 bit)
- RtpHdrExtBit (1 bit)
- RtpCCount (4 bit)
- RtpMarkerBit (1 bit)
- RtpPayloadType (7 bits)
- RtpSequenceNumber (2 bytes)
- RtpTimeStamp (4 bytes)
- RtpSsrcId (4 bytes)

4. In the following cases, this RTP packet will be dropped, its buffer handle will be freed, and no further processing will be done:

    1. RtpVersion is not equal to 2. No support for the old RTP versions.
    2. RtpPaddingBit is set to 1. No padded RTP payload is expected.
    3. RtpHdrExtBit is set to 1. Header extension is not expected.
    4. If RtpCCount is not equal to 0. This application does not expect contributing sources other than the source itself.
    5. If RtpPayloadType is not supported based on the profile identifier currently selected, based on the I.366.2, Annexure P.
    6. If RtpPayloadType, and RtpPacketBytesCount represent a RTP packet containing more than 30 ms worth of audio data (gateway's maximum receive capability).
    7. RtpSequenceNumber is less than the ExpectedRtpSequenceNumber, it means it is a late arrived packet.

5. Based on the above parameters the following values will be calculated:

    (i) Length of the RTP payload.

    - From RtpPacketBytesCount, the length of the RTP header will be subtracted, to get the RtpPayloadLength.

    (ii) Value of the UUI (NextAal2UuiValue) for the next AAL2 packet.

    - If RtpSequenceNumber is same as the ExpectedRtpSequenceNumber, and RtpTimeStamp is same as the ExpectedRtpTimeStamp, the next value of UUI will be NextAal2UuiValue, as received from the RTP/UDP descriptor.
    - *If RtpSequenceNumber is less than the ExpectedRtpSequenceNumber; this RTP packet is already dropped, as described above, so this condition will not be tested here.*
    - If RtpSequenceNumber is greater than the ExpectedRtpSequenceNumber, then NextAal2UuiValue will be incremented to represent the silence for the missed RTP packets. Time difference between ExpectedRtpTimeStamp, and RtpTimeStamp will be calculated, and if it is not 5 ms, or not an exact multiple of 5 ms, it will be converted to represent the next multiple of 5 ms. For each 5 ms; NextAal2UuiValue will be incremented by 1.

- If RtpSequenceNumber is same as the ExpectedRtpSequenceNumber, but RtpTimeStamp is not same as the ExpectedRtpTimeStamp, the value of NextAal2UuiValue will be calculated as described above after getting the time difference in a multiple of 5 ms.

(iii) Size of the AAL2 CPS packets payload, which are going to be formed using this RTP packet (SizeAal2PacketPayload).

- Based on the selected profile identifier and RtpPayloadType, SizeAal2PacketPayload will be identified, as described in Annexure P, I.366.2. If SizeAal2PacketPayload is greater than the RtpPayloadLength, then SizeAal2PacketPayload will be set to the RtpPayloadLength.

(iv) SID presence in the RTP payload.

- Presence of Generic, or G.723.1, or G.729 based SIDs in the RTP payload will be identified based on the following table:

If RTP Payload Type =	**And, if RTP Packet Time (ms) =**	**And, if RTP Payload Bytes Count =**	**Then, Number of SIDs =**
4 (G.723.1)	30	4	1
13 (Generic)	5/10/15/20/25/30	1/2/3/4/5/6	1/2/3/4/5/6
18 (G.729AB)	10/20/30	2/12/22	1
101 (G.729DB)	10/20/30	2/10/18	1
102 (G.729EB, Forward)	10/20/30	2/17/32	1

(v) Number of SIDs present in the RTP payload (CountOfSids).

- Based on the above table, in all other cases CountOfSids will be 0.

(vi) Location of SID present in the RTP payload (LocationOfSid).

- This represents starting from which byte SID starts in the RTP payload. LocationOfSid = 0, will represent that this RTP packet contains SID only. Otherwise, it will be set appropriately, e.g. in 20 ms G.729AB RTP packet, LocationOfSid will be equal to 10. That is 11th and 12th bytes represent G.729B (SID) frame.

(vii) Number of complete and partial AAL2 packets (CountAal2Packets, CountPartialAal2Packets), which can be formed using this arrived RTP packet.

- Based on the selected profile identifier, RtpPayloadLength (to identify SIDs), and RtpPayloadType; CountAal2Packets, and CountPartialAal2Packets will be calculated. Maximum value of CountPartialAal2Packets will be 1. If any SID is present, CountPartialAal2Packets will be equal to 0.

(viii) Processing of FragAal2BufHandle.

- If FragAal2BufHandle is equal to NOT_ALLOCATED, nothing will be done. Otherwise the following cases are possible:
  - If RtpSequenceNumber is same as the ExpectedRtpSequenceNumber, and RtpTimeStamp is same as the ExpectedRtpTimeStamp, and

FragAal2BytesCount is not an exact multiple of 16, re-circulation of the non-aligned bytes from FragAal2BufHandle will be required (CountRecircBytes).

- If RtpSequenceNumber is not same as the ExpectedRtpSequenceNumber, or RtpTimeStamp is not same as the ExpectedRtpTimeStamp, FragAal2BufHandle will be freed, because accumulation of this fragmented AAL2 packet cannot be completed.

- If RtpPayloadType is not same as the LastRtpPayloadType, then also FragAal2BufHandle will be freed, because accumulation of this fragmented AAL2 packet cannot be completed using this RTP packet.

- If CountOfSids is not 0, i.e. any SID is present, then also FragAal2BufHandle will be freed.

(ix) ExpectedRtpSequenceNumber will be calculated using the RtpSequenceNumber of the RTP packet which is being processed, and ExpectedRtpTimeStamp will be calculated using the RtpPayloadType, RtpPayloadLength, and not processed RTP packets sequence numbers (if any, because of loss or out of sequence drop). Wrapping around of time stamp and sequence numbers data types will properly taken care of.

6. Owner ship of the merge space will be checked for the current scope, if it is available, Handler will fill the merge space in the following way, otherwise context will be switched for the Forwarder context:

RtpPayloadType	CountAal2Packets	SizeAal2PacketPayload	CountOfSids
LocationOfSid	RtpPayloadLength	CountRecircBytes	Byte-1
Byte-2	Byte-3	Byte-4	Byte-5
Byte-6	Byte-7	Byte-8	Byte-9
Byte-10	Byte-11	Byte-12	Byte-13
Byte-14	Byte-15		

Figure 18 RtpToAal2Map Merge Space Registers

7. Co-ordination between the RtpToAal2 Handler and Forwarder contexts will be maintained using a control block, RtpToAal2ControlBlock. These blocks will be two in numbers and will be used in a circular fashion. Next block availability will be checked, if next control block is ready to be written after Forwarder processing of it, that is whether ForwarderProcDoneLock variable in the block has become TRUE.

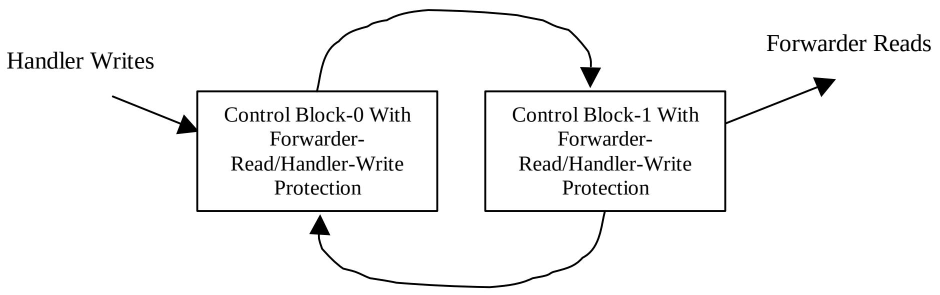
Handler Writes
Control Block-0 With Forwarder-Read/Handler-Write Protection
Control Block-1 With Forwarder-Read/Handler-Write Protection
Forwarder Reads

Figure 19 RtpToAal2Map Control Blocks

If next control block is ready to be written by the handler the following values will be written:

i. CountAal2Packets

ii. CountPartialAal2Packets
iii. NextAal2UuiValue
iv. VPI
v. VCI
vi. CID
vii. VcIndex
viii. CallPresentFlag
ix. RtpPayloadType
x. FragAal2BufHandle
xi. FragAal2BytesCount
xii. ExpectedRtpTimeStamp
xiii. ExpectedRtpSequenceNumber

After writing the control block, handler sets the ForwarderProcDoneLock to FALSE, which means that it is ready to be read by the Forwarder.

8. After filling the merge space and control blocks, handler transfers the new RTP data stored at RtpBufHandle to TxByte processor using PDU services CPI functions, and gives the merge space ownership to TxByte processor for sequencing.

9. Above processing cycle finishes the RtpToAal2Map Handler processing of one RTP packet received from RTP/UDP CP.

8.1.3 RtpToAal2Map TxByte Processor:

1. It polls for the merge space availability, which will be made available by the CP RISC core (handler context).

2. The first 12 bytes of the fixed RTP header will be discarded in the TxByte processor, which are stored at RtpBufHandle, because the Handler has already processed them.

3. After owning the merge space, TxByte processor checks for the CountOfSids field in the merge space. If it is 0, then it checks for the CountRecircBytes value, if 0, then it reads the RTP payload bytes from the SDRAM, and sets the Merge-9 bit in the last byte of each set of SizeAal2PacketPayload number of bytes, representing different AAL2 packets payloads.

4. If CountRecircBytes is not equal to 0, then it transmits re-circulated bytes first, from the merge space, then transmits RTP payload bytes from the SDRAM, and sets the Merge-9 bit accordingly.

5. If CountAal2Packets is 0, then it sets the Merge-9 bit in the last byte of RtpPayloadLength.

6. If CountOfSids is not equal to 0, and LocationOfSid is equal to 0, it means, there are SID bytes only. It sets the Merge-9 bit in the each last SID byte, identifying RtpPayloadType.

7. If CountOfSids and LocationOfSid are not equal to 0, then it sets the Merge-9 bit using SizeAal2PacketPayload, and LocationOfSid values.

8. This entire sequenced stream, in all of the above cases, will be transferred to RxByte processor.

9. After this much processing it gives the merge space ownership to CP RISC core (handler context), and polls for the next availability of merge space as it was doing in the beginning.

8.1.4 RtpToAal2Map RxByte Processor:

1. It polls for the extract space availability, which will be made available by the CP RISC core (forwarder context). After owning the extract space it loads the location address of the Aal2PayloadLength variable of the extract space in the control register. After this, it checks the incoming stream availability in Large FIFO from TxByte processor.

2. When there is some bytes in the Large FIFO, it transfers those bytes to SDRAM location, which is already set by the forwarder context, and writes the count of the received bytes (identifying Data-9 bit) in the extract space location of Aal2PayloadLength using control register. In the last, it gives the ownership of the extract space to CP RISC core, and polls again for its ownership.

8.1.5 RtpToAal2Map Forwarder:

1. Forwarder polls on the availability of the next control block as prepared by the handler as ForwarderProcDoneLock = FALSE.

2. When control block becomes available, it read the following variables from the control block:

   i. CountAal2Packets
   ii. CountPartialAal2Packets
   iii. NextAal2UuiValue
   iv. VPI
   v. VCI
   vi. CID
   vii. VcIndex
   viii. CallPresentFlag
   ix. RtpPayloadType
   x. FragAal2BufHandle
   xi. FragAal2BytesCount
   xii. ExpectedRtpTimeStamp
   xiii. ExpectedRtpSequenceNumber

3. If value of FragAal2BufHandle set in the control block is NOT_ALLOCATED, it allocates fresh SDRAM buffer for each of the CountAal2Packets, where RxByte transfers the AAL2 packet payload. Then it en-queues the CPS packet descriptor to its destination AAL2 CPS Tx CP's queue. This queue id is received from the XP in init descriptor. It decides the UUI value for the AAL2 packets using NextAal2UuiValue, and RtpPayloadType. (LI+1) header value it reads from the extract space for each of the AAL2 packet.

4. If value of FragAal2BufHandle set in the control block is not NOT_ALLOCATED, it means no previous bytes are being re-circulated. Otherwise, it sets the FragAal2BufHandle SDRAM location for RxByte writing. Now it updates the FragAal2BytesCount value from the extract space value. If this value is completely representing the one AAL2 packet for RtpPayloadType, it en-queues it to AAL2 CPS Tx. UUI value for this AAL2 packet will be NextAal2UuiValue.

5. If CountPartialAal2Packets, is 1, then it allocates SDRAM buffer for it, RxByte transfers the AAL2 packet payload to this location, which is not complete so far. Forwarder sets this SDRAM location to FragAal2BufHandle, and FragAal2BytesCount equal to the Aal2PayloadLength, written in the extract space.

6. Forwarder calculates the NextAal2UuiValue, based on the last UUI value used in the last AAL2 packet formed by the Forwarder, except partial AAL2 packet stored in the SDRAM location.

7. The following values will be updated in the table by the forwarder, after processing one control block:
    i. NextAal2UuiValue
    ii. FragAal2BufHandle
    iii. FragAal2BytesCount
    iv. LastRtpPayloadType = RtpPayloadType
    v. ExpectedRtpTimeStamp
    vi. ExpectedRtpSequenceNumber
    vii. CallPresentFlag = 1 (TRUE)

8. After processing this control block, it will be freed for Handler writing, by setting the ForwarderProcDoneLock to TRUE.

9. C-5 Executive Processor and Resources

9.1 Requirements & Overview

XPRC is used for boot and initialization of the C5 chip. The XP application is split into two phases. The first phase performs service initialization, configures system resources and loads the channel processors. The second phase builds the tables, starts the channel processors and sends the initialization descriptors required for each of these channel processors and starts the SDPs before entering the into the infinite loop to monitor the different events from different CPs.

The sequence of operations is as follows:

- **Initialization of system services** (Kernel, Table, Buffer and Queue Services)
- **QMU Configuration –** Each CP is allotted 16 queues each.

CP	Description	
ATM CP	Base Q	ATM cell descriptors from AAL2 TX CP
	Base Q + 1	
	…	
	Base Q + 15	
AAL2 Rx CP	Base Q	ATM cell descriptors from ATM CP
	Base Q + 1	
	…	
	Base Q + 15	
AAL2 Tx CP	Base Q	CPS packet descriptors from RTP to AAL2 Map CP
	Base Q + 1	CU timer expiry messages from XP
	…	
	Base Q + 15	
RTP/UDP CP	Base Q	IP Descriptors from IP/Ethernet CP
	Base Q + 1	RTP/UDP Descriptors from AAL2 to RTP map module
	…	
	Base Q + 15	
Aal2 to RTP Map CP	Base Q	CPS Packet Descriptors from AAL2 RX CP
	Base Q + 1	
	…	
	Base Q + 15	
RTP to Aal2 Map CP	Base Q	RTP Descriptors from RTP/UDP CP
	Base Q + 1	
	…	
	Base Q + 15	
IP CP	Base Q	IP Descriptors from RTP/UDP CP
	Base Q + 1	
	…	
	Base Q + 15	
XP	Base Q	CU Timer Start Request messages from AAL2 TX CP
	Base Q + 1	SONET monitoring messages from ATM CP
	Base Q + 2	OAM/Control messages from ATM CP

	Base Q + 3	Reserved CIDs (1 to 7) and Alarms messages Queue from AAL2 to RTP Map CP
	Base Q + 4	ICMP messages from IP CP
	….	
	Base Q + 15	

- **BMU Configuration**

The following table describes the buffer pools that need to be created.

Number of pool instances	Attached to CP	Contents of the buffer pool	Number of buffers per pool	Size of buffer	Total buffer requirement	Total bytes
1	ATM CP	Incoming ATM SDUs	4096	64	1*4096*64	262144
2	AAL2 Rx CP	CPS SDUs	4096	64	2*4096*64	524288
3	AAL2 Tx CP	Outgoing ATM SDUs	2048	64	3*2048*64	393216
3	AAL2-RTP Mapping CP	RTP Payload	4096	256	3*4096*256	3145728
3	RTP-AAL2 Mapping CP	CPS SDUs	4096	64	3*4096*64	786432
2	RTP/UDP CP	IP PDUs and RTP PDUs	16384	512	2*16384*512	16777216
2	IP CP	IP SDUs	8192	2048	2*8192*2048	33554432

Total SDRAM requirements = 55443456 bytes = 52.875 MB

- **Table Creation**

The following tables need to be created:

- Existing HTK table in existing Aal2Switch App in the ATM module, which is indexed by

12 bits	16 bits	4 bits
VPI	VCI	CpId

The Hash Trie Key Table values shall be modified to support 1K VcIndices. QOS related columns are removed from the table.

Number of entries in Key Table = 1K
Number of Entries in Trie table = 0.5 K
Number of Entries in Hash Table = 2 * 1K

- New HTK table for module that maps from AAL2 to RTP, which is indexed by

8 bits	16 bits	8 bits
PAD	VcIndex	CID

Number of entries in Key Table = 8K
Number of Entries in Trie table = 4K
Number of Entries in Hash Table = 2 * 8K

- New HTK table for module that maps from RTP to AAL2, which is indexed by

32 bits	16 bits
IP Address	UDP Port No

Number of entries in Key Table = 8K
Number of Entries in Trie table = 4K
Number of Entries in Hash Table = 2 * 8K

- New Simple table for timer support of AAL2 to RTP Map Module, which is indexed by

16 bits
CallId

Number of entries in Simple Table = 8K

- **Configuration of SDPs**

1. The SDP (TxByte, TxBit, RxSync, RxBit and RxByte Processor) associated with ATM CP should be configured for OC-3c configuration
2. The SDP (TxByte Processor and RxByte Processor) associated with AAL2 Rx, AAL2 Tx should be configured in Byte-wise recirculation mode.
3. The SDP (TxByte, TxBit, RxBit and RxByte Processor) associated with IP CP should be configured for 100 Mbps Fast Ethernet configurations.

- **Load the CPs and Start the CPs**
- **En-queue the Init Descriptors to each of the CPs**
- **Start the SDPs after starting the CPs**
- **In a while loop {**
  - Process CU Timer start descriptors for AAL2 Tx CPs
  - Sonet related Processing
  - ATM Cells with Payload Type Indicator value equal to non-zero related processing
  - CPS Packets with Reserved CIDs(CIDs from 1 to 7) and UUI value = 31 for OAM alarms processing
  - IP related Control Information Processing
    - Only ICMP messages of type =3 and code =3 shall be processed. The IP Address and UDP Port number are used as a key to launch a table lookup to reset RTP Timestamp, RTP Sequence number values and after ICMP Payload Buffer shall be freed. This mechanism will be used to identify the call closure from the IP end-point.

  }

9.2 Interfaces

The following interfaces exist for XP:

- Interface with all CPs, based on the init descriptor.
- CU Timer start command Interface with AAL2 TX CP, based on the timer start descriptor.
- Sonet Messages from CP, based on Sonet Message Descriptor
- Control and OAM cells from ATM CP,
- Control CPS packet Descriptors (CPS Packets with CIDs from 1 to 7 or UUI = 31)from AAL2 to RTP Map CP,
- ICMP messages from IP CP
- CU Timer expiry message interface to the AAL2 Tx CP.

9.3 Data Structures

The Init Descriptor sent to the AAL2 Rx module by the XP is as follows:

Typedef struct

```
{
   BsPoolId    poolId;
   QsQueueId   destId1; /* Destination Ids of AAL2 to RTP MAP CPs */
   QsQueueId   destId2;
   Int16u      VcIndexLimit; /* Used to identify the MAP-1 CP uniquely */
} Aal2RxCpInitDesc;
```

The Init Descriptor En-queued to the AAL2 TX module by XP shall be modified as below:

```
Typedef struct
{
   BsPoolId    poolId;
   QsQueueId   destId; /* Queue ID of the ATM CP */
    Int8u      Aal2TxNumber; /* To calculate index on DMEM Table */
} Aal2TxCpInitDesc;
```

The Init Descriptor sent to the AAL2 to the RTP Map module by the XP is as follows:

```
Typedef struct
{
   BsPoolId    poolId;
   TsTableId   htkTableId;                  /* For RTP context */
   TsTableId   simpleTableId;               /* For assembly timer */
  QsQueueId    aal2CpsControlQId;           /* For Alarm and CID 1…7 AAL2 packets */
   Int8u       encodingProfileIdSelected; /* Encoding Profile ID Selected as per I.366.2 Annex P */
} Map1CpInitDesc;
```

The Init Descriptor sent to the RTP to the AAL2 Map module by the XP is as follows:

```
Typedef struct
{
   BsPoolId    poolId;
   QsQueueId   aal2TxQueueId;
} Map2CpInitDesc;
```

The Init Descriptor sent to the RTP/UDP CP by the XP is as follows:

```
Typedef struct
{
   TsTableId   tableId;
   BsPoolId    PoolId;
   QsQueueId   ipQueueId;
   Int32u      ethernetPortIpAddress;
   Int16u      udpSourcePortNo;
} RtpUdpCpInitDesc;
```

The Init Descriptor sent to the IP CP by the XP is as follows:

```
Typedef struct
{
   BsPoolId    PoolId;
   QsQueueId   rtpUdpQueueId;
   QsQueueId   icmpXpQueueId;  /* The queue Id of XP to which ICMP messages shall be sent to */
   Int32u     macInternalAddressHi4;
   Int16u     macInternalAddressLo2;
   Int32u     macDestAddressHi4; /* MAC Address for the next-hop router, */
   Int16u     macDestAddressLo2; /* which will be hard-coded for both of the Ethernet ports */
   Int8u        fabricEnabled;
   Int8u        fabricId;
} IpCpInitDesc;
```

The Init Descriptor sent to the ATM CP by the XP is as follows:

```
typedef struct {
   TsTableId   tableId;
   BsPoolId    poolId;
} AtmAal2CpInitDesc;
```

The table entry for the HTK table used by ATM module is as follows:

```
Typedef struct {
   Int8u    Flags;
   Int8u    pad1;
   Int16u   VcIndex;
   Int16u   QueueId;
   Int16u   pad2;
   Int16u   Vpi;
   Int16u   Vci;
   Int32u   atmEgressKey; /* Contains the VPI, VCI and port#  */
} VcUnicastIngress;
```

The table entry for the HTK table used by module that maps from AAL2 to RTP is as follows :

```
Typedef struct {
Int8u     oldEncodingType; /* RTP Payload Type */
Int8u     expectedAal2UuiValue;     /* Expected Aal2 UUI or Sequence Number */
Int16u   presentRtpSeqNumber; /* RTP Sequence Number */
Int32u   presentRtpTimeStamp;  /* RTP TimeStamp */
BsBufHandle rtpPayloadBufHandle; /* Buffer Handle of the assembly buffer */
Int16u rtpPayloadBytesCount; /* Offset in the Buffer */
Int8u    countAal2PacketsNeeded; /* Count of AAL2 packet needed in the RTP packet */
Int32u ipAddress; /* IP Address */
Int16u udpPortNo; /* UDP Port No */
QsQueueId rtpUdpQueue; /* Destination RTP/UDP Cp Queue ID*/
Int32u   rtpSsrc;              /* RTP SSRC */
Int16u   callId;               /* CALL ID */
Int8u      receiversCapability; /* Receiver's Capability of supporting whether 5msec or 10 msec or
20msec or 30 msec RTP Payload*/
} Aal2ToRtpMapTableEntry;
```

The table entry for the HTK table used by module that maps from RTP to AAL2 is as follows :

```
Typedef struct {
Int16u vpi; /* Destination Vpi */
Int16u vci; /* Destination Vci */
Int8u   cid;  /* Destination Channel Id */
Int16u vcIndex; /* Destination VcIndex */
Int8u portNo; /* Port Number */
Int16u   callId; /* Call ID */
Int32u   expectedRtpTimeStamp;  /* Expected RTP TimeStamp */
Int16u   expectedRtpSequenceNumber; /* Expected RTP Sequence Number */
Int8u      nextAal2UuiValue; /* Next Aal2 UUI Value */
Int8u      last RtpPayloadType; /* Last RTP Payload Type */
BsBufHandle fragAal2BufHandle; /* Buffer handle for the fragmented RTP payload */
Int8u     fragAal2Count;  /* Previous count of bytes stored in fragmented RTP payload buffer */
QsQueueId rtpToAal2MapQueue; /* Destination RTP to AAL2 Map CP Queue*/
Int8u     callPresentFlag;     /* Will be reset to 0, by XP using ICMP message processing, will be set to 1 by
          Map-2 */
} RtpToAal2MapTableEntry;
```

The table entry for Simple table for timer support of AAL2 to RTP Map Module is as follows:

```
Typedef struct {
Int16u VcIndex; /* Destination VcIndex */
Int8u   cid;  /* Destination Channel Id */
} Aal2ToRtpTimerTableEntry;
```

The SONET message descriptor en-queued to the XP by the ATM CP is as follows:

```
Typedef struct {
   PsSonetEvent       newEvents;
   PsSonetDefect      newStates;
   int8u              c2PathSignalLabel;
   int8u              traceErrorState; /* J0, bit 1, J1 bit 0 */
   int8u              portId;
   int8u              rdiPathType;
   int8u              S1;
} SonetMsg;
```

The ICMP Descriptor enqueued to the XP, if type = 3 and code = 3 is as follows:

```
/* ICMP Descriptor */
typedef struct {
   BsBufHandle bufHandle;
   Int16u length;
   Int16u VNID_client;
   Int32u appData1; /* IP Address */
   Int16u appData2; /* UDP Port Number */
   Int16u appData3; /* ICMP Type and ICMP Code */
} IcmpDescriptor;
```

The Timeout descriptor en-queued by XP to AAL2 Tx is as follows:

```
Typedef struct _TimeoutDesc
{
   Int8u       seqNo;
   Int16u      vcIndex;
} TimeoutDesc;
```

The Start Timer Descriptor structures used by AAL2 Tx CP to en-queue Timer Start command to XP is as follows :

```
Typedef struct _TimerStartData
{
   Int8u       seqNo;
   Int8u       cpId;
   Int16u      vcIndex;
} TimerStartData;

Typedef union _XpInQueueData
{
   TimerStartData timerStartCommand;
} XpInQueueData;
```

```
Typedef struct _XpInQueueDesc
{
   Int32u       command;
   XpInQueueData data;
} XpInQueueDesc;
```

10. Feature Requirements Trace Matrix

Sl. No.	FRAS Requirement-Id	Design Section	Remarks
1.	R-4.1.1	NA	CDS platform will be used.
2.	R-4.2.1	NA	Proper C-5 board will be used.
3.	R-4.2.2	NA	Proper AAL2/ATM PVC terminations will be configured.
4.	R-4.3.1	NA	These standards were referred for the design.
5.	R-4.4.1	Sections: 2.5.4, 2.5.5. Chapters: 7, and 8.	
6.	R-4.4.2	Sections: 2.5.4, 2.5.5. Chapters: 7, and 8.	
7.	R-4.4.3	Sections: 2.5.4, 2.5.5. Chapters: 7, and 8.	
8.	R-4.4.6	Sections: 2.5.4, 2.5.5. Chapters: 7, and 8.	
9.	R-4.4.9	Chapter: 7, Section: 7.1.2 Chapter: 9, Section: 9.1	
10.	R-4.5.1	Chapters: 7, and 8.	
11.	R-4.5.2	Chapters: 7, 8, and 9.	
12.	R-4.6.1	Chapter: 8, Section: 8.1.2	
13.	R-4.6.2	Chapter: 8, Section: 8.1.2	
14.	R-4.6.3	Chapter: 8, Section: 8.1.2	
15.	R-4.6.4	Chapter: 8, Section: 8.1.2	
16.	R-4.6.5	Chapter: 7, Section: 7.1.5	
17.	R-4.6.6	Sections: 2.5.4, 2.5.5. Chapters: 7, and 8	
18.	R-4.6.7	Chapter: 7, Section: 7.1.5	
19.	R-4.6.8	Chapter: 7, Section: 7.1.5	
20.	R-4.6.9	Chapter: 7, Section: 7.1.5	
21.	R-4.6.10	Chapter: 8, Section: 8.1.2	
22.	R-4.6.11	Chapter: 8, Section: 8.1.2	
23.	R-4.8.1	Chapter: 7, Section: 7.1.2 Chapter: 6, Section 7	
24.	R-4.8.2	Chapter: 7, Section: 7.1.2	
25.	R-4.8.3	Chapter: 7, Section: 7.1.5	
26.	R-4.9.1	Chapter: 8, Section: 8.1.2, and 8.1.5	
27.	R-4.9.2	Chapter: 8, Section: 8.1.2, and 8.1.5	
28.	R-4.10.1	Chapter: 7, Section: 7.1.2 Chapter: 8, Section: 8.1.2	
29.	R-4.10.2	Chapter: 7, Section: 7.1.5 Chapter: 8, Section: 8.1.2	