Prepared (also subject responsible if other)		No.		
UABTOJN		4/1551-APZ 214 03/1 Uen		
Approved	Checked	Date	Rev	Reference
EAB/FLE/T [Olav Tveite]		2011-03-08	A	

XPU DESCRIPTION

1 Revision History

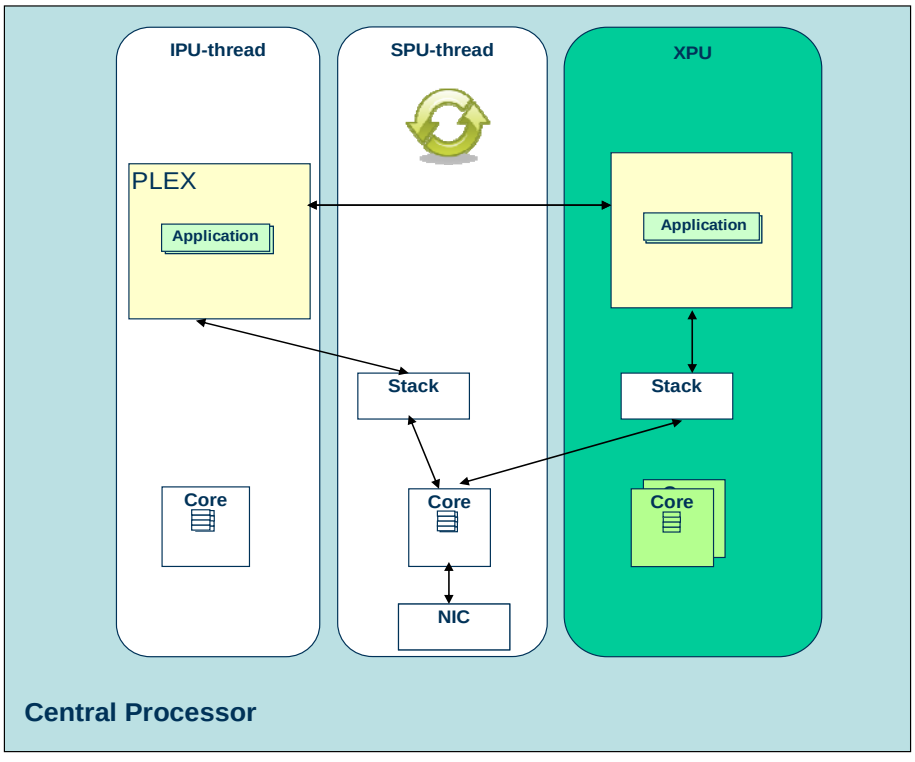
Rev	Date	Author	Changes
A	2011-03-04	uabtojn	First version of the document

2 Introduction

2.1 General

This document describes the XPU (eXtra Processing Unit) -- an additional execution environment in the AXE Central Processor (CP).
The XPU is initially introduced for the CP Cluster System.

2.2 XPU Overview



The XPU environment provides a Linux execution environment in the Central processor of AXE. The APZ Virtual Machine (APZ VM), that provides the PLEX domain in the CP, utilizes two cores. The remaining cores available on modern processors can then be utilized by the XPU domain.

Before the introduction of XPU, the CP consisted of the PLEX environment. With the XPU, the CP consists of two domains, the PLEX domain and the XPU domain.

The XPU runtime environment provides:

- An execution environment in Linux
- A PLEX – XPU communication interface
- TCP/IP communication through an IP-stack.

The XPU applications shall off-load the PLEX domain from CPU intensive tasks. It also provides an environment to execute tasks that are not suitable to implement in PLEX, for instance protocol termination.

It is possible for XPU applications to communicate with it's peer in the PLEX domain. It is also possible to communicate directly with the external world with means of IP.

From an O&M perspective, the XPU can be handled as a separate load unit that is decoupled from the CPHW and APZ VM deliverables.

2.3 Definition of XPU and XM Concepts

The XPU is an extension of the existing execution environment in the AXE CP which makes use of spare cores in the GEP2 and future processors. It is a satellite to the traditional PLEX/APZ VM environment executing in the CP.

The PLEX/APZ VM uses two cores, one for the IPU and one for the SPU, all additional cores are allocated to the XPU. Despite the close relationship with IPU and SPU and the fact that they share the same Linux Operating System instance, hardware etc, the XPU is not a part of the PLEX/APZ VM domain.

The XPU is managed by the PLEX domain. This includes configuration, start/restart, error handling etc. When cloning is performed, the XPU domain is restarted using the configuration included in the new PLEX clone.

The term XPU refers to the system solution (concept) and includes all parts involved to enable the use of the spare cores in the CP.

The XPU environment provides means for internal and external communication, software and fault management.

The XPU software is delivered as an XPU Software Unit, XSU.

The XSU is instantiated in one or several XPU Modules (XM).

An XM may contain several processes or threads.

An XM be can be allocated to one or several cores.

More than one XM can be allocated to the same core.

An XPU is local to the cluster CP, it does not span over a CP cluster.

The XPU does not provide means for storing persistent data for the XPU applications, the PLEX domain must be used for this.

3 Allocation of Functions

The XPU is implemented in a number of different functions briefly described below. Only a few of these are “Function Blocks” in the traditional AXE terminology.

3.1 XPU Execution Environment

The “XPU execution environment” is implemented in CPHW subsystem and provides the Linux environment needed to execute XPU applications. It consists of the Linux OS, XPU bootstrap, PARENTX, IPLIBX, UTILX and shared libraries.

Linux OS: The Linux OS is based on SLES (Suse Linux Enterprise). It includes standard Linux Libraries as well as XPU specific libraries used to provide XPU functionality. The OS is delivered in the bootimage included in the CPHW dump.

XPU bootstrap: The purpose of the XPU bootstrap is to load and start PARENTX. The XPU bootstrap process is always running in the system and is also delivered as a part of the BOOTIMAGE.

PARENTX: PARENTX is responsible for loading and starting of all other XMs than itself.

IPLIBX: IPLIBX provides a IP stack that can be used by applications for TCP/IP communication.

UTILX: UTILX provides debugging and load measurement capabilities for the XPU.

PARENTX, IPLIBX and UTILX are delivered as three different XSU and managed in the system as three separate XMs.

3.2 XPU Administration

The XPU administration is implemented in PLEX block XPULOAD and XPUADM belonging to PES (PLEX Engine Subsystem)

XPULOAD is handling data storage for XPU Software Units (XSUs). It also manages the commands to load XSUs and to and print XSU data.

XPUADM manages configuration, deblocking and blocking of XMs. XPUADM is also holding entry points for traffic signals used by application, but actually received by APZ VM.

3.3 APZ VM Support

APZ VM in PES includes functions for sending signals to and from the XPU. As the APZ VM and the XPU applications are separate processes that should not be able to read o write in each other memory, data is copied when transferred between the PLEX and XPU domain. This copy operation is implemented by a shared memory that functions as a queue between PLEX and each XM.

APZ VM includes functions to share the Ethernet infrastructure between the IP stack used in the XPU, the IP stack used in APZ VM and the Kernel IP stack. APZ VM also includes commands for managing IP configuration.

3.4 XPU Hardware and Software Recovery

The XPU SW recovery function is implemented in the PLEX block MXPREC belonging to MAS (Maintenance Subsystem).

The function XPU Hardware and Software Recovery initiates recovery actions when faults are reported in the XPU.

All errors related to the XPU are reported to the XPU SW recovery function. It is the single point to collect XM error reports and to decide and initiate any specific XPU recovery actions, in cooperation with the ordinary CP SW recovery.

In the current implementation, a fault in one XM will cause a system restart. However, the application interface is prepared to handle restarts of individual XMs. This enables a future improvement where a fault in an XM only forces a restart or blocking of the individual XM instead of restarting the complete system.

4 XPU Execution Platform

The XPU application environment has the following characteristics:

- Runs on Enux, which is a special version of Linux, currently SLES 11.
- Executes on one or several dedicated cores.
- The XPU cores are located in the same processor as the APZ VM, but the APZ VM and XPU does not share any cores.

In order to somewhat limit the impact an XPU application will have on the CP, some limitations/restrictions is forced on the application. The XMs are for instance run as an own user, and this user will have some restrictions. Core affinity is also set on the XM to prevent the XM to run on the IPU or SPU core.

XPU provides a standard linux environment and dynamically linked libraries. The XPU specific libraries that provides support functions that relieves the application design from implementing stuff that must be done by every application. These are functions for interwork with the CP, the IP stack, Fault Management etc.

The library contains functions for signal sending/reception, and all communication with the PLEX domain or the IP stack is done through this library. Reception of signals is signaled through a file descriptor, and the application can then use select or epoll to monitor the file descriptor.

The XPU applications are own processes that execute beside the APZ VM. The main reason is to decouple the XPU from the APZ VM, prepare for separate crash domains and get a memory protection between the XPU and APZ VM.

4.1 XPU Application Interfaces

The main interface is the functionality that is provided by the operating system.

The following interfaces are also provided to the application XPU.

1. PLEX interface: A PLEX communication interface is provided and includes functions for sending and receiving signals to/from the IPU/PLEX domain.
2. TCP/IP interface: An IP stack is provided for external communication. The IP stack interface include a standard POSIXs compliant socket interface and an asynchronous socket interface.

3. Error/event reporting interface: An error/event reporting interface is provided. This is used for the application to log event or errors in runtime and store the output in a file in the AP.

The interfaces are further described in reference [8] and [9].

4.2 Build Support and SEA

The XPU VDE (Virtual Development Environment) and XPU SDK (Software Development Kit) provides an environment to compile and package the XPU application. Even though the XPU is a Linux application, there is a dependency to which version of libraries that is used in the bootimage. This mean that a version controlled environment is needed to compile the application.

The XPU VDE/SDK provides possibilities to basic test the XSU and test the XPU application in SEA.

The development environment is further described in reference [6]

5 XM Owning PLEX Applications

XMs that need to communicate with the PLEX domain in the CP needs an XM owning block. It is not mandatory for an XM to have a XM owning PLEX block.

XM owning PLEX applications must honor a few maintenance signals, for instance received at XM definition and start of an XM.

Only an XM owning PLEX block can send signals to the XM and receive signals form the XM. Signals can only be sent to and received from the “own” XM.

A single PLEX block can “own” several XMs, and these XMs can use the same XSU or use different XSUs.

The PLEX interface is further described in reference [4] and [5].

6 XPU Administration

The XPU administration is implemented in PLEX block XPULOAD and XPUADM belonging to PES.

XPULOAD is handling data storage for XPU software (XSUs). It also manages the commands to load XSU from the APG to the CP and print XSU data.

The XPU Software Unit (XSU) is loaded from the CP file system on AP to data store in the CP by a new command, LAXUL. Command LAXIP can be used to print data for XSU's loaded in data store.

XPULOAD also provide check summing of the XSU, and issue an alarm if the XSU becomes corrupt.

XPUADM manages configuration, deblocking and blocking of XMs. XPUADM is also holding entry points for traffic signals used by application, but actually received by APZ VM.

XPUADM provides a command EXXPI, to define and tie an XM with an XPU software unit together with a CP application block. It is also possible to tie the XM to a specific core and set a memory limit for the XM.

To load the XPU software to Linux file system on the CP and start the XM a new command BLXPE is used. This is synonymous with deblocking the XM.

XPUADM will also provide command EXXPP for printing of data related to XPU administration as well as a command EXXPE for removal of the definitions done with command EXXPI. Command EXXPC provides possibility to change some of the parameters set in EXXPI.

When an XM has been defined and deblocked, the XM is considered part of the system so that start, restart, supervision, program loading and other cooperation towards the XM is possible from the PLEX environment.

A new command is also introduced to block the XM, BLXPI.

6.1 Function Change

Function change of XPU software is done by command EXXPC. The XPU must be blocked by command BLXPI before changes can be made by command EXXPC.

6.2 Start and Restart

APZ VM will terminate all XMs at system restart. The restart of the XM will be managed by PLEX (XPUADM).

The start of an XM is done on initiative and controlled by PLEX (XPUADM). The start will consist of loading the XSU to Linux file system and start of the application.

XPUADM is responsible for performing the start of all XMs at all types of system starts/restarts as well as start of indicated XMs at command BLXPE. However, for each XM to start XPUADM will always first ask permission from MAS to check that there are no fault handling reasons (“Fault Marking”) to omit start of the XM in question.

At system restart, the start of XMs is initiated in absolute restart phase 4. XPUADM then waits in absolute phase 20 to conclude the start of all XMs.

If there is an XM owning PLEX block, the XMs are only started if the XM owning PLEX block is started. This mean that after an initial load before APT is started (command SYATI), the XM will be in state AB (Automatically blocked).

6.2.1 Start Sequence of XPU

The load and start of XMs are divided between bootstrap and PARENTX. The bootstrap process will be used to load and start the PARENTX, and PARENTX will then be used to load and start all other XMs. The reason for the divisions between bootstrap and PARENTX is to include as little functionality in the CPHW/APZ VM deliverables as possible. The System down time is much less when a XSU is replaced than if CPHW/APZ VM is replaced, thus this approach reduces the downtime for changes in the XPU environment.

PARENTX and IPLIBX XMs are referred to as APZ XMs while application XMs are referred as APT XMs. APZ XMs are always started before the APT XMs.

The start of the XPU is done in the steps described below:

1. XPULOAD fetches the XSU for PARENTX from XPULOAD and sends it to the bootstrap process. The bootstrap process extracts the unit and verifies that the PARENTX XSU is compatible with the BOOTIMAGE.

2. XPUADM orders the bootstrap to start PARENTX. The bootstrap than perform “fork” and “exec” on the PARENTX. When PARENTX is started, it send a reply to XPUADM. XPUADM is waiting for this reply before it continues the deblocking sequence.

3. XPUADM loads all other XMs that should be started via PARENTX. PARENTX then extract the XSU and verifies that the XSU is compatible with the BOOTIMAGE.

4. XPUADM orders start of APZ XM,s (IPLIBX and UTILX). PARENTX then performs “fork” and “exec” for the XMs. XPUADM waits until all APZ XMs are started before it continues.

5. XPUADM order start of APT XMs one by one starting with the lowest XM number. Each XMs must send a signal to XPUADM to indicate that it is started. The next XM will be started when the previous XM have signaled that it is started.

A fault in an XM during the deblocking sequence will be considered as a failed deblocking. If a fault occur after the deblocking sequence is completed, a fault will be handled as normal and be escalated to a system restart.

6.3 System Backup

The XSU's and the XM definitions are included in the PLEX software dump, and consequently restored at a system restart or reload.

6.4 Compatibility Control of XSU's

The Linux environment in which the XMs are executing is delivered in the CPHW dump. As the CPHW dump is not included in the PLEX dump, there might be incompatibility issues between the BOOTIMAGE and the XPU applications included in the PLEX dump. This can for instance occur if the BOOTIMAGE is updated to a later Linux release which requires recompilation of XPU applications. To prohibit unpredictable behavior, the XSU software is checked for compatibility.

The compatibility control is performed at two levels. the first level is the HCN (Hardware compatibility number) and SCN (software compatibility Number) that can be seen in "XPU SOFTWARE UNIT INFORMATION" printout (command LAXIP:FULL). The HCN will only change if a completely different processor architecture will be used (e.g. Power PC). In practice, the HCN will remain 1. The SCN will change when a major change in the software compatibility (e.g. replace OS that require recompilation of applications) or a change in the XSU format. If the SCN or HCN is not correct, the deblocking will fail before the XSUs are transferred from PLEX to the XPU domain.

The second level is the version control that is included XSU. When an XSU is created, a versions.txt file is automatically created and included in the XSU. This file contains revisions and compatibility numbers for the VDE and SDK. When the XSU has been transferred from PLEX to the Linux environment at deblocking, the version numbers used to build the XSU are compared with the numbers used to build PARENTX and IPLIBX. If the numbers are not correct, the deblocking will be rejected.

The VDE version number is mainly included to be able to track what version of the VDE that was used to create the XSU.

The SDK compatibility number is used to verify that the header files used to build the XSU are compatible with the runtime environment. This number will be changed if an incompatible change is made in any interfaces.

7 XPU – IPU Communication

Signals sent between PLEX and the XMs consist of register data (PR0, WR0-WR23) and an optional communication buffer (CB). The 16 lowest bits of PR0 is used to address the XM, and the XM number is used as the address. The remaining 96 (4 bytes * 24 working registers + 2 bytes in PR0) can be used for data by the application. When a communication buffer is included, WR0, WR1 and WR2 are used to pass the CB pointer, offset and size of the CB. The communication buffers can be from 128 W16 up to 32 K W16.

Signal numbers is not supported in the PLEX-XPU interface. This means that all signals will be received in the same receive statement. However, each Communication buffer size require an own receive statement in PLEX.

When an XM is defined (command EXXPI), there is a record in block XPUADM which is updated with information about the XM identity, which XSU that is tied with the XM and block number for the XM owning PLEX block. The XM owning PLEX block is also informed about the new XM by a signal from XPUADM. The PLEX application will then know which XM number its XM is allocated to.

When the XPU software has been loaded and started, a deblocking signal is sent to the CP application block to inform that it is allowed to send signals to the XM. Whenever the XPU software is stopped or blocked, a blocking signal is sent to the CP application block.

When the CP application sends a signal to its XM, a combined signal is sent to XPUADM which receives the signal with data including XM number. A Clayton part in XPUADM performs some checks, for instance that the XM is in working state and that the signal comes from the XM owning PLEX block. The signal is then transferred to the XPU Protocol Handler (XPU-PH) and the signal is inserted into the queue connected with XM. Note that there are one queue for each XM.

The APZ VM (where the PLEX is running) and the XMs are separate processes. In order to obtain memory protection, the APZ VM and the XMs are not allowed to access each others memory. To send a signal to an XM, the signal is therefore copied from APZ VM into a shared memory. When the signal is received in the XM application, the application must then copy the data from the shared memory. This shared memory acts as a signal queue to and from the XM.

If for some reason the XM application fetches signals slower than the PLEX application send signals, the shared memory will be congested. In order to protect the shared memory to be fully exhausted, a credit mechanism is used between the Clayton part of XPUADM in the IPU and the shared memory in the SPU. This means that the XPUADM can only send as much data that can be inserted in the shared memory. If XPUADM does not have enough credits to send a signal to the shared memory, the application will be notified immediately and should stop sending until the congestion ceases. The notification is made in the combined backwards signal of the XMSEND. If a communication buffer was included in the signal, it will be returned to the application. When the congestion ceases, the PLEX application will be notified about that by means of a PLEX signal XMSENDRESUME.

The PLEX to XPU communication is reliable and no signals will be lost. Loss of signals can only occur if the XM is restarted.

8 Fault Management

All XPU errors are reported to PLEX block MXPREC in MAS. MXPREC then decides what actions that shall be taken.

8.1 Error Detection

Application detected errors: If an XPU application detects an error, the application can register an event (Non fatal error) or report an error (fatal error) depending on the severity. An event will only cause a logging, while an error will cause a system restart. See chapter 8.4, XPU Logging.

OS detected errors: Some errors in the XPU application will be detected by the OS, for instance memory violation. When these errors occur, the application will be terminated by the OS. The PARENT XM supervises the XMs and will detect that he XM is terminated. PARENTX will then report this to XPU administration (XPUADM) which will forward the information to MAS.

Watchdog: The XPU applications are supervised by a watchdog mechanism. A signal is regularly sent to the XM, and the XM must then respond to this signal. If a XM fails to respond it will be considered as non working and terminated. This polling is sent from block XPUADM and is done with a few seconds interval and the application must reply within a few seconds.

The watchdog can be disabled for debugging purposes and is disabled by the same way as the CP PHC function (command PTPHS/PTPHR). When the watchdog is disabled, the applications are still pooled, but will not be terminated if it fails to respond to a poll.

8.2 Error Recovery

PLEX block MXPREC is responsible to decide what actions that will be taken at XPU errors. The fault handling is simplified to always connect the XPU fault handling to a CP system restart. This means that an error in an XM will always cause a System restart. It also means that whenever the CP is restart (regardless if the restart is due to an XM fault or not) all XMs will be restarted as well.

The XM termination at a system restart is performed by APZ-VM.

The XM restart is done through the normal start procedure by XPU Administration (XPUADM). MXPREC is not involved in the start other than giving permission to XPUADM to start the XMs.

The restart actions will be escalated if the XMs fail to recover during the system restart or if the error is re-occurring and causes cyclic faults. The restart escalation follows the normal flow for restart escalation.

In the Blade Cluster System the Cluster CP will permanently leave cluster CP state active if the XM does not recover after the restart attempt(s). A permanent fault in an XM will bring down the whole CP since the XPU is defined to be an unconditional part of the CP environment.

At the final escalation step when the CP is put to HALT, also the XMs are effectively halted. The APZ VM will terminate all XMs before entering the stop loop at HALT.

The CLUSTER CP FAULT alarm may indicate reason XM FAULT if repeated XPU faults have been reported during the recovery escalation

8.3 Traffic Isolation and CP Stoppage

There is no specific support in APZ for traffic isolation of the XPU applications. APT need to implement any necessary XM isolation actions at reception of the CP isolation request. The XMs are not stopped by APZ at traffic isolation.

Also note that when APZ has requested traffic isolation by APT for certain maintenance activities the cluster CP execution including the XPU may subsequently be stopped by following APZ activities.
For instance the monthly test will perform reboots of the CP as part of the tests.

8.4 XPU Logging

A few different logs can contain information related to the XPU that are useful for troubleshooting.

8.4.1 XPU Log

The AP contains a log called XPU log. This log is used for XPU applications to log events that are useful for understanding the behavior of the application and to analyze faults in the XPU application. This log is often used during normal execution and registrations in this log does not indicate a fault or abnormal situation in the XPU application.

The log can be viewed by AP command xpuls and transferred by command xputran.

8.4.2 XPU Core Files

XPU applications that are terminated due to an error will cause a Core dump file to be created. The core file contains a copy of the memory used by the application at the time of the restart and is vital for understanding the reason for a failure in an XPU application.

The core files can be fetched using the xputran AP command.

8.4.3 Software Recovery Log

XPU applications can register event in the software recovery log (command SYRIP:LOG). The registrations in this log indicate an abnormal event in the application that should not occur normally. In order not to flood the software recovery log with XPU related events, a maximum of 15 events related to the XPU will be recorded for a 30 day period.

Events recorded in the Software recovery log is also stored in the XPU log on the AP.

8.4.4 Restart Information

Faults in the XPU that results in a system restart. As for any other system restart and a log can be found in the restart information printout, (command SYRIP). The restart information contains information which XM that initiated the restart and error information from the application. Error codes reported from APZ are explained in the RESTART INFORMATION POD.

8.4.5 APZ VM and CPHW Logs

Error and Event detected by CPHW (the OS) or APZ VM will be managed by the Central Log Handler (CLH) on the APG. The following logs are of interest for the XPU:

Log entries generated by the XPU bootstrap are sent to syslog.

Event messages from APZ VM (XPU-PH) are stored in VM event log.

Error messages from APZ VM (XPU-PH) are stored in VM error log.

All these logs can be viewed and transferred using the clhls or clhtran AP commands.

8.5 Measurement of XPU Load

Load measurement can be done by use of UTILX. The UTILX XM will act as a telnet server on the CP, and can be connected to by running telnet on the AP. The utilx XM provides a ps command to display the load generated by running XMs.

See reference [1] for more information on using UTILX.

9 TCP/IP Communication

IP based communication is one of the important tasks for the XPU and the XPU application have access to a TCP/IP stack.

The TCP/IP stack is implemented in the IPLIB XM, and is loaded as an XPU application. This provides easy update of the TCP/IP.

Administration of the IP stack (configuring IP addresses, etc.) is be done by the same commands as used to configure the IP stack in the PLEX domain (IHIFI, IHIFC etc.) PLEX block VMIPADM is acting as an XM owning PLEX block for IPLIBX. When IPLIBX has been deblocked, VMIPADM sends configuration data to configure the TCP/IP stack. This configuration is concluded before other XPU applications are started.

The IP stacks in the PLEX and XPU domain share the same physical interfaces, and the incoming traffic is routed to the correct stack based on VLAN. This means that a specific VLAN can’t be shared between the IP stack in the SPU (PLEX) and XPU domain.

The Statistics functionality that exists for the IP stack in the PLEX domain is not supported for the XPU.

10 Overload Protection of XPU

The load regulation and overload protection must be handled differently depending on where the traffic originates from. The XPU platform has no knowledge on how to regulate the load for each application. For this reason the XPU platform does not provide an end to end solution for load regulation and overload protection, but instead provides the mechanism that the application can use to implement these.

The overload protection of XPU has dependency on

- IPU-XPU signaling
- External communication

10.1 Flow Control for Outgoing Signaling

Flow control of outbound signals is used to protect the shared memory between APZ VM and the XPU and is based on a credit system between XPUADM-Clayton and XPU-PH. XPUADM has information coming from XPU-PH about a certain amount of data that can be transferred to each XM at any moment. When this amount has been used up, further transfer is suspended and XPUADM is rejecting the order back to the application with result code indicating congestion. When XPU-PH has detected that new space is available for the signal queue in an XM, a message with credits goes from XPU-PH back to XPUADM which sends a resume indication to the application in question.

Each XM has its own queue and gets individual information. If an application controls multiple XMs, it can happen that one XM is congested and the others are running without problems.

Note that no correlation is made between the occupancy of PLEX-XPU buffer and the transmit buffer of the TCP/IP stack. Therefore it is a responsibility of the XPU application to apply backpressure towards the PLEX block at congestion in the TCP/IP stack.

If the transmit buffer of the TCP/IP stack gets congested due to, for example, a congested network, the send buffer will become congested and the application will be notified. When this happens the application can buffer signals in the XPU or apply backpressure towards the PLEX block.

Watermark can be utilized for the buffers in the TCP/IP stack. The applications will then be informed, on a per connection basis, that the buffer is filled up to the watermark level. In such a case the application shall start to back off and decide to do buffering in the XPU or send congestion indication to the corresponding application block in the IPU (or both) before congestion occurs.

10.2 Flow Control for Incoming Singaling

APZ VM has a “Stop Scan” function which prevents signaling from external peers to cause overload in the CP. When the number of signals in the JOB buffers exceeds certain levels, the APZ VM will stop fetching signals from the external peers. This mechanism also applies for signaling from XPU to PLEX. This means that at CP overload, SPU will no longer fetch signals from the XPU.

If the shared memory between the XPU and PLEX become exhausted, for instance due to stop scan, signal sending towards PLEX will fail and a fault code will be returned indicating buffer congestion. The application must then comply to this backpressure mechanism and stop producing data towards the PLEX domain.

If the application receives data from an external host via TCP/IP, it can decide to buffer for a short while or immediately stop retrieving data from the underlying transport layer. If this is done properly, the receive queues starts getting filled and flow control on TCP (or SCTP) protocol regulates flow of incoming packets from network.

A Resume indication is sent by APZ to XPU applications as soon as the XPU-PH on the SPU starts fetching data and new sending towards IPU is possible. When this happens, the application can empty buffer and restart fetching data from the transport layer. Receiver window of transport protocol will grow progressively and normal network data flow from remote peer will be restored

11 Terminology and Abbreviations

APG	Adjunct Processor Group.
APT	The application part of AXE
APZ	The Control part of AXE
Back Pressure	If the receiving buffers are full and incapable of receiving any more data, Back Pressure is a mechanism which makes the transmitting device to halt the sending of data packets until more data can be received.
Bootimage	An image (file) that contains the operating System and utilities needed to provide the execution environment for APZ VM and the XPU
bootstrap	In this document referred to as XPU bootstrap. Provides support to APZ VM to store the PARENTX XM in Linux file system and start the same.
CLH	Central Log Handler. Manages log files on the APG.
CP	Central Processor
CPHW	Central Processor HardWare
CPU	Central Processing Unit
Flow Control	A mechanism to avoid having the sender send data too fast for the receiver to reliably receive and process the data.
IPLIBX	XM that provide the IP stack to the XPU environment
IPU	Instruction Processor Unit
MAS	Maintenance Subsystem
MXPREC	PLEX block in MAS managing XPU faults

PARENT	Also referred to as PARENTX. A master XM that start and supervise all other XMs
PES	PLEX Engine Subsystem
PHC	Program handling Circuit
PLEX	Programming Language for EXchanges
POD	Print Out Description
SCTP	Stream Control Transmission Protocol
SDK	Software Development Kit
SEA	Simulated Architecture AXE
SLES	SUSE Linux Enterprise Server
SPU	Signal Processing Unit
TCP	Transmission Control protocol
UTILX	XM that provide debug and load measurement capabilities to the XPU
VDE	Virtual Development Environment
VLAN	Virtual LAN
VMIPADM	PLEX block that manages IP configuration in the CP
Watermark	A mechanism where for instance an IP stack informs it's user when certain thresholds for buffering of data is reached.
XM	XPU Module, an XPU application
XPU	eXtra Processing Unit
XPUADM	PLEX block that manages XPU administration
XPULOAD	PLEX block that manages storage of XSU's in the CP
XSU	A file that contains the XPU application, metadata about the application and a configuration file that specify how to start the application.

12 References

[1] XPU Utilx User Guide, 1/198 17-CNZ 211 635

[2] BLOCK XPUADM , 1551-CNZ 225 31 Uen

[3] BLOCK XPULOAD, 1551-CNZ 225 32 Uen

[4] PLEX INTERFACE for IP addresses on APZ CP, 45/155 19-ANZ 225 01/8 Uen

[5] XPU PLEX INTERFACE, 50/155 19-ANZ 225 21/1 Uen

[6] XPU SDK User Guide, 15513CXA 110 4717

[7] XPU Hardware and Software Recovery, 29/155 16-ANZ 214 120

[8] Interwork Description for XM, 110/155 19-ANZ 212 61

[9] TIP Socket API FS, 2/155 17-ANZ 212 61